Projet long de Technologie Objet

Rapport général

EF02

Élèves :

ARRIX-POUGET Baptiste
DEMAZURE Clement
DREUMONT Evann
GIULIANI Astrid
GUTIERREZ Tom
LAGRANGE Angel
SABLAYROLLES Guillaume
THEVENET Louis

2024

Table des matières

1. Introduction

Dans le cadre de notre projet long de TOB, nous avons choisi de développer un éditeur de dessin vectoriel nommé **Pinte** (référence à Paint). Ce choix a été motivé par la difficulté de la tâche et par la variété d'outils mobilisés. Les dessins vectoriels désignent une forme de fichiers décrits par des formes, leurs dimensions et positions, et offrent une flexibilité et une qualité bien particulière, permettant notamment des mises à l'échelle sans perte de qualité et une facilité de modification qui ne sont pas possibles avec des images normales, pratique pour faire des logos par exemple.

1.1. Objectifs du Projet

1.1.1. Objectif Général

L'objectif principal de **Pinte** est de fournir une solution complète pour la création, la modification et l'exportation de dessins vectoriels. En utilisant les principes des méthodes agiles, nous visons à développer un outil qui soit à la fois intuitif et puissant, permettant aux utilisateurs de réaliser des projets graphiques de qualité avec facilité.

1.1.2. Caractéristiques Principales Attendues

- **Choix d'Outils Complet** : Permettre à l'utilisateur de pouvoir utiliser tout les outils connus pour pouvoir dessiner sur un logiciel type Paint (sceau, sélection, construire un carré, tracer une droite, etc..).
- **Facilité d'Utilisation** : Concevoir une interface utilisateur intuitive et ergonomique, permettant même aux novices de créer et de modifier des dessins vectoriels sans courbe d'apprentissage abrupte.
- **Flexibilité et Personnalisation** : Offrir des options de personnalisation pour les raccourcis clavier et l'apparence générale de l'application.
- **Compatibilité** : Assurer la compatibilité avec les formats de fichiers graphiques les plus courants, permettant l'importation et l'exportation en SVG, JPEG, PNG, etc.
- **Performance** : Garantir une performance fluide et réactive.

1.2. Motivations et Contexte

La motivation derrière le développement de **Pinte** nous viens des défauts principaux dans les éditeurs de dessin vectoriel existants. Bien que des outils comme Adobe Illustrator et Inkscape soient extrêmement puissants, ils peuvent être intimidants pour les nouveaux utilisateurs en raison de leur complexité.. De plus, ces outils sont souvent coûteux, nécessitent des abonnements ou demande une machine puissante pour les faire tourner, ce qui peut représenter un obstacle pour les étudiants, les amateurs et les petites entreprises.

Nous avons donc décidé de créer un outil open-source qui comblerait ces lacunes, en mettant l'accent sur l'accessibilité et la facilité d'utilisation, sans sacrifier la puissance et la flexibilité attendues d'un éditeur vectoriel professionnel.

1.3. Méthodologie Agile

Pour mener à bien ce projet, nous avons adopté une approche agile, caractérisée par des cycles de développement itératifs. Cette méthodologie nous permet de rester flexibles et réactifs aux retours de notre encadrant de cours, d'intégrer des améliorations en continu et de s'assurer que le produit final soit qualitatif.

1.3.1. Phases de Développement

1. **Phase de Planification** : Les premiers cours de méthodes agiles nous ont permis plusieurs choses : identification des besoins, définition des fonctionnalités principales et secondaires, et établissement du calendrier de développement.
2. **Phase de Conception** : Suite à la première phase, nous devions élaborer des maquettes de l'interface utilisateur et concevoir des architectures logicielles et des bases de données.
3. **Phase de Développement** : Ensuite, le plus important : l'implémentation des fonctionnalités de base, suivi des sprints et des révisions, et réalisation des tests unitaires et d'intégration.
4. **Phase de Test** : Validation de la stabilité et de la performance de l'application, corrections des bugs et ajustements des fonctionnalités selon les retours des tests utilisateurs.

