Master Informatique — Université Gustave-Eiffel
Première année

Compte-rendu
PROJET

C++

Jolan OSTER – 197451
Groupe Apprenti

2021/2021
Céline Noël

TASK_0

Exercice A. Exécution

C → Appelle un avion
F → Plein écran
X, Q → Ferme le programme

L’avion atterrit à l’aéroport, ce gars, redécolle, fait un tour dans le ciel avant de recommencer.

Affichage lors du parcours de la boucle d’un avion :
<ID Avion> is now landing...
now servicing <ID Avion>...
done servicing <ID Avion>
<ID Avion> lift off

Quand plusieurs avions ce suivent, les 3 premier avion commence à atterrir alors que les suivants attendent en faisant des tours jusqu’à ce que les avions décolle pour laisser leur place.

Exercice B. Exécution

A partir sur la racine, on peut observer plusieurs classes et structure :

Aircraft → represente les avions du projet. Cette classe herite de DynamicObject et Displayable.

distance_to(): renvoir la distance entre cet avion et un point

display() : afficher l'avion

move() : mise à jour de la position de l'avion

AircraftType → le type d'un avion stocke des limites de vitesse et accélération ainsi que la texture, il y a 3 types pré-definit.

Airport → gère l'aéroport, contient les terminaux et le Tower. La classe Tower est friend class de Airport et elle seule peut réserver des terminaux et demander un chemin de décollage.

get_tower() : renvoie la tour de contrôle

display() : affiche l'aéroport

move() : fait bouger tous les terminaux

AirportType → contiennent les coordonnés importantess (relatives à l’aéroport) comme le début/fin des runways (il peut en avoir plusieurs), chaque AirportType peut générer deschemins pour atterrir et pour décoller.

Point2D → classes qui gèrent des maths entre points dans l'espace 2D.

Point3D → classes qui gèrent des maths entre points dans l'espace 3D.

Runway → représente une piste d’atterrissage par rapport à des points 3D.

Terminal → classe qui gère le débarquement d'un avion. Chaque Terminal ne peut pas débarquer qu'un seul avion à la fois.

in_use() : renvoie true s'il est en cours d'utilisation

is_servicing() : renvoie true de moment qu'il n'a pas fini de s'occuper de l'avion.

assign_craft() : assigne un nouvel avion.

start_service() : affiche un message et commence à s'occuper de l'avion.

finish_service() : affiche un message et supprime la reférence sur l'avion.

move() : incremente le conteur le temps du service.

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 …

Tower → classe qui gère la fonctionnalité de la tour de contrôle; des avions peuvent demander des nouvelles instructions ainsi qu'indiquer qu'ils sont arrivés à leur terminal. Chaque Tower contient une affectation des avions aux terminals. Si un avion demande d'atterrir à un moment quand tous les terminals de

l'aéroport sont affectés, alors le tower retourne un "cercle" autour de l'aéroport pour que l'avion redemande quand il a fini son cercle.

