◁ ◁ ◁ COMPTE RENDU FINAL ▷ ▷ ▷

◇ ◇ ◇

PTS8 : SISN

⋆ ⋆ ⋆

Interstellar quest

Chef de groupe : Elliot DREES
Quentin LENET, Jérémie BOULIC, Benjamin HABIÉ

Élèves-ingénieurs à l'École Centrale de Marseille

20 mai 2017

Table des matières

1 Introduction **3**
1.1 Idée générale du jeu . . . 3
1.2 Outils utilisés lors du projet . . . 3
1.2.1 Forge de Centrale Marseille . . . 3
1.2.2 Google Drive . . . 3
1.2.3 Sharelatex . . . 3
1.2.4 Facebook . . . 4
1.3 Répartition du travail dans le groupe . . . 4

2 Présentation de JME **5**
2.1 Introduction . . . 5
2.2 Vue d'ensemble du moteur de jeu . . . 5
2.3 Affichage et logiques de jeu . . . 5
2.4 Open Audio Library . . . 6
2.5 Nifty GUI . . . 6

3 Objectifs du projet **7**
3.1 Logique de jeu . . . 7
3.1.1 Mécanique d'orbite . . . 7
3.1.2 Simulation physique basique . . . 7
3.1.3 Contrôle de vaisseau . . . 7
3.2 Interface développeur . . . 7
3.3 Musique et son de jeu . . . 7
3.4 Modèles 3D . . . 7

4 Travaux réalisés **8**
4.1 Logique de jeu . . . 8
4.1.1 Simulation orbitale . . . 8
4.1.2 Simulation physique basique . . . 9
4.1.3 Une intelligence artificielle basique pour des vaisseaux autonomes . . . 11
4.1.4 Gestion de l'énergie . . . 11
4.2 Interface Développeur . . . 11
4.2.1 Menu basique . . . 12
4.2.2 Intégration avec la logique de jeu . . . 12
4.3 Musique et effets de jeu . . . 12
4.4 Modèles 3D . . . 13
4.4.1 Modèles de base . . . 14
4.4.2 Première faction : Tantra Dawn . . . 14
4.4.3 Seconde faction : Rock Nebulace . . . 16
4.4.4 Missile . . . 17

5 Bibliographie **18**

6 Annexe **19**

1 Introduction

1.1 Idée générale du jeu

Interstellar quest est un jeu de simulation de combat spatial basés sur deux points importants :

— La coopération :

  Le jeu final sera principalement un jeu d'équipe. Chaque joueur fera parti d'un équipage de vaisseau spatial, en tant que tel il aura donc un poste qui contribuera au fonctionnement du vaisseau, allant du capitaine à l'ingénieur en chef en passant par le pilote.

— Le combat :

  Le jeu devra comporter des mécaniques de combat assez complètes avec une gestion de la distribution d'énergie dans les différent sous-systèmes. Différent armements seront disponibles, des munitions laser, des projectiles, des missiles ou autre.

L'idée du jeu vient de deux autres jeux qui comportent chacun un des deux aspects : Artemis (pour la partie coopération) et Star Trek Bridge Commander (pour la partie combat).
L'idée est de pousser l'idée des deux jeux plus loin à la fois en les combinant et développant encore plus certains aspects.

1.2 Outils utilisés lors du projet

1.2.1 Forge de Centrale Marseille

Nous avons mis en place un dépôt Git sur la forge de centrale afin de faciliter le travail en équipe sur le projet.

Nous avions deux branches principales pour le projet :

— Master : La branche principale, elle est la version stable du jeu, c'est sur cette version que se développe l'interface développeur en attendant que la branche de logique de jeu soit assez avancée pour pouvoir faire le lien entre Interface et jeu.

— gameplayDev : La branche secondaire pour travailler la logique de jeu. Cette branche n'est pas une branche stable. C'est sur celle-ci que l'on a travaillé la simulation physique ainsi que le système de gestion d'énergie.

  C'est sur cette branche que les éléments de gameplay se développent et c'est de celle là que les rebase se feront.

1.2.2 Google Drive

Nous utilisons un Google Drive afin de stocker nos dossiers de travail. Tous nos comptes-rendus de séance, avec toutes les données y sont stockés.
Le dossier est scindé en sous-dossiers de la manière suivante :

— Un dossier pour les modèles 3D

— Un dossier pour les diagrammes UML ;

— Un dossier pour les musiques et effets spéciaux ;

— Un dossier pour les comptes rendus et présentations.

Un autre avantage du drive est que l'on peut commenter le travail des autres sur le drive, ce qui facilite l'ajout de remarques, la correction d'oublis ou encore la réponse à certains questionnements.

1.2.3 Sharelatex

Nous utilisons aussi une plateforme collaborative, afin de faire nos comptes-rendus au propre, sous format LaTeX. L'avantage du LaTeX est que l'on peut avoir de nombreux sous-fichiers, qui correspondent à chaque partie de notre compte-rendu et que chaque personne peut le modifier à sa guise et simultanément.

Cela permet à plusieurs personnes de travailler sur différentes parties du document, sans gêner les autres (ex : un sur la présentation, l'autre sur le contenu).

URL de la plateforme :
https://fr.sharelatex.com

1.2.4 Facebook

Pour les discussions et l'organisation temporelle au jour le jour, le groupe a choisi d'utiliser le système de discussion instantanée de Facebook.

Nous utilisons également un groupe Facebook, qui nous permet de disposer de toutes les informations importantes pour le projet et des liens vers notre Google Drive, ainsi que notre Sharelatex, que nous utilisons pour rédiger en simultané nos comptes rendus.

1.3 Répartition du travail dans le groupe

Nous avons réparti le groupe en 3 èquipes de travail :

— Quentin Lenet s'occupait de l'élaboration de l'interface développeur.
— Benjamin Habié était chargé de faire les modèles 3D
— Jérémie Boulic, Elliot Drees étaient chargés de faire la partie logique de jeu et programmation.

2 Présentation de JME

2.1 Introduction

JMonkey Engine est un moteur de jeu open source sur java. Il vient avec sa propre interface qui est une interface modifiée de Netbeans.

Il est conçu pour faire des jeux en 3D en utilisant les technologies modernes, allant des manettes jusqu'au smartphones : il y a en effet une compatibilité Mac/Windows/Linux iOS/Android. Ce moteur est entièrement en java et optimisé pour être efficace. Il utilise OpenGL pour l'affichage, ces librairies sont en partie issues de LWJGL.

2.2 Vue d'ensemble du moteur de jeu

Le moteur de jeu est un système qui permet de faciliter le développement de jeux, il dispose donc de plusieurs outils pour faciliter le développement :

- Un système d'appState qui permettent de basculer dans différent états de jeu.
- Un système nodal pour faire affichage et logique de jeu.
  Le moteur classe tous les éléments dans un graphe. Cela permet d'attacher ou de détacher des entités autant de l'affichage que de la logique de jeu.
- Un système pour les entrées du joueur (InputManager)
  Ce système permet de gérer des entrées clavier, souris, et potentiellement des manettes.
- Un système pour gérer l'affichage
  Les modèles 3D peuvent être importés de modèles blender. Les objets peuvent avoir plusieurs genre de texture : couleur, normale, émission de lumière...
  Il peut aussi faire des effets avec des particules ou alors des post process qui permettent de donner un rendu encore plus joli.
  Pour faire les interfaces utilisateur, le moteur de jeu intègre Nifty GUI.
- Un moteur physique
  JME possède un système pour simuler une physique, cela n'est malheureusement pas applicable dans notre cas en raison de la grande taille du système solaire, qui solliciterait trop le moteur physique.
- OpenAL
  Open Audio Library sert à diffuser du son dans le jeu, il peut faire des sources sonores localisées comme des source omniprésentes.

2.3 Affichage et logiques de jeu

Comme dit ci-dessus il y a deux étapes séparées dans le moteur de jeu :

Les AppState
On peut considérer un jeu comme une machine à états, selon l'état logique du jeu différentes actions seront possible.

