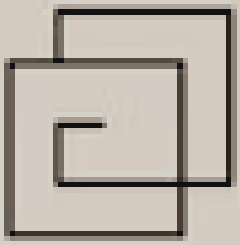
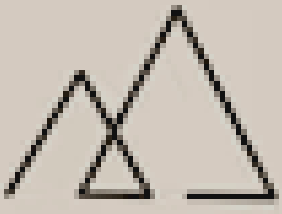
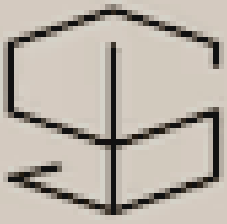






2023 - JAVAFX

PROJET DE PIIA

MINIMALIST LOGOS

PRÉPARÉ PAR

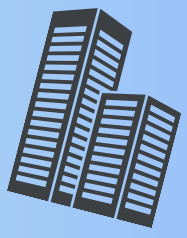
Ahmed BAAROUN

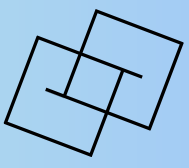
TABLE DES MATIÈRES

A Propos de MINIMALIST LOGOS

Minimalist Logos est une application de dessin, globalement, qui a été pensée à deux, avec Yanki OZAGAC, qui s'est malheureusement retiré du projet.

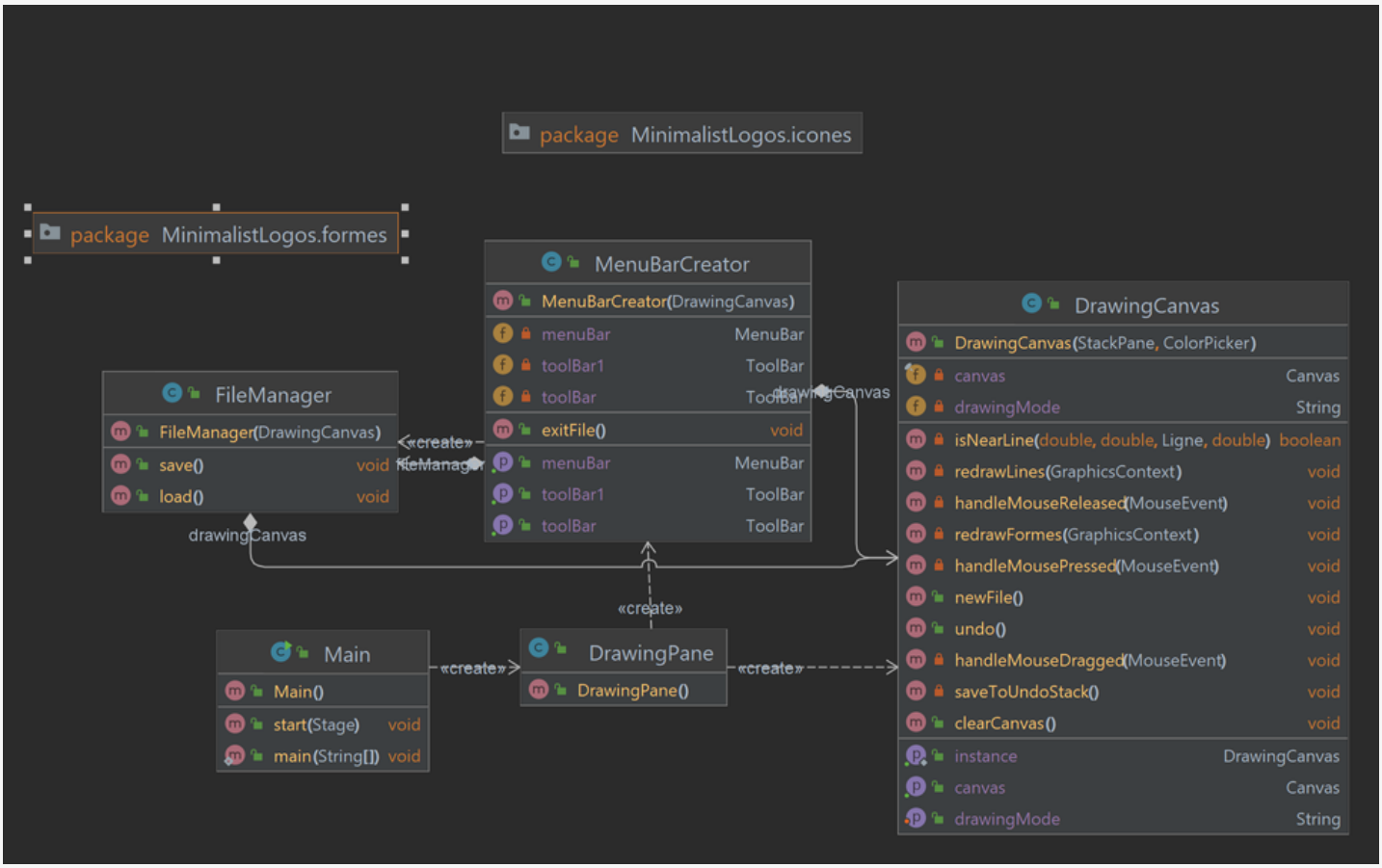
En effet, c'est un outil qui doit être perfectionné et orienté efficacité, dans le cadre de création de logos minimalistes, d'où le titre du projet, permettant aux professionnels de pouvoir se créer une identité visuelle captivante et simple, dans une ère qui favorise le simplisme !





