

# Rapport

Moteur 3D - 7Physics



**Équipe 3 : Noa AMMIRATI, Fanny DELNONDEDIEU, Quentin GENDARME, Pierre LOTTE, Théo PIROUELLE, Éléa TURC**



ENSEEIH  
Département Sciences du Numérique  
1APP SN 2020-2021

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Principales fonctionnalités</b>	<b>4</b>
2.1	Sprint 0 . . . . .	4
2.1.1	Affichage d'une scène 3D . . . . .	4
2.1.2	Manipulation de la caméra . . . . .	4
2.2	Sprint 1 . . . . .	4
2.2.1	Ajout d'un objet 3D . . . . .	4
2.2.2	Amélioration de la caméra . . . . .	5
2.2.3	Création de l'interface graphique . . . . .	5
2.3	Sprint 2 . . . . .	5
2.3.1	Création des principes de base du moteur 3D . . . . .	5
2.3.2	Ajout d'une possibilité de sélection des objets . . . . .	6
2.3.3	Manipulation des forces dans l'interface graphique . . . . .	6
2.3.4	Possibilité de lancer la simulation . . . . .	6
2.4	Sprint 3 . . . . .	6
2.4.1	Ajout de nouvelles formes prédéfinies . . . . .	7
2.4.2	Prévisualisation d'un objet 3D . . . . .	7
2.4.3	Importation et exportation d'un projet *.obj . . . . .	7
2.4.4	Détection des collisions . . . . .	7
<b>3</b>	<b>Découpage de l'application</b>	<b>8</b>
<b>4</b>	<b>Architecture de l'application</b>	<b>9</b>
4.1	7Physics . . . . .	10
4.1.1	GUI . . . . .	10
4.1.2	Files . . . . .	10
4.1.3	Tests . . . . .	10
4.2	Common . . . . .	10
4.2.1	Logger . . . . .	10
4.2.2	Types utilitaires . . . . .	10
4.2.3	Formes prédéfinies . . . . .	10
4.2.4	Tests . . . . .	11
4.3	Engine . . . . .	11
4.3.1	PhysicObject . . . . .	11
4.3.2	World . . . . .	11
4.3.3	Tests . . . . .	11
4.4	Renderer . . . . .	11
4.4.1	Camera . . . . .	11
4.4.2	Object3D . . . . .	11
4.4.3	Scene3D . . . . .	11
4.4.4	Ground . . . . .	11
4.4.5	Tests . . . . .	11
4.4.6	Interaction avec OpenGL . . . . .	11

<b>5</b>	<b>Principaux choix</b>	<b>12</b>
5.1	Conception . . . . .	12
5.1.1	Création de la maquette IHM . . . . .	12
5.1.2	Création d'un diagramme de classe . . . . .	12
5.2	Réalisation . . . . .	12
5.3	Problèmes rencontrés et solutions apportées . . . . .	12
5.3.1	Rafraîchissement du rendu graphique . . . . .	12
5.3.2	Caméra fixée sur la scène . . . . .	12
5.3.3	Rendu pixelisé . . . . .	13
5.3.4	Problèmes de simulation . . . . .	13
<b>6</b>	<b>Organisation de l'équipe et mise en oeuvre des méthodes agiles</b>	<b>14</b>
6.1	Mise en place du projet . . . . .	14
6.2	Estimation des points d'efforts et de valeur métier . . . . .	14
6.2.1	Points d'efforts . . . . .	14
6.2.2	Points de valeur métier . . . . .	14
6.3	Utilisation des cérémonies agiles . . . . .	14
6.3.1	Sprint planning . . . . .	14
6.3.2	Daily scrum . . . . .	15
6.3.3	Sanity Check . . . . .	15
6.3.4	Sprint Review . . . . .	15
6.3.5	Retrospective . . . . .	15
6.4	Choix des tâches à réaliser . . . . .	15
<b>7</b>	<b>Conclusion</b>	<b>16</b>

# 1. Introduction

L'idée de ce projet est de réaliser un moteur 3D permettant de réaliser des simulations de notions de physique élémentaires telles que la gravité, les collisions, etc.

