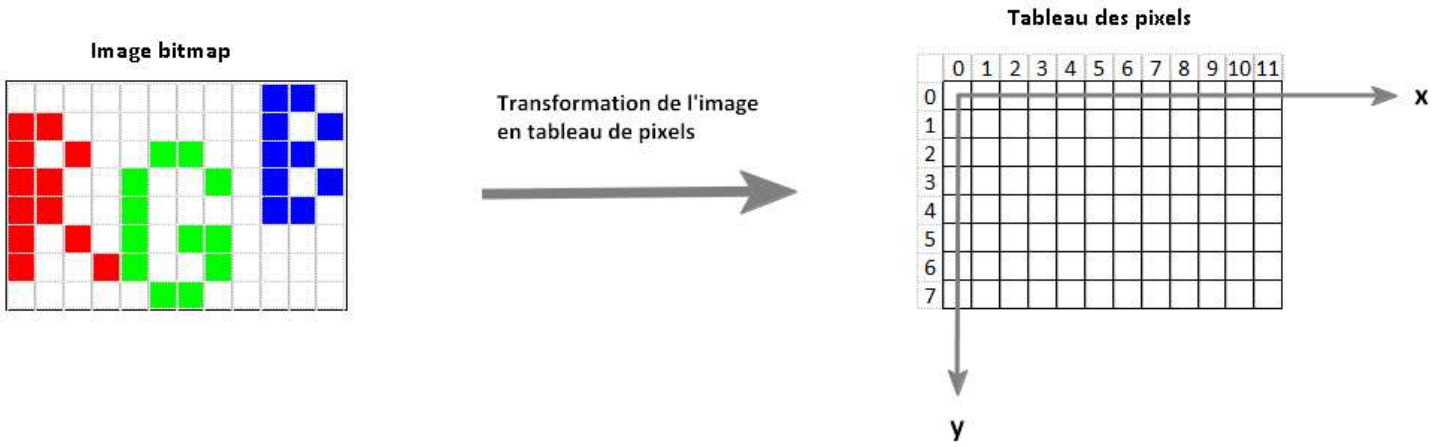
IMAGE

Une image est considérée comme un tableau de dimension *largeur* × *hauteur* contenant le code couleur de chaque pixel de l'image.

Chaque pixel est repéré par ses coordonnées (i ; j) (cf figure ci-dessous). Le pixel en haut à gauche de l'image a pour coordonnées (0;0), le pixel en haut à droite (largeur-1 ;0), le pixel en bas à gauche (0; hauteur-1).



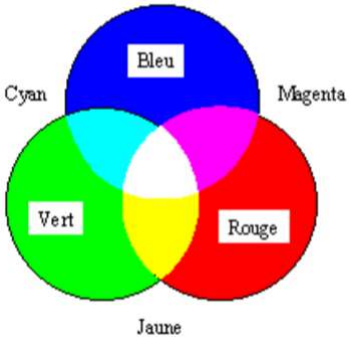
Codage RGB de la couleur

Il utilise la synthèse additive des couleurs.

Le code couleur RGB est composé de 3 composantes: le rouge, le vert et le bleu.

