Génie logiciel orienté objet

GLO-2004

Livrable 2

Projet de session

Présenté à :

M. Jonathan Gaudreault

M. Marc Philippe Parent

Préparé par :

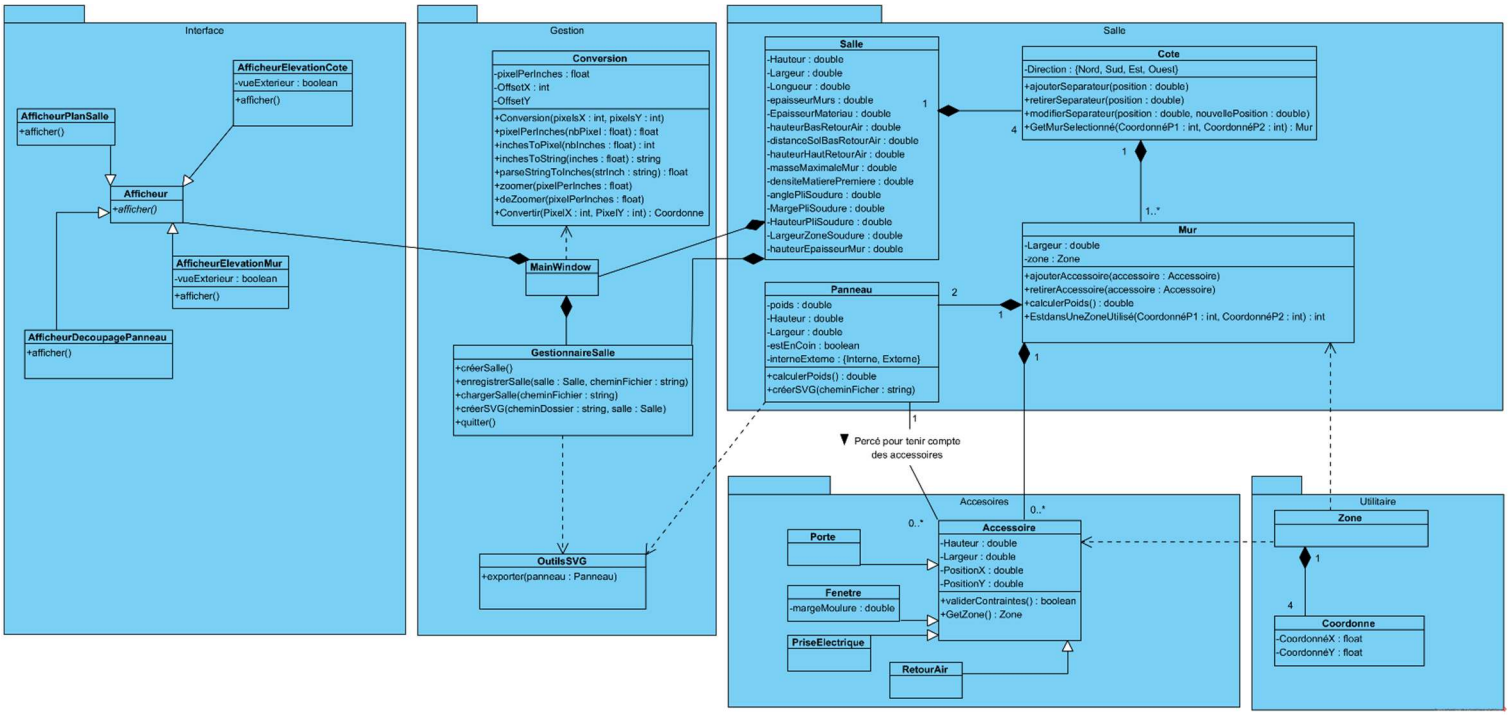
Roxane Dionne – 536 919 108

Alexandre Dicaire – 536 991 773

Bryan Emond Blais – 536 916 138

Hamza Hajjam – 536 998 165

Diagramme de classe de conception



Le « package » interface comporte une classe *Afficheur*, qui est parent à quatre autres classes. Ceux-ci vont nous permettre d'afficher différents éléments de la salle, dont les panneaux de manière individuelles, les murs, le découpage des panneaux ainsi qu'une vue de plan.

