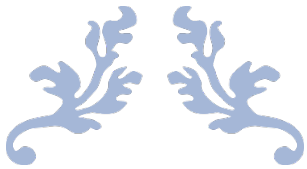

BLAAJJPAINT

RAPPORT DE PROJET

Auteurs :

Benoît Schopfer
Loyse Krug
Adrien Allemand
Antoine Rochat
Jérémie Châtillon
James Smith

Client :

René Rentsch

Mai 2018

heig-vd

Hes·so

Table des matières

1 Introduction

Dans le cadre du projet du 4ème semestre de nos études d'Informatique Logiciel à la HEIG-VD, il nous est demandé de mener de bout à bout un projet de groupe et développer un programme informatique mettant en œuvre nos connaissances acquises jusque-là.

La durée totale de 560 heures, 90 pour chacun des 6 élèves participants, est à répartir sur 15 semaines à raison de 6h par personne par semaine.

Pour ce projet, nous avons décidé de réaliser une application de traitement d'image matricielle, permettant de créer, modifier et exporter une image au format PNG ou JPEG.

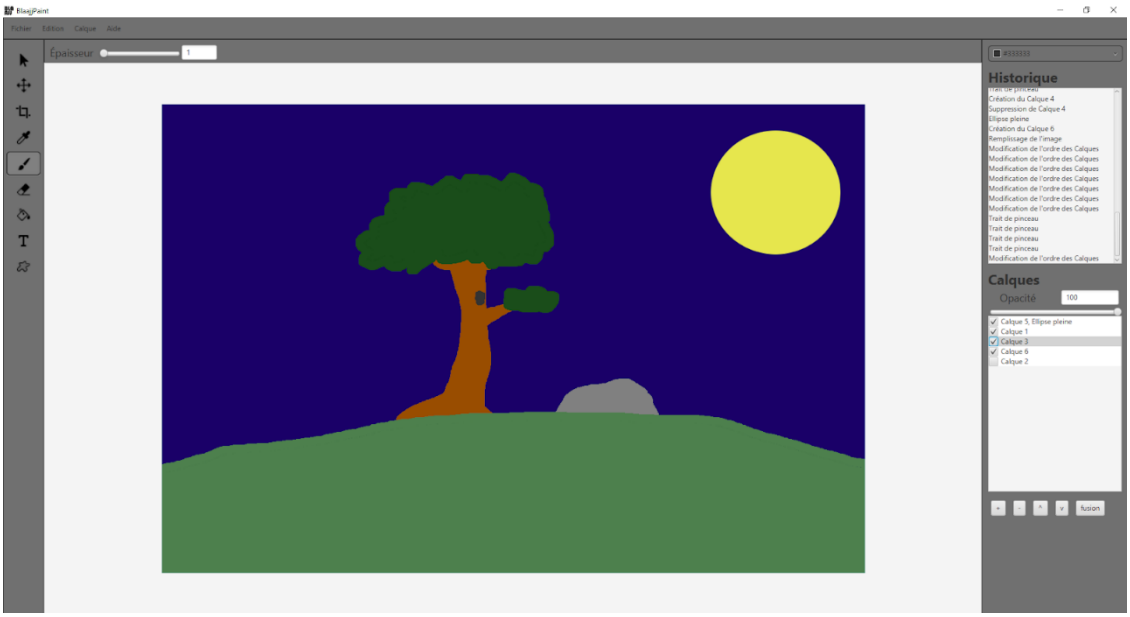

Figure 1: Création d'une image à l'aide du programme BlaajjPaint.

2 Objectifs du projet

2.1 Objectifs pédagogiques

Ce projet avait pour but de mettre en pratique toutes les connaissances acquises durant les 4 premiers semestres de la formation d'ingénieur en Informatique Logiciel à la HEIG-VD. Il demandait une organisation et une gestion de groupe ainsi qu'une mise en commun des connaissances.
Ses objectifs pédagogiques principaux étaient :

- La planification du projet
- L'apprentissage de nouvelles technologies
- L'apprentissage du travail en groupe sur un projet de taille conséquente

2.1.1 Planification du projet

Les objectifs de la planification du projet étaient multiples. Tout d'abord, il nous a fallu discuter et nous mettre d'accord afin de définir précisément le cahier des charges du projet, puis estimer le temps nécessaire à l'implémentation de chacune de ses fonctionnalités. Pour cela, il nous était

demandé de fournir un *Gantt* permettant de visualiser le temps requis pour implémenter chaque fonctionnalité ainsi que la répartition des fonctionnalités entre les différents membres du groupe. Par la suite, l'objectif n'était pas de nous tenir coûte que coûte à cette planification, mais de la comparer à l'avancée effective du projet afin de déceler un éventuel retard et agir en conséquence de sorte à le combler au plus vite.
C'est pourquoi, chaque lundi, nous comparions le planning initial avec l'avancée réelle du projet. Forts de cette analyse, nous nous répartissions le travail pour la semaine à venir et déterminions les objectifs de chacun. Ainsi, lorsque l'un de nous prenait du retard sur l'implémentation d'une fonctionnalité ou rencontrait des difficultés, nous nous arrangions pour l'aider, soit en allouant plus de monde à la résolution du problème, soit en diminuant ses tâches à faire pour la semaine suivante afin qu'il ait plus de temps pour rattraper son retard.

2.1.2 Apprentissage de nouvelles technologies

Un autre objectif pédagogique de ce projet était l'apprentissage de nouvelles technologies. En effet, pour ce travail, nous avons dû apprendre à nous débrouiller seuls, autant pour chercher les informations nécessaires à la réalisation de nos fonctionnalités, que pour prendre des décisions techniques.
Dans notre cas, les nouvelles technologies utilisées étaient principalement la librairie *JavaFX* pour la réalisation de l'interface graphique de notre application ainsi que la librairie *XStream* pour la sérialisation. Nous n'avons d'ailleurs finalement pas utilisé cette seconde libraire car après plusieurs semaines de tests, il s'est avéré que son utilisation n'était pas compatible avec *JavaFX* !

2.1.3 Apprentissage du travail en groupe

Ce projet était à réaliser en équipe de 5 à 6 personnes afin de nous apprendre à collaborer et nous montrer tant les forces que les difficultés d'un travail en groupe.
Pour notre projet, nous étions 6 membres. La collaboration et la coordination étaient d'une importance capitale, étant donné que l'implémentation d'un logiciel de dessin se concentre surtout sur un pôle, celui de l'interface graphique et des interactions avec celle-ci. Ainsi, l'un des défis pour nous était de suffisamment communiquer entre nous et de se coordonner au mieux afin de ne pas se marcher dessus ou refaire des choses qu'un autre avait déjà fait !
Il nous a donc fallu apprendre à nous coordonner, à fixer des conventions de codage et à être aussi clair et précis que possible entre nous.

2.2 Objectifs fonctionnels

Notre logiciel a pour objectif de permettre d'effectuer les traitements de base d'images matricielles. Il offre des outils de dessin classiques, quelques transformations simples ainsi qu'un mécanisme de calque dynamique dans la même idée que propose le logiciel *Photoshop*.

Nous souhaitions qu'à l'issue de ce projet, un utilisateur du logiciel puisse créer un nouveau projet, y importer, une ou plusieurs images, les modifier à l'aide d'outils tels que le pinceau, la gomme, le pot de peinture, ou encore y appliquer des transformations tels que la symétrie ou la rotation. On souhaitait également qu'il puisse enregistrer son projet dans un format .blaajj afin de pouvoir le rouvrir ultérieurement et reprendre sa modification là où il en était, et enfin, nous souhaitions que l'utilisateur puisse exporter son travail terminé dans une image au format *PNG* ou *JPG*.

3 Concepts généraux

Notre projet, *BlaajjPaint* est un logiciel de dessin et de traitement d'images matricielles implémenté en *Java*.