C'est le cas quand on passe du menu principal au jeu en lui même. Chaque écran différent propose des états logiques différents. Et c'est ce qui ce passe avec les appState. On va "Attacher" un appState pour le rendre actif et une fois que son utilité passée on peut le "Détacher. Plusieurs états peuvent êtres actifs en même temps ce permet la superposition d'états (comme par exemple un menu plus d'un affichage en arrière plan).

Les Nodes d'affichages
Pour afficher un objet 3D dans JMonkey Engine, il faut passer par son système nodal.
Chaque objet à être afficher doivent être attachés à la node principale appelée "rootNode". L'afficheur du jeu va ensuite parcourir toute les branches qui partent de la node principale pour les afficher.

Mais ces nodes sont aussi là pour la logique de jeu. On peut attacher des contrôle à ces nodes qui vont leur permettre d'interagir entre elles. Cela va permettre de faire ce que l'on veut avec les éléments qui sont attachés. On peut même créer des nodes qui ne sont pas affichées (qui sont notées "invisible"). Cela permet de faire marcher des logiques de jeu dans l'arrière plan.

Les post process
Un post process est un effet qui se fait après le render de toute la scène. Il permet d'ajouter des effets sur la scène globale.

Cela peut être d'une variation de couleur dans l'ensemble de la scène, mais cela peut aussi être un effet de bloom, un effet de flou, un lense flare. Ces post process sont gérés par des shader qui sont totalement modifiable par le programmeur pour avoir l'effet désiré.

2.4 Open Audio Library

Afin de pouvoir rendre le jeu plus vivant, il est préférable de d'ajouter du son et des effets sonores. C'est la librairie OpenAL qui rentre en compte dans ce cas. Un son peut être diffusé de 3 manière différentes :

- Soit de manière directe sur les hauts parleurs.
- Soit un cône directionnel, cela permet de remplir une zone réduite de son.
- Soit une sphère avec un son qui se dissipe en fonction de la distance du son. Cela permet de simuler des source de sons qui sont éloignées.

Il y a deux manière de jouer un son, soit de le charger au préalable (pre-buffer), soit de le lire en instantané (stream). Le fait de charger au préalable le son permet d'être plus libre au niveau de son utilisation, c'est à dire pouvoir commencer d'où on veut. En revanche le temps de chargement peut être gênant pour l'intégration dans le jeu si le fichier est volumineux.

2.5 Nifty GUI

Nifty GUI est une librairie Java dédiée aux interfaces graphiques. Elle utilise OpenGL pour être affichée à l'écran sous forme de post-process. L'affichage est donc indépendant de la structure nodale de JMonkeyEngine, ce qui permet de bien séparer les parties GUI/logique de jeu. La partie Nifty dispose donc de sa propre appState.
La librairie contient un certain nombre des fonctionnalités intégrées pour l'interactivité, comme des boutons, sliders, consoles etc... Charge est au développeur de créer et développer les fonctions qui seront activées lors de l'interaction avec les éléments de l'interface. Les input manager de JME3 sont également utilisables avec Nifty GUI.
Nifty n'implémente que la partie visuelle et interactive du GUI. C'est le développeur qui doit se charger de réaliser le lien entre l'interface et la logique de jeu.

3 Objectifs du projet

Le but du projet est de réaliser une première esquisse d'un jeu vidéo de simulation de combat spatial. Le jeu dans son état final devra comporter trois aspect principaux :

- Une simulation physique et orbitale acceptable
- Avoir un système de combat élaboré
- Mettre en avant la coopération entre joueurs

3.1 Logique de jeu

3.1.1 Mécanique d'orbite

Le jeu devra comporter une simulation orbitale basique. Pour simplifier, nous nous limiterons simplement à un problème à deux corps, afin de simplifier les calculs et pouvoir les faire en temps réel (pour ne pas ralentir les images par secondes).

Une fois ce système mis en place, il pourra être appliqué aussi bien pour les astres d'un système solaire, que pour les vaisseaux individuels.

3.1.2 Simulation physique basique

Afin de gérer les combats spatiaux, nous ferons une simulation newtonienne basique afin de faciliter les combats et la représentation du joueur dans l'espace. Cette physique ne sera appliquée que lors des combat et changements d'orbites, ceci dans le but d'éviter des calculs inutiles lorsque les vaisseau sont éloignés .

3.1.3 Contrôle de vaisseau

Chaque vaisseau aura des caractéristiques communes :

- Des points de vie
- Un système qui gère l'énergie
- Des sous systèmes séparés pour chaque fonction

Le contrôle des vaisseaux devra donc gérer toutes les différentes action que peut réaliser un vaisseau.

3.2 Interface développeur

Afin de faciliter le développement nous ferons une interface développeur, le but étant d'avoir une interface qui nous permet de tester facilement les nouvelles fonctions qui ont étés implémentés dans le jeu.

Il devra comporter une interface utilisateur simple pour ajouter des éléments de jeu que l'on veut tester.

3.3 Musique et son de jeu

Nous ferons quelques musiques et effets de jeux qui ne seront pas forcément implémentés mais qui pourront l'être dès que le jeu le permet.

3.4 Modèles 3D

Nous créerons des modèles 3D que nous implémenterons dans le jeu. Le but étant d'avoir deux factions discernables que l'on peut intégrer dans le jeu.

4 Travaux réalisés

4.1 Logique de jeu

4.1.1 Simulation orbitale

Comme dit précédemment, nous avons fait une simulation simplifiée de mécanique orbitale. Cela nous a permis d'appliquer des équations simplifiées.

Caractéristiques d'un orbite
Avant de pouvoir faire évoluer au cours du temps l'objet qui gravite, nous devons avoir les caractéristique orbitales de l'objet.

Ces caractéristiques suivantes :

- L'énergie orbitale : E
- Le moment angulaire spécifique : σ_s
- Le demi grand axe : a
- L'excentricité : e
- Temps de passage au périgée : t_p

Une orbite peut être elliptique ($E < 0$), hyperbolique ($E > 0$) ou parabolique ($E = 0$). Afin de simplifier les fonctions, nous ignorerons les équations paraboliques et nous les basculerons sur une équation hyperbolique car c'est un cas marginal lorsqu'on travaille avec des doubles.

Déterminer les caractéristique d'une orbite
En premier, nous devons calculer l'énergie orbitale afin de savoir quel est le type de l'orbite, on calculera donc E de la manière suivante :

$$E = \frac{\|\vec{V}\|^2}{2} - \frac{G \cdot m \cdot m'}{r}$$

Avec G la constante gravitationnelle, m, m' les masses des deux objets, r la distance des centre de gravité et $\vec{V}$ le vecteur vitesse de l'objet orbitant par rapport à l'objet orbité.

Nous cherchons à présent à déterminer la longueur du demi grand axe ainsi que les vecteurs directeurs du grand et petit axe. Nous cherchons donc à avoir l'excentricité de l'orbite pour déterminer le grand axe.

Nous commençons par calculer le moment angulaire spécifique :

$$\vec{\sigma_s} = \vec{r} * \vec{V}$$

Puis nous calculons le vecteur excentricité $\vec{e}$ qui est le vecteur orienté selon le demi grand axe en direction du périgée et de module e :

$$\vec{e} = \frac{\vec{V} * \vec{\sigma_s}}{G \cdot m \cdot m'}$$

Nous avons à présent le sens du demi grand axe et la valeur de l'excentricité de l'orbite. Nous pouvons à présent trouver les caractéristiques de notre orbite.

$$a = -\frac{G \cdot m \cdot m'}{2 \cdot E}$$

On remarquera que a peut être négatif, lorsque l'orbite est hyperbolique.

