

# Rapport final de projet PIIA :

(par Léo Ressayre & Arthur Clot)

## INTRODUCTION :

Une entreprise spécialisée dans l'habitat souhaite fournir un outil à ses clients pour les aider à planifier leur cuisine avec des équipements.

Pour cela, cette entreprise nous a demandé de créer une application permettant de dessiner le plan d'une cuisine en 2 dimensions.

Voici le rapport final qui traitera de l'implémentation de notre application.

## ENVIRONNEMENT DE TRAVAIL & ORGANISATION :

Afin de partir sur des bases connues (les TPs), nous avons décidé d'utiliser l'IDE Eclipse, ainsi que la bibliothèque JavaFX associée à l'outil SceneBuilder. Nous nous sommes aussi aidés de Github pour faciliter notre travail collaboratif.

Au niveau de l'organisation, nous avons choisi d'utiliser l'architecture MVC (Modele Vue Contrôleur), mais sans explicitement séparer ces différentes sections :

Modèle : La partie modèle de notre code est séparée dans un package qui lui est dédié. Cette partie comprend les classes utiles à la logique de notre application.

. Item : cette classe sert à représenter un Item, qui est en fait un meuble, un accessoire électroménager, ou tout autre objet pouvant être ajouté sur notre éditeur. Cette classe nous permet d'associer à chaque Item des coordonnées (utiles aux différents affichages dans le Catalogue ou l'Editeur), un angle de rotation, mais aussi un URL qui sert à garder un lien avec l'image associée à l'Item.

. Projet : cette classe sert à représenter un projet de cuisine. Y sont associés un nom, des dimensions (en mètre), ainsi qu'une liste d'Item qui ont été ajoutés à ce projet, avec leurs coordonnées etc..

. Type : cette classe sert à représenter une catégorie d'Item (Chaise, Table de cuisson, etc). Elle nous est utile pour charger les différents Items de notre catalogue. On retrouve donc un nom (nom de la catégorie), ainsi qu'une liste d'Item qui portera ainsi les différents Items associés à la catégorie.

Vue & Contrôleurs : Cette partie est la plus complexe de toutes. Elle est composée de deux sous-parties :

. Les scripts décrivant nos différentes vues (Page principale, Editeur, Page de Projet, Catalogue, etc) qui sont donc des fichiers fxml. Afin de normaliser notre code, ils sont tous nommés comme ceci : "NomDeLaPage" + Layout. Nous les avons créés, et synchronisés avec leurs contrôleurs grâce au SceneBuilder.

. Les scripts implémentants les contrôleurs de nos vues, qui sont des fichiers java. Leur nommage est normalisé aussi : "NomDeLaVueAssociée" + Layout + Controller.

MainApp : Cette classe est la classe principale de notre application. Elle porte les différentes scènes, elle instancie les vues et les contrôleurs, lance le Thread principale de l'application, et permet de relier toutes les parties de notre projet ensemble.

Par ailleurs, nous avons rajouté un répertoire 'Image' qui contient les différentes images d'Item. Ils sont organisés selon leurs catégories afin de faciliter leur organisation au sein de l'application.

## **IMPLEMENTATION :**

Dans cette section, nous allons décrire chronologiquement l'implémentation de notre application.

### **Mise en place :**

Afin de débuter, nous avons commencé par mettre en place les packages Vues et Modèles, et aussi créer la classe principale MainApp. Nous avons ensuite créé les premières lignes permettant le lancement et l'affichage de l'application. Cette classe implémentant la classe Application de JavaFX, nous avons donc créé la méthode 'start', qui instancie donc le Stage principale, et affiche l'accueil de l'application, ainsi que la méthode main qui sera ainsi le main de notre application.

