Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud (HEIG-VD)

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

29 mai 2017

Table des matières

Table des figures

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

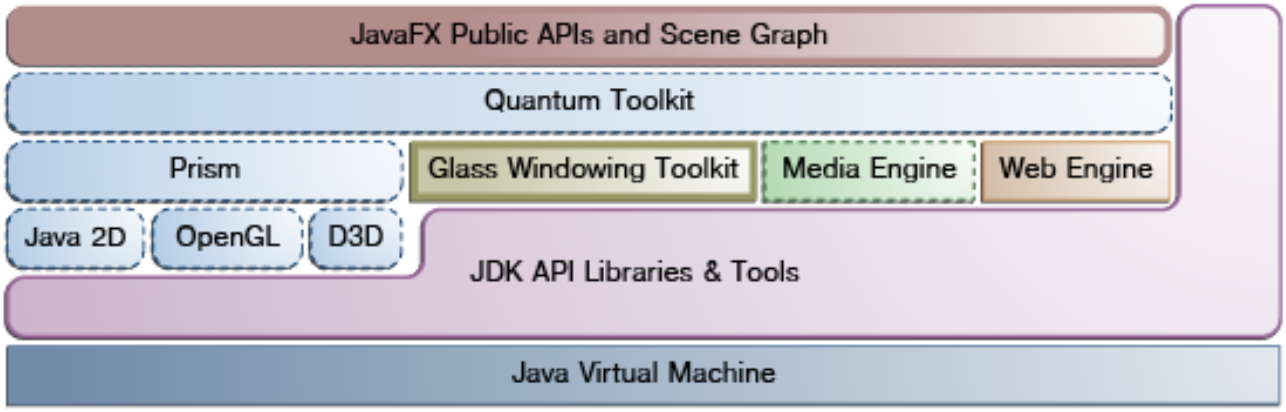
Parmi les deux langages de haut niveau proposés pour l'élaboration de ce projet (Java ou C++), nous avons choisi Java pour sa portabilité, sa sécurité et ses performances. De plus, la dernière version de Java propose la librairie graphique JavaFX qui correspond en tout point à notre projet. Enfin, notre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

