

# LIVRABLE 2

VisuaLigue

À l'intention de:

**Martin Savoie**  
**Maxime Charron**  
**Jean Bouchard**

Maxime Gagnon-Legault (MAGAL106)  
Guillaume Manseau (GUMAN8)  
Olivier Robert (OLROB13)  
Charles-Hubert Van Eyll (CHVAE1)



2016/11/06

## TABLE DES MATIÈRES

<b>1. Programmation de l'interface statique.....</b>	<b>4</b>
1.1. Séparation du travail.....	4
1.2. Description du répertoire de remise .....	4
1.3. Écrans de l'interface (implémentation réelle) .....	5
1.3.1. Éditeur de jeu .....	5
1.3.2. Liste des jeux .....	6
1.3.3. Gestion des sports .....	7
1.3.4. Création/modification d'un sport .....	8
1.3.5. Gestion des obstacles .....	9
1.3.6. Paramètres .....	10
1.4. Fonctionnalités implémentées.....	10
1.5. Fonctionnalités non implémentées.....	11
1.6. Librairies.....	11
1.7. Exécutable .....	11
<b>2. Diagrammes de classe de conception.....</b>	<b>12</b>
2.1. Diagramme de classes du domaine .....	13
2.2. Diagramme de classes des services.....	15
2.3. Diagramme de classes de la persistance .....	17
2.4. Diagramme de classes de l'interface utilisateur .....	18
<b>3. Diagramme de package .....</b>	<b>20</b>
<b>4. Diagramme de séquence de conception .....</b>	<b>22</b>
4.1. Conversion de coordonnées.....	23
4.2. Ajout d'un joueur.....	25
4.3. Déterminer clic sur un joueur .....	27
4.4. Édition image par image .....	29
4.5. Édition temps réel.....	31
4.6. Visionnement du jeu.....	33
<b>5. Diagramme de Gantt (mis à jour) .....</b>	<b>35</b>
5.1. Itération 1.....	36
5.2. Itération 2 .....	37
5.3. Itération 3 .....	38

5.4.	Itération 4.....	39
<b>6.</b>	<b>Annexe : Livrable 1.....</b>	<b>40</b>
6.1.	Vision .....	40
6.1.1.	Introduction .....	40
6.1.2.	Positionnement.....	40
6.1.3.	Descriptions des intervenants.....	41
6.1.4.	Hypothèses et contraintes.....	41
6.1.5.	Coût, prix, licences et installation.....	41
6.1.6.	Résumé des caractéristiques du système.....	42
6.1.7.	Solution proposée.....	42
6.2.	Modèle du domaine .....	43
6.3.	Cas d'utilisation (mis à jour) .....	44
6.3.1.	Diagramme des cas d'utilisation .....	44
6.3.2.	Texte des cas d'utilisation .....	45
6.4.	Glossaire (mis à jour) .....	52
6.4.1.	Correspondance anglophone et francophone des termes .....	52
6.4.2.	Définitions .....	54

# 1. PROGRAMMATION DE L'INTERFACE STATIQUE

## 1.1. SÉPARATION DU TRAVAIL

La programmation statique de l'interface a été programmée par l'étudiant en génie logiciel 4<sup>e</sup> année de cette équipe. Tel qu'entendu avec le professeur, pour des raisons évidentes, les membres de l'équipe en 2<sup>ème</sup> année de génie logiciel devaient produire ce qui devrait normalement être produit à ce stade-ci du projet pour des GLO 2<sup>e</sup> année, c'est-à-dire la programmation de 2 maquettes chaque afin d'apprendre les concepts Java et JavaFX. C'est cette contribution que vous devez évaluer **et non le nombre de ligne de code dans Git.**

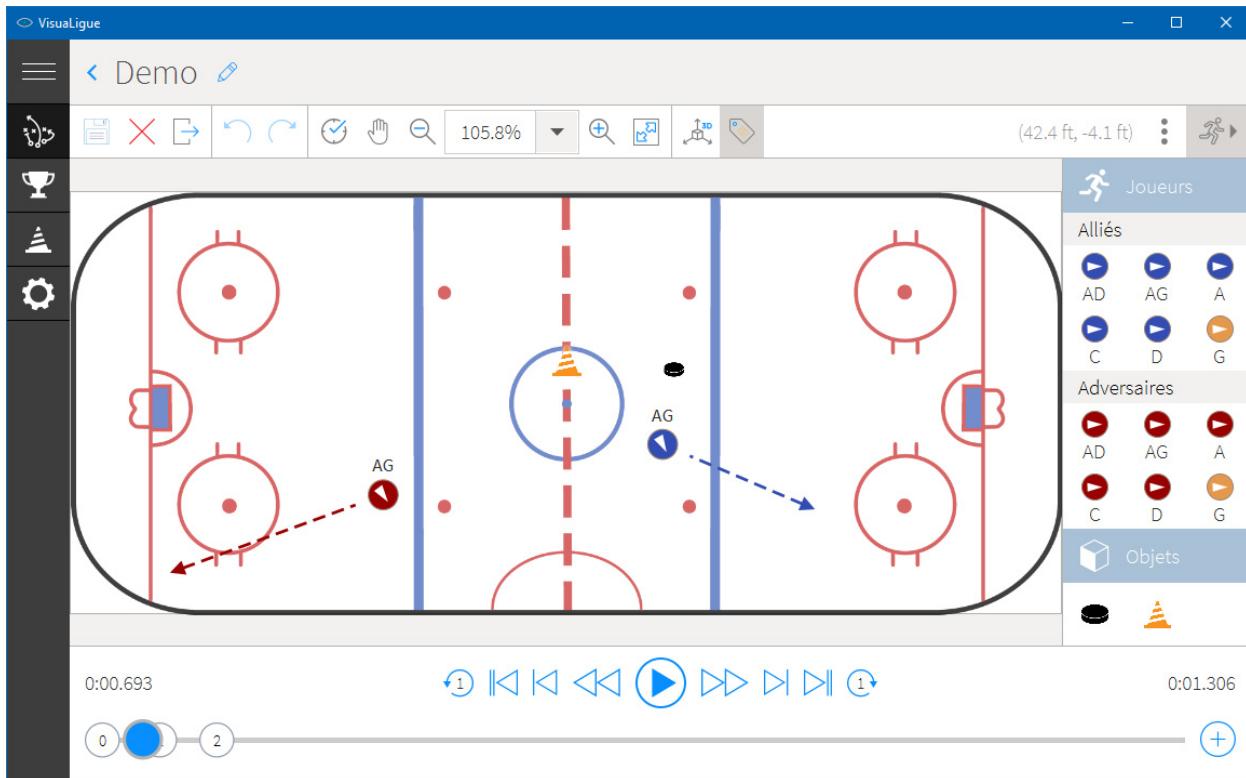
## 1.2. DESCRIPTION DU RÉPERTOIRE DE REMISE

### !IMPORTANT!

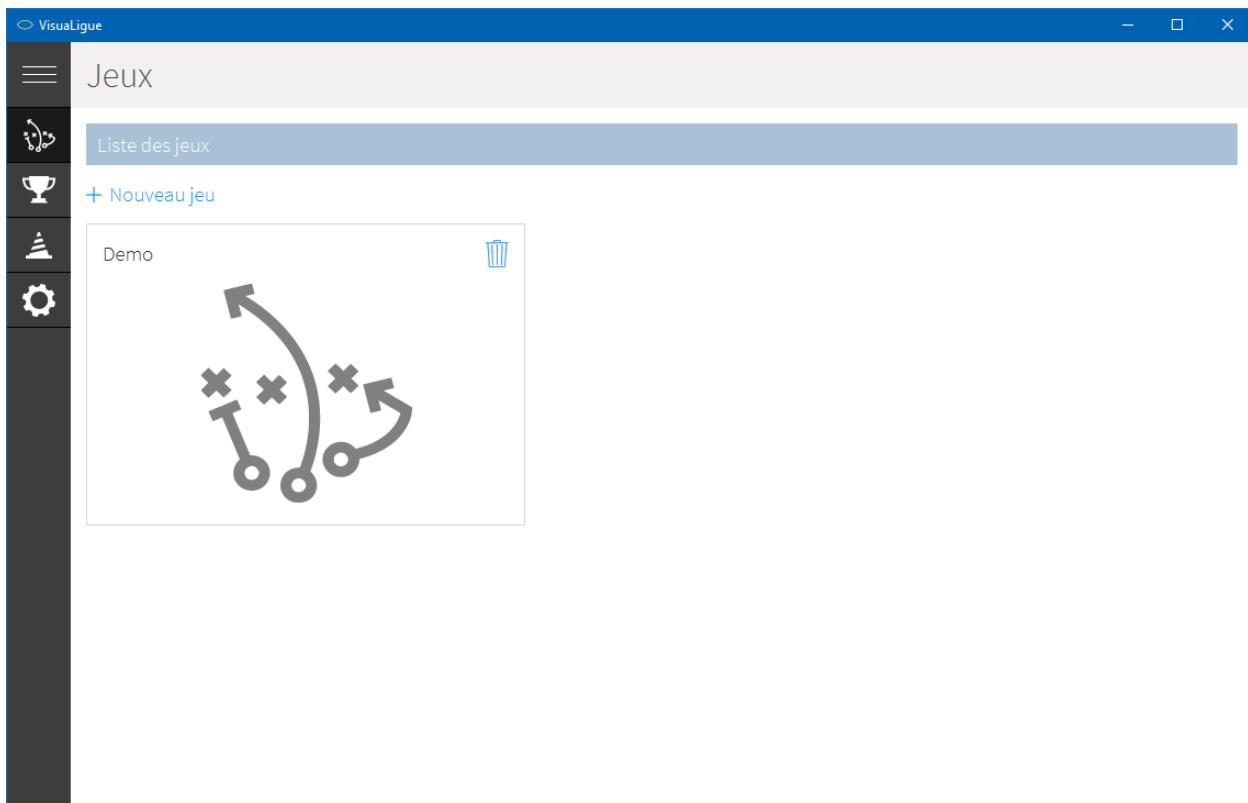
- **/Livrable 1** : Document originaux du livrable 1 (diagramme de Gantt, rapport complet et diagrammes Visual Paradigm)
- **/Livrable 2** : Documents du livrable 2.
  - **/Exécutables.zip** : Exécutables Java pour l'application fonctionnelle.
    - **/VisualLigue-1.0-SNAPSHOT.jar** : Exécutable principal à évaluer pour ce rapport.
    - **/Contribution équipe (nom\_du\_membre)** : Exécutables produit par les GLO 2<sup>ème</sup> année afin de démontrer leur apprentissage et intérêt au projet.
  - **/GLO2004EQ10.mpp** : Diagramme de Gantt (Microsoft Project) mis à jour.
  - **/GLO2004EQ10.vpp** : Diagrammes de classes et de séquences de conception
  - **/GLO2004EQ10.pdf** : Ce rapport.
- **/VisualLigue** : Code source de l'exéutable principal.
- **/Contribution équipe (nom\_du\_membre)** : Code source produit par les GLO 2<sup>ème</sup> années afin de démontrer leur apprentissage et intérêt au projet.

## 1.3. ÉCRANS DE L'INTERFACE (IMPLÉMENTATION RÉELLE)

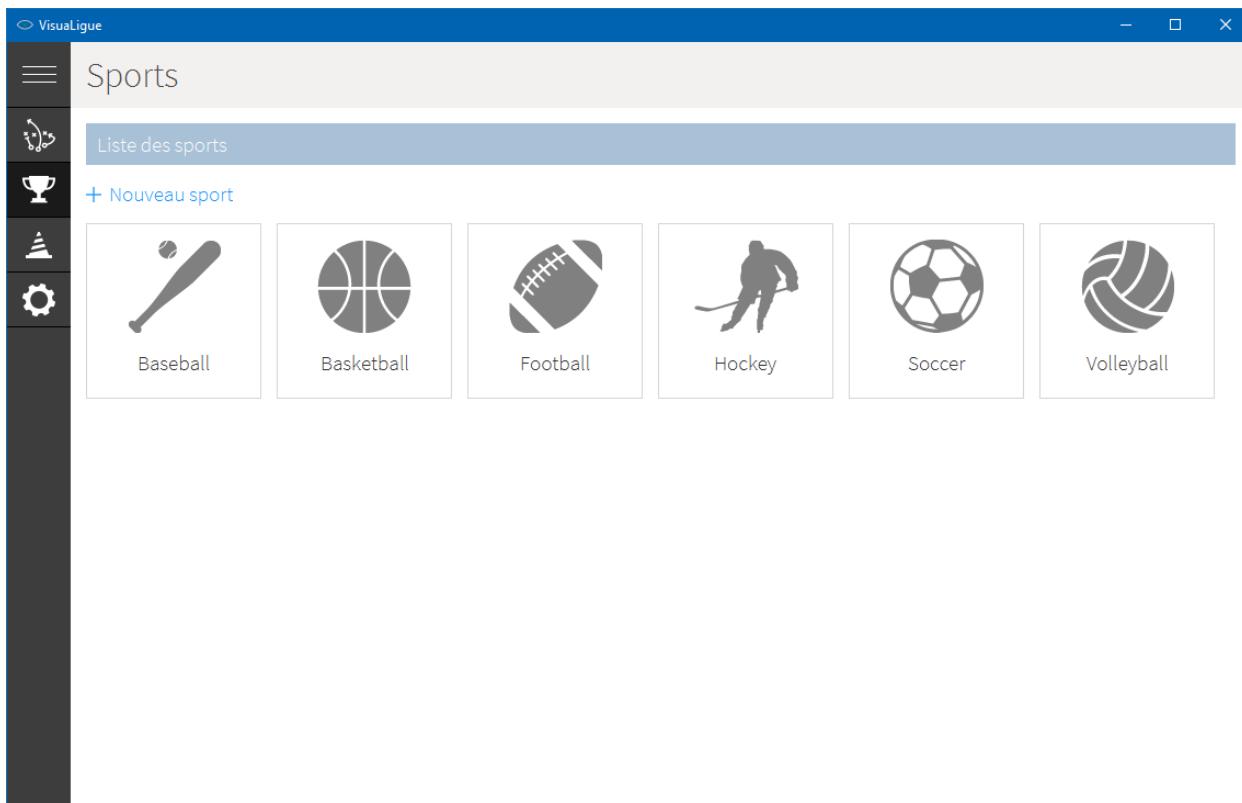
### 1.3.1. ÉDITEUR DE JEU



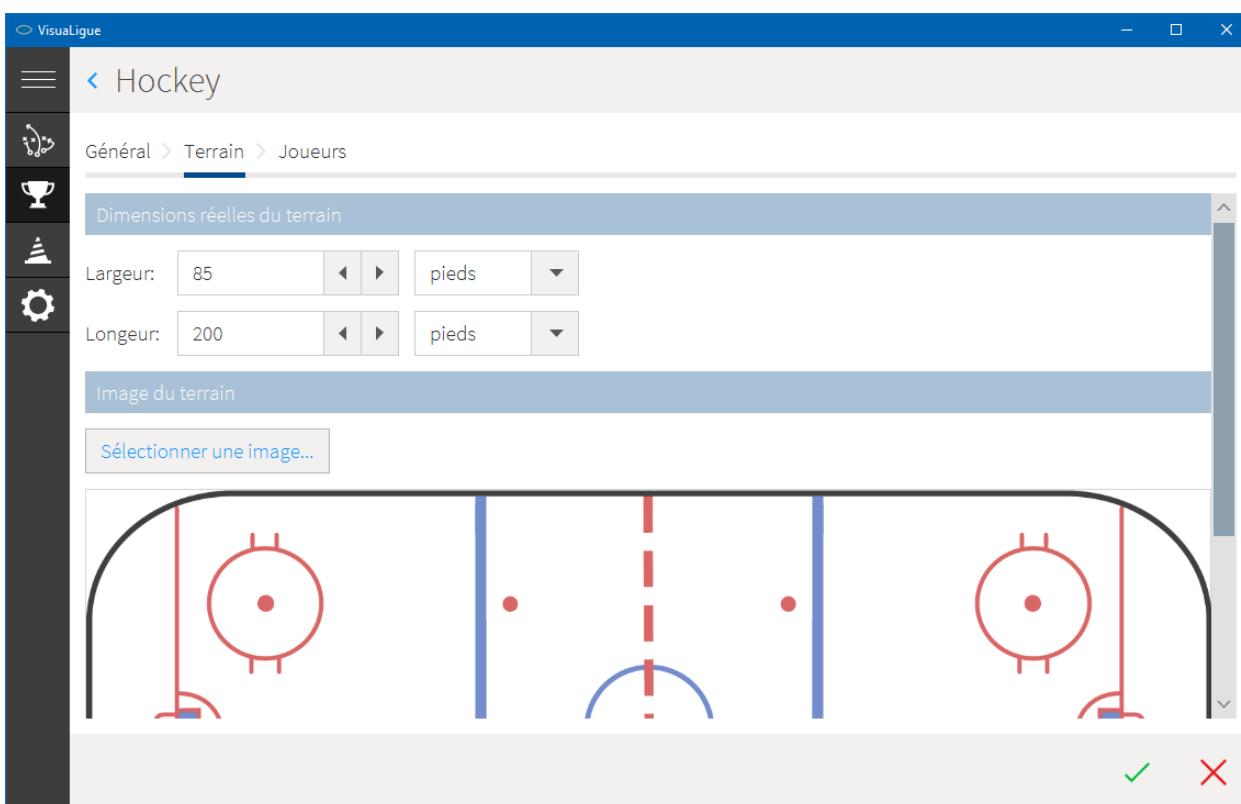
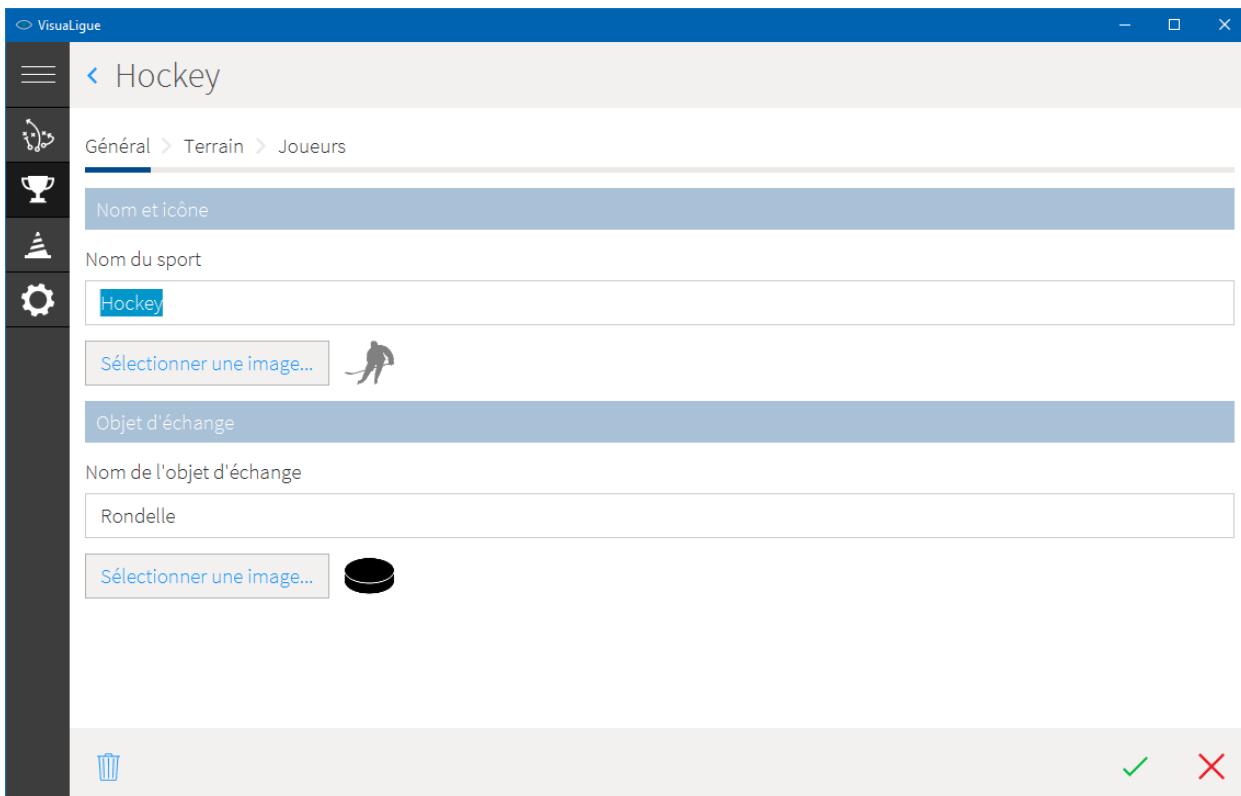
### 1.3.2. LISTE DES JEUX



