
Projet de semestre (PRO)
Editeur d'image GEMMS

Auteur :

Guillaume MILANI
Edward RANSOME
Mathieu MONTEVERDE
Michael SPIERER
Sathiya KIRUSHNAPILLAI

Client :

René RENTSCH

Référent :

René RENTSCH

Table des matières

1	Introduction	1
2	Objectif	1
3	Conception & Architecture	1
3.1	Technologies utilisées	1
3.1.1	Java 8	1
3.1.2	JavaFX 8	1
3.1.3	Scene Builder 8.3.0	2
3.1.4	Maven	2
3.1.5	Git	2
3.1.6	GitHub	2
3.2	Comparaison de l'interface finale avec notre mock-up	3
4	Description technique	4
4.1	Structure	4
4.2	Serialisation	5
4.3	Sauvegarde	6
4.4	Workspace et liste des calques	6
4.5	Copier-coller	6
4.6	Historique	6
4.7	Positionnement	6
4.8	Outils	7
4.8.1	Hiérarchie des outils	7
4.8.2	Réglage des outils	8
4.8.3	Pinceau et gomme	9
4.8.4	Pipette	10
4.8.5	Modification de texte	10
4.8.6	Symétries	10
4.8.7	Déplacement	10
4.8.8	Rotation	10
4.8.9	Redimensionnement	10
4.8.10	Sélection	10
4.8.11	Rognage	10
4.9	Effets	10
4.9.1	Noir et blanc	11
4.9.2	Tint	11
4.9.3	Barres glissantes	11
4.10	Remise à zéro	11
5	Conclusion	11

Table des figures

1	Architecture de JavaFX	2
2	Mock-up de l'interface	3
3	Interface finale du programme	4
4	Diagramme de la sérialisation simplifié	5
5	Diagramme simplifié de la hiérarchie des outils	8
6	Composant de réglages de l'outil de modification texte	8
7	Diagramme simplifié des réglages d'outils	9

1 Introduction

GEMMS est un éditeur d'images réalisé en Java en se basant sur les fonctionnalités graphiques de JavaFX 8. Il a été réalisé dans le cadre de la branche PRO (Projet de semestre) de la deuxième année d'informatique logicielle de la Haute-École d'Ingénierie et Gestion du canton de Vaud (HEIG-VD). Le programme a été élaboré sur une durée de 14 semaines aboutissant le 31 Mai 2017.

2 Objectif

L'application GEMMS a été conçue pour éditer des images de manière rapide, simple et intuitive. Le programme ne nécessite pas d'apprentissage particulier, comme certains programmes plus lourds comme Adobe Photoshop ou encore GIMP.

L'interface est propre, avec des icônes représentant les différents outils et actions possibles dans le programme en essayant de minimiser les menus déroulants surchargés. Des infobulles donnent une description concise de chaque outil lorsqu'on passe la souris dessus.

Bien que plus simple que les applications lourdes mentionnées ci-dessus, un concept très important dans l'édition d'image, les calques, est conservé. Certains éditeurs très basiques comme Paint sous Windows ne fournissent pas cette fonctionnalité. Les calques permettent de superposer des images, du texte, ou des canevas et de les déplacer, modifier ou effacer indépendamment les uns des autres. Lors de l'exportation du projet vers un format image, les calques sont aplatis et l'image est exportée en perdant ces informations.

Un projet GEMMS ne peut cependant pas uniquement être exporté en tant qu'image, mais sauvegardée dans un fichier de projet « .gemms ». Ce type de fichier peut être ouvert par l'application pour restaurer tout le projet en cours, recréant chaque calque ainsi que les transformations effectuées dessus.

