Livrable 2

ChalCLT

Équipe 21

Jérémy Doiron (536 895 119)

Rihab Assabar (111 261 125)

Nadir Berrezouk (536 910 612)

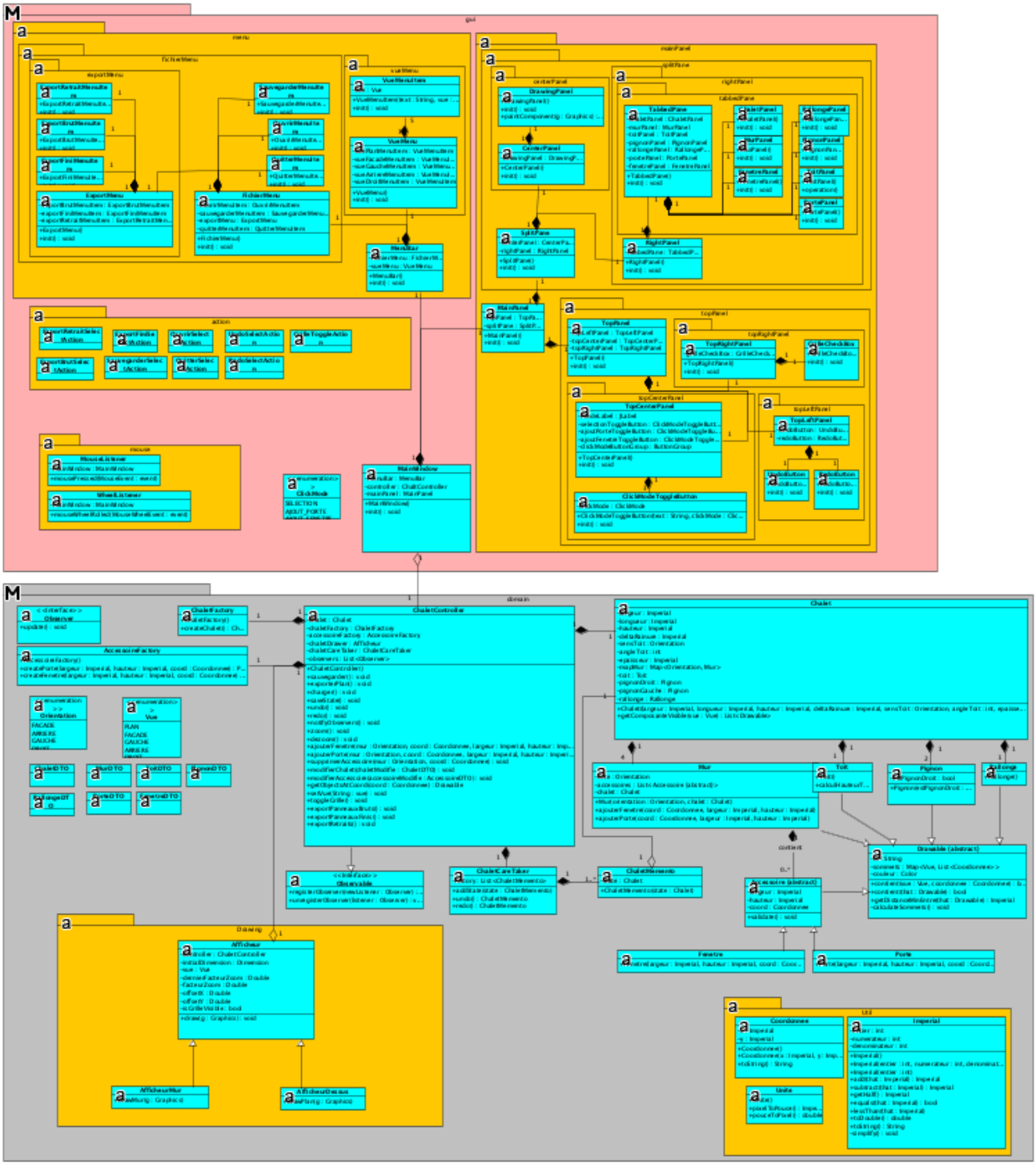
Anass El Hallaoui (536 978 272)

Ziyad Bouazara (536 960 780)

Dans le cadre du cours GLO-2004

Remise le 17 octobre 2023

1. Diagramme de classes de conception



1.1 Description des classes

1.1.1 Package domain

Le package domain s'occupe de la création et le maintien du Chalet en plus du "Drawing" de celui-ci.

1.1.1.1 ChaletController

L'élément central du package domain et le contrôleur du chalet. Il s'agit d'un contrôleur de Larman qui contient les classes ChaletDrawer, ChaletCareTaker et Chalet.

1.1.1.2 Chalet

La classe Chalet contient les paramètres soit la largeur, longueur, hauteur, delta rainure, angle, orientation et epaisseur. De plus, elle contient une map des murs du chalet ainsi que les objets qui composent les panneaux du toit. Chalet est composé de 4 Murs, 2 Pignons, 1 Rallonge et 1 Toit.

1.1.1.3 Drawable

La classe Drawable est une classe abstraite qui est obligatoirement soit un Mur, Toit, Pignon, Accessoire ou Rallonge. Elle représente tous les objets qui sont désirables. Elle contient une map des sommets des différents panneaux selon la vue.

1.1.1.4 Mur

La classe Mur, qui est une spécification de Drawable, contient la liste des accessoires contenus dans les murs ainsi qu'un attribut cote pour savoir de quel côté il s'agit (Façade, Arrière, Gauche ou Droit). Elle contient également une liste des accessoires qu'elle contient.

1.1.1.5 Toit

La classe Toit, qui est une spécification de Drawable, contient les méthodes nécessaires pour calculer la hauteur du toit.

1.1.1.6 Pignon

La classe Pignon, qui est une spécification de Drawable, utilise un booléen afin de déterminer s'il s'agit du pignon droit ou gauche.

1.1.1.7 Rallonge

La classe Rallonge est une spécification de Drawable.

1.1.1.8 Accessoire

La classe abstraite Accessoire, qui est une composition de Mur, contient les paramètres d'un accessoire soit la largeur, hauteur et coordonnée (objet Coordonnee, voir domain.Util.Coordonnee). Un Accessoire est obligatoirement une Porte ou une Fenetre.

1.1.1.9 Porte

La classe Porte est une spécification d'Accessoire et ne prend pas de position en y en paramètre contrairement à Fenetre.

1.1.1.10 Fenetre

La classe Fenêtre est une spécification d'Accessoire.

1.1.1.11 ChaletCareTaker, ChaletMemento

Ces classes servent à l'implémentation du undo/redo. ChaletMemento conserve le state avec un objet Chalet et ChaletCareTaker utilise un attribut history pour stocker dans une liste les objets ChaletMemento. Bref, ChaletCareTaker est composé de ChaletMemento's.

