UNIVERSITÉ LAVAL

Le 14 octobre 2025

Livrable 2

Travail présenté à Anthony Deschênes dans le cadre du cours IFT-2007/GLO-2004 : Analyse et conception des systèmes orientés objets.

Par

Équipe 23

Kémila Bakary – 111 237 502

Wily Roussel Tatow – 536932870

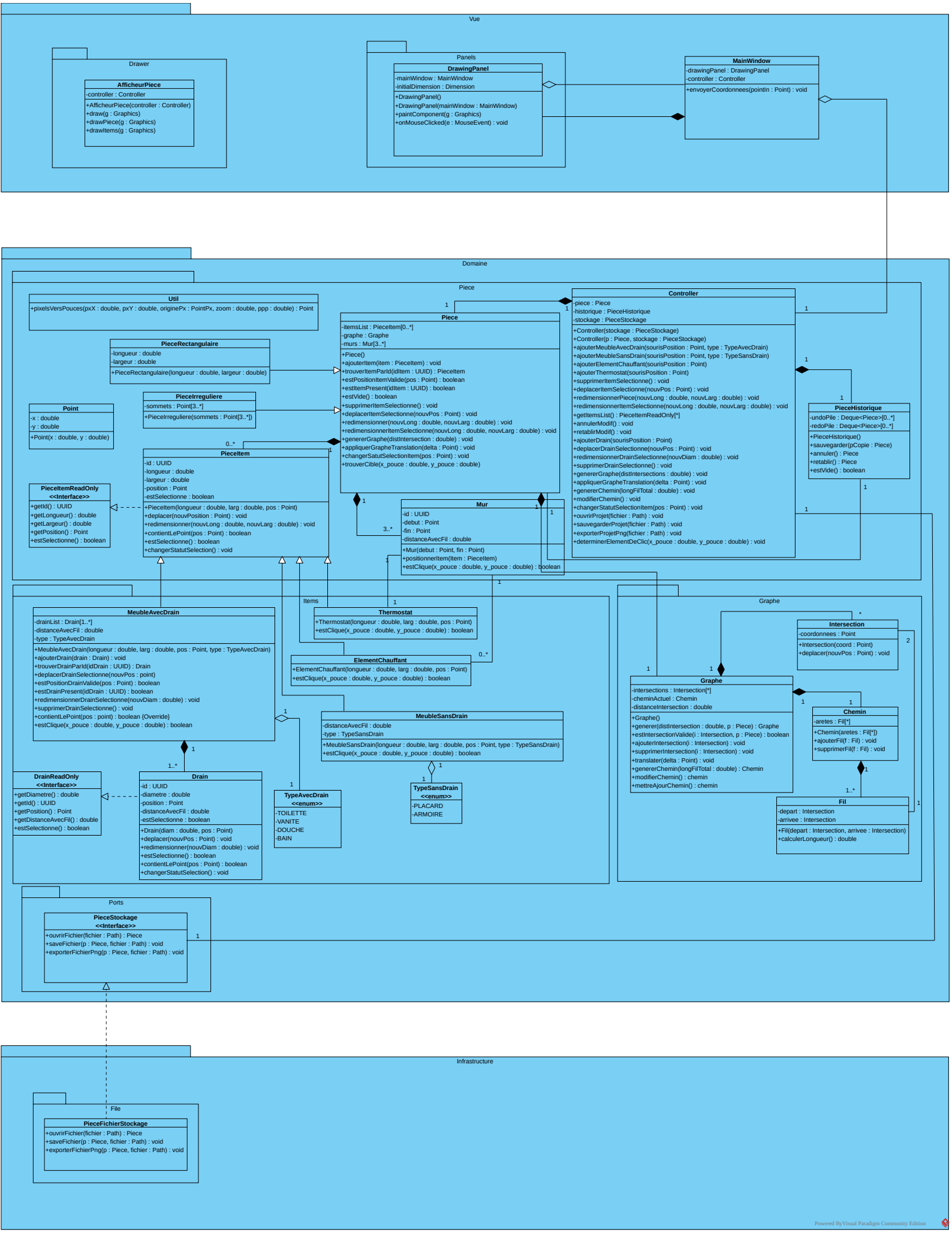
Petiton Wiseley Paul-Enzer – 537 047 716

