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

30 mai 2017

Table des matières

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 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.

Conception & Architecture

Technologies utilisées

Java 8

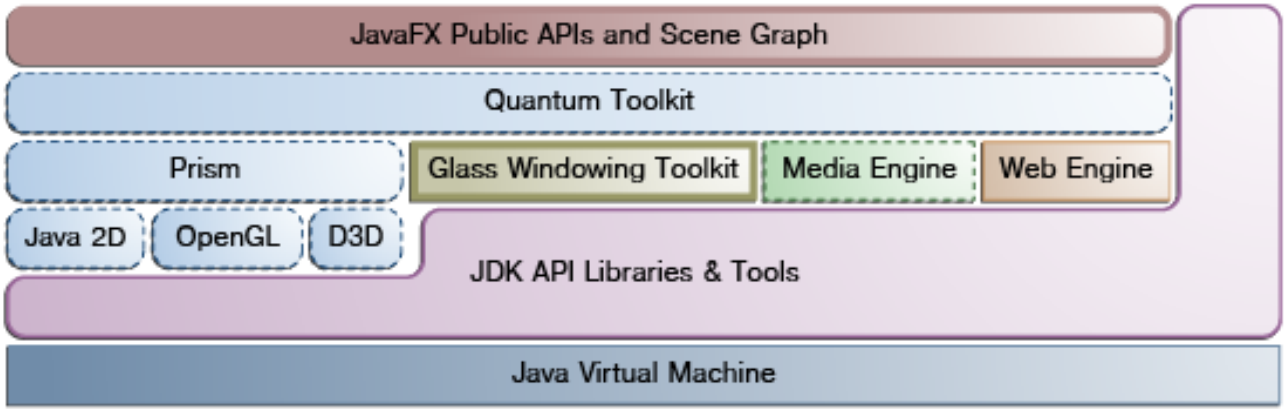
Parmi les deux langages de haut niveau proposés pour l'élaboration de ce projet (Java ou C++), nous avons choisi Java pour sa portabilité, sa sécurité et ses performances. De plus, la dernière version de Java propose la librairie graphique JavaFX qui correspond en tout point à notre projet. Enfin, notre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

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

FIGURE 1 – Architecture de JavaFX



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.

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 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.

