# Rapport de Projet de Fin d'études : Mise en place d'un environnement virtuel interactif pour une installation artistique

Mars 2019

Anthony Delasalle • Teiki Pepin • Enzo Peruzzetto
Master 2 Informatique
Université de Bordeaux

# Table des matières

# 1 Introduction

Échelles Célestes est un projet qui vise à faire éprouver au spectateur la dualité des objets célestes, à la fois géants et microscopiques, au travers d'une dramaturgie incarnée par des personnages. L'objectif général est de proposer à des participants une expérience à la fois collective et personnelle permettant d'explorer ces notions d'échelles et de point de vue.

Nous disposerons d'une maquette physique d'une ville au-dessous d'une voûte sur laquelle apparaissent des astres célestes. Ces derniers seront représentés par des objets physiques observables par tous les participants offrant une expérience collective. La voûte sera représentée par un demi-dôme sur lequel on pourra projeter une nuit étoilée. Sur cette maquette un télescope sera mit a disposition des utilisateurs afin de cibler une zone de la voûte céleste. En regardant au travers d'un dispositif dédié, un utilisateur pourra avoir une vue rapprochée de la zone visée en réalité virtuelle ; il pourra par exemple voyager virtuellement vers l'un des astres ciblés. A travers cette expérience personnelle l'utilisateur aura l'occasion de découvrir un monde virtuel avec lequel il pourra interagir en orientant le télescope et en zoomant pour observer la scène à différentes échelles.


FIGURE 1: Croquis représentant le prototype de la maquette physique

## 2 État de l'art

Cette approche combinant des éléments physiques réels, de la réalité augmentée spatiale et de la réalité virtuelle est un domaine qui a été précédemment étudié par l'équipe de recherche Potioc de notre client. Un premier prototype, visible dans la figure 2, basé sur une projection spatiale sur des éléments physiques enrichi par une vision modifiée par casque de réalitée virtuelle est présenté dans [1]. Une définition des interactions entre les différents niveaux de réalité mixe est présentée dans [2] au sein d'un framework conceptuel.

Le document [3] nous concerne plus dans le cadre de ce projet. Suite à une étude auprès d'utilisateurs, il confirme que les participants peuvent transférer des informations entre des systèmes en réalitée virtuelle et de réalité augmentée spatiale et qu'ils sont capable de représenter une scène unifiée à partir de sources visuelles hétérogènes.

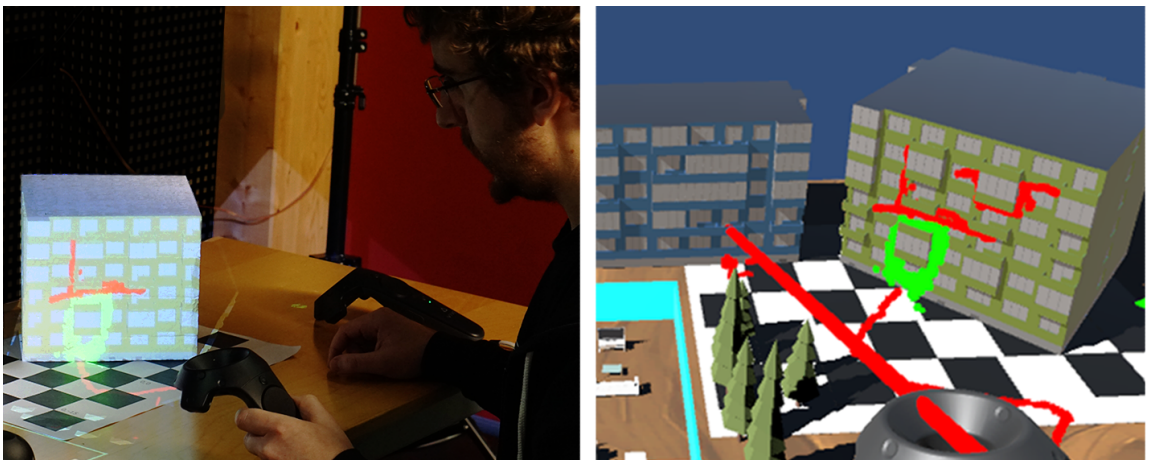


FIGURE 2: Intéraction combinant réalité augmentée spatiale et réalité virtuelle. Gauche : un utilisateur interagi avec une maquette augmentée. Droite : scène représentée par le casque de réalité virtuelle.

Afin d'appliquer la projection et nos éléments sur la surface courbée représentée par le demi-dôme de la maquette, il nous est nécessaire de prendre en compte les difficultés liées à la déformation. En effet, lors de la projection d'une image obtenue par une caméra avec un frustum de vision rectangulaire sur notre surface partiellement sphérique, l'image projetée est déformée et fortement distincte de celle que l'on observerait si celle-ci était projetée sur un plan.

On utilise pour cela le package *Dome Tools* proposé dans le Unity Asset Store. Une visualisation de l'effet de fisheye mis en place par l'asset est montré à la figure 3.

FIGURE 3: Démonstration de l'effet de fisheye proposé par le package *Dome Tools*.

N'étant pas des spécialistes en astrophysique, il nous est nécessaire de pouvoir profiter de ressources externes afin de conserver une représentation réaliste de la voûte céleste. Pour cela, nous mettons à profit Stellarium [4], un logiciel libre existant qui nous a été recommandé par le client représentant le Laboratoire d'Astrophysique de Bordeaux.

Ce logiciel fourni un vaste catalogue d'astres que l'on peut observer, tel qu'on peut le voir à la figure 4. Il dispose aussi d'une fonctionnalité de zoom permettant de représenter ces astres tels qu'on le ferait dans notre programme. Ce programme, pourtant très similaire à ce que l'on cherche à réaliser, ne permet pas d'ajouter des représentations plus artistiques et moins ancrées dans le réel. On s'en sert donc comme référence pour que notre programme maintienne un degré de réalisme.

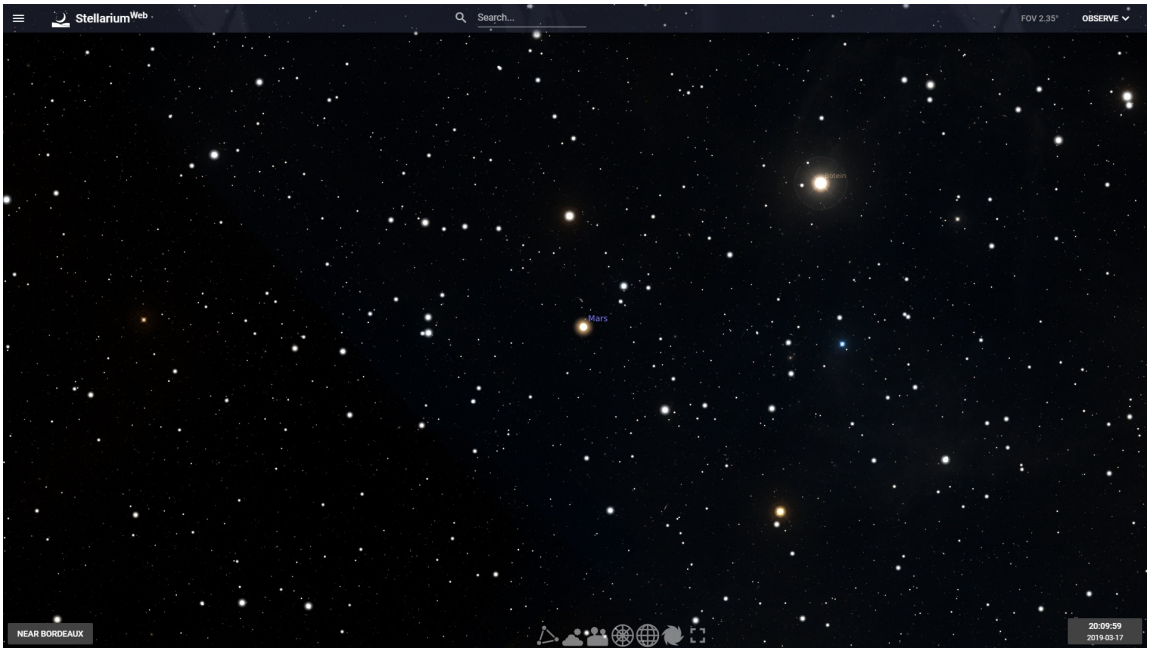



FIGURE 4: Capture d'écran de l'outil Stellarium Web permettant d'utiliser le logiciel Stellarium par navigateur.

Enfin, toujours dans un soucis de réalisme, on utilise certaines ressources mises à disposition par la NASA sur leur site [5] afin représenter nos astres. Par exemple, on peut noter le modèle imprimable de la figure 5 que l'on utilise au sein de notre projet pour représenter une vue à la surface de la Lune.

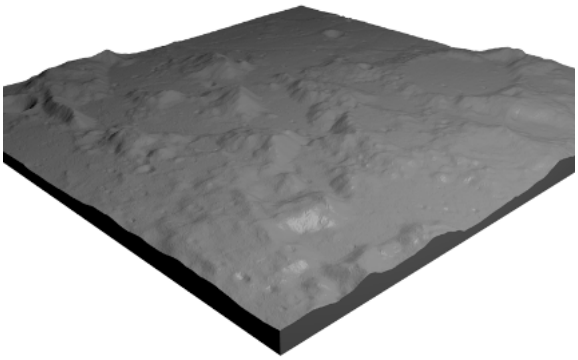


FIGURE 5: Représentation du modèle utilisé correspondant au site d'atterrissage d'Apollo 17 sur la Lune.

# 3 Cahier des Charges

## 3.1 Analyse des Besoins

### Cas d'utilisation

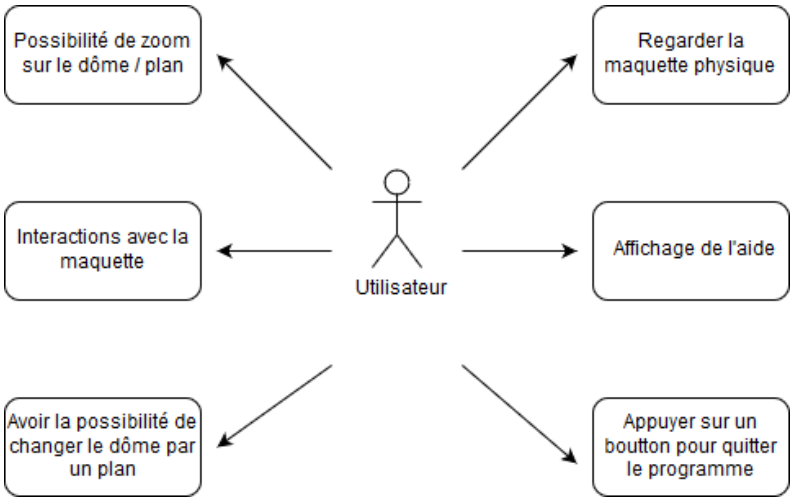



FIGURE 6: Exemples de cas d'utilisation concernant l'affichage de la maquette

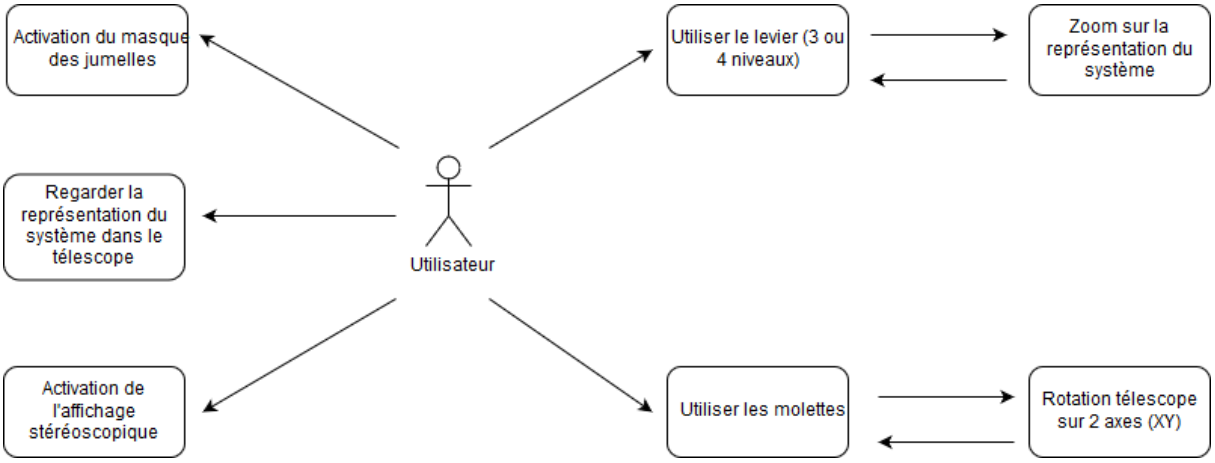



FIGURE 7: Exemples de cas d'utilisation concernant l'affichage de la représentation du système

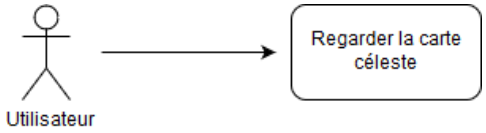



FIGURE 8: Exemple de cas d'utilisation concernant l'affichage de la carte céleste

**Besoins Fonctionnels**

Les besoins fonctionnels suivants ont un niveau de priorité déterminé par une échelle numérique où :
(1) => Priorité élevée.
(5) => Priorité faible.