### 1.3.3. GESTION DES SPORTS



### 1.3.4. CRÉATION/MODIFICATION D'UN SPORT



The screenshot shows the VisualLigue software interface with the title bar "VisualLigue". The main window title is "Hockey". The navigation path is "Général > Terrain > Joueurs". On the left sidebar, there are icons for Home, Terrain, Joueurs, Matchs, and Paramètres. The Joueurs section is active, showing a list of player categories:

Nom	Abréviation	Couleur
Ailier Droit	AD	
Ailier Gauche	AG	
Attaquant	A	
Centre	C	
Défenseur	D	
Gardien	G	

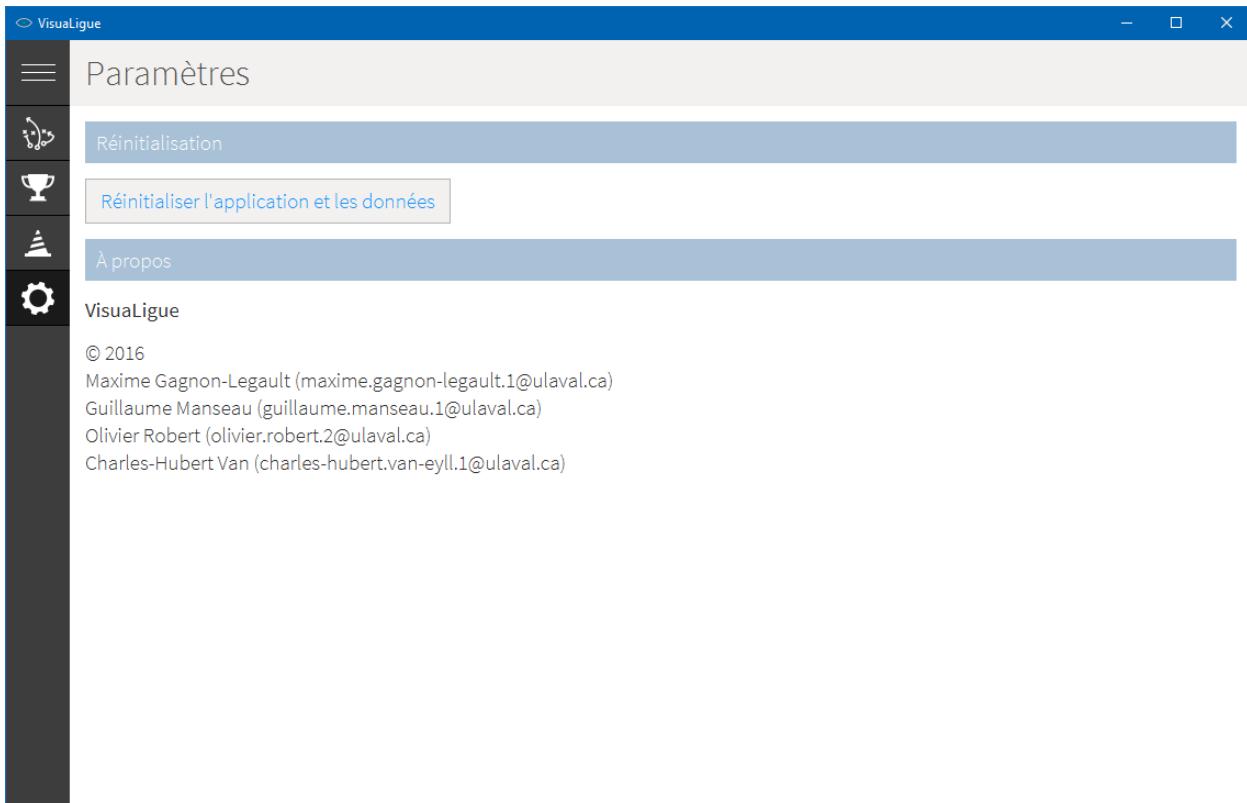
At the bottom right of the list area are two buttons: a green checkmark and a red X.

### 1.3.5. GESTION DES OBSTACLES

The screenshot shows the VisualLigue software interface with the title bar "VisualLigue". The main window title is "Obstacles". The navigation path is "Général > Terrain > Obstacles". On the left sidebar, there are icons for Home, Terrain, Joueurs, Matchs, and Paramètres. The Obstacles section is active, showing a list of obstacle types:

Nom	
Cône	

### 1.3.6. PARAMÈTRES



## 1.4. FONCTIONNALITÉS IMPLÉMÉNTÉES

La version fournie avec ce livrable est une version fonctionnelle de l’application possédant la les fonctionnalités suivantes :

- Gestion des sports (création, modification, suppression), incluant modification du nom, icône, image de la balle, dimensions du terrain, image du terrain et catégories de joueurs.
- Gestion des obstacles (création, modification, suppression).
- Création d’un jeu.
- Ajout d’une image en mode image-par-image.
- Positionnement des joueurs/obstacles/rondelle sur la surface de jeu.
- Enregistrement du jeu.
- Annuler/rétablissement.
- Zoom et navigation dans la scène.
- Affichage/masquage des catégories de joueurs.
- Persistance des données.
- Réinitialisation de l’application.

---

## 1.5. FONCTIONNALITÉS NON IMPLÉMENTÉES

---

Les fonctionnalités suivantes sont planifiées pour le livrable 3 et n'ont pas été implémentées pour ce livrable :

- Le nombre de joueurs par défaut pour une catégorie n'est pas pris en compte lors de la création d'un jeu.
- Affichage des joueurs transparents de l'image précédente lors de l'ajout d'une nouvelle image.
- Déplacement d'un joueur, d'un obstacle ou d'une rondelle/balle.
- Changement de l'orientation d'un joueur
- Assignation de la rondelle à un joueur spécifique.
- Exportation du jeu vers une image.
- Aperçu du jeu dans la liste des jeux (« thumbnail »).

---

## 1.6. LIBRAIRIES

---

Les librairies suivantes ont été utilisées sous autorisation du professeur :

- Maven (automatisation du build)
- JAXB (sérialisation XML)
- Google Guice (injection de dépendance)
- Apache Commons (classes utilitaires StringUtils, FileUtils, IOUtils, etc.)

---

## 1.7. EXÉCUTABLE

---

**Prérequis :**

- JRE 8 Update 102
- Windows 8 et supérieur (non testé sur autres systèmes d'exploitation)

**Emplacement de l'exécutable principal :**

- "/Livrable 2/Exécutables.zip/VisuaLigue-1.0-SNAPSHOT.jar"

**Exécution :** `java -jar "Chemin_du_fichier_jar"`

**Données applicatives :** Les données applicatives sont stockées selon le standard du système d'exploitation, soit le répertoire "%APPDATA%\VisuaLigue" pour Windows.

## 2. DIAGRAMMES DE CLASSE DE CONCEPTION

Dû à l'avancement notable du projet (21 100 lignes de codes et 200 classes), l'emphase a été mise sur la conception de diagrammes permettant de comprendre le fonctionnement du logiciel plutôt qu'une représentation exhaustive et inutile de l'entièreté des classes et des packages.

Les diagrammes de classes suivants **sont fournis** :

- Diagramme de classes du domaine.
- Diagramme de classes des services.
- Diagramme de classes de la persistance (implémentations XML)
- Diagrammes de classes du UI (contrôleurs, modèles et vues MVC) essentiels à la compréhension des diagrammes de séquences demandés.

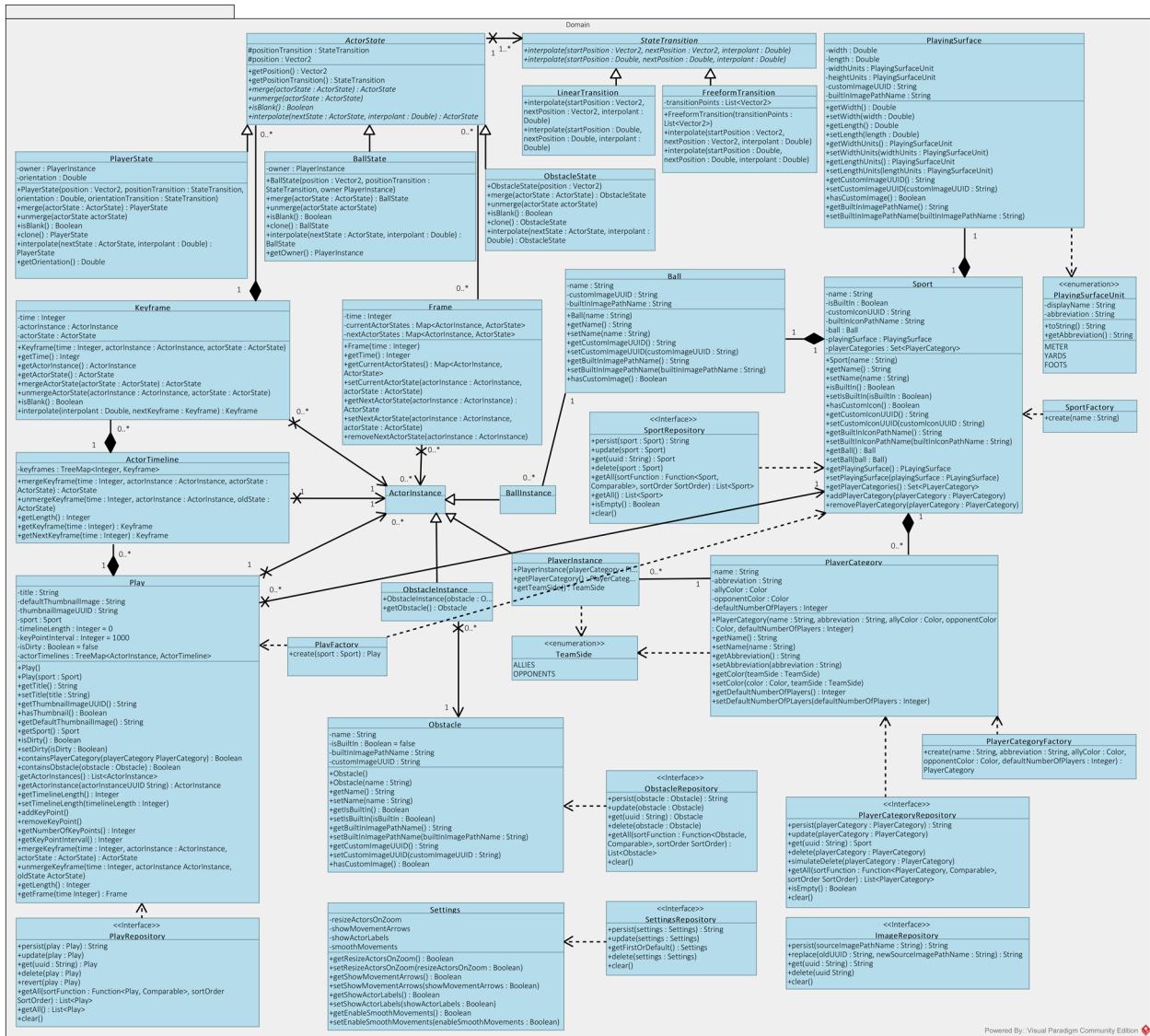
Les diagrammes de classes suivants **ne sont pas fournis** :

- Diagrammes de classes du UI (contrôleurs, modèles et vues MVC) non essentiels à la compréhension des diagrammes de séquences demandés.
- Diagramme de classes utilitaires.

Le [diagramme de package](#) fourni à la section suivante montre une vue globale de l'architecture l'application.

**Note** : Comme l'implémentation des fonctionnalités de l'application n'est pas requise pour ce livrable, il se peut que le code actuel diffère légèrement des classes présentées dans ce rapport.

## 2.1. DIAGRAMME DE CLASSES DU DOMAINE



**Sport** : Défini les propriétés d'un sport (nom, icône, balle, surface de jeu, catégories de joueurs, etc).

**Ball** : Défini les propriétés (nom, image) de la balle/rondelle/disque utilisé dans un sport.

**PlayerCategory** : Défini une catégorie de joueur (nom, abréviation, couleur, nombres de joueurs par défaut, etc.). Un sport peut définir plusieurs catégories de joueur. Celles-ci pourront être utilisées dans un jeu.

**PlayingSurface** : Défini les propriétés de la surface de jeu (largeur, longueur, unités de dimensions, image, etc.) utilisée dans un sport.

**Obstacle** : Défini les propriétés d'un obstacle (nom, image), éventuellement utilisé dans un jeu.

**ActorInstance** : Instance d'un acteur dans le jeu. Ce peut être une instance de joueur, d'un obstacle ou d'une balle/rondelle. Il peut y avoir plusieurs instances d'un même joueur, obstacle ou rondelle dans un même jeu (plusieurs ailiers gauche par exemple).

**ActorState** : Définit l'état d'un acteur (ex. : la position ou l'orientation d'un joueur, d'un obstacle ou d'une rondelle). **La position d'un joueur est définie en coordonnées taille-relative** (« size-relative position »), c'est-à-dire entre 0 et 1 en X et entre 0 et 1 en Y. 0 étant le début du terrain, 1 étant la fin du terrain. Les coordonnées sont indépendantes des pixels et des dimensions réelles du terrain.

**Keyframe** : Associe l'instance d'un acteur à un état de cet acteur à un temps précis (ex. : la position d'un joueur au temps 1000 ms).

**StateTransition** : Définit la transition qui aura lieu entre deux états. Une transition linéaire (LinearTransition) signifie que le joueur se déplacera en ligne droite de l'état 1 à l'état 2 lorsque le visionnement du jeu aura lieu. Une transition libre (FreeformTransition) définit une liste de points qui fera bouger le joueur selon les coordonnées enregistrées lorsque le joueur a été déplacé en mode temps réel.

**ActorTimeline** : Ligne de temps de l'instance d'un acteur. Chaque instance d'acteur possède sa ligne de temps. Une ligne de temps contient plusieurs « keyframe »s définissant les états d'un acteur dans le temps.

**Play** : Définit les propriétés d'un jeu (titre, image d'aperçu, sport associé, longueur de la ligne de temps, intervalle entre les « keypoints » (images), flag de sauvegarde et lignes de temps des acteurs). Un jeu contient une liste de lignes de temps (ActorTimeline) pour chaque acteur ajouté au jeu. Un **KeyPoint** (tel l'attribut *numberOfKeyPoints*) représente des points clés de la ligne de temps (une « image » au sens du langage du client).