Ce projet pourrait alors se séparer en 2 objectifs principaux qui sont aussi les 2 briques principales nécessaires à sa réalisation :

- Créer une bibliothèque de rendu des formes tri-dimensionnelles basiques telles que des cubes, sphères, pyramides, etc. Tout cela avec possibilité de changer le point de vue de l'utilisateur en se déplaçant dans l'espace (concept de caméra à la première personne). Ajouté à cela, il est possible de créer plusieurs fonctionnalités visuelles telles que la présence d'ombre et de lumière, de texture, etc. Afin de créer des objets 3D réalistes.
- Créer un moteur physique responsable de la simulation des concepts évoqués plus tôt (gravité, collisions, etc.). Tous ces concepts pourront alors être manipulés à souhait grâce à des notions de vitesse, de poids, etc. Qui sont autant de paramètres influant sur ces phénomènes physiques.

## 2. Principales fonctionnalités

### 2.1 Sprint 0

Puisque nous avons du temps après la mise en place du projet, nous avons pu mettre en place des fonctionnalités basiques permettant la compréhension de ce que nous allions développer par la suite. Nous avons donc cherché à pouvoir nous repérer dans l'espace.

#### 2.1.1 Affichage d'une scène 3D

Une des premières fonctionnalités à implanter a été la création d'une scène 3D. Cette scène 3D est constituée d'un sol représentant une grille blanche sur fond gris et d'un ciel bleu. Cette scène a pour but de permettre à l'utilisateur de mieux comprendre l'orientation des objets, le placement de sa caméra, etc. Cela permettra alors, lors de simulation physiques, de mieux comprendre les résultats de celles-ci.



Cette fonctionnalité a été complètement réalisée durant cette itération.

#### 2.1.2 Manipulation de la caméra

Par la suite, et dans l'objectif de pouvoir observer la scène sous plusieurs angles, nous avons implanté une première version de gestion de la caméra. Grâce à cela, il nous était alors possible de nous déplacer dans la scène grâce aux raccourcis clavier que nous avons définis.



Cette fonctionnalité a été complètement réalisée durant cette itération.

### 2.2 Sprint 1

L'objectif du sprint 1 a été de fournir une première version de l'interface utilisateur lui permettant d'ajouter des formes 3D prédéfinies à la scène et de les visualiser sous différents angles facilement et intuitivement.

#### 2.2.1 Ajout d'un objet 3D

Tout d'abord, nous avons rendu possible l'ajout de 2 formes prédéfinies qui sont le cube et la sphère.



Cette fonctionnalité a été complètement réalisée durant cette itération.

### 2.2.2 Amélioration de la caméra

Nous nous sommes aperçus que les raccourcis claviers mis en place lors du sprint 0 étaient difficilement utilisables. De plus, la première version ne consistait pas vraiment à faire bouger la caméra mais plutôt à faire tourner la scène sur elle-même. Nous avons donc décidé de changer de manière d’implanter la fonctionnalité et de permettre par la même occasion l’utilisation de la souris pour une meilleure expérience utilisateur.



Cette fonctionnalité a été complètement réalisée durant cette itération.

### 2.2.3 Création de l’interface graphique

Afin de pouvoir une première version utilisable de notre application, nous avons créé, avec l’appui d’une maquette créée lors du Sprint 0, la base de l’interface utilisateur. Pour commencer, nous avons mis en place la structure de la fenêtre. Puis, nous avons ajouté les composants nécessaires à l’utilisation des fonctionnalités développées jusqu’ici. La scène 3D a donc pu être intégrée à la fenêtre et des boutons et formulaires ont été ajoutés pour que l’utilisateur puisse facilement ajouter des formes ou manipuler la caméra.



Cette fonctionnalité a été complètement réalisée durant cette itération.

## 2.3 Sprint 2

L’objectif de ce sprint a été de faire bouger les objets en ajoutant les fonctionnalités liées au moteur physique. Pour cela, nous avons modéliser les concepts de force, d’objets et de monde physique.

