

# Harmonie des Couleurs

## Compte Rendu 5

### Développement de l'application avec Qt Creator (suite) :

Cette semaine nous continué le développement de l'application avec Qt Creator. La semaine précédente, nous avons réussi à ouvrir notre image, à utiliser le générateur de couleurs dominantes et afficher les boutons des couleurs correspondantes.

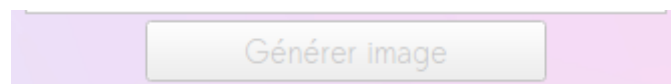
Tout d'abord, nous avons ajouté un bouton non cliquable nous permettant de visualiser la dernière couleur sélectionnée, que ce soit via les boutons des 4 couleurs dominantes ou sur la palette personnalisée. C'est cette couleur que l'on va envoyer dans nos fonctions. Voici un exemple ci-dessous :



Cela n'a pas été aisé de prime abord car nos couleurs dominantes et notre couleur de palette ne sont pas récupérées de la même manière. Comme nous l'avons vu la semaine dernière, les 4 couleurs dominantes sont codées sous la forme de *Pixel* et nous devons les caster afin de modifier la couleur. Cependant la couleur de la palette, elle, est stockée sous forme de *QColor*. Nous devons donc la convertir au format *Pixel*. *Qcolor* nous permet de récupérer les valeurs rgb séparément :

```
colorChosenPixel.r = color.red();  
colorChosenPixel.g = color.green();  
colorChosenPixel.b = color.blue();
```

De plus nous avons décidé de griser le bouton "Générer image" lorsque la couleur n'a pas été encore choisie au moins une fois.



Ensuite nous nous sommes occupés d'ajouter des *boutons radio* afin de sélectionner laquelle des harmonies nous voulons utiliser sur notre image.

Choix de l'harmonie :

☒ Analogue

☐ Complémentaire


☐ Triadique

☐ Quadratique

Chaque harmonie a une largeur de bande. Nous avons donc ajouté un *horizontalSlider* nous permettant de modifier la valeur. Cependant, elles ont toutes une largeur différente, c'est pourquoi nous devons fixer un maximum différent pour chaque sélection. Nous avons également mis des crans afin de deviner plus facilement les valeurs et un affichage de la valeur.

Largeur de la bande :

(sélectionner une valeur)

