1. Présentation du programme

SYNAD est un programme développé pour la fête de la science 2018, dans le cadre des projets de 2ème année FIP EII à Télécom Physique Strasbourg.

Ce programme se décompose en deux sous-programmes :

* Synthèse additive visualisé en faisant varier l’intensité lumineuse des cercles.
* Permutation des intensités lumineuse de chaque pixel d’une image.

Le programme a été codé en Python et utilise une interface graphique « Kivy designer ». Cette interface a été utilisé pour simplifier le code pour maximiser l’ergonomie de l’application et permettre une utilisation facile.

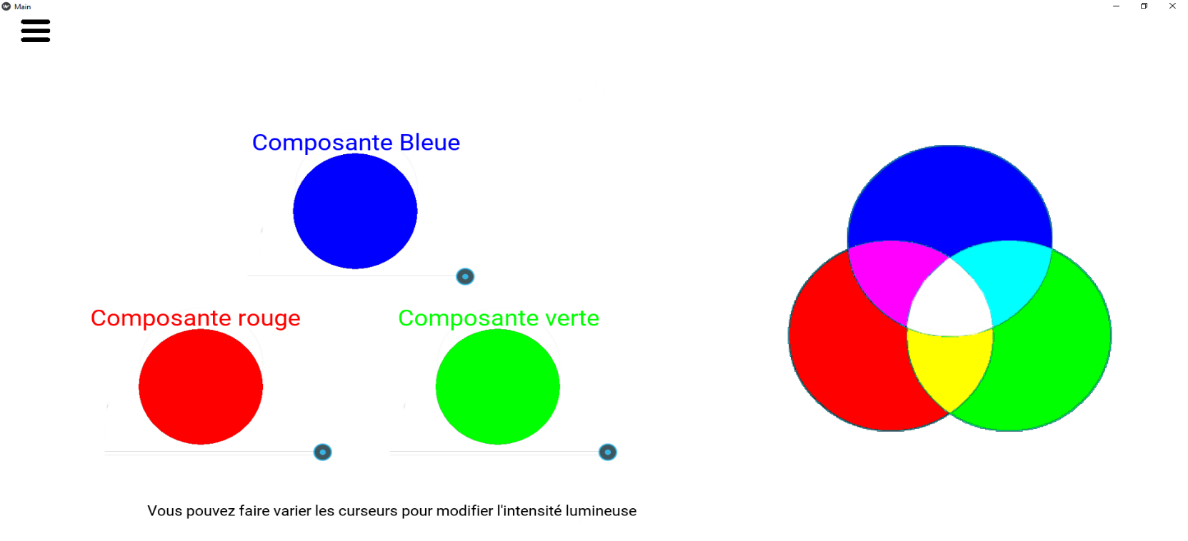
Nom de l’application : SYNAD : - SYNAD = En référence à la Synthèse Additive.

Aperçu :

L’application se lance sur le programme de la synthèse additive,

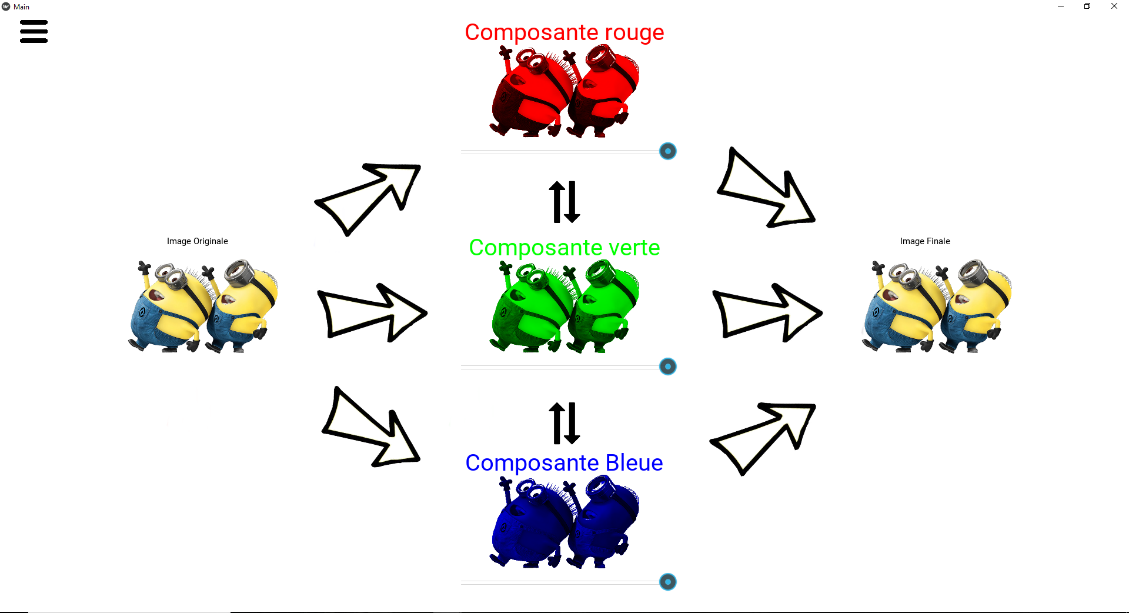
L’interface de la synthèse additive est composée de :