OUEDRAOGO Aliya Imann – 537172383

DONGMEZA Christelle Murielle – 537095548

Table des matières

1. Diagramme de classe de conception


Vue
Drawer
AfficheurPiece
-controller : Controller
+AfficheurPiece(controller : Controller)
+draw(g : Graphics)
+drawPiece(g : Graphics)
+drawItems(g : Graphics)
Panels
DrawingPanel
-mainWindow : MainWindow
-initialDimension : Dimension
+DrawingPanel()
+DrawingPanel(mainWindow : MainWindow)
+paintComponent(g : Graphics)
+onMouseClicked(e : MouseEvent) : void
MainWindow
-drawingPanel : DrawingPanel
-controller : Controller
+envoyerCoordonnees(pointIn : Point) : void
Domaine
Piece
Util
+pixelsVersPouces(pxX : double, pxY : double, originePx : PointPx, zoom : double, ppp : double) : Point
Controller
-piece : Piece
-historique : PieceHistorique
-stockage : PieceStockage
+Controller(stockage : PieceStockage)
+Controller(p : Piece, stockage : PieceStockage)
+ajouterMeubleAvecDrain(sourisPosition : Point, type : TypeAvecDrain)
+ajouterMeubleSansDrain(sourisPosition : Point, type : TypeSansDrain)
+ajouterElementChauffant(sourisPosition : Point)
+ajouterThermostat(sourisPosition : Point)
+supprimerItemSelectionne() : void
+deplacerItemSelectionne(nouvPos : Point) : void
+redimensionnerPiece(nouvLong : double, nouvLarg : double) : void
+redimensionnerItemSelectionne(nouvLong : double, nouvLarg : double) : void
+getItemsList() : PieceItemReadOnly[*]
+annulerModif() : void
+retablirModif() : void
+ajouterDrain(sourisPosition : Point)
+deplacerDrainSelectionne(nouvPos : Point) : void
+redimensionnerDrainSelectionne(nouvDiam : double) : void
+supprimerDrainSelectionne() : void
+genererGraphe(distIntersections : double) : void
+appliquerGrapheTranslation(delta : Point) : void
+genererChemin(longFilTotal : double) : void
+modifierChemin() : void
+changerStatutSelectionItem(pos : Point) : void
+ouvrirProjet(fichier : Path) : void
+sauvegarderProjet(fichier : Path) : void
+exporterProjetPng(fichier : Path) : void
+determinerElementDeClic(x_pouce : double, y_pouce : double) : void
Piece
-itemsList : PieceItem[0..*]
-graphe : Graphe
-murs : Mur[3..*]
+Piece()
+ajouterItem(item : PieceItem) : void
+trouverItemParId(idItem : UUID) : PieceItem
+estPositionItemValide(pos : Point) : boolean
+estItemPresent(idItem : UUID) : boolean
+estVide() : boolean
+supprimerItemSelectionne() : void
+deplacerItemSelectionne(nouvPos : Point) : void
+redimensionner(nouvLong : double, nouvLarg : double) : void
+redimensionnerItemSelectionne(nouvLong : double, nouvLarg : double) : void
+genererGraphe(distIntersection : double) : void
+appliquerGrapheTranslation(delta : Point) : void
+changerSatutSelectionItem(pos : Point) : void
+trouverCible(x_pouce : double, y_pouce : double)
PieceRectangulaire
-longueur : double
-largeur : double
+PieceRectangulaire(longueur : double, largeur : double)
PieceIrreguliere
-sommets : Point[3..*]
+PieceIrreguliere(sommets : Point[3..*])
Point
-x : double
-y : double
+Point(x : double, y : double)
PieceHistorique
-undoPile : Deque<Piece>[0..*]
-redoPile : Deque<Piece>[0..*]
+PieceHistorique()
+sauvegarder(pCopie : Piece)
+annuler() : Piece
+retablir() : Piece
+estVide() : boolean
PieceItem
-id : UUID
-longueur : double
-largeur : double
-position : Point
-estSelectionne : boolean
+PieceItem(longueur : double, larg : double, pos : Point)
+deplacer(nouvPosition : Point) : void
+redimensionner(nouvLong : double, nouvLarg : double) : void
+contientLePoint(pos : Point) : boolean
+estSelectionne() : boolean
+changerStatutSelection() : void
PieceItemReadOnly
<<Interface>>
+getId() : UUID
+getLongueur() : double
+getLargeur() : double
+getPosition() : Point
+estSelectionne() : boolean
Mur
-id : UUID
-debut : Point
-fin : Point
-distanceAvecFil : double
+Mur(debut : Point, fin : Point)
+positionnerItem(Item : PieceItem)
+estClique(x_pouce : double, y_pouce : double) : boolean
Items
MeubleAvecDrain
-drainList : Drain[1..*]
-distanceAvecFil : double
-type : TypeAvecDrain
+MeubleAvecDrain(longueur : double, larg : double, pos : Point, type : TypeAvecDrain)
+ajouterDrain(drain : Drain) : void
+trouverDrainParId(idDrain : UUID) : Drain
+deplacerDrainSelectionne(nouvPos : point)
+estPositionDrainValide(pos : Point) : boolean
+estDrainPresent(idDrain : UUID) : boolean
+redimensionnerDrainSelectionne(nouvDiam : double) : void
+supprimerDrainSelectionne() : void
+contientLePoint(pos : point) : boolean {Override}
+estClique(x_pouce : double, y_pouce : double) : boolean
Thermostat
+Thermostat(longueur : double, larg : double, pos : Point)
+estClique(x_pouce : double, y_pouce : double) : boolean
ElementChauffant
+ElementChauffant(longueur : double, larg : double, pos : Point)
+estClique(x_pouce : double, y_pouce : double) : boolean
MeubleSansDrain
-distanceAvecFil : double
-type : TypeSansDrain
+MeubleSansDrain(longueur : double, larg : double, pos : Point, type : TypeSansDrain)
+estClique(x_pouce : double, y_pouce : double) : boolean
DrainReadOnly
<<Interface>>
+getDiametre() : double
+getId() : UUID
+getPosition() : Point
+getDistanceAvecFil() : double
+estSelectionne() : boolean
Drain
-id : UUID
-diametre : double
-position : Point
-distanceAvecFil : double
-estSelectionne : boolean
+Drain(diam : double, pos : Point)
+deplacer(nouvPos : Point) : void
+redimensionner(nouvDiam : double) : void
+estSelectionne() : boolean
+contientLePoint(pos : Point) : boolean
+changerStatutSelection() : void
TypeAvecDrain
<<enum>>
-TOILETTE
-VANITE
-DOUCHE
-BAIN
TypeSansDrain
<<enum>>
-PLACARD
-ARMOIRE
Graphe
Intersection
-coordonnees : Point
+Intersection(coord : Point)
+deplacer(nouvPos : Point) : void
Graphe
-intersections : Intersection[*]
-cheminActuel : Chemin
-distanceIntersection : double
+Graphe()
+generer(distIntersection : double, p : Piece) : Graphe
+estIntersectionValide(i : Intersection, p : Piece) : boolean
+ajouterIntersection(i : Intersection) : void
+supprimerIntersection(i : Intersection) : void
+translater(delta : Point) : void
+genererChemin(longFilTotal : double) : Chemin
+modifierChemin() : chemin
+mettreAjourChemin() : chemin
Chemin
-aretes : Fil[*]
+Chemin(aretes : Fil[*])
+ajouterFil(f : Fil) : void
+supprimerFil(f : Fil) : void
Fil
-depart : Intersection
-arrivee : Intersection
+Fil(depart : Intersection, arrivee : Intersection)
+calculerLongueur() : double
Ports
PieceStockage
<<Interface>>
+ouvrirFichier(fichier : Path) : Piece
+saveFichier(p : Piece, fichier : Path) : void
+exporterFichierPng(p : Piece, fichier : Path) : void
Infrastructure
File
PieceFichierStockage
+ouvrirFichier(fichier : Path) : Piece
+saveFichier(p : Piece, fichier : Path) : void
+exporterFichierPng(p : Piece, fichier : Path) : void
Powered ByVisual Paradigm Community Edition

- **MainWindow**
  Cette classe représente la fenêtre principale de l'application. C'est elle qui regroupe tous les éléments visuels, comme le panneau de dessin (DrawingPanel) et le contrôleur. Elle gère l'envoi des coordonnées de la souris au contrôleur quand l'utilisateur clique ou effectue une action. En résumé, MainWindow est la passerelle entre l'interface graphique utilisateur (gui) et la logique du programme.