3 Conception & Architecture

3.1 Technologies utilisées

3.1.1 Java 8

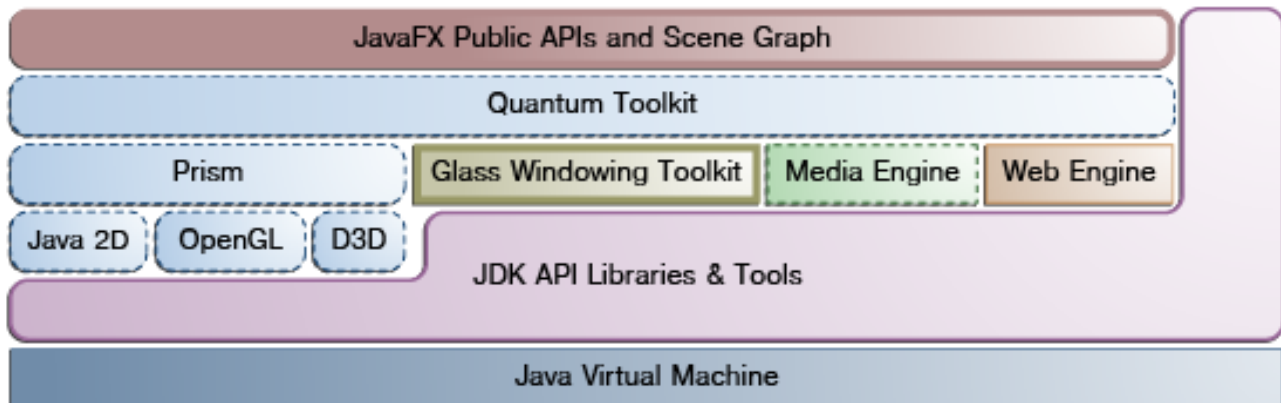
Parmi les deux langages de haut niveau proposés pour l'élaboration de ce projet (Java ou C++), nous avons choisi Java pour sa portabilité, sa sécurité et ses performances. De plus, la dernière version de Java propose la librairie graphique JavaFX qui correspond en tout point à notre projet. Enfin, notre équipe est également plus habile à programmer à l'aide du langage Java.

3.1.2 JavaFX 8

JavaFX, successeur de Swing, est la librairie de création d'interface graphiques officielle de Java. La version 8, utilisée pour ce projet, ajoute de nouvelles fonctionnalités et est la plus récente version utilisable avec Scene Builder.

Comme vous pouvez le voir sur la figure 1, BLA BLA BLA A COMPLETER, A PARLER DE CSS ETC

FIGURE 1 – Architecture de JavaFX



3.1.3 Scene Builder 8.3.0

Scene Builder de Gluon permet de manipuler des objets JavaFX graphiquement et exporter ceux-ci dans un fichier .fxml interprétable par la librairie graphique. L'interface de base a été conçue lors de l'élaboration du cahier des charges pour présenter un exemple de l'interface de l'application finale. Plusieurs mock-ups ont été présentés et c'est sur ceux-ci que nous nous sommes basés pour construire, grâce à Scene Builder, une base d'interface sur la laquelle nous avons rajouté des composants et fonctionnalités tout au long de l'élaboration de l'application. La flexibilité de JavaFX permet d'ajouter des éléments via un fichier externe fxml mais aussi directement dans le code, ce que nous avons aussi utilisé.

3.1.4 Maven

Pour la compilation du projet et l'importation aisée de celui-ci dans un nouvel environnement de travail, nous avons utilisé l'outil Maven de Apache. TODO TODO TODO TODO TODO COMPLETER

3.1.5 Git

Git est un logiciel de gestion de version utilisé pour permettre de stocker tous les fichiers du projet ainsi que toutes les modifications leur ayant été apportés depuis leur création. Pour chaque nouvelle fonctionnalité, nous avons procédé par la création d'une branche à partir de la branche principale (une version fonctionnelle du programme, contenant les fonctionnalités implémentées et testées). Ces nouvelles branches permettent de développer les fonctionnalités du programme indépendamment et de les ajouter à la branche principale une fois inspectées et testées par plusieurs membres de l'équipe.

3.1.6 GitHub

Github est un service web permettant de parcourir visuellement l'historique Git ainsi que de fournir des outils de gestion de Git. Notamment, pour chaque fonctionnalité ou chaque bug découvert, une "issue" (un problème) peut être ouverte et assignée à un ou plusieurs membres de l'équipe. Dès la fin de l'élaboration du planning de notre projet, des issues ont été assignées à chaque développeur. Celles-ci ont permis de mieux se fier au planning et toujours avoir en vue ce qu'il restait à implémenter.

