Nicolas Faucheux
SDSC

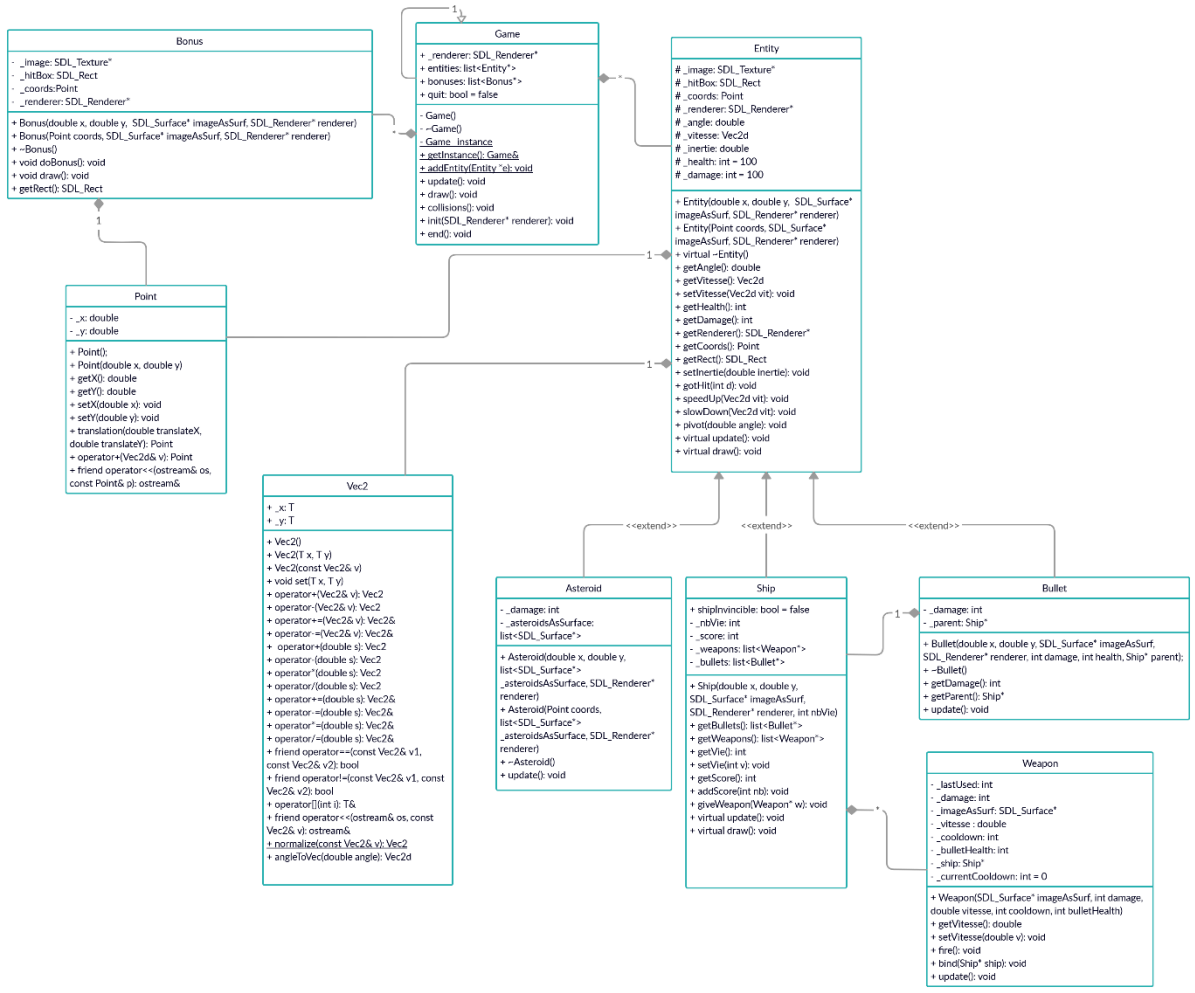
Mardi 05 Mai 2020

Projet Asteroids

- Rapport –

** Tout d'abord, je souhaite préciser que notre projet a le mérite d'avoir été entièrement réalisé par nous-même. Nous n'avons pas téléchargé une version d'Asteroids trouvable sur Internet.*

Diagramme de classe :



Les relations « use » n'ont pas été modélisées sur ce diagramme, pour ne pas le surcharger.

