Mémoire Projet Architecte Logiciel

Thibault Pottier

26 avril 2024

Table des matières

Introduction

Ce projet a été conçu pour répondre aux exigences du cours d'Architecture Logicielle, où l'application des concepts théoriques à des projets concrets est essentielle. L'éditeur de formes géométriques permet aux utilisateurs de créer, modifier et manipuler diverses formes géométriques à travers une interface graphique utilisateur. Ce projet vise non seulement à illustrer l'application pratique des design patterns, mais aussi à fournir une plateforme robuste pour expérimenter avec l'architecture logicielle. Dans notre étude des design patterns, nous avons consulté diverses ressources, y compris des articles détaillés sur Refactoring Guru [1] et des discussions sur Stack Overflow [2] en cas de problèmes majeurs pour essayer de trouver une solution.

Objectifs

Les principaux objectifs de ce projet étaient de :

— Sélectionner un objet depuis un menu graphique (toolbar), et le positionner sur notre dessin – (whiteboard) en utilisant le glisser-déposer (drag and drop).
— Créer des groupes d'objets et sous-groupes d'objets, sous-sous-groupes etc...
— Dissocier un groupe d'objet.
— Modifier la taille, position, etc... de nos objets ou groupes d'objets une fois ceux-ci incorporés dans le dessin.
— Ajouter des groupes d'objets ou des objets paramétrés à notre toolbar en les déposants sur la toolbar (drag and drop).
— Annuler ou refaire une opération.
— Sauvegarder un document et charger un document.
— Sauvegarder l'état du logiciel (toolbar) et le recharger au démarrage.

Méthodologie

La méthodologie adoptée pour ce projet a été soigneusement articuler pour assurer un développement cohérent et efficace autour de ses différentes étapes :

1. **Analyse des besoins** : Cette phase initiale a consisté à identifier et à définir précisément les fonctionnalités essentielles que l'application doit offrir.
2. **Sélection des design patterns** : Chaque design pattern a été choisi en fonction de son adéquation avec les problématiques spécifiques rencontrées lors de la conception.
3. **Conception** : L'utilisation de diagrammes UML a permis de modéliser clairement l'architecture de l'application et les interactions entre ses divers composants.
4. **Implémentation** : Le développement des fonctionnalités de l'application a été réalisé en Java, en essayant de respecter au maximum les différents design patterns sélectionnés.
5. **Tests et validation** : Les tests durant le processus de développement à permis d'avancer sans problèmes majeur.

Développement

Le développement de notre éditeur de formes géométriques a été structuré en plusieurs parties prenantes, de l'analyse initiale des besoins, qui a permis de cerner précisément les fonctionnalités requises et les contraintes techniques, jusqu'aux tests finaux, assurant la robustesse et la qualité du projet. La conception a été soigneusement documentée à l'aide de diagrammes UML, facilitant ainsi la compréhension et l'implémentation des fonctionnalités prévues. L'implémentation à donc été réalisée en Java.

Analyse des besoins

L'analyse des besoins va etre une phase cruciale du développement puisqu'elle définit les exigences de l'application. Pour notre éditeur de formes, cette analyse va permettre d'identifier clairement les fonctionnalités essentielles nécessaires à une interaction efficace et intuitive avec l'utilisateur, ainsi que les attentes en termes de performance et d'extensibilité de l'application. Voici les besoins détaillés qui ont été identifiés :