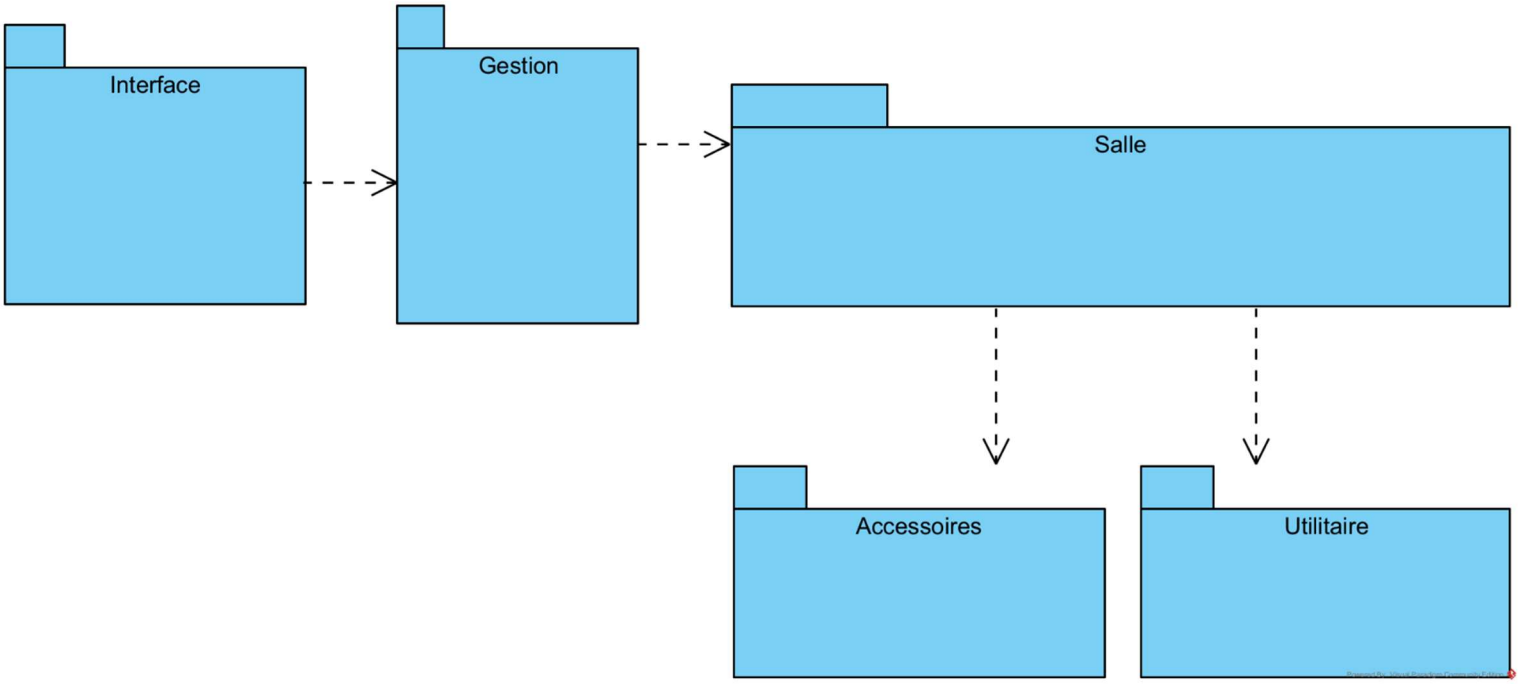
Le « *package* » *Gestion* est le « *package »* principale du projet. Celui-ci contient la fenêtre principale ainsi qui trois autres classes. La classe *Conversion* va nous permettre d'exécuter la plupart de nos calculs à l'intérieur de celle-ci. Par exemple, la conversion de pixels vers les pouces (vice-versa) ainsi que les fonctions pour zoomer. La classe de *GestionnaireSalle* nous permettra d'interagir avec les dimensions et les éléments de la salle. De plus, elle nous permettra de créer une nouvelle salle, d'enregistrer une salle existante, de charger une salle existante et de créer un fichier SVG pour le découpage.