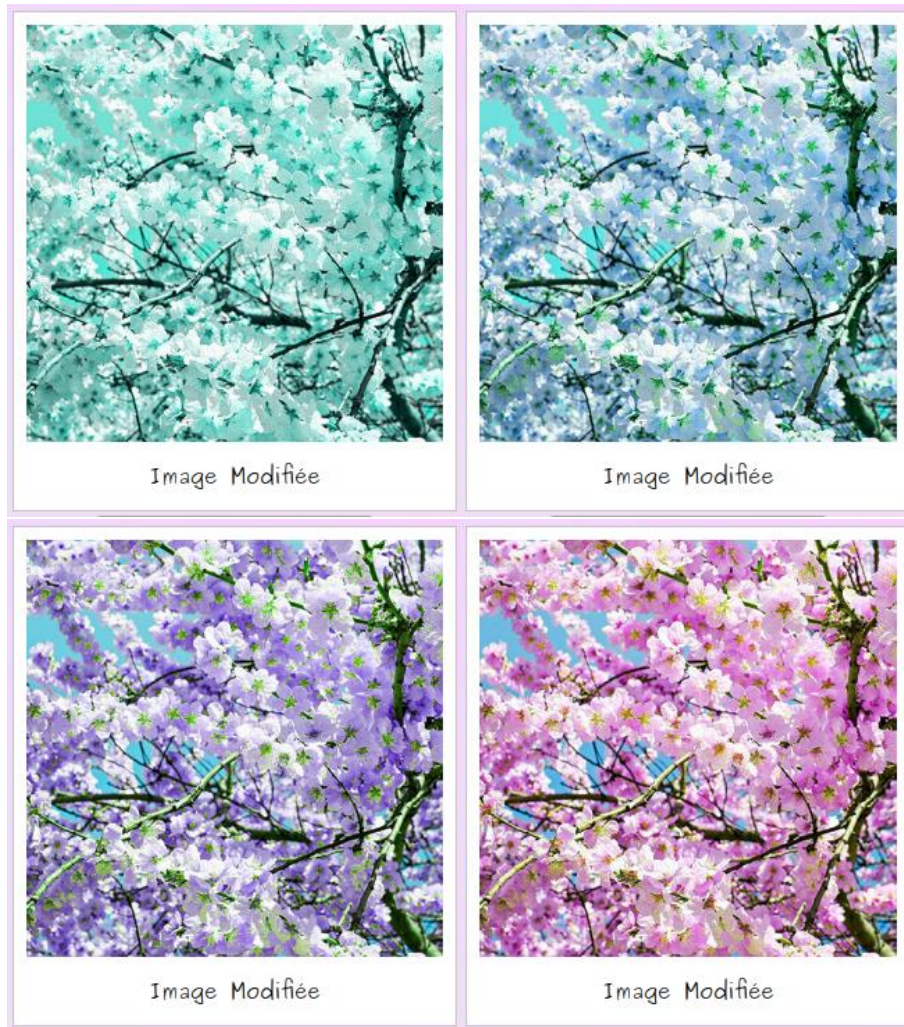


On aura donc :

- Analogue : 1 à 359°
- Complémentaire : 1 à 179°
- Triadique : 1 à 119°
- Quadratique : 1 à 89°

Largeur de la bande : 359	Largeur de la bande : 179
Largeur de la bande : 119	Largeur de la bande : 89

Ci-dessous, voici un exemple de l'effet que cela génère sur notre image. En premier temps l'image harmonisée en version *analogue* avec une bande de 1, puis la même image, avec la même couleur mais une bande de 70, puis 180 et enfin 280. On constate que plus on élargit la bande, plus on retrouve les couleurs originales de notre image.



Pour afficher ces résultats nous avons dupliqué nos fonctions Analogue, Complémentaire et Triadique afin de les modifier en version QT pour pouvoir les utiliser dans notre interface. Entre temps, dans la partie algorithme nous avons ajouté l'harmonie quadratique (voir partie Algorithme). Voici un exemple :

Avant :

```
void AnalogueHarmony(const std::vector<Pixel> &pdominant, const std::vector<Pixel> &listePixels,
int nH, int nW, double sizeBand)
```

Après :

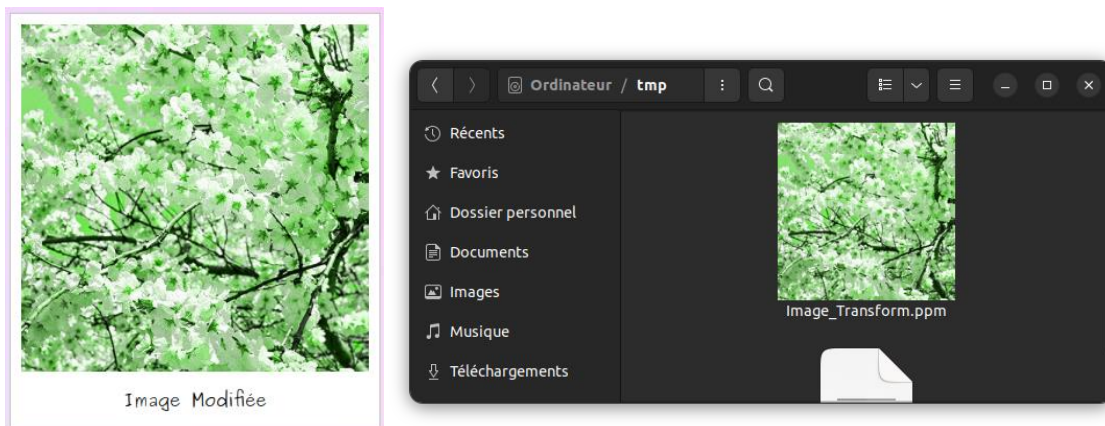
```
void AnalogueHarmonyQT(const Pixel &cdominant, char * filePath, double sizeBand)
```

