24/09/2019

VirtuTuile

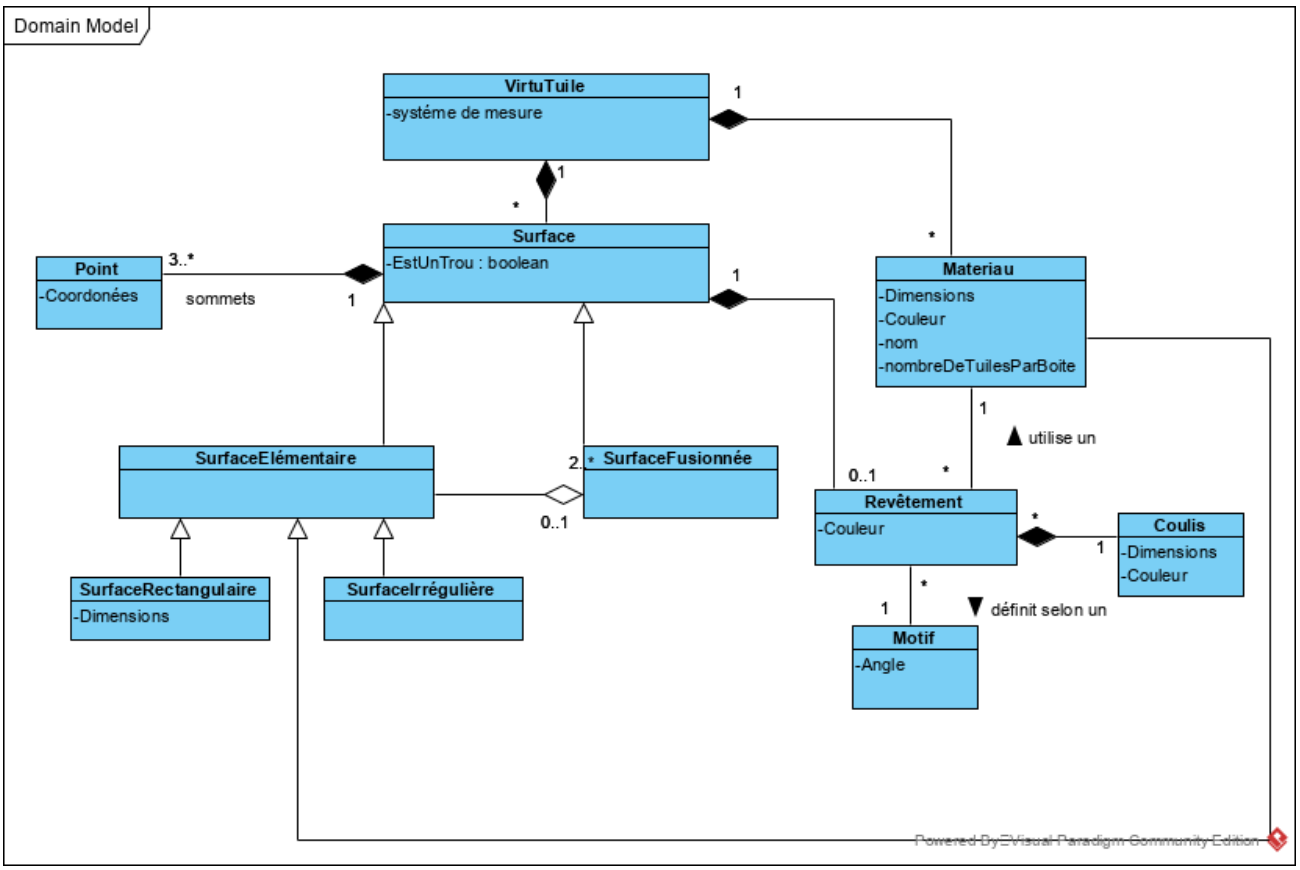
Livrable 3

Nom du rédacteur	Date	Changements
Martin Cotoni	27 septembre 2019	Création du document
Martin Cotoni Thomas Lombard	14 octobre 2019	Refonte de / des / du: - L'énoncé de vision - Modèle du domaine - Modèle des cas d'utilisation Actualisation de / des / du : - Diagramme de Gant Ajout des demandes du Livrable 2
Martin Cotoni	24 novembre 2019	Livrable 3

