DRAW 7

Projet long technologie objet
Rapport général

Équipe 4

Mathieu Malak
Matthieu Trichard
Ethan Boswell
Nathan Albarede
Othmane Elhour
Benjamin Schlögel

Table des matières

I / Récapitulatif du projet

L'objectif était de mettre en place une application permettant de dessiner avec notre souris sur une fenêtre avec la possibilité d'enregistrer nos dessins. Nous voulons que notre application puisse nous permettre de faire des traits continus, de changer la couleur du pointeur, de remplir des espaces vides d'une certaine couleur, de revenir en arrière, enregistrer notre projet. Nous avons donc une application de dessin rudimentaire, mais fonctionnelle qui permet de prendre des notes, faire des schémas rapides ou encore faire office de tableau blanc pour un cours.

Il manque cependant des fonctionnalités importantes que nous voulions implémenter, comme les calques ou les formes dynamiques.

II / Détails des fonctionnalités avec leurs fonctionnements

1 /Le trait continu

Le fonctionnement du trait continu s'appuie essentiellement sur 2 classes : JCanvas et Pointeur. Le rôle de JCanvas est de créer le tableau blanc sur lequel l'utilisateur peut dessiner. Il possède deux attributs tailleX et tailleY qui permettent à l'utilisateur de modifier sa taille. La classe Pointeur hérite de MouseInputAdapter. Les attributs couleur et forme permettent à l'utilisateur de modifier ses propriétés à l'aide de boutons. Le dessin du trait continu s'effectue grâce à la méthode mouseDragged qui récupère la position du MouseEvent grâce à la méthode getPoint, définit la position de la forme à dessiner grâce à la méthode form.setPos (définie dans la classe FormDrawable). On appelle enfin la méthode draw définie dans RectangleDrawable ou CercleDrawable selon la forme choisie, qui se charge d'afficher la forme à l'écran. Cette méthode draw prend en paramètre l'objet Graphics associé au JCanvas, donc le pointeur a besoin d'avoir comme attribut le JCanvas qu'il pointe.

2 /La pipette

Le bouton pipette permet de récupérer la couleur du pixel à l'emplacement actuel de la souris après avoir cliqué dessus. De ce fait, la classe BoutonPipette hérite de la classe parente MouseInputAdapter afin de pouvoir utiliser sa méthode mousePressed(MouseEvent e) qui va se lancer après avoir effectué une pression sur la souris. Dans cette méthode, la classe Robot a été importée pour récupérer la couleur du pixel en appelant sa méthode getPixelColor(int x, int y). De plus, une instance de la classe Toolkit a été créée pour accéder à différentes fonctionnalités du système (ici la taille de l'écran de l'ordinateur). La levée de l'exception AWTException dans cette méthode permet de gérer d'éventuelles erreurs sur la classe Robot, puisque cette dernière permet d'interagir avec les événements du système d'exploitation, et pour des raisons de sécurité, certaines plateformes peuvent restreindre l'utilisation de la classe Robot.

3 /Le remplissage

La classe BoutonRemplissage permet de remplir une zone de couleur uniforme par la couleur actuelle du pointeur. Il est alors nécessaire de récupérer la couleur de la zone cible, ce qui revient à implanter la méthode mousePressed(MouseEvent e) comme la classe BoutonPipette, en ajoutant dans cette méthode un appel à la méthode remplissageZone(int x, int y). Cette méthode va remplir le pixel actuel, puis va s'appeler récursivement en quatre temps pour remplir les quatre pixels voisins (non diagonaux), et ainsi modifier la couleur de la zone cible. Par conséquent, quatre tâches leftTask, rightTask, aboveTask et belowTask ont été créées pour être exécutées en parallèle à l'aide d'instances de la classe Thread. L'exception InterruptedException est levée pour gérer un éventuel appel infini de la méthode remplissageZone avec la récursivité. L'exécution de cette classe est cependant beaucoup trop lente.

4 /Le retour arrière

Pour réaliser cette implémentation, nous avons utilisé une approche basée sur la sauvegarde de l'état de l'application à chaque action de dessin. Lorsque l'utilisateur effectue un tracé ou ajoute une forme, les coordonnées des points ainsi que les informations nécessaires pour représenter la forme

sont enregistrées dans une structure de données appropriée appelée mémoire. L'application utilise les informations de l'action enregistrée pour restaurer l'état précédent, en redessinant toutes les actions précédentes sauf la dernière.

