UNIVERSITÉ LAVAL

16/10/2018

GLO-2004 : Génie logiciel orienté objet

Projet de session

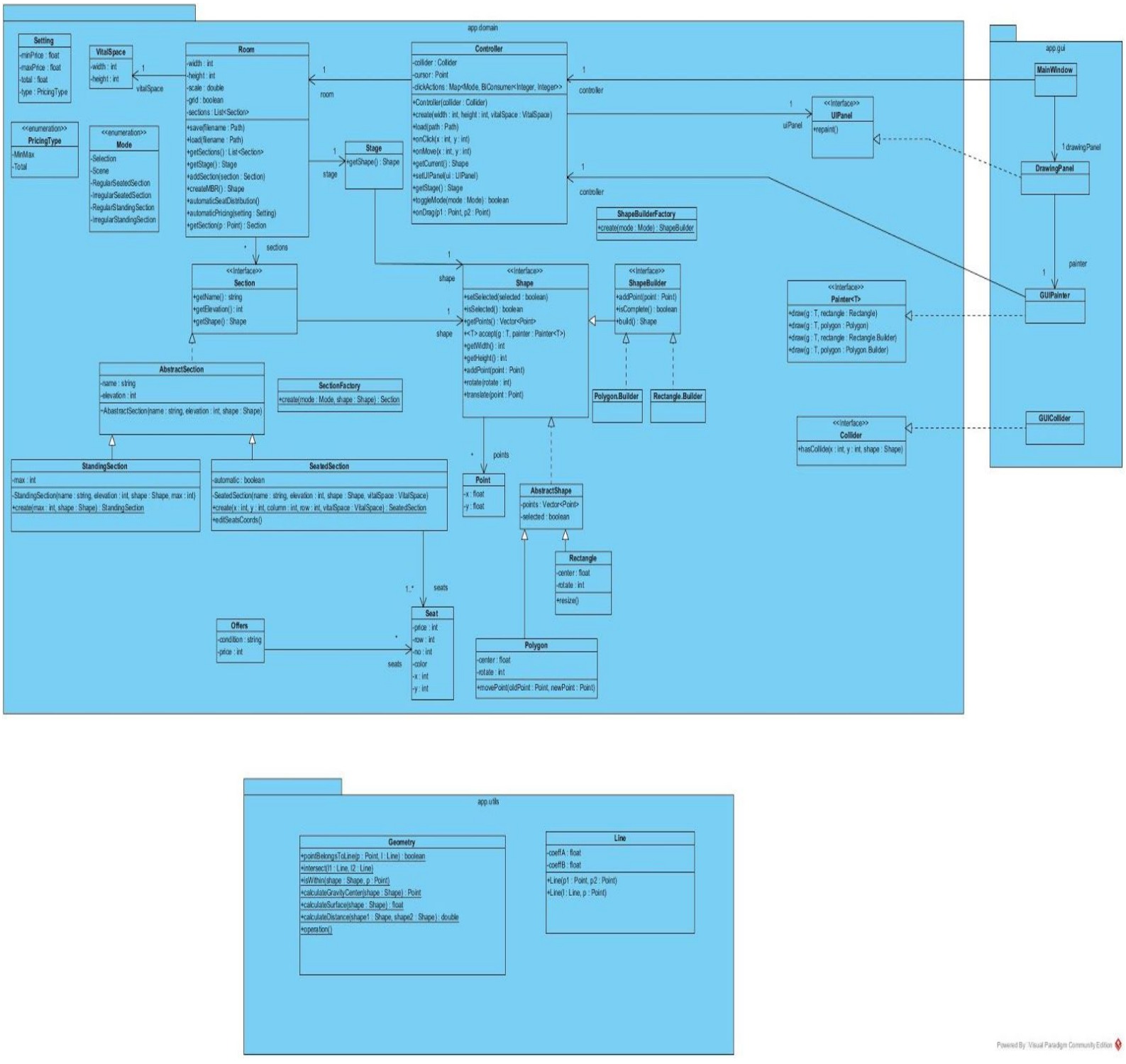
Livrable 2

Equipe 27

Léandre Gagnon-Lewis
Vincent Rasquier
Nicolas Xavier Lasso
Anthony Marchand

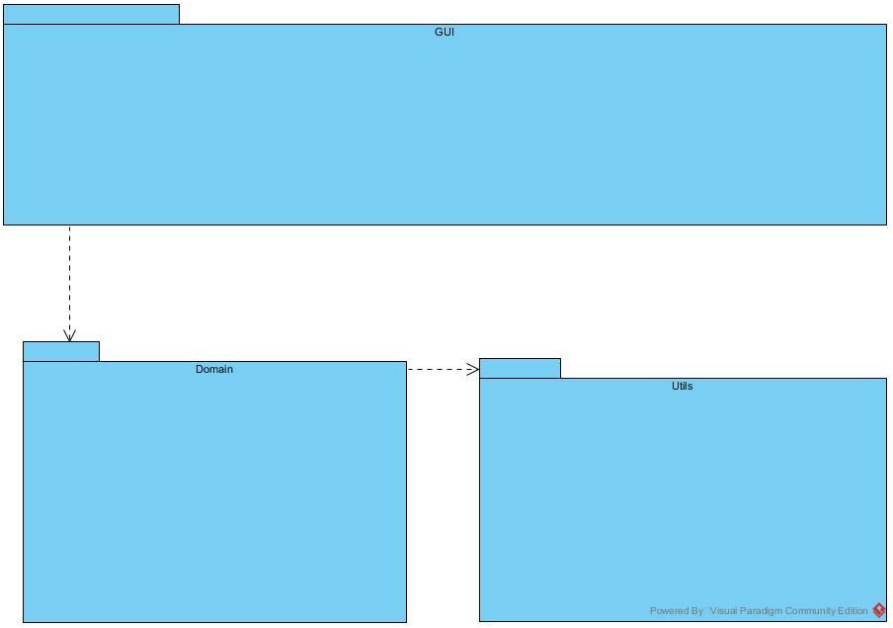
Table des matières

1. Diagramme de classe de conception



2. Architecture logique

Diagramme d'architecture



Texte explicatif

L'architecture de l'application *VenueDesigner* peut être représenté par le diagramme présenté ci-dessus. Il est aussi possible de voir quelles classes appartiennent à quels packages dans le diagramme de classe à la section 1.

La couche la plus élevée correspond à l'interface utilisateur. Toutes les classes de cette couche sont regroupées dans le package GUI. Parmi ces classes, on retrouve la *MainWindow* qui correspond à la fenêtre de l'application et dans laquelle tous les *JComponent* de *Swing* seront imbriqués (ils n'ont pas tous été présentés sur le diagramme de classe à cause de leur nombre élevé). Le *DrawingPanel* est un de ces *JComponent* et fait lui aussi partie de cette classe. De plus, on retrouve les classes *GUIPainter* et *GUICollider* qui servent respectivement à modifier l'affichage du *DrawingPanel* et à détecter les collisions des formes dans celui-ci.

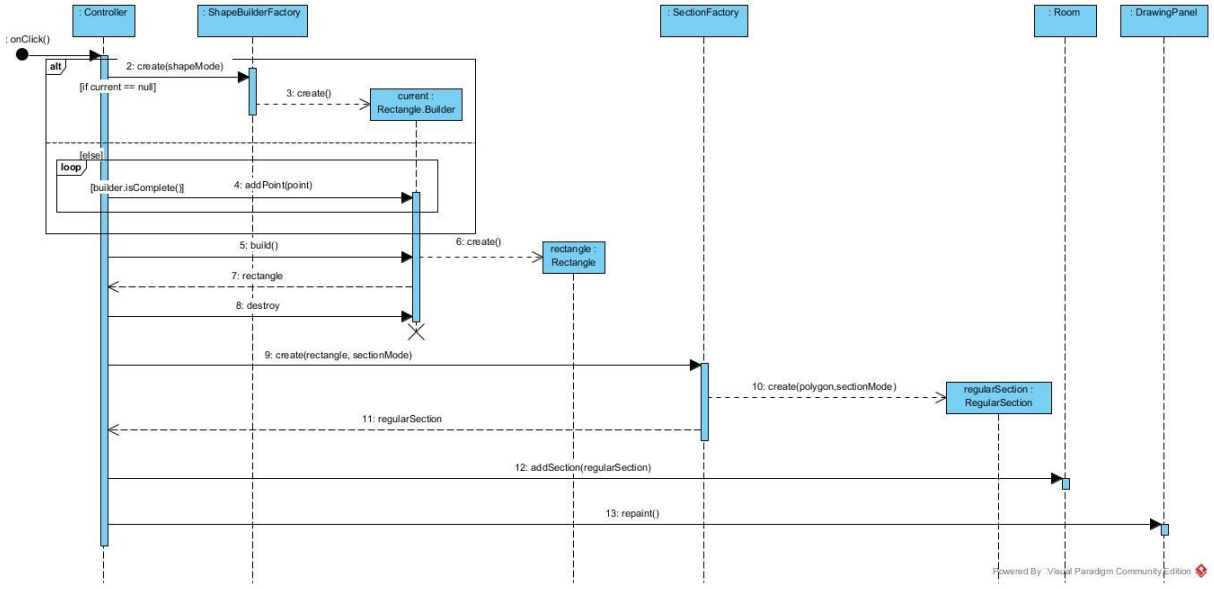
Le package *Domain* comprend les classes correspondantes à la logique applicative. Une classe très importante, faisant partie de ce package, est *Controller* qui est la seule classe de ce package qui interagit avec les classes du *GUI*. Elle a la responsabilité de transmettre les messages aux autres classes du domaine. On y retrouve aussi des interfaces servant à implémenter certaines classes de *GUI*.

Finalement, le Package *Utils* comprend la classe *Geometry* qui contient uniquement des méthodes statiques. Elle contient aussi la classe *Line*. D'autres classes utilitaires pourront être ajoutées à ce package si nécessaire.

3. Diagrammes de séquence de conception

3.1. Création d'une section rectangulaire

Diagramme

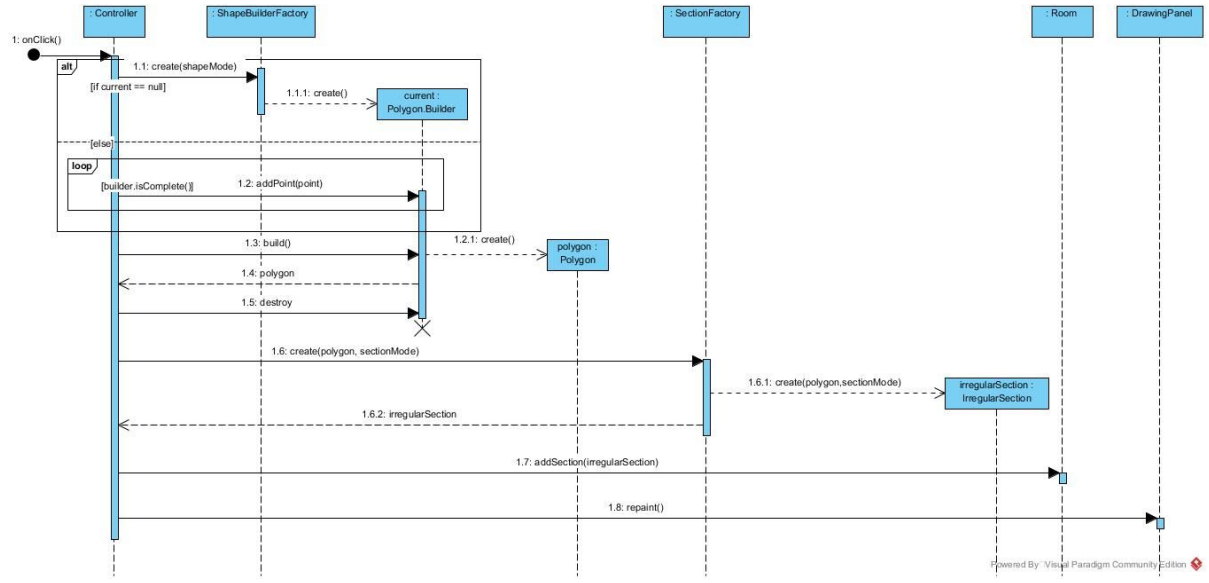


Texte explicatif

Le premier appel est celui de *Controller* avec la méthode *onClick()* qui est appelée à chaque clique de souris. *Controller* va ensuite appeler *ShapeBuilderFactory* en lui envoyant le mode comme information. *ShapeBuilderFactory* sait ainsi quel type de forme créer et, dans notre cas, instancier un *Rectangle.Builder* nommé *current*.*Controller* va ensuite ajouter de nouveaux points à *current*, jusqu'à ce que la forme soit complète (c'est-à-dire que la figure soit fermée). Une fois ceci fait, *Controller* appelle la méthode *build()* de *current* pour que ce dernier puisse créer un *Rectangle*, nommé simplement *rectangle*. *Controller* supprime ensuite *current* qui a joué son rôle et envoie *rectangle* et le mode à *SectionFactory* pour que ce dernier puisse créer une section régulière. Enfin, *Controller* appelle la méthode *addSection()* de *Room* pour ajouter la section créée dans la liste de *Room*, et appelle par la suite la méthode *repaint()* de *DrawingPanel* afin d'afficher graphiquement la mise à jour de la salle.

