RAPPORT PROJET-GUI

Amandine Bouguessa - Eloïse Zaghrini - Armelle Coutaux
2020-2021

INTRODUCTION

Ce rapport présente notre travail pour le projet-GUI du cours de Software Engineering de 2e année au département informatique des Mines Nancy.

Nous avons implémenté une application de traitement d'image en Java permettant de :

- Charger/Sauvegarder une image
- Rogner une image
- Mettre à l'échelle une image
- Redimensionner l'image par seam carving

Extensions réalisées :

- Undo/Redo
- Interface de dessin
- Choix de l'énergie
- Agrandissement de l'image par seam carving

Temps passé à la réalisation du projet :

- Eloïse : 35 heures
- Armelle : 25 heures
- Amandine : 35 heures

Interface

Nous avons choisi de disposer les différents éléments dans un *BorderPane* avec les opérations possibles de traitement d'image dans un menu à l'aide d'un *MenuBar* en haut du *BorderPane*. L'image est affichée à gauche et les actions propres à chaque opération de traitement à droite. Nous avons choisi d'utiliser un *StackPane* pour disposer les différentes options de traitement d'image de façon à les superposer et à ne rendre visible que l'option correspondant au traitement choisi (dans la barre de menu).

Design Pattern

Un controller et son fichier .fxml correspondant ont été créés pour chaque action différentes. Nous avons fait en sorte que le *Controller* (général) soit unique et utilisé par tous les autres controllers pour accéder à l'image. Pour cela, nous avons fait en sorte que le *Controller* ne puisse pas être instancié deux fois en levant une exception si une version du *Controller* existe déjà et en accédant à celui-ci à l'aide d'une méthode *getController()*.

Chargement/Sauvegarde de l'image

On utilise *FileChooser* pour charger et sauvegarder notre image, puis cette image est lue dans une *BufferedImage* afin de pouvoir être modifiée et affichée à l'aide d'une *ImageView*. Une exception est levée si le fichier choisi n'est pas une image. La sauvegarde se fait au format .png.

Recadrage de l'image

Nous avons tenté de recadrer l'image à l'aide d'une fonction déjà existante, mais cela créait trop de problèmes. Nous avons finalement opté pour une méthode "à la main" en copiant les pixels à garder.

Nous avons choisi de représenter cela dans l'interface par 4 sliders qui permettent de rogner à gauche, droite, haut ou bas. Pour cela, les sliders gauche et haut sont initialisés à 0 et droite et bas aux tailles de l'image. Pour rogner à droite on décrémente le slider et pour rogner à gauche, on l'incrémente du nombre de pixels à rogner. Une exception est levée si le slider gauche/haut dépasse le slider droit/bas (demande de taille négative).

Le choix a été fait de ne pas permettre un rognage dynamique mais au clique d'un bouton pour éviter d'augmenter la valeur des sliders ou une erreur de la part de l'utilisateur. De plus, les sliders sont ajustés à la taille de l'image à chaque modification pour ne pas permettre de vouloir l'augmenter.

Mise à l'échelle de l'image

Pour la remise à l'échelle de l'image, nous avons utilisé des bibliothèques déjà présentes dans java, notamment la classe *Graphics2D*. Lorsque l'utilisateur modifie l'échelle de l'image à l'aide des sliders (puis en appuyant sur "Traitement de l'image"), une nouvelle image de type *BufferedImage* est créée avec les nouvelles dimensions souhaitées. On lui ajoute ensuite l'image mise à l'échelle souhaitée à l'aide de méthodes de la classe *Graphics2D*.

Le choix a été fait d'actualiser la valeur maximale des sliders à la taille de l'image à chaque modification pour ne pas perdre en qualité en agrandissant l'image (même si la fonction marchait si le choix était d'agrandir l'image).

