
Projet de semestre (PRO)
Editeur d'image GEMMS

Auteurs :

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	2
3.1.3	Scene Builder 8.3.0	2
3.1.4	Maven	2
3.1.5	Git	2
3.1.6	GitHub	3
3.2	Comparaison de l'interface finale avec notre mock-up	3
4	Description technique	4
4.1	Structure	4
4.1.1	Structure de JavaFX	4
4.2	Serialisation	5
4.3	Sauvegarde	7
4.4	Workspace et liste des calques	7
4.4.1	Structure d'un Workspace	7
4.4.2	Types de calques	8
4.4.3	Gestion des calques	8
4.5	Copier-coller	10
4.5.1	Snapshot	10
4.5.2	Viewport	10
4.5.3	Sérialisation	10
4.5.4	Coller le contenu du presse-papier	10
4.6	Historique	11
4.6.1	Historique visuel	11
4.7	Positionnement	12
4.8	Outils	12
4.8.1	Hiérarchie des outils	12
4.8.2	Réglage des outils	13
4.8.3	Pinceau et gomme	14
4.8.4	Pipette	15
4.8.5	Modification de texte	15
4.8.6	Symétries	16
4.8.7	Déplacement	16
4.8.8	Rotation	16
4.8.9	Redimensionnement	16
4.8.10	Sélection	16
4.8.11	Rognage	17
4.9	Effets	17
4.9.1	Noir et blanc	17
4.9.2	Tint	17

4.9.3	Barres glissantes	17
4.9.4	Remise à zéro	17
5	Conclusion	17
	Glossary	18
6	Annexes	20
A	Journal de travail - Edward Ransome	1
A.1	Semaine 1 : 20 février - 24 février	1
A.2	Semaine 2 : 27 février - 3 mars	1
A.3	Semaine 3 : 6 mars - 10 mars	1
A.4	Semaine 4 : 13 mars - 17 mars	1
A.5	Semaine 5 : 20 mars - 24 mars	1
A.6	Semaine 6 : 27 mars - 31 mars	1
A.7	Semaine 7 : 3 avril - 7 avril	1
A.8	Semaine 8 : 10 avril - 14 avril	1
A.9	Semaine 9 : 24 avril - 28 avril	2
A.10	Semaine 10 : 1 mai - 5 mai	2
A.11	Semaine 11 : 8 mai - 12 mai	2
A.12	Semaine 12 : 15 mai - 19 mai	2
A.13	Semaine 13 : 22 mai - 26 mai	2
A.14	Semaine 14 : 29 mai - 2 juin	2
B	Journal de travail - Mathieu Monteverde	3
B.1	Semaine 1 : 20 février - 24 février	3
B.2	Semaine 2 : 27 février - 3 mars	3
B.3	Semaine 3 : 6 mars - 10 mars	3
B.4	Semaine 4 : 13 mars - 17 mars	3
B.5	Semaine 5 : 20 mars - 24 mars	3
B.6	Semaine 6 : 27 mars - 31 mars	3
B.7	Semaine 7 : 3 avril - 7 avril	3
B.8	Semaine 8 : 10 avril - 14 avril	3
B.9	Semaine 9 : 24 avril - 28 avril	4
B.10	Semaine 10 : 1 mai - 5 mai	4
B.11	Semaine 11 : 8 mai - 12 mai	4
B.12	Semaine 12 : 15 mai - 19 mai	4
B.13	Semaine 13 : 22 mai - 26 mai	4
B.14	Semaine 14 : 29 mai - 2 juin	4
C	Journal de travail - Sathiya Kirushnapillai	5
C.1	Semaine 1 : 20 février - 24 février	5
C.2	Semaine 2 : 27 février - 3 mars	5
C.3	Semaine 3 : 6 mars - 10 mars	5
C.4	Semaine 4 : 13 mars - 17 mars	5
C.5	Semaine 5 : 20 mars - 24 mars	5
C.6	Semaine 6 : 27 mars - 31 mars	5
C.7	Semaine 7 : 3 avril - 7 avril	5

C.8	Semaine 8 : 10 avril - 14 avril	5
C.9	Semaine 9 : 24 avril - 28 avril	5
C.10	Semaine 10 : 1 mai - 5 mai	6
C.11	Semaine 11 : 8 mai - 12 mai	6
C.12	Semaine 12 : 15 mai - 19 mai	6
C.13	Semaine 13 : 22 mai - 26 mai	6
C.14	Semaine 14 : 29 mai - 2 juin	6

D Journal de travail - Michael Spierer 7

D.1	Semaine 1 : 20 février - 24 février	7
D.2	Semaine 2 : 27 février - 3 mars	7
D.3	Semaine 3 : 6 mars - 10 mars	7
D.4	Semaine 4 : 13 mars - 17 mars	7
D.5	Semaine 5 : 20 mars - 24 mars	7
D.6	Semaine 6 : 27 mars - 31 mars	7
D.7	Semaine 7 : 3 avril - 7 avril	7
D.8	Semaine 8 : 10 avril - 14 avril	7
D.9	Semaine 9 : 24 avril - 28 avril	7
D.10	Semaine 10 : 1 mai - 5 mai	8
D.11	Semaine 11 : 8 mai - 12 mai	8
D.12	Semaine 12 : 15 mai - 19 mai	8
D.13	Semaine 13 : 22 mai - 26 mai	8
D.14	Semaine 14 : 29 mai - 2 juin	8

E Journal de travail - Guillaume Milani 9

E.1	Semaine 1 : 20 février - 24 février	9
E.2	Semaine 2 : 27 février - 3 mars	9
E.3	Semaine 3 : 6 mars - 10 mars	9
E.4	Semaine 4 : 13 mars - 17 mars	9
E.5	Semaine 5 : 20 mars - 24 mars	9
E.6	Semaine 6 : 27 mars - 31 mars	9
E.7	Semaine 7 : 3 avril - 7 avril	9
E.8	Semaine 8 : 10 avril - 14 avril	9
E.9	Semaine 9 : 24 avril - 28 avril	10
E.10	Semaine 10 : 1 mai - 5 mai	10
E.11	Semaine 11 : 8 mai - 12 mai	10
E.12	Semaine 12 : 15 mai - 19 mai	10
E.13	Semaine 13 : 22 mai - 26 mai	10
E.14	Semaine 14 : 29 mai - 2 juin	10

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.

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, le concept de calques, très important dans l'édition d'image, 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 également sauvegardé 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.

Le programme doit pouvoir permettre d'effectuer les opérations classiques de manipulation d'image comme le dessin, l'ajout de texte, l'ajout d'image, le changement de taille, le rognage et bien d'autres. Une liste exhaustive des fonctionnalités implémentées ainsi que leur instructions d'utilisation peuvent être trouvées dans le manuel utilisateur.

Conception & Architecture

Technologies utilisées

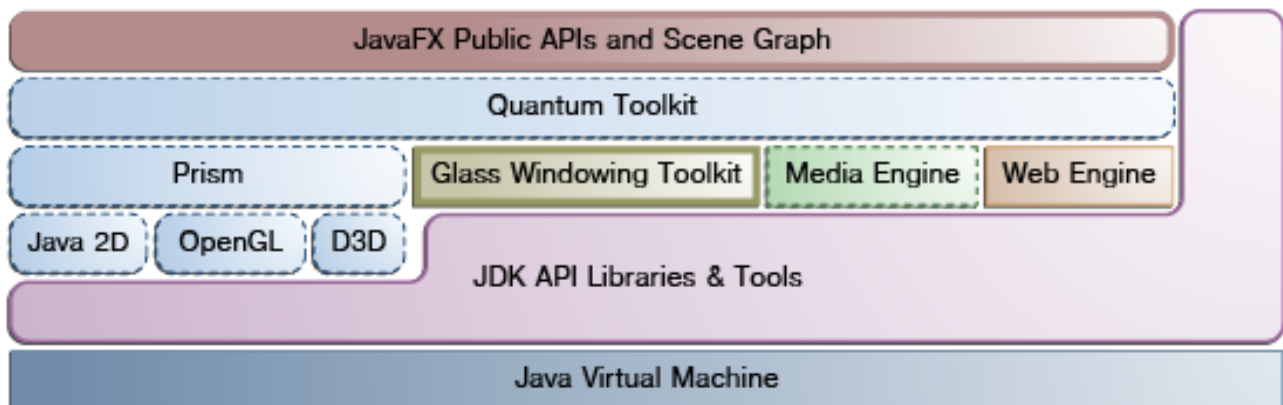
Java 8

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 convient parfaitement à notre projet. Enfin, notre équipe est également plus habile à programmer à l'aide du langage Java.

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.

FIGURE 1 – Architecture de JavaFX



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

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.

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. Il permet de spécifier les dépendances d'un programme ainsi que des instructions permettant de compiler les fichiers source en fichiers exécutables. Ces instructions se trouvent dans le fichier « pom.xml » et est librement modifiable par les développeurs.

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.

GitHub

GitHub est un service web permettant de parcourir visuellement l'historique Git et qui fournit également 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.