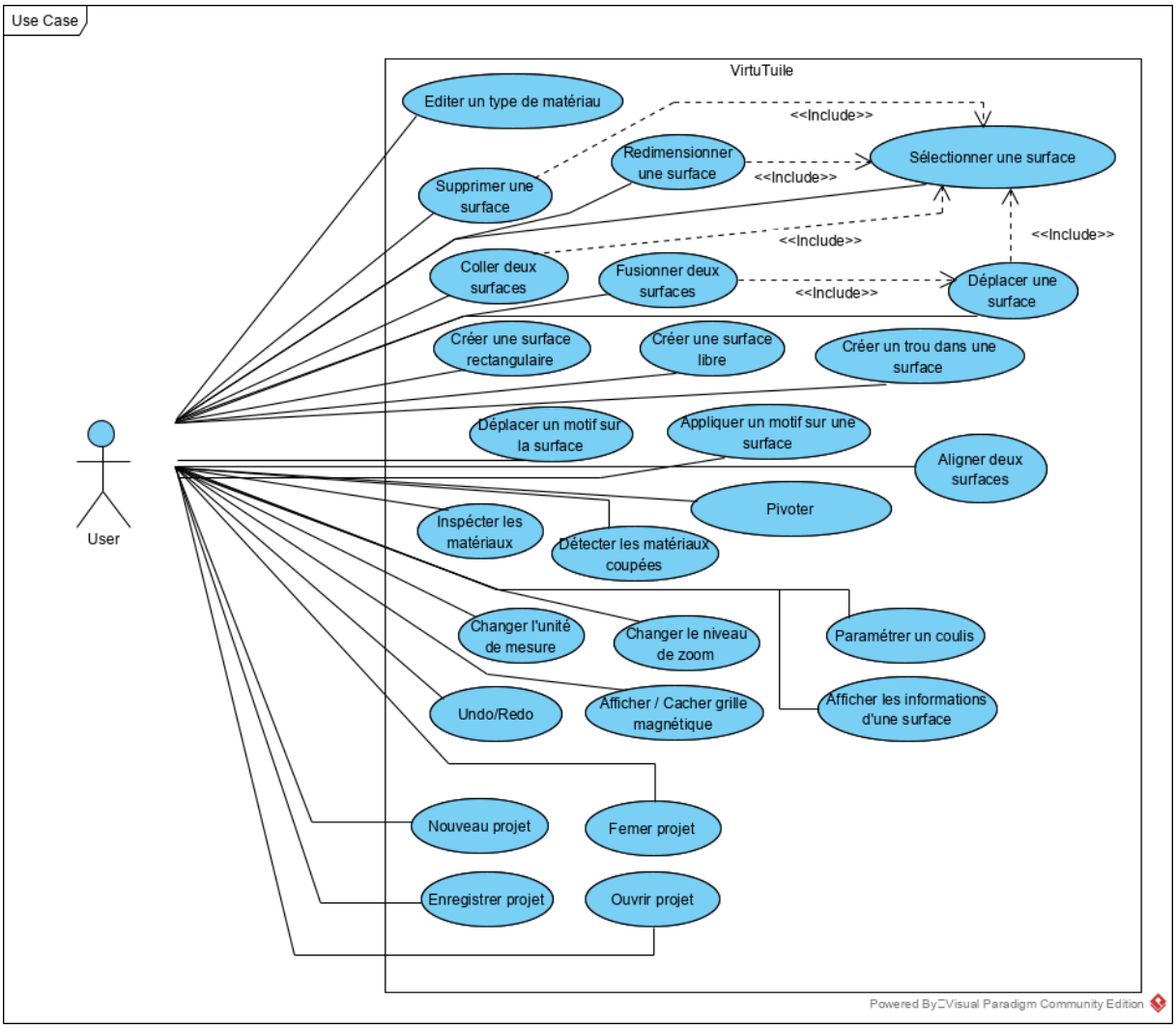
EQUIPE 30
M.FOURRIER – A.PELLETANT – T.LOMBARD – M.COTONI

