Astruc Lélio
Gérard Théva

POLYTECH
NICE-SOPHIA

Rapport de projet : Analyse d'images en Java

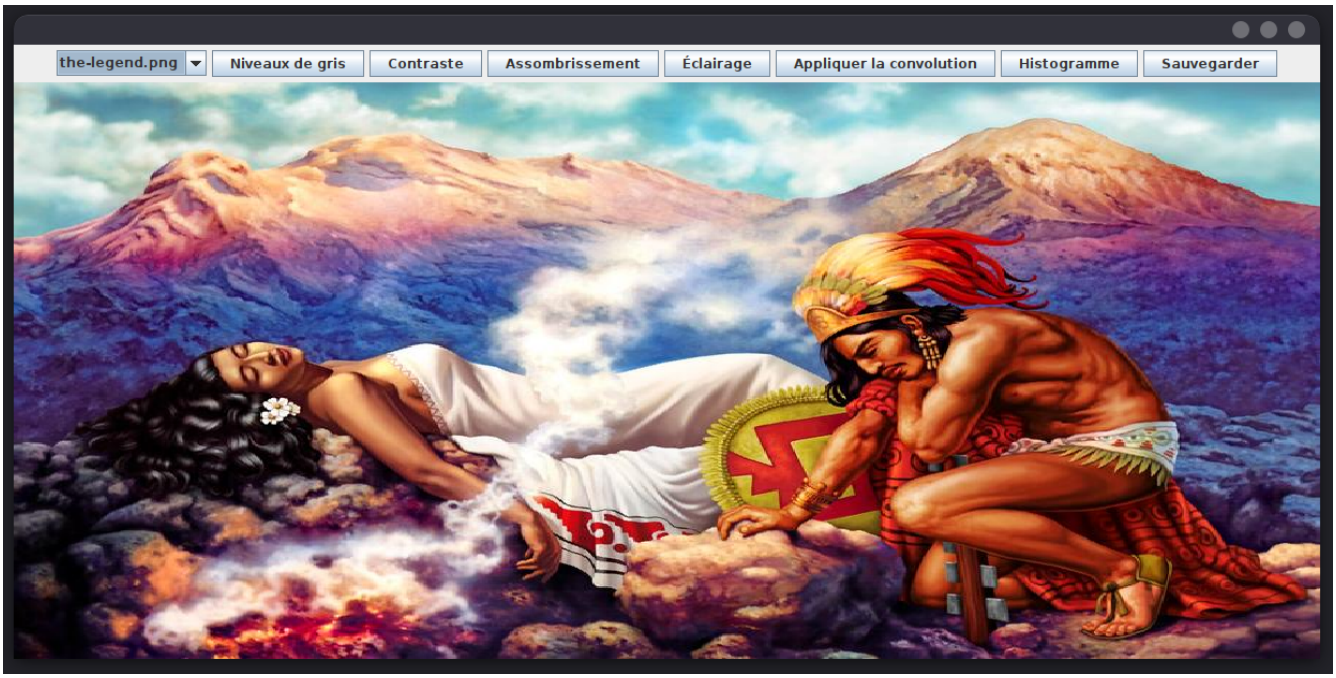


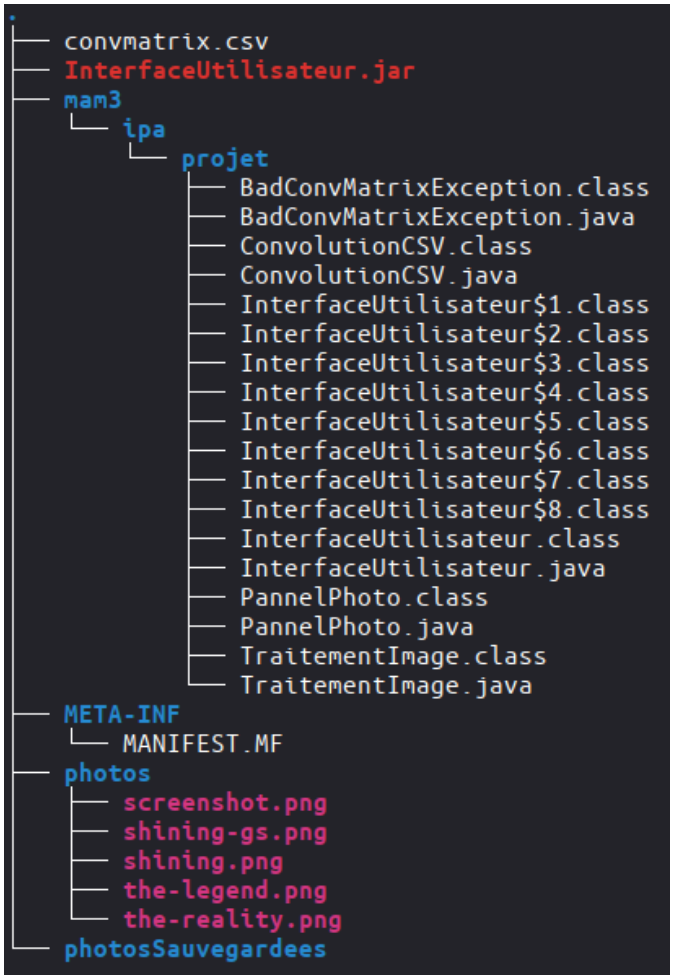
TABLE DES MATIÈRES

Présentation du projet

Dans le cadre de cette UE EIMA523, nous avons eu la possibilité de réaliser un programme permettant d'analyser des images. Notre programme permet la transformation en niveaux de gris, l'éclairage et l'assombrissement de l'image et appliquer le contraste. Il est également possible, grâce à un fichier appelé `convmatrix.csv`, d'appliquer un filtre de convolution à l'image. C'est à l'utilisateur de modifier ce fichier pour appliquer une matrice de convolution différente. Enfin, l'utilisateur a la possibilité de générer un fichier `.txt` de l'histogramme d'une image.

Architecture du projet

L'architecture de notre projet est la suivante :



La classe TraitementImage.java

La classe `TraitementImage.java` est la classe où sont les méthodes de calcul pour les transformations des images.

Transformation en niveaux de gris

```
Fonction appliquerTransformationNiveauxDeGris(image):
Pour chaque ligne dans image :
   Pour chaque colonne dans image:
     couleur = obtenir la couleur du pixel qui a pour coordonnees (ligne, colonne)
     rouge = recuperer le rouge de couleur * 0.21
     vert = recuperer le vert de couleur * 0.72
     bleu = recuperer le bleu de couleur * 0.07
     nouvelleCouleur = (rouge + vert + bleu, rouge + vert + bleu, rouge + vert + bleu)