**Affichage de la carte étoilée :**
- (3) Affichage à plat de la carte étoilée projetée sur le demi-dôme de la maquette.

**Mise en relation entre l'objet détecté sur la maquette et sa visualisation dans l'espace virtuel :**
- (1) Savoir quand un objet est détecté.
- (1) Associer l'objet détecté sur la maquette à son équivalent virtuel.

**Affichage de la visualisation à travers le télescope :**
- (1) Affichage de la visualisation en stéréoscopie (ou non).
- (4) Avoir un bouton qui permet d'activer ou désactiver la vision en stéréoscopie.
- Pouvoir effectuer un zoom :
  - (1) Zoom avec plusieurs niveaux de définition (au moins 3).
  - (4) Effectuer des transitions douces entre les différents niveaux de zoom.
  - (2) Le dernier niveau de zoom permet de se placer sur la planète visualisée en vue à la première personne.

**Affichage d'une simulation de la maquette :**
- (1) Avoir une fenêtre d'affichage dans laquelle on peut visualiser une représentation du bureau avec le dôme et les systèmes d'interaction.
- Projection d'une voûte céleste sur le dôme de la maquette :
  - (1) Effectuer une projection d'une voûte céleste sur un demi-dôme en tenant compte des distorsions sur les bords.
  - (3) Afficher un overlay sur le dôme pour visualiser la zone ciblée par le télescope.
- Bouger le télescope
  - (1) Disposer sur la maquette de deux molettes pour gérer l'orientation du télescope.
  - (1) Disposer sur la maquette d'un levier interactif pour gérer le zoom.
  - (3) Avoir une vitesse de rotation du télescope variable en fonction du niveau de zoom.

**Besoins Non Fonctionnels**

L'application développée devra être fluide, c'est-à-dire 60 images par seconde au minimum sur un ordinateur configuré avec un processeur Intel Core i5 de 2.20Ghz avec une carte graphique Nvidia GeForce 840M et une RAM de 8Go et devra être conforme aux données physiques fournies par les collaborateurs du laboratoire d'astrophysique.

Concernant l'interface, l'application devra être ergonomique pour permettre son utilisation par le grand public avec des interactions simples et intuitives.

**Contraintes**

La seule contrainte est liée à notre environnement de travail qui est l'utilisation de Unity 3D et du langage C# pour la mise en place des éléments décrits ci-dessus et une gestion de développement agile de notre projet avec l'utilisation de SCRUM.

## 3.2 Tests

Pour la validation du projet, nous allons mettre en oeuvre des tests de validations sur les besoins non fonctionnels et des tests unitaires pour les besoins fonctionnels. Toutefois certains besoins correspondant plus à une visualisation, ils seront testés manuellement à l'oeil nu.

Premièrement, pour la fluidité nous disposons d'un outil externe à notre projet qui nous permettra de relever le nombre d'images par seconde atteint par le programme. Afin de satisfaire notre besoin de fluidité et d'ergonomie nous avons pour objectif d'atteindre un minimum de 60 images par seconde sur l'ordinateur décrit plus haut.

Concernant l'ergonomie du programme, nous mettrons en place un questionnaire d'évaluation que l'on distribuera à une quinzaine de personnes issues du domaine informatique.

## 3.3 Architecture

Dans cette partie, nous allons expliquer l'architecture mise en place dans notre projet. Comme déjà dit au parties précédentes, nous utilisons Unity3D qui nous oblige à avoir une architecture particulière (propre à celle d'Unity3D) qui diffère des architectures traditionnelles de classes. Comme vous pouvez le voir dans la figure 9, nous avons séparé notre scène en plusieurs parties.

Le *GameManager* est le contrôleur qui permet de gérer toutes les interactions claviers ou souris de notre projet.

La partie *System Simulation*, figure 10, permet de gérer tout ce qui concerne le comportement des caméras (rotation et zoom). Ces comportement seront expliqués plus en détails dans la partie Implémentations (4.2). La sous-partie *SystemManager* est un *GameObject* qui contient toutes les scènes des planètes du niveau 0 au niveau max que l'on peut définir dans Unity et doit absolument contenir les scripts $MapManager.c$ et $ViewManager.cs$. Attention à l'ordre de la hiérarchie des scènes dans *SystemManager* qui est primordial. En effet, *Map - Level 0* doit être le premier fils de *SystemManager*. Quant aux autres, l'ordre des scènes importe peu sauf qu'elles doivent être des *GameObject* contenant les différentes scènes de l'astre en question. Pour finir, comme montré dans la figure 10, les caméras doivent être contenues dans un *GameObject* comportant le script $CamBehavior.cs$.

La partie *Model Simulation*, figure 11, est la partie modélisation de la maquette, où l'on peut voir ce qui la compose : des caméras pour faire l'affichage de cette maquette, des projecteurs pour

projeter des éléments sur le dôme et la maquette elle-même. Dans cette maquette, des éléments comme *ScrollWheel* et *Lever* sont des éléments cliquables permettant d'effectuer un zoom ou une rotation du télescope.

La partie *Map Simulation*, figure 12, est une carte représentant notre système d'astres que l'on peut visualiser sur le dôme ainsi que dans les jumelles. On y trouve une caméra pour l'affichage de la map, d'éléments qui représentent les astres visibles par notre prototype (attention ces dernières doivent êtres placés de la même manière que celle de la *Map - Level 0* du *SystemManager*). Les *Projector Camera* de cette partie ne sont ni plus ni moins que des caméras dont les images observées seront reprojetées par les projecteurs du *Model Simulation*.

Et pour finir, la partie UI, figure 13, est tout ce qui concerne les bouttons et toggles cliquables par l'utilisateur ainsi que les textes d'instructions.