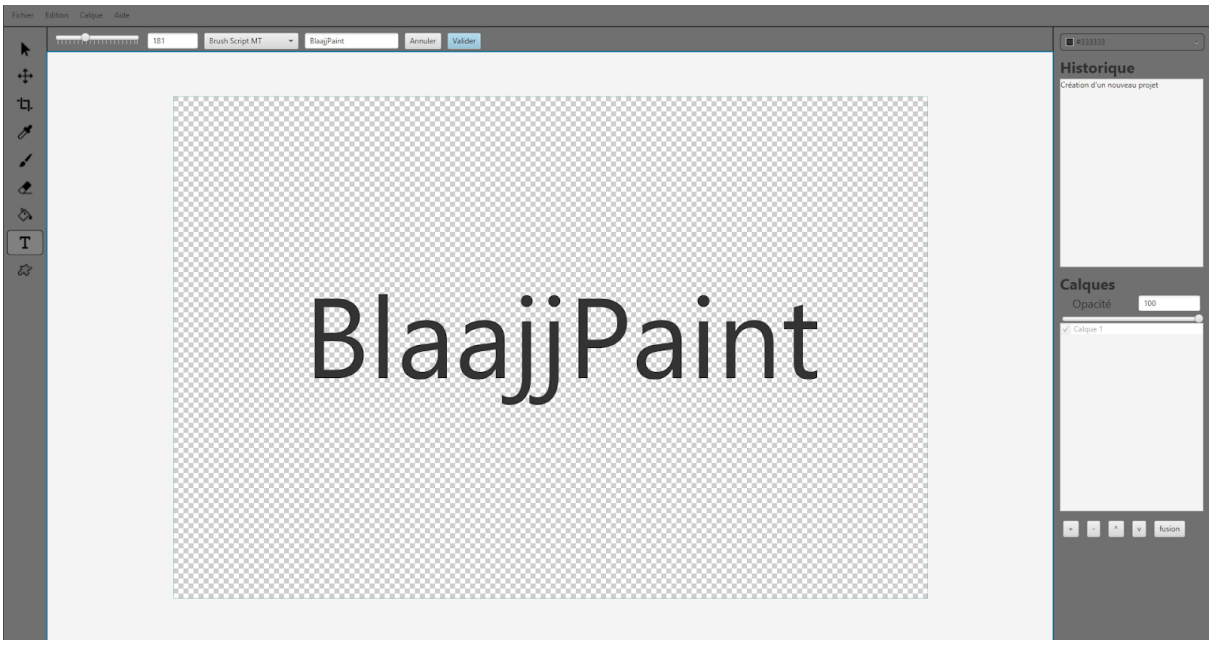

Figure 2 Interface graphique du programme.

Son fonctionnement est centré sur l'utilisation de calques pour former l'image. Un calque est assimilable à une "couche". Les calques formant l'image sont empilés les uns sur les autres. Le calque au sommet de la pile est le calque au premier plan et masque les calques situés au-dessous.
Notre application supporte donc le multi-calque et offre les fonctionnalités principales suivantes :

- Dessiner sur un calque à l'aide de l'outil *pinceau*,
- Effacer les pixels d'un calque avec l'outil *gomme*,
- Effectuer des coloriages de zone à l'aide de l'outil *Pot de peinture*,
- Ajouter des formes prédéfinies,
- Ajouter du texte,
- Rogner et redimensionner un calque
- Déplacer un calque dans l'image,
- Modifier l'opacité d'un calque,
- Ajouter et supprimer des calques,
- Modifier l'ordre de la liste des calques,
- Fusionner deux calques,
- Effectuer une symétrie horizontale ou verticale d'un calque
- Appliquer une rotation à un calque

De plus, un historique est intégré permettant, grâce à un mécanisme d'*Undo/Redo* d'annuler et rétablir toutes les actions persistantes effectuées sur un projet. Le principe est le suivant :
Toutes les actions qui modifient l'état du projet doivent être annulables (*Undo*) et ré-applicables (*Redo*).

Dans notre projet, il y a deux types de variables :

- Les données persistantes qui définissent l'état du projet. Ces données sont l'état de chaque calque du projet (c'est-à-dire leur visibilité, leur nom, leur transparence et leur contenu), le nombre de calques dans le projet et leur ordre. Toutes ces données représentent les informations qui doivent être sauvegardées dans un fichier au format *.blaajj* et rétablies lorsqu'on ouvre une sauvegarde.
- Les données temporaires qui représentent l'état du projet à un certain moment mais ne sont pas essentielles. Ces variables ne sont pas sauvegardées, mais sont tout de même utiles pour l'utilisation de l'application. Parmi ces données, on compte notamment les piles *Undo/Redo* de l'historique des actions, l'outils en cours d'exécution, la couleur choisie etc.

4 Conception du projet

4.1 Technologies utilisées

Figure 3: Principales technologies utilisées : Java, JavaFX et GitHub

Nous avons choisi de réaliser notre application en *Java*.

Pour l'interface graphique, nous nous sommes appuyés sur la librairie *JavaFX* d'Oracle. Cette librairie offre de nombreuses possibilités permettant d'obtenir rapidement des interfaces graphiques relativement élégantes. L'un des grands atouts de cette librairie est les fichiers *FXML* qui, avec des outils tels que *Scene Builder*, permettent de créer l'interface graphique visuellement. De plus, *JavaFX* inclut de nombreuses classes de dessin que nous avons pu utiliser pour réaliser plusieurs fonctionnalités de notre programme.

Pour la sauvegarde d'un projet, nous avons utilisé l'interface *Serializable* du package *java.io*. Nous avons premièrement opté pour la librairie *XStream* mais après de nombreux tests infructueux, nous avons réalisé que certains composants de *JavaFX*, tel que la classe *Canvas* n'était pas sérialisables à l'aide de cette librairie. Comme cette classe est la classe que nous avons utilisé pour implémenter les calques, nous n'avions pas d'autres choix que de trouver une autre solution. Par conséquent, nous nous sommes tournés vers l'interface *Serializable*. C'est pourquoi, toute classe contenant des données persistantes à sauvegarder implémentent cette interface et définissent comment sérialiser et désérialiser ces données.

Afin de gérer le partage de code, résoudre les conflits et être certain de toujours posséder la dernière version du code, nous avons créé un dépôt *git* hébergé sur la plateforme *GitHub*.

4.2 Modèles conceptuels utilisés

Initialement, nous avions prévu de baser notre programme sur le modèle de conception *MVC* (*Modèle-Vue-Contrôleur*). Ce modèle nous paraissait adéquat puisque la séparation *vue-contrôleur* est "incluse" par l'utilisation de *JavaFX* où la vue est représentée par les fichiers *FXML* et les contrôleurs par les classes associées à ces fichiers. Nous pensions alors qu'il nous suffirait de créer le modèle, qui serait notre projet. Seulement, au fur et à mesure de l'apprentissage de *JavaFX*, nous avons réalisé qu'en utilisant cette bibliothèque, le modèle et le contrôleur étaient très difficiles à dissocier. Comme nous ne sommes jamais parvenus à les séparer de façon satisfaisante, nous avons décidé de mettre cette structure de côté.

Notre projet utilise néanmoins deux autres modèles conceptuels réutilisables.
Le modèle *Singleton* utilisé notamment par tous les outils, permet d'avoir une seule et unique instance d'une classe. Il est particulièrement utile car cette classe devient accessible depuis n'importe quelle classe du projet, nous évitant ainsi de devoir faire les liens "à la main".
Ceci a été particulièrement utile pour les deux classes principales, *MainViewControler* et *Project* qui sont toutes deux appelées de partout dans le projet.

