**Arc-Itecture – Projet .NET C#**

*Capocasale Romain, Freiburghaus Jonas et Moulin Vincent*

*He-Arc, INF3dlm-a*

*2019-2020*

**Introduction**

Le but d’Arc-Itecture, est de facilement pouvoir réaliser des plans architecturaux en deux dimensions qui contiennent des murs, des portes, des fenêtres, des étiquettes pour les salles et une icon qui représente la position de la caméra.

Ceci, car ce projet est utilisé en complément à notre projet d’infographie, qui à partir d’un fichier JSON réalise l’extraction du plan en 3D. Il nous est donc nécessaire d’avoir des fonctions d’importation et d’exportation de fichiers JSON. Le projet d’infographie possède certaines contraintes qu’il a fallu respecter au travers de la réalisation de ce projet, notamment le fait que lorsqu’on place une porte, il faut prévoir un trou dans le mur. Une seconde contrainte est que les murs doivent être placés parallèlement à l’abscisse et l’ordonnée.

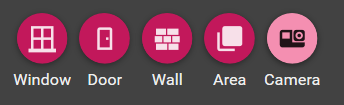
**Guide utilisateur**

L’application a pour but de permettre à un utilisateur de dessiner en 2D un plan de maison ou d’appartement. Le programme offre la possibilité d’ajouter des murs, des fenêtres, des portes ou encore de définir des zones (salle de bain, salon, …)

L’utilisateur a la possibilité de charger et sauvegarder un plan au format JSON. Ce fichier pourrait être utilisé pour notre projet d’infographie en Unity.

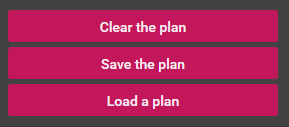
**Pour commencer :**

Lancer le programme, puis choisissez un outil. Les outils sont représentés par les icônes suivantes :

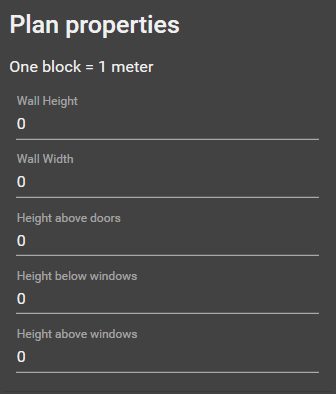


Effet de chaque outil :

* **Window** : Permet de dessiner une fenêtre sur un mur (Wall), il s’agit d’effectuer un click suivi d’un déplacement de la souris et d’un second click pour dessiner. (Raccourci : F)
* **Door** : Permet de dessiner une porte, il s’agit d’effectuer un click suivi d’un déplacement de la souris et d’un second click pour dessiner. La porte doit être dessinée entre 2 extrémités de murs (Wall). (Raccourci : D)
* **Wall** : Permet de dessiner un mur. Tant que la touche « N » n’est pas appuyée, le mur suivant débute depuis le dernier point du dernier mur dessiné. (Raccourci : D)
* **Area** : Permet de définir une zone dans une pièce : par exemple définir un salon, une salle de bain, etc… il s’agit d’effectuer un click suivant d’un déplacement de la souris et d’un second click pour dessiner. (Raccourci : A)
* **Camera** : Permet à l’aide d’un click de définir la position de la caméra (Sera utilisé comme point de départ lorsque le plan sera chargé dans notre application Unity réalisée pour le cours d’infographie). (Raccourci : C)

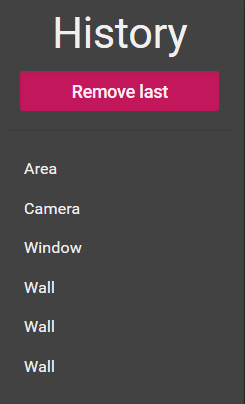
**Menu :**

* « Create a plan » : Supprime l’ensemble des objets sur le canvas de dessin (raccourci : Ctrl + N).
* « Save the plan » : Ouvre une fenêtre pour sauvegarder le plan sur son PC (raccourci : Ctrl + S).
* « Load a plan » : Ouvre une fenêtre pour charger un plan depuis son PC (raccourci : Ctrl + O).

**Propriétés du plan :**

Permet de définir les tailles des différentes propriétés 3D. Notamment :

* La hauteur des murs
* L’épaisseur des murs
* La distance entre le plafond et le haut de la porte
* La distance entre le plafond et le haut de la fenêtre et la distance entre le sol et le bas de la fenêtre

**Historique :**

Permet à un utilisateur de pouvoir revenir en arrière s’il commet une erreur.

On peut revenir en arrière soit en cliquant sur le bouton « Remove Last » ou en réalisant le raccourci clavier « Ctrl + z »

Le dernier élément est alors retiré de la liste et supprimé du canvas.

**Quadrillage :**

Le quadrillage a pour but de fournir une échelle pour l’utilisateur. Chaque carré de la grille est de taille 1m X 1m

**Packages :**

Utilisation de :

* JSON.NET de [Newtonsoft](https://www.newtonsoft.com/json), lecture et sauvegarde des plans en JSON
* MaterialDesign pour XAML

*Les packages ont été installés via le gestionnaire de paquets NuGet*

**Implémentation**

Notre programme respecte le pattern MVVM(Model-view-viewmodel). Dans notre cas :

* **La View** : correspond à notre interface graphique et contient les différents composants de notre application ainsi que le canvas.
* **Les Model :** correspond aux différents composants de notre plan. C’est-à-dire les murs, les fenêtres, les zones, la caméra et les portes. Ce sont ces modèles qui vont contenir les données tout au long de l’exécution du programme.
* **Le ViewModel :** permet de faire le lien entre l’interface graphique et les données des modèles. Cela permet de répercuter les actions de la GUI directement dans les modèles et inversement.

Notre programme implémente le pattern Command[[1]](#footnote-1). Ce pattern s’adapte bien à notre programme car il permet de changer efficacement d’outil de dessin et d’adapter la technique de dessin en fonction de l’outil. Ce pattern se décompose dans notre projet de la manière suivante :

* **Invoker :** qui permet de définir l’outil actuellement en cours d’utilisations.
* **Receiver :** qui permet de dessiner l’objet voulu sur le canvas en fonction de la commande sélectionnée.
* **IDrawCommand :** interface qui permet de définir les méthodes pour le dessin sur la canvas.
* **ConcereteCommand :** implémentation de l’interface commande, permet de redéfinir les méthodes de dessin en fonction de l’outil sélectionné.

1. <https://en.wikipedia.org/wiki/Command_pattern> [↑](#footnote-ref-1)