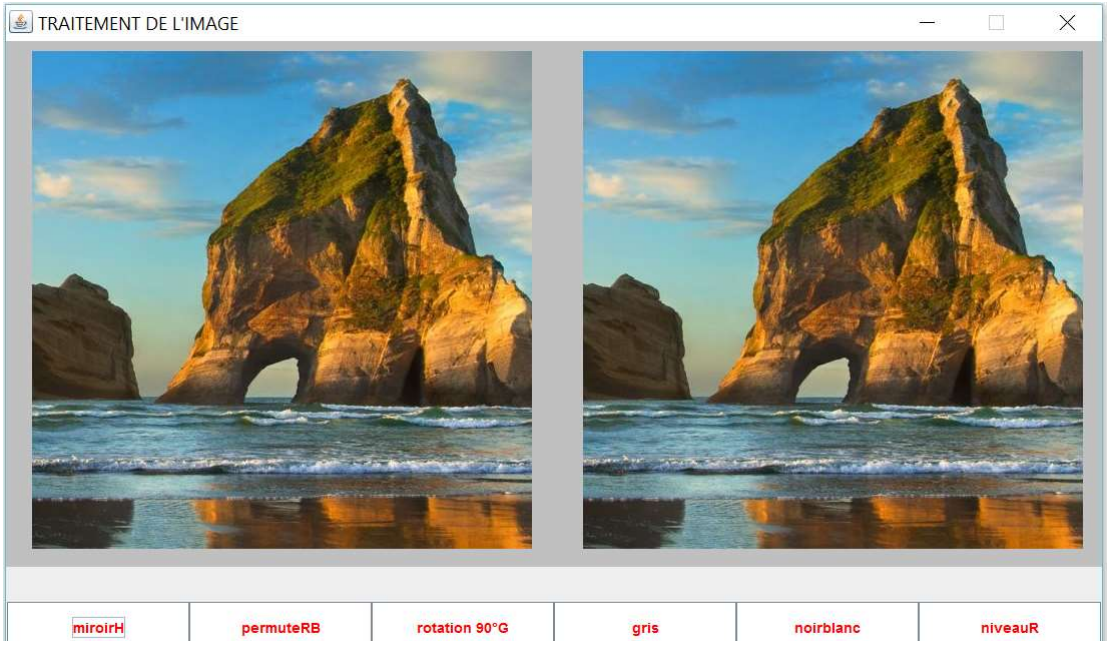
Chaque composante a une valeur entière comprise entre 0 et 255 soit 256 niveaux.

(0: pas de couleur, 255: intensité maximale)



Couleur	Rouge	Vert	Bleu	Magenta	Jaune	Cyan	Blanc	Noir
Code RGB	255,0,0	0,255,0	0,0,255	255,0,255	255,255,0	0,255,255	255,255,255	0,0,0

OBJECTIF: CREER L'APPLICATION SUIVANTE QUI EFFECTUE DIFFERENTS TRAITEMENTS SUR DES IMAGES



BUFFEREDIMAGE

BufferedImage est une image tampon ie une copie de l'image que l'on veut modifier. C'est un tableau contenant la couleur de chaque pixel.

METHODES POUR UNE BUFFEREDIMAGE

Imag=**BufferedImage(largeur_image,hauteur_image,BufferedImage.*TYPE_INT_RGB*)** définit une image tampon de type RGB nommée *imag*.

Imag.**getWidth()** donne la largeur de l'image *imag.*

Imag.**getHeight()** donne la hauteur de l'image *imag.*

imag.**setRGB(**i, j, rgb**)** Le pixel de coordonnées (i ; j) de *imag* prend la couleur dont le code RGB est rgb. (i, j, rgb entiers)

Imag.**getRGB(**i,j**)** donne le code RGB de la couleur du pixel de coordonnées (i ; j) de *imag*.

METHODES POUR MODIFIER LE CODE COULEUR RGB

r= **getRed(** RGB**)** donne la composante rouge du code couleur RGB avec r entier et $0 \leq r \leq 255$

g= **getGreen(** RGB**)** donne la composante verte du code couleur RGB avec g entier et $0 \leq g \leq 255$

b= **getBlue(** RGB**)** donne la composante bleue du code couleur RGB avec b entier et $0 \leq b \leq 255$

rgb= **makeRGB(**r**,** g**,** b**)** crée le code RGB à partir des valeurs des 3 composantes r, g et b. rgb entier