Pour l'implémentation de la fonctionnalité *Undo/Redo*, nous utilisons une variante du modèle *Commande*. Ce patron de conception a pour but d'unifier toutes les commandes par une même interface lors de leur exécution. Dans notre projet, cette interface se nomme *ICmd* et est implémentée par toutes les classes internes enregistrant une action persistante.

5 Architecture

L'architecture globale du projet est basée sur le paterne "*Modèle-Vue-Contrôleur*" qui, comme expliqué précédemment s'apparente plus à un paterne "*Vue-Contrôleur*" où les contrôleurs et le modèle sont fusionnés.
Dans notre situation, la classe *Project* pourrait s'apparenter au modèle car elle contient les variables d'états du projet. Cependant, elle gère aussi des évènements, ce qui devrait, selon le modèle *MVC* être fait dans un contrôleur… C'est pour cette raison que nous avons plus un patron "*Vue-Contrôleur*" où le contrôleur et le modèle sont fusionnés qu'un réel modèle *MVC*.

Tous les contrôleurs du projet se trouvent dans le package *controller*, tandis que les vues (les fichiers *FXML*) se trouvent dans le package *view* des ressources. Le *main* est situé dans un package *main* et le package *utils* contient quelques classes utilitaires tel que la classe gérant la sauvegarde.
L'interface graphique est entièrement créée en *FXML*. Nous avons vraiment cherché à utiliser au maximum les possibilités offertes par *FXML* en le chargeant d'un maximum de tâches. Ainsi, c'est lui qui crée les contrôleurs lors de l'initialisation de l'application, fait le lien entre le *FXML* et le contrôleur et initialise les variables du contrôleur. C'est lui aussi qui est chargé de créer les *listeners* de chaque bouton de l'interface graphique et d'appeler la méthode correspondant à l'action du bouton lorsqu'il est cliqué.
La vue (package *view*) est séparée en quatre parties :

- La fenêtre principale (*mainView.fxml*) qui est le nœud racine de la vue et contient tous les autres fichiers *FXML*. Elle contient aussi la zone de travail où les calques apparaissent et peuvent être modifiés par l'utilisateur.

- La *menubar* (package *view.menubar*) contient tous les éléments gérant la barre de menus. Il s'agit de la barre située tout en haut de l'application et regroupant quatre menus principaux, *Fichier*, *Edition*, *Calque* et *Aide* dont les détails sont donnés dans le manuel d'utilisateur.
- Le *rightMenu* (package *view.rightMenu*) contient les fichiers *FXML* gérant trois éléments importants de l'application : le sélecteur de couleur, l'historique *Undo/Redo* et la liste de tous les calques du projet. Le *rightMenu* contient également les boutons permettant de gérer les calques et plus précisément d'ajouter, supprimer, changer la position ou l'opacité et fusionner les calques. Le *rightMenu* est affiché à droite de l'espace de travail.
- La *toolBar* (package *view.tools*) affiche la barre d'outils à gauche de l'interface graphique. Elle contient tous les outils nécessaires à l'édition d'un calque.

5.1 Calques

Pour implémenter les calques, nous avons choisi d'utiliser la classe *Canvas* de *JavaFX*. Cette classe met à disposition une zone rectangulaire de pixels éditables. De plus, il est possible de masquer un *Canvas* et de modifier son opacité. Un autre avantage de cette classe est qu'il est possible d'extraire les pixels au format *Image*.
Par conséquent, nous avons fait hériter la classe *Layer* de la classe *Canvas* de *JavaFX* pour pouvoir y ajouter des attributs et y ajouter nos méthodes d'éditions. Les attributs ajoutés sont :

- Un nom pour l'affichage de la liste des calques.
- Un id qui permet de donner un nom unique au calque. L'id est déterminé à l'aide d'une variable statique incrémentée lors de la création de chaque nouvelle instance de *Layer* qui n'est pas temporaire.
- Une description de calque précisant lors de l'affichage s'il s'agit d'un calque normal ou d'un calque de texte, ou de forme.
- Une variable qui indique si c'est un calque temporaire. Si c'est le cas, il n'est pas affiché dans la liste de calque et n'incrémente pas le nombre de calques du projet.

Cette classe implémente l'interface *Serializable* nous permettant d'enregistrer les calques du projet. Vous trouverez plus d'informations à ce propos au chapitre 5.3 Sauvegarde.
Comme dit précédemment, il est possible de récupérer un objet de type *Image* à partir d'un objet de type *Canvas*. Nous avons utilisé cette fonctionnalité afin de stocker des images des calques lorsqu'il nous fallait enregistrer l'état d'un calque, comme par exemple lors de la sauvegarde, lors de l'enregistrement de l'état d'un calque avant et après modification pour le *Undo/Redo*, et à de nombreuses autres reprises. Cela nous a permis de réduire l'utilisation de la mémoire lors de nos manipulations et d'améliorer les performances de certaines fonctionnalités.

5.2 Undo/Redo

Pour implémenter la fonctionnalité d'*Undo/Redo*, nous avons utilisé le modèle commande. Ce modèle est implémenté par l'interface *ICmd*, définissant les méthodes que chaque commande doit implémenter.

Pour chaque fonctionnalité ou outil modifiant l'état du projet tel que défini dans les 3 Concepts généraux, nous avons créé une classe interne sauvegardant la modification et implémentant l'interface *ICmd*. Ainsi, lorsqu'une modification est appliquée, une instance de la classe interne de sauvegarde est créée puis enregistrée dans le singleton *RecordCmd*. De cette manière, chaque outil est responsable au moment où il applique une modification, de générer un objet *ICmd* enregistrant

l'état du projet avant et après sa modification et définissant les méthodes permettant d'annuler ou de rétablir la modification concernée. Le singleton *RecordCmd* contient l'historique des modifications (sous la forme d'une liste de commande exécutées et d'une liste de commandes annulées) et permet d'annuler ou rétablir une *ICmd* de la liste en appelant la méthode *undo()* ou *redo()* de la commande.

Comme les fonctionnalités modifiant l'état du calque sont relativement nombreuses, nous avons un nombre important de classes internes de sauvegarde, comme vous pouvez le constater dans le diagramme *UML* de notre projet joint en annexe.

5.3 Sauvegarde

Pour gérer la sauvegarde du projet ainsi que l'ouverture d'une sauvegarde, nous avons créé la classe *SaveProject*. Cette classe a pour but de définir comment écrire un projet dans un fichier *.blaajj* pour l'enregistrer et, à l'inverse, comment lire les données d'un fichier *.blaajj* afin de pouvoir rétablir le projet enregistré. *SaveProject* implémente également le modèle singleton afin d'être certain de n'avoir qu'une seule instance à la fois et de pouvoir appeler cette classe depuis n'importe quel contrôleur.
Les deux méthodes principales de cette classe sont les méthodes *SaveAs(File f)* et *Open(File f)* :

- SaveAs(File f): Enregistre le projet dans le fichier reçu en paramètre en le transformant en *ObjectOutputStream*, puis en appelant sa méthode *writeObject* avec l'instance du singleton *Project* en paramètre afin de l'enregistrer.
- Open(File f): Lit le fichier reçu en paramètre, le transforme en *ObjectInputStream* et restaure le projet qu'il contient à l'aide de la méthode *readObject()*.

Les méthodes *writeObject()* et *readObject()* appellent les méthodes du même noms situées dans les classes *Project* et *Layer* afin de sérialiser ces objets lorsqu'ils sont rencontrés.

Dans la classe *Project se trouve* toutes les données à sauvegarder. Ces données sont :

- La hauteur et la largeur du projet,
- Le nombre de calques,
- La liste de calques (liste d'objets de type *Layer*).

Pour les calques (classe *Layer*) les données que nous enregistrons sont :

- La hauteur et la largeur du calque
- La position X, Y du coin supérieur gauche du calque par rapport au coin supérieur gauche de l'espace de travail,
- L'opacité du calque,
- La visibilité du calque,
- Le contenu du calque enregistré sous forme d'image.

