Sae traitement d'images

1. Mise en oeuvre de notre solution personnelle pour le traitement d'images

1.1 Fonctionnement de notre solution

Notre solution est une solution qui ne nécessite pas de longs temps de traitements, elle ne parcourt l'image qu'une seule fois, se contente de recenser les pixels ayant des couleurs proches les unes des autres et de toutes les regrouper dans un même groupe. Le nombre de groupes créés dépend du nombre de couleurs présentes dans l'image ainsi que d'une variable "tolérance" qui permet de choisir la distance maximale entre deux couleurs pour qu'elles soient regroupées dans un même groupe. Plus la tolérance est grande, plus le nombre de groupes sera petit, permettant un traitement plus rapide, mais une image moins fidèle à l'originale, parfaite pour les nombres de couleurs faibles. A l'inverse, plus la tolérance est faible, plus le nombre de groupes sera grand, permettant un traitement plus long, mais une image plus fidèle à l'originale, parfait pour les nombres de couleurs élevés.

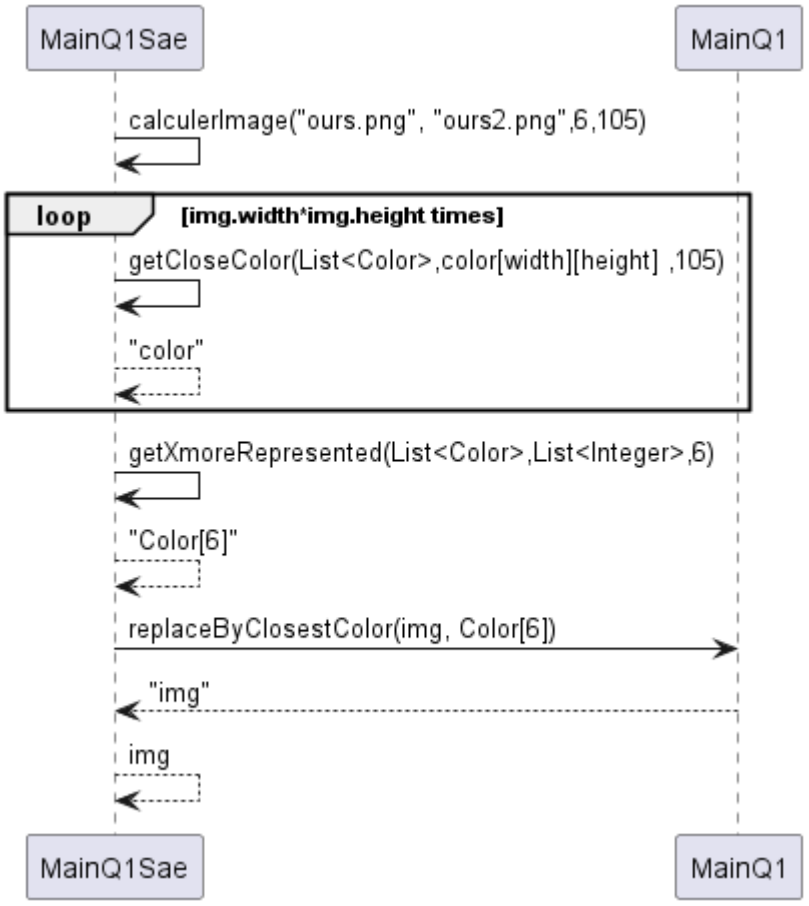
1.2 algorithme de notre solution

```
entrées : n>0 : nombre de couleurs
entrées : t>0 : tolerance
entrées : D{di} : données
resultat : C{ci} : couleurs

*/ parcours des données et affectation des couleurs
pour i de 1 à n faire
    pour j de 1 a m faire
        si distance(D{i}, C2{j}.donnée < tolerance alors
            C2{j}.nombre = C2{j}.nombre + 1
        sinon
            C2.ajouter(D{i})
            C2{C.taille}.nombre = 1
        fin si
    fin pour
fin pour

    */ tri des couleurs par ordre décroissant
    pour i de 1 à C2.taille faire
        pour j de i+1 à C2.taille faire
            si C{i}.donnée < C2{j}.donnée alors
                temp = C2{i}
                C2{i} = C2{j}
                C2{j} = temp
            fin si
        fin pour
    fin pour

        */ affectation des couleurs
        pour i de 1 à n faire
            C{n}.couleur = C2{n}.couleur
        fin pour
rendre C
```

1.3 conception de la solution

Cette solution ne comporte qu'une seule nouvelle classe et utilise la méthode du tp3 pour remplacer les pixels par la couleur la plus proche contenue dans un tableau de couleurs.

