**1.Introduction**

Le but de ce projet est de créer un éditeur de photo qui permettra d’effectuer des opérations simples comme l’ajout de décoration ou des filtres de manière rapide. Cette application sera destinée aux adolescents qui veulent éditer leurs photos de profil pour les publier sur les réseaux sociaux. Pour l’implémentation de ce projet on utilisera Java, plus précisément la bibliothèque javaFx et FXML.

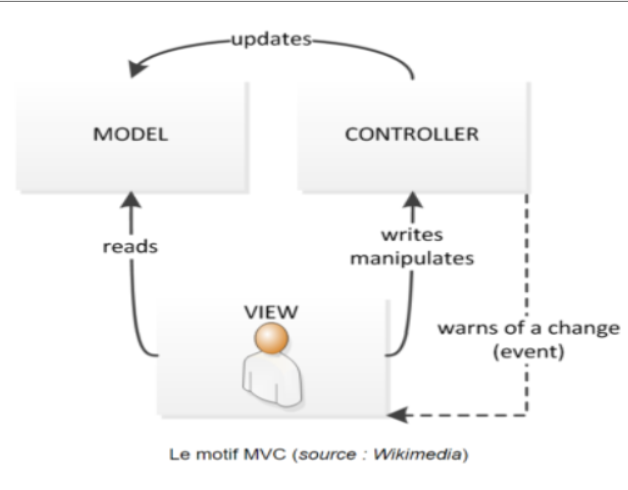
**2. Analyse globale**

Notre projet sera implémenté suivant le modèle MVC et contiendra une vue dans laquelle les éléments suivants sont présents :

* Une partie qui inclue des boutons et des textField pour manipuler les différents composants sur la photo.
* Une Canvas qui affichera notre photo et les modifications effectuées dessus.
* Un menu qui permet d’importer, sauvegarder une photo, créer des composants et appliquer des filtres.

**2.1. Conception générale**

Puisque notre programme gère une interface graphique, nous avons adopté le design pattern MVC.

Notre fenêtre sera gérée par la vue qui s’occupera de l’affichage des éléments à dessiner sur l’écran. Tandis que la création des composants et la gestion des boutons de l’application seront gérer par le contrôleur qui notifie la vue des changements qui ont eu lieu tout en modifiant les variables du modèle. Voici un diagramme qui illustre le modèle MVC :

**3.Conception détaillée**

**3.1. Les composants**

Les composants : Rectangle, Ovale et Emoji hériteront tous la classe Component et utiliseront le contrôleur ComponentControl.

**3.1.1. ComponentControl**

La classe ComponentControl gère la réaction des composants aux inputs de l’utilisateur.

Les méthodes de la classe sont les suivantes :

*Attraper* : Une méthode qui déplace un composant suivant les mouvements de la souris, on teste la position de la souris sur notre Canvas et on met à jour la variable grabbed si le curseur fait contacte avec un composant.

*Déplacer* : Une méthode qui déplace un composant qui a été attrapé et met à jour l’affichage.

Déposer : Une méthode qui fixe la position du composant qui a été déplacée et met à jour l’affichage

*Select* : Une méthode qui réagit aux cliques de la souris et teste si on fait contact avec un composant. Cette méthode permettra d’appliquer de diverses modifications sur le composant choisi.

*ApplyDimChanges* : Cette méthode permet d’appliquer des nouvelles dimensions au composant qui a été sélectionné. Ces composants seront extraits des textField présent dans l’application et les valeurs saisies sont vérifiée pour garantir qu’elles sont numériques.

*Delete* : Cette méthode permet de supprimer un composant sélectionné si aucun composant n’a été sélectionné un message d’erreur s’affichera pour aider l’utilisateur.

**3.1.2. Rectangle, Ellipse**

Les classes Rectangle et Ellipse représenteront les formes géométriques de l’application, elles héritent de la classe abstraite Component et implémentent donc les mêmes méthodes.

*Draw* : Cette méthode permet de dessiner la forme choisie sur l’écran.

*inForm* : Cette méthode teste si un clique de la souris est sur le composant ou pas. Elle prend en paramètre les coordonnées de la souris.

updatePos : Cette méthode permet de mettre à jour la position du composant. Elle prend en paramètre les coordonnées de la souris.

Ces deux classes possèdent aussi des getteurs et des setteurs pour les attributs.

**3.1.3Emoji**

La classe emoji héritent de Component, l’intégralité de ces méthodes sont codés de la même façon que Rectangle a l’exception de l’attribut Icon qui représente l’image de l’emoji. La méthode draw dessinera donc une emoji aux coordonnées x et y et avec une largeur width et une hauteur height.