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

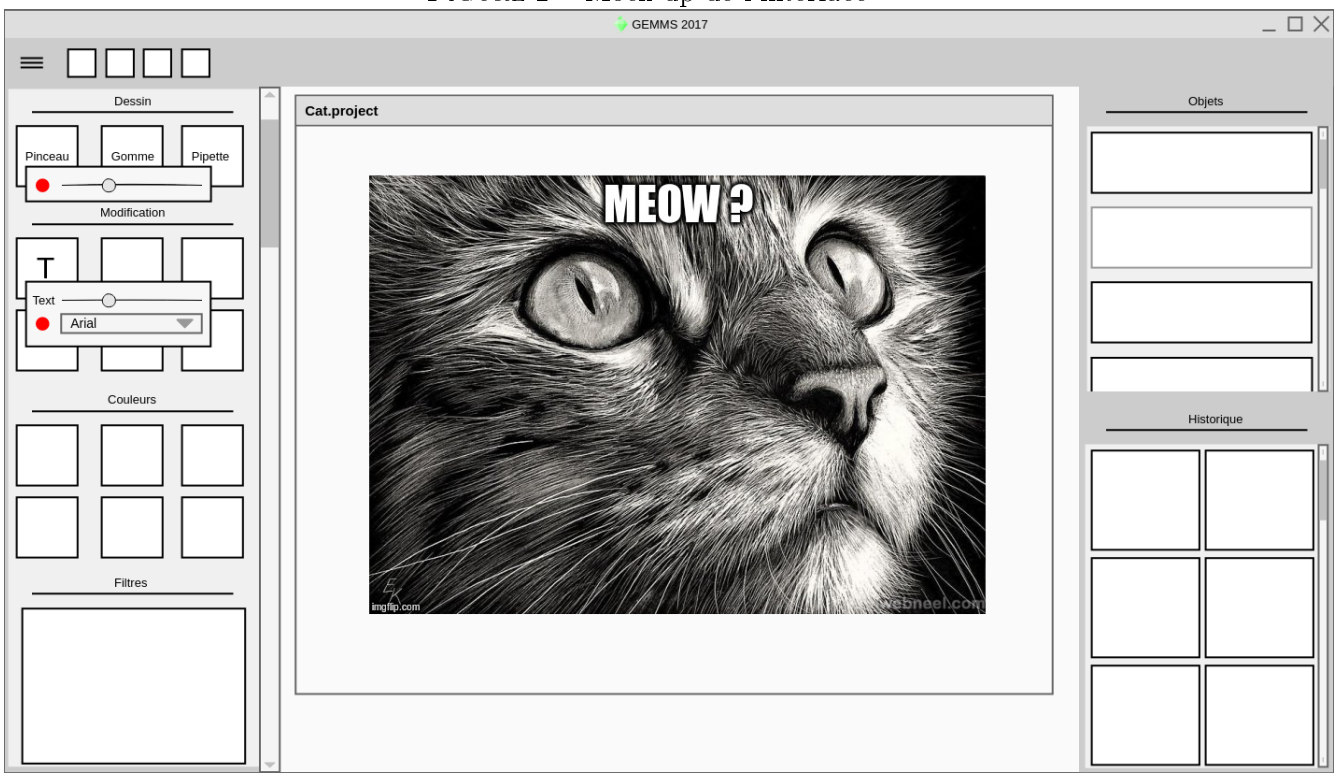
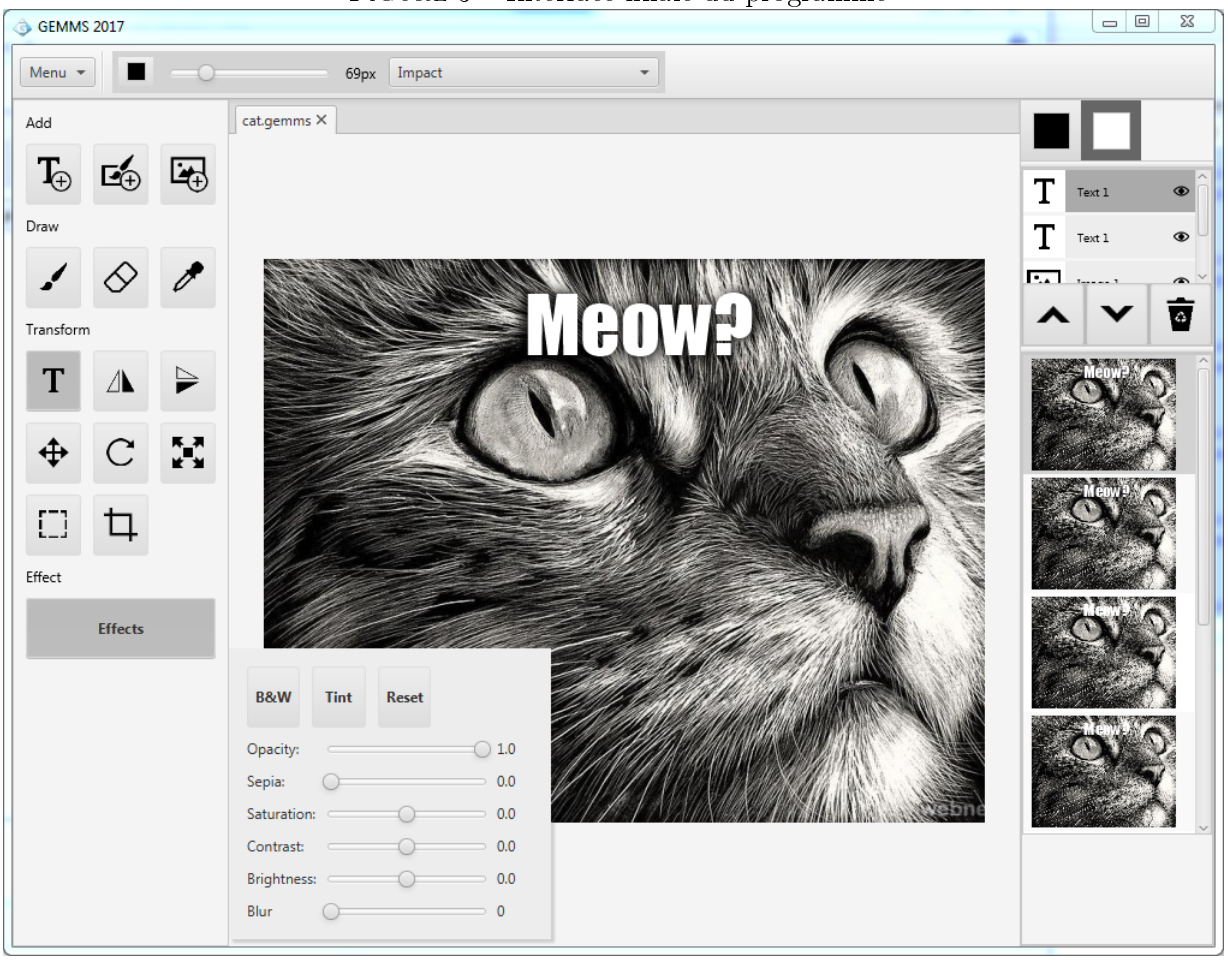