### 2.3.1 Création des principes de base du moteur 3D

Afin de compléter les fonctionnalités voulues de notre projet, nous avons commencé à mettre en place les briques nécessaires à la simulation de la physique. Pour cela, nous avons modélisé le concept d’objet physique. Ce concept a pour but de représenter la physique d’une forme. Nous avons donc utilisé ce concept pour calculer la position d’un objet à tout moment d’une simulation ainsi que sa vitesse à partir de toutes les forces qui lui sont appliquées. Pour cela nous avons utilisé les équations de cinématique suivantes.

$$x(t) = x_0 + v_0 \cdot t + \frac{1}{2} \cdot a \cdot t^2$$

$$v(t) = v_0 + a \cdot t$$

Grâce à ces équations et en mesurant le temps qui s’écoule au lancement d’une simulation, nous avons réussi à déplacer la position de tous les objets 3D en fonction du temps. Puisque nous réaffichons fréquemment les objets et que les calculs sont réalisés à partir de la position de l’objet, les objets peuvent désormais se déplacer selon les forces appliquées.



Cette fonctionnalité a été complètement réalisée durant cette itération.

### 2.3.2 Ajout d'une possibilité de sélection des objets

Afin de manipuler les objets plus facilement, nous avons ajouté, dans l'interface, une liste de boutons. Chacun de ces boutons est lié à un objet et nous permet de le sélectionner pour ensuite le manipuler. L'affichage ainsi réalisé est le suivant :

Puisque nous pouvons désormais sélectionner une table, il nous faut désormais rendre l'ajout des forces à un objet donné possible.



Cette fonctionnalité a été complètement réalisée durant cette itération.

### 2.3.3 Manipulation des forces dans l'interface graphique

Nous avons donc cherché à rendre la manipulation de forces (création et suppression notamment) possible depuis l'interface graphique de notre application. Ainsi, une fois l'objet sélectionné grâce à la fonctionnalité du point précédent, nous affichons les détails de la forme dans l'interface avec la possibilité d'ajouter une force sur l'objet sélectionné. Toutes les forces sont affichées et elles-mêmes sélectionnables, nous avons donc rajouté un bouton permettant la suppression de la force actuellement sélectionnée.



Cette fonctionnalité a été complètement réalisée durant cette itération.

### 2.3.4 Possibilité de lancer la simulation

Pour finir, toujours dans le but de permettre aux objets de se déplacer, nous avons ajouté un bouton permettant de lancer la simulation. Une fois ce bouton pressé, nous demandons 60 fois par secondes à tous les objets physiques de mettre leurs positions à jour.



Cette fonctionnalité a été complètement réalisée durant cette itération.

## 2.4 Sprint 3

L'objectif de ce dernier sprint a été de finaliser le projet en ajoutant les dernières fonctionnalités réalisables dans le temps imparti : l'ajout de nouvelles formes prédéfinies, la prévisualisation d'un objet 3D, l'importation et l'exportation d'un projet \*.obj et la détection de collisions.

### 2.4.1 Ajout de nouvelles formes prédéfinies

Afin de proposer à l'utilisateur un panel un peu plus large de formes 3D, l'ajout de nouvelles formes a été réalisé. En plus du cube et de la sphère, il est maintenant possible d'ajouter une pyramide (à base carré ou triangulaire), un cône et un cylindre à la scène 3D.



Cette fonctionnalité a été complètement réalisée durant cette itération.

### 2.4.2 Prévisualisation d'un objet 3D

Nous avons ensuite voulu enrichir l'expérience utilisateur en mettant en place la prévisualisation d'un objet 3D avant de valider l'ajout ou non de l'objet à la scène.



Cette fonctionnalité a été complètement réalisée durant cette itération.

### 2.4.3 Importation et exportation d'un projet \*.obj

En prenant en compte toutes les fonctionnalités précédentes, le choix des formes était donc restreint aux formes prédéfinies présentes sur l'interface. Afin d'accroître le champ des possibles, nous avons ajouté la possibilité d'importer un projet \*.obj. Grâce à cette fonctionnalité, l'utilisateur a plus de liberté et peut ajouter de nouvelles formes qu'il a préalablement créé dans un logiciel de modélisation 3D.

