Aix*Marseille université
Initiative d'excellence
Année 2018-19

Domaine Sciences et Technologies
Licence informatique

Programmation 2 : TP 3
Code UE : SIN3U02

Manipulation d'images

1 Introduction

On s'intéresse à la représentation et la manipulation d'images. Ces images seront constituées de pixels caractérisés par une couleur représentant un niveau de gris.

1.1 Gestion de versions

Ce projet est l'occasion d'apprendre à utiliser des outils de développement professionnel. En plus de l'IDE IntelliJ, vous aurez à utiliser un outil de gestion de versions, en l'occurrence `git`. Un tel outil permet d'enregistrer à distance votre projet et de permettre à plusieurs personnes de travailler simultanément, sans devoir échanger les fichiers par courriel, et sans se marcher sur les pieds (par exemple modification sans le savoir du même fichier).

Le principe de `git` est d'avoir un *dépot* distant : une version de votre projet stockée sur un serveur sur Internet (en l'occurrence par `github`). Vous disposez en plus de dépots locaux sur les ordinateurs sur lesquels vous travaillez. Vous faites vos modifications sur votre ordinateur, et lorsque vous avez accompli une amélioration qui fonctionne bien, vous pouvez la faire enregistrer (`commit`) par `git`. Ces enregistrements sont locaux à votre ordinateur, et vous pouvez en faire autant que vous le souhaitez. Si vous voulez partager votre travail avec votre équipe, il vous faut l'envoyer vers le dépot distant (`push`). À l'inverse, si vous souhaitez récupérer le travail fait par vos co-équipiers, il faut ramener ces modifications depuis le dépot distant (`pull`). IntelliJ est capable de gérer `git` ; vous trouverez dans le menu `VCS` l'option `Commit`, et l'option `Git` qui contient `push` et `pull`.

Vous travaillerez de la façon suivante :

- Si vous ne possédez pas de compte Github, aller sur github et créez-vous un compte gratuit.
- Allez sur le lien du projet, créer une équipe ou joignez une équipe existante (2 personnes maximum par équipes). Mettez vous d'accord avant !
- Au tout début, vous lancerez IntelliJ, puis commencerez un nouveau projet depuis un dépot `git` (menu `File`, `New`, `Project from version control`, `Git`). Renseignez l'adresse du dépot `git` fournie par `github`, vous la trouverez en cliquant sur le bouton vert `clone or download` sur la page `github` du projet. IntelliJ va initialiser votre dépot local, en vous demandant dans quel répertoire le placer.
- Modifiez le fichier `README.md`. Mettez votre nom ainsi que le nom de votre éventuel co-équipier. Faites un `commit` avec pour message "inscription d'un membre de l'équipe", puis un `push`.
- À chaque avoir fini une modification de code, faites un `commit` en sélectionnant les fichiers modifiés. Chaque `commit` doit contenir un message précisant la nature des modifications effectuées.
- À chaque tâche terminée, faites un `push` de votre travail.
- Ceci est le minimum. Vous pouvez faire plus de `commit` et plus de `push`, ainsi que des `pull` pour récupérer le travail de votre co-équipier.
- Si vous avez un problème et souhaitez l'aide de votre instructeur en dehors des séances, un `push` lui permet de voir votre programme.

1.2 Maven

Ce projet est compilé grâce Maven qui est un moteur de production pour Java. Lorsque vous importez votre projet il faut donc choisir le *template* maven et accepter d'ajouter le projet en tant que projet Maven (*add as maven project*).

1.3 Consignes du projet

Quelques consignes à respecter pour ce TP :
- Chaque classe ou interface devra être dans un fichier séparé et être contenu dans le répertoire `src/main/java` du projet.
- Tester dès que possible les fonctionnalités que vous avez codées.
- Demander de l'aide si vous êtes bloqué.

2 Définitions des couleurs et des images

2.1 Définition des couleurs

Les fonctionnalités d'une couleur représentant un niveau de gris sont décrites dans l'interface `GrayColor` :

```
public interface GrayColor extends Comparable<GrayColor> {
    int getGrayLevel();
    void setGrayLevel(int grayLevel);
    Color getColor();
}
```

Les niveaux de gris considérés seront codés par un entier sur un octet, et donc entre 0 et 255. De telles couleurs sont représentées par la classe `ByteGrayColor`.