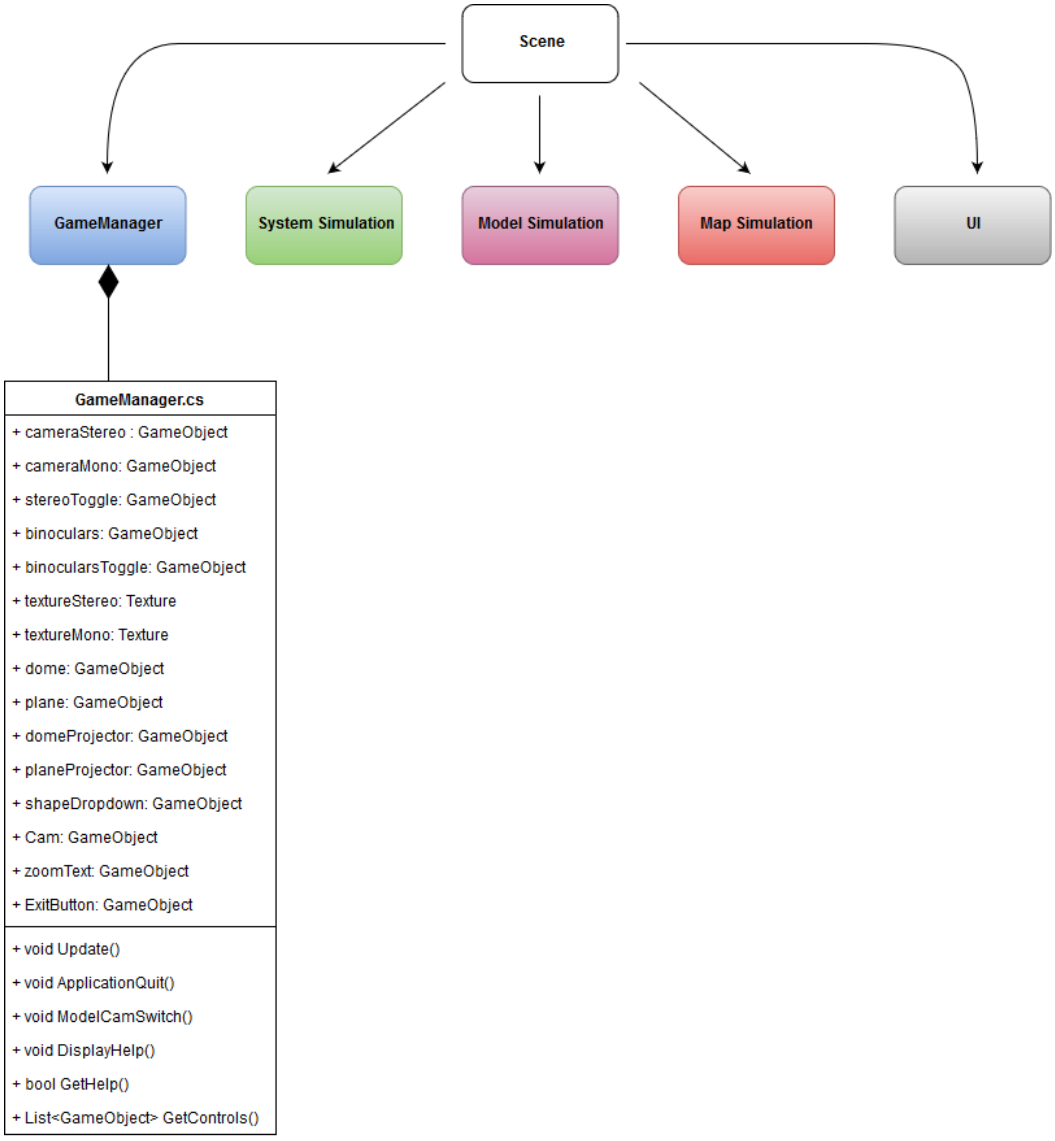



FIGURE 9: Architecture du projet

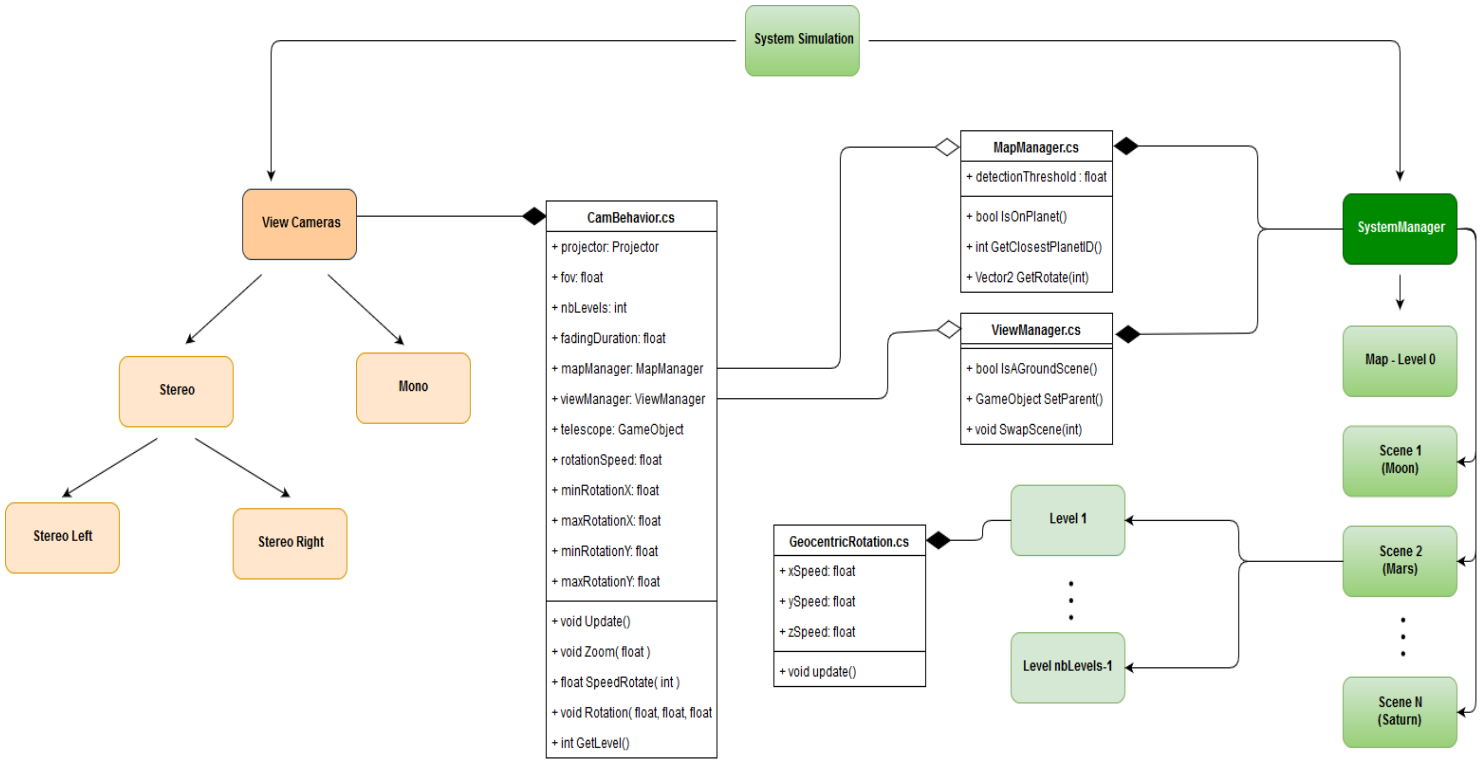



FIGURE 10: Architecture de la partie *System Simulation*

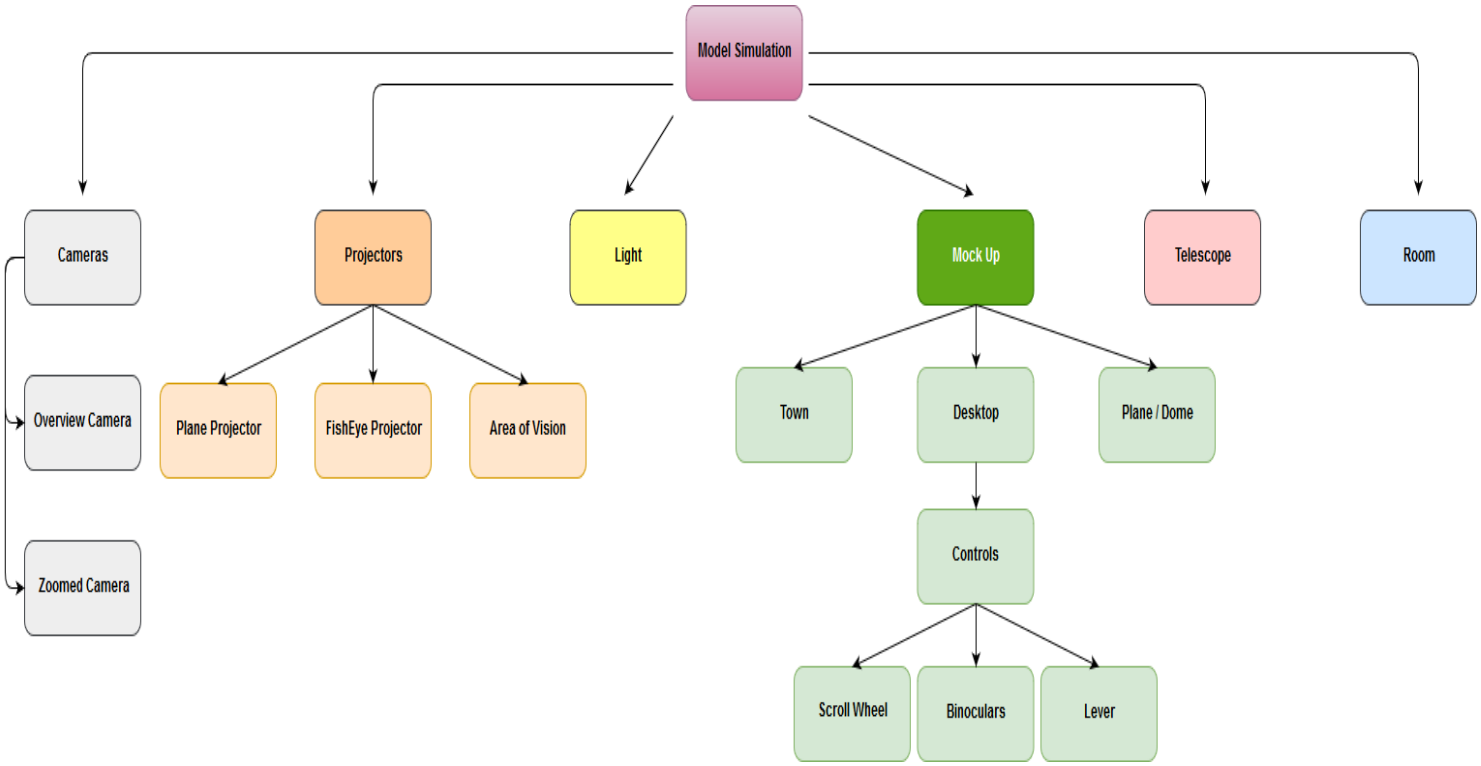



FIGURE 11: Architecture de la partie *Model Simulation*

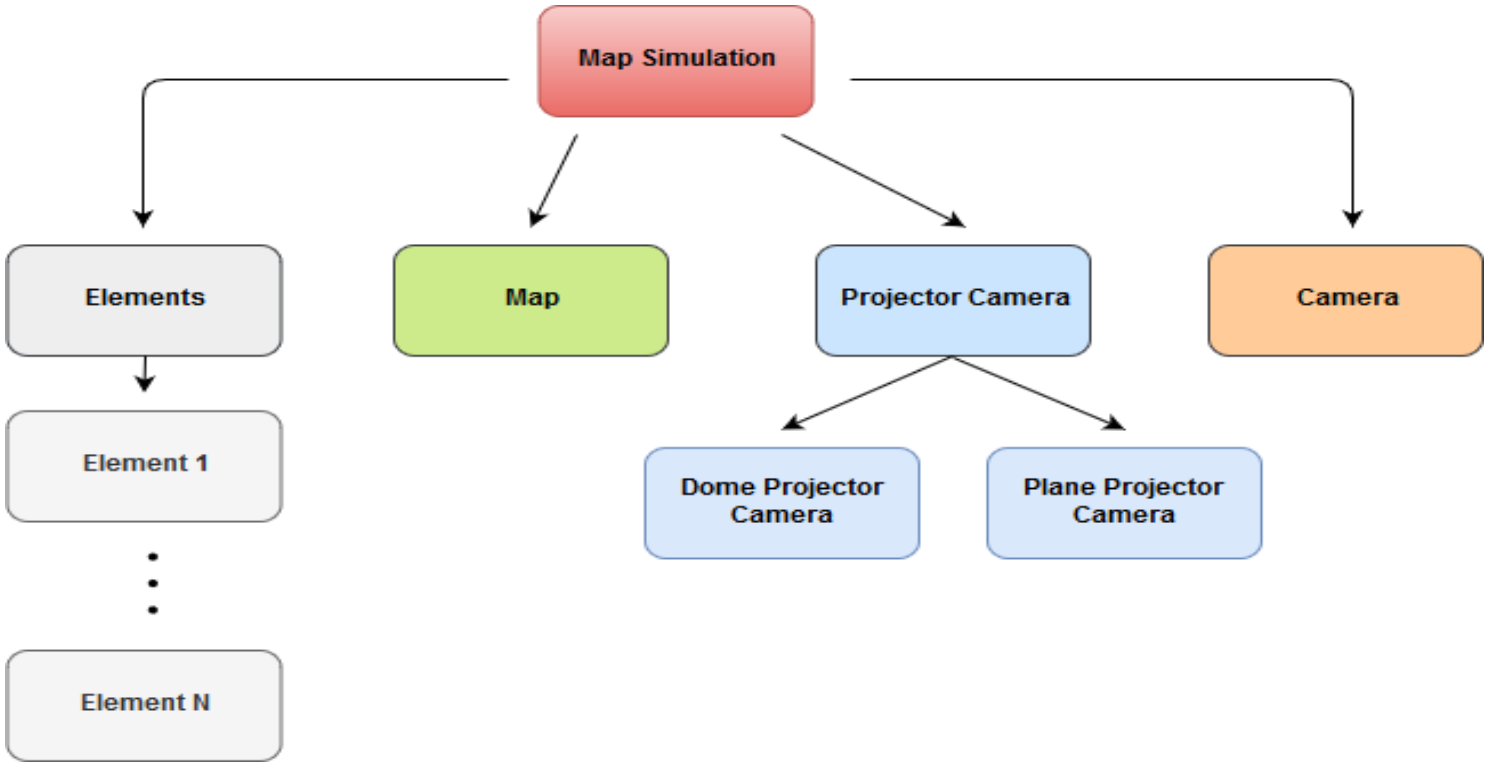



FIGURE 12: Architecture de la partie *Map Simulation*

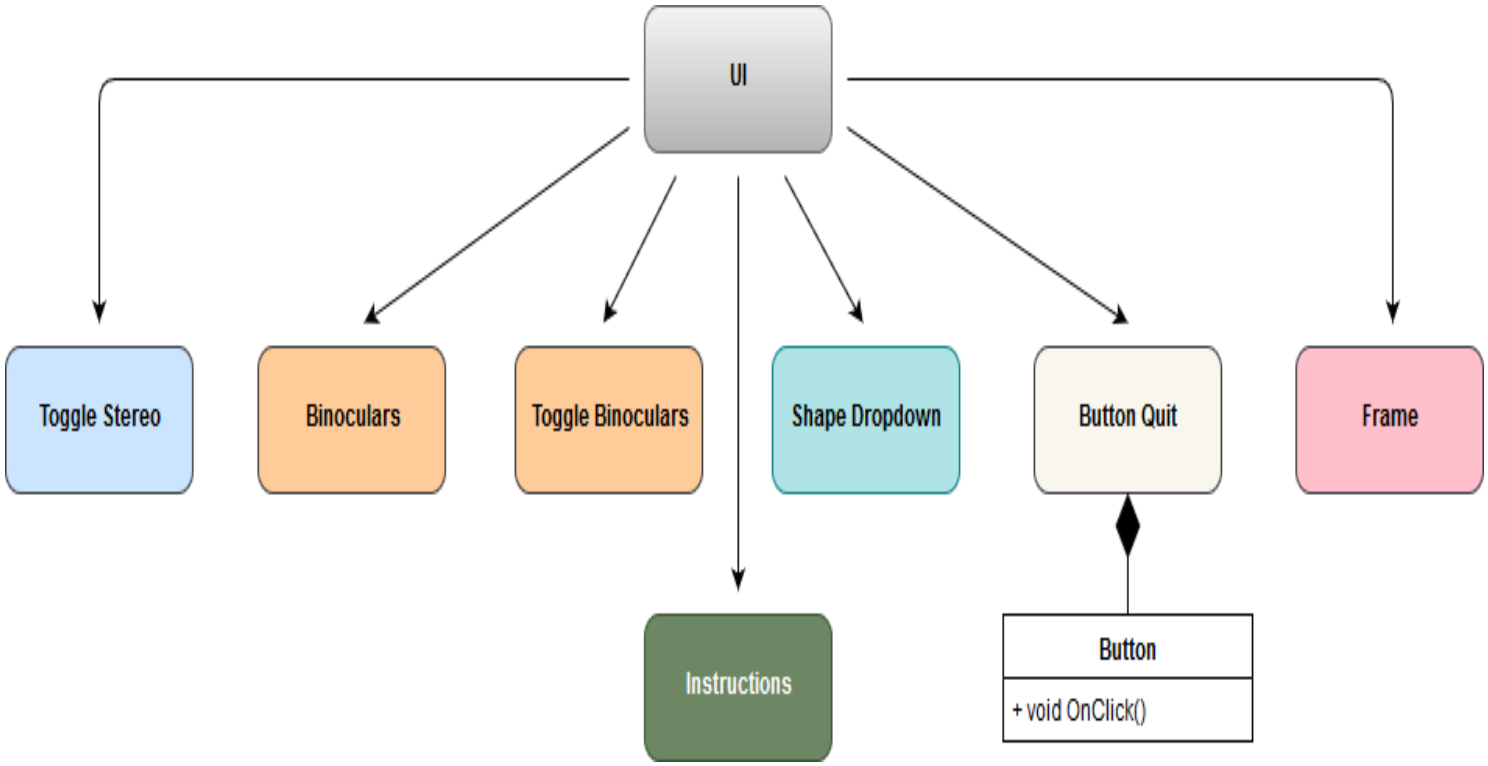



FIGURE 13: Architecture de la partie UI

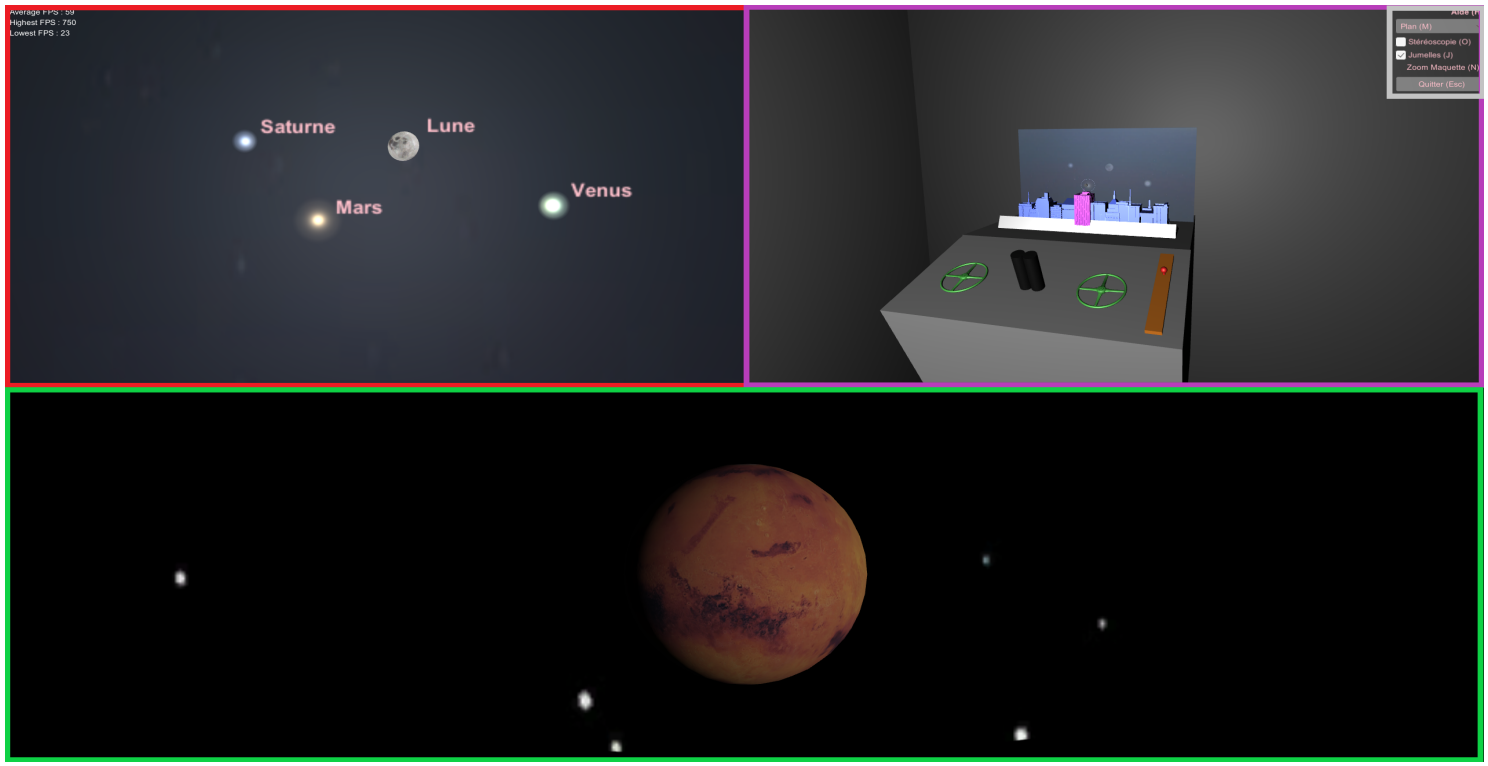



FIGURE 14: Prototype finalisé avec représentation des parties

## 3.4 Diagramme de Gantt

### Diagramme de Gantt prévisionnel

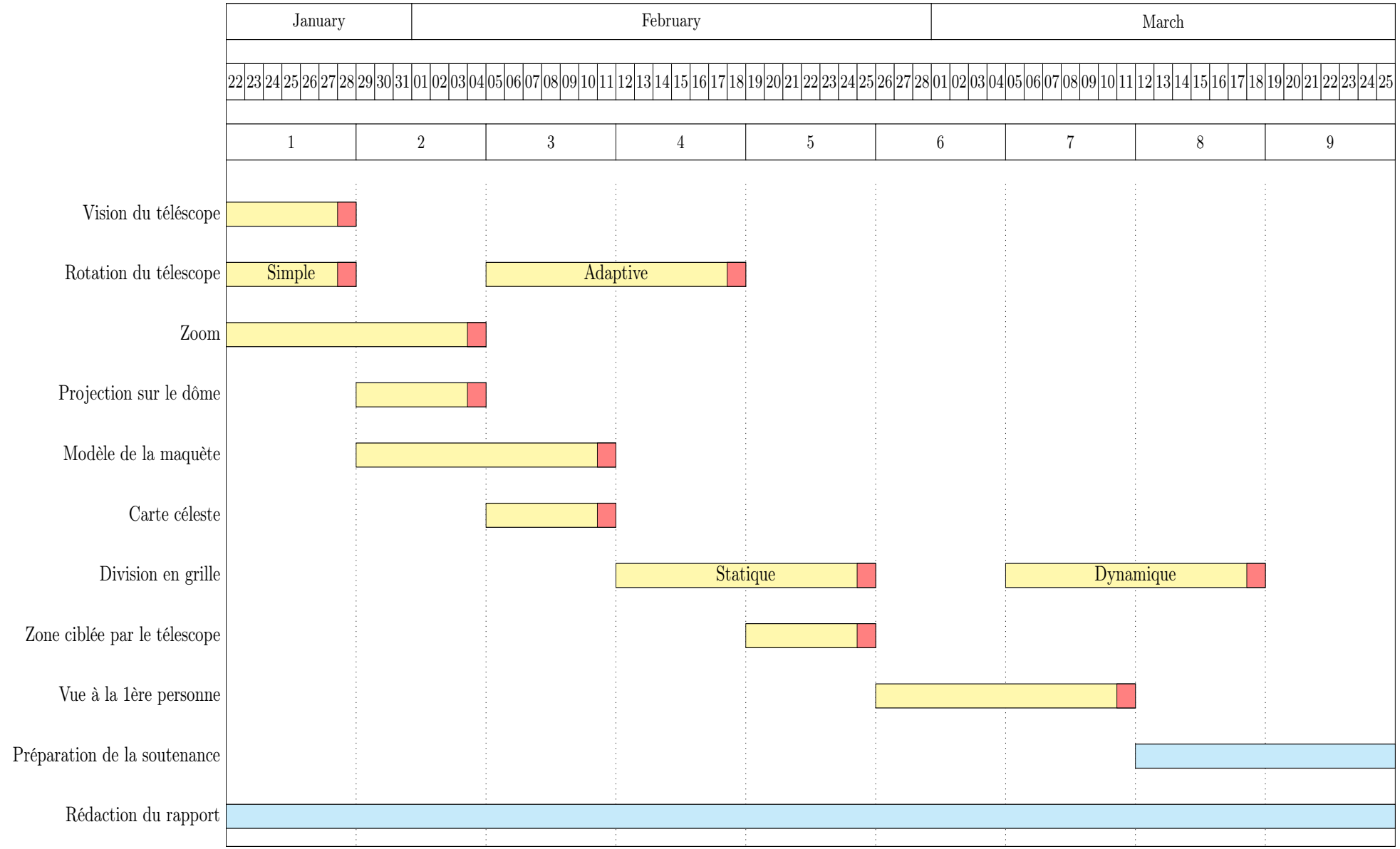



FIGURE 15: Diagramme de Gantt représentant le planning prévisionnel du projet. Les implémentations du programme sont en jaune. Les phases de tests d'intégration sont représentées en rouge. Les tâches de rédaction sont indiquées en cyan.

**Diagramme de Gantt réel**

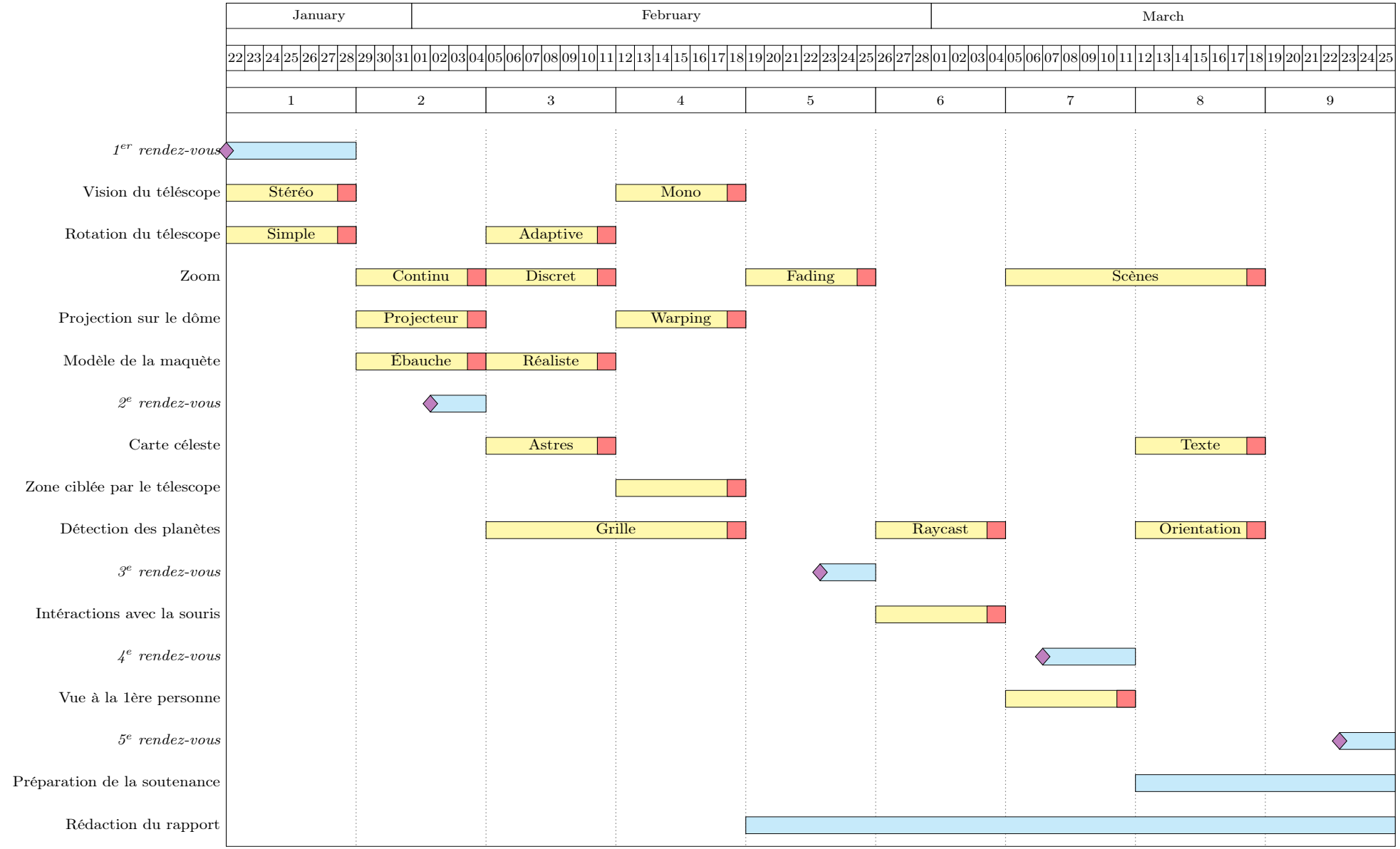



FIGURE 16: Diagramme de Gantt représentant le planning réel du projet. Les implémentations du programme sont en jaune. Les phases de tests d'intégration sont représentées en rouge. Les tâches de rédaction sont indiquées en cyan. Les rendez-vous avec le client sont notés en violet et sont toujours suivis d'une mise à jour du cahier des charges.

# 4 Implémentations

## 4.1 Interactions

### 4.1.1 GameManager

Afin de centraliser toutes les interactions de l'utilisateur, on utilise un script *GameManager.cs* rattaché à un *GameObject* éponyme. Ce script gère les parties de l'interface utilisateur interactives. La plupart des fonctions proposées peuvent être activées à l'aide du clavier ou de la souris. Cela est géré par le script en séparant les fonctions détectant l'activation d'une fonctionnalité et les fonctions mettant à jour le système en fonction de la fonctionnalité. Ces parties correspondent respectivement aux parties Contrôleur et Modèle d'un design pattern MVC, la partie Vue étant automatiquement effectuée par Unity en fonction de l'état courant du Modèle.