Table des matières

I. Modèle du domaine

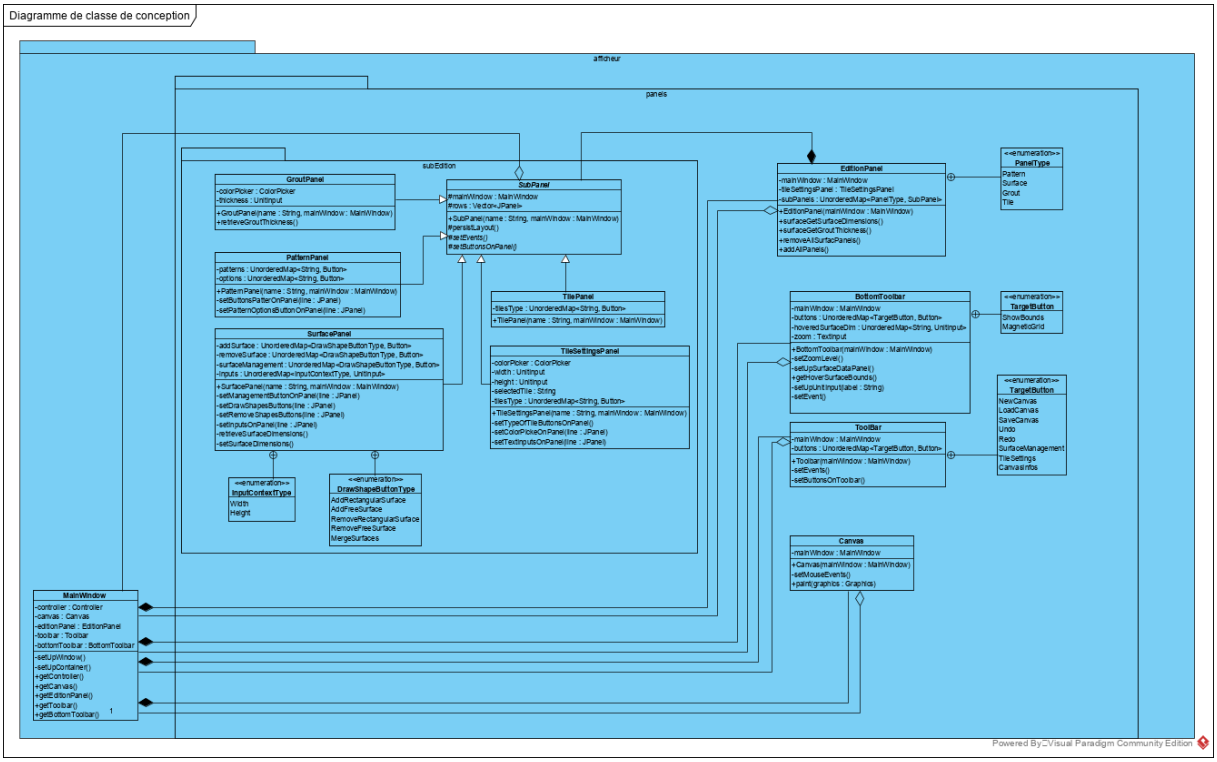


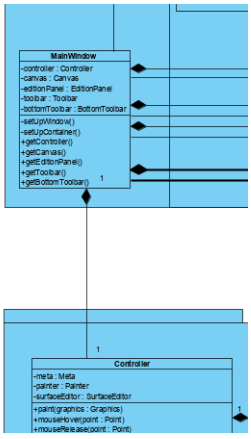
II. Modèle des cas d'utilisation



III. Modèle de conception

Pour un soucis de lisibilité, le diagramme à été coupé en deux sur ce document, avec en première partie, le package Afficheur puis « un zoom » sur la relation entre **MainWindow** et **Controller** et enfin le package Domaine.





