Tom Redon
UGE

Rapport du projet tower_control

Table des matières

Réponses aux questions des diverses tâches:

Tâche 0 :

A - Exécution :

J'ai donc exécuté les commandes suivantes pour compiler et exécuter le programme :

cmake .

make

./tower

La fonction responsable des inputs du programme est : create_keystrokes().

Il faut appuyer sur C pour créer un avion.

Il faut appuyer sur X ou sur Q pour quitter le programme.

La touche F permet de mettre en full screen le programme.

L'avion tourne, atterit sur la piste, va au terminal puis redécolle et recommence.

Voici ce qui se passe en console :

AY9583 is now landing...

now servicing AY9583...

done servicing AY9583

AY9583 lift off

Si on ajoute plus de 3 avions, les autres avions attendent que les terminaux

soient disponibles pour attérir, ils tournent en rond tant qu'un terminal n'est pas disponible.

B - Analyse du code :

Commencant avec le repo GL/. Nous avons donc :

Displayable - c'est une classe abstrait qui forme la base pour tout les choses qui peuvent être dessiné sur l'ecran. La classe contient une coordonné z qui permet de trier les objets de cette classe.

DynamicObjject - une autre classe abstrait qui forme la base pour tout les choses qui peuvent "bouger" (en sense large, par example, un Terminal est aus

si un DynamicObject, son movement est le débarquement des avions)

opengl-interface - pas de classe ici, juste quelques fonctions necessaires pour interagir avec OpenGL; on remarque, par contre, la fonction timer dans laquelle tout les objets dans la move_queue ont leur move() appelé!

Texture2D - une texture qui contient un pointeur vers un img::Image (qui contient les octets vrac de l'image); la texture peut être affichée avec Texture2D::draw

Dans img/ on trouve les suivantes:

Image - une classe qui gère des octets d'un image dans la memoire pour être utilisé avec Texture2D

MediaPath - une classe qui gère l'acces aux PNGs qui vont avec le code

stb_image - pas de classe ici, c'est en fait une bibliotheque C qui sait lire les PNGs correctement

Finalement, les classes important pour la logique du projet:

Point2D / Point3D - classes qui gèrent des maths entre points dans l'espace 2D et 3D

Waypoint - un point sur un chemin d'un avion, c'est juste un Point3D avec l'information si ce point se trouve au sol, chez un terminal ou dans l'air; ici, on voit aussi qu'un "chemin" (WaypointQueue) est un deque de Waypoints.

Runway - stocke le debut et le fin d'un piste de décollage

Terminal - classe qui gère le débarquement d'un avion; chaque Terminal peut débarquer qu'un seul avion à la fois

Aircraft - un avion qui peut (1) être déssiner (Displayable) et (2) bouger (DynamicObject); chaque avion peut retourné son "flight number" ansi que sa distance à un point donné

AircraftType - le type d'un avion stocke des limites de vitesse et acceleration ansi que la texture; il y a 3 types pré-definit

Airport - gère l'aeroport, contient les terminals et le tower; seulment son friend class Tower peut réserver des terminals et démander un chemin de décollage

AirportType - contient les coordonnés importants (relatives au centre de chaque aeroport) comme le debut/fin des runways (il peut en avoir plusieurs); chaque AirportType peut génerer des chemins pour atterir et pour décoller

Tower - classe qui gère la fonctionalité du tour de control; des avions peuvent démander des nouvelles instructions ansi qu'indiquer qu'ils sont arrivés à leur Terminal; Chaque Tower contient une affectation des avions aux terminals. Si un avion X démande d'atterir à un moment quand tout les Terminals de l'aeroport sont affectés, alors le Tower retourne un "cercle" autour de l'aeroport pour que X re-démande quand il a fini son cercle

TowerSimulation - une classe pour la gestion de la simulation: creation de l'aeroport, affichage de l'usage sur la ligne de commande, creation des avions, etc

config - pas de classe ici, mais des constantes qui determinent quelques comportements de la simulation, par example le nombre d'intervals necessaire pour débarquer un avion

Tower :

get_instructions : l'avion récupère les instructions de la tour.

arrived_at_terminal : défini si l'avion est arrivé au terminal.

