24/09/2019

VirtuTuile

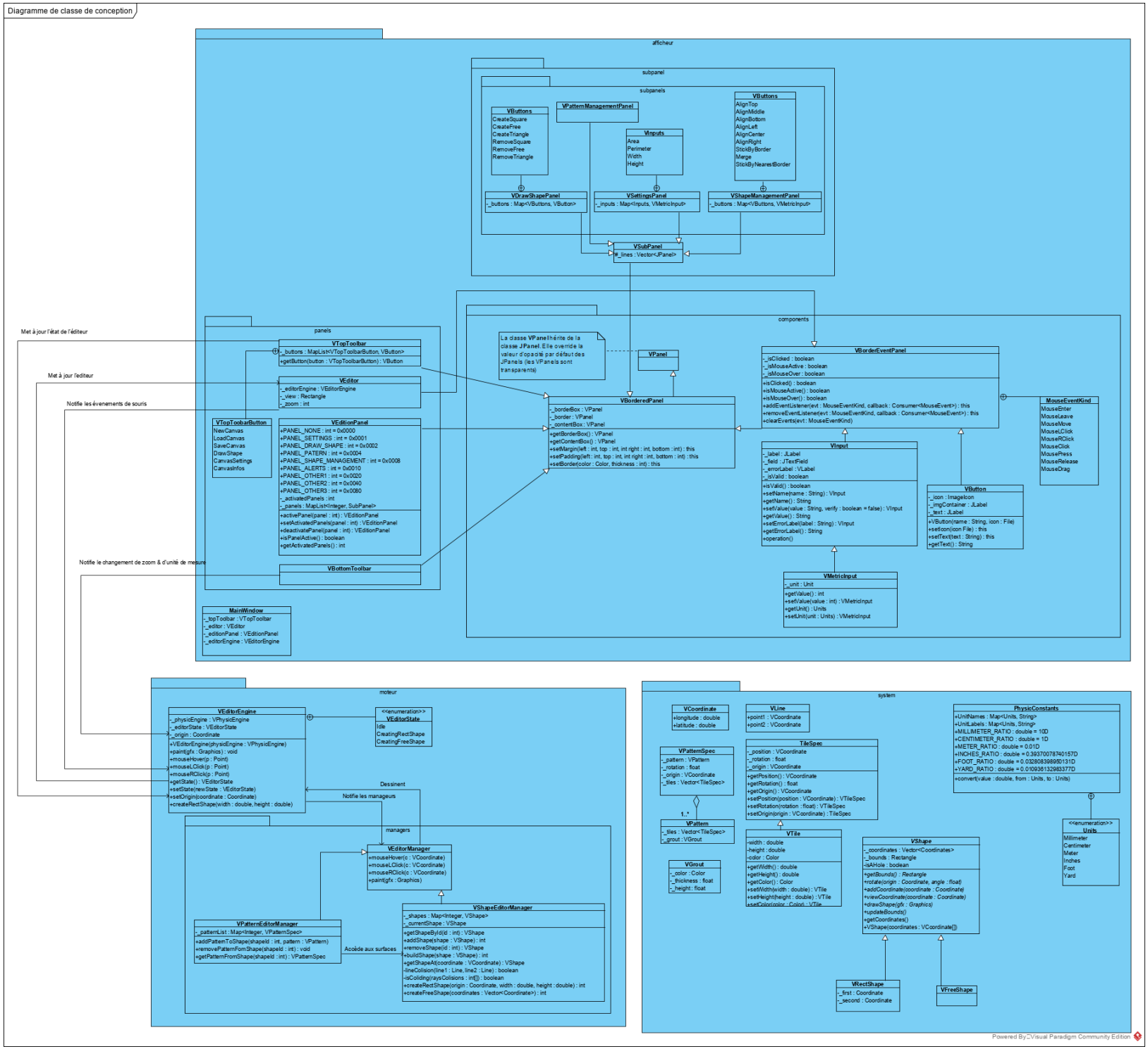
Livrable 2

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

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

Table des matières

I. Diagramme de classe de conception



1. Package Afficheur

Ce package contient tout le code spécifique à l'affichage.

- **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.

1.1. Package panels

- **VTopToolbar** : Hérite de **VBorderedPanel**. Cette classe gère et contient tous les boutons principaux. Créer un nouveau Canvas, Sauvegarder, Editer le contenu du Canvas, …
- **VTopToolbarButtons** : Cet enum représente la liste des boutons contenus par **VTopToolbar**. Les valeurs associées sont les liens entre ces boutons et le reste du code. Elles servent d'accesseurs pour ces boutons.
- **VEditor** : Hérite de **VBorderedEventPanel**. Cette classe gère et transfère tous les événements reçu par l'éditeur au **VEditorEngine**.
- **VEditonPanel** : Hérite de **VBorderedPanel**. 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
- **VBottomToolbar** : Hérite de **VBorderedPanel**. Cette classe gère et affiche des informations comme le système métrique choisi, le zoom …

1.2. Package subpanel

- VSubPanel : Hérite de VBorderedPanel. Cette classe abstraite gère tout ce qui est relatif à l'implémentation d'un panneau de configuration / édition pour la classe VEditionPanel.

1.2.1. Package subpanels

- **VDrawShapePanel** : Hérite de **VSubPanel**. Cette classe est contient et gère tous les outils relatifs à l'édition et au dessin de formes. Elle notifie **VEditorEngine** à chaque type de changements.
- **VButtons** : Cet enum représente la liste des boutons contenus par **VDrawShapePanel**. Les valeurs associées sont des liens entre ces boutons et le reste du code. Elles savent d'accesseurs pour ces boutons.
- **VPatternManagementPanel** : Hérite de **VSubPanel**. Cette classe contient et gère tous les outils relatifs aux choix et configurations des motifs. Elle notifie **VEditorEngine** à chaque type de changements.
- **VShapeManagementPanel** : Hérite de **VSubPanel**. Cette classe contient et gère toutes les entrées utilisateurs relatifs à l'édition et la configuration d'une face.
- **VInputs** : Cet enum représente la liste des inputs contenus par **VShapeManagementPanel**. Les valeurs associées sont des liens entre ces inputs et le reste du code. Elles savent d'accesseurs pour ces inputs.
- **VSettingsPanel** : Hérite de **VSubPanel**. Cette classe contient et gère toutes les entrées utilisateurs relatifs à la configuration du canvas / projet.
- **VButtons** : Cet enum représente la liste des boutons contenus par **VSettingsPanel**. Les valeurs associées sont des liens entre ces boutons et le reste du code. Elles savent d'accesseurs pour ces boutons.

1.3. Package components

Ce package contient toutes les classes qualifiables de composants. C'est à dire que ce sont des briques élémentaires du programmes et réutilisables.

- **VPanel** : Hérite de **JPanel**. Cette classe a pour rôle principal de surcharger la valeur d'opacité par défaut des **JPanel**s. Ainsi contrairement aux **JPanel**s, les **VPanel**s sont transparents par défaut.
- **VBorderedPanel** : Hérite de **VPanel**. Cette classe a pour rôle principal de simplifier la gestion des marges, bordures et marges-internes des **JPanels**. Par défaut elles n'ont pas de bordure, mais ont des marges internes et externes de 5px.
- **VBorderedEventPanel** : Hérite de **VBorderedPanel**. Cette classe a pour rôle d'intercepter et de dispatcher tous les évènements de souris. Ainsi elle permet d'ajouter des écouteurs sur des évènements précis afin de simplifier la gestion de ceux-ci. La liste des évènements gérés par **VBorderedEventPanel** est disponible dans l'enum **MouseEventKind**.
- **VInput** : Hérite de **VBorderEventPanel**. Cette classe a pour rôle d'encapsuler des **JTextField** dans des composants de plus haut niveau permettant d'automatiser certains processus comme la validation du contenu de l'input, la gestion d'erreur ou encore l'affichage de message d'erreurs.
- **VMetricInput** : Hérite de **VInput**. Cette classe ajoute un type à **VInput**. Ce type doit être métrique. (cm, m, km, in, …)
- **VButton** : Hérite de **VBorderEventPanel**. Cette classe a pour rôle d'encapsuler et de simplifier la gestion des boutons avec icône.

2. Package moteur

Ce package contient tout le code dit spécifique. C'est à dire entre autres, le code concernant la gestion des surfaces, la gestion des motifs, du coulis …

- **VEditorEngine**. C'est notre controlleur de Larman. Il reçoit tous les événements et les dispatche aux manageurs dédiés. Il possède également un « état ». Cet état détermine vers quel manageur doit traiter un évènement. Chacune des barres d'outils et panneaux d'édition notifie le contrôleur à chaque fois que l'état doit changer. Ainsi, quand la classe **VTopToolbar** est notifiée d'un changement d'état - création d'une surface par exemple - elle notifie **VEditorEngine** qui utilisera donc **VShapeEditorManager** pour gérer ces traitements.
- **VEditorState** : cet enum réprésente tous les états possibles de l'éditeur. C'est ce qui fait le lien entre les barres d'état et d'édition et **VEditorEngine**

2.1. Package managers

- **VEditorManager** : Cette classe abstraite est ce qui caractérise un manageur. Elle définit les principales méthodes qu'appellera **VEditorEngine**.
- **VShapeEditorManager** : hérite de **VEditorManager**. Elle est responsable de la gestion des formes dans le programme. C'est ce manageur qui contient toute la liste des formes en cours dans l'éditeur.

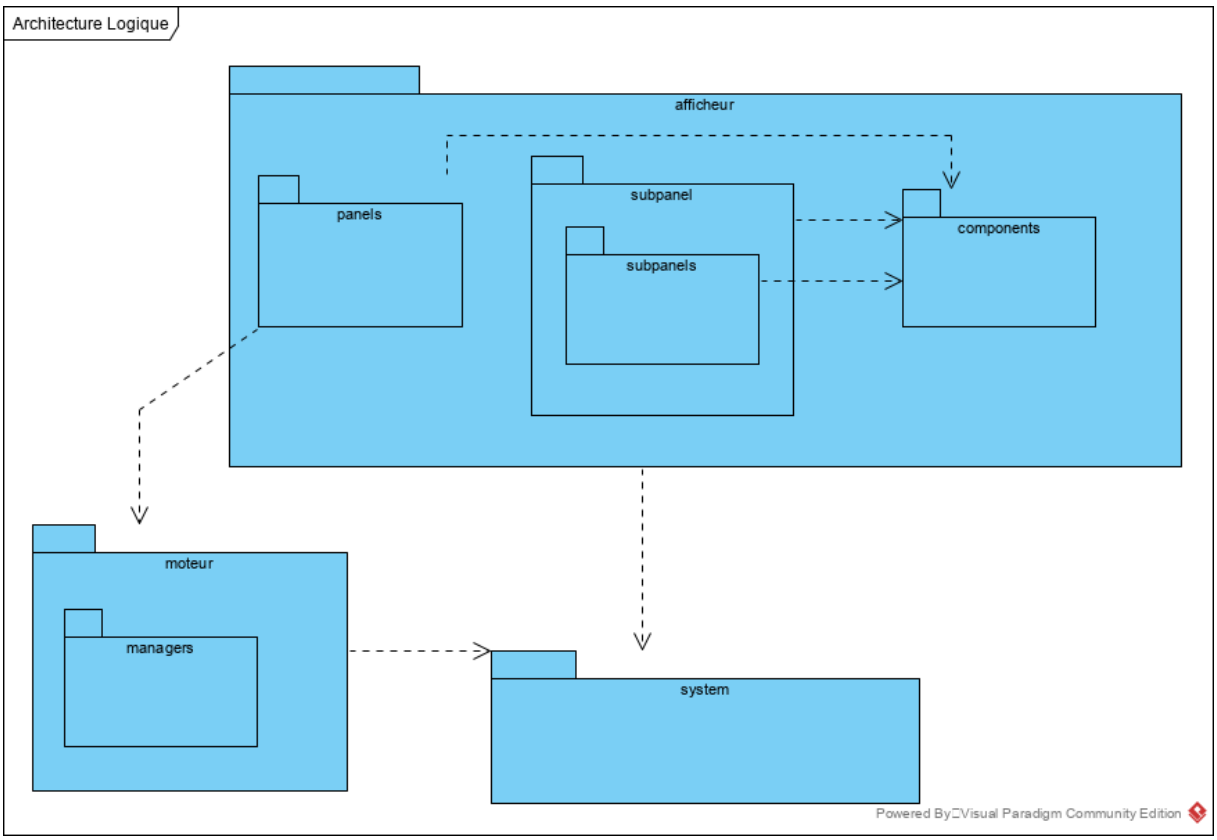
- **VPatternEditorManager** : hérite de **VEditorManager**. Elle est responsable de la gestion et du traitement des différents motifs. C'est ce manageur qui contient toutes les spécifications des motifs appliqués aux surfaces.

