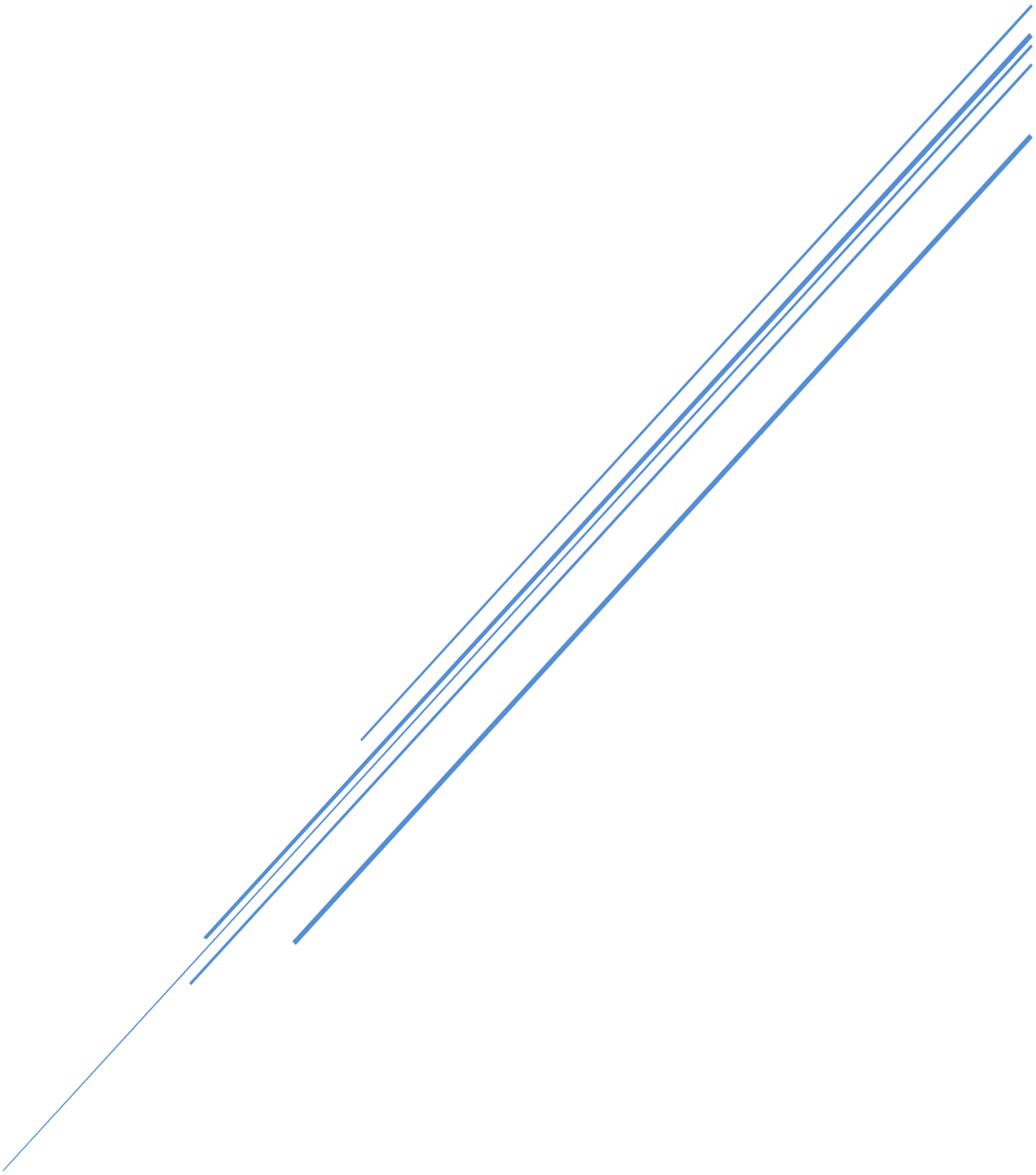
PROJET C++

Couvreur Alexis



Master Informatique IFI-RIF-MBDS 1ère année
Université de Nice-Sophia Antipolis

Introduction

Ce projet consiste à créer un jeu de type tower defense. On pourra sur un damier, placer des vaisseaux qui devront défendre le côté gauche contre des astéroïdes qui proviennent de l'extrémité droite. Ce projet va notamment mettre en évidence les concepts d'héritage, définition de fonctions virtuelles, purement virtuelles, création et destruction dynamique d'objets.