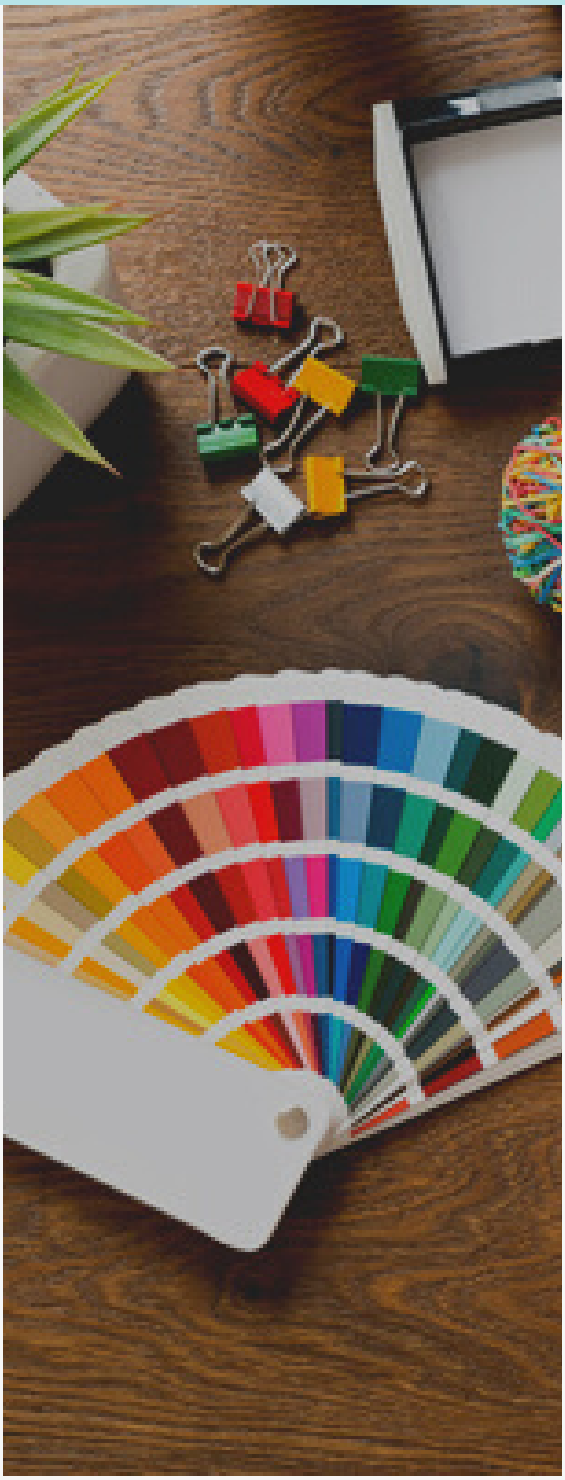
Organisation de M.L.

