Équipe

Chheang, Raphaël; Fontaine, Cédric ; Gagnon, Cédric; Mainardi, Maxime; Marsolais, Cédric-B.

Génie logiciel orienté objet

GLO-2004

Projet de session

Livrable 2

Travail présenté à

Marc Philippe Parent

Anthony Deschênes

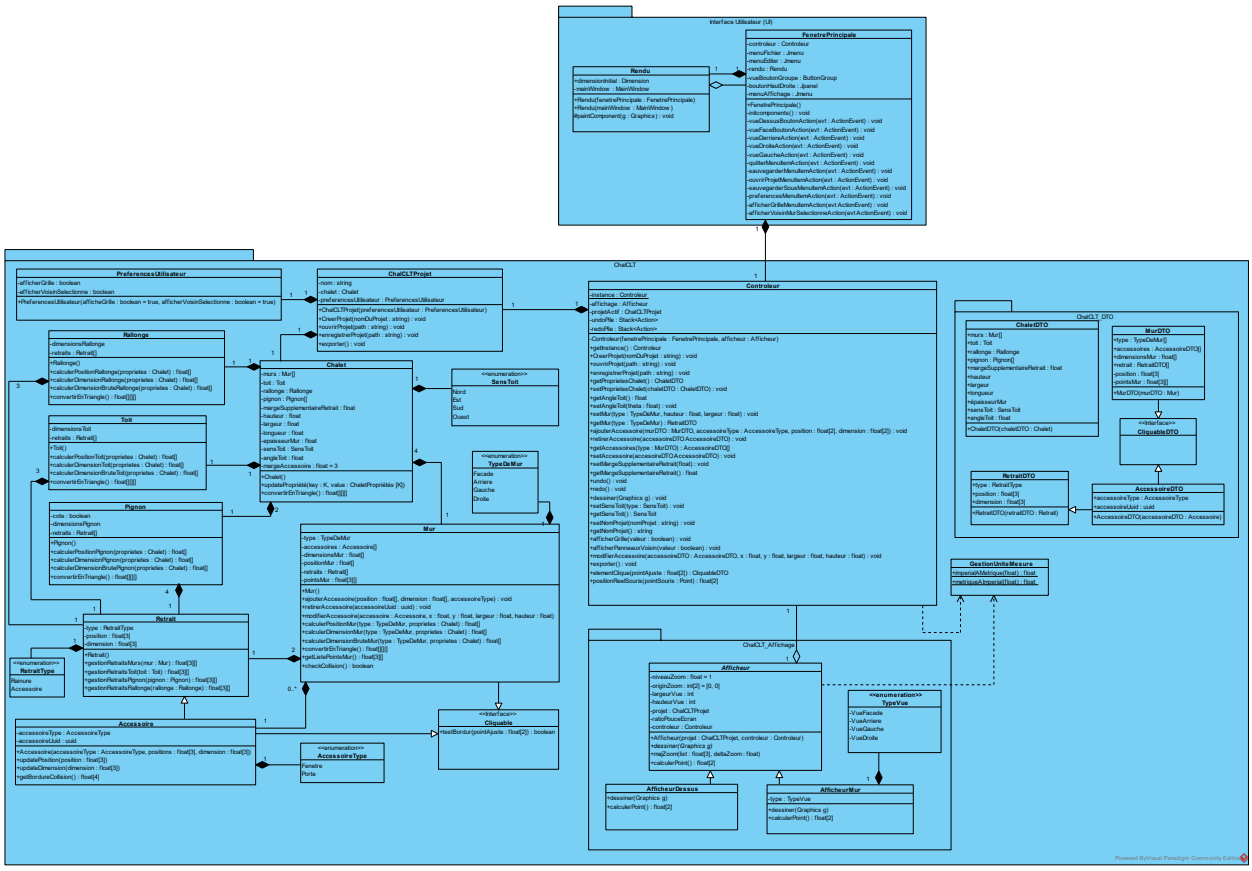
Génie Logiciel

Université Laval

Automne 2023

Table des matières

Diagramme de classes



La classe **ChalCLTProjet** est responsable de la création, l'ouverture, l'enregistrement et l'exportation d'un projet par ses méthodes et a comme attributs un nom, un **Chalet** et des **PreferencesUtilisateur**.

La classe **PreferencesUtilisateur** contient les préférences de l'utilisateur quant à l'affichage de son chalet. Plus précisément, elle détermine l'affichage de la grille et des panneaux voisins.

La classe **Chalet** contient toutes les informations et méthodes nécessaire au chalet lui-même. Elle comporte des **Murs**, un **Toit**, une **Rallonge** et des **Pignons** et elle possède les dimensions du chalet (hauteur, largeur, longueur), l'épaisseur des panneaux, une marge supplémentaire aux retraits pour compenser les erreurs possibles lors de l'usinage et l'assemblage, l'angle du toit et le sens du toit (énumération Nord, Sud, Est, Ouest nommée **SensToit**) comme attributs. Ses méthodes peuvent changer les propriétés du chalet et convertir les points de ses composantes en triangles.

La classe **Rallonge** représente le panneau qui relie un mur à la partie surélevée du toit. Composée de ses dimensions et de ses **Retraits**, ses méthodes permettent de calculer sa position, ses dimensions et ses dimensions brutes ainsi que de convertir ses points en triangles.