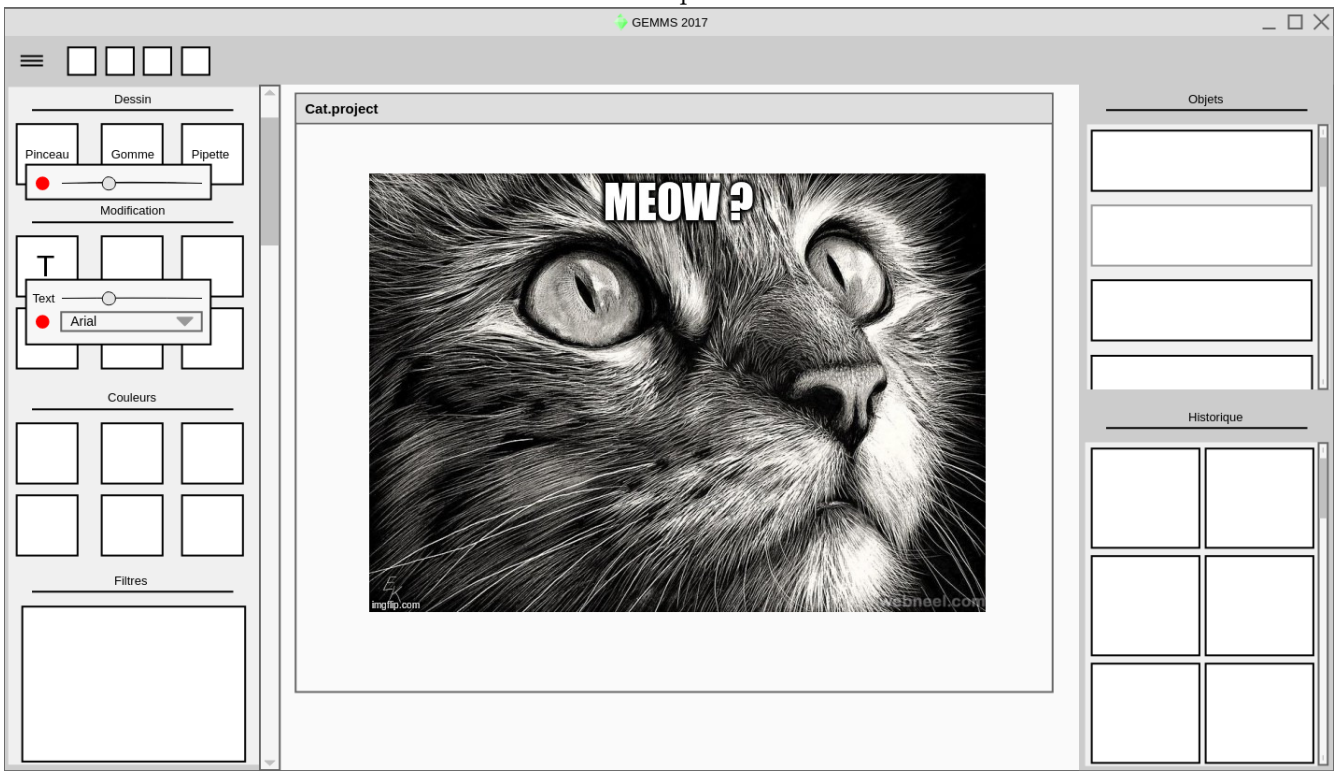
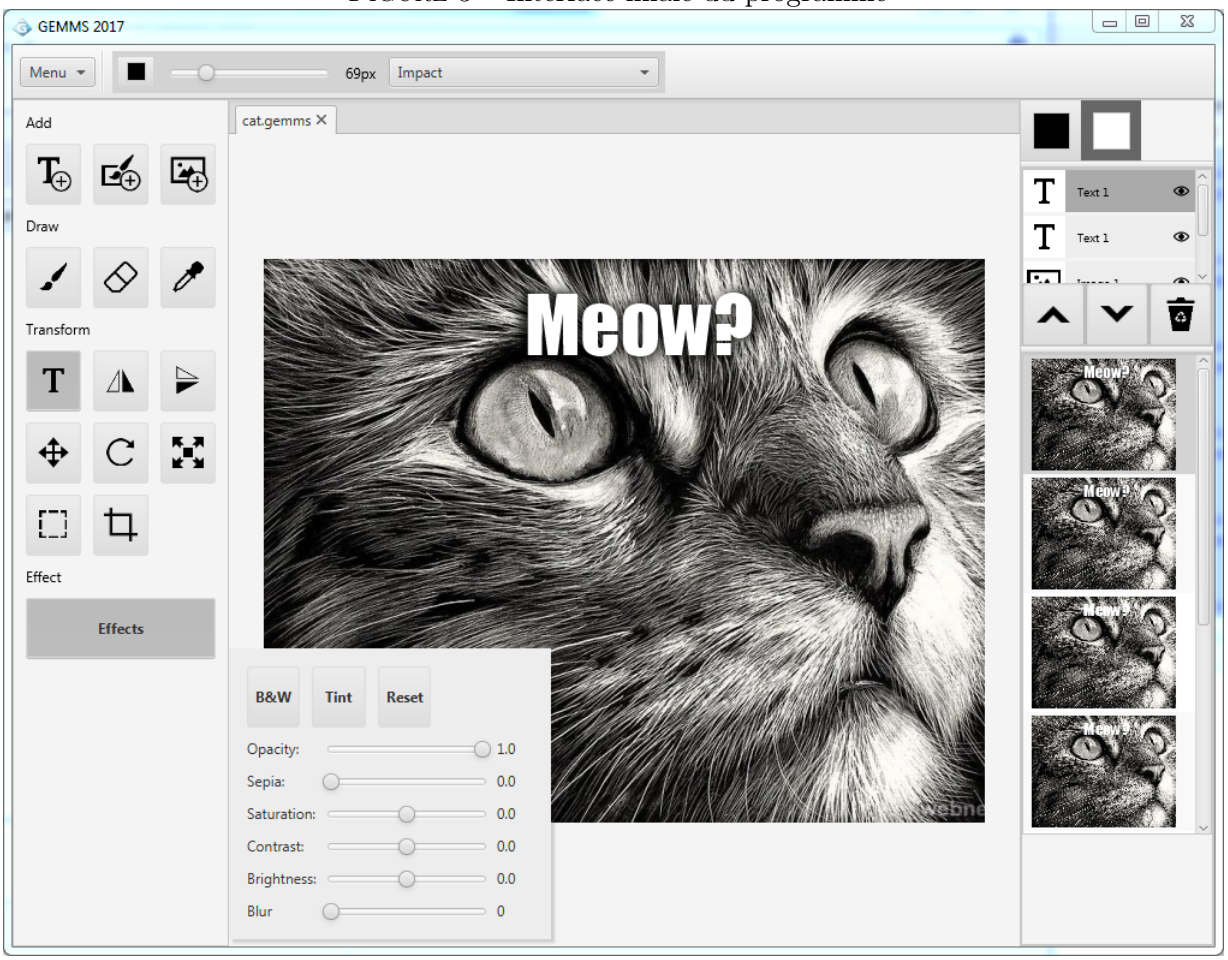


