Projet long de Technologie Objet

Rapport général

Itération n°3

Groupe IJ-1

Fainsin Laurent
Guillemin Johan
Guillotin Damien
Heurtebise Tom
Jourdan Pierre-eliot

Département Sciences du Numérique
Première année
2020 — 2021

Table des matières

1 Rappel du projet

Sagittarius est un jeu type arcade tour par tour pouvant être joué de 2 à 5 joueurs. Le but d'un joueur est d'éliminer ses adversaires grâce à un arc, des flèches et, la mécanique principale du jeu, la gravité !
Ce projet a pour but de recréer ce jeu existant tout en l'améliorant par divers ajouts/fonctionnalités. Nous n'en revendiquons pas la propriété intellectuelle et nous n'avons pas consulté le code source pour élaborer notre projet !

2 Principales fonctionnalités

La fonctionnalité principale de l'application est de pouvoir jouer à une ou plusieurs parties. Le joueur doit donc pouvoir tirer, ce qui prend en compte l'angle de tire ainsi que la puissance du tire, et il doit pouvoir se déplacer.
Il est également important que le joueur puisse régler les paramètres qui vont directement impacter sa partie tel que le nombre de joueurs et donc le nombre de planètes. Cependant, nous n'avons pas eu le temps d'implanter de mode solo ni de mode équipe. Chaque joueur est désigné par une couleur, cependant il n'a pas moyen de la choisir par lui-même, sa couleur lui est assignée aléatoirement.

Pour la génération de l'environnement (ce qui comprend en compte la position, la taille et la masse des planètes), l'utilisateur ne peut pas changer les paramètres avancés mais elle reste tout de même différente à chaque partie. Au niveau des paramètres audiovisuel, l'utilisateur peut par exemple retirer les sons et musiques. De plus, il peut changer les touches de contrôle.

Toutes ces fonctionnalités sont rangées dans des menus tels que le menu settings situé dans le menu principal ou dans le menu pause. A tout moment, il est tout à fait possible de quitter le jeu, quitter une partie ou en recommencer une.

3 Gestion Agile du projet avec Vision de l'application

Durant les séances dédiées à l'application des méthodes de projet Agile nous avons pu découper notre application en fonctionnalités elle même redécoupées et ainsi de suite jusqu'à obtenir des "Users stories". Sur la page suivante vous pourrez retrouver le découpage que nous avons effectué.

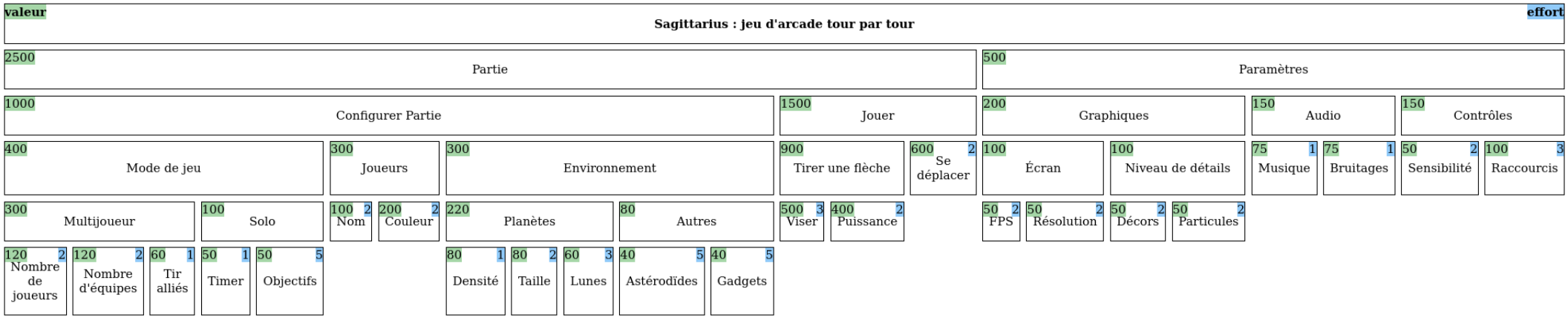

FIGURE 1 – Découpage de la vision du projet jusqu'au Users Stories (voir Méthodes Agiles pour le détail des termes)

