Projet C++ - Tower Sim

Table des matières

TASK 0

A – Exécution

Pour ajouter un avion, il faut appuyer sur la touche *c*.

Pour quitter le programme, il faut appuyer sur la touche *q*.

La touche *f* sert à mettre en plein écran. Cependant sur MacOs, la mise en grand écran produit une erreur.

Le comportement de l'avion est qu'il atterrit, qu'il décharge son contenu dans un terminal qui lui est réservé et après il redécolle.

Les informations dans la console sont :

- un message lorsque l'avion atterrit
- un message lorsque l'avion décharge son contenu
- un message lorsque l'avion a fini de décharger son contenu
- un message lorsque l'avion décolle

Lorsqu'on ajoute 4 avions, 3 d'entre eux atterrissent et se dirige vers le terminal qui leur est réservé et font leur déchargement et le 4ème attend en faisant des tours dans les airs le temps qu'un terminal se libère.

B – Analyse du code

Les classes

Classe	Description
aircraft	Cette classe regroupe toutes les fonction qui permettent de faire bouger l'avion, de le faire atterrir avec les bonnes vitesses, de prévenir la tour que l'avion est arrivé au terminal et de l'afficher dans la fenêtre graphique.
aircraft_type	Cette classe permet de lister les types d'avion qu'on a.
airport	Cette classe permet de réserver un terminal pour un avion, de lui envoyer la route à suivre jusqu'au terminal, d'afficher un aéroport avec ses terminaux
airport_type	Cette classe permet de créer un aéroport aves ses caractéristiques (nombre de terminaux, nombre de piste, forme, ...)

config	Cette classe liste des variables globales utiles aux autres classes.
geometry	Cette classe regroupe toutes les fonctions liées à la géométrie. Elle redéfinit les opérateurs +=, *=, +, * de point en 2D et de point en 3D, elle donne accès aux fonctions de calculs de distance diverses.
main	C'est le main du projet, la classe qui permet de lancer le projet.
runway	Cette classe permet de créer les pistes d'un aéroport
terminal	Cette classe regroupe les fonctions qui sont en lien avec un terminal, comme le fait qu'un terminal est déjà utilisé, l'état du débarquement, d'assigner un terminal à un avion, de commencer ou de terminer le débarquement, …
tower	Cette classe joue le rôle de la tour de contrôle, elle essaye de réserver un terminal à un avion si cela est possible sinon la tour dit à l'avion de faire un tour d'attente, elle permet de lancer le débarquement lorsque l'avion est arrivé à son terminal.
tower_sim	Cette classe permet de configurer l'application, elle crée, de manière aléatoire ou non, les avions, elle initialise l'aéroport, elle permet d'afficher l'aide si l'utilisateur la demande, elle assigne les actions aux touches.
waypoint	Cette classe permet de définir si un avion est dans les airs, en train d'atterrir ou bien au terminal

La classe Tower

La fonction **get_circle()** retourne le cercle que doivent faire les avions dans les airs si ils sont en attente de réservation d'un terminal.

La fonction **find_craft_and_terminal(const Aircraft& aircraft)** permet de trouver un avion et le terminal qui lui est associé.

La fonction **get_instructions(Aircraft& aircraft)** permet de gérer un avion, en lui disant ce qu'il doit faire et à quel moment, si l'avion est près de l'aéroport et qu'il y a un terminal de libre alors il faut le lui réserver sinon on lui dit de faire un cercle au dessus de l'aéroport pour attendre et lorsque l'avion est déjà à son terminal, on gère son débarquement.

La fonction **arrived_at_terminal(const Aircraft& aircraft)** permet de lancer le débarquement lorsque l'avion est arrivé à son terminal.

La classe Aircraft

La fonction **get_flight_num( )** permet de récupérer le numéro de vol d'un avion afin de pouvoir l'identifier.

La fonction **distance_to(const Point3D& p)** permet de calculer la distance entre cet avion et le point p entré en paramètre. Elle permet donc de savoir où est l'avion par rapport à certain point.

La fonction **display()** sert à effectuer l'affichage d'un avion dans la fenêtre graphique.

La fonction **move()** permet de faire bouger l'avion et de le faire suivre une route selon sa position.

La classe Airport

La fonction **get_tower()** permet d'avoir accès à cette tour de contrôle.