1.3.2. Outils et Technologies

Pour le développement en JAVA de **Pinte**, nous avons choisi de travailler avec des outils nous permettant l'agilité de notre projet :

- **API JavaFX** : Cette API nous permet de gérer l'affichage de notre application. Nous avions le choix entre celle vue en cours (SWING) ou JavaFX mais par raison d'efficacité, de fonctionnalités et du fait qu'elle est plus récente nous avons choisi JavaFX.
- **GITHUB** : Pour la gestion des versions et la collaboration en équipe, la gestion temporelle des différentes itérations et la gestion des différentes features à développer.

Fig. 1. – Gestion des itérations avec GitHub

- **DISCORD** : Pour pouvoir communiquer ensemble sur l'avancée de chacun et garder une bonne cohésion d'équipe.

1.3.3. Mises au Point hebdomadaires

Nous avons mis en place des réunions hebdomadaires pour discuter des avancées, des obstacles rencontrés et des prochaines étapes. Chaque TD de méthodes agiles nous permettaient aussi de faire le point avec notre encadrant afin de déterminer si nous prenions du retard ou non.

2. Architecture

L'application est découpée en plusieurs morceaux :

- Le `Canvas` qui est le point central de l'application, il permet notamment
  - de contenir les objets dessinés par l'utilisateur
  - contenir les paramètres du projet actuel
  - contenir l'état de l'éditeur de dessin (presse-papier, couleurs, taille de police, …)
  - fournir de quoi afficher les objets
- La classe `CanvasObject`
  - classe abstraite sur laquelle s'appuient les différents objets du canvas
  - permet d'avoir une interface commune pour les différentes classes filles
  - fournit des méthodes pour manipuler les objets (déplacer, supprimer, …)
  - sous-classes : `CanvasPolygon`, `CanvasTextField`, `CanvasEllipse`, …
- La classe `State`
  - classe abstraite sur laquelle s'appuient les classes `AddEllipseState`, `AddRectangleState`, `SelectionState`, `TranslateState`, …
  - sert à indiquer l'état de l'application
  - fournit la méthode d'ajout propre à chaque chaque forme (poser les points de polygones, tracer une droite, …)
  - fournit des méthodes pour manipuler les objets (déplacer, supprimer, …)
- Classes `Main`, `Menu`, etc
  - classes qui servent à créer les éléments graphiques de l'application
  - notamment la classe `ContextualMenu` qui fait le lien entre l'interface et les objets du canvas
  - la classe `Save` sert à sauvegarder le projet actuel
  - la classe `Export` sert à exporter au format PNG
- La classe `CanvasObjectParser`
  - classe utile à la lecture des fichiers SVG
  - permet d'instancier les objets en fonction du type de balise SVG rencontrée

3. Déroulement

Dans cette partie, nous décrirons itération par itération les différentes User Stories implantées et les choix de conception effectués.

Chaque membre du groupe a pioché parmi les User Stories prêtes à être réalisées du backlog.

3.1. Itération 1

La première itération avait pour objectif de mettre en place l'architecture du projet et l'affichage simple de notre application avec des fonctionnalités primaires telle que afficher des formes etc... Les US suivantes sont combinées avec l'itération 0 qui a permis la mise en place des outils nécessaires pour le développement de Pinte.

3.1.1. User Stories réalisées

Fig. 2. – User stories réalisées ou non dans l'itération 1

3.1.2. Code réalisé

3.1.2.1. Formes

Classe `CanvasObject` abstraite implantée par les classes d'objets spécifiques : rectangle, ellipse et polygon Ainsi nous pouvons générer différentes formes et les afficher sur un canva. D'autres formes ont été ajoutées par la suite.

3.1.2.2. Conversion SVG

Une classe `CanvasObjectParser` nous permet de récupérer des valeurs de clé dans un SVG et des méthodes `toSVG` et `createFromSVG` pour passer d'une forme de notre canva à un SVG et inversement.