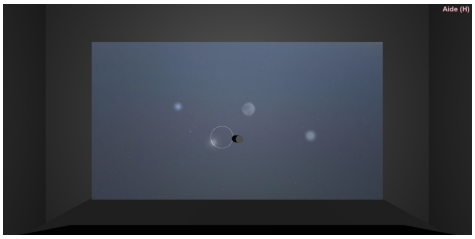
**Aide :** En premier lieu, on a un simple texte qui est affiché dans le coin haut droit de l'écran. Celui-ci lit simplement "*Aide (H)*" et indique que la touche *H* peut être utilisée afin d'afficher le reste de l'interface utilisateur (UI). À l'exception du compteur de FPS, le reste de l'UI s'affiche à la suite de ce premier message, dans le coins haut droit tel que visible dans la figure 17.


FIGURE 17: Capture d'écran de l'interface utilisateur avec les différentes options proposées.

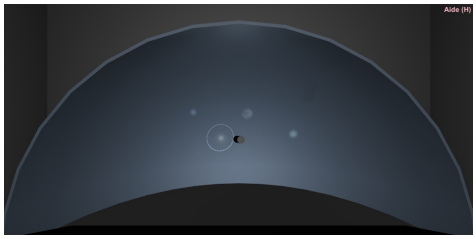
**Forme :** Une fois l'UI affichée, on peut choisir dans un menu déroulant, ou via la touche *M*, la forme sur laquelle la projection est appliquée dans la vue de la maquette. Les deux options sont : un plan qui permet d'observer notre projection sans risques de distorsions (non présentes dans la version courante du projet) et un demi-dôme qui correspond à la maquette physique qui sera produite pour le projet. Les deux formes sont montrées à la figure 18.

**Zoom maquette :** On dispose également, via la touche *N*, de la possibilité de zoomer sur le plan ou le demi-dôme de la maquette, afin d'observer de plus proche et avec plus de détail la projection. Cela nous empêche alors de pouvoir observer ou d'utiliser le reste de la maquette tant que l'on est dans un état rapproché. On peut désactiver cela on reappuyant sur la même touche.

**Stéréoscopie :** Est également mis à disposition une checkbox cliquable, que l'on peut aussi activer avec la touche *O*. Cela nous permet d'activer, ou de désactiver, la caméra stéréoscopique. Une fois

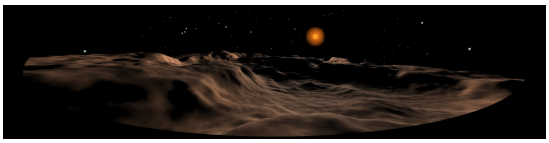


(a) Maquette avec plan.