* 3 cercles (rouge, vert, bleu)
* D’une image finale
* 3 sliders permettant de faire varier l’intensité lumineuse des cercles et de l’image finale.
* D’un bouton paramètre permettant de changer de programme.



L’interface permettant la permutation des intensités lumineuses est composée de :

* 1 image initiale
* 3 images pour chaque composante
* 1 image finale
* 3 sliders pour modifier l’intensité lumineuse des composantes et de l’image finale
* 2 images échange des intensités lumineuses.
* 6 flèches décomposition/recomposition de l’image permettant d’améliorer la compréhension.



2. Exigences de fonctionnement

Pour faire fonctionner le programme, voici les éléments requis :

- Un ordinateur sous Linux ou équivalent

- La totalité du programme SYNAD fourni

Ceci comporte le main.py, le fichier contenant les fonctions pour la modification de l’interface graphique et la permutation des intensités lumineuse.

Du fichier main.kv pour la création de l’interface graphique. Du fichier buildozer.spec qui est un fichier de configuration pour construire l’exécutable.

Le dossier Cercle, Img,et ImgAnnexe qui contient toutes les image utilisé dans l’application.

La suite des éléments requis est inclue dans un fichier « requirement.txt » à partir duquel on peut tout installer en une seule commande à l’aide de « *pip install*».

- Python 3.6.4

- Kivy 1.10.1.dev0 # Utilisé pour créer l’IHM du programme

- Kivy-Garden 0.1.4 # Module concentrant tous les add-ons Kivy crées par les utilisateurs

- Pillow 5.0.0 # Librairie de traitement d’image

Le lancement de l’application s’effectue par la commande : *python main.py*

L’arrêt de l’application s’effectue par une pression de la touche « Échap » ou en fermant la fenêtre principale.

3 Kivy Designer

SYNAD est une application avec une interface très chargé, car l’application de la synthèse additive et de la permutation de l’intensité lumineuse des pixels se trouve sur une même fenêtre.

Kivy designer est une interface graphique pour la librairies kivy, elle permet de créer rapidement des interfaces graphiques de manière visuelle.

L’ergonomie et la clarté étant un facteur important de l’application, cela a été la raison principale pour laquelle on a décidé d’utiliser Kivy designer pour réaliser le développement de l’interface graphique.

Pour installer Kivy designer on a besoin de :

* *La librairie kivy 1.9+*
* *Les module Python suivants* 
  + *Watchdog*
  + *Pygments*
  + *Docutils*
  + *Jedi*
  + *Gitpython*
  + *Six*
  + *Kivy-garden*
* *Le widget FileBrowser depuis Kivy Garden*

Pour installer tous les fichiers ci-dessus, on peut utiliser les commandes :

*git clone* [*http://github.com/kivy/kivy-designer/*](http://github.com/kivy/kivy-designer/)

