

TRAITEMENT D'UNE IMAGE

Algorithmique

Bachir KARI

27/01/2020

SOMMAIRE

Page 2 – PROBLEMATIQUE

Page 3 – ACCENTUATION DU CONTRASTE

Page 5 – FONCTION MOYENNE

Page 6 – DIMINUTION DU CONTRASTE

Page 7 – INVERSION DES COULEURS

Page 8 – CONCLUSION

PROBLEMATIQUE

Une image en nuances de gris est représentée par un tableau de cases modélisé par une matrice 5 x 6 dont les éléments sont des entiers compris entre 0 à 100 appelés **saturation**.

Une saturation de 0 correspond à une case noire et une saturation de 100 correspond à une case blanche.

On cherche à implémenter 3 fonctions portant sur ces matrices :

- Une fonction permettant d'**accentuer le contraste** d'une image donnée , c'est-à-dire qu'une case claire devient plus claire et une case foncée devient plus foncée.
- Une fonction permettant de **diminuer le contraste**, appliquant l'effet inverse.
- Une fonction permettant **d'inverser la saturation** de chaque case.

IMPLEMENTATIONS

ACCENTUATION DU CONTRASTE

L'accentuation du contraste s'effectuera en appliquant le processus suivant sur chaque case :

Si la saturation est supérieure à 50 et inférieure ou égale à 75 , alors elle devient 75

Si la saturation est supérieure à 75 , alors elle devient 100

Si la saturation est inférieure à 50, alors elle est divisée par 2 (arrondi à l'entier inférieur)

Fonction **ACCENTUER_CONTRASTE** (tableau de ENTIER : Matrice[][]) : tableau de ENTIER