- **DrawingPanel**
  C'est le panneau de dessin où la pièce et ses éléments (meubles, drains, murs, fil chauffant, etc...) sont affichés. Il est directement relié à la fenêtre principale. DrawingPanel gère le rendu graphique grâce à la méthode paintComponent (Graphics). C'est la surface de travail où l'utilisateur peut faire ses simulations.

- **AfficheurPiece**
  Cette classe est responsable de l'affichage de la pièce dans le panneau de dessin. Elle utilise le contrôleur pour accéder aux données de la pièce et dessiner chaque élément. Les méthodes drawPiece() et drawItems() s'occupent de tracer respectivement les contours de la pièce et les différents objets qu'elle contient (meubles, drains, thermostat, etc...). AfficheurPiece fait donc le lien entre les données du modèle et leur représentation graphique.

- **Controller**
  Le contrôleur est le point d'entrée principal du système. C'est lui qui reçoit les actions de l'utilisateur (par exemple : ajouter une douche, déplacer un meuble ou sauvegarder un projet). Il applique le principe du contrôleur de Larman, c'est-à-dire qu'il ne fait pas directement les calculs, mais coordonne les autres classes. Il communique avec la classe Piece pour gérer les objets de la pièce, avec la classe Graphe pour les mises à jour du fil chauffant, et avec les classes de stockage pour sauvegarder ou exporter les projets.

- **Piece**
  Cette classe représente la pièce où sera installé le plancher chauffant. Elle contient plusieurs éléments comme les meubles, les murs, le thermostat ou encore le graphe. Elle gère les principales opérations liées à la pièce : ajouter, supprimer, déplacer ou redimensionner un objet. Elle permet aussi de gérer la sélection d'un item et de générer le graphe représentant la membrane chauffante.

- **PieceItem**
  C'est la classe de base pour tous les éléments qu'on peut placer dans la pièce (meubles, drains, thermostat, éléments chauffants, etc.). Elle contient les attributs communs comme un identifiant, la position (sous forme d'un Point), la taille et l'état de sélection. Elle définit aussi des fonctions communes, comme déplacer un objet, redimensionner un élément ou vérifier si la souris clique sur lui. Les autres classes (comme MeubleAvecDrain ou Thermostat) héritent de cette classe et réutilisent son comportement.

- **PieceItemReadOnly (interface)**
  Cette interface fournit une version en lecture seule d'un PieceItem. Elle permet à certaines parties du programme, comme l'affichage, d'accéder aux informations d'un élément (sa position, sa taille, son état) sans pouvoir les modifier. Elle sert aussi à protéger le modèle contre les changements non désirés.

- **Point**
  La classe Point représente une position dans la pièce avec deux coordonnées : x et y. Ces coordonnées sont exprimées en pouces plutôt qu'en pixels pour garder des valeurs précises et indépendantes de la résolution de l'écran. Les points servent à définir les positions des éléments, des drains, des intersections dans le graphe, ainsi que l'identification du clic de souris.

- **Mur**
  Chaque mur définit un segment qui forme la limite de la pièce. Un mur possède un point de départ et un point de fin, tous deux représentés par des objets Point. Les murs permettent d'aligner certains éléments comme l'élément chauffant, les drains ou les thermostats.

- **PieceRectangulaire / PieceIrreguliere**
  Ces deux classes représentent les deux formes possibles d'une pièce.
  - *PieceRectangulaire* stocke la longueur et la largeur de la pièce et permet un redimensionnement simple.
  - *PieceIrreguliere* contient une liste de points représentant les coins de la pièce, ce qui permet de gérer des formes plus complexes.

  Ces classes sont utilisées pour créer les murs et valider la surface totale de la pièce.

- **MeubleAvecDrain**
  C'est une classe qui représente un meuble nécessitant un drain, comme une douche, un bain ou une toilette. Elle hérite de PieceItem et ajoute une liste de drains (composition). Elle permet d'ajouter, de supprimer ou de déplacer un drain, et de vérifier si sa position respecte les contraintes (distance minimale avec les murs ou le fil chauffant). Elle contient aussi un type (TypeAvecDrain) qui indique la catégorie du meuble.

- **MeubleSansDrain**
  Cette classe représente les meubles qui n'ont pas de drain, comme une armoire ou un

placard. Elle hérite aussi de PieceItem. Même si elle est plus simple, elle doit respecter certaines contraintes de distance par rapport au fil chauffant.

- **Drain**
  Le drain est un élément qui appartient à un meuble avec drain. Il est créé en même temps que le meuble ou ajouté après. Il a un diamètre et une position dans la pièce. Il peut être déplacé ou redimensionné si nécessaire. Le drain est souvent utilisé comme point de référence pour calculer les zones où le fil ne doit pas passer. Une version lecture seule (DrainReadOnly) existe pour l'affichage.

- **Thermostat**
  Cette classe représente le thermostat du système de plancher chauffant. Il est placé sur un mur ou à proximité et permet de contrôler la température. Il sert aussi souvent de point de départ du fil chauffant. Il hérite de PieceItem et possède des attributs liés à sa position et à sa taille.

- **ElementChauffant**
  L'élément chauffant représente une portion du fil électrique chauffant. Il hérite lui aussi de PieceItem. Il est utilisé lors des calculs du graphe pour vérifier la couverture du plancher et s'assurer que les contraintes (distance entre fils, distance aux murs) sont respectées.

- **TypeAvecDrain / TypeSansDrain**
  Ces deux énumérations servent à classer les types de meubles selon qu'ils possèdent ou non un drain. Par exemple, une douche ou une toilette appartiennent à TypeAvecDrain, tandis qu'une armoire appartient à TypeSansDrain. Elles facilitent la création et la validation des objets dans le programme.

- **Graphe**
  Le graphe représente la structure logique du plancher chauffant. Il est composé de plusieurs intersections (les points de la membrane) et d'un chemin (le parcours du fil). Il sert à générer la grille, à calculer le tracé du fil, à vérifier les contraintes de distance et à mettre à jour le chemin si un nouvel élément est ajouté. Il est directement lié à la classe Piece.