4 Découpage de l'application en sous-systèmes (en paquetages)

Le jeu est actuellement divisé en deux paquets, un paquet pour toutes les classes du modèle physique du jeu, et un deuxième paquet regroupant les écrans du jeu.
Nous avons utilisé le sous-paquet scene2D de libGDX, mais celui-ci ne respecte pas le template MVC. S'obstiner à appliquer le MVC impliquait de contourner les systèmes mis en place par scene2D ou bien à réécrire la totalité de libraire nous même, nous avons donc choisi de ne pas appliquer le MVC. À noter tout de même que le modèle MVC est peu adapté à des jeux plus dynamiques d'un puissance 4 ou un tic-tac-toe, comme le nôtre. Nous avons tout de même essayé de créer une version compatible avec swing mais, les deux moteurs graphiques étant incompatibles, lors de l'ajout d'une nouvelle fonctionnalité au jeu, résoudre les problèmes de compatibilité était trop chronophage. De même la granularité des contrôles des entrées que proposait Swing n'était pas comparable avec celle de libGDX, surtout pour un jeu en temps réel comme le nôtre. Après quelques recherches il semble que le patron de conception le plus couramment utilisé pour des jeux en temps réel soit l'ECS. Scene2D ne propose pas un système de composant pour les entitées, mais seulement des entitées (appelées ici Actors).

5 Diagramme de classe et architecture de l'application

Dans cette section nous allons vous présenter l'architecture de l'application sous la forme de deux diagrammes UML. Le premier présente l'architecture du modèle physique et le deuxième l'architecture de la vue du jeu.
Voici d'abord le moteur physique du jeu :