3. Package système

Ce package contient tout le code dit générique. C'est du code susceptible d'être utilisé par les deux autres packages. Il renferme majoritairement des entités ou des constantes.

- **VCoordinate** : Cette classe se comporte comme une structure en C++. Ses attributs sont publics car elle n'en contient que deux et qu'il n'y a pas de traitement spécifique nécessitant une protection de ceux-ci. Elle contient deux doubles : longitude et latitude.
- **VLine** : Cette classe se comporte comme une structure en C++. Ses attributs sont publics pour la même raison que **VCoordinate**. Elle contient deux **VCoordinate** ce qui caractérise une ligne.
- **VTile** : Cette classe contient toutes les informations caractérisant une tuile seule.
- **VTileSpec** : Cette classe contient toutes les informations permettant de placer une tuile dans un motif. Elle étend **VTile** en y ajoutant une gestion de la rotation, de l'origine - point autour duquel faire pivoter la tuile - et la position de la tuile dans le motif.
- **VPattern** : Cette classe contient toutes les informations caractérisant un motif tels que les tuiles à placer dans ce motif - **VTileSpec** -, et le coulis.
- **VPatternSpec** : Cette classe contient toutes les informations caractérisant un motif appliqué à une surface avec entre autres l'origine de la répétition, les tuiles utilisées et la rotation éventuelle du motif.
- **VGrout** : Cette classe se comporte comme une structure en C++ pour la même raison que **VCoordinate**. Elle contient 3 attributs publics : color, thickness et height qui sont respectivement la couleur du coulis, l'épaisseur du coulis entre les tuiles et l'épaisseur du coulis entre la tuile et le sol.
- **VShape** : Cette classe abstraite détermine et définit ce qui caractérise une forme. Elle contient tous les sommets de cette forme ainsi que la rotation de celle-ci.
- **VRectShape** : Cette classe étends **VShape**. Elle y ajoute la possibilité de créer une forme via sa hauteur et sa largeur.
- **VFreeShape** : Cette classe étends **VShape**. Elle y ajoute la possibilité de créer une forme via ses sommets.
- **VPhysicConstants** : Cette classe est statique et publique. Elle contient toutes les constantes relatives à la physique du programme tels que les conversions d'unités.

II. Architecture Logique



Cette architecture logique se compose deux couches différentes.

La première est celle qui va interagir avec l'utilisateur, la couche nommée **afficheur**. Nous allons donc retrouver les interfaces utilisateurs, ainsi que tous les objets graphiques et les classes de Swing (encapsulée).
Ces classes sont encapsulées dans le package **components**. Ce package contient toutes les briques graphiques élémentaires tels que des panneaux déjà configurés, des boutons …
On y trouve également le package **panels**. Il contient tous les éléments de plus haut niveau tels que les barres d'outils ou les panneaux d'édition. Afficheur contient également deux autres packages, **subpanel** et **subpanels**.
Ces deux packages sont imbriqués (**subpanel** contient **subpanels**). Ils contiennent respectivement ce qui caractérise un élément des panneaux d'édition et les éléments des panneaux d'édition.

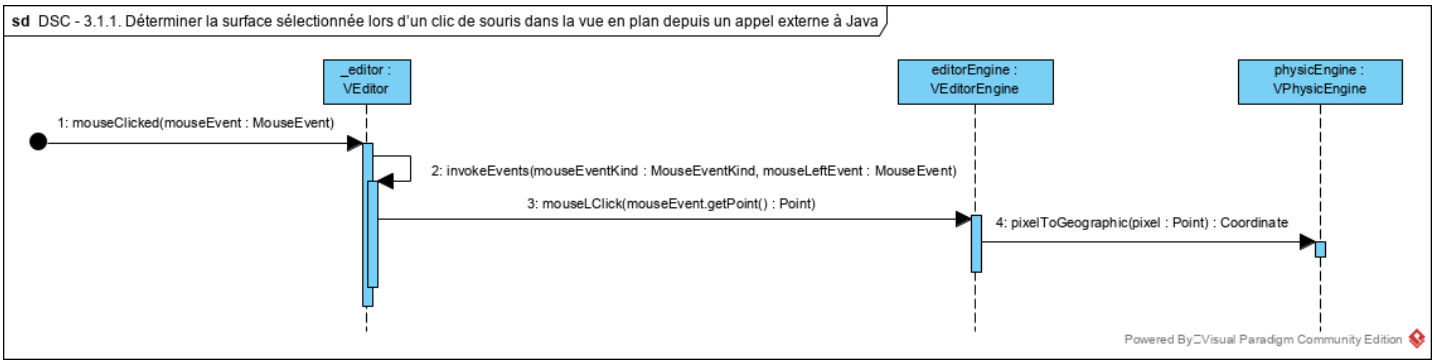
La seconde, est composée de deux packages différents. Le premier nommé **moteur**, contient toute la partie intelligente de l'application, notamment la classe **VEditorEngine** qui représente notre contrôleur unique, le point d'entrée et de sortie entre **afficheur** et **moteur**. Etant donné que notre contrôleur n'effectue pas de traitement logique à proprement parler, il existe un package **managers**, contenu dans **moteur** qui est responsable desdits manageurs. Ce sont eux qui effectueront les traitements logiques du programme.
Le deuxième package de la couche non-graphique est **System**, il contient toutes les classes nécessaires au fonctionnement de **moteur** et **afficheur**. Il peut être vu comme en entité commune au

programme, que le contrôleur régit. **System** contient des modèles et de constantes. C'est la raison pour laquelle il est à part et qu'il ne dispose pas de sous-packages.

III. Diagrammes de séquence de conception

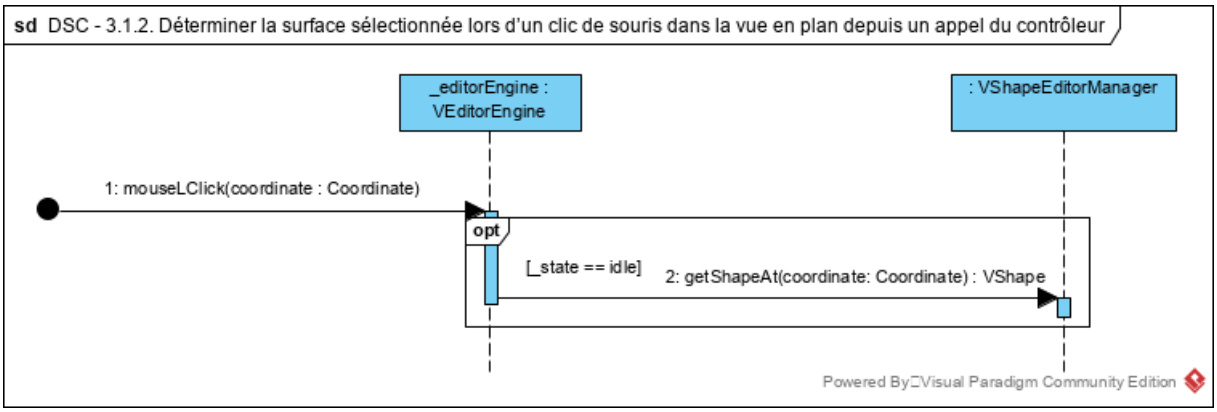
1. Déterminer la surface sélectionnée lors d'un clic de souris dans la vue en plan.

1.1. Depuis un appel externe à Java



Ce diagramme de séquence de conception démarre tout d'abord par un événement de la souris, définit ici par l'appel **mouseClicked**. A la suite de cet appel, un appel à **invokeEvents** va déterminer quel événement doit être déclenché. Grâce à un **eventListener** définit dans *VEditor* au lancement du projet (prenant en paramètre une lambda), la méthode **mouseLClick** va être appelé depuis *VEditorEngine* (contrôleur). Enfin, un appel à **pixelToGeographic** va nous renvoyer les pixels du clic en unité de mesure.

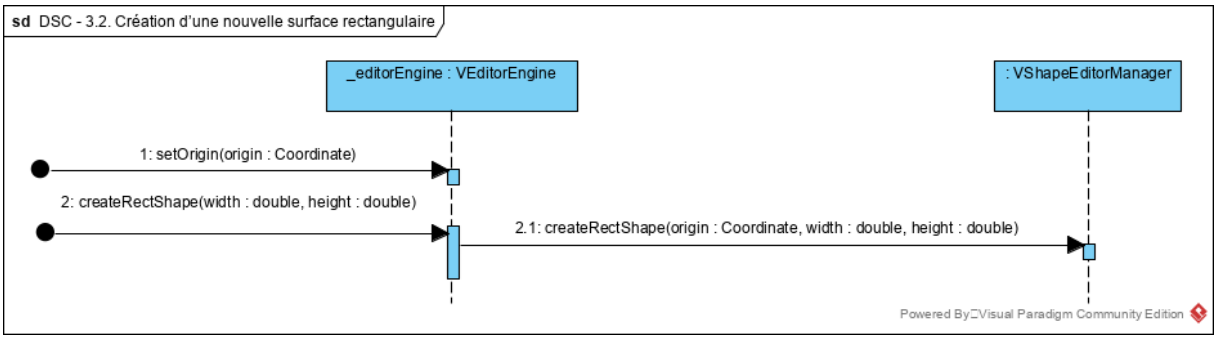
1.2. Depuis un appel du contrôleur



Le premier appel dans ce diagramme va être **mouseLClick**, ayant en paramètre des coordonnées en unité de mesure. *VEditorEngine* possède un attribut nommé **state**, qui définira la fonction à effectuer. S'il possède la valeur « idle », alors un appel vers *VShapeEditorManager* va être effectué,