Le « *package »* Salle comporte quatre classes. La classe panneau va avoir les dimensions principales de celle-ci (hauteur, largeur) ainsi que le type de panneaux, sont poids, s'il est en coin et une liste d'accessoires. La classe panneau va aussi nous être utile lors de la création du fichier SVG. La classe *mur* va comprendre deux panneaux dont un extérieur et un intérieur. Cette classe va avoir les fonctions pour l'ajout d'accessoires, la suppression d'accessoires et si l'accessoire ajouter est dans une zone déjà utilisé. La classe *Cote* définie un coté par une cardinalité (nord, sud, est, ouest). Les côtés vont avoir une liste de murs unique dont ceux-ci vont être séparés par des séparateurs. Les fonctions pour l'ajout, la suppression et la modification des séparateurs vont et contenue dans cette classe. La classe Salle contient les informations générales de la pièce ainsi que des informations sur la masse de la salle, sur plie de soudure et sur les retours d'aire.

Le « package » Accessoire comporte la classe parente Accessoire qui hérite sur les classes Porte, Fenêtre, Prises électriques, retours d'air. Chaque classe enfants va avoir des dimensions (hauteur, largeur) ainsi que leur position relative à l'intérieur du panneau. La classe Fenêtre va contenir la marge de moulure de plus que les autres classes.

Le « package » Zone va nous permettre de définir une zone lors d'un click de la souris. Les zones vont être définies par les murs et les accessoires. L'objet *zone* va avoir une fonction qui va retourner un objet mur, accessoire ou nulle lorsque le click est à l'intérieur ou à l'extérieur d'une zone.