FIGURE 3 – Interface finale du programme



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.

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.

Structure

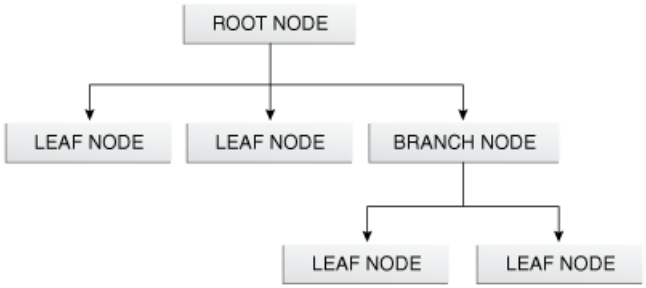
Structure de JavaFX

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 point-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 avions besoin. Ils sont ensuite ajoutés au Scene Graph de la scène principale, et nous pouvons ainsi construire notre application.

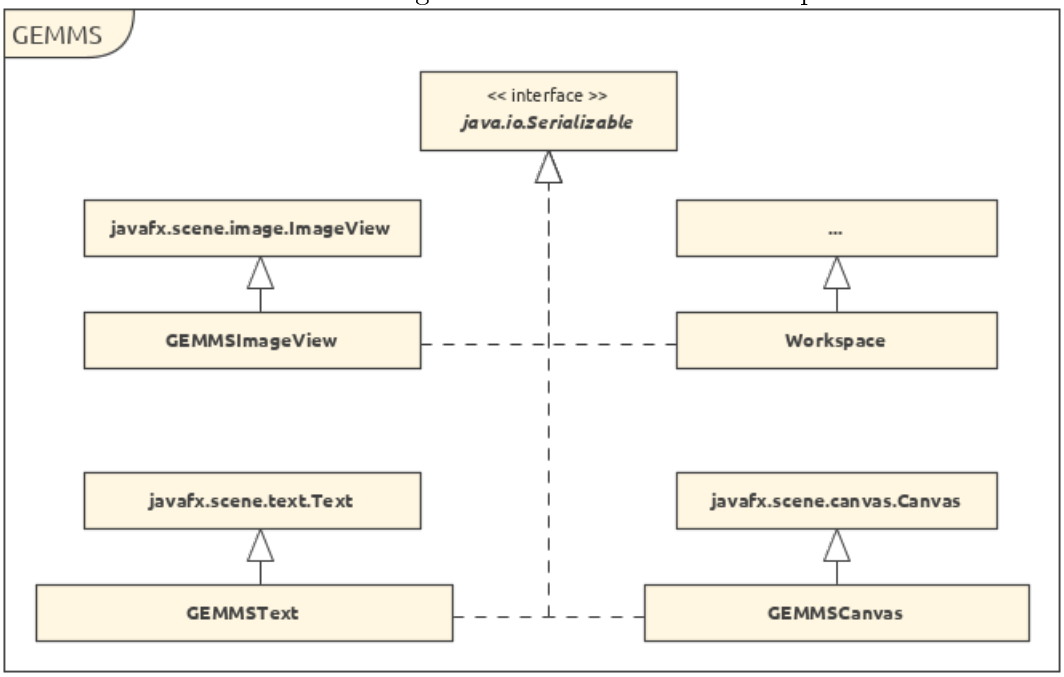
Serialisation

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'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-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.