*cd kivy-designer*

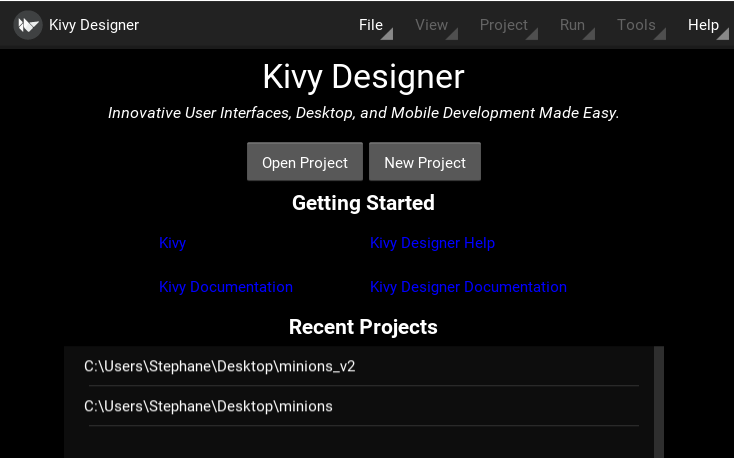
*pip install -Ur requirements.txt*

Pour installer le FileBroswer :

*garden install filebrowser*

Pour lancer Kivy designer :

*Python -m designer*

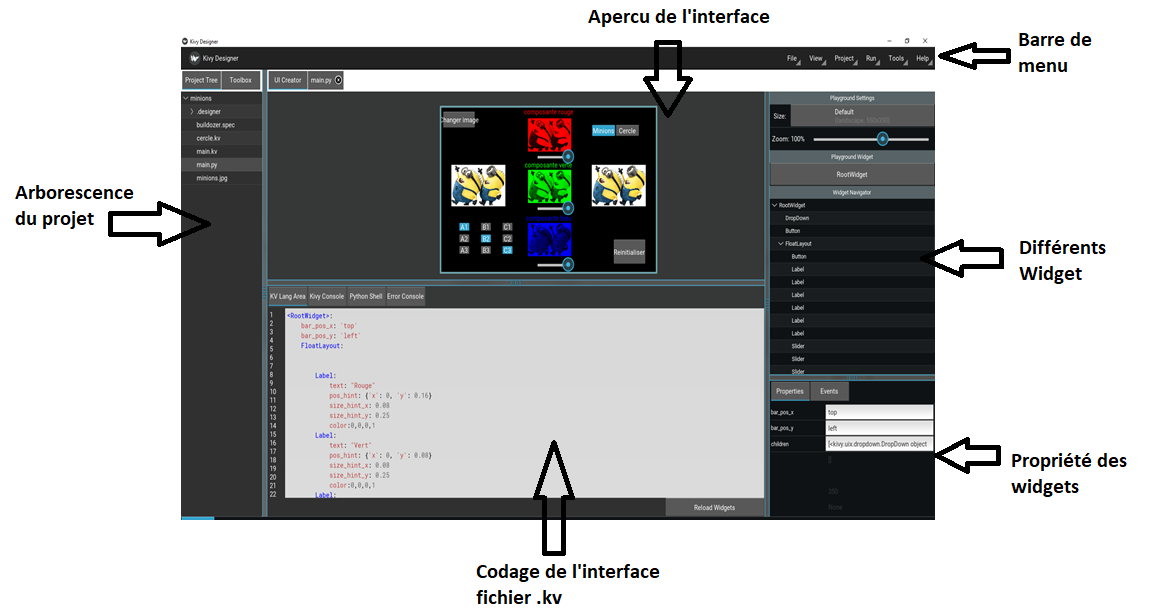


L’arborescence d’un nouveau projet Kivy créer à l’aide de Kivy designer se décompose de la manière suivante :

* Un fichier kv, elle permet de créer l’interface graphique, de placer différents widgets (texte, image, bouton) et de donner leurs spécifications (taille, place, couleur …).
* Un fichier py elle permet de développer les fonctionnalités de l’application (changement de programme, permutation des intensités lumineuse).
* Un fichier buildozer.spec elle permet de donner les spécifications de l’application (titre de l’application, dossier source, configuration du débogage …).

Tous les fichiers crées peuvent être modifiés indépendamment de l’interface graphique kivy designer.

L’utilisation des fichiers kv est très intéressant notamment lors de la réalisation d’interface complexe que l’on peut retrouver dans l’application SYNAD.