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. Les figures 2 et 3 ci-dessous proposent une comparaison de celui-ci avec l'interface finale.

FIGURE 2 – Mock-up de l'interface

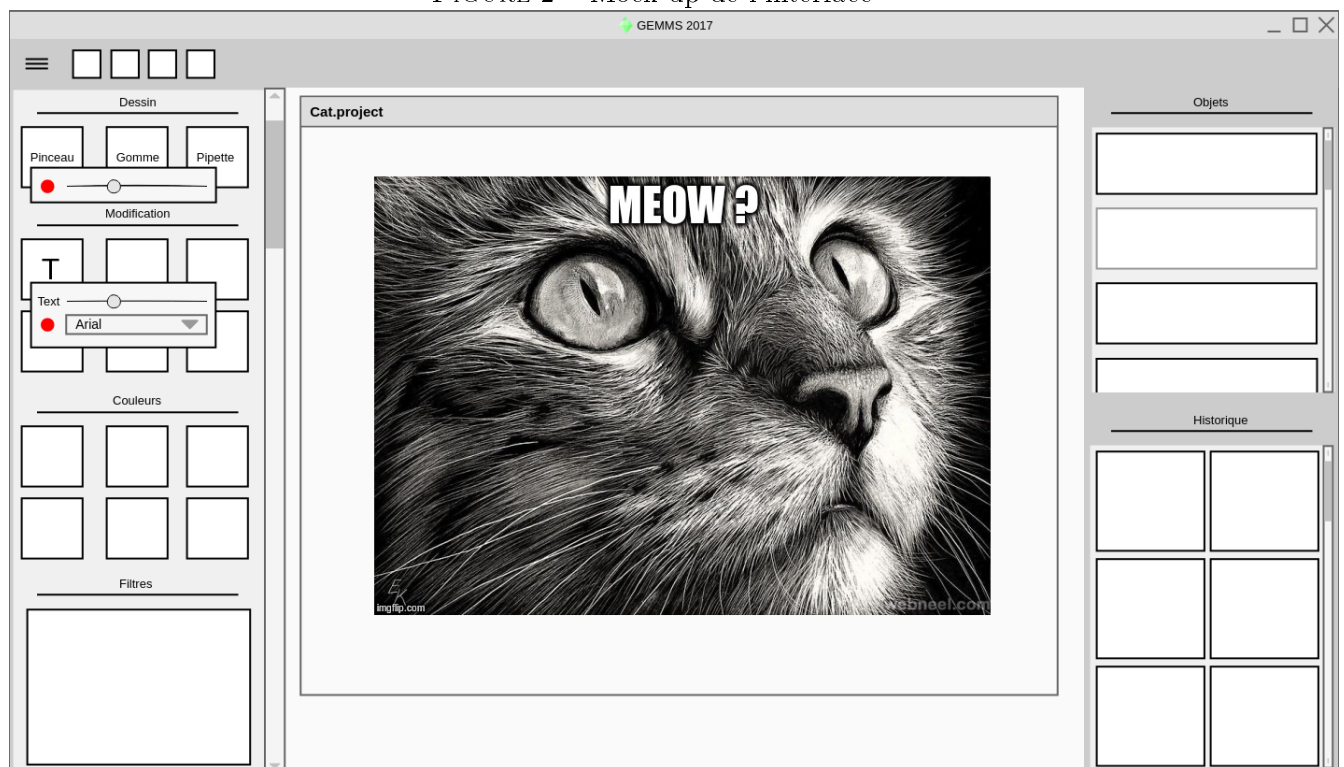
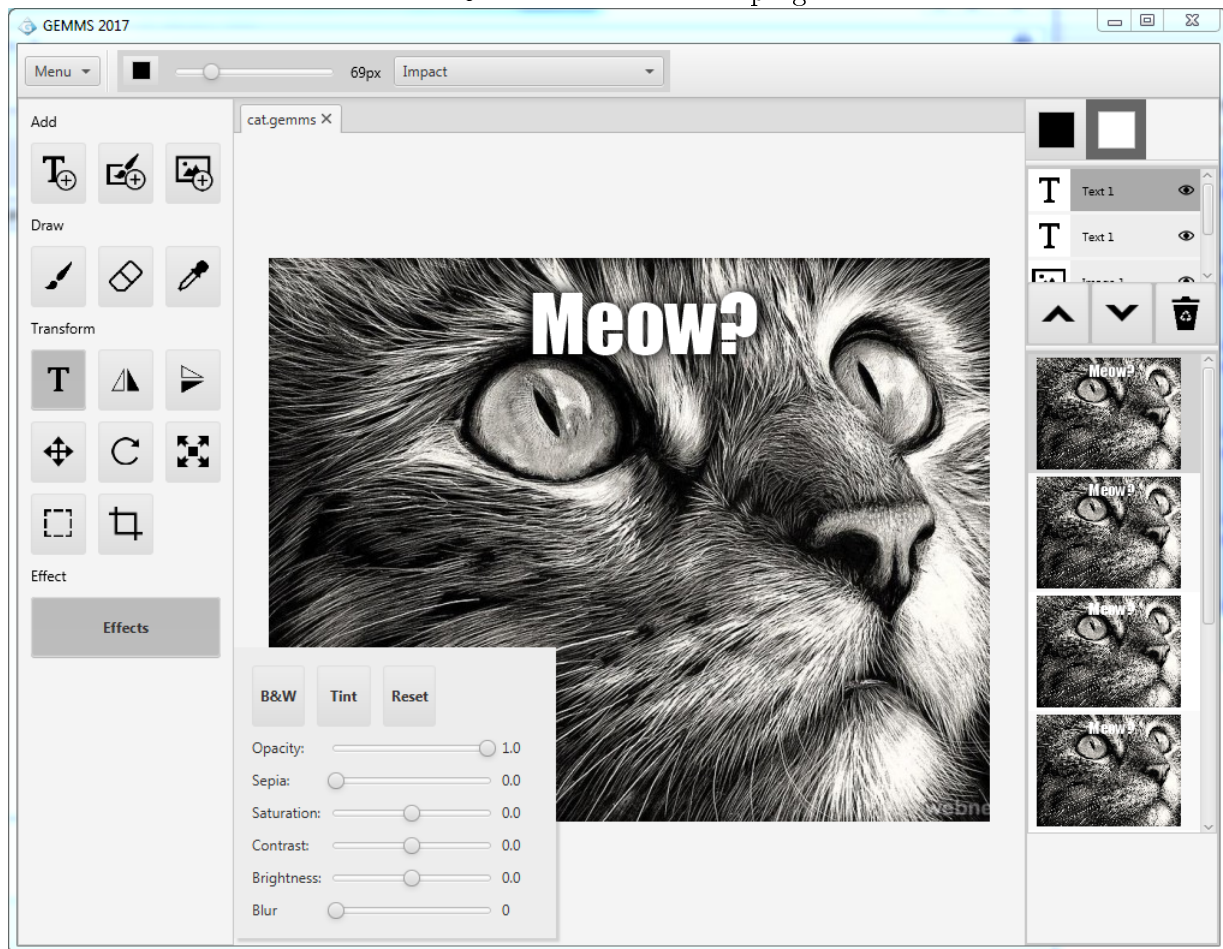


FIGURE 3 – Interface finale du programme



Comme le montrent les figures 2 et 3, l'interface conçue lors de l'élaboration du cahier des charges a 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.

Description technique

Structure

Structure de JavaFX

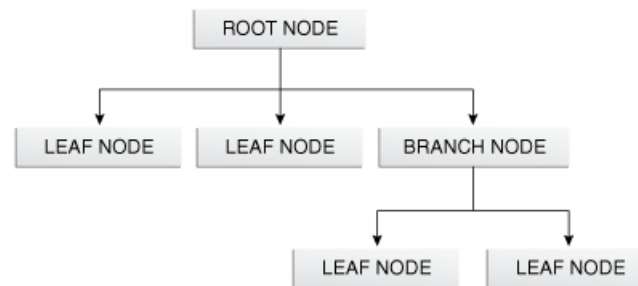
Notre application se base sur la structure officielle d'une application JavaFX, qui se prête parfaitement à notre type d'application. Sa compréhension est essentielle pour discuter de l'architecture de notre logiciel.

Les trois points-clés sont la notion de Stage, de Scene et de Scene Graph. Le Stage est le container haut-niveau d'une application JavaFX [3]. Il doit contenir tout le contenu d'une fenêtre. La Scene est

le container pour un Scene Graph. Chaque scène doit avoir un et un seul nœud racine du Scene Graph [2].

Le Scene Graph est une structure en arbre qui garde une représentation interne des éléments graphiques de l'application. En tout temps, il sait quels éléments afficher, quelles zones de l'écran doivent être rafraîchies, et comment le faire de la manière la plus efficace [1].