La classe **Pignon** représente les deux panneaux triangulaires de chaque côté du toit. Cette classe est très similaire à la classe **Rallonge**, possédant des propriétés et des méthodes homologues. La classe **Pignon** possède un attribut supplémentaire permettant de déterminer sur quel côté du toit il se situe.

La classe **Toit** représente le panneau du toit, c'est-à-dire celui qui fermera le chalet du haut. Il possède les mêmes attributs et propriétés que la classe **Rallonge**.

La classe **Mur** représente les 4 panneaux de chaque côté du chalet. Possédant les même attributs et méthodes que la classe **Rallonge**, elle possède aussi un attribut qui indique où se situe le mur par rapport au chalet (énumération Façade, Arrière, Gauche, Droite nommée **TypeDeMur**) et une liste d'**Accessoires** présents sur le mur. Elle comporte également les méthodes permettant l'ajout, le retrait et la modification de ses accessoires.

La classe **Retrait** représente les découpes des différents panneaux. Elle a pour attributs les dimensions et la position du retrait en plus de son type (énumération Retrait, Accessoire nommée **RetraitType**). Ses méthodes servent à générer les points du panneau auquel le retrait est associé.

La classe **Accessoire** dérive de la classe **Retrait** et représente les différents accessoires pouvant être ajoutés aux murs. En plus de tous les attributs et méthodes de **Retrait**, elle contient un attribut déterminant son type (énumération Fenetre, Porte nommée **AccessoireType**) ainsi qu'un identificateur permettant de faciliter l'accès aux différents accessoires d'un mur. Cette classe contient des méthodes pouvant mettre à jour sa position et ses dimensions et une méthode pour assurer que son placement est valide.

La classe **Cliquable** est une classe d'interface pour tous les objets avec lesquels l'utilisateur peut interagir en cliquant dessus avec la souris. Elle possède une méthode permettant de savoir si la souris se situe à l'intérieur de l'aire de l'objet lors du clic. Elle est la classe générique des classes **Mur** et **Accessoire**.

La classe abstraite **Afficheur** sert de classe de base aux classes qui permettent d'afficher le chalet à l'écran. Elle a comme attributs un point d'origine et un niveau de zoom, une largeur et une hauteur de vue, un projet de chalet ainsi qu'un ratio permettant de faire la transition des dimensions du chalet (en pouces) et pixels. Ses méthodes permettent de mettre à jour le niveau de zoom et de calculer la position de la souris à l'aide des points. Elle possède également une méthode abstraite permettant de dessiner les composantes du chalet à l'écran.

Deux classes dérivent de la classe **Afficheur**. La classe **AfficheurDessus** permet d'afficher le chalet en vue d'oiseau (n'affichant pas le toit) tandis que la classe **AfficheurMur** permet d'afficher le chalet en vue de côté (affichant un des 4 murs). Cette dernière possède un attribut supplémentaire qui indique le mur à afficher (énumération VueFacade, VueArriere, VueGauche, VueDroite nommée **TypeVue**).

La classe **GestionUniteMesure** contient des méthodes permettant de convertir les mesures de pouces vers centimètres et vice-versa, dépendamment de la préférence de l'utilisateur.

Le projet comporte plusieurs classes de type DTO (Data Transfer Object). Ces classes permettent à l'interface utilisateur l'accès aux données membres des classes du domaine ChalCLT sans pouvoir les modifier. Les classes **ChaletDTO, MurDTO, RetraitDTO, AccessoireDTO** et **CliquableDTO** sont des classes qui contiennent uniquement les attributs des classes **Chalet**, **Mur**, **Retrait**, **Accessoire** et **Cliquable** respectivement. La classe **MurDTO** dérive de la classe **CliquableDTO** et la classe **AccessoireDTO** dérive des classes **RetraitDTO** et **CliquableDTO**.

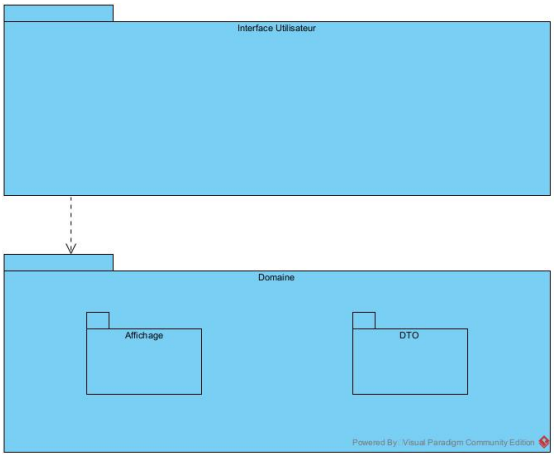
La classe **FenetrePrincipale** représente la fenêtre qui affichera l'application ChalCLT à l'écran. Comme attributs, elle possède un **Controleur**, un **Rendu** et toutes les composantes nécessaires à l'interface

utilisateur. Ses méthodes servent à déterminer les actions de l'utilisateur sur l'interface et à envoyer les informations appropriées au **Controleur** ainsi qu'à afficher les composantes de l'interface utilisateur.

La classe **Rendu** sert à dessiner toutes les composantes visuelles de l'application ChalCLT. Contenant une **FenetrePrincipale** et les dimensions de la fenêtre où s'affiche le chalet, elle possède une méthode permettant de dessiner toutes les composantes du chalet sur la fenêtre.