La fonction **display()** sert à effectuer l'affichage de la tour de contrôle.

La fonction **move()** permet de faire avancer le débarquement de chaque terminaux relié à cette tour de contrôle. En effet, il faut un certain nombre de cycle pour que le débarquement soit terminé.

La classe Terminal

La fonction **in_use()** permet de savoir si un terminal est en train d'être utilisé par un avion afin de ne pas l'attribuer à un autre avion.

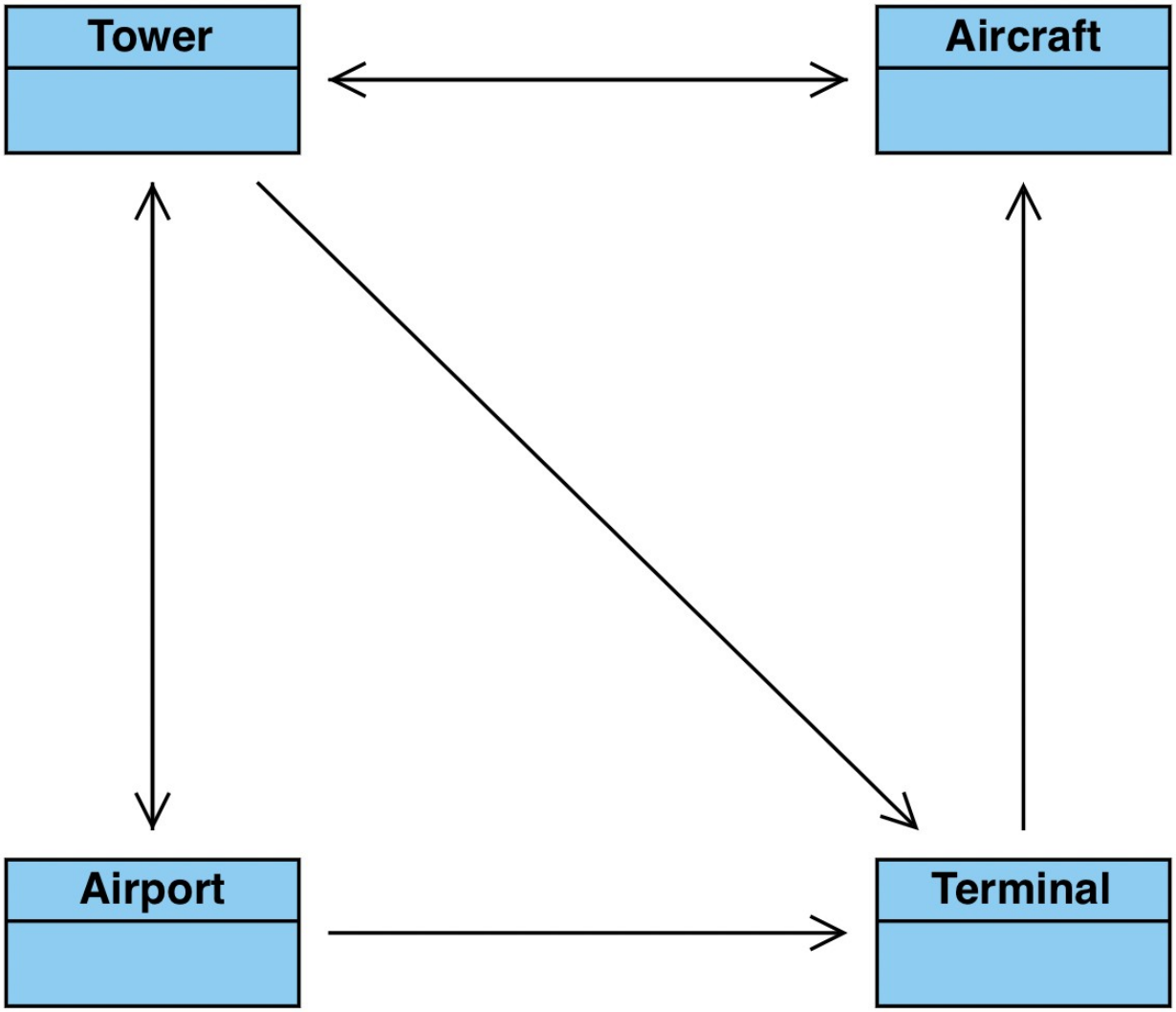
La fonction **is_servicing()** permet de savoir si le débarquement est en cours ou pas.