FIGURE 4 – Représentation du Scene Graph de JavaFX. Source : documentation Oracle [1]



Comme on peut le voir sur la figure 4, chaque nœud de l'arbre du Scene Graph appartient à la hiérarchie de la classe Node. De plus, chacun de ces nœuds est soit une feuille (ne pouvant pas contenir d'enfants), soit une branche (pouvant alors contenir des enfants).

Cette structure particulière permet donc de créer facilement des interfaces graphiques, car il suffit que chaque élément visuel de notre application soit un objet (spécialisé ou non) d'une classe héritant de Node pour que l'affichage de cet élément soit géré automatiquement par JavaFX.

Pour beaucoup de nos composants, nous avons donc spécialisé une des classes offertes par JavaFx proposant la fonctionnalité recherchée, en y ajoutant les comportements dont nous avons besoin. Ils sont ensuite ajoutés au Scene Graph de la scène principale, et nous pouvons ainsi construire notre application.

Serialisation

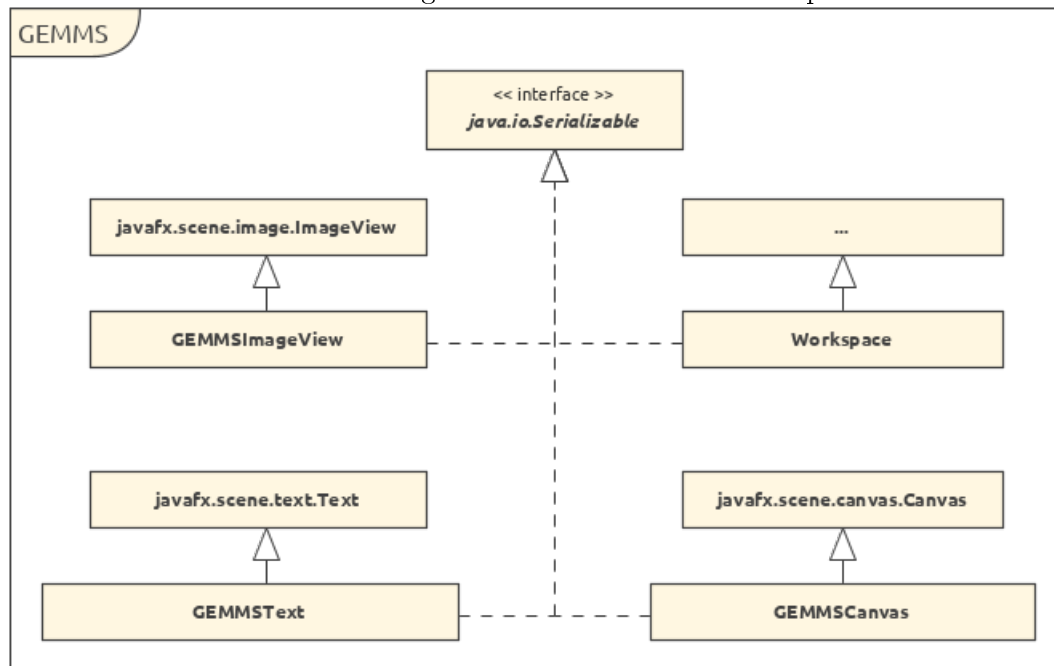
En Java, la sérialisation s'effectue à l'aide de l'interface « Serializable ». Par conséquent, chaque classe de Java implémentant cette dernière telle que « String », peut être sérialisée et désérialisée à volonté. Cependant, la majorité des classes JavaFX n'implémentent pas cette interface. En effet,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

Pour combler ce manque, nous devons nous-mêmes implémenter la sérialisation des classes JavaFX que nous sommes susceptibles d'utiliser.

Sur la figure 5, 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 spécialisables. 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

FIGURE 5 – Diagramme de la sérialisation simplifié



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.

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.

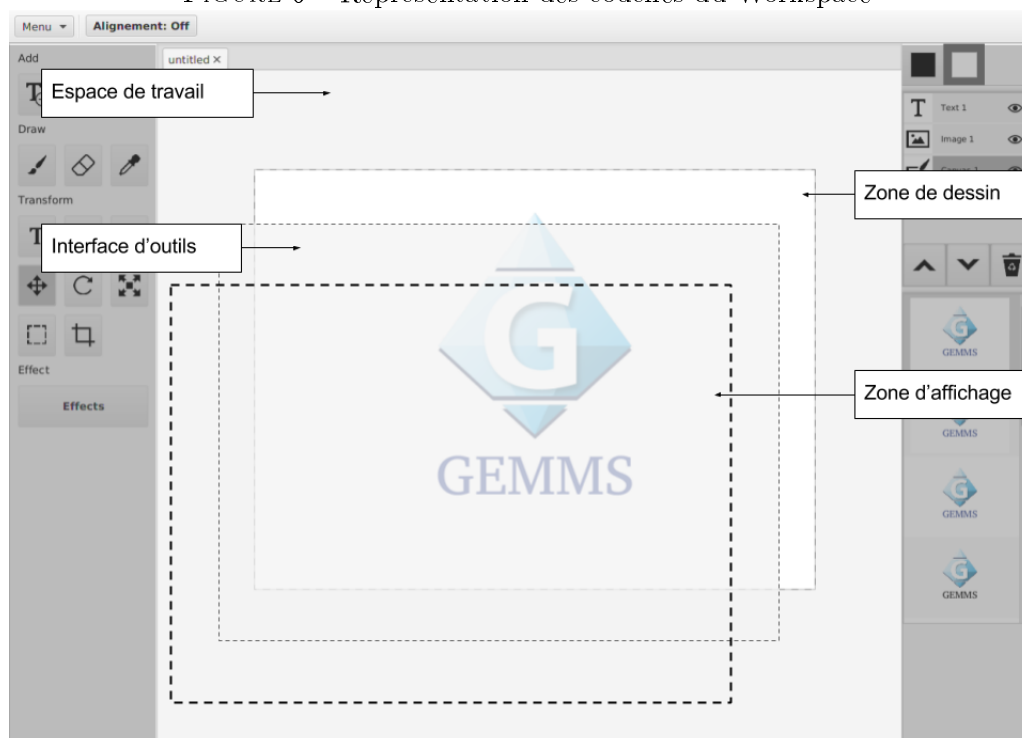
Workspace et liste des calques

Le Workspace correspond à l'espace de travail d'un Document GEMMS. Workspace est une classe personnalisée héritant de la classe StackPane de JavaFX.

Structure d'un Workspace

Chaque document ouvert possède un objet Workspace permettant à l'utilisateur de manipuler le contenu du fichier. Il est constitué de plusieurs couches de composants, comme représenté sur la figure 6.

FIGURE 6 – Représentation des couches du Workspace



L'espace de travail (Workspace) est un container contenant deux enfants (au sens de l'arbre de JavaFX) : une zone de dessin (une branche du Scene Graph) où sont ajoutés tous les calques créés et modifiés par l'utilisateur, et une interface des outils permettant de dessiner les retours visuels des outils

(comme le curseur du pinceau par exemple). Tous deux sont de type `AnchorPane`, un type spécifique de container JavaFX, car cette classe permet de positionner leurs enfants selon des coordonnées (x, y).

La zone d’affichage correspond à un masque appliqué à l’espace de travail pour éviter que les calques ne dépassent de la zone dessin. En effet les parents de type `AnchorPane` ne cachent pas leur éléments enfants sortant de leurs frontières.

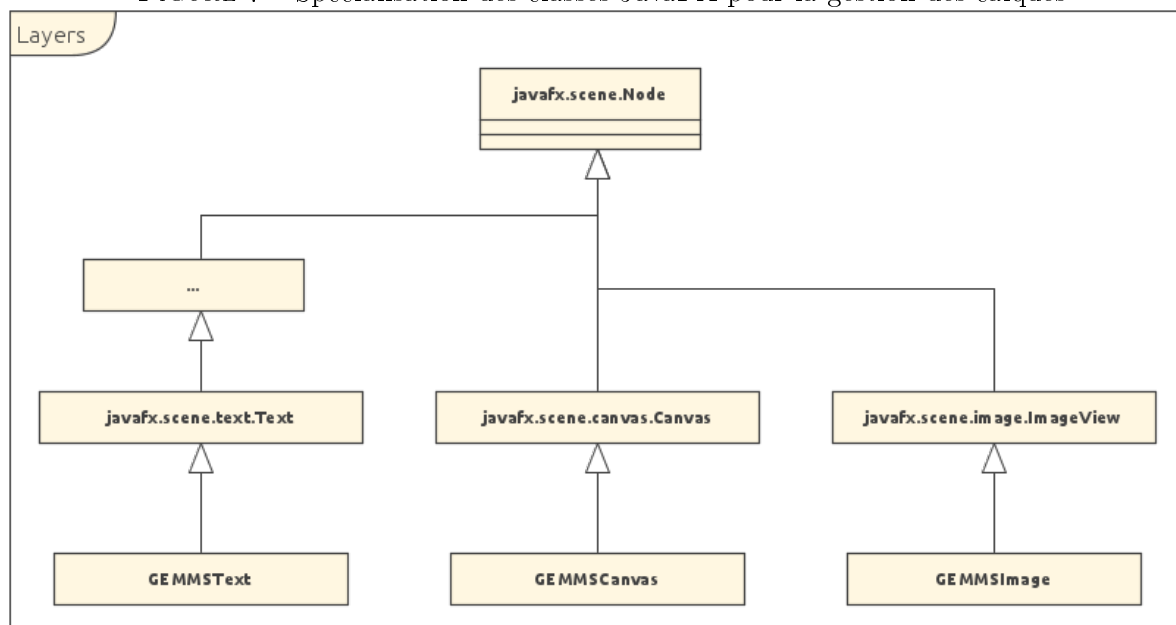
Le `Workspace` détecte les mouvements de souris sur l’interface des outils et se charge d’appeler les méthodes correspondantes de l’objet `Tool` activé (se référer à la section 4.8.1).