La classe **Controleur** assure la communication entre l'interface utilisateur et le domaine ChalCLT en utilisant les classes DTO comme intermédiaire. Il suit le principe de Larman. Il ne peut en avoir qu'une seule instance à la fois dans l'application. Cette classe possède deux piles d'actions utilisées lors des undo/redo, un **Afficheur** et une instance de projet **ChalCLTProjet**. Ses méthodes servent à accéder et modifier les composantes et les propriétés du chalet, à annuler et rétablir les actions, à accéder aux préférences de l'utilisateur, à créer, enregistrer, ouvrir et exporter des projets, puis à obtenir la position de la souris et l'élément cliqué par l'utilisateur.

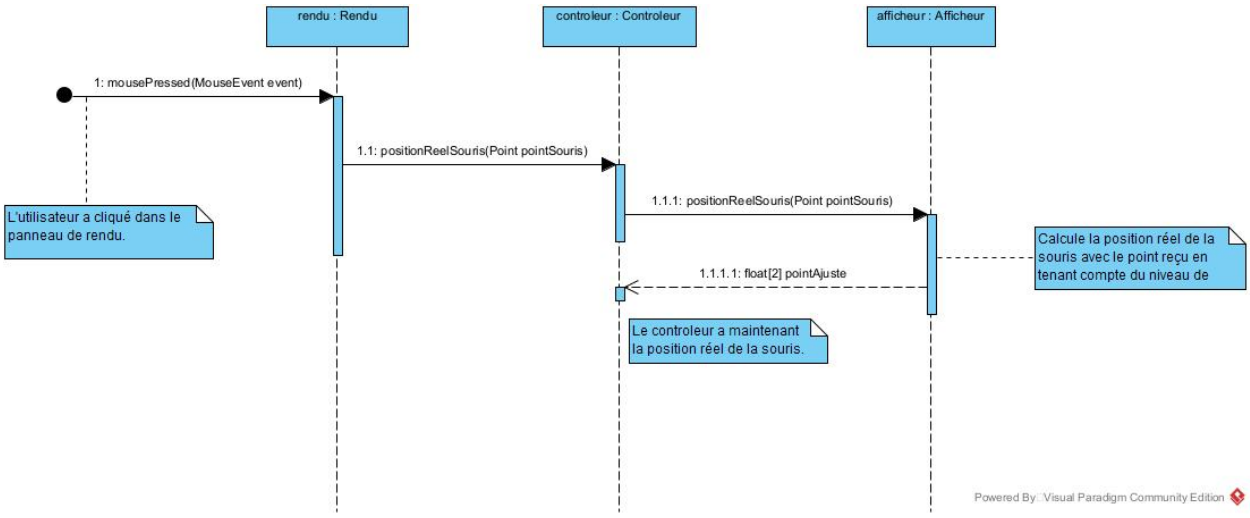
Architecture logique



Le projet ChalCLT comportera plusieurs paquets. Premièrement, le paquet **ChalCLT** sera le paquet principal du programme. Il joue le rôle du domaine et comprend toutes les classes associées à la conception du chalet telle quelle. À l'intérieur de ce paquet, on compte également deux sous paquets. Le paquet **ChalCLT_Affichage** sera celui qui contient les classes et méthodes nécessaires à l'affichage du chalet à l'écran, tandis que le paquet **ChalCLT_DTO** regroupera les classes contenant les données importantes au chalet. De plus, un autre paquet inclus dans le projet est le paquet **Interface Utilisateur**. Ce paquet englobe l'information de la fenêtre et du rendu, c'est celui qui se charge de l'affichage de l'application globale. Il communique donc avec la classe **Controleur** du paquet **ChalCLT** pour obtenir les informations nécessaires à ses fonctions.