- **Intersection**
  Chaque intersection est un point du graphe où deux lignes de la membrane se croisent. Elle contient ses coordonnées et ses liens avec les autres intersections du chemin. Elle sert à construire le tracé complet du fil chauffant.

- **Chemin**
  Le chemin correspond au parcours du fil chauffant sur la membrane. Il est composé d'une liste ordonnée d'intersections et permet de calculer la longueur totale du fil. Il peut être régénéré automatiquement si la configuration de la pièce change.

- **Util**
  Cette classe contient des fonctions utilitaires utilisées un peu partout dans le programme. Par exemple, elle permet de convertir les coordonnées entre pixels et pouces, de faire des calculs de distance ou de gérer des valeurs numériques. Elle ne contient pas d'état, seulement des méthodes statiques.

- **PieceStockage**
  C'est une interface qui sert à gérer la sauvegarde et le chargement des projets. Elle permet d'enregistrer une pièce dans un fichier, de l'ouvrir plus tard ou de l'exporter en image (PNG) ou en PDF. Le contrôleur utilise cette interface pour communiquer avec les fichiers sans dépendre d'un format particulier.

- **PieceFichierStockage (Infrastructure)**
  Cette classe fait partie de la couche Infrastructure. Elle s'occupe de la gestion des fichiers liés au projet. Elle permet d'ouvrir un fichier pour charger une pièce existante, d'enregistrer les modifications effectuées, ou encore d'exporter la pièce sous forme d'image (PNG). Les méthodes ouvrirFichier(), saveFichier() et exporterFichierPng() assurent la communication entre le système et le disque dur. Elle implémente concrètement l'interface PieceStockage utilisée par le contrôleur.

2. L'architecture logique

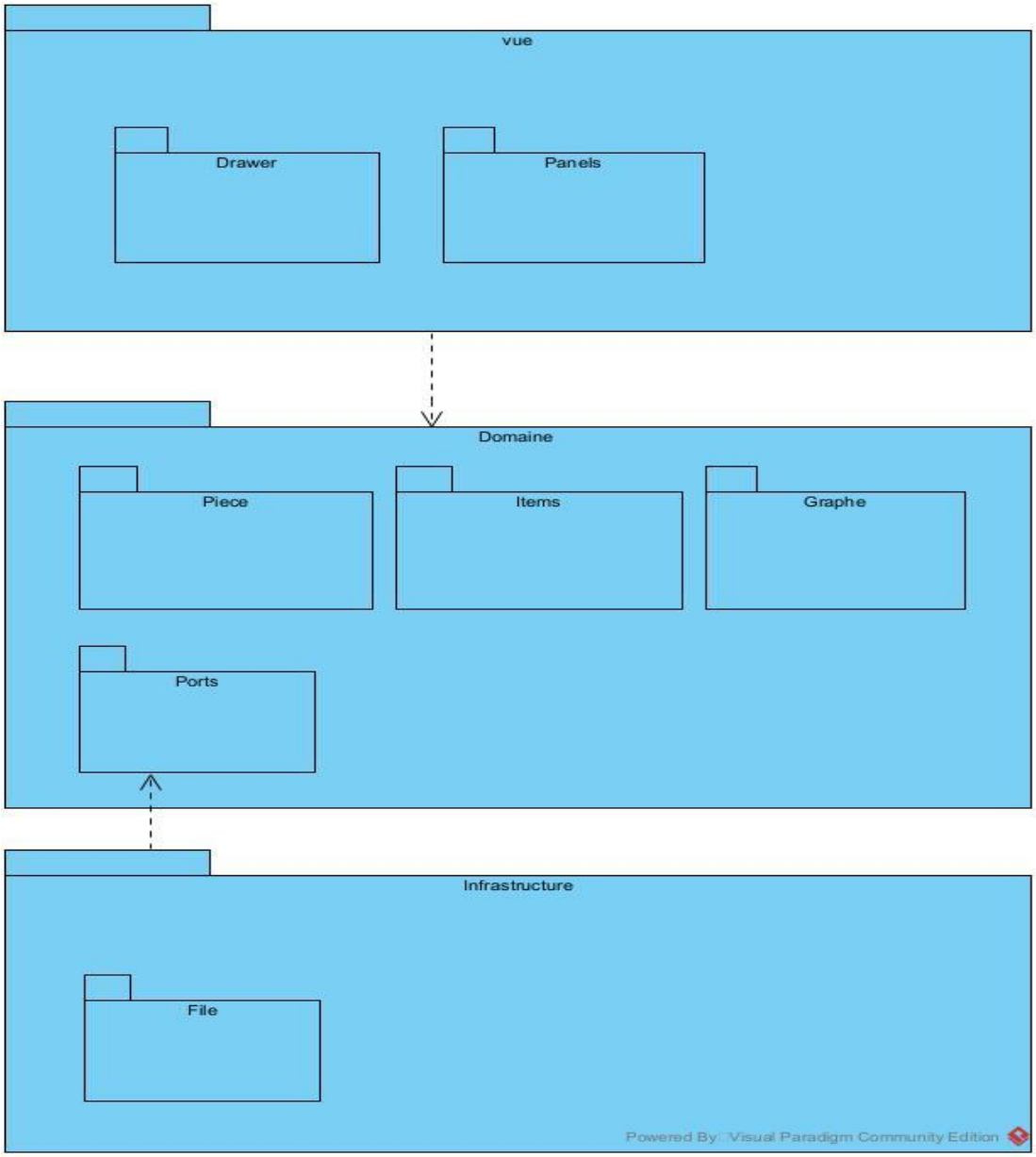
Le diagramme suivant illustre l'architecture logique du projet *HeatMyFloor*.

Le projet est organisé selon une architecture multicouche, composée des packages **Vue**, **Domaine** et **Infrastructure**.

Le package **Vue** regroupe les éléments qui concernent l'interface utilisateur, c'est-à-dire tout ce qui permet d'afficher les informations et d'interagir avec le système.
Le package **Domaine** contient la logique principale du programme, c'est-à-dire les éléments qui représentent le cœur fonctionnel du système, les règles métiers et la gestion interne des données.
Enfin, le package **Infrastructure** s'occupe de la partie technique, notamment de la gestion des fichiers et de la persistance des données.