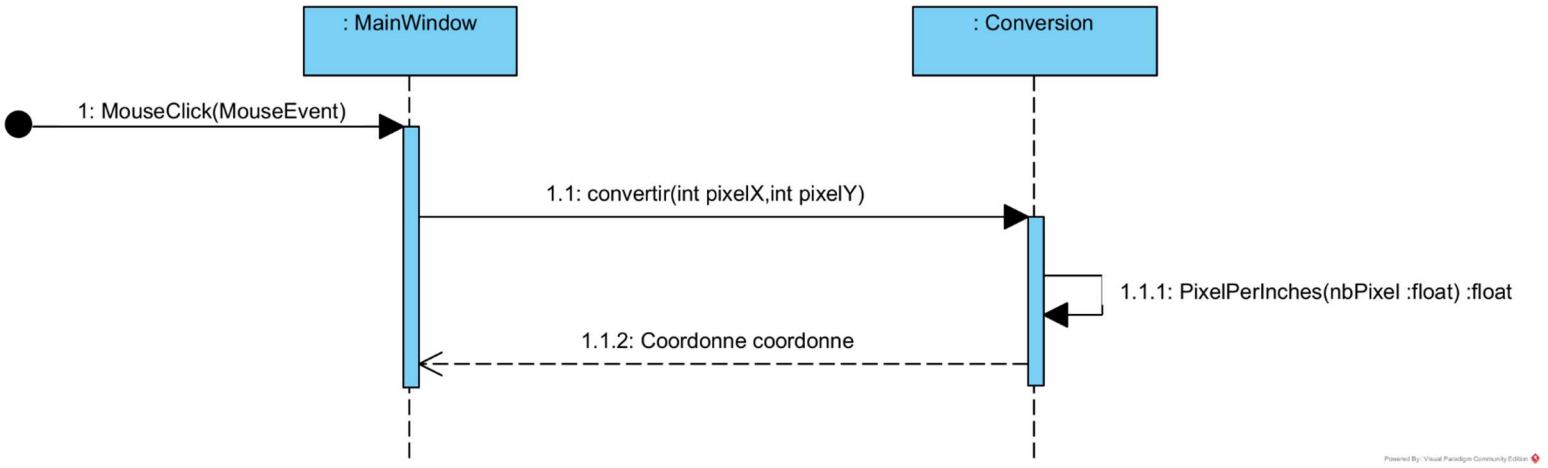
L'architecture logique



Ce diagramme représente l'architecture globale de notre logiciel. Nous avons cinq « package » qui distinguent les différentes classes. Le « package » interface représente toutes les classes servant à l'interface de notre logiciel. Le « package » gestion, de son côté, sert de pont entre les classes techniques et les classes visuelles. Les «*package» salle, accessoire* et *utilitaire* servent quant à eux, à définir les éléments techniques nécessaires au logiciel.