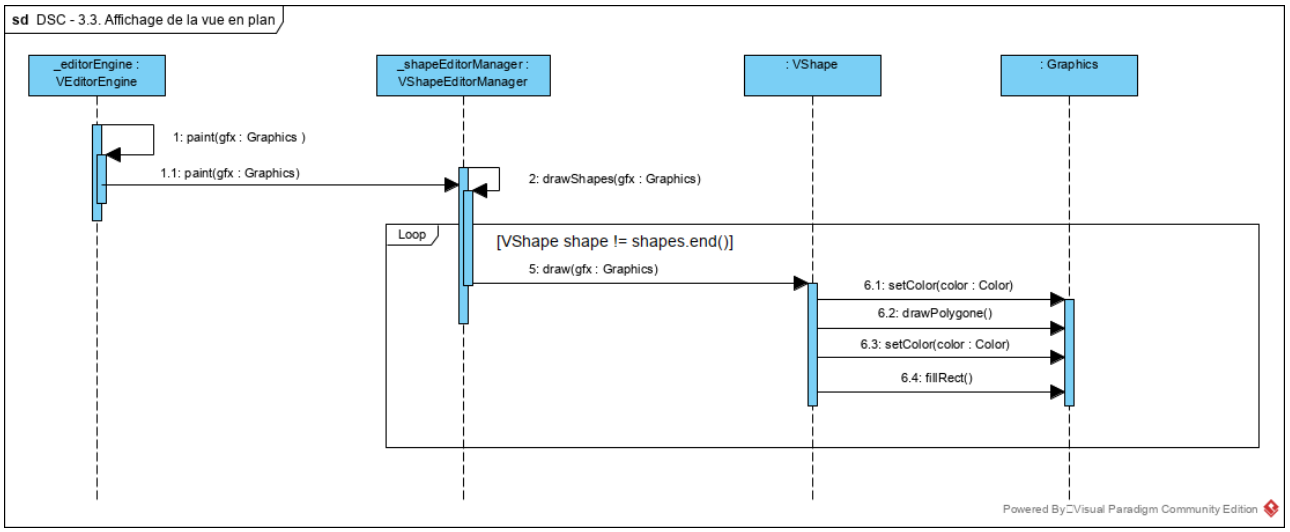
avec la méthode **getShapeAt** (que vous pouvez retrouver dans la partie *IV. Pseudo Code*). De cette manière, soit une forme (**VShape**) nous est renvoyée en cas de succès, soit *null*.

2. Création d'une nouvelle surface rectangulaire



Un premier appel est fait avec la méthode **setOrigin**, qui permet de définir les coordonnées de la forme à créer. Ensuite, **createRectShape** est appelé afin de définir les dimensions de la forme à créer. Cette forme est ensuite créée par le *VShapeEditorManager* avec sa méthode **createRectShape** recevant en paramètre les coordonnées ainsi que les dimensions. **createRectShape** ajoute ensuite la forme créé à la liste de **VShape** que *VShapeEditorManager* contient.

3. Affichage de la vue en plan



Lorsqu'il sera nécessaire d'actualisé le visuel de l'application, un événement sera déclenché. Cet événement localisé dans la classe *VEditorEngine,* va appeler la fonction **paint** que nous avons surchargé. Ensuite, la fonction **paint** de *VShapeEditorManager* va s'occuper d'appeler la fonction **drawShapes** contenant des appels à **draw** présent dans chaque **VShape**, à l'aide d'un *forEach*. Finalement, chaque **VShape** va pouvoir update l'élément *Graphics*.

IV. Pseudo-code d'un algorithme qui permet de déterminer si un point se trouve à l'intérieur d'une surface irrégulière

```
Type Line : tableau de Point de deux cases

Début Structure Point :
   x en tant que virgule flottante
   y en tant que virgule flottante
Fin Structure
```

```
// Fonction qui pour une position donnée return soit une shape si
// la position donnée est à l'intérieur de cette shape ou null si
// la position donnée n'est pas à l'intérieur d'une shape.

Début Procédure VShapeEditorManager::getShapeAt(position)
        Entrées
                position en tant que VCoordinate

        Sortie
                La forme trouvée en tant que VShape, null autrement

        Locales
                pos1 en tant qu'entier
                pos2 en tant qu'entier
                first en tant que Line
                second en tant que Line
                third en tant que Line
                fourth en tant que Line
                results en tant que tableau d'entier de 4 cases

        Procédure
                Pour (shape parmi toutes les shapes)
                        Pour (i inférieur à nombre de points dans shape)
                                pos1 = shape.points[ i ]
                                pos2 = shape.points[ i + 1 ]
                                first = {{pos1.x, pos1.y}, {pos1.x, +∞}}
                                second = {{pos1.x, pos1.y}, {pos1.x, -∞}}
                                third = {{pos1.x, pos1.y}, {+∞, pos1.y}}
                                fourth = {{pos1.x, pos1.y}, {-∞, pos1.y}}
                                results = {0, 0, 0, 0}

                                Si (lineColision(first, {{pos1.x, pos1.y}, {pos2.x, pos2.y}}) est
vrai)
```

```
                                        results[0] incrémente de 1;
                                        Si (firstShape vaut null)
                                                firstShape = shape
                                Si (lineColision(second, {{pos1.x, pos1.y}, {pos2.x, pos2.y}})
est vrai)
                                        results[1] incrémente de 1;
                                        Si (firstShape vaut null)
                                                firstShape = shape
                                Si (lineColision(third, {{pos1.x, pos1.y}, {pos2.x, pos2.y}}) est
vrai)
                                        results[2] incrémente de 1;
                                        Si (firstShape vaut null)
                                                firstShape = shape
                                Si (lineColision(fourth, {{pos1.x, pos1.y}, {pos2.x, pos2.y}})
est vrai)
                                        results[3] incrémente de 1;
                                        Si (firstShape vaut null)
                                                firstShape = shape
                Si (isColiding(resultat) est vrai)
                        renvoi firstShape
                Sinon
                        renvoi null
Fin Procédure
```

```
// Fonction qui pour deux ligne donnée, return true si les deux
// ligne entre en collision ou false si les deux lignes ne rentre
// pas en collision. 'ab' et 'cd' sont les deux segments à tester

Début Procédure VShapeEditorManager::lineColision(ab, cd)
        Entrées
                ab en tant que Line
                cd en tant que Line

        Sortie
                booléen (vrai si collision, faux autrement)

        Locales
                dénominateur en tant que virgule flottante
                numérateur1 en tant que virgule flottante
                numérateur2 en tant que virgule flottante
                r en tant que virgule flottante
                s en tant que virgule flottante

        Procédure
                dénominateur = ((ab[1].x - ab[0].x) * (cd[1].y - cd[0].y)) - ((ab[1].y - ab[0].y) *
(cd[1].x - cd[0].x))
                numérateur1 = ((ab[1].y - ab[0].y) * (cd[1].x - cd[0].x)) - ((ab[1].x - ab[0].x) *
(cd[1].y - cd[0].y))
                numérateur2 = ((ab[0].y - cd[0].y) * (ab[1].x - ab[0].x)) - ((ab[0].x - cd[0].x) *
(ab[1].y - ab[0].y))
                Si (dénominateur vaut 0)
                        Si (numérateur1 vaut 0 et numérateur2 vaut 0)
                                renvoi vrai
                        Sinon
                                renvoi faux

                r = numérateur1 / dénominateur
                s = numérateur2 / dénominateur

                Si (0 <= r <= 1 et 0 <= s <= 1)
                        renvoi vrai
                Sinon
                        renvoi faux
Fin Procédure
```

```
Début Procédure VShapeEditorManager::isColiding(collisions)
        Entrées
                collisions en tant que tableau d'entiers

        Sortie
                booléen (vrai si plus de segments ou autant se croissent un nombre pair de
fois)

        Locales
                isNotColliding en tant qu'entier
                isColliding en tant qu'entier
                i en tant qu'entier
                collision en tant qu'entier

        Procédure
                isNotColliding = 0
                isColliding = 0

                Pour (i <= 4)
                        collision = collisions[ i ]
                        Si (collision % 2 vaut 0)
                                isNotColliding incrémente de 1
                        Sinon
                                isColliding incrémente de 1
                        i incrémente de 1

                Si (isColliding >= isNotColliding)
                        renvoi vrai
                Sinon
                        renvoi faux
Fin Procédure
```

Source : https://gamedev.stackexchange.com/questions/26004/how-to-detect-2d-line-on-line-collision

V. Diagramme de Gantt

Il reste désormais **5 semaines** (S42 (semaine du rendu du livrable 2) et S48 (semaine précédente du rendu du livrable 3)). Ici, **une semaine** compte comme **une itération**.

Itération 1	Itération 2	Itération 3	Itération 4	Itération 5	Itération 6	Itération 7	Itération 8
			Sélectionner une surface	Paramètrer un coulis	Afficher les informations d'une surface	Nouveau Projet	Charger projet
			Créer une surface rectangulaire	Redimensionner une surface	Undo / Redo	Sauvegarder projet	Afficher / Cacher grille magnétique
			Créer une surface libre	Fusionner deux surfaces	Pivoter	Changer le niveau de zoom	
			Créer un trou dans une surface	Coller deux surfaces	Détecter les matériaux coupés		
			Supprimer une surface	Appliquer un motif sur une surface	Aligner deux surfaces		
			Déplacer une surface	Editer un type de matériaux	Inspecter les matériaux		

VI. Contribution des membres

Compte tenu des tâches à réaliser, l'équipe s'est une nouvelle fois réparti le travail. Thomas Lombard s'est majoritairement concentré sur la réalisation du **diagramme de classe de conception**, ainsi que sur l'**implémentation du squelette** de VirtuTuile. Maxence Fourrier, Antoine Pelletant et Martin Cotoni ont quant à eux réalisés **les diagrammes de séquence de conception**, en collaboration avec Thomas afin d'effectuer des changements pour affiner la justesse de l'architecture. Deux réunions formelles ont eu lieu, réunions dans lesquelles nous avons pu affiner le diagramme de conception et retravailler **le domaine du modèle**, **l'énoncé de vision** et les **modèles de cas d'utilisation**. Martin Cotoni s'est occupé du réarrangement du diagramme de Gantt, ainsi que de l'unification du travail.