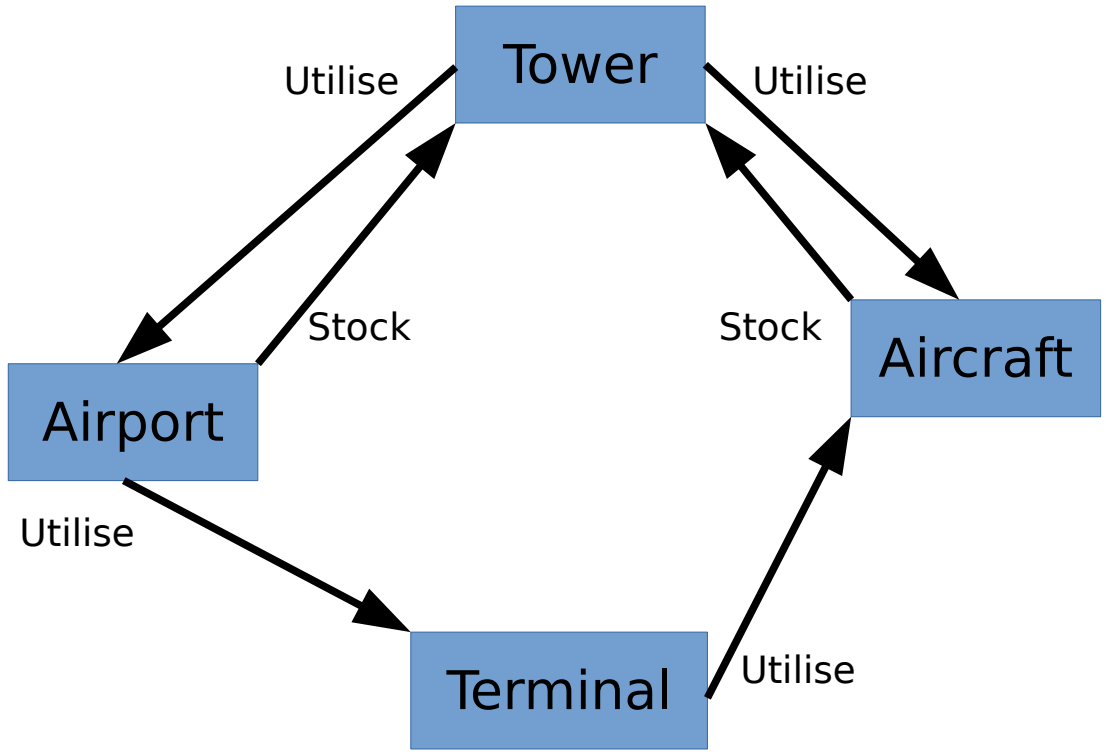
get_instructions (Aircraft&) : génère des instructions pour l'avion.

arrived_at_terminal (Aircraft&) : récupère le terminal sur lequel doit se poser l'avions et ensuite lui donne l'avion.

Displayable → c'est une classe abstraite qui représente les objets qu'on peut dessiner dans le projet. La classe contient une coordonné qui permet de trier les objets de cette classe.

DynamicObject → une classe abstraite qui forme la base pour tous les objets qui ont la spécificité de bouger (de faire bouger un autre objet pour le cas du terminal).

Texture2D → représente l'affichage d'un objet grâce a une image et a ses caractéristiques.



Les classes Waypoint, Airport et Tower sont impliquées dans la génération du chemin d’un avion par le biais des fonctions get_instructions, reserve_terminal, get_circle, reserve_terminal, et start_path. Pour représenter le chemin d’un avion on utilise un queue.

Une queue parce qu'on veut ajouter des points à la fin et pouvoir les récupérer à partir du début.

Exercice C. Exécution

1- Dans le fichier « aircraft_types.hpp », dans la méthode « init_aircraft_types », chaque ligne est une déclaration d’un nouvelle avion et la deuxième valeur est le nombre max de vitesse possible.
La vitesse du concorde passe de 0.05 à 0.1.

2- Dans le fichier « opengl_interface.hpp » il y a une variable « ticks_per_sec » qui représente les images afficher par seconde et donc le framerate.
La variable « ticks_per_sec » est utilisé dans le dénominateur d’un division pour le calcul du framerate, si on essaie de mettre pause en mettant ticks_per_sec » à 0 il y aura donc une division par 0 et donc une erreur.
Pour mettre en pause sans variable globale, il faut créer une variable globale « pause » a l’intérieur du fichier « opengl_interface.hpp » avant de l’utiliser dans le fichier « opengl_interface.cpp » dans la fonction « timer » pour bloquer la boucle.

3- Dans le fichier «config.hpp » c’est la variable « SERIVCE_CYCLES » qui est utilisé pour le temps de débarquement des avions.

4- Un avion doit être supprimé quand il a terminé son atterrissage et qu’il est repartie. Dans la méthode « get_instructions », on peut

observer un « return get_circle() » à un certaine endroit, il faut donc modifier cette endroit.