Pour que les méthodes *open()* et *saveAs()* de *SaveProject* puissent sérialiser des objets de type *Project* et *Layer*, il faut que ces deux classes implémentent l'interface *Serializable* et redéfinissent les méthodes *writeObject(ObjectOutputStream s) et readObject(ObjectInputStram s)* afin de définir comment sérialiser et désérialiser ces objets.
L'ordre de sérialisation des variables est important. La méthode *writeObject* écrit les variables dans un ordre précis que la méthode *readObject* doit impérativement lire dans le même ordre!

Pour sérialiser la liste de calques contenue dans l'instance de *Project*, nous avons tout d'abord dû enregistrer le nombre de calques de la liste, puis sérialiser chaque calque l'un après l'autre dans leur ordre d'apparition dans la liste. Il est indispensable d'enregistrer le nombre de calques de la liste afin de savoir combien la désérialisation doit en lire.

Pour la classe *Layer*, nous aurions voulu sérialiser son canevas directement, mais comme la classe *Canvas* de *JavaFX* ne le permet pas, il nous a fallu trouver un autre moyen. Pour contourner le problème, nous avons dû convertir le calque en un objet de type *Image*, qui lui, est sérialisable.

5.4 FXML et contrôleurs

Avec *JavaFX*, on a la possibilité d'associer un contrôleur à chaque fichiers *FXML*. Un contrôleur peut être n'importe quelle classe *Java*. Nous avons donc créé, pour chaque fichier *FXML* de l'interface graphique, un contrôleur associé. Afin de facilement pouvoir déterminer quelle classe contrôle quel fichier *FXML*, nous avons toujours nommé les contrôleurs de la façon suivante : *NomFXMLAssocié + Controller.java*
Par exemple, le fichier *MainView.fxml* est associé au contrôleur *MainViewController.java*, *ToolBar.fxml* au contrôleur *ToolBarController.java* etc.

Les interactions entre les fichiers *FXML* et les contrôleurs sont simples. Lorsque l'utilisateur fais une action dans l'interface graphique, comme par exemple cliquer sur un bouton, le *FXML* contenant ce bouton appelle la méthode associée à ce bouton située dans son contrôleur.
C'est le contrôleur qui est alors chargé d'appliquer l'action du bouton cliqué. Par exemple, dans notre application, si l'utilisateur clique sur le menu *Fichier -> Enregistrer sous*, cela aura pour effet d'appeler la méthode *handleSaveAs()* du contrôleur *MenuBarController*. Le code de cette méthode défini donc le comportement du sous-menu. Ici, la méthode ouvrira une fenêtre invitant l'utilisateur à choisir l'emplacement et le nom à donner à la sauvegarde, puis appellera le singleton *SaveProject* afin d'exécuter la sauvegarde comme expliqué au point 5.3 Sauvegarde.

Il est important de noter que tout ce qui est fait via le fichier *FXML* aurait également pu être fait en ligne de code dans les contrôleurs. En revanche, *FXML* n'intègre pas toutes les possibilités offertes par le code. C'est pourquoi, certains contrôleurs définissent des comportements supplémentaires à leur fichier *FXML associé*. Par exemple, la fonction *handleNew()* du contrôleur *MenuBarController* ouvre une nouvelle fenêtre en chargeant un nouveau fichier *FXML*, *windowsNewProject.fxml* qui, automatiquement, charge le contrôleur associé (*WindowsNewProjectController*). Ce contrôleur aussi ajoute des fonctionnalités au fichier *FXML* qui lui est associé. En effet, il ajoute des listener aux textFields définit dans le *FXML* afin de vérifier le contenu entré par l'utilisateur et empêcher la validation si ce contenu n'est pas correct.

6 Description des difficultés rencontrées

Nous nous sommes heurtés à de nombreuses difficultés durant la réalisation de ce projet. Ces problèmes peuvent être séparés en 2 catégories. La majorité de ces problèmes étaient des difficultés techniques souvent dues à une mauvaise connaissance des outils utilisés, notamment de *JavaFX* et *FXML,* ou encore à de mauvais choix au niveau de l'implémentation. Nous avons aussi dû faire face à des soucis organisationnels dû au nombre de personnes dans le groupe et aux difficultés que cela implique au niveau de la coordination et de la communication.

6.1 Difficultés techniques

Les difficultés techniques que nous avons rencontrées étaient la plupart du temps dues soit à notre inexpérience sur des projets de cette envergure, soit au manque de connaissances sur les librairies utilisées.

Comme discuté précédemment, nous avons rencontré un problème majeur au niveau de la mise en place du modèle *MVC* dans notre architecture. Mais au fur et à mesure que nous avancions dans le projet et que nous prenions *JavaFX* en main, nous nous sommes rendus compte qu'en utilisant cette librairie, il devenait très difficile de séparer le contrôleur du modèle. Nous avons donc dû adapter notre modèle et notre architecture pour qu'il suive le fonctionnement de *JavaFX*. Ce problème est entièrement dû à notre méconnaissance de la librairie utilisée.

Cette méconnaissance de la libraire *JavaFX* prend source dans son apprentissage, qui a été un véritable défi pour chacun et qui a posé des difficultés tout au long de la réalisation du projet. Nous avons réellement sous-estimé le temps nécessaire à la compréhension de cet outil et avons réalisé, souvent trop tard que nous avions implémenté quelque chose d'une façon bien plus compliquée que nécessaire. L'exemple criant étant l'outil texte implémenté dans la classe *TextTool*. Nous avons passé un temps non négligeable à l'implémenter et à le déboguer avant de nous rendre compte, par hasard, quelques jours avant le rendu que *JavaFX* incluait une classe *Text* permettant de faire tout ce dont nous avions besoin !

Cette méconnaissance a également engendré plusieurs choix durant la réalisation du projet qui n'ont pas toujours été les bons. Par exemple, nous avons beaucoup trop utilisé la classe *Canvas*. Pour plusieurs outils, tel que l'outil de texte ou celui des formes, nous avons utilisé un calque temporaire afin d'offrir à l'utilisateur un aperçu avant qu'il ne finalise la forme. Ces calques temporaires sont devenus un vrai cauchemar lorsque nous sommes passés aux derniers tests du programme. Ils étaient en effet très régulièrement la cause de problèmes détectés.

Les problèmes de ce genre ont été nombreux durant le projet, et sont au final, à l'origine de nombreux bugs décelés dans l'application mais que nous n'avons pas pu corriger faute de temps. Ces bugs sont listés au point 7.2 Bugs restants.

Une autre méconnaissance qui nous a coûté beaucoup de temps est la méconnaissance de *git*. L'outil que nous utilisions pour partager notre code nous a, à plusieurs reprises, joué des tours suite à la mauvaise manipulation de l'un ou l'autre membre du groupe. Nous avons bien trop régulièrement dû réécrire du code ou passer des heures à corriger des bugs dû à un merge ou un push mal fait !

6.2 Difficultés organisationnelles

Tout au long de ce projet, l'ambiance au sein de notre groupe a été excellente. Nous avons su nous soutenir, nous aider et nous motiver pour arriver à un résultat dont nous sommes fiers, compte tenu de l'ampleur du projet.
Mais malgré cette bonne entente au sein du groupe, nous avons fait face à plusieurs problèmes organisationnels. Les deux problèmes majeurs dont a souffert notre groupe sont la gestion des divergences d'opinion et surtout le manque de communication.