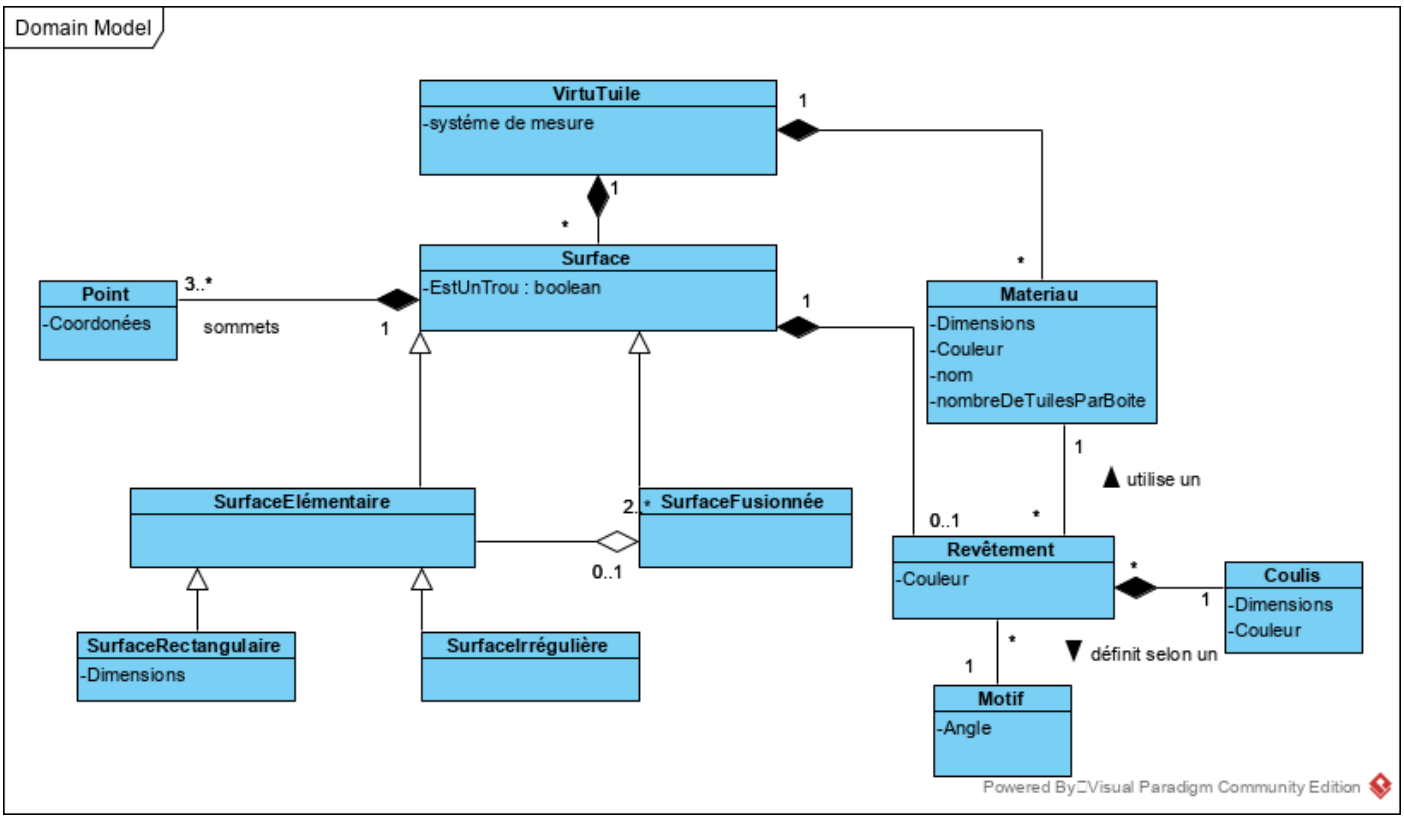
Annexe : Livrable 1

I. Enoncé de Vision

Le but de **Virtutuile** est de faciliter les travaux d'architecture d'une personne dans le monde du carrelage, qu'il soit un professionnel ou un particulier. Le but est de proposer un logiciel qui va réunir de nombreux points importants de ce domaine. Après que l'utilisateur ait spécifié le nombre de tuiles présentes dans les cartons qu'il souhaite acheter, il va pouvoir définir une ou plusieurs surfaces à couvrir. Sur ces surfaces, l'utilisateur pourra appliquer un motif à sa convenance, en y ajoutant ses spécificités, comme par exemple les dimensions du coulis ou encore les couleurs des différents éléments. Il pourra également coller ou fusionner des surfaces, afin de représenter au mieux la réalité. Enfin, VirtuTuile se chargera d'informer l'utilisateur de toutes les informations dont il aura besoin, comme par exemple le nombre de boites de tuiles à acheter ou encore le nombre de tuiles « non-pleines ».

II. Modèle du domaine

A. Diagramme du modèle du domaine



B. Texte Explicatif

VirtuTuile représente le programme, dans lequel il est possible de sélectionner une unité de mesure. Cette classe est composé de *Surface* et de *Matériau.*

La **Surface** représente la zone qui va être défini par l'utilisateur afin d'être remplie d'un revêtement. Elle contient un booléen nommé « EstUnTrou » afin de savoir si cette surface est pleine. Elle est composée de la classe *Point* et de la classe *Revêtement*.

La classe **Point** définit les sommets d'une *Surface*. Un point contient des Coordonnées x et y.

Le **Revêtement** représente la couche qui va être appliquer sur une *Surface*. Il possède un angle, ainsi que d'une couleur. Il est composé d'un *Coulis* et est définit selon un *Motif* en utilisant un *Matériau*.

Le **Motif** représente une disposition précise de *Matériau* sur une *Surface* via un *Revêtement*.

La classe **Matériau** (aussi appelé « type de tuile ») représente l'élément qui va se répéter dans un *Revêtement*. Il possède une couleur, un nom, des dimensions ainsi qu'un nombre de tuile présent dans une boite (définit par l'utilisateur).

Le **Coulis** représente la surface de séparation entre chaque *Matériau* dans un revêtement. Le coulis possède une couleur ainsi que des Dimensions.

Une **Surface Elémentaire** est un type de *Surface*. Elle est peut-être fusionnée avec une autre et donc via une agrégation, devenir une *SurfaceFusionée*. Elle peut également être rectangulaire (et être une *SurfaceRectangulaire*) ou être irrégulière (et être une *SurfaceIrrégulière*).

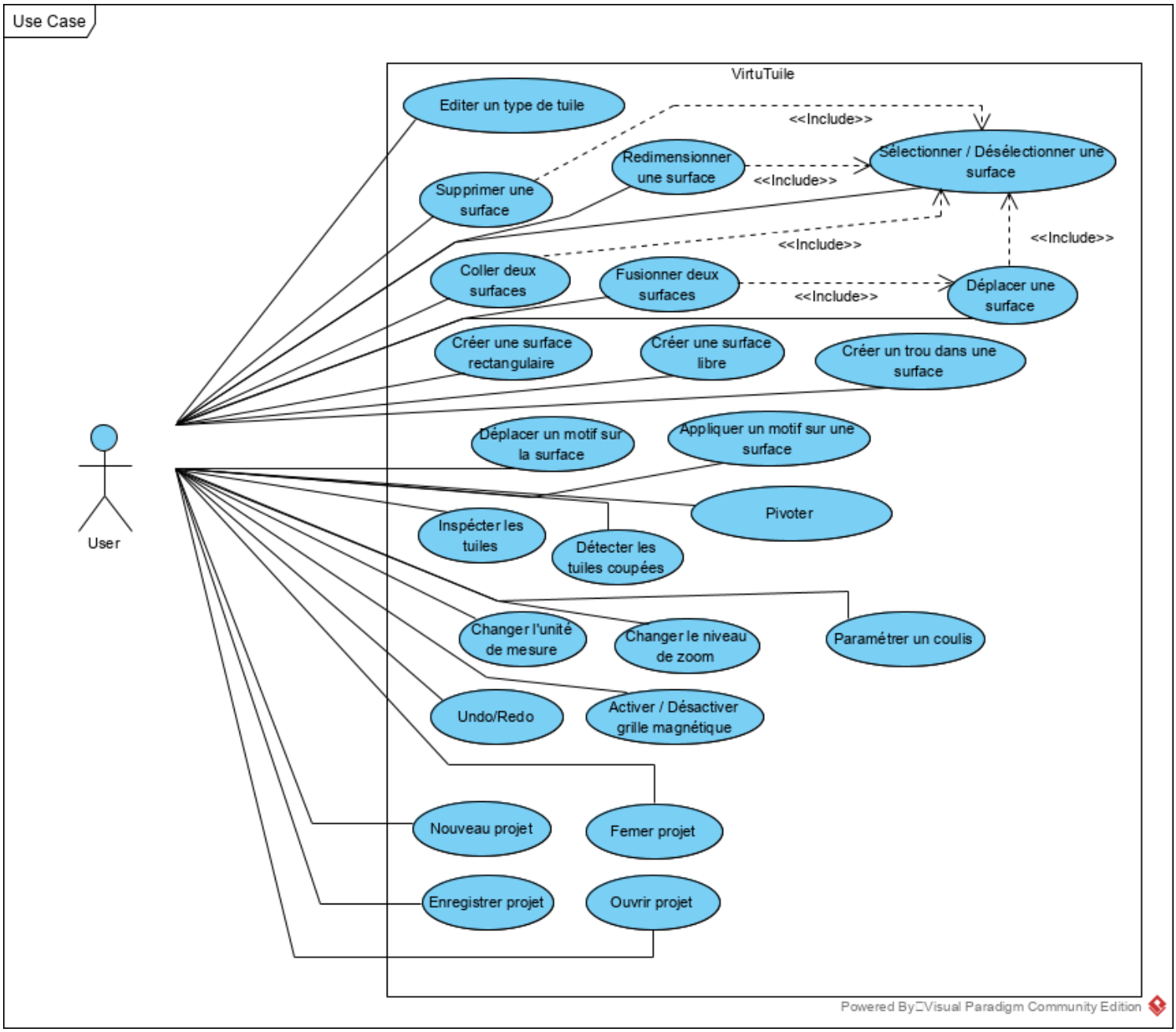
Une **Surface Fusionnée** représenté par une agrégation, correspond à la fusion de deux (ou plusieurs) *SurfaceElémentaire*.

Une **Surface Rectangulaire** représente une *SurfaceElémentaire* ayant des dimensions (longueur et largeur) et une forme rectangulaire. De la même manière qu'une *Surface*, elle possède des *Points*.

Une **Surface Irrégulière** représente une *SurfaceElémentaire* ayant des *Points* comme sommets pour repère. Il s'agit ici d'un polygone quelconque.

III. Modèle des cas d'utilisation

A. Diagramme des cas d'utilisation



B. Texte des cas d'utilisation

Cas d'utilisation :	**Sélectionner une surface**
Système :	VirtuTuile
Acteur(s) :	Utilisateur
Partie prenante et intérêts :	Utilisateur : Sélectionner une surface
Préconditions :	Une surface est créée.
Garantie en cas de succès :	La surface sélectionnée est en surbrillance et ses informations sont affichées.
Scénarios Principal :	1. L'utilisateur appuie sur une surface avec le clic-gauche de sa souris. 2. VirtuTuile affiche en surbrillance la surface sélectionnée. 3. VirtuTuile affiche les informations de la surface sélectionnée.
Scénarios Secondaires :	

Cas d'utilisation :	**Créer une surface rectangulaire**
Système :	VirtuTuile
Acteur(s) :	Utilisateur
Partie prenante et intérêts :	Utilisateur : Créer une surface rectangulaire.
Préconditions :	
Garantie en cas de succès :	La surface rectangulaire est créée.
Scénarios Principal :	1. L'utilisateur appuie sur le bouton créer une forme rectangulaire. 2. L'utilisateur définit un premier point. 3. VirtuTuile définit ce point comme origine. 4. VirtuTuile affiche une prévisualisation de la forme en suivant la souris par rapport au point d'origine.

	5. L'utilisateur sélectionne un deuxième point pour définir l'angle opposé.	6. VirtuTuile créé la surface avec pour référence ces deux points opposés. 7. VirtuTuile affiche la surface.
Scénarios Secondaires :		

Cas d'utilisation :	**Créer une surface libre**
Système :	VirtuTuile
Acteur(s) :	Utilisateur
Partie prenante et intérêts :	Utilisateur : Créer une surface libre
Préconditions :	
Garantie en cas de succès :	La surface libre est créée.
Scénarios Principal :	1. L'utilisateur appuie sur le bouton créer une forme libre. 2. L'utilisateur définit une suite de point sur le plan de travail. 3. VirtuTuile relie cette suite de point par des lignes. 4. L'utilisateur double-clic droit sur sa souris afin de valider les points. 5. VirtuTuile ferme la forme en reliant le premier et dernier point, crée une surface. 6. VirtuTuile affiche la surface à l'écran.
Scénarios Secondaires :	<u>Ligne 4 :</u> Les points définit ne peuvent pas se relier étant donné une intersection entre deux liens, VirtuTuile invite l'utilisateur à réessayer.