3.1.3. Graphes UML

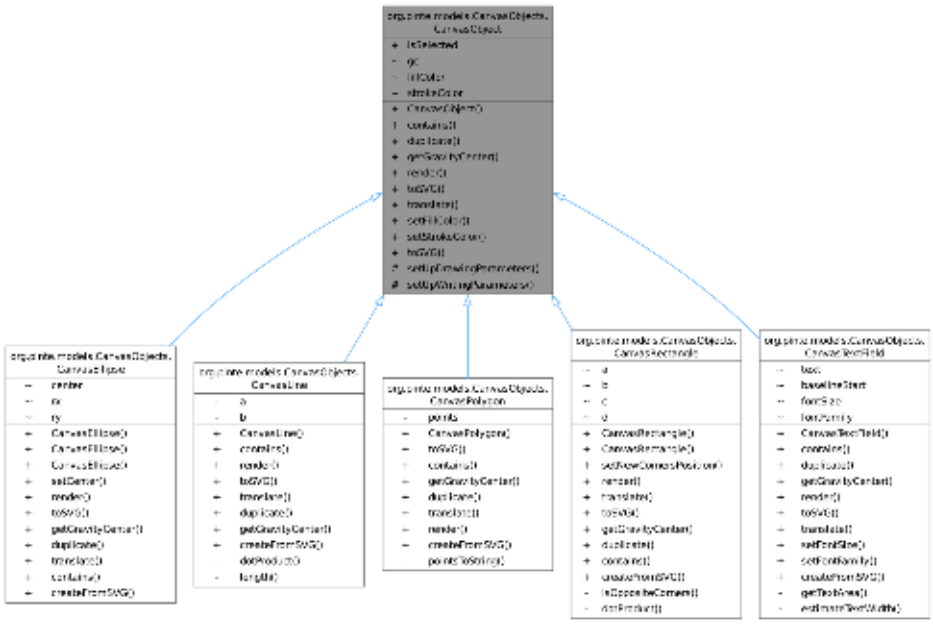


Fig. 3. – Graphe d'héritage de `CanvasObject`

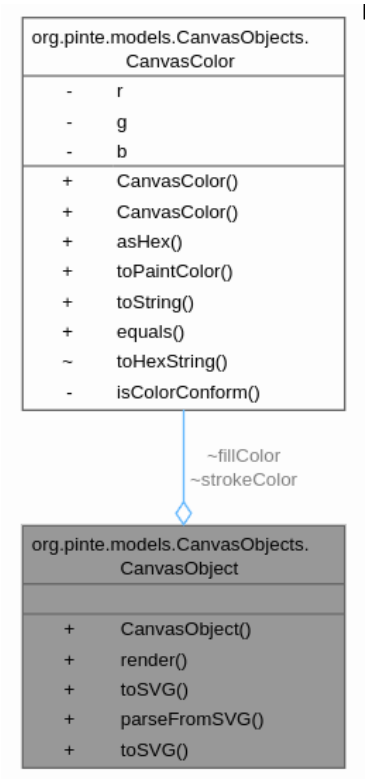


Fig. 4. – Graphe de collaboration `CanvasColor`

3.1.4. Rendu

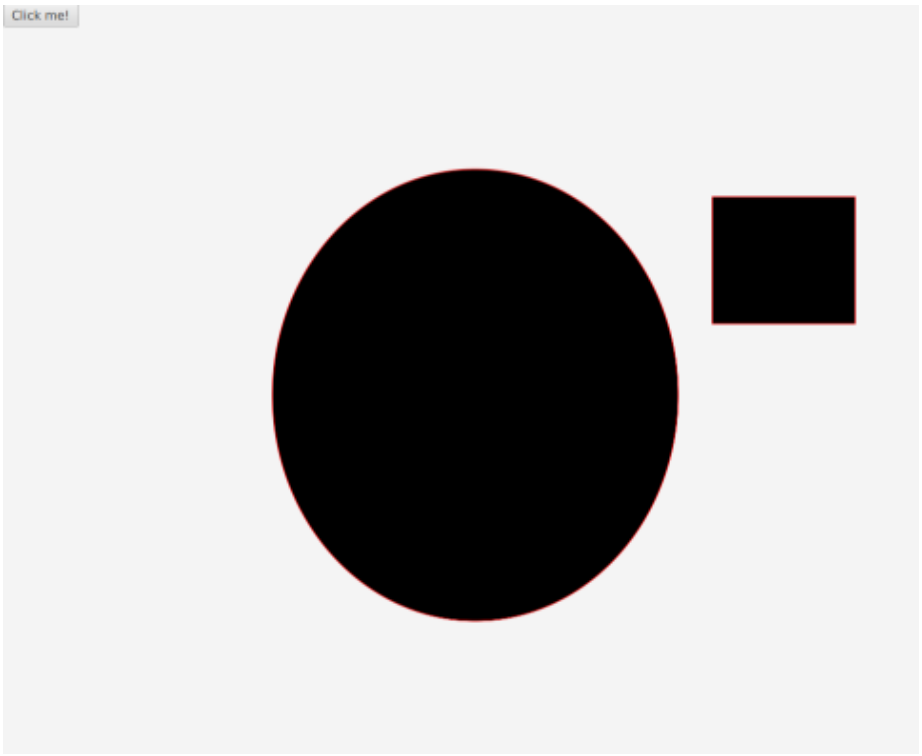


Fig. 5. – Rendu de la 1ère itération

Le bouton « click me » ici permet d'afficher une forme aléatoire.

3.2. Itération 2

Durant cette itération, l'attention s'est portée sur la gestion des fichiers, afin de pouvoir sauvegarder et rouvrir des fichiers SVG. Mais également pouvoir poser des formes à l'endroit voulu. Un outil de sélection a également été développé.

3.2.1. User Stories réalisées

Fig. 6. – User stories réalisées ou non dans l'itération 2