Utiliser le programme

Lancer le jeu

Ouvrir la Solution Visual studio fournie et lancer en mode **Debug**, les options d'includes / lib, etc. sont configurés et fonctionnent et la lib glut32.dll est dans le dossier **Debug**.

Jouer

Lorsque le jeu est lancé on peut alors placer les vaisseaux, et changer de type de vaisseau sélectionné en cliquant sur les cases. On peut alors lancer la vague en cliquant sur **Start !** dans le menu gauche.

Lorsqu'un astéroïde atteint le côté gauche, on perd des points de vie, et lorsque notre vie atteint zéro la partie est finie. Un astéroïde peut donc être détruit par les vaisseaux et gagnent de l'argent lors de la destruction de ceux-ci permettant d'acheter plus de vaisseaux pour résister de mieux en mieux au fil des vagues.

Les vaisseaux ne tirent pas en ligne droite, ils tirent sur l'astéroïde le plus, car les astéroïdes ne sont pas restreints au centre d'une ligne mais peuvent apparaitre à n'importe quelle hauteur.

Il existe trois types de vaisseaux :

- le Hispania : 15 de dégâts et une portée moyenne ;
- le Elysium : 30 de dégâts et une portée moyenne ;
- le Nostromo : 60 de dégâts et une portée longue.

Il existe trois types d'astéroïdes :

- le Eros : 100 points de vie, vitesse rapide et de valeur 25$;
- le Blume : 200 points de vie, vitesse moyenne et de valeur 100$;
- le Zephyr : 1000 points de vie et 500 de bouclier (enfin c'est un astéroïde donc plutôt une surface recouvrant l'astéroïde qu'il faut détruire avant), un vitesse lente et de valeur 500$.

Chaque niveau comporte un nombre d'astéroïde, et il n'existe pas de limite de niveaux, les astéroïdes apparaissent plus vite après le niveau 5, et encore plus vite après le niveau 10. Le jeu est un peu dur au début, et il faut bien placer ses vaisseaux sinon cela risque de poser problème.

Structure et héritage

Interface **ITickable**

La classe **ITickable** ne possède aucun membre et seulement une seule méthode **Tick** sans arguments.

Définie dans le header en tant que méthode purement virtuelle (mot clé **virtual** et valeur à 0), elle rend la classe purement virtuelle et par définition est une interface, elle va pouvoir définir donc un comportement pour tous les objets du modèle qui pourront être identifiés tels des **ITickable**.

Interface **IDrawable**

La classe **IDrawable** ne possède aucun membre et seulement une seule méthode **Draw** sans arguments.

Pour les même raisons qu'**ITickable** il s'agit d'une interface.

Classe **DrawableObject**

La classe **DrawableObject** est la base de toutes les entités qui vont pouvoir être dessinées sur le jeu, elle permet d'implémenter toutes les variables nécessaires, une position (**x** et **y**), une largeur, une hauteur et une couleur (**r**, **g**, **b**). Les getters et des fonctions utiles telles que :

- **rotate_point :** la rotation d'un vecteur par rapport à sa position suivant l'angle donné ;
- **angle_to_drawable :** l'angle entre cet objet et un objet **DrawableObject ;**
- **distance_to** : la distance entre cet objet et un objet **DrawableObject**.

Cette classe hérite de **IDrawable** et n'implémente pas cette même fonction, faisant de la classe **DrawableObject** une classe abstraite.

La classe **Spacecraft** représentant les vaisseaux, hérite directement de cette classe.