Afin de pouvoir créer notre orbite, nous créons la base orthonormée qui sera le support du traçage de notre orbite et pour les déplacements sur l'orbite.

$$\vec{x} = \frac{\vec{e}}{\|e\|}$$

$$\vec{y} = \frac{\vec{\sigma_s} * \vec{e}}{\|\vec{\sigma_s} * \vec{e}\|}$$

Utiliser les caractéristiques des orbites pour placer un objet

Dans le cas ou l'on a les caractéristiques du système, on peut passer de la position au temps ou l'inverse (on passe justement de la position initiale au temps de position au périgée après avoir calculé les caractéristique pour avoir t_p).
Pour faire cela, on va devoir passer par plusieurs étapes, les équations varient en fonction du type d'orbite, le equations si dessous sont les équations dans le cas d'orbite Hyperboliques :

1. Passer de la position à l'anomalie vraie
   On utilise soit un calcul d'angle entre la position et la base orthonormée de l'orbite, soit la base pour passer de l'angle à la position. (C'est plus un calcul géométrique et n'est pas vraiment lié à une équation particulière)
2. Passer de l'anomalie vraie à l'anomalie excentrique
   $$cosh(\theta_e) = \frac{e + cos(\theta)}{1 + e \cdot cos(\theta)}$$
3. Passer de l'anomalie excentrique à l'anomalie moyenne
   $$\theta_M = e \cdot sinh(\theta_e) - \theta_e$$
4. Passer de l'anomalie moyenne au temps
   $$t - t_p = \theta_M \cdot \sqrt{\frac{(-a)^3}{G \cdot m \cdot m'}}$$

Remarque : ces équations ne sont pas facile à inverser, on utilise alors la méthode de Newton pour approximer la solution à l'étape 2 et 3 quand on veut passer du temps à la position.

Vous pourrez voir des exemples d'orbites en annexe.

4.1.2 Simulation physique basique

Discrétisation du Principe Fondamental de la Dynamique

Principe Fondamental de la Dynamique :

$$m\vec{a} = \vec{F} - f\vec{v}$$

Discrétisation :
le paramètre tpf (time per frame) permet d'obtenir le temps séparant 2 trames. On notera donc par la suite le temps infinitésimal $\delta t = tpf$

Les forces de friction :
Admettons que les forces appliquées à l'entité soient nulles. La discrétisation de l'équation précédente aboutit à l'expression suivante :

$$m\frac{v_{t+1} - v_t}{\delta t} = -fv_t \text{ soit } v_{t+1} = (1 - \frac{\delta t}{m}f)v_t$$

Quelques calculs permettent de se rendre compte qu'avec cette relation, le comportement de la friction dépend de δt, et donc de la performance de l'ordinateur. L'expression que nous utilisons est donc la suivante :

$$v_{t+1} = (1 - f)^{\frac{\delta t}{m}} v_t$$

Cette expression (dont la précédente est un développement limité en $\frac{\delta t}{m}$) présente un double intérêt : Tout d'abord, on peut facilement vérifier que la valeur de la vitesse après un laps de temps donné ne varie pas en fonction de $\frac{\delta t}{m}$, et donc de la performance de l'ordinateur, mais également, la valeur de $1 - f$ est facilement interprétable puisqu'elle correspond au coefficient par lequel on multiplie la vitesse chaque seconde (tpf nous étant donné en secondes).

Les autres forces :
Elles sont appliquées de la manière la plus simple possible :

$$v_{t+1} = v_t + \frac{\delta t}{m} F$$

Finalement, à chaque trame, la mise à jour de la position d'une entité se déroule donc en 3 étapes :

- Mise à jour de la vitesse de l'entité par application des forces de friction
- Mise à jour de la vitesse par application des forces appliquées au vaisseau
- Mise à jour de la position à partir de la nouvelle vitesse de l'entité

Principe Fondamental de la Dynamique appliqué aux moments :
On utilise pour cela très largement les quaternions, qui ont en 3D certaines des propriétés très intéressantes qu'ont les complexes en 2D. Définis par 4 composantes desquelles on peut déduire un angle et une direction, une simple multiplication (non commutative) entre eux permet de combiner plusieurs rotations successives autour de différents axes. Tous les quaternions utilisés sont normalisés (équivalents aux $e^{i\theta}$ complexes).

Hormis cela, les calculs sont très similaires à ce qui a été vu précédemment :

- Les couples de friction : L'angle du quaternion *vitesse angulaire* sont multiplié par $(1-f)^{\frac{\delta t}{m}}$
- Les autres couples : Le quaternion *vitesse angulaire* est multiplié par le quaternion représentant le couple, et dont l'angle est multiplié par $\frac{\delta t}{m}$
- Enfin, l'ensemble des rotations appliquées à l'entité est contenu dans un quaternion *rotation*, multiplié à chaque trame par *vitesse angulaire* (dont l'angle a, encore une fois, été multiplié par $\frac{\delta t}{m}$.

Commentaires sur l'application de ces équations aux vaisseaux :

A partir de l'accélération, on met à jour la position du vaisseau : cela nécessite une double intégration, et donc l'accumulation d'erreurs d'intégration qui peuvent être importantes après un certain temps de jeu, malgré l'utilisation de variables de type *double*. Il est paru en particulier très vite évident qu'il n'était pas possible d'utiliser le vecteur vitesse du vaisseau pour calculer sa nouvelle position d'un côté, et translater le modèle 3D qui lui est associé de l'autre. Il est nécessaire :

- de calculer la position du vaisseau puis de déplacer le modèle 3D à la position tout juste calculée ou alors

- de déplacer le modèle 3D selon un vecteur égal à la vitesse du vaisseau, puis d'affecter comme position du vaisseau un vecteur égal à la translation du modèle 3D

La seconde solution a été choisie, pour des questions de simplicité d'écriture du programme.

Une caméra suivant le vaisseau a été par la suite implémentée. Malgré ces précautions, on observait un décalage progressif entre l'orientation de la caméra et celle du vaisseau, alors que la caméra, pour se positionner correctement, avait accès à la position et à l'orientation du vaisseau.

Le quaternion rotation (celui donc, qui permet d'obtenir à tout moment, la position des 3 axes principaux du vaisseau), qui est sensé être en permanence normalisé, ne l'était plus au bout d'un certain temps, ce qui aboutissait à un mauvais calcul des axes principaux du vaisseau.

Il est donc nécessaire pour corriger les erreurs d'orientation de la caméra d'effectuer systématiquement (ou du moins régulièrement) la normalisation des quaternions, en exécutant en particulier *angularSpeed.normalizeLocal()*, et surtout *rotation.normalizeLocal()*.

Enfin, les angles contenus dans les quaternions sont identiques modulo 2 pi. L'accélération angulaire maximale que l'on peut obtenir lorsque la masse du vaisseau est égale à 1 est donc limitée à $\pi rad.s^{-2}$, de la même manière que la vitesse angulaire maximale est limitée à $\pi rad.s^{-1}$.

Une solution envisagée serait par exemple de diviser l'angle du quaternion par un facteur n s'il est supérieur à π afin de le ramener à une valeur comprise entre 0 et π (entre 0 et $-\pi$ si l'angle est négatif], puis de multiplier la valeur du temps entre chaque trame par ce facteur n.

4.1.3 Une intelligence artificielle basique pour des vaisseaux autonomes

De manière à pouvoir tester les mécaniques de simulation physique de base, de remplir la scène, voire de tester certaines fonctionnalités comme l'ajout de vaisseaux fonctionnels, il nous a semblé intéressant d'implémenter une intelligence artificielle basique pour des vaisseaux qui fonctionneraient de manière autonomes. Cette intelligence artificielle, codée en dur, exécute à chaque trame les actions suivantes :

- Lorsque le vaisseau est situé au-dessus d'une grille que l'on fait apparaître dans le jeu et à une altitude comprise entre certaines bornes arbitraires, le vaisseau se dirige de manière aléatoire : Un nombre aléatoire est choisi entre 0 et 9 (nombre de momuvements que peux choisir d'exécuter l'IA), et selon le nombre choisi, l'action est exécutée
- En dehors de la grille, le vaisseau tourne selon 1 de ses 3 axes principaux : l'axes le plus pertinent pour se diriger vers le centre de la grille est calculé, on détermine ensuite le sens de rotation, puis l'action est exécutée

4.1.4 Gestion de l'énergie

Architecture du système énergétique

Un aspect important dans ce jeu est le système de combat qui doit être assez détaillé pour inclure un système de distribution d'énergie. Cette distribution sera assurée par la personne au poste d'ingénieur (ou de l'IA s'il n'y a personne qui occupe ce poste).