FIGURE 5 – Diagramme de la sérialisation simplifié



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.

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.

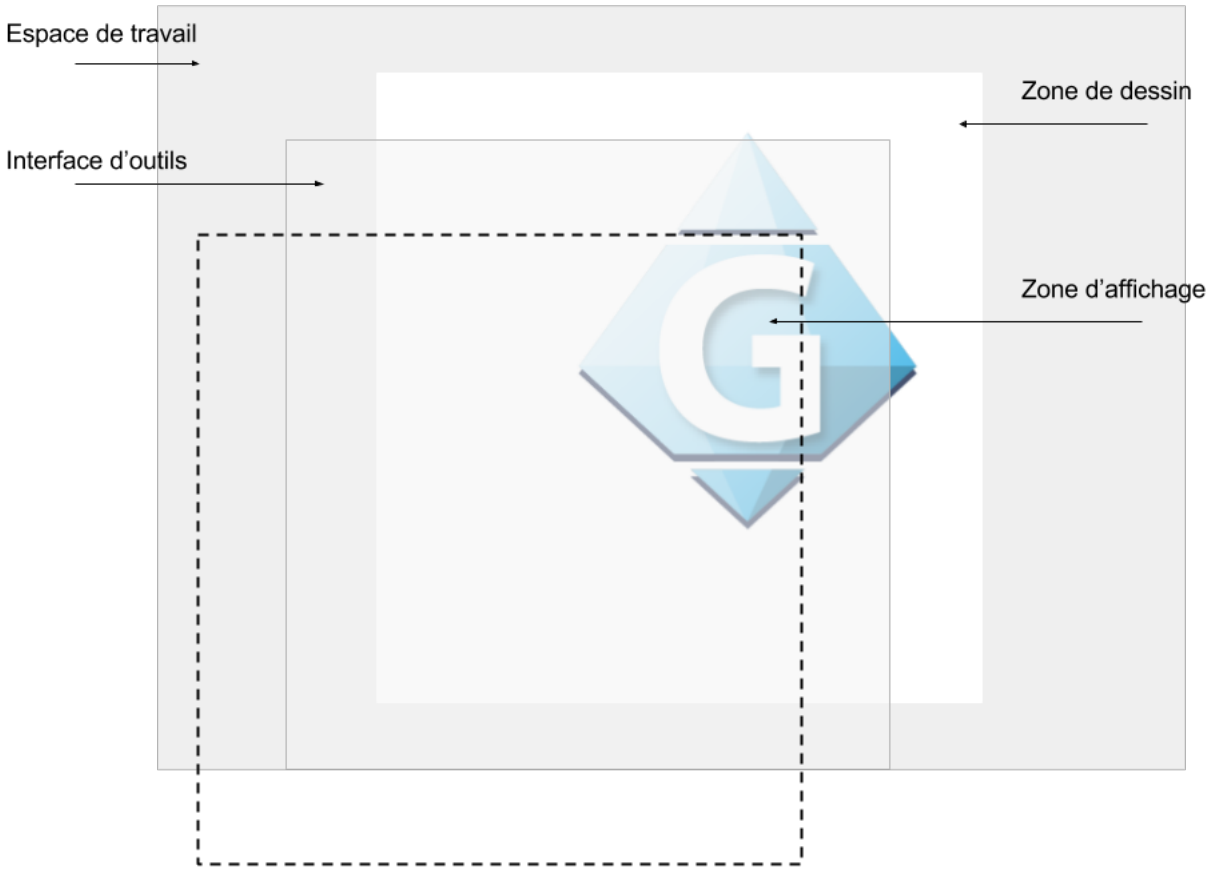
Workspace et liste des calques

Le Workspace correspond à l'espace de travail d'un Document GEMMS. Workspace est une classe personnalisée héritant de la classe StackPane de JavaFX. Il s'insère donc dans le graphe de scène 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 couches 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 graphe de scène) ou seront 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, car cela permet de positionner leurs enfants selon des coordonnées (x, y).