- **Sélection et positionnement d'objets :** Les utilisateurs vont devoir pouvoir sélectionner divers objets graphiques sur un menu (toolbar) et les placer sur une surface de dessin (whiteboard) via un mécanisme de glisser-déposer. Cette fonctionnalité nécessite une interface réactive et précise pour le positionnement et une représentation visuelle claire des objets disponibles.
- **Création de groupes d'objets :** Il est essentiel que l'application supporte la création de groupes et sous-groupes d'objets, permettant de manipuler des ensembles d'objets comme une entité unique. Cette capacité doit être récursive, permettant la création de structures hiérarchiques complexes (sous-sous-groupes, etc.).
- **Dissociation de groupes :** Les utilisateurs doivent pouvoir dissocier un groupe d'objets, les ramenant à leur état individuel initial sans affecter leur position ou leurs propriétés.
- **Modification d'objets :** Une fois les objets placés sur le whiteboard, les utilisateurs doivent avoir la possibilité de modifier leurs propriétés telles que la taille, la position et d'autres attributs pouvant varier selon leurs type, à travers une interface utilisateur intuitive.
- **Gestion des objets dans la toolbar :** Les utilisateurs doivent pouvoir ajouter des objets ou groupes d'objets à la toolbar par un glisser-déposer ou alors pouvoir supprimer un objet déjà inséré, pour une utilisation ultérieure, ce qui implique des fonctionnalités de personnalisation de la toolbar.
- **Annuler et refaire des opérations :** L'application doit fournir des commandes robustes pour annuler et refaire des opérations, permettant aux utilisateurs de corriger facilement des erreurs ou de réexaminer des étapes sans perte de données.
- **Sauvegarde et chargement de documents :** Les utilisateurs doivent pouvoir sauvegarder leur travail dans un document et le charger ultérieurement. Ce besoin inclut la sauvegarde de l'état complet de l'application, y compris la configuration de la toolbar, afin de permettre une continuité dans le travail après un redémarrage de l'application.

Sélection des design patterns

La sélection des design patterns a été essentielle pour répondre aux défis architecturaux et fonctionnels de l'application. Chaque pattern a été choisi pour optimiser certains aspects du développement et de l'interaction utilisateur, en facilitant la maintenance et l'évolutivité de l'application. Voici une explication de chaque design pattern utilisé et son application spécifique dans le projet :

- **Singleton** : Ce pattern a été utilisé pour gérer le contexte global de l'application, assurant qu'une instance unique de la configuration globale soit accessible dans toute l'application. Ceci est crucial pour maintenir la cohérence des états de l'application, en particulier pour des fonctionnalités comme les paramètres de l'interface utilisateur et la gestion des sessions.
- **Factory** : Le pattern Factory a été appliqué pour la création dynamique des menus de l'application ainsi que des différents shapes. Cela permet de centraliser la création des éléments de menu en un seul lieu, facilitant ainsi les modifications et l'extension de ces éléments sans altérer le reste du code.
- **Composition** : Utilisé pour structurer les différentes formes géométriques que les utilisateurs peuvent créer et manipuler. Ce pattern permet de traiter des groupes de formes comme des objets uniques, simplifiant les opérations complexes comme le déplacement ou la transformation de groupes entiers.
- **Prototype** : Ce pattern a été employé pour la gestion de nouvelles instances de formes géométriques. Il fournit une interface pour cloner les shapes, ce qui facilite l'ajout de nouvelles formes à l'application.
- **Command** : Le pattern Command a été implémenté pour encapsuler les actions effectuées par l'utilisateur, telles que les commandes de création, modification ou suppression de formes. Cela permet de réaliser des opérations complexes de gestion des commandes, comme l'annulation et la répétition des actions (undo/redo), de manière plus structurée et modulaire.
- **Memento** : Ce pattern est utilisé pour gérer les fonctionnalités d'annulation et de rétablissement (undo/redo), ainsi que pour la sauvegarde de l'état de l'application. Il permet de restaurer l'état précédent de l'application sans exposer les détails internes de l'objet dont l'état est sauvegardé.

Conception Implementation

Dans cette partie nous irons essayer de prévisualiser l'architecture selon les différents designs patterns que l'on à selectionner précédemment avec un exemple de chaque implémentation

1. **Singleton**
   L'implémentation du Singleton a été réalisée pour garantir que les paramètres de configuration globaux restent uniques et accessibles de manière cohérente et spécifiquement conçu pour contrôler l'accès à la configuration du système, en évitant les instanciations multiples.

FIGURE 1 – Diagramme de classe pour le Singleton

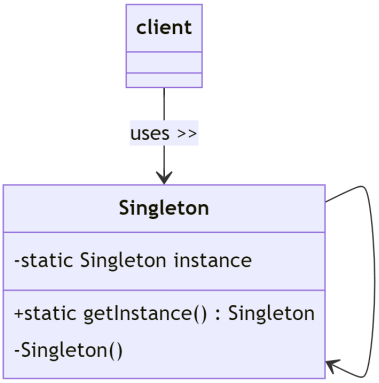
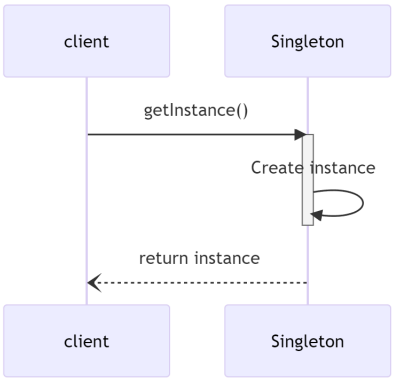