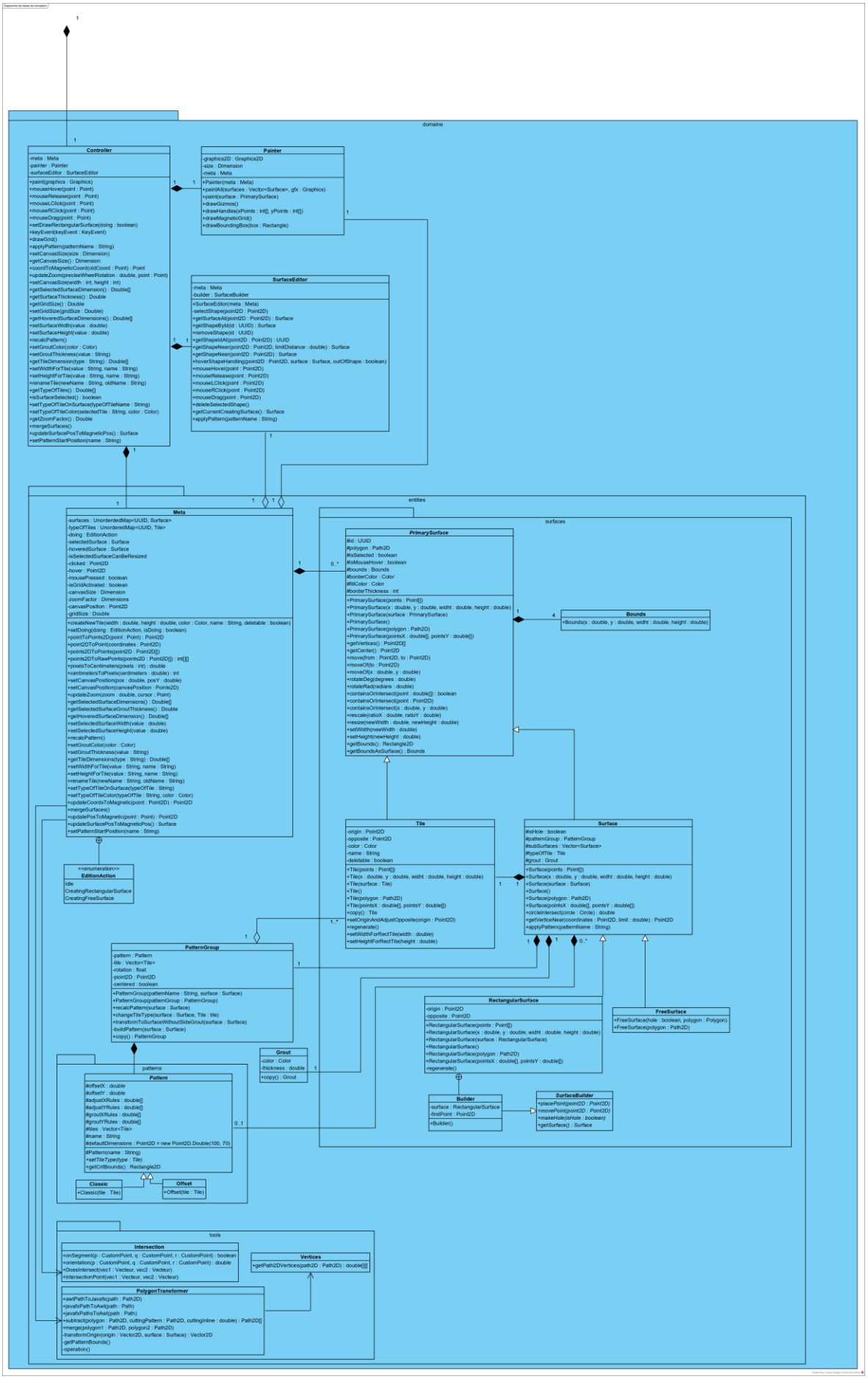


Diagramme de classe de conception
domaine
Controller
-meta : Meta
-painter : Painter
-surfaceEditor : SurfaceEditor
+paint(graphics : Graphics)
+mouseHover(point : Point)
+mouseRelease(point : Point)
+mouseLClick(point : Point)
+mouseRClick(point : Point)
+mouseDrag(point : Point)
+setDrawRectangularSurface(doing : boolean)
+keyEvent(keyEvent : KeyEvent)
+drawGrid()
+applyPattern(patternName : String)
+setCanvasSize(size : Dimension)
+getCanvasSize() : Dimension
+coordToMagneticCoord(oldCoord : Point) : Point
+updateZoom(precaiseWheelRotation : double, point : Point)
+setCanvasSize(width : int, height : int)
+getSelectedSurfaceDimension() : Double[]
+getSurfaceThickness() : Double
+getGridSize() : Double
+setGridSize(gridSize : Double)
+getHoveredSurfaceDimensions() : Double[]
+setSurfaceWidth(value : double)
+setSurfaceHeight(value : double)
+recalcPattern()
+setGroutColor(color : Color)
+setGroutThickness(value : String)
+getTileDimension(type : String) : Double[]
+setWidthForTile(value : String, name : String)
+setHeightForTile(value : String, name : String)
+renameTile(newName : String, oldName : String)
+getTypeOfTiles() : Double[]
+isSurfaceSelected() : boolean
+setTypeOfTileOnSurface(typeOfTileName : String)
+setTypeOfTileColor(selectedTile : String, color : Color)
+getZoomFactor() : Double
+mergeSurfaces()
+updateSurfacePosToMagneticPos() : Surface
+setPatternStartPosition(name : String)
Painter
-graphics2D : Graphics2D
-size : Dimension
-meta : Meta
+Painter(meta : Meta)
+paintAll(surfaces : Vector<Surface>, gfx : Graphics)
+paint(surface : PrimarySurface)
+drawGizmos()
+drawHandles(xPoints : int[], yPoints : int[])
+drawMagneticGrid()
+drawBoundingBox(box : Rectangle)
SurfaceEditor
-meta : Meta
-builder : SurfaceBuilder
+SurfaceEditor(meta : Meta)
-selectShape(point2D : Point2D)
+getSurfaceAt(point2D : Point2D) : Surface
+getShapeById(id : UUID) : Surface
+removeShape(id : UUID)
+getShapeIdAt(point2D : Point2D) : UUID
+getShapeNear(point2D : Point2D, limitDistance : double) : Surface
+getShapeNear(point2D : Point2D) : Surface
+hoverShapeHandling(point2D : Point2D, surface : Surface, outOfShape : boolean)
+mouseHover(point : Point2D)
+mouseRelease(point : Point2D)
+mouseLClick(point : Point2D)
+mouseRClick(point : Point2D)
+mouseDrag(point : Point2D)
+deleteSelectedShape()
+getCurrentCreatingSurface() : Surface
+applyPattern(patternName : String)
entities
Meta
-surfaces : UnorderdedMap<UUID, Surface>
-typeOfTiles : UnorderedMap<UUID, Tile>
-doing : EditionAction
-selectedSurface : Surface
-hoveredSurface : Surface
-isSelectedSurfaceCanBeResized
-clicked : Point2D
-hover : Point2D
-mousePressed : boolean
-isGridActivated : boolean
-canvasSize : Dimension
-zoomFactor : Dimensions