**3.1.4Line**

La classe Line est un peu particulière elle hérite aussi de Component mais possède deux attribut x2 et y2 qui représente les coordonnées du deuxième point de la ligne. La méthode draw s’adapte donc pour afficher une ligne à partir des coordonnées des deux points.

Le contrôleur qui permet la création des lignes est LineCreator.

**3.1.5. LineCreator**

La gestion de la mise à jour de la position de la ligne s’est avérée difficile. Ce qu’on choisit de faire est de laisser l’utilisateur choisir de manière précise la position des deux points. Les attributs de la classe sont les suivantes :

Textes : une liste des textes crée par l’utilisateur.

Positions : la position des textes crée par l’utilisateur.

Fonts : la taille des textes crée par l’utilisateur.

Colors : la couleur des textes crée par l’utilisateur.

defaultFont : la taille par défaut du texte crée par l’utilisateur.

Les méthodes de cette classe sont les suivantes :

spawnLine : Cette méthode est déclenchée si l’utilisateur choisi de créer un ligne. Elle vérifie si on a choisi de créer une ligne et pose le premier point sur la clique de la souris. Elle vérifie ensuite si on a posé le deuxième point et l’ajoute sur l’écran.

**3.1.6. TextControler**

La classe TextControler permet de gérer le textField dédié à la création du texte sur la photo. Les méthodes de cette classe sont les suivantes :

createText : Cette méthode teste si on clique sur le bouton « addText » et ajoute le texte présent sur le textField en récupérant la taille de l’espace dédié, elle vérifie donc si cette valeur est numérique et l’applique au texte si elle ne l’est pas on utilise une valeur par défaut.

printTexte : Cette méthode affiche les textes crée par l’utilisateur en appliquant les tailles et couleurs associés.

setReady : Cette méthode permet d’indiquer que le texte est prêt à être affiché.

**3.1.7. Les modifications sur les composants :**

Les modifications sur les composants possibles sont les suivantes :

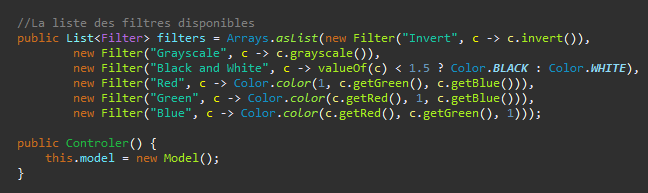
* Changer les dimensions d’un composant : à l’aide de deux TextField qui contiendront les nouvelles dimensions du composant sélectionné on vérifie si ces valeurs sont numériques et on clique sur apply pour appliquer les modifications.
* Changer la position d’un composant : On attrape avec la souris le composant à l’aide des méthodes de la classe ComponentControl et on met à jour sa position.
* Supprimer un composant : On clique sur le bouton delete qui fait appel à la méthode delete et supprime le composant choisie.

**3.2. Les filtres**

Les filtres seront gérés par la classe Filtre qui implémente l’interface Function<Image,Image> , en effet un filtre est une fonction qui prend une image en entré, effectue des modification sur les pixels de cette image et renvoie une image en sortie. La classe possède un attribut colorMap de type Function<Color,Color> cette fonction caractérise chaque filtre et on utilisera cette fonction pour effectuer des changements sur chaque pixel de l’image.

Cette classe possède une seule méthode apply, elle permet d’appliquer le filtre sur l’image. On parcoure l’image pixel par pixel et on applique la fonction du filtre pour changer les valeurs de chaque pixel.

La liste des filtres qui seront disponible pour notre application est la suivante :



* Grayscale : permet de convertir les couleurs à une variation de gris.
* Black and White : teste si la valeur des pixel et supérieur a un seuil et convertit vers noire si on dépasse, vers blanc sinon.
* Red, Green et Blue : maximise la valeur de chaque couleur dans la composition RGB.

On pourra aussi supprimer les filtres appliqués à une image avec la méthode undoFilter de la classe Control cette méthode sera liée au bouton undo. UndoFIlter récupère une version non modifiée de l’image et l’affiche à la place de celle qu’on a modifiée.

**3.3. Importer une image**

L’import des images est effectué grâce à la méthode importAction de la classe Control.

On utilise un FileChooser pour choisir une image locale, pour s’assurer qu’on importe une image on applique un filtre, on ajuste après les dimensions de notre Canvas et on affiche l’image. Cette méthode sera liée au MenuItem Import.