Les dépendances entre les packages suivent une logique descendante : la **Vue** dépend du **Domaine** pour obtenir les informations à afficher, et le **Domaine** dépend de l'**Infrastructure** pour les opérations de lecture et d'écriture de données.
Ainsi, chaque couche peut être développée et testée de façon indépendante, ce qui facilite la maintenance et la réutilisation du code.

Cette structure hiérarchique permet de séparer clairement les responsabilités : l'interface se concentre sur la présentation, le domaine sur la logique métier et l'infrastructure sur les aspects techniques.



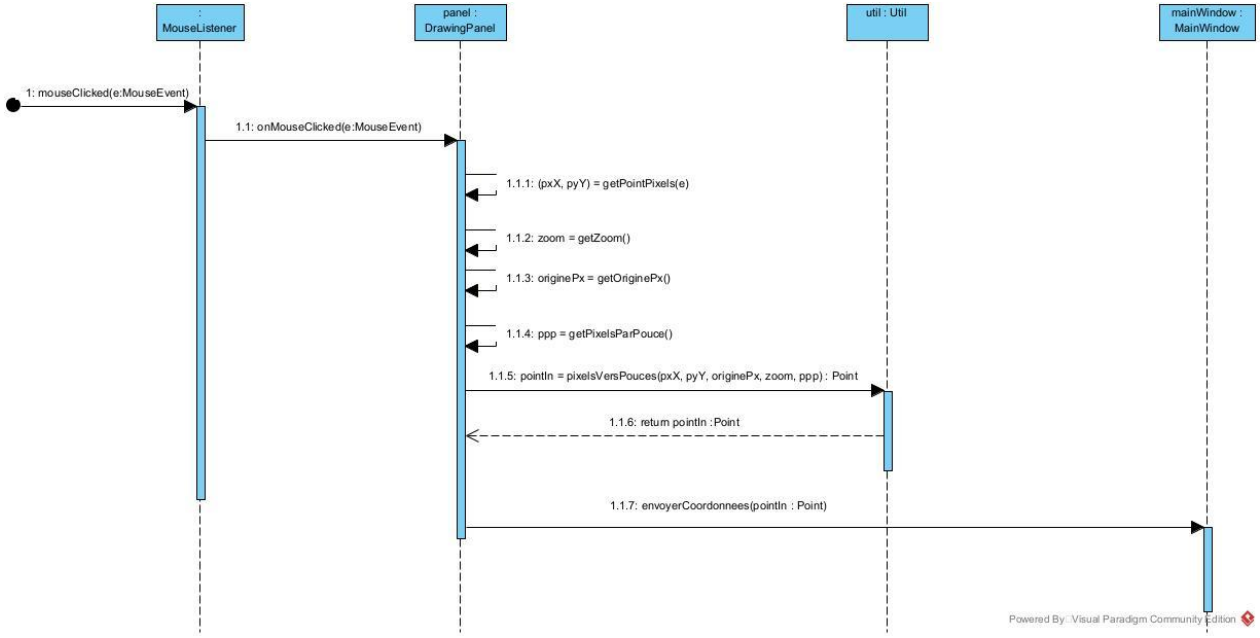
3. Diagramme de séquence de conception (DSC)

Les diagrammes de séquence de conception (DSC) présentés ci-dessous décrivent en détail les interactions entre différentes classes du projet. Ils montrent comment les objets communiquent

entre eux en réponse à des événements tels que les clics de souris, l'ajout d'un meuble ou l'affichage de la vue.

3.1 Déterminer l'élément sur lequel a lieu un clic de souris

3.1.1 Un appel externe fait par Java lors d'un clic de souris dans le JPanel



Ce diagramme illustre le traitement d'un clic de souris dans le **DrawingPanel** et la conversion des coordonnées du clic en pouces avant leur transmission à la fenêtre principale.
Le scénario débute par un appel externe provenant du moteur d'événements **Java Swing**.

Lorsqu'un utilisateur clique à l'intérieur du panneau de dessin, **Java Swing** déclenche automatiquement la méthode **mouseClicked(MouseEvent e)** sur l'objet **MouseListener** associé au panneau.
Cet appel externe constitue le **point d'entrée** du diagramme.

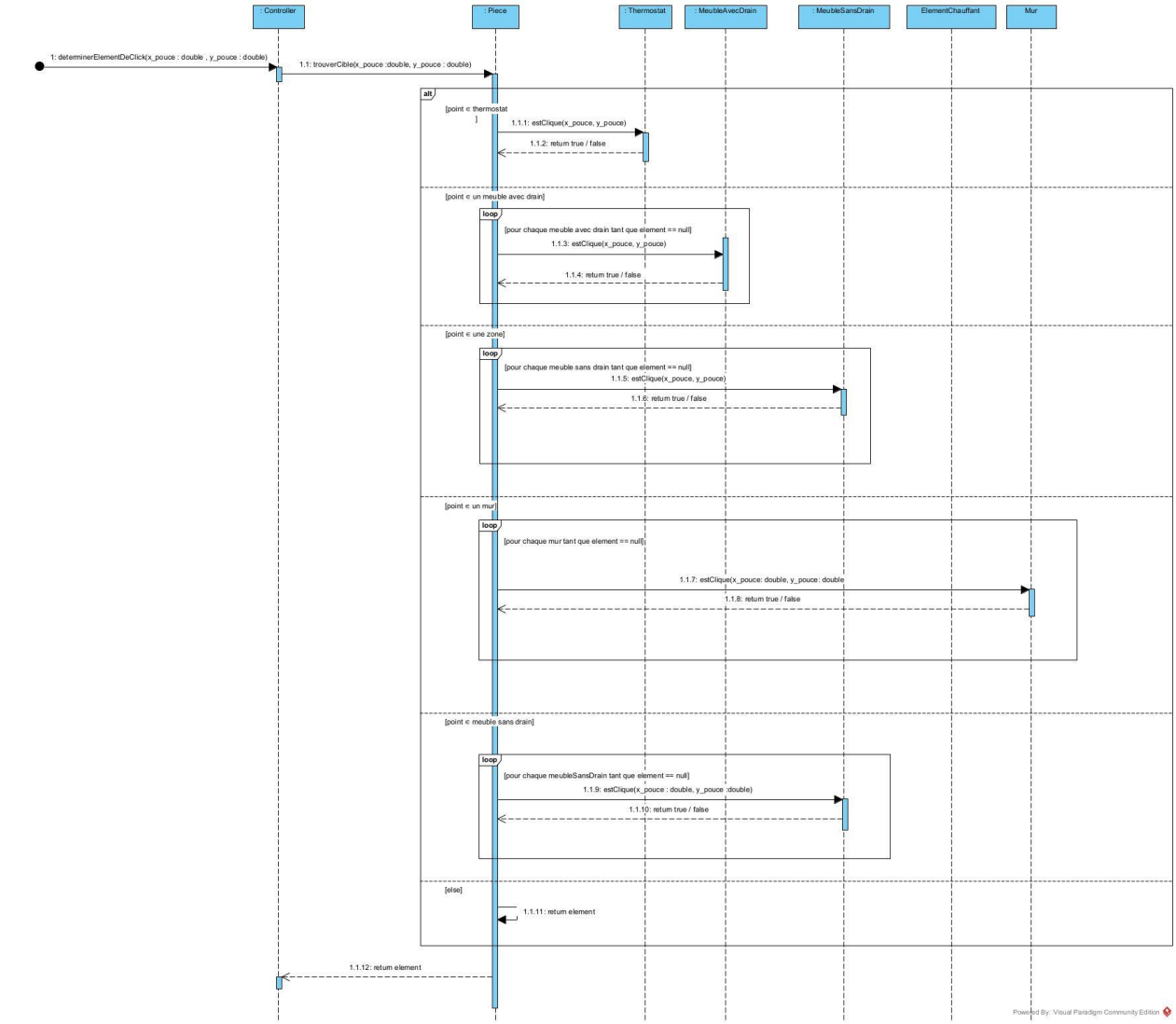
Le **MouseListener** délègue la gestion du clic à la méthode **onMouseClicked(e)** de la classe **DrawingPanel**.
Le panneau commence alors la lecture des données de l'événement :