LA méthode main appel à la suite toutes les méthodes de la classe afin de calculer les meilleurs couleurs a utiliser pour remplacer toutes celles de l'image



Une fois l'image calculée elle est écrite dans le fichier passé en paramètre du programme

1.4 lancement de notre solution

Notre solution se présente sous la forme d'une classe "MainQ1Sae" qui contient une méthode "main" qui permet de lancer le traitement. Elle prend comme arguments le chemin de l'image à traiter, le chemin de l'image de sortie, le nombre de couleurs, et la tolérance.

A la fin de l'exécution, les temps de lecture, traitement et écriture sont donnés

```
temps de lecture: 71
temps de calcul: 151
temps d'ecriture: 31
temps total: 253
```

Tous les paramètres ont des valeurs par défaut permettant de dessiner l'ours en 32 couleurs avec une tolérance de 37

1.5 Améliorations possibles

Comme améliorations possibles on peut déjà penser à un calcul automatique de la tolérance optimal pour chaque image et chaques nombre de couleurs, ça n'a pas été mis en place dans le code car ceci ralentissait fortement l'exécution du traitement sans pour autant donner de bien meilleurs résultats que ce que l'on obtiendrait en testant 2 ou 3 tolérances manuellement ce qui prendrais bien moins de temps étant donné la rapidité d'exécution de cet algorithme

2. Mise en oeuvre de la solution proposée dans la SAE :

2.1. Principe de KMeans :

KMeans est un algorithme qui consiste à placer un nombre arbitraire n de points nommés centroïdes dans un plan de dimension nd. Il va ensuite déterminer quel est le centroïde le plus proche de chaque point du plan et les stocker dans un "cluster" qui contient également le centroïde en question. Il va ensuite placer le centroïde au centre de son cluster et recommencer ces deux étapes jusqu'à ce que la condition d'arrêt soit remplie, dans notre code, celle-ci est quand le coût moyen du cluster augmente, ce qui n'arrive que quand les résultats d'une itération à l'autre ne changent pas beaucoup.

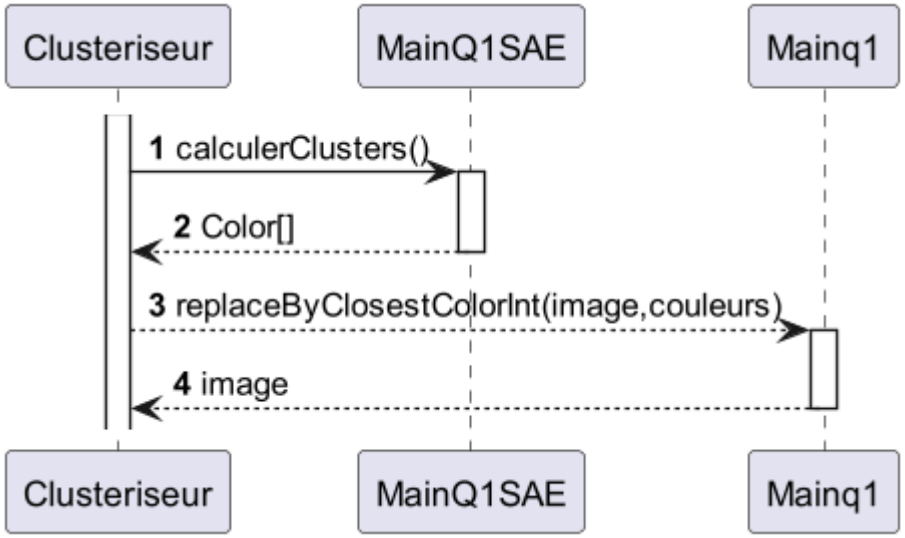
2.2. Algorithme KMeans :