Nos divergences d'opinion, dues aux différences de connaissances et d'expérience entre les membres du groupe, ont fortement impacté la structure de notre projet. En effet, elles nous ont menées à de longues discussions pour déterminer quelles étaient les solutions optimales et de quelle façon il fallait implémenter les fonctionnalités. Nous avons sans doute perdu beaucoup de temps à débattre sur des détails techniques bien trop prématurés alors que nous avions encore des lacunes sur l'implémentation globale de notre projet !

De plus, ce projet a touché plusieurs d'entre nous, étant utilisateurs de logiciels de traitement d'image. Chacun avait à cœur d'implémenter certaines fonctionnalités qui selon son expérience des logiciels de dessin étaient indispensables, ce qui nous a mené à voir beaucoup trop grand. En effet,

nous avons proposé un cahier des charges vraiment conséquent et il nous a fallu un travail acharné pour parvenir à le tenir !

Nous avons également pu constater la difficulté à mettre d'accord un groupe de 6 personnes ! Notre groupe étant composé de plusieurs caractères de meneur, aux opinions divergentes, il a souvent fallu de longs débats pour trouver une solution satisfaisant tout le monde.
Malgré la nomination d'un chef de projet, nous avons pendant trop longtemps fonctionné en essayant d'intégrer l'avis de tout le monde à part égale et de satisfaire tout le monde, alors que nous aurions dû, dès le départ, donner plus de poids au chef de projet.
C'est ce que nous avons fait dès la moitié du projet environ et notre productivité s'en est vue grandement améliorée. Dès lors, le chef de projet était chargé d'écouter les opinions de chacun et de prendre les décisions finales. On s'est également mieux répartis les tâches. Le chef de projet était chargé de donner du travail à chacun au fur et à mesure qu'ils étaient désœuvrés et chaque membre est devenu le spécialiste d'un certain domaine. La coordination en a été grandement améliorée. Lorsqu'une question taraudait un membre du groupe, il venait la poser au chef de projet afin qu'il puisse y répondre au mieux ou renvoyer vers la personne la plus apte à répondre.

Une autre difficulté rencontrée est liée au manque de communication. À six sur un projet, la communication est un point crucial pour la réalisation du code. Nous en avions conscience et avons fait de notre mieux pour communiquer autant que possible. Seulement, pendant toute la première partie du projet, la communication était trop peu claire car il n'y avait pas de référent. Tout le monde devait être au courant de tout et chacun devait communiquer ses avancées à tout le monde ! Résultat, il nous est arrivé de travailler chacun de notre côté sans communiquer sur ce qui avait été fait ni comment cela avait été fait. Ce manque de communication a engendré des divergences au niveau de l'implémentation que nous avions élaboré au départ de notre projet et a posé de gros problèmes lors du regroupement des fonctionnalités.

En revanche, la communication est devenue très précise et efficace dès la deuxième moitié du projet. Dès lors, le chef de projet était le référent de tout le monde. Chaque membre devait le tenir au courant de ses avancées de sorte à ce que le chef de projet soit au courant de toutes les avancées. Ainsi, le chef de projet avait une vision d'ensemble de l'avancée réelle du projet et pouvait coordonner et organiser les efforts de chacun.

7 Tests

7.1 Techniques de test

Nous avions initialement prévu d'utiliser des tests unitaires *JUnit* afin de tester notre programme. Seulement, avec *JavaFX*, l'implémentation de tests *JUnit* s'est vite retrouvée compromise pour la plupart des fonctionnalités.
Nous avons donc décidé de procéder à des tests "en condition réelles" en lançant le programme et en testant des combinaisons de fonctionnalités à chaque ajout ou modification de code.
Il n'a pas été possible de tester tous les cas, étant donné le nombre de fonctionnalités et de combinaisons possibles, mais nous avons tout de même pu déceler et résoudre de très nombreux bugs.

De plus, nous avions fixé certains points à tester régulièrement tels que :

- La création d'un nouveau projet,
- L'ouverture d'un projet existant,

- La sauvegarde d'un projet,
- L'exportation d'un projet en *JPG* et en *PNG*,
- L'importation d'une image au format *JPG* ou *PNG* dans le projet,
- Pour chaque outil de la barre d'outils :
    - Vérifier que l'outil fonctionne si on le sélectionne en premier,
    - Vérifier qu'il est sélectionnable après l'utilisation de chaque autre outil,
    - Effectuer 3 *undo* et 3 *redo* (sauf pour la *pipette*),
    - Effectuer 3 *undo* puis utiliser l'outil,
    - Pour les outils qui demandent validation (*rogner* et *texte*), vérifier qu'un changement d'outil fonctionne avant et après validation,
    - Pour les outils qui demandent validation (*rogner* et *texte*), vérifier que les *undo/redo* fonctionnent avant et après validation,
    - Pour les outils possédant des paramètres (épaisseur, couleur, …), vérifier que ces paramètres sont modifiables et que le résultat varie comme souhaité,

- Tests sur les calques. Pour tous ces tests, vérifier le fonctionnement de l'*Undo/Redo* :
    - Ajout de calque avec le bouton **+** et via le menu *Calque*
    - Suppression de calque avec le bouton **-** et via le menu *Calque*
    - Déplacement de calque dans la pile de calques
    - Duplication d'un calque via le menu *Calque*
    - Fusion de deux calques via le bouton fusion de l'interface graphique et via le menu *Calque*
    - Aplatissement de tous les calques via le menu *Calque*
    - Redimensionnement d'un calque avec et sans les proportions maintenues
    - Redimensionnement d'un calque avec et sans l'ajustement du calque
    - Rotation d'un calque
    - Symétrie horizontale et verticale d'un calque
    - Changement d'opacité d'un calque

En plus de ces tests, nous avons chacun testé des variantes lors de l'implémentation de chaque outils et lors de la phase finale de tests.

Grâce à ces tests, nous avons réussi à isoler des bugs apparaissant de manière récurrente lors de la réalisation de certaines actions. La liste des bugs non résolus établie ci-dessous n'est donc pas exhaustive, mais couvre une majorité de ceux-ci.

7.2 Bugs restants

Description du bug	Comment il se produit	Solution envisagée
Problème d'arrondi avec l'utilisation du *Pot de peinture*	Le *Pot de peinture* ne remplit pas toujours parfaitement bien les bords du conteneur. Ceci est dû au fait que le *PixelReader* utilisé fonctionne avec des réels et non des entiers.	Réduire la taille de parcours du *Pot de peinture* mais cela le rendrait encore plus lent...
Problème avec l'historique en utilisant l'outil *Texte*.	Durant l'ajout d'un texte (pas encore validé), si l'on revient en arrière	Problème dû aux calques temporaires. C'est un problème complexe à

	dans l'historique, celui-ci ne fonctionne plus.	résoudre, qui nécessite probablement une réflexion sur la modélisation et l'implémentation des calques temporaires.
Autre problème avec l'historique en utilisant l'outil *Texte*.	Durant l'ajout d'un texte (pas encore validé), si l'on ajoute ou supprime un calque, ou si l'on déplace le calque sélectionné tout en haut de la liste, l'historique ne fonctionne plus correctement.	Problème dû aux calques temporaires. 2 solutions possibles : 1) Verrouiller les boutons situés en dessous de la liste des calques durant l'utilisation de l'outil *Texte*. 2) Même « solution » que pour le bug précédent.
Problème avec l'historique en utilisant l'outil de rognage.	Les deux bugs précédent peuvent aussi se reproduire avec l'outil de rognage.	Même solution qu'aux deux bugs précédents.
Problème lorsque l'on sélectionne une zone en dehors du calque à rogner.	Si on sélectionne un rectangle qui sort du calque, le rognage ne s'effectue pas correctement	Serait résolu si on ajoutait la possibilité au rognage d'agrandir le calque et pas que de le diminuer.
Problèmes avec l'épaisseur des outils *Pinceau* et *Gomme*.	L'épaisseur des traits verticaux et horizontaux des outils *Pinceau* et *Gomme* n'est pas la même que pour des traits diagonaux.	Changer l'implémentation des outils *Pinceau* et *Gomme*. (le bug ne se produit que sur certains ordinateurs)
Le dessin de l'espace de travail n'est pas centré et n'est pas aux bonnes dimensions lorsqu'on crée le projet alors que le programme n'est pas en plein écran.	Lors de la création d'un nouveau projet, l'espace de travail est centré dans la zone de dessin du logiciel. Si cette zone est plus petite que les dimensions du projet (programme lancé en mode fenêtré) le projet ne s'ouvre pas dans les bonnes dimensions.	Modifier la formule permettant de centrer l'espace de travail et faire en sorte de redessiner tout l'espace de travail, clip compris lorsque la fenêtre est redimensionnée.