FIGURE 2 – Diagramme de séquence pour le Singleton



2. **Factory**
   La Factory a été employée pour décentraliser la création d'éléments de l'interface utilisateur, facilitant la gestion et l'extension des menus ainsi que les shapes. Les classes Factory concrètes sont utilisées pour instancier les objets menu en fonction des besoins de l'utilisateur sans coupler le code à des implémentations spécifiques.

FIGURE 3 – Diagramme de classe pour le Factory

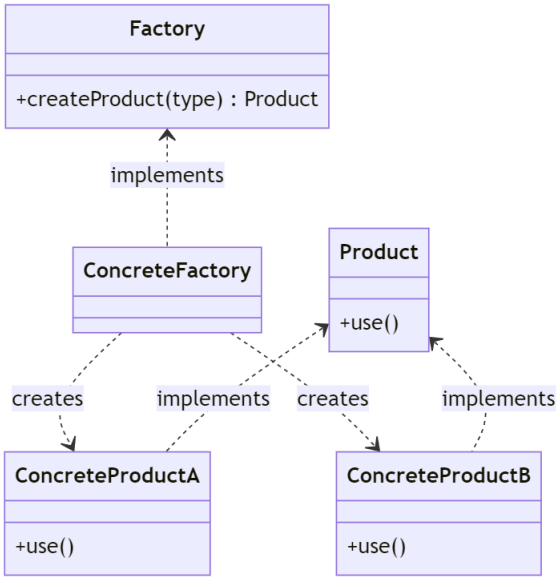
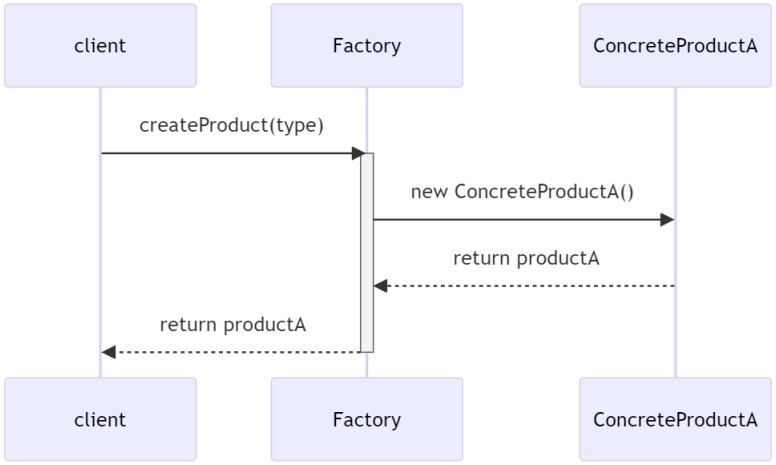


FIGURE 4 – Diagramme de séquence pour le Factory



3. **Composition**
   Le pattern Composition a été mis en place pour organiser les formes géométriques, permettant aux utilisateurs de manipuler des ensembles de formes comme s'ils étaient une entité unique. Cette structuration aide à gérer des modifications groupées et des interactions complexes entre objets composites.

