GLO-2004 : Génie logiciel orienté objet

Livrable 2 - Conception

Projet InnoEvent

Rémi Gastaldi - IDUL : REGAS3 - NI : 111 244 692
Léo Hubert - IDUL : LEHUB2 - NI : 111 244 584
Maud Marel - IDUL : MAMAR758 - NI : 111 244 577
Khaled Nasri - IDUL : KHNAS1 - NI : 111 088 348

Sommaire

1. Diagramme de classe de conception

1.1. UI

1.1.1. View

Méthodes :

- openView : Permet d'ouvrir une nouvelle fenêtre
- closeView : Permet de fermer une fenêtre
- openViewWithAnimation : Permet d'ouvrir une nouvelle en effectuant une animation
- start : Cette méthode est appelé automatiquement quand JavaFx est prêt et lancé.
- main : Cette méthode est le point d'entré de notre projet.

1.1.2. MainView

1.1.2.1. MainViewController

Contrôleur graphique de la classe MainView.

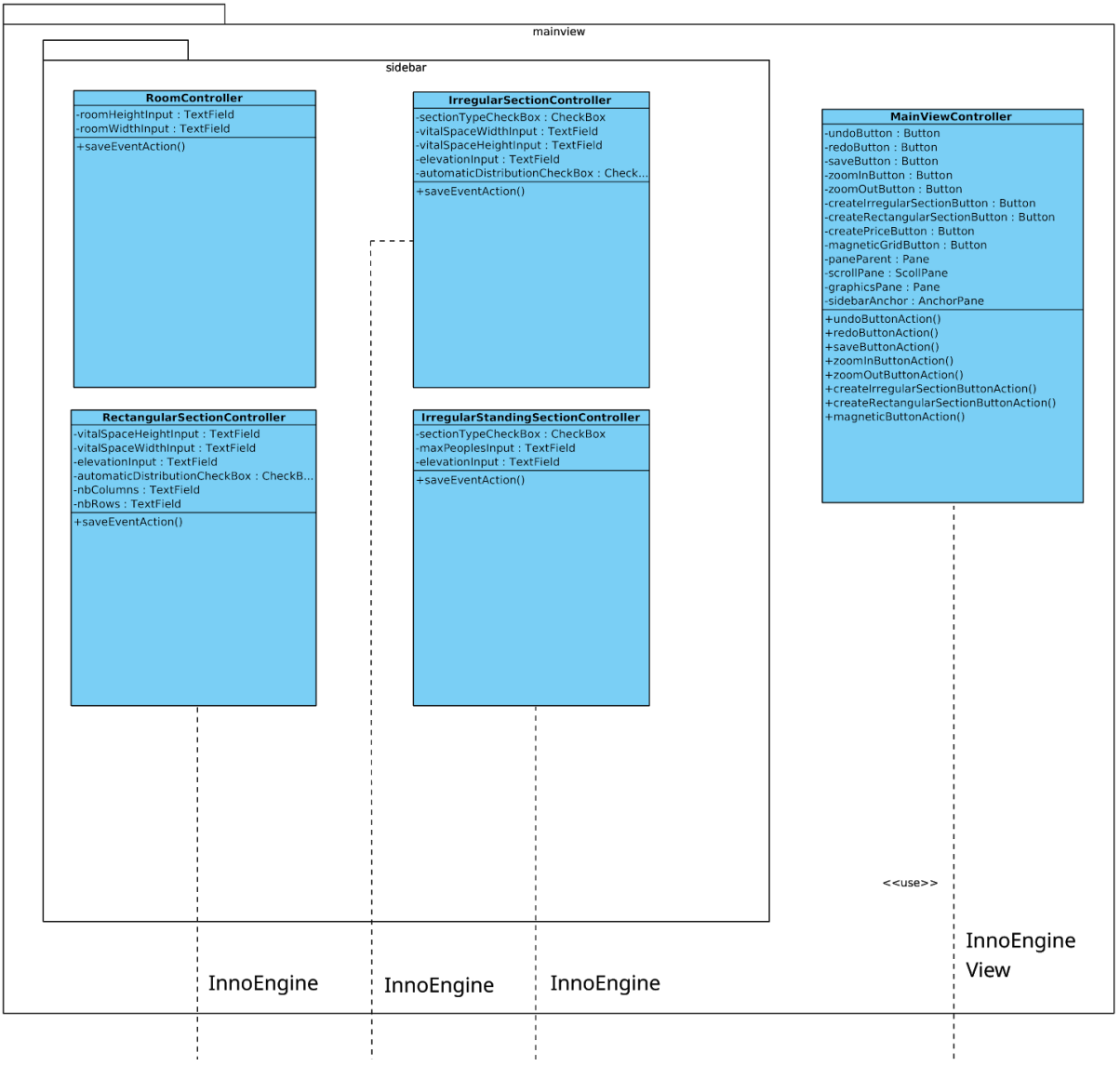
Attributs :

- undoButton est un Bouton.
- redoButton est un Bouton.
- saveButton est un Bouton.
- zoomInButton est un Bouton.
- zoomOutButton est un Bouton.
- createIrregularSectionButton est un Bouton.
- createRectangularSectionButton est un Bouton.
- createPriceButton est un Bouton.
- magneticGridButtonest un Bouton.
- PaneParent est une Pane.
- scrollPane est une ScrollPane.
- graphicsPane est une Pane.
- sidebarAnchor est une AnchorPane.

Méthodes :

- undoButtonAction : Permet d'annuler l'action précédente.
- redoButtonAction : Permet de refaire la dernière action annulée.
- saveButtonAction : Permet de sauvegarder en passant par le **Core**.
- zoomInButtonAction : Permet de zoomer en avant.
- zoomOutButtonAction : Permet de Zoomer en arrière.
- createIrregularSectionButtonAction : Permet de creer une section de forme irreguliere.
- createRectangularSectionButtonAction : Permet de créer une section de forme rectangulaire.
- magneticButtonAction : Permet d'activer la grille magnétique.

1.1.2.2. Sidebar



1.1.2.2.1. RoomController

Contrôleur graphique de la classe Room.

Attributs :

- roomHeightInput est un champ de texte.
- roomWidthInput est un champ de texte.

Méthodes :

- saveEventAction change la taille de la room.

1.1.2.2.2. IrregularSectionController

Contrôleur graphique de la création d'une section irrégulière.

Attributs :

- sectionTypeCheckBox est une CheckBox.
- vitalSpaceWidthInput est un champ de texte.

- vitalSpaceHeightInput est un champ de texte.
- elevationInput est un champ de texte.
- automaticDistributionCheckBox est une CheckBox.

Méthodes :

- saveEventAction change les valeurs d'une section irrégulière.

1.1.2.2.3. RectangularSectionController

Contrôleur graphique de la création d'une section rectangulaire.

Attributs :

- vitalSpaceHeightInput est un champ de texte.
- vitalSpaceWidthInput est un champ de texte.
- elevationInput est un champ de texte.
- automaticDistributionCheckBox est une CheckBox.
- nbColumns est un champ de texte.
- nbRows est un champ de texte.

Méthodes :

- saveEventAction change les valeurs d'une section régulière.

1.1.2.2.4. IrregularStandingSectionController

Contrôleur graphique de la création d'une section irrégulière debout.

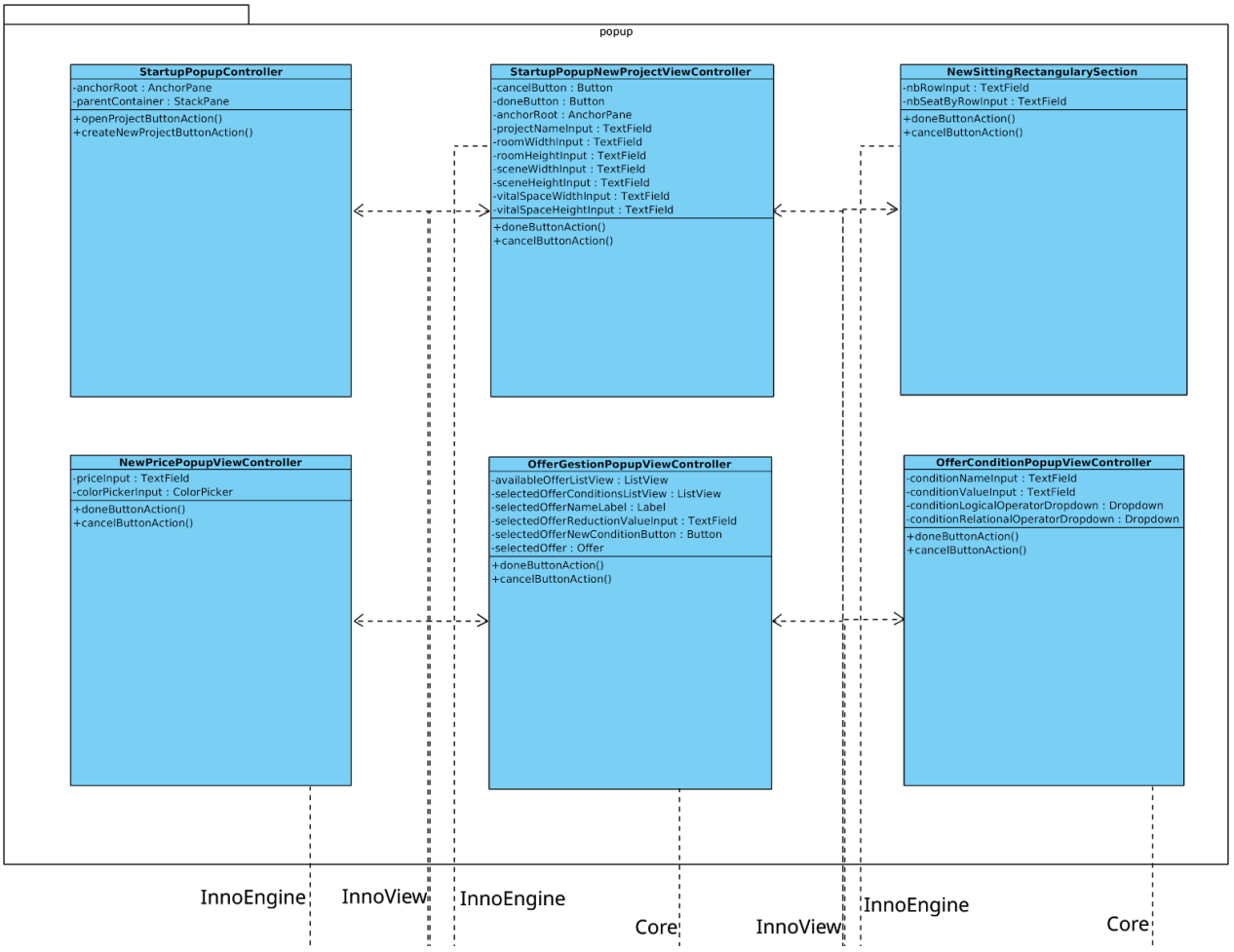
Attributs :

- sectionTypeCheckBox est une CheckBox.
- maxPeopleInput est un champ de texte.
- elevationInput est un champ de texte.

Méthodes :

- saveEventAction change les valeurs d'une section irrégulière debout.

1.1.3. Pop-up



1.1.3.1. StartupPopupController

Contrôleur graphique de la pop-up de démarrage.

Attributs :

- anchorRoot est une AnchorPane.
- parentContainer est une StackPane.

Méthodes :

- openProjectButtonAction permet d'ouvrir un projet en passant par le **Core**.
- createNewProjectButtonAction permet de créer un projet en passant par le **Core**.

1.1.3.2. StartupPopupNewProjectViewController

Contrôleur graphique de la pop-up de création d'un nouveau projet.

Attributs :

- cancelButton est un Bouton.
- doneButton est un Bouton.
- anchorRoot est une AnchorPane.
- projectNameInput est un champs de texte.

- roomWidthInput est un champs de texte.
- roomHeightInput est un champs de texte.
- sceneWidthInput est un champs de texte.
- sceneHeightnIput est un champs de texte.
- vitalSpaceWidthinput est un champs de texte.
- vitalSpaceHeightInput est un champs de texte.

Méthodes:

- doneButtonAction permet de confirmer les informations.
- cancelButtonAction permet d'annuler la création.

1.1.3.3. NewPricePopupViewController

Contrôleur graphique de la création d'un nouveau prix.

Attributs :

- priceInput est un champs de texte.
- colorPickerInput est un champs de texte.

Méthodes :

- doneButtonAction permet de confirmer les informations.
- cancelButtonAction permet d'annuler la création.

1.1.3.4. OfferGestionPopupViewController

Contrôleur graphique de la fenêtre pop-up de gestion d'offres.

Attributs :

- availableOfferListView est un afficheur en liste.
- selectedOfferConditionsListView est un afficheur en liste.
- selectedOfferNameLabel est un label.
- selectedOfferReductionValueInput est un champs de texte.
- selectedOfferNewConditionButton est un bouton.
- selectedOffer affiche l'offre selectionnee.

Méthodes :

- doneButtonAction permet de confirmer les informations.
- cancelButtonAction permet d'annuler la création.

1.1.3.5. NewSittingRectangularySection

Attributs :

- nbRowInput est un champ de texte.
- nbSeatByRowInput est un champ de texte.

Méthodes :

- doneButtonAction permet de confirmer les informations.
- cancelButtonAction permet d'annuler la création.

1.1.3.6. OfferConditionPopupViewController

Contrôleur graphique de la pop-up de création d'une nouvelle condition.

Attributs:

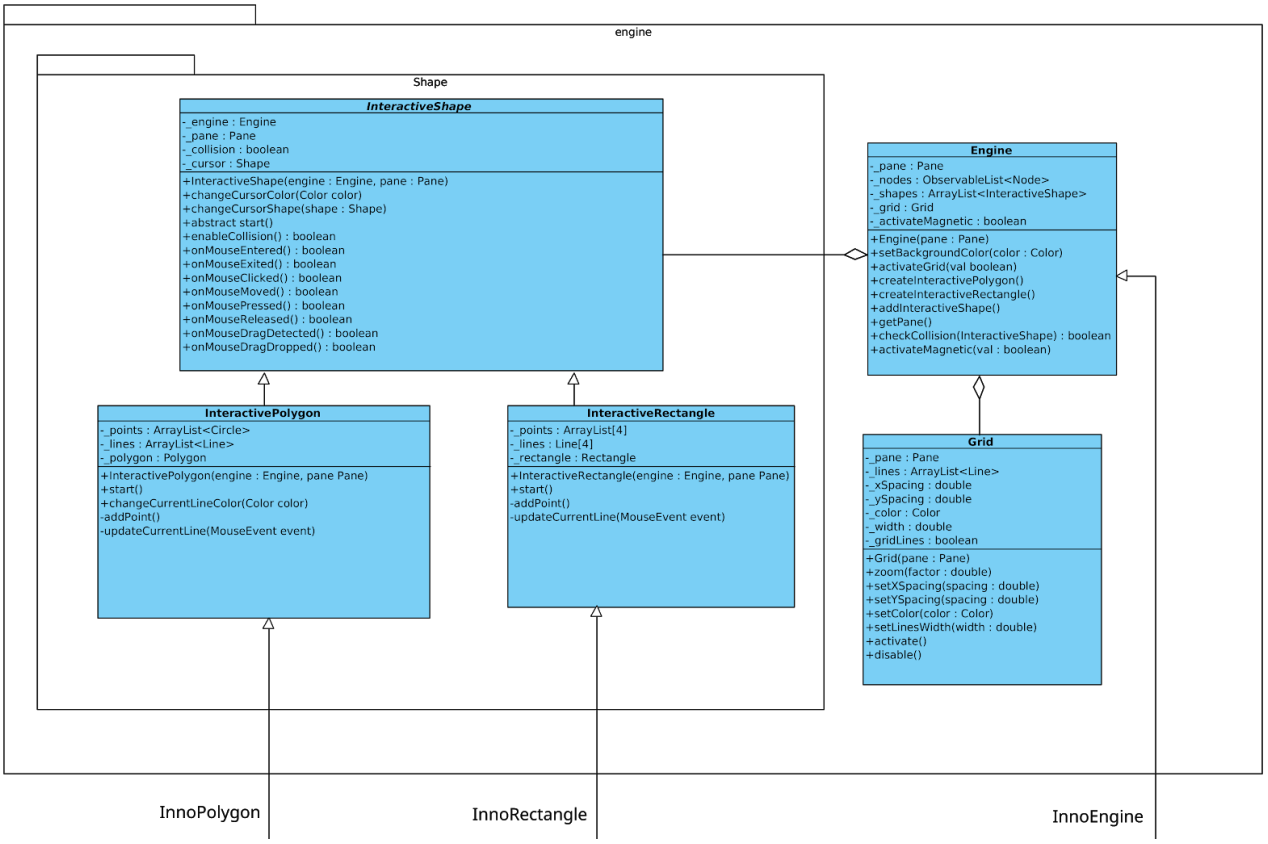
- conditionNameInput est un champs de texte.
- conditionValueInput est un champs de texte.
- conditionLogicalOperatorDropdown est un DropDown.
- conditionRelationalOperatorDropdown est un DropDown.

Méthodes:

- doneButtonAction : Permet de confirmer les informations.
- cancelButtonAction : Permet d'annuler la création.

1.1.4. Engine

Le package **Engine** est un ensemble de Classes utilitaire permettant de créer interactivement des formes avec l'utilisateur ainsi qu'une grille magnétique au choix, ce package est pensé pour n'avoir aucun lien avec notre projet et pouvant donc être réutilisable dans un autre.