### **RootLayout :**

Nous avons ensuite créé la première brique de notre application : la vue RootLayout. Cette vue est composée d'un BorderPane qui va porter dans la partie TOP le header de toute l'application (visible sur chaque page) et qui est composée du titre "Cuisine Edit" (accessoirement hyperLien vers l'accueil afin de respecter le standard des sites où en cliquant sur le titre dans le header nous permet de revenir à l'accueil), ainsi que des trois boutons Accueil, Projet, et Catalogue qui mènent chacun à leur page associée.

Ce layout sera la base visuelle de toute notre application puisqu'on viendra, au fil de l'utilisation, greffer dans la partie CENTER chaque différentes pages. Cela permet une fluidité visuelle puisque la base reste fixe, et la corp change au gré de l'utilisateur.

Associé à cette vue, le RootLayoutController vient ajouter à notre vue les fonctions utiles, associées aux différents boutons. Elles sont introduites par les '@FXML' qui permet de déclarer ces fonctions comme fonctions associées à un fichier FXML, et ainsi, au sein du SceneBuilder, d'édition les champs 'OnAction' des différentes features.

Enfin, dans le but d'instancier et d'afficher ce RootLayout au lancement, la méthode 'start' de notre classe MainApp va appeler une fonction 'initRootLayout()' qui va charger le fichier FXML (fxmlLoader), instancier le contrôleur, et finalement créer une scène qui portera notre vue.

### **Développement :**

Le reste de notre application a été créé de cette façon pour chaque pages différentes :

#### **Etape 1 : Mise en place :**

On crée deux fichiers, l'un fxml et l'autre java. L'un sera destiné à la vue, et l'autre portera le contrôleur de cette vue. On ajoute en plus une fonction ' show+"NomDeLaVue" ' dans la MainApp qui permettra de supprimer la vue actuelle, puis de charger le FXML de la vue à afficher et de la greffer au centre de notre RootLayout, d'instancier le contrôleur associé, ainsi que de l'initialiser. Cette initialisation aura la plupart du temps pour but principal de charger les données à afficher avant toute action de l'utilisateur, mais aussi de lier ce contrôleur à notre MainApp, et donc au reste du projet.

#### **Etape 2 : FXML & SceneBuilder :**

On édite, grâce à l'outil SceneBuilder, le layout de la vue. Cette édition consiste à organiser notre vue de manière très simpliste, avec les différents boutons, les différentes séparations de notre affichage, etc... A cette étape-ci, nous ne nous occupons pas encore du style de nos pages car nous nous concentrions surtout sur la logique de l'application.

#### **Etape 3 : Contrôleur**

Cette étape est l'étape qui de manière générale est la plus longue et complexe. Elle consiste dans un premier temps en la synchronisation des différents Labels, Canvas, TextFields du fichier FXML avec le contrôleur, puis en la création des premières méthodes, comme les fonctions d'initialisation de notre classe. Par exemple, dans notre page d'édition, nous utilisons un Canvas pour afficher le projet de cuisine. Mais ses dimensions

changent selon les dimensions de la cuisine, et son contenu, sauvegardé lorsque l'on sort du projet, doit être affiché lorsque l'on arrive sur l'éditeur. On doit donc initialiser ce canvas avant que l'utilisateur ne fasse quoi que ce soit.

Ces fonctions d'initialisation sont appelées dans la classe MainApp.

Puis, après une courte phase d'analyse, nous créons les éventuelles classes modèles (Item, Projet...) nécessaires.

Ensuite, il faut s'occuper des différentes méthodes propres aux vues. Il s'agit donc de créer les fonctions associées aux boutons, au clique, ou drag&drop de souris etc... Selon la complexité de ces méthodes, nous créons parfois des fonctions annexes.

