Le Noir, le Blanc, le Gris et les Couleurs

(Semaines du 27 novembre au 8 décembre)

Une image sera codée avec trois vecteurs à deux dimensions, chaque vecteur 2D codera une composante couleur de l'image. Chaque case de ces vecteurs à deux dimensions codera une couleur d'un pixel. Chaque pixel a 3 composantes : Rouge, Vert et Bleu. Chaque composante est un entier entre 0 et 255 et indique la quantité de cette couleur dans le pixel. Un pixel (0, 0, 0) est noir car il n'y a aucune couleur, un pixel (255, 0, 0) est rouge car il y a tout le rouge possible et aucune autre couleur.
Quelle est la couleur d'un pixel à (255, 255, 255) ? d'un pixel à (0, 255, 0) ? d'un pixel à (0, 255, 255) ? d'un pixel à (127, 127, 127) ?

Par exemple, l'image suivante est représentée par les trois vecteurs 2D qui suivent, tous des vecteurs2D de taille 4x2, avec chaque carré représentant un pixel. Les données sont stockées ligne par ligne.

rouge	**vert**	**bleu**	**bleu ciel**
jaune	**blanc**	**noir**	**gris**

rouge

255	0	0	102
255	255	0	191

vert

0	255	0	204
255	255	0	191

bleu

0	0	255	255
0	255	0	191

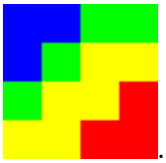
Pour représenter et manipuler une image dans votre programme, vous devez définir une classe Image avec comme attributs : 3 vecteurs d'entiers à deux dimensions nommés `rouge`, `vert` et `bleu` stockant la valeur de la composante respective de chaque pixel, deux entiers `largeur` et `longueur` stockant les dimensions de l'image.

Image
rouge : vecteur 2D d'entiers vert : vecteur 2D d'entiers bleu : vecteur 2D d'entiers longueur : entier largeur : entier

1. (★) Ecrire la déclaration de la classe `Image`.

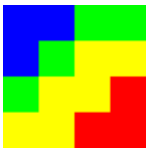
2. (★) Ecrire un constructeur permettant d'initialiser un objet de la classe `Image` à partir de trois vecteurs 2D d'entiers. Si les 3 vecteurs 2D n'ont pas les mêmes dimensions il faudra lancer une exception.

Tester votre constructeur en créant des images de quelques pixels de côté. Ces images pourront servir pour tester vos méthodes plus tard.

Construire un objet de la classe `Image` pour représenter l'image  .
(voir l'annexe à la fin de ce fichier)

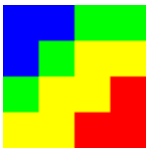
Pour vérifier le contenu de votre objet `image`, vous pourrez écrire des méthodes pour afficher le vecteur2D de chacune des trois composantes et une méthode pour récupérer les valeurs pour un pixel donné.

3. (★) Ecrire une méthode `composanteRouge` de la classe Image permettant de créer une nouvelle image à partir de l'image cible qui a la même composante Rouge et les autres composantes à zéro.

   Exemple : On passe de l'image en  à l'image 

4. (★) Ecrire une méthode `detection` qui prend en paramètre trois entiers représentant le code RVB d'une couleur et qui retourne vrai si il existe un pixel de cette couleur dans l'`image` cible.

5. (★) Ecrire une méthode `niveauxGris` de la classe Image permettant de créer une nouvelle image à partir de l'image cible dont chaque pixel est gris. Un pixel gris est un pixel dont les composantes Rouge, Vert et Bleu sont égales. Pour obtenir le niveau de gris d'un pixel, on fait la moyenne des trois composantes.

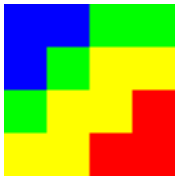
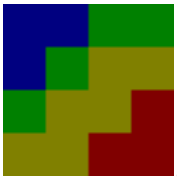
   Exemple : On passe de l'image en  à l'image 

6. (★★★★) Ecrire des méthodes `visionDeuteranopie`, `visionProtanopie` et `visionTritanopie` qui permettent de rendre compte de ce que voient les gens atteints des différentes formes de daltonisme.