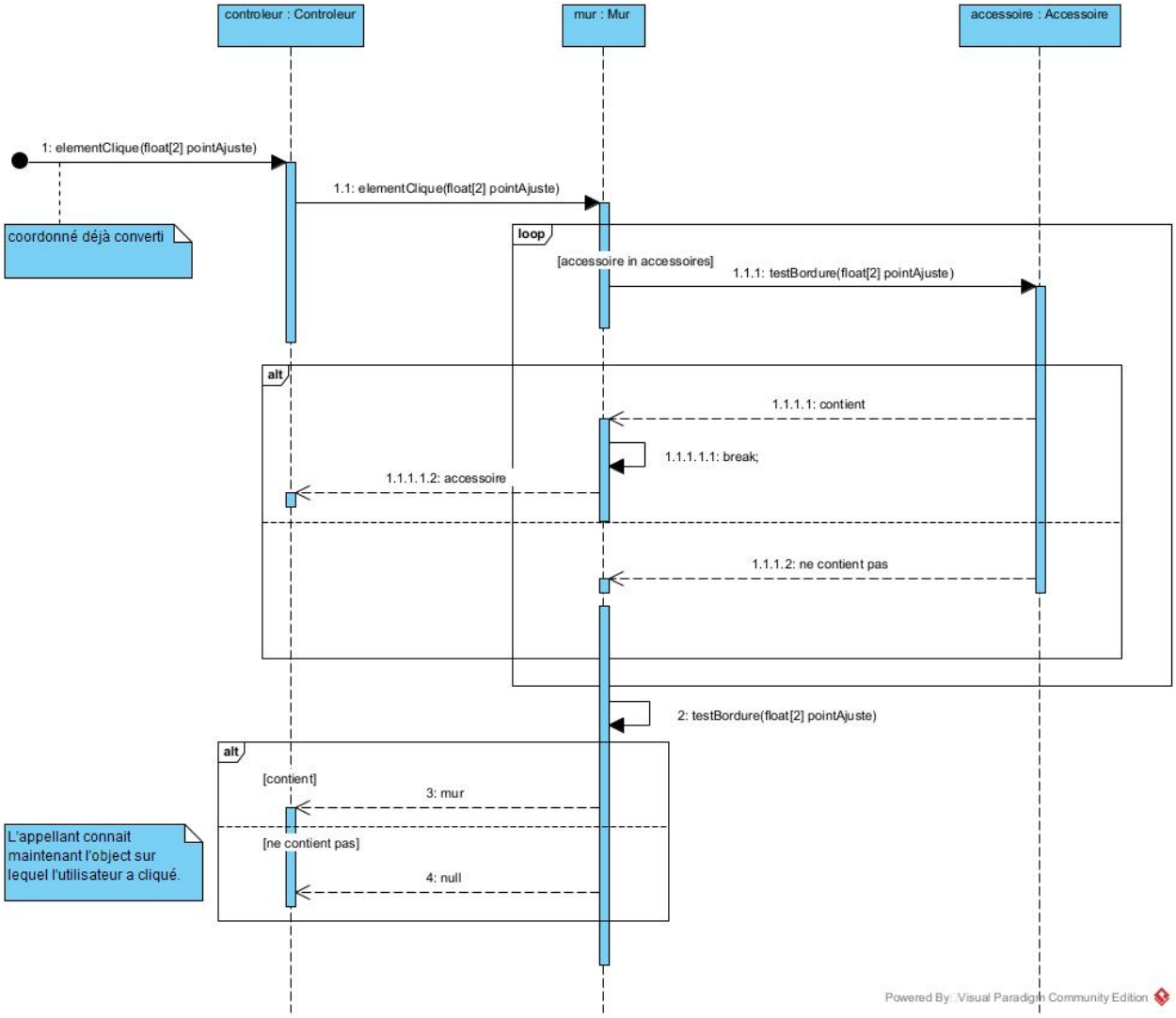
Diagrammes de séquence de conception

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



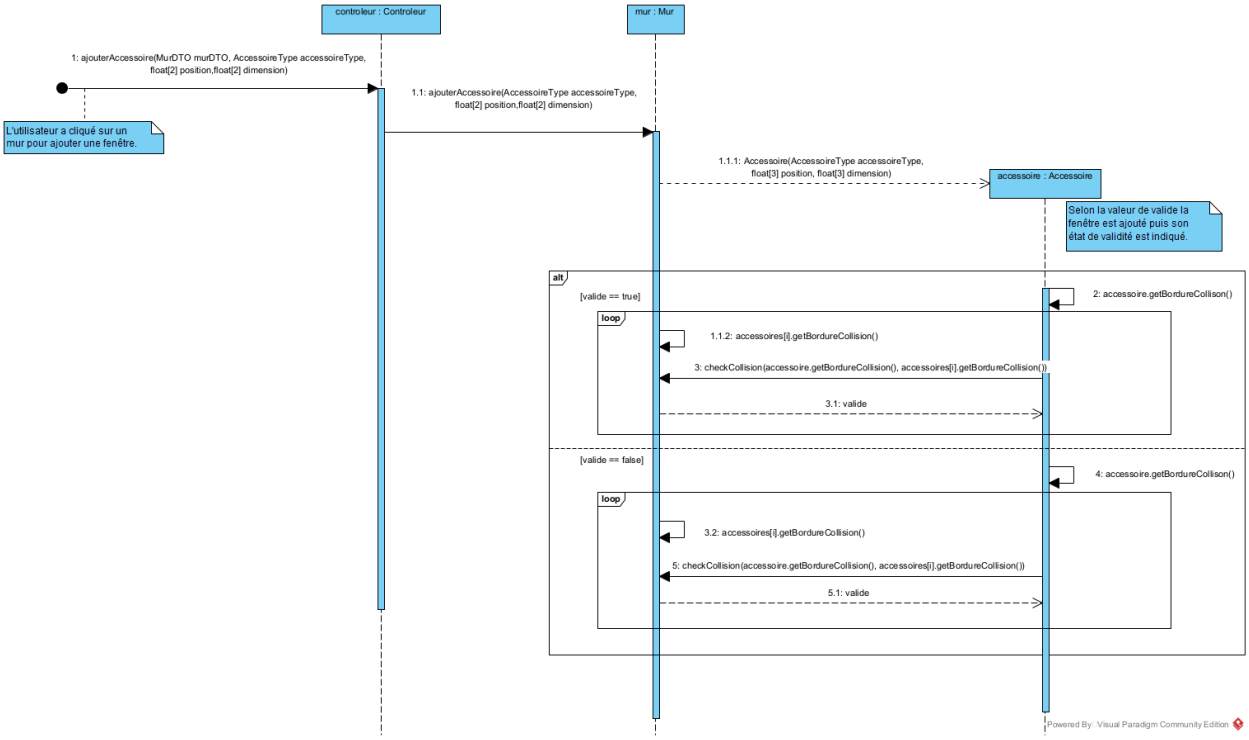
Lorsque l'utilisateur clique sur le panneau de rendu, celui-ci reçoit un événement **mousePressed** indiquant la position relative du curseur. Celle-ci est relayée au contrôleur par **positionReelSouris**, qui la retransmet vers l'afficheur. Avec l'information qu'il possède sur le niveau et le centre de zoom, celui-ci est en mesure de retourner à l'appelant la position de la souris, **pointAjuste**, dans le système de coordonnées du chalet.