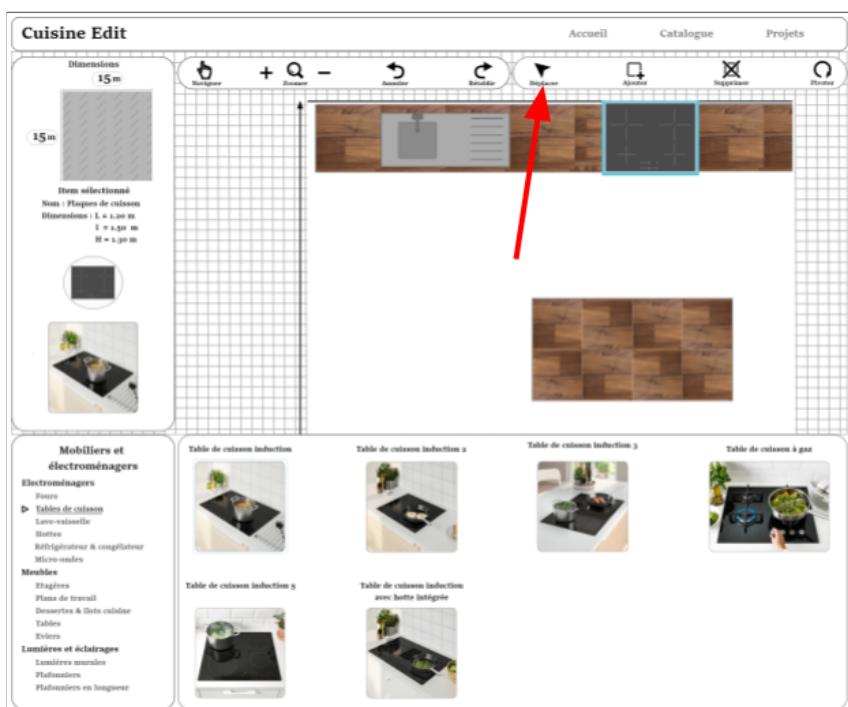
Par exemple, toujours sur notre éditeur, nous avons créé une fonction 'drawCanvas' qui, grâce à un attribut 'currentProjet' de notre classe contrôleur (qui représente l'état actuel du projet), va afficher les différents Items sur le canvas. Cette fonction sera alors par exemple appelée dans notre méthode 'deplace' qui est associée au drag&drop de la souris.

## Améliorations :

Après avoir créé les grandes parties de notre projet (nous avions page par page, en faisant dans l'ordre Root, Accueil, Projet, Editeur, Catalogue), nous avons décidé de faire tester l'application à nous et à nos familles. L'idée était de se rendre compte de la façon dont ils se servaient des différentes fonctionnalités, afin de les rendre plus intuitives et ergonomiques.

Nous avons ainsi pu remarquer quelques problèmes que notre Cahier des charges de notre rapport préalable induisait, notamment pour l'éditeur.

Une des premières choses qui est apparue est que le bouton initialement appelé 'Déplacer' n'était pas du tout intuitif. A l'origine, notre éditeur présentait ce bouton qui servait à sélectionner un des items sur le canvas, puis en drag&drop de le déplacer, mais aussi de sélectionner un des items du catalogue avant de l'ajouter en cliquant sur le bouton ajouter :



En effet, nos proches nous ont rapidement dit que cliquer sur un bouton de la barre d'outils pour simplement sélectionner un Item sur le canvas ou le catalogue n'était pas du tout intuitif pour eux.

De la même façon, il fallait cliquer sur 'Sélectionner', sélectionner un item puis cliquer sur le bouton 'Supprimer' afin de supprimer cet item (3 cliques en tout).

Un autre aspect problématique était qu'à l'inverse des autres pages, la page de l'éditeur était plus grande que la fenêtre, ce qui obligeait l'utilisateur à scroller vers le bas pour voir le catalogue de l'éditeur, et ainsi ajouter des éléments.

Pour remédier à ces problèmes, nous avons décidé d'enlever la partie catalogue de notre éditeur. A la place, pour ajouter un élément, il faudra cliquer sur le bouton 'Ajouter', ce qui ouvrira une nouvelle fenêtre flottante, puis sélectionner l'item voulu et cliquer sur le bouton 'Ajouter' de cette fenêtre flottante. Ainsi, on garde un aspect très organisé des différentes parties de l'éditeur, mais en laissant plus d'espace au schéma de cuisine, et sans forcer l'utilisateur à scroller.

