Programmation objet II	**420-216-FX**
Travail pratique #3 : Transformations bijectives d'images	**18%**
Prof.: Karine Filiatreault	***Hiver 2016***

Travail en équipe de deux. Le professeur doit approuver les équipes.

Date de remise:

- ***Au plus tard lundi le 23 mai 2016 avant 11h00.***

Objectif

L'objectif de ce travail est de construire une petite application effectuant des transformations bijectives sur des images. Une propriété importante d'une transformation bijective est qu'après un certain nombre fini d'étapes, l'image originale est reconstruite.

Dans le cadre de ce travail, les images seront considérées comme des matrices de pixels; ceci permettra de mettre en pratique la manipulation de matrices rectangulaires, les indexeurs et la surcharge des opérateurs.

Transformations bijectives à programmer

Vous devez programmer les transformations bijectives suivantes :

1. **Miroir horizontal** (inversion gauche-droite).
2. **Miroir vertical** (inversion haut-bas).
3. **Transposition** (dimensions identiques) selon la diagonale qui part du coin supérieur gauche au coin inférieur droit (inversion selon la diagonale).
4. **Décalage horizontal** vers la droite (une colonne à la fois).
5. **Décalage en diagonale** vers la droite et vers le bas (une colonne et une ligne à la fois).
6. **En colonnes** (dimensions paires) : Les colonnes de la section de droite sont intercalées avec celle de la section de gauche. Ainsi, les colonnes initiales 0, 1, 2, 3, 4, 5, 6, 7 (8 colonnes) sont déplacées de la manière suivante : 0, 4, 1, 5, 2, 6, 3, 7
7. **Photomaton** (dimensions paires). Pour des explications, voir : *http://sebastien.verspecht.com/2008/06/algorithme-la-transformation-du-photomaton.html*
8. **Boulanger** (dimensions paires). Pour des explications, voir : *http://sebastien.verspecht.com/2008/06/algorithme-la-transformation-du-boulanger.html*
9. **Fleur** (dimensions paires). Pour des explications, voir le début de l'article suivant (arrêtez à la section "Généralisation") : *http://sebastien.verspecht.com/2008/07/algorithme-la-transformation-de-la-fleur.html*
10. **Svastika** (dimensions paires et identiques).

Pour une démo de la plupart des transformations, voir le logiciel " transfo-bijectives.jar" fourni par le professeur.

Effets à programmer

Vous devez programmer les effets suivants à appliquer aux deux images (source et modifiée):

1. **Niveau de gris** : Utilisez les ratios suivants : 0.299 Rouge + 0.587 Vert + 0.114 Bleu.
2. **Sépia** : Voir la première réponse de la question suivante sur *stackoverflow* :

   http://stackoverflow.com/questions/1061093/how-is-a-sepia-tone-created

Fonctionnalités offertes par l'application

Votre interface graphique comportera une seule fenêtre comportant les éléments suivants :

1. Un **menu** (*MenuStrip*) offrant les deux options suivantes :
   - **Sélectionner** un **fichier image** sur le disque dur et de l'afficher dans l'interface. Ceci causera aussi la réinitialisation de la section pour l'image transformée.
   - **Quitter** l'application.
2. Un **ensemble de boutons** permettant de **contrôler les transformations** (voir la section suivante).
3. Un *Label* qui affiche le **numéro de l'itération courante** (doit être zéro si l'image source est affichée).
4. Un *TextBox* permettant de préciser **à quelle itération on désire s'arrêter** lorsqu'on utilise le bouton *PlayStop*. **Validations** et **mises en forme** requises pour ce champ.
5. Un *TextBox* permettant de préciser la **durée minimale entre deux itérations** en mode automatique; la valeur par défaut, doit être 100 millisecondes. **Validations** et **mises en forme** requises pour ce champ.
6. Un *Label* qui affiche un **message** indiquant qu'une **séquence de transformations a été complétée** et le **nombre d'itérations** requises pour reproduire l'image originale.
7. Un *ComboBox* (*DropDownList*) permettant de sélectionner le **type de transformation** à effectuer. Lorsqu'un changement est effectué dans ce *ComboBox*, vous devez réinitialiser la section pour l'image transformée.
8. Un *Button* permettant **d'afficher l'image source en niveau de gris** et de réinitialiser la section pour l'image transformée. Les transformations subséquentes devront être appliquées sur l'image en niveau de gris.
9. Un *Button* permettant **d'afficher l'image source en sépia** et de réinitialiser la section pour l'image transformée. Les transformations subséquentes devront être appliquées sur l'image en sépia.