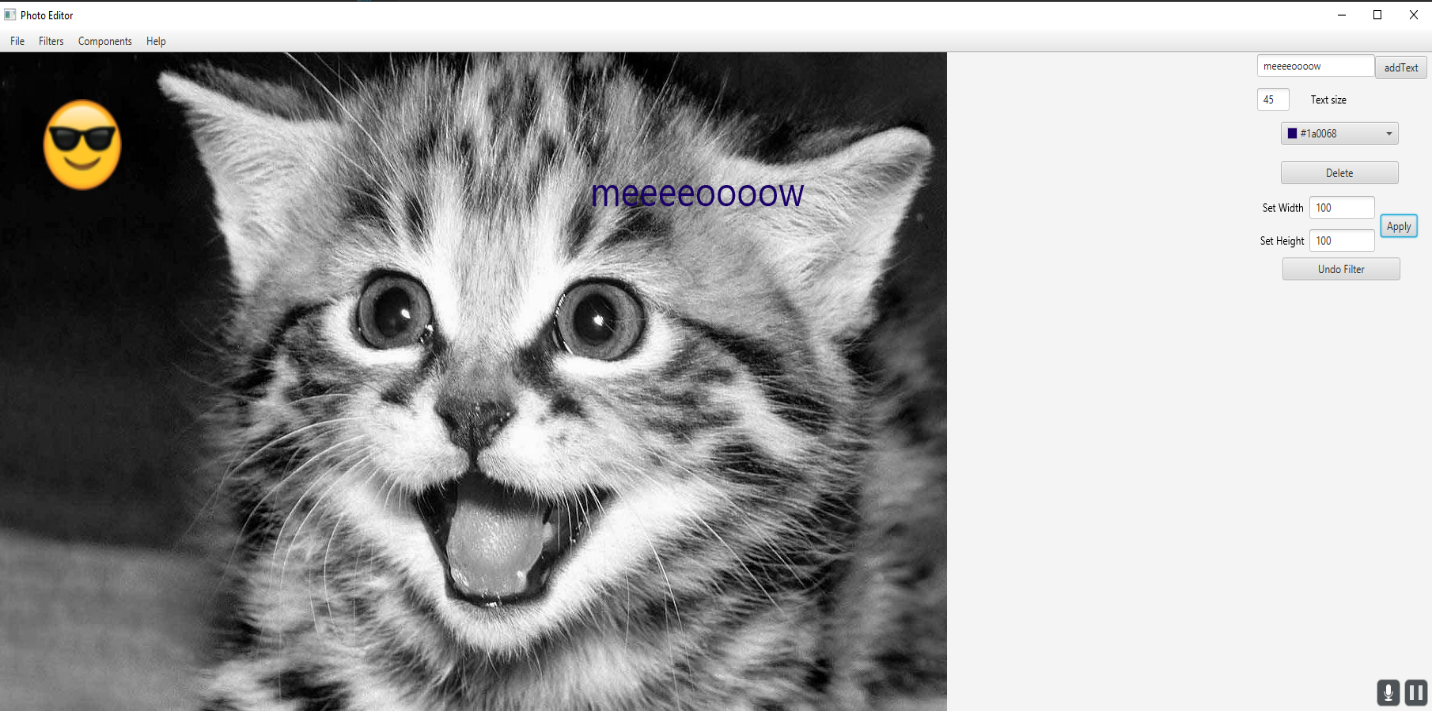
**3.4. Sauvegarder**

Par faute de temps l’implémentation d’un type de sauvegarde personnalisée n’as pas été réaliser néanmoins la méthode onSave de classe Control, permet de prendre une capture de notre Canvas et le sauvegarder. Cette capture contiendra aussi les modifications effectuées sur l’image.

**4.Résultat**

**4.1 Grille des fonctionnalités**

|  |  |
| --- | --- |
| Ajouter des composants | Réalisé |
| Déplacer/supprimer/redimensionner | Réalisé |
| Ajouter des filtres aux photos | Réalisé |
| Mettre un composant au premier plan ou au dernier plan | Non Réalisé |
|  Sauvegarder la photo | Réalisé |

**4.2** **L’interface graphique**

**5. Conclusion**

On a pu réaliser à travers ce projet une application d’édition de photo simple qui permet d’effectuer des modifications comme l’ajout des composants (forme géométrique, emoji, texte), application des filtres… On peut aussi sauvegarder notre photo pour retravailler dessous. Ce projet nous a permis d’approfondir nos connaissances en Java et d’apprendre de nouvelles notions de javaFx ainsi que l’utilisation de sceneBuilder pour la gestion des fichier FXML.