Cependant on peut pas supprimer un avion a ce moment la, il faut faire remonter l’information jusqu’à la boucle qui effectue les actions de chaque objet pour pouvoir les supprimer.

5- Pour insérer et supprimer automatiquement un avion on doit faire l’insertion dans le constructeur et la suppression dans le destructeur.

6- Pour pouvoir remplacer le « Vector » pour quelque chose de plus efficace a la recherche d’éléments enregistré, il est préférable d’utiliser une « Map ».
Cela permet d’avoir une complexité de recherche optimisé et de ce passer de la fonction de recherche par itérator.

Exercice D. Théorie

1- Pour bloquer toutes les autres classes d’utiliser la fonction de réservation on pourrait la mettre en privé. Ensuite, il faudrait indiqué à la classe « Airport » que la classe « Tower » est une classe safe qui a la permission d’utiliser les informations donné en privé.

2- Il est préférable de ne pas passer par une référence pour que le chemin ne puissent pas ce modifier après sa création, cela pourrait créer des problèmes.

Sachant qu’un Point3D est une classe qui représente un point, il est probable de bloquer une copie grâce au constructeur de copie.

TASK_1

Analyse de la gestion des avions

Il faudrait chercher cette avion dans les deux files et faire en sorte que ce soit bel et bien le même, sachant que l'endroit en question doit avoir accès a ses files. Cela ne semble pas réalisable correctement.

Objectif 1. Référencement des avions

A – Choisir l'architecture

Pour référencer les avions présent il faut donc avoir une nouvelle classe qui puisse gérer ce genre d'information.

Cependant il faut choisir entre : Créer une nouvelle classe ou intégrer cette capacité a une classe existante.

Il serait préférable de créer une nouvelle classe par soucie d'organisation. Chaque classe possède son propre objectif, il faut donc éviter d'intégrer de nouvelles capacités aux classes déjà existantes, dans le bute d'avoir des classes claire et bien organisé.

B – Déterminer le propriétaire de chaque avion

1- Quand une avion doit être détruit, l'information remonte jusqu'au fichier « opengl_interface.cpp », qui fait la demande de suppression auprès des différentes listes de stockages.

2- La classe « Displayable » et « DynamicObject » possèdes tous deux une liste qui peuvent stocker un avion qui doit être supprimer au moment ou l'on décide qu'il faut en supprimer un.

3- Pour supprimer la référence de l'avion, on utilise l'itérateur qui parcours les objets déplacer pour le retirer de la « move_queue ». Puis on supprime l'itérateur qui utilisera le destructeur de la classe

« Displayable » pour pouvoir recréer un itérateur capable d’être utilisé pour supprimer l’objet de la « display_queue ».

4- Il n’est pas très judicieux d’essayer de faire la même chose pour « AircraftManager », cela va dupliquer la recherche des informations concernant les avions (parcourir deux fois une liste similaire).

C – C'est parti !

Pour stocker les avions, il faut donc utiliser un vector de Aircraft.

L’idée est de remplacer tous les avions de la move_queue, par un unique objet « AircraftManager » qui fera donc le travail de move tous les avions qu’il possède au moment ou on lui demande de le move.

Le aircraft_manager sera initialiser comme un airport, avec un création a nullptr ainsi qu’une véritable initialisation a l’aide d’une fonction faite pour avant d’être introduit dans la move_queue.

Pour finir, il faut utiliser une fonction d’insertion pour le « AircraftManager » a chaque que l’on créer un nouveau aicraft.

Pour faire en sorte que « AircraftManager » soit l’unique possesseur différents avions, il faut utiliser des « unique_ptr ».

Objectif 2. Usine à avions

A – Création d’une factory

Après avoir créé une classe « AicraftFactory » il faut regrouper toutes les informations nécessaire a l’intérieur de cette classe. Les variables globale deviennent des champs privé, car ils ne sont pas nécessaire ailleurs.