3.2. Création d'une section irrégulière

Diagramme



Texte explicatif

Afin de créer une section irrégulière, il nous faut cliquer sur la souris, et la méthode *onClick()* de *Controller* sert de déclencheur. *Controller* appelle *ShapeBuilderFactory*, qui va créer une *Shape* particulière selon un certain mode, et ces modes sont regroupés dans une classe *Enum* nommée *Mode*.

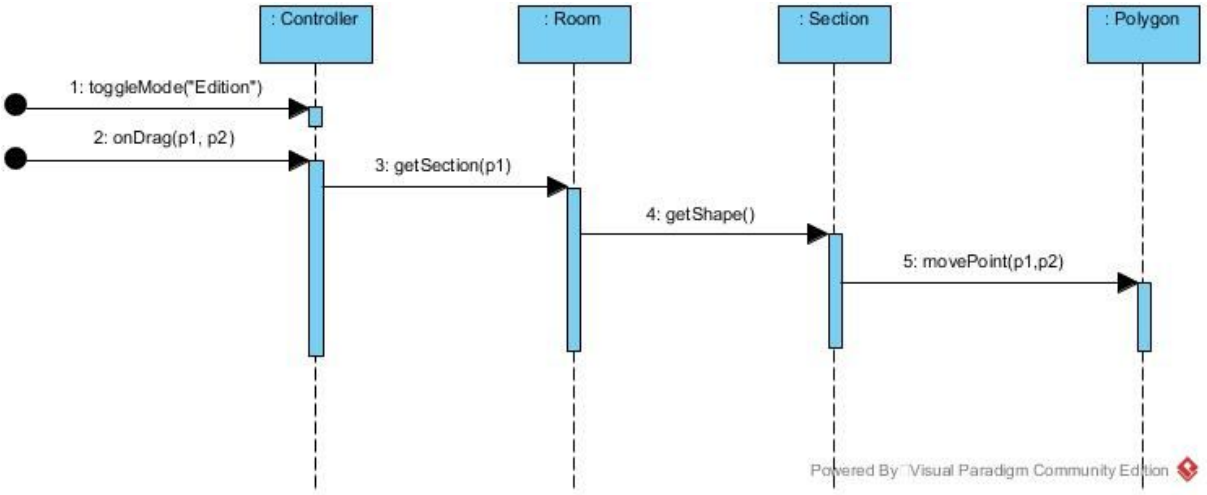
Dans le cas des sections irrégulières, si la figure n'est pas en cours de construction, alors, un objet *current* de *Polygon.Builder* est créé, sinon, la méthode *addPoint()* est appelée en boucle jusqu'à ce que la figure soit complète, c'est-à-dire que l'on ajoute des points à notre polygone jusqu'à ce que que la méthode *isComplete()* retourne *true* . Le cas échéant, le current de *Polygon.Builder* est créé ainsi qu'un objet *polygon*. Par la suite, le *current* est détruit.

Controller fait alors un appel à *SectionFactory* avec la méthode *create()*, en prenant en paramètre l'objet *polygon* et le *mode*. Par la suite, *SectionFactory* crée un objet *section* avec la méthode *create()* prenant en paramètre l'objet *polygon* et *sectionMode* afin de créer une section assise ou debout, mais dans tous les cas, cette section sera de type irrégulière.

Ensuite, *Controller* utilise la méthode *addSection* pour ajouter la section créée dans la salle. Et enfin, *Controller* utilise la méthode *repaint()* pour dessiner dans *DrawingPanel*.

3.3. Modifier la position d’un point d’une section irrégulière

Diagramme



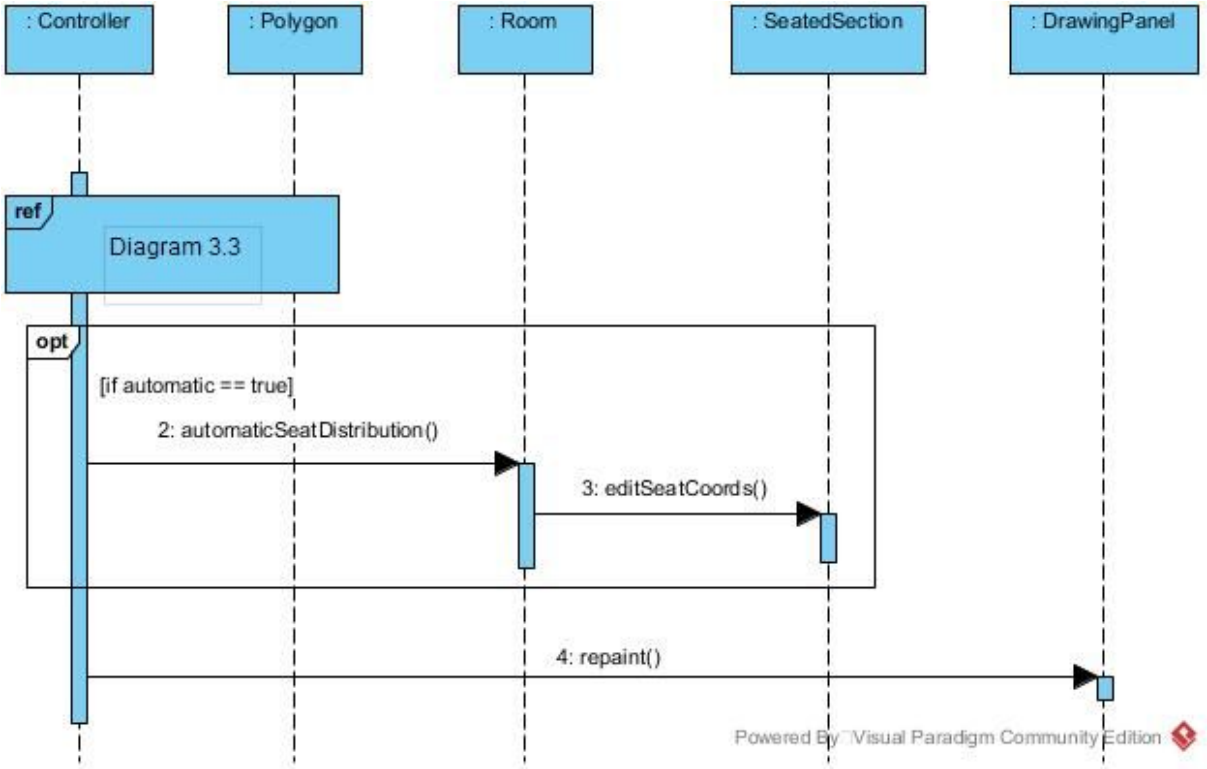
Texte explicatif

Le diagramme de séquence ci-dessus permet de représenter comment les objets de *VenueDesigner* collaborent entre-eux pour modifier la position d’un point d’une section irrégulière. Premièrement, il est important que *Controller* soit en mode “Edition”, pour cela la méthode *controller.toggleMode(“Edition”)*, doit être appelée.
Ensuite, quand la méthode *controller.onDrag(p1,p2)* est appelée, *Controller* appelle la méthode *room.getSection(p1)* de la classe *Room* qui lui retourne la section à modifier. De plus, la méthode *section.getShape()* sera appelée sur cette section pour obtenir la forme qui la définit.

Puisque la section est irrégulière, la classe retournée sera de type *Polygon* (classe enfant de la classe *AbstractShape*), la méthode *polygon.movePoint(p1,p2)* sera donc appelée et modifiera la position du point. Finalement, *Controller* appellera la fonction *drawingPanel.repaint()* pour afficher les changements.

3.4. Redistribution automatique des sièges lorsqu'il y a une modification

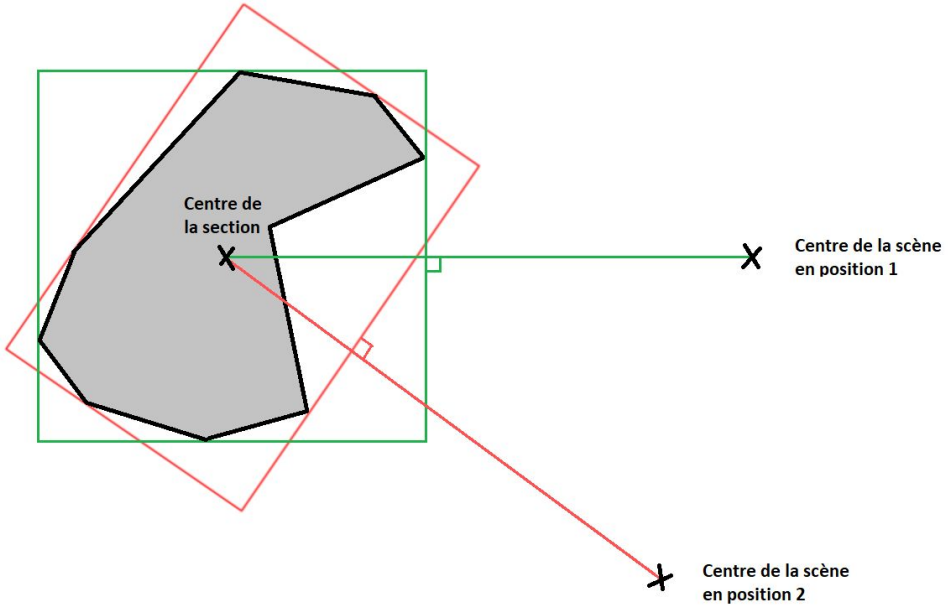
Diagramme



Texte explicatif et pseudocode

La distribution automatique des sièges se fait en plusieurs étapes. La première étape consiste à tracer le **rectangle à limite minimum** (*minimum bounding rectangle (MBR)* en anglais) de la section. Ce rectangle se fera en fonction de la position de la scène, afin que les sièges soient orientés dans sa direction. Pour cela, on tracera la ligne passant par le centre de la scène et par le centre de la section, puis, à partir du centre de la scène comme point de départ, on se déplacera le long de cette ligne pour tester si une ligne perpendiculaire à celle-ci contient un point de la section. La première ligne répondant à ces critère constitue le premier côté du rectangle.

Afin de trouver les autres côtés, on applique la même logique : on vérifie en partant du centre de la section si chaque ligne perpendiculaire au premier côté passe dans la section. Une fois qu'une droite passant par l'extérieur de la section est trouvée, on prend la précédente qui est donc le deuxième côté du rectangle. On applique la même chose pour trouver le troisième côté à partir du deuxième côté jusqu'à trouver le quatrième et dernier côté.



Exemple de rectangles à la limite minimum en fonction de deux positions différentes de la scène

Une fois le rectangle à la limite minimum trouvé, on parcourt ce dernier, en itérant en fonction de la largeur et de la longueur de l'espace vital, et on vérifie si chaque point se trouve à l'intérieur de la section. Chaque point dans la section trouvé est ainsi le centre d'un siège.

En détaillant tout ce processus, on arrive à devoir utiliser différents algorithmes de géométrie. Tout d'abord, afin de tracer toutes les droites nécessaires, il faut déterminer toutes les équations de droite de la forme $y = ax + b$, afin d'itérer dessus, tracer leur perpendiculaire, voir si un point appartient à une droite...

Les méthodes *Line()* sont des constructeurs de la classe *Line*.