En effet, nous avons seulement besoin du chemin du fichier, la couleur dominante et la largeur de la bande. Chaque fonction est indépendante. Chacune finit en générant l'image modifiée avec la fonction `ecrire_image_ppm()`. Dans celle-ci, nous mettons en

paramètre le chemin dans lequel nous voulons stocker cette image. Cette dernière est une image temporaire dont nous ne voulons pas que l'utilisateur ait connaissance. Donc on y met le nom `"/tmp/Image_Transform.ppm"`.

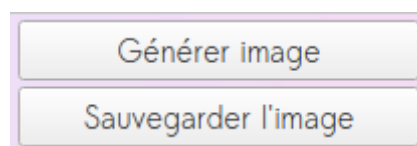
Il est très important que chacune écrive le même nom au même endroit. En effet, cela va permettre de facilement supprimer l'image précédemment générée et sauvegarder la nouvelle. De ce fait, lorsqu'on génère l'image on va pouvoir facilement retrouver le chemin pour l'afficher avec `QImage`.

```
QImage image("/tmp/Image_Transform.ppm");  
image = image.scaledToWidth(ui->final_image_label->width(), Qt::SmoothTransformation);  
ui->final_image_label->setFixedSize((image.size()));  
ui->final_image_label->setPixmap(QPixmap::fromImage(image));
```

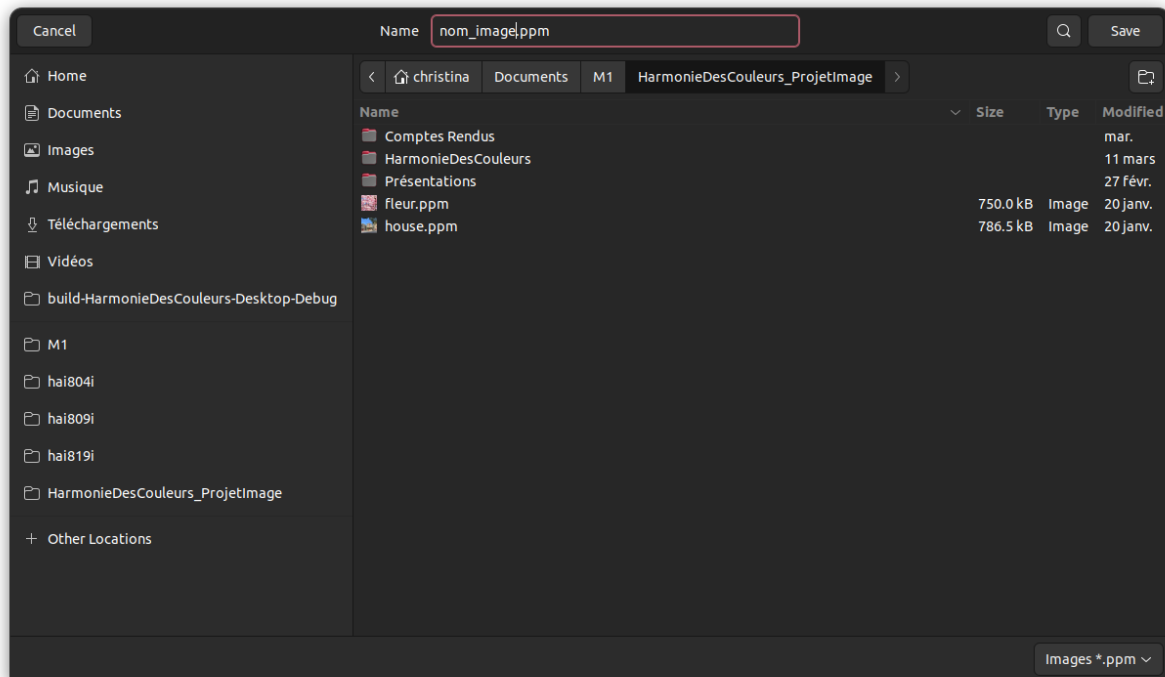


Ensuite ce que nous voulons faire, c'est pouvoir sauvegarder cette image. En effet, comme dit précédemment, l'utilisateur ne sait pas où elle se situe et l'on veut qu'il puisse facilement l'enregistrer dans le dossier de son image de base s'il le souhaite.