Une classe implémentant cette interface doit donc implémenter les quatre méthodes suivantes :
- `int getGrayLevel()` : renvoyant le niveau de gris de la couleur.
- `void setGrayLevel(int grayLevel)` : modifiant le niveau de gris de la couleur.
- `Color getColor()` : renvoyant une couleur pour l'affichage.
- `compareTo(GrayColor o)` : renvoie la différence entre les `greyLevels`, les couleurs ayant un `greyLevel` plus petit que d'autres couleurs seront considérées comme plus petites pour l'ordre de compareTo.

2.1.1 Tâche 1 : classe `ByteGrayColor`

Compléter le code de la classe `ByteGrayColor` implémentant l'interface `GrayColor`. La méthode `Color getColor()` est déjà implémentée. En plus de compléter les méthodes, il vous faudra compléter le code des deux constructeurs suivants :
- `ByteGrayColor()` : construit une couleur avec un niveau de gris égal à `MINIMUM_GRAY_VALUE`.
- `ByteGrayColor(int grayLevel)` : construit une couleur avec un niveau de gris égal à l'argument.

2.2 Définition des images

Les fonctionnalités d'une couleur représentant un niveau de gris sont décrites dans l'interface `GrayColor` :

```
public interface GrayImage extends Image{
    void setGrayLevel(int graylevel, int x, int y);
    int getGraylevel(int x, int y);
    GrayColor getPixelGrayColor(int x, int y);
}
```

L'interface `Image` est définie par :

```
public interface Image {
    Color getPixelColor(int x, int y);
    int getWidth();
    int getHeight();
}
```

Une classe implémentant cette interface doit donc implémenter les six méthodes suivantes :

— `GrayColor getPixelGrayColor(int x, int y)` : renvoie la `GrayColor` du pixel (x, y).
— `int getWidth()` : renvoie la largeur de l'image.
— `int getHeight()` : renvoie la hauteur de l'image.
— `int getGrayLevel(int x, int y)` : renvoie le niveau de gris de la couleur du pixel (x, y).
— `void setGrayLevel(int graylevel, int x, int y)` : modifie le niveau de gris de la couleur du pixel (x, y).
— `Color getPixelColor(int x, int y)` : renvoie la couleur du pixel (x, y) pour l'affichage.

Pour une image, x représente la coordonnée « horizontale » et y la coordonnée « verticale ». Le point de coordonnées $(0, 0)$ est le point situé en haut à gauche de l'image. L'axe des x est donc orienté vers la droite et celui des y vers le bas comme pour le précédent TP.

2.2.1 Tâche 2 : classe `MatrixGrayImage`

Compléter le code de la classe `MatrixGrayImage` implémentant l'interface `GrayImage`. En plus de compléter les méthodes, il vous faudra compléter le code d'un constructeur `MatrixGrayImage(int width, int height)`qui initialise un image de taille `width` × `height` avec toutes les couleurs de la matrice `pixels` construites grâce au constructeur `ByteGrayColor()`. La matrice `pixels` stocke les couleurs des pixels de l'image : la case à la ligne `x` et colonne `y` contient donc la couleur du pixel (x, y).

Vous devriez obtenir l'affichage suivant :

3 Transformations d'images

On souhaite faire des manipulations simples d'images. Pour cela, vous allez définir l'interface `Transform` suivante :

```
public interface Transform {
    void applyTo(GrayImage image);
}
```

Un appel à la méthode `applyTo` devra modifier l'image suivant la transformation.
Vous allez donc définir des classes implémentant cette interface.

3.1 Inversion des niveaux de gris

La première transformation consiste à modifier chaque pixel de l'image de sorte à ce que le nouveau niveau de gris de chaque pixel soit égal au niveau de gris maximum auquel on soustrait l'ancien niveau de gris.

3.1.1 Tâche 3 : classe `Invert`

Avant d'écrire le code de la classe `Invert` qui va nous permettre de faire la transformation d'image, on va procéder aux changements suivants pour accéder au niveau de gris maximal d'un pixel :

- Ajouter à l'interface `GrayColor` la méthode `int getMaximumGrayLevel()`.
- Implémenter la méthode `int getMaximumGrayLevel()` dans la classe `ByteGrayColor` de manière à ce qu'elle retourne le niveau du gris maximum.
- Ajouter à l'interface `GrayImage` la méthode `int getMaximumGrayLevel(int x, int y)`.
- Implémenter la méthode `int getMaximumGrayLevel(int x, int y)` dans la classe `MatrixGrayImage` de manière à ce qu'elle retourne le niveau du gris maximum du pixel (x, y).