Types de calques

Dans GEMMS, les éléments d’un document sont gérés au moyen de calques. L’application utilise trois sortes de calques, qui sont en fait des types de `Node` proposés par JavaFX. La classe `Text` permet de représenter des textes, la classe `Canvas` permet de dessiner des éléments graphiques (lignes, formes, etc.) et la classe `ImageView` permet de représenter des images.

Pour les besoins de l’application, nous ne pouvions pas utiliser ces classes telles qu’elles sont proposées par l’API JavaFX. Nous avons donc créé des classes spécialisées (`GEMMSText`, `GEMMSCanvas` et `GEMMSImage`) de chacun de ces éléments afin de pouvoir ajouter les fonctionnalités nécessaires. La figure 7 illustre cette spécialisation.

FIGURE 7 – Spécialisation des classes JavaFX pour la gestion des calques

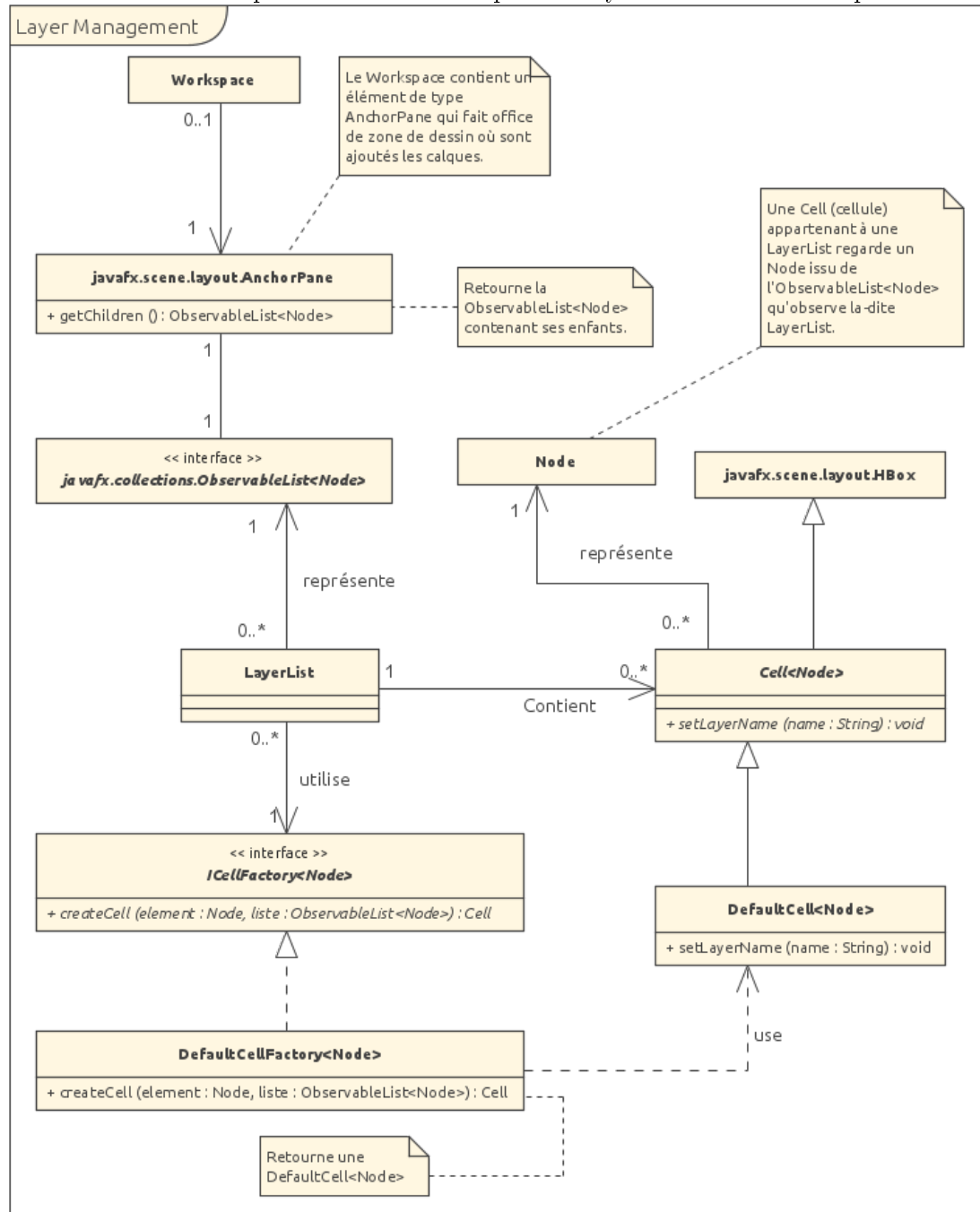


Gestion des calques

Une des fonctionnalités indispensables de GEMMS est de permettre à l’utilisateur de gérer les calques de son document au moyen d’une liste dans un panneau de l’interface. Il s’agit d’une représentation visuelle de l’état du Scene Graph dans la branche de la zone de dessin dont nous parlons plus haut. Elle doit donc être en tout temps synchronisée avec l’état actuel du document et tout changement doit se répercuter dans ce listing de calques.

La classe `ListView` de JavaFX propose cette fonctionnalité. Malheureusement, ce composant n'étant pas paramétrable comme nous le souhaitions pour les besoins ergonomiques de l'application, nous avons dû implémenter notre propre système de gestion de calques. Il est cependant fortement inspiré du fonctionnement de `ListView`.

FIGURE 8 – Implémentation du composant `LayerList`. Structure simplifiée.



Copier-coller

Deux façon d'aborder le copier-coller ont été implémentées : la première consiste à enregistrer dans le presse-papier un ou des calques sélectionnés, la seconde consiste à enregistrer uniquement la partie sélectionnée et visible à l'écran.

Trois procédés sont à l'origine de cette fonction de l'application : le snapshot, le viewport et la sérialisation.

Snapshot

Le snapshot est un outil proposé par JavaFX qui permet d'effectuer une «capture» d'un élément ou un groupe d'élément. C'est ce qui nous intéresse lorsque l'on fait une copie d'une zone sélectionnée de l'écran. L'image ainsi capturée est alors écrite sur un **GEMMSCanvas** qui est sérialisé et enregistré dans le presse-papier.

Viewport

La notion de Viewport intervient dans le cas d'un snapshot. Elle permet de définir une «fenêtre» de capture rectangulaire à laquelle se restreindra le snapshot, on la passe en paramètre au moment d'effectuer ce dernier.

Ce mécanisme est utilisé quand on copie une zone sélectionnée de l'espace de travail (voir 4.8.10 pour la sélection). Les coordonnées de la sélection (position et taille) sont récupérées et utilisées pour créer un viewport. Le défi principal a été de gérer le système de coordonnées de JavaFX. En effet, chaque nœud possède son propre système de coordonnées et un point de ce nœud peut être exprimé selon le système de coordonnées local au nœud, selon le système du parent direct du nœud ou encore selon le système de la fenêtre, de l'écran etc.

La notion à maîtriser ici est d'exprimer au viewport ses dimensions dans le système de coordonnées du parent du nœud sur lequel on effectue le snapshot. Une fois cette notion comprise, le snapshot est restreint au viewport (et donc à la sélection) et le comportement voulu est adopté.

Il reste ensuite à écrire l'image capturée sur un **GEMMSCanvas** qui sera affiché par la suite dans l'espace de travail.

Sérialisation

Le presse-papier ne permet d'enregistrer que des données textuelles, il faut donc sérialiser le ou les calques qui seront stockés dans le presse-papier. Pour plus de détails sur la sérialisation, voir la section 4.2.