     definir le pixel qui a pour coordonnees (ligne, colonne) avec nouvelleCouleur
```

où les coefficients $0.21\%, 0.72\%$ et 0.07% sont les coefficients donnés dans l'énoncé du sujet.
Complexité de l'algorithme : $O(ligne * colonne)$ où ligne et colonne sont la longueur et la largeur de l'image.

Contraste

```
Fonction appliquerContraste(image):
Pour chaque colonne dans limage :
   Pour chaque ligne dans limage :
     couleur = obtenir la couleur du pixel qui a pour coordonnees (ligne, colonne)
     rouge = recuperer le rouge de couleur
     vert = recuperer le vert de couleur
     bleu = recuperer le bleu de couleur
     couleurContraste = (255 - rouge, 255 - vert, 255 - bleu)
     definir le pixel qui a pour coordonnees (ligne, colonne) avec couleurContraste
```

Complexité de l'algorithme : $O(ligne * colonne)$ où ligne et colonne sont la longueur et la largeur de l'image.

Assombrissement

```
Fonction appliquerAssombrissement(image):
Pour chaque colonne dans limage :
   Pour chaque ligne dans limage :
     couleur = obtenir la couleur du pixel qui a pour coordonnees (ligne, colonne)
     rouge = recuperer le rouge de couleur
     vert = recuperer le vert de couleur
     bleu = recuperer le bleu de couleur
     couleurAssombrissement = ((int)(rouge^2)/255, (int)(vert^2)/255,
   (int)(bleu^2)/255)
     definir le pixel qui a pour coordonnees (ligne, colonne) avec
   couleurAssombrissement
```

Complexité de l'algorithme : $O(ligne * colonne)$ où ligne et colonne sont la longueur et la largeur de l'image.

