GLO-2004-IFT-2007

Automne 2022

Projet de session

Livrable no 2 - Modèle de conception et Architecture logique

Université Laval

Faculté des sciences et de génie

Réalisé par:

Jérémy Caron

Antoine Buquet

Anthony Laliberté

Jeammy Côté

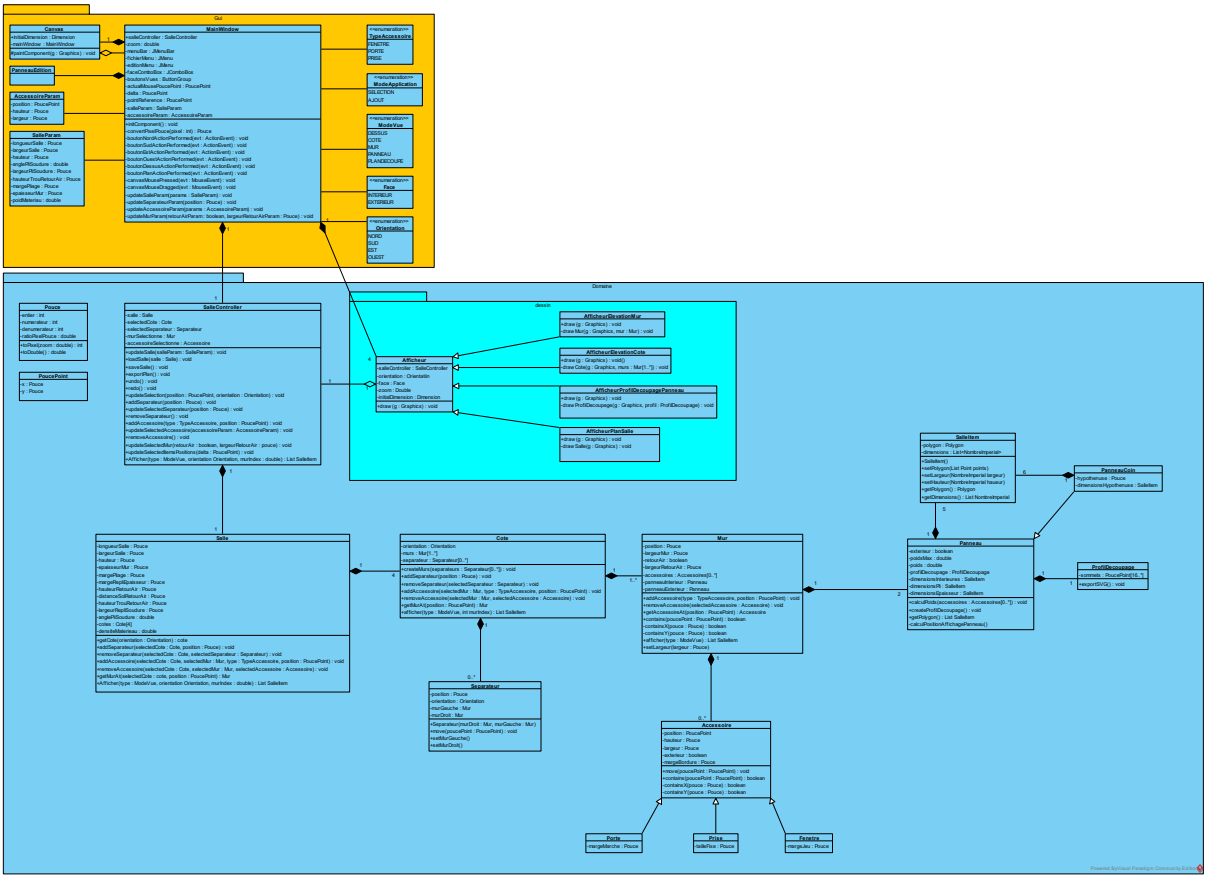
Professeurs responsables :

Jonathan Gaudreault

Marc Philippe Parent

1. Diagramme de classe de conception

1.1. Diagramme



1.2. Texte explicatif décrivant les classes et leurs relations

<u>Classes de l'interface utilisateur:</u>

TypeAccessoire: Un enum qui sert à décrire le type d'un **Accessoire**.