1.1.1.12 Observer

Cette classe se fait notifier par le contrôleur lors de changement et s'actualise par la suite.

1.1.1.13 Observable
Cette classe permet de s'abonner au contrôleur.

1.1.1.14 AccessoireFactory
Cette classe permet d'instancier des accessoires.

1.1.1.15 ChaletDTO, MurDTO, ToitDTO, RallongeDTO, PignonDTO, FenetreDTO, PorteDTO
Les classes DTO sont chacune basées sur la classe originale avec leurs attributs public. Le contrôleur retourne les DTO au gui afin de ne pas retourner d'objet du domaine complexe.

1.1.1.16 ChaletFactory
Cette classe permet d'instancier un chalet.

1.1.2 Package domain.Util
Ce package consitute les utilitaires. Il se trouve dans le package domain.

1.1.2.1 Coordonnee
La classe Coordonnee stocke les coordonnées en x et en y.

1.1.2.2 Imperial
La classe Imperial contient l'entier, le numérateur et le dénominateur des nombres impériaux.

1.1.2.3 Unite
La classe Unite permet de faire la conversion de pixels à pouces ou encore de pouces à pixels.

1.1.3 Package domain.Drawing
Contient l'affichage et les méthodes pour dessiner.

1.1.3.1 ChaletDrawer
ChaletDrawer est une généralisation d'AfficheurMur et AfficheurDessus. Elle Contient ChaletController, les dimensions initiales, la vue et l'orientation du Chalet.

1.1.3.2 AfficheurMur et AfficheurDessus
Ces deux classes possèdent chacune une méthode draw qui utilise java.awt.Graphics pour dessiner l'élément voulu.

1.1.4 Package gui
Le package gui contient les outils qui permettent de réaliser l'affichage de l'application.

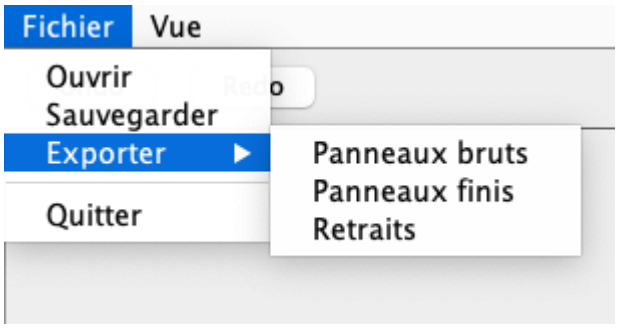
1.1.4.1 MainWindow
La classe MainWindow est le centre du package. MainWindow se subdivise en deux packages, soit "menu" et "mainPanel". C'est grâce à une instance de ChaletController contenu comme attribut dans MainWindow que toutes les composantes de l'interface utilisateur pourront accéder et modifier les propriétés du Chalet. Il est important de noter que toutes communications entre le gui et le domain sont fait uniquement entre MainWindow et ChaletController.

1.1.5 Package gui.menu
Le package gui.menu contient les classes pour effectuer l'export et manipuler les vues.

1.1.5.1 MenuBar
Cette classe contient les objets qui font référence aux différents types de menu, soit le VueMenu et le FichierMenu.

1.1.5.2 FichierMenu

Cette classe est composée de quatre composantes distinctes du menu, soit un OuvrirMenuItem, SauvegarderMenuItem, QuitterMenuItem et ExportMenu. Voici la représentation de ce menu dans l'interface graphique :



1.1.5.3 ExportMenu

Cette classe est composée d'un ExportFiniMenuItem, ExportBrutMenuItem et ExportRetraitMenuItem.

1.1.5.4 ExportFiniMenuItem

Permet l'exportation des panneaux finis.

1.1.5.5 ExportBrutMenuItem

Permet l'exportation des panneaux bruts.

1.1.5.6 ExportRetraitMenuItem

Permet l'exportation des retraits des panneaux.

1.1.5.7 QuitterMenuItem

Permet de quitter l'application

1.1.5.8 OuvrirMenuItem

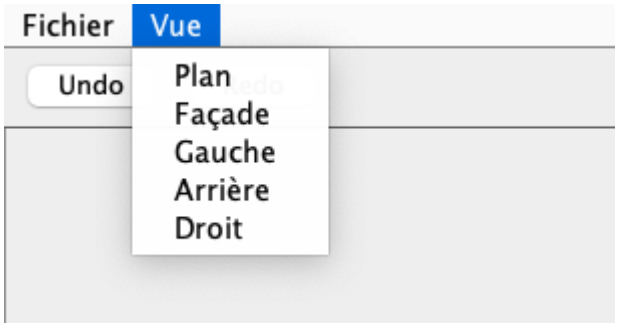
Permet d'ouvrir et naviguer à travers les fichiers.

1.1.5.9 SauvegarderMenuItem

Permet de sauvegarder le travail.

1.1.5.10 VueMenu

Cette classe est composée de cinq VueMenuItem qui représentent les 5 vues soit : Plan, Façade, Arrière, Gauche et Droit. Elle permet donc de choisir la vue du Chalet. Voici la représentation de ce menu dans l'interface graphique :



1.1.5.11 VueMenuItem

Cette classe est une composition de VueMenu. Elle contient l'objet Vue.

1.1.6 Package gui.mainPanel

Le package gui.mainPanel contient les différentes classes des panels.

1.1.6.1 MainPanel

Cette classe est composée d'un TopPanel et d'un SplitPanel.

1.1.6.2 TopPanel

Cette classe est composée de 3 panels différents, soit un TopRightPanel, TopLeftPanel et TopCenterPanel. Voici la représentation de ce panel dans l'interface graphique:

1.1.6.3 TopRightPanel

Cette classe est composée d'un objet GrilleCheckBox

1.1.6.4 GrilleCheckBox

Cette classe permet de choisir si on affiche la grille ou non.

1.1.6.5 TopLeftPanel

Cette classe est composée d'un objet UndoButton et d'un RedoButton.

1.1.6.6 UndoButton et RedoButton

Ces classes permettent de déclencher le undo et redo.

1.1.6.7 TopCenterPanel

Cette classe est composée de trois objets ClickModeToggleButton et permet de choisir le mode soit : Sélection, Ajout porte, Ajout fenêtre.

1.1.6.8 ClickModeToggleButton

Cette classe permet de détecter si le bouton est cliqué.

1.1.6.9 SplitPane

Cette classe est composée de deux panels différents, soit un CenterPanel et RightPanel. Voici la représentation de ce panel dans l'interface graphique :

1.1.6.10 RightPanel

Cette classe est composée d'un TabbedPanel

1.1.6.11 TabbedPane

Cette classe est composée d'une instance de chacune de ces classes : ChaletPanel, MurPanel, FenetrePanel, RallongePanel, PignonPanel, ToitPanel et PortePanel.