3.2.2. Code réalisé

3.2.2.1. `States`

Des classes ont été créées pour gérer les différentes états de l'application, création de forme, sélection, etc.

3.2.2.2. Menu

Pour pouvoir choisir quellles formes nous voulons ajouter, il nous à fallu créer un menu qui répertorie les formes et les outils disponibles.

3.2.3. Graphe UML

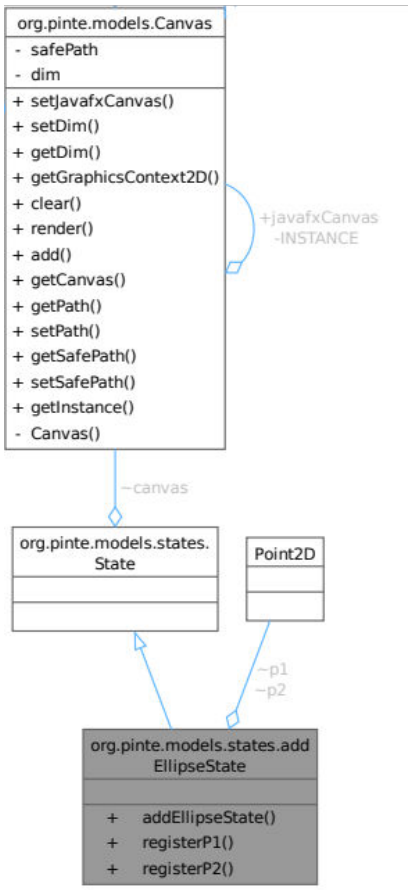
Voici le graphe pour mettre un exemple sur la relation entre les states et le canva.

Dans cette classe on y voit les méthodes :

- `registerP1()`
- `registerP2()`

(Qui servent à enregistrer deux points de l'ellipse pour définir son centre et ses rayons verticaux et horizontaux.)

Et un lien d'agrégation avec la classe abstraite CanvasObject.