ModeApplication: Un enum qui sert à décrire le statut de l'application pour gérer l'édition.

ModeVue: Un enum qui sert à décrire les vues de la **Salle**.

Face: Un enum qui sert à décrire un des **Panneaux** d'un **Mur**.

Orientation: Un enum qui sert à décrire l'orientation d'un **Cote**.

MainWindow: La fenêtre principale de l'application. Elle contient notamment un **Canvas** et un **PanneauEdition**. Elle contient une référence vers le **SalleController** qu'elle utilise pour faire des modifications sur la **Salle**.

Canvas: L'objet graphique utilisé pour afficher les vues de la salle. Le traitement lié au dessin est contenu dans les objets de type **Afficheur** respectif aux vues de la salle.

PanneauEdition: La zone de la fenêtre principale où l'utilisateur pourra modifier les paramètres de la salle et autres.

AccessoireParam: L'objet contenant les paramètres de l'accessoire qui sera envoyer au **SalleController** pour mettre à jour l'accessoire

SalleParam: L'objet contenant les paramètres de la salle qui sera envoyer au **SalleController** pour mettre à jour l'accessoire

Classes du domaine:

Pouce: Un objet qui sert à manipuler des mesures impérial notamment des pouces. L'objet est utilisé un peu partout dans l'application, mais majoritairement dans le domaine.

PoucePoint: Un point dont les coordonnées x et y sont représentées sous la forme d'un objet **Pouce**.

SalleController: Un objet qui applique le principe du "contrôleur de Larman". Il contient plusieurs fonctions qui servent à modifier ou faciliter la modification de la **Salle**. Le contient une référence vers la **Salle**.

Salle: L'objet au centre de l'application. La **Salle** contient plusieurs paramètres "globaux" et un **Cote** pour chacune des **Orientations.**

Cote: Un objet qui sert à regrouper tous les Mur d'une Orientation. Le **Cote** contient une liste de "n" **Mur** et une liste de "n - 1" **Separateur** où "n" est un nombre entier plus grand que 0.

Mur: Un objet qui sert à diviser un **Cote** en plusieurs segments. Chaque **Mur** contient l'information liée à leurs dimensions, une référence vers les **Panneaux** qui le compose ainsi qu'une liste de ses **Accessoires**.

Separateur: Un objet qui représente la séparation en deux **Mur** d'un **Cote**. Il est principalement utilisé dans le traitement de l'ajout et de la modification d'un **Mur**.

Panneau: Un objet qui représente un des panneau d'un **Mur**. Il possède une référence vers son profil de découpage.

ProfilDecoupage: Un objet qui représente le profil de découpage d'un **Panneau**

Accessoire: Classe abstraite qui décrit les caractéristiques communes des **Accessoires** d'un **Mur**.

Porte: Un type d'accessoire. Contient la mesure d'une marge pour une marche. La **Porte** hérite de la classe abstraite **Accessoire**.

Prise: Un type d'accessoire. La **Prise** hérite de la classe abstraite **Accessoire**.

Fenetre: Un type d'accessoire. Contient la mesure d'une marge pour des moulures. La **Fenetre** hérite de la classe abstraite **Accessoire**.

Classes de gestion d'affichage:

Afficheur: Classe abstraite qui sert à gérer le traitement de l'affichage sur un objet **Graphics**.

AfficheurPlanSalle: Classe qui sert à faire le traitement de l'affichage de la vue en plan de la **Salle**. Elle hérite de la classe abstraite **Afficheur**.

