Projet C++

Bleyfuesz Yonathan

1 Introduction

Le but de ce projet est de coder en C++ un mini-jeu de tower defense. Dans ce jeu, des astéroïdes vont apparaître du côté droit de l'écran, et avancer peu à peu vers le côté gauche, et pour s'en défendre, il va falloir acheter et placer des vaisseaux qui pourront tirer sur les-dits astéroïdes.

Ce projet de programmation met notamment en évidence les problèmes d'héritage et de fonctions virtuelles, de redéfinition d'opérateurs, de création et destruction dynamiques, dont les différents exemples et applications seront détaillés par la suite, après une courte explication des étapes à suivre pour lancer le projet, et des règles du jeu et fonctionnalités implémentées.

2 Lancement du projet

Ouvrir la solution Visual Studio et la lancer en mode debug doit normalement suffire pour lancer le projet, puisque les options d'includes et de librairie ont été préconfigurées. Cependant, si cette préconfiguration a été perdue, il faut changer les paramètres suivants dans Projet > Propriétés de `bleyfuesz_yonathan_code` :

- Propriétés de configuration > Général > Version du Windows SDK : choisir une version présente sur l'ordinateur
- C/C++ > Autres répertoires include : mettre `$(SolutionDir)include`
- Editeur de liens > Répertoires de bibliothèques supplémentaires : mettre `$(SolutionDir)libWin`
- Editeur de liens > Entrée > Dépendances supplémentaires : mettre `glut32.lib; libGraph.lib;`
- Editeur de liens > Système > Sous-système : choisir `Console (/SUBSYSTEM:CONSOLE)`

3 Règles du jeu et fonctionnalités

Principe Le jeu comporte une grille de 144 cases dont chacune peut accueillir au maximum un vaisseau. Ces vaisseaux peuvent être placés sur la grille en les achetant avec des crédits qui sont soit donnés de base, soit gagnés en détruisant des astéroïdes. En effet, ces vaisseaux vont tirer en permanence, afin d'essayer de toucher des astéroïdes qui apparaissent sur la droite, et se déplacent vers la gauche. Si un astéroïde atteint le côté gauche de la grille, la partie est perdue. Le but est donc de détruire tous les astéroïdes. Les astéroïdes apparaissent un à un à l'intérieur d'une vague, et le joueur a la possibilité de commencer une nouvelle vague au moment où il le souhaite.

Commandes de jeu Toutes les commandes du jeu sont indiquées à l'écran, soit en permanence, soit lorsqu'elles sont utiles.

- Lancer une nouvelle vague : `N`
- Sélectionner un type de vaisseau : `A`, `E` ou `Z` suivant le type de vaisseau.
- Acheter un vaisseau et le placer sur la grille de jeu : clic gauche de souris. Le vaisseau placé sera du type précédemment sélectionné, et ne sera acheté et placé que si le joueur possède assez de crédits.

Types d'astéroïdes Trois types d'astéroïdes peuvent apparaître du côté droit de l'écran, et avancer peu à peu vers le coté gauche :

Average Astéroïde commun. Couleur brune, 75 points de vie.

Warping Astéroïde téléporteur. Couleur bleue, 100 points de vie. Ils se téléportent à intervalle régulier sur une ligne avoisinante.

Elliptical Astéroïde elliptique. Couleur grise, 120 points de vie. Après avoir subi 5 attaques, ils acquièrent un bouclier qui les rend bleus, et insensibles aux 2 prochaines attaques. Après ce temps d'insensibilité, leur bouclier disparaît, et ils redeviennent gris.

Types de vaisseaux Trois types de vaisseaux, ayant chacun un tir visuellement différent, sont implémentés dans le projet :

Enterprise Rapide et faible. Couleur rouge, double tir rouge et vert qui oscille, 15 points de dégats par double tir, haute fréquence de tir.

Serenity Peu rapide et puissant. Couleur bleue, tir rotatif blanc à une case de distance du vaisseau, 50 points de dégats par tir rotatif, basse fréquence de tir.

DeathStar Lent et très puissant. Couleur rose, tir pénétrant blanc qui s'étend à partir de la tête du vaisseau jusqu'à atteindre le bord droit et disparaître, 4 point de dégats par cycle, très basse fréquence de tir.

Amélioration des vaisseaux Chaque vaisseau peut être renforcé en positionnant un autre vaisseau derrière le premier : dans ce cas, vu qu'il y a deux vaisseaux sur la même ligne (limite maximale), le vaisseau de derrière ne pourra pas tirer tant que celui de devant est présent. Le vaisseau de devant obtiendra les améliorations suivantes, en fonction du type de vaisseau de derrière :

Enterprise Le temps d'attente entre deux tirs est diminué de 33%.