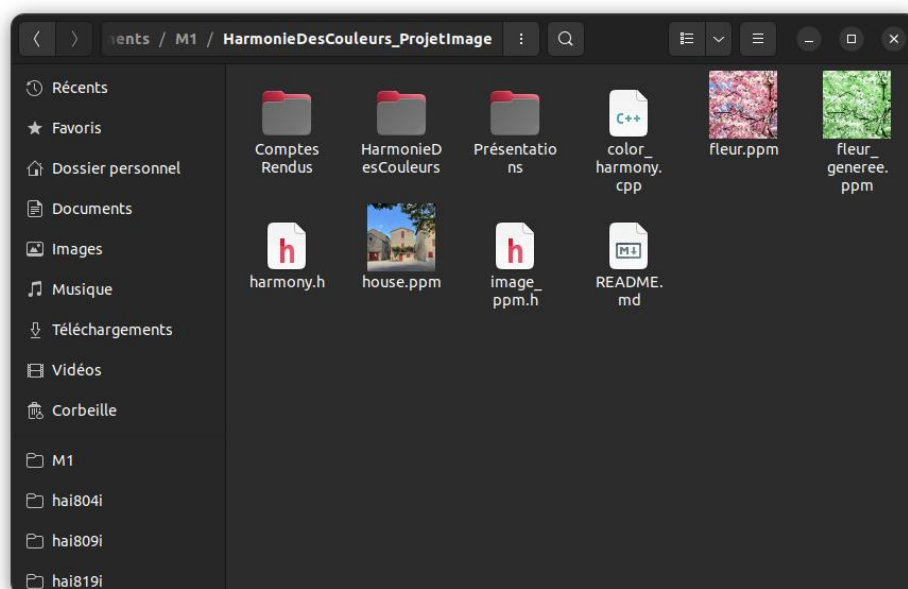
Nous avons donc créé un bouton "sauvegarder l'image".



Lorsque l'on appuie dessus, une fenêtre de dialogue apparaît :



Le nom de base est *"nom\_image.ppm"* afin que l'utilisateur ne se trompe pas et n'écrase pas l'image originale. Pour l'exemple, nous l'avons ainsi modifié en *"fleur\_generee.ppm"*. Nous pouvons constater qu'elle a bien été enregistrée dans le dossier original.



Pour faire tout cela il nous a fallu récupérer le chemin de l'image originale, faire *info(cheminFichier)* puis appliquer *path()* pour récupérer le chemin du dossier. Nous avons ensuite créé un *QString nomFichierSaved* qui est généré grâce au *QFileDialog::setSaveFileName()* qui ouvre la fenêtre de dialogue pour choisir l'endroit où



sauvegarder et renommer l'image si besoin. Ensuite si les conditions sont respectées, on utilise *save(nomFichierSaved, "ppm")* pour sauvegarder (l'image qui est dans le dossier tmp) au format ppm avec le bon nom à l'endroit voulu.