1.1.4.1. Engine

La classe **Engine** est la classe principale de son package, elle permet de facilement créer des formes et de gérer toute la logique interne de javaFX en l'encapsulant

Attributs :

- _pane

- _node
- _shapes
- _grid
- _activateMagnetic

Méthodes :

- Engine(**Pane**): Constructeur de l'objet, prends une **Pane** en paramètre qui est le node sur lequel l'engine va interagir.
- setBackgroundColor(**Color**): Permet de changer la couleur en arrière du pane.
- activateGrid(boolean): Permet d'activer ou non la grille visuelle.
- createInteractivePolygon(): Permet d'instancier un **InteractivePolygon** et démarrer l'interaction utilisateur pour qu'il puisse set les points graphiquement.
- createInteractiveRectangle(): Permet d'instancier un **InteractiveRectangle** et démarrer l'interaction utilisateur pour qu'il puisse set les points.
- addInteractiveShape(**InteractiveShape**): Permet d'ajouter une **InteractiveShape** déjà instancié au moteur de l'engine.
- getPane(): Retourne la **Pane** courante d l'engine.
- checkCollision(**InteractiveShape**): Permet de vérifier si une collision a lieu avec un autre **InteractiveShape**, retourne true en cas de collision, false dans le cas contraire.
- activateMagnetic(boolean): Permet d'activer ou non la fonction magnétique de la grille.

1.1.4.2. Grid

Grid est une classe de gestion de la grille visuelle et magnétique, elle permet de choisir l'espacement, la couleur ainsi que l'accrochage magnétique.

Attributs:

- _pane
- _lines
- _xSpacing
- _ySpacing
- _color
- _width
- _gridLines

Méthodes:

- Grid(**Pane**): Construteur, prends en paramètre le Pane avec lequel interagir.
- zoom(double): Permet de zoomer sur la grille pour mettre à l'échelle.
- setXSpacing(double): Permet de régler l'espacement entre les lignes sur l'axe des X
- setYspacing(double): Permet de regler l'espacement entre les lignes sur l'axe des Y
- setColor(**Color**): Permet de changer la couleur des lignes de la grille.
- setLinesWidth(double): Pernet de changer changer l'épaisseur des lignes.
- activate(): Pernet d'activer la grille.
- disable(): Permet de desactiver la grille.

1.1.4.3. Shape

Package comprenant toutes les classes de forme interactive du package **Engine**, formes, polygon, rectangle etc.

1.1.4.3.1. InteractiveShape

Attributs:

- _engine
- _pane
- _collision
- _cursor

Méthodes:

- interactiveShape(**Engine**, **Pane**): Construteur, prends en paramètre l'**Engine** et le Pane avec lequel interagir.
- changeCursorColor(**Color**): Prends une couleur en paramètre pour changer la couleur du curseur.
- changeCursorShape(**Shape**): Permet de changer la forme du curseur.
- start
- enableCollision
- onMouseEntered(**MouseEvent**) : Permet de traiter l'action de l'entrée du curseur dans la fenêtre.
- onMouseExited(**MouseEvent**) : Permet de traiter l'action de la sortie du curseur de la fenêtre.
- onMouseclicked(**MouseEvent**) : Permet de traiter l'action d'un clic de souris.
- onMouseMoved(**MouseEvent**) : Permet de traiter l'action d'un mouvement de souris.
- onMousePressed(**MouseEvent**) : permet de traiter le maintien d'un clic de souris.
- onMouseReleased(**MouseEvent**) : Permet de traiter l'action de libérer un click de souris.
- onMouseDragDetected(**MouseEvent**) : permet de traiter l'action de détection d'un glissement du curseur.
- onMouseDragDropped(**MouseEvent**): Permet de traiter l'action de libérer le glissement de souris.

1.1.4.3.2. InteractivePolygon

Attributs:

- _points : Les points du polygone.
- _lines : Les lignes du polygone.
- _polygon

Méthodes:

- InteractivePolygon(**Engine**, **Pane**): Constructeur de l'objet prenant l'engine ainsi que le **Pane** avec lequel interagir.
- start(): Permet de commencer la création d'un polygone interactif.
- changeCurrentLineColor(**Color**): Permet de changer la couleur de la ligne actuelle.
- addPoint(**MouseEvent**): Permet d'ajouter un point à la position du clic de la souris.
- updateCurrentLine(**MouseEvent**): Mets à jour la position de la ligne courante en fonction de la position de la souris.

1.1.4.3.3. InteractiveRectangle

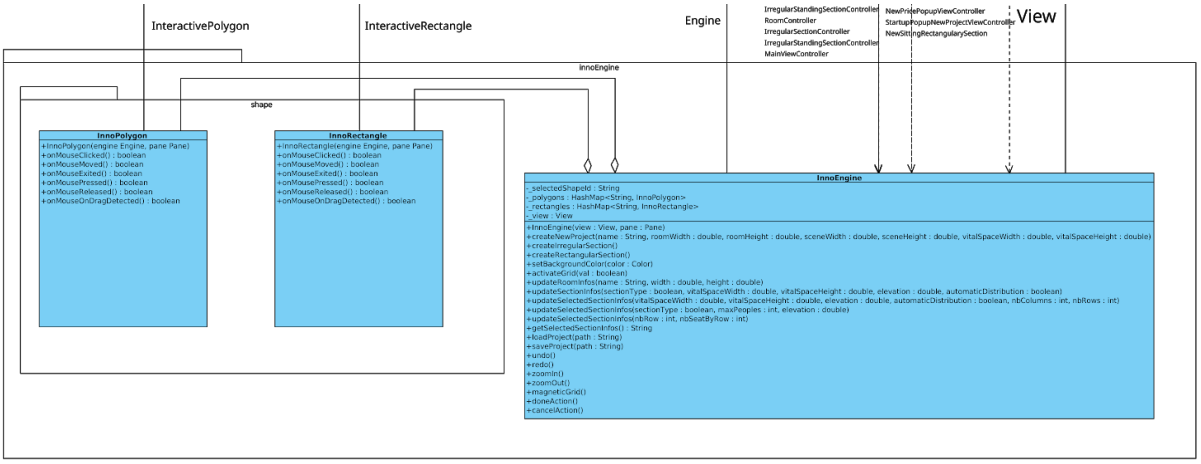
Attributs:

- _points : Les points du rectangle.
- _lines : Les lignes du rectangle.
- _rectangle

Méthodes:

- interactiveRectangle(**Engine**, **Pane**): Constructeur de l'objet prenant l'engine ainsi que le **Pane** avec lequel interagir.
- start(): Permet de commencer la création d'un rectangle interactif.
- addPoint(**MouseEvent**): Permet d'ajouter un point à la position du clic de la souris
- updateCurrentLine:(**MouseEvent**): Mets à jour la position de la ligne courante en fonction de la position de la souris.

1.1.5. InnoEngine



1.1.5.1. InnoEngine

InnoEngine est une classe héritant de la classe Engine pour pouvoir étendre ses fonctionnalités pour pouvoir y rajouter des comportement propre à notre projet.

Attributs:

- _selectedShapeId: id de la forme sélectionnée.
- _polygons
- _rectangles

Méthodes:

- InnoEngine(**View**, **Pane**): Constructeur de InnoEngine, prends en paramètre les objets nécessaire à la construction de la classe **Engine** dont il hérite.
- createNewProject(String): Permet de lier la création d'un nouveau projet du contrôleur graphique au **Core.**
- createIrregularSection: Permet d'instancier un **InteractivePolygon** et démarrer l'interaction utilisateur pour qu'il puisse set les points graphiquement.
- createRectangularSection() :Permet de démarrer la création interactive d'une section de type rectangulaire.

- setBackgroundColor(**Color**): Permet de changer la couleur en arrière du pane
- activateGrid(boolean): Pernet d'activer ou non la grille.
- updateRoomInfos(String, double, double): Permet de mettre à jour les informations de la **Room** provenant du controlleur graphique au niveau de la couche Domaine ainsi que graphique.
- getSelectedSectionInfos(): Retourne les informations de la section actuellement sélectionnée.
- updateSelectedSectionInfos(): Permet de mettre à jour les informations provenant du controlleur graphique pour la section actuellement sélectionné au niveau de la couche Domaine ainsi que graphique.
- loadProject() : Permet de lier le chargement d'un projet du contrôleur graphique au **Core**.
- saveProject(String) : Permet de lier la sauvegarde du contrôleur graphique au **Core.**
- undo() : Permet d'annuler la dernière action.
- redo (): Permet de refaire la dernière action annulée.
- zoomIn() : Permet de zoomer en avant.
- zoomOut() : Permet de zoomer en arrière.
- magneticGrid() : Permet d'activer ou non la grille magnétique.
- doneAction() : Permet de valider l'action en cours.
- cancelAction(): Permet d'annuler l'action en cours.

1.1.5.2. Shape

1.1.5.2.1. InnoPolygon

Méthodes:

- InnoPolygon
- onMouseClicked(**MouseEvent**) : Permet de traiter l'action d'un clic de souris
- onMouseMove(**MouseEvent**) : Permet de traiter l'action d'un mouvement de souris
- onMouseExited(**MouseEvent**) : Permet de traiter l'action de la sortie de la souris de la fenêtre.
- onMousePressed(**MouseEvent**) : Permet de traiter l'action de maintenir un clic de souris.
- onMouseReleased(**MouseEvent**) : Permet de traiter l'action de libérer le clic de souris.
- onMouseOnDragDetected(**MouseEvent**) : Permet de traiter l'action de glisser la souris.

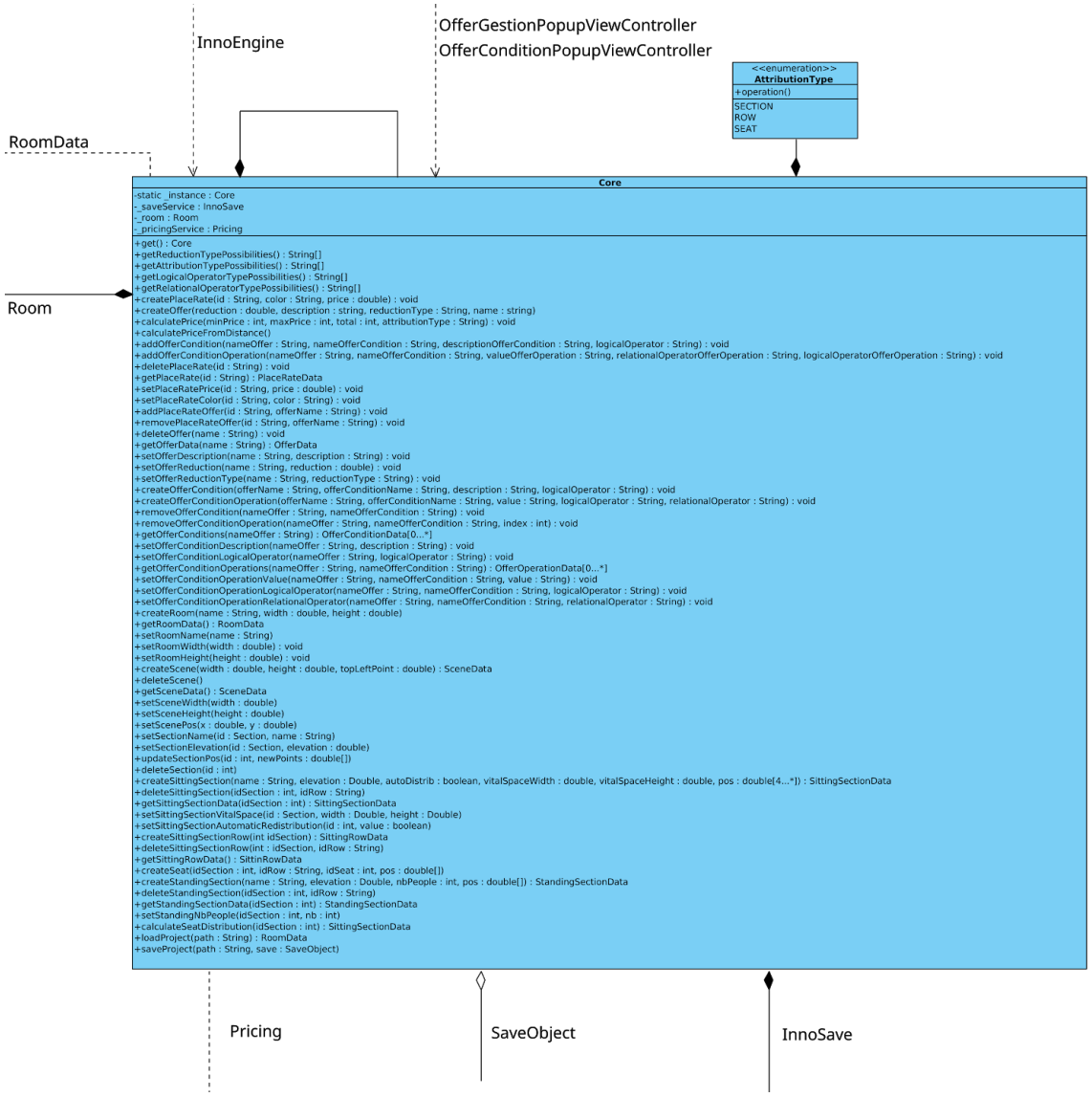
1.1.5.2.2. InnoRectangle

Méthodes:

- InnoRectangle
- onMouseClicked(**MouseEvent**) : Permet de traiter l'action d'un clic de souris
- onMouseMoved(**MouseEvent**) : Permet de traiter l'action d'un mouvement de souris
- onMouseExited(**MouseEvent**) : Permet de traiter l'action de la sortie de la souris de la fenetre.
- onMousePressed(**MouseEvent**) : Permet de traiter l'action de maintenir un clic de souris.
- onMouseReleased(**MouseEvent**) : Permet de traiter l'action de libérer le clic de souris.
- onMouseOnDragDetected(**MouseEvent**) : Permet de traiter l'action de glisser la souris.

1.2. App

1.2.1. Core



La classe **Core** est notre contrôleur de Larman, elle permet d'effectuer une liaison entre **la couche UI** et **la couche du Domaine** et la couche des Technical Services, elle est statique et à un attribut qui correspond à l'instance d'elle même ce qui signifie que le Core est un Singleton, il est accessible à n'importe quel endroit dans notre Projet
Cette classe possède 3 attributs :

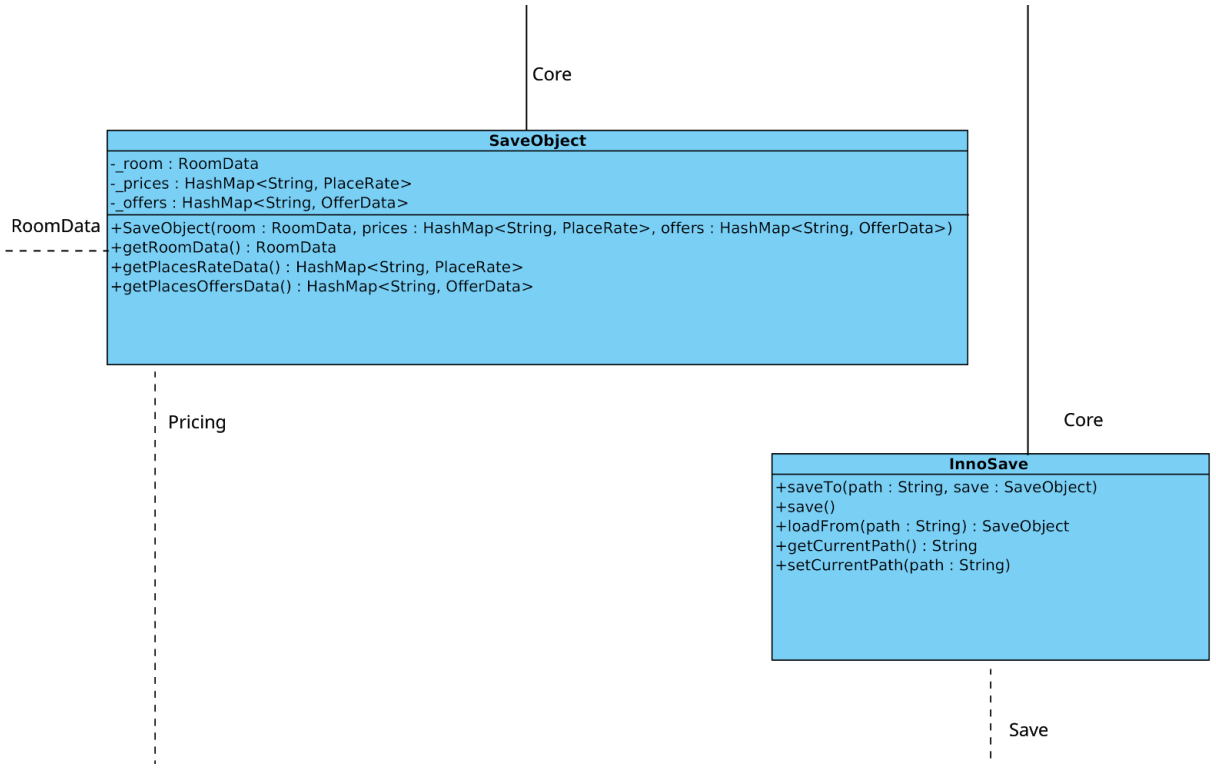
- *static _instance* correspond à une instance de la classe **Core**.
- *_saveService* correspond à une instance de la classe **Save.**
- *_princingService* correspond à une instance de la classe **Pricing.**
- *_room* correspond à une instance de la classe **Room.**

Elle contient x méthodes:

- *get* permet de récupérer l'attribut *_instance*.