- Extraire les coordonnées du clic en pixels via **getPointPixels(e)** ;
- Récupérer le facteur de zoom avec **getZoom()** ;
- Obtient la position de l'origine du plan en pixels via **getOriginePx()** ;
- Lit le nombre de pixels par pouce avec **getPixelsParPouce()**.

Le **DrawingPanel** appelle ensuite la méthode **pixelsVersPouces(pxX, pyY, originePx, zoom, ppp)** de la classe **Util**.
Cette classe utilitaire est responsable de la **conversion des coordonnées en pouces**, en tenant compte du facteur de zoom et de l'origine du repère.
Le résultat de cet appel est un **objet Point**, représentant la position du clic exprimée en pouces par rapport à la référence du plan.

La méthode de la classe **Util** retourne ce point au **DrawingPanel**, qui le transmet à la fenêtre principale **MainWindow** via la méthode **envoyerCoordonnees(pointIn)**.
Ce transfert rend les coordonnées disponibles pour le **contrôleur**, qui se chargera, dans le **DSC 3.1.2**, de déterminer sur quel élément du plan (meuble, thermostat, élément chauffant, etc.) le clic a réellement eu lieu.

3.1.2 Un appel au contrôleur auquel on passe les coordonnées du point sur lequel l'utilisateur a cliqué



Le scénario débute lorsque la fenêtre principale ou le panneau de dessin envoie les coordonnées du clic au **Controleur**, qui appelle la méthode **determinerElementDeClic(x_pouce, y_pouce)**. Cet appel constitue le point d'entrée de la séquence.

Le **Controleur** délègue ensuite la recherche à la méthode **trouverCible(x_pouce, y_pouce)** de la classe **Piece**.
C'est cette méthode qui orchestre la vérification de chaque type d'élément contenu dans la pièce.

La classe **Piece** effectue une série de vérifications représentées par un grand **bloc alt** et plusieurs **boucles loop**.
Elle parcourt successivement les différents types d'éléments présents dans la pièce — **thermostats, meubles avec drain, meubles sans drain, éléments chauffants et murs** — en appelant, pour chacun, la méthode **estClique(x_pouce, y_pouce)**.
Chaque boucle se poursuit tant qu'aucun élément n'a été trouvé (condition **element == null**).

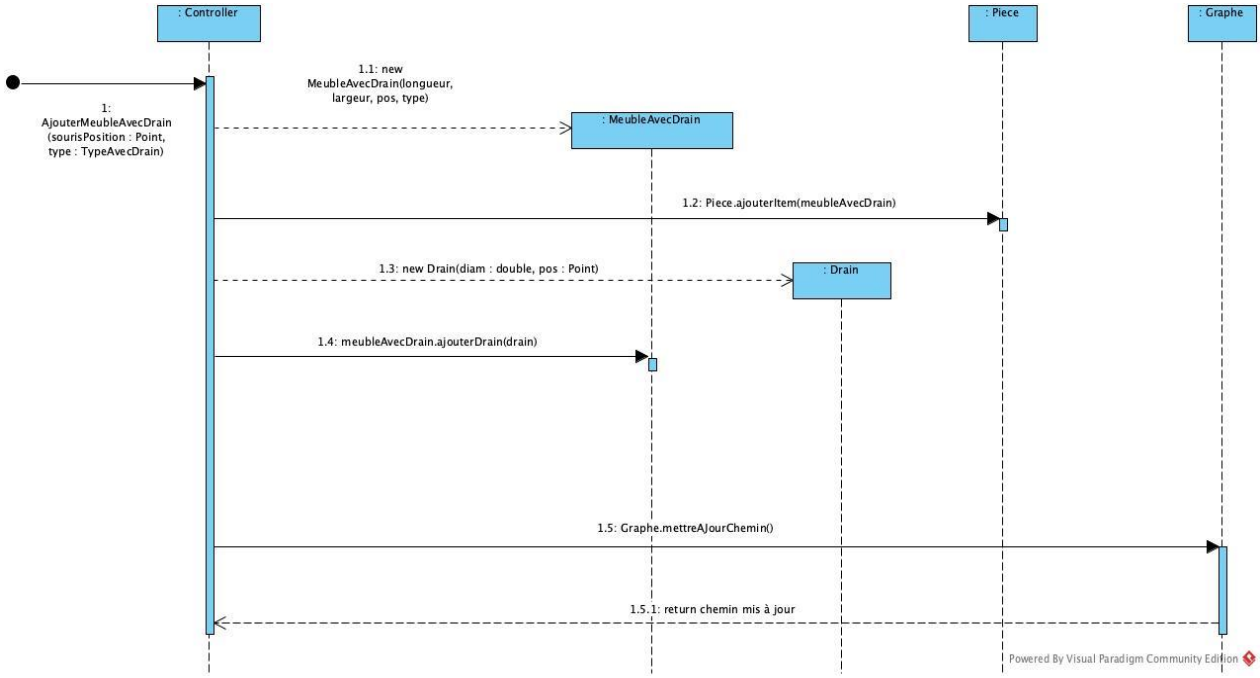
Cette approche permet d'identifier efficacement l'objet cliqué dans le plan tout en interrompant la recherche dès qu'un élément est reconnu.