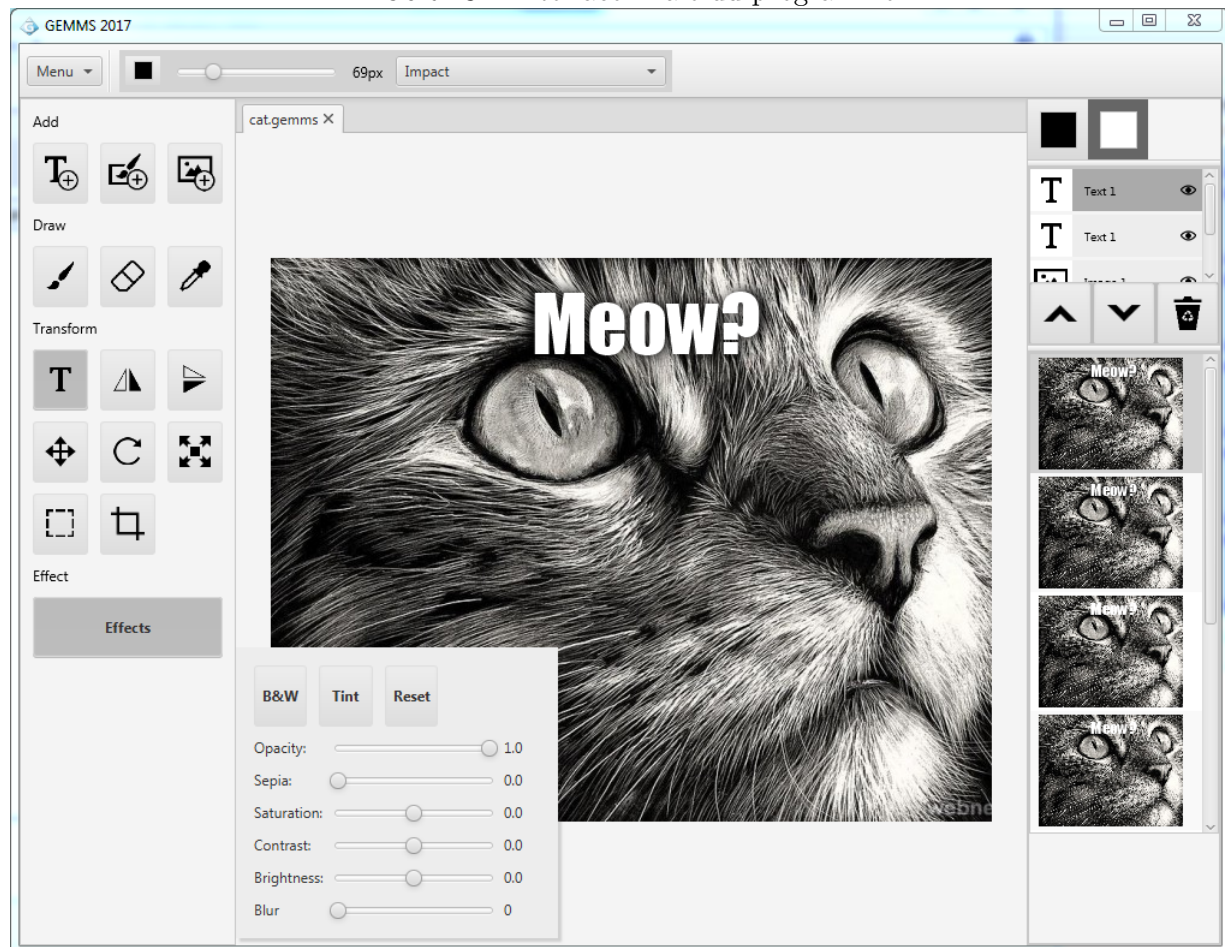
3.2 Comparaison de l'interface finale avec notre mock-up

Le mock-up de l'interface a été conçu au début du projet en prenant compte de toutes les fonctionnalités que le programme devait fournir. Voici une comparaison de celui-ci avec l'interface finale.

FIGURE 2 – Mock-up de l'interface



FIGURE 3 – Interface finale du programme



Comme le montrent les figures 2 et 3, l'interface conçue lors de l'élaboration du cahier des charges à été reprise presque entièrement pour notre programme. Les différences majeures sont les paramétrages des outils, qui ont lieu en haut de l'interface à côté du menu déroulant, ainsi que les filtres & effets, qui ont lieu dans une fenêtre qui peut être ouverte ou fermée à souhait pour rendre l'interface moins chargée.

4 Description technique

Comme cité précédemment, notre application a été codée à l'aide de la librairie JavaFX. Ainsi, toute notre implémentation technique est basée sur cette dernière.