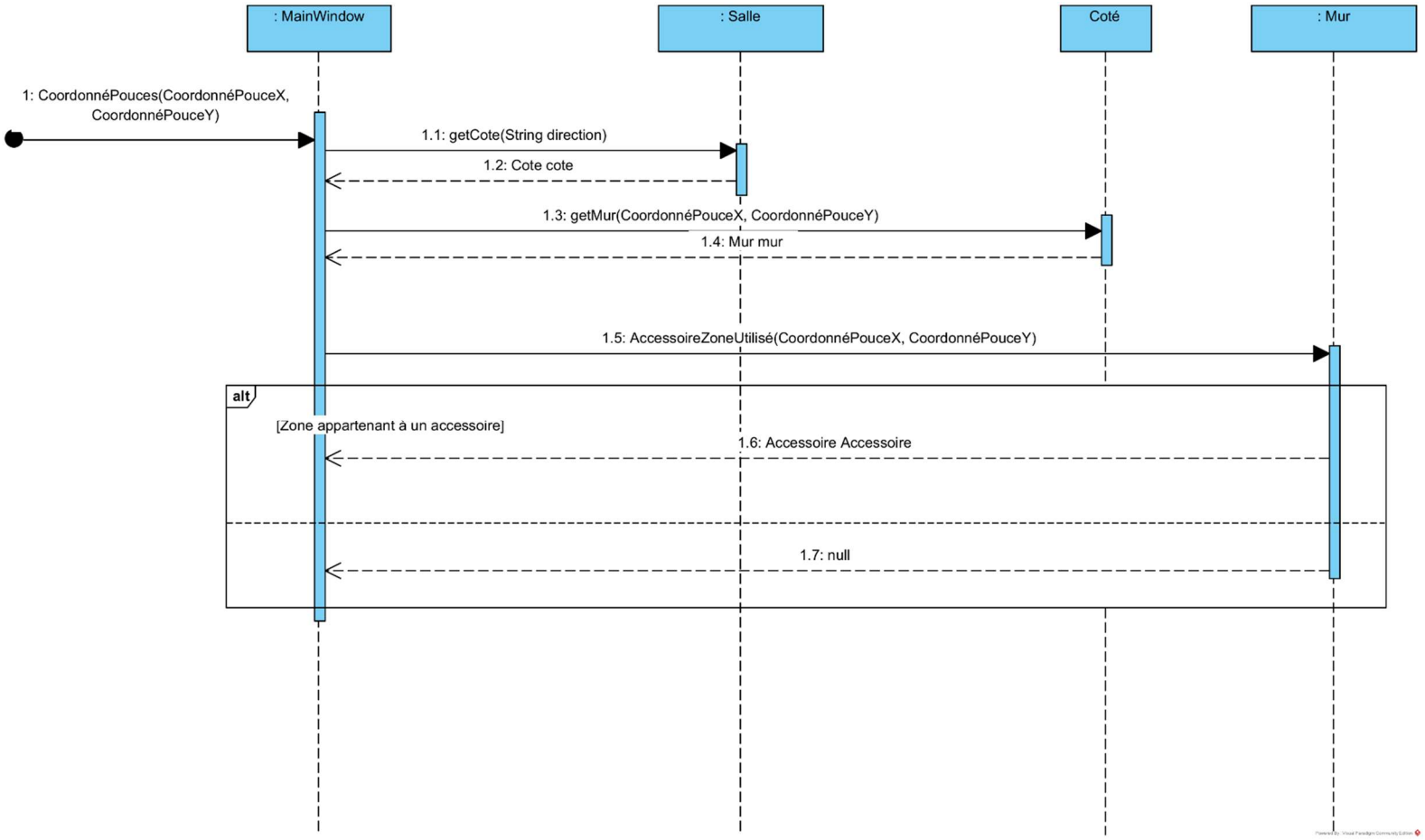
Diagramme de séquence de conception

3.1.1



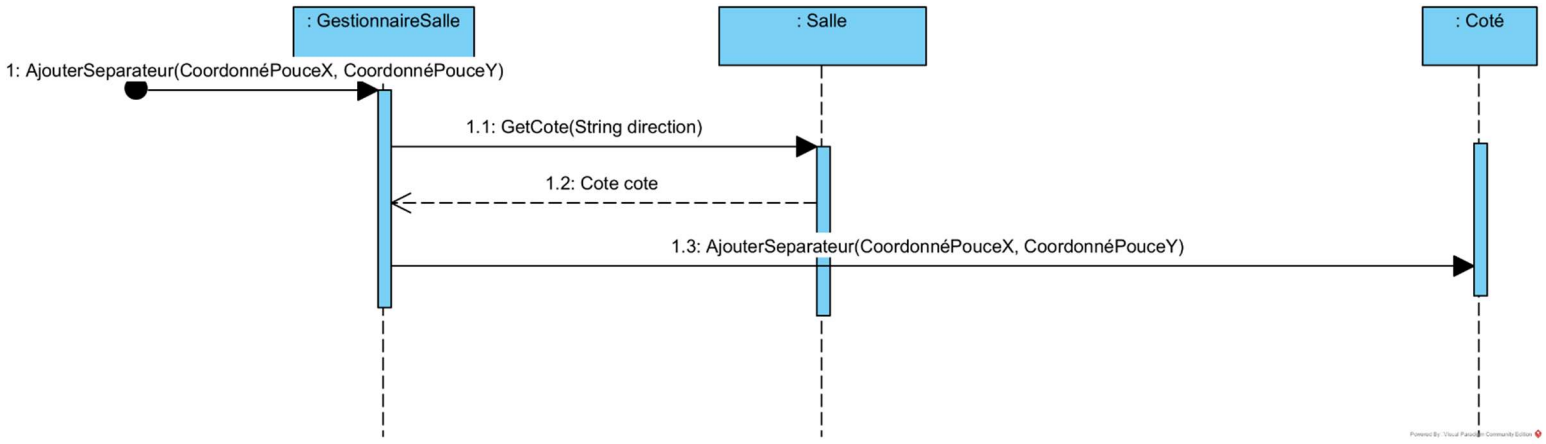
Ce premier diagramme de séquence de conception représente le travail réalisé pour obtenir l'emplacement d'un clic sur notre interface. Le clic déclenche un *mouseEvent* qui est envoyé à la classe *mainWindow* pour le gérer. Celui-ci va donc convertir les pixels en pouces par la fonction convertir, qui fait appel à la classe Conversion. Pour convertir les pixels en pouce, la classe Conversion a se sert de *PixelPerInches.* Cette fonction détermine le nombre de pixels qui est nécessaire pour former un pouce et renvoie les coordonnés du clic par le fait même.