La zone d'affichage correspond à un masque appliqué à l'espace

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 a 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

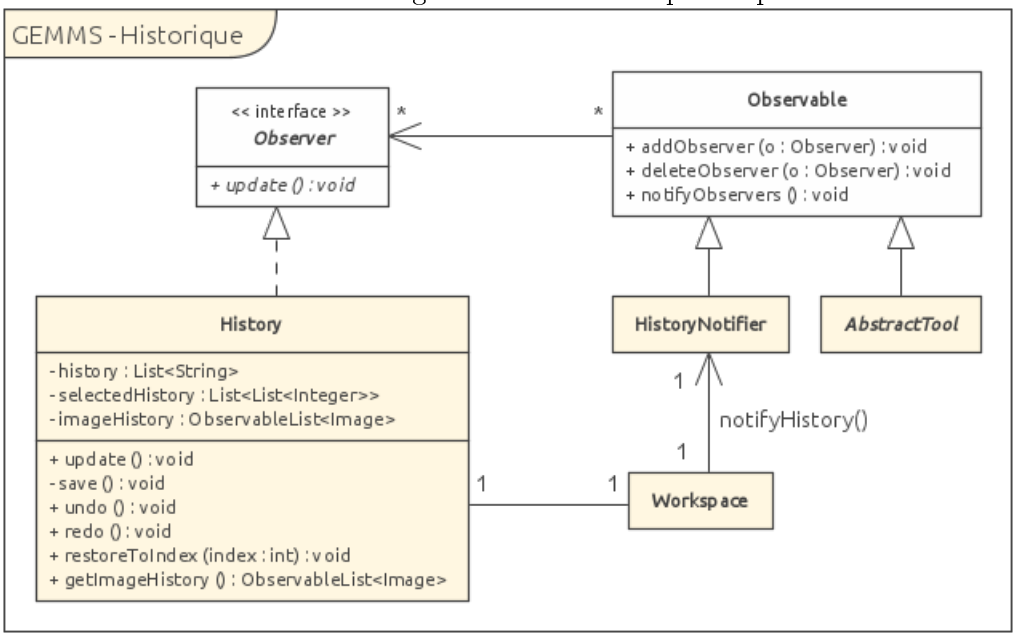
Le collage consiste à désérialiser le contenu du presse-papier et à insérer dans le Workspace courant le ou l'ensemble de calques désérialisés.

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 7.

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

FIGURE 7 – Diagramme de l'historique simplifié



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 (désérialise) 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.

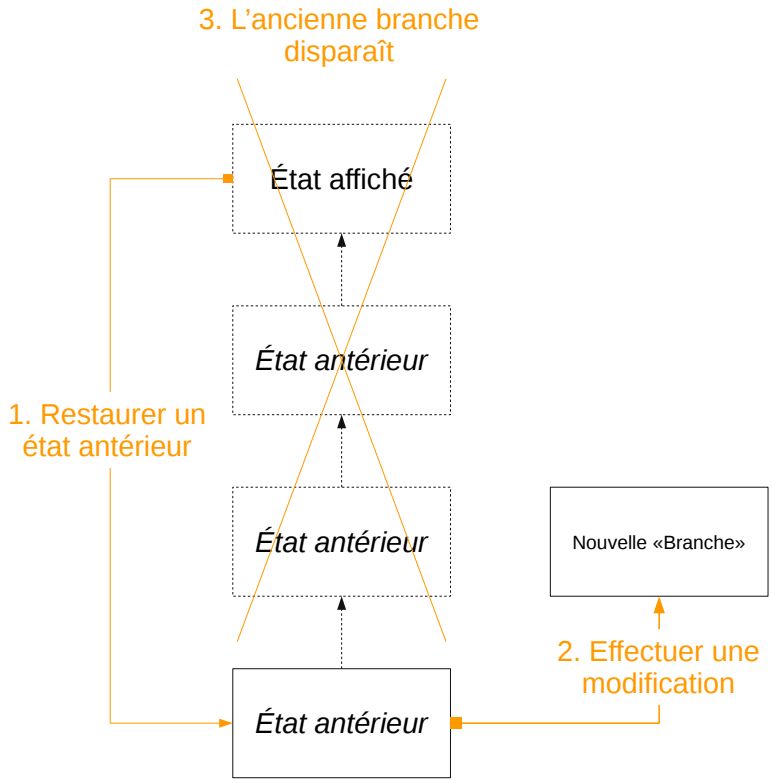
Historique visuel

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 travaille 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 8.