CONSTANTES

ENTIER : NL <- 5

ENTIER : NC <- 6

VARIABLES

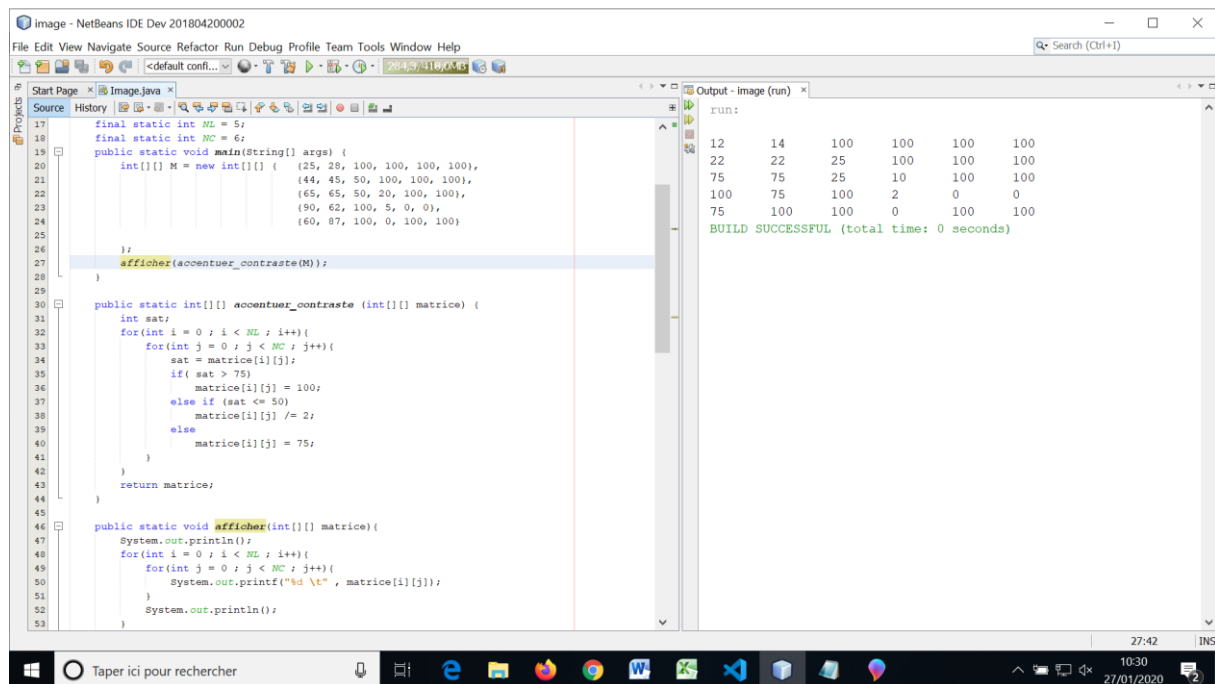
ENTIER : i, j, Sat

DEBUT

```
POUR i ALLANT_DE 0 à NL – 1 FAIRE
    POUR j ALLANT_DE 0 à NC – 1 FAIRE
        Sat <- Matrice[i][j]
        SI Sat > 75 ALORS
            Matrice[i][j] <- 100
        SINON_SI Sat <= 50 ALORS
            Matrice[i][j] <- Sat / 2
        SINON
            Matrice[i][j] <- 75
        FIN_SI
    FIN_POUR
FIN_POUR
RETOURNE Matrice ;
```

FIN

Implémentation en Java ci-dessous - en prenant un jeu de test permettant de tester le bon fonctionnement des 3 branches de notre structure alternative , c'est-à-dire au moins une saturation inférieure à 50 , au moins une saturation supérieure à 75 et au moins une comprise entre 50 et 75.



DIMINUTION DU CONTRASTE

Afin de diminuer le contraste d'une image, on décide de « tirer » les saturations de chaque case vers la saturation moyenne de l'image.

On implémente donc tout d'abord une fonction MOYENNE qui calcule la saturation moyenne d'une image donnée :

Fonction **MOYENNE**(tableau de ENTIER : Matrice[][]) : REEL

CONSTANTES

ENTIER : NL <- 5

ENTIER : NC <- 6

VARIABLES

ENTIER : i, j, somme

REEL : moyenne

DEBUT

Somme <- 0

POUR i ALLANT_DE 0 à NL – 1 FAIRE

POUR j ALLANT_DE 0 à NC – 1 FAIRE

somme <- somme + Matrice[i][j]

FIN_POUR

FIN_POUR

Moyenne <- somme / (NL * NC)

RETOURNE moyenne

FIN

```
image - NetBeans IDE Dev 201804200002
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Start Page x Image.java x
Source History
7
8 /**
9  *
10  * @author formation_gep
11  */
12 public class Image {
13
14     /**
15      * @param args the command line arguments
16      */
17     final static int NL = 5;
18     final static int NC = 6;
19     public static void main(String[] args) {
20         int[][] M = new int[][] {
21             {25, 28, 100, 100, 100, 100},
22             {44, 45, 50, 100, 100, 100},
23             {65, 65, 50, 20, 100, 100},
24             {90, 62, 100, 5, 0, 0},
25             {60, 87, 100, 0, 100, 100}
26         };
27
28         System.out.printf("Moyenne : %.2f\n", moyenne(M));
29     }
30
31     public static double moyenne(int[][] matrice){
32         int somme = 0;
33         for(int i = 0; i < NL; i++){
34             for(int j = 0; j < NC; j++){
35                 somme += matrice[i][j];
36             }
37         }
38         return (double) somme / (NL * NC);
39     }
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
Output - image (run) x
run:
Moyenne : 66,53
BUILD SUCCESSFUL (total time: 0 seconds)
```

On peut maintenant implémenter la fonction permettant de diminuer le contraste :

FONCTION DIMINUER_CONTRASTE (tableau de ENTIER : Matrice[][]) : tableau de ENTIER

CONSTANTES

ENTIER : NL <- 5

ENTIER : NC <- 6

VARIABLES

REEL : moy

ENTIER : i, j

DEBUT

moy <- MOYENNE(Matrice)