5 /Le changement de couleur

La méthode setColor s'occupe du changement de couleur du côté du pointeur. Cette méthode change la valeur de l'attribut couleur, et modifie la forme du pointeur pour qu'elle prenne la bonne couleur (une manière d'éviter de créer un nouvel objet FormDrawable à chaque changement de couleur pourrait être de mettre la couleur actuelle du pointeur en argument de la méthode draw).

6 /Le changement de forme de tracé

Deux classes CercleDrawable et RectangleDrawable s'occupent de gérer la forme du tracé lorsque l'utilisateur clique sur le canvas. Au niveau de l'interface graphique, nous avons ajouté dans un menu déroulant deux boutons qui permettent à l'utilisateur de changer de forme. Lorsque cliqués, ces boutons appellent la méthode setForme de la classe Pointeur qui prend en paramètre la forme correspondante.

7 /Enregistrement d'un projet

La classe BoutonEnregistrer utilise seulement la méthode actionPerformed(ActionEvent e), qui va effectuer une capture d'écran du canvas actuel en créant une instance 'image' de la classe BufferedImage et en utilisant la méthode createScreenCapture(Rectangle canvasRect) de la classe Robot. Ensuite, on affiche une fenêtre de saisie à l'utilisateur pour qu'il choisisse le nom de cette image qui est ensuite écrite dans le nouveau fichier .png du répertoire courant à l'aide de la méthode ImageIO.write(image,"png",file). La méthode actionPerformed(ActionEvent e) peut alors lever deux exceptions pour gérer les erreurs possibles liées à l'utilisation de la classe Robot (voir 2 /La Pipette) et à l'utilisation de la classe

ImageIO si des erreurs se produisent lors de l'écriture de l'image dans le fichier, comme des problèmes de permission d'écriture ou des erreurs de disque.

8 /Ouverture d’un projet

Après avoir sélectionné ce bouton, une fenêtre de saisie apparaît pour demander d’entrer le chemin d’accès au fichier .png souhaité, cette image est ensuite convertie en canevas avec la méthode createCanvasWithImage(File file), puis une nouvelle fenêtre est créée et affichée avec ce canevas. La nouvelle classe FenetreOuvrir a donc été créée pour éviter une boucle infinie (car la classe Ouvrir appelle la classe Fenetre et réciproquement), l’appel à la classe Ouvrir a été supprimé dans cette classe. La méthode createCanvasWithImage(File file) crée une instance de la classe BufferedImage à l’aide de la méthode ImageIO.read(file) qui permet de lire une image à partir d’un fichier. Un canvas de mêmes dimensions que l’image est ensuite créée, suivi d’une instance de JLabel créée à partir de l’icône de l’image souhaitée, qui va permettre d’afficher l’image sur le canvas. L’exception IOException peut être levée si le fichier saisi n’existe pas, s’il y a un problème de permission de lecture ou des erreurs de disque.

9 /Palette de couleur

Pour créer la palette de couleur, on crée dans le constructeur de la classe Fenêtre (le premier, Fenêtre(JCanvas jc, Memory Memory)) une liste listeCouleurs qui contient toutes les couleurs de base du module java.awt.Color. Cela permet de créer plusieurs boutons en itérant sur cette liste avec un foreach. Cela se fait en deux étapes : on crée un JButton avec une icône dont le nom est celui de sa couleur associée (par exemple, si la couleur actuelle est Color.RED, on crée un bouton avec l’icône nommée java.awt.Color[r=255,g=0,b=0]) et ensuite on associe à ce bouton un ActionListener nommé ActionCouleur(c) avec c la couleur courante, qui se charge uniquement d’appeler la méthode setColor avec la couleur c passée en paramètre de cette classe. Cette méthode possède plusieurs avantages : elle permet d’ajouter rapidement un grand nombre de boutons en peu de lignes de codes, et l’utilisation de la classe ActionCouleur permet de

s’affranchir de créer un grand nombre de classes qui font la même chose à une couleur près.