Classe **MovableObject**

La classe **MovableObject** est la base de toutes les entités du jeu en mouvement**,** elle étend la classe **DrawableObject** décrite précédemment et y ajoute seulement les valeurs nécessaires pour un objet en mouvement, à savoir une vitesse et une direction. Mais aussi l'interface **ITickable** qui va permettre à chaque **MovableObject** d'implémenter la méthode **Tick** pour pouvoir évoluer au fil des ticks.

Les classes **Asteroid** et **LaserShot** étendent cette classe.

Factoy

Pour créer les astéroïdes et les vaisseaux, une factory est utilisée, elle est implémentée sous forme de singleton et permet de faciliter grandement l'ajout de ces derniers, par exemple la factory est appelée dans **ISpacecraftStrategy** pour pouvoir créer facilement à la volé le vaisseau voulu au bon endroit. Les valeurs de création sont dans la factory car cela permettra plus tard si on veut ajouter un fichier de configuration d'utiliser les valeurs ici.

Un semblant de modèle MVC

Classe **Environment**

La classe **Ennvironment** est le modèle, il comporte les données de la partie à savoir les vaisseaux dans un vecteur de pointeur sur **Spacecraft**, les astéroïdes dans un vecteur de pointeur sur **Asteroid**, le niveau courant, l'argent, les points de vie, une matrice de booléen permettant de savoir si une case est occupée ou non, et une référence vers la vue (**GraphivalDrawinngBoard**). Cette classe étend **GameEngineBase** et implémente donc la méthode **idle** qui va donc se charger d'appeler **Tick** sur chacun des astéroïdes et vaisseaux.

Classe **GraphicalDrawingBoard**

Cette classe permet l'interprétation totale du visuel, elle a comme membre principal un vecteur de pointeur de **IDrawable** qui seront dessinés à chaque appel de la méthode **Draw** hérité de **GraphicEngineBase**. Mais elle sert surtout à traiter les coordonnées

Classe **Controller**

Cette classe permet la mise à jour de la classe **Environment** en vérifiant les informations auprès de **GraphicalDrawingBoard** après avoir reçu une entrée clavier / souris.

Mise en place d'un **Strategy** pour la sélection des vaisseaux, lorsque l'on clique sur le menu du bas, on demande alors au **GragicalDrawingBoard** de nous donner la S**trategy** de vaisseau correspondant. Ainsi lors d'un clic sur le plateau de jeu, on appel seulement la méthode **getSpacecraft** de la **Strategy** (voir **ISpacecraftStrategy**), cela permet une évolutivité relativement facile à implémenter et évite d'avoir des conditions en brut dans le **Controller**.

Améliorations / Evolutivité

On peut déjà imaginer donner aux vaisseaux des stratégies d'attaque, ici ils en ont une mais c'est la seule qui existe : ils attaquent l'astéroïde le plus proche d'eux. On peut imaginer par exemple, attaque l'astéroïde le plus fort, le plus proche de côté gauche, etc.

Créer un fichier de propriété permettant la modification des valeurs à travers celui-ci.

Le projet est réalisé de sorte à ce qu'on puisse rajouter des types d'astéroïdes / vaisseaux très facilement, même changer la taille du plateau, le nombre de cases etc.

Conclusion

Les fonctionnalités attendues sont présentes, avec une légère modification au niveau des tirs des vaisseaux et du fonctionnement des astéroïdes, on peut atteindre le niveau 15 en réfléchissant un peu, le seul problème étant que je n'ai pas implémenté le fait de retirer un vaisseau pour en mettre un autre à la place, donc on devient vite limité quand le plateau devient complet.

C'est le premier projet réalisé en C++, et cela m'a permis, je pense, de beaucoup mieux comprendre le C++ malgré encore beaucoup de concepts inexplorés comme les templates.

Les choix de conceptions qui s'offraient à moi avec la richesse du langage étaient tous très intéressent, et j'aurai pu faire plein de choses de différentes manières. Les cours et TD ont grandement servit à l'appréhension de certain concept qu'offre le C++.