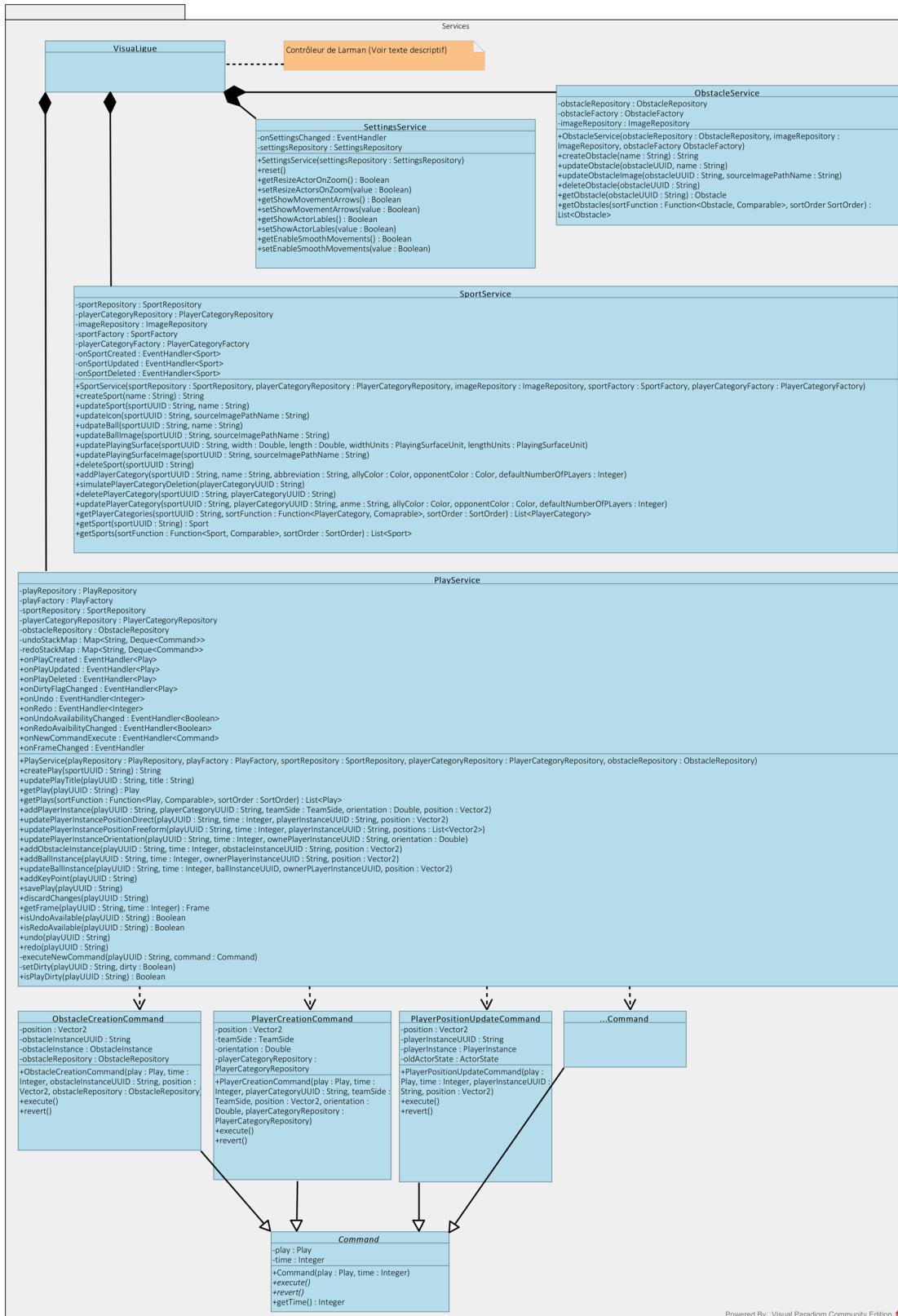
**Settings** : Définit les paramètres généraux de l'application.

**\*\*\*Repository** : Interface utilisée par les services pour la persistance des objets du domaine. Voir la section Diagramme de package.

**\*\*\*Factory** : Classes permettant la création des objets du domaine et facilitant la testabilité.

**Note** : Afin de ne pas alourdir inutilement le diagramme, les rôles (descriptions des associations) ont été omis, ceux-ci n'apporteraient absolument rien de significatif.

## 2.2. DIAGRAMME DE CLASSES DES SERVICES



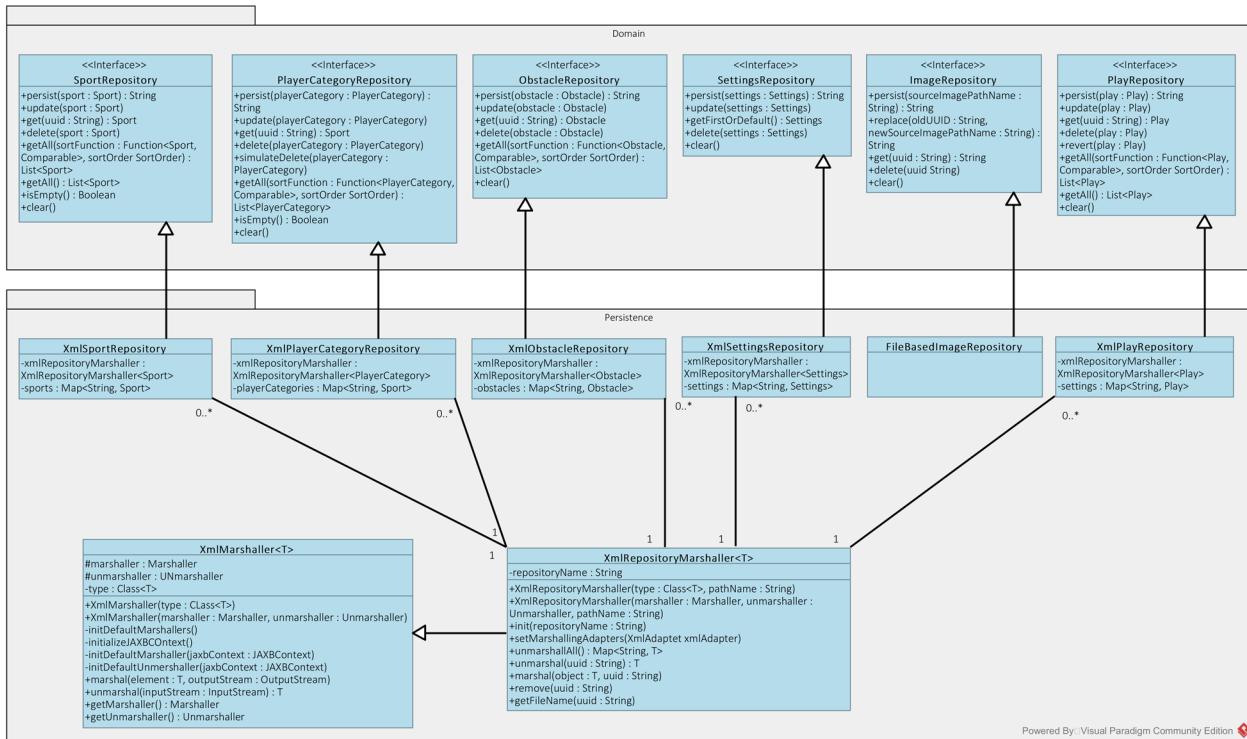
**PlayService** : Définit des méthodes permettant la gestion et la création d'un jeu, notamment la gestion de l'ajout de joueurs sur la surface de jeu, changement de position, sauvegarde d'un jeu, etc. Cette classe gère l'aspect des commandes pour l'annulation et le rétablissement (undo/redo). Des piles de commandes (une pile « undo » et « redo » pour chaque jeu) permettent l'accumulation des commandes qui pourront être annulées ou répétées.

**\*\*\*Services** : L'application dispose de plusieurs services séparés logiquement, telle une architecture logicielle normale. C'est le contrôleur de Larman séparé en plusieurs services logiques. Notez bien ici qu'on ne parle pas de service « technique » tel que décrit dans le livre de Larman mais bien de services opérant au niveau de la couche du domaine. Les services sont le point d'entrée de la logique d'affaires (après le contrôleur de Larman) et se chargent de synchroniser les objets du domaine avec la persistance via les dépôts.

**VisuaLigue** : Contrôleur de Larman. Ce sont les services décrits ci-haut mais tous mis en commun dans une géante « godclass ». Pour éviter la redondance et l'alourdissement inutile du diagramme de classes présenté, celui-ci n'a pas été détaillé. **Le contrôleur de Larman n'est ni plus ni moins que toutes les méthodes et attributs des services SportService, ObstacleService, PlayService et SettingsService regroupés sous une même classe géante et celui-ci ne fait que rediriger les appels vers les bons services.** Ceci a été fait dans le but de "presque" éviter l'Implémentation d'une « godclass » tout en respectant les directives du cours. Le contrôleur de Larman dans cette application n'est qu'une classe bidon n'agissant que d'intermédiaire entre le UI et les services. Les contrôleurs UI font affaire avec le contrôleur de Larman et celui-ci redirige vers les bons services individuellement.

**\*\*\*Command** : Classes définissant une action spécifique à exécuter. Chaque action spécifique peut être annulée et répétée. Le « undo »/« redo » est de type « generate state » (pas de sérialisation).

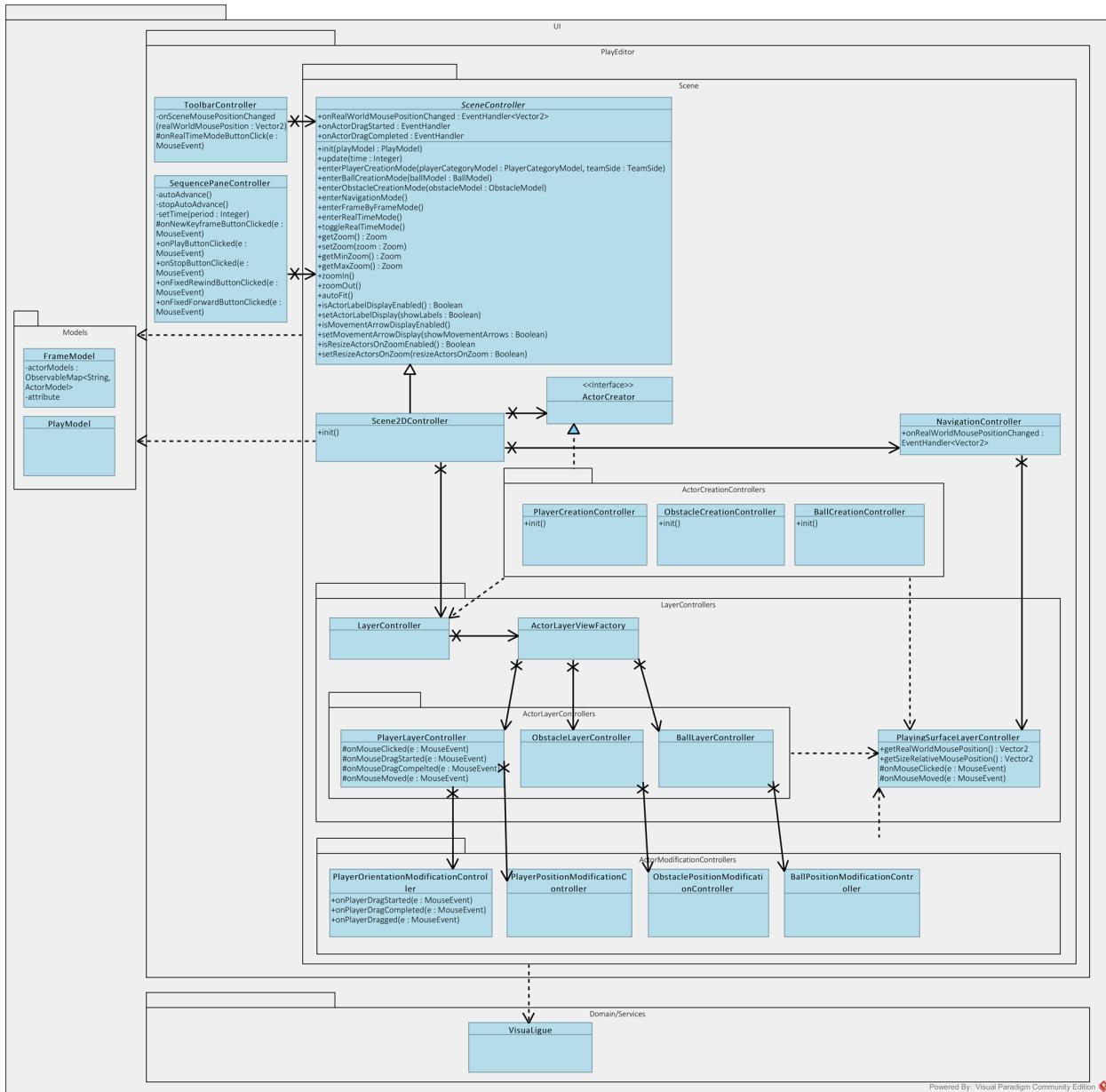
## 2.3. DIAGRAMME DE CLASSES DE LA PERSISTANCE



Couche permettant de gérer la persistance des objets du domaine via un concept de dépôts.

- **\*\*\*Repository :** Les dépôts implémentent l'interface définie dans le domaine. Implémentant typiquement l'ajout, la suppression ou la recherche d'objets. Cette application utilise une implémentation XML.
- **XmIMarshaller :** Classe générique permettant de sérialiser n'importe quel objet vers un flux de sortie ou de désérialiser n'importe quel objet depuis un flux d'entrée.
- **XmiRepositoryMarshaller :** Classe générique permettant la sérialisation et la désérialisation de n'importe quel objet dans un fichier XML selon les contraintes et besoins spécifiques d'un dépôt dans le système.

## 2.4. DIAGRAMME DE CLASSES DE L'INTERFACE UTILISATEUR

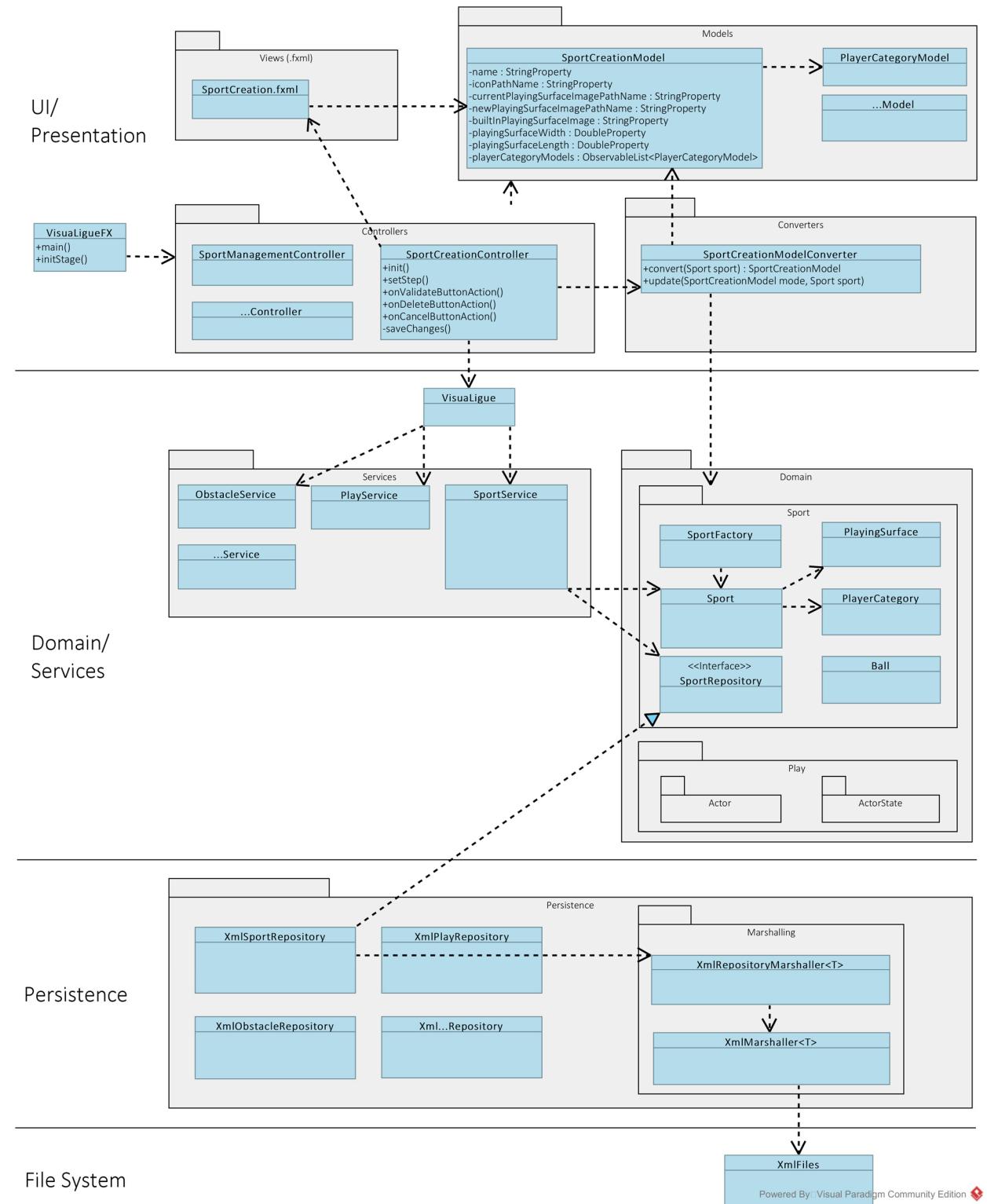


Ce diagramme est un diagramme de classe (partiel) montrant les différentes classes intervenant principalement dans la gestion de la scène au niveau UI.

- **SceneController** : Classe abstraite définissant les méthodes que les scènes 2D et 3D devront implémenter.
- **Scene2DController** : Implémentation 2D de la scène.
- **ToolbarController** : Contrôleur de la barre d'outils. Permet notamment à l'utilisateur de passer en mode temps réel.