Déterminer l'élément sur lequel a lieu un clic de souris dans la vue d'un Mur (partie 2)



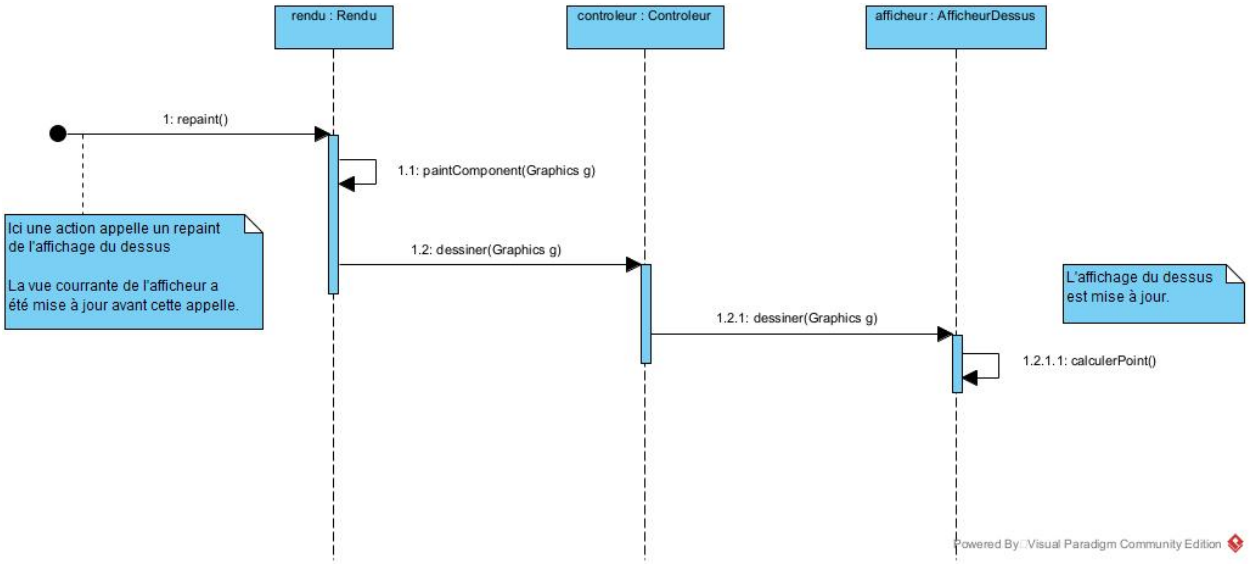
Lorsque l'élément cliqué doit être déterminé, le contrôleur relaye la demande par **elementClique** au mur couramment sélectionné. Celui-ci effectue un **testBordure**, pour chacun de ses accessoires, si la position du clique se situe à l'intérieur; si tel est le cas, celui-ci est retourné à l'appelant. Si aucun des accessoires ne correspond, un **testBordure** supplémentaire est effectué pour voir si le mur lui-même est sélectionné, dans quel cas celui-ci est retourné à la place. Si le clique ne correspond à aucun élément, un élément nul est retourné.

Créer une fenêtre



Quand l'utilisateur clique sur un mur pour ajouter une fenêtre, le **contrôleur** appelle sa méthode **ajouterAccessoire** utilisant un objet MurDTO, qui permet à l'affichage d'indiquer le mur affecté. Ensuite, la méthode **ajouterAccessoire,** qui se situe dans la classe Mur, est utilisé pour construire et ajouter un accessoire de type fenêtre sur le mur ayant les dimensions choisies, instanciant alors un objet du type Accessoire. Dans la vérification de la validité de l'accessoire ajouté, deux cas alternatifs se distinguent; si la position est valide (valide == true), ou si elle ne l'est pas (valide == false). La méthode **getBordureCollision** de l'objet est appelée, retournant la position du coin supérieur gauche et les dimensions de la fenêtre en prenant en compte la marge de l'accessoire (voir l'attribut de **Chalet** margeAccessoire). Par la suite, dans une boucle, on compare la bordure du nouvel accessoire à celles de chaque accessoire du mur, grâce à la méthode **checkCollision**, retournant la validité en boolean. La fenêtre est ajoutée qu'elle est valide ou non, étant marquée comme invalide, avec un simple changement de propriétés visuelles, dans le cas échéant.