= Minimalist Logos



Minimalist Logos suit une architecture MVC (Modèle-Vue-Contrôleur), et est largement développée avec JavaFX afin d'en soigner la présentation et lui attribuer une IHM efficace. Pour commencer, l'application détient tout d'abord une classe importante qu'est "DrawingCanvas", abritant toutes les méthodes et fonctionnalités relatives aux réalisations de dessins en elles-même, ce qui en fait la classe la plus étendue avec 391 lignes de code.
On y trouve par exemple toutes les méthodes de contrôle de la souris, ainsi que les méthodes permettant de construire le Canva principal sur lequel apparaît le dessin réalisé.
Ensuite, on a une classe "MenuBarCreator" contenant toutes les barres de menus déroulants (ex : Fichier, Affichage...), ainsi que des barres de boutons permettant de choisir son outil de création (Crayon, Ligne...), mais aussi d'avoir un contrôle sur son travail (Annuler et Rétablir des actions). Les barres, les événements et les canvas sont par la suite intégrés dans une méthode "DrawingPane" permettant de tout implémenter et d'organiser l'affichage des éléments, affichés finalement grâce à la classe "Main" qui intègre tout cela dans une fenêtre. Voici par ailleurs un diagramme de classes ci-dessus pour illustrer l'architecture de ML.

Concernant les fonctionnalités...



Pour commencer, ML étant une application de dessin, on y trouve des fonctionnalités de dessin, naturellement !

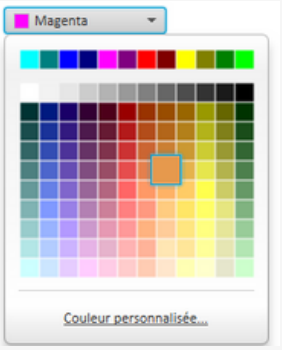
Tout d'abord, nous avons le choix entre la liberté du crayon, ou la régularité de certaines formes implémentées dans des classes avec des méthodes de dessin, elles-mêmes héritées de l'interface "Forme" suivante :

```java
package MinimalistLogos.formes;

import javafx.scene.canvas.GraphicsContext;

// Interface Forme définissant les méthodes communes pour toutes les formes géométriques
public interface Forme {

    // Méthode pour dessiner la forme sur le GraphicsContext donné
    void dessiner(GraphicsContext gc);

    // Méthode pour définir les coordonnées de départ (x, y) de la forme
    void setDebut(double x, double y);

    // Méthode pour définir les coordonnées de fin (x, y) de la forme
    void setFin(double x, double y);

    // Méthode pour vérifier si un point (x, y) est contenu dans la forme
    boolean contains(double x, double y);

    // Méthodes pour obtenir les coordonnées de départ et de fin de la forme
    double getStartX();

    double getStartY();

    double getEndX();

    double getEndY();
}
```

On y trouve tout type de signatures de méthodes de dessin, de test, ou de retour de valeurs indispensables au fonctionnement de ML.

Ergonomie au service du dessin

Sans oublier le crayon, on peut bien évidemment choisir la couleur à appliquer, grâce au ColorPicker, importé par la bibliothèque "javafx.scene.control.ColorPicker", et intégré directement au MenuBar à la suite des autres éléments-barres.

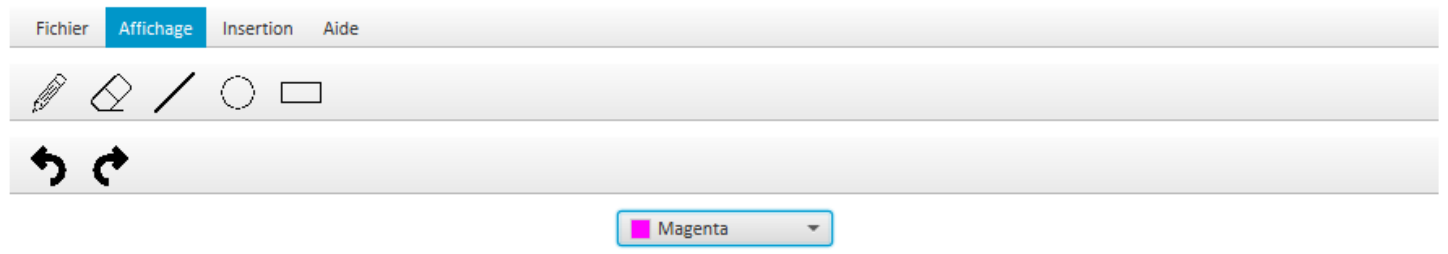


Des icônes ont été placées sur les boutons des outils de dessin afin d'avoir une reconnaîssance systématique de ces derniers.