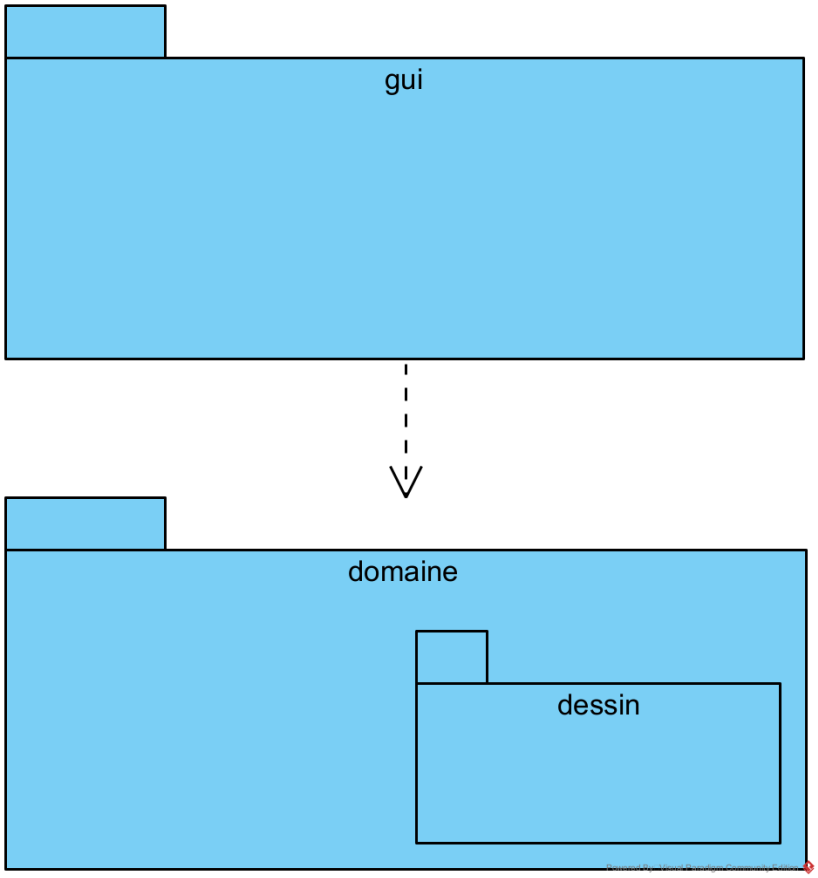
AfficheurElevationCote: Classe qui sert à faire le traitement de l'affichage de la vue en élévation d'un **Cote** de la **Salle**. Elle hérite de la classe abstraite **Afficheur**.

AfficheurElevationMur: Classe qui sert à faire le traitement de l'affichage de la vue en élévation d'un **Mur** de la **Salle**. Elle hérite de la classe abstraite **Afficheur**.

AfficheurProfileDecoupagePanneau: Classe qui sert à faire le traitement de l'affichage du plan de decoupage d'un **Panneau** de la **Salle**. Elle hérite de la classe abstraite **Afficheur**.