Pseudocode pour déterminer l'équation d'une droite à partir de deux points :

```
function Line(Point A, Point B)
        coefA = (B.getY() - A.getY()) / (B.getX() - A.getX())
        coefB = A.getY() - coefA * A.getX()
```

Pseudocode pour déterminer l'équation d'une droite perpendiculaire à une autre à partir d'une équation de droite et d'un point :

```
function Line(Line L, Point A)
        coeffA = -1 / L.getCoeffA()
        coeffB = A.getY() - coeffA * A.getX()
```

Pseudocode pour déterminer si un point appartient à une droite :

```
function pointBelongsToLine(Line L, Point A)
        y = D.getCoeffA() * A.getX() + L.getCoeffB()
        if y == A.getY()
                return true
        else
                return false
```

Pseudocode pour déterminer le point d'intersection de deux droites à partir de leur équation :

```
function intersect(Line L1, Line L2)
        x = (L2.getCoeffB() - L1.getCoeffB()) / (L1.getCoeffA() - L2.getCoeffA())
        y = L1.getCoeffA() * coeffX + coeffB
        Point p = new Point(x, y)
        return p
```

Pseudocode pour déterminer si un point est dans un polygone (méthode du Ray casting[1]) :

```
function isWithin(Shape shape, Point point)
        bool isWithin = false
        i, j = shape.getPoints().size() - 1
        For i = 0 to shape.getPoints().size()
                if ((shape.getPoints()[i].getY() < point.getY() and shape.getPoints()[j].getY()
                >= point.getY()) or  (shape.getPoints()[j].getY() < point.getY() and
                shape.getPoints()[i].getY() >= point.getY()))
                        if (shape.getPoints()[i].getX() + (point.getY() -
                        shape.getPoints()[i].getY()) / (shape.points[j].getY() -
                        shape.getPoints()[i].getY()) * (shape.getPoints()[j].getX() -
                        shape.getPoints()[i].getX()) < point.getX())
                                isWithin  != isWithin
                j = i
        return isWithin
```

Pseudocode pour tracer le rectangle à la limite minimum :

```
function createMBR(shape, stage)
        Shape mbr = new Shape()
        line = new Line(shape.getCenter(), stage.getCenter())
        bool firstBorder = false
```

[1] sources : https://en.wikipedia.org/wiki/Point_in_polygon#Ray_casting_algorithm
http://alienryderflex.com/polygon/

```
// On cherche le premier côté
while (firstBorder != true)
        For i = stage.getCenter().getX() to shape.getCenter().getX()
                j = line.getCoeffA() * i + line.getCoeffB()
                Point tempPoint = new Point(i, j)
                Line tempLine = new Line(line, tempPoint)
                        For y = DrawingPanel.getMinimumSize().getHeight() to
                        DrawingPanel.getMaximumSize().getHeight()
                        x = (y - tempLine.getCoeffB()) / tempLine.getCoeffA()
                        tempPoint = new Point(x, y)
                        if (isWithin(shape, tempPoint))
                                line = tempLine
                                firstBorder = true
// On cherche le deuxième côté
firstLine = line
finish = false
For i = shape.getCenter().getY() to DrawingPanel.getMaximumSize().getHeight() and
finish != true
        j = (i - line.getCoeffB()) / line.getCoeffA()
        Point tempPoint = new Point(j, i)
        Line tempLine = new Line(line, tempPoint)
                For x = DrawingPanel.getMinimunSize().getWidth() to
                        DrawingPanel.getMaximumSize().getWidth()
                        y = tempLine.getCoeffA() * x + tempLine.getCoeffB()
                        tempPoint = new Point(x, y)
                        if (isWithin(shape, tempPoint))
                                break
                        if (x = DrawingPanel.getMaximumSize().getWidth() - 1 and
                        !isWithin(shape, tempPoint)
                                mbr.addPoint(intersect(line, tempLine))
                                line = tempLine
                                finish = true
// On cherche le troisième côté
finish = false
For i = shape.getCenter().getX() to DrawingPanel.getMinimunSize().getWidth()
and finish !=true
        j = line.getCoeffA() * i + line.getCoeffB()
        Point tempPoint = new Point(i, j)
        Line tempLine = new Line(line, tempPoint)
                For y = DrawingPanel.getMinimumSize().getHeight() to
                        DrawingPanel.getMaximumSize().getHeight()
                        x = (y + tempLine.getCoeffB()) / tempLine.getCoeffA()
                        tempPoint = new Point(x, y)
                        if (isWithin(shape, tempPoint))
                                break
                        if (x = DrawingPanel.getMaximumSize().getHeight() - 1 and
```

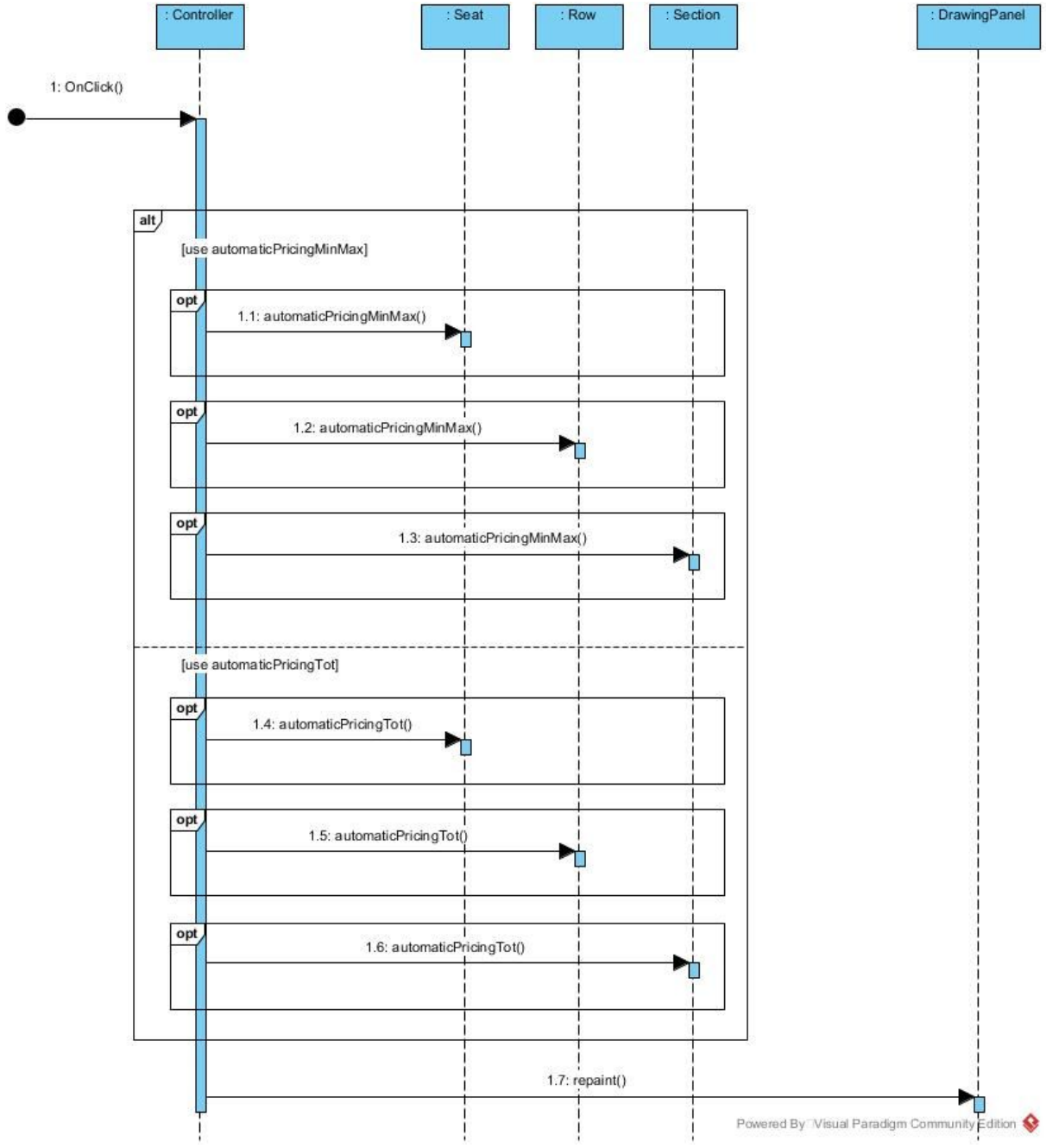
```
                        !isWithin(shape, tempPoint)
                                mbr.addPoint(intersect(line, tempLine))
                                line = tempLine
                                finish = true
        // On cherche le quatrième côté
        finish = false
        For i = shape.getCenter().getY() to DrawingPanel.getMinimumSize().getHeight()
        and finish != true
                j = (i - line.getCoeffB()) / line.getCoeffA()
                Point tempPoint = new Point(j, i)
                Line tempLine = new Line(line, tempPoint)
                        For x = DrawingPanel.getMinimunSize().getWidth() to
                                DrawingPanel.getMaximunSize().getWidth()
                                y = tempLine.getCoeffA() * x + tempLine.getCoeffB()
                                tempPoint = new Point(x, y)
                                if (isWithin(shape, tempPoint))
                                        break
                                if (x = DrawingPanel.getMaximunSize().getWidth() - 1 and
                                !isWithin(shape, tempPoint)
                                        mbr.addPoint(intersect(line, tempLine))
                                        mbr.addPoint(intersect(line, firstLine))
                                        finish = true
```

Pseudocode pour déterminer tous les points de coordonnées des sièges (on utilise les attributs *VitalSpace*, *shape* et *stage* de la classe *Room*) :

```
function  automaticSeatDistribution()
        Shape mbr = createMBR(shape, stage)
        List<Point> coord = new List<Point>()
        For i = (VitalSpace.getWidth()/2) to mbr.getWidth()-(VitalSpace.getWidth()/2)
                For j = (VitalSpace.getHeight()/2) to mbr.getWidth()-(VitalSpace.getHeight()/2)
                        Point temp = new Point(i, j)
                        if temp.IsWithin(shape)
                                coord.add(temp)
        return coord
```

3.5. Affectation automatique des prix en fonction de la distance avec la scène

Diagramme



Texte explicatif et pseudocode

La création des prix automatique peut être attribuée à des sièges, des sections ou des rangées, suivant deux schémas principaux, à savoir :

1) Le cas ou l'on n'attend pas spécialement un revenu total en particulier

Dans ce cas, on rentre en paramètre le minimum et le maximum des prix que l'on veut assigner à un objet. Par conséquent, on ne contrôle pas le revenu final généré par l'événement.

2) Le cas ou l'on veut un revenu total

Dans ce cas, on rentre en paramètre le revenu total de l'événement, et l'application calcule le prix des objets proportionnellement au rendement souhaité. Par conséquent, on ne contrôle pas le prix minimum et maximum attribué à un objet. Si c'était le cas, on pourrait avoir un problème d'inconsistance.