- **SequencePaneController** : Contrôleur du panneau de séquence permettant à l'utilisateur de démarrer le visionnement, l'avancer, le reculer et l'arrêter, etc. Lorsqu'il y a défilement, c'est ce contrôleur qui indique la scène de se mettre à jour à un temps X.
- **FrameModel** : Modèle représentant une « frame » à un temps X devant être affiché dans la scène. Ce modèle contient une liste de sous-modèles d'acteurs.
- **ActorCreator** : Une interface définissant des méthodes à implémenter pour qu'une classe puisse agir en tant que créateur d'acteur.
- **PlayerCreationController**, **ObstacleCreationController** et **BallCreationController** : Contrôleurs implémentant ActorCreator et permettant la création d'acteurs (joueurs, obstacles, balle/rondelle).
- **NavigationController** : Contrôleur s'occupant de gérer l'aspect navigation de la scène (déplacement avec la souris, zoom, etc.);
- **LayerController** : Contrôleur s'occupant de gérer les couches de la scène. Chaque acteur est sur sa propre couche.
- **ActorLayerViewFactory** : Factory permettant la création d'une couche d'acteur.
- **PlayerLayerController**, **ObstacleLayerController**, **BallLayerController** : Contrôleur s'occupant des couches d'acteurs spécifiques. Chaque contrôleur gère sa couche respective. Il y a un contrôleur pour chaque instance d'acteur.
- **PlayingSurfaceLayerController** : Contrôleur spécial s'occupant de gérer la couche inférieure représentant la surface de jeu. Celle-ci reçoit les événements souris et peut en notifier ses observateurs.
- **ActorModificationControllers** : Contrôleurs permettant la gestion d'événements spécifiques à un acteur (déplacer un joueur, changer l'orientation d'un joueur, etc).
- **VisuaLigue** : Contrôleur de Larman. Les contrôleurs ci-haut font affaire directement avec ce contrôleur. Voir le Diagramme de classes des services pour les détails concernant ce contrôleur.

### 3. DIAGRAMME DE PACKAGE



Ce diagramme montre une vue globale de l'architecture de l'application. Afin de mieux visualiser les relations entre les classes et les packages, le diagramme présente une vue « transversale » des classes et relations associées à la création de sport. Les mêmes relations sont appliquées pour tous les use-cases.

**Couche UI/Presentation :** Couche effectuant la gestion de l'interface utilisateur via une implémentation MVC. **Attention!** MVC a été implémenté côté UI et ne viole aucunement les directives du cours au sujet du contrôleur de Larman. Cette application a été conçue selon les bonnes pratiques de JavaFX, c'est-à-dire l'utilisation de MVC. Voir le professeur à ce sujet.

- **Views** : Vues FXML.
- **Models** : Modèles permettant le « binding » avec la vue en JavaFX. Les modèles sont créés ou mis à jour par les « converters ». Les modèles sont munis de propriétés « observables » permettant la mise à jour en temps réel de l'interface avec très peu de code.
- **Controllers (MVC)**: Gèrent la logique d'affichage. Les contrôleurs MVC font affaire avec le contrôleur de Larman (VisualLigue) lorsque l'exécution de la logique d'affaires est requise ou lorsque que l'affichage de données du domaine est requis (via les « converters » et « models »).
- **Converters** : Effectuent la conversion d'objets du domaine vers des modèles spécifiques à un use case. Les modèles résultants seront utilisés par une ou plusieurs vues. Les objets du domaine ne se promènent jamais dans le UI.

**Couche Domain/Service** : Couche effectuant la logique d'affaires.

- **VisualLigue** : Contrôleur de Larman. Toutes les requêtes des contrôleurs MVC UI passent par ce contrôleur. Cette classe n'est qu'une classe anémique redirigeant vers les services regroupant des fonctionnalités logiques. N'est présente que pour se plier aux directives du cours
- **Services**: Points d'entrée de la logique d'affaires (après le contrôleur de Larman) et se chargent de synchroniser les objets du domaine avec la persistance via les dépôts.
- **Domain** : Objets du domaine. La majorité de la logique d'affaires est opérée à cet endroit.

**Couche Persistence** : Couche permettant de gérer la persistance des objets du domaine via un concept de dépôts. Voir le Diagramme de classes de la persistance pour les détails concernant cette couche.

**Couche File System** : Système de fichier des dépôts. Le système de fichier pour cette application est le système de fichier local.

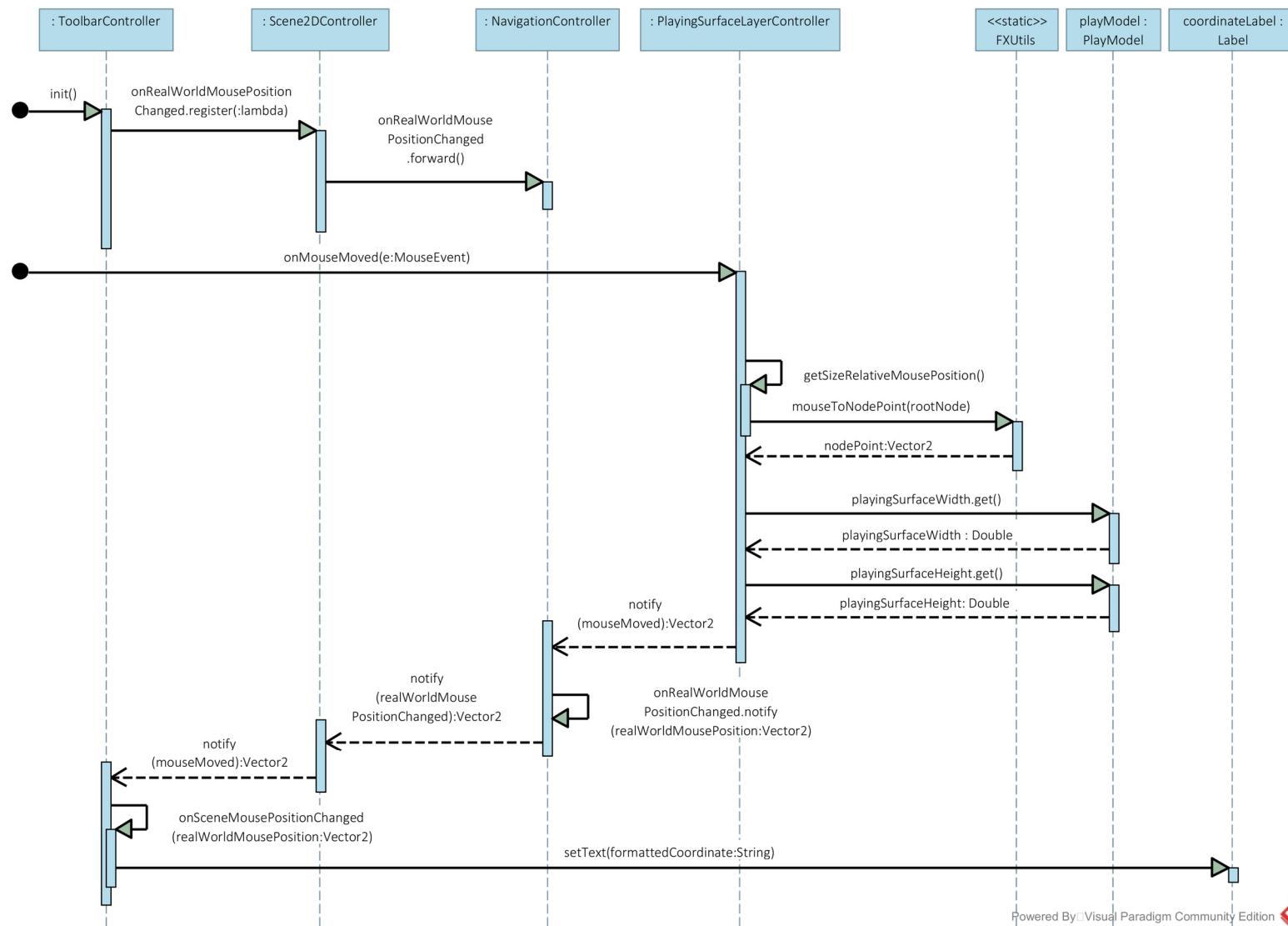
**Note concernant l'interaction domaine -> UI** : La communication du domaine (contrôleur de Larman / services) vers la couche UI se fait via de l'événementiel (classe *EventHandler*, une implémentation du patron observateur de C# en Java). Par exemple, le *SportListController* s'enregistre auprès du *SportService* via l'événement *onSportUpdated* pour mettre à jour l'affichage de la liste des sports lorsqu'un sport est modifié. Le domaine, le contrôleur de Larman et les services ne connaissent pas le UI, ne connaissent pas JavaFX et ne connaissent pas un clic de souris. La logique UI est gérée côté UI (par les contrôleurs MVC) et la logique d'affaires est gérée côté domaine.

## 4. DIAGRAMME DE SÉQUENCE DE CONCEPTION

**Note** : Les numéros de séquence ont été masqués afin d'améliorer la visibilité du diagramme.

**Note** : Les diagrammes de séquence suivants fonctionnent, ils ont été implémentés (le point 4 et le point 5 n'ont pas été livrés, car l'implémentation n'est pas finale).

## 4.1. CONVERSION DE COORDONNÉES

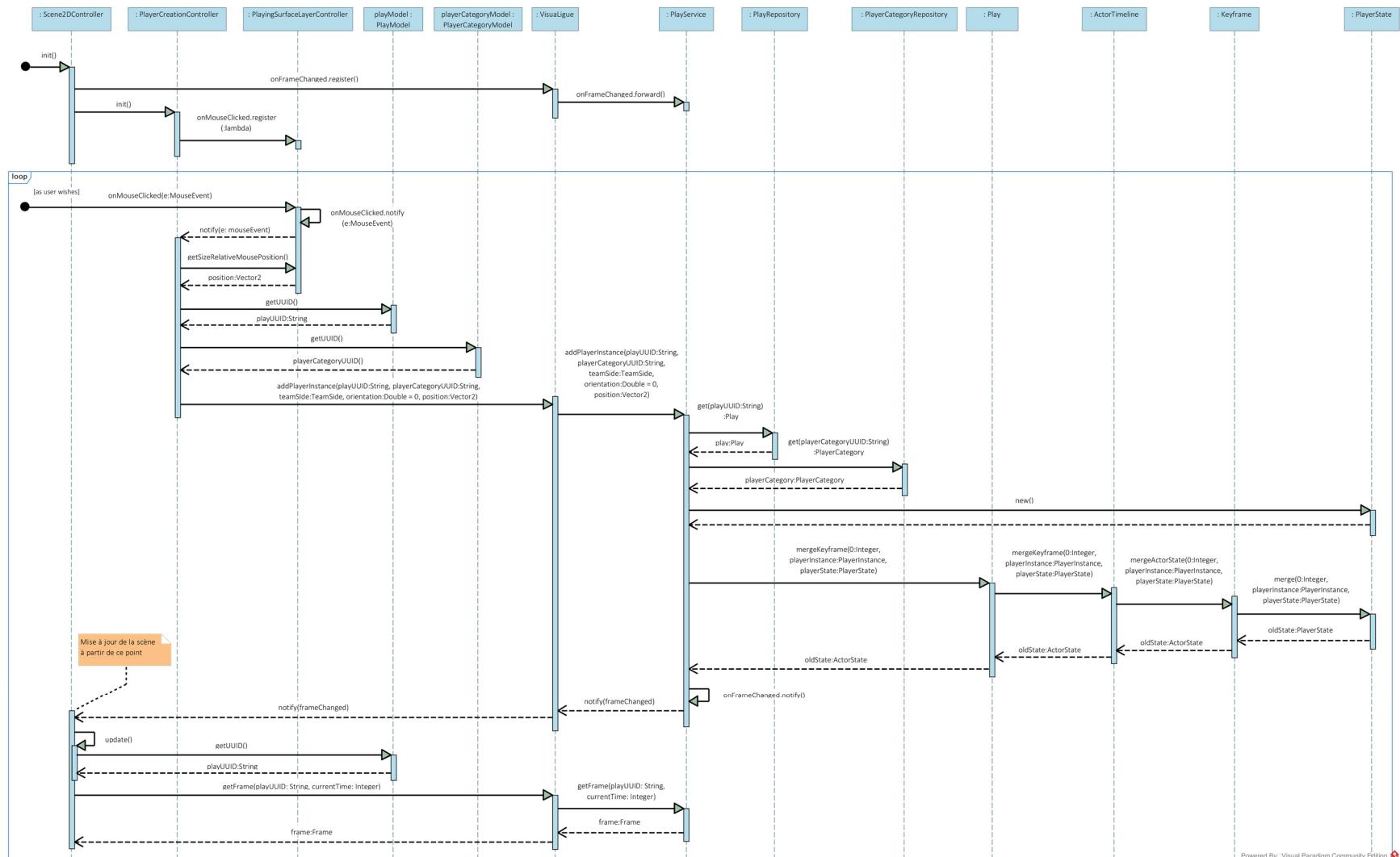


Cette question ne fait aucun sens dans une application où la couche de présentation/UI a été correctement séparée de la couche de domain/logique d'affaires (contrôleur de Larman ou non). Les points en « pixel », le zoom et la position courante du terrain sont des concepts UI, gérés par les contrôleurs du UI. Différentes technologies UI (JavaFX, une application Web JavaScript, etc.) gèrent ces concepts différemment et ne peuvent donc être intégrés au domaine. Le domaine ne connaît pas ces concepts et il n'y aucune logique d'affaire intervenant dans ce cas d'utilisation.

1. Le contrôleur de la barre d'outils (où sont affichées les coordonnées) s'enregistre auprès du contrôleur de la scène pour les changements de coordonnées,
2. Celui-ci fait de même au contrôleur de navigation et redirige l'observateur.
3. Lorsque la souris se déplace, le panneau de scène contenant la surface de jeu détecte un mouvement.
4. Celui-ci demande à une classe utilitaire (utilisant des fonctions génériques JavaFX) afin de déterminer les coordonnées du point de la souris. Le zoom et la position du terrain sont sans importance car les coordonnées souris sont obtenues en coordonnées relatives (entre 0 et 1).
5. Le contrôleur de surface de jeu ne fait que multiplier ces coordonnées relatives par la grosseur de la surface de jeu contenue dans le modèle, ce qui donne les coordonnées réelles.
6. Ces coordonnées sont ensuite retournées tour à tour aux observateurs jusqu'à la barre d'outils, qui les affiche.

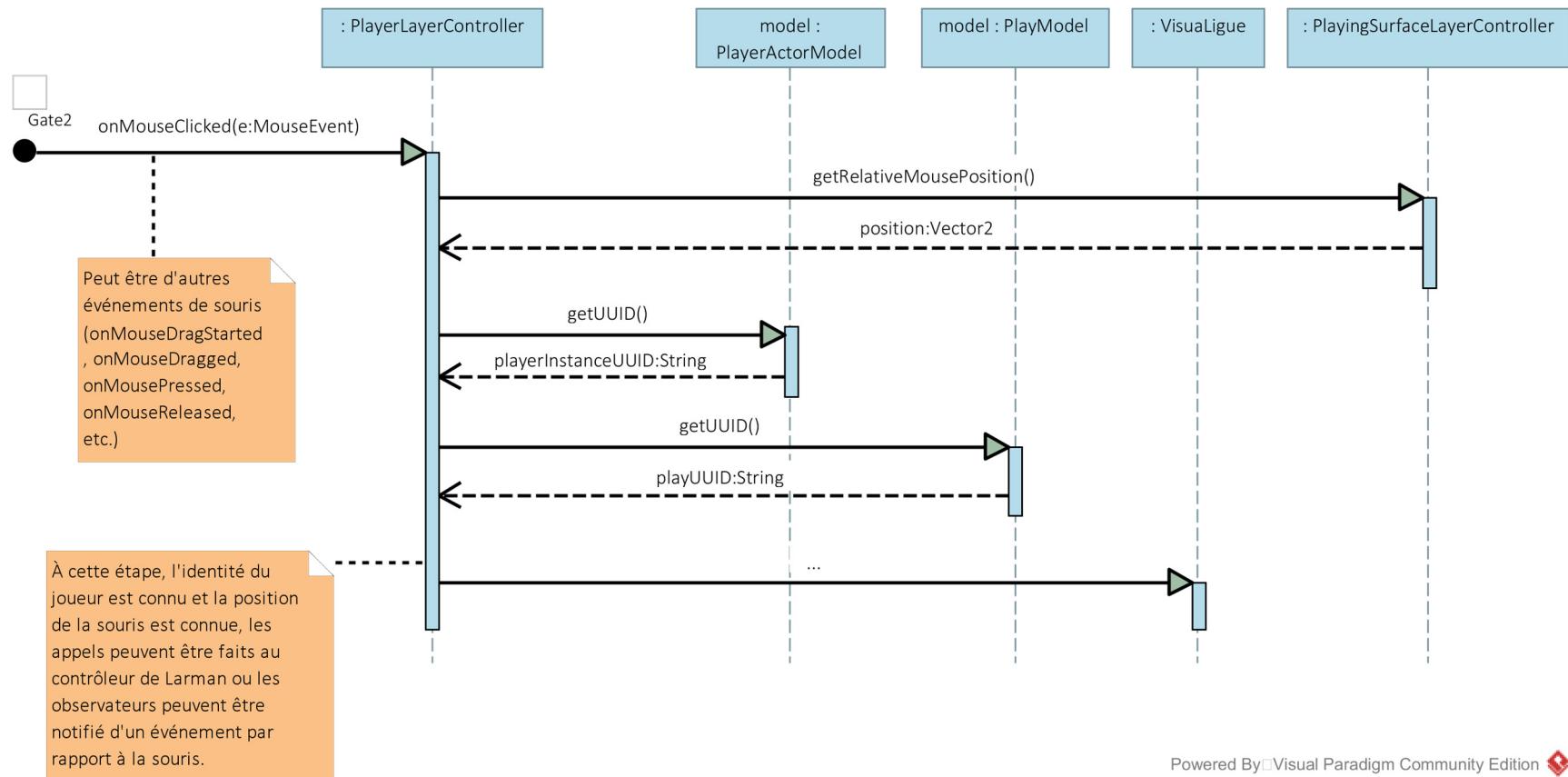
**Note :** Il n'y a aucune logique d'affaire dans ce cas d'utilisation. L'opération de multiplication est débatteable, mais architecturalement parlant c'est « overkill » de faire ça dans le domaine. Le contrôleur de Larman n'intervient pas.