La fonction **assign_aircraft(*const* Aircraft& aircraft)** permet d'allouer un avion à ce terminal, c'est la réservation de ce terminal pour l'avion.

La fonction **start_service(*const* Aircraft& aircraft)** permet de commencer le débarquement d'un avion qui se trouve sur ce terminal.

La fonction **stop_service()** permet d'indiquer si le débarquement est fini. On utilise la fonction i**s_servicing()** dans cette fonction.

La fonction **move()** permet d'augmenter la variable correspond aux cycles de débarquement.



Génération du chemin d’un avion

Les classes impliquées dans la génération du chemin sont les classes *waypoint, tower, aircraf.*

Le conteneur de la librairie standard pour représenter le chemin choisi est std::deque.

Les intérêts de ce choix sont que ce conteneur est random access donc l’accès se fait en temps constant, l’ajout en début ou en fin se fait aussi en temps constant, augmenter ou réduire la taille de deque est automatique et coute moins chère qu’augmenter la taille d’un vector.

C – Bidouillons !

Les vitesses maximales et l’accélération de chaque avions sont définies dans la classe *aircraft_type.*

La variable ticks_per_sec de la classe opengl_interface contrôle le framerate de la simulation.

Les deux nouveaux inputs sont *a* qui permet d'augmenter le framerate et *b* qui permet de le diminuer.

Pour mettre en pause, j'ai essayé de mettre la framerate à 0 pour que l'image se fige mais cela ne produit une erreur car dans la fonction timer de la même classe, on fera une division par 0.

La variable qui contrôle le temps de débarquement des avions est SERVICE_CYCLE de la classe *config*. Elle passe de 20 cycles à 40.

On peut savoir quand l'avion doit être supprimé dans la méthode get_instructions de Tower en passant la variable *was_at_airport* à true signifiant que l'avion a été au terminal et a fait son débarquement.

Il ne faut pas supprimer l'avion dans cette méthodes car on pourrait avoir des problèmes avec les itérateurs. On fait la suppression dans la méthode timer de opengl_interface en appelant la méthode delete_asap qui dit si l'avion doit être supprimé ou non.

La méthode delete_asap renvoie la variable *was_at_airport* qui, elle, est changée dans Tower de false à true après le débarquement.

D – Théorie

Comment a-t-on fait pour que seule la classe Tower puisse réserver un terminal de l'aéroport ?

La classe Tower est une classe ami de la classe Terminal et le champs *reserved_terminal* est privée dans la classe Tower donc seule cette classe peut le manipuler.

En regardant le contenu de la fonction void Aircraft::turn(Point3D direction), pourquoi selon-vous ne sommes-nous pas passer par une référence ?

Cela est du au fait que la méthode cap_length de Point3D va modifier cette valeur

Pourquoi n'est-il pas possible d'éviter la copie du Point3D passé en paramètre ?

Il n'est pas possible d'éviter la copie car on ne veut pas le modifier si on le passe plusieurs fois dans la méthode.

Task 1

Gestion mémoire

Analyse de la gestion des avions

Pour accéder à un avion à un moment ou à un autre du programme, il faudrait aller regarder dans le vector *GL::displayable_queue* auquel on a accès et en utilisant la méthode *get_flight_num()*. Cependant, ce vecteur est rempli de *Displayable*, il faudra donc vérifier dynamiquement si l'objet est bien un avion.

Objectif 1 - Référencement des avions

A – Choisir l'architecture

	Avantages	Inconvénients
Créer une nouvelle classe	• Plus optimisé • La classe n'aura que ce rôle à faire	• Plus long à implémenter
Donner le rôle à une classe existante	• Plus rapide à mettre en place	• Classe plus lourde • La classe aura plusieurs rôles

B – Déterminer le propriétaire de chaque avion

Qui est responsable de détruire les avions du programme ?

La méthode *timer()* de **opengl_interface** est responsable de la destruction des avions.

Quelles autres structures contiennent une référence sur un avion au moment où il doit être détruit ?

La map *AircraftToTerminal* de Tower, *GL::display_queue* et *GL::move_queue* contiennent une référence sur un avion.

Comment fait-on pour supprimer la référence sur un avion qui va être détruit dans ces structures ?