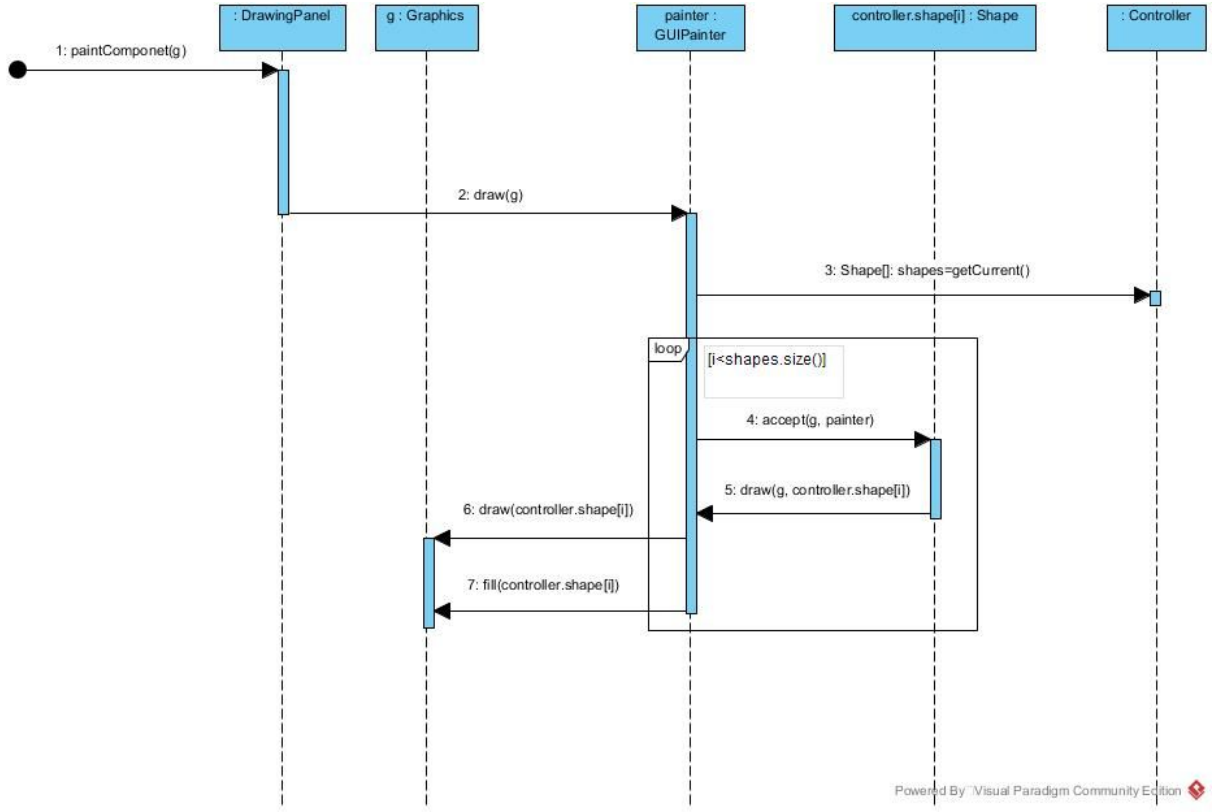
Pseudocode pour la distribution des prix automatique:

```
public float automaticPricing (MinPrix , MaxPrix, TotalRevenus, TypePrix, TypeObjet){

VARIABLES LOCALES:
        distObjet(float) = calculateDistance(scene, objet)    // Ici, l'objet représente un siège,
                                                              // une rangée ou une section
        distMax(float) = calculateDistance(scene, objetLePlusLoin)
        coeff(float) = 1-(distObjet/distMax)    // C'est un résultat en pourcentage plus il est
                                                // grand, plus le prix est élevé
        prix(float)
   INSTRUCTIONS:

   Lire TypePrix
   Si TypePrix == MinMax      // On ne connaît pas le total des revenus que l'on souhaite
        Lire TypeObjet
        Si TypeObjet = "siège" Alors
                prix = coeff*MaxPrix
                Si prix < MinPrix Alors
                        prix = MinPrix
                Fin Si
        Fin Si
        Sinon et Si TypeObjet = "rangée" Alors
                prix = coeff*MaxPrix
                Si prix < MinPrix Alors
                        prix = MinPrix
                Fin Si
        Fin Sinon et Si
```

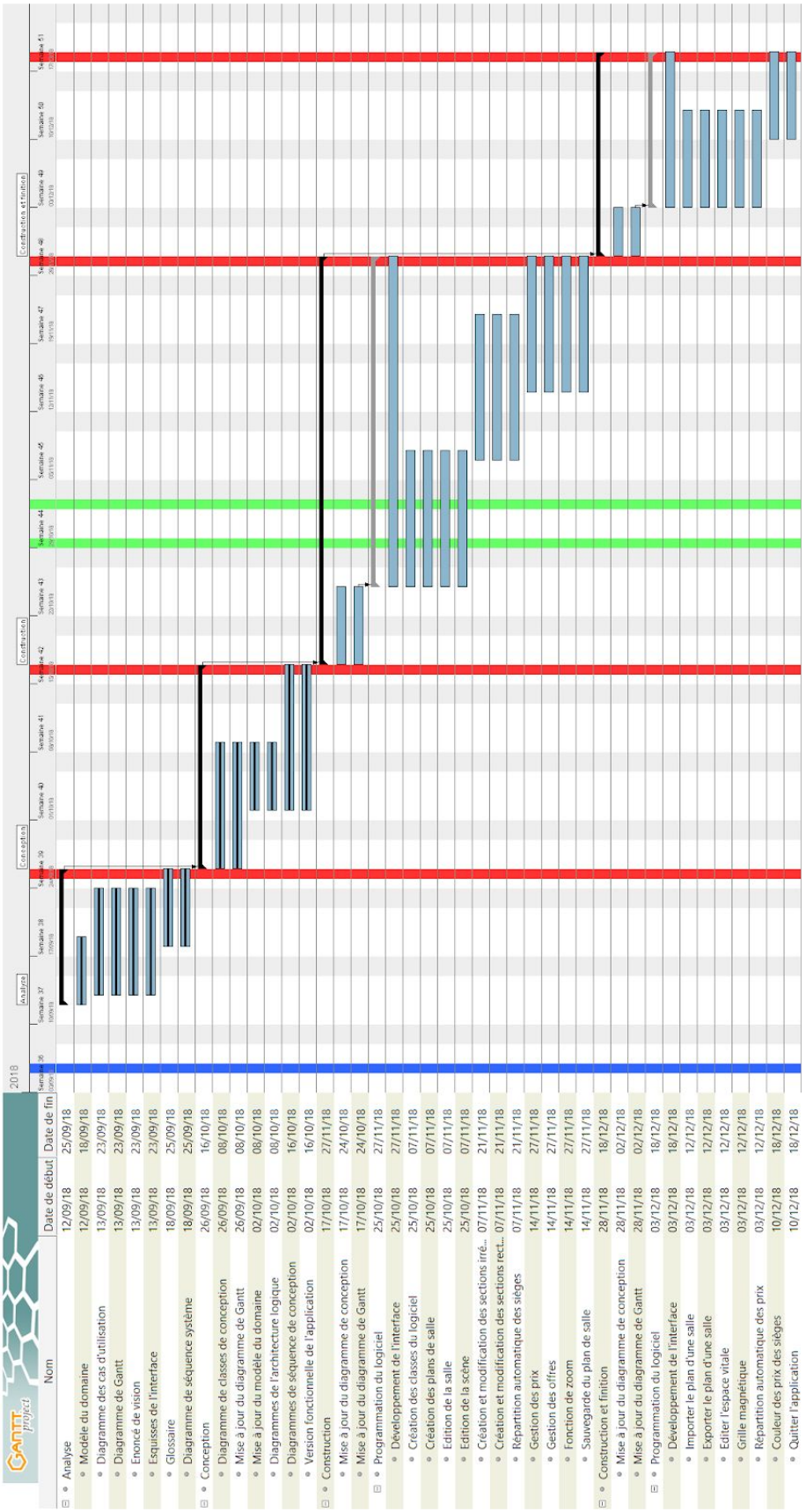
```
        Sinon et Si TypeObjet = "section" Alors
                prix = coeff*MaxPrix
                Si prix < MinPrix Alors
                        prix = MinPrix
                Fin Si
        Fin Sinon et Si

        Sinon Lancer une Exception
        Fin Sinon
        retourner prix
   Sinon et si TypePrix == Total        // On connaît le total des revenus que l'on souhaite
                Lire TypeObjet
                Si TypeObjet = "siège" Alors
                        nbreSieges(entier) = sommeDesSiegesDeLaSalle()
                        prix = coeff*(Total/nbresieges)         // Ici le coeff est calculé en
                                                                // fonction de distance
                                                                // siège-scène
                Fin Si

                Sinon et Si TypeObjet = "rangée" Alors
                        nbreRangee(entier) = sommeDesRangeesDeLaSalle()
                        prix = coeff*(Total/nbreRangee)         // Ici le coeff est calculé en
                                                                // fonction de distance
                                                                // rangée-scène
                Fin Sinon

                Sinon et Si TypeObjet = "section" Alors
                        nbreSection(entier) = sommeDesSectionsDeLaSalle()
                        prix = coeff*(Total/nbreSection)        // Ici le coeff est calculé en
                                                                // fonction de distance
                                                                // section-scène
                 Fin Sinon et Si

                Sinon Lancer une Exception
                Fin Sinon
                retourne prix

   Fin Sinon et si
}
```

3.6. Affichage du plan de salle

Diagramme



4. Plan de travail



5. Annexes

5.1. Énoncé de vision

Ce projet vise à répondre aux besoins de l'entreprise *AmzEvent* qui souhaite remplacer son ancien logiciel de création d'évènements, complexe et peu ergonomique. Ce logiciel doit permettre aux employés de la compagnie *AmzEvent*, de définir les plans d'une salle pour un événement donné, c'est-à-dire de gérer l'emplacement de la scène et des sièges, ainsi que le nombre et le prix de ces derniers.

Notre objectif est ainsi d'élaborer un logiciel, dans le temps imparti, répondant aux besoins de la société *AmzEvent*, et qui soit convivial, facile d'utilisation, intuitif et robuste.

Pour cela, à travers une interface agréable, l'utilisateur devra pouvoir créer, dimensionner et positionner une scène dans une salle. Il devra aussi être capable de dessiner facilement des sections de sièges de la forme qu'il souhaite à l'aide de la souris. Une fois la section créée, les sièges seront automatiquement répartis à l'intérieur de la section. L'utilisateur aura alors la possibilité d'éditer l'espace vital d'un spectateur et d'attribuer des prix ou des offres aux sections, rangées de siège ou à un siège en particulier. Des sections d'admission générale où les spectateurs sont debout peuvent également être dessinées. A chaque rotation, déplacement ou modification de section, tous les paramètres de cette dernière (répartition des sièges, prix...) sont mis à jour.

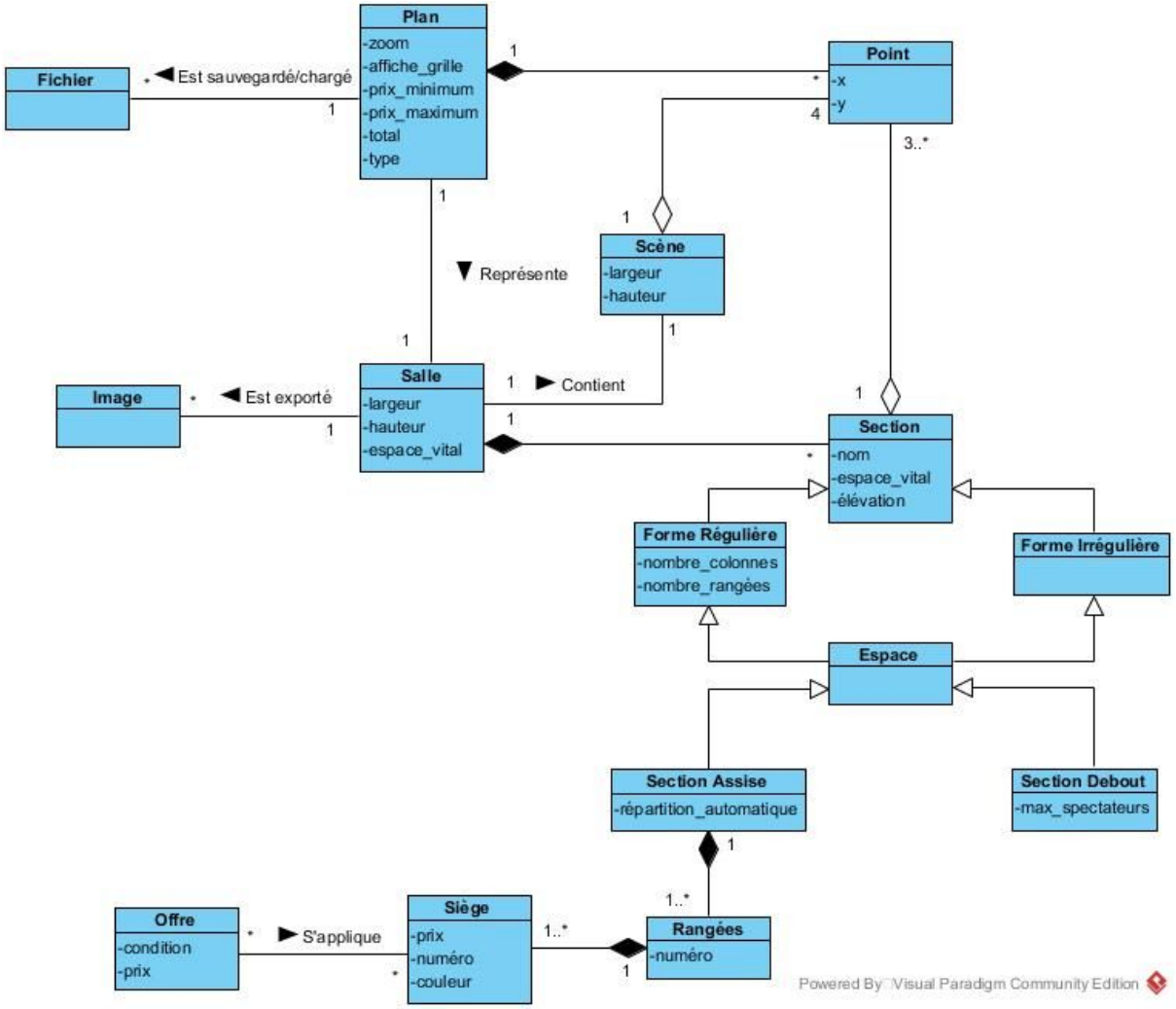
De plus, d'autres fonctionnalités secondaires seront intégrées à ce logiciel, comme la possibilité d'utiliser une grille magnétique pour faciliter le placement des sections, activer ou non la distribution automatique des sièges, définir le prix des sièges automatiquement en fonction de la distance par rapport à la scène, attribuer des couleurs aux différents tarifs…

Finalement, l'application permettra aux utilisateurs d'enregistrer le projet en cours, pour pouvoir y accéder de nouveau lors d'une prochaine session, sans que les modifications apportées ne soient perdues. Ils pourront aussi, s'ils le souhaitent, exporter le plan dans un format d'image courant.