## 4.2. AJOUT D'UN JOUEUR



1. Le contrôleur de la scène s'enregistre auprès du contrôleur de Larman pour être notifié de la mise à jour de la « frame » courant (et celui-ci redirige l'observateur au service de jeu).
2. Le contrôleur de création de joueurs s'enregistre auprès du contrôleur de la surface de jeu pour les clics de souris.
3. Lorsqu'un clic survient, le contrôleur de création de joueurs est notifié.
4. À l'aide de l'information contenue dans les modèles et de la position du clic de souris (convertie en position relative), le contrôleur de création de joueur appelle le contrôleur de Larman pour ajouter un joueur.
5. Cet appel est simplement transféré au service de jeu.
6. Le service de jeu va chercher dans les dépôts le jeu et la catégorie de joueur spécifié en paramètre.
7. À l'aide ces éléments, le service de jeu crée un nouvel état de joueur « PlayerState » contenant la position du joueur (et son orientation initiale à 0).
8. Le service fusionne cet état dans le jeu.
9. Le fusionnement est effectué à l'aide d'appels en cascade jusqu'à ce que l'état soit ajouté (ou fusionné si un état existe déjà pour cet acteur au temps spécifié).
10. Le service de jeu notifie ensuite ces observateurs d'une modification
11. La scène se met à jour par la suite en demande au service de jeu un recalcul de la « frame » au temps courant.

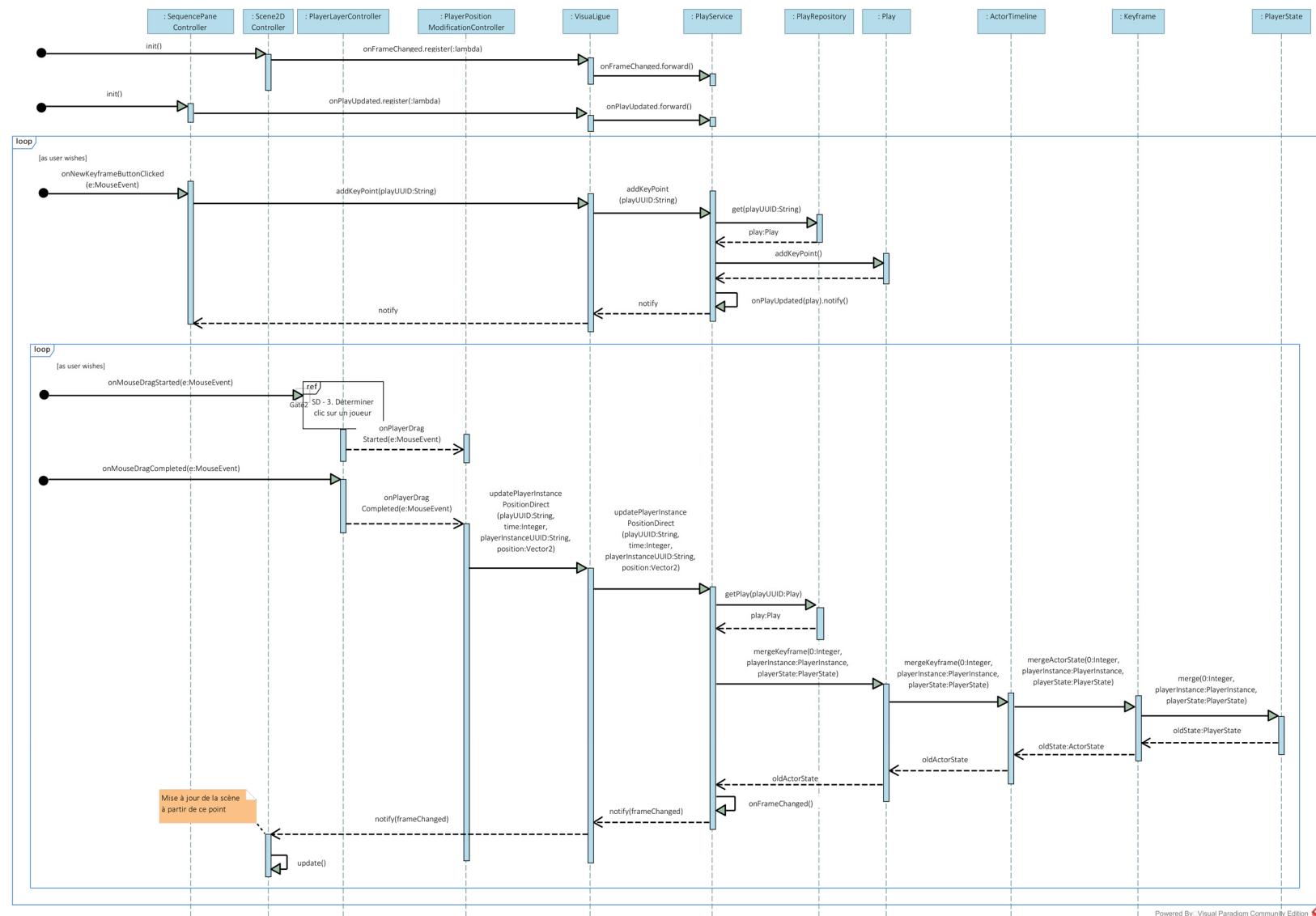
### 4.3. DÉTERMINER CLIC SUR UN JOUEUR



Encore une fois, ceci ne fait aucunement intervenir le domaine. Le clic est un événement purement JavaFX sur un objet dans la scène. L'objet est associé à un modèle, contenant le UUID du joueur associé dans le domaine.

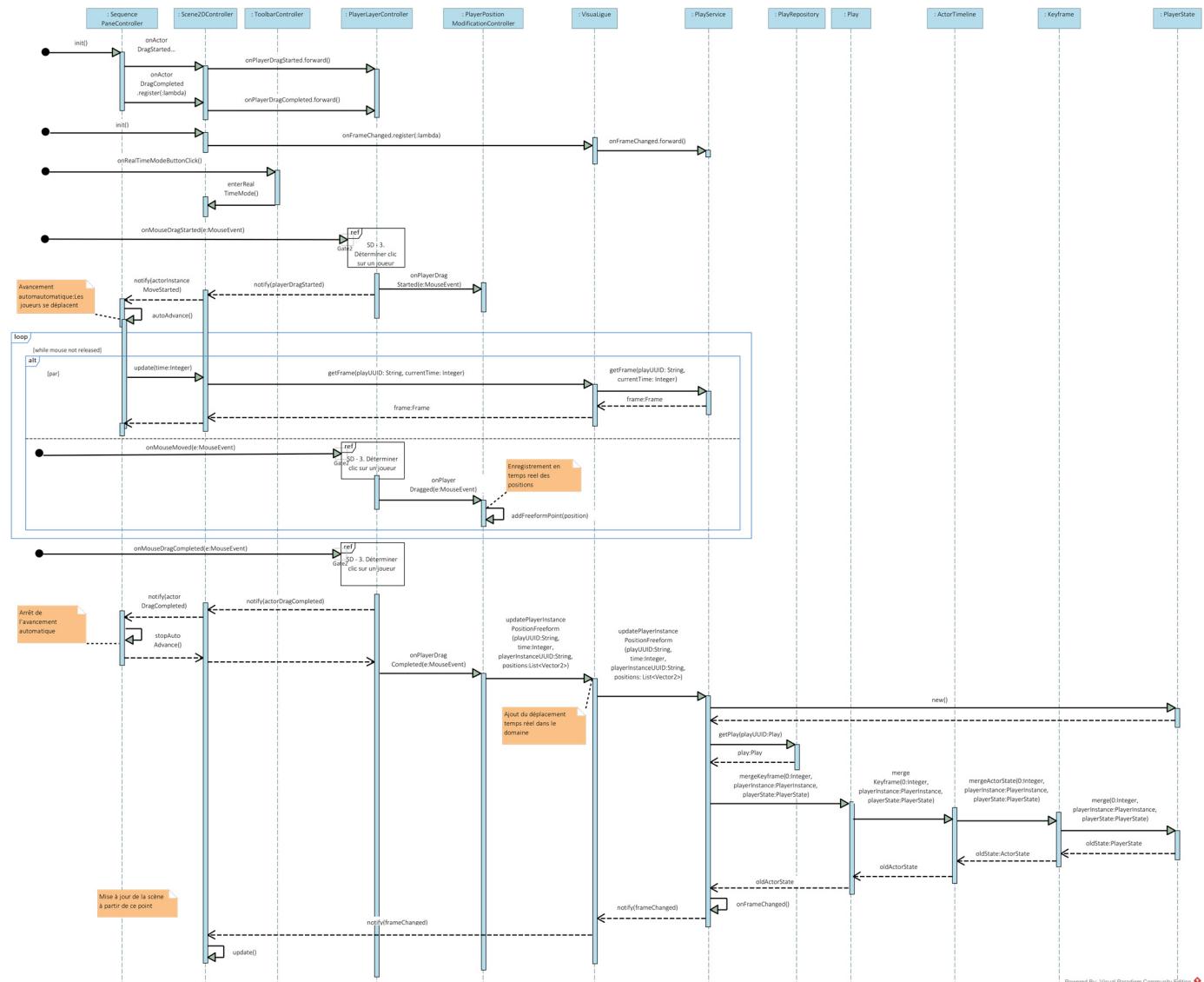
1. Le contrôleur de la couche de joueur est notifié d'un clic sur sa couche (événement purement JavaFX).
2. Le contrôleur peut aller chercher la position taille-relative du clic à l'aide d'un appel au contrôleur de la surface de jeu.
3. À l'aide de l'information précédente et des informations contenues dans les modèles, le contrôleur peut faire des appels au contrôleur de Larman pour des actions relatives à ce joueur.

#### 4.4. ÉDITION IMAGE PAR IMAGE



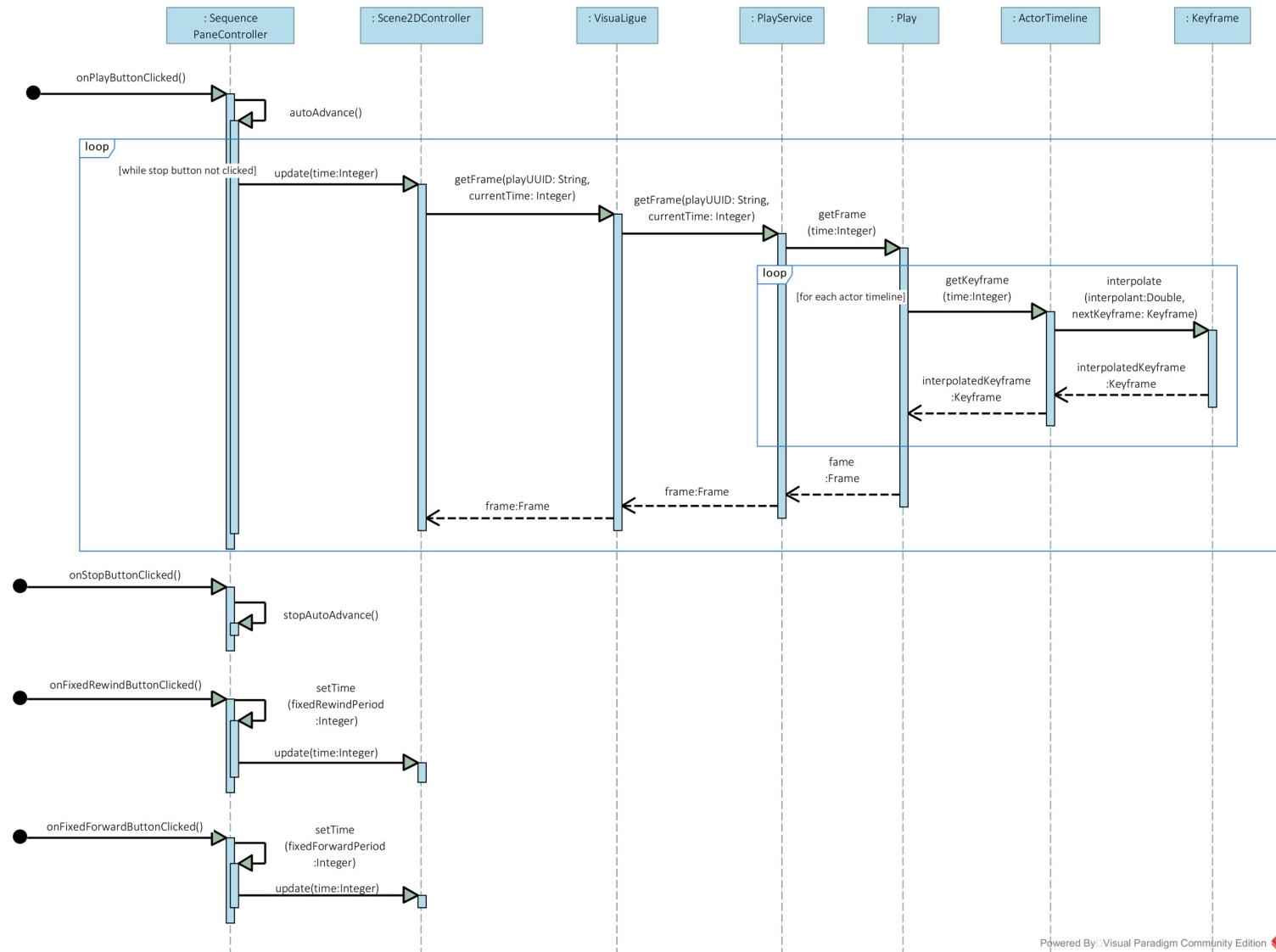
1. Initialisation : Le contrôleur de la scène s'enregistre auprès du contrôleur de Larman pour être notifié de changements de la « frame » courante (celui-ci redirige l'observateur au service de jeu).
2. Initialisation : Le contrôleur du panneau de séquence s'enregistre auprès du contrôleur de Larman pour être notifié de changements au jeu (notamment du nombre de « keypoints » à afficher sur la ligne de temps). Celui-ci redirige l'observateur au service de jeu.
3. Lorsque l'utilisateur clique sur le bouton pour ajouter un « keypoint » (un « image » dans le langage du client), celui-ci appelle la méthode pertinente du contrôleur de Larman (qui redirige l'appel au service de jeu).
4. Le contrôleur du panneau de séquence est notifié de ce changement et peut mettre à jour les « keypoints » affichés sur la ligne de temps et se déplacer à ce nouveau « keypoint ».
5. Lorsque l'utilisateur clic sur un joueur et le déplace (voir le diagramme de séquence Déterminer clic sur un joueur pour déterminer le clic et le joueur), le contrôleur de modification de position de joueur est notifié par le contrôleur de la couche de joueur.
6. Celui-ci demande au contrôleur de Larman de mettre à jour la position du joueur au temps courant.
7. S'en suit de la fusion de l'état du joueur et la mise à jour de la scène tel que décrits au diagramme de séquence Ajout d'un joueur.

## 4.5. ÉDITION TEMPS RÉEL



1. Initialisation : Le contrôleur du panneau de séquence s'enregistre auprès de la scène pour être notifié du début et de la fin d'un déplacement d'un joueur. Celui-ci redirige l'observateur au contrôleur du joueur.
2. Initialisation : Le contrôleur de la scène s'enregistre auprès du service de jeu pour être notifié de changements de la « frame » courante.
3. L'utilisateur passe en mode temps réel en cliquant sur le bouton à cet effet dans la barre d'outils.
4. L'utilisateur clique sur un joueur et commence à le déplacer (voir le diagramme de séquence Déterminer clic sur un joueur pour déterminer le clic et le joueur).
5. Le contrôleur du panneau de séquence est notifié et celui-ci fait avancer le jeu automatiquement (via un Timer) pour faire apparaître les mouvements des autres joueurs pendant le déplacement. Ceci est de la logique purement UI.
6. En se déplaçant, le panneau de séquence demande à la scène de se mettre à jour au temps courant. Ceci faire apparaître les mouvements des autres joueurs sur la scène pendant le déplacement du joueur. La scène demande la « frame » du temps courant au contrôleur de Larman et l'affiche.
7. Pendant ce temps, le contrôleur de modification de position de joueur enregistre les déplacements du joueur en temps réel.
8. Lorsque l'utilisateur relâche le joueur (voir le diagramme de séquence Déterminer clic sur un joueur pour déterminer le clic et le joueur), le contrôleur de modification de position de joueur est notifié.
9. Celui-ci demande au contrôleur de Larman de mettre à jour la position du joueur (en mode temps réel, c'est-à-dire une liste de points définissant son déplacement).
10. S'en suit de la fusion de l'état du joueur et la mise à jour de la scène tels que décrits au diagramme de séquence Ajout d'un joueur.