1.1.6.12 ChaletPanel, MurPanel, FenetrePanel, RallongePanel, PignonPanel, ToitPanel et PortePanel

Ces classes représentent les composantes du RightPanel.

1.1.6.13 CenterPanel

Cette classe est composée d'un DrawingPanel.

1.1.6.14 DrawingPanel

La classe DrawingPanel étend la classe javax.swing.JPanel. Elle permet de dessiner des images à l'aide de l'objet java.awt.Graphics.

1.1.7 Package gui.mouse

Ce package contient les différentes classes qui permettent de traquer les actions faites par la souris de l'utilisateur telle que le fait de cliquer et d'utiliser la roulette.

1.1.7.1 MouseListener

La classe MouseListener étend java.awt.event.MouseAdapter. Cette classe permet de capturer un java.awt.event.MouseEvent, contenant les informations de position du clic sur la fenêtre et même la position relative à l'écran de l'utilisateur. C'est entre autres à l'aide de ces mécanismes que nous faisons la transformation d'une position en pixels vers des pouces.

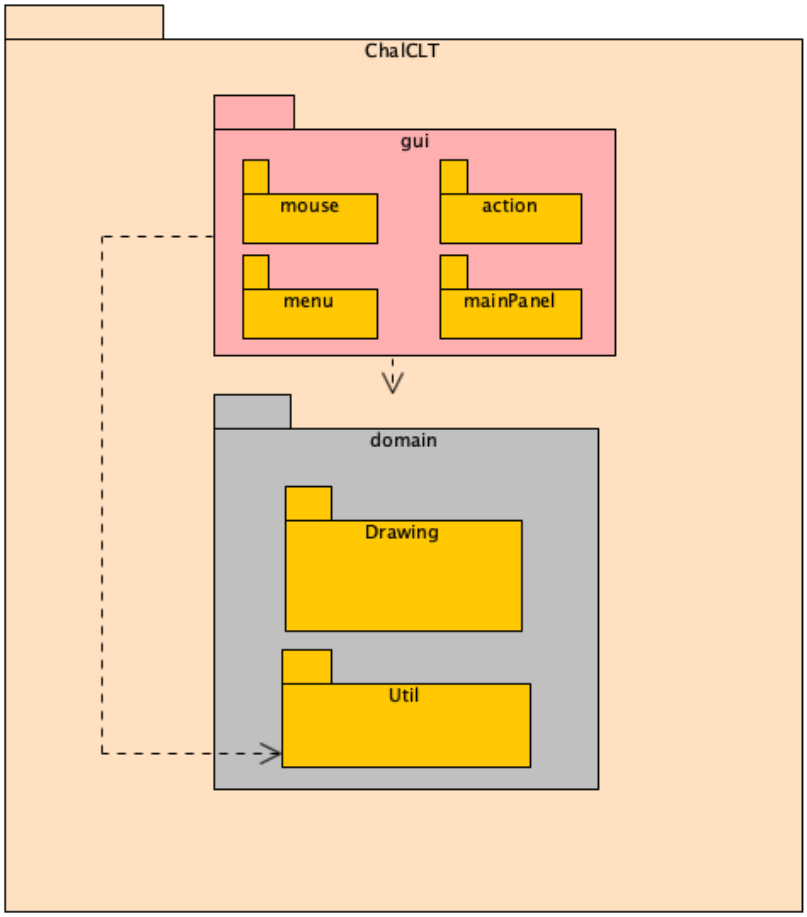
1.1.7.2 WheelListener

La classe WheelListener implémente l'interface java.awt.event.MouseWheelListener, et permet de capturer un java.awt.event.MouseWheelEvent qui servira à l'implémentation du zoom/dezoom à fait par l'utilisateur avec la roulette de la souris.

1.1.8 Package gui.action

Ce package contient les différents ActionListener pour écouter les actions effectuées sur les boutons.

2. Architecture logique



2.1 Texte explicatif de l'architecture logique

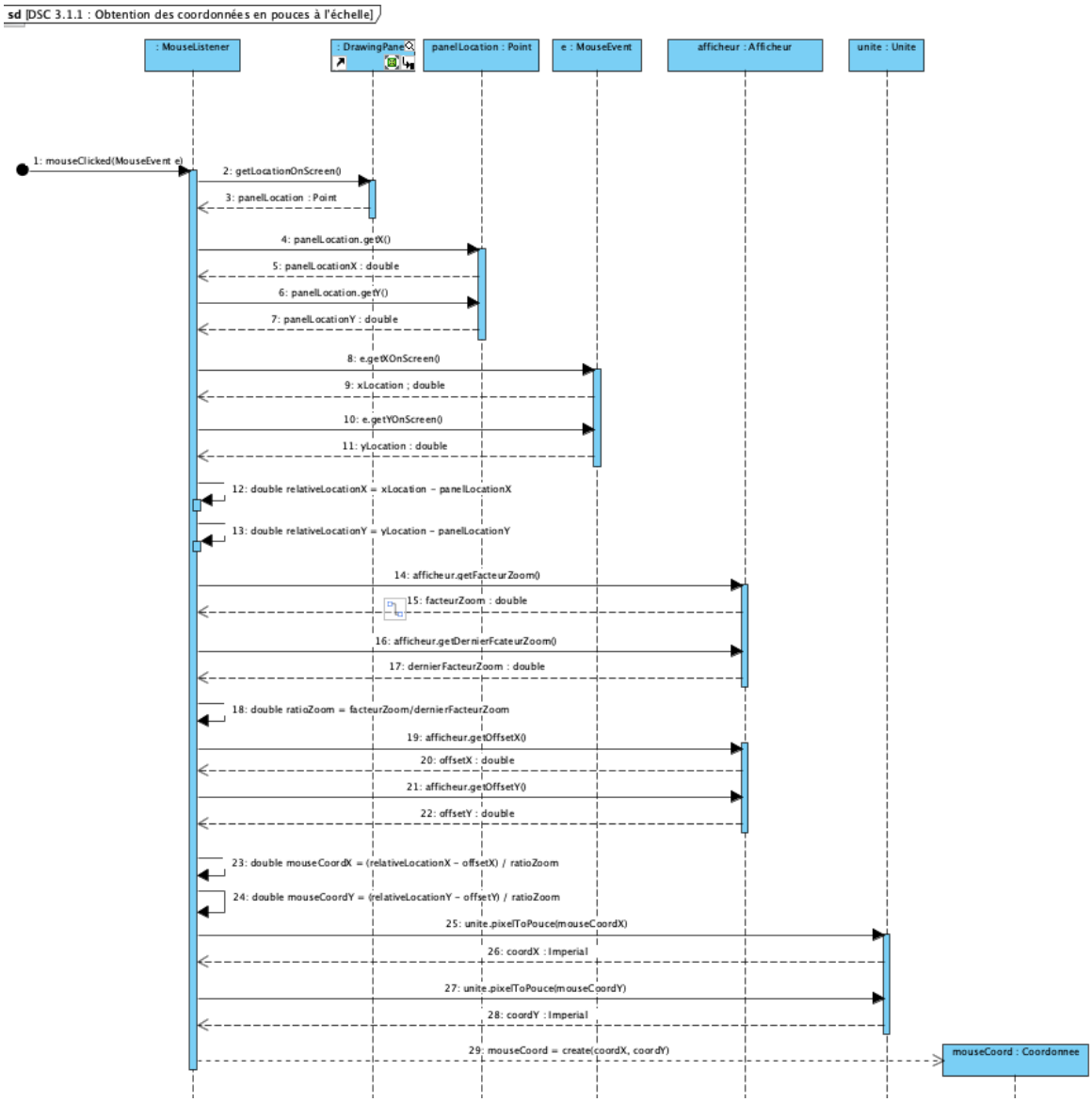
Notre architecture logicielle est constituée de deux packages principaux, chacun ayant des rôles spécifiques bien définis. En premier lieu, le package "gui" est chargé de la gestion de l'interface utilisateur, garantissant ainsi une expérience conviviale pour les utilisateurs.