Afin de réaliser ce projet, notre équipe de développement programmera le logiciel en Java avec l'IDE *IntelliJ*, et utilisera *Swing* pour concevoir l'interface utilisateur. Le projet s'articulera en différentes phases d'analyse, d'élaboration et de construction. Enfin, du début jusqu'à la fin du projet, prévue au milieu du mois de décembre, différents livrables seront présentés au client afin d'obtenir un retour de ce dernier et de lui montrer l'avancement du projet. Le document ci-présent est le premier livrable présentant la phase d'analyse du projet.

5.2. Modèle du domaine

5.2.1. Diagramme des classes conceptuelles



5.2.2. Texte explicatif des classes conceptuelles

Plan

Le **Plan** est une classe qui nous permet de décrire et de représenter l'espace de travail de l'utilisateur pour la création de la salle. Ce plan contient des informations spécifiques concernant le zoom, l'affichage de la grille magnétique, et les paramètres directement liés aux calculs des prix automatiques (prix minimum, prix maximum, total et type). Ce plan a une relation vers un ou des **Fichiers** car il peut être sauvegardé et chargé afin de pouvoir

récupérer son travail. De plus, le plan a une relation vers des **Points** car la totalité du plan contient un ensemble de points x et y. Cette liaison peut être apparentée aux pixels de l'écran, bien que dans ce cas précis elle nous permet simplement d'exprimer que les points appartiennent à un plan donné. Enfin, le plan nous permet également de représenter une **Salle.** Il existe une nuance entre le plan et la salle, car le plan est une classe permettant de stocker également des informations sur les préférences de l'utilisateur (comme indiqué plus haut, le zoom par exemple). La salle n'est qu'une représentation concrète du plan de la salle.

Salle

La **Salle** est la classe qui permet de définir concrètement le plan de la salle. Elle dispose d'une largeur, d'une hauteur et d'un espace vital par défaut. La salle est liée aux **Images** car elle peut être exportée vers un format d'image spécifique. La Salle contient également tout ce qui permet de faire un plan de salle. Ainsi nous retrouvons une liaison 1-1 avec la classe **Scène** et une liaison 1-* avec des **Sections**.

Scène

La classe **Scène** permet de conceptualiser la scène du spectacle. Il ne peut y en avoir qu'une pour une salle donnée. Elle est définie par une largeur et une hauteur ainsi que 4 points. La scène, étant de forme rectangulaire, a une relation 1-4 vers des **Points** puisqu'elle contient 4 points pour définir sa forme.

Section

Une **Section** est une classe conceptuelle définissant l'espace d'une section de siège ou d'admission générale (respectivement **Section assise** et **Section debout**). Une section a un nom, un espace vital (potentiellement hérité de la **Salle**) ainsi qu'une élévation. Une section contient également des **Points** (au minimum 3) qui permettent de définir sa forme (potentiellement rectangulaire ou polygonale).

Section assise

La **Section assise** hérite de la section et permet de représenter une section comportant des sièges. Cette section peut disposer de la fonctionnalité de répartition automatique des sièges. Une section assise contient des **Sièges** (au minimum 1).

Section debout

La **Section debout** hérite de la section et permet quant à elle de représenter une section d'admission générale. C'est-à-dire qu'il n'y a pas de sièges dans cette section. Les spectateurs seront debout. Afin de garantir que chaque spectateur puisse entrer dans cette section, un paramètre définissant le nombre maximum de spectateurs est présent (max_spectateur).

Siège

Un **Siège** est une classe conceptuelle permettant de définir un siège contenu dans une **Section assise**. Un siège est défini par son prix, sa rangée, son numéro (selon la rangée) et une couleur optionnelle. Un siège peut avoir des **Offres**.

Offre

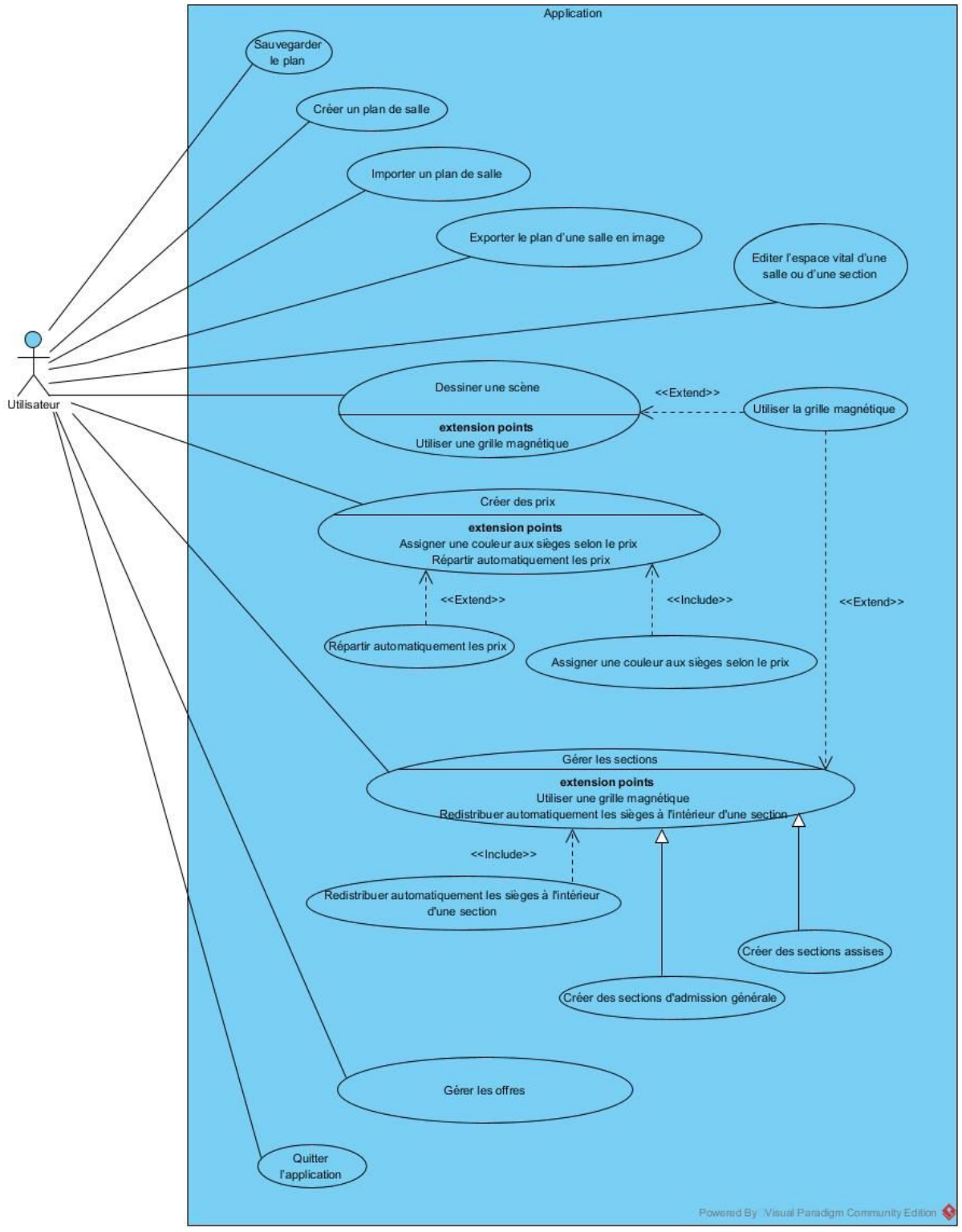
Une **Offre** est une classe conceptuelle définissant les réductions concernant un **Siège**. Une offre est définie par une condition (ex: être étudiant, avoir plus de 70 ans...) et un prix. Une offre s'applique sur un **Siège** spécifique ou un ensemble de sièges.

Spectateur

La classe conceptuelle **Spectateur** nous permet de représenter les actions d'un spectateur. Elle permet majoritairement de visualiser la plus-value de certaines actions sur notre design. Ainsi un spectateur peut être sujet à des **Offres**, il peut également réserver un **Siège** spécifique ou réserver son entrée dans une **Section debout** (d'admission générale). Bien que le logiciel ne permet pas la réservation de siège ni l'utilisation d'offre, la classe **Spectateur** permet d'orienter nos classes **Siège**, **Offre** et **Section** en prenant en compte ce paramètre.

5.3. Modèle des cas d'utilisation

5.3.1. Diagramme des cas d'utilisation



5.3.2. Texte des cas d'utilisation

B1- Créer un plan de salle

Cas d'utilisation:	Créer un plan de salle
Système:	VenueDesigner
Acteurs:	Utilisateur
Parties prenantes et intérêts:	Utilisateur : Il souhaite accéder à un espace de travail vierge dans l'application.
Préconditions:	N/A
Garanties en cas de succès:	Un nouveau fichier est créé sur le poste local de l'utilisateur.
Scénario principal:	1. L'utilisateur lance VenueDesigner sur son poste. 2. Présentation des options d'accueil. 3. L'utilisateur choisit de créer un nouveau plan de salle. 4. Sollicitation d'un nom de fichier. 5. L'utilisateur choisit un nom de fichier. 6. Sollicitation des dimensions de la salle. 7. L'utilisateur entre les dimensions de la salle. 8. Création du nouveau fichier de plan de salle. 9. Présentation des options de création de salles.
Scénarios alternatif:	Ligne 5 et 7: L'utilisateur choisit d'annuler la création d'un nouveau plan de salle. VenueDesigner doit revenir à la présentation des options d'accueil sans créer le nouveau fichier.

B2 - Dessiner une scène

Cas d'utilisation:	Dessiner une scène
Système:	VenueDesigner
Acteurs:	Utilisateur
Parties prenantes et intérêts:	Utilisateur : Il souhaite définir les dimensions et l'emplacement de la scène.
Préconditions:	- Avoir un fichier de plan de salle d'ouvert dans VenueDesigner.
Garanties en cas de succès:	La scène s'affiche bien au bon emplacement et avec les bonnes dimensions sur le plan de la salle.
Scénario	1. L'utilisateur dessine la scène 2. La scène apparaît dans la salle.

principal:	rectangulaire. 3. L'utilisateur choisit les dimensions de la scène. 4. Les nouvelles dimensions de la scène sont prises en compte. 5. L'utilisateur déplace la scène dans la salle. 6. La nouvelle position de la scène dans la salle est prise en compte. 7.L'utilisateur réoriente la scène. 8. La scène est correctement orientée.
Scénarios alternatif:	*a. À n'importe quel moment, l'utilisateur décide de supprimer une offre.

B3 - Redistribuer automatiquement les sièges.

Cas d'utilisation:	Redistribuer automatiquement les sièges.
Système:	VenueDesigner
Acteurs:	Utilisateur
Parties prenantes et intérêts:	Utilisateur: Il souhaite que les sièges soient automatiquement répartis dans une section, de sorte à ce que tout l'espace à l'intérieur de la section soit occupé de manière optimale.
Préconditions:	- Avoir un fichier de plan de salle d'ouvert dans VenueDesigner.
Garanties en cas de succès:	Les sièges sont placés de façon automatique dans la section.
Scénario principal:	1. L'utilisateur dessine une section. 2. Redistribution automatique des sièges dans la section.
Scénarios alternatif:	*a. À tout moment : l'utilisateur choisit d'annuler la redistribution automatique des sièges. *b. À tout moment : l'utilisateur choisit d'éditer l'espace vital de la salle ou d'une section, et la répartition des sièges se met à jour. *c. À tout moment : l'utilisateur modifie la section, et la répartition des sièges se met à jour.