Cas d'utilisation :	**Inspecter les matériaux**
Système :	VirtuTuile
Acteur(s) :	Utilisateur
Partie prenante et intérêts :	Utilisateur : Connaitre les caractéristiques du matériau.
Préconditions :	Une surface est créée avec un motif appliqué.
Garantie en cas de succès :	Les informations du matériau s'affichent à l'écran.
Scénarios Principal :	1. L'utilisateur survole le matériau. 2. VirtuTuile affiche les caractéristiques du matériau.

Scénarios Secondaires :	

Cas d'utilisation :	**Détecter les matériaux coupés**
Système :	VirtuTuile
Acteur(s) :	Utilisateur
Partie prenante et intérêts :	Utilisateur : Savoir quels matériaux sont coupés.
Préconditions :	Une surface est créée avec un motif appliqué.
Garantie en cas de succès :	Les matériaux coupés sont affichés en surbrillance à l'écran.
Scénarios Principal :	1. L'utilisateur clic sur le bouton de détection des matériaux coupés. 2. VirtuTuile recherche les matériaux découpées sur les surfaces. 3. VirtuTuile affiche en surbrillances les matériaux concernées.
Scénarios Secondaires :	

Cas d'utilisation :	**Déplacer une surface**
Système :	VirtuTuile
Acteur(s) :	Utilisateur
Partie prenante et intérêts :	Utilisateur : Déplacer une surface sur un plan de travail
Préconditions :	Une surface est créée.
Garantie en cas de succès :	La surface est créée est déplacé à une nouvelle position
Scénarios Principal :	1. (voir use case « Sélectionner une surface ») 2. L'utilisateur maintient le clic sur la surface et la déplace sur le plan de travail. 3. VirtuTuile définit la nouvelle position de la surface. 4. VirtuTuile affiche la surface et ses informations à l'écran.
Scénarios Secondaires :	<u>Ligne 2 :</u> La surface initialisée ne contient pas assez de points, VirtuTuile indique un message d'erreur et invite l'utilisateur à réessayer.

Cas d'utilisation :	**Appliquer un motif sur une surface**
Système :	VirtuTuile

Acteur(s) :	Utilisateur
Partie prenante et intérêts :	Utilisateur : Ajouter un motif sur une surface.
Préconditions :	Une surface est créée.
Garantie en cas de succès :	La surface est affichée avec le motif sélectionné par l'utilisateur.
Scénarios Principal :	1. (voir use case « Sélectionner une surface ») 2. L'utilisateur sélectionne un motif dans le panneau d'édition. 3. VirtuTuile applique le motif sélectionné sur la surface sélectionnée. 4. VirtuTuile affiche le motif sur la surface.
Scénarios Secondaires :	

Cas d'utilisation :	**Créer un trou dans une surface**
Système :	VirtuTuile
Acteur(s) :	Utilisateur
Partie prenante et intérêts :	Utilisateur : Définir une zone « vide », un trou, dans une surface.
Préconditions :	Une surface est créée
Garantie en cas de succès :	La surface est affichée avec une zone « vide » aux dimensions définies par l'utilisateur.
Scénarios Principal :	1. L'utilisateur clic sur le bouton « Créer un trou ». 2. L'utilisateur définit le périmètre du trou. 3. VirtuTuile applique ce trou sur la surface. 4. VirtuTuile affiche la surface modifiée.
Scénarios Secondaires :	<u>Ligne 2 :</u> Le trou définit est plus grand que la surface sur lequel il doit être appliqué, auquel cas VirtuTuile indique un message d'erreur et invite l'utilisateur à réessayer.

Cas d'utilisation :	**Paramétrer un coulis**
Système :	VirtuTuile
Acteur(s) :	Utilisateur
Partie prenante et intérêts :	Utilisateur : Remplacer les propriétés par défaut du coulis dans une surface sélectionnée.
Préconditions :	Une surface doit être créée.
Garantie en cas de succès :	Les propriétés du coulis correspondent aux caractéristiques définies par l'utilisateur et les modifications sont affichées.

Scénarios Principal :	1. (voir use case « Sélectionner une surface ») 2. L'utilisateur définit l'épaisseur du coulis et sa couleur. 3. VirtuTuile applique les modifications sur les propriétés du coulis. 4. VirtuTuile affiche les modifications sur la surface.
Scénarios Secondaires :	Ligne 2 : L'épaisseur du coulis est inférieure ou égale à 0, auquel cas VirtuTuile indique une erreur et invite l'utilisateur à renseigner une valeur supérieure à 0.

Cas d'utilisation :	**Nouveau Projet**
Système :	VirtuTuile
Acteur(s) :	Utilisateur
Partie prenante et intérêts :	Utilisateur : Créer un nouveau projet
Préconditions :	
Garantie en cas de succès :	Un nouveau projet est affiché.
Scénarios Principal :	1. L'utilisateur clic sur le bouton de création d'un nouveau projet. 2. VirtuTuile informe l'utilisateur que les données non-sauvegardés seront supprimées et propose un bouton « Valider ». 3. VirtuTuile affiche le projet
Scénarios Secondaires :	Ligne 2 : Le projet est sauvegardé, auquel cas le projet passe directement à la ligne 3.

Cas d'utilisation :	**Enregistrer projet**
Système :	VirtuTuile
Acteur(s) :	Utilisateur
Partie prenante et intérêts :	Utilisateur : Enregistrer un projet
Préconditions :	
Garantie en cas de succès :	Le projet est enregistré.

Scénarios Principal :	1. L'utilisateur clic sur le bouton d'enregistrement du projet. 2. VirtuTuile ouvre l'explorateur de fichier. 3. L'utilisateur sélectionne un emplacement et nomme son projet**.** 4. VirtuTuile enregistre le fichier. 5. VirtuTuile ferme l'explorateur de fichier
Scénarios Secondaires :	

Cas d'utilisation :	**Charger projet**
Système :	VirtuTuile
Acteur(s) :	Utilisateur
Partie prenante et intérêts :	Utilisateur : Charger un projet
Préconditions :	Un enregistrement d'un projet doit exister.
Garantie en cas de succès :	Le projet est chargé et affiché à l'écran.
Scénarios Principal :	1. L'utilisateur clic sur le bouton « Charger projet ». 2. VirtuTuile affiche un explorateur de fichier. 3. L'utilisateur sélectionne le projet désiré. 4. VirtuTuile charge le fichier. 5. VirtuTuile ferme la fenêtre. 6. VirtuTuile affiche le projet issu du chargement.
Scénarios Secondaires :	<u>Ligne 5 :</u> Le fichier sélectionné n'est pas un projet, l'utilisateur est invité à réessayer en sélectionnant un fichier approprié.

Cas d'utilisation :	**Editer un type de matériau**
Système :	VirtuTuile
Acteur(s) :	Utilisateur
Partie prenante et intérêts :	Utilisateur : Remplacer les propriétés par défaut des matériaux dans une surface sélectionnée.
Préconditions :	Une surface est créée.
Garantie en cas de succès :	Les propriétés d'un type de matériaux dans la surface sélectionnée correspondent aux caractéristiques définies par l'utilisateur.
Scénarios Principal :	1. (voir use case « Sélectionner une surface »)

	2. L'utilisateur sélectionne le type de matériau dans le panneau d'édition. 3. L'utilisateur modifie ses propriétés (dimension, couleur). 4. VirtuTuile applique les propriétés au type de matériau sélectionné.
Scénarios Secondaires :	

Cas d'utilisation :	**Fusionner deux surfaces**
Système :	VirtuTuile
Acteur(s) :	Utilisateur
Partie prenante et intérêts :	Utilisateur : Fusionner deux surfaces.
Préconditions :	Deux surfaces sont créées.
Garantie en cas de succès :	Les deux surfaces sélectionnées se fusionnent et s'affichent comme une seule surface sur le plan de travail. Le motif présent sur la plus grande surface s'applique sur la surface finale.
Scénarios Principal :	1. (L'utilisateur sélectionne et déplace une surface sur une autre surface, voir use case « Déplacer une surface ») 2. VirtuTuile propose de fusionner ces deux surfaces. 3. L'utilisateur valide la fusion. 4. VirtuTuile fusionne les deux surfaces. 5. VirtuTuile applique le motif de la plus grande surface sur la surface fusionnée. 6. VirtuTuile affiche les changements.
Scénarios Secondaires :	

Cas d'utilisation :	**Coller deux surfaces**
Système :	VirtuTuile
Acteur(s) :	Utilisateur
Partie prenante et intérêts :	Utilisateur : Coller deux surfaces verticalement ou horizontalement.
Préconditions :	Deux surfaces sont créées.
Garantie en cas de succès :	Les deux surfaces sont collées l'une à l'autre, verticalement ou horizontalement.
Scénarios Principal :	1. L'utilisateur sélectionne une surface (voir use case

	« Sélectionner une surface »). 2. VirtuTuile propose de coller la surface à une autre, horizontalement ou verticalement. 3. L'utilisateur valide l'une des propositions. 4. L'utilisateur sélectionne une deuxième surface (voir use case « Sélectionner une surface »). 5. VirtuTuile colle les deux surfaces en fonction de ce que l'utilisateur a décidé. 6. VirtuTuile affiche les modifications.
Scénarios Secondaires :	

Cas d'utilisation :	**Aligner deux surfaces**
Système :	VirtuTuile
Acteur(s) :	Utilisateur
Partie prenante et intérêts :	Utilisateur : Aligner une surface par rapport à la / le gauche/droite/haut/bas d'une autre surface.
Préconditions :	Deux surfaces sont créées.
Garantie en cas de succès :	La surface à aligner est aligner conformément au choix de l'utilisateur par rapport à l'autre surface.
Scénarios Principal :	1. L'utilisateur sélectionne une surface (voir use case « Sélectionner une surface »). 2. VirtuTuile propose à l'utilisateur d'aligner la surface à une autre par rapport à un côté. 3. L'utilisateur valide l'alignement en sélectionnant un côté. 4. L'utilisateur sélectionne la deuxième surface (voir use case « Sélectionner une surface »). 5. VirtuTuile aligne la surface par rapport à la deuxième conformément à la demande. 6. VirtuTuile affiche les modifications.
Scénarios Secondaires :	

Cas d'utilisations :	**Redimensionner une surface**
Acteur(s) :	Utilisateur
Type :	
Description :	L'utilisateur sélectionne la surface puis le sommet de la surface qu'il souhaite repositionner. Puis, il déplace le sommet à son nouvel emplacement. Les côtés liés au sommet sélectionné se repositionnent afin d'être relié au sommet.

Cas d'utilisations :	**Supprimer une surface**
Acteur(s) :	Utilisateur
Type :	
Description :	L'utilisateur sélectionne une surface en cliquant dessus. Puis il clique sur la touche « SUPPR » de son clavier ou sur. La surface sélectionnée est supprimée.

Cas d'utilisations :	**Afficher/Cacher une grille magnétique**
Acteur(s) :	Utilisateur
Type :	
Description :	L'utilisateur clique sur le bouton « Grille magnétique » pour afficher ou cacher la grille magnétique.

Cas d'utilisations :	**Changer le niveau de zoom**
Acteur(s) :	Utilisateur
Type :	
Description :	L'utilisateur utilise des boutons précis afin de zoomer ou dézoomer à l'infini sur le plan de travail.