Éclairage

```
Fonction appliquerEclairage(image):
Pour chaque colonne dans limage :
   Pour chaque ligne dans limage :
     couleur = obtenir la couleur du pixel qui a pour coordonnees (ligne, colonne)
     rouge = recuperer le rouge de couleur
     vert = recuperer le vert de couleur
     bleu = recuperer le bleu de couleur
     couleurEclairage = ((int) sqrt(rouge)*16, (int) sqrt(vert)*16, (int)
   sqrt(bleu)*16)
     definir le pixel qui a pour coordonnees (ligne, colonne) avec couleurEclairage
```

Complexité de l'algorithme : $O(ligne * colonne)$ où ligne et colonne sont la longueur et la largeur de l'image.

Convolution

```
Fonction appliquerConvolution(image, matrice):
longueurMatrice = longueur de matrice
hauteurMatrice = hauteur de matrice
moitieLongueurMatrice = longueurMatrice / 2
moitieHauteurMatrice = hauteurMatrice / 2
imageTemporaire = nouvelle image de meme taille et type que image
Pour y allant de moitieHauteurMatrice a hauteurImage - moitieHauteurMatrice:
   Pour x allant de moitieLongueurMatrice a largeurImage - moitieLongueurMatrice:
     rouge = 0
     vert = 0
     bleu = 0
     Pour matriceY allant de 0 a hauteurMatrice:
       Pour matriceX allant de 0 a longueurMatrice:
           pixelX = x + matriceX - moitieLongueurMatrice
           pixelY = y + matriceY - moitieHauteurMatrice
           couleurPixel = couleur du pixel en (pixelX, pixelY) de image
           rouge = rouge + (couleurPixel.obtenirRouge() * matrice[matriceY][matriceX])
           vert = vert + (couleurPixel.obtenirVert() * matrice[matriceY][matriceX])
           bleu = bleu + (couleurPixel.obtenirBleu() * matrice[matriceY][matriceX])
     rouge = min(max(rouge, 0), 255)
     vert = min(max(vert, 0), 255)
     bleu = min(max(bleu, 0), 255)
     couleurConvolution = (rouge, vert, bleu)
     definir le pixel qui a pour coordonnees (x, y) de imageTemporaire avec
   couleurConvolution

Pour y allant de 0 a hauteur de image:
Pour x allant de 0 a largeur de image:
definir le pixel qui a pour coordonnees (x, y) de image avec le pixel qui a pour
   coordonnees (x, y) de imageTemporaire
```

La convolution est la partie qui nous a causé le plus de soucis. En effet, nous avons mis beaucoup de temps à comprendre ce qu'était réellement le procédé de filtre de convolutions sur une image. Voici l'essentiel de la méthode que nous avons appliqué :

1. On calcule la longueur et la hauteur de la matrice
2. On calcule la moitié de la longueur et de la hauteur de la matrice
3. On crée une image temporaire de même taille et type que l'image initiale
4. Pour chaque pixel de l'image :
   (a) On initialise les composantes rouge, verte et bleu à 0
   (b) Pour chaque élément de la matrice :
      i. On calcule la position du pixel de l'image
      ii. On récupère la couleur du pixel de l'image en ces positions
      iii. On calcule la valeur des composantes rouge, verte et bleu d'un pixel en prenant en compte les pixels qui l'entourent
   (c) On limite les valeurs de chaque composante rouge, verte et bleu entre 0 et 255
   (d) On crée une couleur à partir de ces valeurs
   (e) On affecte cette couleur au pixel de l'image temporaire
5. Pour chaque pixel de l'image initiale :
   (a) On affecte la couleur du pixel de l'image temporaire au pixel de l'image initiale

Complexité de l'algorithme : $O(ligne*colonne*ligneMatrice*colonneMatrice)$ où ligne et colonne sont la longueur et la largeur de l'image.