B4 - Gérer les offres

Cas d'utilisation:	Gérer les offres
Système:	VenueDesigner
Acteurs:	Utilisateur
Parties prenantes et intérêts:	Utilisateur: Il souhaite assigner une ou des offres à un ou plusieurs emplacements. Il doit aussi pouvoir retirer une offre d'un emplacement.
Préconditions:	- Avoir un fichier de plan de salle d'ouvert dans VenueDesigner. - Pour assigner une offre à un emplacement: avoir au moins une section de

	créée.
Garanties en cas de succès:	L'utilisateur peut visualiser facilement les sièges attribués à des offres et en connaître le nombre.
Scénario principal:	1. L'utilisateur choisit de créer une offre. 2. Sollicitation d'un nom d'offre. 3. L'utilisateur choisit un nom d'offre. *L'utilisateur répète les étapes 1 à 3 jusqu'à ce que toutes les offres soient créées.* 4. L'utilisateur sélectionne un ou plusieurs emplacements. 5. L'utilisateur choisit une offre à assigner aux emplacements sélectionnés. 6. Assignation de l'offre aux emplacements sélectionnés. *L'utilisateur répète les étapes 4 à 6 jusqu'à ce que toutes les offres soient assignées à leurs emplacements.*
Scénarios alternatif:	*a. À n'importe quel moment, l'utilisateur décide de supprimer une offre.
	1. L'utilisateur sélectionne une offre. 2. L'utilisateur choisit de supprimer l'offre. 3. Si l'offre sélectionnée est assignée à au moins un siège, affichage d'un avertissement annonçant que l'offre sera automatiquement désassigner des sièges. Sinon étape 5. 4. L'utilisateur décide de continuer. 5. L'offre est désassignée de tous les sièges auxquels elle était assignée et est supprimée.
	*b. À n'importe quel moment, l'utilisateur décide de désassigner une offre d'un ou de plusieurs sièges.
	1. L'utilisateur sélectionne un ou plusieurs sièges 2. L'utilisateur choisit de désassigner l'offre. 3. L'offre est désassignée de tous les sièges sélectionnés.

B5 - Répartir automatiquement les prix.

Cas d'utilisation:	Répartir automatiquement les prix.
Système:	VenueDesigner
Acteurs:	Utilisateur
Parties	Utilisateur: Il souhaite que les prix soient automatiquement attribués aux

prenantes et intérêts:	emplacements en fonction de la distance avec la scène, soit selon un prix maximum et un prix minimum par billet, soit selon un montant de revenu total souhaité.	
Préconditions:	- Avoir un fichier de plan de salle d'ouvert dans VenueDesigner. - Avoir au moins une section de créée.	
Garanties en cas de succès:	Les emplacements se voient attribuer un prix en fonctions de la distance avec la scène en respectant les paramètres fournis par l'utilisateur.	
Scénario principal:	1. L'utilisateur choisit d'attribuer automatiquement les prix aux emplacements. 3. L'utilisateur choisit la répartition selon un prix maximum et minimum par billet. 5. L'utilisateur fournit les prix maximum et minimum.	2. Sollicitation de choisir entre une répartition selon un prix maximum et minimum par billet ou selon un montant de revenu total. 4. Sollicitation des prix maximum et minimum. 6. Répartition automatique des prix selon les paramètres fournis et la distance avec la scène.
Scénarios alternatif:	*a. À tout moment: l'utilisateur choisit d'annuler la répartition automatique des prix. Venue Designer doit revenir à l'état dans lequel il était avant la requête.	
	3a. L'utilisateur choisit la répartition selon un montant de revenu total. 5a. L'utilisateur fournit le montant de revenu total.	4a. Sollicitation du montant de revenu total.

B6 - Gérer les sections.[2]

Cas d'utilisation :	Gérer les sections.
Système :	VenueDesigner.
Acteurs :	Utilisateur
Parties prenantes et intérêts :	Utilisateur : Il souhaite créer ou modifier des sections assises ou des sections d'admission générale régulières ou irrégulières.
Préconditions :	Pour créer une section, il faut que l'utilisateur ait créé un plan de salle. Pour modifier une section, il faut que cette dernière ait été créée.
Garanties en cas de succès :	Une nouvelle section est créée ou une section a été modifiée.

[2] Nous sommes conscient que ce cas d'utilisation aurait pu être découpé en de plus petits, créer une section et modifier une section par exemple. Toutefois nous avons fait le choix de les regrouper pour simplifier notre diagramme des cas d'utilisation.

Scénario principal :	1. L'utilisateur choisit de créer une nouvelle section.	2. Sollicitation du nom de la section.
	3. L'utilisateur choisit un nom pour la section ou laisse celui par défaut.	4. Présentation des options de création de sections (admission générale ou de sièges).
	5. L'utilisateur choisit de créer une section d'admission générale.	6. Sollicitation du type de section (régulière ou irrégulière).
	7. L'utilisateur choisit de créer une section régulière.	8. Sollicitation des dimensions de la section.
	9. L'utilisateur rentre les dimensions de la section.	10. Création d'une nouvelle section d'admission générale régulière.
Scénario Alternatif 1 :	1. L'utilisateur choisit de créer une nouvelle section.	2. Sollicitation du nom de la section.
	3. L'utilisateur choisit un nom pour la section ou laisse celui par défaut.	4. Présentation des options de création de sections (admission générale ou de sièges).
	5. L'utilisateur choisit de créer une section d'admission générale.	6. Sollicitation du type de section (régulière ou irrégulière).
	7. L'utilisateur choisit de créer une section irrégulière.	8. Sollicitation de l'utilisateur pour créer cette section.
	9. L'utilisateur choisit des points sur le plan,, reliés d'une certaine manière entre eux et qui définiront une section	10. Création d'une nouvelle section d'admission générale irrégulière.

	irrégulière, à la manière d'un polygone irrégulier.	
Scénario Alternatif 2 :	1. L'utilisateur choisit de créer une nouvelle section. 3. L'utilisateur choisit un nom pour la section ou laisse celui par défaut. 5. L'utilisateur choisit de créer une section de sièges. 7. L'utilisateur choisit de créer une section régulière. 9. L'utilisateur choisit un point du plan, qui sera le coin avant gauche de la future section et étire la section pour la former.	2. Sollicitation du nom de la section. 4. Présentation des options de création de sections (admission générale ou de sièges). 6. Sollicitation du type de section (régulière ou irrégulière). 8. Sollicitation de l'utilisateur pour créer cette section. 10. Création d'une nouvelle section de sièges régulière.
Scénario alternatif 3 :	1. L'utilisateur choisit de créer une nouvelle section. 3. L'utilisateur choisit un nom pour la section ou laisse celui par défaut. 5. L'utilisateur choisit de créer une section de sièges. 7. L'utilisateur choisit de créer une section irrégulière.	2. Sollicitation du nom de la section. 4. Présentation des options de création de sections (admission générale ou de sièges). 6. Sollicitation du type de section (régulière ou irrégulière). 8. Sollicitation de l'utilisateur pour créer cette section . 10. Création d'une nouvelle

	9. L'utilisateur choisit des points sur le plan,, reliés d'une certaine manière entre eux et qui définiront une section irrégulière, à la manière d'un polygone irrégulier.	section de sièges irrégulière.
Scénario alternatif 4 :	1. L'utilisateur choisit de modifier une section existante. 3. L'utilisateur clique sur un point de la section et peut l'étirer si c'est une section régulière, ou déplacer le coin de la section comme il le souhaite s'il s'agit d'une section irrégulière. 5. L'utilisateur a fini son action	2. Sollicitation de la section. 4. L'application vérifie qu'il n'y a pas de problème avec la modification sinon, renvoie un message d'erreur. 6. La modification de la section est effectuée.
Scénario alternatif 5 :	L'utilisateur choisit d'annuler la création d'une section, VenueDesigner doit revenir à la présentation des options de créations de sections sans en créer.	

B7 - Créer des prix.

Cas d'utilisation :	Créer des prix.
Système :	VenueDesigner.
Acteurs :	Utilisateur.
Parties prenantes et intérêts :	Utilisateur : Il souhaite créer des prix pour des ensembles de sièges, des sièges, des sections ou des rangées.
Préconditions :	Il faut que l'utilisateur ait créé un plan de salle. Il faut que l'utilisateur ait créé une section.
Garanties en cas de succès :	Un prix sera assigné à un siège, un ensemble de sièges, une

	section ou une rangée.
Scénario principal :	1. L'utilisateur sélectionne l'objet pour lequel il souhaite assigner un certain prix. 2. Sollicitation de l'objet sélectionné et présentation des options de création de prix. 3. L'utilisateur rentre un prix pour l'objet souhaité manuellement ou automatiquement. 4. Validation du prix assigné à l'objet et attribution de la couleur.
Scénario Alternatif :	Ligne 3 et 4 : l'utilisateur choisit d'annuler la création d'un prix pour l'objet souhaité. VenueDesigner doit revenir à la présentation des options de créations de prix sans en assigner.

B8 - Importer un plan de salle.

Cas d'utilisation :	Importer un plan de salle.
Acteur(s) :	Utilisateur
Type :	Primaire
Description :	Une fois VenueDesigner lancé, l'utilisateur choisit d'importer un plan de salle. Il sélectionne le fichier à importer à partir de son disque local. VenueDesigner retrouve le plan dans l'état dans lequel il a été sauvegardé et affiche les options de création de salle.

B9 - Assigner une couleur aux sièges selon le prix.

Cas d'utilisation :	Assigner une couleur aux sièges selon le prix.
Acteur(s) :	Utilisateur

Type :	Primaire
Description :	Une fois que des prix sont attribués aux sièges, l'utilisateur peut choisir d'assigner une couleur au siège selon le prix. Il peut choisir une couleur parmi une palette et y attribuer un intervalle de prix qui y correspondra. Il peut créer autant d'intervalles qu'il veut tant que ceux-ci ne se chevauchent pas et sont inclus dans les valeurs limites des prix.

B10 - Sauvegarder le plan

Cas d'utilisation :	Sauvegarder le plan
Acteur(s) :	Utilisateur
Type :	Primaire
Description :	Dès le moment où l'utilisateur modifie le plan, celui-ci peut choisir de sauvegarder ses modifications dans le fichier du plan. Toutes les modifications sont enregistrées par VenueDesigner.

B11 - Exporter le plan d'une salle en image.

Cas d'utilisation :	Exporter le plan d'une salle en image.
Acteur(s) :	Utilisateur
Type :	Primaire
Description :	Après avoir créé le plan de la salle et réalisé toutes les modifications qu'il souhaitait dans cette dernière, l'utilisateur choisit d'exporter le plan en une image. Un aperçu du résultat s'affiche. Il choisit ensuite sous quel format d'image le plan doit être exporter, puis l'emplacement du fichier. Le fichier est ensuite sauvegardé selon les directives de l'utilisateur.

B12 - Editer l'espace vitale d'une salle ou d'une section.

Cas d'utilisation :	Editer l'espace vital d'une salle ou d'une section.
Acteur(s) :	Utilisateur
Type :	Primaire
Description :	Une fois le pan de la salle créé, l'utilisateur peut définir un espace vital pour toute la salle. Il modifie alors la valeur par défaut de l'espace vital dans les propriétés de la salle. Si l'utilisateur avait déjà changé l'espace vital d'une section, celui-ci n'est pas modifié. Sinon, après avoir dessiné une section, l'utilisateur peut sélectionner cette section, et modifier l'espace vital dans les propriétés de la section.