Définissez l'interface `Transform` puis créez la classe `Invert` implémentant l'interface `Transform`.

Changer le code de la méthode `initialize` à l'intérieur de la classe `Display` de sorte à ce que vous appliquez la transformation `Invert` à l'attribut `image` entre le chargement de l'image (appel à `createImageFromPGMFile`) et son rendu (appel à `render`).

Vous devriez obtenir l'affichage suivant :

3.2 Diminution des niveaux de gris

On cherche maintenant à modifier une image en diminuant le nombre de niveaux de gris. Ce nombre de niveaux de gris sera un attribut `nbGrayLevels` de la classe `DecreaseGrayLevels` qu'on supposera être une puissance de 2 comprise entre `2` et `128`.

On peut travailler de manière assez simple : on décompose l'intervalle $[0, 255]$ en `nbGrayLevels` sous-intervalles de taille t. Dans la nouvelle image, chaque pixel de l'image initial de couleur $c \in [k \times t, (k+1)t[$ est remplacé par un pixel de couleur $k \times t$.

3.2.1 Tâche 4 : classe `DecreaseGrayLevels`

Créer la classe `DecreaseGrayLevels` implémentant l'interface `Transform`.

Changer le code de la méthode `initialize` à l'intérieur de la classe `Display` de sorte à ce que vous appliquez la transformation `DecreaseGrayLevels` à l'attribut `image` entre le chargement de l'image (appel à `createImageFromPGMFile`) et son rendu (appel à `render`). L'attribut `nbGrayLevels` de l'instance de `DecreaseGrayLevels` devra être égal à `8`.

Vous devriez obtenir l'affichage suivant :

3.3 Dessin des contours

Vous allez créer une classe `Outline` qui modifie une image par extraction de contours.

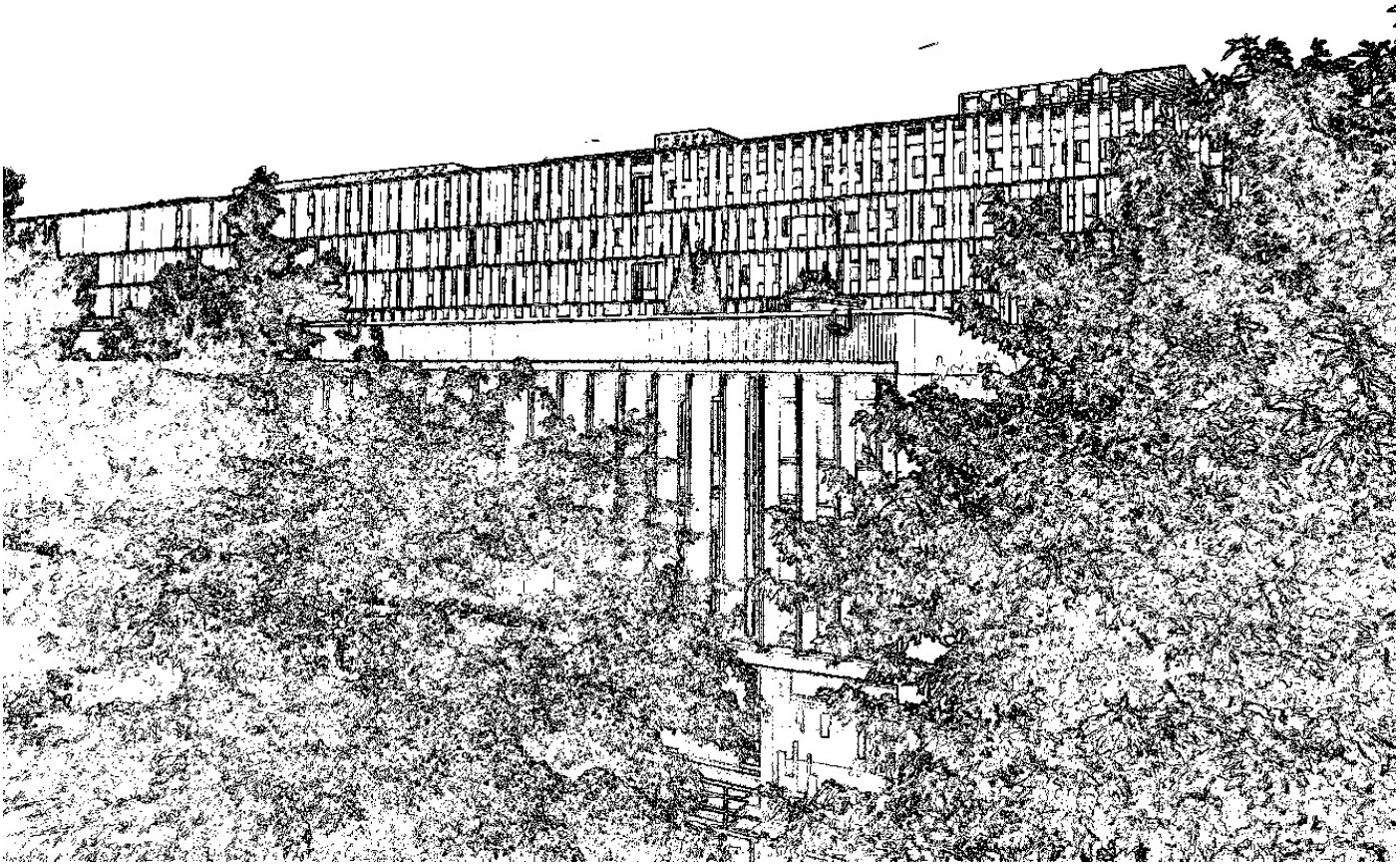
Il n'est pas très compliqué de modifier l'image. En effet, une manière de procéder est la suivante : un pixel de l'image résultat est noir s'il appartient au contour de l'image initiale, c'est-à-dire s'il est très différent de l'un des deux pixels situés à sa droite ou en-dessous de lui dans l'image initiale. L'expression **Très différent** signifie que la valeur absolue (fonction `Math.abs`) de la différence entre les niveaux de gris de deux pixels est supérieure à un seuil fixé. Les autres pixels de l'image (qui ne sont pas identifiés comme appartenant à un contour) sont blancs. Le seuil à prendre en compte pour l'extraction des contours sera un attribut de la classe `Outline` nommé `threshold`. Cet attribut devra être initialisé par un constructeur `Outline(int threshold)`.