-canvasPosition : Point2D
-gridSize : Double
+createNewTile(width : double, height : double, color : Color, name : String, deletable : boolean)
+setDoing(doing : EditionAction, isDoing : boolean)
+pointToPoints2D(point : Point) : Point2D
+point2DToPoint(coordinates : Point2D)
+points2DToPoints(point2D : Point2D[])
+points2DToRawPoints(points2D : Point2D[]) : int[][]
+pixelsToCentimeters(pixels : int) : double
+centimetersToPixels(centimeters : double) : int
+setCanvasPosition(pox : double, posY : double)
+setCanvasPosition(canvasPosition : Point2D)
+updateZoom(zoom : double, cursor : Point)
+getSelectedSurfaceDimensions() : Double[]
+getSelectedSurfaceGroutThickness() : Double
+getHoveredSurfaceDimension() : Double[]
+setSelectedSurfaceWidth(value : double)
+setSelectedSurfaceHeight(value : double)
+recalcPattern()
+setGroutColor(color : Color)
+setGroutThickness(value : String)
+getTileDimensions(type : String) : Double[]
+setWidthForTile(value : String, name : String)
+setHeightForTile(value : String, name : String)
+renameTile(newName : String, oldName : String)
+setTypeOfTileOnSurface(typeOfTile : String)
+setTypeOfTileColor(typeOfTile : String, color : Color)
+updateCoordsToMagnetic(point : Point2D) : Point2D
+mergeSurfaces()
+updatePosToMagnetic(point : Point) : Point2D
+updateSurfacePosToMagneticPos() : Surface
+setPatternStartPosition(name : String)
surfaces
PrimarySurface
#id : UUID
#polygon : Path2D
#isSelected : boolean
#isMouseHover : boolean
#bounds : Bounds
#borderColor : Color
#fillColor : Color
#borderThickness : int
+PrimarySurface(points : Point[])
+PrimarySurface(x : double, y : double, widht : double, height : double)
+PrimarySurface(surface : PrimarySurface)
+PrimarySurface()
+PrimarySurface(polygon : Path2D)
+PrimarySurface(pointsX : double[], pointsY : double[])
+getVertices() : Point2D[]
+getCenter() : Point2D
+move(from : Point2D, to : Point2D)
+moveOf(to : Point2D)
+moveOf(x : double, y : double)
+rotateDeg(degrees : double)
+rotateRad(radians : double)
+containsOrIntersect(point : double[]) : boolean
+containsOrIntersect(point : Point2D)
+containsOrIntersect(x : double, y : double)
+rescale(ratioX : double, ratioY : double)
+resize(newWidth : double, newHeight : double)
+setWidth(newWidth : double)
+setHeight(newHeight : double)
+getBounds() : Rectangle2D
+getBoundsAsSurface() : Bounds
Bounds
+Bounds(x : double, y : double, widht : double, height : double)
Tile
-origin : Point2D
-opposite : Point2D
-color : Color
-name : String
-deletable : boolean
+Tile(points : Point[])
+Tile(x : double, y : double, widht : double, height : double)
+Tile(surface : Tile)
+Tile()
+Tile(polygon : Path2D)
+Tile(pointsX : double[], pointsY : double[])
+copy() : Tile
+setOriginAndAdjustOpposite(origin : Point2D)
+regenerate()