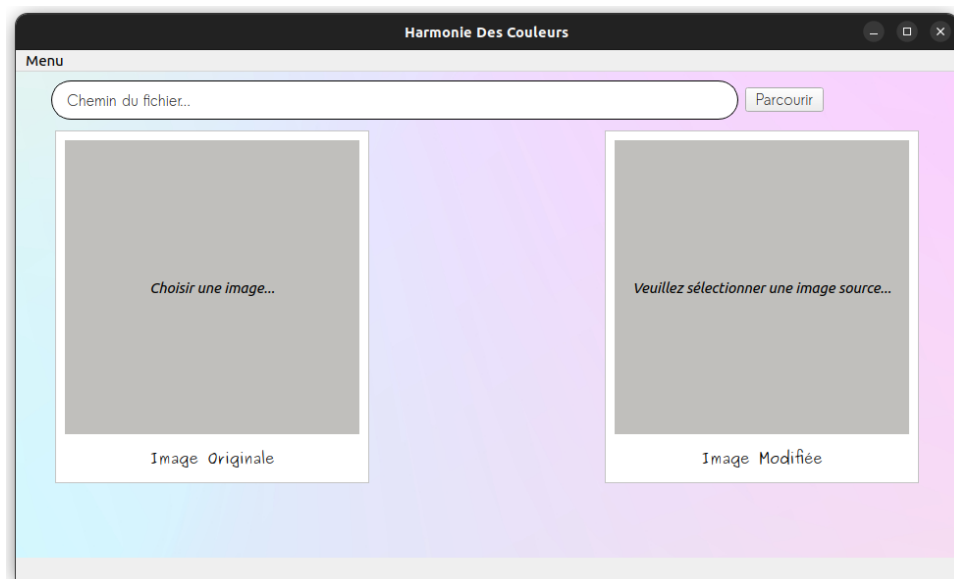
```
void HarmonieDesCouleurs::on_Save_image_clicked()
{
    if(imageModifiée){
        QImage imageTransformee("/tmp/Image_Transform.ppm");
        QFileInfo info(cheminFichier);
        QString cheminFichierDossier = info.path();

        QString nomFichierSaved = QFileDialog::getSaveFileName(this, tr("Sauvegarder l'image"), cheminFichierDossier + "/nom_image.ppm", tr("Images *.ppm"));

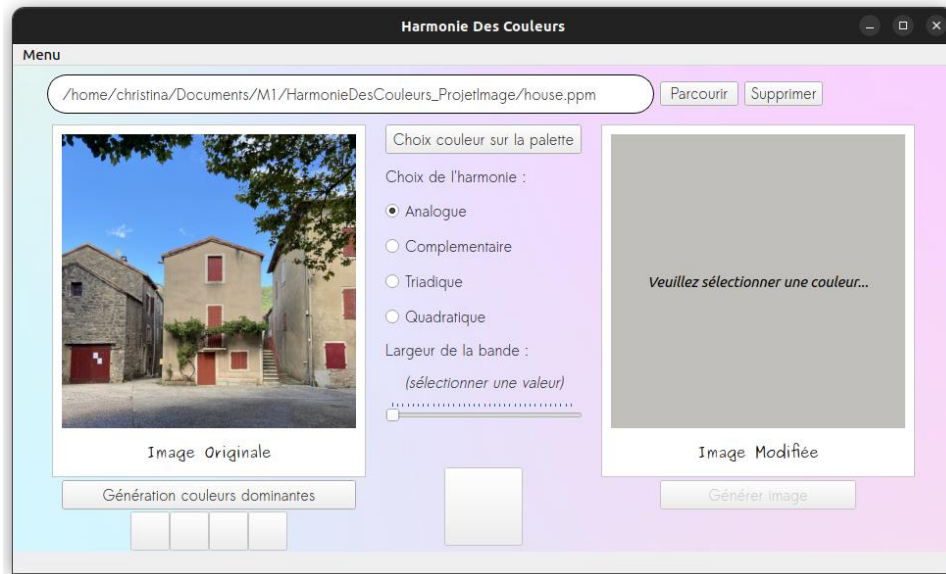
        if (!nomFichierSaved.isEmpty()){
            if(imageTransformee.save(nomFichierSaved,"ppm")){
                qDebug() << "Image Sauvergardée.";
            }
        }else{
            return;
        }
    }
}
```

Enfin, nous avons fait plusieurs transformations pour un meilleur confort visuel et une interface plus intuitive pour l'utilisateur.