Aircraft :

turn_to_waypoint : se dirige vers le point défini

turn : tourne l'avion

get_speed_octant : donne la vitesse pendant un huitième de cercle

arrive_at_terminal : défini si l'avion est au terminal

operate_landing_gears : amorce l'atterissage

add_waypoint : ajoute un waypoint

move : bouge l'avion

display : dessine l'avion dans la fenêtre

Airport :

get_tower : donne la tour de l'aéroport

display : affiche la tour dans la fenêtre

move : fait bouger les avions dans le terminal

Terminal :

in_use : défini si le terminal est utilisé

is_servicing : défini si l'avion est en cours de débarquement

assign_craft : associe un avion à un terminal

finish_service : défini si l'avion a fini son débarquement

move : fais bouger l'avion sur le terminal.

Les classes impliquées sont waypoints et tower.

Les fonctions impliquées dans la génération du chemin sont get_instructions et get_circle.

Une deque parce que il est facile d'obtenir le premier élément.

C – Bidouillons ! :

1 - On peut modifier la vitesse dans aircraft_type.hpp.

2 - La variable est DEFAULT_TICKS_PER_SEC dans le fichier config.hpp. Les avions iront plus vite si les ticks sont augmentés.

3 - La variable qui contrôle le débarquement des avions est SERVICE_CYCLES.

4 - On peut supprimer les avions de display_queue et move_queue l'or de l'update.

5 - Dans le destructeur de displayable, on peut le supprimer de la queue. Il n'est pas pertinent d'en faire de même dans DynamicObject car les objets stockés dans move_queue ne sont pas const.

6 - On va plutôt utiliser une map pour cela.

D – Théorie :

1 - C'est grâce à la ligne suivante et au mot-clef :

```
friend class Tower;
```

2 – On est passé par une copie car la direction est modifiée par la fonction cap_length. Il ne me semble pas possible d'éviter la copie.

Tâche 1 :

Objectif 1 :

Les « contre » : Cela est un peu plus long d'implémenter une classe.

Les « pour » : Cela rend l'architecture simple à comprendre, le code plus simple à lire, cela est relativement simple à implémenter.

1 - La fonction timer dans opengl_interface détruit les avions dans le programme.

2 - Les structures contenant les références sur un avion sont : Tower, Terminal, move_queue et display_queue.

3 - On le détruit dans la fonction timer, dans Tower et Terminal, l'avion ne peut pas y être donc pas besoin de le supprimer et le destructeur de displayable le retire de display_queue et la fonction timer le retire de la move_queue.

4 - Car le conteneur d'avion est le propriétaire et donc il doit s'en occuper seul.

Vous pouvez voir l'implémentation de aircraft_manager dans le fichier aircraft_manager.hpp.

Objectif 2 :

Vous pouvez voir l'implémentation de aircraft_factory dans le fichier aircraft_factory et cela a impliqué des modifications dans tower_sim.cpp et dans les fonctions create_aircraft et create_random_aircraft.

Tâche 2 :

Objectif 1 :

On peut utiliser une structure [key, value] car cela est plus adapté qu'une structure pair qui sera moins lisible pour un utilisateur externe et qui nous permettra donc de séparer la paire de façon simplifiée.

A - On peut utiliser :

```
aircrafts.erase(std::remove_if(aircrafts.begin(), aircrafts.end(),[this] (std::unique_ptr<Aircraft>& aircraft_it)
```

Cela simplifie la boucle for et permet de supprimer les avions de la move_queue en une passe.

B - On peut utiliser :

```
for (auto i = '0'; i < '8'; i++)
{
GL::keystrokes.emplace(i, [this, i]() { std::cout << manager.airlines_number(factory.get_airline(i - '0')) << std::endl; });
}
```

Cela permet de simplifier le code et de ne pas répéter l'écriture 8 fois de la ligne nécessaire.

C - On peut utiliser les fonctions transform (incluse dans <algorithm>) et reduce (incluse dans <numeric>).

On aura donc juste à appliquer ces fonctions sur le tableau nouvellement créée en utilisant des lambdas ou d'autres fonctions telles que std::plus, std::minus ou bien encore std::multiplies.

Objectif 2 :

A - Pour le fuel, on rajoute juste un attribut dans la classe Aircraft auquel on associera une valeur dans l'AircraftFactory. Ensuite, on modifie juste la variable lors de l'update des aircrafts pour retirer le fuel petit à petit.