Ensuite, le package "domain" se consacre à la création et à la maintenance du chalet, en implémentant la logique nécessaire pour ces opérations. Il englobe le package "Util", regroupant les utilitaires essentiels pour effectuer des conversions de données et afficher des informations au format impérial. De plus, il inclut le package "Drawing", permettant de dessiner les différents éléments du chalet, tels que les côtés, les murs, les panneaux du toit, les accessoires, et bien d'autres.

À l'intérieur du package "gui", il y a quatre sous-packages : "action", "mouse", "menu", et "mainPanel". Le sous-package "action" est spécifiquement dédié à la capture des événements propres à l'application, tandis que le sous-package "mouse" se focalise sur la gestion des événements liés aux clics de souris et au défilement de la roulette. Les packages "menu" et "mainPanel" sont des extensions du MainWindow. Ainsi, "menu" contient les classes permettant d'accéder au menu d'exportation et des vues, tandis que "mainPanel" comprend les classes permettant de manipuler les différents panels du gui tel que le TopPanel, le SplitPanel et le DrawingPanel.

3. Diagramme de séquence de conception (DSC)

3.1.1 Obtention des coordonnées en pouces à l'échelle

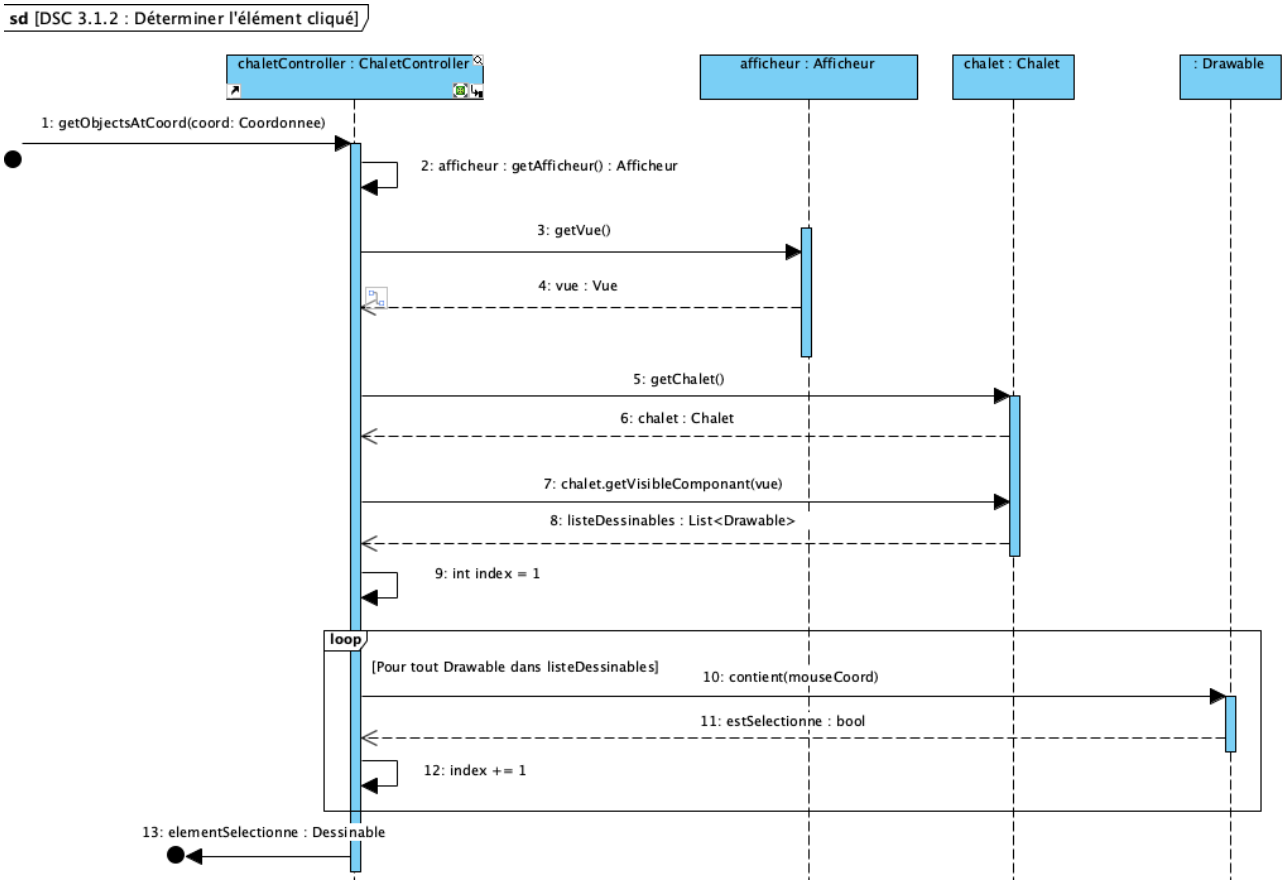


Le diagramme ci-dessus décrit la méthode complète que nous utilisons pour obtenir les coordonnées en pouces à l'échelle. Tout d'abord, nous capturons un événement de souris grâce au MouseListener. Ensuite, nous obtenons la position du DrawingPanel par rapport à l'écran de l'utilisateur à l'aide de la méthode getLocationOnScreen() qui nous renvoie un objet Point. Ainsi, nous obtenons les positions X et Y de notre DrawingPanel avec les méthodes getX() et getY(). Les coordonnées X et Y de la position de la souris sur l'écran sont obtenues à l'aide des méthodes du MouseEvent getXOnScreen() et getYOnScreen(). Nous sauvegardons ensuite les positions en X et en Y de ces deux coordonnées, puis nous effectuons la différence entre les deux pour obtenir une position relative en X et une position relative en Y. Ceci nous donne la position du curseur dans le DrawingPanel, quel que soit l'état du zoom.