Nous avons ensuite implanter la possibilité d'exporter un projet \*.obj. L'utilisateur peut donc à tout moment sauvegarder son projet afin de le récupérer et de continuer ses expérimentations plus tard.



Cette fonctionnalité a été complètement réalisée durant cette itération.

### 2.4.4 Détection des collisions

Une fonctionnalité importante d'un moteur 3D est la détection des collisions. Des recherches ont été effectuées mais la complexité de cette fonctionnalité ne nous a pas permis de réaliser une version complètement finalisée et stable. Toutefois, une première version expérimentale a été implantée.



Cette fonctionnalité n'a pas pu être complètement finalisée. Une première version est cependant proposée à l'utilisateur (cf. Manuel utilisateur).



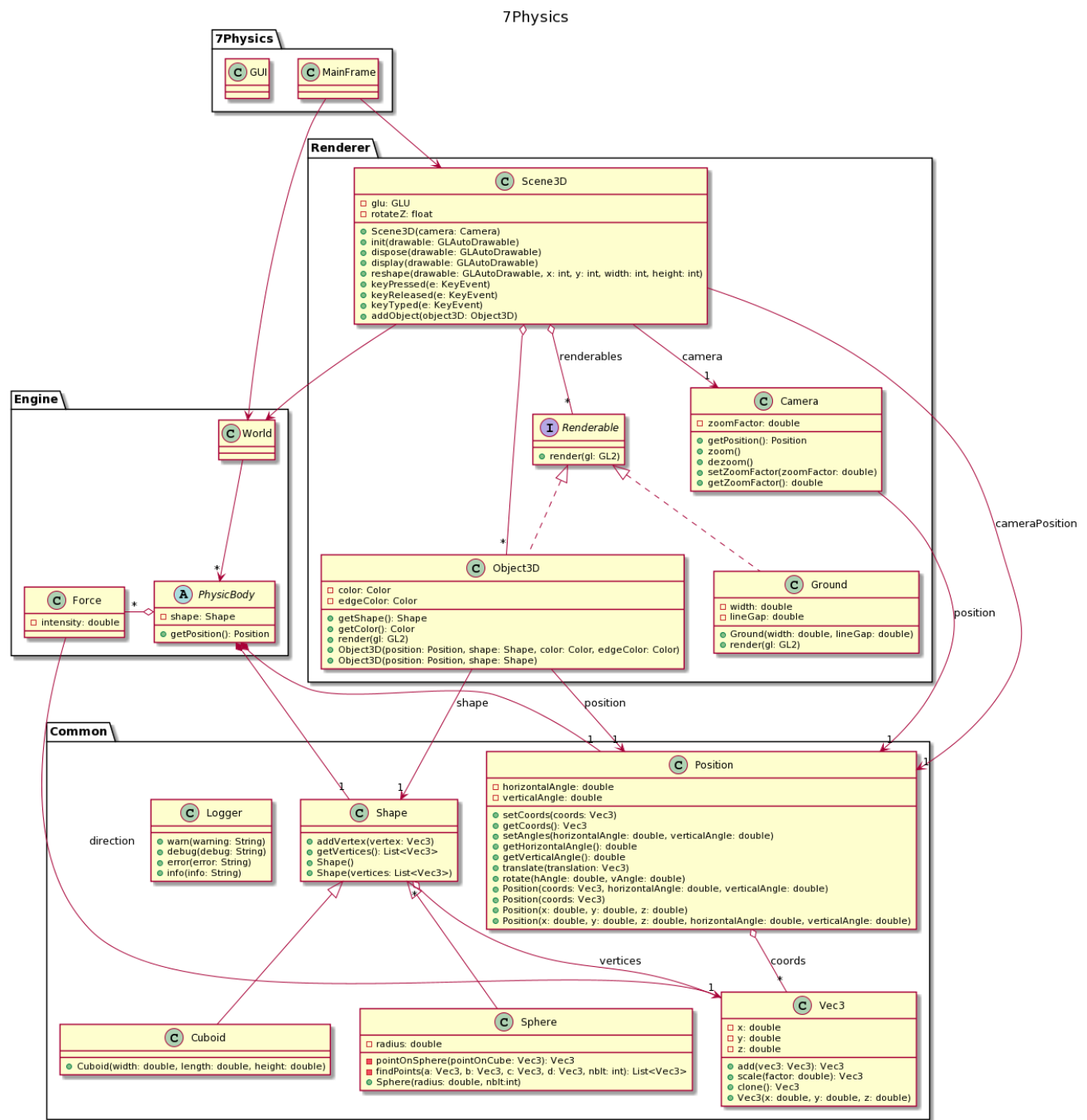
### 3. Découpage de l'application

Afin de séparer les rôles le plus possibles et permettre l'utilisation, à termes, de parties de l'application indépendamment, nous avons commencé par créer 4 répertoires git différents. Cela permet alors de n'utiliser que les fonctions de rendus dans une librairie minimaliste si besoin et de même pour la partie simulation physique.

Dans chaque répertoire était alors développée une des 4 briques de l'application finale :

- **7Physics** : Répertoire contenant le code de l'interface graphique. Ce répertoire se base sur les 3 autres et contient l'exécutable final. Son seul rôle est d'articuler les différentes fonctionnalités du moteur et ce dans une interface graphique agréable à utiliser.
- **Common** : Répertoire contenant tout le code commun est nécessaire au fonctionnement des autres. Dans ce répertoire, nous pouvons trouver la représentation des formes, la définition d'une classe représentant la position, un logger permettant de tracer les appels systèmes, etc.
- **Render** : Répertoire contenant uniquement les interactions avec le contexte OpenGL. Afin de fonctionner, ce répertoire ne nécessite que l'utilisation du répertoire Common. Il est alors tout à fait pensable de n'utiliser que cette partie de l'application et de lui demander de fournir un affichage en 3 dimensions à partir de classes créées par l'utilisateur.