Serenity Ajoute un bonus de 2 de dégats à chaque tir, et le temps d'attente entre deux tirs est diminué de 20 cycles.

DeathStar Double les dégats.

Difficulté A chaque niveau impair, une nouvelle ligne peut être choisie pour faire apparaître des astéroïdes. De plus, régulièrement, la vitesse, les points de vie, le nombre d'astéroïdes augmentent. Le nombre d'astéroïdes est cependant limité à 20.

Fin de jeu Etant donné qu'il n'y a pas de limite supérieure au nombre de vagues, et que la difficulté ne fait qu'augmenter, le jeu se terminera forcément par une défaite du joueur : un astéroïde arrivera à atteindre le bord gauche du jeu. Un message "Game over" s'affichera, et le jeu continuera d'afficher la vague atteinte, le nombre d'astéroïdes et les crédits restants. Pour recommencer une nouvelle partie, il suffit d'appuyer sur `N` : attention, en plus de réinitialiser les données de jeu, cela fera immédiatement démarrer la vague 1.

Comportements surprenants Certains comportements peuvent sembler être erratiques : par exemple, les astéroïdes ne détruisent que par l'avant, mais ce comportement permet ainsi de pouvoir replacer plus rapidement les vaisseaux. On peut aussi citer les vaisseaux téléporteurs qui n'apparaissent pas forcément sur la ligne qui a été choisie pour faire apparaître des vaisseaux, mais sur celle au-dessus ou en-dessous ce qui oblige à attendre avant de placer les vaisseaux. Enfin, quand une DeathStar tire sur un astéroïde elliptique, le couleur de ce dernier oscille entre bleu et gris : étant donné que la DeathStar fait des dégâts à tous les cycles, l'astéroïde fait très rapidement apparaître son bouclier puis le perd presqu'instantanément, et ce, en boucle.

4 Classe `Environment`

Pour gérer tous les éléments de jeux (graphique, contrôle et action), j'ai décidé de créer une classe `Environment` qui est un singleton et qui contient tous les vecteurs d'objets. Elle contient aussi la classe `GraphicalInterface` (voir section 6). On initialise l'unique instance de cette classe dans le `main` et on la passe aux différents `engine`. Cette classe n'avait pas besoin d'être initialisée plusieurs fois, d'où le choix du singleton.

5 Itérateurs et classe `vector`

Fréquemment dans le projet, je dois parcourir les vecteurs d'éléments, afin d'en supprimer certains. Pour ce faire, au début, j'utilisais des itérateurs "à la java" :

```
for(auto pSpacecraft:_pSpacecrafts)
{
    pSpacecraft->doSomething;
}
```

Cependant, ce type de parcours a un comportement erratique lorsqu'on retire un élément du vecteur, et générait une erreur de type `BADACCESS`, due à un accès mémoire non valide. J'ai donc ensuite utilisé directement l'itérateur implémenté dans la classe `vector` :

```
for(auto it= _pSpacecrafts->begin(); it!=_pSpacecrafts->end();) { ... }
```

Ce code fonctionnait sur Mac. Cependant, il semblerait que sur Windows, la modification du vecteur est interdite lors de l'utilisation de ce type d'itérateur. J'ai donc fini par instancier, lors d'un premier parcours, une liste des éléments et des indices que je devais retirer, puis j'ai parcouru le vecteur dans l'autre sens pour effectuer mes modifications.

6 Classe façade pour le GUI

J'ai choisi de faire une classe par élément de l'interface graphique (menu haut, gauche, bas, damier). Toutes ces classes sont réunies dans une unique classe façade, par valeur. Ainsi, les interactions entre l'environnement et les différentes parties de l'interface graphique se font uniquement avec la classe façade `GraphicalInterface`. On notera que pour des raisons pédagogiques, j'ai choisi d'implémenter les éléments graphiques et les arguments des fonctions qu'ils utilisent par valeurs. Je suis conscient qu'à chaque appel de ces fonctions, les constructeurs de copie de ses arguments seront appelés. Cependant, vu que chacun d'eux n'est instancié qu'une seule fois (à l'initialisation de l'environnement), le nombre d'appels est fixé et donc non dérangeant. Si je n'avais pas fait cela, je n'aurais pas eu à écrire le moindre constructeur de copie qui aurait été véritablement appelé.