FIGURE 5 – Diagramme de classe pour le Composition

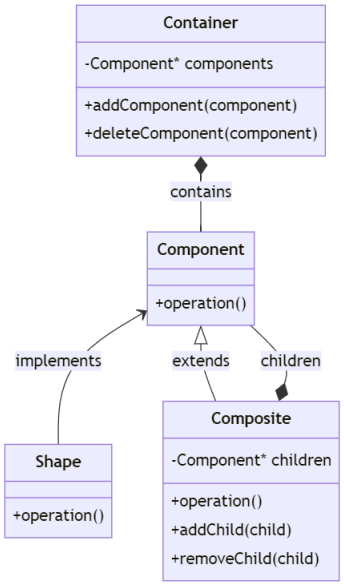
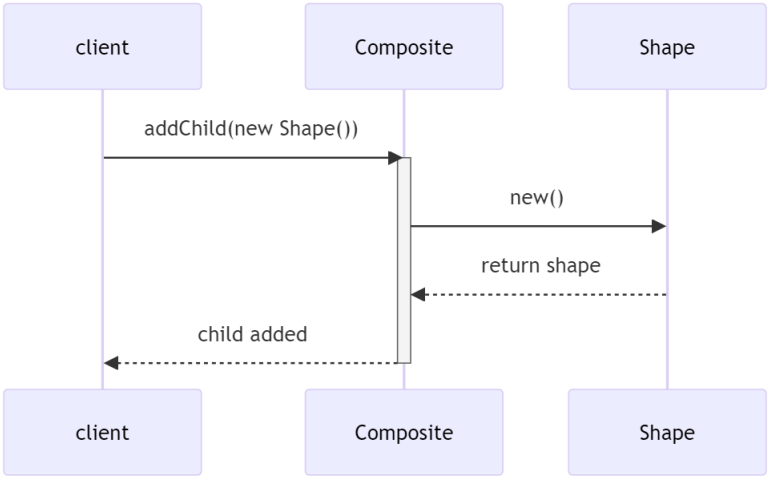


FIGURE 6 – Diagramme de séquence pour le Composition



4. **Prototype**
   Abstract Factory a été utilisé pour fournir une interface pour la création de familles de formes sans spécifier leurs classes exactes. Cela permet l'extension de l'application avec de nouvelles formes sans modifier le code existant.

FIGURE 7 – Diagramme de classe pour Prototype

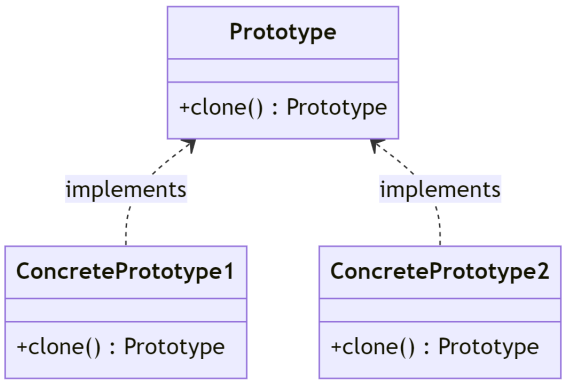
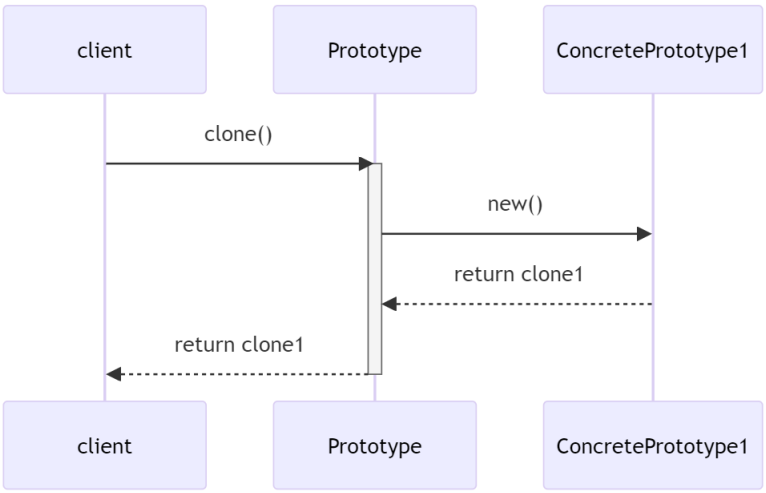


FIGURE 8 – Diagramme de séquence pour Prototype



5. **Command**
   L'architecture Command a été intégrée pour encapsuler toutes les interaction des utilisateurs possibles en tant qu'objets de commande, pour une plus grande modularité dans le traitement des interactions utilisateur, rendant le code plus adaptable et plus facile à tester.