- *getReductionTypePossibilities* appelle la fonction *getReductionTypePossibilities* de la classe **Pricing**
- *getAttributionTypePossibilities* appelle la fonction *getAttributionTypePossibilities* de la classe **Pricing**
- *getLogicalOperatorTypePossibilities* appelle la fonction *getLogicalOperatorTypePossibilities* de la classe **Pricing.**
- *getRelationalOperatorTypePossibilities* appelle la fonction *getRelationalOperatorTypePossibilities* de la classe **Pricing.**
- *createPlaceRate* appelle la fonction *createPlaceRate* de la classe **Pricing.**
- *createOffer* appelle la fonction *createOffer* de la classe **Pricing.**
- *calculatePrice* calcule le prix par automatiquement par apport aux renseignements donnés.
- *calculatePriceFromDistance* calcule un prix par apport à la distance d'une place à la scene.
- *addOfferCondition* appelle la fonction *addOfferCondition* de la classe **Pricing.**
- *addOfferConditionOperation* appelle la f*addOfferConditionOperationbilities* de la classe **Pricing.**
- *deletePlaceRate* appelle la fonction *deletePlaceRate* de la classe **Pricing.**
- *getPlaceRate* appelle la fonction *getPlaceRate* de la classe **Pricing.**
- *setPlaceRatePrice* appelle la fonction *setPlaceRatePrice* de la classe **Pricing.**
- *setPlaceRateColor* appelle la fonction *setPlaceRateColor* de la classe **Pricing.**
- *addPlaceRateOffer* appelle la fonction *addPlaceRateOffer* de la classe **Pricing.**
- *removePlaceRateOffer* appelle la fonction *removePlaceRateOffer* de la classe **Pricing.**
- *deleteOffer* appelle la fonction *deleteOffer* de la classe **Pricing.**
- *getOfferData* appelle la fonction *getOfferData* de la classe **Pricing.**
- *setOfferDescription* appelle la fonction *setOfferDescription* de la classe **Pricing.**
- *setOfferReduction* appelle la fonction *setOfferReduction* de la classe **Pricing.**
- *setOfferReductionType* appelle la fonction *setOfferReductionType* de la classe **Pricing.**
- *removeOfferCondition* appelle la fonction *removeOfferCondition* de la classe **Pricing.**
- *removeOfferConditionOperation* appelle la fonction *removeOfferConditionOperation* de la classe **Pricing.**
- *getOfferConditions* appelle la fonction *getOfferConditions* de la classe **Pricing.**
- *setOfferConditionDescription* appelle la fonction *setOfferConditionDescription* de la classe **Pricing.**
- *setOfferConditionLogicalOperator* appelle la fonction *setOfferConditionLogicalOperator* de la classe **Pricing.**
- *getOfferConditionOperations* appelle la fonction *getOfferConditionOperations* de la classe **Pricing.**
- *setOfferConditionOperationValue* appelle la fonction *setOfferConditionOperationValue* de la classe **Pricing.**
- *setOfferConditionOperationLogicalOperator* appelle la fonction *setOfferConditionOperationLogicalOperator* de la classe **Pricing.**
- *setOfferConditionOperationRelationalOperator* appelle la fonction *setOfferConditionOperationRelationalOperator* de la classe **Pricing.**
- *createRoom* appelle la fonction *createRoom* de la classe **Room.**

- *getRoomData* appelle la fonction *getRoomData* de la classe **Room.**
- *setRoomName* appelle la fonction setRoomName de la classe **Room.**
- *setRoomHeight* appelle la fonction *setRoomHeight* de la classe **Room.**
- *setRoomWidth* appelle la fonction *setRoomWidth* de la classe **Room.**
- *createScene* appelle la fonction *createScene* de la classe **Room.**
- *deleteScene* appelle la fonction *deleteScene* de la classe **Room.**
- *getSceneData* appelle la fonction *getSceneData* de la classe **Room.**
- *setSceneWidth* appelle la fonction *setSceneWidth* de la classe **Room.**
- *setSceneHeight* appelle la fonction *setSceneHeight* de la classe **Room.**
- *setScenePos* appelle la fonction *setScenePos* de la classe **Room.**
- *setSectionName* appelle la fonction *setSectionName* de la classe **Room.**
- *setSectionElevation* appelle la fonction *setSectionElevation* de la classe **Room.**
- *updateSectionPos* appelle la fonction *updateSectionPos* de la classe **Room.**
- *deleteSection* appelle la fonction *deleteSection* de la classe **Room.**
- *createSittingSection* appelle la fonction *createSittingSection* de la classe **Room.**
- *deleteSittingSection* appelle la fonction *deleteSittingSection* de la classe **Room.**
- *getSittingSectionData* appelle la fonction *getSittingSectionData* de la classe **Room.**
- *setSittingSectionVitalSpace* appelle la fonction *setSittingSectionVitalSpace* de la classe **Room.**
- *setSittingSectionAutomaticRedistribution* appelle la fonction *setSittingSectionAutomaticRedistribution* de la classe **Room.**
- *createSittingSectionRow* appelle la fonction *createSittingSectionRow* de la classe **Room.**
- *deleteSittingSectionRow* appelle la fonction *deleteSittingSectionRow* de la classe **Room.**
- *getSittingRowData* appelle la fonction *getSittingRowData* de la classe **Room.**
- *createSeat* appelle la fonction *createSeat* de la classe **Room.**
- *createStandingSection* appelle la fonction *createStandingSection* de la classe **Room.**
- *deleteStandingSection* appelle la fonction *deleteStandingSection* de la classe **Room.**
- *getStandingSectionData* appelle la fonction *getStandingSectionData* de la classe **Room.**
- *setStandingNbPeople* appelle la fonction *setStandingNbPeople* de la classe **Room.**
- *calculateSeatDistribution* appelle la fonction *calculateSeatDistribution* de la classe **Room.**
- *loadProject* appelle la fonction *loadProject* du service **Save.**
- *saveProject* appelle la fonction *loadProject* du service **Save.**



1.2.2. SaveObject

La classe **SaveObject** contient les objets qui vont être sérialisée au moment de la sauvegarde.
Elle contient 3 attributs:

- *_room* contient les informations de la salle sous forme de **RoomData**
- *_prices* contient les informations des prix qui ont été créée sous forme de **PlaceRateData**
- *_offers* contient les informations des offres disponible sous form de **OfferData**

Cette class possède 3 methods:

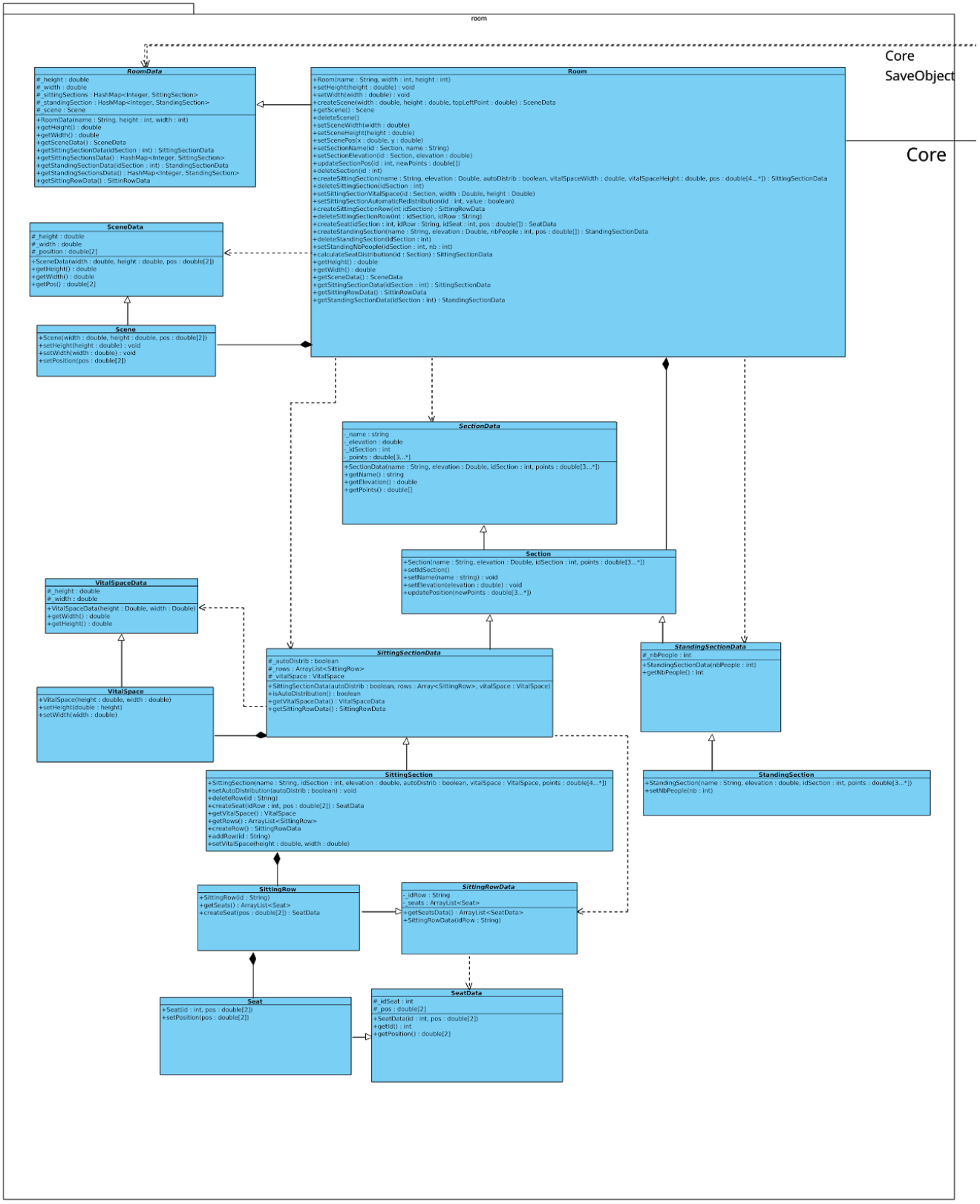
- *getRoomData* permet de récupérer l'attribut *_room*
- *getPlacesRateData* permet de récupérer l'attribut *_prices*
- *getOffersData* permet de récupérer l'attribut *_offers*

1.2.3. InnoSave

La class **InnoSave** permet de sauvegarder n'importe quel type d'objet dans un fichier.
Elle possède 5 methods:

- setCurrentPath va modifier le path du fichier du projet dans le service **Save.**
- getCurrentPaht va récupérer le path du fichier du projet dans le service **Save.**
- loadFrom va charger un fichier et le transformer en **SaveObject.**
- save va sauvegarder le projet dans un fichier.
- saveTo permet de sauvegarder le projet dans un nouveau fichier.

1.2.4. Room


room
Core
SaveObject
Core
RoomData
height : double
width : double
sittingSections : HashMap<Integer, SittingSection>
standingSection : HashMap<Integer, StandingSection>
scene : Scene
+RoomData(name : String, height : int, width : int)
+getHeight() : double
+getWidth() : double
+getSceneData() : SceneData
+getSittingSectionData(idSection : int) : SittingSectionData
+getSittingSectionsData() : HashMap<Integer, SittingSection>
+getStandingSectionData(idSection : int) : StandingSectionData
+getStandingSectionsData() : HashMap<Integer, StandingSection>
+getSittingRowData() : SittinRowData
Room
+Room(name : String, width : int, height : int)
+setHeight(height : double) : void
+setWidth(width : double) : void
+createScene(width : double, height : double, topLeftPoint : double) : SceneData
+getScene() : Scene
+deleteScene()
+setSceneWidth(width : double)
+setSceneHeight(height : double)
+setScenePos(x : double, y : double)
+setSectionName(id : Section, name : String)
+setSectionElevation(id : Section, elevation : double)
+updateSectionPos(id : int, newPoints : double[])
+deleteSection(id : int)
+createSittingSection(name : String, elevation : Double, autoDistrib : boolean, vitalSpaceWidth : double, vitalSpaceHeight : double, pos : double[4..*]) : SittingSectionData
+deleteSittingSection(idSection : int)
+setSittingSectionVitalSpace(id : Section, width : Double, height : Double)
+setSittingSectionAutomaticRedistribution(id : int, value : boolean)
+createSittingSectionRow(int idSection) : SittingRowData
+deleteSittingSectionRow(int : idSection, idRow : String)
+createSeat(idSection : int, idRow : String, idSeat : int, pos : double[]) : SeatData
+createStandingSection(name : String, elevation : Double, nbPeople : int, pos : double[]) : StandingSectionData
+deleteStandingSection(idSection : int)
+setStandingNbPeople(idSection : int, nb : int)
+calculateSeatDistribution(id : Section) : SittingSectionData
+getHeight() : double
+getWidth() : double
+getSceneData() : SceneData
+getSittingSectionData(idSection : int) : SittingSectionData
+getSittingRowData() : SittinRowData
+getStandingSectionData(idSection : int) : StandingSectionData
SceneData
height : double
width : double
position : double[2]
+SceneData(width : double, height : double, pos : double[2])
+getHeight() : double
+getWidth() : double
+getPos() : double[2]
Scene
+Scene(width : double, height : double, pos : double[2])
+setHeight(height : double) : void
+setWidth(width : double) : void
+setPosition(pos : double[2])
SectionData
_name : string
_elevation : double
_idSection : int
_points : double[3...*]
+SectionData(name : String, elevation : Double, idSection : int, points : double[3...*])
+getName() : string
+getElevation() : double
+getPoints() : double[]
Section
+Section(name : String, elevation : Double, idSection : int, points : double[3...*])
+setIdSection()
+setName(name : string) : void
+setElevation(elevation : double) : void
+updatePosition(newPoints : double[3...*])
VitalSpaceData
height : double
width : double
+VitalSpaceData(height : Double, width : Double)
+getWidth() : double
+getHeight() : double
VitalSpace
+VitalSpace(height : double, width : double)
+setHeight(double : height)
+setWidth(width : double)
SittingSectionData
autoDistrib : boolean
rows : ArrayList<SittingRow>
vitalSpace : VitalSpace
+SittingSectionData(autoDistrib : boolean, rows : Array<SittingRow>, vitalSpace : VitalSpace)
+isAutoDistribution() : boolean
+getVitalSpaceData() : VitalSpaceData
+getSittingRowData() : SittingRowData
StandingSectionData
nbPeople : int
+StandingSectionData(nbPeople : int)
+getNbPeople() : int
SittingSection
+SittingSection(name : String, idSection : int, elevation : double, autoDistrib : boolean, vitalSpace : VitalSpace, points : double[4...*])
+setAutoDistribution(autoDistrib : boolean) : void
+deleteRow(id : String)
+createSeat(idRow : int, pos : double[2]) : SeatData
+getVitalSpace() : VitalSpace
+getRows() : ArrayList<SittingRow>
+createRow() : SittingRowData
+addRow(id : String)
+setVitalSpace(height : double, width : double)
StandingSection
+StandingSection(name : String, elevation : double, idSection : int, points : double[3...*])
+setNbPeople(nb : int)
SittingRow
+SittingRow(id : String)
+getSeats() : ArrayList<Seat>
+createSeat(pos : double[2]) : SeatData
SittingRowData
_idRow : String
_seats : ArrayList<Seat>
+getSeatsData() : ArrayList<SeatData>
+SittingRowData(idRow : String)
Seat
+Seat(id : int, pos : double[2])
+setPosition(pos : double[2])
SeatData
idSeat : int
pos : double[2]
+SeatData(id : int, pos : double[2])
+getId() : int
+getPosition() : double[2]

1.2.4.1. Room

La classe **Room** permet la gestion de la salle. Elle hérite directement de **RoomData** qui contient les attributs nécessaires .
Elle possède 29 méthodes :

- *setHeight* permet de modifier l'attribut *_height* de la classe abstraite **RoomData**.
- *setWidth* permet de modifier l'attribut *_width* de la classe abstraite **RoomData**.
- *createScene* permet d'appeler le constructeur de la classe **Scene**.
- *getScene* permet de récupérer l'attribut *_scene* de la classe abstraite **RoomData**.
- *deleteScene* permet de supprimer la scène de la classe abstraote **RoomData**.
- *setSceneWidth* permet d'appeler la méthode *setWidth* de la classe **Scene**.
- *setSceneHeight* permet d'appeler la méthode *setHeight* de la classe **Scene**.
- *setScenePos* permet d'appeler la méthode *setPosition* de la classe **Scene**.
- *setSectionData* permet d'appeler la méthode *setName* de la classe **Section**.
- *setSectionElevation* permet d'appeler la méthode *setElevation* de la classe **Section**.
- *updateSection* permet d'appeler la méthode *updatePosition* de la classe **Section**.
- *deleteSection* permet de supprimer une section par rapport à son id.
- *createSittingSection* permet de créer l'objet **SittingSection**.
- *deleteSittingSection* permet de supprimer une section assise par rapport à son id.
- *setSittingSectionVitalSpace* permet d'appeler la méthode *setVitalSpace* de la classe **SittingSection**.
- *setSittingSectionAutomaticRedistribution* permet d'appeler la méthode *setAutoDistribution* de la classe **SittingSection**.
- *createSittingSectionRow* permet d'appeler la méthode *createRow* de la classe **SittingSection** en renvoyant les informations propres à cette rangée.
- *deleteSittingSectionRow* permet d'appeler la méthode *deleteRow* de la classe **SittingSection** par rapport à son id.
- *createSeat* permet d'appeler la méthode *createSeat* de la classe **SittingRow**.
- *createStandingSection* permet de créer l'objet **StandingSection**.
- *deleteStandingSection* permet de supprimer une section debout par rapport à son id.
- *setStandingNbPeople* permet d'appeler la méthode *setNbPeople* de la classe **StandingSection**.
- *calculateSeatDistribution* permet de distribuer automatiquement les sièges d'une section en nous renvoyant les informations de cette section.
- *getHeight* permet d'appeler la méthode *getHeight* de la classe abstraite **RoomData***.*
- *getWidth* permet d'appeler la méthode *getWidth* de la classe abstraite **RoomData***.*
- *getSceneData* permet d'appeler la méthode *getSceenData* de la classe abstraite **RoomData.**
- *getSittingSectionData* permet d'appeler la méthode *getSittingSectionData* de la classe abstraite **RoomData** en fonction de son id.
- *getSittingRowData* permet d'appeler la méthode *getSittingRowData* de la classe abstraite **RoomData**.
- *getStandingSectionData* permet d'appeler la méthode *getStandingsSectionData* de la classe abstraite **RoomData** en fonction de son id.

Le constructeur permet d'initialiser la salle en fonction de sa longueur et de sa largeur.

1.2.4.2. RoomData

La classe **RoomData** est une class abstraite contenant les attributs et les getters nécessaire à la classe **Room**, grâce à cela cette classe ne contenant que des getters peut être utilisé par l'**UI**.
Cette classe possède 5 attributs :

- ***_height*** correspond à la longueur de la salle.
- ***_width*** correspond à la largeur de la salle.
- ***_sittingSections*** correspond à un index de **SittingSections** c'est-à-dire des sections assises.
- ***_standingSections*** correspond à un index de **StandingSection** c'est-à-dire des sections debouts.
- ***_scene*** correspond à l'objet **Scene**.