10 /Personnalisation des couleurs

Nous avons pour cela créé un nouveau bouton ouvrant une fenêtre pop-up qui offre le choix de saisir la couleur choisie en RVB avec un aperçu de la couleur. Cette fenêtre a été implémenter de manière à ne pas pouvoir être fermé sans avoir fait de choix pour pouvoir maîtriser les changements de couleur sans risque de bug. Si l’utilisateur clique sur “Annuler” alors on reste sur la couleur d’origine et s'il clique sur “Ok” on change la couleur courante et on met à jour l’aperçu de la couleur en haut à droite de la fenêtre.

III / Points qui auraient pu être améliorés

Notre projet n’est pas parfait, il y a de nombreux points pouvant être améliorés et des fonctionnalités qui n’ont pas été mises en place.

Dans un premier temps, la mise en place des calques n’a pas pu être faite par manque de temps car d'autres fonctionnalités plus importantes ont été priorisées.

De plus, l’action de remplissage de zone est beaucoup trop lente et les conditions de sortie sont alors à revoir pour améliorer les performances car la fonctionnalité n’est pas utilisable en l’état.

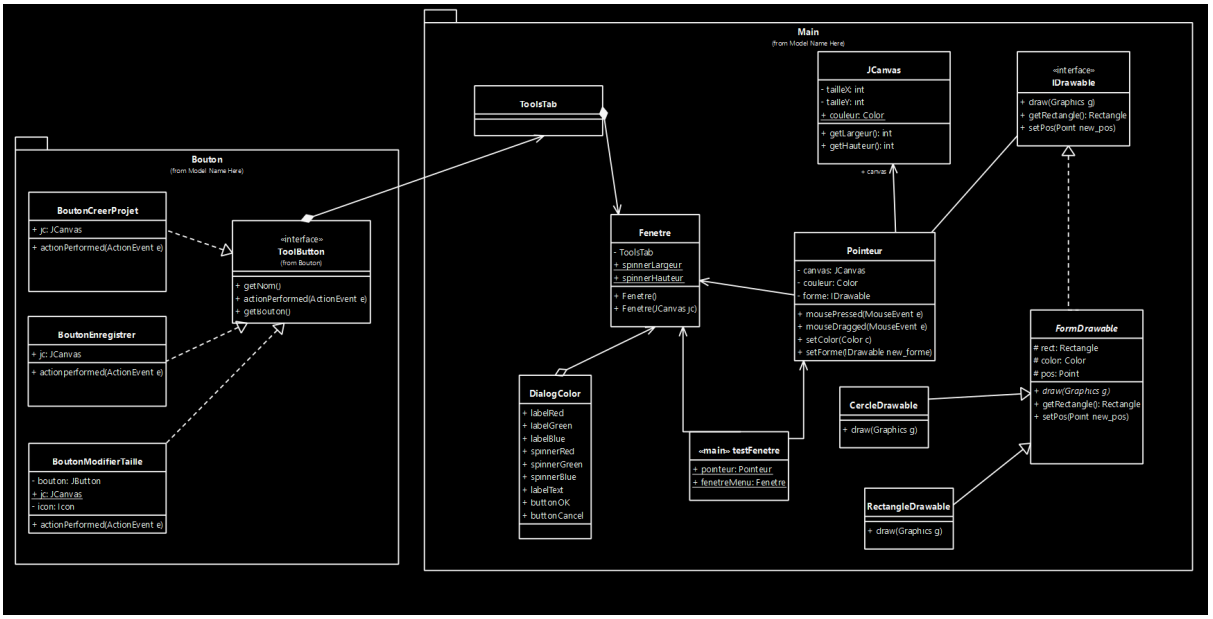
Encore, l’ouverture d’une image l’a convertie en canvas pour pouvoir dessiner dessus, mais l’utilisation des boutons de la barre d’outils n’est alors pas possible car il manque des Actionlistener au niveau des boutons car ces derniers sont cliquables, mais ne sont pas actifs, c'est-à-dire que le Actionperformed ne se lance pas. Nous n’avons pas réussi à régler ce problème.

En outre, la fonctionnalité de retour en arrière ne fonctionne pas lorsqu’on utilise le changement de couleur RVB.