2. L’architecture logique

2.1. Diagramme de package



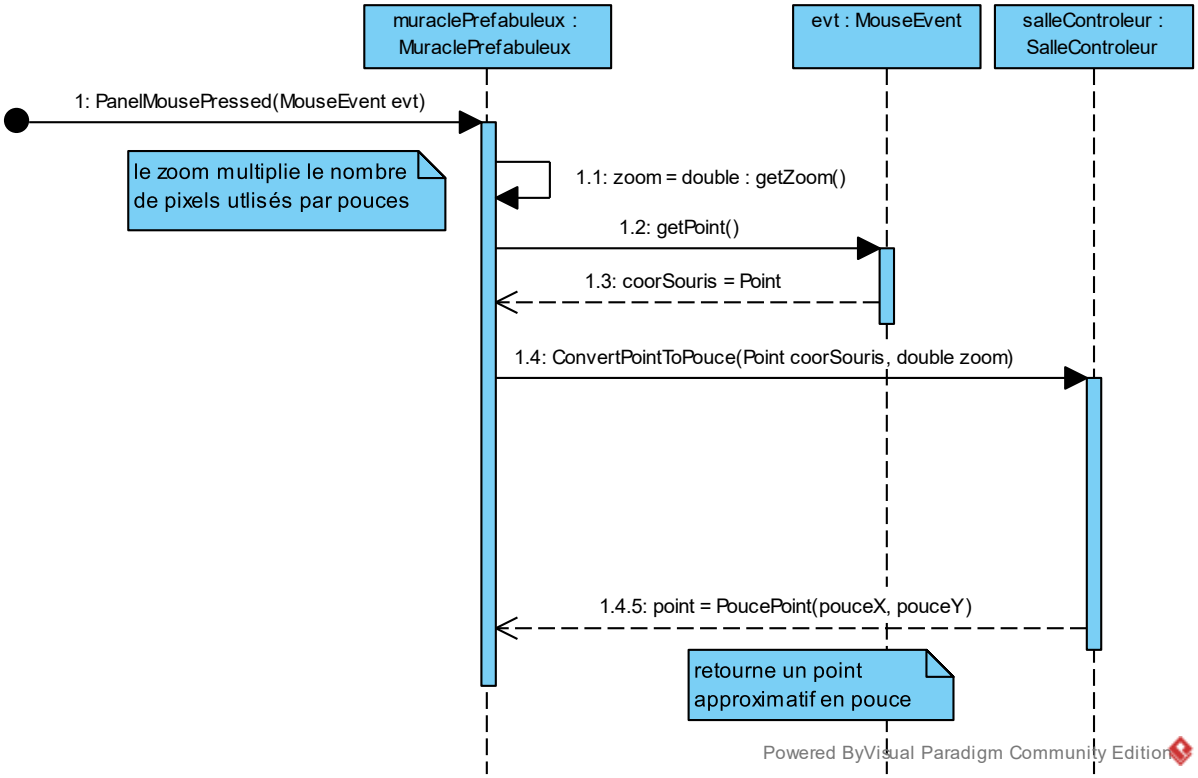
2.2. Texte explicatif

Nous avons deux packages principaux qui sont le package “gui” et le package “domaine”. Le package “gui” correspond à l’interface utilisateur, c’est avec lui que le client va interagir avec notre application. Ce package utilise ensuite notre deuxième package principal nommé “domaine”. On y retrouve toute notre logique d’affaire et nos objets correspondant à notre domaine. C’est donc lui qui s’occupe de modifier notre salle après que l’utilisateur ai fait une action sur la fenêtre de notre application. Au sein de ce package “domaine” existe un autre package portant le nom de “dessin”. C’est le package qui va contenir les différentes classes d’affichage de notre application.

3. Diagrammes de séquence de conception

3.1. Déterminer l'élément sur lequel a lieu un clic de souris dans la vue d'élévation d'un côté

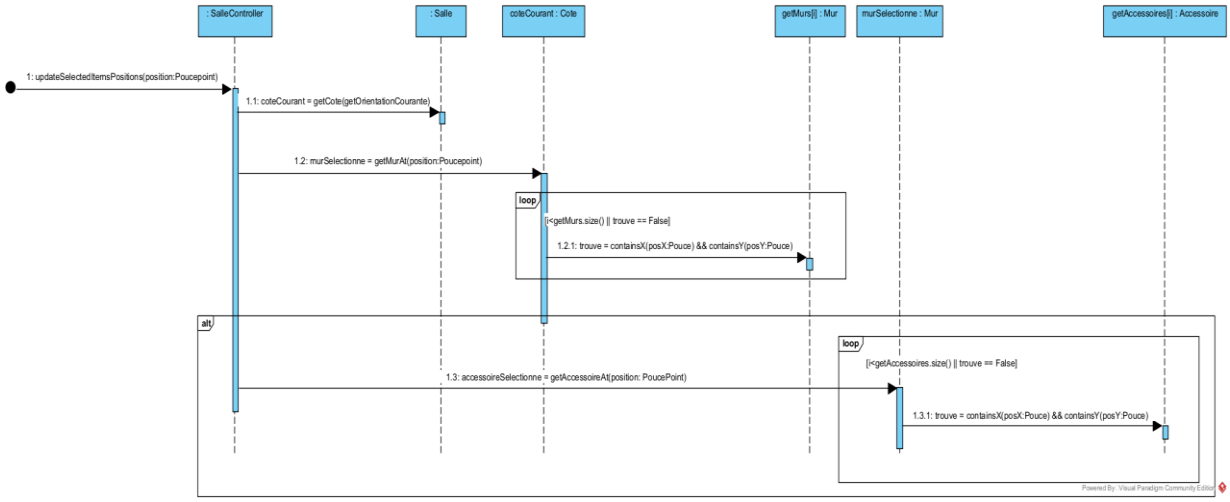
3.1.1. Premier diagramme : déterminer les coordonnées d'un clic en pouce