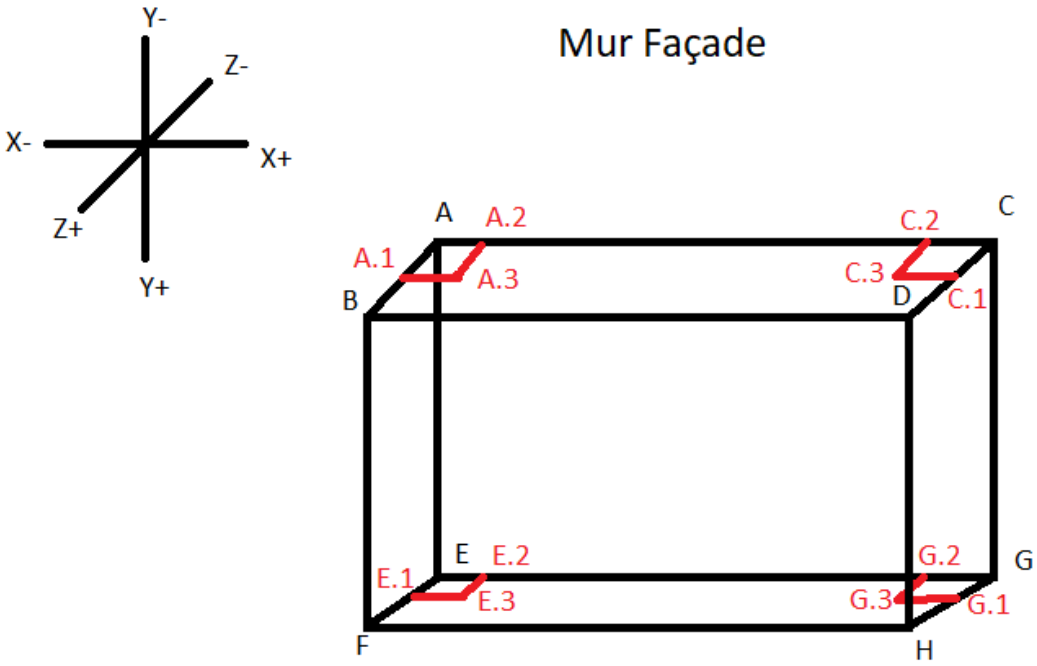
L’affichage de la vue du dessus



Quand un appel est fait à la méthode **repaint** de l’élément de rendu, la méthode **paintComponent** de celui-ci est appelée, menant à un appel vers la méthode **dessiner** du controleur, puis de l’afficheur. L’affichage est mis préalablement à la vue du dessus si cela était requis; pour chaque mur, les polygones à dessiner sont alors passés dans la fonction **calculerPoint** pour les convertir des coordonnées physiques à leurs positions en pixels, avant d’être dessinés sur l’élément Graphics du rendu.

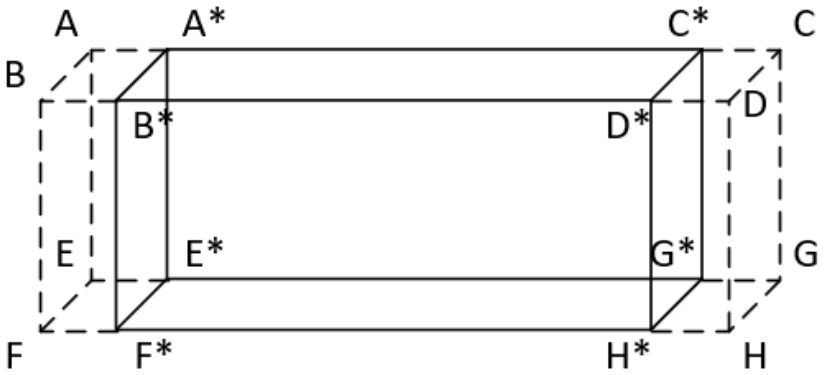
Pseudo-code

Le schéma ci-dessous contient le nom des points que le pseudo-code utilise comme référence dans les commentaires.



Le schéma ci-dessous démontre l'ajustement des points du mur lorsque besoin selon le sens du toit.