FIGURE 8 – 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

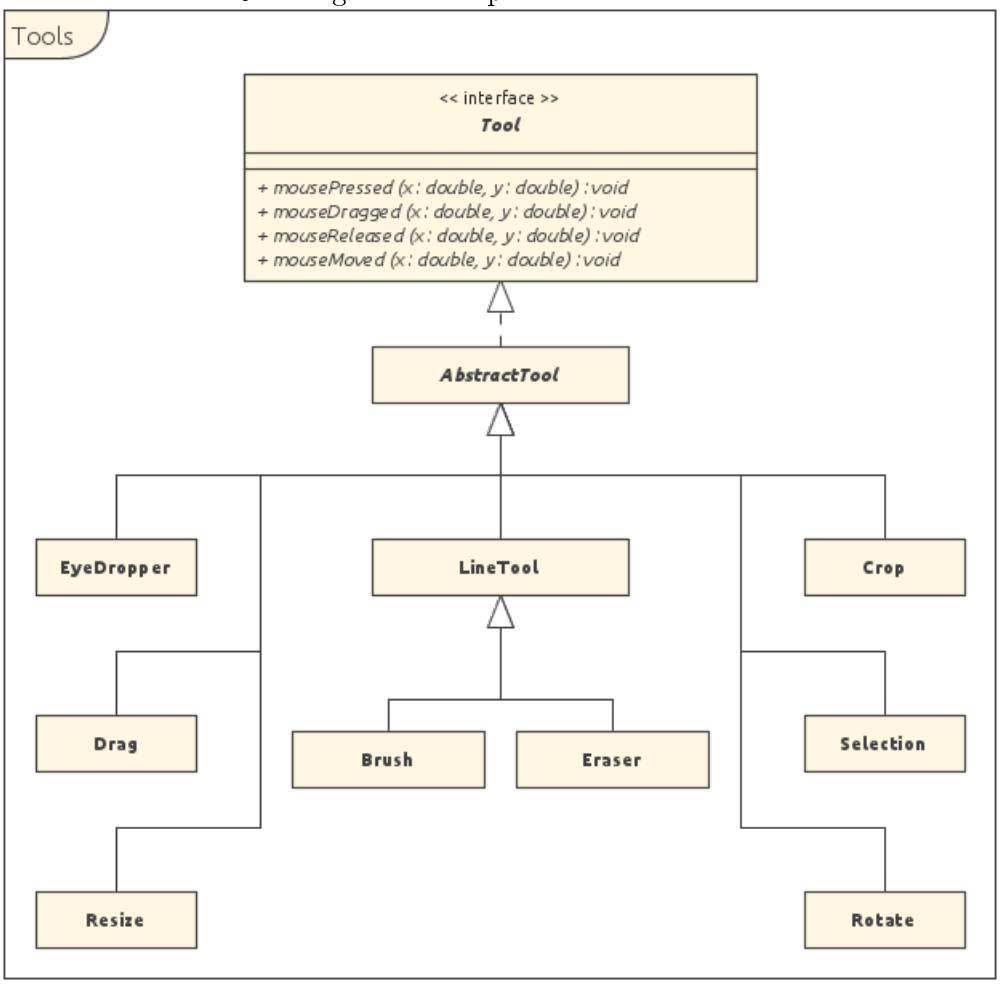
La plupart des outils de l'application fonctionnent grâce à ces trois événements. On pensera notamment au pinceau qui doit dessiner un trait en suivant la souris lors d'un MouseDragged.

Comme on peut le voir sur la figure 9, 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 événement MousePressed, MouseDragged ou MouseReleased.

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

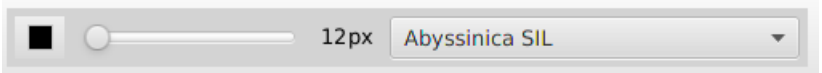
FIGURE 9 – Diagramme simplifié de la hiérarchie des outils



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 même besoin concernent la gestion de la couleur du pinceau, de la taille et de la police de l'outil de modification de texte. La figure 10 illustre l'apparence des réglages de l'outil de modification de texte.