De plus, on peut annuler ou rétablir les actions réalisées grâce aux boutons d'une autre barre de boutons, toutes deux grâce à des ToolBar-s.

Au-dessus de tout ce beau monde on trouve une barre de menu assez conventionnelle avec des Menu-s et des MenuItem-s, ainsi que des sous-Items dans certains menus tels que les actions de zoom.

Avant de continuer, voici un aperçu de cette partie de la fenêtre (par ailleurs stockée dans une VBox dans la classe "DrawingPane", avec le colorPicker en plus, en l'affichant en haut (TOP) d'un BorderPane :



Voici le code, vulgairement, de l'affichage de cette partie de l'application :

```
// Création de l'objet MenuBarCreator en lui passant l'objet drawingCanvas
MenuBarCreator menuBarCreator = new MenuBarCreator(drawingCanvas);
// Création d'un conteneur VBox pour accueillir la barre de menu et le colorPicker
VBox topContainer = new VBox(menuBarCreator.getMenuBar(), menuBarCreator.getToolBar(), menuBarCreator.getToolBar1(), colorPicker) ;

// Alignement des éléments au centre
topContainer.setAlignment(Pos.TOP_CENTER);
//couleurs.setAlignment(Pos.BOTTOM_CENTER);

// Définition des marges autour du conteneur
topContainer.setPadding(new Insets( v: 10,  v1: 10,  v2: 10,  v3: 10));

// Définition de l'espacement entre les éléments du conteneur
topContainer.setSpacing(10);
// Ajout du conteneur en haut du BorderPane
setTop(topContainer);

setCenter(canvasContainer);
```

Fonctionnalités non graphiques

Dans un souci de contrôle de travail, il a été rendu possible d'annuler ses actions, avec une méthode d'annulation undo() avec une méthode "annexe" saveToUndoStack() permettant de sauvegarder l'état du canvas en cours dans une pile dite d'annulation afin de pouvoir justement revenir dessus avec undo(), elle-même reliée au bouton undoButton dans MenuBarCreator :

```java
// Méthode pour annuler la dernière action effectuée sur le canvas
public void undo() {
    if (!undoStack.isEmpty()) {
        double width = gc.getCanvas().getWidth();
        double height = gc.getCanvas().getHeight();

        // Sauvegarde de l'état actuel du canvas dans la pile de rétablissement (redoStack)
        WritableImage currentImage = new WritableImage((int) width, (int) height);
        canvas.snapshot( snapshotParameters: null, currentImage);
        redoStack.push(currentImage);

        // Restauration de l'état précédent du canvas à partir de la pile d'annulation (undoStack)
        WritableImage previousImage = undoStack.pop();
        gc.drawImage(previousImage,  v: 0,  v1: 0);
    }
}

// Méthode pour sauvegarder l'état actuel du canvas dans la pile d'annulation (undoStack)
private void saveToUndoStack() {
    WritableImage currentImage = new WritableImage((int) canvas.getWidth(), (int) canvas.getHeight());
    canvas.snapshot( snapshotParameters: null, currentImage);
    undoStack.add(currentImage);
}
```

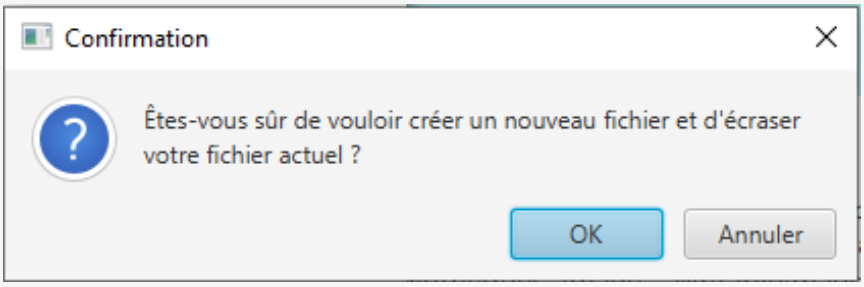
, soit aussi dans MenuBarCreator :

```java
undoButton.setOnAction(event -> drawingCanvas.undo());
```