FIGURE 3 – Interface finale du programme



Comme le montrent les figures 2 et 3, l'interface conçue lors de l'élaboration du cahier des charges à été repris presque entièrement pour notre programme. Les différences majeurs sont les paramétrages des outils, qui ont lieux en haut de l'interface à coté du menu déroulant, ainsi que les filtres & effets, qui ont lieux 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 codé à l'aide de la librairie JavaFX. Ainsi, toute notre implémentation technique est basée sur cette dernière.

4.1 Structure

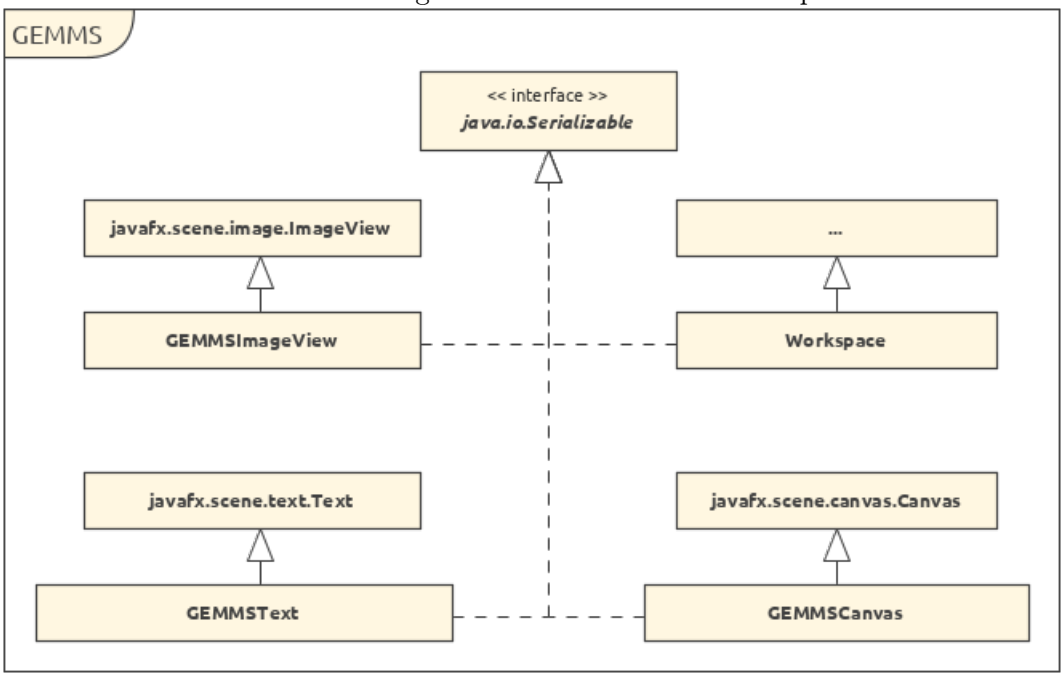
JavaFX utilise des fichiers FXML pour séparer la logique de la vue TODO TODO TODO ImageView, Canvas et Text Controller

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 utilisé 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 à un, c'est-à-dire si le Ctrl + Y était précédé d'un Ctrl + Z).

4.7 Positionnement

TODO TODO TODO

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 à 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.

–travail en groupe sur une interface chaud

Références