Tout d'abord, parlons des différentes classes de sous systèmes qu'un vaisseau peut avoir. Nous avons les sous systèmes qui ne sont pas alimentés et ceux qui sont alimentés (c'est à dire que pour que ce sous système fonctionne, il faut qu'il ait un apport en énergie).

Voici une liste de sous-systèmes qui ont été codé, même s'ils ne sont pas opérationnel :

- Sous-systèmes non alimentés :
  - Générateur (fonctionnel)
  - Batterie (fonctionnel)
- Sous-systèmes alimentés :
  - Armure énergétique
  - Moteurs sub-luminique (fonctionnel)
  - Lanceur plasma (quasi fonctionnel)

La distribution de l'énergie

La distribution se fait en deux étapes :

1. Charge de la batterie du vaisseau
   Le générateur du vaisseau va générer une quantité d'unité d'énergie par seconde qui va dépendre de son état, et cette énergie sera envoyée directement dans la batterie du vaisseau.
2. Distribution de l'énergie à travers le vaisseau
   La personne qui est l'ingénieur du vaisseau va distribuer l'énergie dans l'ensemble du vaisseau, choisissant l'énergie allouée à chaque sous-système.

On peut décider de distribuer plus d'énergie que ce que le générateur fournit, dans ce cas on utilise la batterie du vaisseau comme "tampon". Cela peut permettre de suralimenter certains sous-systèmes pour augmenter leur efficacité et avoir l'avantage stratégique dans certaines situations particulières.

4.2 Interface Développeur

La partie interface est réalisée à l'aide de la librairie Nifty GUI, qui permet de réaliser des post-process recouvrant l'écran. L'implémentation de ce module étant orthogonal au reste du code de logique de jeu, il a été possible de s'entraîner à la réalisation de menus sans impacter le reste du développement. En conséquence, l'AppState de l'affichage du GUI est séparé du reste.

Le principe est assez classique : un fichier xml contient les informations relatives à la structure et aux fonctions utilisées par l'interface graphique voulue, comme pourrait l'être un fichier html sur une page web. Au lieu de balises html, on utilise des balises propres à Nifty. Il est à noter que des styles et effets

de base, implémentés dans Nifty GUI, peuvent être directement utilisés.
Ce fichier xml est alors relié à un ou des contrôleurs java, qui implémentent les fonctionnalités voulues. et permettent l'interactivité avec l'utilisateur.

4.2.1 Menu basique

La première étape a été de réaliser un menu basique, permettant de naviguer entre les écrans, et de réaliser un certain nombre d'opérations simples.

Les différentes balises permettent d'arranger notre interface graphique. <screen> permet de définir plusieurs écrans entre lesquels ils est possible de naviguer. <layer> permet de définir plusieurs couches (fond d'écran, textures etc...) et <panel> permet d'arranger les éléments sur l'écran.

Ont été implémentés :

- Menu classique avec navigation entre différents écrans au moyen de boutons.
- Menu désactivable par pression de la touche F1
- Champs permettant la saisie et le traitement d'un texte ou d'un nombre.
- Sliders permettant de régler de manière visuelle la valeur d'un paramètre, avec affichage de la valeur pointée en temps réel.

4.2.2 Intégration avec la logique de jeu

Notre GUI développeur doit avoir accès à toutes les briques du jeu pour pouvoir les utiliser ou non selon l'option choisie. Il s'agit donc de raccorder le menu aux fonctionnalités du jeu et d'y faire appel à volonté.

Le contrôleur java du GUI devient alors l'AppState général de notre jeu en développement. C'est lui qui, en plus d'assurer l'interaction avec l'utilisateur, va attacher/détacher les AppStates de logique de jeu. Pour cela, l'instance d'application en cours est attribut de la classe du contrôleur. On peut alors accéder à toutes les méthodes possibles de l'application (caméra, logique, physique, graphisme...)
Il a également accès aux méthodes des AppStates de jeu pour régler les paramètres (nombre de vaisseaux, énergie etc...).
3 modes ont été implémentés :

- Le mode nuée : on peut générer jusqu'à 700 vaisseaux qui suivent une proto IA : Une cible est disposée au centre. Chaque vaisseau s'éloigne de manière aléatoire, puis doit rejoindre la cible en temps minimale.
- Le mode contrôle : nous avons la vue en 1ere personne d'un vaisseau que nous contrôlons en orbite autour d'un astre.
- Un mode galerie qui montre les modèles 3D créés.

Il reste encore un souci de saturation de RAM à régler (même en détachant une appState, les données créées ne sont pas supprimées par le ramasse miettes de Java). Mais chaque mode fonctionne bien séparément.

4.3 Musique et effets de jeu

Nous avons utilisé Logic Pro X afin de faire des musique de jeu ainsi que des effets de jeu.
Logic Pro est un logiciel de musique assisté par ordinateur qui peut être utilisé pour toute sorte de projets audio. Ce logiciel est généralement utilisé par des professionnels de la musique dans des studios d'enregistrement.

Nous avons utilisé le plugin audio Alchemy pour la plupart des sons crées. Alchemy est un outil qui permet de manipuler de samples (son audio très court), mais possède aussi tout un panel de synthétiseur afin de pouvoir modeler le son à sa guise.

Nous avons créé deux Musiques :

- Une musique pour le menu principal
- Une musique pour les combats

— Une petite musique pour quand on se pose sur un vaisseau/station

Mais nous avons aussi créé différents effets :

— Tir de lanceur plasma
— Tir de laser lourd
— Plusieurs impacts
— Le son en entrée et sortie d'hyperespace

Après la création de la musique de jeu, les fichiers audio peuvent être importés dans le jeu afin d'être utilisé pour faire les sons du jeu.

Pour faire du son dans le jeu, nous utilisons OpenAL, c'est une librairie qui permet de faire des sons localisés ainsi que directionnels. C'est donc ce qu'il faut dans le jeu vidéo pour pouvoir distinguer un son lointain d'un son proche.

Tous les sons du jeu qui ont été faits jusqu'a présent sont sur le drive suivant :
`https://drive.google.com/open?id=0B6npgIjioUnpYXg3ZWd0RGh3Tm8`

4.4 Modèles 3D

Les modèles 3D des vaisseaux et stations qui forment l'ensemble des éntités du jeu ont été réalisées à l'aide du moteur 3D de modélisation et d'animation Blender. Il s'agit d'un logiciel open source et se présente sous une interface basé sur OpenGL. Ses fonctionnalités sont très étendues et permettent la modélisation de structures 3D à partir de nombreus outils de générations d'objets géométriques modelables, ainsi que le choix de la texture ou encore l'éclairage de la scène. Bien qu'il soit aussi possible, au sein même de blender, de réaliser des animations mettant en scène nos modèles, et même de leur attribuer des propriétés physiques basiques (solide, liquide, etc...),cette fonctionnalité d'autant plus interessante dans le cadre de notre projet n'a pas été utilisée ici pour des raisons de praticité à implémenter les modèles blender dans le jeu.
Les différentes structures sont séparés en deux factions adverses comportant chacune une station et deux vaisseaux. Dans l'idée de garder une certaine logique mais aussi bien de créer une identité ditincte à chaque camp, les vaisseaux et station d'une même faction ont été modélisés dans des styles similaires, avec des éléments constitutif communs ou similaires. Toujours dans cette optique, nous avons attribué un nom à chacune.

4.4.1 Modèles de base

4.4.2 Première faction : Tantra Dawn

Cette faction prototype a été réalisée dans un style classique inspiré de l'aspect général des vaisseaux spatiaux militaires dans la littérature de science-fiction.