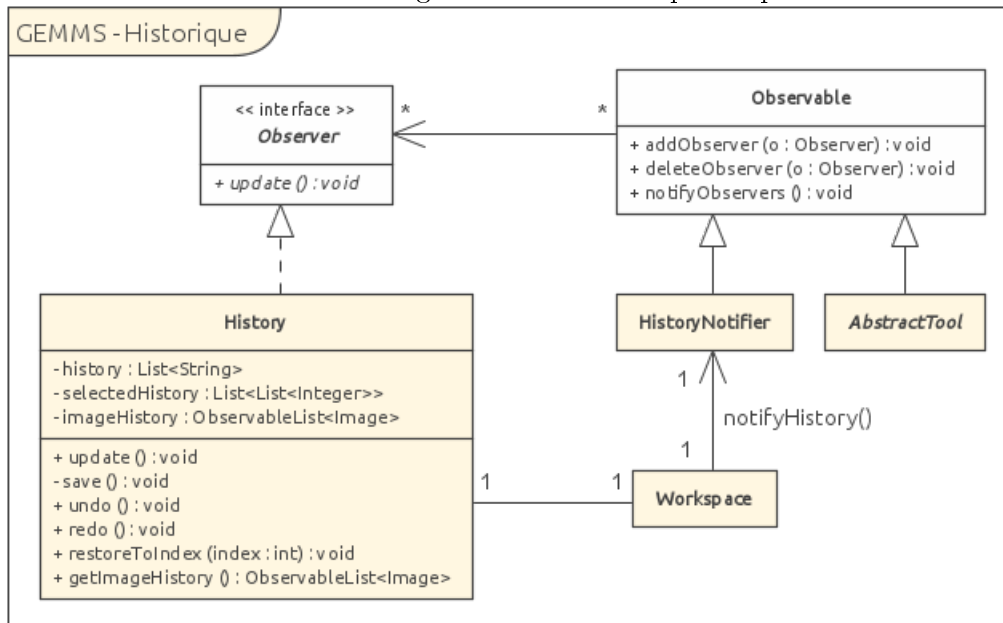
Coller le contenu du presse-papier

Le collage consiste à désérialiser le contenu du presse-papier et à insérer dans le Workspace courant le ou l'ensemble de calques désérialisés.

Historique

L'historique des actions effectuées se base principalement sur deux concepts : la sérialisation et le patron de conception «Observable» comme l'indique la figure 9.

FIGURE 9 – Diagramme de l'historique simplifié



Après chaque action que l'on souhaite pouvoir annuler, on notifie l'instance de la classe **History** liée au **Workspace** en cours d'utilisation. **History** implémente donc **Observer** et observe les outils ainsi que le **HistoryNotifier**. Lorsqu'il reçoit une notification, l'historique sauvegarde l'entier du **Workspace**. C'est à ce moment qu'intervient la sérialisation. Il aurait été possible d'utiliser l'interface **Cloneable** pour les différents **GEMMSNode** et sauvegarder systématiquement des copies des objets, cependant le code aurait été très similaire à celui de la sérialisation. Nous avons donc décidé de sérialiser les calques de l'espace de travail et d'enregistrer la chaîne de caractères en résultant. Note : l'utilisation de **HistoryNotifier** se justifie par le fait que **Workspace** étend la classe **StackPane** et ne pouvait donc pas être lui-même **Observable**.

Lorsque les méthodes **undo()** et **redo()** sont appelées (notamment par les commandes Ctrl + Z et Ctrl + Y), l'historique restaure (déséréalise) la sauvegarde faite dans la liste d'éléments sérialisés correspondant à l'action demandée.

De la même manière que pour les calques, la liste des calques sélectionnés au moment de la sauvegarde est enregistrée afin que lorsque l'utilisateur restaure l'état précédent, les calques qu'il avait sélectionnés le soient à nouveau.

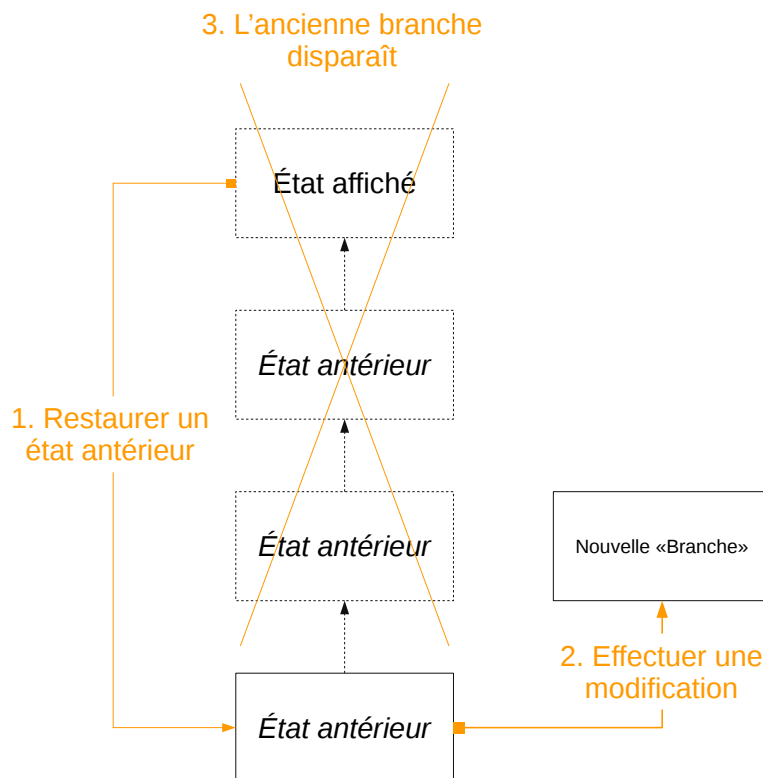
Historique visuel

L'affichage de l'historique visuel utilise le même mécanisme que décrit ci-dessus. À chaque action de l'utilisateur, on effectue une capture («snapshot») miniature de l'espace de travail qui sera stockée dans une liste. Cette liste est observable et alimente une **ListView** qui affiche les images dans l'interface de GEMMS.

Lorsque l'utilisateur clique sur un élément de la `ListView`, la méthode `restoreToIndex(index)` de l'historique est appelée. Elle restaure l'espace de travail non pas à l'état précédent, mais à l'état indiqué par le paramètre entier positif `index`.

On notera finalement que si l'utilisateur effectue une modification après avoir restauré un état précédent, il se trouvera sur une nouvelle «branche» et il ne pourra plus «revenir en avant», comme sur pratiquement tous les logiciels que l'on connaît, voir figure 10.

FIGURE 10 – Fonctionnement de l'historique



Positionnement

TODO TODO TODO

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.

Hiérarchie des outils

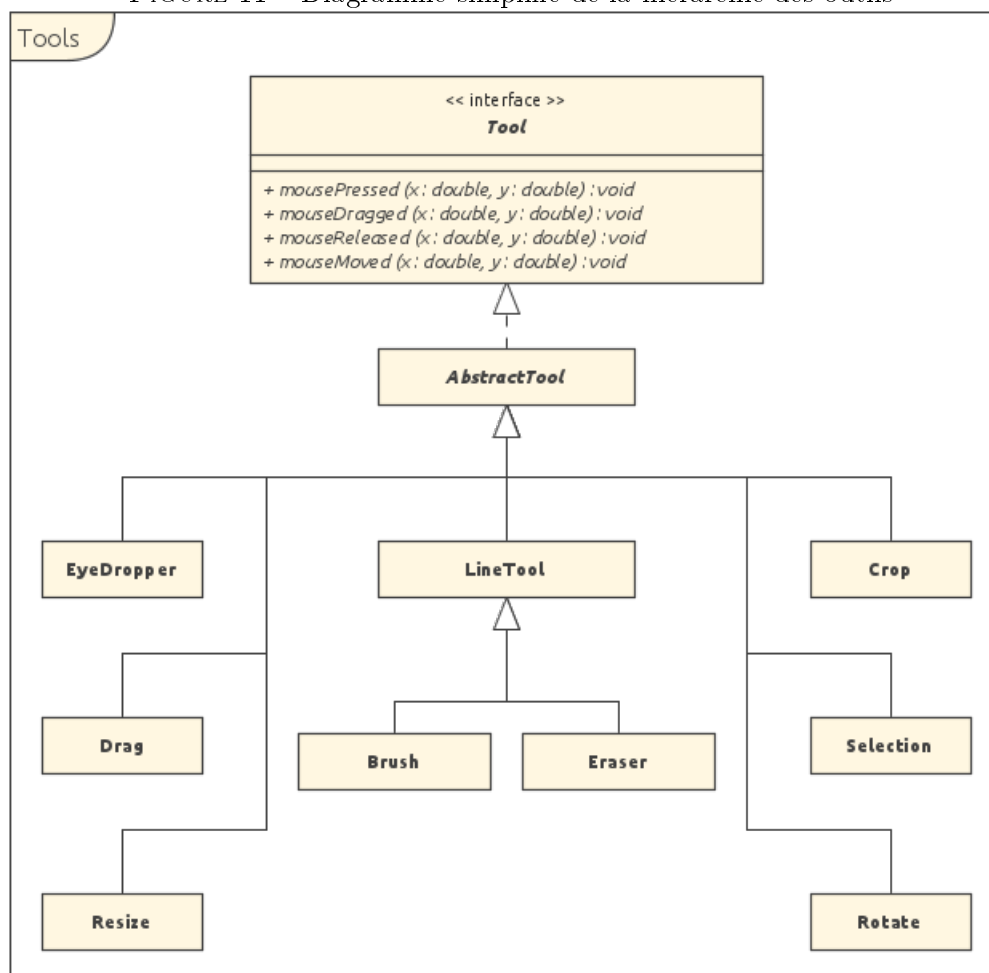
La plupart des outils de l'application fonctionnent grâce à ces trois événements. On pensera notamment au pinceau qui doit dessiner un trait en suivant la souris lors d'un `MouseDragged`.