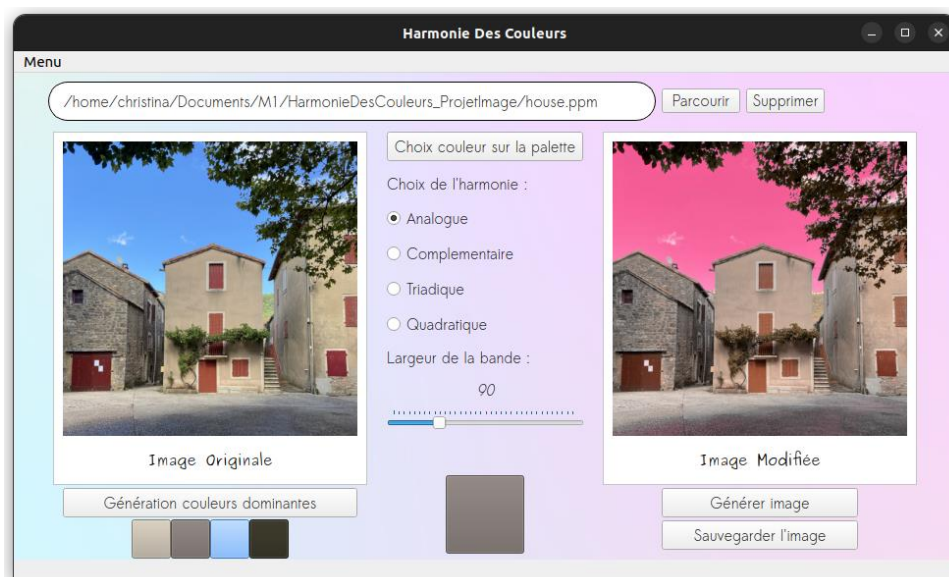
Tout d'abord, nous avons enlevé toute l'interface qui ne doit pas être utilisée tant que l'on n'a pas choisi une image source :