FIGURE 9 – Diagramme de classe pour le Command

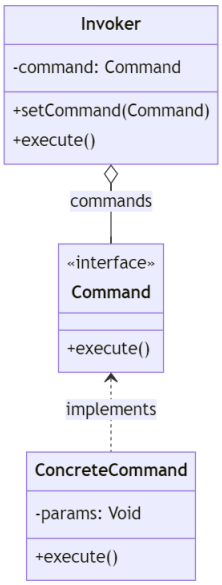
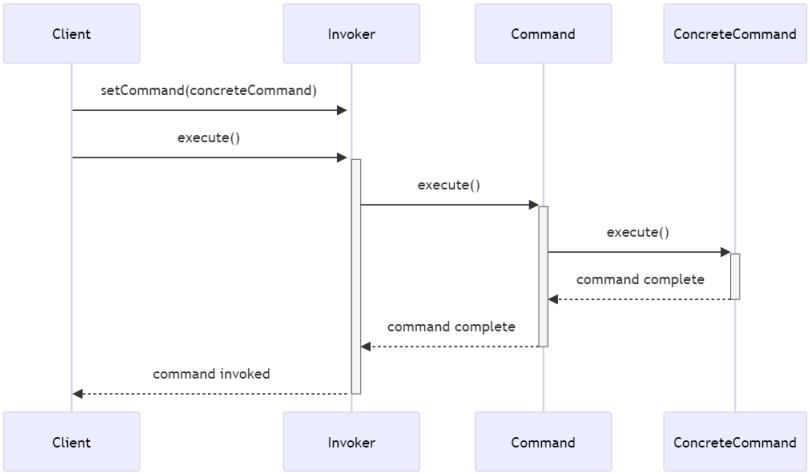


FIGURE 10 – Diagramme de séquence pour le Command



6. **Memento**
   Memento a été crucial pour l'implémentation des fonctionnalités d'undo/redo ainsi qu'aux système de backup, permettant aux utilisateurs de revenir à un état précédent sans perdre l'intégrité de l'état de l'application. Ce système stocke les états précédents de manière efficace, en veillant à ce que l'utilisateur puisse faire des modifications sans risque.

FIGURE 11 – Diagramme de classe pour le Memento

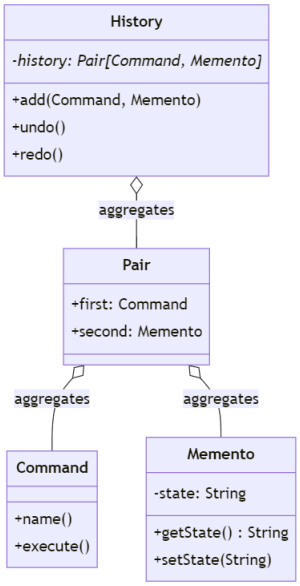
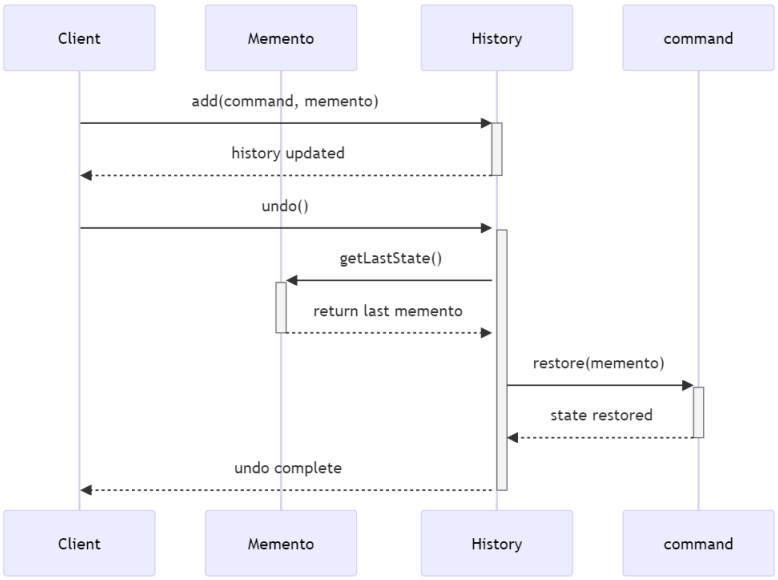


FIGURE 12 – Diagramme de séquence pour le Memento



Tests et validation

Tous les cas d'utilisation définis pour l'application ont été exécutés avec succès, confirmant ainsi que chaque fonctionnalité fonctionne correctement et répond aux exigences spécifiées.