Ensuite, nous avons besoin du facteur de zoom actuel ainsi que du dernier facteur de zoom pour être en mesure de savoir à partir de quelle échelle et vers quelle échelle l'affichage a été transformé. À l'aide de l'Afficheur, qui contient le facteur de zoom actuel (facteurZoom) ainsi que le dernier facteur de zoom (dernierFacteurZoom), on calcule le quotient de ces deux propriétés, nous obtenons le ratio de zoom.

Ensuite, nous avons également besoin du décalage du zoom en X et en Y (offsetX et offsetY), puisque le zoom se fait en fonction de la position du curseur. Nous obtenons ensuite la position relative au ratio de zoom et au décalage à l'aide des équations : mouseCoordRelativeX = (relativeLocationX - offsetX) / ratioZoom et mouseCoordRelativeY = (relativeLocationY - offsetY) / ratioZoom. Étant donné que les coordonnées obtenues sont en pixels, pour les traduire en pouces, nous faisons appel à la fonction pixelToPouce, cette méthode utilise la densité de pixels par pouce de l'écran, que nous obtenons en appelant la méthode java.awt.Toolkit.getScreenResolution().

3.1.2 Déterminer l'élément cliqué



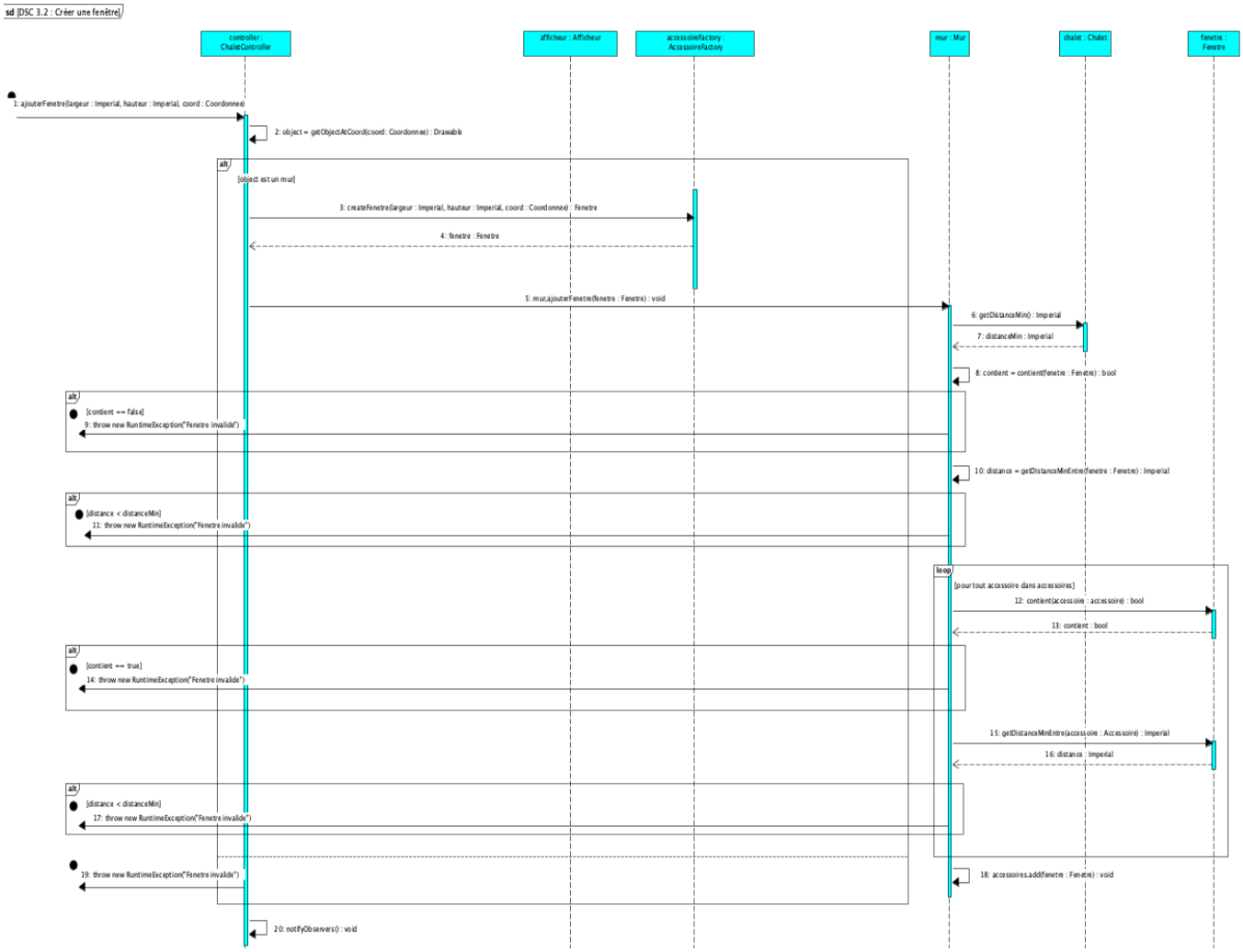
Pour obtenir l'élément cliqué, il est essentiel de déterminer les éléments sur lesquels un clic est possible. C'est pourquoi nous avons besoin de la classe Afficheur, qui contient la vue actuelle de l'utilisateur.

En utilisant la fonction getDessinable(vue) avec la vue en paramètre, nous pouvons accéder aux objets dessinables. Cette fonction nous renvoie une liste d'objets Drawable sur lesquels nous allons itérer pour déterminer quel objet se trouve entre les quatre sommets des coordonnées où se trouve la souris.

Il est important de noter que dans notre concept, tous les objets dessinables héritent de la classe abstraite Dessinable, qui contient les coordonnées de tous les éléments en pouces.

Avec la liste List<Dessinable> contenant tous les accessoires, murs et toits à notre disposition, nous itérons simplement pour vérifier si les coordonnées initiales se trouvent à l'intérieur de l'un des objets Dessinable. Si tel est le cas, nous changeons la valeur du booléen estSelectionne en true, puis nous retournons l'objet cliqué.