Elle possède 8 méthodes :

- *getHeight* permet de renvoyer l'attribut ***_height***.
- *getWidth* permet de renvoyer l'attribut ***_width***.
- *getSceneData* permet de renvoyer les informations provenant de la classe abstraite **SceneData**.
- *getSittingSectionData* permet de renvoyer les informations provenant classe abstraite **SittingSectionData**.
- *getSittingSectionsData* permet de renvoyer un index d'objet de type **SittingSection**.
- *getStandingSectionData* permet de renvoyer les informations provenant de la classe abstraite **StandingSectionData**.
- *getStandingSectionsData* permet de renvoyer un index d'objet de type **StandingSection**.
- *getSittingRowData* permet de renvoyer les informations provenant de la classe abstraite **SittingRowData**.

1.2.4.3. SceneData

La classe **SceneData** est une class abstraite contenant les attributs et les getters nécessaire à la classe **Scene**, grâce à cela cette classe ne contenant que des getters peut être utilisé par l'**UI**.
Cette classe possède 3 attributs :

- ***_height*** correspond à la longueur de la scène.
- ***_width*** correspond à la largeur de la scène.
- ***_position*** correspond au point en haut à gauche de la scène.

Elle possède 3 méthodes :

- *getHeight* permet de retourner l'attribut ***_height.***
- *getWidth* permet de retourner l'attribut ***_width.***
- *getPos* permet de retourner l'attribut ***_position.***

1.2.4.4. Scene

La classe **Scene** permet la gestion de la scène positionnée dans la salle.
Elle hérite directement de **SceneData** qui contient les attributs nécessaires.
Elle possède 3 méthodes :

- *setHeight* permet de modifier l'attribut ***_height*** de **SceneData**.
- *setWidth* permet de modifier l'attribut ***_width*** de **SceneData**.
- *setPosition permet de modifier l'attribut* ***_position de SceneData.***

Son constructeur permet d'initialiser la scene en fonction de sa longueur, de sa largeur et du point en haut à gauche.

1.2.4.5. SectionData

La classe **SectionData** est une class abstraite contenant les attributs et les getters nécessaire à la classe **Section**, grâce à cela cette classe ne contenant que des getters peut être utilisé par l'**UI**.
Cette classe possède 4 attributs :

- ***_idSection*** correspond à l'identifiant de la section.
- ***_name*** correspond au nom de la section.
- ***_elevation*** correspond à la valeur de l'élévation de la section.
- ***_points*** correspond au tableau des valeurs des points constituant la section.

Elle possède 3 méthodes :

- *getName* permet de renvoyer l'attribut ***_name de la section.***
- *getElevation* permet de renvoyer l'attribut ***_elevation*** de la section.
- *getPoints* permet de renvoyer l'attribut ***_points*** de la section.

Le constructeur de la classe prend en paramètre le nom de la section, l'index de celle-ci, son élévation ainsi que les coordonnées de ses points.

1.2.4.6. Section

La classe **Section** permet la gestion des multiples sections présentes dans la salle.
Elle hérite directement de **SectionData** qui contient les attributs nécessaires.
Elle possède 4 méthodes :

- *setIdSection* permet de modifier l'attribut ***_idSection*** de **SectionData**.
- *setName* permet de modifier l'attribut ***_name*** de **SectionData.**
- *setElevation* permet de modifier l'attribut ***_elevation*** de **SectionData.**
- *updatePosition* permet de modifier l'attribut ***_points*** de **SectionData.**

Son constructeur permet d'initialiser une section en fonction de son nom, son élévation, son index et de ses points le délimitant.

1.2.4.7. StandingSectionData

La classe **StandingSectionData** est une class abstraite contenant les attributs et les getters nécessaire à la classe **StandingSection**, grâce à cela cette classe ne contenant que des getters peut être utilisé par l'**UI**.
Elle hérite directement de **Section** ainsi que de **SectionData.**

Cette classe possède 1 attribut :

- *_nbPeople* correspond au nombre maximum de personne pouvant être présent dans la section.

Elle possède 1 méthode :

- *getNbPeople* permet de renvoyer l'attribut *_nbPeople de la section.*

Son constructeur permet d'initialiser une section debout en fonction du nombre maximum de personne pouvant être présent dans la section.

1.2.4.8. StandingSection

La classe **StandingSection** permet la gestion des sections debouts présentes dans la salle.
Elle hérite directement de **StandingSectionData** qui contient les attributs nécessaires.
Elle possède 1 méthode :

- *setNbPeople* permet de modifier l'attribut *_nbPeople* de **StandingSectionData.**

Son constructeur permet d'initialiser une section debout en fonction de son nom, son élévation, son index et de ses points le délimitant.

1.2.4.9. SittingSectionData

La classe **SittingSectionData** est une class abstraite contenant les attributs et les getters nécessaire à la classe **SittingSection**, grâce à cela cette classe ne contenant que des getters peut être utilisé par l'**UI**.
Elle hérite directement de **Section** ainsi que de **SectionData.**
Cette classe possède 3 attributs :

- *_autoDistrib* permet de savoir si il y a une auto distribution des sièges dans la section
- *_rows* correspond à une liste d'objet **SittingRow** correspondant aux rangées.
- *_vitalSpace* correspond à l'objet **VitalSpace.**

Elle possède 3 méthodes :

- *isAutoDistribution* permet de renvoyer l'attribut *_autoDistrib* de la section.
- *getVitalSpaceData* permet de renvoyer les informations de l'objet **VitalSpaceData.**
- *getSittingRowData* permet de renvoyer les informations de l'objet **SittingRowData**.

Son constructeur permet d'initialiser une section assise en fonction de l'autodistribution, l'espace vital et un tableau d'objet **SittingRow**.

1.2.4.10. SittingSection

La classe **SittingSection** permet la gestion des sections assises présentes dans la salle.
Elle hérite directement de **SittingSectionData** qui contient les attributs nécessaires.
Elle possède 8 méthodes :

- *setAutoDistribution* permet de modifier l'attribut *_autoDistrib* de **SittingSectionData.**
- *deleteRow* permet de supprimer une rangée en fonction de son id.
- *createSeat* permet d'appeler la fonction *createSeat* de la classe **SittingRow** en renvoyant les informations de la classe abstraite **SeatData.**
- *getVitalSpace* permet de renvoyer l'attribut *_vitalSpace* de **SittingSectionData.**
- *getRows* permet de renvoyer l'attribut *_rows* de **SittingSectionData.**

- *createRow* permet de créer une rangée et de récupérer les informations de **SittingRowData.**
- *addRow* permet d'ajouter une rangée en fonction de son id.
- *setVitalSpace* permet de modifier l'attribut ***_vitalSpace*** de **SittingSectionData.**

Son constructeur permet d'initialiser une section assise en fonction de son nom, son élévation, son index, son autodistribution, son espace vital et ses points le délimitant.

1.2.4.11. SittingRowData

La classe **SittingRowData** est une classe abstraite contenant les attributs et les getters nécessaire à la classe **SittingRow**, grâce à cela cette classe ne contenant que des getters elle peut être utilisé par l'**UI**.
Cette classe contient 2 attributs:

- ***_idRow*** correspond à l'identifiant de la rangée.
- ***_seats*** est une liste de **Seat.**

Elle possède 1 méthode:

- *getSeatsData* permet de récupérer la liste de **Seat** sous forme de **SeatData.**

1.2.4.12. SittingRow

La classe **SittingRow** permet de gérer une section de siège elle hérite de **SittingRowData**
Elle possède 2 attributs:

- *getSeats* permet de récupérer l'attribut _seat.
- *createSeat* permet de créer un siège.

1.2.4.13. SeatData

La classe **SeatData** est une classe abstraite contenant les attributs et les getters nécessaire à la classe **Seat**, grâce à cela cette classe ne contenant que des getters elle peut être utilisé par l'**UI**.
Cette classe possède 2 attributs:

- ***_idSeat*** correspond à l'identifiant du siège.
- ***_pos*** correspond à la position du siège.

Elle possède 2 méthodes:

- *getIdSeat* permet de récupérer l'identifiant du siège.
- *getPos* permet de récupérer la position du siège.

1.2.4.14. Seat

La classe **Seat** permet la gestion des sièges présents dans une section assise elle hérite de **SeatData**.
Elle possède 1 méthode :

- *setPosition* permet de modifier la position d'un siège.

1.2.4.15. VitalSpaceData

La classe **VitalSpaceData** est une classe abstraite contenant les attributs et les getters nécessaire à la classe **VitalSpace**, grâce à cela cette classe ne contenant que des getters elle peut être utilisé par l'**UI**.
Cette classe possède 2 attributs:

- *_height* correspond à la longueur de l'espace vital.
- *_width* correspond à la largeur de l'espace vital.

Elle possède 2 méthodes:

- *getHeight* permet de récupérer l'attribut _height.
- *getWidth* permet de récupérer l'attribut _width.

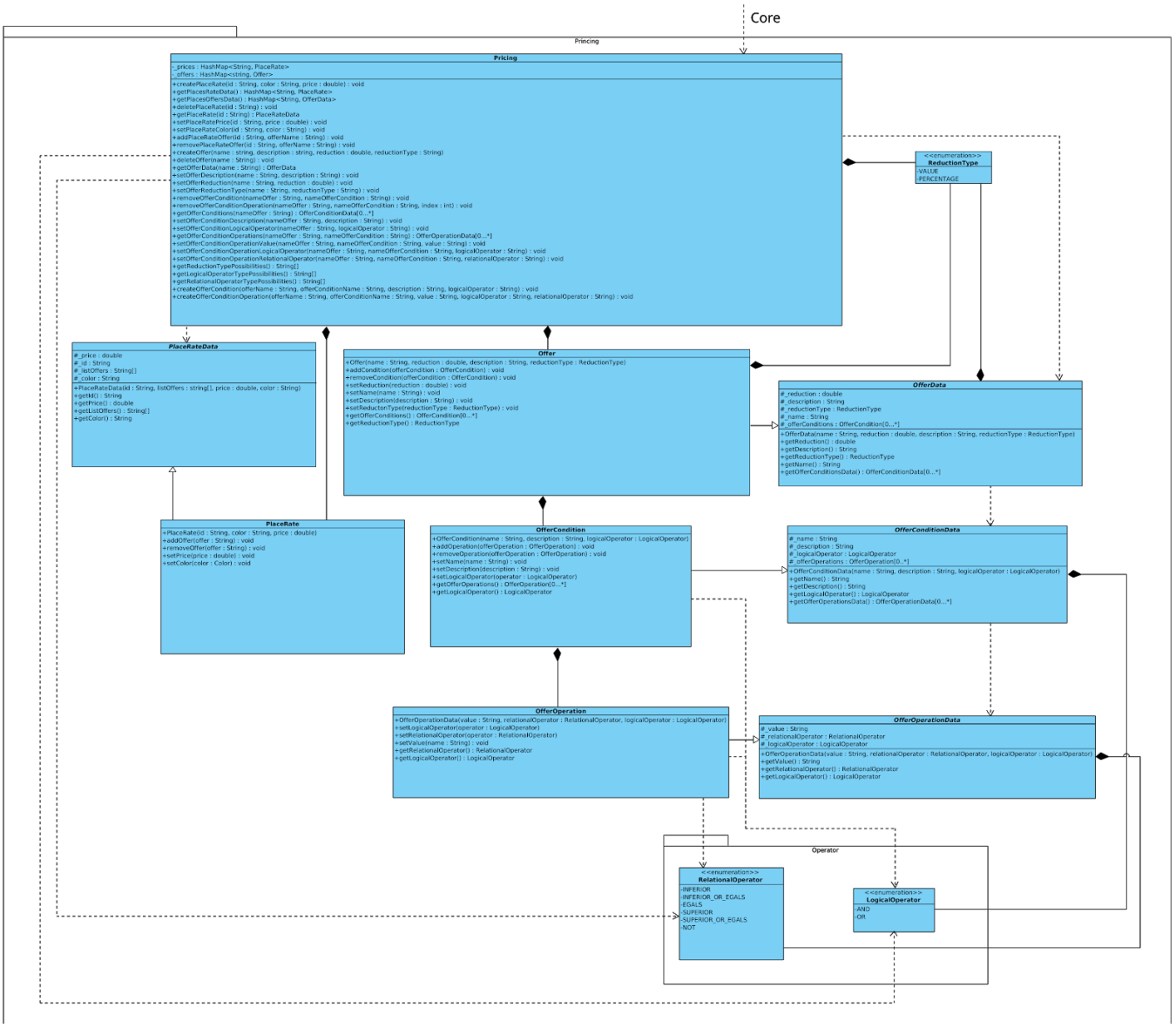
1.2.4.16. VitalSpace

La classe **VitalSpace** permet la gestion de l'espace vital des sièges présents dans les sections elle hérite de **VitalSpaceData.**
Elle possède 2 méthodes :

- *setHeight* permet de modifier l'attributs *_height.*
- *setWidth* permet de récupérer l'attributs *_width.*

1.3. Services

1.3.1. Pricing



1.3.1.1. Pricing

La classe **Pricing** est un service permettant d'attribuer un prix à une place grâce à un identifiant, gérer les offres de celle-ci en créer de nouvelles, gérer les offres existantes, modifier un prix, supprimer le prix d'une place.
Pour nous la classe **Pricing** figure parmis les services car par sa conception elle n'est pas liée à notre projet, elle pourrait utilisée comme un service par un autre projet de la même manière.
Elle possède 2 attributs:

- *_prices* est un index de **PlaceRate** qui correspond à la liste des prix sur une place qui ont été créés.
- *_offers* est un index d'**Offer** qui correspond à la liste des offres disponibles.
  Cette classe contient 4 méthodes:

- *createPlaceRate* permet de créer un prix pour une place avec une couleur, un prix et un identifiant unique afin d'identifier la place, une fois créée, le prix sera contenu dans *_prices*, il sera par la suite possible d'ajouter ou retirer une offre au prix avec les fonctions *addPlaceRateOffer, removePlaceRateOffer*.
- *deletePlaceRate* permet de supprimer le prix d'une place grâce à son identifant.
- *getPlaceRate* permet de récupérer une classe abstraite dont **PlaceRate** hérite qui contient les attributs et les getters nécessaire, celle-ci se nomme **PlaceRateData.**
- *setPlaceRatePrice* permet de modifier le prix d'une place.
- *setPlaceRateColor* permet de modifier la couleur d'une place.
- *addPlaceRateOffer* permet d'ajouter une offre à une place.
- *removePlaceRateOffer* permet de retirer une offre à une place.
- *createOffer* permet de créer une offre ce qui va par la suite l'ajouter dans la liste des offres disponible à l'ajout d'une place. (*_offers*).
- *deleteOffer* permet de supprimer une offre de la liste des offres disponible (*_offers).*
- *getOfferData* permet de récupérer une classe abstraite dont **Offer** hérite qui contient les attributs et les getters nécessaire, celle-ci se nomme **OfferData**.
- *setOfferDescription* permet de changer la description d'une offre. (**Offer**)
- *setOfferReduction* permet de changer la valeur de la réduction d'une offre. (**Offer**)
- *setOfferReductionType* permet de changer le type d'une offre, si celle-ci s'appliquera avec un pourcentage ou avec une valeur fixe. (**Offer**)
- *createOfferCondition* permet de créer une condition à une offre.
- *removeOfferCondition* permet de supprimer une condition à une offre.
- *createOfferConditionOperation* permet de créer une opération à une condition qui elle même est assigné à une offre.
- *removeOfferConditionOperation* permet de supprimer une opération à une condition qui est elle même associé à une offre.
- *getOfferConditions* permet de récupérer une liste de conditions associé à une offre.(**OfferConditionData**)
- setOfferConditionDescription permet de changer la description d'une condition. (**OfferCondition**)
- *setOfferConditionLogicalOperator* permet de changer l'attribut *_logicalOperator* d'une condition. (**OfferCondition**)
- *getOfferConditionOperations* permet de récupérer une liste des opérations associé à une condition. (**OfferOperationData**)
- *setOfferConditionOperationValue* permet de modifier la valeur d'une opération d'une condition d'une offre. (**OfferOperation**)
- *setOfferConditionOperationLogicalOperator* permet de modifier l'opérateur logique d'une opération d'une condition d'une offre. (**OfferOperation**)
- *setOfferConditionOperationRelationalOperator* permet de modifier l'opérateur relationnel d'une opération d'une condition d'une offre. (**OfferOperation**)
- *getReductionTypePossibilities* permet de récupérer une liste sous forme de tableau de **String** décrivant tous les différents types de réductions disponible.
- *getLogicalOperatorTypePossibilities* permet de récupérer une liste sous forme de tableau de **String** décrivant tous les différents types de d'opérateur de logique disponible.

- *getRelationalOperatorTypePossibilities* permet de récupérer une liste sous forme de tableau de **String** décrivant tous les différents types de d'opérateur de relations disponible.

1.3.1.2. PlaceRateData

La classe **PlaceRateData** est une class abstraite contenant les attributs et les getters nécessaire à la classe **PlaceRate**, grâce à cela cette classe ne contenant que des getters peut être utilisé par l'**UI**.
Elle possède 4 attributs:

- *_price* correspond à la valeur du prix de la place.
- *_id* correspond à l'identifiant de la place.
- *_listOffers* est une liste d'offres associé à la place.
- *_color* correspond à la couleur de la place.

Cette classe contient 4 methodes:

- *getPrice* permet de récupérer l'attribut *_price*.
- *getListOffers* permet de récupérer l'attribut *_listOffers*
- *getColor* permet de récupérer l'attribut *_color*.
- *getId* permet de récupérer l'attribut *_id*.

Le constructeur de la classe prend en paramètre l'identifiant de la place, une couleur, la valeur de la place.

1.3.1.3. PlaceRate

La classe **PlaceRate** gère le prix d'une place, sa couleur et ses offres. Elle hérite directement de **PlaceRateData** qui contient les attributs nécessaires .
Cette classe contient 4 methodes:

- *setPrice* permet de modifier l'attribut *_price*.
- *setColor* permet de modifier l'attribut *_color*.
- addOffer permet d'ajouter une offre dans *_listOffers*
- *removeOffer* permet de retirer une offre à la place

Le constructeur de la classe prend en paramètre l'identifiant de la place, une couleur, la valeur de la place.