3.3.1 Tâche 5 : classe `Outline`

Créer la classe `Outline` implémentant l'interface `Transform`.

Changer le code de la méthode `initialize` à l'intérieur de la classe `Display` de sorte à ce que vous appliquez la transformation `Outline` à l'attribut `image` entre le chargement de l'image (appel à `createImageFromPGMFile`) et son rendu (appel à `render`). L'attribut `threshold` de l'instance de `Outline` devra être égal à `10`.

Vous devriez obtenir l'affichage suivant :



3.4 Pixelisation

L'idée de cette transformation est de découper l'image en carré d'une certaine taille. La figure ci-dessous illustre un découpage en carré de taille 10.

L'idée est de calculer le niveau de gris moyen au sein d'un carré puis de modifier tous les pixels du carrés de sorte à ce qu'ils aient ce niveau de gris. La taille d'un coté d'un tel carré sera défini par un attribut `newPixelSize` de la classe `Pixelate`.

3.4.1 Tâche 6 : classe `Pixelate`

Créer la classe `Pixelate` implémentant l'interface `Transform`.

Changer le code de la méthode `initialize` à l'intérieur de la classe `Display` de sorte à ce que vous appliquez la transformation `Pixelate` à l'attribut `image` entre le chargement de l'image (appel à `createImageFromPGMFile`) et son rendu (appel à `render`). L'attribut `newPixelSize` de l'instance de `Pixelate` devra être égal à `10`.

Vous devriez obtenir l'affichage suivant :

3.5 Composition de transformations

L'idée de cette transformation est de composer plusieurs transformations et de les appliquer successivement à l'image. Cette transformation devra donc appliquer une séquence de transformations qui sera un tableau `Transform[] transforms` passé au constructeur.