8 Conclusion

8.1 Niveau du projet

Nous nous sommes rapidement rendu compte, durant la réalisation de notre projet, que nous avions vu grand dans notre cahier des charges et que nous allions avoir de la peine à le tenir. Au final, nous y sommes presque ! En effet, toutes les fonctionnalités obligatoires stipulées dans le cahier des charges ont bien été implémentées, à l'exception de l'outil *Zoom* que nous avons dû abandonner, faute de temps, après plus d'une semaine passée à tenter, sans succès, de le faire fonctionner ! Cependant, nous avons réalisé certaines tâches qui étaient facultatives dans le cahier des charges, tel que l'outil *Pipette* ou encore la gestion de raccourcis clavier.
Les fonctionnalités qui ont été réalisées ou non sont exposées plus clairement dans le fichier *BlaajjPaint-Fonctionnalite.xlsx* joint en annexe.

Globalement, nous avons terminé 24 des 25 tâches obligatoires et réalisé 5 tâches optionnelles. Voici les raisons qui nous ont poussé à réaliser des tâches optionnelles avant d'avoir terminé les tâches obligatoires :

- La pipette est un outil très pratique et souvent utilisé.
- La sélection de couleurs parmi un spectre complet de couleurs et la possibilité de sauvegarder des couleurs personnalisées étaient des fonctionnalités offertes par le *colorPicker* disponible par défaut avec *JavaFX*.
- Les raccourcis clavier ont été implémentés car ils nous ont été très pratiques pour la phase de test du projet. En effet, il est bien plus rapide d'utiliser le raccourci clavier pour, par exemple, créer un nouveau projet que de passer par le menu.
- Pour l'ajout de l'historique de modification, nous avons souhaité relever le challenge de le réaliser, car nous voulions exploiter le plein potentiel du *Undo/Redo*.

En ce qui concerne la qualité du code, nous pensons avoir une solution correcte et relativement bien factorisée. Ce n'était pas facile car certains composants avaient un comportement similaire mais pas suffisamment pour pouvoir tous bien les regrouper dans une grosse classe mère.

Le gros bémol de notre code est la complexité parfois démesurée de certaine classe alors que *JavaFX* propose une implémentation déjà toute faite et bien plus performante. C'est le cas, par exemple de notre implémentation de l'outil *Texte*, ou encore des *Formes*, qui sont complexes et entièrement "fait maison" alors qu'on aurait pu bien plus s'appuyer sur les fonctionnalités offertes par *JavaFX*.

Aussi, avons-nous peut être sous-exploité les concepts vus dans le cours de *MCR* (modèles conceptuels réutilisables). Il aurait pu être intéressant, par exemple d'implémenter le modèle *Composite*, mais il était compliqué d'ajouter de nouveaux modèles dans notre implémentation, d'autant plus qu'on a vu ce modèle alors que le projet était entamé depuis de nombreuses semaines. On a essayé d'approcher le modèle *Modèle-Vue-Contrôleur* (*MVC*), mais il s'avéra impossible de séparer clairement ces trois composants au sein de notre code.
En revanche, nous avons énormément utilisé le modèle *Singleton*, peut-être même un peu trop, mais il nous a été extrêmement utile pour l'organisation du code et sa factorisation.

Au niveau des performances de notre code, en revanche, nous ne sommes que moyennement satisfaits. En effet, le temps d'exécution de certaines actions est de loin pas optimal. Pour l'améliorer significativement, il faudrait mettre en place un système de threads s'exécutant en parallèle, ce qui était impensable dans le temps imparti.

Une grande satisfaction de notre code reste la sérialisation et la désérialisation de la sauvegarde. Ce fut probablement une des fonctionnalités la plus compliquée et laborieuse à implémenter. Cela est notamment dû à nos manques critiques de connaissances dans ce domaine. Néanmoins nous sommes fiers d'avoir réussi à enregistrer proprement notre programme sans atteindre des temps d'exécutions trop important ou des tailles de fichier astronomiques.

La réalisation de la fonction *Undo/Redo* a aussi nécessité d'intenses réflexions. Malgré le fait que nous avons été obligés de limiter la taille de l'historique à 100 éléments maximum, nous sommes contents d'avoir implémenté cette fonctionnalité. Nous avons dû ajouter cette limite car nous stockons, pour chaque action, une image de l'état précédent du calque. En fonction de la taille du calque, ces images peuvent prendre une place non négligeable en mémoire et il nous est arrivé de faire exploser la mémoire… C'est pourquoi, nous avons placé cette limite de 100 éléments maximum dans la liste des actions pouvant être annulées.

La représentation graphique de notre projet est très épurée et sobre, ce qui la rend plaisante à utiliser. Le résultat n'est pas d'un design époustouflant mais est suffisamment élégant et abouti pour avoir été réalisé par des personnes qui n'ont pas de formation dans le graphisme et qui, il y a quelques mois, n'avaient encore aucune connaissance en *JavaFX*. Néanmoins, il est possible que la qualité des images de certains boutons ne soit pas parfaite si l'utilisateur a une petite résolution d'image et un grand écran.

8.2 Fonctionnement du groupe

Dans l'ensemble nous avons eu une très bonne dynamique de groupe. Bons amis de base, nous avons su en profiter pour nous motiver mutuellement et parvenir à ce résultat.
Nous avons chacun su intégrer les points forts de nos personnalités dans ce travail afin de pouvoir être le plus productif possible. Il n'y a pas eu de grosse mésentente durant la réalisation et personne n'a été mis de côté. Toutes les idées ont été écoutées, prises en compte ou débattues, que ce soit au niveau de la modélisation comme de l'implémentation.
Le seul point négatif majeur dans le fonctionnement du groupe fût la communication. Comme décrit au point 6.2 Difficultés organisationnelles, il nous a fallu trop longtemps pour trouver une bonne organisation du groupe et instaurer une communication simple et efficace entre les membres.
Ces problèmes de communication ont apporté quelques tensions, négligeables, notamment lorsque des personnes ont modifié du code existant, engendrant des dysfonctionnements dans le code des autres collaborateurs.
En revanche, ces soucis nous ont fait prendre conscience, une fois pour toute, de l'importance capitale des commentaires dans le code afin de savoir ce qui a été fait, pourquoi, comment et par qui. Il nous est arrivé à plusieurs reprises de nous marcher sur les pieds, de refaire quelque chose qu'un autre avait déjà fait ailleurs etc, mais cela n'a jamais pu enlever la bonne humeur qui régnait constamment dans le groupe.

Nous nous sommes organisé plusieurs séances de travail intenses, souvent réparties sur 2 jours chez l'un d'entre nous. Ces grosses séances de travail en commun ont été extrêmement bénéfiques. Elles ont permis de donner de gros coups d'accélérateur au projet et nous ont permis de faire remonter les problèmes très rapidement, ce qui nous a sans doute évité de partir dans de mauvaises directions. Ces séances de travail ont également favorisé le partage de nos connaissances notamment concernant *JavaFX,* et ont fait économiser du temps à tout le monde. Il est toujours plus rapide de demander à son voisin que de chercher la réponse à notre question sur internet, et comme nous étions 6 dans le groupe, il était assez fréquent que l'un de nous ait la réponse !