4.1 Structure

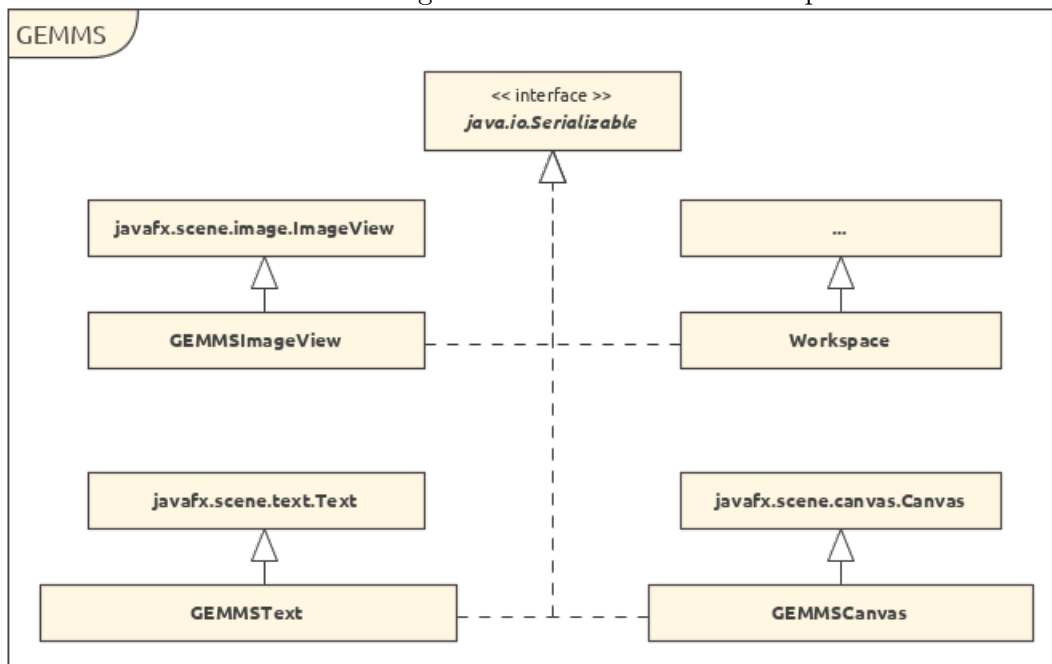
JavaFX utilise des fichiers FXML pour séparer la logique de la vue TODO TODO TODO ImageView, Canvas et Text Controller

4.2 Serialisation

En Java, la sérialisation s'effectue à l'aide de l'interface « `Serializable` ». Par conséquent, chaque classes de Java implémentant cette dernière telle que « `String` », peut être sérialisé et désérialisé à volonté. Cependant, la majorité des classes JavaFX n'implémente pas cette interface. En effet, cette librairie utilise grandement des mécanismes et des liaisons dynamiques tel que les listeners qui sont pour l'instant des sous-systèmes non-sérialisable. C'est pourquoi, JavaFX contient peu d'objet sérialisable.

Pour combler ce manque, nous devons nous même implémenter la sérialisation des classes JavaFX que nous sommes susceptible d'utiliser.

FIGURE 4 – Diagramme de la sérialisation simplifié



Sur la figure 4, nous pouvons voir un diagramme simplifié de l'implémentation de la sérialisation. Dans notre application, nous allons utiliser des classes de base telles que `ImageView`, `Text`, `Canvas`, `Color`, etc. Nous devons donc spécialiser ces classes afin qu'elles puissent implémenter l'interface « `Serializable` ». Toutefois, certaines classes comme « `Color` » ne sont malheureusement pas spécialisable. Il faut donc sérialiser les paramètres un par un à l'aide des accesseurs et mutateurs de cette dernière.

Étant donné que les classes JavaFX possèdent énormément de fonctionnalités, sérialiser l'entier de celles-ci nous demanderait beaucoup trop de temps. C'est pourquoi nous nous contentons uniquement des paramètres utilisés au sein du projet tel que la largeur, la hauteur, la position, etc.

Listing 1 – Exemple de sérialisation

```
private void writeObject(ObjectOutputStream s) {
    // ...

    // Write the size
    s.writeDouble(width);
    s.writeDouble(height);
}
```

```
    // ...  
}  
  
private void readObject(ObjectInputStream s) {  
    // ...  
  
    // Get the size of the canvas  
    double width = s.readDouble();  
    double height = s.readDouble();  
  
    // ...  
}
```

Bien que la sérialisation soit possible, ceci engendre des contraintes et des pertes de performances. Par exemple, les classes spécialisées ne peuvent plus étendre d'une classe commune et bénéficier de ses méthodes. De plus, les objets comme Canvas et ImageView devront sérialiser pixel par pixel, ce qui peut être long et volumineux selon la taille.