Enfin, le bouton ModifierTaille ne fonctionne pas entièrement, il crée un autre canvas de la taille choisie au lieu de modifier la taille du canvas actuel. Cela peut être utile, mais ce n'était pas la fonctionnalité souhaitée au départ. Lorsque nous avons essayé de l'implémenter, le problème que nous avions

rencontré était que le dessin disparaissait lorsqu'on changeait la taille du canvas.

IV / Diagramme UML



On lance le programme depuis la classe testFenetre. Elle dispose de deux attributs, la fenetreMenu qui représente la Fenêtre de départ à afficher, et le pointeur qui représente le pointeur que l'utilisateur utilisera pour dessiner jusqu'à ce qu'il quitte l'application. Il y a en tout 3 fenêtres différentes, celle correspondant au menu, celle où l'on dessine et celle où l'on peut personnaliser la couleur du trait. Les deux premières étant des fenêtres classiques et la troisième étant une pop-up.

La partie droite du diagramme constitue la structure générale du pointeur, tandis que la partie gauche du diagramme est plutôt liée à la gestion de l'interface graphique et des boutons.

Nous avons fait le choix de réunir tous les boutons dans ToolTab pour simplifier et généraliser l'architecture, cela nous a permis d'ajouter de nouveaux boutons facilement.

La classe Pointeur quant à elle nous permet de réunir dans une même classe tous les attributs du futur trait. En effet il y a plusieurs aspects à prendre en compte, la couleur, la taille, la forme etc… Il y a ensuite les différentes formes de trait possible, nous avons implémenté les formes Cercle et Rectangle mais il aurait été facile d'en ajouter avec notre architecture versatile.

V / Méthodes agiles

Pour les méthodes agiles, nous nous sommes appuyés sur deux applications, Trello et Discord.

Trello nous a permis d'organiser notre travail en Story, User Story et en tâches. Cette application nous a aussi permis d'organiser notre travail en fonction des priorités de chacune des features. Cela nous a permis de bien visualiser les tâches à faire en priorité pour obtenir une application de plus en plus fonctionnelle. Elle nous a aussi permis de visualiser l'état d'avancement des différentes tâches, pour savoir si la personne en charge de la tâche a besoin de soutien. Trello fournit une visualisation en calendrier de nos tâches et nous avons utilisé des labels de couleurs différentes pour représenter l'avancement des tâches. Cela nous a permis d'obtenir un outil pratique et agréable à utiliser. De plus que Trello peut notifier les participants qu'ils leur restent des tâches à faire pour une certaine date.

Discord nous a servi pour la communication orale et textuelle entre les membres du groupe. Nous avons créé un serveur pour le projet avec différents canaux de discussion pour chaque partie du projet comme l'interface graphique ou les outils. Cela a permis de séparer les discussions et de rapidement retrouver des informations. De plus, discord permet de créer des événements pour tous les membres du serveur comme pour les réunions ou les rendus à faire. Discord nous a aussi permis de garder dans un canal précis les différents documents pour que tout le monde y ait accès partout.

Pour l'organisation des sprints, nous avons fait des réunions tous les dimanches en fin d'après-midi pour faire un compte-rendu de ce qui a été fait pendant la semaine. Cela nous a permis de voir les tâches qui allaient prendre plus de temps et ainsi nous adapter et fournir de l'aide à la personne chargée de cette tâche. De plus, nous avons pu découper notre projet en plusieurs Story pour pouvoir mieux organiser nos semaines de travail. Chaque semaine, nous donnions une tâche à des groupes de personnes, rarement une personne seule, pour que les tâches avancent plus rapidement et qu'il y ait plusieurs points de vue sur le problème à résoudre.

Nous avons rencontré plusieurs problèmes avec des tâches qui ont été difficiles à résoudre comme pour le remplissage d'une zone. Pour ces cas, nous avons changé les personnes chargées de cette tâche pour essayer de renouveler le point de vue sur le problème et essayer de régler le problème.

Nous avons aussi pendant certaines réunions essayé de voir les ressentis du groupe pour mettre les personnes sur des sujets où ils étaient le

plus à l'aise pour que tout le monde aient le sentiment d'avancer et d'être utile au projet.