- **Engine** : Répertoire permettant la simulation physique. Ce répertoire ne nécessite que l'utilisation de Common pour fonctionner. En effet, tout comme Render, il est tout à fait pensable de n'utiliser que ce répertoire comme une librairie de calcul physique simple permettant les calculs de position, de vitesse et de collisions.

## 4. Architecture de l'application



## 4.1 7Physics

Ce répertoire est composé de deux packages : *gui* et *file* ainsi que de la classe *Main.java* permettant de lancer l'application.

### 4.1.1 GUI

Dans ce répertoire, on retrouve le package *gui* composé de la classe *GUI.java* qui représente la fenêtre principale de l'application. Elle hérite donc de *JFrame*, le composant *SWING* permettant la création d'une fenêtre principale avec une bordure et une barre de titre.

C'est donc dans cette classe que l'interface principale est définie avec les différents conteneurs (*JPanel*, *JMenu*) contenant d'autres composants (*JButton*, *ImageIcon*, *JMenuItem*, ...).

Dans cette classe, le patron de conception *Singleton* a été implémenté. En effet, nous souhaitons nous assurer qu'une seule fenêtre principale ne puisse exister et donc qu'un unique point d'accès soit possible.

Afin de séparer le code, certains composants ont été séparés de la classe *GUI*. Les différents *JPanel* utilisés pour l'ajout des formes prédéfinies ont été créés dans des classes séparées. De même concernant certains *ActionListener*, les interfaces "auditrices" pour différents types d'événements à traiter.

### 4.1.2 Files

Dans ce même répertoire, on retrouve aussi le package *file*. Ce package contient la classe *OBJFile.java* qui permet de manipuler des fichiers *\*.obj*. Elle permet de fournir une méthode permettant d'extraire le contenu d'un fichier *\*.obj* et de faire le traitement nécessaire afin de fournir les formes à afficher à la scène 3D. Elle permet aussi la création d'un fichier *\*.obj* à partir du contenu de la scène 3D courante.