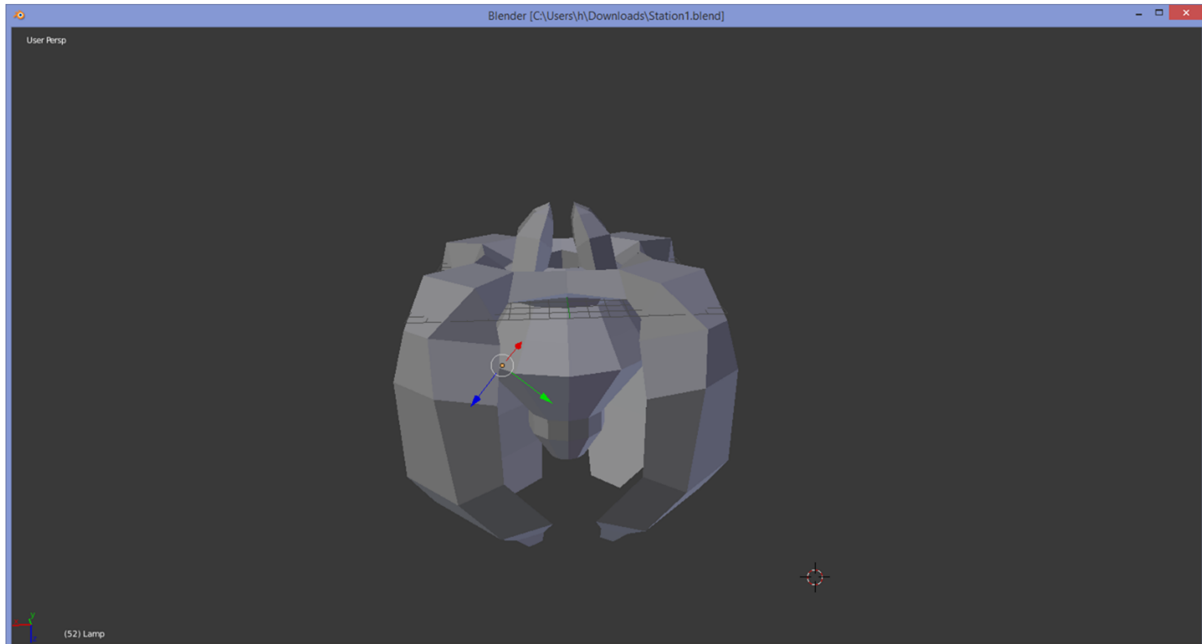

FIGURE 1 – Station

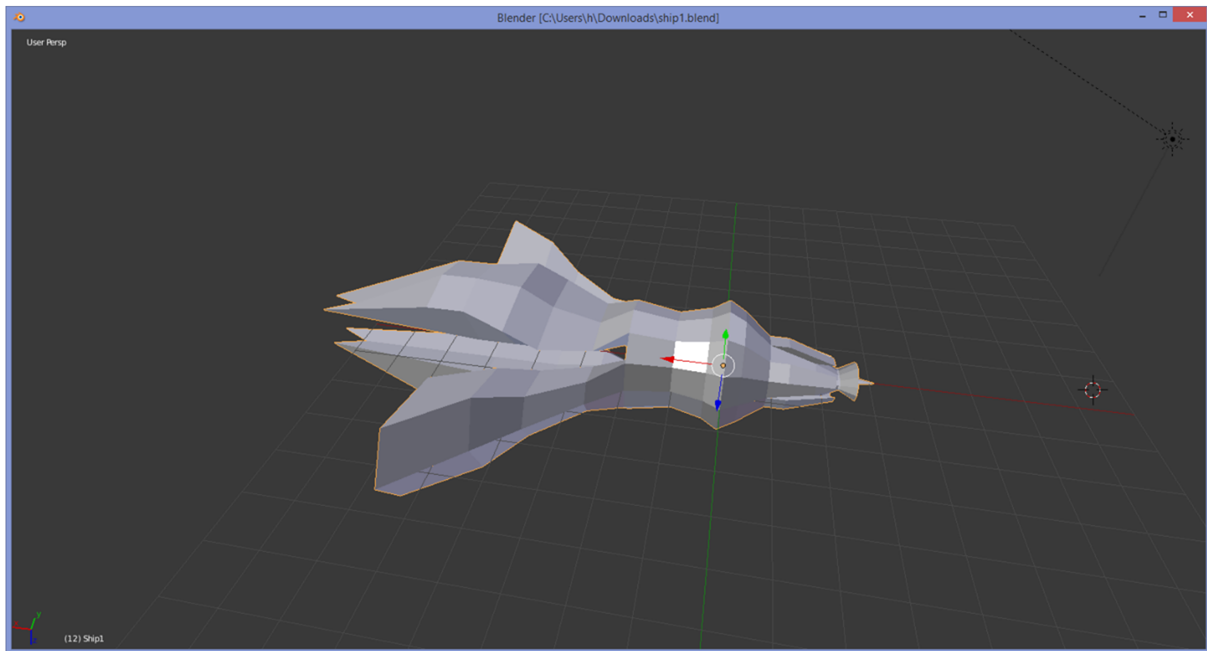

FIGURE 2 – Premier vaisseau

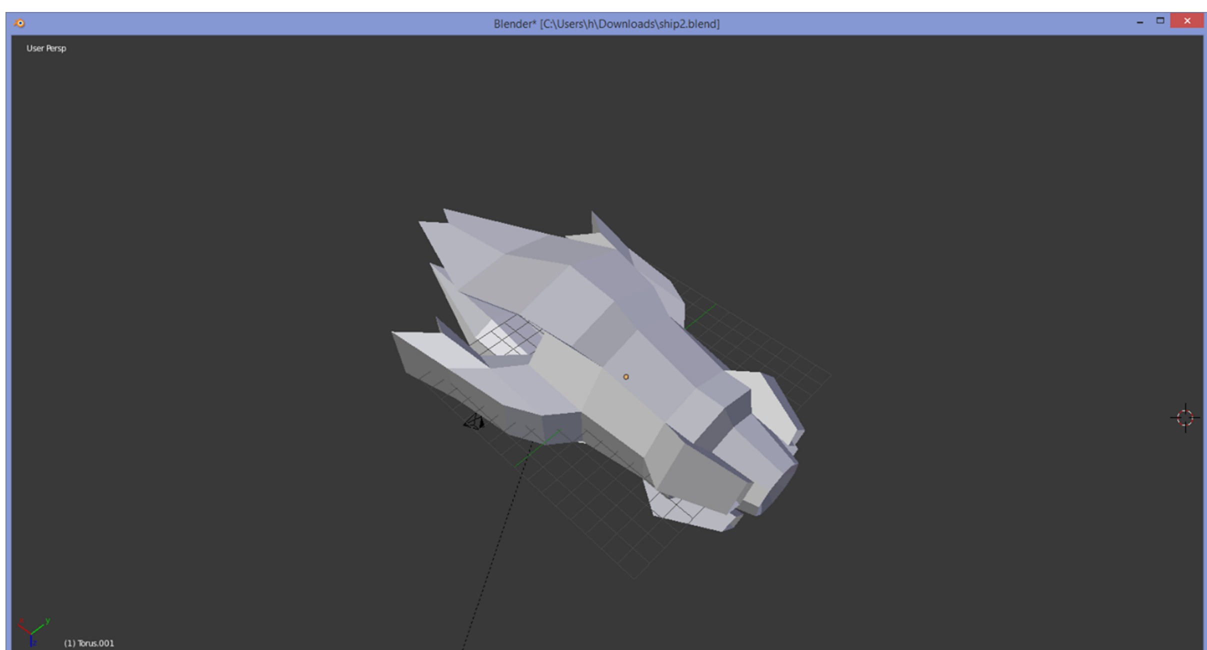

FIGURE 3 – Second vaisseau

4.4.3 Seconde faction : Rock Nebulace

Une faction imaginée dans un style rocailleux et rotatif.