B - On a donc géré la réservation de terminal en faisant en sorte que les avions puissent les réserver au fur et à mesure. On a introduit deux fonctions qui vérifient les états de l'avion (has_terminal() et is_circling()). On peut donc se servir de has_terminal() pour construire la fonction dans Tower, reserve_terminal(), en se basant aussi sur le code de la fonction get_instructions().

On a par la suite, juste eu besoin de modifier un tant soit peu l'update dans Aircraft et de rajouter un if vérifiant si l'avion est en attente de réservation de terminal.

J'ai tenté de mettre en place un système de réservation et de déreservation des terminals pour les avions crashés et cela semble bien fonctionné.

C - J’ai mis en commentaire cette partie car elle provoquait une segfault.

D - J’ai implémenté la fonction is_low_on_fuel assez facilement.

Tâche 3 :

Objectif 1 :

1 - On peut faire de la récupération sur erreur dans l'update des avions.

2 - J'ai ajouté un attribut crash_number qui s'affichera, en appuyant sur m.

3 - J'ai rajouté une gestion d'erreur quand le fuel dans l'avion est à 0 dans l'update et j'invoque donc une erreur de type AircraftCrash.

Objectif 2 :

J'ai ajouté deux assert(airport) dans les fonctions create_aircraft et create_random_aircraft pour éviter d'avoir un aéroport nul.

De plus, j'ai pu vérifier que l'on ne renvoyait bien pas deux pointeurs vides (des nullptr).

J'ai ajouté un assert(image) dans la création d'airport pour être sûr que l'on ne passe pas une image vide.

J'ai ajouté un assert dans TowerSimulation pour vérifier que l'objet TowerSimulation n'est bien appelé qu'une fois.

Tâche 4 :

Objectif 1 :

1 - J’ai modifié le code comme demandé dans la question.

2 - J’ai pu utiliser un if constexpr qui regardera l’argument front et qui ajoutera en avant ou en arrière le waypoint. Il faudra bien préciser, entre chevrons, la variable front qui sera observée à l’exécution de la fonction.

3 - En comparant les deux codes, on peut facilement voir que le compilateur produit du code en assembleur pour l'un et non pour l'autre car il évalue que la partie de code ne sera jamais vérifiée.

Objectif 2 :

1 - J’ai rajouté toutes les fonctions qui étaient disponibles dans Point2D et Point3D pour être sûr de pouvoir, au futur, pouvoir remplacer ces deux structures par celle nouvellement créée.

2 - J’ai donc ajouté la fonction dans le fichier main.cpp, elle semble fonctionner correctement.

3 - On a donc ajouté les deux constructeurs nécessaires au déploiement de la structure pour fonctionner comme une structure englobant les deux anciennes structures Point2D et Point3D.

4 - Le programme ne compile pas, en m’indiquant qu’il y a trop d’arguments pour construire le point. L’erreur est dû à la globalisation des fonctions de constructions pour les Points2D et 3D.

5 - La valeur non-initialisée vaut toujours 0 lorsqu’on essaye de construire un point3D avec deux arguments. On peut donc finalement rajouter des static assert dans les deux constructeurs et dans les fonctions y() et z().

6 - Je ne sais pas vraiment comment traiter cette question.

Précisions :

J’ai répondu à tous les choix d’architecture dans les réponses de non-code. Mon mot d’ordre a été d’essayer de simplifer ou de produire le code le plus facile et le plus simplement lisible par un humain.

Il est fortement possible qu’il y ait de nombreuses choses à améliorer dans le projet au niveau de l’architecture mais j’ai tout de même essayé de faire quelque chose d’assez propre dans son organisation.

Commentaires personnels :

Le projet était assez intéressant, mais j’ai trouvé quelques tâches plus complexes de par leur longueur, tel que la tâche 2 qui m’a demandé beaucoup de temps, pour obtenir seulement un résultat mitigé.

J’ai cependant apprécié travailler dessus et je pense avoir appris de nombreuses choses et cela m’aura surtout donné la chance d’appliquer des éléments du cours pour les rendre un peu plus concrets.

J’ai apprécié le fait que le projet soit tourné dans certaines parties comme une correction de code.