Finalement, nous sommes tous très contents d'avoir réalisé ce projet et nous pensons vraiment que notre dynamique de groupe a aidé à la réalisation d'un projet dont nous sommes grandement satisfaits.

8.3 Si c'était à refaire

Durant ce projet nous avons rencontré de nombreux problèmes souvent dû à nos mauvaises connaissances des librairies utilisées. Nous nous sommes demandé comment nous implémenterons ce même projet, si, forts de l'expérience acquise au cours de ces derniers mois, il nous était demandé de le refaire.

8.3.1 À garder

Nous pensons que s'il nous était demandé de refaire le projet, une bonne partie de l'architecture serait refaite à l'identique. L'organisation *FXML - contrôleur* resterait probablement la même. Nous tenterions toutefois de plus et mieux utiliser les *FXML* de sorte à décharger et simplifier les codes des contrôleurs autant que possible.

De nombreuses choses que nous avons implémentées sont bonnes et seraient gardée, mais la plupart peuvent être améliorées.

8.3.2 À améliorer

Tout d'abord, notre connaissance accrue dans la librairie *JavaFX*, nous permettrait de bien mieux implémenter les outils. Nous pensons notamment aux outils *formes* et *texte* qui n'ont pas du tout utilisé les possibilités offertes par *JavaFX* !
Nous chercherions également à améliorer les performances globales des outils, en particulier celles du *Pot de peinture*, particulièrement lent pour remplir des zones importantes.
Enfin, nous utiliserions beaucoup moins les calques. Nous bannirions sans doute les calques temporaires qui ont été la cause de tant de bugs !
En plus de ces changements majeurs, nous apporterions quelques améliorations mineures aux fonctionnalités présentes. Nous avons regroupé ces améliorations possibles dans le tableau ci-dessous :

Description	Pourquoi c'est un problème	Comment le résoudre
Il n'est pas possible de ne rien avoir dans les textFields définissant la taille des outils crayon, gomme et forme, ainsi que dans le textFields "Opacité" des calques.	Pas pratique lorsque l'on veut passer de la taille 2 à 3 par exemple.	Modifier la vérification de la saisie.
Lorsque l'on quitte le programme avec un projet ouvert, aucune demande de sauvegarde ou de confirmation n'est faite.	Risque de perte du projet actuel en cas de mauvais manipulation.	Fonctionnalité à créer.

Lorsque l'on utilise le pinceau ou la gomme avec une épaisseurs conséquente, on perd de la précision avec la souris.	Il est difficile d'être précis avec la souris puisque la souris n'indique que le centre du trait qui sera dessiné.	Définir un curseur de différentes tailles dont la taille correspondrait à l'épaisseur de l'outil.
Difficile de déterminer le résultat et la position des ellipses créées.	On ne sait pas où cliquer pour faire un cercle autour d'un point précis. Impossible de savoir ou l'ellipse va être positionnée après avoir définis sa taille.	Changer la façon de dessiner les ellipses en demandant une validation comme pour l'outil texte. Déplacer le calque de la forme après sa création.

D'autres améliorations possibles seraient d'utiliser la programmation concurrente afin de pouvoir optimiser les temps d'exécution de divers algorithmes en multi-threadant les calculs. On pourrait également mieux factoriser le code en utilisant des modèles conceptuels réutilisables comme l'*abstract factory* pour l'outil *formes* ou encore le modèle *Composite*.
Au niveau utilisateur, cela pourrait être intéressant d'ajouter une barre de chargement pour chacune des exécutions qui peuvent prendre un peu de temps.
Une autre fonctionnalité utile serait de faire un gestionnaire d'erreur afin, par exemple, de faire apparaître une fenêtre d'erreur quand l'utilisateur ouvre un fichier corrompu.
Bref, la liste des améliorations possibles est infinie !

8.4 Avis personnels

Cette rubrique contient les avis personnels de tous les membres du groupe qui ont participé à l'élaboration du projet.

8.4.1 Allemand Adrien

La réalisation de *BlaajjPaint* a été une véritable aventure chevaleresque. Il y a eu l'engouement du départ alors que nous n'étions encore que des rêveurs, où dans un brainstorming aussi désordonné que remplis de bonne volonté nous avons tous jetés nos idées et nos souhaits, peut-être un peu trop même. Puis, il y a eu l'entraînement et la préparation (principalement à *JavaFX*) qui nous a semblé durer une éternité et qui nous a laissé un peu craintif face à la suite. S'en est suivit le développement en lui-même qui fut comparable à un champ de bataille où nous avons appris que le plus important dans un projet, c'est non le code en lui-même mais bien la communication entre les différents intervenants qui évite bien des sacrifices et des détours inutiles. Pour finir, nous avons eu l'occasion de reparcourir notre projet en entier et d'en peindre les moindres détails dans ce rapport, ce qui m'a empli d'une certaine fierté en repassant sur les nombreuses fonctionnalités que nous avons réussi à compléter et en repensant à toutes les difficultés que nous avons surmontées.

Si je devais recommencer dans les mêmes conditions, je le ferais avec plaisir mais en prenant un chemin légèrement différent à certains endroits. Premièrement j'utiliserais une méthode de gestion de projet du type *AGILE* car la planification initiale s'est avérée relativement inefficace et de ce fait la direction générale du projet fastidieuse à coordonner. Ensuite, je n'utiliserais pas de *Framework* tel que *JavaFX*, avec le recul, je pense que nous aurions été beaucoup plus efficace et aurions pu créer un code bien mieux factorisé si nous avions fonctionné sans toutes ces classes "toutes faites" que nous ne comprenions pas bien au départ. Pourquoi pas travailler en C++ avec des structures brutes

mais qu'au moins nous aurions codé à 100% et dont nous aurions pu planifier les moindres détails lors de leur conception.

En revanche, je tiens à faire l'éloge de notre équipe car l'ambiance de travail a été décontractée durant tout le projet. Chacun a su trouver sa place dans le groupe, à la fois en exploitant ses forces et en comptant sur les autres pour le soutenir dans ses faiblesses. Lorsqu'il y avait divergence dans les avis, celles-ci étaient réglées calmement, en discutant des points de vue de chacun et en prenant en considération les avantages et les défauts des solutions proposées.

En conclusion, même si nous n'avons malheureusement pas (encore) réussi à développer le logiciel qui va remplacer Photoshop, ce projet a été très instructif sur les mécaniques du travail en équipe qui, je suis sûr, seront parmi les plus utiles pour notre avenir professionnel.

8.4.2 Châtillon Jérémie

J'ai beaucoup aimé réaliser ce projet. Je suis globalement très content de celui-ci même si son élaboration fut laborieuse. Je suis surpris en bien du résultat obtenu. Je ne pensais pas pouvoir aboutir à un projet tel qu'il est. Le point négatif de ce projet est que, à mon avis, nous avons voulu faire trop de fonctionnalités. Cela a impliqué que nous avons dû précipiter l'implémentation de certaines choses sans avoir assez de connaissances dans le domaine. Nous aurions dû nous limiter à moins afin de pouvoir mieux factoriser notre code.
Cependant, je suis quand même fier de ce que nous avons réalisé, et content d'avoir pu partager cette expérience avec toutes les personnes présentes dans le groupe.