Histogramme

creerHistogramme

```
Fonction creerHistogramme(image) :
histogram = int[3][256]
Pour chaque colonne dans limage :
   Pour chaque ligne dans limage :
     couleur = obtenir la couleur du pixel qui a pour coordonnees (ligne, colonne)
     rouge = recuperer le rouge de couleur
     vert = recuperer le vert de couleur
     bleu = recuperer le bleu de couleur
     histogram[0][rouge]++
     histogram[1][vert]++
     histogram[2][bleu]++
return histogram
```

Complexité de l'algorithme : $O(ligne * colonne)$ où ligne et colonne sont la longueur et la largeur de l'image.

appliquerHistogramme

```
Fonction appliquerHistogramme(image, nomHistogrammeTxt) :
histogram = creerHistogramme(image)
Essayer :
ecrireTexte = BufferedWriter(new FileWriter(nomHistogrammeTxt))
 Pour i allant de 0 a 256 :
     ligne = i+","+histogram[0][i]+","+histogram[1][i]+","+histogram[2][i]
     ecrireTexte.write(ligne)
     ecrireTexte.newLine()
Si erreur :
 Afficher la trace des appels des methodes qui ont conduit a l exception
```

La classe ConvolutionCSV.java

La classe ConvolutionCSV permet de lire le fichier `convmatrix.csv` du dossier source et de le transformer en un tableau de réels à deux dimensions (i.e. la matrice de convolution).

CSVtoMatrix

```
Fonction CSVtoMatrix() :
scanner = lecture("convmatrix.csv")
matrice = double [3][3]
ligne = 0
Tant que la matrice a une ligne suivante :
   rangee = caractere de scanner
   valeur = nombre de caracteres
   Pour colonne allant de 0 a valeur.length :
       Si valeur.length > 3 :
           Lever BadConvMatrixException()
       matrice[ligne][colonne] =  (double) valeurs[colonne])
   ligne = ligne + 1
   Si ligne > 3 :
       Lever BadConvMatrixException()
return matrice
```

Complexité de l'algorithme : $O(9)$

La classe BadConvMatrixException.java

La classe `BadConvMatrixException.java` est une exception à lever si la matrice de convolution choisie par l'utilisateur n'est pas bonne. Celà arrive si la matrice n'est pas de dimensions 3 par 3 (en réalité il faudrait que le programme marche avec n'importe quelle matrice carrée de dimensions impaire).
La classe se compose uniquement du constructeur qui renvoie un message d'erreur indiquant à l'utilisateur que les dimensions ne sont pas correctes.
Cette classe étend la classe `Exception`.

La classe PannelPhoto.java

Cette classe sert à actualiser l'image à chaque fois qu'une opération est appliquée.
Elle ne contient qu'un constructeur qui prend en paramètre un objet de type `Graphics` et lui applique la fonction `drawImge()`, ainsi que de ses accesseurs.

La classe InterfaceUtilisateur.java

C'est ici que tout s'imbrique pour donner naissance au programme. Les attributs de la classe sont le menu déroulant qui permet de choisir l'image, les différents boutons qui permettent d'appliquer les transformations sur les images ainsi qu'un objet du type `PannelPhoto`.
Le constructeur de la classe ne prend pas d'arguments et initialise l'interface graphique. On y ajoute des `addActionListener` qui permettent d'associer une action aux boutons et au menu déroulant. Il nous semble que c'est à cause de ceci que plusieurs `InterfaceUtilisateur$n.class`, $n \in \mathbb{N}$, sont créés. Ce sont apparemment des classes anonymes. Étant donné que le code marche quand même, nous avons décidé de laisser le code ainsi.
C'est également dans cette classe que le `main()` se trouve. C'est donc cette classe que l'on va executer.

