Rapport

Choix de structure:

```
typedef struct noeud{
    int pixel[4]; // contiendra la pixel moyen du noeud
    int hauteur; // contiendra la hauteur du noeud
    struct noeud *no; //les 4 fils du noeud
    struct noeud *ne;
    struct noeud *se;
    struct noeud *so;
    int x1,y1,x2,y2; // contient les coordonnées du noeud
    double erreur; // contiendra l'erreur du noeud
}Noeud, *Arbre;
```

Pour plus d'efficacité, nous avons choisi de donner en plus des variables logiques (les 4 fils et le tableau de pixel contenant les valeurs R,G,B,A) 6 variables.

2 vont correspondre aux coordonnées du coin en haut à gauche du noeud ou de la feuille, et 2 autres aux coordonnées du coin en bas à droite du noeud ou de la feuille.
Ces 4 variables serviront à pouvoir dessiner dans la fenêtre plus facilement mais également à utiliser les différentes fonctions d'approximation, de minimisation et de sauvegarde plus facilement, elles serviront à comparer 2 noeuds ou 2 feuille par exemple, ou à retrouver un noeud ou une feuille.

La variable hauteur quant à elle servira notamment aux fonctions de minimisation, elle sauvegardera la hauteur du noeud dans l'arbre, ainsi le premier noeud sera de hauteur 0, et ses 4 fils de hauteur (hauteur du père +1)=0+1=1.

La variable erreur elle contiendra le taux d'erreur du noeud, elle servira à savoir quel noeud diviser en 4 quadtree.

Parcours d'Arbre:

```
sauvegardeNB(pixels->no,fichier);
sauvegardeNB(pixels->ne,fichier);
sauvegardeNB(pixels->se,fichier);
sauvegardeNB(pixels->so,fichier);
```

La majorité des fonctions de ce projet sont de type récursive, il a fallu mettre en place un parcours d'arbre simple comme montré ci-dessus avec la fonction sauvegarde. Ce parcours d'arbre fonctionnera si lorsque l'on arrive à une feuille, on renvoie NULL.

Comment savoir que l'on est arrivé a une feuille?
Chaque noeud a obligatoirement 4 fils, pas 1, pas 2, pas 3 mais 4, tout le temps 4, ainsi, si un des fils est NULL les 4 le seront, c'est pourquoi pour vérifier si l'on est en présence d'une feuille, il suffit de poser la condition par exemple pour un arbre 'a' « if (a->no==NULL) ».

Les sauvegardes minimisées:

Pour effectuer des sauvegardes d'arbres minimisées comme demandées dans l'énoncé du projet, il faut parcourir en largeur l'arbre minimisé. Pour cela, on va créer un tableau d'Arbre et un Arbre a.
La premiere valeur du tableau sera le premier noeud de l'arbre et a pointera vers ce noeud.
Ensuite, la deuxième valeur de l'arbre sera son son fils Nord-Ouest (no), ainsi on obtiendra [a, a->no] pour l'instant.
Comme l'arbre a déjà été minimisé, si a->no et a->ne par exemple sont égaux, alors on aura déjà fait pointer a->ne vers a->no. Ainsi, lorsque l'on va arriver a a->ne dans, on va lancer l'appel a la fonction indice qui renverra l'indice du tableau correspondant à a->ne si a->ne pointe vers un des arbres déjà dans le tableau, indice renvoie -1, a->ne ne pointe vers aucun arbre du tableau, on ajoute donc a->ne au tableau, sinon, on se contente d'imprimer dans le fichier l'indice de l'arbre vers lequel pointe a->ne dans le tableau.
Et on fera ca pour a->no, a->ne, a->se, a->so.

(Par exemple, pour le damier dans l'énoncé, la tableau, à la fin du parcours de l'arbre minimisé donnera [a, a->no, a->no->no, a->no->no->no, a->no->no->ne])

Cette méthode permet un parcours en largeur de l'arbre.
Ainsi, lorsque l'on rencontre une feuille, on va imprimer dans le fichier l'indice de la feuille et 0 si elle est blanche, noire sinon.
Et lorsque l'on rencontre un noeud, on va imprimer dans le fichier, l'indice du noeud, et l'indice des noeuds vers lesquels pointent ses 4 fils.

Reprenons l'exemple du damier, on arrive au premier noeud de l'arbre donc son indice est égal à 0, on imprime 0 dans le fichier, ensuite son fils no n'est pas encore dans le tableau donc on l'ajoute dans le tableau et il a comme indice 1 donc on imprime 1, ensuite son fils ne pointe vers son fils no dont l'indice est 1 donc on imprime 1, idem pour ses fils se et so.
La première ligne donnera donc « 0 1 1 1 1 ».
Même fonctionnement pour les 3 noeuds suivants.
En revanche lorsque l'on va arriver aux feuilles blanches et noires d'indice 4 et 5, on imprimera « 4 0 » puisque la première feuille rencontrée est blanche et que son indice est 4 et « 5 1 » puisque la deuxième feuille rencontrée est noire et que son indice est 5.

Autres explications:

Toutes les autres explications sont expliquées en commentaires de code et ne nécessitent pas d'explication particulière.

Répartition du travail:

Des difficultés ont été rencontrées pour Hugo qui a eu du mal a utiliser le git, ainsi malgré peut-être une divergence sur les rendus de git, le travail a correctement été partagé au sein du binôme.