3.1.2



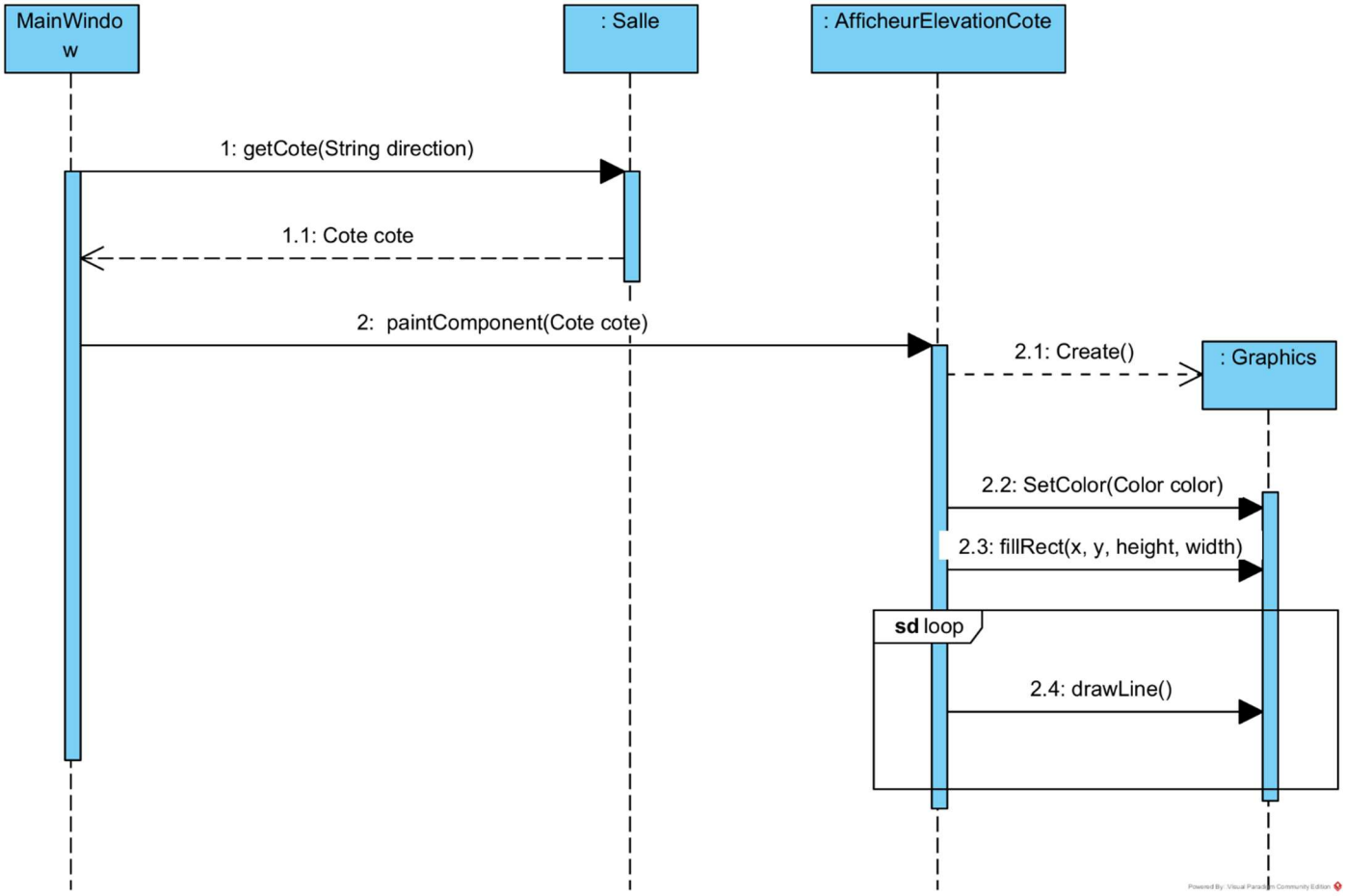
Ce deuxième diagramme de séquence de conception démontre le travail effectué pour déterminer sur quel élément l'utilisateur a cliqué. On commence donc par l'envoi des coordonnées en pouce au contrôleur *MainWindow*. Le contrôleur demande donc à la classe *Salle* de lui retourner un côté sur lequel l'utilisateur était dans sa fenêtre. Ensuite, selon les coordonnés, le contrôleur demande à la classe *Cote* de lui retourner le mur dans lequel l'utilisateur as cliquer. Enfin, le contrôleur cherche à déterminer si l'utilisateur a cliqué sur un accessoire et si c'est le cas, de quel accessoire il s'agit. Pour cela, Il appelle la boucle *AccessoireZoneUtilise* dans la classe Mur. Cette classe va ensuite faire un appel if/else et retourner un accessoire dans la zone cliqué s'il y a lui.