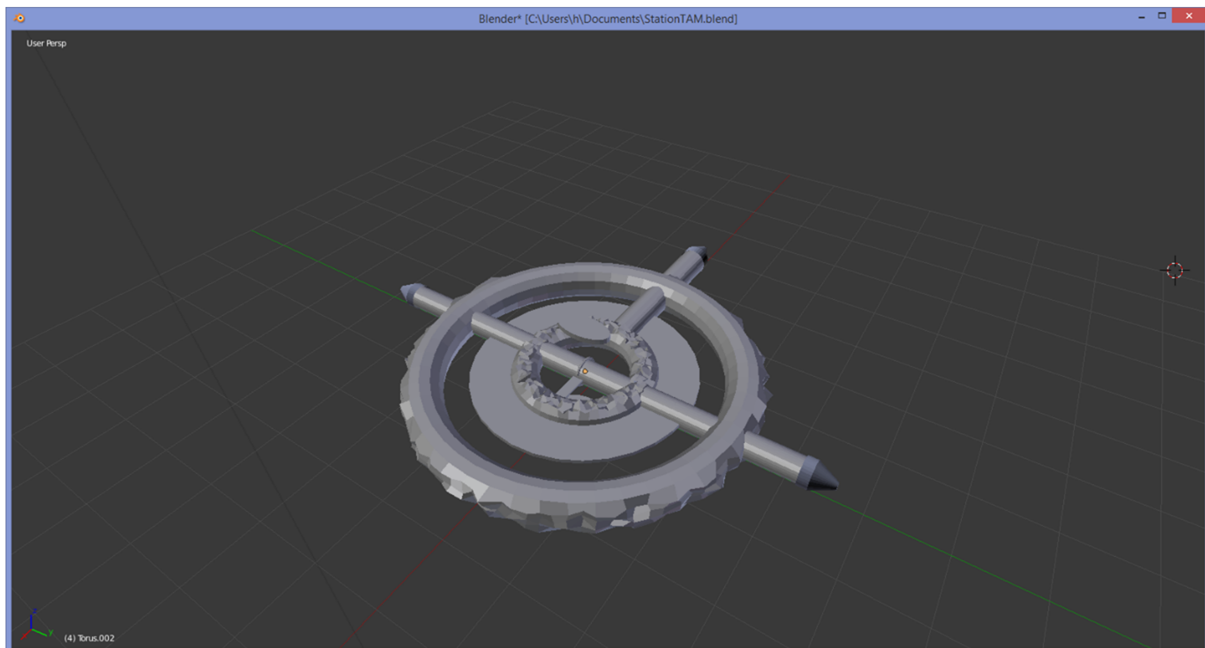

FIGURE 4 – Station

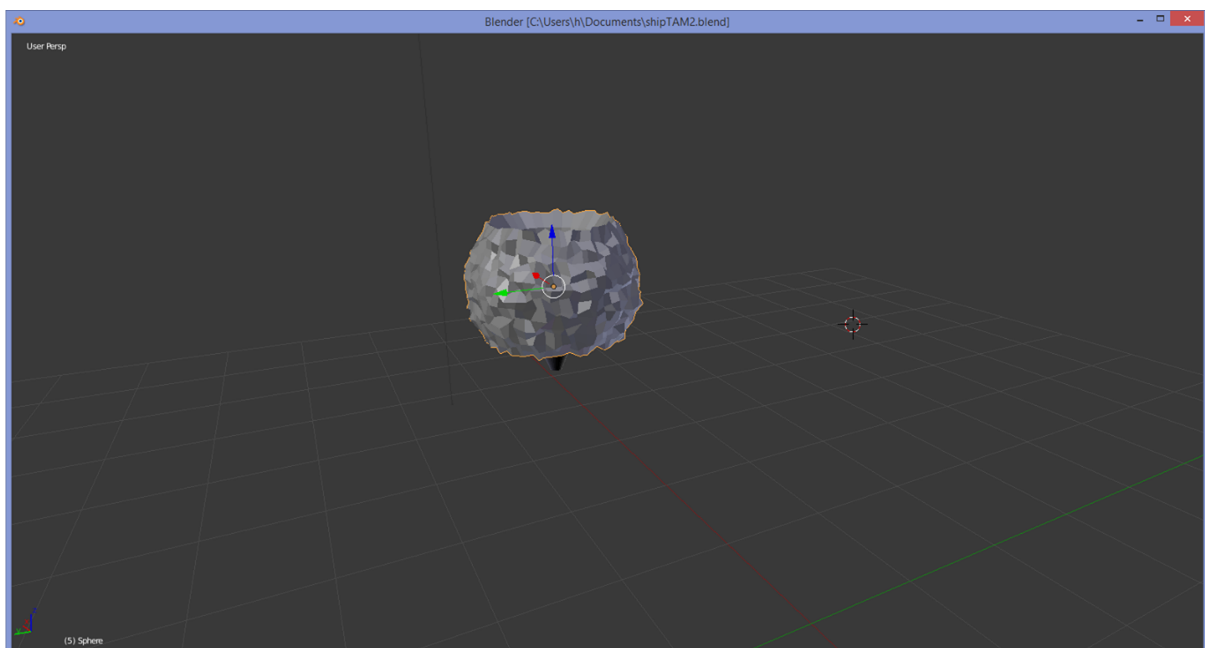

FIGURE 5 – Premier vaisseau

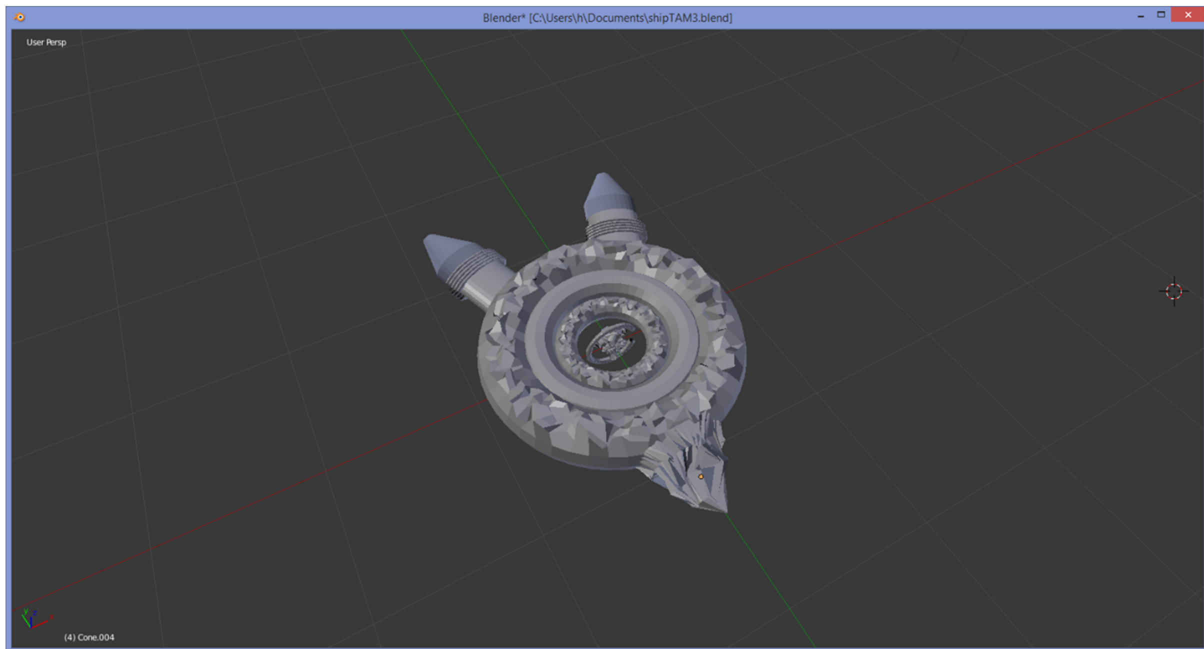

Figure 6 – Second vaisseau

4.4.4 Missile

Il s'agit de l'élément projectile utilisés par tous les vaisseaux pour les affrontements.