+setWidthForRectTile(width : double)
+setHeightForRectTile(height : double)
Surface
#isHole : boolean
#patternGroup : PatternGroup
#subSurfaces : Vector<Surface>
#typeOfTile : Tile
#grout : Grout
+Surface(points : Point[])
+Surface(x : double, y : double, widht : double, height : double)
+Surface(surface : Surface)
+Surface()
+Surface(polygon : Path2D)
+Surface(pointsX : double[], pointsY : double[])
+circleIntersect(circle : Circle) : double
+getVerticeNear(coordinates : Point2D, limit : double) : Point2D
+applyPattern(patternName : String)
<<enumeration>>
EditionAction
Idle
CreatingRectangularSurface
CreatingFreeSurface
PatternGroup
-pattern : Pattern
-tile : Vector<Tile>
-rotation : float
-point2D : Point2D
-centered : boolean
+PatternGroup(patternName : String, surface : Surface)
+PatternGroup(patternGroup : PatternGroup)
+recalcPattern(surface : Surface)
+changeTileType(surface : Surface, Tile : tile)
+transformToSurfaceWithoutSideGrout(surface : Surface)
-buildPattern(surface : Surface)
+copy() : PatternGroup
Grout
-color : Color
-thickness : double
+copy() : Grout
RectangularSurface
-origin : Point2D
-opposite : Point2D
+RectangularSurface(points : Point[])
+RectangularSurface(x : double, y : double, widht : double, height : double)
+RectangularSurface(surface : RectangularSurface)
+RectangularSurface()
+RectangularSurface(polygon : Path2D)
+RectangularSurface(pointsX : double[], pointsY : double[])
+regenerate()
FreeSurface
+FreeSurface(hole : boolean, polygon : Polygon)
+FreeSurface(polygon : Path2D)
Builder
-surface : RectangularSurface
-firstPoint : Point2D
+Builder()
SurfaceBuilder
+placePoint(point2D : Point2D)
+movePoint(point2D : Point2D)
+makeHole(isHole : boolean)
+getSurface() : Surface
patterns
Pattern
#offsetX : double
#offsetY : double
#adjustXRules : double[]
#adjustYRules : double[]
#groutXRules : double[]
#groutYRules : double[]
#tiles : Vector<Tile>
#name : String
#defaultDimensions : Point2D = new Point2D.Double(100, 70)
#Pattern(name : String)
+setTileType(type : Tile)
+getCritBounds() : Rectangle2D
Classic
+Classic(tile : Tile)
Offset
+Offset(tile : Tile)
tools
Intersection
+onSegment(p : CustomPoint, q : CustomPoint, r : CustomPoint) : boolean
+orientation(p : CustomPoint, q : CustomPoint, r : CustomPoint) : double
+DoesIntersect(vec1 : Vecteur, vec2 : Vecteur)
+intersectionPoint(vec1 : Vecteur, vec2 : Vecteur)
Vertices
+getPath2DVertices(path2D : Path2D) : double[][]
PolygonTransformer
+awtPathToJavafx(path : Path2D)
+javafxPathToAwt(path : Path)
+javafxPathsToAwt(path : Path)
+subtract(polygon : Path2D, cuttingPattern : Path2D, cuttingInline : double) : Path2D[]
+merge(polygon1 : Path2D, polygon2 : Path2D)
-transformOrigin(origin : Vector2D, surface : Surface) : Vector2D
-getPatternBounds()
-operation()