3.2 Créer une fenêtre



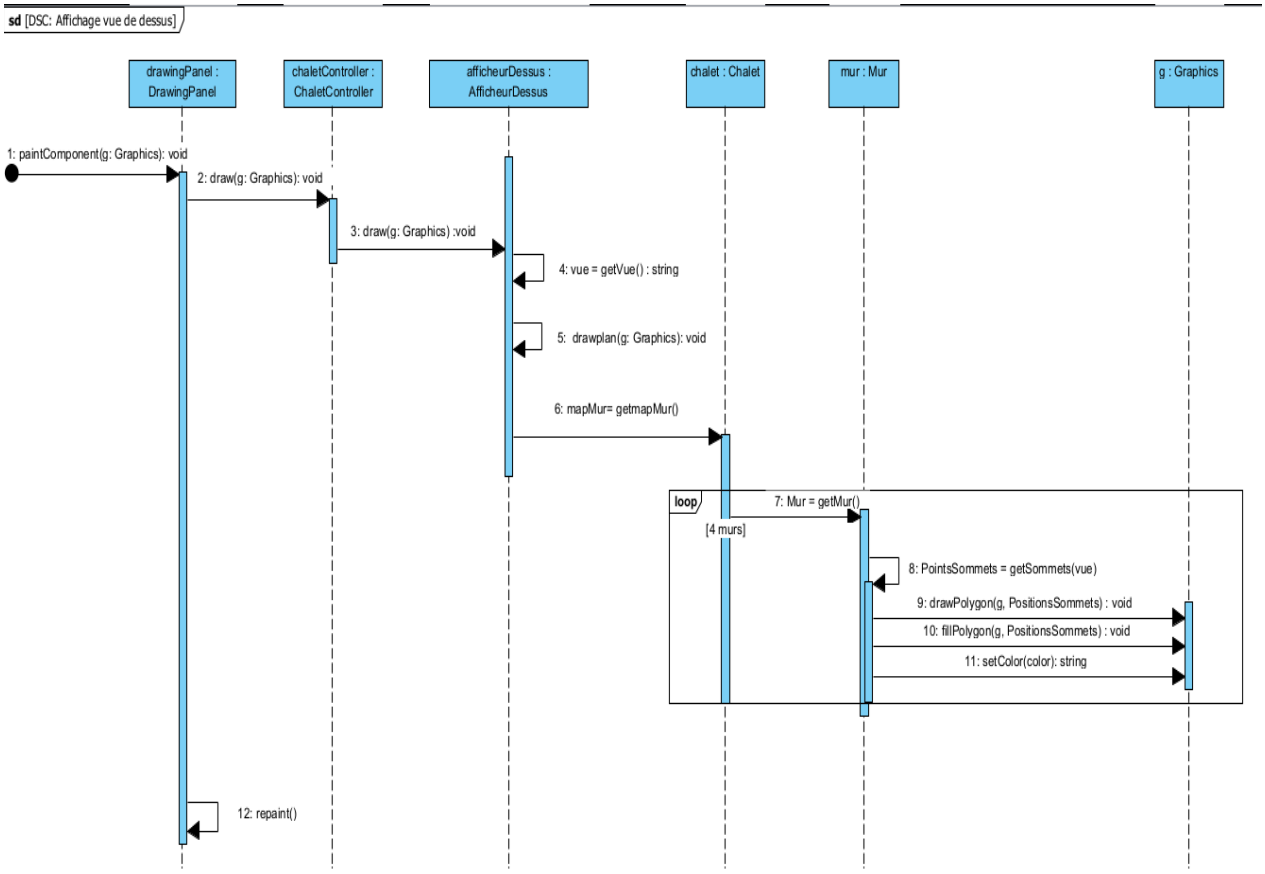
Pour créer une fenêtre, l'utilisateur commence par cliquer sur le bouton "Ajouter fenêtre". Ensuite, il clique sur l'emplacement où il souhaite la placer sur le mur existant. L'application vérifie si le clic s'est produit à un emplacement valide, c'est-à-dire s'il n'y a pas déjà un autre accessoire à cet endroit et si la distance minimale entre les accessoires est respectée.

On vérifie également si le clic s'est produit sur l'un des murs du chalet. Si le clic ne correspond pas à un mur, l'application affiche une erreur, indiquant qu'il est impossible d'ajouter une fenêtre à cet endroit, car la position n'est pas valide.

Si le clic est sur un mur, on identifie sur quel mur l'utilisateur souhaite ajouter la fenêtre. Si aucun mur est trouvé l'application affiche une erreur signalant qu'aucun mur n'a été trouvé à cette position. Si un mur est trouvé, on crée une nouvelle fenêtre et l'insère dans la liste des accessoires du mur,

immédiatement après. On ajoute également les coordonnées de la nouvelle fenêtre à la liste des accessoires. Enfin, l'affichage est mis à jour pour refléter les modifications apportées lors de l'ajout de la fenêtre sur le mur.

3.3 Affichage de la vue de dessus



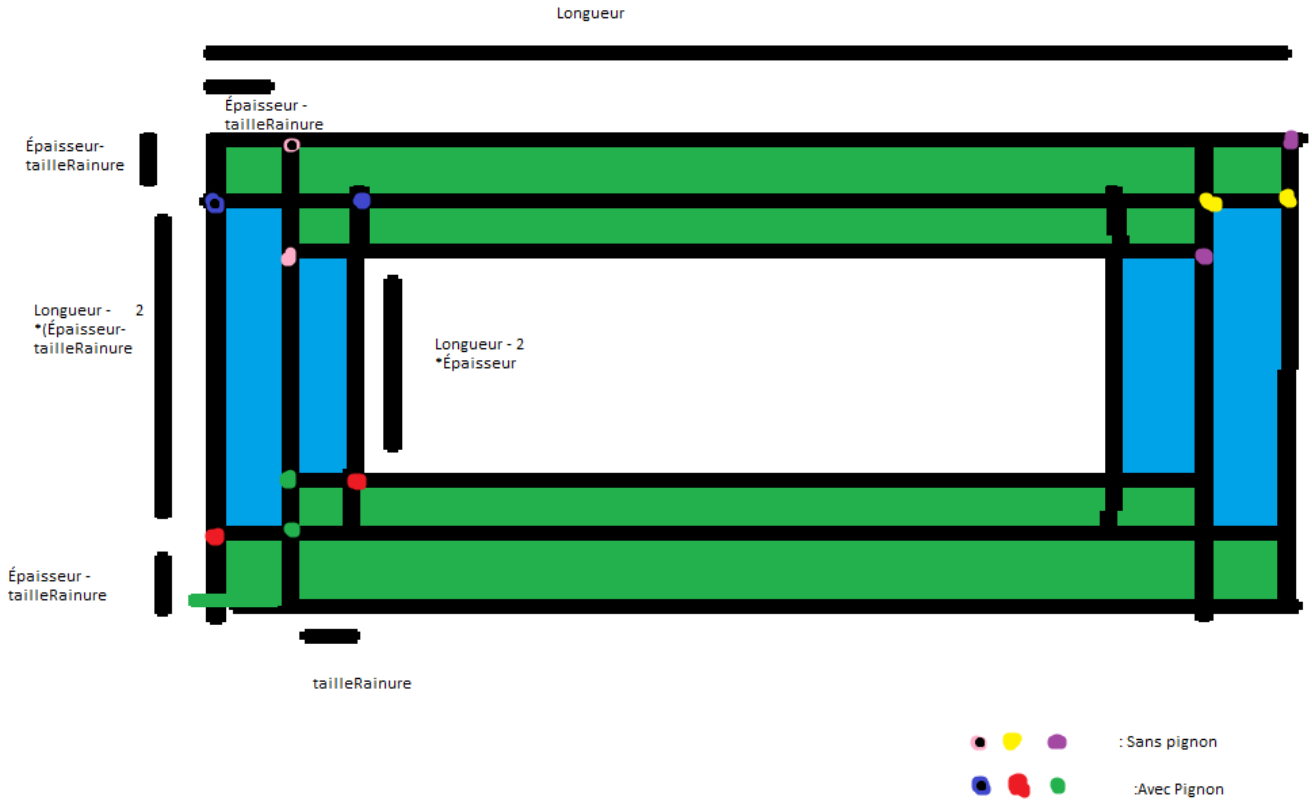
Dans ce diagramme, le processus est déclenché par l'appel à *paintComponent(g)* de la classe *DrawingPanel* pour permettre de redessiner la vue sélectionnée.