3.2



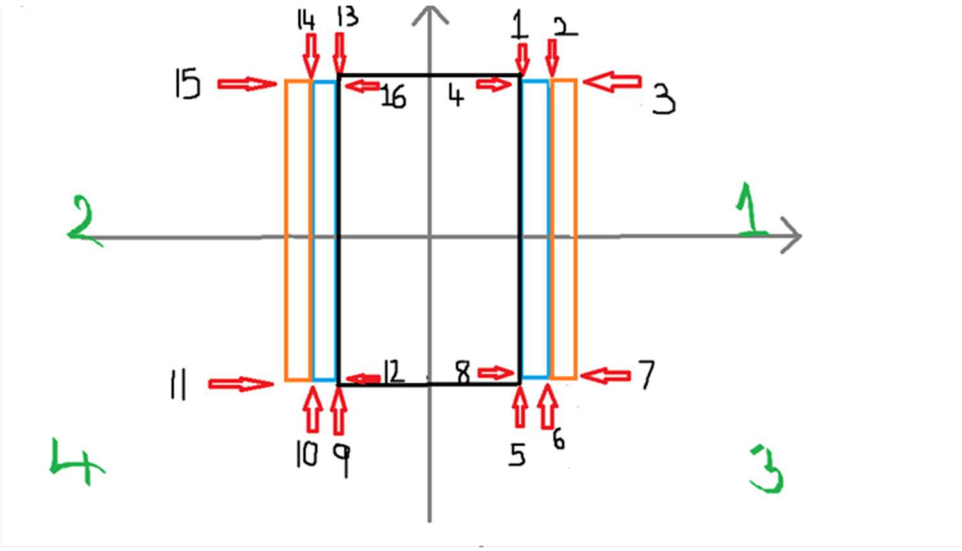
Ce troisième diagramme de séquence de conception représente le travail du logiciel pour créer un séparateur sur un mur. L'utilisateur commence par déclencher la fonction *AjouterSéparateur* en demandant un ajout. La classe *GestionnaireSalle* fait donc appel à la salle pour obtenir le côté. Elle renvoie donc le côté en cours d'utilisation par l'utilisateur. Puis, Le *GestionnaireSalle* ajoute un séparateur en le demandant à l'objet *Cote*. Le tout en prenant compte des positions X et Y désirés en pouces.

3.3.



Ce dernier diagramme de séquence de conception représente le travail du logiciel pour afficher une vue en élévation d'un côté externe réalisé. La classe « *MainWindow* » commence par appeler la classe *Salle* par la fonction « *getCote* » pour obtenir le côté étant sélectionné par l'utilisateur. Ensuite la classe Mainwindow appelle la classe *AfficheurElevationCote* par la fonction « *paintComponen*t » de swing qui crée à la de la classe « *graphics* » la page de l'élévation demandée. Elle va aussi décider de la couleur et de d'autres éléments pour créer la page.