### 4.1.3 Tests

Une classe de test a été réalisée afin de tester la classe *OBJFile* avec différents fichiers *\*.obj* (taille et contenu différent).

## 4.2 Common

Le répertoire *Common* est composé de deux packages : *logger* et *geom*.

### 4.2.1 Logger

Dans le package *logger*, la classe *Logger* représente la classe utilitaire gérant les logs du projet.

### 4.2.2 Types utilitaires

Dans le package *geom*, il y a différentes classes définissant des types utilitaires. Nous avons par exemple, la classe *Vec3* qui représente un vecteur à trois dimensions, la classe *Position* représentant une position et la classe *Shape* représentant une forme en trois dimensions.

### 4.2.3 Formes prédéfinies

Dans le package *geom*, nous avons un nouveau package *shape* représentant les différentes formes prédéfinies tels que le cone, le cube, la pyramide, la sphere, ...

#### 4.2.4 Tests

Une classe de test a été réalisée afin de tester avec différents formes 3D (cube et sphère), la classe *BoundingBox* qui représente les bordures des objets 3D.

### 4.3 Engine

Ce répertoire est composé de plusieurs classes : *PhysicObject*, *World*.

#### 4.3.1 PhysicObject

La classe *PhysicObject* représente un objet 3D dans l'environnement ayant des caractéristiques physiques.

Un *PhysicObject* est créé à partir d'une *Shape*, d'une *Position* et d'une vitesse initiale. Nous pouvons lui ajouter/supprimer des forces, mettre à jour sa position et sa vitesse en fonction des forces appliquées.

#### 4.3.2 World

La classe *World* représente un monde 3D avec un sol d'une certaine dimension. Il est possible d'ajouter un *PhysicObject*, d'appliquer la gravité au monde 3D, de gérer les collisions.

#### 4.3.3 Tests

Deux classes de test ont été définies : *TestCollisions*, *TestPhysicBody* afin de vérifier l'effet des collisions et les résultats obtenus après application de plusieurs forces sur un *PhysicObject*, .

### 4.4 Renderer

Ce dernier répertoire est composé d'éléments permettant le rendu graphique 3D.

#### 4.4.1 Camera

#### 4.4.2 Object3D

#### 4.4.3 Scene3D

#### 4.4.4 Ground

#### 4.4.5 Tests

Des tests graphiques ont été réalisés afin de vérifier visuellement la suppression d'un objet de la scène par exemple.

#### 4.4.6 Interaction avec OpenGL

# 5. Principaux choix

## 5.1 Conception

### 5.1.1 Création de la maquette IHM

A l'aide de l'outil Figma, une maquette de l'interface graphique a été réalisée afin de concrétiser les idées des membres de l'équipe et de représenter concrètement le logiciel à construire. Nous avons choisi ce logiciel de design car il offrait la possibilité de collaborer sur un projet unique. En apportant tous nos avis sur cette maquette, nous avons pu obtenir une représentation réaliste qui a été utile lors du développement de l'IHM.

### 5.1.2 Création d'un diagramme de classe

Après avoir défini les besoins auxquelles notre application devra répondre, nous avons réfléchi au découpage du projet (voir Découpage de l'application). A partir de ce découpage, nous avons créé un diagramme de classe initial qui nous a servi de base à nos développements. Au fur et à mesure de l'affinement de nos visions sur chaque partie de l'application nous avons adapté le diagramme de classe en conséquence. Nous avons utilisé l'outil plantuml pour créer ce diagramme ce qui nous a permis de ne pas perdre du temps de mise en page.

## 5.2 Réalisation

Comme nous l'avons évoqué dans la partie concernant le découpage de l'application, nous avons choisi de réaliser cette application au travers de 4 répertoires git différents. Cette décision a été prise dans l'objectif de fournir 2 librairies indépendantes tout en ayant le code commun nécessaire à ces 2 librairies non dupliqué.