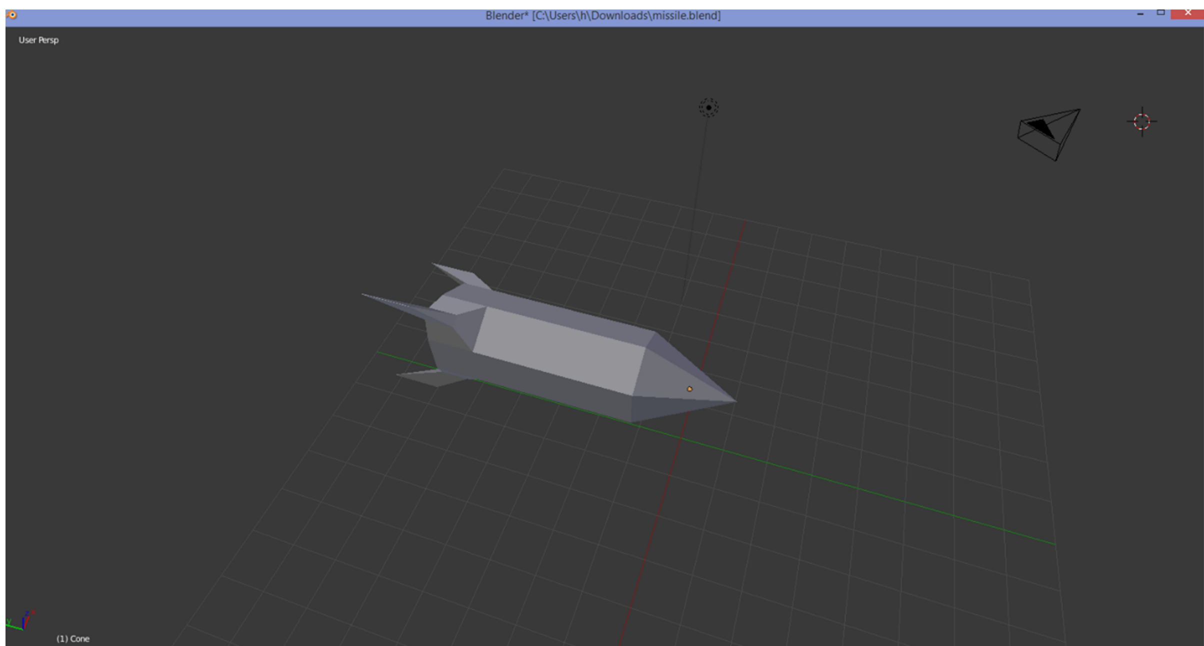

Figure 7 – Missile

5 Bibliographie

— Jmonkey engine 3.0 cookbook

— Jmonkey engine 3.0 Beginners guide

— https ://openclassrooms.com/courses/debutez-dans-la-3d-avec-blender

6 Annexe

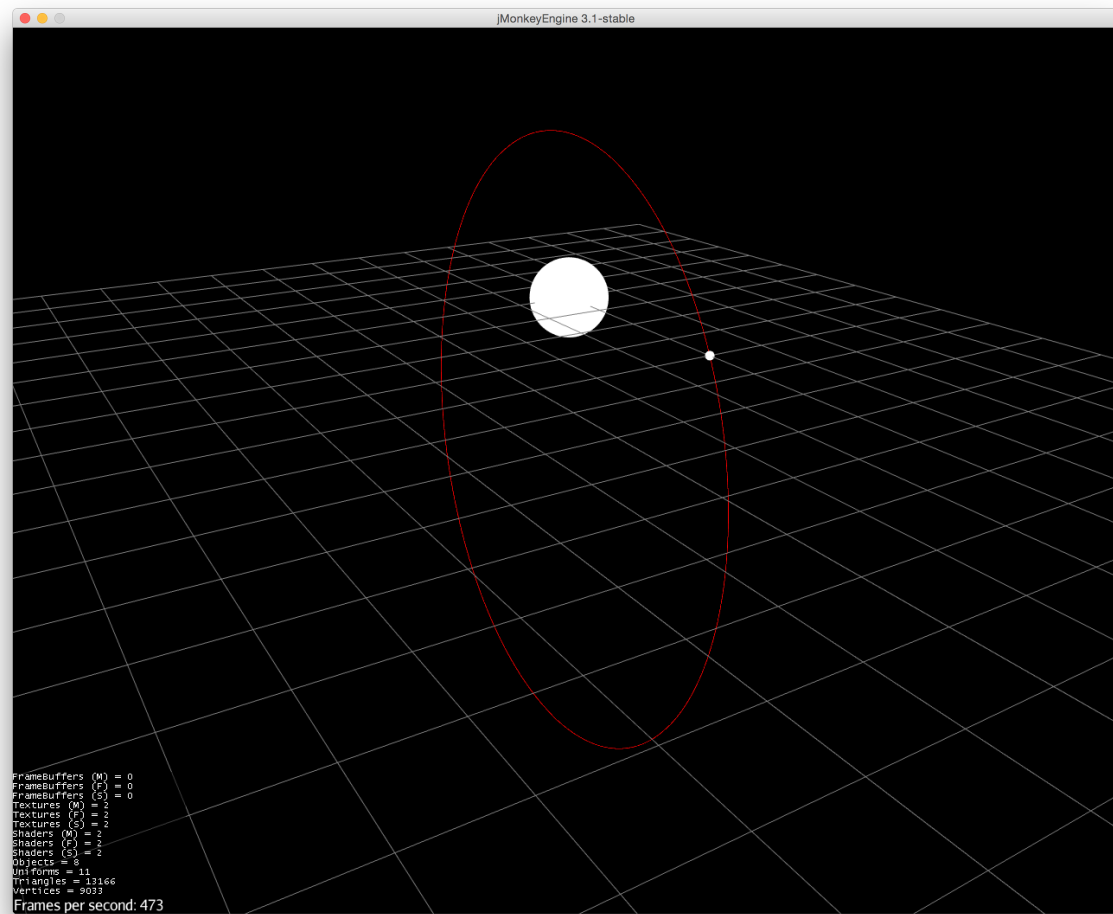