Si aucun élément ne satisfait les conditions de clic, la classe **Piece** assigne **element = null** dans la branche **[else]** du bloc **alt**.
Cela signifie qu'aucun objet cliquable n'a été trouvé à la position indiquée.

À la fin du traitement, la méthode **trouverCible(...) retourne l'élément trouvé** (ou **null**) au **Controleur** via un message de retour **return element**.

Le **Controleur** pourra ensuite, dans les étapes suivantes du système, transmettre cette information à la vue ou exécuter une action contextuelle selon le type d'élément sélectionné.

3.2 Ajouter une douche

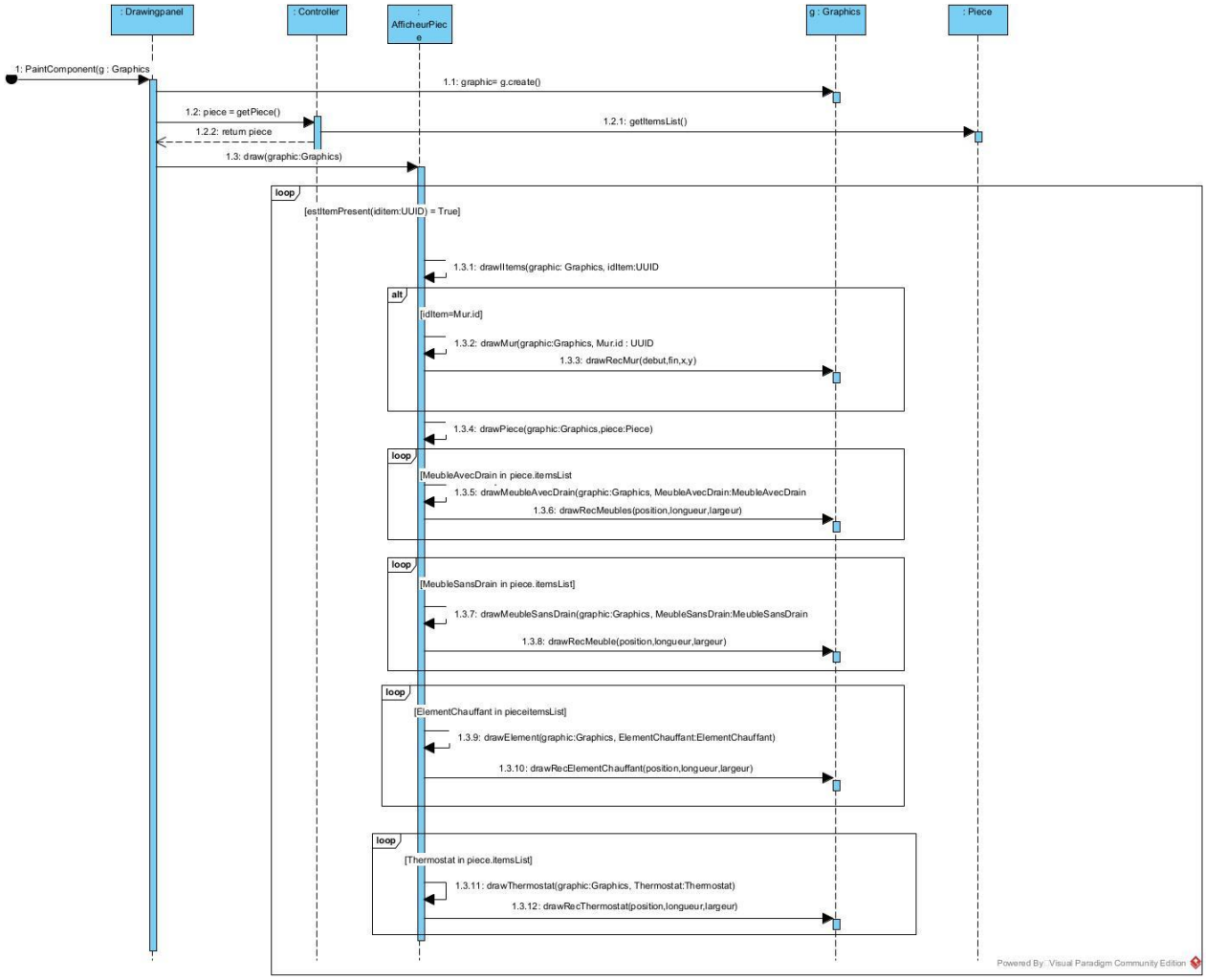


Le diagramme illustre la séquence d'opérations permettant d'ajouter une douche dans la pièce. Cette fonctionnalité correspond à l'appel de la méthode ajouterMeubleAvecDrain(sourisPosition : Point, type : TypeAvecDrain) du contrôleur, qui implémente le cas d'utilisation « Ajouter une douche ». Le contrôleur crée d'abord un nouvel objet MeubleAvecDrain à l'aide du constructeur new MeubleAvecDrain(longueur, largeur, pos, type). L'objet est ensuite ajouté à la pièce par la méthode Piece.ajouterItem(meubleAvecDrain), ce qui l'insère dans la liste des éléments du plancher. Le contrôleur instancie ensuite un nouvel objet Drain via new Drain(diam : double, pos

: Point), et rattache ce drain au meuble correspondant en appelant meubleAvecDrain.ajouterDrain(drain) . Enfin, la mise à jour du graphe du plancher chauffant est effectuée à l'aide de Graphe.mettreAJourChemin() , lequel retourne le chemin recalculé .

Ainsi, le diagramme montre comment une douche est ajoutée en tant que meuble avec drain, puis reliée à son drain et intégrée au réseau du plancher chauffant.