* = ajusté

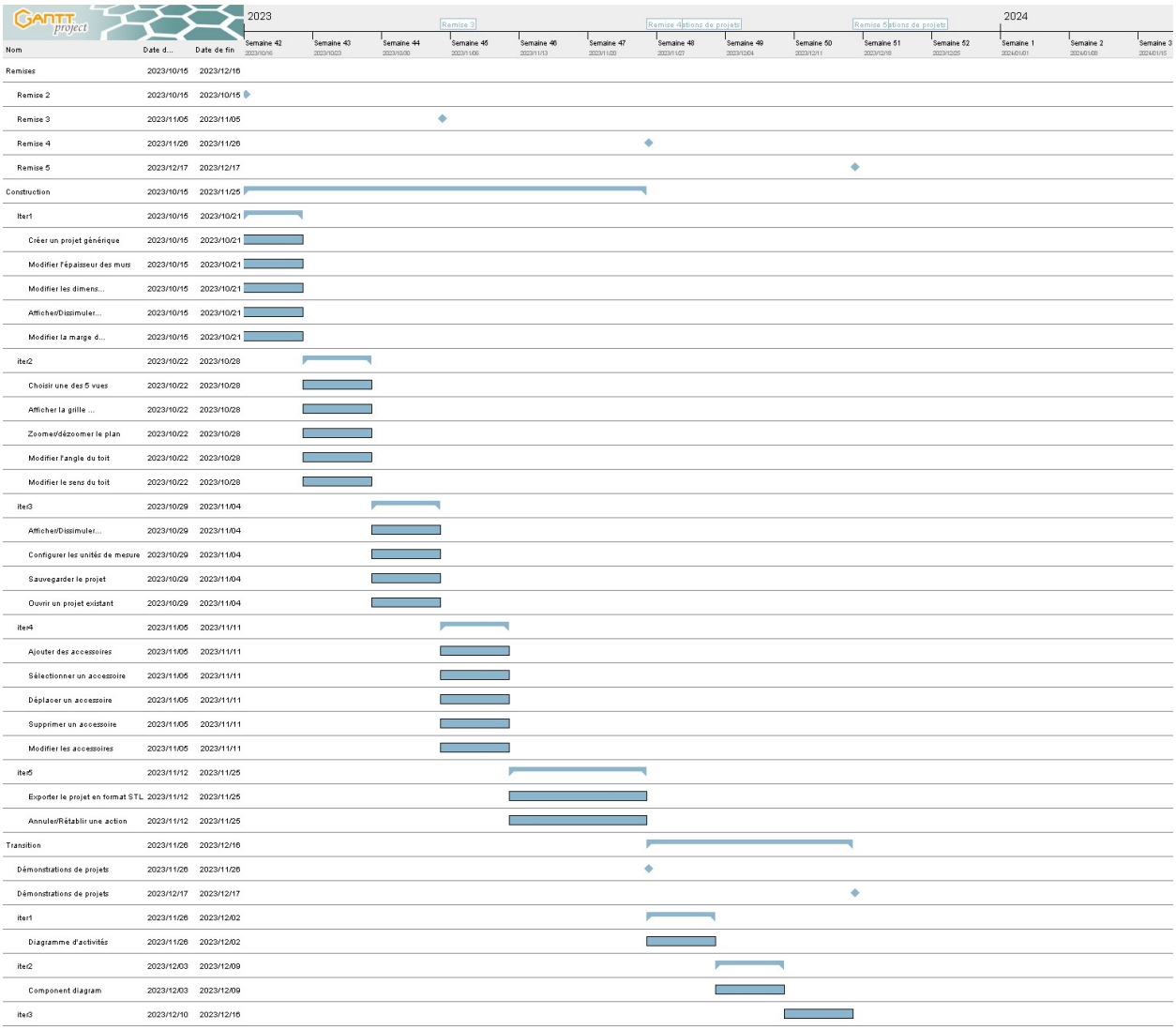


```
Fonction gestionRetraitsMurs(mur Mur,SenseToit sensToit)
{

    pointsMurModifie = copieDe(mur.pointsMur); // pointsMurModifie copie la liste des points du mur

    //X=0; Y=1; Z=2;
    largeur = mur.dimensionsMur[X]; //largeur du mur
    hauteur = mur.dimensionsMur[Y]; //hauteur du mur
    epaisseur = mur.dimensionsMur[Z]; //epaisseur du mur

    si mur.type est "Facade" alors
            si sensToit est "Nord" ou sensToit est "sud" alors

                //ajouts des nouveaux points
                nouveauPoint = copieDe(pointsMurModifie[A]) //donne à nouveauPoint la valeur du point A
                nouveauPoint[Z] = nouveauPoint[Z] + epaisseur/2
                ajouter nouveauPoint à pointsMurModifie        //ajoute le point A.1

                nouveauPoint[Y] = nouveauPoint[Y] + hauteur
                ajouter nouveauPoint à pointsMurModifie        //ajoute le point E.1

                nouveauPoint = copieDe(pointsMurModifie[A]) //met nouveauPoint à la valeur du point A
                nouveauPoint[X] = nouveauPoint[X] + epaisseur/2
                ajouter nouveauPoint à pointsMurModifie        //ajoute le point A.2

                nouveauPoint[Y] = nouveauPoint[Y] + hauteur
                ajouter nouveauPoint à pointsMurModifie        //ajoute le point E.2

                nouveauPoint[Z] = nouveauPoint[Z] + epaisseur/2
                ajouter nouveauPoint à pointsMurModifie        //ajoute le point E.3

                nouveauPoint[Y] = nouveauPoint[Y] - hauteur
                ajouter nouveauPoint à pointsMurModifie        //ajoute le point A.3

                nouveauPoint = copieDe(pointsMurModifie[C]) //met le nouveauPoint à la valeur du point C
                nouveauPoint[Z] = nouveauPoint[Z] + epaisseur/2
                ajouter nouveauPoint à pointsMurModifie    //ajoute le point C.1

                nouveauPoint[Y] = nouveauPoint[Y] + hauteur
                ajouter nouveauPoint à pointsMurModifie        //ajoute le point G.1

                nouveauPoint = copieDe(pointsMurModifie[C]) //met le nouveauPoint à la valeur du point C
                nouveauPoint[X] = nouveauPoint[X] - epaisseur/2
                ajouter nouveauPoint à pointsMurModifie        //ajoute le point C.2

                nouveauPoint[Y] = nouveauPoint[Y] + hauteur
                ajouter nouveauPoint à pointsMurModifie        //ajoute le point G.2

                nouveauPoint[Z] = nouveauPoint[Z] + epaisseur/2
                ajouter nouveauPoint à pointsMurModifie        //ajoute le point G.3

                nouveauPoint[Y]= nouveauPoint[Y] - hauteur
                ajouter nouveauPoint à pointsMurModifie        // ajoute le point C.3
                //fin des ajouts

                //retire les points obsolètes
                supprimer le point G de pointsMurModifie  // retire le point G
                supprimer le point E de pointsMurModifie  // retire le point E
                supprimer le point C de pointsMurModifie  // retire le point C
                supprimer le point A de pointsMurModifie  // retire le point A
```

```
        sinon //si le sens du toit est "Est" ou "Ouest"

            // ajustement des points du mur par faute du sens du toit
            pointAjuster = copieDe(pointsMurModifie[A]) //donne la valeur du point A à pointAjuster
            pointAjuster[X] = pointAjuster[X] + epaisseur/2
            pointsMurModifie[A] = pointAjuster  // ajustement du point A

            pointAjuster[Y] = pointAjuster[Y] + hauteur
            pointsMurModifie[E] = pointAjuster // ajustement du point E

            pointAjuster = copieDe(pointsMurModifie[B]) //met la valeur du point B à pointAjuster
            pointAjuster[X] = pointAjuster[X] + epaisseur/2
            pointsMurModifie[B] = pointAjuster //ajustement du point B

            pointAjuster[Y] = pointAjuster[Y] + hauteur
            pointsMurModifie[F] = pointAjuster //ajustement du point F

            pointAjuster = copieDe(pointsMurModifie[C]) //met la valeur du point C à pointAjuster
            pointAjuster[X] = pointAjuster[X] - epaisseur/2
            pointsMurModifie[C] = pointAjuster  //ajustement du point C

            pointAjuster[Y] = pointAjuster[Y] + hauteur;
            pointsMurModifie[G] = pointAjuster  //ajustement du point G

            pointAjuster = copieDe(pointsMurModifie[D]) //met la valeur du point D à pointAjuster