Algorithme 1 : K-Means

Entrées : $n_g \geq 0$ nombre de groupes
Entrées : $D = \{d_i\}_i$ données
Résultat : Centroïdes mis à jour

```
   /* Initialisation centroïdes
1  pour i ∈ [0, n_g] faire
2      c_i ← random(D)
   /* Boucle principale
3  tant que (non(fini)) faire
       /* Initialisation Groupes
4      pour i ∈ [0, n_g] faire
5          G_i ← ∅
       /* Construction des Groupes
6      pour d ∈ D faire
7          k ← indiceCentroidePlusProche(d, {c_i}_i)
8          G_k ← G_k ∪ d
       /* Mise à jour des centroïdes
9      pour i ∈ [0, n_g] faire
10         c_i ← barycentre(G_i)
```

2.3 Conception :



2.4. Comment lancer cette solution :

Compiler le fichier CluteriseurDeCouleur.java, l'exécuter avec les paramètres, tous optionnels suivants nbCouleurs inputFile outputFile et le booléen “better” ce qui donne l’exemple suivant:
“java CluteriseurDeCouleur 64 ../images_diverses_small/animaux/poulpe.png ../imgCompressees/copie.jpg false”
(les paramètres de l’exemple sont ceux par défaut si vous n’en mettez pas)
Le programme vous donnera le temps de calcul à la fin:

```
Temps d'execution : 4232ms

Process finished with exit code 0
```

Note: on peut également appeler la méthode statique calculerImage de la classe avec seulement 2 premiers arguments vu que la méthode retourne l’image sans l’écrire.

Note2: le booléen better permet de passer d’une initialisation des clusters aléatoires à une initialisation par histogramme empruntée sur notre solution. On gagne ainsi en performances.

2.5. Pistes d’améliorations :

Le programme s’avère plutôt lent au vu du nombre d’itérations nécessaires, nous l’avons optimisé comme nous avons pu (notament en supprimant le facteur aléatoire) mais nous manquons probablement d’expérience ou de recul pour l’optimiser encore plus.

3 Tests

3.1 Base de tests utilisée

les Tests ont été réalisés sur l'image "ours.png", rendu avec 2, 4, 8, 16, 32 et toutes les puissance de 2 jusqu'à 4096 couleurs

3.2 Résultats obtenus

nb couleurs	temps 1(ms)	temps 2(ms)	distance 1	distance 2	t2/t1	d1/d2
2	152	506	2,784E+09	2,299E+09	3,3289474	1,210962
4	128	833	1,951E+09	763080222	6,5078125	2,5561297
8	104	1114	1,089E+09	417024079	10,711538	2,6106825
16	170	4037	449344703	211171727	23,747059	2,1278639
32	317	14513	270514209	118907352	45,782334	2,2749999
64	635	12655	158983756	77049145	19,929134	2,0634071
128	1589	13295	82861006	48411279	8,3668974	1,7116054
256	4914	37256	47901086	31004446	7,5816036	1,5449747
512	13582	69222	33189210	20688444	5,0965984	1,6042391
1024	27855	137625	25171336	13309551	4,9407647	1,8912235
2048	47063	264105	18018850	8621460	5,6117332	2,0899998
4096	81627	395623	13222053	5473259	4,8467174	2,415755
moyennes :					12,204262	2,0084869

3.3 analyse et conclusion sur les résultats

Nous nous attendions à voir des images plus rapides mais moins précises par notre algorithme que par celui proposé en cours, en effet notre algorithme itère une seul fois dans l'image et donne des couleurs représentatives a l'issu de cette iteration là où l'algorithme K-mean itère plusieurs fois dans l'image et déplace ses barycentre de façon les placés de façon le plus optimal possible et ne s'arrête que lorsqu'il cessent de bouger et ont donc de bonnes chances d'être optimales

Ces résultats montrent clairement que notre méthode est bien plus rapide pour donner des résultats (en moyenne 12X plus rapide) mais donne des résultats moins précis (en moyenne 2X plus éloignés de l'original), ce qui corresponds aux résultats attendus, en effet parcourire une unique fois l'image et donner des tendances général de couleurs représentatives de l'image est plus rapide que de faire de multiples itérations a travers les images a la recherche de barycentres optimaux

3.4 reproduire ces tests

Pour produire ces tests, il suffit de lancer le main de la classe MainTests.

3.5 : amélioration possibles

Pour Améliorer les résultats de nos tests, il serait possible de faire tourner les tests sur plus d’images ou a de plus hautes résolutions afin d’affiner les résultats et obtenir des moyennes sur plusieurs images