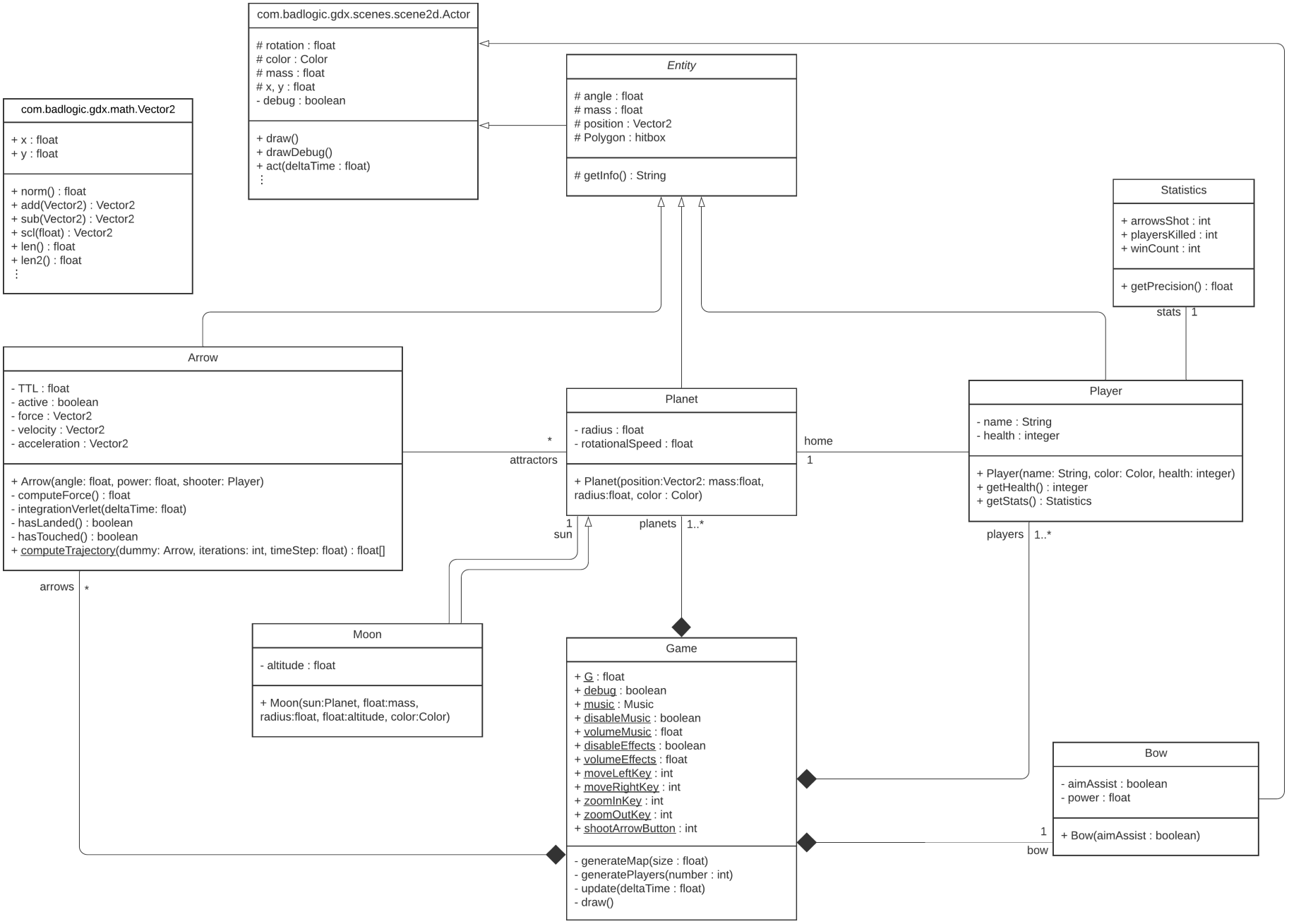

FIGURE 2 – Diagramme UML simplifié du moteur physique du jeu

La classe initiale de tout le projet est SagittariusGame.java. On le constate aisément sur l'UML puisqu'elle agrège d'autres classes. Du point de vue du code, c'est elle qui initie tout le jeu en construisant l'écran de démarrage situé dans la vue (StartScreen.java). Cet écran fait lui-même appel à la construction de l'écran d'une partie (cf classe GameScreen.java) qui va lancer tous les constructeurs du modèle physique. Ainsi c'est dans la classe GameScreen que toutes les "entités" sont générées et que l'on crée donc les joueurs, les planètes, l'arc et les flèches. Il s'agit vraisemblablement de la classe centrale du projet puisque c'est elle qui s'occupe de la gestion et de la destruction de tous ces objets (un joueur tué est supprimé de la partie par GameScreen).
A notre sens la classe centrale du modèle physique est néanmoins Arrow.java puisque c'est cette classe qui est la plus technique avec le calcul de trajectoire par exemple.

A présent, abordons le diagramme UML de la vue. Sur celui-ci vous retrouverez de nombreux éléments issus de libGDX puisque nous nous sommes largement appuyés sur le framework proposé par cette librairie. Ainsi comme vous pourrez le noter sur le diagramme la vue proposée par libgdx est organisée sous forme de "scènes" (cad des écrans) sur lesquelles figurent des acteurs. Les classes que nous avons développées pour la vue sont celles dont le suffixe est *Screen.java. Dans chacune de ces classes vous trouverez notamment les méthodes initiales et update qui se chargent respectivement de la création des écrans et de leur rafraîchissement perpétuels (avec la prise en compte des évènements comme la mort d'un joueur par exemple). Le code important se trouve dans les méthodes update pour les écrans et pour les acteurs ils se trouvent dans les méthodes act.