1.3.1.4. OfferRate

La class **OfferData** est une class abstraite contenant les attributs et les getters nécessaire à la classe **Offer**, grâce à cela cette classe ne contenant aucun setters elle peut être utilisé par l'**UI**.
Elle possède 5 attributs:

- *_name* est le nom de l'offre
- *_reduction* correspond à la valeur de l'offre
- *_description* correspond à la description de l'offre
- *_listConditions* est une liste de **OfferCondition**

- *_reductionType* est le type de la réduction (**ReductionType**), par pourcentage ou par valeur.

Cette classe contient 5 méthodes:

- *getReductionTypeValue* permet de récupérer l'attribut *_reductionType sous forme d'une **String.***
- *getOfferConditionsData* permet de récupérer l'attribut *_listConditions* sous forme d'une liste de **OfferConditionData.**
- *getDescription* permet de récupérer l'attribut *_description.*
- *getReduction* permet de récupérer l'attribut *_reduction.*
- *getName* permet de récupérer l'attribut *_name.*

1.3.1.5. Offer

La classe **Offer** permet de gérer les offres qui pourront être attribué à une place.
Elle possède 1 type:

- **ReductionType** correspond au type de la réduction, réduction avec pourcentage ou réduction avec valeur.

Cette classe contient 8 méthodes:

- *addCondition* permet d'ajouter une condition dans *_listConditions.*
- *removeCondition* permet de retirer une condition de la list _listConditions.
- *setReduction* permet de modifier l'attribut *_reduction.*
- *setName* permet de modifier l'attribut *_name.*
- *setDescription* permet de modifier l'attribut *_description.*
- *setReductonType* permet de modifier l'attribut *_reductionType.*
- *getOfferConditions* permet de récupérer l'attribut *_listConditions* sous forme d'une liste de **OfferCondition.**
- *getReductionType* permet de récupérer l'attribut *_reductionType.*

1.3.1.6. OfferConditionData

La class **OfferConditionData** est une class abstraite contenant les attributs et les getters nécessaire au bon fonctionnement de la classe **OfferCondition**, étant donnée que cette classe ne contient aucune méthode de modification elle peut être utilisé par l'**UI**.
Elle possède 4 attributs:

- *_name* correspond au nom de la condition.
- *_description* correspond à la description de la condition.
- *_listOfferOperations* est une liste d'**OfferOperation**.
- *_logicalOperator* est un enum de type **LogicalOperator**, il défini si la condition doit être validé indépendamment ou dépendant des autres conditions.

Cette classe contient 4 méthodes:

- *getName* permet de récupérer l'attribut *_name*
- *getLogicalOperatorValue* permet de récupérer l'attribut *_logicalOperator sous forme de **String**.*
- *getDescription* permet de récupérer l'attribut *_description*

- *getOfferOperationsData* permet de récupérer l'attribut *_listOfferOperations* sous forme d'une liste de **OfferOperationData.**

1.3.1.7. OfferCondition

La class **OfferCondition** permet de gérer les conditions d'une offre.
Cette classe contient 7 méthodes:

- *addOperation* permet d'ajouter une opération dans *_listOfferOperations*
- *removeOperation* permet de retirer une opération dans *_listOfferOperations*
- *setLogicalOperator* permet de modifier l'attribut *_logicalOperator*
- *setName* permet de modifier l'attribut *_name*
- *setDescription* permet de modifier l'attribut *_description*
- *getOfferOperations* permet de récupérer l'attribut *_listOfferOperations* sous forme d'une liste de **OfferOperation.**
- *getLogicalOperator* permet de récupérer l'attribut *_logicalOperator*

1.3.1.8. OfferOperationData

La class **OfferOperationData** est une class abstraite contenant les attributs et les getters nécessaire au bon fonctionnement de la classe **OfferOperation**, étant donnée que cette classe ne contient aucune méthode de modification elle peut être utilisé par l'**UI**.
Elle possède 3 attributs:

- *_value* est la valeur de l'opération qui permettra de valider ou non l'opération.
- *_relationalOperator* est un enum de type **LogicalOperator**, il déterminera l'opération devra être traité avec les autres opérations ou non.
- *_logicalOperator* est un enum de type **RelationalOperator**, il déterminera la façon dont la valeur devra être vérifié, supérieur ou égal à celle donné, inférieur etc.

Cette class est composé de 3 méthodes:

- *getValue* permet de récupérer l'attribut *_value.*
- *getLogicalOperatorValue* permet de récupérer l'attribut *_logicalOperator* sous forme de **String**.
- *getRelationalOperatorValue* permet de récupérer l'attribut *_relationalOperator sous forme de **String**.*

1.3.1.9. OfferCondition

La class **OfferOperation** permet de gérer une opération qui servira à la classe **OfferCondition**.
Cette class est composé de 5 méthodes:

- *setLogicalOperator* permet de modifier l'attribut *_logicalOperator*
- *setRelationalOperator* permet de modifier l'attribut *_relationalOperator*
- *setValue* permet de modifier l'attribut *_value*
- *getLogicalOperator* permet de récupérer l'attribut *_logicalOperator*
- *getRelationalOperator* permet de récupérer l'attribut *_relationalOperator.*

1.3.1.10. Operator

1.3.1.10.1. LogicalOperator

LogicalOperator est un enum, il permet de définir un opérateur logique dans une condition tel que le “ >= “ , “ < ”, “ == “, “ != “ etc.

1.3.1.10.2. RelationalOperator

RelationalOperator est un enum, il permet de définir un opérateur relationnel dans une condition tel que le “ & “ , “ | ”, “ && “ etc.

1.3.2. Save

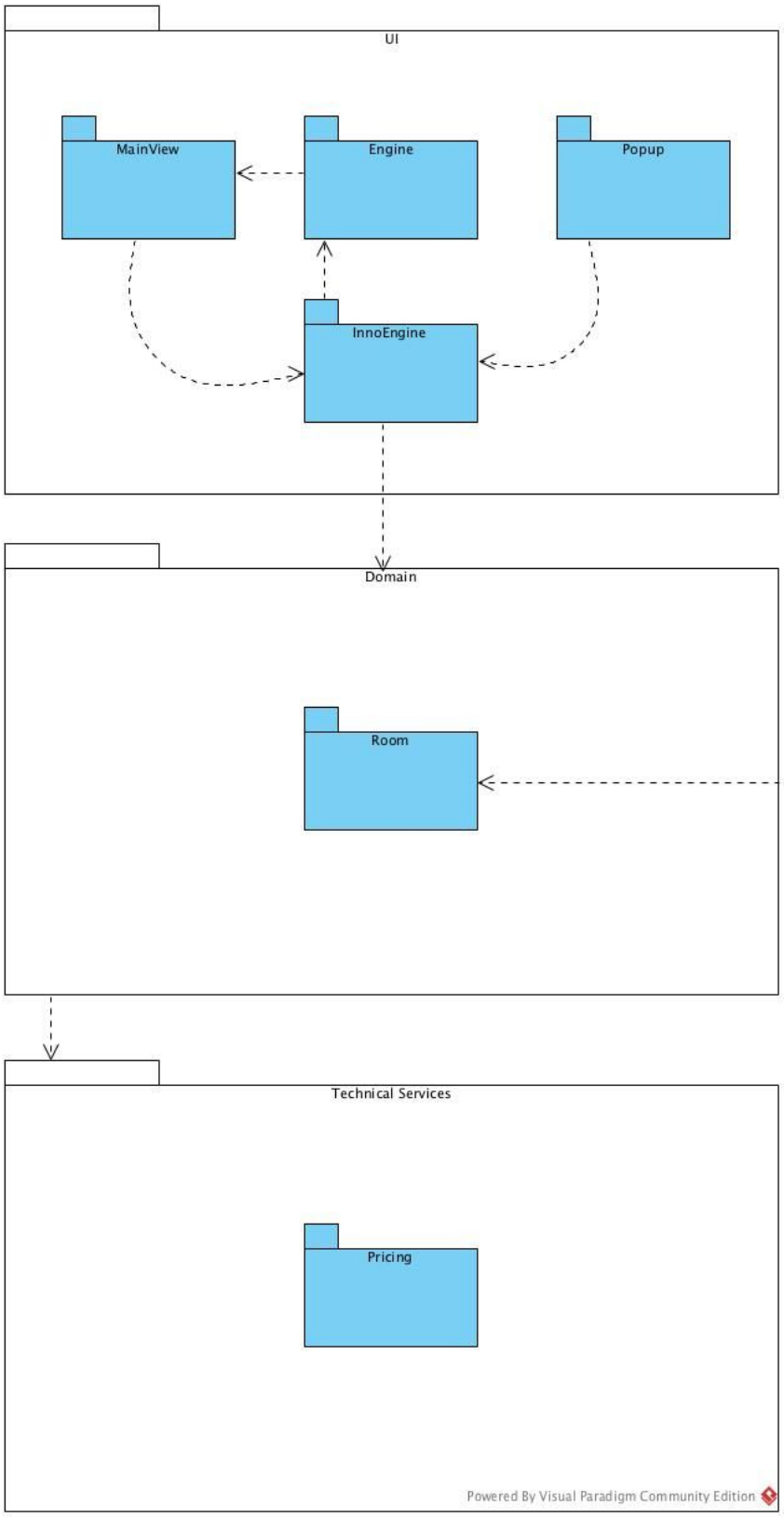
La classe **Save** permet de sauvegarder le plan de salle en cours de réalisation ou à la fin de sa réalisation.
La classe possède 1 attribut :

- *_currentPath* correspond au chemin pour accéder au fichier de sauvegarde.

Elle se compose de 4 méthodes :

- *saveTo* permet de sauvegarder un objet à un endroit précis de l'ordinateur c’est-à-dire à l’emplacement du path désiré.
- *loadFrom* permet de charger un fichier depuis un path et de retourner un objet.
- *getCurrentPath* permet de récupérer le path.
- *setCurrentPath* permet de modifier le path.

2. Explication diagrammes d’architecture logique

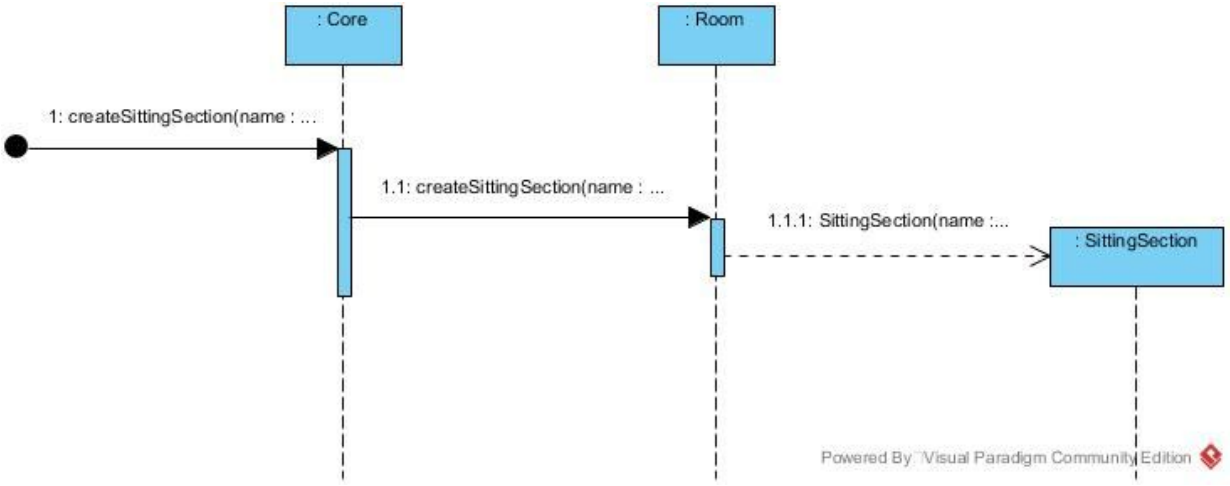


- **package MainView, Popup**: Afin d'effectuer des interactions avec l'Interface Utilisateur les controllers qui se trouves dans **MainView** et **Popup** auront besoin d'interagir avec **InnoEngine**, qui permet de créer une section, déplacer une section, modifier sa taille, en sommes **InnoEngine** va permettre de gérer la retransmission visuel des informations se trouvant dans la couche du Domaine.
- **package InnoEngine**: Ce package va devoir interagir avec **Engine** qui lui permet de faire des actions simples comme créer une forme rectangulaire ou irrégulière avec des points données, il va aussi devoir interagir avec **la couche du Domaine** pour retranscrire les données visuels en données internes comme par exemple lors de la création d'une section, l'**InnoEngine** va gérer l'affichage de cette section mais aussi de retranscrire les données dans **la couche du Domaine**.
- **package Engine:** Celui-ci va devoir interagir avec le package **MainView** afin d'effectuer les actions qui lui aura été demandé, il va par exemple devoir dessiner un polygone afin de représenter une section irrégulière, des rectangles pour la salle ou une section régulière etc.
- **la couche du Domaine** va devoir interagir avec **la couche des Technical Services** par exemple le package **Pricing**, ce package permet d'avoir une gestion des places complètement indépendante du projet en lui même, c'est pour cela qu'il est un service, il ne contient pas des rangées, des sections ou même des sièges, il contient des places. Chaques places ont leurs identifiant unique, et peuvent contenir des offres indépendamment des autres places. Le package **Pricing** permet aussi de créer des offres qui seront par la suite assignable à une place si demandé par l'utilisateur. **La couche du Domaine** (plus spécifiquement le controller de Larman) va devoir interagir avec le package **Room** afin de gérer la salle, ses sections etc.

3. Diagrammes de séquence de conception

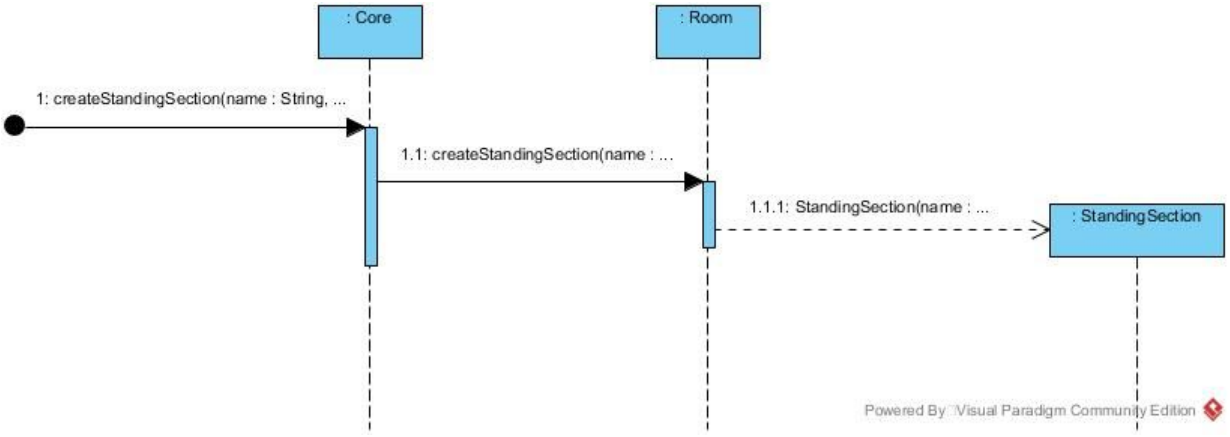
3.1. Création d'une section rectangulaire

3.1.1. Création d'une section rectangulaire assise



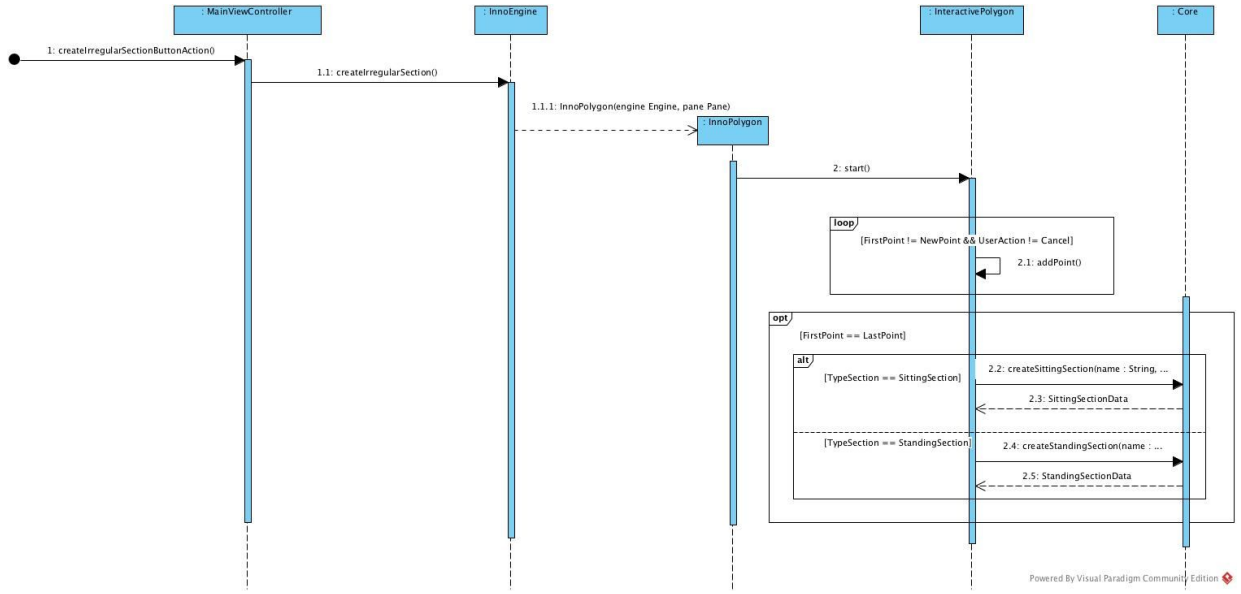
Pour créer une section rectangulaire de type assis, la classe **Core** appelle la méthode *createSittingSection* qui va ensuite permettre d'appeler la méthode *createSittingSection* de la classe **Core** et enfin le constructeur qui va créer l'objet **SittingSection**, *SittingSection* et créant ainsi la section.