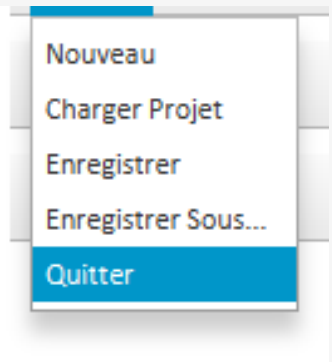
Au niveau des menus

Il est possible de sauvegarder son travail dans une image png, ou d'insérer une image png par la même occasion, le tout grâce à une seule classe "FileManager" qui utilise les bibliothèques "Image", "WritableImage", "FileChooser" et "FileInputStream" pour ce faire, avec des gestions d'erreurs au cas où !

Dans ML, il est aussi possible d'écraser le fichier en cours et d'en créer un nouveau (Canvas), après avoir confirmé, grâce à "Alert" de JavaFX, de la manière suivante :



Enfin, hormis la "croix rouge" de fermeture de fenêtre, on peut quitter l'application grâce à "Platform" de JavaFX par un élément du menu "Fichier", appelé "Quitter" tout simplement, et qui doit être confirmé tout comme l'écrasement de fichier :



Tout pour satisfaire J. Nielsen

CERTAINES FONCTIONNALITÉS PERMETTENT DE RESPECTER CERTAINES DES HEURISTIQUES DE NIELSEN POUR UNE "BONNE" IHM

Voici un exemple par heursitique respectée :

- Visibilité du statut du système : Couleur choisir du ColorPicker toujours affichée
- Correspondance entre le système et le monde réel : Des icônes "connues" (Crayon...)
- Contrôle et liberté de l'utilisateur : Possibilité d'annuler son action
- Cohérence et standards : Sentiment de déjà vu du logiciel donc mise à l'aide de l'utilisateur
- Prévention des erreurs : Besoin de confirmation pour écraser ou quitter ML
- Esthétique et design minimaliste : Pas de présentation tape-à-l'oeil et fichier CSS pour le style

Documentation de l'utilisateur (développeur)

Dans le cadre de l'usage de JavaFX, il faut faire appliquer certaines manipulations nécessaires :

- Commencer par télécharger la bibliothèque JavaFX facilement trouvable sur internet
- Après ouverture/lancement de votre projet, accéder (sur IntelliJ) à File -> Project Structure -> Modules -> Depencies, puis appuyer sur "+", choisir "JARs or Directories" et sélectionner le dossier des exécutables de JavaFX, préalablement téléchargés, souvent appelé "lib".
- Cocher ce dossier et appliquer les changements
- Dans la configuration de l'exécution, ajouter l'option "VM Options" et y intégrer : "--module-path CHEMIN DU DOSSIER DE JARs --add-modules javafx.controls,javafx.fxml"

Les sujets qui fâchent

Pas mal de difficultés ont honnêtement été rencontrées lors du développement de cette application.

En effet, dans le cadre notamment d'un souci de volonté de respecter les heuristiques de Nielsen, certains bouts de code sont mis en commentaires car le temps n'a pas été géré correctement afin de pouvoir les ajouter en tant que fonctionnalités à ML.

Mon binôme Yanki OZAGAC a par ailleurs manqué à l'appel assez tôt ce qui m'a mis dans une situation délicate, sachant les travaux à faire hors PÏIA ainsi que la recherche de master et d'alternance.

Globalement, tous les boutons et éléments de menus devaient avoir des portions fonctionnelles dans le code source.

De plus, en guise d'exemple, il devait aussi y avoir une barre d'état indiquant l'outil de dessin en cours d'utilisation !

Pour Finir

La programmation d'interfaces interactive est très subtile mais très intéressante, néanmoins, beaucoup de fonctionnalités n'ont malheureusement pas été ajoutées.