1. **Glissé-déposer depuis la toolbar** :
   (a) L'utilisateur clique sur un élément de la toolbar.
   (b) Il effectue un glissé-déposer.
   (c) Il relâche son clic :
   — Au-dessus de la feuille blanche, une nouvelle instance de l'objet est ajoutée à la feuille de dessin.
   — Dans une icône "Poubelle", l'élément est supprimé de la toolbar.
   — Ailleurs, rien ne se passe.
2. **Groupage** :
   (a) L'utilisateur réalise une sélection potentiellement multiple d'objets.
   (b) La sélection peut se faire soit via un rectangle de sélection sur le whiteboard, soit par control-clics successifs.
   (c) Il sélectionne l'option "Group" d'un menu en clic-droit.
   (d) Les objets deviennent enfants d'un groupe d'objets.
3. **Dissociation d'un groupe** :
   (a) L'utilisateur sélectionne un groupe d'objets.
   (b) Il sélectionne l'option "De-group" d'un menu en clic-droit.
   (c) Les objets sont dégroupés, c'est-à-dire rajoutés au groupe parent. Ils ne doivent pas changer de position, etc.
4. **Édition des propriétés des objets** :
   (a) L'utilisateur sélectionne un objet ou un groupe d'objets.
   (b) Il réalise un clic droit et sélectionne "Edit" dans un menu déroulant.
   (c) Une boîte de dialogue s'ouvre : elle contient les paramètres que l'on peut modifier.
   (d) L'utilisateur change un ou plusieurs des paramètres.
   (e) Il appuie sur :
   — "Appliquer" : les modifications sont appliquées, et il peut continuer à changer des paramètres.
   — "Ok" : les modifications sont appliquées et le dialogue se ferme.
   — "Annuler" : toutes les modifications réalisées depuis l'ouverture du dialogue sont annulées.
5. **Glissé-déposé vers toolbar** :
   (a) L'utilisateur sélectionne un objet ou groupe d'objets.
   (b) Il effectue un drag'n'drop.
   (c) S'il relâche la souris sur la toolbar, un nouvel élément est ajouté dans la toolbar.
   (d) S'il relâche la souris sur la poubelle, les objets sont supprimés.

Difficultés rencontrées

Le développement de l'application a été marqué par plusieurs défis techniques et conceptuels. Voici les principales difficultés rencontrées :

- **Démarrage avec AWT** : L'utilisation de Java AWT pour la gestion des interfaces graphiques a présenté des défis dès le début du projet. Des incompréhensions sur l'utilisation de certains composants ont ralenti les premières phases de développement. La solution a été de consulter des ressources complémentaires et des forums en ligne, ainsi que de réaliser de petits prototypes pour tester les fonctionnalités avant leur intégration.
- **Gestion du redimensionnement** : La fonctionnalité de redimensionnement des formes géométriques a impliqué des calculs complexes pour maintenir les proportions et les alignements corrects. Les difficultés résidaient dans la coordination des dimensions en fonction des transformations et des contraintes d'affichage.
- **Copie des objets** : La mise en œuvre de la fonctionnalité de copie s'est heurtée à des problèmes liés à la copie partielle des objets via l'interface Cloneable de Java, qui ne permettait pas de copier en profondeur des structures d'objets complexes. Cela a conduit à des bugs liés aux pointeurs entre les objets clonés. Une méthode de copie profonde personnalisée a du être implémentée pour s'assurer que chaque composant de l'objet était correctement reproduit sans partager de références avec l'original.
- **Gestion du focus** : La coordination du focus entre le clavier et la souris s'est révélée complexe, particulièrement dans un environnement riche en éléments interactifs. Les problèmes concernaient la détermination de l'élément ayant le focus pour les entrées du clavier après des interactions avec la souris. Des ajustements dans la gestion des événements à été nécessaires pour garantir une manipulation intuitive et cohérente du focus.

Ajout supplémentaires

Pour améliorer l'efficacité et l'intuitivité de l'utilisation de l'application, plusieurs raccourcis clavier et fonctionnalités supplémentaires ont été ajoutés. Ces ajouts sont destinés à optimiser les flux de travail des utilisateurs et à accroître la rapidité des interactions avec l'application.