3.1.2. Création d'une section rectangulaire debout



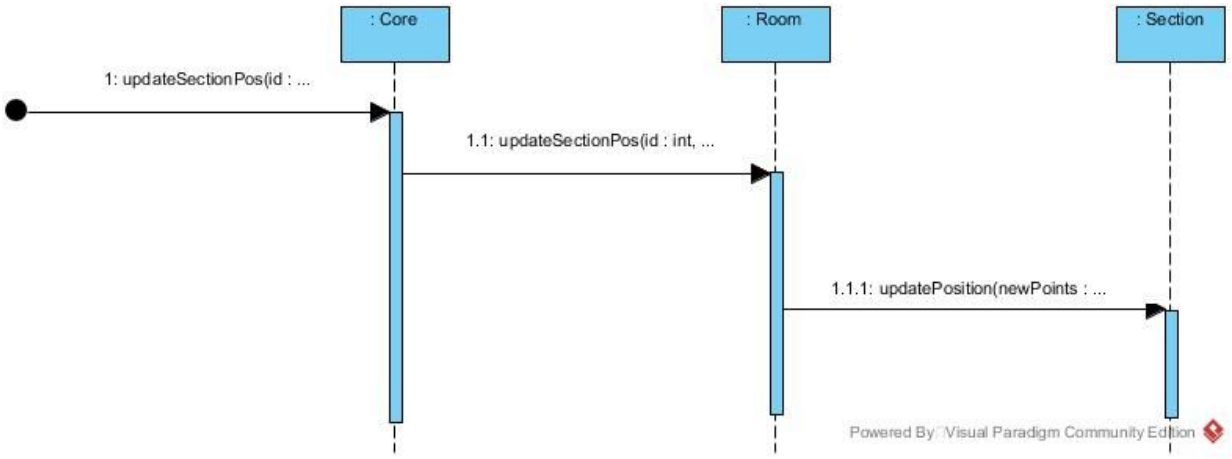
Pour créer une section rectangulaire de type debout, la classe **Core** appelle la méthode *createStandingSection* qui va ensuite appeler la méthode *createStandingSection* de la classe **Core** et enfin le constructeur qui va créer l'objet **StandingSection**, *StandingSection* et créant ainsi la section.

3.2. Création d'une section irrégulière



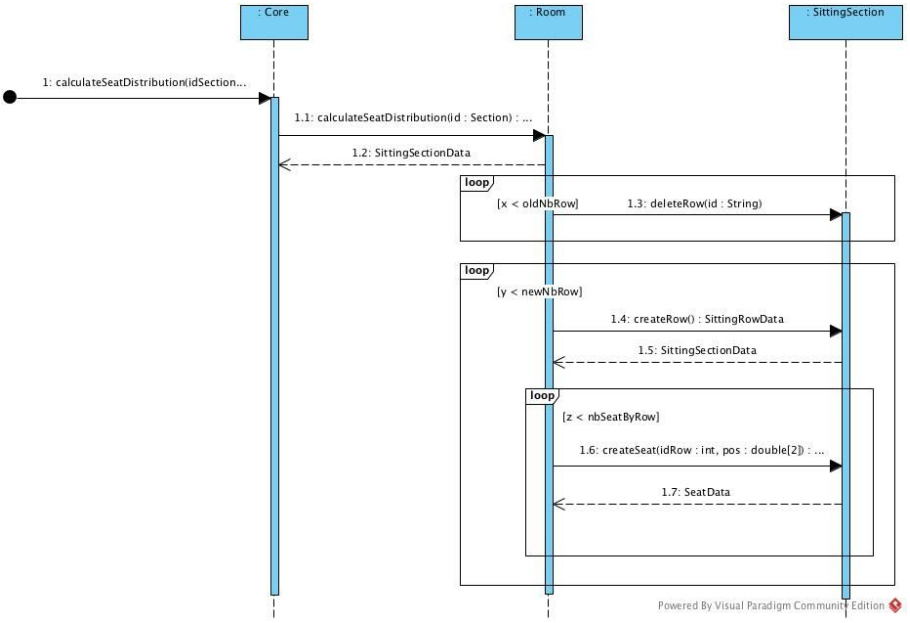
Pour créer une section irrégulière, la classe **MainViewController** appelle la méthode *createIrregularSectionButton* qui va ensuite appeler la méthode *createIrregularSection* de la classe **InnoEngine**. Par la suite, l'objet **InnoPolygon** va être créé par le biai de son constructeur *InnoPylgon* qui va permettre d'appeler la méthode *start* de l'objet **InteractivePolygon.**
A ce moment-là, l'utilisateur va ajouter des points ce qui est récupéré par la méthod *addPoint* de la classe **InteractivePolygon**. Cette action pourra être répétée autant de fois tant que le nouveau point ne correspond pas au premier point et tant que l'utilisateur n'a pas cliqué sur annuler. Par la suite, si le premier point correspond au dernier point ajouté et si le type de section est assis, la classe **Core** va appeler la méthode *createSittingSection* afin de créer la section assise souhaitée et retourner les informations *SittingSectionsData*. Par contre, si le premier point correspond au dernier point ajouté et si le type de section est debout, la classe **Core** va appeler la méthode *createStandingSection* afin de créer la section debout souhaitée et retourner les informations *StandingSectionsData*.

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



Afin de modifier la position d'un point d'une section irrégulière, la classe **Core** va appeler la méthode *updateSectionPos* qui va appeler à son tour la méthode *updateSectionPos* de la classe **Room** pour enfin appeler la méthode *updatePosition* de la classe **Section** qui comme son nom l'indique va mettre à jour les positions des points.

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

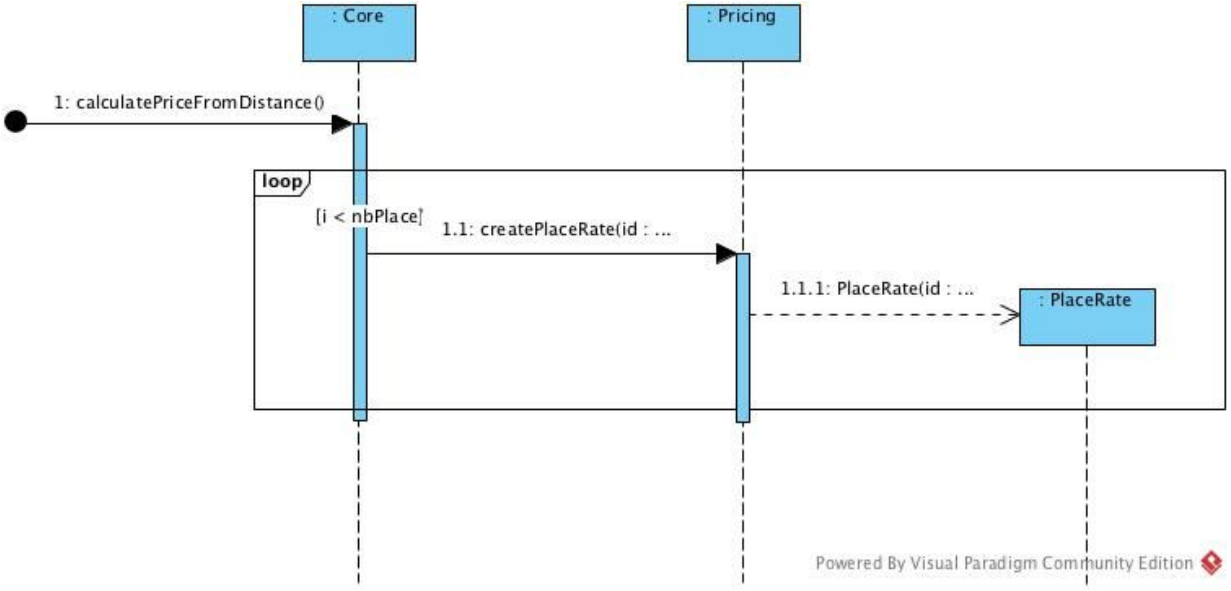


Lorsqu'il y a une modification, la redistribution automatique s'effectue. Tout d'abord, la classe **Core** va appeler la méthode *calculateSeatDistribution*, puis la méthode du même nom mais de la classe **Room** va être appelée et va renvoyer les informations *SittingSectionData*. C'est cette dernière méthode qui va réaliser les calculs permettant de réaliser une nouvelle distribution. Par la suite, avant d'ajouter la nouvelle distribution des sièges, il faut supprimer l'ancienne. Pour cela, la méthode *deleteRow* de la classe **SittingSection** est utilisée pour supprimer une à une les rangées. Après suppression des anciennes rangées, les nouvelles rangées vont pouvoir être ajoutées à l'aide de la méthode *createRow* de la classe **SittingSection** qui renvoit les informations *SittingSectionData*. A chaque ajout d'une rangée, tous les sièges de la rangée vont être ajoutés un par un à l'aide de la méthode *createSeat* de la classe **SittingSection** qui renvoit les informations *SeatData*.

Pseudocode:

```
Si redistribution automatique actif, alors
  X1 := x de la section le plus proche de la scène
  X2 := x de la section le plus loin de la scène
  Y1 := y de la section le plus proche de la scène
  Y2 := y de la section le plus loin de la scène
 a := (Largeur espace vital)/2
  b := (Profondeur espace vital)/2
Pour k de 1 jusqu'à nombre_des_sièges faire
      supprimer siège(i)
Pour i de X1 + a jusqu'à X2 avec un pas de 1 pixel faire
        Pour j de Y1 + b jusqu'à Y2 avec un pas de 1 pixel faire
              Si (point(i+a, j+b) dans section et
                 point(i+a, j+b) dans section et
                 point(i+a, j+b) dans section et
                 point(i+a, j+b) dans section) faire
                 Placer_siege au point (i,j)
                       j += profondeur espace vital – 1pixel
                           Si j > Y2 alors
                              i += largeur espace vital – 1pixel
```

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



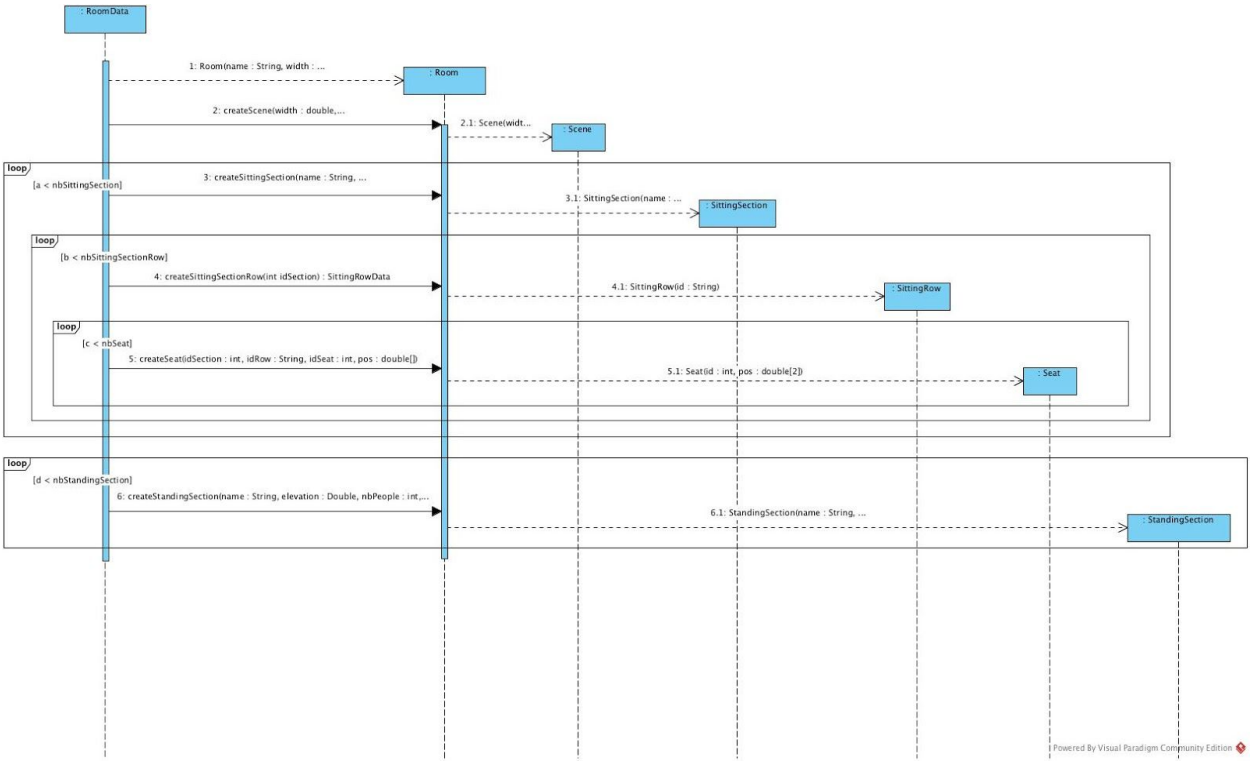
Pour réaliser l'affectation automatique des prix en fonction de la distance avec la scène, la méthode *calculatePriceFromDistance* de la classe **Core** est utilisée. Cette méthode va calculer les prix en fonction de quelques paramètres tels que le prix minimum, le prix maximum et le revenu total souhaité pour l'évènement mais surtout de la distance. Après réalisé les calculs, pour toutes les places, la méthode *createPlaceRate* de la classe **Pricing** va être appelé et le constructeur *PlaceRate* va créer l'objet **PlaceRate**. Chaque objet **PlaceRate** représentera une place avec son prix, prix établi auparavant.

Pseudocode:

```
MinP := Prix minimum
MaxP := Prix maximum
total := Revenu_souhaité – total_déjà_attribué

Si type_attribution == par siège alors
        Limite := nombreSiègesSélectionnés
Si type_attribution == par rangée alors
        Limite := nombreRangéeSélectionnés
Si type_attribution == par section alors
        Limite := nombreSectionsSélectionnés

MaxD := la distance de l'élément le plus loin de la scène
MinD := la distance de l'élément le plus proche de la scène

Fonction Calcul_en_fonction_de_la_distance(éléments_Sélectionnés):
```

```
Pour k de 1 à Limite faire
        Dist(k) := Distance de élément(k) de la scène
        Pourcentage_élément(k) := (MaxD – Dist(k))/(MaxD – MinD)
        Prix_élément(k) := (1-p)*Min+p*Max

Fonction Calcul prix en fonction du revenu(éléments_Sélectionnés):
         Fonction Calcul_en_fonction_de_la_distance(éléments_Sélectionnés)
        diff := (total – total_des_prix )/nombre_d'éléments
        reste := 0
        limites_comptés := 0
Répéter{
        Si diff < 0 alors
        Pour k de nombre_d'éléments a (1+ limites_comptés ) faire
                Si(prix(k) + diff) < Min
                        reste += prix(k) + diff
                prix(k) := Min
                limites_comptés += 1
                Sinon
                Prix(k) += diff
    Si diff > 0 alors
          Pour k de 1 a (nombre_d'éléments - limites_comptés ) faire
                Si(prix(k) + diff) > Max
                        reste += prix(k) - diff
                prix(k) := Max
                limites_comptés += 1
                Sinon
                Prix(k) -= diff
}tant que reste > 0
```

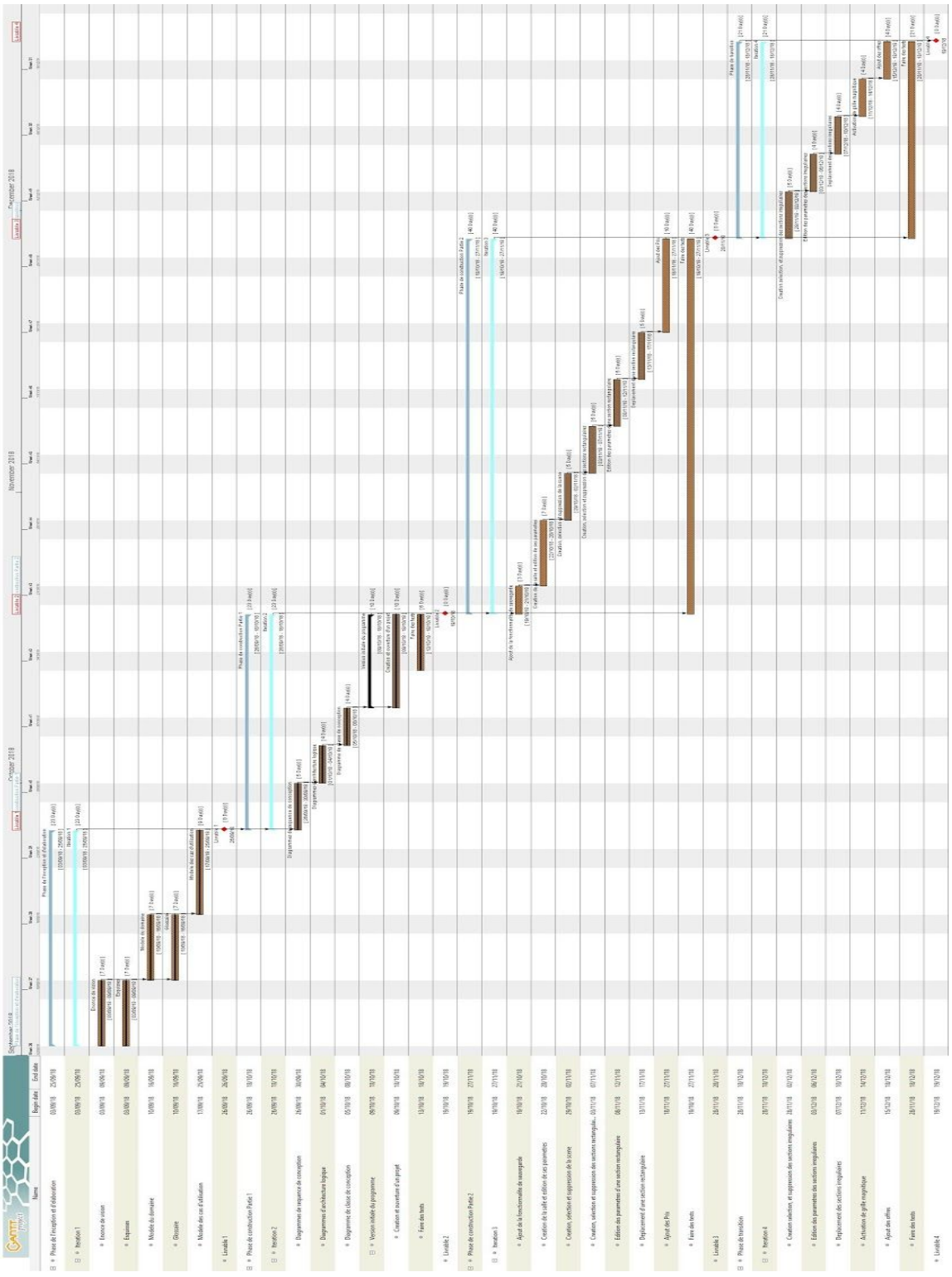
3.6. Synchroniser les objets de l'interface utilisateur avec votre domaine