Cas d'utilisations :	**Afficher les informations d'une surface**
Acteur(s) :	Utilisateur
Type :	
Description :	L'utilisateur clique sur une surface pour consulter les données de celle-ci dans la barre d'état.

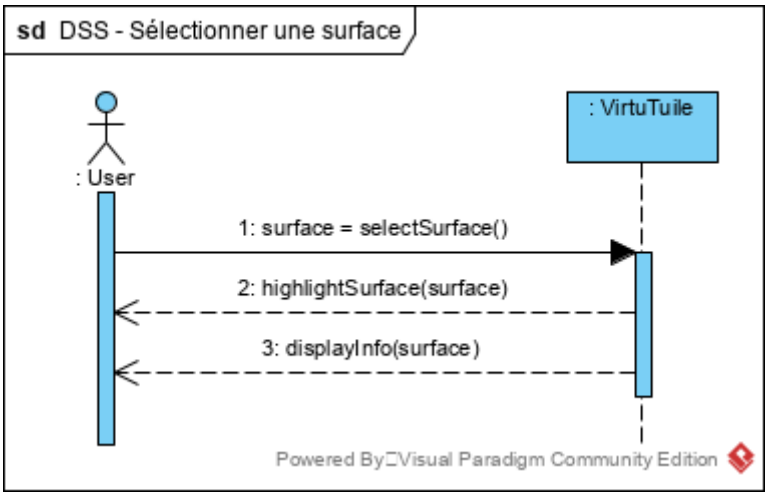
Cas d'utilisations :	**Undo / Redo**
Acteur(s) :	Utilisateur
Type :	
Description :	L'utilisateur utilise un bouton lui permettant d'annuler la dernière modification ou d'annuler la dernière annulation.

Cas d'utilisations :	**Changer d'unité de mesure**
Acteur(s) :	Utilisateur
Type :	
Description :	L'utilisateur utilise un bouton dans la barre d'état lui permettant de changer l'unité de mesure.

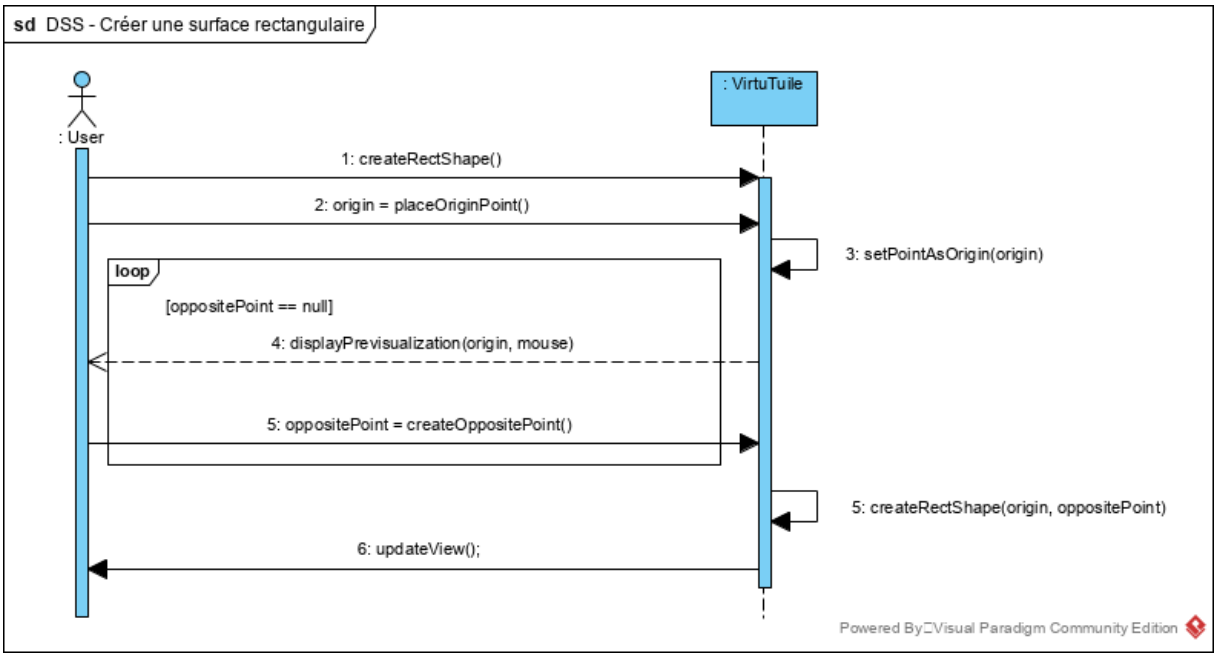
Cas d'utilisations :	**Pivoter**
Acteur(s) :	Utilisateur
Type :	
Description :	L'utilisateur utilise un bouton pour faire pivoter une surface ou un motif.