## 5.3 Problèmes rencontrés et solutions apportées

### 5.3.1 Rafraîchissement du rendu graphique

Lors de la première version de notre projet, la caméra avait un déplacement saccadé rendant l'expérience utilisateur médiocre. Nous avons alors cherché à régler ce problème en augmentant le taux de rafraîchissement de la caméra à 60 images par seconde. De plus, nous avons diminué les valeurs utilisées pour déplacer la caméra afin de donner plus de contrôle et rendre l'animation plus fluide.

### 5.3.2 Caméra fixée sur la scène

Nous n'avions, au début du projet pas connaissance de la bonne utilisation et réalisation de la caméra dans un contexte OpenGL. Nous avons donc commencé par faire tourner la scène sur elle même sans bouger le point de vue. Cela donnait alors la possibilité de voir la scène sous tous les angles mais n'était en aucun cas une bonne utilisation de la caméra. Nous avons fini par trouver une solution dans une des classes fournies par la librairie nous permettant d'utiliser OpenGL. Cela nous a alors permis d'obtenir une réelle caméra augmentant ainsi l'expérience utilisateur car la visualisation était beaucoup plus intuitive.

### 5.3.3 Rendu pixelisé

Lors de l'utilisation d'OpenGL dans sa version brute, nous avons remarqué de gros problèmes de résolutions qui se traduisaient alors par des lignes d'avantages semblables à des escaliers qu'à de vraies lignes droites. Pour corriger cet effet, nous avons utilisé l'option d'anti-aliasing disponible. Cela a alors amélioré le rendu graphique des objets et de la scène en général.

### 5.3.4 Problèmes de simulation

Lorsque nous avons ajouté la possibilité d'ajouter des forces sur un objet, nous avons utilisé, afin de calculer la position et la vitesse, des référentiels de temps absolus. Nous avons donc utilisé, dans nos équations, la position et la vitesse de base et nous avons utilisé comme mesure de temps, le temps écoulé depuis le début de la simulation. Cependant, cela avait pour effet lors de la modification de forces lors d'une simulation, de téléporter les objets à la position obtenue si la force avait été présente depuis le début de la simulation.

Pour corriger ce problème, nous avons alors changé la façon de réaliser les calculs de cinétique en nous basant désormais sur un référentiel de temps relatif. Chaque position était alors calculée via la dernière position enregistrée et le temps écoulé depuis le calcul de celle-ci.

## 6. Organisation de l'équipe et mise en oeuvre des méthodes agiles

### 6.1 Mise en place du projet

L'objectif du sprint 0 a été de mettre en place le projet. Pour cela, il a tout d'abord fallu déterminer les différents objectifs et les différents besoins utilisateur au travers de User stories. Ensuite, l'équipe a défini les outils à utiliser concernant la gestion de projet, la gestion du code et la communication au sein de l'équipe. Pour finir, le projet a été structuré en différents répertoires au sein de l'organisation GitHub créée pour le projet long et l'environnement de travail a été configuré pour chaque membre de l'équipe.

Une fois cela fait, nous avons pu commencer à réaliser les premières User Stories après les avoir estimées en termes de points d'efforts et de valeur métier.

### 6.2 Estimation des points d'efforts et de valeur métier

#### 6.2.1 Points d'efforts

Afin d'estimer au mieux les points d'efforts, nous nous sommes réunis et avons procédé à un planning poker. Le déroulement de ce planning poker était alors le suivant :

1. L'un d'entre nous lisait l'énoncé de la user story que nous devions estimer
2. Chacun d'entre nous réfléchissait alors de ce qu'il pensait de cette tâche et préparait alors une estimation
3. Pour finir, nous discutons de ces estimations et nous mettons d'accord sur la valeur finale à accorder à cette tâche.

#### 6.2.2 Points de valeur métier

Une fois cela fait pour chaque User Story, nous nous sommes penchés sur les points de valeur métier. Pour cela, nous avons procédé d'une façon similaire. Nous avons commencé par attribuer un nombre total de valeur métier à obtenir puis avons partagé ce total et attribué des points de valeur métier en fonction de l'importance des fonctionnalités à nos yeux.

### 6.3 Utilisation des cérémonies agiles

Lors du déroulement du projet, nous avons mis en place 4 types de cérémonies des méthodes agiles.