4.3 Sauvegarde

La sauvegarde d'un document utilise la sérialisation des objets. Comme mentionné précédemment, la sérialisation de certaines classes peut être volumineux. Ainsi, les données sont compressées dans le format GZIP.

4.4 Workspace et liste des calques

TODO TODO TODO

4.5 Copier-coller

TODO TODO TODO

4.6 Historique

Pour garder un historique de chaque action effectuée, on utilise la sérialisation des composants présentée précédemment. A la fin de chaque action modifiant l'espace de travail, une fonction va être appelée permettant de sauvegarder intégralement l'espace de travail courant et le placer sur une pile. A chaque détection de la commande Ctrl + Z, la sauvegarde sera chargée et la modification sera donc effacée. De même, à la détection de la commande Ctrl + Y, on va charger un espace de travail plus récent (s'il y en a un, c'est-à-dire si le Ctrl + Y était précédé d'un Ctrl + Z).

4.7 Positionnement

TODO TODO TODO

4.8 Outils

JavaFX offre (entre autres) les événements `MousePressed`, `MouseDragged` et `MouseReleased`. Ils correspondent respectivement à l'action de presser la souris, de la déplacer en gardant le clic gauche enfoncé ou de relâcher le clic gauche de la souris.

4.8.1 Hiérarchie des outils

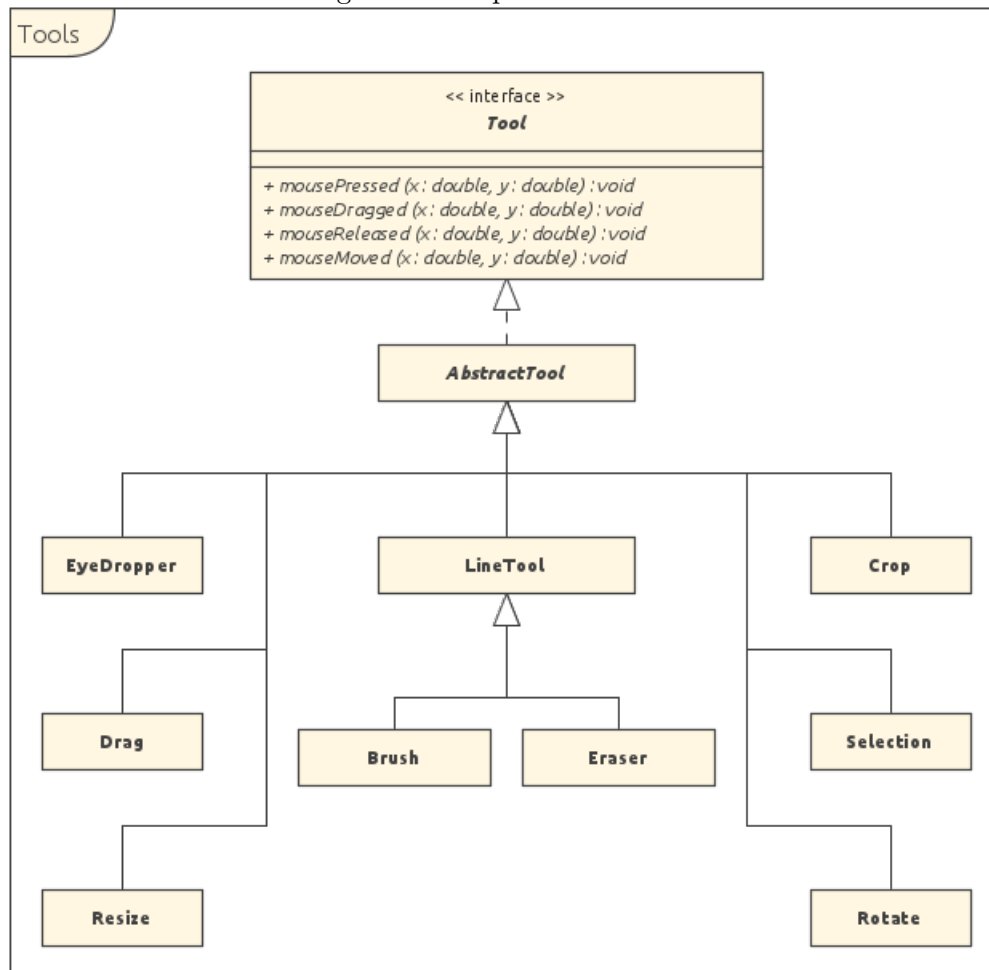
La plupart des outils de l'application fonctionnent grâce à ces trois événements. On pensera notamment au pinceau qui doit dessiner un trait en suivant la souris lors d'un `MouseDragged`.