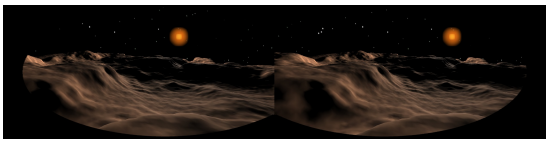


(b) Maquette avec demi-dôme.

FIGURE 18: Différentes options possible pour la forme de la maquette.



(a) Vision par caméra mono.



(b) Vision par caméra stéréo.

FIGURE 19: Activation de la stéréoscopie avec le masque des jumelles actif.

celle-ci activée, elle remplace alors la caméra mono et permet d'observer la réalisation en 3D via un matériel adapté. On peut observer la différence entre les deux vue à la figure 19.

**Jumelles :** Liée à la touche *J*, cette checkbox permet d'afficher ou non le masque correspondant aux jumelles. Ce masque, si il est actif, se met aussi à jour en fonction de la checkbox de stéréoscopie. En effet, la forme du masque s'adapte en fonction de si on voit au travers d'une caméra mono ou stéréo.

**FPS :** Enfin, on a aussi un compteur de FPS dans le coin haut gauche de l'écran. Celui-ci affiche une moyenne des derniers FPS obtenus, ainsi qu'un relevé des FPS maximum et minimum atteint. Ces deux valeurs aident à détecter lorsqu'une action particulière a un impact conséquent sur la fluidité du programme.

### 4.1.2 Maquette

On peut également effectuer certaines interactions directement sur certain composants de la maquette visibles dans la figure 20. Ces actions correspondent à des actions du télescope que l'utilisateur effectuerait d'une autre manière (souris, clavier, joystick) mais pour lesquels on permet tout de même d'utiliser directement la maquette afin de simuler le produit physique final.

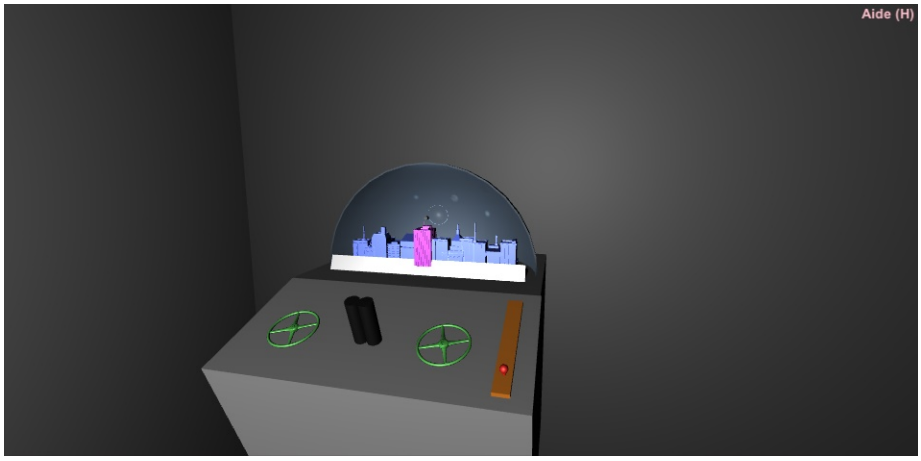



FIGURE 20: Capture d'écran de la maquette où l'on peut observer les deux volants verts et le levier rouge permettant d'interagir avec le télescope.

**Rotation :** Un script nommé *RotateScrollwheel* est attaché à chacun des deux volants de la maquette. Ce script permet de détecter lorsque le volant auquel il est rattaché est sélectionné par la souris de l'utilisateur. Lorsqu'une telle sélection est active, on change la couleur du volant afin de l'indiquer à l'utilisateur et on récupère les valeurs de déplacement sur l'axe des X de la souris (qui est glissée par l'utilisateur). Ces valeurs sont transmises à *CamBehavior* qui se chargera alors de mettre à jour la rotation du télescope sur l'axe vertical pour le volant gauche et sur l'axe horizontal pour le volant droit.

**Zoom :** De manière similaire, on a sur notre modèle de levier un script nommé *TranslateLever*. Celui-ci permet, tout comme pour les volants, de sélectionner le levier et de le déplacer à l'aide de la souris. Cela indique alors au script *CamBehavior* d'appliquer un zoom (ou un dé-zoom) en fonction de l'action de l'utilisateur. Il n'est possible de ne zoomer ou de dé-zoomer que d'un seul niveau à la fois par sélection du levier. Ce script est également en charge de mettre à jour la position visuelle du modèle du levier en fonction du niveau de zoom courant. Le levier suit ainsi les zooms effectués par l'utilisateur, y compris à la molette de la souris.

## 4.2 CamBehavior

### 4.2.1 Rotation

L'un de nos objectifs était de pouvoir bouger le télescope selon deux axes (vertical = axe X et horizontal = axe Y), tout en bougeant la vue dans les jumelles. Pour cela, il nous suffisait juste d'appliquer une rotation avec les mêmes quaternions sur le télescope et les caméras. Deux spécificités sont rajoutées à cette rotation de base : la limitation de la rotation sur le demi-dôme et ne plus appliquer de rotation au télescope une fois que le niveau de zoom est supérieur à 0. Pour la première spécificité, nous avons mis le télescope au centre du demi-dôme et limité les rotations selon quatres limitations que l'on peut changer sur Unity. Ces quatres limites sont le min et max sur les deux

axes où l'on rajoute pour le min et enlève pour le max la moitié du *field of view* des caméras, ce qui limite au demi-dôme la région couverte par le télescope (voir figure 21).

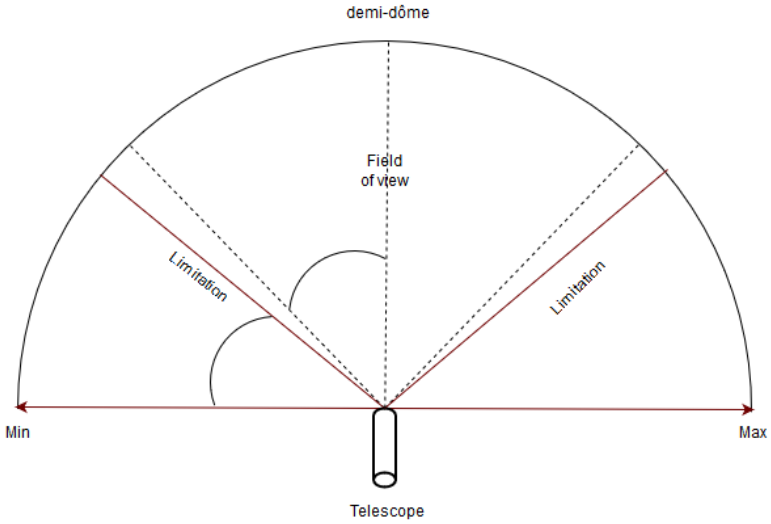



FIGURE 21: Schéma de la limitation de la rotation sur l'axe Y (Horizontal)

**Vitesse de Rotation :** Un autre besoin du client était de ralentir la vitesse de rotation selon le niveau de zoom. Pour cela on applique juste à la rotation un coefficient qui dépends du niveau de zoom.

$$coef = rotationSpeed * (\frac{nbLevels - currentLevel}{nbLevels})^{dynamicRotationFactor}$$

Où $rotationSpeed$ et $dynamicRotationFactor$ sont réglable sur unity.

### 4.2.2 Détection de planètes

Une fois la rotation effectuée, il faut pouvoir déterminer quelle étoile sommes nous en train de regarder. Pour cela, nous avons fait plusieurs méthodes pour la détection.

**Utilisation d'une grille :** Tout d'abord, le client avait proposer de mettre en place un système de grille permettant d'associer un astre à une case tout en ayant la possibilité de projeter cette grille sur la carte céleste ou le demi-dôme. La mise en place de la grille et sa visualisation ayant été faite, nous nous sommes rendu compte que le déplacement case par case n'était pas réaliste et nous avons cherché d'autres solutions.

**Utilisation d'un raycast :** On a ensuite pensé à faire un raycast depuis la caméra pour savoir quel astre l'utilisateur est en train de regarder. Cette méthode était assez simple à mettre en place mais on s'est rendu compte que, plus tard sur le projet physique du client, il serait plus compliqué de faire un raycast physiquement. On s'est donc portés sur notre dernière solution.

**Utilisation de coordonnées de rotation :** Grâce à deux coordonnées de rotation (ici X et Y) de la caméra, on peut déterminer l'objet que l'on est en train de regarder. En effet, comme le réglage de parabole à nos jour, on connaît la position de la parabole et la position du satellite, on peut donc en déduire les deux angles de rotation pour aligner la parabole au satellite.
On a donc mis en place une classe *MapManager.cs* qui, au démarrage du programme, stocke dans un tableau les coordonnées de rotation X et Y de chaque astres, et lorsque l'on fait une rotation de la caméra on a plus qu'à comparer les rotations de la caméra aux rotations du tableau avec une zone de tolérance réglable dans Unity.

### 4.2.3 Zoom

**Déplacement de la caméra :** Tout d'abord nous avons mis en place un zoom qui se déplaçait selon l'axe Z pour zoomer. Or, on s'est vite rendu compte que lorsque l'on appliquait un zoom puis une rotation, la position de la caméra en X et en Y avait changé. On a donc opter pour une autre solution.

**Field of view :** Pour le zoom nous avons commencé par mettre en place un zoom continu qui changeait juste le *field of view* de la caméra (zoom = réduction du fov, dé-zoom = augmentation du fov). Cependant après le deuxième rendez-vous avec le client, le client nous demanda de charger une scène différente à chaque niveau de zoom pour permettre plus de liberté dans les mise en place des scènes.

**Chargement de scènes :** Le chargement de scènes pour le niveau de zoom se fait avec les classes *MapManager.cs* et *ViewManager.cs*. En effet, *MapManager* implémente des fonctions qui permettent de détecter si on vise un astre (voir 4.2.2). Lorsque c'est le cas, on récupère l'ID correspondant à la position de l'astre visé dans la hiérarchie de *SystemManager*. Dans *ViewManager*, on sélectionne alors l'objet correspondant au niveau de zoom souhaité parmi les objets enfants de l'ID conservé. Cette objet, qui correspond donc à la nouvelle scène à afficher, est alors activé alors que la scène précédemment active est désactivée.

## 4.3 Déformation de la projection sur le dôme

Lorsque l'on effectue une projection sur une surface non planaire des déformations peuvent apparaître. Dans notre cas, ces déformations sont particulièrement visibles à proximité des bords de notre surface de projection qui est un dôme comme on peut le constater dans la figure 22.

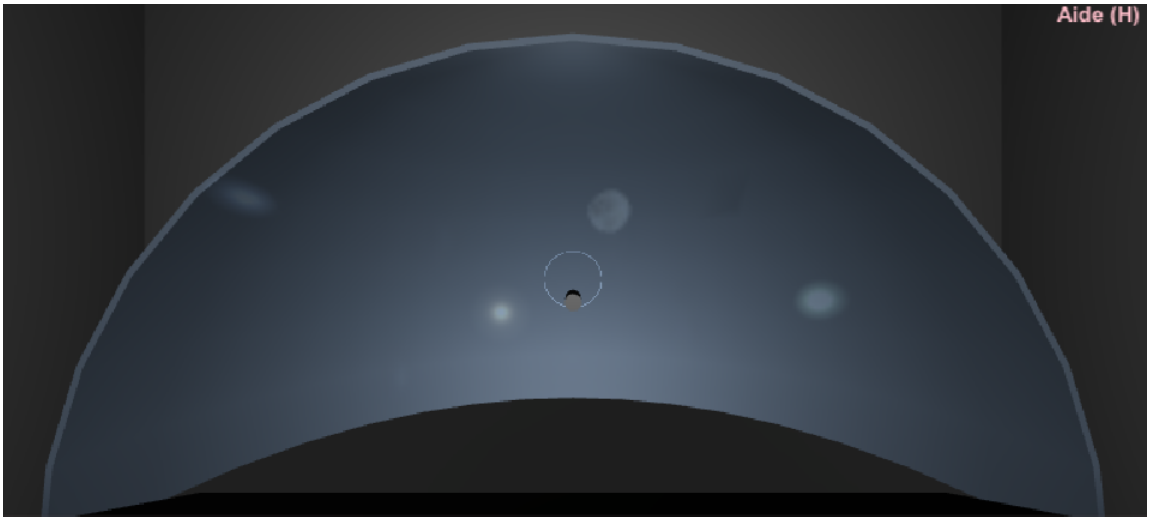



FIGURE 22: Capture d'écran de la projection déformée sur les bords.

Déformer l'image avec un effet *fisheye* avant de la projeter permet de palier à ce problème. Sur la figure 23 on peut constater qu'après application d'un effet *fisheye* les déformations précédemment perceptibles sur les bords du dôme ont été grandement atténuées.