La classe possède donc une méthode permettant de créer un avion sur commande, regroupant toutes les informations nécessaire avant d’envoyer un unique pointeur.

La classe « TowerSimulation » possédera une instance de « AicraftFactory » pour qu’elle puisse demander la création d’un nouvelle avion lorsque le joueur appuie sur la bonne touche.

B – Conflits

Pour gérer ce problème, il faut ajouter un set dans les champs privé de la casse « AicraftFactory », qui liste tous les numéro de vol des avions créé par la classe.

Il faut aussi ajouter un do-while, qui boucle jusqu’à ce qu’un numéro de vol original soir créé pour terminer la création d’un nouvelle avion.

TASK_2

Objectif 1. Refactorisation de l’existant

A – Structured Bindings

La strucutured binding adapté serait de remplacer la simple variable pair par « [key, value] » qui sépare déjà les deux valeur.

B – Algorithmes divers

1- Pour remplacer la boucle for par un « remove_if » il faut appliquer la méthode « move » pour tous les éléments du vector, chaque avion qui doit être supprimé a une méthode « move » qui renvoie true. A la fin du « remove_if » il faut supprimer tous les éléments entre la fin de la nouvelle liste et de l’ancienne, ce qui supprimer tous les éléments qui ont renvoyé true.

2- Pour avoir le nombres d’avions en fonction de leur type d’airlines, il faut plusieurs choses : que AicraftFactory donne les string par rapport aux id (1, 2, 3..) et que AicraftManager donne le nombre rapport au string précédent.
Pour arriver a les compter de manière efficace, on utilise la fonction « count_if » qui compte le nombre d’occurrence qui passe le test donné en paramètre. Le test en question étant un « compare » du début du numéro de vol et du type recherché.

C – Relooking de Point3D

Après avoir remplacer le tableau par un « std::array », on peut utiliser les fonction « std::transform » qui appliquera une fonction fonction des paramètre donné.
Chaque Méthode a remplacé aura donc une utilisation unique de cette fonction.

Objectif 2. Rupture de kérosène

A – Consommation d'essence

Pour utiliser correctement les bornes utilisé pour la valeur aléatoire de l’essence, il faut en faire des variables globale qui ce situe dans le fichier « config.cpp ». Une fois la variable correspondant a la quantité d’essence initialisé dans le constructeur, il faut l’utiliser dans la fonction « move » de l’aircraft. Retournant la valeur « true » si jamais il n’y a plus d’essence pour pouvoir supprimer l’avion (c’est la même stratégie quand un avion a terminer son chemin).

B – Un terminal s’il vous plaît

1- La fonction « has_terminal » utilise le « waypoints » pour savoir ses instructions sont vide et si un terminal est déjà réservé.

2- La fonction «is_circling» a besoin d’une nouvelle variable de la classe « Aircraft » pour savoir si son service a déjà été effectué. En plus de cela il faut savoir si l’avion est sur le sol ou sur un terminal pour savoir si l’avion est en train d’effectuer une boucle en vol.

3- La fonction « reserve_terminal » réserve donc un terminal pour un avion si jamais l’avion n’est pas sur un terminal et si jamais il y a une terminal de disponible.

4- Pour utiliser les fonctions créé précédemment, il faut donc vérifier (dans la fonction « move ») que l’avion est en attente dans sa boucle. Si oui, il faut donc tenter de réserver un terminal qui sera ajouter dans son chemin si la réservation foncitonne.

C – Minimiser les crashs

Pour pouvoir ordonner correctement les avions dans la liste, il faut pouvoir récupérer leur quantité de carburant et donc créer une fonction « getFuel ». Ensuite, la fonction de trie pourra organiser les avions en fonction des avions sur un terminal et de leurs niveau de carburant.

D – Réapprovisionnement

1- Pour créer la fonction « is_low_on_fuel », il faut de nouveau une variable globale qui ce placera dans le fichier « config.hpp » afin de définir la quantité minimum d'essence nécessaire pour repartir. Ensuite, il faut modifier la fonction « move » de la classe « terminal.hpp » pour ajouter une condition a l'avancé du service.