Comme on peut le voir sur la figure 11, 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 11 – Diagramme simplifié de la hiérarchie des outils



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 12 illustre l'apparence des réglages de l'outil de modification de texte.

FIGURE 12 – 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 13), 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).

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 :

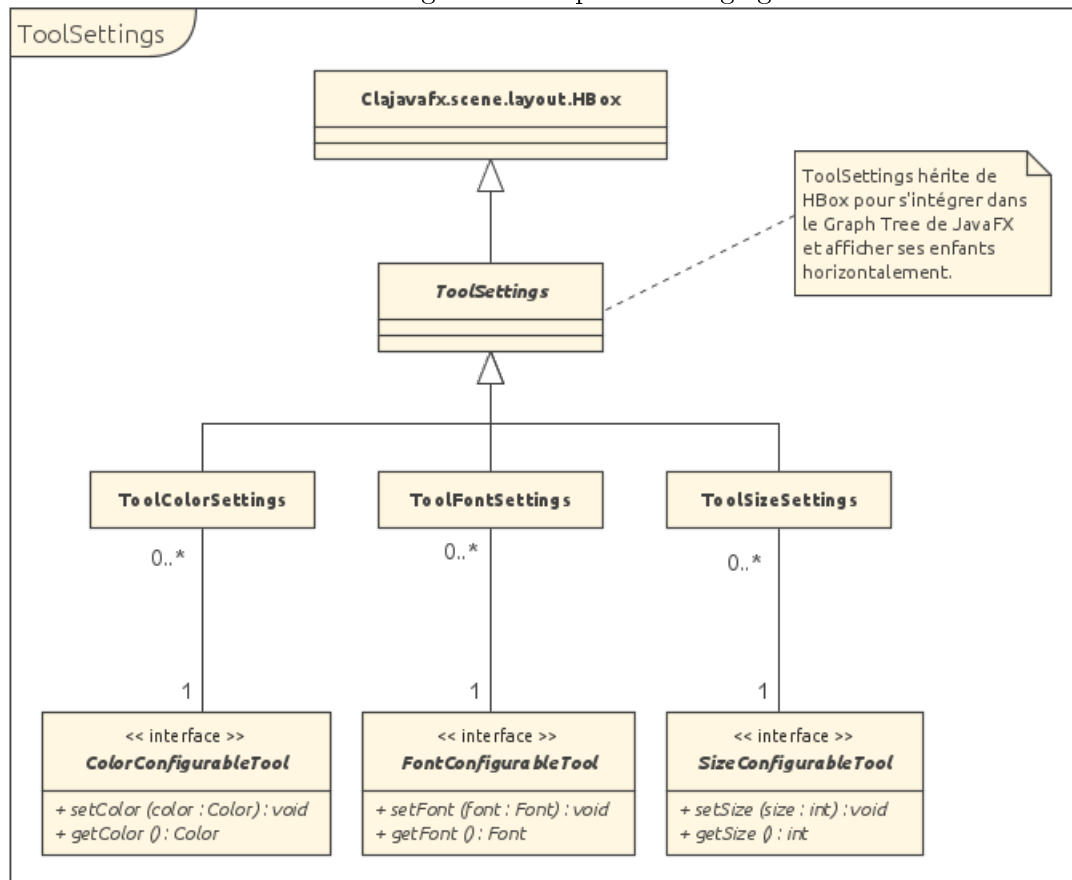
L'événement `MouseDragged` est déclenché à intervalles réguliers, tant que l'utilisateur effectue cette action. Pour chaque répétition, il est facile de garder trace de la position de la souris au dernier événement et à l'événement actuel. Connaissant ces deux points, il est possible de dessiner une 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 [5].

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.

FIGURE 13 – Diagramme simplifié des réglages d'outils



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.

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.

Symétries

Les deux symétries, horizontale et verticale, s'effectuent non pas grâce à un outil qui implémente une interface *Tool* mais grâce à des transformations JavaFX sur les nœuds sélectionnés. Cela est principalement dû au fait que les outils implémentant cette interface sont pratiques pour une utilisation avec la souris alors que pour une symétrie, il n'y en a pas besoin. Une symétrie effectue donc une transformation de rotation d'un angle de 180 degrés avec un pivot centré au milieu du nœud et un axe de rotation correspondant à la symétrie. Par exemple, pour la symétrie horizontale, on applique la rotation sur l'axe Y.

Concernant la sérialisation de cette transformation, comme il s'agit de la transformation d'un nœud et que les transformations de JavaFX ne sont pas sérialisables, il faut enregistrer tous les paramètres de la rotation (angle, pivot et axe) afin de pouvoir réappliquer cette transformation lors de la désérialisation.

Déplacement

Le rôle de l'outil de déplacement est de pouvoir déplacer un ou des nœuds. Pour se faire, il implémente l'interface *Tool* afin de pouvoir déplacer les nœuds avec la souris. Il retient la position de la souris lors du clic, calcule un offset lors du déplacement de celle-ci et applique une translation pour chaque nœud sélectionné qui consiste à rajouter l'offset aux coordonnées du nœud.

Il y a deux types de déplacement : celui avec l'alignement automatique et celui sans. Ce dernier, activable via un bouton, permet d'afficher des repères qui traversent le centre du workspace et qui permettent d'aligner un nœud sur ceux-ci. De plus, avec ces repères, il y a un mécanisme d'aimant qui permet, lorsque la souris est assez proche d'un repère d'alignement, d'attirer le nœud sélectionné sur lui. Ce mécanisme d'attraction ne marche que s'il n'y a qu'un nœud de sélectionné.

Rotation

Plusieurs approches sont possibles pour le mécanisme de rotation mais nous avons finalement opté pour un mécanisme très similaire à l'outil de déplacement : on calcule un offset via la position de la souris et on applique une rotation sur les nœuds sélectionnés.

Redimensionnement

Comme pour l'outil déplacement et l'outil rotation, plusieurs approches ont été pensées mais celle qui a été retenue est très semblable à la rotation et au déplacement. On calcule un offset avec le déplacement de la souris, on a un certain facteur de grossissement que l'on multiplie à l'offset afin d'obtenir le nouveau grossissement que l'on va appliquer aux nœuds.

Sélection

La sélection permet à l'utilisateur de sélectionner une zone du Workspace à l'aide d'un rectangle. Ce dernier peut être déplacé ou recréé si la zone de sélection ne convient pas. Une fois la zone choisie, si le calque courant est un GEMMSCanvas, il est possible de supprimer à l'aide de la touche DEL ou de copier à l'aide de la touche CTRL+C.

Pour la suppression de la zone, la classe `GraphicsContext` qui s'occupe de la partie graphique d'un Canvas offre la méthode `clearRect` qui permet d'effacer une partie rectangulaire du Canvas (Remplace les pixels présents par des pixels de couleur transparent). Il suffit donc de récupérer la position, la largeur et la hauteur de la zone de sélection et d'utiliser `clearRect`.

Et enfin pour le copier-coller, voir la section 4.5 pour plus d'informations sur son fonctionnement.

Rognage

Effets

La section « Effect » 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'une future modification des effets, de simplement devoir faire varier les paramètres de ces effets.

Noir et blanc

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

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`.

Barres glissantes

Les barres glissantes (« Sliders » JavaFX) permettent d'ajuster tous les autres effets implémentés. La transparence règle directement un attribut « `Opacity` » d'un nœud. 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`.

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.

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 a 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 tâches initiale a cependant pris en compte ces dépendances et l'ordre d'implémentation des fonctionnalités a 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.

Glossaire

AnchorPane AnchorPane est une classe de JavaFX permettant (entre autres) de disposer des noeuds selon des coordonnées (x,y).. 8, 17

Canvas Type de Node spécialisé permettant de dessiner des lignes, des formes et des éléments graphiques dans un canevas dans le Scene Graph.. 17

ColorAdjust Effet JavaFX permettant d'ajuster la couleur d'un nœud en modifiant son contraste, saturation, teinte et luminosité.. 17

GaussianBlur Effet JavaFX permettant d'affecter à un nœud un effet de flou gaussien d'intensité variable.. 17

ImageView Type de Node personnalisé permettant de représenter une image dans le Scene Graph. 17

Node La classe Node est une classe de JavaFX. Il s'agit de la classe de base de tout élément présent dans le Scene Graph. 5, 17

Scene La Scene, dans une application JavaFX, est le container pour un Scene Graph. Chaque scène doit avoir un et un seul nœud racine du Scene Graph. 4, 17

Scene Graph Le Scene Graph (graphe de scène en français) est une structure en arbre qui garde une représentation interne des éléments graphiques de l'application. 4, 17

SepiaTone Effet JavaFX permettant d'affecter à un nœud un effet sépia d'intensité variable.. 17

Stage Le Stage est le container haut-niveau d'une application JavaFX. Il doit contenir tout le contenu d'une fenêtre. 4, 17

Text Type de Node spécialisé permettant de représenter un texte dans le Scene Graph.. 17

Workspace Workspace est une classe de l'application GEMMS représentant la zone de travail de l'utilisateur dans un Document GEMMS. Un objet contient notamment la liste de tous les calques utilisés dans le document. 7, 17

Références

- [1] Oracle. Working with the javafx scene graph, 2013.
- [2] Oracle. Scene (javafx), 2015.
- [3] Oracle. Stage (javafx), 2015.
- [4] R. M. Star. *Foo Bar Baz*. MIT Press, Cambridge, MA, 1989.
- [5] Wikipedia. Bresenham-algorithmus, 2007.