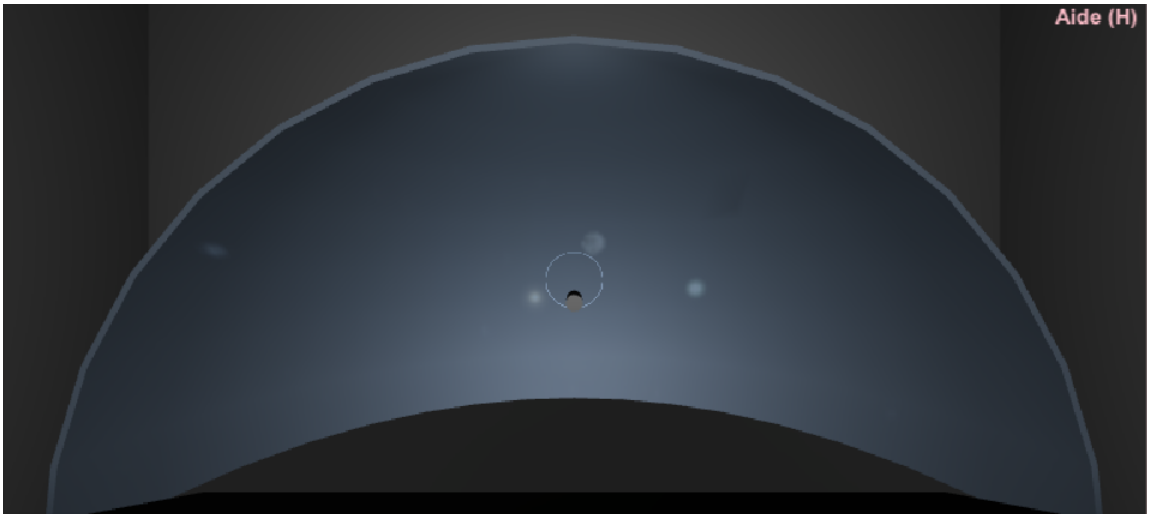



FIGURE 23: Capture d'écran de la projection avec effet *fisheye*.

Nous mettons en place l'effet *fisheye* en utilisant un *asset* disponible sur le Unity Asset Store à l'adresse suivante : `https://assetstore.unity.com/packages/vfx/shaders/fullscreen-camera-effects/dome-tools-62664`. Ce *package* se compose de deux script, d'un shader et d'un *GameObject* qui joue le rôle d'une caméra récupérant l'image à projeter.

Le script *DomeProjection.cs*, attaché au *GameObject*, récupère l'image observée par le *GameObject* sous forme de cubemap et la transforme par la suite à l'aide du shader pour avoir au rendu un effet *fisheye*. Le second script, lui aussi attaché au *GameObject*, à pour rôle de permettre de jouer sur certains paramètres dans l'éditeur de Unity afin d'avoir le rendu qui convient le mieux à l'utilisateur.

Ce package est simple d'utilisation : on place le *GameObject* fourni dans la scène, orienté de manière à ce que la caméra qui le compose récupère ce qui l'on souhaite projeter. Il faut ensuite récupérer dans un *Render Texture* le rendu de la caméra (celui-ci est déjà un rendu *fisheye*) pour pouvoir le projeter grace à un projecteur.

# 5 Tests

## Test de la stéréoscopie

**Protocole**

Notre stéréoscopie est effectuée en disposant de deux fenêtres d'affichage correspondant à deux caméras différentes, de même orientation et espacées l'une de l'autre d'une distance correspondant à la distance entre deux yeux. L'objectif est d'obtenir, à l'aide d'un visiocasque, tel que le Google Cardboard par exemple, un rendu en 3D.
Notre test consiste donc à créer un léger exécutable qui met en place notre système de stéréoscopie sur une scène réduite et à observer le rendu a l'aide d'un visiocasque. Par la suite nous avons aussi écrit un shader qui nous permet de superposer nos deux rendu et à l'aide de l'asset Post Processing Stack (disponible sur le Unity Asset Store) qui nous permet d'appliquer des filtres de couleurs sur nos rendus de caméra, nous avons mis en place un système de stéréoscopie anaglyphe. Celle-ci permet de visionner la scène en 3D à l'aide de lunette de vision stéréoscopique anaglyphe.

**Résultats**

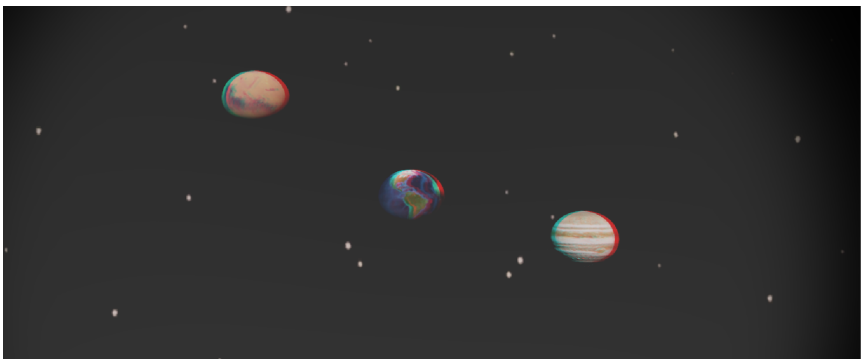


FIGURE 24: Capture d'écran du rendu en stéréoscopie anaglyphe.

| | |
|---|---|
| **Conclusion** | Le rendu premièrement effectué et visualisé avec un visiocasque est en 3D. De même, l'utilisation de lunettes de stéréoscopie anaglyphe permet de visualiser la figure 24 en 3D : notre stéréoscopie fonctionne donc. |

### Test du niveau de zoom en plusieurs niveaux

| | |
|---|---|
| **Protocole** | Pour zoomer ou dézoomer on charge une nouvelle scène. Ce test est en deux parties. Premièrement, on s'assure que la fonction de zoom et de dézoom fonctionne correctement. Ensuite, on s'assure que notre zoom soit limité au nombre maximum de niveaux.<br>Pour ce faire, on commence par orienter la caméra sur un astre, puis on applique ensuite un zoom et on vérifie que la nouvelle scène activé soit la bonne (scène de l'astre au niveau 1) et que la scène *Map - Level 0* est désactivé. On réitère un zoom pour vérifier que la scène chargée soit la suivante du même astre, puis on dézoom jusqu'à revenir à la *Map - Level 0*.<br>Pour tester la limitation sur le nombre de niveaux de zooms, on se place au niveau de zoom le plus dézoomé, on essaye de dézoomer et on s'assure que notre champ de vision soit le même. On test ensuite qu'on ne puisse pas zoomer plus que le nombre maximum de fois en regardant si la scène correspond au dernier niveau de l'astre. |
| **Résultats** | Lors de la première passe de test les scènes chargées correspondent à celles attendues, de même lors de la seconde passe les scènes aux limites restent les mêmes. |
| **Conclusion** | Le fait que les scènes activées correspondent à nos attentes signifie que le zoom et le dézoom sont fonctionnels et correspondent à nos prévisions. Lorsque nous sommes aux niveaux de zoom minimum ou maximum les scènes activées ne sont pas modifiées respectivement pour un dézoom et pour un zoom. Cela signifie que nous avons bien borner nos zooms à un nombre de niveaux fixe. |

### Test de la fenêtre de la maquette

**Protocole**

On souhaite disposer d'un fenêtre où l'on représenterait la maquette physique présentée à la figure 25. On compare une capture d'écran de notre programme aux croquis fournis par le client afin de s'assurer de la similarité entre notre réalisation et celle attendue.

FIGURE 25: Croquis représentant le prototype de la maquette physique

**Résultats**

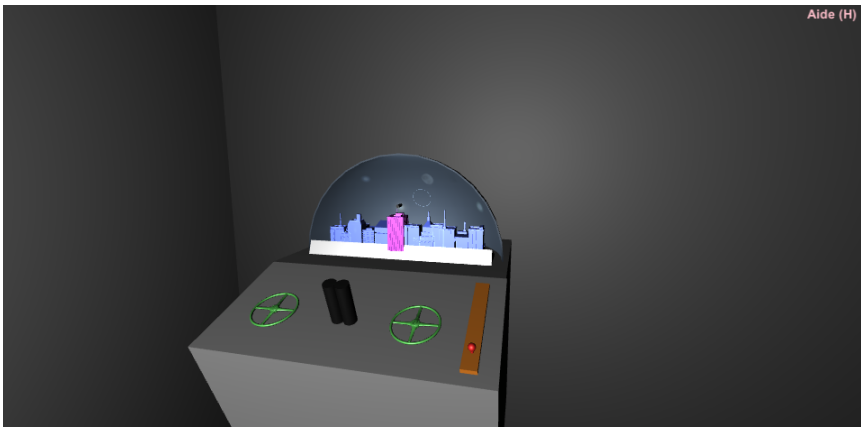



FIGURE 26: Capture d'écran d'une partie de notre programme où l'on peut voir la représentation virtuelle de la maquette.

**Conclusion**

Le résultat visible dans la figure 26 se montre similaire à la représentation de référence. On y observe bien un bureau, sur lequel est posé la ville qui est sous le demi-dôme où est affichée la voûte céleste. On voit aussi les outils interactifs : deux molettes (en vert) pour diriger le télescope et le levier de zoom (en rouge).

## Test de la projection

**Protocole**

Un projecteur est utilisé pour projeter sur le dôme les différents astres et nous voulons nous assurer que les éléments projetés à proximité des bords de la surface ne sont pas déformés. Nous effectuons ce test par comparaison visuelle entre la projection de la vue originale de la scène et la projection de cette même vue, déformée par un effet *fisheye*.

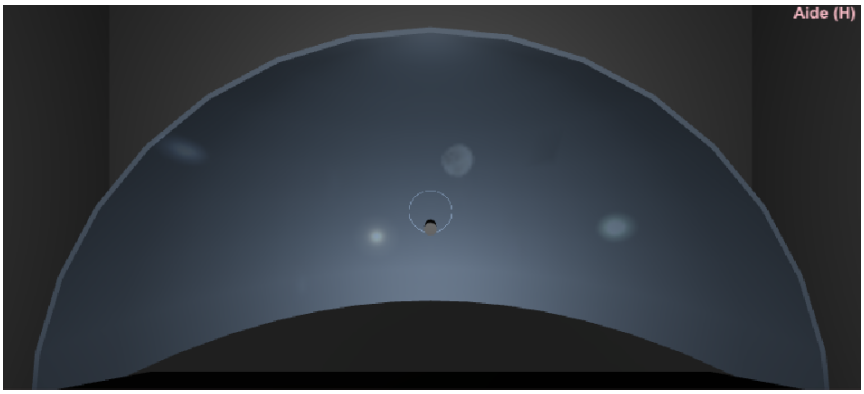



FIGURE 27: Capture d'écran de la projection déformée sur les bords.

**Résultats**

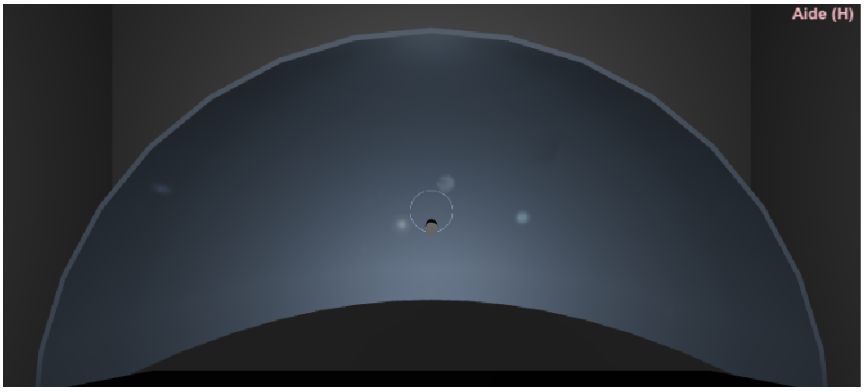



FIGURE 28: Capture d'écran de la projection avec l'effet fisheye.

**Conclusion**

L'utilisation d'un effet *fisheye* appliqué à la projection permet de réduire la déformation perceptible sur les bords du dôme, comme on peut le constater sur les captures d'écrans 27 et 28.

### Test de la rotation du télescope

**Protocole**

Pour tester que l'on puisse déplacer la vue du télescope, on lui applique une rotation arbitraire et on s'assure que la rotation aie bien été appliquée à l'objet. De plus, la rotation du télescope ne s'effectue que sur deux axes ; on s'assure donc aussi que lorsqu'une rotation est appliquée sur l'un des deux axes de rotation le troisième n'est pas modifié.

**Résultats**

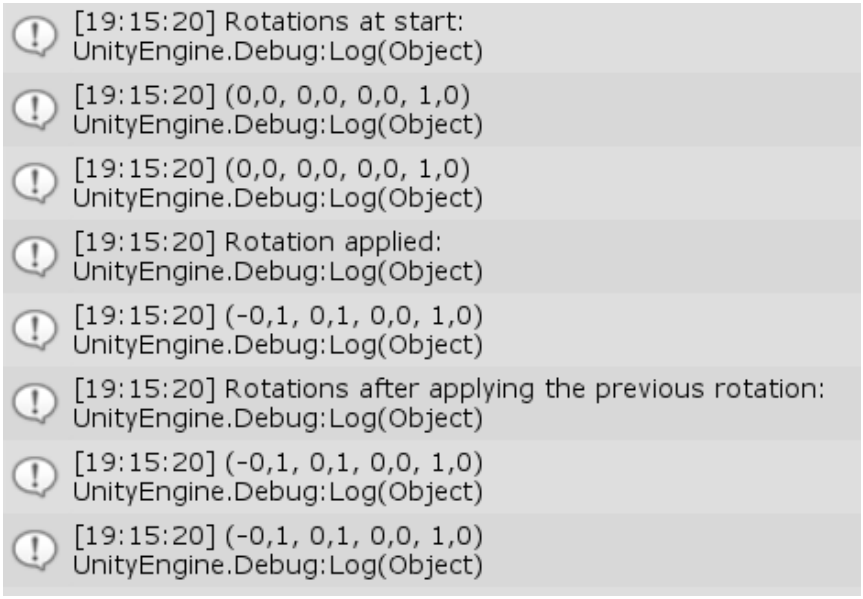



FIGURE 29: Capture d'écran de la console de test.