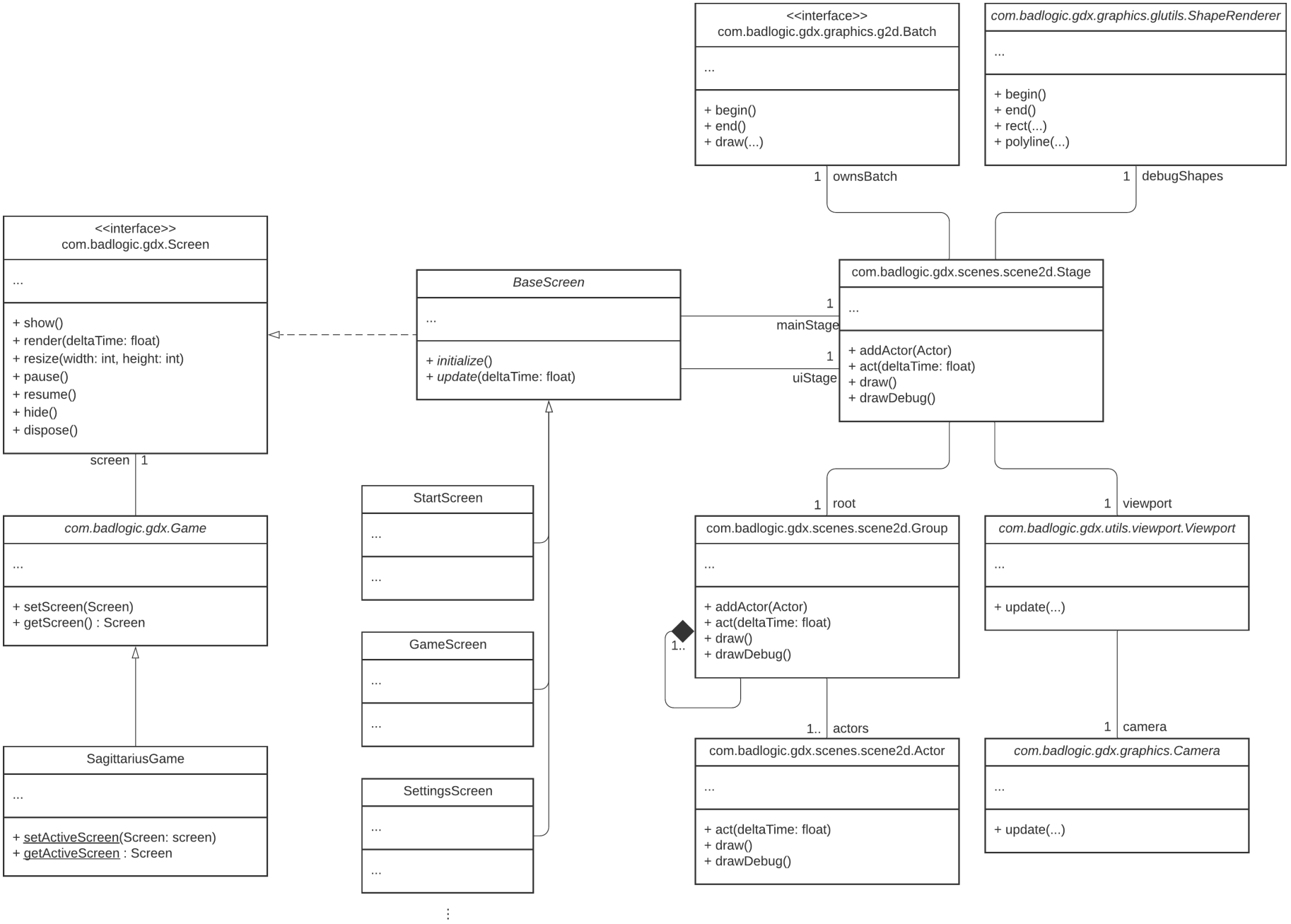

FIGURE 3 – Diagramme UML simplifié de la vue du jeu

6 Choix de conception et réalisation

Un choix de conception important que nous avons dû réaliser et qui ne respecte pas bien le principe du MVC est le fait que nous ayons utilisé de nombreuses méthodes de LIBGDX dans la plupart de nos classes. Nous avons déjà abordé ce point dans les sections et dans les rapports précédents.

Pour une classe donnée, par exemple Player.java, l'ensemble des éléments en rapport avec lui sont contenus dans celle-ci (en partie à cause de la manière dont est écrit scene2D). Ainsi la classe Player.java gère son modèle physique, sa vue, le lancement d'effets sonores, la gestion des contrôles en rapport avec son déplacement, sa destruction... Ce patron de conception va à l'encontre du MVC, mais de cette manière toutes les entités sont bien séparées.

7 Organisation de l'équipe

Nous avons choisi d'utiliser le git de l'enseeiht (git.inpt.fr) pour notre gestionnaire de version.

Laurent s'est occupé de créer une base avec libGDX pour que nous puissions tous commencer à programmer dessus. Il s'est aussi globalement occupé du modèle physique, du tracking de la caméra et du menu permettant de gérer les contrôles.
Damien s'est occupé de l'affichage des textures, de la génération de la carte, de la gestion des joueurs (système de tours)...
Pierre-Eliot s'est chargé avec Damien de la réalisation des textures du jeu, mais aussi du modèle physique avec Laurent (classes Entity, Player, Planet...).
Tom s'est principalement occupé de la création des menus et du système permettant de naviguer entre les menus. De même, il s'est occupé de la gestion de la musique et des effets sonores.

Globalement, nous avons toujours respecté ce que nous avions planifié à chaque itération.

8 Points à améliorer

Par manque de temps, nous n'avons pas eu l'occasion d'implémenter certaines fonctionnalités. Ainsi il manque actuellement un système d'équipe ainsi qu'un mode de jeu solo. Certains menus pourraient être mieux désignés, mais ceux-ci remplissent tout de même leur rôle. Le code peut évidemment être optimisé à certains endroits, de même quelques classes/fonctions pourraient être factorisé/réifié.

9 Conclusion

Nous avons trouvé ce projet intéressant puisqu'il nous a permis de découvrir pour la première fois ce qu'était la collaboration sur un projet ambitieux. De plus, il nous a fallu nous organiser entre nous et apprendre à utiliser des outils tels que git qui devraient nous être utiles dans notre future vie professionnelle.