Annexes

Projet de semestre (PRO)
Editeur d'image GEMMS

Auteurs :

Guillaume MILANI
Edward RANSOME
Mathieu MONTEVERDE
Michael SPIERER
Sathiya KIRUSHNAPILLAI

Client :

René RENTSCH

Référent :

René RENTSCH

Journal de travail - Edward Ransome

Semaine 1 : 20 février - 24 février

Création du groupe et recherche d'idées. Discussion et proposition des deux principaux sujets au professeur, un éditeur d'images et un programme de manipulation de graphes.

Semaine 2 : 27 février - 3 mars

Programme d'édition d'images accepté, début d'élaboration du cahier des charges. Discussion sur le fonctionnement, l'architecture, les fonctionnalités voulues ainsi que des mock-ups d'interface. Travail simultané de tout le groupe.

Semaine 3 : 6 mars - 10 mars

Fin de rédaction du cahier des charges, avec mock-ups finaux et une liste des fonctionnalités indispensables et optionnelles. Rendu de celui-ci ainsi qu'un planning du travail sous forme de diagramme de Gant.

Semaine 4 : 13 mars - 17 mars

Retour sur le cahier des charges, pas de problèmes majeurs. Planification rédigée sous forme d'un tableau Excel pour faciliter la compréhension et mieux voir le travail effectué à chaque membre du groupe. Début du travail individuel selon le planning.

Semaine 5 : 20 mars - 24 mars

Étude de JavaFX. Tutoriel JavaFX 8 de Code Makery effectué, qui comprends une introduction à Scene Builder, création de fenêtres, effets et autres fonctionnalités de cette librairie. Étude de la documentation Oracle de JavaFX.

Semaine 6 : 27 mars - 31 mars

Début de création de l'interface avec Guillaume Milani en se basant sur les mock-ups élaborés pour le cahier des charges.

Semaine 7 : 3 avril - 7 avril

Fin de l'interface Scene Builder avec contrôleurs des boutons dans le code ainsi que des références aux éléments du fichier « .fxml » dans le code pour pouvoir les modifier hors de Scene Builder.

Semaine 8 : 10 avril - 14 avril

Aide à l'implémentation du Workspace dans l'interface graphique principale avec déplacement et zoom.

Semaine 9 : 24 avril - 28 avril

Continuation de l'implémentation du Workspace dans l'interface.

Semaine 10 : 1 mai - 5 mai

Début d'élaboration de la zone « Effet » de l'interface, recherche sur l'implémentation des effets en JavaFX ainsi que leur application sur divers éléments de la librairie (Image, texte, canvas).

Semaine 11 : 8 mai - 12 mai

Création de différents boutons permettant d'augmenter l'opacité, saturation et l'effet sepia.

Semaine 12 : 15 mai - 19 mai

Changements dans l'implémentation des effets, un « Slider » JavaFX par effet incrémentable, permettant de modifier l'opacité, saturation, contraste et sepia sur un ou plusieurs calques. Implémentation d'une remise à zéro des effets. Début de réflexion sur la sérialisation des effets.

Semaine 13 : 22 mai - 26 mai

Ajout de l'effet de flou, ainsi que un bouton « Tint » qui crée une nuance de couleur sur un calque. Implémentation de la sérialisation des effets et correction de l'ordre d'application des effets pour permettre leur application dans un ordre quelconque sans recréer tous les effets.

Semaine 14 : 29 mai - 2 juin

Élaboration du rapport.

Journal de travail - Mathieu Monteverde

Semaine 1 : 20 février - 24 février

Constitution des groupes et choix du sujet. De nombreux sujets ont été proposés, la plupart ne faisant pas l'unanimité. Finalement, deux idées ont fait été retenues par le groupe : un programme de manipulation de graphes, et en premier lieu, un programme d'édition d'images.

Semaine 2 : 27 février - 3 mars

Attribution des sujets de projet. Le projet de programme d'édition d'images a été accepté. Nous avons donc pu commencer la réflexion autour des fonctionnalités et la planification du projet.

Semaine 3 : 6 mars - 10 mars

Discussion en groupe. Nous faisons le tri des fonctionnalités indispensables et utiles. Une fois celles-ci fixées, nous établissons le cahier des charges et la planification Gant qui va avec.

Semaine 4 : 13 mars - 17 mars

Le professeur nous fourni un retour sur notre cahier des charges et notre planification. Pour plus de clarté, nous remplissons une nouvelle planification dans un format Excel. Nous nous mettons d'accord sur le fait de prendre deux semaines pour étudier la technologie JavaFX que nous utiliserons pour le projet et qu'aucun de nous ne connaît.

Semaine 5 : 20 mars - 24 mars

Étude de JavaFX.

Semaine 6 : 27 mars - 31 mars

Étude de JavaFX. Début de la mise en place de la structure de la classe Workspace avec la spécification des méthodes. Nous réalisons également que des difficultés sont à attendre pour la gestion des calques, de la sérialisation et des éléments issus de JavaFX en général.

Semaine 7 : 3 avril - 7 avril

Implémentation du Workspace avec insertion et suppression de calques.

Semaine 8 : 10 avril - 14 avril

Implémentation de la gestion du Workspace pour le déplacement et le zoom de l'utilisateur dans l'interface. Premières recherches pour la gestion de calques au moyen d'une ListView de JavaFX.

Semaine 9 : 24 avril - 28 avril

Ajout des prototypes recherchés en semaine 9 au reste du projet. Le Workspace permet maintenant d'ajouter des calques, de les supprimer, et de se déplacer dans l'interface.

Semaine 10 : 1 mai - 5 mai

Des changements ont eu lieu pour le Workspace. De mon côté, il faut encore améliorer la gestion des calques. On se rend compte que l'élément ListView de JavaFX, qui pourrait pourtant de fonctionner parfaitement pour l'affichage des calques, ne permet de pas de changer l'ordre d'affichage.

Semaine 11 : 8 mai - 12 mai

Il va falloir trouver une solution pour la gestion des calques. Le module étant cependant fonctionnel, on se charge des autres fonctionnalités. Début de la réalisation du pinceau et de la gomme.

Semaine 12 : 15 mai - 19 mai

Remplacement de la ListView de gestion des calques JavaFX par un composant fait main pour pouvoir répondre aux besoins de l'application. Les outils pinceaux, gommes et pipette sont fonctionnels. Beaucoup d'éléments ont pu être ajoutés cette semaine. La gestion de la couleur, le paramétrage des outils (taille du pinceau et de la gomme, édition de textes et autres).

Semaine 13 : 22 mai - 26 mai

Il y a eu beaucoup de problèmes avec les transformations de calques (rotation, taille, symétrie, etc.) Mais après beaucoup d'efforts, les problèmes ont pu être résolus. Maintenant, il s'agit de peaufiner l'interface (espacement, ergonomie, retour visuels pour l'utilisateur, etc.),

Semaine 14 : 29 mai - 2 juin

Finition du rapport et du manuel utilisateur.

Journal de travail - Sathiya Kirushnapillai

Semaine 1 : 20 février - 24 février

Constitution des groupes et choix du sujet. De nombreux sujets ont été proposés, la plupart ne faisant pas l'unanimité. Finalement, deux idées ont fait été retenues par le groupe : un programme de manipulation de graphes, et en premier lieu, un programme d'édition d'images.

Semaine 2 : 27 février - 3 mars

Attribution des sujets de projet. Le projet de programme d'édition d'images a été accepté. Nous avons donc pu commencer la réflexion autour des fonctionnalités et la planification du projet.

Semaine 3 : 6 mars - 10 mars

Discussion en groupe. Nous faisons le tri des fonctionnalités indispensables et utiles. Une fois celles-ci fixées, nous établissons le cahier des charges et la planification Gant qui va avec.

Semaine 4 : 13 mars - 17 mars

Le professeur nous fourni un retour sur notre cahier des charges et notre planification. Pour plus de clarté, nous remplissons une nouvelle planification dans un format Excel. Nous nous mettons d'accord sur le fait de prendre deux semaines pour étudier la technologie JavaFX que nous utiliserons pour le projet et qu'aucun de nous ne connaît.