Boutons de contrôle

Votre interface graphique doit comporter les boutons suivants permettant de contrôler les transformations :

1. ***Play*** : Permet d'avancer d'une seule étape.
2. ***FastPlay*** : Permet d'avancer automatiquement et d'autant d'étapes que nécessaire pour pouvoir reconstruire l'image originale (l'image doit être actualisée après chaque étape). Il est important d'arrêter automatiquement lorsque l'image originale est reconstruite.

3. ***Stop*** : Permet d'arrêter lorsqu'on est en mode automatique (FastPlay).
4. ***PlayStop*** : Permet de se déplacer à une itération précise de la transformation et de s'y arrêter. L'image est uniquement rafraichie pour l'itération où l'on doit s'arrêter (on n'affiche pas les images transformées intermédiaires).
5. ***GoToStart*** : Permet de revenir au début (sans transformation) et de réinitialiser la section pour l'image transformée dans l'interface (ne pas réinitialiser les *TextBox*).

Codage de la classe "BitmapMatricielle"

Pour manipuler les images vous devez utiliser uniquement la classe personnalisée "BitmapMatricielle". Une partie de cette classe vous est fournie dans le dossier de départ. Vous devrez compléter cette classe en y ajoutant les éléments suivants :

1. Un **indexeur** avec deux indices `[i,j]` permettant de manipuler les images comme une **matrice rectangulaire** de pixels. Le premier indice (`i`) dénote la ligne et le deuxième (`j`) dénote la colonne. Sauf dans cet indexeur, **vous ne devez pas utiliser les méthodes "SetPixel" et "GetPixel"** pour modifier une image. Attention : l'indice "i" correspond à la coordonnée "y" et l'indice "j" correspond à la coordonnée "x".
2. La **surcharge des opérateurs d'égalité et d'inégalité** permettant de vérifier si deux images sont identiques ou différentes. Les images sont considérées identiques si elles comportent exactement les mêmes pixels aux mêmes positions.
3. La **méthode** "**Equals(...)**" qui doit aussi être redéfinie lorsqu'on définit l'opérateur d'égalité.
4. **Pour chaque transformation** demandée, **une méthode** effectuant la transformation sur l'image (une seule étape). **Attention** : vos algorithmes doivent être **efficaces** et vous ne devez pas utiliser d'images bitmap temporaires si ceci n'est pas nécessaire. Par exemple, lorsqu'il est facile de créer l'image transformée par permutation, vous devez le faire.
5. **Pour chaque effet** demandé, **une méthode** générant cet effet autant sur l'image source que sur l'image transformée.
6. Vous devez **lancer des exceptions** (instruction ***throw***) du bon type lorsqu'une transformation demandée n'est pas possible. Assurez-vous de considérer tous les cas.

Directives spécifiques

- Pour la construction des **interfaces graphiques**, voir les **aperçus fournis**. Ce n'est pas essentiel de reproduire exactement la même interface graphique que les aperçus mais vous devez avoir les mêmes types de contrôle et les mêmes fonctionnalités. Les *PictureBox* doivent être de taille 512 pixels X 512 pixels.
- Vous pouvez attraper toutes les exceptions dans votre formulaire en utilisant un ***catch*** global (classe ***Exception***) à l'endroit approprié. Un seul catch pour ce travail suffira. Vous pouvez utiliser la méthode "**TryParse**" dans ce travail.
- Vos algorithmes pour les transformations doivent être efficaces et éviter de construire de nouveaux objets ***Bitmap*** lorsque c'est possible. Utilisez des variables temporaires lorsque cela permet d'améliorer l'efficacité (inversion de pixels, entre autres).
- Assurez-vous que votre application ne plante pas lorsqu'aucune image n'est chargée comme au chargement de l'application.

- Ne modifiez pas les espaces de noms (*namespace*) des fichiers de départ qui vous sont fournis.