B13 - Quitter l'application

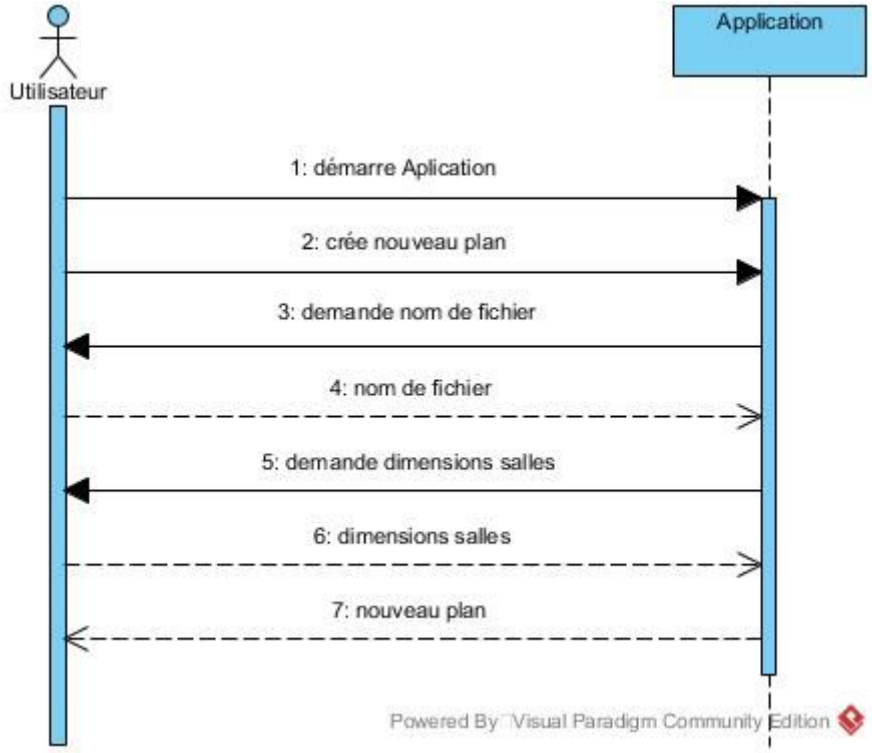
Cas d'utilisation :	Quitter l'application
Acteur(s) :	Utilisateur
Type :	Primaire
Description :	Lorsque l'utilisateur veut quitter l'application, une fenêtre apparaît pour lui demander s'il souhaite enregistrer son travail avant de quitter. Il choisit ainsi de sauvegarder les dernières modifications apportées ou non.

B14 - Utiliser la grille magnétique

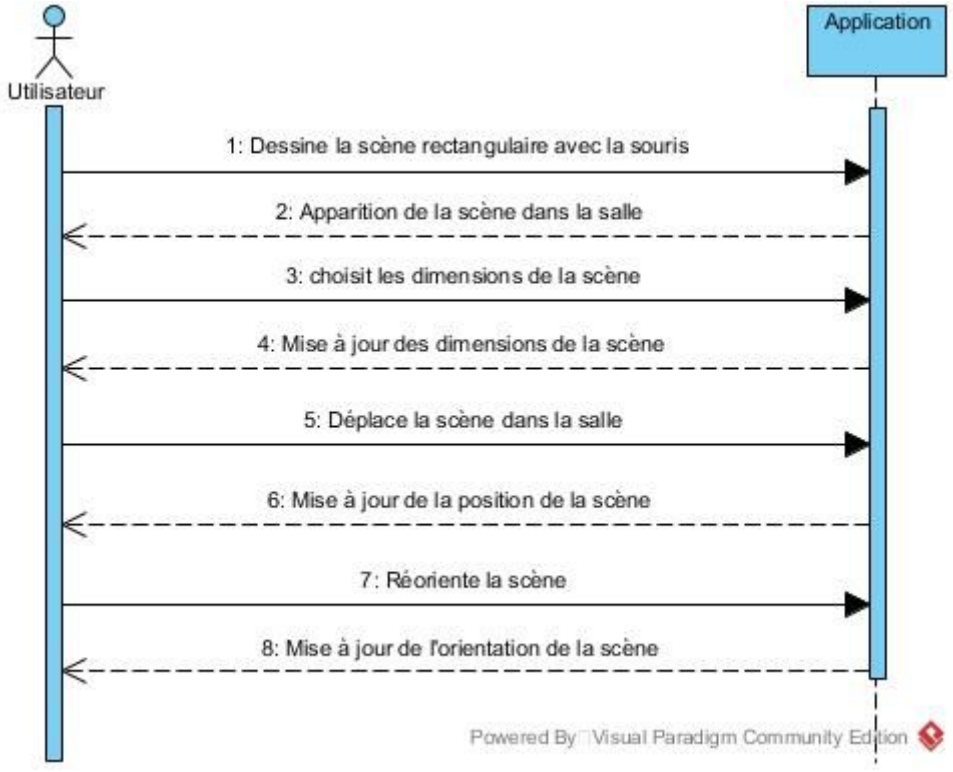
Cas d'utilisation :	Utiliser la grille magnétique
Acteur(s) :	Utilisateur
Type :	Primaire
Description :	L'utilisateur peut dessiner une scène rectangulaire, créer ou modifier une section à l'aide d'une grille magnétique qui lui permet de distinguer des points particuliers sur le plan de salle, et lui facilite ainsi sa vision des repères et des distances avec les objets qu'il manipule. Cette grille peut être activée ou désactivée à volonté.

5.3.3. Diagrammes de séquence système

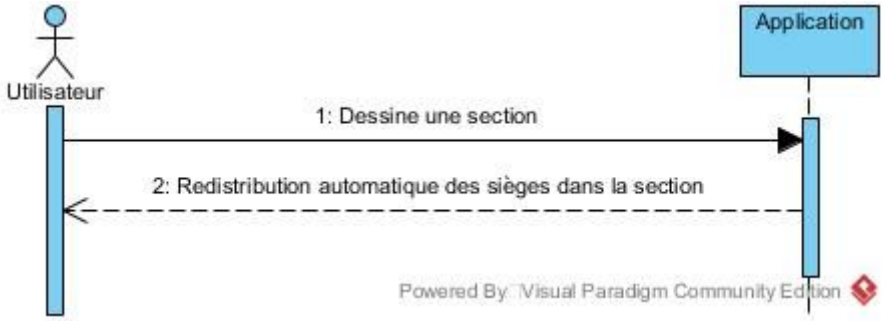
C1 - Créer un plan de salle



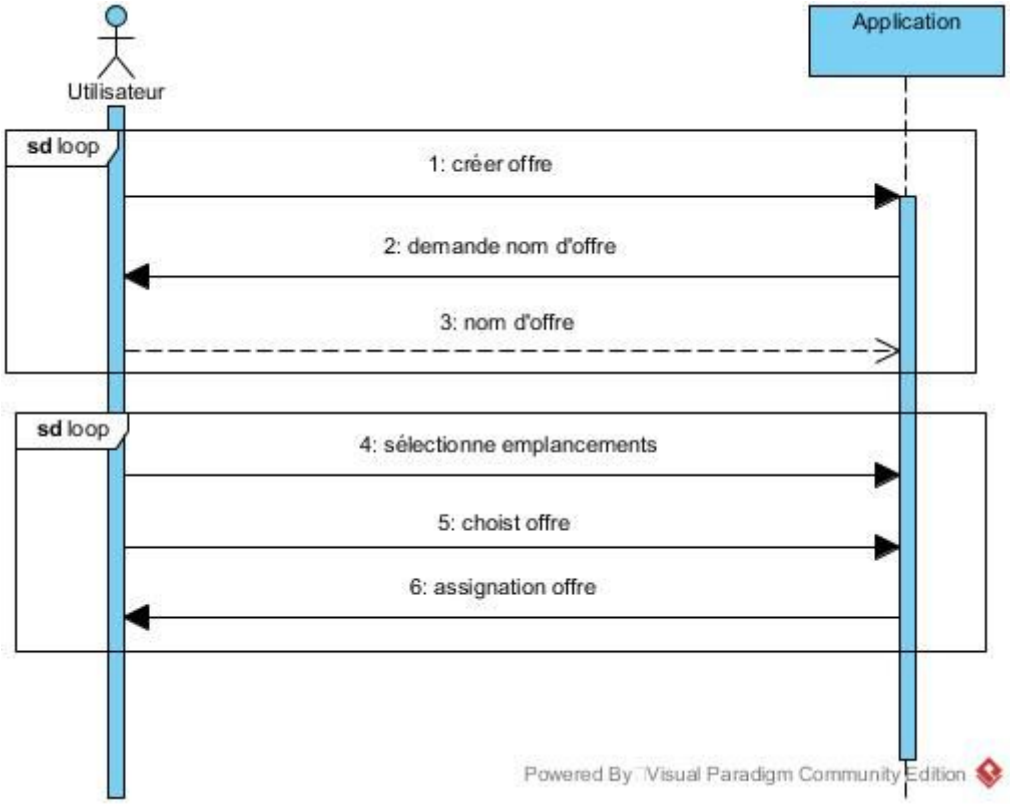
C2 - Dessiner une scène



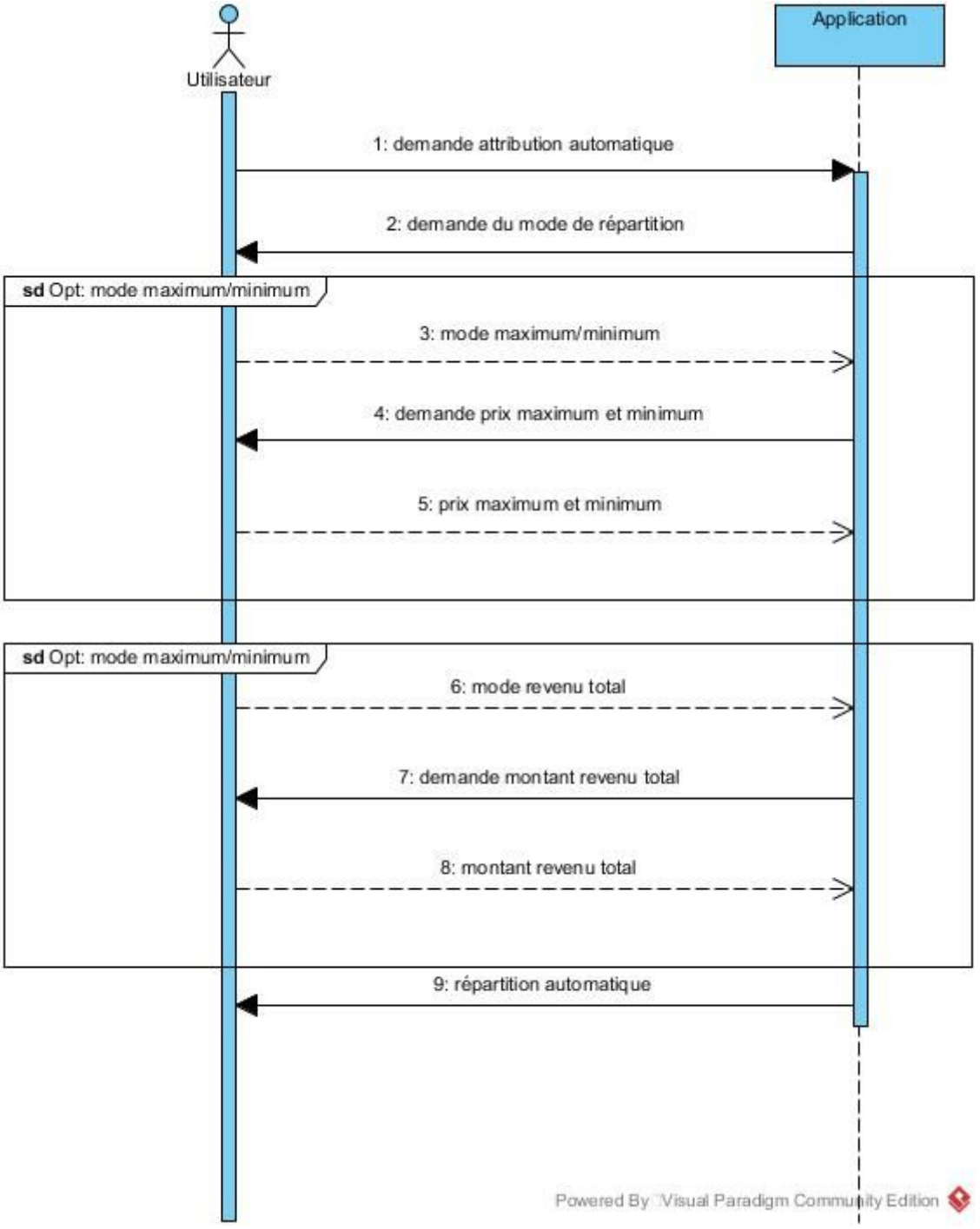
C3 - Redistribuer automatiquement les sièges