            pointAjuster[X] = pointAjuster[X] - epaisseur/2;
            pointsMurModifie[D] = pointAjuster //ajustement du point D

            pointAjuster[Y] = pointAjuster[Y] + hauteur;
            pointsMurModifie[H] = pointAjuster //ajustement du point H
            //fin des ajustement dù au sens du toit

            //ajouts des nouveaux points
            nouveauPoint = copieDe(pointsMurModifie[A]) //donne à nouveauPoint la valeur du point A
            nouveauPoint[Z] = nouveauPoint[Z] + epaisseur/2
            ajouter nouveauPoint à pointsMurModifie        //ajoute le point A.1

            nouveauPoint[Y] = nouveauPoint[Y] + hauteur
            ajouter nouveauPoint à pointsMurModifie        //ajoute le point E.1

            nouveauPoint = copieDe(pointsMurModifie[A]) //met nouveauPoint à la valeur du point A
            nouveauPoint[X] = nouveauPoint[X] + epaisseur/2
            ajouter nouveauPoint à pointsMurModifie        //ajoute le point A.2

            nouveauPoint[Y] = nouveauPoint[Y] + hauteur
            ajouter nouveauPoint à pointsMurModifie        //ajoute le point E.2

            nouveauPoint[Z] = nouveauPoint[Z] + epaisseur/2
            ajouter nouveauPoint à pointsMurModifie        //ajoute le point E.3

            nouveauPoint[Y] = nouveauPoint[Y] - hauteur
            ajouter nouveauPoint à pointsMurModifie        //ajoute le point A.3

            nouveauPoint = copieDe(pointsMurModifie[C]) //met le nouveauPoint à la valeur du point C
            nouveauPoint[Z] = nouveauPoint[Z] + epaisseur/2
            ajouter nouveauPoint à pointsMurModifie    //ajoute le point C.1

            nouveauPoint[Y] = nouveauPoint[Y] + hauteur
            ajouter nouveauPoint à pointsMurModifie        //ajoute le point G.1

            nouveauPoint = copieDe(pointsMurModifie[C]) //met le nouveauPoint à la valeur du point C
            nouveauPoint[X] = nouveauPoint[X] - epaisseur/2
            ajouter nouveauPoint à pointsMurModifie        //ajoute le point C.2

            nouveauPoint[Y] = nouveauPoint[Y] + hauteur
```

```
                ajouter nouveauPoint à pointsMurModifie        //ajoute le point G.2

                nouveauPoint[Z] = nouveauPoint[Z] + epaisseur/2
                ajouter nouveauPoint à pointsMurModifie        //ajoute le point G.3

                nouveauPoint[Y]= nouveauPoint[Y] - hauteur
                ajouter nouveauPoint à pointsMurModifie    // ajoute le point C.3
                //fin des ajouts

                //retire les points obsolètes
                supprimer le point G de pointsMurModifie  // retire le point G
                supprimer le point E de pointsMurModifie  // retire le point E
                supprimer le point C de pointsMurModifie  // retire le point C
                supprimer le point A de pointsMurModifie  // retire le point A

        sinon si mur.type est "Arriere" alors
            //traitement pour le mur arrière
        sinon si mur.type est "Gauche" alors
            //traitement pour le mur gauche
        sinon si mur.type est "Droite" alors
            //traitement pour le mur droit
        fin si
    retourner pointsMurModifie
fin fonction
```

Plan de travail (Gantt)



Contribution de chacun des membres de l’équipe

Les membres de l’équipe ont tous équitablement contribué à la conception du diagramme de classes de conception et des diagrammes de séquence de conception. Tous les membres ont aussi aidé à la rédaction du texte décrivant les classes et texte de l’architecture logique.

Cédric Fontaine a fait individuellement le texte du DSC pour déterminer l’élément sur lequel a lieu un clic de souris dans la vue d’un Mur et a fait la version fonctionnelle du projet.

Cédric Gagnon a fait individuellement le plan de travail (Gantt), ainsi que les textes de DSC pour le calcul du point sélectionné, et l’affichage vue du dessus

Cédric B. Marsolais a fait individuellement le texte du DSC pour la création d’une fenêtre et la mise en forme du document.

Maxime Mainardi a fait la modélisation des différents diagrammes de séquence de conception et du diagramme de classes de conception et a fait le pseudo code en grande majorité.

Raphaël Chheang a fait individuellement le texte du DSC pour l’affichage de la vue de la vue du dessus, Même si tous les membres ont aidé à le rédiger, il a aussi fait en majorité le texte décrivant les classes et leurs relations et le texte de l’architecture logique.