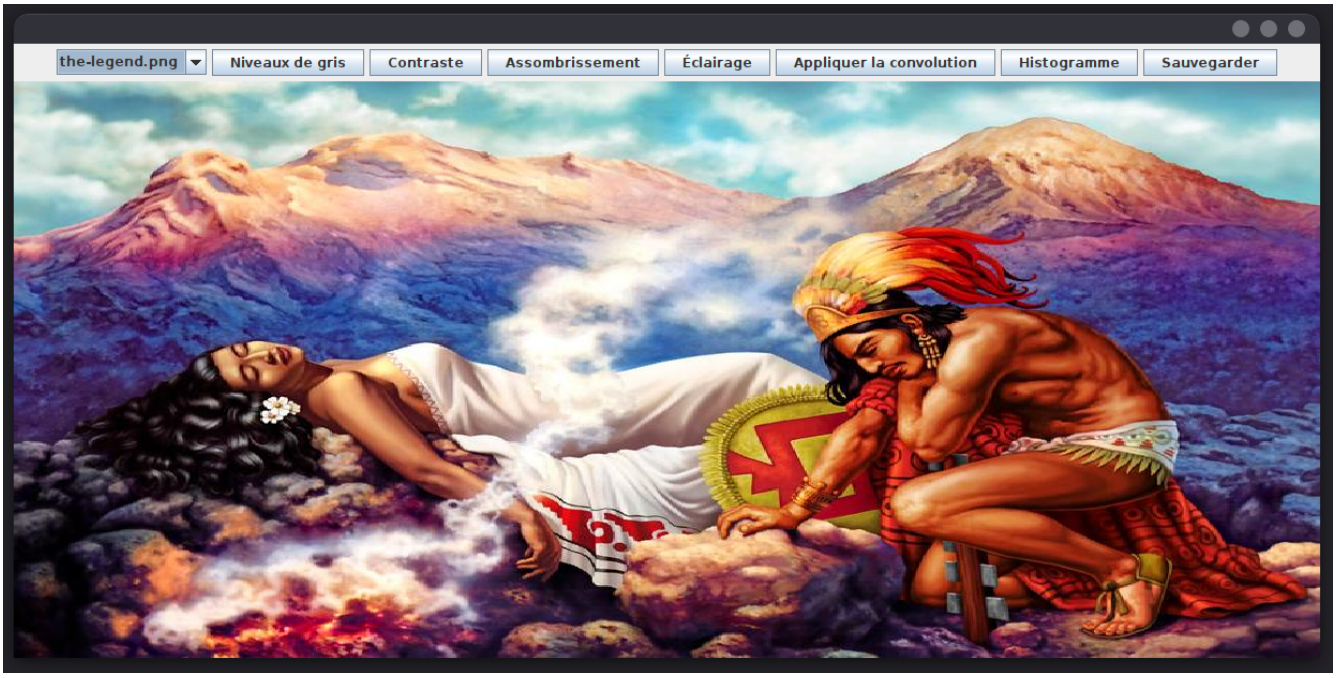
Installation

Voici une explication pour installer le projet :
Téléchargez le dossier et ouvrez un terminal dans le dossier dans lequel vous avez téléchargé le fichier.
Executez les commandes suivantes :