C. Diagramme de séquence système

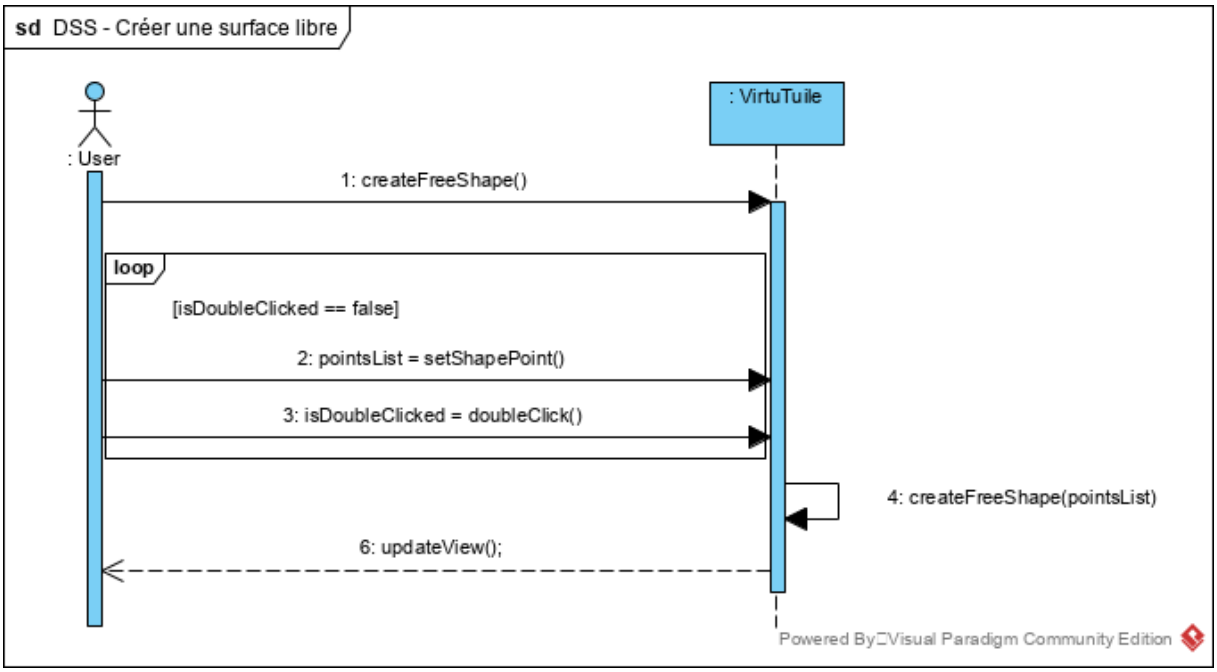
Sélectionner une surface



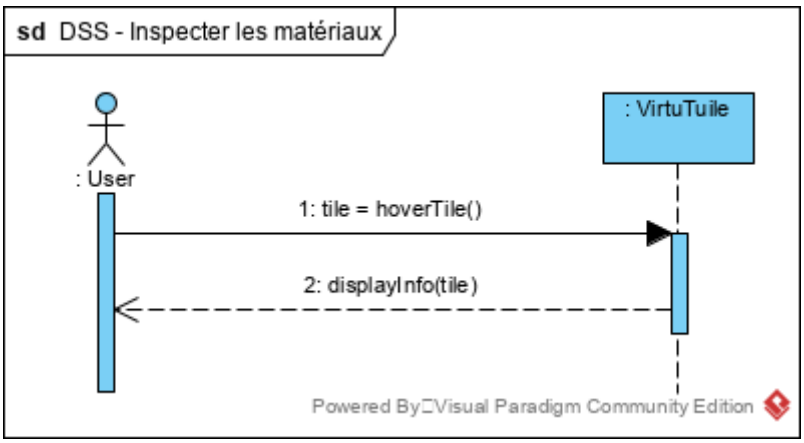
Créer une surface rectangulaire



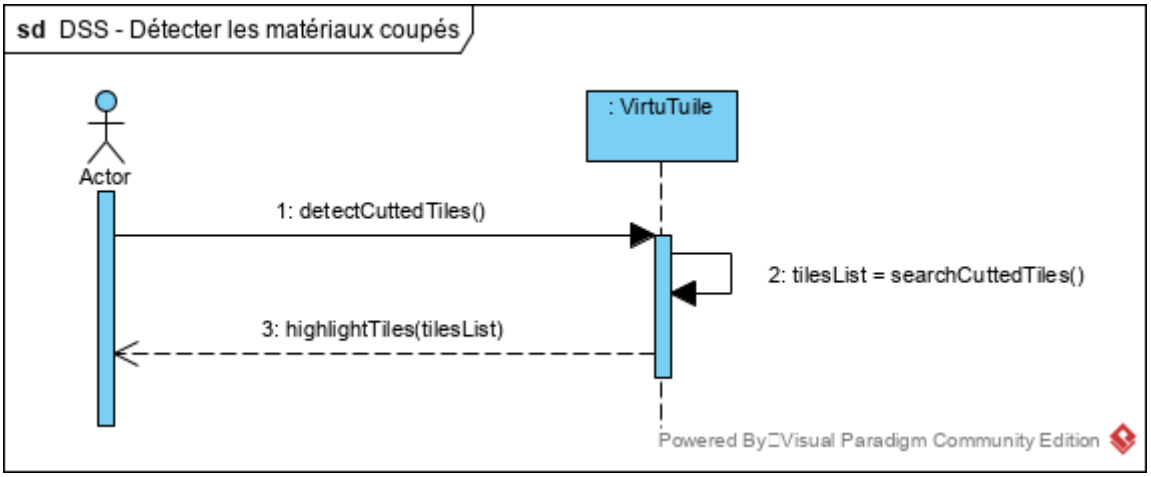
Créer une surface libre



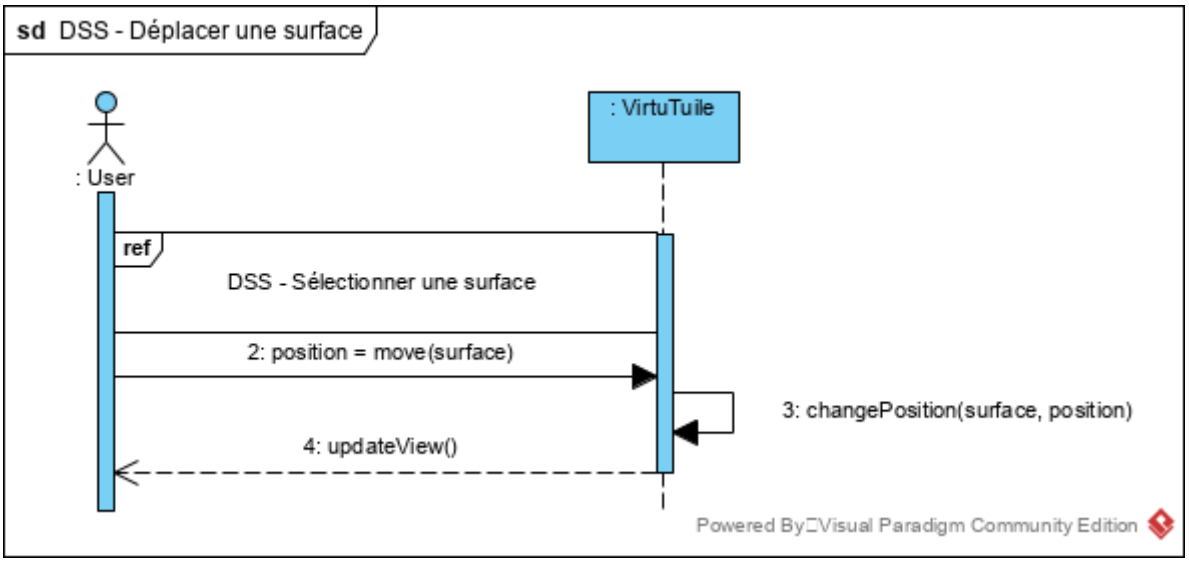
Inspecter les matériaux



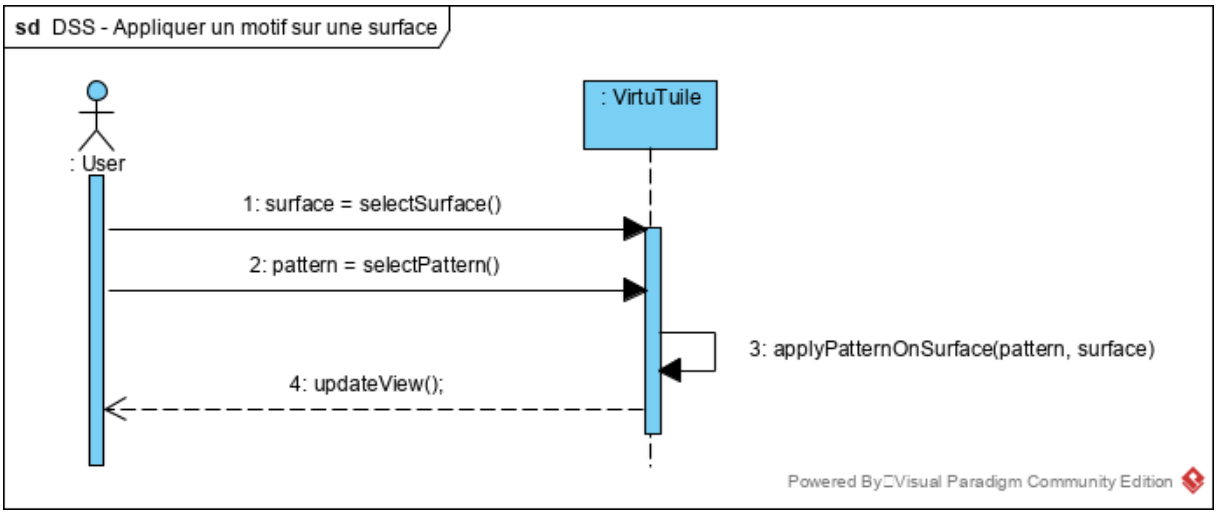
Détecter les matériaux coupés



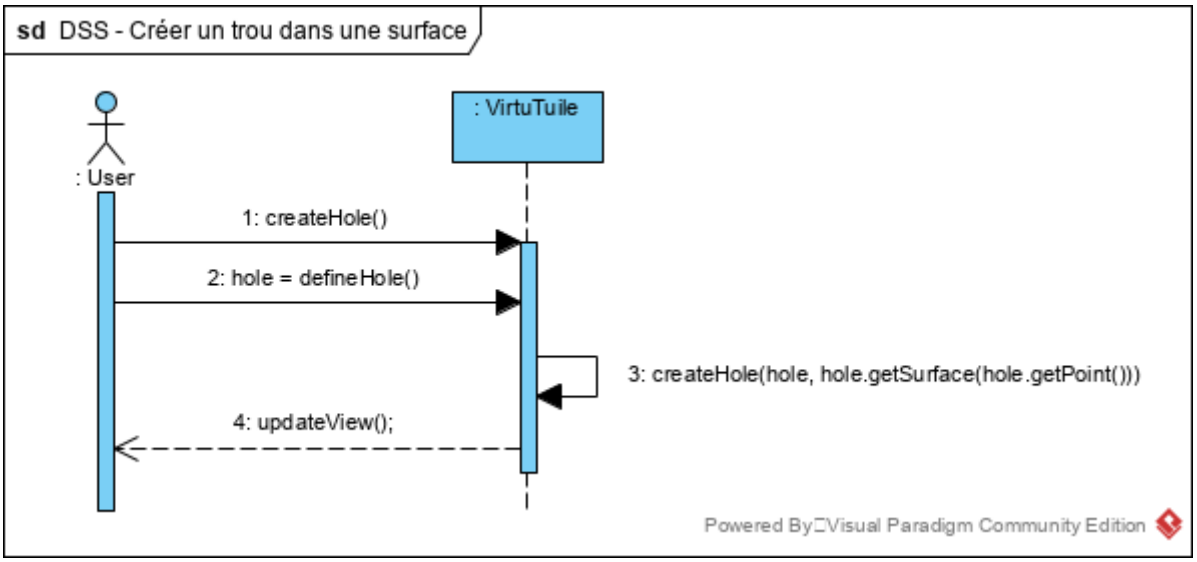
Déplacer une surface



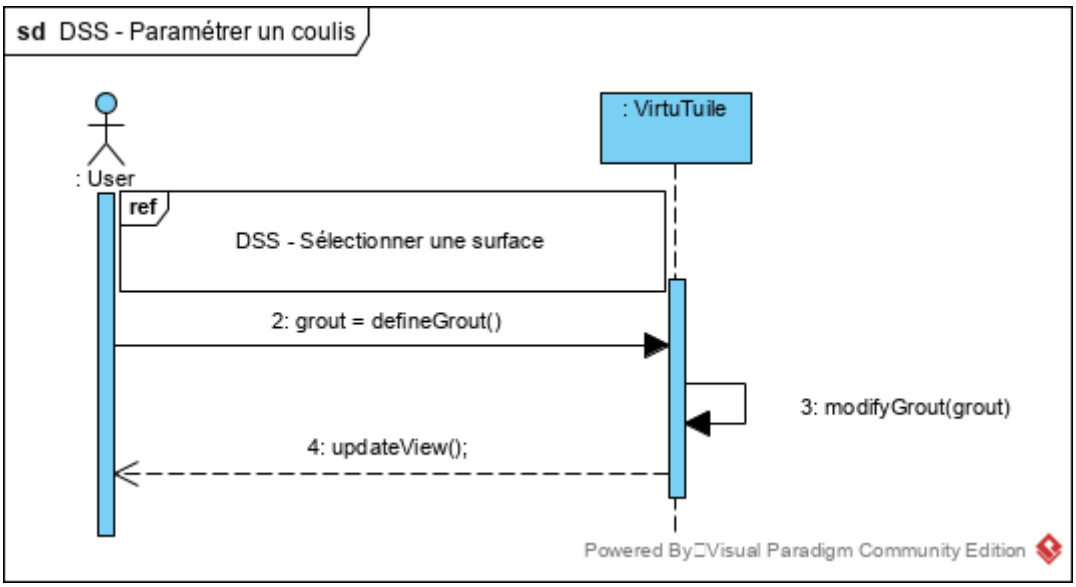