## 4.6. VISIONNEMENT DU JEU



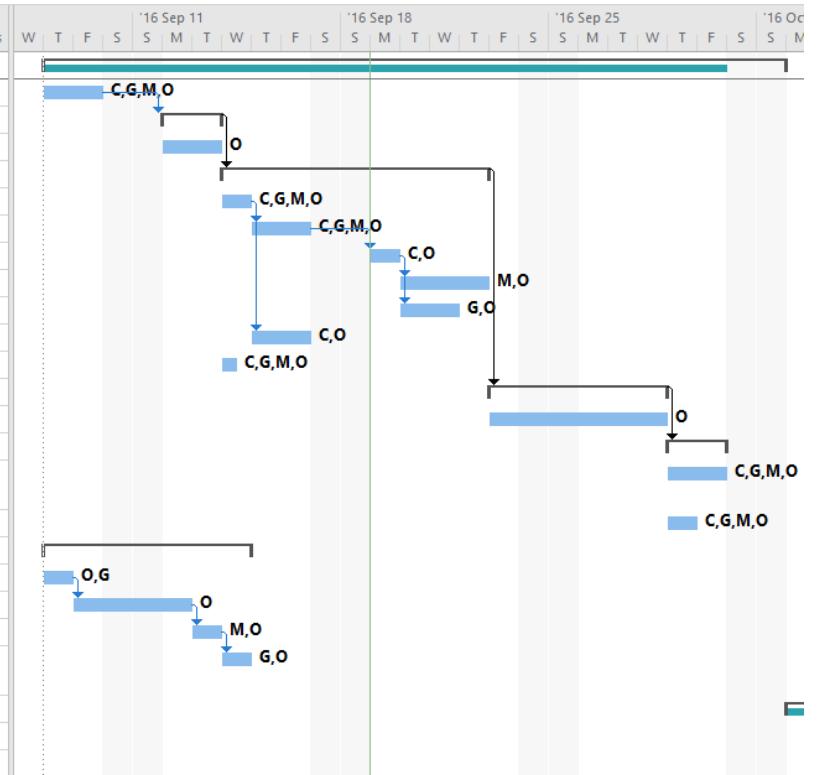
1. Lorsque l'utilisateur clique sur le bouton « Débuter » sur le panneau de séquence, ce dernier exécute l'avancement automatique.
2. À chaque avancement (60 fois par secondes), le contrôleur du panneau de séquence demande à la scène de se mettre à jour au temps courant.
3. La scène demande au contrôleur de Larman la « frame » au temps courant, dont l'appel est redirigé vers le service de jeu et ce dernier vers le jeu lui-même.
4. Pour chaque ligne de temps d'acteur (« ActorTimeline »), le jeu demande à la ligne de temps d'acteur de lui fournir un keyframe au temps donné. Ce qui « keyframe » peut être interpolé si un « keyframe » n'existe pas directement au temps spécifié.
5. La ligne de temps trouve le keyframe plancher (« floor ») et le keyframe plafond (« ceiling ») (méthodes internes en Java sur un NavigableMap) et une interpolation est effectuée sur le « keyframe ».
6. La « frame » résultante contenant l'état précis de tous les acteurs au temps demandé est retourné.
7. Le processus de mise à jour de la scène à partir d'une « frame » se fait via des « bindings » JavaFX (ObservableMap et autres propriétés « observables »), donc aucun appel direct. Dû à la difficulté de représentation de ce fonctionnement sur un diagramme de séquence, cette étape n'est pas documentée.
8. Lorsque l'utilisateur clique sur le bouton « Pause » (ou « Arrêter »), le contrôleur du panneau de séquence arrête son avancement automatique en annulant son « Timer ».
9. Lorsque l'utilisateur clique sur le bouton de recul prédéfini, le contrôleur du panneau de séquence recul au temps prédéfini spécifié (modifiable par l'utilisateur), et la mise à jour de la scène s'en suit tel que décrit plus haut.
10. Lorsque l'utilisateur clique sur le bouton d'avancement prédéfini, le contrôleur du panneau de séquence recul au temps prédéfini spécifié (modifiable par l'utilisateur), et la mise à jour de la scène s'en suit tel que décrit plus haut.

## 5. DIAGRAMME DE GANTT (MIS À JOUR)

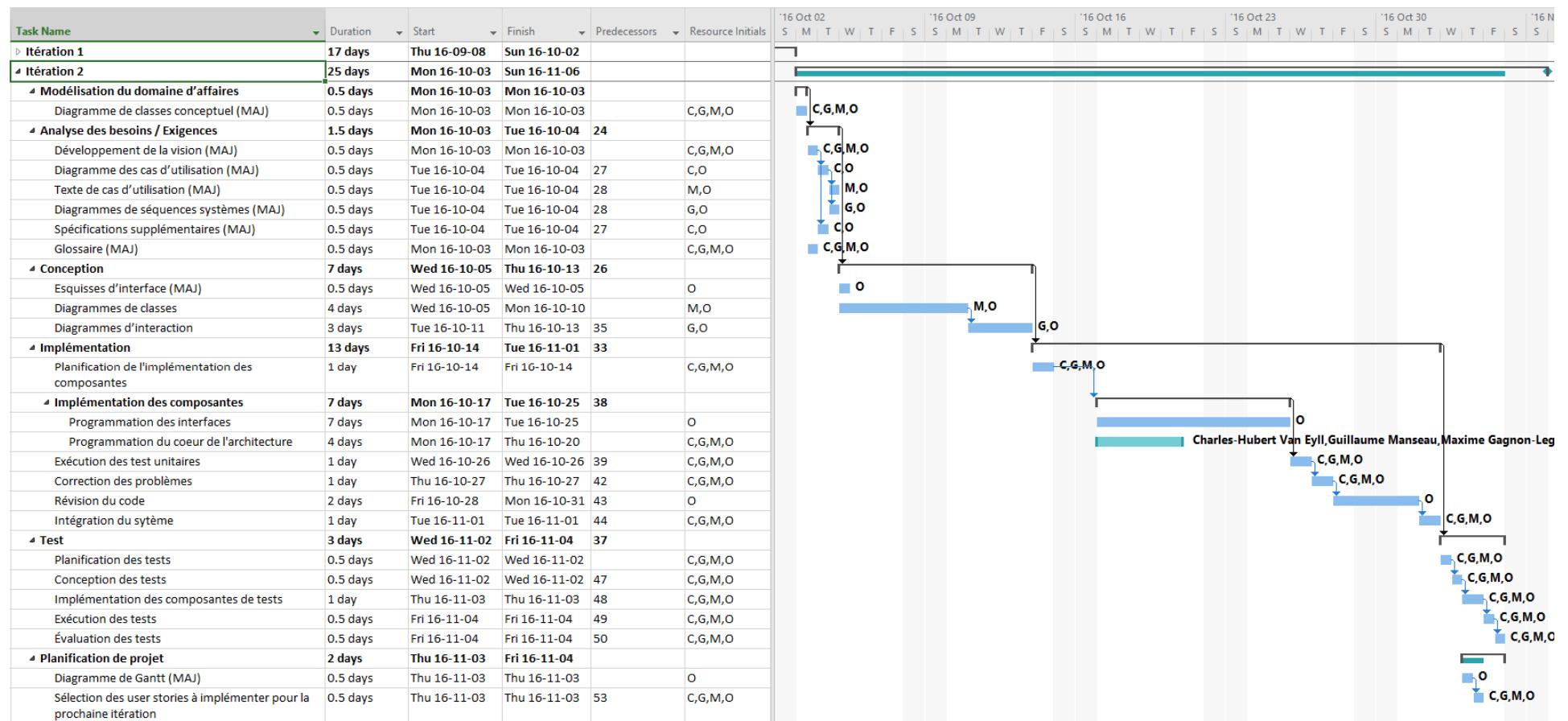
Voir le document Microsoft Project fourni avec ce document (/Livrable 2/GLO2004EQ10.mpp) pour les détails du diagramme.

## 5.1. ITÉRATION 1

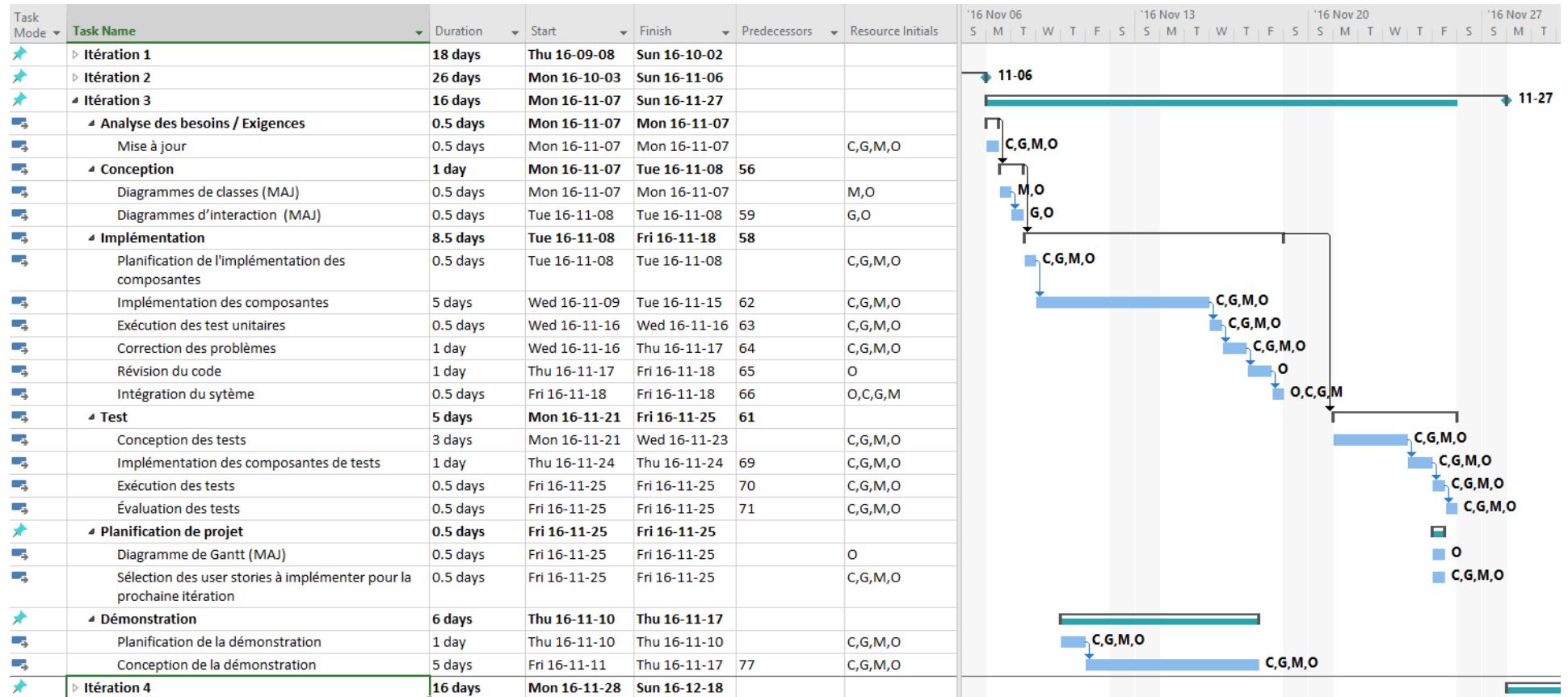
Task Mode	Task Name	Duration	Start	Finish	Predecessors	Resource Initials
▷	▷ Itération 1	17 days	Thu 16-09-08	Sun 16-10-02		
➡	Préparation de l'environnement et des outils	2 days	Thu 16-09-08	Fri 16-09-09		C,G,M,O
➡	▷ Modélisation du domaine d'affaires	2 days	Mon 16-09-12	Tue 16-09-13	2	O
➡	Diagramme de classes conceptuel	2 days	Mon 16-09-12	Tue 16-09-13		O
➡	▷ Analyse des besoins / Exigences	7 days	Wed 16-09-14	Thu 16-09-22	3	
➡	Compréhension du problème	1 day	Wed 16-09-14	Wed 16-09-14		C,G,M,O
➡	Développement de la vision	2 days	Thu 16-09-15	Fri 16-09-16	6	C,G,M,O
➡	Diagramme des cas d'utilisation	1 day	Mon 16-09-19	Mon 16-09-19	7	C,O
➡	Texte de cas d'utilisation	3 days	Tue 16-09-20	Thu 16-09-22	8	M,O
➡	Diagrammes de séquences systèmes	2 days	Tue 16-09-20	Wed 16-09-21	8	G,O
➡	Spécifications supplémentaires	2 days	Thu 16-09-15	Fri 16-09-16	6	C,O
➡	Glossaire	0.5 days	Wed 16-09-14	Wed 16-09-14		C,G,M,O
➡	▷ Conception	4 days	Fri 16-09-23	Wed 16-09-28	5	
➡	Esquisses d'interface	4 days	Fri 16-09-23	Wed 16-09-28		O
➡	▷ Implémentation	2 days	Thu 16-09-29	Fri 16-09-30	13	
➡	Planification de l'implémentation des composantes	2 days	Thu 16-09-29	Fri 16-09-30		C,G,M,O
➡	Implémentation des composantes	1 day	Thu 16-09-29	Thu 16-09-29		C,G,M,O
➡	▷ Planification de projet	5 days	Thu 16-09-08	Wed 16-09-14		
➡	Plan sommaire du projet	1 day	Thu 16-09-08	Thu 16-09-08		O,G
➡	Diagramme de Gantt	2 days	Fri 16-09-09	Mon 16-09-12	19	O
➡	Budget	1 day	Tue 16-09-13	Tue 16-09-13	20	M,O
➡	Identification des risques et élaboration des stratégies de gestion du risque	1 day	Wed 16-09-14	Wed 16-09-14	21	G,O
▷	▷ Itération 2	25 days	Mon 16-10-03	Sun 16-11-06		
▷	▷ Itération 3	16 days	Mon 16-11-07	Sun 16-11-27		
▷	▷ Itération 4	16 days	Mon 16-11-28	Sun 16-12-18		



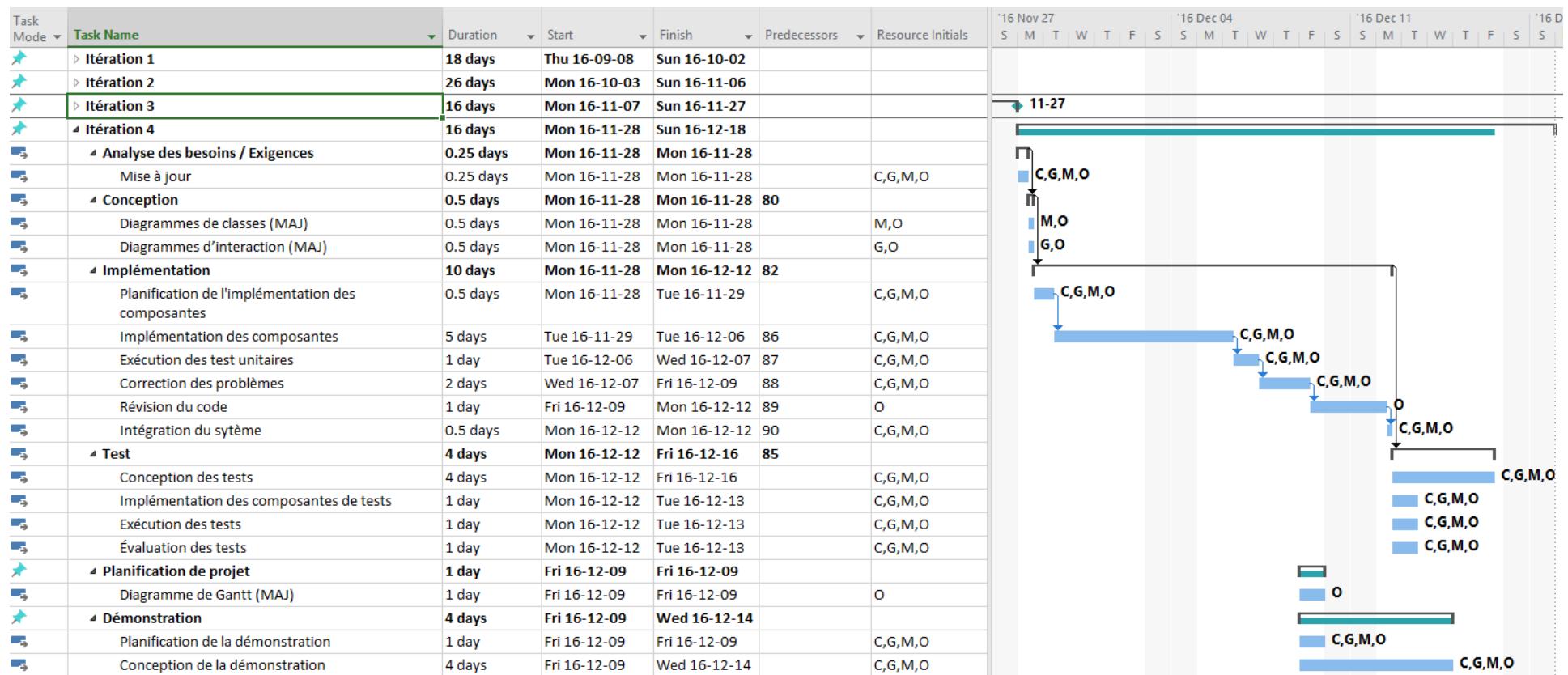
## 5.2. ITÉRATION 2



### 5.3. ITÉRATION 3



## 5.4. ITÉRATION 4



## 6. ANNEXE : LIVRABLE 1

### 6.1. VISION

#### 6.1.1. INTRODUCTION

L'AEMQ donne le mandat à GLO-2004-EQ10 la réalisation d'une application permettant de simplifier l'enseignement des stratégies de jeu pour les entraîneurs (*VisuaLigue*, VL). Cette application permettra notamment la création de stratégies de jeux de façon dynamique et interactive.

#### 6.1.2. POSITIONNEMENT

##### 6.1.2.1. OPPORTUNITÉ D'AFFAIRES

Il existe présentement quelques solutions sur le marché (*Daximation*, *DrillDraw*, *HockeyShare*, etc.), mais ces produits sont relativement incomplets (ils n'offrent pas les fonctionnalités que l'AEMQ demande) et non intuitifs pour la plupart. L'AEMQ mandate GLO-2004-EQ10 la réalisation de ce produit.

##### 6.1.2.2. ÉNONCÉ DU PROBLÈME

Les entraîneurs de l'AEMQ effectuent présentement des dessins sur un tableau blanc pour expliquer aux membres de l'équipe les stratégies qu'ils devront appliquer dans certaines situations. Il n'est pas facile de comprendre ce que l'entraîneur veut enseigner aux joueurs et il n'est pas si simple de visualiser les stratégies sur ce support visuel. Le produit à développer vise à faciliter cette visualisation.