Répartition des tâches :

Une grande partie de ce projet a consisté à modéliser le jeu. Nous avons réalisé cette tâche ensemble. Nous avons beaucoup discuté pour nous mettre d'accord sur les meilleures ou ce que nous pensons les meilleures façons de modéliser telle ou telle chose.
Nous avons réalisé une première ébauche des différentes classes moins aboutie que l'UML présenté ci-dessus.
Nous avons par la suite ajouté des classes, toujours en se concertant, pour apporter des améliorations ou des fonctionnalités supplémentaires à notre jeu.

Concernant la répartition des tâches, nous ne nous sommes pas imposés de tâches. Nous choisissions une tâche et nous prévenions l'autre que nous travaillons dessus, pour ne pas faire la même tâche en même temps.
Dès que nous avions une idée ou qu'une tâche devait être réalisée, l'un de nous le faisait.

Nous avons travaillé avec GitLab pour le partage de code et Discord pour la communication.

Concernant le design de notre jeu. Nous avons utilisé des images pour nos objets car nous trouvions ça plus « joli » que des DrawLine. Le constructeur de nos objets prend un chemin vers une image donc il serait tout à fait possible de changer d'image ou encore d'avoir des astéroïdes avec des images différentes. Nous avons préféré laisser le design de côté pour nous concentrer sur la partie fonctionnelle de notre jeu.

Un choix d'implémentation

J'ai commencé par nous créer une base pour pouvoir commencer à travailler avec un bon environnement, j'ai modifié le code fourni ainsi que le CMake pour qu'il soit facile de commencer à réaliser ce projet.

J'ai ensuite commencé à faire le jeu de base : dessiner un vaisseau et essayé de le faire se déplacer, puis Ezriel a continué cette partie en ajoutant les rotations ainsi que des images pour nos entités.

Remarquant que nous avions souvent besoin de coordonnées x et y pour placer nos objets. J'ai réalisé une classe Point, qui s'est avérée être très utile. J'ai surchargé plusieurs opérateurs pour nous simplifier la programmation par la suite. J'ai aussi réalisé une classe template Vec2 en même temps, qui ressemble à la classe Point, mais qui représente un vecteur de deux objets, nous l'avons utilisé pour faire des calculs d'angles, des translations de points pour déplacer nos objets. Cette classe étant template, elle peut stocker toutes sortes d'objets.

Je me suis occupé de la partie Bonus, notre code étant réfléchi en amont, j'ai pu facilement implémenter les Bonus. Il m'a suffi d'ajouter une classe Bonus, que le Game Manager utilise pour créer et faire apparaître aléatoirement ces bonus sur la fenêtre.

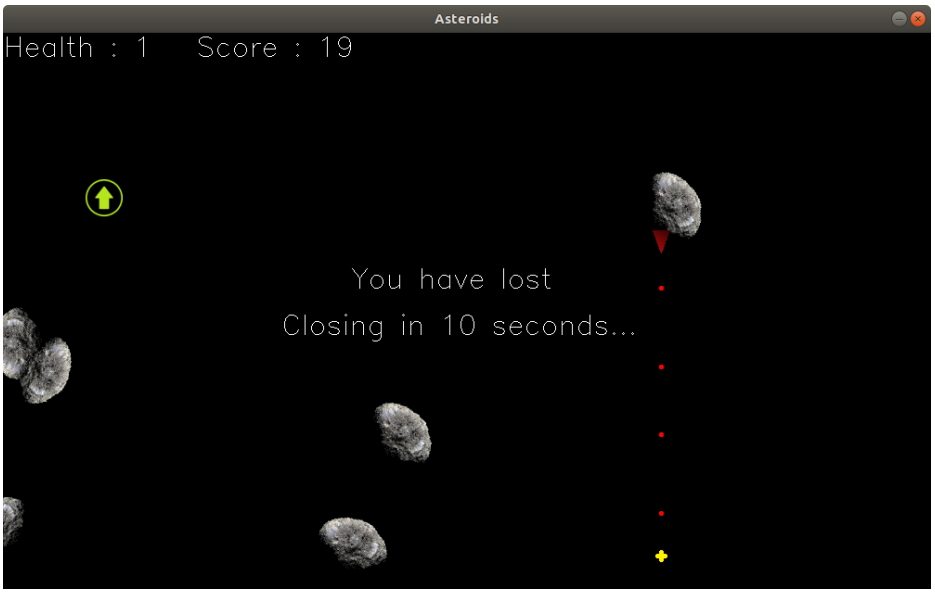
J'ai aussi ajouté des threads dans le code pour gérer le changement de certaines sans rendre le code bloquant. Par exemple, quand un vaisseau est touché par un astéroïde, il devient invincible pendant 2 secondes. Cette invincibilité est gérée par un thread pour que le jeu puisse continuer à s'exécuter.