**Conclusion**

On a bien des rotations sur les deux axes désirés sans que le troisième axe ne soit affecté, ce qui signifie que l'on peut effectivement déplacer la vue du télescope.

### Test de la vue en première personne

**Protocole**

On souhaite pour ce besoin confirmer que l'on peut donner à l'utilisateur une sensation d'être posé sur une planète à la première personne, sans que cela soit observé par un télescope lointain. On observe visuellement une image de résultat de notre programme afin de confirmer cela.

**Résultats**

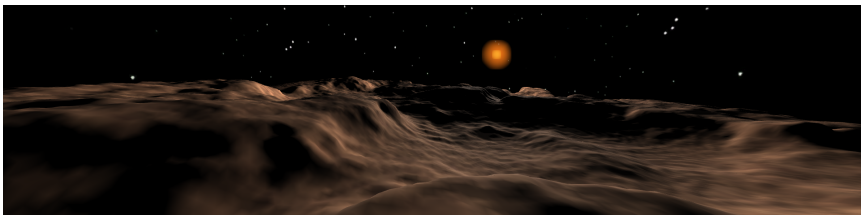


FIGURE 30: Capture d'écran d'une partie de notre programme où l'on peut voir une scène représentant Mars vue à la 1ère personne.

**Conclusion**

Le résultat présenté par la figure 30 nous montre bien que la caméra de l'utilisateur semble posée sur le relief de la planète. L'utilisateur ne peut cependant pas se déplacer ou tourner sa caméra, afin d'empêcher qu'il ne sorte de la scène modélisée pour cette affichage.

## Test de la fenêtre de la carte étoilée

**Protocole**

On souhaite confirmer que nous disposons, conformément aux besoins, d'une fenêtre affichant une carte étoilée qui correspond à un affichage à plat de ce qui est visible sur le demi-dôme de la maquette (fig 31). On prends une capture d'écran de notre programme afin de la comparer visuellement au résultat attendu représenté par notre croquis.

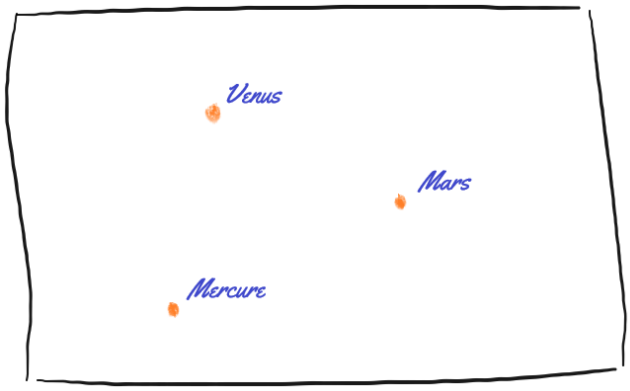



FIGURE 31: Croquis d'un affichage type de la carte céleste avec des astres quelconques affichés et nommés.

**Résultats**


FIGURE 32: Capture d'écran d'une partie de notre programme où l'on peut voir dans la carte céleste des astres implémentés.

**Conclusion**

On remarque que le résultat présenté par la figure 32 est conforme à ce qui a été présenté ci-dessus. On dispose bien d'une fenêtre distincte pour la visualisation de la carte céleste. De plus, cette carte céleste est projetée sur la maquette, les deux représentations sont donc identiques entre elles.

## Test de l'overlay

**Protocole**

On souhaite disposer en *overlay* sur la projection d'une indication pour que l'utilisateur puisse repérer plus facilement à l'oeil nu, sur le dôme, la zone ciblée par le télescope. Pour ce test, on s'assure donc premièrement que le télescope et le projecteur de l'*overlay* ont la même rotation. On applique ensuite une rotation au télescope et on vérifie à nouveau que leurs rotations sont identiques.

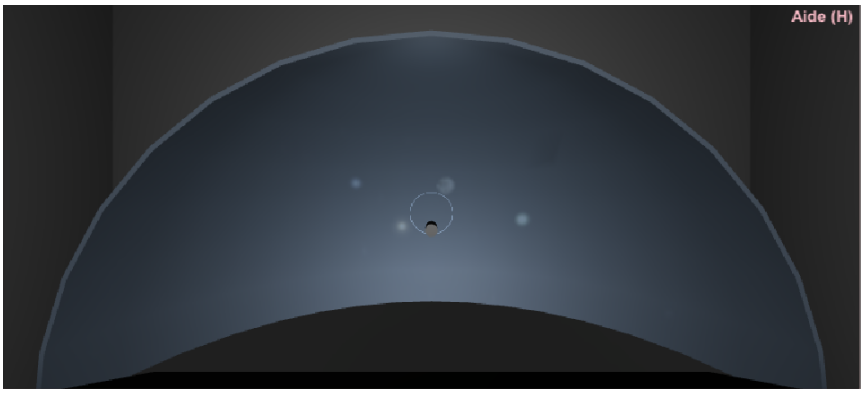



FIGURE 33: Capture d'écran du télescope et de l'*overlay* au démarrage.

**Résultats**

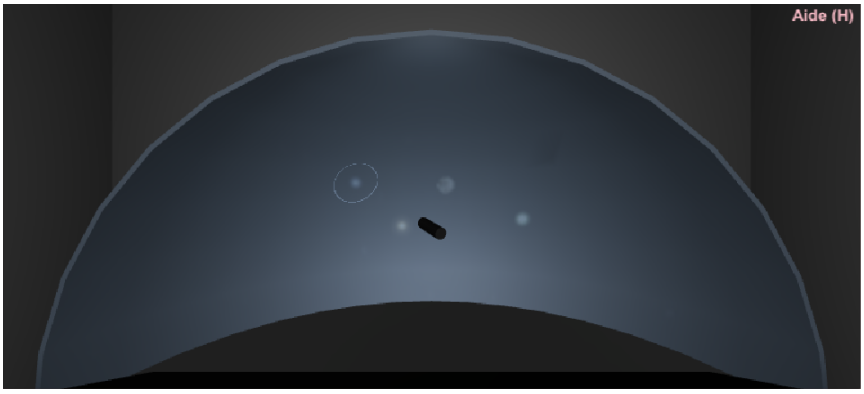



FIGURE 34: Capture d'écran du télescope et de l'*overlay* après rotation du télescope.

```
Rotations at start
Rotation of the telescope : (0,0, 0,0, 0,0)
Rotation of the projector : (0,0, 0,0, 0,0)
Rotations after applying a rotation to the telescope
Rotation of the telescope : (350,0, 10,0, 0,0)
Rotation of the projector : (350,0, 10,0, 0,0)
```

FIGURE 35: Rotations des deux objets avant et après avoir appliqué une rotation au télescope.

**Conclusion** Les rotations appliquées au télescope sont aussi appliquées au projecteur de l'*overlay*, ce dernier indique donc correctement la zone ciblée par le télescope.

### Test de variation de la vitesse de rotation du télescope

**Protocole** Ce test a pour objectif de vérifier que notre vitesse de rotation du télescope est bien réduite en fonction du niveau de zoom (un zoom important implique une vitesse de rotation plus faible).
Pour se faire on dispose d'une valeur indiquant la vitesse de rotation de l'objet. On récupère cette valeur correspondant on la vitesse de rotation initiale, on applique un zoom puis on récupère une nouvelle fois cette valeur en la comparant à la première.

**Résultats**

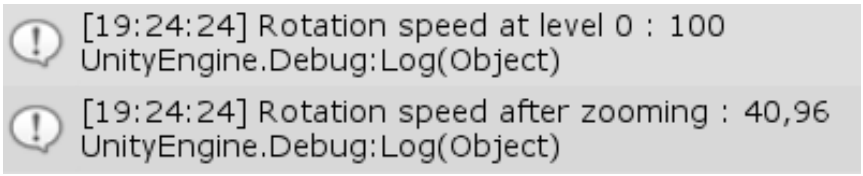



FIGURE 36: Capture d'écran des résultats de la console de test.

**Conclusion** Une fois le zoom effectué la vitesse de rotation est plus faible, ce qui signifie que cette vitesse s'adapte correctement au niveau de zoom.

### Test de l'activation de la stéréoscopie

**Protocole** L'objectif de ce test est de s'assurer que l'on puisse activer et désactiver la vue stéréoscopique. On dispose pour cela d'une caméra mono et de deux caméras pour la stéréoscopie. Lorsqu'on active la stéréoscopie on vérifie que seules les caméras pour la stéréoscopie sont actives et lorsqu'on la désactive on vérifie que seule la caméra mono soit active.

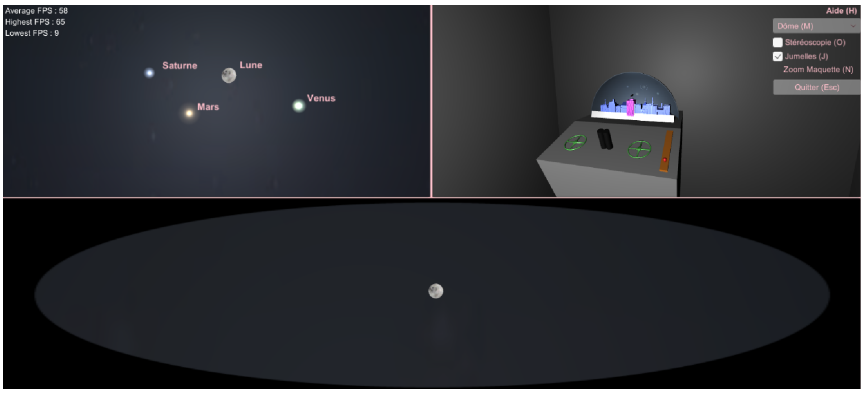



FIGURE 37: Capture d'écran sans la stéréoscopie.

**Résultats**

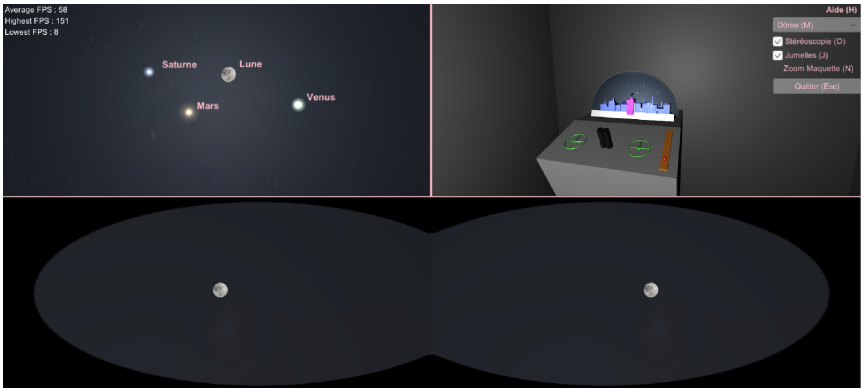



FIGURE 38: Capture d'écran avec la stéréoscopie.

**Conclusion**

Avec la stéréoscopie activée, seules les caméras dédiées à son fonctionnement sont actives et dans le cas contraire seule la caméra mono est active. On peut de plus constater ce changement directement à l'écran dans la fenêtre de rendu comme le montrent les figures 37 et 38.