##### 6.1.2.3. ÉNONCÉ DE POSITION DE PRODUIT

Le système cible les entraîneurs de hockey désirant créer et visualiser des stratégies de jeu, mais aussi les entraîneurs de tout sport d'équipe nécessitant l'élaboration de stratégies. Le produit pourra permettre, à l'aide d'une interface visuelle épurée, dynamique et adaptée à tout support visuel, la création et la visualisation de stratégies de jeu selon différents modes, dont un mode temps réel.

### 6.1.3. DESCRIPTIONS DES INTERVENANTS

#### 6.1.3.1. RÉSUMÉ DES PARTIES PRENANTES (NON-UTILISATEUR)

**Président de l'AEMQ** (Association des entraîneurs mineurs du Québec) : Le client du l'équipe ayant donné le mandat de développement du produit à GLO-2004-EQ10.

**Martin Savoie** : L'intermédiaire entre le président de l'AEMQ (le client) et GLO-2004-EQ10 (l'entreprise de développement).

**GLO-2004-EQ10** : L'entreprise ayant été mandatée par le président de l'AEMQ (le client) pour le développement de l'application. L'équipe de développement est composée des membres énumérés en page titre de ce document.

#### 6.1.3.2. RÉSUMÉ DES UTILISATEURS

**Entraîneur** : C'est celui qui élabore les stratégies de jeu. C'est le principal utilisateur de l'application.

**Joueurs** : Ce sont les joueurs de l'équipe qui visualisent, à l'aide de l'application développée, les stratégies élaborées par l'entraîneur.

#### 6.1.3.3. BUT DES ACTEURS

**Entraîneur** : Créer un sport, créer des stratégies/jeux, visualiser les stratégies/jeux, exporter les jeux.

**Joueurs** : Observer les stratégies/jeux.

#### 6.1.4. HYPOTHÈSES ET CONTRAINTES

- L'équipe de développement dispose d'un délai raisonnable pour réaliser le projet dans son ensemble.
- Une solution au problème posé est possible.

#### 6.1.5. COÛT, PRIX, LICENCES ET INSTALLATION

Le logiciel est développé gratuitement dans le cadre du cours. Se référer aux responsables du cours pour les détails concernant la licence d'utilisation et l'installation du logiciel.

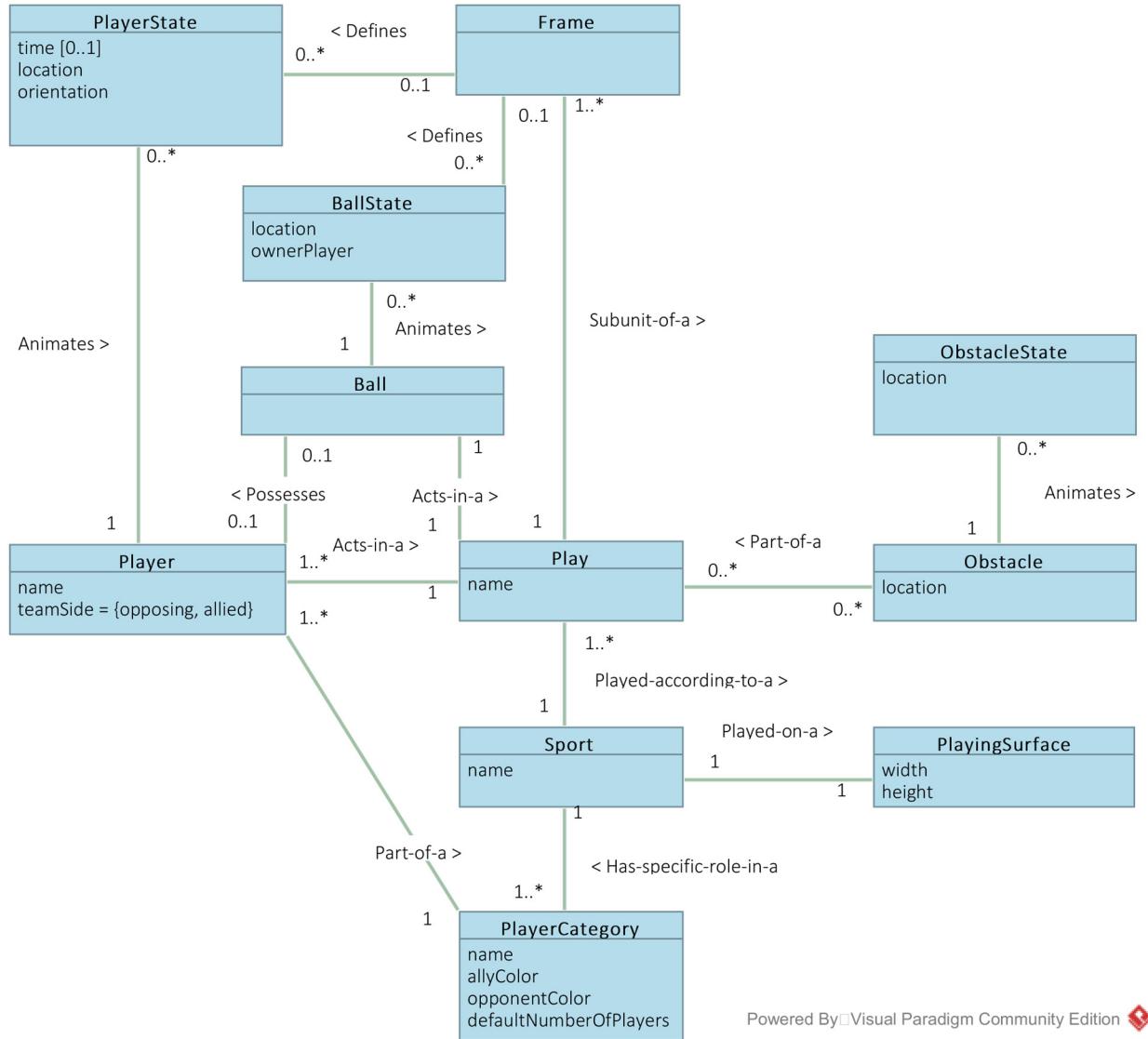
### 6.1.6. RÉSUMÉ DES CARACTÉRISTIQUES DU SYSTÈME

- Création et sauvegarde de jeux
- Stratégies visuellement accessibles au chargement de l'application, avec aperçu et titre
- Support de 2 modes de création pour les stratégies : mode image par image et mode temps réel.
- Contrôle du visionnement (débuter, pause, reculer, avancer et ajustement de vitesse)
- Doit être réutilisable pour d'autres sports : Création d'un sport, ajout d'une image de terrain, spécification des dimensions réelles, définition du nombre de joueurs et des catégories.
- Support pour annuler/rétablissement (undo/redo) les actions
- Exportation des stratégies dans un format image
- Zoom
- Affichage des coordonnées de la souris en unité réelles
- Ajustement de l'orientation des joueurs
- Affichage/masquage en tout temps du rôle des joueurs
- Ajout, édition et suppression d'obstacles.

### 6.1.7. SOLUTION PROPOSÉE

Un logiciel en Java 8 multiplateforme et parfaitement adaptée aux besoins de l'AEMQ. Celui-ci permettra la gestion des sports et permettra la création de stratégies de jeu de façon dynamique, fluide, intuitive et efficace. Il sera doté d'une interface utilisateur moderne et épurée avec une disposition s'adaptant à tous les appareils. Cette interface sera adaptée autant à la navigation tactile qu'à la navigation à l'aide d'une souris conventionnelle.

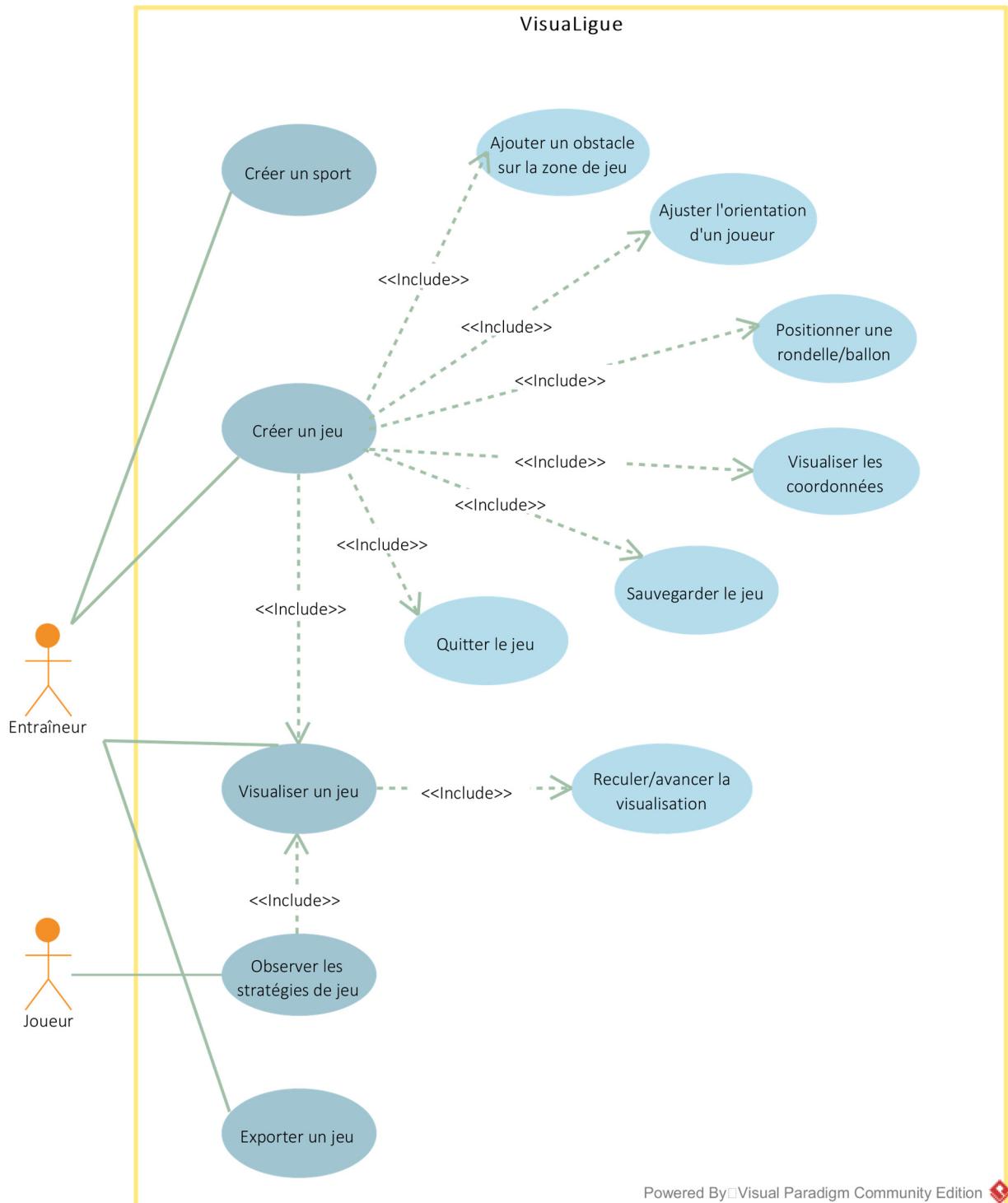
## 6.2. MODÈLE DU DOMAINÉ



Powered By Visual Paradigm Community Edition

## 6.3. CAS D'UTILISATION (MIS À JOUR)

### 6.3.1. DIAGRAMME DES CAS D'UTILISATION



### 6.3.2. TEXTE DES CAS D'UTILISATION

Cas d'utilisation : <b>Créer un sport</b>	
Système :	Application <i>VisuaLigue</i>
Niveau :	But de l'utilisateur
Acteurs :	Entraîneur
Parties prenantes et intérêts :	<b>Entraîneur</b> : Il désire créer un sport inexistant dans le système.
Préconditions :	Un sport du même nom n'existe pas dans le système.
Garanties en cas de succès :	Le sport est créé et il est possible de créer des stratégies en utilisant ce sport.
Scénario principal :	<ol style="list-style-type: none"> <li>1. L'entraîneur entre le nom du sport.</li> <li>2. L'entraîneur ajoute une image représentant le terrain.</li> <li>3. L'entraîneur spécifie les dimensions du terrain en valeurs réelles.</li> <li>4. L'entraîneur définit les catégories de joueurs.             <ol style="list-style-type: none"> <li>4.1. L'entraîneur ajoute une nouvelle catégorie de joueurs.</li> <li>4.2. L'entraîneur spécifie le nom de la catégorie.</li> <li>4.3. L'entraîneur spécifie une couleur pour les joueurs alliés et une autre pour les joueurs adverses.</li> <li>4.4. L'entraîneur définit le nombre de joueurs par défaut pour chaque catégorie de joueurs.</li> </ol> </li> </ol>
Scénarios alternatifs :	<p>1a. Le nom du sport existe déjà :</p> <ol style="list-style-type: none"> <li>1. L'entraîneur doit spécifier un autre nom.</li> </ol> <p>*a. En tout temps, l'entraîneur annule l'opération</p>

Cas d'utilisation : <b>Créer un jeu</b>	
Système :	Application <i>VisuaLigue</i>
Niveau :	But de l'utilisateur
Acteurs :	Entraîneur
Parties prenantes et intérêts :	<b>Entraîneur</b> : Il désire créer un jeu dans le système qui lui permettra ensuite de le visualiser.
Préconditions :	Le sport pour lequel le jeu veut être créé existe.
Garanties en cas de succès :	Le jeu est créé et il est possible de le visualiser par la suite.

Scénario principal :	<ol style="list-style-type: none"> <li>1. L'entraîneur demande de créer un jeu.</li> <li>2. L'entraîneur spécifie le sport pour lequel il veut créer le jeu.</li> <li>3. L'entraîneur spécifie le titre du jeu.</li> <li>4. L'entraîneur crée le déroulement de la stratégie :             <ol style="list-style-type: none"> <li>4.1. L'entraîneur positionne un joueur (depuis la liste des joueurs) sur la surface de jeu.</li> <li>4.1.a. En mode image par image :                     <ol style="list-style-type: none"> <li>1a. L'entraîneur déplace un joueur existant sur la surface de jeu, si joueur il y a.</li> <li>1b. L'entraîneur ajoute une image (fait avancer l'image).</li> </ol> </li> <li>4.1.b. En mode temps réel :                     <ol style="list-style-type: none"> <li>1a. L'entraîneur déplace le joueur et lui fait effectuer un mouvement qui sera enregistré en temps réel pendant que les autres joueurs se déplacent.</li> </ol> </li> <li>4.2. L'entraîneur peut répéter cette dernière étape autant de fois qu'il le désire, jusqu'à ce que tous les mouvements formant la stratégie soient définis.</li> </ol> </li> </ol>
Scénarios alternatifs :	<p>*a. En tout temps, l'entraîneur peut activer/désactiver le mode temps réel : Inclure <b>Error! Reference source not found..</b></p> <p>*b. En tout temps, l'entraîneur peut annuler/rétablissement une action : Inclure <b>Error! Reference source not found..</b></p> <p>*c. En tout temps, l'entraîneur peut afficher/cacher le rôle des joueurs : Inclure <b>Error! Reference source not found..</b></p> <p>*d. En tout temps, l'entraîneur peut agrandir/rétrécir la surface de jeu: Inclure <b>Error! Reference source not found..</b></p> <p>*e. En tout temps, l'entraîneur peut ajouter un obstacle sur la surface de jeu : Inclure <b>Ajouter un obstacle sur la zone de jeu.</b></p> <p>*f. En tout temps, l'entraîneur peut positionner une rondelle/ballon sur la surface de jeu : Inclure <b>Positionner une rondelle/ballon.</b></p> <p>*g. En tout temps, l'entraîneur peut visualiser les coordonnées de la souris en unités réelles : Inclure <b>Visualiser les coordonnées.</b></p> <p>*h. En tout temps, l'entraîneur peut ajuster l'orientation d'un joueur : Inclure <b>Ajuster l'orientation d'un joueur.</b></p> <p>*i. En tout temps, l'entraîneur peut visualiser le jeu en cours de création : Inclure <b>Visualiser un jeu.</b></p> <p>*j. En tout temps, l'entraîneur peut sauvegarder le jeu créé : Inclure <b>Sauvegarder le jeu.</b></p> <p>*k. En tout temps, l'entraîneur peut quitter la création du jeu Inclure <b>Quitter le jeu.</b></p>

Cas d'utilisation : Ajouter un obstacle sur la zone de jeu	
Niveau :	Sous-fonction
Préconditions :	<ul style="list-style-type: none"> <li>▪ Un jeu est en cours de création/visualisation.</li> <li>▪ Au moins un obstacle a été créé.</li> </ul>
Garanties en cas de succès :	<ul style="list-style-type: none"> <li>▪ L'obstacle est ajouté à l'endroit spécifié par l'utilisateur.</li> <li>▪ L'obstacle est présent sur toutes les images.</li> <li>▪ L'obstacle ne bouge pas pendant la visualisation.</li> </ul>
Scénario principal :	<ol style="list-style-type: none"> <li>1. L'entraîneur sélectionne un obstacle dans la liste.</li> <li>2. L'entraîneur positionne l'obstacle sur la surface de jeu.</li> </ol>
Scénarios alternatifs :	<ol style="list-style-type: none"> <li>1a. L'entraîneur définit un nouvel obstacle:             <ol style="list-style-type: none"> <li>1. L'entraîneur spécifie le nom du nouvel obstacle.</li> <li>2. L'entraîneur ajoute l'image représentant l'obstacle.</li> </ol> </li> </ol>

Cas d'utilisation : Ajuster l'orientation d'un joueur	
Niveau :	Sous-fonction
Préconditions :	<ul style="list-style-type: none"> <li>▪ Un jeu est en cours de création/visualisation.</li> </ul>
Garanties en cas de succès :	<ul style="list-style-type: none"> <li>▪ L'orientation du joueur est modifiée.</li> <li>▪ La modification de cette orientation ne modifie pas la direction de déplacement du joueur.</li> </ul>
Scénario principal :	<ol style="list-style-type: none"> <li>1. L'application offre un contrôle permettant d'effectuer la rotation du joueur lorsque celui-ci est survolé.</li> <li>2. L'entraîneur effectue la rotation désirée du joueur.</li> </ol>

Cas d'utilisation : Positionner une rondelle/ballon	
Niveau :	Sous-fonction
Préconditions :	<ul style="list-style-type: none"> <li>▪ Un jeu est en cours de création/visualisation.</li> </ul>
Garanties en cas de succès :	<ul style="list-style-type: none"> <li>▪ La rondelle/ballon est positionnée à l'endroit spécifié par l'utilisateur.</li> <li>▪ La position n'affecte pas les autres images (en mode image par image) ou les autres séquences (en mode temps réel).</li> </ul>
Scénario principal :	<ol style="list-style-type: none"> <li>1. L'entraîneur sélectionne une nouvelle rondelle/ballon.</li> <li>2. L'entraîneur met la rondelle/ballon en possession d'un joueur.</li> </ol>

Scénarios alternatifs :	1a. L'entraîneur sélectionne une rondelle/ballon déjà sur la surface de jeu. 2a. L'entraîneur positionne librement la rondelle sur la surface de jeu.
-------------------------	--

Cas d'utilisation : Visualiser les coordonnées	
Niveau :	Sous-fonction
Préconditions :	<ul style="list-style-type: none"> <li>▪ Un jeu est en cours de création/visualisation.</li> <li>▪ La souris est à l'intérieur de la surface de jeu.</li> </ul>
Garanties en cas de succès :	<ul style="list-style-type: none"> <li>▪ Les coordonnées de la souris sont affichés en unités réelles.</li> </ul>
Scénario principal :	1. L'application affiche les coordonnées de la souris en tout temps lors de la création/visualisation d'un jeu.
Scénarios alternatifs :	1a. La souris est à l'extérieur de la surface de jeu : <ul style="list-style-type: none"> <li>1. La dernière coordonnée valide est affichée.</li> </ul>

Cas d'utilisation : Sauvegarder le jeu	
Niveau :	Sous-fonction
Préconditions :	<ul style="list-style-type: none"> <li>▪ Un jeu est en cours de création/visualisation.</li> </ul>
Garanties en cas de succès :	<ul style="list-style-type: none"> <li>▪ Le jeu est sauvegardé dans son état actuel.</li> <li>▪ L'état est rétabli lorsque le jeu est ré-ouvert.</li> </ul>
Scénario principal :	1. L'entraîneur sauvegarde le jeu.