POUR i ALLANT_DE 0 à NL – 1 FAIRE

POUR j ALLANT_DE 0 à NC – 1 FAIRE

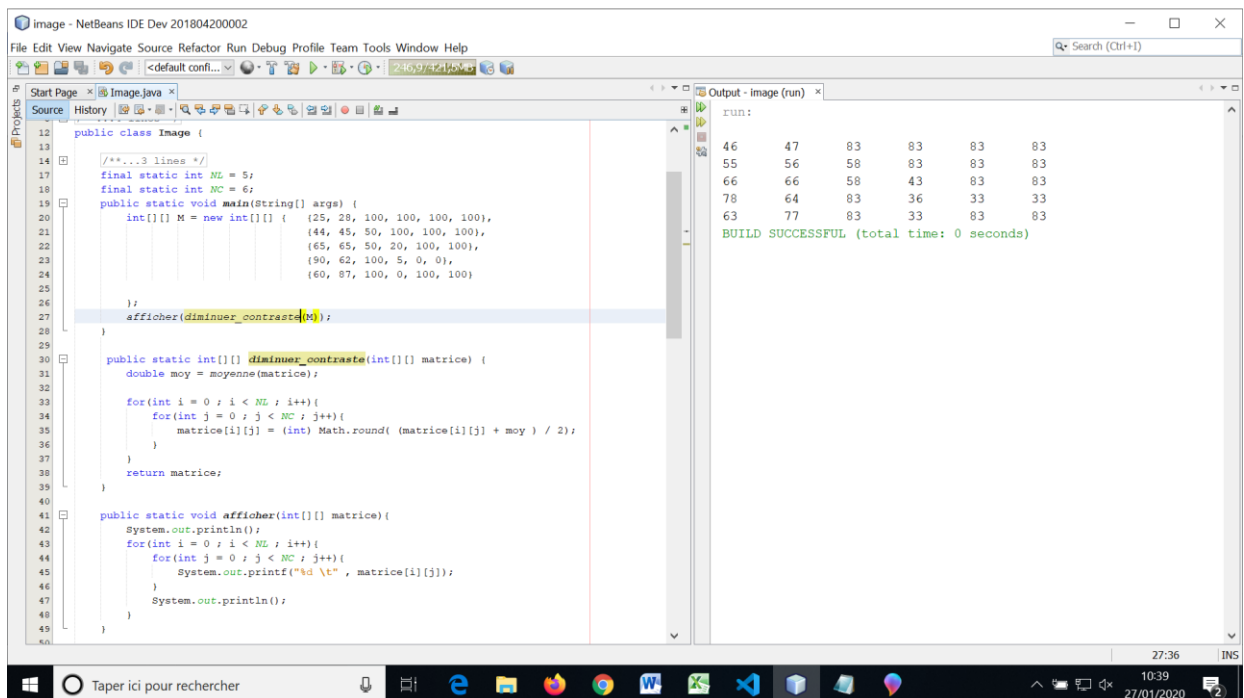
Matrice[i][j] = (Matrice[i][j] + moy) / 2

FIN_POUR

FIN_POUR

RETOURNE Matrice

FIN



```
public class Image {
    /**...3 lines */
    final static int NL = 5;
    final static int NC = 6;
    public static void main(String[] args) {
        int[][] M = new int[][] {
            {25, 28, 100, 100, 100, 100},
            {44, 45, 50, 100, 100, 100},
            {65, 65, 50, 20, 100, 100},
            {90, 62, 100, 5, 0, 0},
            {60, 87, 100, 0, 100, 100}
        };
        afficher(diminuer_contraste(M));
    }

    public static int[][] diminuer_contraste(int[][] matrice) {
        double moy = moyenne(matrice);
        for(int i = 0 ; i < NL ; i++){
            for(int j = 0 ; j < NC ; j++){
                matrice[i][j] = (int) Math.round( (matrice[i][j] + moy) / 2);
            }
        }
        return matrice;
    }

    public static void afficher(int[][] matrice){
        System.out.println();
        for(int i = 0 ; i < NL ; i++){
            for(int j = 0 ; j < NC ; j++){
                System.out.printf("%d\t", matrice[i][j]);
            }
            System.out.println();
        }
    }
}
```

run:

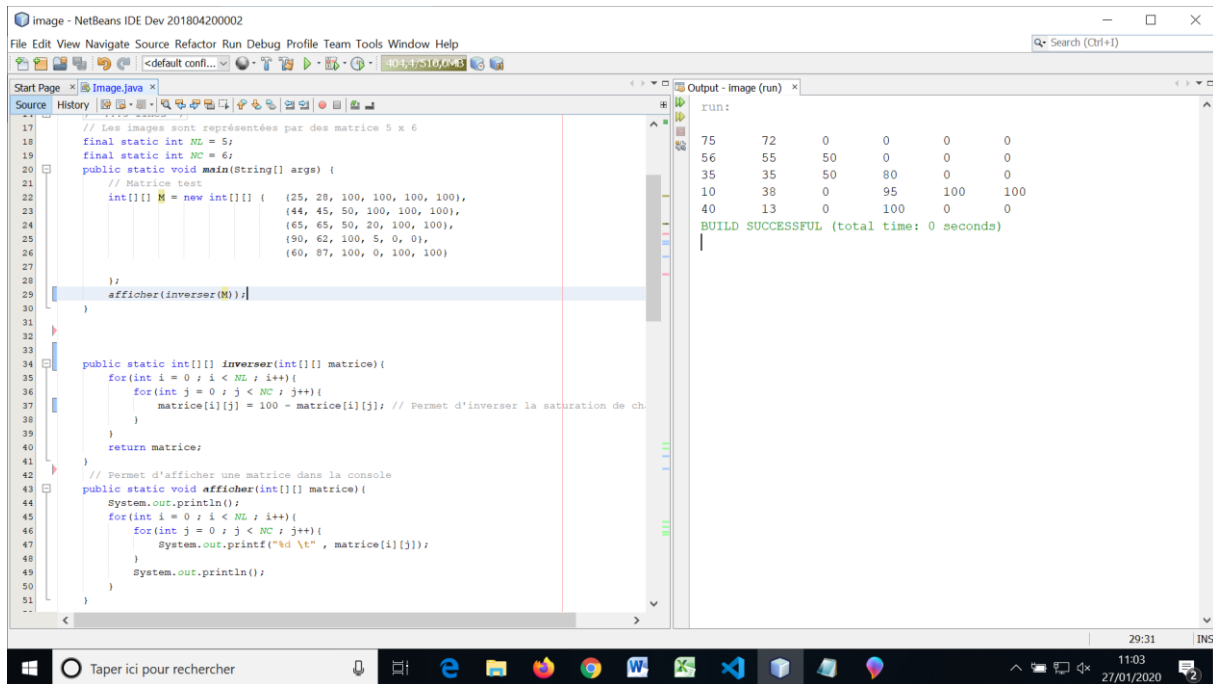
46	47	83	83	83	83
55	56	58	83	83	83
66	66	58	43	83	83
78	64	83	36	33	33
63	77	83	33	83	83

BUILD SUCCESSFUL (total time: 0 seconds)

Il n'y a aucune structure alternative cette fois , donc un seul jeu de test quelconque devrait suffire à vérifier le bon fonctionnement du programme.

INVERSION DES COULEURS

Implémentation en Java de la fonction permettant d'inverser la saturation de chaque case de l'image – une case blanche doit devenir noire et une case noire doit devenir blanche.



The screenshot shows the NetBeans IDE with a Java file named 'Image.java'. The code defines a 5x6 matrix and a function to invert its saturation. The output window shows the resulting matrix values.

```
// Les images sont représentées par des matrices 5 x 6
final static int NL = 5;
final static int NC = 6;
public static void main(String[] args) {
    // Matrice test
    int[][] M = new int[NL][NC] {
        {25, 28, 100, 100, 100, 100},
        {44, 45, 50, 100, 100, 100},
        {65, 65, 50, 20, 100, 100},
        {90, 62, 100, 5, 0, 0},
        {60, 87, 100, 0, 100, 100}
    };

    afficher(inverser(M));
}

public static int[][] inverser(int[][] matrice) {
    for(int i = 0 ; i < NL ; i++) {
        for(int j = 0 ; j < NC ; j++) {
            matrice[i][j] = 100 - matrice[i][j]; // Permet d'inverser la saturation de ch.
        }
    }
    return matrice;
}

// Permet d'afficher une matrice dans la console
public static void afficher(int[][] matrice) {
    System.out.println();
    for(int i = 0 ; i < NL ; i++) {
        for(int j = 0 ; j < NC ; j++) {
            System.out.printf("%d\t", matrice[i][j]);
        }
        System.out.println();
    }
}
```

Output - image (run):

```
run:
75    72    0    0    0    0
56    55    50    0    0    0
35    35    50    80    0    0
10    38    0    95    100    100
40    13    0    100    0    0
BUILD SUCCESSFUL (total time: 0 seconds)
```

CONCLUSION

Une image numérique peut être représentée par une grande matrice de pixels dont les valeurs représentent la colorisation de chaque pixel.

Les fonctions que nous avons établies peuvent donc très bien s'appliquer à plus grande échelle, au traitement de véritables images.