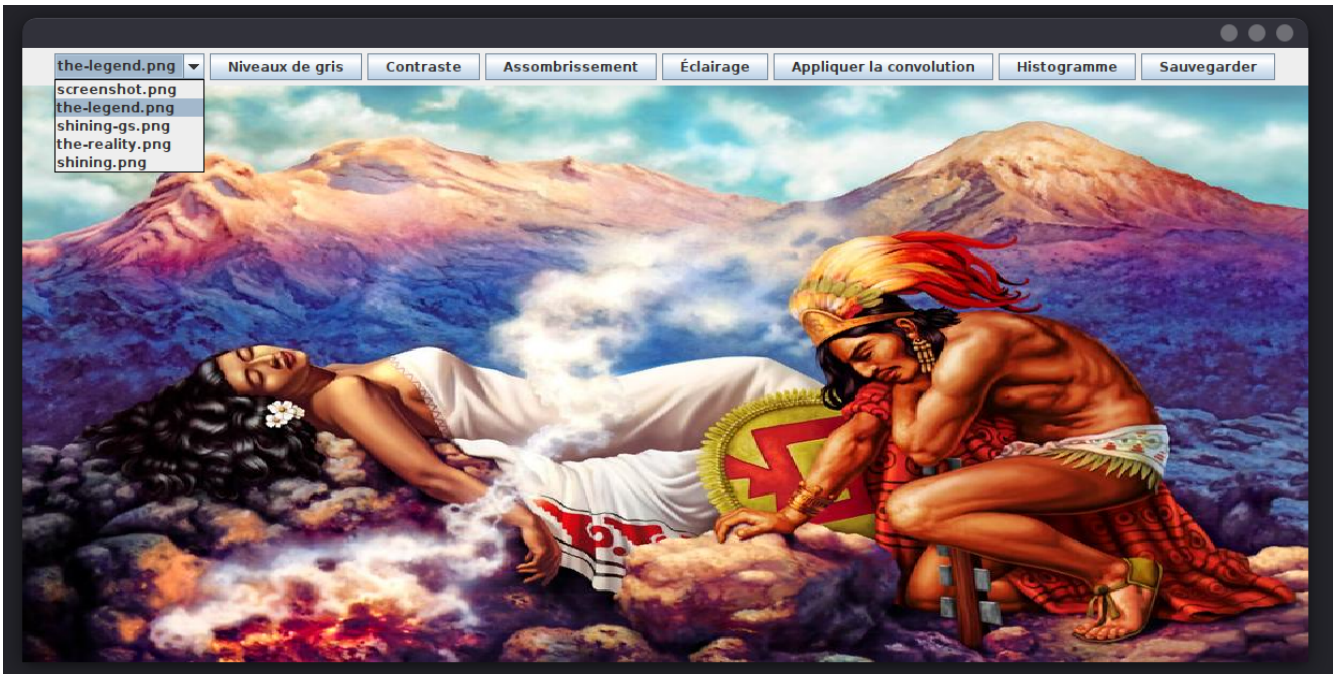
```
$ tar -xvf AstrucGerard.tar.xz
$ cd AstrucGerard
$ java -jar AstrucGerard.jar
ou
$ cd mam3/ipa/projet && javac *.java
$ cd .. && cd .. && cd ..
$ java mam3.ipa.projet.InterfaceUtilisateur
```

Utilisation

Maintenant que le programme est installé et prêt à être utilisé, voici comment l'utiliser :



Le menu déroulant permet de sélectionner l'image sur laquelle on veut appliquer les transformations



Les boutons à droite permettent d'effectuer les transformations dont nous avons déjà parlé ; chaque transformation donne un effet visible à part Histogramme qui génère un fichier `nomImage-h.txt` dans le dossier du programme. Enfin, le bouton Sauvegarder permet d'enregistrer l'image dans le

dosser `photosSauvegardees`.
De plus, comme dit précédemment, le bouton Appliquer la convolution utilise le fichier `convmatrix.csv`. L'utilisateur doit donc éditer ce fichier par lui même. Voici quelques matrices que nous recommandons

$$\text{Détection de bords} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

$$\text{Amélioration de la netteté} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

$$\text{Box blur} = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

ToDo

Voici une liste des choses que nous aurions aimé implémenter dans le programme mais que nous n'avons pas pu faire par manque de temps ou car nous n'avons simplement pas réussi :

* Faire en sorte de pouvoir appliquer des matrices de convolution de n'importe quelle taille (dimension impaire)
* Modifier le fichier `InterfaceUtilisateur.java` de manière à ne pas créer plusieurs `InterfaceUtilisateur$n.class`
* Permettre à l'utilisateur de modifier le fichier `convmatrix.csv` pendant que le programme est ouvert. En effet, pour appliquer une autre matrice de convolution il faut nécessairement fermer le programme puis le ré-ouvrir après avoir modifié le fichier.
* Permettre à l'utilisateur d'annuler une transformation
* Modifier la méthode de sauvegarde de `InterfaceUtilisateur.java`. En effet, si peu de transformations sont appliquées le nom de l'image sauvegardée est assez lourd.

Il n'y a pas de bugs connus.