Directives générales

- Vous devez respecter les **conventions** fournies par le professeur pour tous les **identificateurs en C#** (variables, méthodes, classes, contrôles de formulaire, etc.) ainsi que les autres conventions de codage du cours Programmation Objet II (420-216-FX). Voir le fichier "**Conventions pour les identificateurs en C#.pdf**" et "**Normes de codage C#.pdf**". De plus, tout identificateur doit avoir **un nom significatif** considérant le contexte dans lequel il apparaît.
- Tous les **contrôles de formulaire doivent être renommés**; il ne faut pas conserver les noms par défaut tels "label1", "button1". N'oubliez pas de **renommer** aussi les **formulaires** (ne pas utiliser les noms de classe par défaut) et de leur donner un titre.
- Vous devez mettre des **commentaires en format XML** pour tous les éléments suivants : **classes, énumérations, constantes, attributs, constructeurs, propriétés et méthodes**. À noter que pour les classes correspondant à des formulaires, il n'est pas nécessaire de mettre des commentaires XML pour les méthodes liées aux événements dans l'interface graphique.
- À **l'intérieur** des **méthodes**, vous devez **ajouter des commentaires** pour expliquer clairement ce que fait chaque section de code. Entre autres, toutes les **variables locales** doivent être accompagnées d'un commentaire expliquant son rôle.
- Vous devez vous assurez qu'il n'y ait pas **d'erreurs** (*errors*) selon ***ReSharper*** et la configuration fournie par le professeur. De plus, vérifiez minutieusement les **avertissements** (*warnings*) car il ne devrait presque pas y en avoir. Exécutez la commande suivante : *ReSharper/Inspect/Code Issues in Solution*.
- Les classes doivent toutes être présentées à l'aide du **gabarit de fichier** (*File Layout*) de ***ReSharper*** (Ctrl-E, F).

Précisions sur l'évaluation

L'évaluation portera sur les éléments suivants :

1. (**65 %**) Bon fonctionnement des **transformations et effets** :
   - ✓ Efficacité des algorithmes pour les transformations.
   - ✓ Utilisation d'une nouvelle image seulement lorsque c'est vraiment nécessaire.
   - ✓ Un message d'erreur apparaît lorsqu'une opération impossible est demandée.
   - ✓ En mode « auto play », arrête automatiquement lorsqu'on revient à l'image initiale.
2. (**15 %**) **Interfaces graphiques** :
   - ✓ Bon fonctionnement de l'interface graphique (menu, boutons de contrôle, champs de saisie, etc.)
   - ✓ Validation des entrées (durée minimale et numéro d'itération).
3. (**20 %**) **Codage et qualité du code** :
   - ✓ Bon codage de la classe « BitmapMatricielle ».
   - ✓ Gestion des exceptions (« throw » le bon type d'exception).
   - ✓ Erreurs et avertissements selon *ReSharper*.

- ✓ Respect des normes de codage.
- ✓ Noms significatifs pour les identifiants (en français).
- ✓ Noms pour les contrôles des formulaires (avec préfixe).
- ✓ Commentaires XML
- ✓ Commentaires à l'intérieur des méthodes.
- ✓ Présentation du code (gabarit *ReSharper*, structure, présentation, etc.)

Outre les éléments à réaliser dans le cadre du travail, les aspects suivants viendront influencer la note finale:

- Retard.
- Remise incorrecte du travail.
- Français.
- Contribution insuffisante au travail d'équipe.
- D'autres éléments selon le besoin.

À remettre

- **Aucune remise papier.**
- Créez le **dossier "TP3_Nom1_Nom2"** (utilisez vos prénoms et vos noms de famille complets) dans lequel vous devez copier votre dossier pour le projet C# qui doit inclure le **fichier solution (.sln)**. Vérifiez que tout fonctionne bien avant de le remettre en copiant le dossier et en tentant d'ouvrir votre projet à partir du fichier solution copié.

  Note : Il est essentiel de créer ce dossier car autrement je ne peux pas savoir qui est votre coéquipier.
- **Compressez** le dossier en format ".**zip**". Utilisez un autre format de compression uniquement s'il vous est impossible de le faire en format ".zip".
- **Remettez le dossier compressé par LÉA**.

Bon travail et bonne fin de session !