Cas d'utilisation : Quitter le jeu	
Niveau :	Sous-fonction
Préconditions :	<ul style="list-style-type: none"> <li>▪ Un jeu est en cours de création/visualisation.</li> </ul>
Garanties en cas de succès :	<ul style="list-style-type: none"> <li>▪ Le jeu quitte.</li> </ul>
Scénario principal :	<ul style="list-style-type: none"> <li>▪ L'entraîneur quitte le jeu.</li> </ul>
Scénarios alternatifs :	1a. Le jeu a subi des modifications depuis la dernière sauvegarde ou n'a jamais été sauvegardé :

	1. L'application demande à l'utilisateur s'il désire sauvegarder son jeu.
--	---

Cas d'utilisation :	Visualiser un jeu
Système :	Application <i>VisuaLigue</i>
Niveau :	But de l'utilisateur
Acteurs :	Entraîneur
Parties prenantes et intérêts :	<b>Entraîneur</b> : Il désire visualiser un jeu créé dans le système.
Préconditions :	<ul style="list-style-type: none"> <li>▪ Un jeu est ouvert.</li> <li>▪ La visualisation est arrêtée.</li> <li>▪ Il y a au moins 2 images de définies (mode image par image) ou un mouvement effectué (mode temps réel).</li> </ul>
Garanties en cas de succès :	<ul style="list-style-type: none"> <li>▪ La visualisation débute.</li> <li>▪ Les joueurs et rondelles/balles se déplacent selon les positions définies lors de la création du jeu.</li> </ul>
Scénario principal :	<ol style="list-style-type: none"> <li>1. L'entraîneur débute la visualisation.</li> <li>2. La visualisation s'arrête automatiquement lorsque la fin de la séquence de jeu est atteinte.</li> </ol>
Scénarios alternatifs :	<p>*a : En tout temps, l'entraîneur peut arrêter la visualisation : Inclure <i>Error! Reference source not found.</i></p> <p>*b. En tout temps, l'entraîneur recule/avancer la visualisation : Inclure <i>Reculer/avancer la visualisation.</i></p> <p>*c. En tout temps, l'entraîneur peut modifier le temps de recul/avancement prédéfini : Inclure <i>Error! Reference source not found..</i></p>

Cas d'utilisation :	Reculer/avancer la visualisation
Niveau :	Sous-fonction
Préconditions :	<ul style="list-style-type: none"> <li>▪ Un jeu est ouvert.</li> <li>▪ Il y a au moins 2 images de définies (mode image par image) ou un mouvement effectué (mode temps réel).</li> </ul>
Garanties en cas de succès :	<ul style="list-style-type: none"> <li>▪ La visualisation recule ou avance, de façon continue ou prédéfinie.</li> </ul>

	<ul style="list-style-type: none"> <li>▪ Pour recul/avancement linéaire, les mouvements des joueurs et rondelles/balles se font en vitesse proportionnelle par rapport à la vitesse de recul/avancement.</li> <li>▪ Pour un recul/avancement prédéfini, l'état est rétabli au moment défini par le temps d'avancement/recul prédéfini.</li> </ul>
Scénario principal :	<ol style="list-style-type: none"> <li>1. L'entraîneur avance ou recule la visualisation, de façon linéaire ou prédéfinie.</li> </ol>
Scénarios alternatifs :	<p>*a. En tout temps, l'entraîneur peut accélérer ou réduire la vitesse de recul/avancement.</p>

Cas d'utilisation : <b>Observer les stratégies de jeu</b>	
Système :	Application <i>VisuaLigue</i>
Niveau :	But de l'utilisateur
Acteurs :	Joueur
Parties prenantes et intérêts :	<b>Entraîneur</b> : Il désire montrer les stratégies créées à ses joueurs. <b>Joueur</b> : Ils désirent comprendre la stratégie créée par l'entraîneur.
Préconditions :	<ul style="list-style-type: none"> <li>▪ Un jeu est ouvert.</li> </ul>
Garanties en cas de succès :	<ul style="list-style-type: none"> <li>▪ Les joueurs ont pu observer le jeu.</li> </ul>
Scénario principal :	<ol style="list-style-type: none"> <li>1. L'entraîneur sélectionne le jeu désiré dans la liste des jeux (si un jeu n'est pas déjà ouvert).</li> <li>2. L'entraîneur effectue les actions spécifiées dans <i>Visualiser un jeu</i>.</li> </ol>
Scénarios alternatifs :	Aucun.

Cas d'utilisation : <b>Exporter un jeu</b>	
Système :	Application <i>VisuaLigue</i>
Niveau :	But de l'utilisateur
Acteurs :	Entraîneur
Parties prenantes et intérêts :	<b>Entraîneur</b> : Il désire exporter un jeu créé dans le système.
Préconditions :	<ul style="list-style-type: none"> <li>▪ Le jeu a été créé (partiellement ou complètement).</li> </ul>

Garanties en cas de succès :	<ul style="list-style-type: none"> <li>▪ Le jeu est exporté dans un format d'image (png, jpeg, etc.) avec une visualisation du déplacement des joueurs par des flèches.</li> </ul>
Scénario principal :	<ol style="list-style-type: none"> <li>1. L'entraîneur sélectionne le jeu désiré dans la liste des jeux (si un jeu n'est pas déjà ouvert).</li> <li>2. L'entraîneur exporte le jeu.             <ol style="list-style-type: none"> <li>2.1. L'entraîneur choisit le format de fichier.</li> <li>2.2. L'entraîneur choisit le nom de répertoire de destination et le nom de fichier pour l'exportation.</li> </ol> </li> </ol>
Scénarios alternatifs :	<ol style="list-style-type: none"> <li>2a. Un erreur d'entrée-sortie survient lors de l'exportation :             <ol style="list-style-type: none"> <li>1. L'exportation est annulée.</li> <li>2. L'entraîneur est notifié de l'erreur.</li> </ol> </li> </ol>

---

## 6.4. GLOSSAIRE (MIS À JOUR)

### 6.4.1. CORRESPONDANCE ANGLOPHONE ET FRANCOPHONE DES TERMES

**Note :** Un mot barré signifie un terme synonyme utilisé dans la définition de projet, mais qui ne sera pas utilisé par souci de cohérence.

Anglais	Français	Français	Anglais
Allied	Allié	Action	Action → Movement
Actor	Acteur	Acteur	Acteur
Actor Instance	Instance d'un acteur	Allié	Allié
Actor State	État d'un acteur	Application	Application
Actor Timeline	Ligne du temps d'un acteur	Avancer	Fast Forward
Application	Application/Système	Balle	Ball
Ball	Balle/Disque/Rondelle/Ballon	Ballon	Ball
Coach	Entraîneur	Catégorie de joueur	Player Category
Creation Mode	Mode de création	Côté de jeu	Playing Side
Drawings	Dessins	Débuter	Start
Fast Forward	Avancer	Déplacement	Movement
Fast Forward Speed	Vitesse d'avancement	Dessins	Drawings
Frame	Image	Direction du joueur	Player Direction
Frame by Frame	Image par image	Disque	Puck → Ball
Keyframe	Image-clé	Écran	Screen
Mouse	Souris	Équipe	Team
Movement	Mouvement/Déplacement/Action	Entraîneur	Coach
Opposing	Opposé	État d'un acteur	Actor State
Pause	Pause	Image	Frame
Play	Jeu/Stratégie	Image-clé	Keyframe

<b>Player</b>	Joueur	<b>Image par image</b>	Frame by Frame
<b>Player Category</b>	Catégorie de joueur	<b>Instance d'un acteur</b>	Actor Instance
<b>Player Direction</b>	Direction du joueur	<b>Jeu</b>	Play
<b>Player Orientation</b>	Orientation du joueur	<b>Joueur</b>	Player
<b>Player Position</b>	Position du joueur	<b>Ligne du temps</b>	Timeline
<b>Playing Side</b>	Côté de jeu	<b>Ligne du temps d'un acteur</b>	Actor Timeline
<b>Playing Surface</b>	Surface de jeu/Terrain	<b>Mode de création</b>	Creation Mode
<b>Real Time</b>	Temps réel	<b>Mouvement</b>	Movement
<b>Rewind Speed</b>	Vitesse de recul	<b>Opposé</b>	Opposing
<b>Rewind</b>	Reculer	<b>Orientation du joueur</b>	Player Orientation
<b>Screen</b>	Écran	<b>Pause</b>	Pause
<b>Sequence</b>	Séquence	<b>Position du joueur</b>	Player Position
<b>Sport</b>	Sport	<b>Reculer</b>	Rewind
<b>Start</b>	Débuter	<b>Rondelle</b>	Puck → Ball
<b>Team</b>	Équipe	<b>Souris</b>	Mouse
<b>Timeline</b>	Ligne du temps	<b>Sport</b>	Sport
		<b>Stratégie</b>	Strategy → Jeu
		<b>Surface de jeu</b>	Playing Surface
		<b>Séquence</b>	Sequence
		<b>Système</b>	System → Application
		<b>Temps réel</b>	Real Time
		<b>Terrain</b>	Field → Playing Surface
		<b>Vitesse de recul</b>	Rewind Speed
		<b>Vitesse d'avancement</b>	Fast Forward Speed

#### 6.4.2. DÉFINITIONS

**Action** : Voir *Mouvement*.

**Allié** : Joueur allié à l'équipe pour laquelle le jeu est visionné.

**Application** : L'ensemble du système devant être développé.

**Avancer** : Action d'avancer le visionnement du jeu d'un intervalle quelconque.

**Balle** : Voir *Ballon*.

**Ballon** : Objet échangé entre les joueurs lors d'un jeu. Ce terme sera aussi utilisé pour tous les sports utilisant d'autres formes d'objet d'échanges, tels un disque ou une rondelle.

**Catégorie de joueur** : Définit la catégorie d'un joueur relatif au sport présenté (exemple pour le hockey: gardien, ailier, centre, bloqueur, attaquant, etc.)

**Côté de jeu** : Parti pris du joueur dans un sport. Un joueur fait partie de l'équipe alliée (équipe pour laquelle le jeu est visionné) ou de l'équipe opposée.

**Débuter** : Action de débuter le jeu affiché. Durant cette action, le déplacement des joueurs se fait à l'écran.

**Déplacement** : Voir *Mouvement*.

**Dessins** : Illustrations à l'aide de formes, flèches ou autres éléments visuels dessinées par l'entraîneur sur la surface de jeu.

**Direction du joueur** : La position vers laquelle le joueur se dirige.

**Disque** : Voir *Ballon*.

**Écran** : L'appareil utilisé pour afficher les stratégies/jeu au joueur et qui permet l'affichage des interactions avec le système par l'entraîneur ou tout autre utilisateur.

**Entraîneur** : Personne qui, par des exercices gradués, entraîne l'équipe et les prépare à une compétition. Dans le cadre de ce projet, c'est l'utilisateur qui interagira principalement avec l'application.

**Équipe** : Ensemble des joueurs associés ou du même camp dans le cadre de l'illustration d'une stratégie.

**État d'acteur** : Défini l'état d'un acteur (ex. : la position ou l'orientation d'un joueur, d'un obstacle ou d'une rondelle). **La position d'un joueur est définie en coordonnées taille-relative** (« size-relative position »), c'est-à-dire entre 0 et 1 en X et entre 0 et 1 en Y. 0 étant le début du terrain, 1 étant la fin du terrain. Les coordonnées sont indépendantes des pixels et des dimensions réelles du terrain.

**Image** : Représente un point bien défini sur la ligne de temps où les joueurs sont positionnés et organisés d'une façon particulière. Ne pas confondre avec l'image affichée ou l'image d'une surface de jeu.

**Image-clé** : Associe l'instance d'un acteur à un état de cet acteur à un temps précis (ex. : la position d'un joueur au temps 1000 ms).

**Image par image** : Mode de création de jeu permettant de positionner les joueurs sur la zone de jeu individuellement pour chaque image. Lors de la création d'une nouvelle image, l'utilisateur clique sur un joueur et le glisser vers la prochaine position.

**Instance d'acteur** : Instance d'un acteur dans le jeu. Ce peut être une instance de joueur, d'un obstacle ou d'une balle/rondelle. Il peut y avoir plusieurs instances d'un même joueur, obstacle ou rondelle dans un même jeu (plusieurs ailiers gauches par exemple).

**Jeu** : Un ensemble d'action ou bien de déplacement qui doivent être effectués par les joueurs lors d'une situation précise.

**Joueur** : Personne pratiquant le sport pour lequel la stratégie est réalisée et prenant part à celle-ci en tant qu'acteur.

**Ligne de temps** : Représentation linéaire des ensembles de mouvements positionnés dans le temps ; elle associe les mouvements des joueurs à leurs positions dans le temps le long d'une échelle graduée, ce en quoi elle se rapproche d'une chronologie. Dans le mode *image par image*, une position peut se définir en terme d'images (exemple : image numéro 5) ou bien en terme de temps (exemple : 5 secondes du début). Dans le mode *temps réel*, une position se définit en terme de temps seulement.

**Ligne de temps d'acteur** : Ligne de temps de l'instance d'un acteur. Chaque instance d'acteur possède sa ligne de temps. Une ligne de temps contient plusieurs « keyframes » définissant les états d'un acteur dans le temps.

**Mode de création** : Mode de création de jeu, soit le mode *image par image* ou le mode *temps réel*.

**Mouvement** : Déplacement individuel d'un joueur du point A au point B.

**Opposé** : Joueur opposé à l'équipe pour laquelle le jeu est visionné.

**Orientation du joueur** : Orientation statique du joueur par rapport à son environnement, indépendamment du mouvement.

**Pause** : Action d'arrêter le visionnement du jeu en cours.

**Position du joueur** : Position d'un joueur en coordonnées cartésiennes sur la surface de jeu à moment donné.

**Reculer** : Action de reculer le visionnement du jeu d'un intervalle quelconque.

**Rondelle** : Voir *Ballon*.

**Souris** : Élément d'entrée principal permettant l'interaction avec l'application.

**Sport** : Ensemble des exercices physiques se présentant sous forme de jeux individuels ou collectifs, donnant généralement lieu à compétition, pratiqués en observant certaines règles précises. Dans ce projet, le sport représente le cadre dans lequel se déroule une stratégie.

**Stratégie** : Voir *Jeu*.

**Surface de jeu** : Espace cartésien délimitant le déroulement d'un jeu et dans lequel est restreinte la position des joueurs.

**Séquence** : Succession d'images dans un temps fini constituant un sous-ensemble d'un jeu.

**Temps réel** : Mode de création de jeu permettant le mouvement d'un joueur en temps réel, de façon synchrone et dont le début et la fin du mouvement ne sont pas nécessairement coordonnés avec les autres joueurs. Dès qu'il clique sur le joueur en question, l'utilisateur peut déplacer le joueur et lui faire effectuer un mouvement qui sera enregistré en temps réel. Lorsque l'utilisateur clique sur un joueur, la simulation démarre et les autres joueurs se déplacent. Il peut donc faire déplacer le joueur préalablement sélectionné tout en voyant le déplacement des autres joueurs.

**Terrain** : Voir *Surface de jeu*.

**Vitesse de recul** : Vitesse à laquelle le visionnement du jeu est accéléré lors de l'action de recul.

**Vitesse d'avancement** : Vitesse à laquelle le visionnement du jeu est accéléré lors de l'action d'avancement.