Ensuite, nous avons décidé de supprimer le bouton ‘Déplacer’. A la place, nous avons fait en sorte qu’un simple clique sur un item du canvas permettrait de le sélectionner, et en draguant, de le déplacer. Ainsi, pour supprimer un item, il suffit de cliquer sur cet item, ce qui le sélectionnera (un encadrement bleu permettra de le signifier à l’utilisateur afin de respecter l’heuristique 1), puis de cliquer sur le bouton de la barre d’outil ‘Supprimer’.

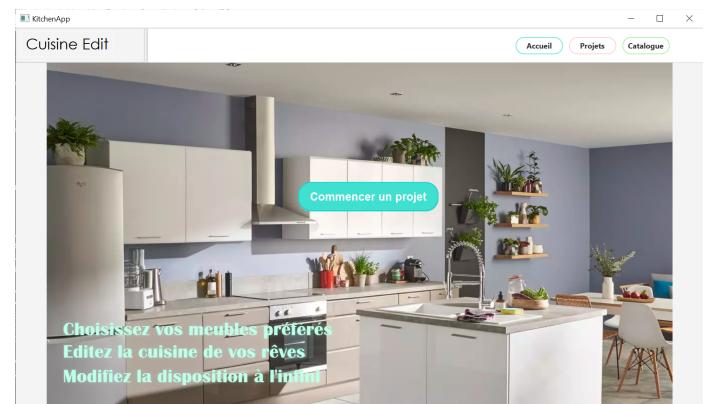
### Style :

Pour finir, nous nous sommes attardé sur le style de notre application. En effet, à l’origine, le logiciel n’était composé que de textes, de blocs gris, et de boutons peu esthétiques. Nous avons donc rajouté du CSS à nos fichiers FXML afin de rendre le tout plus agréable à l’œil. Nous avons ainsi rajouté des photos pour la page d’accueil, quelques couleurs, ainsi qu’un style différent pour les boutons, afin de rendre le tout plus joli, sans pour autant chercher trop dans l’originalité (heuristique 4). Cette étape a, néanmoins, surtout consisté en la copie de notre prototype fait pour le rapport préalable.

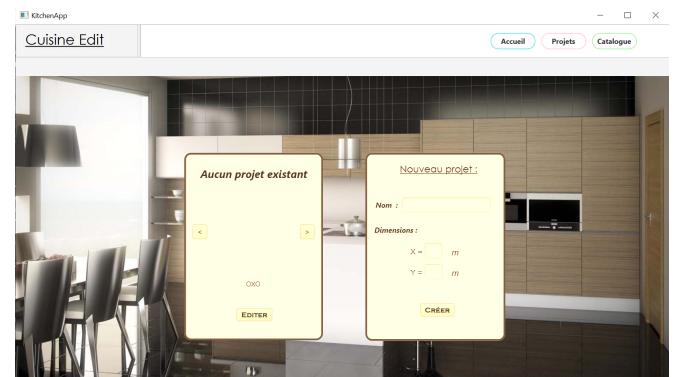
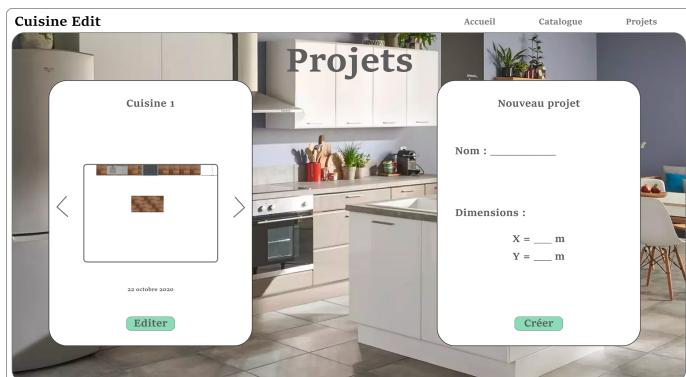
## RÉSULTAT & FONCTIONNALITÉS :