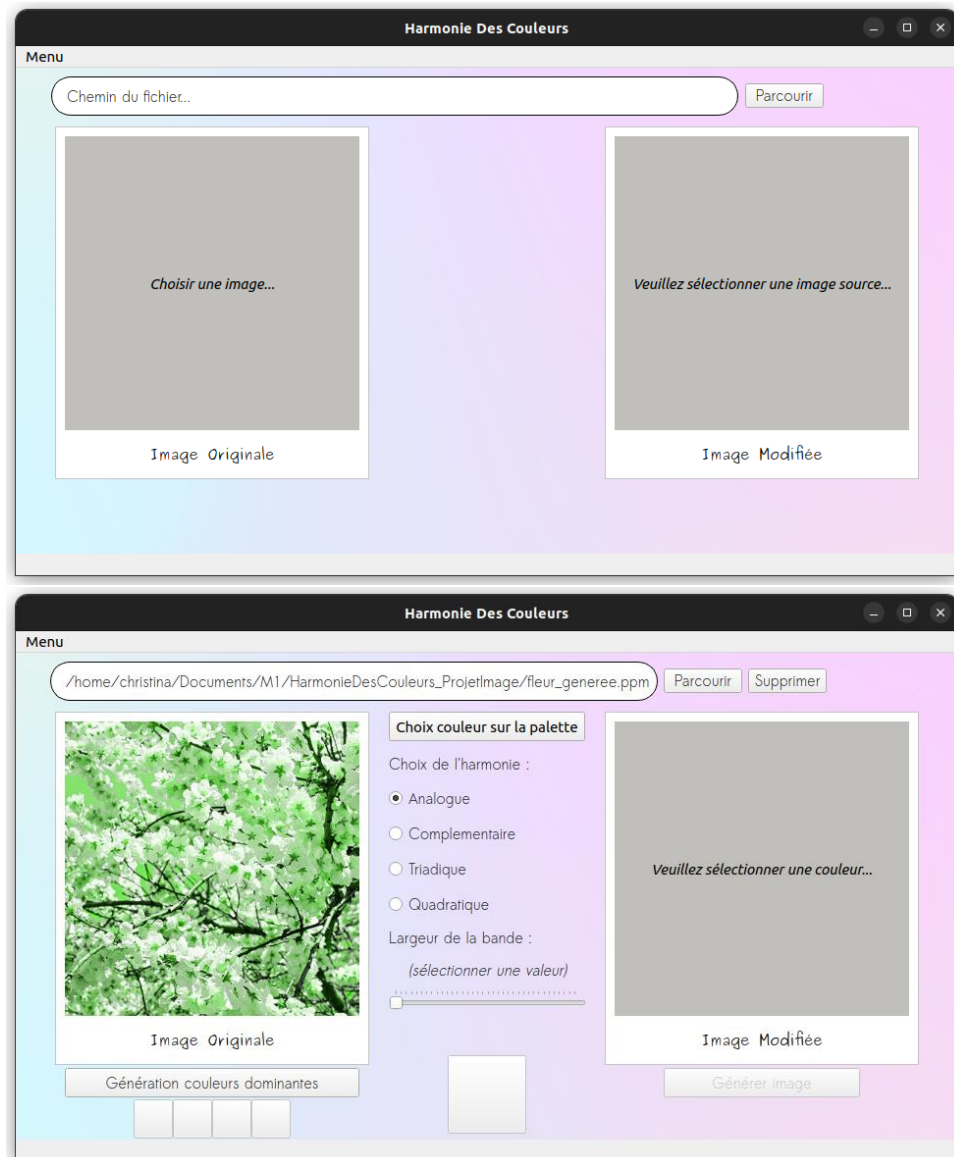
Ensuite, comme expliqué plus tôt, on a grisé le bouton "générer image" si l'on n'a pas choisi de couleur. Le reste s'affiche. On remarque que le texte dans le cadre de l'image modifiée a été changé de manière à ce que l'utilisateur comprenne la consigne.



Ensuite, on fait apparaître le bouton “sauvegarder l’image” si les conditions sont remplies.



Enfin, si l’on clique sur supprimer, on a fait en sorte que tout retourne à leurs valeurs initiales. Voici ci-dessous, l’affichage lorsque l’on supprime puis ajoute une nouvelle image.



Voilà pour cette semaine.



## Algorithme :

Cette semaine nous nous sommes concentrés en particulier sur l'intégration des fonctions vu précédemment dans l'application en devant ainsi modifier les paramètres demandé et l'image obtenue (comme l'emplacement du fichier de sortie par exemple)

Niveau algorithmique nous avons continué à investiguer sur la manière à utiliser afin de prendre en compte l'harmonisation spatiale.

Également nous avons mis au point un nouveau type d'harmonisation, la « **square tetradric harmonisation** »



Exemple d'harmonisation square

Comme on peut le voir au niveau des bourgeons par exemple, il y a des soucis au niveau de l'harmonisation spatiale.

Également on remarque que ce type d'harmonisation change beaucoup moins par rapport à la photo d'origine, ce qui est normal car nous avons à présent un choix de couleur beaucoup plus vaste à l'aide des quatre points obtenus lors de la mise en place du carré, ainsi le choix de couleur est beaucoup plus vaste.

## Pour la semaine prochaine :

Pour la semaine prochaine nous prévoyons de nous concentrer grandement sur les algorithmes d'harmonisation spatiale afin d'avoir un résultat plus propre et convenable.

Nous regarderons comment améliorer notre application pour qu'elle soit la plus intuitive possible pour un utilisateur lambda.

### Sources :

<https://www.techniques-de-peintre.fr/theorie-des-couleurs/harmonie-coloree/>

[https://en.wikipedia.org/wiki/HSL\\_and\\_HSV](https://en.wikipedia.org/wiki/HSL_and_HSV)

<https://doc.qt.io/>