2- Utilisant la fonction « accumulate » pour faire le calcul de la somme, il était aussi nécessaire d'ajouter le fait que la classe « AircraftManager » pouvait avoir accès aux champs privé de la classe « Aircraft » afin de rendre le code plus facile.

3- Pour faire en sorte que la classe « Aiport » ait accès au AicraftManager, il faut qu'il possède le champ grâce a son constructeur donné par la classe « tower_sim » qui ce charge de créer l'airport. Après avoir corrigé le code pour que cela puisse être faisable, il suffit de donner l'adresse a la classe airport.

4- Pour gérer le fait que le stock de fuel peut être trop faible, il faut faire en sorte de le mettre a zéro tout en utilisant le reste si jamais il n'y a pas asse de fuel pour remplir l'avion.

5- Pour créer la fonction « refill_aircraft_of_neede » il aura fallu modifier la variable « current_aircraft » pour qu'elle ne soit plus en constante.

6- Celui a cela il fallu corriger quelque morceau de programme qui faisais planter le programme, comme le mauvais type de int, et il problème de calcul de carburant.

A ce stade le programme fonctionne bien mais si il y a trop d'avion il y a un « Segmentation fault » qui est impossible de régler. (Je ne sais pas d'où sort cette erreur.

TASK_3

Objectif 1. Crash des avions

1- Pour gérer les crash d’avions et donc les exception de type « AircraftCrash » il faut que l’erreur soit récupéré dans « AicraftManager » qui sera la pour supprimer l’avion.

2- Le compteur est simple a mettre en place, car il suffit de le placer a l’endroit ou les exceptions sont récupéré pour l’incrémenté et être stocké dans le manager. La touche choisit est la touche « m ».

2- Au lieu de retourner true (qui fera supprimer l’avion), on lève une exception quand l’avion tombera a 0 de carburant.

Objectif 2. Détecter les erreurs de programmation

Les asserts seront a implanter a certains endroit, pour être sur que le programme fonctionne bien. Au début des fonctions pour que le programme s’arrête si jamais certains paramètres n’ont pas les bonnes valeurs.

TASK_4

Objectif 1. Devant ou derrière ?

1- 2- Pour changer l’appel de la fonction « add_waypoint » il faut utiliser un template avec un boolean qui ce chargera de la variable front. Il faut changer la variable qui ce charge du front en constante.

Objectif 2. Points générique

1- On peut donc fusionner les classes Point2De et point3D en une seul et unique classe qui utilise un template pour définir la taille du tableau de point. Il faut faire attention a définir toutes les fonction de manière a s’adapter pour des points 2D et 3D.

2- Tous les tests fonctionne.

3- Il faut donc créer deux constructeur en fonction des deux point possible, faisant attention a ce que les tailles soit respecté. Les alias sont pratique pour définir des points avec la bonne taille de tableau.

4- L’erreur s’effectue pendant la compilation, le programme ne souhaitant pas intégré une troisième variable a cause de la taille du tableau. Car informatiquement le programme détecte la trop grande quantité de valeurs pour son tableau.

5- Cependant pour une instance de Point3D avec 2 arguments aucune erreur ne ce produit, il faut donc mettre des asserts pour bloquer la moindre tentative d’erreur a ce niveau la.

ATTENTION : Il y a un problème avec mon programme que je n’arrive pas a résoudre.

Aléatoirement dans le programme (pour moi, car je ne sais pas d’ou ça vient) Il y a une Segmentation fault dans le fichier « stl_deque.h ». Cette erreur a été vue pour la première fois pendant la Task 2, mais elle ne semble pas lié a la gestion du carburant. Le programme fonctionne correctement mais est interrompu sans grande justification.
Il me semble que cette erreur provient de la gestion des Waypoint, mais aucune idée de ou elle pourra provenir. (L’erreur peut survenir tot ou tard)