FIGURE 8 – Orbite elliptique

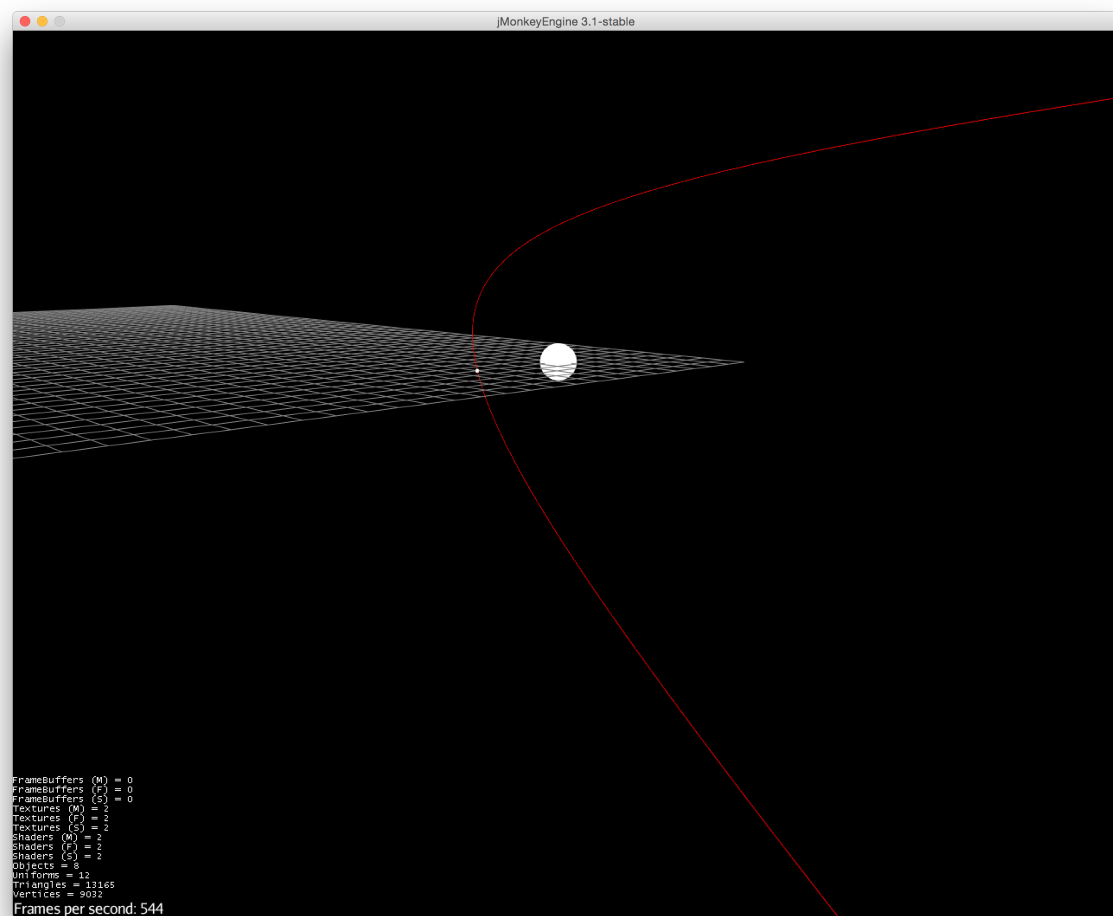

Figure 9 – Orbite Hyperbolique

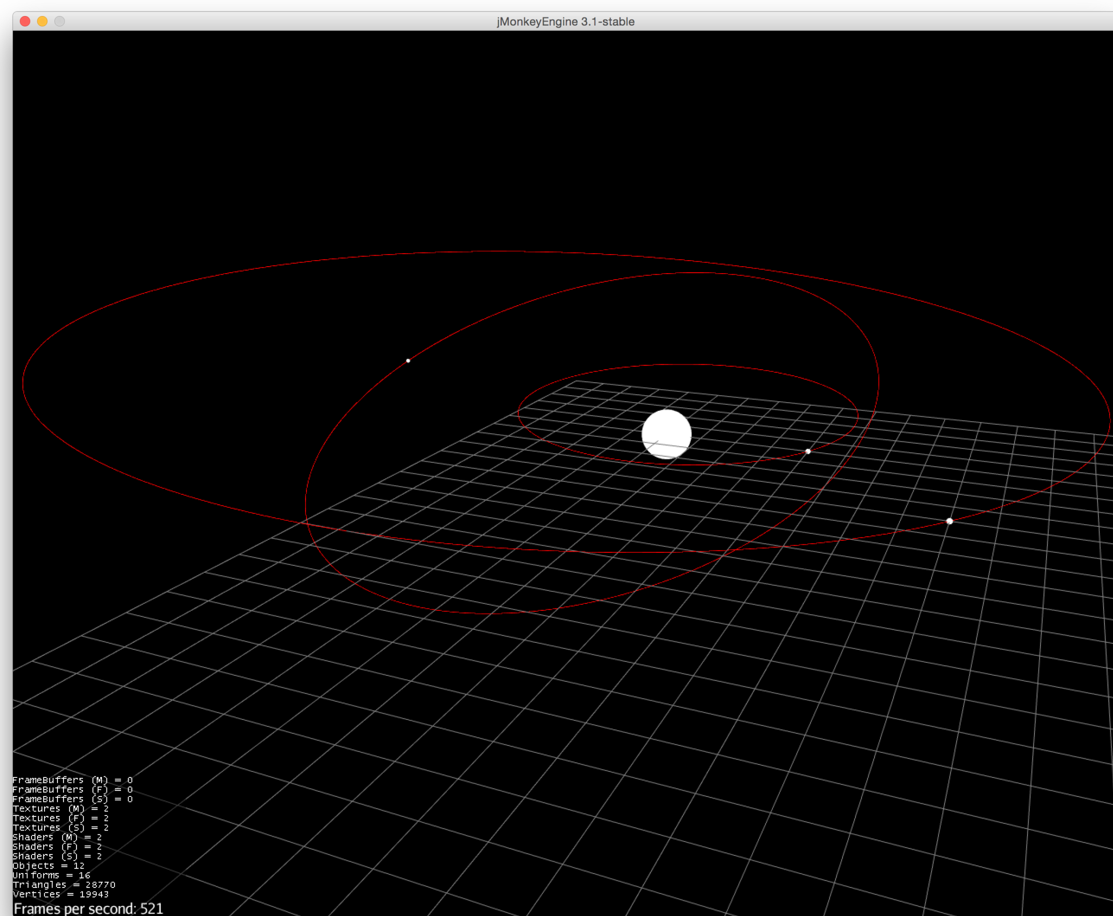

FIGURE 10 – Exemple de système solaire

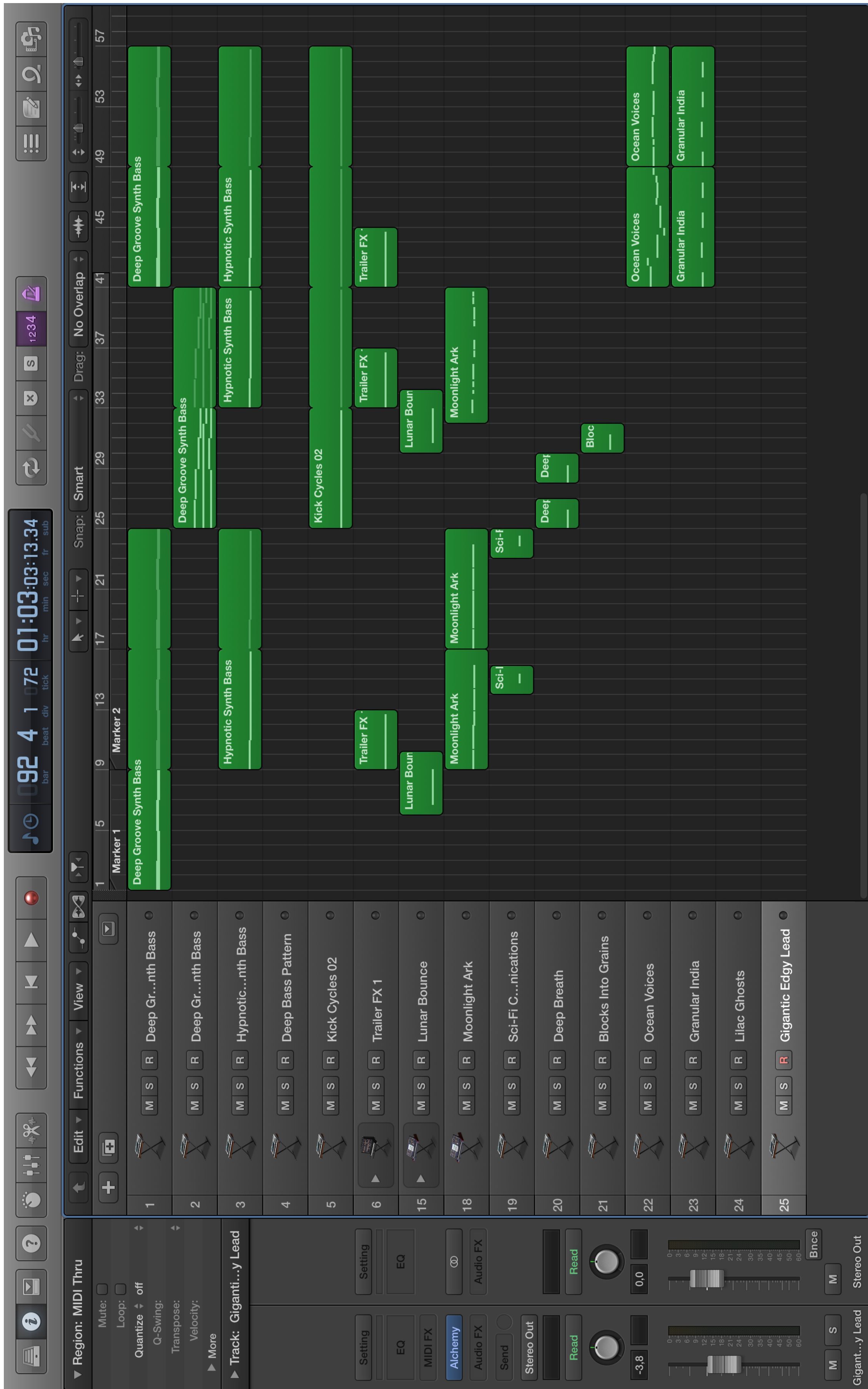

FIGURE 11 – Logic pro