Une fois l'afficheur de dessus sollicité, il récupert la vue grâce à la méthode getVue() et met à jour la variable vue. C'est également à cette étape que les vues sont généralement mises à jour et donc que les différentes actions sont mises en place pour dessiner la vue correspondante. La classe intiei ensuite le dessin avec la méthode drawPlan(g). Finalement, cette classe nécessite les différentes informations sur les murs et pour cela, elle fait appel à la méthode getmapMur de la classe Chalet. Celle-ci retroune la map des murs qui sont stockés dans la variable Mur.

On initie ensuite une boucle à partir de Chalet. On sort de la boucle lorsque les 4 murs sont dessinés. Pour chaque itération:

- On appelle la méthode getMur de la classe Mur afin d'avoir les informations sur le mur dessiné.
- À partir de Mur, on récupère les sommets selon la vue stockée précédemment avec la méthode getSommets, qu'on stocke dans l avariable PointsSommets.
- Pour les deux étapes suivantes, on utilise ces sommets pour dessiner le contour avec drawPolygon et le remplir avec fillPolygon.
- Finalement, on définit la couleur avce la méthode setColor.

4. Pseudo-code de la fonction retirer_un_prisme(deltaRainure,pointDepart)

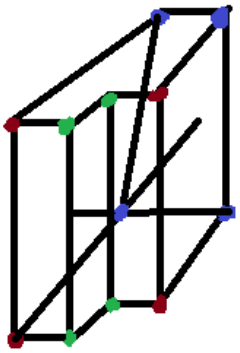


Dans ce problème, il nous faut seulement trouver comment retirer **une** rainure, donc on garde les même longueur et épaisseur (on utilise le terme largeur dans ce pseudo-code), mais le schéma ci-dessous illustre mieux le résultat final de notre liste de sommet à partir du prisme de départ.

Classe PrismRectangulaire

Attributs :

Largeur // x
Hauteur // y
Longueur // z
Pignon



Méthode Constructeur (largeur, hauteur, longueur,pignon)

this.largeur = largeur
this.hauteur = hauteur
this.longueur = longueur
this.pignon = pignon
//si le prisme rectangulaire est un mur qui soutient un pignon, retourne vrai. Sinon, faux.

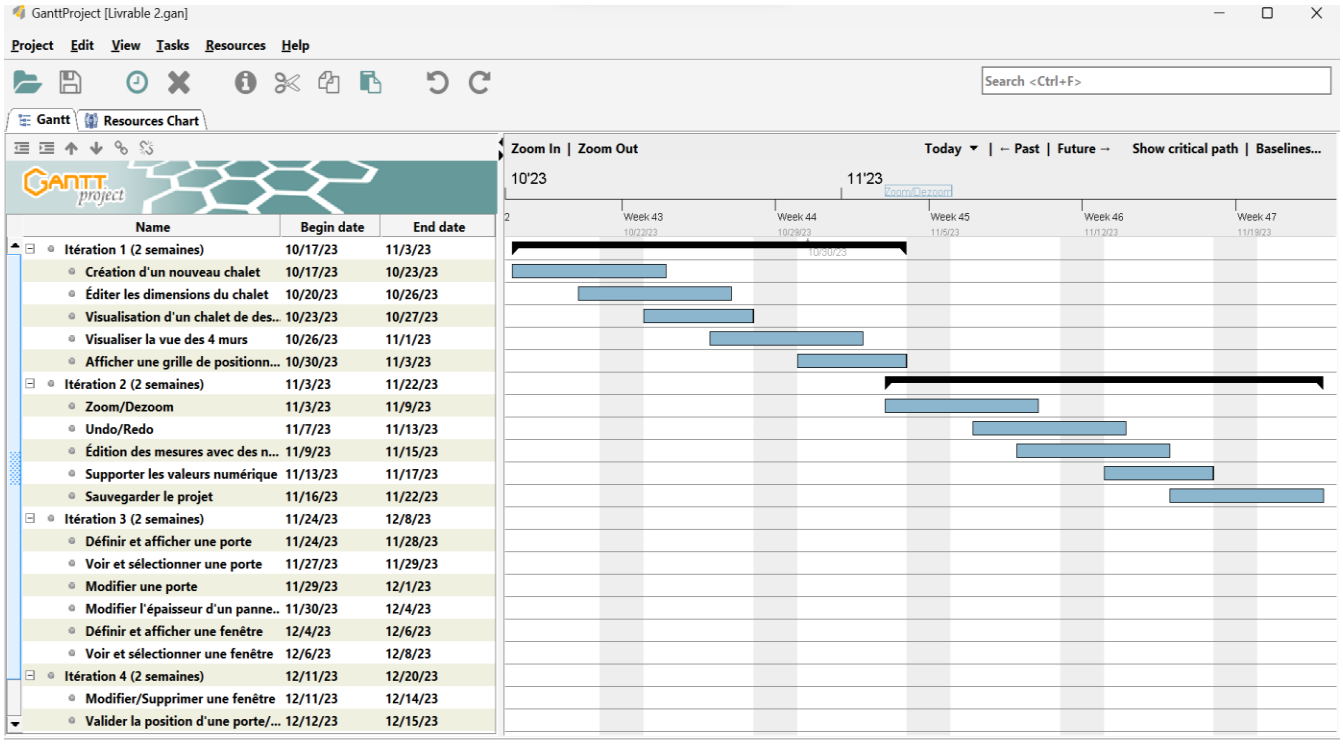
Méthode retirer_un_prisme(deltaRainure)