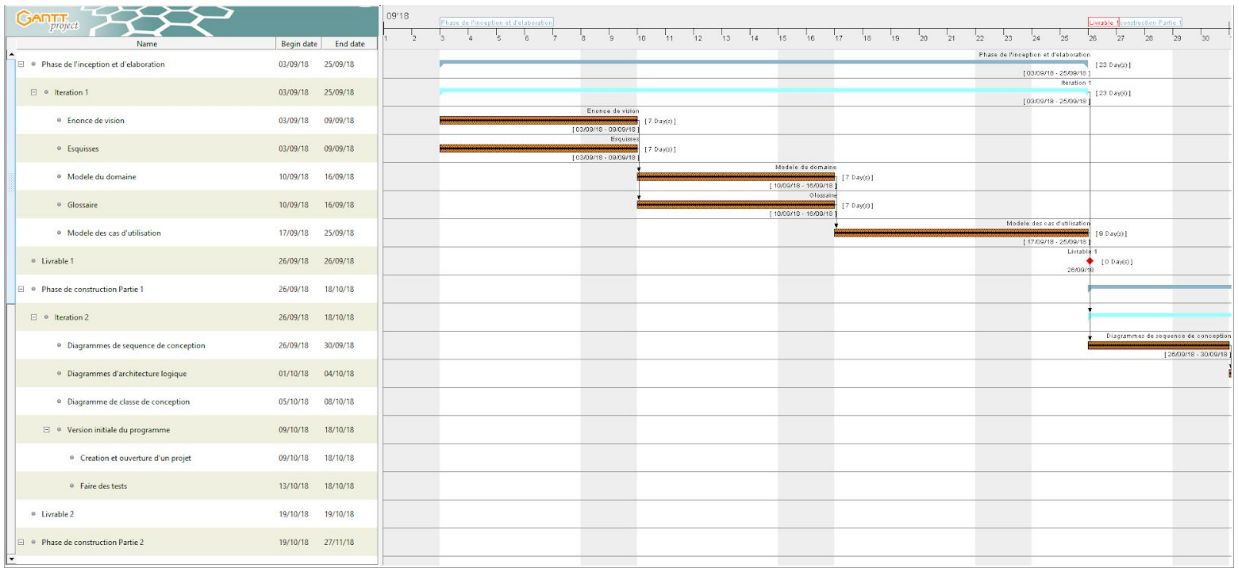
Pour la synchronisation des objets de l'interface utilisateur avec notre domaine lorsqu'un fichier d'être chargé, les informations du fichiers sont tout d'abord récupéré et stocker dans la classe abstraite **RoomData**.
Par la suite, la classe **Room** va être créée à l'aide de son constructeur *Room.*
Room va appeler la méthode *createScene* qui va appeler ensuite le constructeur *Scene* créant ainsi l'objet **Scene**.
Ensuite, chaque section assise présente dans **RoomData**, la méthode *createSittingSection* va être appelée par **Room** qui appelle ensuite le constructeur *SittingSection* créant ainsi l'objet **SittingSection.** Pour chaque section assise, la méthode *createSittingSectionRow* va être appelé par **Room** qui appelle ensuite le constructeur *SittingRow* créant ainsi l'objet **SittingRow**. Cette itération pour la création des rangées sera faite autant de fois qu'il y a de rangée dans la section associée. Enfin, pour chaque rangée, la méthode *createSeat* va être appelée par **Room** qui appelle ensuite le constructeur *Seat* créant ainsi l'objet **Seat**. Cette itération pour la création des sièges sera faite autant de fois qu'il y a de siège dans la rangée associée.
Et, pour chaque section debout présente dans **RoomData**, la méthode *createStandingSection* va être appelée par **Room** qui appelle ensuite le constructeur *StandingSection* créant ainsi l'objet **StandingSection.**

4. Gantt

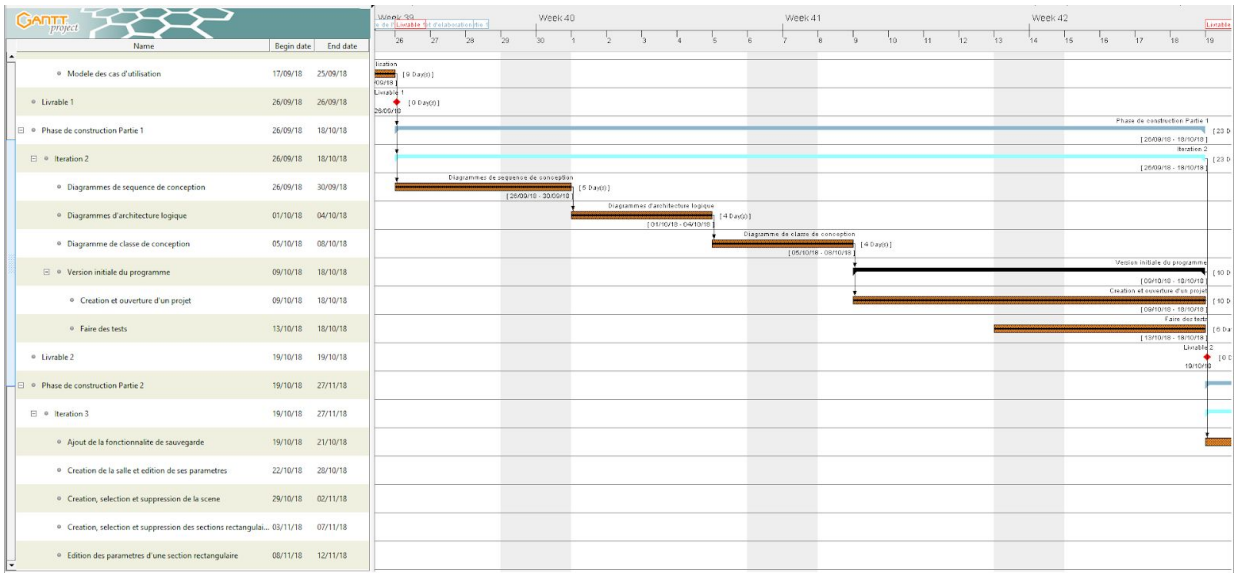
4.1. Plan de travail



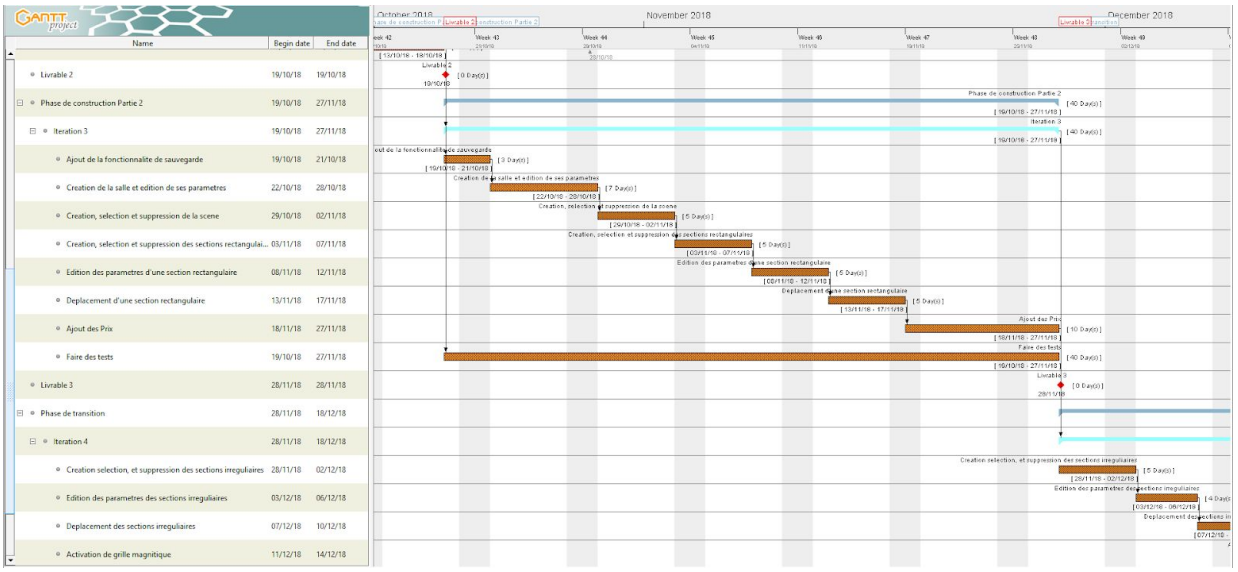
4.2. Itération 1



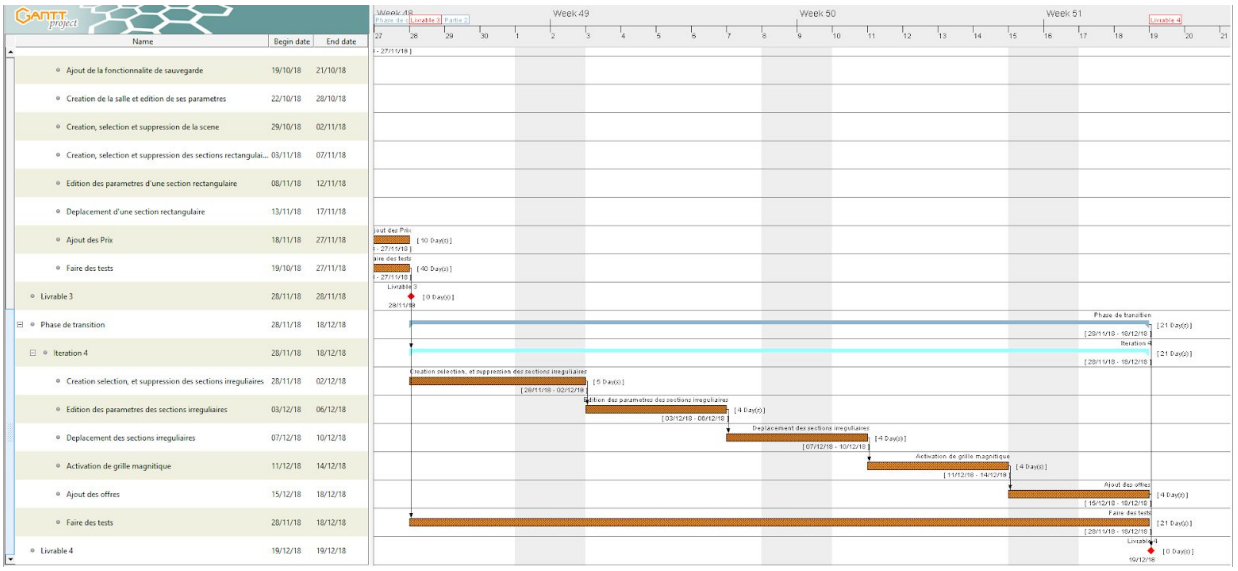
4.3. Itération 2



4.4. Itération 3



4.5. Itération 4



5. Annexes

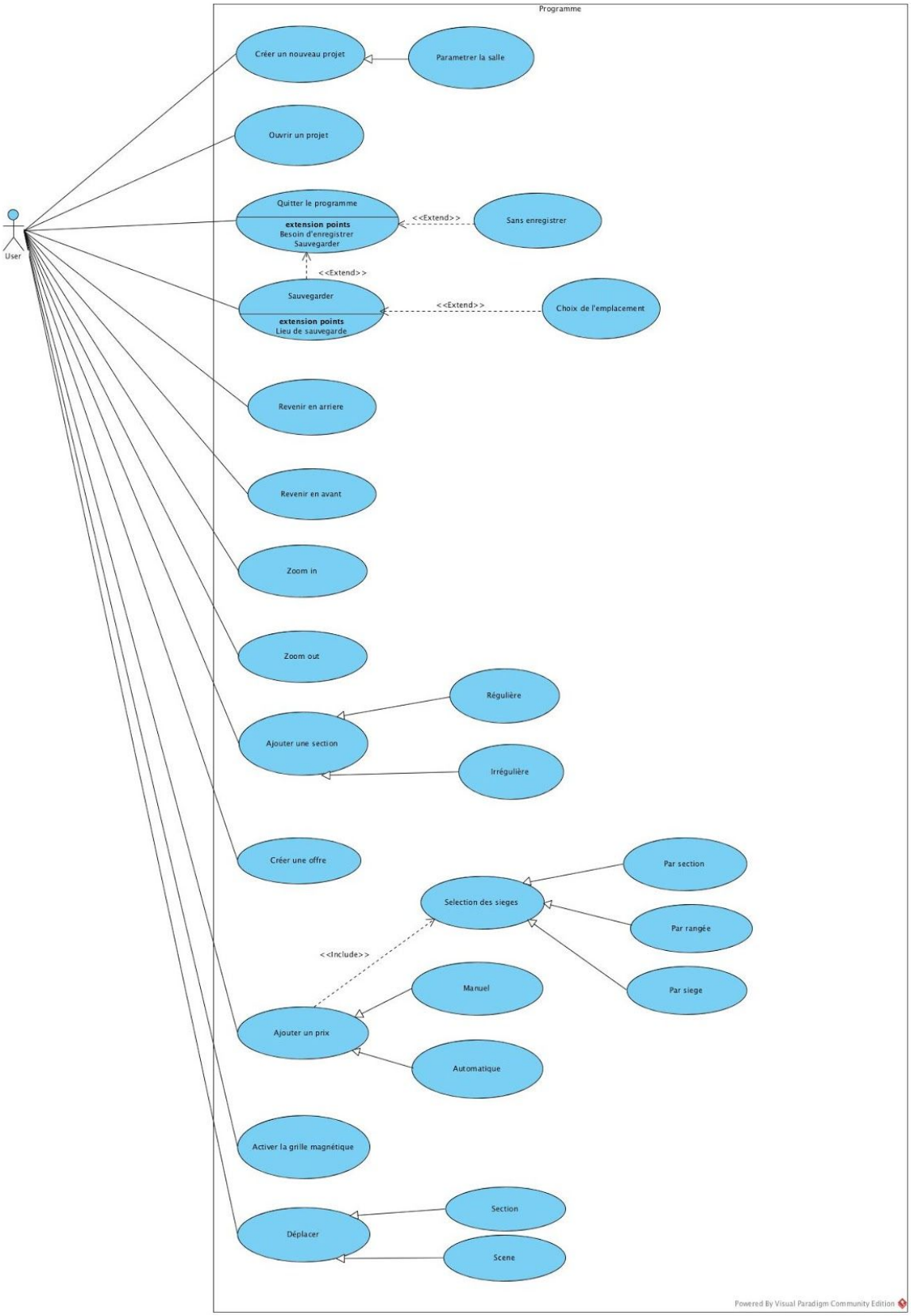
5.1. Énoncé de vision

InnoEvent est une application de création d'événements permettant de définir le plan de salle pour des concerts, des ballets, des matchs de sport par exemple.
L'application permet de configurer le plan de salle de différentes manières grâce à ses différentes fonctionnalités.
Ainsi, il est possible d'ajouter des sections réparties autour d'une unique scène. Ces sections peuvent être des section d'admission générale (debout) ou des sections de siège. Pour ces deux catégories de section, on peut distinguer deux types : les sections régulières correspondant aux sections de forme rectangulaire et les sections irrégulières correspondant aux sections de forme quelconque.
Après avoir représenté votre salle avec la scène et ses multiples sections, différentes options pour les prix sont disponibles.
Tout d'abord, il est possible d'attribuer les prix manuellement en choisissant d'associer un prix à un ou plusieurs sièges, une ou plusieurs rangées voir même une ou plusieurs sections. Pour cette option, il faut indiquer le prix pour un siège et y associer une couleur si le prix n'a jamais été attribué. Ensuite, la distribution des prix peut-être automatisée. Ainsi, il faut juste indiquer le revenu total de la vente et la répartition des prix en fonction de la position du siège par rapport au centre de la scène est réalisée automatiquement.
De plus, pour chaque siège(s), rangée(s) ou section(s), une association à une ou plusieurs offres est réalisable. Ces offres peuvent être créées, modifiées, ajoutées, effacées par l'utilisateur. L'utilisateur choisit l'attribution de ces offres en fonction de ses envies et du revenu total souhaité.
Cette application permet donc d'avoir un plan de salle conforme à la réalité avec une disposition précise de nos éléments, une répartition des prix modulables selon nos souhaits et une gestion d'offres efficaces.

5.2. Modèle du domaine

A FAIRE APRÈS QUE LE DIAGRAMME DE CLASSES DE CONCEPTION SOIT FINI

5.3. Modèle des cas d'utilisation



5.3.1. Abrégé

Cas d'utilisation	Sauvegarder
Acteur (s)	Utilisateur
Type	Primaire
Description	L'utilisateur clique sur le bouton "Sauvegarder", rentre le nom du fichier, choisit l'emplacement où va être enregistrer le fichier et clique sur le bouton "Sauvegarder".

Cas d'utilisation	Revenir en arrière
Acteur (s)	Utilisateur
Type	Secondaire
Description	L'utilisateur clique sur le bouton "Annuler".

Cas d'utilisation	Revenir en avant
Acteur (s)	Utilisateur
Type	Secondaire
Description	L'utilisateur clique sur le bouton "Rétablir".

Cas d'utilisation	Zoomer en avant
Acteur (s)	Utilisateur
Type	Secondaire
Description	L'utilisateur clique sur le bouton "Zoom avant".

Cas d'utilisation	Zoomer en arrière
Acteur (s)	Utilisateur
Type	Secondaire
Description	L'utilisateur clique sur le bouton "Zoom arrière".