Texte explicatif:

Tout commence par l'événement "MousePressed" du JPanel utilisé. On obtient le zoom sélectionné dans l'interface utilisateur. À partir du paramètre de l'événement on obtient les coordonnées du point du clic en pixel. Par la suite, on demande au **SalleControleur** de convertir le point en pouce. Le SalleControleur contient une constante pour la marge entre le début du Canvas et du Cote qui seront utilisé dans le calcul du point. Le contrôleur calcul des valeurs décimales de coordonnées en pouce, puis instancie des objets de type **Pouce** avec. Le contrôleur retourne à l'événement un objet de type **PoucePoint** ayant pour coordonnées les coordonnées calculées précédemment.

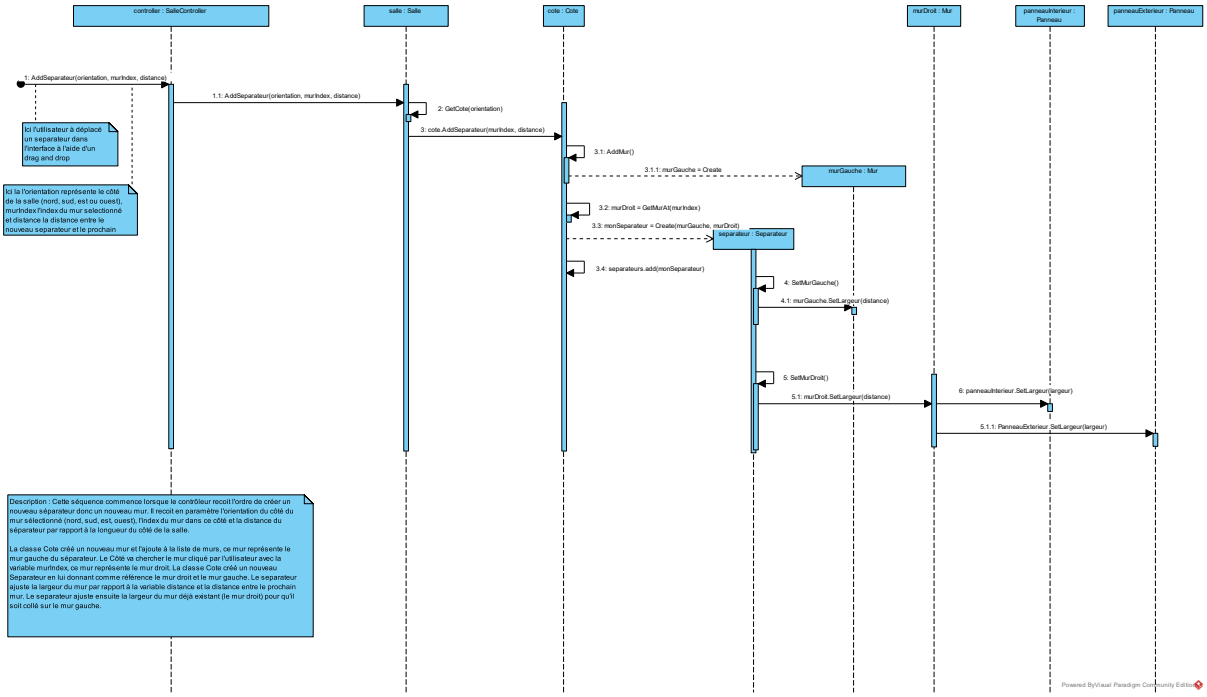
3.1.2. Deuxième diagramme : Déterminer les éléments sélectionnés grâce aux coordonnées obtenue en 3.1.1