I. Afficheur

MainWindow : hérite de **JFrame**. Le rôle de cette classe est de gérer tout ce qui est relatif à la gestion de la fenêtre : que faire lorsqu'on clique sur la croix, ou la gestion de son contenu. Elle possède **Controller** et également tous les éléments de la fenêtre tel que **Canvas**, **EditionPanel**, **Toolbar** & **BottomToolbar**.
Cette classe, via des agrégations est possédés par **Canvas**, **EditionPanel**, **Toolbar** & **BottomToolbar**, afin que tout élément graphique puisse interagir entres elles et avec **Controller**. (Tout cela via des getters).

1. Panels

Canvas : Hérite de **BorderedEventPanel** (classe non présente dans le diagramme, surcharge d'un **JPanel** pour gérer des évenements). Cette classe contient des « Listeners » d'événements qui interagiront en fonction avec **Controller** via un getter de **MainWindow**.

EditionPanel : Hérite de **BorderedPanel** (classe non présente dans le diagramme, surcharge d'un **JPanel**). Cette classe contient et gère toutes les palettes d'outils contenues dans l'éditeur comme l'édition de formes, les paramètres du canvas, la gestion du coulis entre autres. Elle contient des **SubPanels**.

Toolbar : Hérite de **BorderedPanel** (classe non présente dans le diagramme, surcharge d'un **JPanel**). Cette classe gère et contient tous les boutons principaux. Créer un nouveau Canvas, Sauvegarder, Editer le contenu du Canvas, …

BottomToolbar : Hérite de **BorderedPanel** (classe non présente dans le diagramme, surcharge d'un **JPanel**). Cette classe gère et affiche des informations comme le système métrique choisi, le zoom …

2. SubEdition

SubPanel : Hérite de **PanelEvents** (classe non présente dans le diagramme, surcharge d'un **JPanel** pour gérer des évenements). Il s'agit d'une classe abstraite, utilisé par **GroutPanel**, **PatternPanel**, **SurfacePanel**, **TilePanel**, **TileSettingsPanel**. Elle contient, grâce a une référence, **MainWindow**

GroutPanel : Hérite de **SubPanel**. Cette classe gère les paramètres d'un coulis (couleur, taille) pour une surface sélectionnée.

PatternPanel : Hérite de **SubPanel**. Cette classe permet de choisir, pour une surface sélectionnée, quel type de motif doit être appliqué.

SurfacePanel : Hérite de **SubPanel**. Cette classe gère la création de **Surface**s ainsi que son édition (hauteur, largeur).

TilePanel : Hérite de **SubPanel**. Cette permet de choisir quel type de tuile appliqué pour une surface sélectionnée.

TileSettingsPanel : Hérite de **SubPanel**. Cette classe permet de configurer les différents types de tuile.

II. Domaine

Controller : Cette classe représente l'unique point d'entrée de la couche Domaine. Elle contient toutes les méthodes qui peuvent être appelé par l'Afficheur ou Swing. (Controller de Larman)
Elle possède trois attributs, **Meta**, **SurfaceEditor** et **Painter**.

Painter : Cette classe, contenu dans **Controller**, permet d'afficher les éléments qui ont été créé par le programme et l'utilisateur (surfaces, motifs, etc.). Elle contient **Meta** en tant qu'attribut (référence).

SurfaceEditor : Cette classe contient toute la méthode relative à la création d'une surface, ainsi que les méthodes qui vont être appelé par **Canvas** depuis **Controller**.

