Fiche explicative – Bidule Market

Baptiste CERDAN, Janos FALKE, Thomas STEINMETZ, Victor VOGT

Sommaire

Introduction

Nous vous mettons également à disposition le schéma de classes pour que vous puissiez comprendre plus facile les explications qui vont suivre.

Notre projet s'appuie sur le seed de départ et permet une adaptation facile. Nous pouvons ajouter d'une manière très simplifiée des objets, des plans, des boîtes à outils, des outils, des fenêtres de message d'information et des fenêtres de détails.

Notre objectif était de créer une structure et architecture facilement compréhensible, modifiable, adaptable et extensible.

L'application se construit via la classe principale App qui se chargera d'instancier les fenêtres à messages, les boîtes à détails, les boîtes à outils, les outils et le plan de base.

Par la suite, nous allons expliquer plus en détail les instanciations qui sont faites dans la classe principale ainsi que leurs liaisons.

Classes mères : Messagebox, Toolbox et Detailbox

Les fenêtres à messages d'information, les boîtes à détails et les boîtes à outils fonctionnent de la même façon.

Tous les 3 sont intégré dans un containeur respectif qui permet l'ajout de multiples boîtes (par exemple dans le cas des boîtes à outils « ColorToolbox », « ItemToolbox », etc. sont dans le même containeur).

En plus de cela, les classes héritent de la même classe mère selon leur type, c'est-à-dire par exemple les objets des boîtes à outils héritent d'une classe abstraite boîte à outil (Toolbox). Cela permettra aux boîtes à outils spécifiques d'avoir les mêmes propriétés.

Boîte à outils

Il y a des boîtes à outils qui sont plus complexe et particulier que les fenêtres à messages d'informations ainsi que des boîtes à détails.

Il s'agit des boîtes à outils concernant les objets (ItemToolbox) et les plans (PlanToolbox).

Dans ces deux, nous construisons une liste d'objets/plans avec des icônes qui sera affiché dans la boîte à outil respectif. Pour cela nous avons créé des classes objets/plans qui serviront uniquement pour l'affichage de la boîte à outils (ToolboxItems, ToolboxPlans).

Concernant la partie des objets, nous allons dessiner, au clic sur un objet qui se trouve dans la boîte à outils, un objet sous forme de path paper JS. Après cela, nous allons remplir les données de ce path dessiné grâce à une ItemFactory qui va créer un objet qui servira comme data.

Concernant la partie des plans, nous allons, comme pour les objets, dessiner, au clic sur un objet qui se trouve dans la boîte à outils, un plan sous forme de path paper JS. En revanche ici, compte tenu qu'il y a uniquement un seul plan de base, ce dernier sera remplacé par celui sélectionné.

Objets (Items)

Nous avons choisi de créer une classe abstraite d'objet (Item) qui sera la base pour tous les autres objets (i.e. Etagères, bloc caisses, etc.). Nous devrons alimenter des propriétés pour chaque propre objet comme son titre, sa couleur et des attributs pour savoir s'il a une taille modifiable, est suspendu ou peut être tourné.

Au moment de dessiner un objet, cette classe (selon l'objet choisi) sera créé et placé dans les données du path paper JS créé.

Par la suite, nous allons pouvoir identifier pour chaque path qui se trouve dans la variable globale du projet paper JS un objet (i.e. étagères, bloc caisses, etc.).

Tout ceci facilitera la compréhension et également l'extensibilité de l'application.

Nous pourrons ajouter des objets à la volé et devrons uniquement créer un objet qui hérite de la classe abstraite objet (Item) puis l'ajouter sa création dans la ItemFactory et finalement créer un objet avec une icône (ToolboxItem) qui sera affiché dans la boîte à outils. Cet objet dans la boîte à outils permettra ensuite de dessiner le path paper JS et prendra l'objet (Item) préalablement créé et le placera dans ses données.

Outils (Tools)

La partie d'outils (Tools) a déjà été initialisé avec le seed de départ. Nous avons décidé de garder cette architecture et d'également créer nos outils qui seront présentes dans la barre à outils par la suite. Ces classes d'outils vont tous hériter de la classe PaperTool qui permettra d'interagir avec le projet et d'avoir des évènements (souris et clavier).

Nous aurons également la possibilité de passer dans le constructeur d'un outil des boîtes à outils. Cela va faciliter l'utilisation des propres boîtes à outils à des outils comme par exemple le fait de pouvoir utiliser la fenêtre des détails d'un objet dessiné lorsque nous sommes en train d'utiliser l'outil qui permet l'ajout d'objets.

Nous avons choisi de gérer uniquement les évènements dans ces classes là pour avoir une propre utilisation à chaque outil.

Fonctions « Helpers »

Nous avons aussi créé un dossier qui regroupe des fonctions helpers qui vont nous aider à ne pas devoir réécrire plusieurs fois la même méthode quand elle est utilisée dans plusieurs classes différentes.

Nous avons pour l'instant deux classes. Une première pour pouvoir globalement traiter les couleurs des contours ainsi que des contenus des objets/plans tout comme pour les messages d'informations. Une deuxième qui servira comme aide concernant la détection des intersections qui se fait dans plusieurs outils (Tools) différents.

Commentaires / Documentation

Nous avons également mis en place des commentaires sous forme de mark down dans toutes les classes pour permettre la création facile d'une documentation. Cette documentation pourra être créer via la commande « npx typedoc ».

Conclusion

Pour conclure, nous avons mis en place un système qui est à la fois plus facile à comprendre et dont l'extensibilité n'est pas très complexe.

Nous pouvons facilement ajouter des objets, des plans, des outils, des boîtes à outils, des fenêtres de message d'information et des fenêtres de détails.