FIGURE 10 – 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

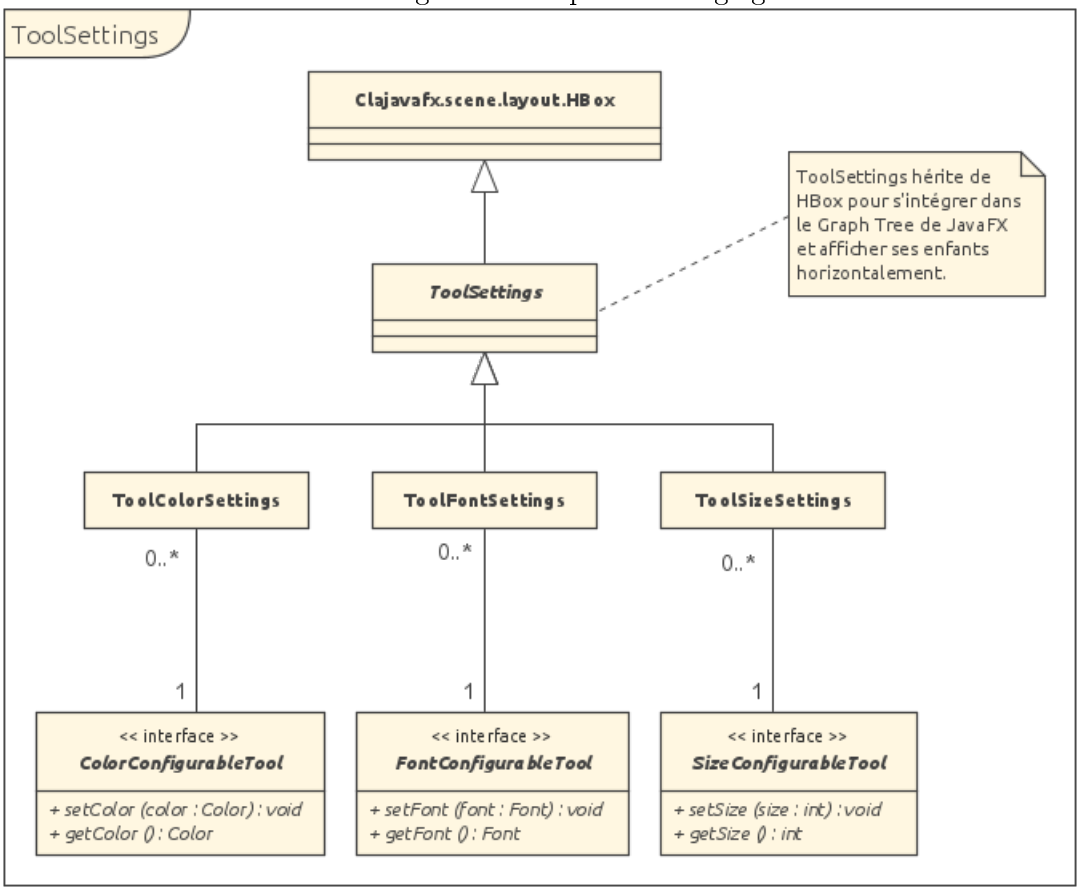
actuelle du pinceau et la garder en mémoire.

Pour ce faire, les réglages sont gérés au moyen d'une hiérarchie de classes (se référer à la figure 11), 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 FontConfigruableTool. 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 11 – Diagramme simplifié des réglages d'outils



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.

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

Déplacement

Rotation

Redimensionnement

Sélection

La sélection permet à l'utilisateur de sélectionner une zone du workspace à l'aide d'un rectangle. Ce dernier peut être déplacer ou recréer si la zone de sélection ne convient pas. Une fois la zone choisie, si le calque courant est un GEMMSCanvas, il 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. Pour cela, il faut

Rognage

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 initialiement 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.

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 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.

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 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.

Glossaire

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. 14

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. 14

Stage Le Stage est le container haut-niveau d'une application JavaFX. Il doit contenir tout le contenu d'une fenêtre. 14

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, 14

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

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

30 mai 2017

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 réfection sur la sérialisation des effets.

Semaine 13 : 22 mai - 26 mai

Ajout de l'effet de flou, ainsi que un bouton « Tint » qui créé 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'élément 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 résolu. 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. 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'élément 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 résolu. 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 - Michael Spierer

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 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

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 desérialiser avec l'aide de Sathiya car il avait déjà bosser 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, deplacement 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, en plus de sérialiser les calques qui sont de base non-sérialisable, sérialiser les transformations.

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 regler 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 reévalue 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

Finition du rapport et journal de bord.