Texte explicatif :

Le contrôleur demande à la salle d'obtenir le côté qui est affiché présentement. Nous pouvons ensuite demander à ce côté de trouvé le mur qui se trouve à la position donnée en paramètre au contrôleur avec la méthode getMurAt(). Cette méthode boucle dans tous les murs du côté pour trouver le mur qui contient cette position. Ensuite, on vérifie si ce mur contient au moins un accessoire. Si c'est le cas, on répète un processus similaire, mais cette fois-ci c'est la méthode getAccessoireAt() du mur trouvé précédemment qui est utilisée. On obtient donc l'accessoire qui contient le point position, ou Null si ce point n'est pas sur un accessoire.

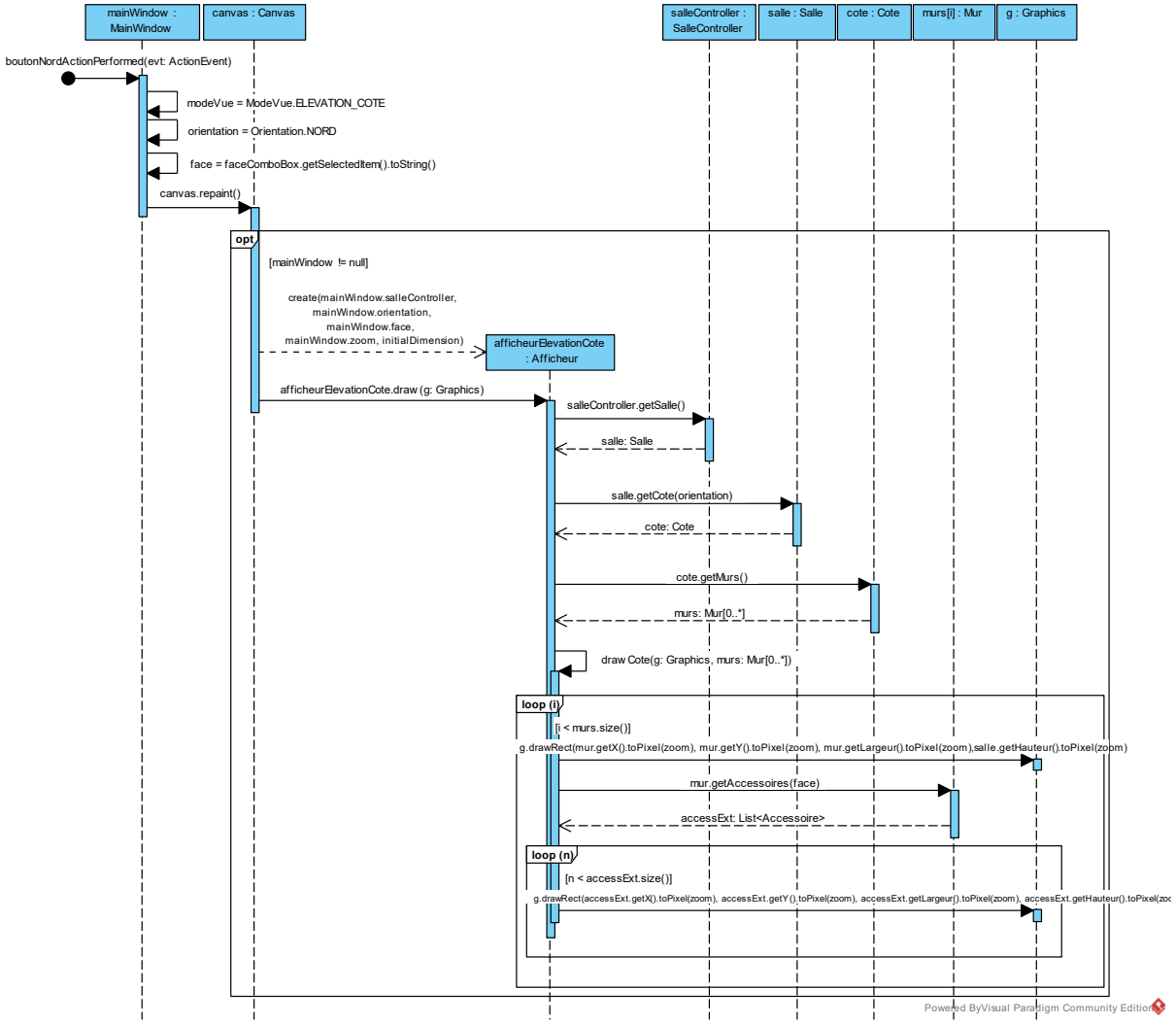
3.2. Créer un séparateur



Texte explicatif : Cette séquence commence lorsque le contrôleur reçoit l'ordre de créer un nouveau séparateur donc un nouveau mur. Il reçoit en paramètre l'orientation du côté du mur sélectionné (nord, sud, est, ouest), l'index du mur dans ce côté et la distance du séparateur par rapport à la longueur du côté de la salle.

La classe Cote créé un nouveau mur et l'ajoute à la liste de murs, ce mur représente le mur gauche du séparateur. Le Côté va chercher le mur cliqué par l'utilisateur avec la variable murIndex, ce mur représente le mur droit. La classe Cote créé un nouveau Separateur en lui donnant comme référence le mur droit et le mur gauche. Le Separateur