On passe par le parcours d'iterator et on utilise la méthode erase qui va renvoyer un nouvel iterator pour ne pas casser le parcours de la structure. Pour le pointeur sur Aircraft, on le fait juste pointer vers *nullptr*.

Pourquoi n'est-il pas très judicieux d'essayer d'appliquer la même chose pour votre AircraftManager?

On va utiliser un meilleur système, plus adapté et intelligent pour AircraftManager.

Objectif 1 – Usine à avions

A – Création d’une factory

A quelle ligne faut-il définir context_initializer dans TowerSimulation pour s'assurer que le constructeur de context_initializer est appelé avant celui de factory ?

Il faut le définir avant la définition de factory.

Task 4

Templates

Objectif 2 - Points génériques

Dans la fonction *test_generic_points*, essayez d'instancier un *Point2D* avec 3 arguments. Que se passe-t-il ? Comment pourriez-vous expliquer que cette erreur ne se produise que maintenant ?

On a une erreur de compilation car le tableau de Point2D est de taille 2 et on essaye de lui mettre 3 valeurs. L’erreur ne se produit que maintenant car Point2D est un alias et que le constructeur ne tient pas compte du type à gauche.

Que se passe-t-il maintenant si vous essayez d'instancier un *Point3D* avec 2 arguments ?

Le code compilera mais, à l’exécution, nous avons un *undefined behavior* car la troisième valeur n’est pas initialisé.

Résumé personnel

Choix de l’architecture du programme

Lors de la suppression des avions, j’ai fait le choix de partir sur une méthode qui indiquait si oui ou non un avion pouvait être supprimé. Cette méthode est ***delete_asap*** et comme son nom laisse présumé indique quand il est possible de supprimé un avion le plus vite possible. Cette méthode renvoie une variable indiquant si un avion a fait son débarquement ou non.

Problèmes

J’ai eu un problème causé par le crash d’un avion. En effet, lorsqu’un avion a un terminal de réserver mais qu’avant d’atterrir il se crash, je ne gérait pas bien le crash et le terminal restait réserver pour rien et donc aucun autre avion ne pouvait le réserver. Si on laissait le programme tourné un certain temps, on pouvait voir que aucun avion n’atterrissait plus car tous les terminaux étaient réservé alors qu’aucun avion n’était dessus, ce qui faisait tombé les avions dans les airs en panne d’essence les uns après les autres. J’ai corrigé ce problème en créant une méthode dans ***Tower*** qui s’occupe d’enlever la réservation d’un terminal d’un avion qui s’est crash. Cette méthode s’appelle ***delete_crashed_aircraft*** et s’occupe de libérer le termina libéré de l’avion qui l’avait réservé afin de ne pas le bloquer à l’infini. Et le problème est réglé !

J’ai aussi eu du mal à bien comprendre la notion de template qui m’a pris plus de temps à assimiler que le reste. Je n’ai malheureusement pas réussi à faire la partie sur les variadic-template à cause de ces quelques lacunes.

Ce que j’ai aimé / détesté

J’ai aimé le fait de partir d’un code déjà existant car c’est majoritairement à cela qu’on sera confronté dans le monde du travail. De plus, les consignes étaient très claires et bien présentées avec ce concept de *task*. Le coté très graphique du projet aide aussi à se plonger dedans et j’ai plutôt vu ça comme un « jeu », ce qui aide à se motiver. De plus, le thème m’a plu aussi.

Faire ce projet m’a permis de conforter le fait que je n’utiliserai pas ce langage en priorité pour d’éventuelles futurs projets. En effet, la syntaxe ne me convient pas trop et la complexité du langage me ferait fuir je pense, surtout quand Java existe...

Ce que j’ai appris

J’ai pu faire de la programmation orienté objet dans un autre langage que Java ce qui m’a permis de découvrir d’autre façon de faire les choses. J’ai pris en expérience sur le fait de m’adapter au code de quelqu’un d’autre et cela est une chose que je recherche de plus en plus. Plus particulièrement au langage, j’ai appris à utiliser une multitude de pointeurs et de conteneurs différents. J’ai appris à les comparer, à mieux les choisir. Pour finir, ce projet m’a permis de me faire une meilleures idée sur ce que je voudrais faire plus tard, avec quelles technologies je serais le plus à l’aise.