Seam Carving

1. Implémentation

Deux classes sont utilisées pour réaliser le Seam Carving :

- Une classe *Couture* qui représente une couture de pixel de l'image
- Une classe *SeamCarver* qui réalise le Seam Carving en prenant une BufferedImage

Le tableau d'énergie des pixels est calculé en faisant un gradient sur chaque couleur R G B de chaque pixel de l'image. Les pixels de bord sont eux initialisés au maximum. 2 méthodes permettent de trouver la couture d'énergie minimale, en calculant cette énergie Dynamiquement puis en retrouvant les pixels parcourus par Backtracking. La méthode *CarveImage* retire une à une les coutures d'énergie minimale pour atteindre la hauteur et largeur souhaitée par l'utilisateur. Le tableau d'énergie de l'image est recalculé à chaque retrait de couture, ce qui rend l'opération plus longue mais permet d'avoir un meilleur résultat final.

2. Choix de l'énergie

Nous avons implémenté deux manières différentes de calculer l'énergie, en norme 1 et en norme 2, au choix de l'utilisateur dans l'interface qui a le choix entre deux *RadioButton*. En pratique, les deux normes donnent un résultat assez similaire.

3. Agrandissement de l'image

Nous avons rajouté une classe *SeamAdder* qui permet d'augmenter la taille de l'image par un procédé similaire au Seam Carving. On commence par retirer le nombre de coutures horizontales ou verticales que l'on souhaite ajouter en Seam Carving sur une copie de l'Image. Ces coutures sont stockées dans un *ArrayList* de coutures et on les rajoute ensuite à l'image dans l'ordre pour retrouver les pixels qui ont été retirés. Ces mêmes pixels sont ensuite tous dédoublés pour créer l'image agrandie. Il faut trouver toutes les n coutures à ajouter d'un seul

coup cette fois, car dédoubler au fur et à mesure les coutures d'énergie minimale ne ferait que dédoubler n fois la même couture.

4. Choix de l'ordre optimal en redimensionnement bidirectionnel

Nous n'avons pas implémenter un choix de l'ordre optimal en backtracking par faute de temps et également par peur que le processus déjà long du Seam Carving le devienne encore plus. De plus, la littérature montre que les différences entre cette méthode et celle que nous avons implémentée ne sont pas très significatives. Cependant, pour ne pas choisir arbitrairement de retirer des coutures horizontales ou Verticales, nous comparons à chaque retrait possible de coutures horizontales ou verticales les énergies des deux et retirons celle d'énergie minimale.

Interface de dessin

Nous avons choisi d'utiliser un canva qui s'adapte à l'image et l'incorpore pour pouvoir dessiner dessus et l'enregistrer et pas d'écrire directement sur les pixels car le lien observer/listener avec la souris se faisait mieux dans le premier cas. Nous utilisons des listeners pour suivre les mouvements de la souris et tracer une ligne tant que la souris est maintenue appuyée. Le TD1 a été réutilisé pour que l'utilisateur puisse choisir la couleur avec laquelle dessiner sur l'image. Et nous avons rajouté un slider pour choisir l'épaisseur du pinceau.

Undo/Redo

Nous avons implémenté l'option Undo/Redo à l'aide d'une liste chaînée d'images de type *BufferedImage* stockée dans une nouvelle classe *ImageChainee*. Cette classe permet de rajouter deux méthodes primordiales à une BufferedImage : *getNext()* et *getPrev()*, qui lie chaque image avec une image suivante (redo) et précédente (undo). Lorsqu'une nouvelle opération sur une image est réalisée, elle est ajoutée "au début" de la liste chaînée. Le bouton "Undo" permet donc d'afficher les images précédemment affichées, dans l'ordre inverse d'apparition. De même, le bouton Redo permet de revenir à l'image suivante. Cela peut poser des problèmes de mémoire si le nombre d'images stockées est trop important, mais une image ne prend pas énormément de place donc dans le cadre de cette application cela paraît suffisant.

Limites

Les images trop grandes empêchent de bien voir les options de traitement. Les images sans fonds posent également des problèmes, avec le *Seam Carving* et le *Undo Redo* principalement. Le *Seam Carving* a un temps de traitement qui peut être long en recalculant à chaque fois toutes les énergies. Il serait possible de recalculer les énergies seulement autours de la couture pour un compromis entre qualité et efficacité. Dans le menu, il faut également appuyer exactement sur le texte du *label*, sinon le click n'est pas pris en compte.