3.2.4. Rendu

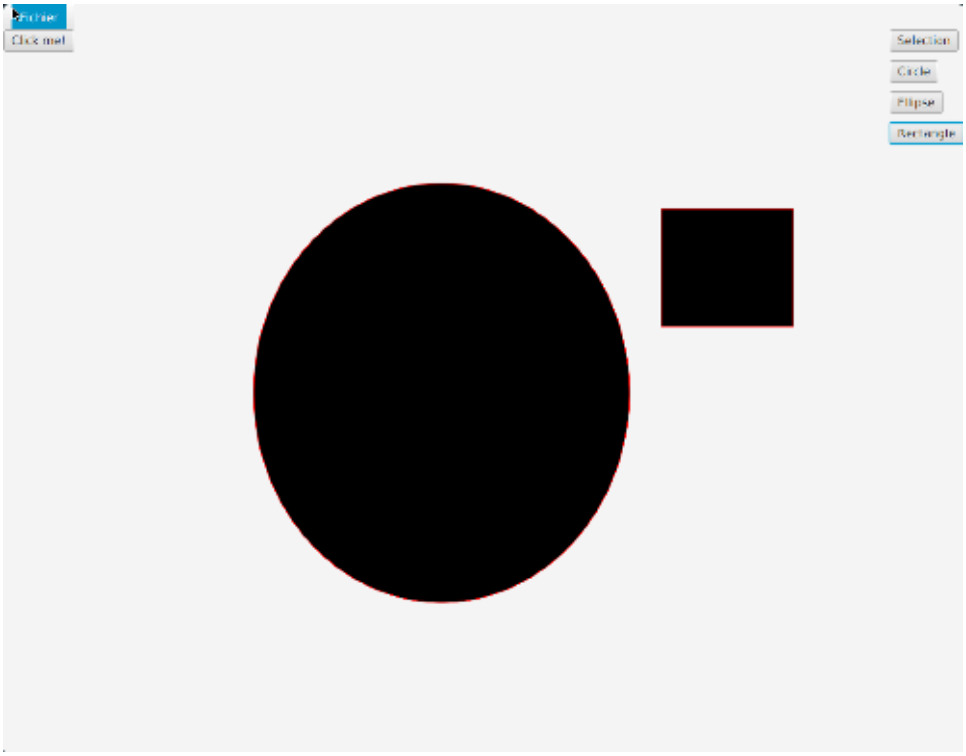


Fig. 7. – Rendu de la 2ème itération

L'interface n'est pas parfait mais on y voit les boutons outils permettant d'ajouter différentes formes et de pouvoir sélectionner celle que l'on souhaite.

3.3. Itération 3

La troisième itération avait pour objectif de rendre Pinte plus interactif, avoir une meilleure interface, et pouvoir utiliser les outils complets de ce que peut faire un outil de dessin vectoriel (remplir une forme avec une couleur, créer des segments, des polygones, copier coller, enregistrer un projet, etc…). L'interface finale est restée cependant rudimentaire mais fonctionnelle.

3.3.1. User Stories réalisées

Fig. 8. – User stories réalisées ou non dans l'itération 3

3.3.2. Code réalisé

3.3.2.1. Menu contextuel

Ajout d'un menu contextuel `CanvasContextualMenu` lors du clic droit pour pouvoir appliquer des opérations sur la sélection actuelle.

3.3.2.2. Coloration

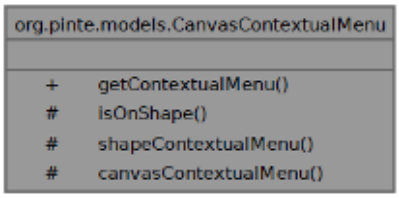
Choisir la couleur à utiliser pour créer la nouvelle forme, remplir ou changer le contour d'une forme.

3.3.2.3. Sauvegarde

Enregistrer le projet pour pouvoir le continuer ultérieurement et pouvoir l'exporter dans un format d'image souhaité.

3.3.3. Graphe UML

Voici le diagramme de classe de `CanvasContextualMenu` :



La méthode **shapeContextualMenu()** permet d'obtenir les actions à performer sur le canvas selon les touches de souris et clavier appuyées.

3.3.4. User Stories non réalisées

Fig. 9. – User stories non traitées

3.3.5. Rendu

Fig. 10. – Rendu de la 3e itération