Comme on peut le voir sur la figure 5, les outils implémentent une interface `Tool`, possédant des méthodes correspondant à ces événements. Au long de l'exécution du programme, le `Workspace` garde une référence vers un outil considéré actif (qui peut aussi être référence nulle), et lorsqu'il détecte un des événements cités plus haut, il se charge d'appeler la ou les méthodes correspondantes de cet outil.

Lorsque l'utilisateur clique sur un bouton pour activer un outil, le programme crée une nouvelle instance de ce type d'outil, et le `Workspace` utilise cet outil pour traiter les événements `MousePressed`, `MouseDragged` ou `MouseReleased`.

D'autres outils plus simples, comme la symétrie horizontale ou verticale ou les effets de couleurs sont implémentés en ajoutant une action à des composants de bases de JavaFX comme des `Button` ou des `Slider`.

FIGURE 5 – Diagramme simplifié de la hiérarchie des outils



4.8.2 Réglage des outils

Certains de ces outils nécessitent d'être paramétrés en temps réel en fonction des calques sélectionnés par l'utilisateur. Par exemple, la taille de la gomme est stockée dans l'objet représentant l'outil, et est utilisée pour déterminer la taille du rectangle à effacer. L'utilisateur peut la régler au moyen d'un Slider (il s'agit d'un composant JavaFX). Les mêmes besoins concernent la gestion de la couleur du pinceau, de la taille et de la police de l'outil de modification de texte. La figure 6 illustre l'apparence des réglages de l'outil de modification de texte.

FIGURE 6 – Composant de réglages de l'outil de modification texte



Ces réglages doivent pouvoir s'adapter à un outil existant, comme par exemple récupérer la taille actuelle du pinceau et la garder en mémoire.

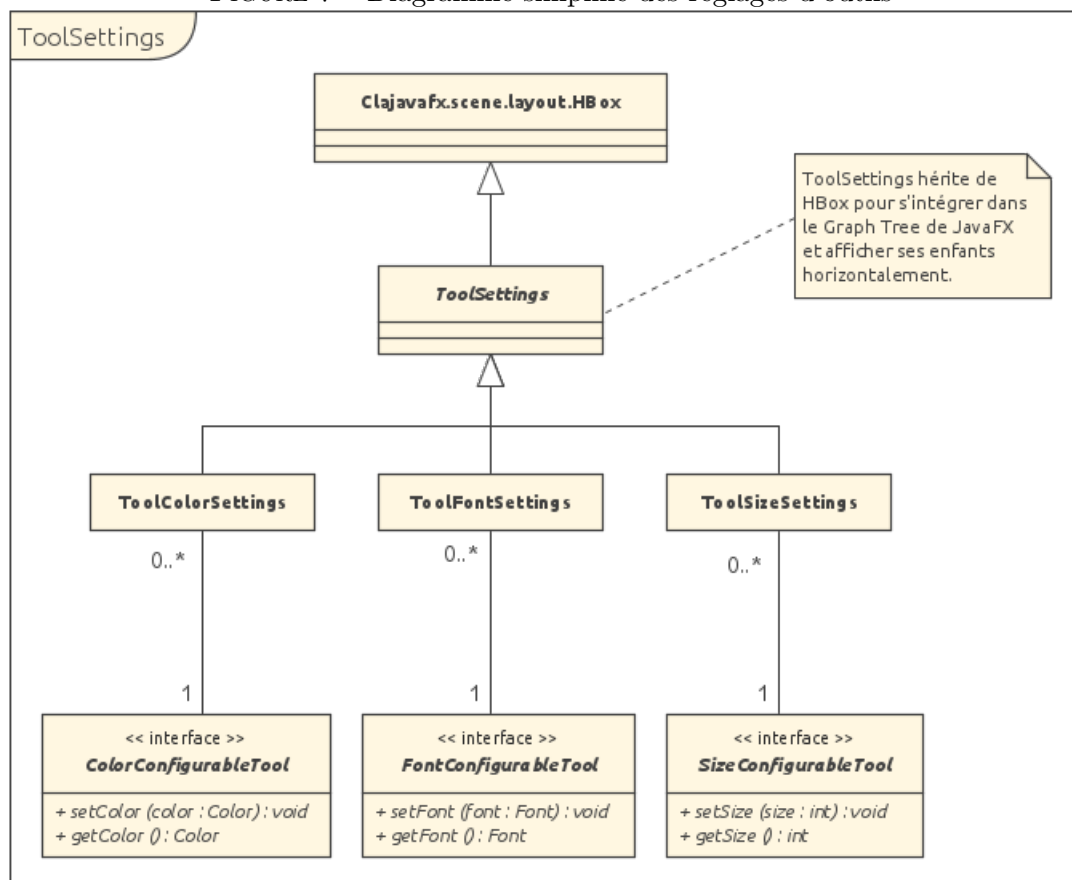
Pour ce faire, les réglages sont gérés au moyen d'une hiérarchie de classes (se référer à la figure 7), qui sont en fait des spécialisations de composants JavaFX permettant à l'utilisateur de paramétrer les