Comme dit précédemment, nous avons tenté de nous rapprocher au maximum du modèle fait lors du cahier des charges. Cela s'est avéré compliqué, ne maîtrisant pas encore suffisamment javaFX ainsi que CSS. Voici une comparaison générale entre le modèle, et l'application finale :

### La page d'accueil:



### La page de choix et de création de projet :



## La page de l'éditeur :

The screenshot shows the KitchenApp editor interface. At the top, there are tabs for 'Cuisine Edit', 'Accueil', 'Catalogue', and 'Projets'. Below the tabs is a toolbar with icons for 'Nouveau', 'Zoomer', 'Annuler', 'Rétablir', 'Déplacer', 'Ajouter', 'Supprimer', and 'Rotater'. The main workspace is a 3D grid where a wooden cabinet is being placed. To the left, there's a sidebar for 'Mobilier et électroménagers' with categories like 'Electroménagers' (Fours, Lave-vaisselle, Micro-ondes, Réfrigérateurs & congélateurs, Planchers), 'Tables' (Tables de cuisine induction, Table de cuisine induction 2, Table de cuisine induction 3, Table de cuisine à gaz), and 'Etagères' (Plans de travail, Dessertes & îlots cuisine, Tables). On the right, there's a catalog window titled 'Catalogue' with sections for 'Catalogue de cuisine' (Electroménager: Fours, Micro-ondes, Tables de cuisson, Lave-vaisselle, Hotte, Réfrigérateurs & Congélateurs, Evier), 'Mobilier' (Tables, Etagères, Plans de travail, Dessertes & îlots de cuisine, Chaises), and 'Lumière & Décoration' (Lampes murales, Plafonniers, Plantes). A red box highlights the 'Catalogue de cuisine' section.

## Page supplémentaire : Catalogue :

The screenshot shows the KitchenApp Catalogue page. At the top, there are tabs for 'Cuisine Edit', 'Accueil', 'Projets', and 'Catalogue'. The main area is divided into three columns: 'Electroménager' (Eviers, Fours, Lave-vaisselles, Micro-ondes, Réfrigérateurs & congélateurs, Tables de cuisson), 'Mobilier' (Chaises, Dessertes & îlots de cuisine, Etagères, Plans de travail, Tables), and 'Lumière & Décoration' (Lampes murales, Plafonniers, Plantes). Below the columns, there are three images of cooktops labeled 'Table cuisson I', 'Table cuisson II', and 'Table cuisson III'.

Voici maintenant la liste des fonctionnalités. Vous y trouverez une comparaison entre les fonctionnalités du cahier des charges et celles finalement implémentées :

Fonctionnalité du cahier des charges :	Fonctionnalité réalisée :
communes au trois fenêtres :	
2 boutons en haut à droite permettant d'accéder directement à : -Accueil -Projets	3 boutons en haut à droite permettant d'accéder directement à : -Accueil -Projets -Catalogue