ajuste la largeur du mur par rapport à la variable distance et la distance entre le prochain mur. Le Separateur ajuste ensuite la largeur du mur déjà existant (le mur droit) pour qu'il soit collé sur le mur gauche.

3.3. Affichage de la vue d'élévation



Texte explicatif :

Notre application est divisée en plusieurs zones (barre de menu, panel d'édition) dont une qui est consacrée à l'affichage de notre salle (c'est la zone principale de notre application). Cette zone va contenir un canvas, c'est la toile sur laquelle on va venir dessiner notre salle. Lors d'un clic sur le bouton pour passer en vue d'élévation d'un côté (sur le côté nord par exemple), cela déclenche automatiquement un appel à la méthode *boutonNordActionPerformed()* dans notre application. On met à jour certains attributs qui vont nous être utiles pour la suite. Une fois fait on appelle la méthode *repaint()* de notre canvas. Notre canvas a préalablement été créé par notre application qui s'est donné lui-même en paramètre. Il va ensuite créer la bonne instance de notre classe Afficheur en lui donnant les paramètres nécessaires et va faire appel à sa méthode *draw(g)*. L'afficheur peut donc aller chercher les différents objets du domaine à l'aide du contrôleur qui lui a été fourni. Avec ces objets et leurs différents attributs, l'afficheur est capable de dessiner sur le canvas en utilisant l'objet Graphics.