7. (★) Ecrire une méthode `noirEtBlanc` de la classe Image permettant de créer une nouvelle image à partir de l'image cible dont chaque pixel est ou noir, ou blanc.
   On peut partir de l'image en niveaux de gris, et si le niveau de gris est plus clair qu'un seuil passé en paramètre le pixel est blanc et si il est plus sombre que le seuil le pixel devient noir.

8. (★★) Ecrire une méthode `histogrammeGris` retournant un vecteur d'entiers représentant l'histogramme de l'image en niveau de gris. Un niveau de gris est une valeur comprise entre 0 et 255. Cet histogramme peut être représenté en `C++` en utilisant un vecteur : dans la case d'indice i on inscrit le nombre de pixels dont le niveau de gris est i.

9. (★★★) Ecrire une méthode `histogrammeCouleur` retournant un vecteur de dimension 3 d'entiers représentant l'histogramme de l'image en couleurs. Une couleur est un triplet de valeurs chacune entre 0 et 255. Vous utiliserez donc des vecteurs 3D pour faire cette question.

10. (★★) Ecrire des méthodes `luminosityUp` et `luminosityDown` de la classe Image permettant de créer une nouvelle image à partir de l'image cible dont la luminosité a été augmentée ou diminuée. Pour augmenter (resp. diminuer) la luminosité, il faut multiplier toutes les composantes par une même valeur supérieure à 1 (resp. inférieure à 1).
Attention : les valeurs des composantes doivent rester entre 0 et 255 inclus.
Exemple : si on diminue la luminosité d'un facteur de 0.5 des 3 pixels (0,127,138) (255,200,127) (0,64,120), on obtient les pixels (0,63,79) (127,100,63) (0,32,60). Vérifiez les calculs.
L'image  avec une luminosité de moitié devient  .

11. (★★) Ecrire des méthodes `contrasteUp` et `contrasteDown` de la classe Image permettant de créer une nouvelle image à partir de l'image cible dont le contraste a été augmenté ou diminué. Pour augmenter (resp. diminuer) le contraste d'un pixel, on va multiplier la distance de chaque composante à 128 par une même valeur supérieure à 1 (resp. inférieure à 1).
Attention : les valeurs des composantes doivent rester entre 0 et 255 inclus.
Exemple : on augmente le contraste d'un facteur de 2 des 3 pixels suivants :
(0,128,138) (255,200,128) (0,64,120)
On obtient les pixels : (0,128,148) (255,255,128) (0,0,112).
Vérifiez ces valeurs en effectuant les calculs à la main.

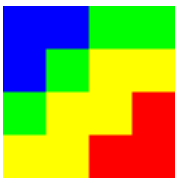
12. (★★★) Ecrire une méthode `reglageAuto` de la classe Image permettant de créer une nouvelle image à partir de l'image cible et qui effectue un réglage automatique de l'image de la façon suivante :

    - La luminosité est changée de façon à avoir une luminosité moyenne de l'ensemble des pixels de 128.
    - Le contraste est augmenté autant que possible sans que la valeur d'une composante d'un pixel de l'image dépasse 255 ou 0.

13. (★★★) Ecrire une méthode `reglageAutoGris` qui utilise l'histogramme de niveau de gris pour proposer un réglage automatique de l'image en niveau de gris. Il s'agit d'étaler les niveaux de gris de façon à utiliser l'ensemble de la palette des gris, de noir (0,0,0) à blanc (255,255,255).
Pour chaque niveau de gris $k = 0,1,..,255$, on note n_k le nombre de pixel qui ont le niveau de gris k, n le nombre total de pixel et L le nombre de niveau de gris effectivement utilisé. La nouvelle valeur du niveau de gris d'un pixel de niveau k est donnée par la formule $\frac{(L-1)}{n}\sum_{i=0}^{k} n_i$.

14. (★★★★) Ecrire une méthode `réglageAutoCouleur` qui utilise l'histogramme des couleurs pour étaler les couleurs afin d'améliorer le contraste et la luminosité.

Annexe

L'image  ,utilisée dans de nombreux exemples, est représentée par les trois vecteurs2D suivants :

Rouge

0	0	0	0
0	0	255	255
0	255	255	255
255	255	255	255

Vert

0	0	255	255
0	255	255	255
255	255	255	0
255	255	0	0

Bleu

255	255	0	0
255	0	0	0
0	0	0	0
0	0	0	0