Cas d’utilisation	Activer la grille magnétique
Acteur (s)	Utilisateur
Type	Secondaire
Description	L'utilisateur clique sur le bouton “Activer la grille magnétique”.

Cas d’utilisation	Déplacer une section
Acteur (s)	Utilisateur
Type	Secondaire
Description	L'utilisateur clique sur la section qu’il souhaite déplacer et réalise un cliquer-déposer vers l’endroit souhaité.

Cas d’utilisation	Déplacer la scène
Acteur (s)	Utilisateur
Type	Secondaire
Description	L'utilisateur clique sur la scène et réalise un cliquer-déposer vers l’endroit souhaité.

Cas d’utilisation	Créer une offre
Acteur (s)	Utilisateur
Type	Secondaire
Description	L'utilisateur clique sur le bouton “Prix”, clique sur “Gestion d’offres”, crée l’offre souhaité et clique sur “Créer offre”.

5.3.2. Détaillé

Cas d'utilisation	Créer un nouveau projet
Système	Programme
Acteur (s)	Utilisateur
Parties prenantes et intérêts	Utilisateur: Il désire créer un nouveau projet.
Préconditions	Le logiciel est ouvert.
Garanti en cas de succès	Le logiciel crée le projet.
Scénario principal	1. L'utilisateur clique sur le bouton "créer un nouveau projet". 2. Le système ouvre une popup d'option de création du nouveau projet comprenant les paramètres de la taille de la salle ainsi que celle de la scène. 3. L'utilisateur saisit le nom du projet, la taille de la salle, la taille de la scène et l'espace vital par défaut. 4. L'utilisateur valide son choix. 5. Le système crée la salle ainsi que la scène.
Scénario alternatif	Ligne 4 : L'utilisateur annule son choix. Le système retourne sur le menu précédent.

Cas d'utilisation	Ouvrir un projet
Système	Programme
Acteur (s)	Utilisateur
Parties prenantes et intérêts	Utilisateur: Il désire ouvrir un nouveau projet.
Préconditions	Le logiciel est ouvert.
Garanti en cas de succès	Le logiciel charge le projet.
Scénario principal	1. L'utilisateur clique sur le bouton "ouvrir un projet". 2. Le système ouvre une popup de navigation parmi les dossiers de l'utilisateur pour le projet à ouvrir.

	3. L’utilisateur choisit le fichier contenant le projet dont il souhaite charger le contenu. 4. L’utilisateur valide son choix. 5. Le système ouvre le projet désiré.
Scénario alternatif	Ligne 4 : L’utilisateur annule son choix. Le système retourne sur le menu précédent.

Cas d’utilisation	Créer une section de sièges de type régulier
Système	Programme
Acteur (s)	Utilisateur
Parties prenantes et intérêts	Utilisateur: Il désire créer une nouvelle section de siège régulière.
Préconditions	Le logiciel est ouvert et un projet est chargé.
Garanti en cas de succès	La section est créée.
Scénario principal	1. L'utilisateur clique sur le bouton “créer une section de type régulière”. 2. Le système sélectionne l’outil de création de section régulière. 3. L’utilisateur clique à l’emplacement désiré. 4. Le système ouvre une popup avec les choix du nombre de rangées ainsi que le nombre de sièges par rangées. 5. L’utilisateur définit le nombre de rangées et le nombre de sièges par rangées souhaités. 6. L’utilisateur valide son choix. 7. Le système crée la section à l’emplacement désiré.
Scénario alternatif	Ligne 6 : L’utilisateur annule son choix. Le système annule l’action précédente.

Cas d’utilisation	Créer une section de sièges de type irrégulier
Système	Programme
Acteur (s)	Utilisateur
Parties prenantes et intérêts	Utilisateur: Il désire créer une nouvelle section de siège irrégulière.

Préconditions	Le logiciel est ouvert et un projet est chargé.
Garanti en cas de succès	La section est créée.
Scénario principal	1. L'utilisateur clique sur le bouton “créer une section de type irrégulière”. 2. Le système sélectionne l’outil de création de section irrégulière. 3. L’utilisateur clique aux différents emplacements des points en reliant le premier au dernier à la fin. 4. Le système crée la section et attribue automatiquement les sièges.
Scénario alternatif	Ligne 3 : L’utilisateur annule la sélection. Le système annule l’action précédente.

Cas d’utilisation	Créer une section d’admission générale de type irrégulière
Système	Programme
Acteur (s)	Utilisateur
Parties prenantes et intérêts	Utilisateur: Il désire créer une nouvelle section d’admission générale de type irrégulière.
Préconditions	Le logiciel est ouvert et un projet est chargé.
Garanti en cas de succès	La section est créée.
Scénario principal	1. L'utilisateur clique sur le bouton “créer une nouvelle section de type irrégulière”. 2. Le système sélectionne l’outil de création de section irrégulière. 3. L’utilisateur clique aux différents emplacements des points en reliant le premier au dernier à la fin. 4. Le système crée la section. 5. L’utilisateur coche la case “Section d’admission générale”. 6. Le système change le type de la section à section d’admission générale.
Scénario alternatif	

Cas d’utilisation	Créer un prix

Système	Programme
Acteur (s)	Utilisateur
Parties prenantes et intérêts	Utilisateur: Il désire attribuer un prix à des sièges, rangées ou sections.
Préconditions	Une ou plusieurs sections contenant des siège sont créées.
Garanti en cas de succès	Le prix demandé est attribué aux sièges sélectionnés.
Scénario principal	1. L'utilisateur clique sur le bouton “créer un nouveau prix manuel”. 2. Le système ouvre une popup pour le choix du prix. 3. L’utilisateur sélectionne les sections, rangées, sièges auxquels il souhaite attribuer le prix. 4. Le système affiche un indicateur visuel indiquant les sections, rangées, sièges sélectionnés. 5. L’utilisateur définit le prix des sièges, choisit la couleur et valide son choix. 6. Le système attribue les prix et la couleur à la sélection.
Scénario alternatif	Ligne 5 : L’utilisateur annule son choix. Le système annule toute action précédente.

Cas d’utilisation	Quitter le programme
Système	Programme
Acteur (s)	Utilisateur
Parties prenantes et intérêts	Utilisateur: Il désire quitter le programme.
Préconditions	Le programme est ouvert.
Garanti en cas de succès	Le programme se quitte.
Scénario principal	1. L'utilisateur clique sur le bouton “quitter le programme”. 2. Le système ouvre une popup avec la possibilité de quitter avec sauvegarde, de quitter sans sauvegarde et d’annuler. 3. L’utilisateur clique sur “Quitter en

	sauvegardant”. 4. Le système ouvre une popup de navigation parmi les dossiers de l’utilisateur. 5. L’utilisateur choisit l'endroit où il souhaite sauvegarder ses modifications. 6. Le système sauvegarde le fichier et ses modifications à l’endroit désiré. 7. Le système ferme le programme.
Scénario alternatif	Ligne 3-a : L'utilisateur clique sur le bouton, “Quitter sans sauvegarde”. Le scénario reprend à l’étape 7. Ligne 3-b : L’utilisateur clique sur “Annuler”. Le système annule l’action précédente.

5.3.3. Deux colonnes

5.3.3.1. Rappel

1. L'utilisateur lance le programme. 3. L'utilisateur crée un nouveau projet. 5. L'utilisateur rentre le nom de la salle et ses dimensions, ainsi que les dimensions de la scène. 7. L'utilisateur positionne la scène où il le désire en la déplaçant. 9. L'utilisateur ajoute les sections. 11. L'utilisateur assigne des prix. 13. L'utilisateur assigne les offres. 15. L'utilisateur sauvegarde.	2. Possibilité de charger ou de créer un nouveau. 4. Popup d'options de création du nouveau projet. 6. Création de la salle et de la scène. 8. Mise à jour de la salle en fonction de l'emplacement de la scène. 10. Création des sections. 12. Création de prix. 14. Mise à jour des offres. 16. Sauvegarde du projet à l'emplacement désiré.

5.3.3.2. Détaillé

Cas d'utilisation	Créer un nouveau projet
Système	Programme
Acteur (s)	Utilisateur
Parties prenantes et intérêts	Utilisateur: Il désire créer un nouveau projet.
Préconditions	Le logiciel est ouvert.
Garanti en cas de succès	Le logiciel crée le projet.
Scénario principal	1. L'utilisateur clique

	sur le bouton “créer un nouveau projet”. 2. Le système ouvre une popup d’option de création du nouveau projet comprenant les paramètres de la taille de la salle ainsi que celle de la scène. 3. L’utilisateur saisit le nom du projet, la taille de la salle, la taille de la scène et l’espace vital par défaut. 4. L’utilisateur valide son choix. 5. Le système crée la salle ainsi que la scène.
Scénario alternatif	Ligne 4 : L’utilisateur annule son choix. Le système retourne sur le menu précédent.

Cas d’utilisation	Ouvrir un projet
Système	Programme
Acteur (s)	Utilisateur
Parties prenantes et intérêts	Utilisateur: Il désire ouvrir un nouveau projet.
Préconditions	Le logiciel est ouvert.
Garanti en cas de succès	Le logiciel charge le projet.
Scénario principal	1. L'utilisateur clique sur le bouton “ouvrir un projet”. 2. Le système ouvre une popup de navigation parmi les dossiers de l’utilisateur pour le projet à ouvrir. 3. L’utilisateur choisit le fichier contenant le projet dont il souhaite charger le

	contenu. 4. L'utilisateur valide son choix. 5. Le système ouvre le projet désiré.
Scénario alternatif	Ligne 4 : L'utilisateur annule son choix. Le système retourne sur le menu précédent.

Cas d'utilisation	Créer une section de sièges de type régulier
Système	Programme
Acteur (s)	Utilisateur
Parties prenantes et intérêts	Utilisateur: Il désire créer une nouvelle section de siège régulière.
Préconditions	Le logiciel est ouvert et un projet est chargé.
Garanti en cas de succès	La section est créée.
Scénario principal	1. L'utilisateur clique sur le bouton "créer une section de type régulière". 2. Le système sélectionne l'outil de création de section régulière. 3. L'utilisateur clique à l'emplacement désiré. 4. Le système ouvre une popup avec les choix du nombre de rangées ainsi que le nombre de sièges par rangées. 5. L'utilisateur définit le nombre de rangées et le nombre de sièges par rangées souhaités. 6. L'utilisateur valide son choix. 7. Le système crée la

	section à l'emplacement désiré.
Scénario alternatif	Ligne 6 : L'utilisateur annule son choix. Le système annule l'action précédente.

Cas d'utilisation	Créer une section de sièges de type irrégulier
Système	Programme
Acteur (s)	Utilisateur
Parties prenantes et intérêts	Utilisateur: Il désire créer une nouvelle section de siège irrégulière.
Préconditions	Le logiciel est ouvert et un projet est chargé.
Garanti en cas de succès	La section est créée.
Scénario principal	1. L'utilisateur clique sur le bouton "créer une section de type irrégulière". 2. Le système sélectionne l'outil de création de section irrégulière. 3. L'utilisateur clique aux différents emplacements des points en reliant le premier au dernier à la fin. 4. Le système crée la section et attribue automatiquement les sièges.
Scénario alternatif	Ligne 3 : L'utilisateur annule la sélection. Le système annule l'action précédente.

Cas d'utilisation	Créer une section d'admission générale de type irrégulière
Système	Programme

Acteur (s)	Utilisateur
Parties prenantes et intérêts	Utilisateur: Il désire créer une nouvelle section d'admission générale de type irrégulière.
Préconditions	Le logiciel est ouvert et un projet est chargé.
Garanti en cas de succès	La section est créée.
Scénario principal	1. L'utilisateur clique sur le bouton "créer une nouvelle section de type irrégulière". 2. Le système sélectionne l'outil de création de section irrégulière. 3. L'utilisateur clique aux différents emplacements des points en reliant le premier au dernier à la fin. 4. Le système crée la section. 5. L'utilisateur coche la case "Section d'admission générale". 6. Le système change le type de la section à section d'admission générale.
Scénario alternatif	

Cas d'utilisation	Créer un prix
Système	Programme
Acteur (s)	Utilisateur
Parties prenantes et intérêts	Utilisateur: Il désire attribuer un prix à des sièges, rangées ou sections.
Préconditions	Une ou plusieurs sections contenant des siège

	sont créées.
Garanti en cas de succès	Le prix demandé est attribué aux sièges sélectionnés.
Scénario principal	1. L'utilisateur clique sur le bouton “créer un nouveau prix manuel”. 2. Le système ouvre une popup pour le choix du prix. 3. L’utilisateur sélectionne les sections, rangées, sièges auxquels il souhaite attribuer le prix. 4. Le système affiche un indicateur visuel indiquant les sections, rangées, sièges sélectionnés. 5. L’utilisateur définit le prix des sièges, choisit la couleur et valide son choix. 6. Le système attribue les prix et la couleur à la sélection.
Scénario alternatif	Ligne 5 : L’utilisateur annule son choix. Le système annule toute action précédente.

Cas d’utilisation	Quitter le programme
Système	Programme
Acteur (s)	Utilisateur
Parties prenantes et intérêts	Utilisateur: Il désire quitter le programme.
Préconditions	Le programme est ouvert.
Garanti en cas de succès	Le programme se quitte.
Scénario principal	1. L'utilisateur clique sur le bouton

	“quitter le programme”. 2. Le système ouvre une popup avec la possibilité de quitter avec sauvegarde, de quitter sans sauvegarde et d’annuler. 3. L’utilisateur clique sur “Quitter en sauvegardant”. 4. Le système ouvre une popup de navigation parmi les dossiers de l’utilisateur. 5. L’utilisateur choisit l'endroit où il souhaite sauvegarder ses modifications. 6. Le système sauvegarde le fichier et ses modifications à l’endroit désiré. 7. Le système ferme le programme.
Scénario alternatif	Ligne 3-a : L'utilisateur clique sur le bouton, “Quitter sans sauvegarde”. Le scénario reprend à l’étape 7. Ligne 3-b : L’utilisateur clique sur “Annuler”. Le système annule l’action précédente.

5.4. Glossaire

Phase : Une étape regroupant un certain nombre d'itérations qui correspondent au même thème de travail.
Itération : est une répétition d'une séquence d'instructions ou d'une partie de programme un nombre de fois fixé à l'avance ou tant qu'une condition définie n'est pas remplie.
Diagramme de Gantt : Un diagramme permettant de visualiser dans le temps les diverses tâches composant un projet.
Classes conceptuelles : Les idées ou Les objets du système selon la manière dont on les visualise dans des situations réelles.
Classe : Une collection d'objets ayant les mêmes responsabilités et comportement. Une classe est représentée par une boite avec un nom, des attributs et des méthodes.
Attribut : une valeur nommée qu'un objet ou une classe est responsable de maintenir. Ils sont représentés dans le premier compartiment sous le nom de la classe.
Méthode : une fonctionnalité qu'une classe sait comment faire. Elles sont représentées dans le dernier compartiment de la classe.
Paramètre : Une valeur fournie lors de la génération d'une instance de certaines classes, ou pour l'utilisation de certaines méthodes. Ils sont représentés dans le nom de la méthode entre des parenthèses en spécifiant leurs noms et leurs types.
Constructeur : Une méthode particulière qui crée un nouvel objet de la classe correspondante. Des paramètres peuvent être requis selon le constructeur.
Héritage : la propriété qui fait bénéficier une sous-classe de la structure et du comportement de sa surclasse.
Encapsulation : consiste à masquer la structure et le comportement internes et propres au fonctionnement de l'objet.
Objet : est une instance unique d'une classe qui occupe un espace de stockage.
Package : est un dossier regroupant un ensemble de classes et d'autres packages qui sont liés entre eux sémantiquement.
Couche : Un ensemble qui regroupe les packages et les classes entre trois groupes selon leurs rôles et relations (Couche service, couche domaine et coche Interface) et qui est responsable d'un aspect majeur du fonctionnement du système.
Architecture logique : L'organisation des classes du système en des packages et des couches selon la logique.
Couche de service : Regroupe les packages et les classes qui sont à usage général et peuvent être utilisés indépendamment du système actuel et implémentés selon le besoin.
Couche de domaine : Regroupe les packages et les classes appartenant au fonctionnement interne du système comme les calculs et la logique applicative.
Couche Interface : Regroupe les packages et les classes qui servent à représenter les interfaces utilisateur et faire le lien entre les actions utilisateur et la couche domaine.
Interface Utilisateur : L'ensemble des interfaces intuitives comme les fenêtres, qui permettent à un utilisateur d'interagir avec le système et qui font le lien Homme-machine.
Pop-up : Une fenêtre spécifique qui s'affiche à la suite d'un évènement particulier afin d'obtenir un certain résultat.
Évènement : Une action particulière comme un clic de souris ou pression d'un bouton.

Contrôleur : Le premier objet de la couche Domaine qui interagit avec la couche Interface et qui délègue les tâches aux autres objets. Dans le cas de ce projet on le nomme Core.
Diagramme de séquence de conception : représente les interactions entre les objets et le séquencement des flots de contrôle.
Diagramme de classe : un diagramme représentant les classes et leurs associations.
Modèle de domaine : La représentation visuelle des classes conceptuelles et leurs associations.
Cas d'utilisation : un ensemble de scénarios partageant des acteurs similaires et qui produisent des résultats spécifiques. Ils sont documentés par un texte en langage naturel et représentés sous la forme d'icônes ovales.
Acteur : Une entité qui déclenche des scénarios et en obtient des résultats. Il est représenté par un bonhomme bâton.
Scénario : La description de la manière dont un acteur particulier obtient un résultat spécifique.
Enum : Un type de donnée qu'on définit en listant ses valeurs possibles.
Loop : Une boucle qu'on itère un certain nombre de fois tant que la condition de garde est vérifiée.
Alt : Plusieurs alternatives, chacune avec une condition de garde . Une seule alternative peut être vraie.
Opt : Une alternative simple avec une condition de garde.
Type : peut être une classe ou un type standard. Les types peuvent être: Integer pour des entiers, String pour des chaînes de caractères, Boolean pour des booléens, des double pour des réels.
Path : chemin d'accès d'un fichier ou d'un répertoire