Exemples:

makeRGB(255, 0, 0) donne le rouge

makeRGB(0, 255, 0) donne le vert

makeRGB(0, 0, 255) donne le bleu

makeRGB(0, 0, 0) donne le noir

makeRGB(255, 255, 255) donne le blanc

makeRGB(*r*, 0, 0) donne une nuance de rouge

makeRGB(255, 0, 255) donne le magenta

makeRGB(255, 255, 0) donne le jaune

makeRGB(0, 255, 255) donne le cyan

makeRGB(*r,g,b*) donne une nuance de gris si *r = g = b*

LE PROGRAMME

Dans le programme ApplicationImage, vous avez ***uniquement*** ces 3 parties du programme à modifier:

La partie 1 (**ligne 7**) concerne l'ajout de boutons , la partie 2 (**ligne 15**) la gestion des actions des boutons et la partie 3 (**ligne 43**) le traitement de l'image.

```
#*********** PARTIE 1  *********** PARTIE 1  *********** PARTIE 1  *********** PARTIE 1*
msg=[ "miroirH" , "permuteRB" ] <- on ajoute les noms des nouveaux boutons

filename= "SNT/Traitement_image/falaise.png" <- on ajoute # devant filename pour enlever l'image carrée
#filename="SNT/Traitement_image/bauhaus.png" <- on enlève # pour avoir une image rectangulaire
dimX=5 # dimension du tableau des boutons  <- on change le nombre de boutons
dimY=2
#*********** FIN PARTIE 1  *********** FIN PARTIE 1  *********** FIN PARTIE 1  ***********



#*********** PARTIE 2  *********** PARTIE 2  *********** PARTIE 2  *********** PARTIE 2***********
# gestion des différents traitement de l'image
def traitement(i):# associer le n° du bouton à son traitement
  global imgF
  i=int(i)
  if i==0: # bouton n°1
    miroirH()
  elif i==1: # bouton n°2
   permuteRB()
  elif i==2: # bouton n°3
   son traitement


#*********** FIN PARTIE 2  *********** FIN PARTIE 2 *********** FIN PARTIE 2  *********** FIN PARTIE 2



#*********** PARTIE 3  *********** PARTIE 3  *********** PARTIE 3  *********** PARTIE 3
def permuteRB():
  global imgO,imgF,largeur,hauteur
  imgF=BufferedImage(largeur,hauteur,BufferedImage.TYPE_INT_RGB)#déclaration des dimensions
de l'image résultat et du type de codage de la couleur

  for i in range(largeur):
   for j in range (hauteur):
      rgb=imgO.getRGB(i, j) # récupération du code rgb du pixel (i,j)
      r=getRed(rgb) # récupération de la compsante rouge du pixel (i,j)
      g=getGreen(rgb) # récupération de la composante verte du pixel (i,j)
      b=getBlue(rgb) # récupération de la composante bleue du pixel (i,j)
      ImgF.setRGB(i,j,makeRGB(b,g,r)) # le pixel(i,j) prend la couleur créée avec makeRGB
```

Pour créer un nouveau traitement, il suffit de copier permuteRB et de modifier ce qui est nécessaire.

```
#*********** FIN PARTIE 3  *********** FIN PARTIE 3 *********** FIN PARTIE 3  *********** FIN PARTIE 3
```

TOUTE MODIFICATION EN DEHORS DE CES 3 PARTIES ENTRAINERA L'ECHEC DU PROGRAMME.

Déplacement de la couleur

Etape 1:

Récupération des coordonnées du pixel de l'image originale

Etape 2:

Récupération du code RGB de ce pixel

Etape 3:

Affectation du code RGB au pixel de l'image finale

Exemple: ***Effet du miroir horizontal***

imageOriginale

N° des lignes \ N° des colonnes (i / j)	0	1	2	3	4
0	W		W	W	R
1	G	G	G	R	W
2	W	G	W	B	B
3	W	G	W	Y	G

Image originale de dimension 5x4

⇒

imageFinale

N° des lignes \ N° des colonnes (i / j)	0	1	2	3	4
0	R	W	W		W
1	W	R	G	G	G
2	B	B	W	G	W
3	G	Y	W	G	W