- **CTRL+S (Sauvegarde)** : Implémenter un raccourci pour la sauvegarde rapide des projets en cours. Cela permettra aux utilisateurs de sécuriser régulièrement leur travail sans interrompre leur processus créatif.
- **CTRL+O (Ouverture)** : Un raccourci pour ouvrir rapidement un projet existant. Ce raccourci facilitera l'accès aux travaux précédents, rendant le processus de chargement des fichiers plus fluide et moins intrusif.
- **CTRL+Z (Annuler)** : Renforcer la fonctionnalité d'annulation pour permettre aux utilisateurs de revenir facilement sur leurs dernières actions. Ce raccourci est essentiel pour une exploration créative sans risque.
- **CTRL+Y (Refaire)** : Complémentaire à l'annulation, ce raccourci permettra de rétablir les actions précédemment annulées.
- **CTRL+G (Groupage)** : Un raccourci pour grouper rapidement plusieurs objets sélectionnés. Ce raccourci accélérera les modifications complexes en permettant aux utilisateurs de manipuler des groupes d'objets comme s'ils étaient une entité unique.
- **Suppression depuis la toolbar d'une shape** : Permettre aux utilisateurs de supprimer directement des formes de la toolbar via un mécanisme de suppression pour ne pas surchargé la toolbar inutilement. Cette fonctionnalité ajoutera une couche de personnalisation à l'interface utilisateur, permettant aux utilisateurs de mieux organiser et personnaliser leur espace de travail.

Pistes d'amélioration

En vue d'une amélioration à grande échelle de l'application, voici une liste exhaustive de certaines améliorations qui pourraient être possibles. Ces propositions visent à améliorer la performance, l'utilisabilité et l'efficacité de l'application, tout en tenant compte des défis technologiques actuels et futurs.

1. **Déplacement simultané** : Actuellement, le déplacement de plusieurs objets non groupés doit être effectué individuellement. L'intégration d'une fonctionnalité permettant le déplacement simultané de plusieurs éléments sélectionnés qui ne sont pas groupés améliorerait significativement l'expérience utilisateur.
2. **Gestion de la mémoire cache** : L'implémentation d'une gestion de la mémoire cache pourrait considérablement améliorer la réactivité et l'efficacité de l'application, particulièrement pour les opérations qui nécessitent un accès rapide et répété à certaines données, comme le rendu des formes ou les fonctionnalités d'undo/redo. Cela pourrait également réduire la charge sur la mémoire principale et accélérer le traitement des données, en particulier pour les utilisateurs travaillant sur des projets complexes avec de nombreux éléments graphiques.
3. **Ajout de thread** : L'intégration de threads pour gérer les calculs en parallèle pourrait significativement augmenter la vitesse et la réactivité de l'application lors de l'exécution de tâches multiples ou complexes. Cela serait particulièrement utile pour des opérations comme le recalcul de la géométrie des formes, le rendu en temps réel ou encore la gestion simultanée de multiples entrées utilisateur.
4. **Rétrocompatibilité** : Assurer la rétrocompatibilité des sauvegardes lors des mises à jour du logiciel pour éviter la perte de données. Il est important que les fichiers sauvegardés avec les versions antérieures restent utilisables avec les nouvelles versions du logiciel.

Exécution

Exécution .exe

Commande d'exécution

Il est possible d'exécuter l'application via le .exe.

```
cd ./executable
./FigmaV2.exe
```

Listing 1 – Command execution .exe

Exécution .jar

Prérequis

- **JDK Requis :** JDK 21.

Assurez-vous que au moins le JDK 21 est installé sur votre machine. La version du JDK est disponible depuis le site officiel d'Oracle ou directement via ce lien : https://download.oracle.com/java/21/latest/jdk-21_windows-x64_bin.exe

Commande d'exécution

Pour exécuter l'application sous Windows, utilisez la commande suivante dans PowerShell. Assurez-vous que le chemin d'accès à votre JAR et à votre JDK est correct et que votre classe principale est correctement spécifiée :

```
& "C:\Program Files\Java\jdk-21\bin\java.exe" -cp ".\FigmaV2-1.0-SNAPSHOT.jar"
    thibault.kuraima.Main
```

Listing 2 – Command execution .jar

Références

[1] Design patterns. https://refactoring.guru/fr/design-patterns. Accédé durant la continuité du projet.

[2] Stack overflow. https://stackoverflow.com/. Accédé durant la continuité du projet.