outils, et d'une série d'interfaces représentant des outils pouvant être paramétrés sur divers aspects (la taille, la couleur, la police et ainsi de suite).

Ainsi un objet `ToolFontSettings` permet de paramétrer la Font (les paramètres de police d'écriture) d'un objet implémentant l'interface `FontConfigurableTool`. Dans le cas de l'outil texte, qui implémente cette interface, les réglages doivent s'adapter aux paramètres d'un calque texte et ainsi se mettre à jour en temps réel.

Lorsque l'utilisateur modifie la valeur du Slider contenu dans l'objet `ToolFontSettings`, celui-ci met à jour sa cible `SizeConfigurableTool` en temps réel (ici, l'outil de modification de texte).

FIGURE 7 – Diagramme simplifié des réglages d'outils



4.8.3 Pinceau et gomme

Le pinceau et la gomme ont un comportement et une implémentation quasiment identiques. Dans le programme GEMMS, ce sont les classes `Brush` et `Eraser` qui se chargent d'implémenter ces fonctionnalités. Toutes deux héritent d'une classe commune : `LineTool`.

Pour ces deux outils, la problématique était la suivante :

L'événement `MouseDragged` est déclenché à intervalles réguliers, tant que l'utilisateur effectue cette action. Pour chaque répétition, il est facile de garder trace de la position de la souris au dernier événement et à l'événement actuel. Connaissant ces deux points, il est possible de dessiner une droite.

Pour ce faire, nous avons utilisé l'algorithme de tracé de segment de Bresenham. Mis au point en 1962, cet algorithme permet de déterminer quels pixels sont à colorer pour relier harmonieusement deux points par une ligne droite.

Il existe de nombreuses implémentations de cet algorithme, et nous avons choisi l'implémentation compacte, que l'on peut trouver sur la page allemande de l'article Wikipédia dédié à ce sujet [2].

La classe LineTool se charge donc d'implémenter cet algorithme et pour chaque pixel à colorer, elle appelle une méthode abstraite drawPixel que ses sous classes se chargent de définir. Ainsi, Brush dessiner un disque correspondant à la taille du pinceau, et Eraser efface un carré de pixel, correspondant à la taille de la gomme.

4.8.4 Pipette

Le rôle de l'outil pipette est de permettre à l'utilisateur de choisir une couleur en la prélevant sur un élément existant du document. Il s'agirait typiquement de récupérer la couleur d'un calque GEMMSText, GEMMSCanvas ou GEMMSImage.

Dans le cas d'un texte, la pipette retourne simplement la couleur de celui-ci. Dans le cas d'un objet de type GEMMSCanvas ou d'une GEMMSImage, l'outil lit le pixel exact cliqué par l'utilisateur, si celui-ci se trouve à l'intérieur des bornes du calque sélectionné.

4.8.5 Modification de texte

L'outil de modification de texte permet à l'utilisateur de modifier les propriétés d'un calque texte existant. Lorsque il clique dessus, si le clic est effectué dans les bornes visuelles du texte, l'outil ouvre une fenêtre de dialogue invitant à changer le contenu du texte.