Fenêtre d'accueil :	
Un bouton placé au centre permettant d'accéder à la fenêtre <b>Projets</b> .	Un bouton placé au centre permettant d'accéder à la fenêtre <b>Projets</b> .
Fenêtre de choix du projet :	
<ul style="list-style-type: none"> <li>- Un bouton pour créer un nouveau projet de cuisine en choisissant un nom et des dimensions</li> </ul>	<ul style="list-style-type: none"> <li>- Un bouton pour créer un nouveau projet de cuisine en choisissant un nom et des dimensions</li> </ul>
<ul style="list-style-type: none"> <li>- La liste des projets de cuisine déjà existants, que l'utilisateur pourra faire défiler grâce aux flèches</li> </ul>	<ul style="list-style-type: none"> <li>- La liste des projets de cuisine déjà existants, que l'utilisateur pourra faire défiler grâce aux flèches</li> </ul>
Fenêtre d'édition :	
Catalogue dans la partie inférieure	Catalogue dans une nouvelle fenêtre ouverte quand on appui sur le bouton ajouter
Outil Zoom permettant de zoomer (bouton +) ou de dézoomer (bouton -)	Outil Zoom permettant de zoomer (bouton +) ou de dézoomer (bouton -)
Outil Navigation permettant de déplacer le canvas de la cuisine vers le haut ou le bas	Non réalisé
Outil annuler/rétablir (boutons) pour les actions sur le canvas	Outil annuler/rétablir (boutons) pour <u>certaines actions</u> sur le canvas
Outil déplacer (boutons)	Outil déplacer (press and drag)
Outil ajouter qui ajoute l'élément sélectionné dans le catalogue sur le canvas	Outil ajouter qui ouvre le catalogue pour pouvoir sélectionner un élément et l'ajouter sur le canvas
Outil supprimer qui supprime l'élément sélectionné	Outil supprimer qui supprime l'élément sélectionné
Outil pivoter qui modifie l'orientation d'un élément sélectionné	Outil pivoter qui modifie l'orientation d'un élément sélectionné
Chevauchement qui surligne en rouge les élément se chevauchant	Chevauchement qui surligne en rouge les élément se chevauchant

On remarque ici que l'on a omis certaines fonctionnalités du cahier des charges. La raison principale à cela est que, comme par exemple pour la fonctionnalité de navigation, qui permettait de naviguer sur le canvas d'édition une fois que l'on a zoomé dessus, ces fonctionnalités étaient trop compliquées à implémenter. Cette complexité vient la plupart du temps de choix techniques pris en amont, qui finissent par empêcher ou rendre trop complexe l'implémentation de nouvelles fonctionnalités. Il nous aurait ainsi fallu plus de temps pour revenir sur nos choix techniques afin de faciliter la création de ces nouvelles fonctionnalités.

Un autre aspect que nous n'avons pas pu implémenter consiste en la persistance des données. En effet, les projets créés puis édités sont sauvegardés lors d'une session, mais lorsque l'on ferme l'application et qu'on la ré-ouvre, les anciens projets ne sont pas sauvegardés sur la machine. La sauvegarde automatique est donc bien implémentée, mais elle n'est pas persistante.

## DIFFICULTÉS ET BUGS PERSISTANTS

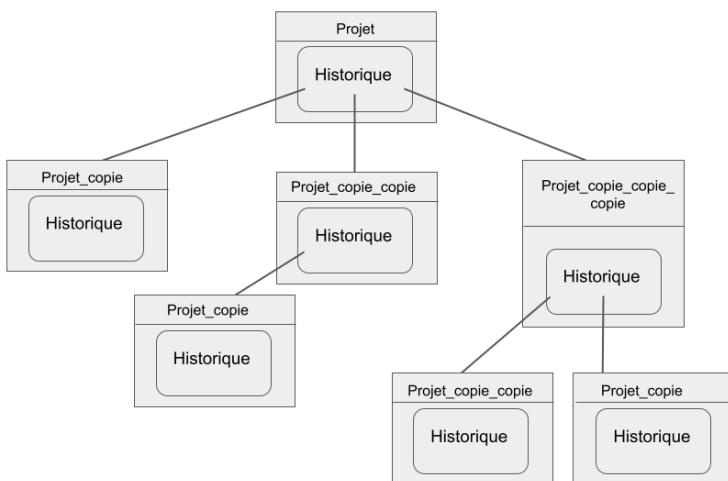
Lors de l'implémentation, nous avons rencontré certaines difficultés.

Une des premières était liée au travail collaboratif sur Github. En effet, en utilisant Eclipse sur nos deux machines, nous avions des différences au niveau de nos versions de Java et de logiciels. Il a donc fallu nous accorder sur le contenu du répertoire Github et sur les réglages des projets que nous allions créer afin de pouvoir facilement nous partager le travail. Nous avons ainsi dû découvrir les fonctionnalités de merge propres à Git par nous même (Git étant un outil largement utilisé dans le monde professionnel, cela ne nous en sera qu'enrichissant).