7 Eléments de jeu dynamiques

Pour ces éléments (vaisseaux, tirs, astéroïdes, etc), j'ai décidé de tous les stocker par pointeurs : cela permet d'éviter de constamment appeler les constructeurs de copie des classes. Les pointeurs ont aussi été choisis afin de rester cohérent avec l'exemple du vecteur à papillons fait en TD. De plus, il est impossible de créer des vecteurs de références car la classe `vector` n'admet comme élément que des types assignables.

8 Hiérarchie de classe

La plupart des classes sont issues d'une hiérarchie de classe. Les classes qui sont au sommet de ces hiérarchies sont en fait des classes abstraites (implémentant une ou plusieurs fonctions virtuelles pures). Plus on va descendre dans cette hiérarchie, plus ces classes virtuelles seront implémentées, jusqu'à ce qu'on arrive aux classes instanciables qui auront redéfini toutes les fonctions virtuelles pures. Les constructeurs sont eux aussi virtuels afin de les appeler à la destruction des classes héritées. On notera qu'il arrive, dans notre hiérarchie, que certaines classes implémentent des méthodes avec du code vide. Ceci est une erreur d'un point de vue purement conceptuelle, cependant, une telle pratique existe à l'intérieur même du kernel linux.

9 Hitboxs

Pour des raisons pédagogiques, j'ai décidé de baser tout mon système de détection et gestion des collisions sur la surcharge des opérateurs. Ainsi l'opérateur `&` a été redéfini pour prendre des hitboxs de différents objets du jeu et gérer les collisions.

10 Amélioration des vaisseaux

J'ai choisi de rajouter une fonctionnalité qui permet aux vaisseaux de renforcer d'autres vaisseaux. Pour ce faire, j'ai redéfini l'opérateur `+` dans la classe `CombatSpacecraft`. Cet opérateur effectue maintenant des downcasts dynamiques pour vérifier le type de l'objet qui lui est passé comme opérande de droite. En fonction de ce type, il effectue le renforcement sur le vaisseau de l'opérande de gauche. J'avais auparavant essayé d'implémenter cette fonctionnalité en redéfinissant plusieurs fois l'opérateur, avec directement le type de vaisseaux comme opérande de droite, et en utilisant donc la technique de "forward definition". Cependant, cette manière de faire les choses aurait nécessité d'effectuer énormément de downcasts dans le code de mon environnement ce qui était exclu.

11 Réécriture de fonctions dans la librairie

Dans la librairie, la fonction de dessin de polygones en 2 dimensions, donnés par la liste des coordonnées des sommets, permettant de dessiner des triangles ou des quadrilatères par exemple, ne permettait pas de dessiner des hexagones, ou des ellipses comme je le désirais. J'ai donc réécrit cette fonction afin de pouvoir dessiner des ellipses et autres polygones convexes.

12 Conclusion

Le projet implémente bien les fonctionnalités demandées (multi-lignes avec une grille de 12x12, 3 vaisseaux, 3 astéroïdes), avec l'ajout de l'amélioration possible de vaisseaux. Sans stratégie, le niveau 10 est atteint facilement, en testant plusieurs stratégies, le niveau 20 peut être atteint. On peut cependant regretter que les premières vagues sont vraiment simples, et qu'il faille attendre la vague 9 pour faire apparaître le premier astéroïde elliptique. Néanmoins, cela correspond complètement à l'esprit du type de jeu, et les paramètres de la première vague sont de toute façon modifiables via des define dans l'environnement (`STARTCREDIT`, `STARTWAVE`). Pour autant que je puisse en juger, le projet fonctionne.

A plusieurs reprises, j'ai fait un choix de conception ou d'implémentation qui me permettait de mettre en pratique des notions qui m'ont été apportées durant le cours de C++ du premier semestre. On pourra citer quelques notions de bases du C++ que le programme utilise, tels les constructeurs par valeurs/défauts/copies et les destructeurs, passage par valeur, par référence ou directement pointeurs, mais aussi des notions plus avancées telles que la redéfinitions d'opérateurs, l'héritage et les fonctions virtuelles, ainsi que le downcasting.

J'ai également pu expérimenter la non-portabilité du C++, notamment dans sa gestion des itérateurs, en codant de base sur Mac et en le passant ensuite sous Windows. Les itérateurs de vecteurs ont d'ailleurs été une source majeure de problèmes, qui fut fort fastidieuse à résoudre. Enfin, la structure du projet, en demandant de détruire et construire des objets rapidement, m'a obligé à adopter une certaine rigueur au niveau notamment des constructeurs et des destructeurs d'objet.

Au final, je pense que ce projet m'a permis d'améliorer mon niveau en C++.