#### 6.3.1 Sprint planning

Au début de chaque sprint nous avons commencé par se réunir tous ensemble pour faire un point sur l'état d'esprit de tout le monde. Nous échangeons alors sur le ressenti par rapport au projet et l'état d'avancement

de celui-ci ainsi que les difficultés éprouvées. Cette première étape nous permettait alors de s'assurer de la bonne entente entre les membres de l'équipe.

Une fois cela fait, nous continuions par choisir les prochaines User Stories à implémenter au sein de l'application en se basant sur la vélocité de l'équipe lors du sprint précédent.

Le sprint était alors prêt à commencer.

### 6.3.2 Daily scrum

Lors du déroulement de chaque Sprint, nous avions des réunions régulières. N'ayant pas forcément le temps d'avancer chaque jour sur le projet, nous avons donc décidé de nous réunir tous les 2 à 3 jours pour faire le point sur l'avancement de chacun depuis le dernier daily scrum, les choix de conception et d'implémentation que nous devions revoir en cours de projet, etc.

### 6.3.3 Sanity Check

Au milieu de chaque sprint, nous nous réunissions afin de discuter de l'état courant du sprint. Le but était alors, dans le cas où tout avait déjà été fait, d'ajouter de nouvelles User Stories à réaliser, ou, dans le cas contraire, d'alléger la charge de travail prévue.

Cette cérémonie nous permettait alors d'être sûr que notre charge de travail était réalisable lors du temps imparti et de s'assurer que tout se déroulait comme prévu.

### 6.3.4 Sprint Review

Pour clôturer le sprint, nous organisons une réunion ayant pour but de revenir sur les événements du sprint. Cette cérémonie était importante car elle nous permettait notamment de discuter des difficultés rencontrées et calculer la vélocité à utiliser lors du prochain sprint planning.

### 6.3.5 Retrospective

Enfin, l'équipe se réunissait une dernière fois dans le sprint pour analyser le déroulement du sprint. Cette analyse permet d'avoir le ressenti réel de chaque membre de l'équipe sur le projet. Cette réunion faisait ressortir les défauts du sprint réalisé afin de tenter de les corriger lors du prochain.

## 6.4 Choix des tâches à réaliser

Afin de choisir au mieux les tâches à effectuer lors de chaque sprint, nous nous sommes basés sur les estimations de valeur métier et de points d'efforts que nous avons évoqués précédemment ainsi que sur notre vélocité. Nous avons alors une vélocité relativement constante de 80 points d'efforts par sprint.

Puisque nous avons donc les estimations fixées pour chaque tâche et une vélocité donnée, notre façon de procéder était la suivante, les tâches à réaliser étaient les suivantes :

- La tâche doit être **réalisable** (pas de dépendances avec une tâche pas encore effectuée).
- La tâche doit avoir une **valeur métier élevée** autant que possible.
- La **somme des points d'efforts** doit être environ **égale à notre vélocité** calculée.

Grâce à cette ligne directrice, nous avons pu mettre en place des fonctionnalités clés du projet sans problèmes liés à la gestion du projet en lui-même.



## 7. Conclusion

En conclusion, nous avons, pendant ces trois semaines de projet, tenté de réaliser un moteur physique 3D simpliste qui permet à un utilisateur de modéliser des formes simplistes par le biais de l'interface que nous avons créé, aussi bien que des projets complexes exportées au format .obj. Ces formes une fois placées dans une scène 3D peuvent alors être déplacées par le biais de forces qui leur seront appliquées.

Afin de réaliser ce projet nous avons utilisé les méthodes agiles au travers de 3 sprints distincts d'une durée d'une semaine. Sprints durant lesquels nous avons mis en place plusieurs cérémonies permettant d'avancer efficacement tout en maintenant une bonne cohésion d'équipe.

```
# Supprimer java de son système  
sudo apt list --installed | grep jdk
```

```
# Supprimer les résultats de la commande précédente  
sudo apt remove #
```

```
# Installer java8  
sudo apt install openjdk-8-jdk
```

```
# Vérifier avec la commande  
java -version
```