### Test des transitions de niveau de zoom

| | |
|---|---|
| **Protocole** | Pour ce test, on souhaite s'assurer qu'un effet de fading approprié est bien appliqué à notre caméra lorsqu'un zoom est effectué. Ceci est effectué par deux tests.<br>Le premier vérifie que toutes les étapes de la transition sont appliquées lors d'un zoom : pas de fading –> en applique le fading –> fading effectué –> on retire le fading –> pas de fading.<br>Le second utilise un timer pour vérifier que l'effet de fading s'applique sur une durée qui correspond à celle qui lui est passée en paramètre. |
| **Résultats** | Pour le premier test, on confirme par des assertions régulières que le cycle d'étapes est entièrement parcouru une fois un zoom entamé.<br>Pour le second test, avec une durée de fading de 1 seconde, on confirme que l'on atteint le point central du cycle (étape "fading effectué") 500ms, avec une marge d'erreur de 50ms, après avoir commencé le zoom. On atteint aussi la fin du cycle de transition 1000ms après le zoom, avec une marge d'erreur de 100ms. |
| **Conclusion** | On a bien un effet de fading appliqué. Celui-ci se déroule de la manière attendue est s'applique bien sur la durée souhaitée, ce qui permet de le synchroniser correctement avec l'effet de zoom en lui-même. |

### Test d'expérience utilisateur

| | |
|---|---|
| **Protocole** | L'objectif de ce test est d'avoir un retour sur l'expérience des utilisateurs sur notre prototype, afin de savoir les modifications qui apporterai le plus d'impact. Pour cela, nous avons envoyé à une quinzaine de personnes un questionnaire (Annexe B) sur l'utilisation et l'ergonomie de notre prototype. |

Le questionnaire a été répondu par neuf personnes à l'heure de la rédaction de ce rapport. Certains des résultats sont présentés ci-dessous :

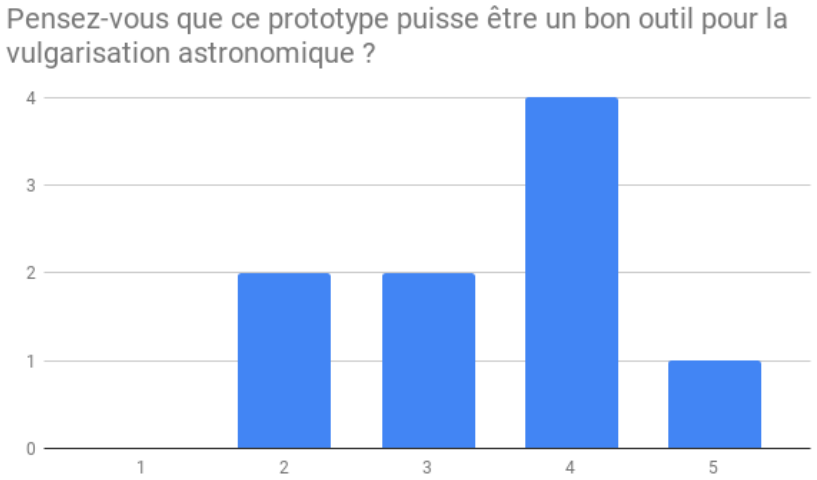



FIGURE 39: Graphique des réponses de la vulgarisation astronomique de notre prototype

**Résultats**

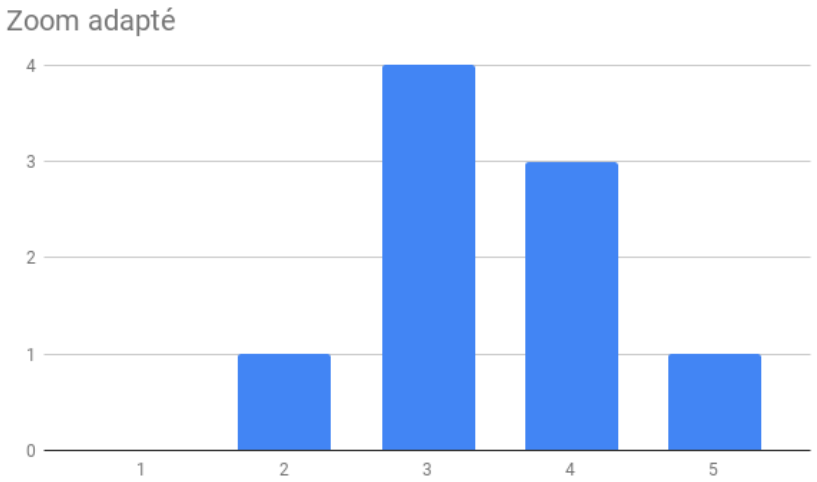



FIGURE 40: Graphique des réponses d'un zoom adapté pour notre prototype

Vous pouvez trouver sur ce lien toutes les réponses : `https://docs.google.com/forms/d/1PfSdzUtX-qHY2Jf_Yoltyqv2Hl5kibZoBo7btvnnIYQ/edit?usp=sharing`

**Conclusion**

D'après ce test, il faudra apporter des modifications sur l'ergonomie et sur certaines interactions pour que le prototype soit plus adapté aux utilisateurs.

### Test de fluidité

**Protocole**

L'application nécessite, pour avoir une expérience de la réalité virtuelle agréable, un taux d'images par seconde suffisamment important. On considère qu'à partir de 60 images par secondes la fluidité est suffisamment importante pour l'oeil humain et ainsi éviter à l'utilisateur d'être malade. Le test consiste donc à calculer le taux d'images par seconde lorsque l'application est lancée pour s'assurer qu'on atteigne ce seuil ou que l'on s'en rapproche suffisamment. Ce test est réalisé avec les configurations machine détaillées dans le cahier des charges.

**Résultats**

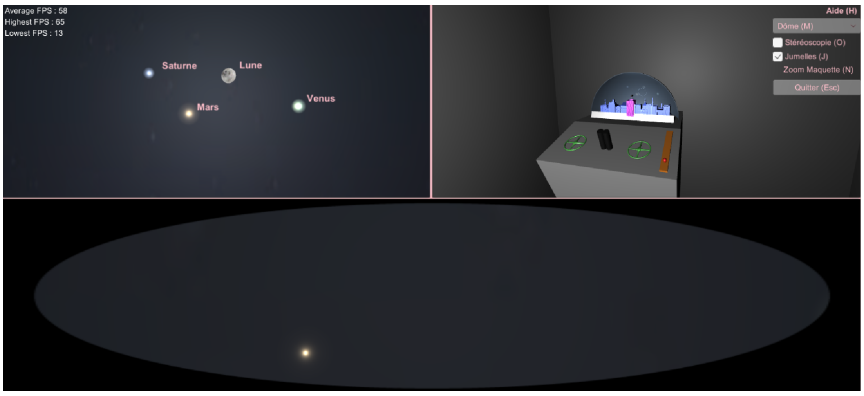



FIGURE 41: Capture d'écran avec affichage des images par seconde.

**Conclusion**

Comme on peut le constater sur la figure 41 la fluidité est convenable. En moyenne on avoisine les 58 images par seconde ce qui est suffisamment proche du seuil des 60 images par seconde pour que l'utilisation de l'application soit agréable à l'oeil. Le minimum affiché semble bas mais est dû à la manière dont la capture d'écran est prise, qui passe pendant quelques secondes l'application en arrière plan et diminue le nombre d'images par seconde sur cette période.

# 6 Conclusion

L'objectif était de disposer d'une application permettant à l'utilisateur d'expérimenter une simulation d'un visualisation à travers un télescope en réalité virtuelle. Le dispositif physique qui

accompagnera l'application n'est pas encore mis en place et une fenêtre permettant de simuler ce dispositif était aussi nécessaire afin de pouvoir se projeter plus aisément dans ce que sera l'expérience finale pour l'utilisateur. Ces deux fonctionnalités ont été mises en place et interagissent entre elles, permettant à l'utilisateur d'expérimenter à la fois une vision de l'espace en réalité virtuelle et l'utilisation future du dispositif physique.

Cependant, les objets visualisés en réalité virtuelle sont actuellement détectés virtuellement dans l'application. L'objectif serait, une fois le dispositif physique fabriqué, de disposer d'objets physiques positionnés dessus et détectés. Sans le matériel notre application n'en est donc actuellement pas capable. Une solution possible à mettre en place serait que pour chaque objet physique à détecter lui associer une rotation pour que le télescope soit dessus et la position de son équivalent virtuelle dans la scène au niveau 0. Ces associations devront être faites dans Unity et donc en amont de l'utilisation du dispositif.
Une autre solution serait de conserver la projection des éléments virtuels, de positionner ces derniers selon les besoins et de s'en servir en amont pour calibrer le dispositif et positionner par les suite les objet physiques avant de désactiver cette projection. Ainsi chaque objet physique serait positionné exactement à l'emplacement correspondant à son équivalent virtuel.

# 7 Références


[1] J. Sol Roo and M. Hachet, "Towards a Hybrid Space Combining Spatial Augmented Reality and Virtual Reality," in *3DUI - IEEE Symposium on 3D User Interfaces*, ser. 3DUI 17. Los Angeles, United States : IEEE, Mar. 2017. [Online]. Available : https://hal.archives-ouvertes.fr/hal-01453385

[2] ——, "One Reality : Augmenting How the Physical World is Experienced by combining Multiple Mixed Reality Modalities," in *UIST 2017 - 30th ACM Symposium on User Interface Software and Technology*, ser. ACM UIST'17. Quebec City, Canada : ACM, Oct. 2017. [Online]. Available : https://hal.archives-ouvertes.fr/hal-01572490

[3] J. Sol Roo, J. Basset, P.-A. Cinquin, and M. Hachet, "Understanding Users' Capability to Transfer Information between Mixed and Virtual Reality : Position Estimation across Modalities and Perspectives," in *CHI '18 - Conference on Human Factors in Computing System*. Montreal, Canada : ACM Press, Apr. 2018. [Online]. Available : https://hal.archives-ouvertes.fr/hal-01718590

[4] F. Chéreau, "Stellarium," accessed : 2019-03-15. [Online]. Available : https://stellarium.org/

[5] National Aeronautics and Space Administration, "NASA 3D Resources," accessed : 2019-03-15. [Online]. Available : https://nasa3d.arc.nasa.gov/

# Annexe A - Ressources utilisées

| Ressource | Format | Source |
|---|---|---|
| Apollo 17 Landing Site | stl | https://nasa3d.arc.nasa.gov/detail/Apollo17-Landing |
| Buildings | obj | https://free3d.com/3d-model/19-low-poly-buildings-974347.html |
| Curiosity Landing Site | stl | https://nasa3d.arc.nasa.gov/detail/CuriosityQR |
| Dome Tools | N/A | https://assetstore.unity.com/packages/vfx/shaders/fullscreen-camera-effects/dome-tools-62664 |
| Globe Icon | png | https://www.psdgraphics.com/psd-icons/photoshop-world-globe-icon/ |
| Leyland Steering Wheel | obj | https://sketchfab.com/3d-models/leyland-steering-wheel-01e3c476a3cd4d1db9dc8ded131a6d13 |
| Mars | jpg | http://www.supinemusic.net/Files%20for%20Cloud%20projects/Cinema%204D/plugins/Planet%20X%20Generator%20R12/presets/ |
| Mars - NormalMap | jpg | http://www.supinemusic.net/Files%20for%20Cloud%20projects/Cinema%204D/plugins/Planet%20X%20Generator%20R12/presets/ |
| Moon | jpg | http://www.supinemusic.net/Files%20for%20Cloud%20projects/Cinema%204D/plugins/Planet%20X%20Generator%20R12/presets/ |
| Moon - NormalMap | jpg | http://www.supinemusic.net/Files%20for%20Cloud%20projects/Cinema%204D/plugins/Planet%20X%20Generator%20R12/presets/ |
| Post Processing Stack | N/A | https://assetstore.unity.com/packages/essentials/post-processing-stack-83912 |
| Saturn | jpg | http://www.supinemusic.net/Files%20for%20Cloud%20projects/Cinema%204D/plugins/Planet%20X%20Generator%20R12/presets/ |
| Saturn - NormalMap | jpg | http://www.supinemusic.net/Files%20for%20Cloud%20projects/Cinema%204D/plugins/Planet%20X%20Generator%20R12/presets/ |
| Saturn Ring | obj | http://spacecraftkits.com/SatRings.html |
| Saturn Ring Color | jpg | http://planetpixelemporium.com/saturn.html |
| Starry Night | jpg | https://all-free-download.com/free-photos/download/starry-night_192398.html |
| Venus | jpg | http://www.supinemusic.net/Files%20for%20Cloud%20projects/Cinema%204D/plugins/Planet%20X%20Generator%20R12/presets/ |
| Venus - NormalMap | jpg | http://www.supinemusic.net/Files%20for%20Cloud%20projects/Cinema%204D/plugins/Planet%20X%20Generator%20R12/presets/ |

# Annexe B - Formulaire UX



## Échelles Célestes

*Obligatoire

### Utilisateur

1. **Êtes-vous étudiant en Master 2 Informatique ?** *
   *Une seule réponse possible.*
   - ◯ Oui
   - ◯ Non

2. **Avez-vous des bonnes connaissances en astronomie ?** *
   *Une seule réponse possible.*
   - ◯ Oui
   - ◯ Non

### Utilisation du Système

3. **Trouvez-vous l'utilisation de ce prototype intuitive ?** *
   *Une seule réponse possible.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| fortement en désaccord | ◯ | ◯ | ◯ | ◯ | ◯ | fortement en accord |

4. **Imaginez-vous que la plupart des gens pourront apprendre à utiliser ce prototype rapidement ?** *
   *Une seule réponse possible.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| fortement en désaccord | ◯ | ◯ | ◯ | ◯ | ◯ | fortement en accord |

5. **Pensez-vous que ce prototype puisse être un bon outil pour la vulgarisation astronomique ?** *
   *Une seule réponse possible.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| fortement en désaccord | ◯ | ◯ | ◯ | ◯ | ◯ | fortement en accord |

6. **Quelles parties du prototype sont les plus ou les moins importantes pour vous ?**

7. **Comment pourrions-nous présenter l'information d'une manière plus significative ?**

8. **Il y a-t-il quelque chose que vous changeriez, ajouteriez ou enlèveriez pour améliorer le prototype ?**

## Ergonomie

9. *Une seule réponse possible.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Déplaisant | ◯ | ◯ | ◯ | ◯ | ◯ | Plaisant |

10. *Une seule réponse possible.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Laid | ◯ | ◯ | ◯ | ◯ | ◯ | Beau |

11. *Une seule réponse possible.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Rotation gênante | ◯ | ◯ | ◯ | ◯ | ◯ | Rotation adaptée |

12. *Une seule réponse possible.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Zoom gênant | ◯ | ◯ | ◯ | ◯ | ◯ | Zoom adapté |