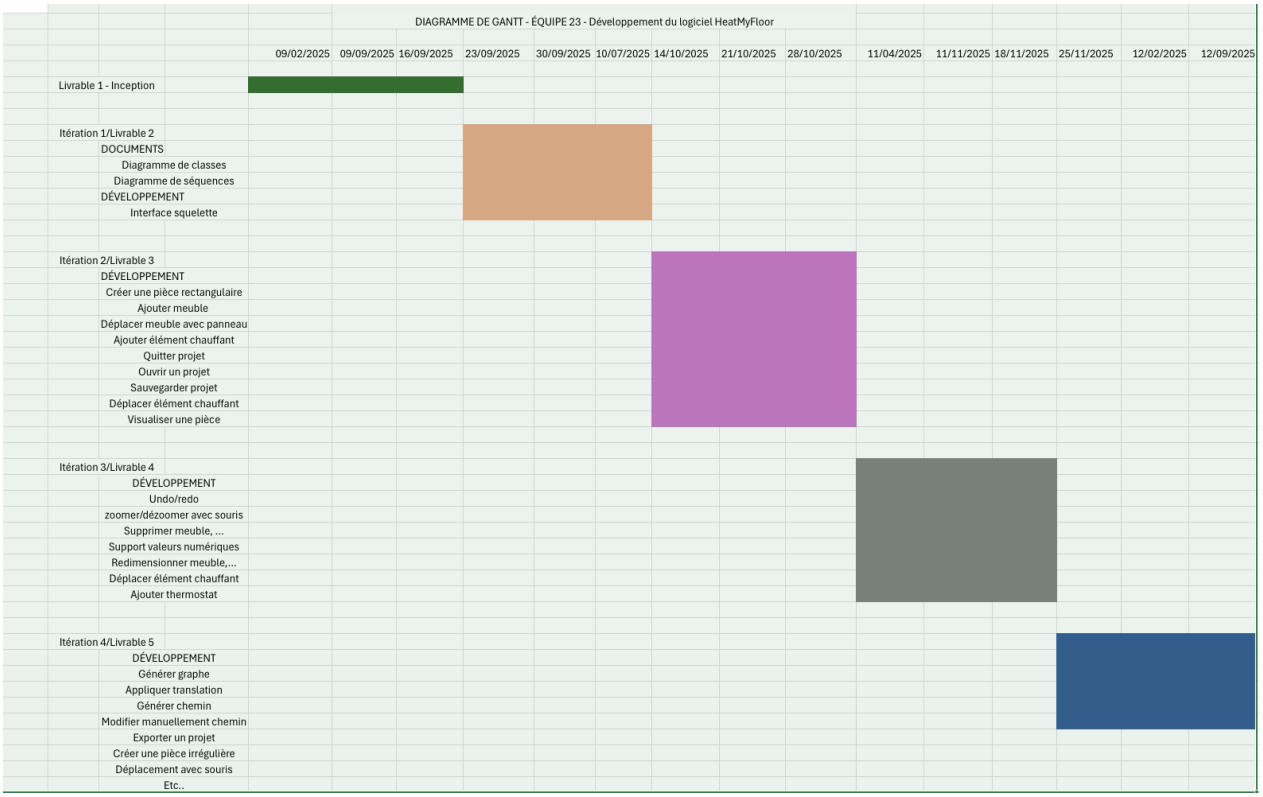
3.3 Affichage de la vue



L'affichage de la *vue*, représentée par le *DrawingPanel*, commence lorsque Swing demande un rafraîchissement via la méthode *paintComponent(Graphics g)*. À ce moment-là, la vue prépare le contexte graphique, puis transmet la tâche au *Controller* en appelant *draw(g)*. Le *Controller* agit

comme un pont entre la vue et le modèle : il récupère la pièce à afficher avec *getPiece()* et demande à l'*AfficheurPiece* de dessiner les éléments. Ce dernier parcourt les objets du modèle — murs, meubles (AvecDrain et SansDrain), éléments chauffants et thermostats — et utilise des méthodes spécifiques comme *drawMur()*, *drawMeubleSansDrain(),drawMeubleAvecDrain()*, *drawElement()*et *drawThermostat* () pour les afficher. Chaque boucle de dessin correspond à une collection d'objets dans la pièce. Une fois le dessin terminé, le contrôle revient à l'*AfficheurPiece*, puis au *Controller*, et enfin à la *vue*, qui montre le résultat final à l'écran.

4. Diagramme de Gantt



5. Contribution de chacun des membres de l'équipe

Diagramme de classe de conception: tous les membres de l'équipe

DSC: Kémila Bakary, OUEDRAOGO Aliya Imann, DONGMEZA Christelle Murielle

Code de l’interface: Wily Tatow

Texte explicatif des classes: OUEDRAOGO Aliya Imann, Wily Tatow, DONGMEZA Christelle Murielle

Diagramme de Gantt: Petiton Wiseley

Architecture logique: Kémila Bakary, Petiton Wiseley

Nous avons eu deux réunions par semaine et chaque membre de l’équipe a pu contribuer et valider toutes les parties du livrable.

6. Version fonctionnelle du projet

HeatMyFloor s’ouvre sur une interface en trois zones.

En haut, un ruban d’outils regroupe les commandes par thèmes : Projet (Nouveau, Ouvrir, Enregistrer, Exporter), Modélisation (Pièce, Fil), Affichage (Vue 2D, Vue 3D), Formes (Rectangle, Cercle), Meubles (Sans drain, Avec drain) et Autres (Thermostat, Zones).

À gauche, le panneau Propriétés propose des cartes de saisie claires et défilables : dimensions de la pièce, paramètres de membrane, dimensions de meuble et réglages du fil, avec des boutons dédiés (Générer un graphe, Générer un chemin).

Au centre, l’espace de modélisation, organisé en onglets (Projet 1, Projet 2, …), affiche la scène sur un canevas gris et réagit aux outils actifs (sélection, création, etc.).

En bas, une barre d’état indique la position (X, Y), la translation et la rotation de l’élément courant, tandis qu’une zone Messages d’erreur à droite présente les alertes contextuelles (p. ex. déplacement interdit).

Pour référence, le squelette exécutable de cette interface est fourni dans le fichier **equipe23.jar** inclus avec le projet.