L’interface de kivy designer se décompose de la façon suivante :

* Arborescence du projet : permet d’ouvrir dans l’interface kivy designer les différents fichiers du projet (py, kv, spec, image, …)
* Aperçu de l’interface : permet de visualiser en temps réel l’évolution graphique du programme.
* Barre de menu : permet de choisir différents paramètres, d’exécuter, sauvegarder …
* Arborescence widget : permet de voir et sélectionner les différents Widget utilisé dans l’application.
* Propriété des widgets : permet de spécifier (la position, taille, couleur du widget).
* Codage de l’interface : permet de voir l’évolution et de modifier le code de l’interface.
* La toolbox : permet l’ajouts de différents widgets (text, bouton, image) par un drag and drop.

En conclusion Kivy designer est idéal pour faire une esquisse simple et rapide de l’interface. Quand l’interface devient trop complexe il est préférable de modifier directement le fichier kv.

4. Le code

Les fichiers main.py et main.kv permettent le fonctionnement de l’application. Ils ont été commentés pour permettre une meilleure compréhension lors de sa lecture. Néanmoins, nous allons résumer son fonctionnement dans cette partie.

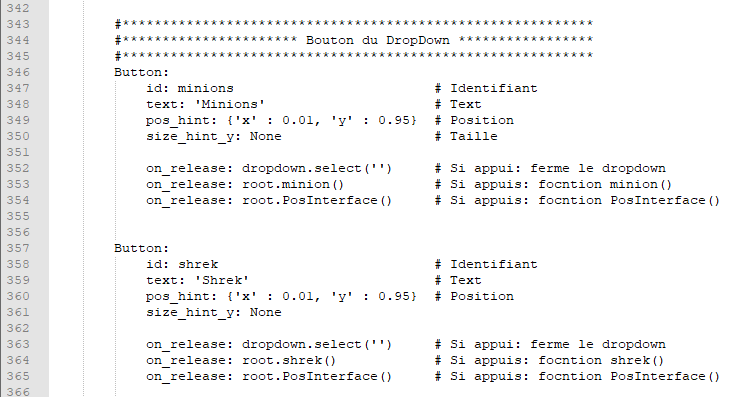
Le fichier main.py comporte :

- 2 classes (RootWidget, MainAPP)

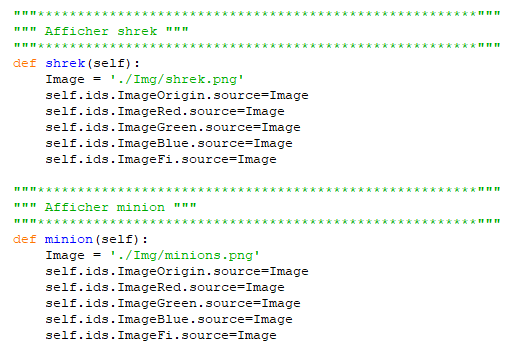
- 7 fonctions (Init, Synthese, PosInterfaces, minion, cercle, change1, change2)

La classe RootWidget, c’est dans cette classe que l’on peut appeler chaque Widget crée dans le fichier main.kv

La classe MainApp c’est dans cette classe que l’on appelle le fichier main.kv contenant tous les Widgets de l’interface graphique.

**Ajouter une image :** *Créer un bouton supplémentaire dans le DropDown*

*Modifier l’image source sur l’appuis du bouton.*



4.1. Classe « RootWidget »

La classe RootWidget permet de gérer le changement d’interface et d’image entre les 2 sous programmes et la permutation des composantes RGB de chaque pixel de l’image.

Fonctionnement des différentes fonctions de la classe :

***Init***

Arguments en entrée : *self*, et *\*\*kwargs*

Éléments de sortie : *Void*

Cette fonction permet d’initialiser les variables position des composantes RGB et sliders RGB

***synthese***

Arguments en entrée : *red, green, blue*

Éléments de sortie : *Void*

Cette fonction permet des réaliser la synthèse additive des couleurs, en prenant l’intensité lumineuse des sliders (rouge vert et bleu).

***PosInterface***

Arguments en entrée : *self*f

Éléments de sortie : *Void*

La fonction permet de repositionner tous les Widgets pour le programme sur la permutation des intensités lumineuse. Le repositionnement de l’interface est donné manuellement dans le fichier main.py et l’initialisation est réalisé dans le fichier graphique Main.kv. Cela implique que pour modifier la position/taille d’une il faut la modifier dans le fichier main.py et dans le fichier main.kv.