1. Entities

Meta : Cette classe va contenir toutes les informations relatives à la session de l'utilisateur (liste de surfaces, état de ce qu'il est en train de faire, surface survolée avec la souris, etc.). Elle contient également les méthodes qui vont être appelés par les menus d'éditions, par le biais de **Controller** (applyPattern, setHoveredSurface, etc.)
Par défaut, **Meta** instancie 3 types de Tuile

Grout : Cette classe, possédée par **Surface**, représente l'espacement, le coulis qui va être présent entre deux tuiles. Elle possède une couleur ainsi qu'une épaisseur.

PatternGroup : Cette classe, possédée par **Surface**, représente un groupe de **Pattern** dans une surface. Une de ses méthodes permet de construire le pattern sur une surface donnée.

1.1. Pattern

Pattern : Cette classe abstraite représente un motif.

Classic : Cette classe hérite de **Pattern.** Elle instancie simplement une tuile.

Offset : Cette classe hérite de **Pattern.** Elle instancie deux tuiles, de manière à avoir une tuile décalée par rapport à l'autre, sur un axe horizontal.

1.2. Surfaces

PrimarySurface : Cette classe abstraite représente une surface primaire, dont **Bounds**, **FreeSurface, Rectangular Surface, Surface** et **Tile** vont hérités. Elle définit toutes les méthodes communes à une surface de base.

Bounds : Cette classe représente les limites d'une surface régulière et irrégulière, si cette surface était rectangulaire. Cette classe est contenu dans **Surface**.

FreeSurface : Cette classe hérite de **Surface** et implémente toutes les méthodes spécifiques à une surface irrégulière.

RectangularSurface : Cette classe hérité de **Surface.** Elle implémente toutes les méthodes spécifiques à une surface régulière. Elle contient également **Builder**, une classe statique qui permet de créer une surface de rectangulaire.

Surface : Cette classe hérite de **PrimarySurface**. Elle implémente toutes les méthodes et caractéristique d'une surface « Avancée », comme le fait d'être ou non, un trou. Elle possède également une liste de **Surface**, pouvant être une autre surface (comme un trou).

SurfaceBuilder : Cette classe abstraite permet de définir ce que les différents builders (pour le moment, il n'y a que **Builder**) doivent implémenter.

Tile : Cette classe, qui hérite directement de **PrimarySurface**, représente une tuile qui va être appliqué sur une surface via **PatternGrout**. Il s'agit plus exactement d'un « type » de tuile.

1.3. Tools

Intersection : Cette classe permet grâce à des méthodes statiques, de connaitre le point d'intersection entre deux segments.

PolygonTransorfmer : Cette classe permet grâce à des méthodes statiques, d'exploiter des fonctionnalités mathématiques de JavaFx sur des Polygon d'awt.

Vertices : Cette classe contient une méthode statique permettant de récupérer les sommets d'un Path2D.

IV. Contribution des membres

Pour ce livrable 3, le travail à été plus compliqué que prévu. Maxence Fourrier à décidé d'abandonner le cours (GLO-2004), il n'a donc rien produit pour cette itération.

Thomas Lombard et Martin Cotoni ont réalisés ensemble la tâche concernant **l'association d'un motif simple à une surface** et **la combinaison de deux surfaces**.

Martin Cotoni à réaliser les tâches suivantes : **Editer les propriétés d'un matériau**, **Editer les paramètres d'une surface**, **Redimensionner des surfaces**, **Associer un matériau à une surface**, **Recalcule du motif lors du redimensionnement d'une surface**. Il a également développé **les menus de l'interface** ainsi que les **interactions** qui n'étaient pas présent / présentes dans le livrable 2 et à aussi refondu tout le projet sur la base de la correction fournie au livrable 2. Il s'est par ailleurs occupé de réaliser et de corriger les diagrammes pour ce livrable.

Thomas Lombard à réaliser les tâches suivantes : **Créer une surface rectangulaire**, **Supprimer une surface**, **Zoomer / Dezoomer**, **Déplacement de surfaces**, **Sélection / Désélection**. Il à également réaliser toutes les classes « Outils » ou encore les classes de gestions d'événements dont Martin Cotoni s'est servi pour réaliser les interactions avec le controller.

Antoine Pelletant à réaliser les tâches suivantes : **Grille magnétique**, **Afficher les informations d'une surface survolée**. L'implication est arrivée tardivement, ce qui explique pourquoi Thomas Lombard et Martin Cotoni ont réalisé l'ensemble des tâches.