Image finale de dimension 5x4

Compléter les deux tableaux.

	largeur	**hauteur**
image originale		
image finale		

image originale		**image finale**	
pixel (,)	⇒	pixel (0,0)	**1ère colonne**
pixel (,)	⇒	pixel (0,1)	
pixel (,)	⇒	pixel (0,2)	
pixel (,)	⇒	pixel (0,3)	
pixel (,)	⇒	pixel (1,0)	**2ème colonne**
pixel (,)	⇒	pixel (1,1)	
pixel (,)	⇒	pixel (1,2)	
pixel (,)	⇒	pixel (1,3)	

Quelles observations peut-on faire?

En déduire la relation entre les pixels de deux images.

	image originale	⇒	**image finale**
pixel (	,)	⇒	pixel (i , j)

Modification de la couleur

Etape 1:

Récupération du code RGB du pixel

Etape 2:

Extraction des 3 composantes du code RGB

Etape 3:

Création du nouveau code RGB

Etape 4:

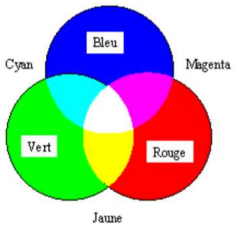
Affectation du nouveau code RGB au pixel.

Exemple: ***<u>Permutation des composantes Rouge et Bleu</u>***

A partir du code RGB de la couleur originale et de la synthèse additive de couleurs, déterminer le nouveau code RGB et sa couleur.

Couleur originale	**Rouge**	Vert	**Bleu**	Magenta	**Jaune**	**Cyan**	Blanc	Noir
Code RGB original	255,0,0	0,255,0	0,0,255	255,0,255	255,255,0	0,255,255	255,255,255	0,0,0

⇓



Code RGB final	0,0,255							
Couleur finale	**Bleu**							

Cette permutation appliquée à l'image de gauche ci-dessous donne celle de droite.

N° des lignes \ N° des colonnes (i / j)	0	1	2	3	4
0	C		W	W	R
1	G	G	G	R	W
2	C	G	M	B	B
3	Y	G	M	Y	G

Image originale de dimension 5x4

⇒

N° des lignes \ N° des colonnes (i / j)	0	1	2	3	4
0	Y		W	W	R
1	G	G	G	R	W
2	Y	G	M	B	B
3	C	G	M	C	G

Image finale de dimension 5x4

Après avoir analyser les deux méthodes précédentes, créer les méthodes suivantes:

- miroirV: miroir vertical
- filtreR: filtre rouge (On ne conserve que la composante rouge ie g=b=0)
- renforceR: renforce la composante rouge r avec la contrainte suivante:

si 0<r<205 alors r=r+50 sinon r=255.

Pour aller plus loin:

- Qui permet de permuter le vert et le bleu
- Qui permet d'obtenir le négatif d'une image. (r= 255-r, g= 255-g et b=255-b)
- Qui permet d'obtenir une image en niveaux de gris en utilisant la règle suivante:

r= (r+g+b)/3, g= (r+g+b)/3 et b=(r+g+b)/3.

- une rotation de 180° de l'image.
- Qui coupe verticalement l'image en deux parties et qui permute la partie droite et la partie gauche de l'image.
- Qui permet une rotation de 90° gauche.
- Qui permet de permuter le rouge et le vert.
- Qui remplace la couleur d'un pixel par sa composante dominante.

Par exemple: si r=100, g=102 et b=56 alors r=0, g=255, b=0.

- Qui permet d'obtenir une image en rouge et noir en utilisant la règle suivante:

si r>125 alors r=255, g=0 et b=0 sinon rgb=0.

- Qui permet d'obtenir une image en noir et blanc en utilisant la règle suivante:

si r+g+b>765/2 alors rgb=255 sinon rgb=0