***Minion***

Arguments en entrée : *self*

Éléments de sortie : *Void*

Cette fonction permet de charger l’image minion

***Cercle***

Arguments en entrée : *self*

Éléments de sortie : *Void*

Cette fonction permet de repositionner tous les Widgets pour le programme sur la synthèse additive et charge les images cercle. Aucun code n’est utilisé pour gérer la synthèse additive, tout est géré graphiquement dans le fichier main.kv.

***Change1***

Arguments en entrée : *self*

Éléments de sortie : *Void*

La fonction permet d’échanger les intensités de l’image du haut avec l’image du milieu et de modifier l’image finale.

Pour l’aspect graphique l’application recherche la couleur de l’image et slider qui sont en haut et au milieu et les changent de position.

Pour l’aspect technique, il ouvre l’image source de l’image Finale et échange l’intensité de chaque pixels en fonction de la composante du haut et du milieu, puis l’enregistre en image temporaire pour réactualiser l’image finale.

***Change2***

Arguments en entrée : *self*

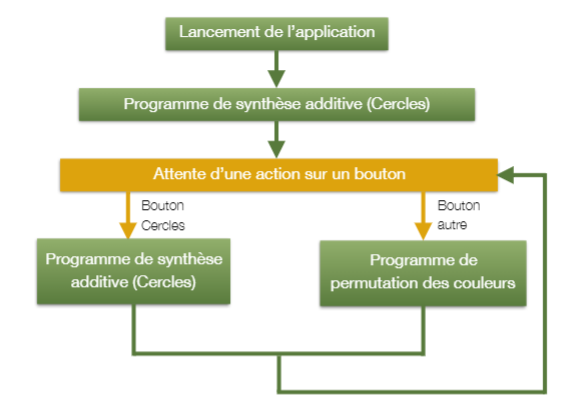
Éléments de sortie : *Void*

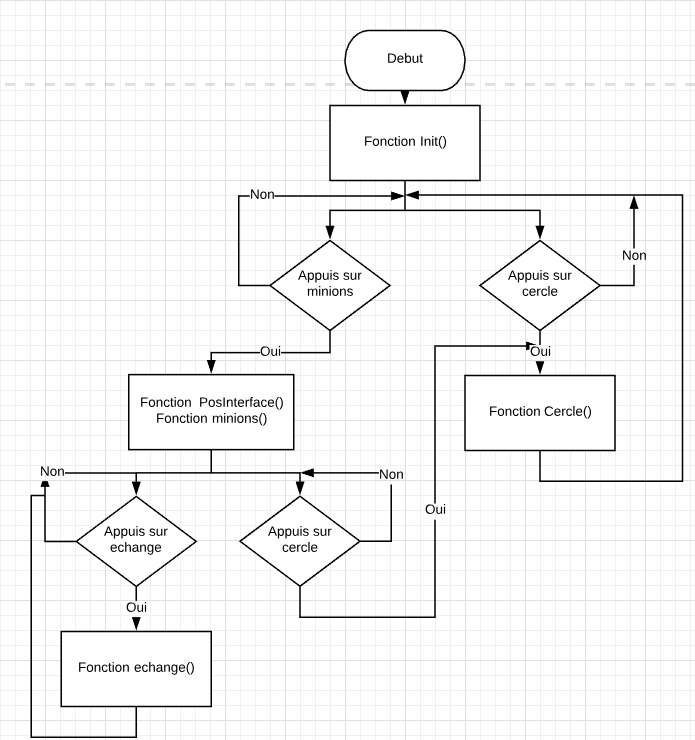
La fonction permet d’échanger les intensités de l’image du milieu avec l’image du bas et de modifier l’image finale.

Pour l’aspect graphique l’application recherche la couleur de l’image et slider qui sont au milieu et en bas et les changent de position.

Pour l’aspect technique, il ouvre l’image source de l’image Finale et échange l’intensité de chaque pixels en fonction de la composante du haut et du milieu, puis l’enregistre en image temporaire pour réactualiser l’image finale.