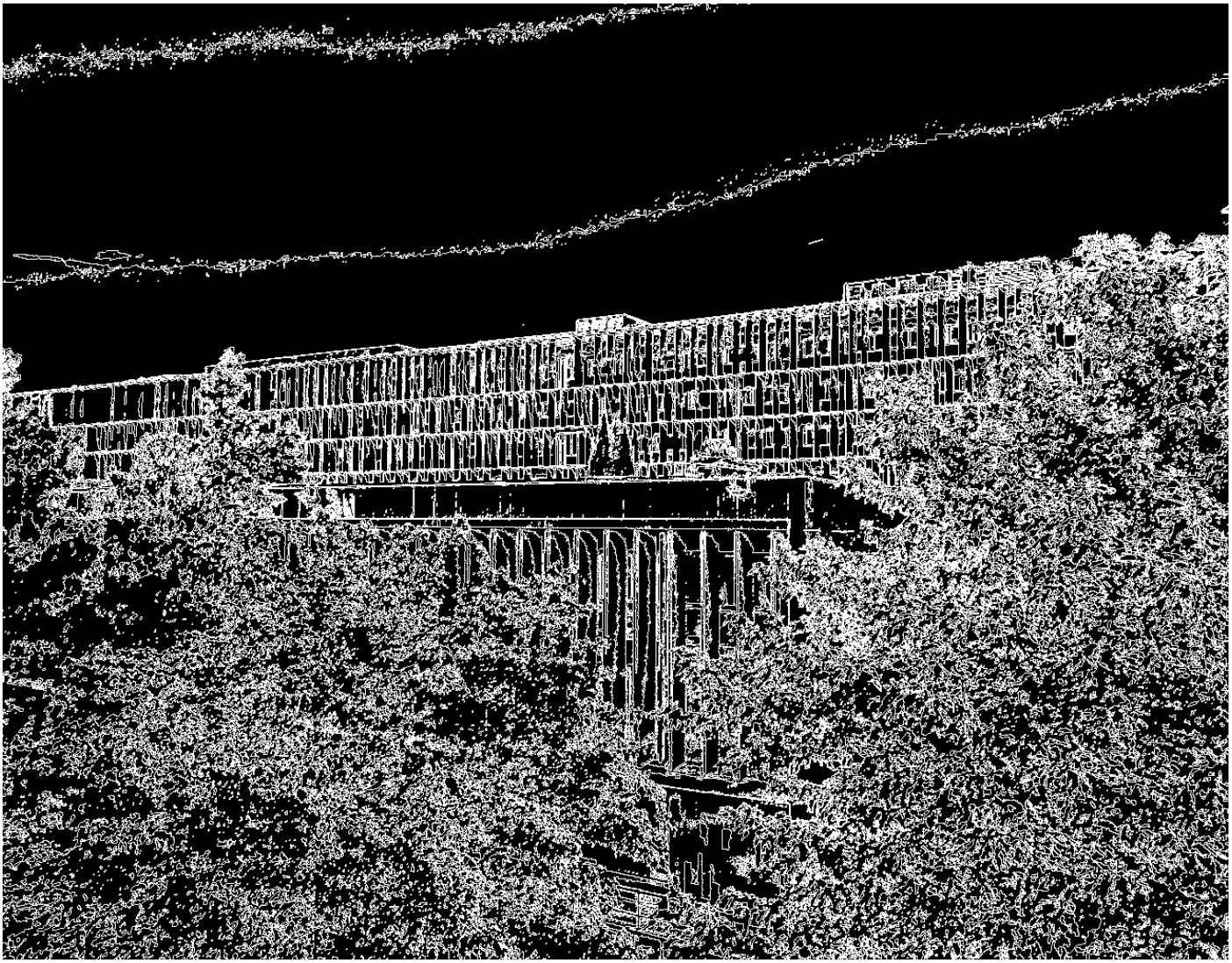
3.5.1 Tâche 7 : classe `CompositeTransform`

Créer la classe `CompositeTransform` implémentant l'interface `Transform`.

Changer le code de la méthode `initialize` à l'intérieur de la classe `Display` de sorte à ce que vous appliquez la transformation `CompositeTransform` à l'attribut `image` entre le chargement de l'image (appel à `createImageFromPGMFile`) et son rendu (appel à `render`). L'attribut `transforms` de l'instance de `CompositeTransform` devra contenir les trois transformations suivantes (dans cet ordre) :

— `DecreaseGrayLevels` avec `8` pour la valeur de `nbGrayLevels`
— `Outline` avec `20` pour la valeur de `threshold`
— `Invert`

Vous devriez obtenir l'affichage suivant :



4 Tâches supplémentaires

4.1 Rotation

Créer une classe de transformation permettant de faire des rotations (d'un angle multiple de 90°). Que doit-ont ajouter à la classe image pour permettre ces rotations ?

4.2 Images en couleurs

Rajouter des classes et extensions pour gérer les images en couleurs au format .ppm similaire au pmg sauf que les couleurs sont définies par trois entiers au format RGB

4.3 Menu

Rajouter un menu dans l'application permettant de contrôler les transformations et la lecture/écriture de fichier.