Une autre difficulté rencontrée était liée au SceneBuilder. En effet cet outil bien que très pratique reste néanmoins assez complexe à prendre en main, et plutôt mal conçu parfois (impossibilité de zoomer sur l'environnement de travail sous windows par exemple). Nous avons donc souvent rencontré des problèmes liés à la synchronisation entre le SceneBuilder et le fichier FXML dans Eclipse, nous faisant perdre un certain temps lorsque cela engendrait des erreurs.

Mais finalement, la plupart des difficultés rencontrées étaient d'ordre technique. La rotation des images sur un canvas qui est assez complexe à mettre en place, la sauvegarde automatique qui nous a fait revenir sur bon nombre de nos choix techniques (grandes modifications au sein des classes du package Modele), ou encore les options d'annulation et de rétablissement d'actions.

Certaines difficultés ont pu être résolues facilement, comme par exemple le fait d'empêcher l'utilisateur de déplacer les items en dehors de sa cuisine (et donc du canvas). En revanche d'autres nous ont parfois fait revoir nos choix techniques assez profondément. Par exemple, lorsque nous avons implémenté la fonctionnalité d'annulation. Cette fonctionnalité devait permettre à l'utilisateur de revenir sur une action (déplacement, rotation, ajout d'item...) qu'il venait d'effectuer. Nous avons d'abord réfléchi à sauver l'état du projet dans un attribut 'Historique' que chaque projet porterait. Ainsi, chaque projet viendrait avec une liste vide ou non d'anciennes versions, et ainsi cela permettrait d'y revenir en supprimant la version actuelle et en la remplaçant par une des versions antérieures. Cela s'est révélé être une mauvaise solution car à chaque action, on ajoutait un état du projet (donc une instance de projet entière) à l'attribut 'Historique' du projet édité. Or chaque nouvel état à sauvegarder venait déjà avec un historique, ce qui, par récursion, nous faisait ajouter une quantité de projet bien trop grande. Au bout de 8 actions différentes (déplacements, rotation, suppression etc...), l'application prenait un temps considérable à l'effectuer.



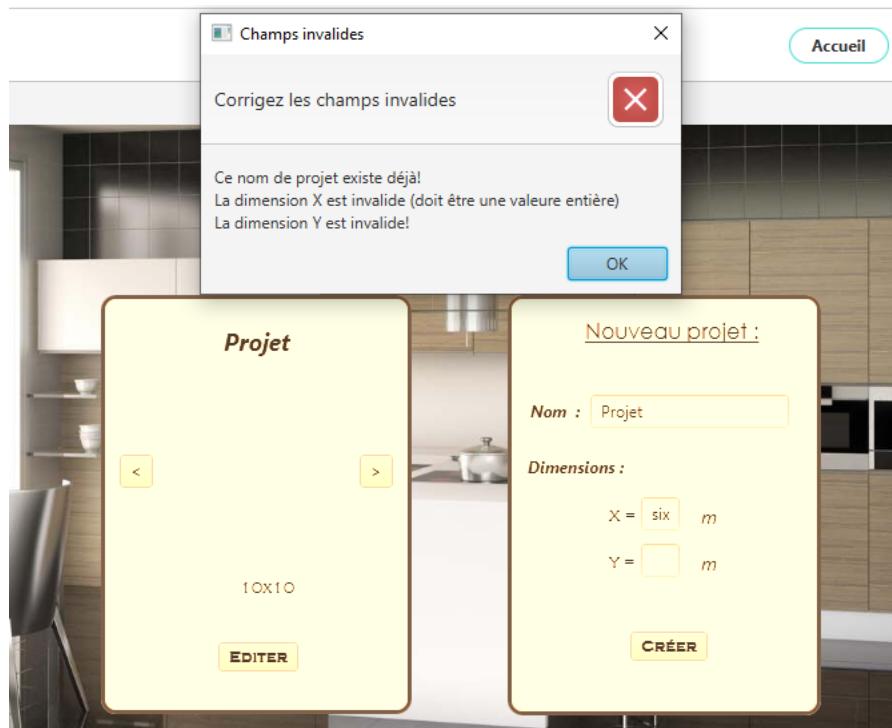
Afin de résoudre ce problème, nous avons décidé de créer au sein du contrôleur de l'éditeur, une liste d'état de projet, à laquelle viendraient s'ajouter chaque sauvegarde du projet une fois qu'une action est effectuée.