5. Organigramme de fonctionnement





6. Justification des choix

Dans cette partie, nous allons expliquer pourquoi nous avons décidé d’utiliser tel ou tel outil dans notre programme.

5.1. Kivy

Kivy est une librairie python OpenSource pour le développement rapide d’applications avec des interfaces innovantes. Ces idées nous ont séduit et lors de nos recherches, nous avons apprécié les réalisations que d’autres utilisateurs ont produit. Kivy est également Cross-platform.

Cela permet le développement sur Windows, Linux, Mac et d’obtenir toujours le même rendu. En résumé, c’est sa simplicité d’utilisation, sa compréhension facile, son design et sa multicompatibilité qui nous ont fait choisir Kivy.

5.2. Pillow

Pillow est la librairie principale que j’ai utilisé pour le traitement d’image, permutations des pixels et synthèse additive. Elle permet de réaliser les fonctions dont a besoin l’application de manière simple rapide et optimisé.

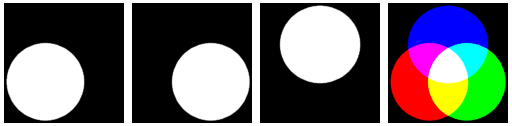
La permutation des pixels se fait en 2 lignes de code à l’aide de getpixel() et putpixel().

La Synthese\_additive a été développé de 3 manière différente avec leurs avantages / désavantages.

La première fonction est réalisée de la manière suivante :

* On crée trois images en niveau de gris (blanc et noir) avec un fond entièrement noir
* Superposer toutes les images.
* On dessine un cercle rouge décaler à gauche sur le fond noir qui représentera la composante rouge.
* On dessine un cercle vert décaler à droite sur le fond noir qui représentera la composante verte.
* On dessine un cercle bleu au milieu décaler vers le haut sur le fond noir qui représentera la composante verte.
* On crée une nouvelle image RGB qui fusionne l’ensemble des images à bande unique crée précédemment dans une nouvelle image multibande.

Le code utilisé : (Image.merge('RGB', (image\_R, image\_G, image\_B)))



Le problème principal de cette méthode est le fond noir qui hôte toutes l’esthétique a notre application. L’avantage est la rapidité d’éxécution.

La deuxième fonction a été réalisée suite à la découverte de la méthode Imagechops :

* On crée trois images RGBA ou le fond est transparent.
* On superpose toutes les images.
* On dessine un cercle rouge décaler à gauche sur le fond transparent qui représentera la composante rouge.
* On dessine un cercle vert décaler à droite sur le fond transparent qui représentera la composante verte.
* On dessine un cercle bleu au milieu décaler vers le haut sur le fond transparent qui représentera la composante verte.
* On crée une nouvelle image RGB qui fusionne l’ensemble des images à bande unique crée précédemment dans une nouvelle image multibande.

Le code utilisé : out = ImageChops.add(image1,image2,0.5)

out = ImageChops.add(out,image3,0.5)

C’est technique qui est privilégié dans notre programme car elle permet d’éliminer le fond noir et ainsi de rendre l’application plus esthétique. Car elle présente le meilleur compromis.

La troisième méthode est d’utiliser directement l’interface graphique pour charger et gérer la couleur de chaque partie de l’image.

Les avantage sont les suivant :

- Toute l’interface graphique est créé dans un seul fichier.

- Simplicité : pas besoin de code supplémentaire.

Les inconvénients sont :

* Charger 4 image supplémentaire dans une interface déjà bien remplis.
* Garder la même taille et la même position pour tous les cercle (modification plus importante).
* Les contours des cercle sont visible.

7. Limites et améliorations possibles

De nombreuse améliorations peuvent être apportés à SYNAD afin d’améliorer certaines fonctionnalités de l’application. Cela nécessite de gérer la création de certain Widgets autrement que dans un fichier graphique main.kv. On peut donc gérer l’interface graphique en faisant un mixte avec le fichier main.py et main.kv.

L’ajout automatique d’images nécessite de créer un dopdown dans le fichier main.py et d’incrémenter le nombre de bouton à l’intérieur en fonction du nombre d’image.

On peut également ajouté la synthèse soustractive pour faire le contraste avec la synthèse additive.