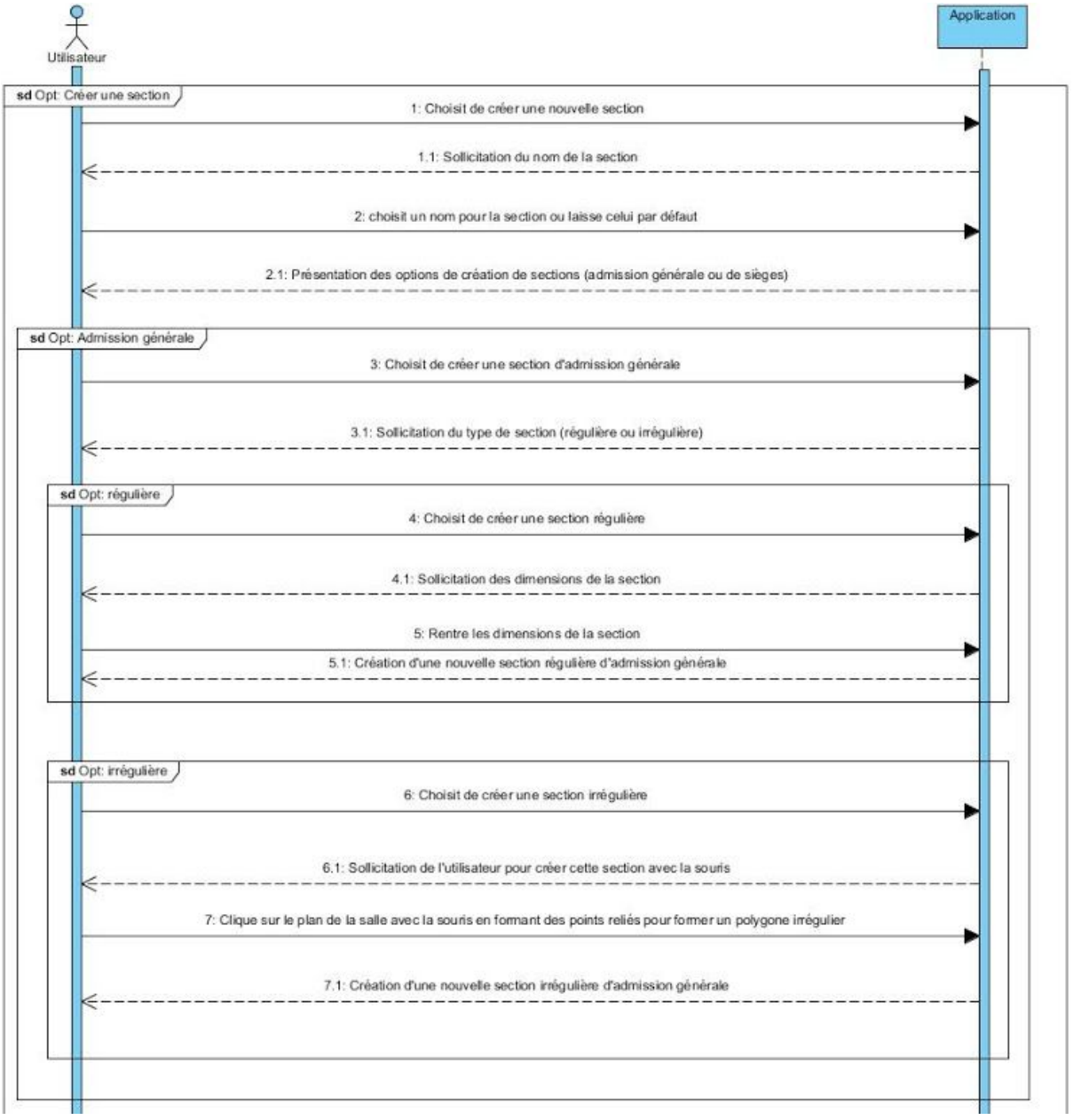
C4 - Gérer les offres



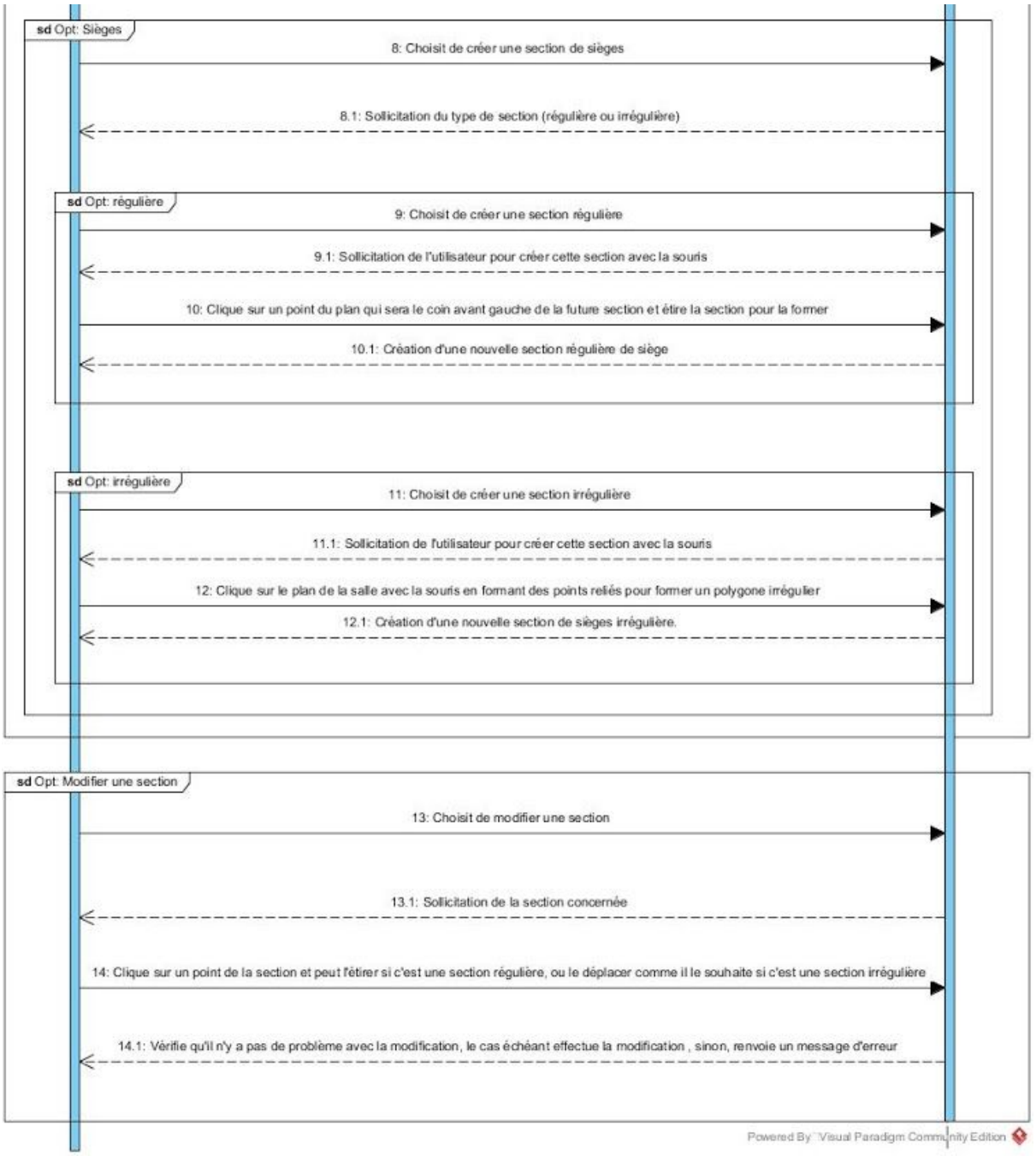
C5 - Répartir automatiquement les prix



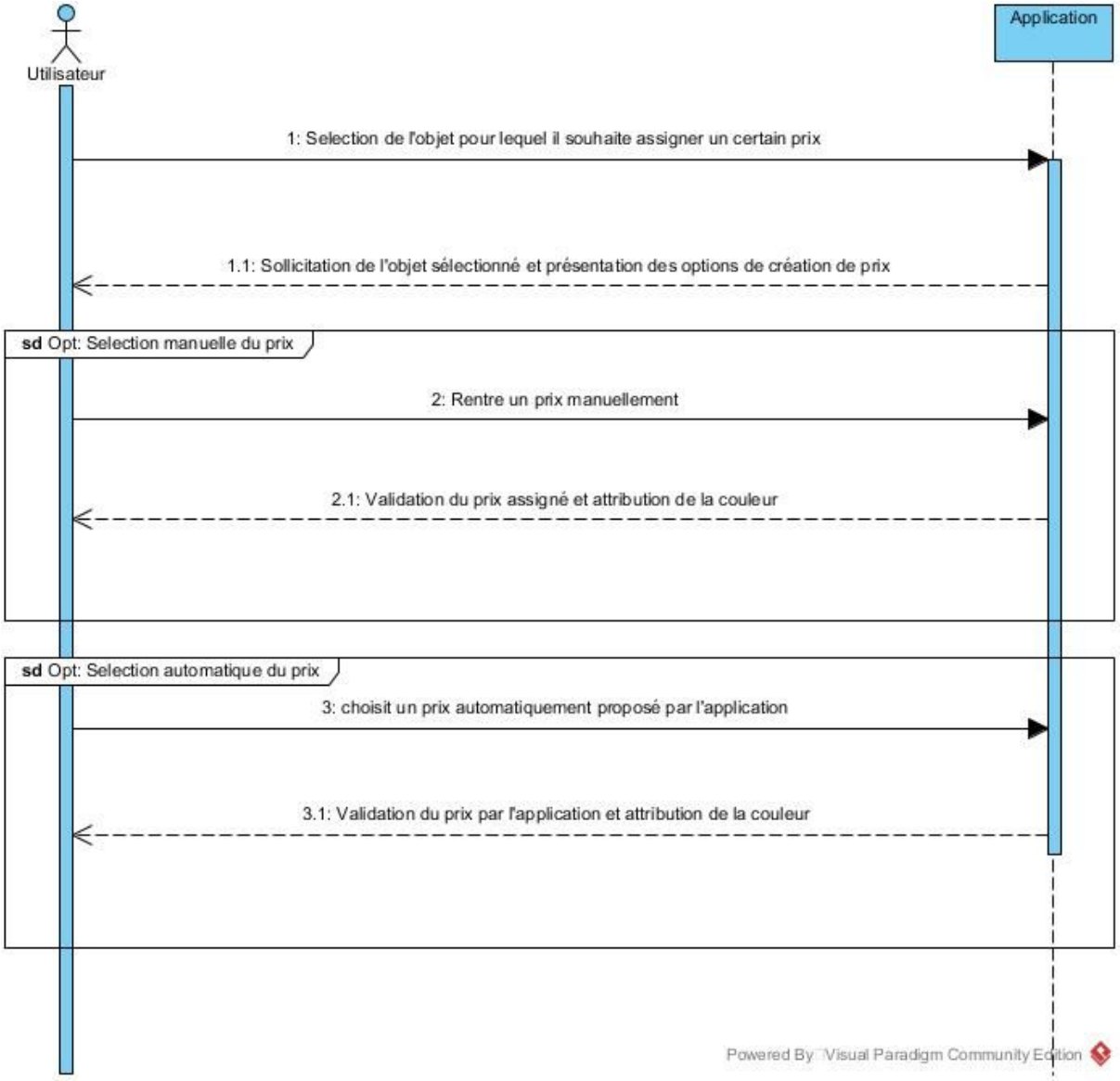
C6 - Gérer les sections.



C6 - Gérer les sections (suite)



C7 - Créer des prix


Utilisateur
Application
1: Selection de l'objet pour lequel il souhaite assigner un certain prix
1.1: Sollicitation de l'objet sélectionné et présentation des options de création de prix
sd Opt: Selection manuelle du prix
2: Rentre un prix manuellement
2.1: Validation du prix assigné et attribution de la couleur
sd Opt: Selection automatique du prix
3: choisit un prix automatiquement proposé par l'application
3.1: Validation du prix par l'application et attribution de la couleur
Powered By Visual Paradigm Community Edition

5.4. Glossaire

VenueDesigner	Nom de l'application.
Section debout	Section d'admission générale ne comportant aucun siège mais uniquement des personnes debout.
Objet	Ensemble de sièges ou section.
IDE	Environnement de développement.
Robustesse	Capacité d'une application à avoir des performances stables.
Offre	Remise en pourcentage ou en comptant sur un siège en fonction de certains critères donnés (âge du spectateur, jour de la semaine...).
Admission générale	Places où les spectateurs sont debout.