Version classique :

Nous avons réalisé une version classique du jeu, c'est-à-dire que le jeu est fonctionnel et que nous avons rajouté le fait de pouvoir obtenir des bonus au cours de la partie. Nous en avons créé qu'un

pour montrer que notre implémentation permettait d'en créer facilement et pour nous garder du temps pour ajouter d'autres fonctionnalités.

Nous avons également ajouté une deuxième arme, car notre modélisation permet de le faire en quelques lignes.



En rouge une arme plus rapide mais moins puissante, en jaune, plus lente mais plus puissante.
La flèche verte est un bonus de vitesse d'attaque.

La version multi-joueur :

Cette version n'est pas totalement finie, elle a été réalisée en une trentaine de minutes. Le code est juste « dupliqué » pour un deuxième joueur. Il faudrait mieux organiser ça pour avoir moins de duplication. Mais nous avons préféré montrer qu'il est possible de réaliser une version multi-joueur avec notre implémentation. En ajoutant seulement un deuxième vaisseau, on voit que tout le reste du code suit et que tout fonctionne parfaitement (gestion des scores, des bonus, des points de vies).

La gestion des scores :

On peut voir sur les screens, qu'elle est fonctionnelle et qu'elle utilise notre propre font basée sur la Hershey Vector Font. Cette partie a été réalisée par Ezriel.

Multi-thread :

Notre jeu n'est pas vraiment multi-thread, mais il utilise tout de même des threads pour réaliser certaines opérations sans bloquer notre jeu.
Par exemple, quand un joueur est touché par un astéroïde, il devient invulnérable pendant 2 secondes. Au lieu de faire une attente active dans notre code ou que cette invulnérabilité dure plus longtemps à cause d'une boucle conditionnelle. Nous avons utilisé un thread qui va mettre à jour une variable au bout de 2 secondes. Par conséquent, le jeu tourne comme si rien ne s'était passé.

Modélisation :

Nous avons bien sûr utilisé l'héritage, nous avons une classe abstraite Entity qui représente toutes les entités qui se déplacent dans notre jeu (Ship, Asteroid, Bullet). Comme on peut s'en douter, ces classes ont des attributs et méthodes communes. Des méthodes pour se déplacer, des coordonnées dans l'espace, des points de vie, des dégâts, mais aussi tout simplement des méthodes pour être affichées à l'écran ou pour mettre à jour leur position.

Par conséquent, nous avons forcément du polymorphisme, certaines méthodes communes à ces 3 classes, sont implémentées légèrement différemment pour chacune d'elle. Elles ont chacune un comportement propre à l'objet. Par exemple, un astéroïde ne va que tout droit, il ne peut pas pivoter.

Nous avons utilisé beaucoup les list de la bibliothèque standard pour stocker nos objets et donc les itérateurs pour les parcourir.

Les listes semblaient très optimisées pour nos utilisations. Insertion en première position, par exemple.

Comme dit plus haut, nous avons réalisé une classe template Vec2, qui pouvait contenir deux éléments de n'importe quel type. Dans notre cas, le Vec2 de float a été très utile pour les vitesses et les calculs de coordonnées par conséquent.

Concernant les design-pattern, la grille de notation a été annoncé assez tardivement. Nous aurions voulu utiliser le design-pattern MVC, mais nous aurions perdu trop temps à restructurer tout notre code.
Néanmoins, nous avons utilisé le design-pattern Singleton pour la classe Game qui doit s'occuper de gérer la partie (un manager de partie). En effet, il n'existe qu'une seule instance de la classe Game par exécution. Le design-pattern Singleton permet d'en être sûr.

Nous avons utilisé les conventions de codage de Java car il s'agit de celle que nous connaissons le mieux, en utilisant donc le Camel Case, une variable commence par une minuscule, une classe par majuscule…

Nous avons commenté tous les fichiers d'en-tête (.h) en utilisant le formalisme doxygen.

Bien entendu, pour se partager le code nous avons utilisé Git :
https://git.unistra.fr/nfaucheux/m1_asteroids