4. Pseudo-code

```
class Panneau
{
   private SalleItem dimensionsInterieurs
   private SalleItem dimensionsPli
   private SalleItem dimensionsEpaisseur
   private double inclinaisonPli = 10
   private double margeEpaisseur = 5
   private double margeVue = 50
   private bool exterieur

   public Panneau(bool exterieur){
      this.exterieur = exterieur
   }

   public List<SalleItem> GetPolygon(){
      List listSalleItems = new List<SalleItem>()
      listSalleItems.add(dimensionsInterieurs)
      listSalleItems.add(dimensionsPli)
      listSalleItems.add(dimensionsEpaisseur)
      return listSalleItems
   }

   public CalculPositionAffichagePanneau()
   {
      if (exterieur)
      {
         xMaxPrecedent = 0
         yMaxPrecedent = 0

         \\pour pli
         List<Pouce> dimensionsPli = dimensionsPli.GetDimensions()
         double largeurPli = dimensionsPli.GetDimensions()[0].toDouble()
         double hauteurPli = dimensionsPli.GetDimensions()[1].toDouble()
         List<Point> newPoints
         newPoints.add(xMaxPrecedent + margeVue,inclinaisonPli + margeVue)
         newPoints.add(largeurPli + margeVue, yMaxPrecedent + margeVue)
         newPoints.add(largeurPli + margeVue, hauteurPli + margeVue)
         newPoints.add(xMaxPrecedent + margeVue, hauteurPli - inclinaisonPli + margeVue)
         dimensionPli.SetPolygon(newPoints)
         xMaxPrecedent = largeurPli
         yMaxPrecedent = hauteurPli

         \\pour epaisseur
         List<Pouce> dimensionsEpaisseur = dimensionsEpaisseur.GetDimensions()
         double largeurEpaisseur = dimensionsEpaisseur.GetDimensions()[0].toDouble()
         double hauteurEpaisseur = dimensionsEpaisseur.GetDimensions()[1].toDouble()
         List<Point> newPoints
         newPoints.add(xMaxPrecedent + margeVue, margeEpaisseur + margeVue)
```

```
            newPoints.add(largeurEpaisseur + xMaxPrecedent + margeVue, margeEpaisseur + margeVue)
            newPoints.add(largeurEpaisseur + xMaxPrecedent + margeVue , hauteurEpaisseur + margeVue)
            newPoints.add(xMaxPrecedent + margeVue, hauteurEpaisseur + margeVue)
            dimensionsEpaisseur.SetPolygon(newPoints)
            xMaxPrecedent = largeurEpaisseur
            yMaxPrecedent = hauteurEpaisseur

            \\Ainsi de suite pour les quatres polygones du panneau...
        }
        else
        {
            \\Si c'est un panneau exterieur
        }
    }
}

class SalleItem
{
    private Polygon polygon
    private List<Double> dimensions

    public Polygon GetPolygon(){
        return polygon
    }
    public SetPolygon(List<Double> listePoints){
        polygon = new Polygon()
        foreach(point in listePoints){
            polygon.add(point.GetX(),point.GetY())
        }
    }
    public List<Double> GetDimensions(){
        return dimensions
    }
    public SetLargeur(double largeur){
        dimensions[0] = largeur
    }
    public SetHauteur(double hauteur){
        dimensions[1] = hauteur
    }
}
```

5. Gantt

Itération:	**Itération 1**	**Itération 2**	**Itération 3**	**Itération 4**
Date:	27 Septembre - 18 Octobre	18 Octobre - 8 Novembre	8 Novembre - 20 Novembre	29 Novembre - 20 Décembre
Durée:	3 semaines	3 semaines	3 semaines	3 semaines
Cas d'utilisation:	Créer une nouvelle salle Visualiser le plan de haut Editer les propriétés de la salle	Zoomer/Dézoomer Ajouter un séparateur de mur Séléctionner un séparateur Déplacer séparateur avec souris Déplacer séparateur avec propriétés Supprimer un séparateur de mur Afficher la vue de coté Changer de côté	Basculer vue intérieur/extérieur Modifier une salle Modifier un mur spécifique Ajouter un accessoire Sélectionner un accessoire Déplacer un accessoire avec propriétés Déplacer un accessoire avec souris Supprimer un accessoire	Visualiser le profil de découpage Exporter les plans en SVG Quitter Sauvegarder la salle Revenir en avant/en arrière Ouvrir une salle

6. Contribution de chacun des membres de l'équipe

Jeammy Côté :

- Contribution au diagramme de classe de conception
- Diagramme de séquence de conception 3.2 et texte explicatif
- Pseudo-code pour l'étape 4

Antoine Buquet :

- Diagramme de classe de conception et contribution aux textes explicatifs
- Diagramme de package et texte explicatif
- Diagramme de séquence de conception 3.3 et texte explicatif
- Contribution à l'interface

Anthony Laliberté :

- Contribution diagramme de classe de conception
- Texte explicatif diagramme de classe de conception
- Diagramme de séquence 3.1.1 et texte explicatif

Jérémy Caron :

- Mise à jour du Gantt
- Création du squelette de l'interface
- Diagramme de séquence 3.1.2 et texte explicatif
- Ajustement du diagramme de classe