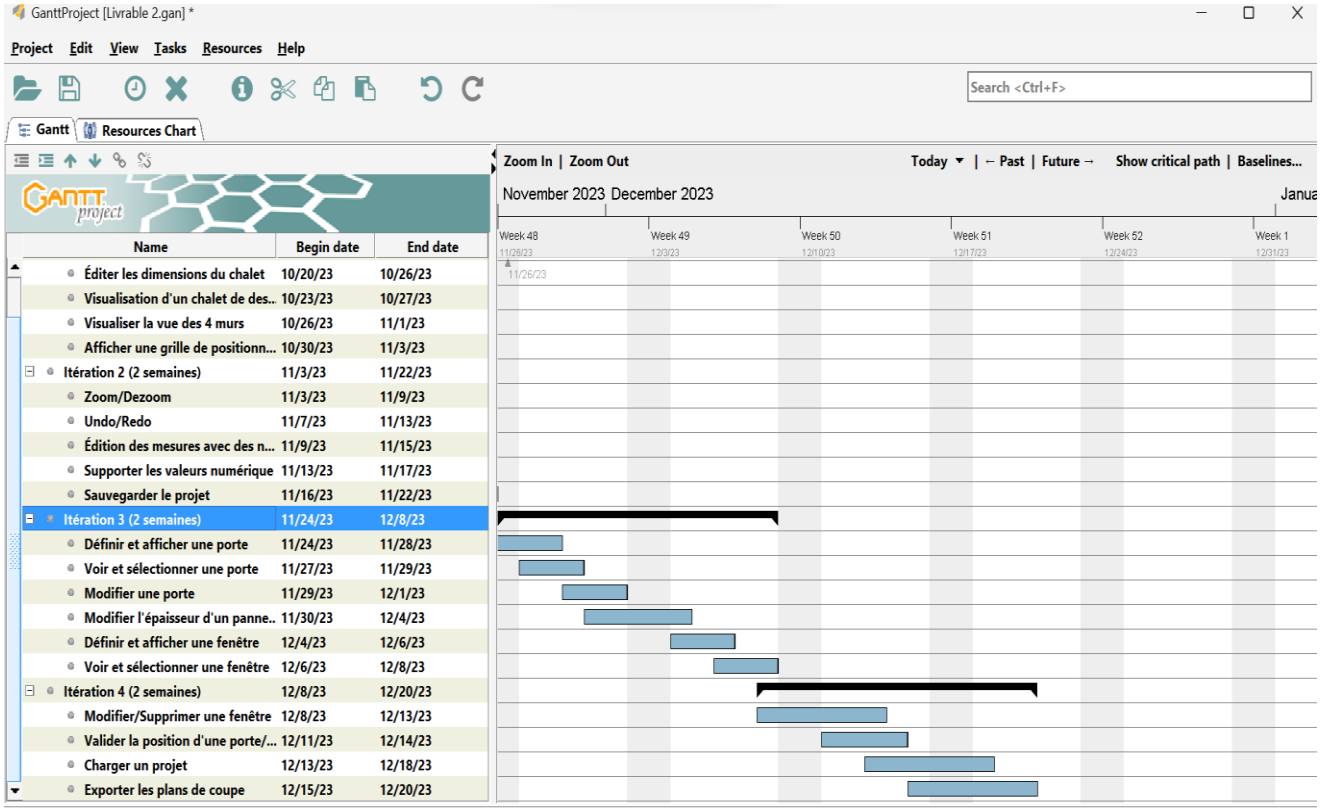
sommetPolygone= ListeVide

```
    rainureTaille= moitieLargeur + deltaRainure


    // Sommet de l'arrière (point bleu (P-S: les lignes ne sont pas représentatives, car ça doit être un
       prisme rectangulaire pas un prisme bizarre))
    sommetPolygone.ajouter (Nouveau Point3D(0, 0, 0))
    sommetPolygone.ajouter(Nouveau Point3D(largeur, 0, 0))
    sommetPolygone.ajouter (Nouveau Point3D(largeur, hauteur, 0))
    sommetPolygone.ajouter (Nouveau Point3D(0, hauteur, 0))

    Si pignon:

        // Sommet pour la création de la rainure (point vert)
        sommetPolygone.ajouter (Nouveau Point3D (largeur-tailleRainure, 0, longueur-(largeur-
tailleRainure)))
        sommetPolygone.ajouter (Nouveau Point3D (largeur-tailleRainure, hauteur, longueur-
(largeur-tailleRainure)))
         sommetPolygone.ajouter (Nouveau Point3D(rainureTaille, 0, longueur-largeur))
        sommetPolygone.ajouter (Nouveau Point3D(largeur-tailleRainure, hauteur, longueur-
largeur))

         // Les quatre sommets restants de l'avant (point rouge)
        sommetPolygone.ajouter (Nouveau Point3D(0, 0, longueur-(largeur-tailleRainure)))
        sommetPolygone.ajouter (Nouveau Point3D(largeur, 0, longueur-largeur))
        sommetPolygone.ajouter (Nouveau Point3D(0, hauteur, longueur-(largeur-tailleRainure)))
        sommetPolygone.ajouter (Nouveau Point3D(largeur, hauteur, longueur-largeur))

     Sinon:
          // Sommet pour la création de la rainure (point vert)
        sommetPolygone.ajouter(Nouveau Point3D (largeur-tailleRainure, 0, longueur-(largeur-
tailleRainure)))
        sommetPolygone.ajouter(Nouveau Point3D (largeur-tailleRainure, 0, longueur))
         sommetPolygone.ajouter(Nouveau Point3D (largeur-tailleRainure, hauteur, longueur-
(largeur-tailleRainure)))
        sommetPolygone.ajouter(Nouveau Point3D (largeur-tailleRainure, 0, longueur))

          // Les quatre sommets restants de l'avant (point rouge)
        sommetPolygone.ajouter(Nouveau Point3D (largeur,0, longueur-(largeur-tailleRainure)))
         sommetPolygone.ajouter(Nouveau Point3D (0,0, longueur))
          sommetPolygone.ajouter(Nouveau Point3D (largeur, hauteur, longueur-(largeur-
tailleRainure)))
         sommetPolygone.ajouter(Nouveau Point3D(0, hauteur,longueur))

    Retourner sommetPolygone
Fin de Classe

Classe Point3D
  Attributs :
```

```
        x
        y
        z

    Méthode Constructeur (x, y, z)
        this.x = x
        this.y = y
        this.z = z

    Méthode ToString()
        Retourner "(" + x + ", " + y + ", " + z + ")"
Fin de Classe

Fonction Principale
    prism = Nouveau PrismRectangulaireComplexe(10, 8, 6)
    sommets = prism.retirer_un_prisme(deltaRainure)

    Pour chaque i de 0 à sommets.Taille() - 1
        Afficher "Sommet " + (i + 1) + " : " + sommets[i].ToString()
    Fin Pour
Fin de Fonction
```

5. Diagramme de Gantt





6. Justification des contributions

Petite contribution : P

Grande contribution : G

	Jérémy	Rihab	Nadir	Anass	Ziyad
Diagramme de classes de conception	G	P	P	P	G
Architecture logique	P				G
Diagramme de séquence de conception (DSC)	P	P	G	P	P
Pseudo-code				G	
Diagramme de Gantt		G			
Code interface	G				