Appliquer un motif sur une surface



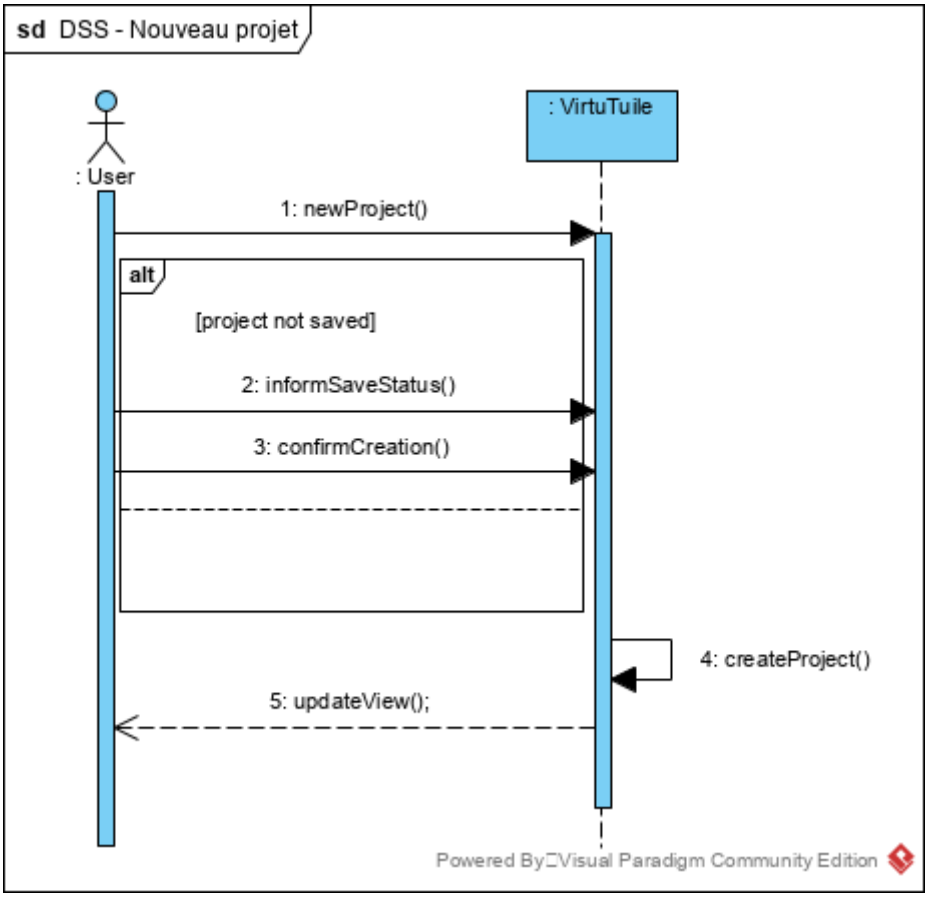
Créer un trou dans une surface



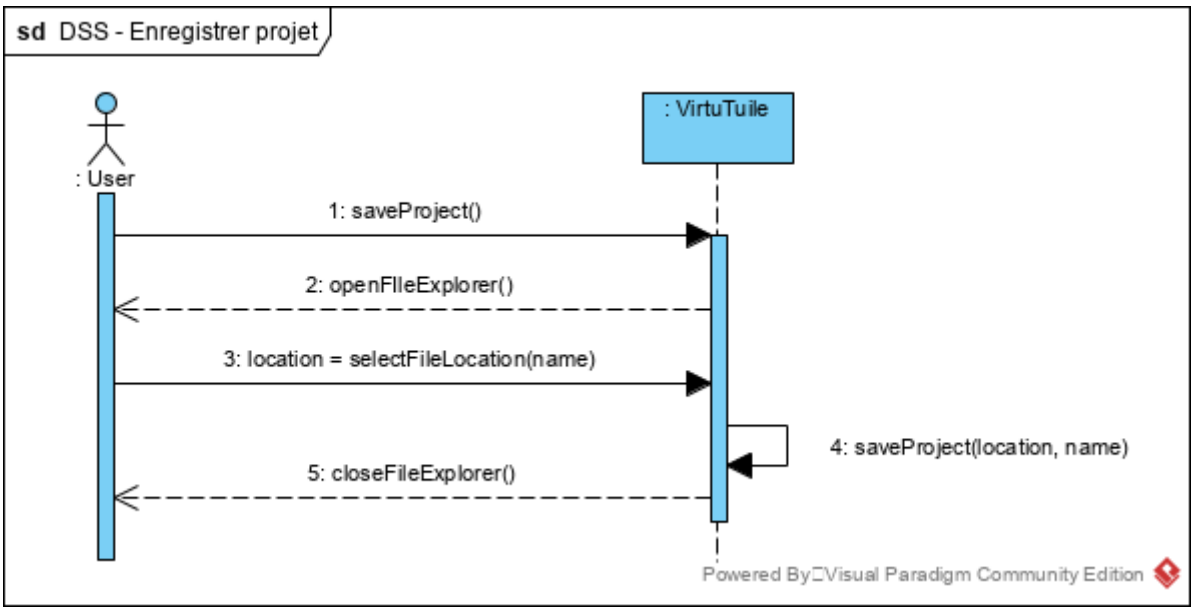
Paramétrer un coulis



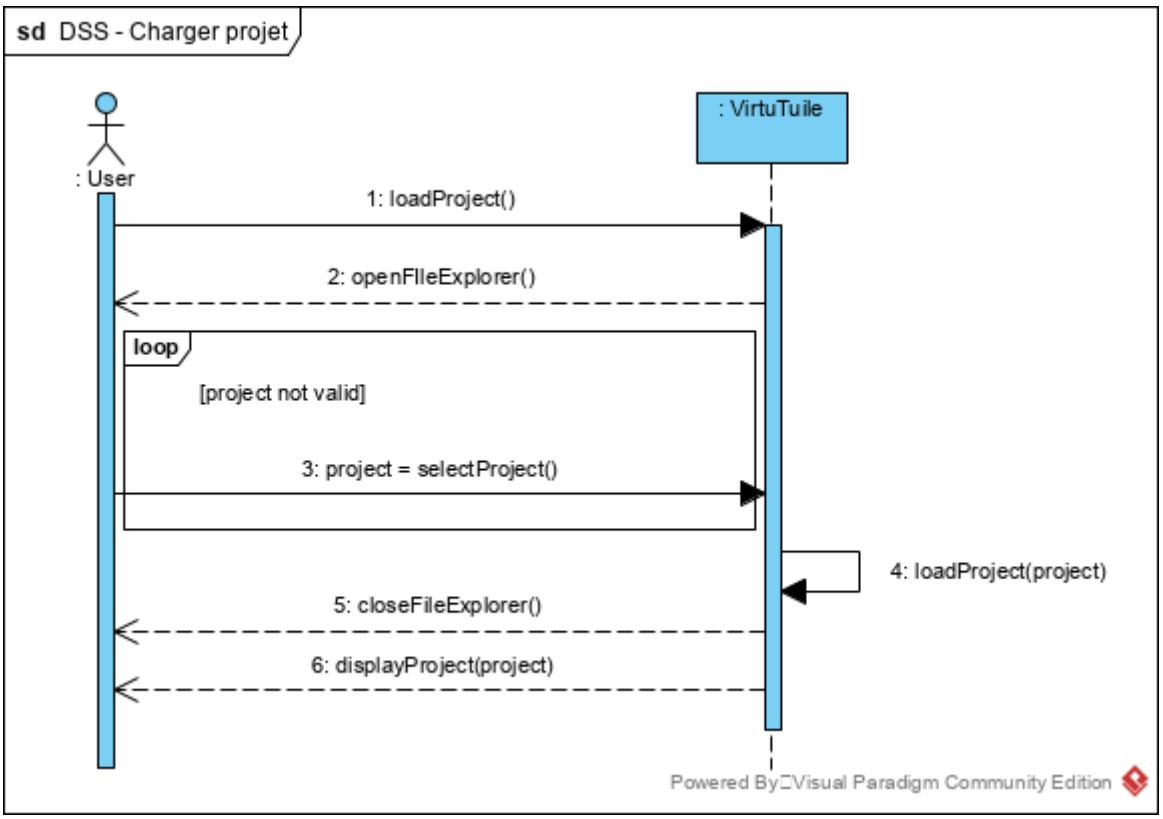
Nouveau projet



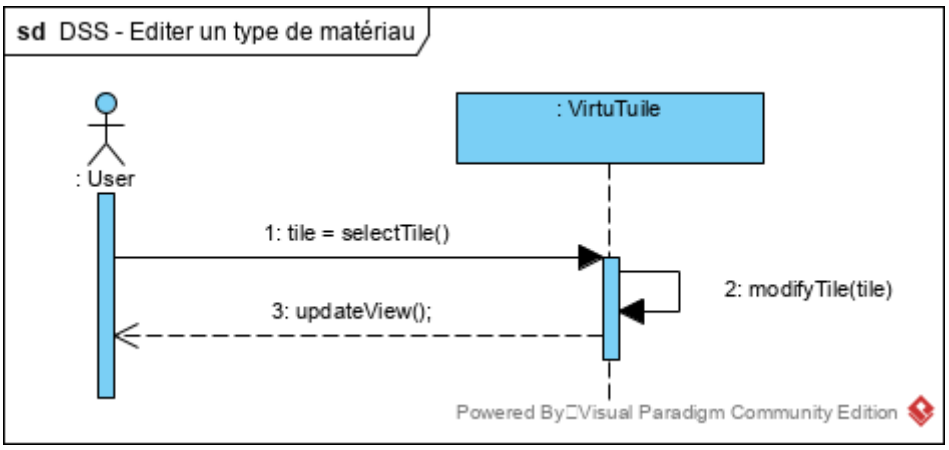
Enregistrer projet



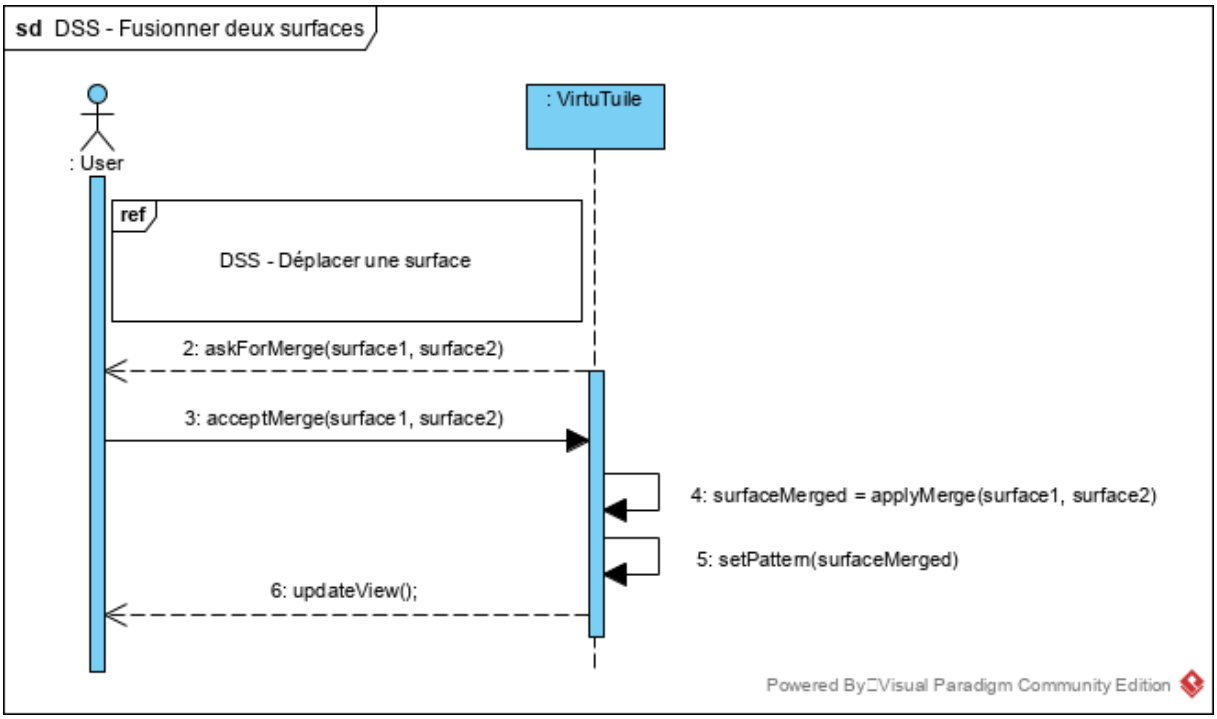
Charger projet



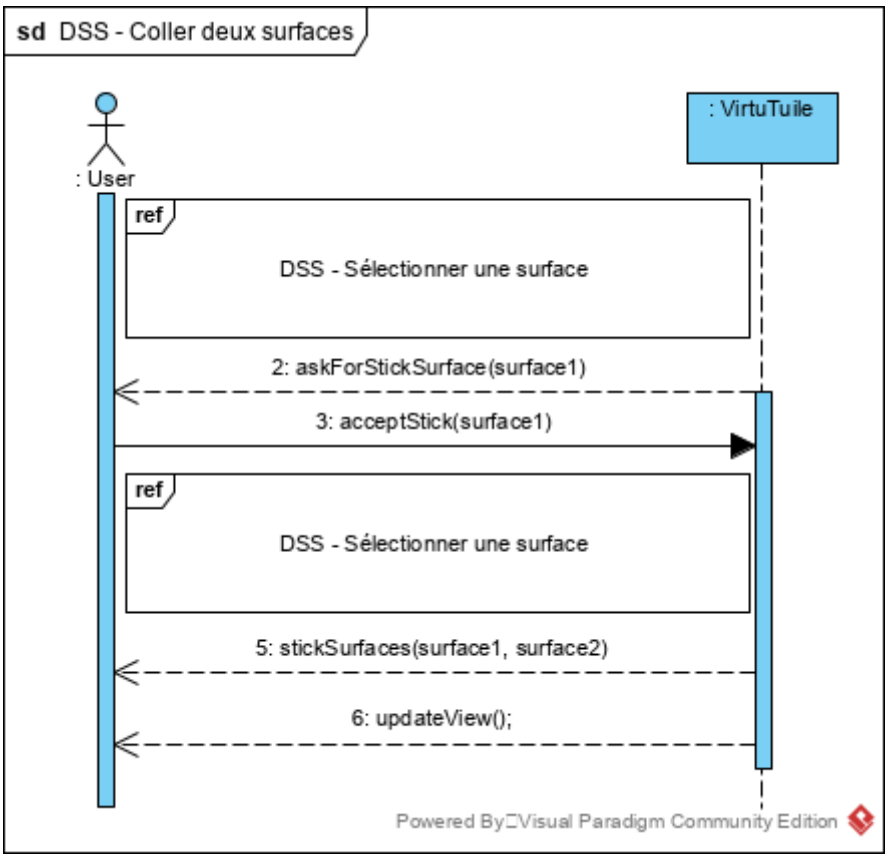
Editer un type de matériau



Fusionner deux surfaces



Coller deux surfaces



Aligner deux surfaces