Algorithme en pseudo code




```
Classe OutilsSVG

{

private Salle salle;

        // Le premier int représente L'axe des x et le deuxième celui des y

private List<Point>  coordonnees;


Public OutilsSVG(Salle s) {

salle = s;

coordonnees = new List()

        }


        Public  List< Point >  GenererCoordonneesUnPanneau {

        double EpaisseurMur = salle.epaisseurMur

        double hauteur = salle.hauteur

        double largeur = salle.cote[0].mur[1].largeur

        // Données zone 1 {

        Point point1 = new Point()
```

```
Point point2 = new Point()

Point point3 = new Point()

Point point4 = new Point()


point1.X = largeur/2

Point1.Y = hauteur/2


point2.X = largeur/2 + EpaisseurMur + salle.plieSoudure

point2.Y = salle.hauteurEpaisseurMur /2


point3.X = largeur/2 + salle.plieSoudure  +  EpaisseurMur +
salle.largeurZoneSoudure + salle.plieSoudure

        point3.Y = salle.hauteurEpaisseurMur /2


point4.X = largeur/2

point4.Y = salle.hauteurEpaisseurMur /2


// Données zone 1

coordonnees.add(point1)

coordonnees.add(point2)

coordonnees.add(point3)

coordonnees.add(point4)


// Données zone 2

// point 15

coordonnees.add(new Point(point3.X *-1, point3.Y ))

// point 14

coordonnees.add(new Point(point2.X *-1, point2.Y ))

// point 13
```

```
coordonnees.add(new Point(point1.X *-1, point1.Y ))
// point 16
coordonnees.add(new Point(point4.X *-1, point4.Y ))

// Données zone 3
// point 7
coordonnees.add(new Point(point3.X, point3.Y *-1))
// point 6
coordonnees.add(new Point (point2.X, point2.Y *-1))
// point 5
coordonnees.add(new Point (point1.X, point1.Y *-1))
// point 8
coordonnees.add(new Point(point4.X, point4.Y *-1))

// Données zone 4
// point 11
coordonnees.add(new Point (point3.X *-1, point3.Y *-1))
// point 10
coordonnees.add(new Point (point2.X *-1, point2.Y *-1))
// point 9
coordonnees.add(new Point (point1.X *-1, point1.Y *-1))
// point 12
coordonnees.add(new Point (point4.X *-1, point4.Y *-1))

// S'il faut que les sommets soient positifs il suffit d'additionner point1.Y à toutes les
// données Y et point3.X à toutes les données X
// for (int i = 0; i < coordonnees.length; i++)
                {
      coordonnees[i].Y += point1.Y
```

```
                coordonnees[i].X += point3.X

                        }
            //
            return coordonnees
            }

}
```

Plan de travail Gant mis à jour

Itération 1	Itération 2	Itération 3	Itération 4
3 semaines (2022-09-28 à 2022-10-18)	**3 semaines (2022-10-19 à 2022-11-08)**	**3 semaines (2022-11-09 à 2022-11-29)**	**3 semaines (2022-11-30 à 2022-12-20)**
diagramme de classe de conception			
Architecture logique			
premier essaie pseudo-code			
diagramme séquence			
interface			
	creation nouvelle salle		
	visualisation salle en plan		
	visualisation cote		
	Changer la vue		
	Gestion Zoom		
	definir, afficher, voir, sélectionner, Mur		
	définir, afficher, voir, sélectionner, modifier Séparateur		
	definir, afficher, voir, sélectionner, Accessoire		
		Support valeur numérique par default	
		Édition valeur numérique par défault	
		Exporter un document	
		editer propriétés salle	
			Charger une salle
			Enregistrer une salle
			Déplacement objet avec souris
			Afficher la vue de coupe
			Afficher la grille d'aide au positionnement
			Tests et correction de bugs

Détaillant de la contribution de chacun des membres de l'équipe

Tâche	**Membres**
Diagramme de classe de conception: diagramme	Roxane, Hamza, Alexandre, Bryan
Diagramme de classe de conception: texte	Bryan
L'architecture logique: diagramme	Roxane, Hamza, Alexandre, Bryan
L'architecture logique: texte	Roxane
Diagramme de séquence de conception: diagramme	Roxane, Hamza, Alexandre, Bryan
Diagramme de séquence de conception : texte	Roxane
Algorithme en pseudo code	Hamza
Plan de travail Gant mis à jour	Roxane
Version fonctionnelle du squelette de l'interface utilisateur	Alexandre