Dans notre application finale, certains problèmes restent irrésolus.

Tout d'abord, notre application n'est pas responsive : nous avons imposé une taille de fenêtre assez petite afin que l'affichage soit le même pour chaque taille d'écran, mais tout redimensionnement est un risque de manipuler l'affichage et ainsi de désorganiser l'application. Pour l'éditeur, cela est problématique car, alors que l'utilisateur devrait pouvoir voir son éditeur en grand sur la totalité de son écran, il doit redimensionner la fenêtre (par un double clique sur la barre supérieure de la fenêtre par exemple) afin que celle-ci s'adapte à sa taille d'écran.

Ensuite, certains 'bugs' persistent, notamment pour la fonctionnalité d'annulation et de rétablissement. Ces deux fonctionnalités ne prennent pas en compte la rotation des éléments, ce qui crée quelques bugs lorsque l'utilisateur pense annuler sa rotation, et que finalement ce sont tous les items qui voient leur rotation s'annuler.

En revanche, en dehors de cela, notre application reste assez robuste. Beaucoup de cas d'erreurs sont prévenus, notamment la sortie du canvas lors d'un déplacement ou d'une rotation d'items, la fonction annuler qui n'a, à l'inverse de ce que spécifie le cahier des charges, aucune limite de nombre d'action (on prévoyait de limiter le nombre d'annulation à 5 pour éviter de surcharger la mémoire de notre application, mais cela s'est révélé inutile, le système étant suffisamment rapide pour éviter les ralentissements et garder une utilisation confortable), ou encore les fautes de frappe d'utilisateur qui ne peut pas créer de projet avec pour dimension une chaîne de caractère par exemple (les dimensions doivent être des entiers). Ces cas d'erreurs sont le plus souvent indiqués par une fenêtre d'erreur qui s'affiche en décrivant le problème de manière précise.



## **CONCLUSION :**

Pour conclure, nous avons grandement apprécié réaliser ce projet. En dehors des connaissances en Java que nous avons pu retravailler, ainsi que des nouvelles techniques et librairies auxquelles nous avons été introduite comme JavaFX et le SceneBuilder, nous avons pu mettre en pratique l'apprentissage que nous avons eu sur les interfaces interactives. Ce cours nous a apporté une approche nouvelle pour la conception d'une application, et aussi sensibilisé à des aspects que nous ne soupçonnions pas. Nous avons, à travers ce projet, tenté de mettre en œuvre les techniques du cours, en respectant au maximum les conseils et restrictions relatives à la conception d'une bonne interface, mais aussi d'adopter une méthodologie de tests visant l'utilisateur et son parcours afin d'améliorer notre logiciel.

Le résultat, malgré les quelques problèmes qui persistent, nous semble cohérent avec l'énoncé, ainsi que notre vision du projet et le cahier des charges.

Afin d'aller plus loin, nous pourrions étudier la question de la persistance des données, et aussi faciliter encore l'expérience utilisateur, en ajoutant par exemple un indicateur indiquant à l'utilisateur sur quelle page il se trouve, ou sur quelle catégorie du catalogue il se trouve. Enfin, nous pourrions aussi retravailler le style de notre application qui, malgré nos efforts, pourrait encore être affiné.