Les réglages de couleur et de police de cet outils utilisent la structure expliquée à la section 4.8.2 en utilisant des objets de type ToolColorSettings et ToolFontSettings.

4.8.6 Symétries

4.8.7 Déplacement

4.8.8 Rotation

4.8.9 Redimensionnement

4.8.10 Sélection

4.8.11 Rognage

4.9 Effets

La section « Effet » permet d'appliquer des effets colorimétriques, régler la transparence et ajouter un flou à un ou plusieurs calques. Chaque calque qui doit être modifié passe par une vérification : s'il n'a aucun effet appliqué, on lui applique trois effets : un ColorAdjust, un SepiaTone et un GaussianBlur permettant d'effectuer respectivement des ajustements de la couleur, un effet sépia, et un flou. Ces

effets sont initialement tous réglés pour n'effectuer aucun changement visuel. Ceci permet, lors d'un futur changement des effets, de simplement devoir faire varier les paramètres de ces effets.

4.9.1 Noir et blanc

Le bouton « B&W » a pour effet de régler la saturation de l'image à sa valeur minimale, donnant pour effet une image en nuances de gris.

4.9.2 Tint

Applique une teinture de la couleur sélectionnée à l'image. Ceci est fait en calculant une valeur de la teinte et en l'appliquant à l'effet ColorAdjust.

4.9.3 Barres glissantes

Les barres glissantes (« Sliders » JavaFX) permettent d'ajuster tous les autres effets implémentés. La transparence règle directement un attribut « Opacity » d'un noeud. La saturation, le contraste et la luminosité peuvent être ajustés en modifiant l'effet ColorAdjust. L'effet sépia et le flou ajustent respectivement l'effet SepiaTone et GaussianBlur.

4.10 Remise à zéro

Le bouton « Reset » permet de remettre à zéro tous les effets sur tous les calques sélectionnés, les restaurant à leur état visuel initial.

5 Conclusion

Le programme fourni connaît certains problèmes de performances dû aux choix d'implémentation effectués. La taille des sauvegardes même après compression reste grande, ce qui implique que notre façon de garder un historique des changements est gourmand. Après chaque changement, on effectue une sauvegarde intégrale de l'espace de travail pour qu'il puisse être restauré par la suite. Ainsi, si l'on travaille sur des grandes images, on peut voir apparaître un délai après chaque action ou chaque utilisation d'un Ctrl + Z. En plus de ces ralentissements, la quantité de mémoire utilisée peut s'avérer problématique.

À notre avis, ceci est le plus grand défaut de notre application. Pour corriger ceci, l'implémentation de l'historique aurait pu être effectuée différemment : pour chaque action effectuée, il aurait été possible de sauvegarder un objet permettant d'effectuer l'action inverse. Par exemple, pour une symétrie horizontale, une implémentation possible aurait été d'empiler sur l'historique un objet effectuant une autre symétrie horizontale. Avec cette façon de faire, on évite de sauvegarder l'intégralité de l'espace de travail et on ne stocke qu'un objet très petit. Le problème de cette implémentation est sa difficulté d'implémentation. Pour chaque transformation, il doit être possible de coder son inverse et stocker les calques sur lesquelles il a été effectué. Avec un grand nombre d'actions possibles dans le programme cela représente une charge de travail considérable et, bien que plus performant, le temps nécessaire à son implémentation n'était simplement pas disponible.

Pour ce qui concerne le travail de groupe, il à été difficile au début du projet de répartir des tâches pouvant être effectuées individuellement. Vu que le programme est composé entièrement d'une unique interface graphique, il existe beaucoup de dépendances entre les fonctionnalités. Travailler sans se marcher dessus et sans devoir attendre des fonctionnalités n'a pas toujours été facile. Notre planification des tâche initiale à cependant pris en compte ces dépendances et l'ordre d'implémentation des fonctionnalités à permis, dès l'élaboration de l'interface principale, d'éviter la plus part des conflits. Cependant, il s'est de temps en temps avéré qu'un changement de l'un des membres du groupe affecte les outils d'autres membres, notamment en ce qui concerne les systèmes de coordonnées.

Au final, le programme réalisé nous semble correspondre au cahier des charges proposé et, malgré certains problèmes de performance, est un outil qui nous restera utile en dehors du cadre de ce cours.

Références

- [1] R. M. Star. *Foo Bar Baz*. MIT Press, Cambridge, MA, 1989.
- [2] Wikipedia. Bresenham-algorithmus, 2007.