Semaine 5 : 20 mars - 24 mars

Étude de JavaFX.

Semaine 6 : 27 mars - 31 mars

Étude de JavaFX et première réalisation du diagramme de classe.

Semaine 7 : 3 avril - 7 avril

Création de la classe Document avec les fonctionnalités suivantes : Ouvrir, sauver et exporter un document. Cependant, le Workspace n'est pas encore disponible pour compléter la classe Document.

Semaine 8 : 10 avril - 14 avril

Étude de la sérialisation avec JavaFX. Malheureusement, les classes de JavaFX n'implémentent pas Serializable. Recherche d'une solution afin de combler ce manque pour Canvas, Text, Image, etc.

Semaine 9 : 24 avril - 28 avril

Étude de la sérialisation avec JavaFX. Implémentation de la sérialisation sur des classes de base tel que Text et Canvas.

Semaine 10 : 1 mai - 5 mai

Réécriture de la classe Document avec le Workspace enfin disponible. Implémentation concrète de la sérialisation et réalisation de tests avec des classes ajouter dans le Workspace.

Ajout des boutons permettant de créer, ouvrir et sauver un document.

Semaine 11 : 8 mai - 12 mai

Gestion de plusieurs documents à l'aide d'onglets (Adaptation de la classe Document).

Ajout du bouton permettant l'export d'un document en image PNG.

Semaine 12 : 15 mai - 19 mai

Ajout de l'importation d'une image et de sa sérialisation.

Ajout de l'outil sélection.

Redimensionnement du Workspace à l'aide d'un popup.

Amélioration visuelle de l'interface des outils.

Ajout d'un fond de couleur à la création d'un document

Semaine 13 : 22 mai - 26 mai

Il y a eu beaucoup de problèmes avec les transformations de calques(rotation, taille, symétrie, etc). Recherche d'une solution avec Mathieu Monteverde.

Implémentation du déplacement du rectangle de sélection et de la suppression à l'aide de l'outil sélection.

Semaine 14 : 29 mai - 2 juin

Finition du code (Commentaires, bugs, structures) et du rapport.

Journal de travail - Michael Spierer

Semaine 1 : 20 février - 24 février

Nous avons fait un brainstorming afin de trouver des idées, retenu les deux meilleures afin de les proposer pour le projet.

Semaine 2 : 27 février - 3 mars

Notre idée de faire un éditeur d'images a été acceptée et nous avons pu commencer à trouver quelles fonctionnalités nous voulons avoir et à commencer à les planifier.

Semaine 3 : 6 mars - 10 mars

Cahier des charges et planification Gant.

Semaine 4 : 13 mars - 17 mars

Planification dans un format Excel.

Semaine 5 : 20 mars - 24 mars

Étude de JavaFX, tests de certaines fonctionnalités de JavaFX, tutoriels de JavaFX.

Semaine 6 : 27 mars - 31 mars

Étude de JavaFX, tests de certaines fonctionnalités afin de jeter un coup d'oeil aux fonctionnalités que je vais devoir implémenter.

Semaine 7 : 3 avril - 7 avril

Début d'implémentation du Workspace.

Semaine 8 : 10 avril - 14 avril

Recherche de la gestion de calques avec Mathieu, on rencontre déjà des problèmes qui vont nous faire nous poser pas mal de questions, par exemple, un Node n'est pas sérialisable, donc on essaie de trouver des solutions à ce problème.

Semaine 9 : 24 avril - 28 avril

Gestions de calques avec Mathieu. On a fait en sorte de rendre les calques sérialisables (on a ajouté des méthodes pour sérialiser et désérialiser avec l'aide de Sathiya car il avait déjà bossé sur la question de la sérialisation). On peut ajouter les calques au workspace.

Semaine 10 : 1 mai - 5 mai

Implémentation des claques de texte.

Semaine 11 : 8 mai - 12 mai

Redimensionnement des calques, pivotement, déplacement d'un ou plusieurs calques. Il y a plusieurs manières d'aboutir aux effets escomptés, plusieurs choses rentrent en compte : vu que quasiment rien n'est sérialisable, il faut trouver comment sérialiser les transformations, en plus de sérialiser les calques qui sont de base non-sérialisable.

Semaine 12 : 15 mai - 19 mai

Toujours sur les redimensionnement de calques. Il y a de nouveau soucis : le mappage de coordonnées des tools (pas juste les transformations que j'implémente mais aussi brush etc) ne se fait pas bien -> On parvient à régler le soucis. De plus, les symetries m'ont donné du fil à retordre mais marchent parfaitement en fin de compte.

Semaine 13 : 22 mai - 26 mai

Essaie de faire un bonus qui n'est pas dans le cahier des charges pour la réalisation d'un bucket (afin de remplir une zone d'une couleur en un clique). Le prototype marche bien sauf à partir du moment où il y a une transformation sur le calque. Je réévalue la situation, pas de temps à perdre sur cette fonctionnalité non demandée, je passe à la suite. Implémentation de l'alignement automatique d'un calque lors du drag.

Semaine 14 : 29 mai - 2 juin

Rapport et journal de bord.

Journal de travail - Guillaume Milani

Semaine 1 : 20 février - 24 février

Constitution des groupes et choix du sujet. Le choix du sujet n'est pas facile, plusieurs idées sont proposées et un vote doit avoir lieu. Au final nous proposerons un projet de résolution de graphe et un éditeur d'images.

Semaine 2 : 27 février - 3 mars

Les sujets sont attribués, le projet d'éditeur d'image est préféré car un autre groupe est aussi intéressé par l'éditeur de graphe. Nous commençons la liste des fonctionnalités à proposer et l'écriture du cahier des charges.

Semaine 3 : 6 mars - 10 mars

Nous fixons la liste des fonctionnalités que nous souhaitons voir apparaître dans notre programme. Nous rédigeons le cahier des charges ainsi que la planification Gantt (très importante car notre projet sera basé presque entièrement sur une seule et unique vue, interface).

Semaine 4 : 13 mars - 17 mars

À la demande de M. Rentsch, nous remplissons une planification au format Excel, plus claire que le Gantt précédemment. Nous commençons l'étude de JavaFX chacun de notre côté, il s'agit pour nous d'une première expérience avec ce Framework.

Semaine 5 : 20 mars - 24 mars

Étude de JavaFX et création des issues sur Github que nous utiliserons par la suite. Je reproduis la planification sous forme de Milestones et attribue chaque issue aux personnes concernées.

Semaine 6 : 27 mars - 31 mars

Début de la création de l'interface de l'application à l'aide de l'outil Scene Builder en essayant de reproduire à l'identique la maquette présentée dans le cahier des charges.

Semaine 7 : 3 avril - 7 avril

Edward et moi-même mettons en place le contrôleur Java qui interagira avec la vue au format FXML.

Semaine 8 : 10 avril - 14 avril

Commentaire du contrôleur et mise en place d'une fonction pour générer dynamiquement les boutons des outils afin de faciliter le travail des autres membres de l'équipe.

Semaine 9 : 24 avril - 28 avril

Documentation sur les Popups, implémentation d'un popup qui permettra de modifier la taille et la couleur d'un outil. Ce popup sera finalement abandonné par la suite pour simplifier l'interface.

Semaine 10 : 1 mai - 5 mai

Réflexion sur la manière d'effectuer le copier / coller. Je dois d'abord parcourir et comprendre le code de Sathiya qui a mis en place la sélection d'une zone de l'espace de travail ainsi que la sérialisation des calques.

Je rencontre des problèmes pour enregistrer une capture de l'espace de travail, je ne comprend pas bien comment les couleurs sont gérées lors de la création d'une image.

Semaine 11 : 8 mai - 12 mai

Une première version du copier / coller est implémentée et fonctionne avec quelques problèmes (le calque collé ne se place pas au bon endroit). J'ai résolu les problèmes de couleur et de copie d'image en copiant les pixels un à un et en ne copiant pas les pixels transparents, pour éviter une conversion en blanc.

Semaine 12 : 15 mai - 19 mai

Les problèmes de placement du calque collé sont résolus. Il faut être attentif au système de coordonnées que l'on utilise lorsqu'on exécute un snapshot et ne pas oublier de convertir les coordonnées au besoin.

La fonction copier / coller est fonctionnelle, commentée et nettoyée au mieux.

Semaine 13 : 22 mai - 26 mai

Implémentation de la fonction d'historique. Malgré une réflexion sur le sujet depuis le début de la phase d'implémentation en mars, je n'aurai pas le temps d'implémenter la version «efficace» qui consisterait à enregistrer chaque action et une fonction permettant de l'inverser. J'opte donc pour la solution plus facile à mettre en place qui consiste à enregistrer l'état de l'espace de travail après chaque action de l'utilisateur.

En milieu de semaine l'historique fonctionne et je commence à implémenter la partie «visuelle» qui permettra d'afficher les miniatures dans l'interface de l'application.

Semaine 14 : 29 mai - 2 juin

Fin de la partie visuelle de l'historique et rédaction du rapport.