8.4.3 Loyse Krug

La réalisation de ce projet m'a beaucoup appris, tant au niveau du code à écrire qu'au niveau de la dynamique de groupe. Du point de vue implémentation, j'ai rencontré certaines difficultés avec *JavaFX* et me rend maintenant compte de l'ampleur du travail nécessaire pour maîtriser correctement cette bibliothèque. En ce qui concerne le travail de groupe, j'ai pu parfois constater les difficultés à nous mettre d'accord et à communiquer notre travail de manière général. Malgré cela, j'ai beaucoup apprécié travailler avec cette équipe qui, même si elle a eu quelques soucis de temps en temps, a toujours essayé de prendre en compte l'avis de tout le monde. Tous les membre de notre groupe se sont impliqué activement dans le projet dans la mesure de ses moyens. Avec du recul, je pense que nous avons choisi un sujet très ambitieux et que nous aurions peut-être dû nous fixer moins de fonctionnalités. Si ce projet était à refaire, je m'y lancerais sans hésiter, les idées plus claires et la soif d'en découvrir davantage plus présente que jamais.

8.4.4 Rochat Antoine

Ce projet fut personnellement une expérience très enrichissante. C'était la première fois qu'il m'était donné de travailler sur un projet avec un groupe aussi nombreux. J'ai pu me rendre compte de toutes les difficultés qu'un projet de groupe impliquait, notamment au niveau de la communication et de la répartition des tâches. Néanmoins, je suis reconnaissant d'avoir fait partie de ce groupe où tout le monde s'est montré disponible et impliqué dans le projet. Une autre difficulté que j'ai personnellement rencontrée était d'apprendre une nouvelle technologie, *JavaFX*, qui m'était alors inconnue. J'ai perdu de précieuses heures à chercher une solution qui s'est pour finir avérée très simple car je n'avais clairement pas les connaissances nécessaires.

Ces difficultés rencontrées ont néanmoins été très enrichissantes, et je suis sûr que ce projet me sera très utile pour le futur. Je me sens désormais plus en confiance à l'idée de me confronter à un nouveau projet de groupe.

8.4.5 Schopfer Benoît

J'ai personnellement adoré réaliser ce projet et en être le chef de projet a été pour moi un honneur et une expérience très enrichissante !
L'entente au sein du groupe a été au beau fixe durant toute la réalisation du projet. Il n'y a jamais eu de problèmes, ou de conflits plus graves que de savoir quelle implémentation était la meilleure ou quel paquet de chips nous allions ouvrir !
Nos séances de travail ont toujours été joyeuses, animées, joviales tout en restant extrêmement efficaces et productives !

J'ai également trouvé très intéressant de se lancer dans un projet de cette envergure. C'est la première fois que je réalise un projet conséquent de bout à bout, et je suis réellement content du résultat, même si une ou deux semaines de plus auraient permis de rendre un programme presque parfait, je trouve le résultat actuel tout à fait respectable. J'ai cru un temps qu'on s'était placé la barre un peu trop haute, avec un cahier des charges pour le moins conséquent et de très nombreuses fonctionnalités à implémenter, mais l'implication de chaque membre du groupe a rapidement effacé ces doutes. Nous sommes parvenus à surmonter obstacle après obstacle et à remplir quasi entièrement le cahier des charges, et j'en suis pas peu fier ! Malheureusement, on s'est heurté à un os avec le zoom, qui a fait resurgir toutes les erreurs d'implémentations que nous avions commises et que nous ne sommes finalement pas parvenus à implémenter. Mais à part cet outil, le groupe complet à fait un travail que je trouve remarquable.

Pour ma part, je me suis extrêmement impliqué dans ce projet, peut-être un peu trop au regard du nombre d'heure que j'y ai passé… Mais ce projet m'a vraiment plu et je le faisais avec plaisir et envie. Faut dire qu'entre déboguer BlaajjPaint ou faire un labo de SER pénible à vouloir se pendre, le choix était facile… Cela explique sans doute mon nombre d'heure important.
En plus du plaisir à coder ce projet, j'ai trouvé très intéressant et instructif d'être le chef de projet. J'aime beaucoup avoir des responsabilités et je voulais être chef de projet car c'est une voie qui m'intéressait potentiellement pour mon avenir. Je me questionnais notamment à propos des avantages et inconvénients de cette responsabilité. S'il y a une chose que je sais, c'est que je veux pouvoir écrire du code et réaliser des programmes concrets lors de mon parcours professionnel. Je me demandais donc à quel point un chef de projet avait encore la possibilité de coder ou s'il se retrouvait à passer l'entièreté de son temps à coordonner les équipes et faire de l'administratif.
Être le chef de ce projet m'a permis de me rendre compte, toute proportion gardée, des avantages et inconvénients de cette responsabilité. J'aime beaucoup avoir la vision d'ensemble du projet qu'offre cette position et apprécie être au courant de tout ce qui concerne le projet, connaître les problèmes et avancées de chacun et pouvoir donner un coup de main lorsque nécessaire. En revanche, même à la petite échelle d'un groupe de 6, les séances de travail en groupe étaient pour moi très peu productives. En effet, j'étais constamment interrompu par mes camarades qui venaient me poser telle ou telle question, me demander une nouvelle tâche ou me demander comment implémenter telle ou telle fonctionnalité, m'empêchant bien souvent d'avancer ma propre tâche.
Après cette expérience, je pense qu'être chef de projet au niveau professionnel doit tôt ou tard rimer avec cesser de coder et en venir à ne faire plus que de l'administratif et de la coordination. Je ne banni pas pour autant cette possibilité, mais ce projet m'a offert un premier aperçu qui est le bienvenu.

8.4.6 Smith James

Ce projet fut une sacrée expérience que ce soit niveau développement, organisation ou travail d'équipe. L'utilisation de *JavaFX* qui était pour nous tous, une technologie inconnue au début du projet, nous a posé de nombreuses difficultés. Il est très compliqué de planifier la réalisation de tâches et d'estimer leurs durées sans connaître les outils à disposition. Le fait de travailler sur un projet où tout le monde travaille sur les mêmes fichiers fût complexe mais intéressant. Je suis très content d'avoir travaillé avec toutes les personnes qui ont participé au projet. Il fût facile de se voir pour des week-ends, de communiquer et d'apprendre entre nous.
Malgré les difficultés rencontrées, je suis très content du résultat final du projet et je suis fière d'avoir fait partie de cette équipe.

9 Webographie

Sérialisation :

- https://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html
- https://www.tutorialspoint.com/java/java_serialization.htm
- http://x-stream.github.io/

Documentation oracle sur les canvas :

- https://docs.oracle.com/javafx/2/canvas/jfxpub-canvas.htm

Récupération d'expérience d'autres utilisateurs de *JavaFX* :

- https://www.youtube.com/user/thenewboston
- https://openclassrooms.com/courses/les-applications-web-avec-javafx/la-scene-graphique-de-javafx
- https://docs.oracle.com/javafx/2/canvas/jfxpub-canvas.htm
- www.stackoverflow.com
- www.github.com

Fonction Historique (Undo/Redo) :

- https://en.wikipedia.org/wiki/Command_pattern
- https://sourcemaking.com/design_patterns/command
- https://alvinalexander.com/java/java-command-design-pattern-in-java-examples
- https://www.javaworld.com/article/2076698/core-java/add-an-undo-redo-function-to-your-java-apps-with-swing.html

10 Table des illustrations

11 Annexes

Les fichiers fournis en annexes sont les suivants :

- *Cahier des charges.pdf* --> le cahier des charges rédigé au tout début du projet
- *AnalyseDeRépartitionDuTemps.pdf* --> analyse planification début / fin du projet
- *Gantt.pdf* --> diagramme de Gantt réalisé au tout début du projet
- *Répartition_heures.xlsx* --> tableau de répartition des heures réalisé au tout début du projet.
- *manuelUtilisateur.pdf* --> le manuel d'utilisateur
- *dossier Journaux de travail --> dossier contenant le journal de travail de tous les membres*
- *dossier BlaajjPaint* --> l'ensemble des sources du projet
- *BlaajjPaint_UML.png* --> diagramme UML du projet