VONG Stéphane

RAPPORT Cpp : Tower simulator

Table des matières :

TASK 0

A- Exécution

Pour ajouter un avion : touche c

Pour quitter le programme : touche q

La touche f permet de mettre en plein écran.

Lorsqu'un avion est créé, l'avion va se préparer au lancer puis, une fois lancer, fais un tour dans les airs pour revenir sur le sol. Chaque avion aura sa propre ronde a effectuer.

B- Analyse du code

Liste des classes du programme du dossier src :

- aircraft -> permet de créer un avion
- aircraft_types -> permet de définir les types d'avion
- airport -> permet de créer un aéroport
- tower -> permet de créer une tour
- tower_sim -> permet d'initialiser le simulateur (initialise les tours/avions)
- terminal -> permet de notifier la tour de l'action d'un avion (départ/arrivé)
- waypoint -> permet de désigner la position d'un avion (coordonné d'un avion, sur terre/dans les airs)

Tower:

WaypointQueue get_instruction(Aircraft& aircraft) :

la fonction-membre va donner les instructions aux avions.

Dans le cas où l'avion, donné en argument, a l'ordre de se poser, si il est trop éloigné par rapport à la tour de contrôle, on va lui réserver une place pour se poser. Si une place a été assigné a un avion, alors l'avion va se poser, sinon l'avion va continuer a tourner autour de la tour.

Dans le cas où l'avion est proche de la tour, on va lui trouver un chemin pour qu'il se pose puis on lui enlève sa place et on lui enlève l'ordre de se poser et on retournera son ordre de départ.

void arrived_at_terminal(const Aircraft& aircraft) :

la fonction-membre va signaler si l'avion s'est posé et dans le cas où il est posé, on lui ordonne de commencer son service.

Aircraft:

const std::string& get_flight_num() const : retourne le numéro du vol de l'avion.

float distance_to(const Point3D& p) const : retourne la position entre le point p et la position de l'avion.

void display() const override : affiche le sprite du type de l'avion

void move() override : la fonction-membre va faire bouger l'avion.

Airport:

Tower& get_tower() : retourne la tour de l'aéroport

void display() const override : affiche le sprite de l'aéroport

void move() override : pour chaque avion posé, on va les faire bouger (commencer le service)

Terminal:

bool in_use() const : retourne si la place est utilisé par un avion

bool is_servicing() const: renvoie la barre de progression de la place, si elle renvoie true, c'est que la place est encore occupée, sinon la place est libre.

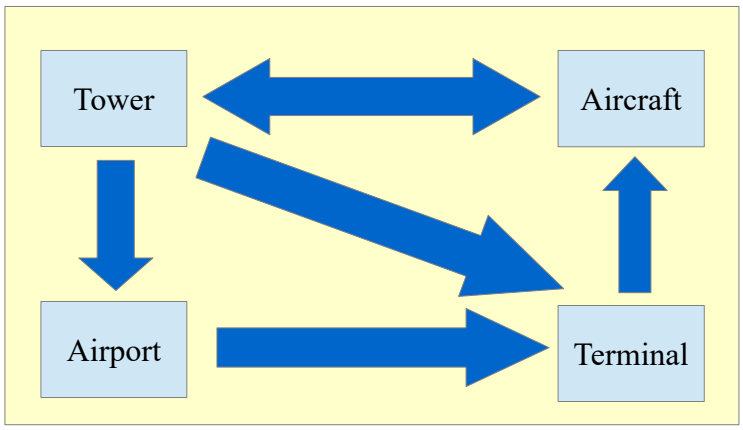
void assign_craft(const Aircraft& aircraft) : assigne un avion a la place

void start_service(const Aircraft& aircraft) : signale que l'avion débute son service et va lancer l'avion

void finish_service() : signale que l'avion a fini son service et va assigner un nouveau numéro de vol pour l'avion

void move() override : incrémente le temps que la place est occupé

Schéma UML :



Les classes impliquées dans la génération du chemin d'un avion sont Aircraft et Tower avec, respectivement, les fonctions Aircraft::move() et Tower::get_instruction(). Le conteneur std::dequeue<Waypoint> a été choisi car nous voulons plus tard rajouter des Waypoints au début et à la fin de la queue.

C- Bidouillons

1. les vitesses maximales sont définies dans aircraft_types

2. ticks_per_sec dans opendl_interface.hpp

void slowfast_tick(int tick); --> modifie le framerate

Le jeu s'arrete car il y aura une divisin par 0 dans la fonction timer ce qui va faire planter le programme. J'ai ajouté un boolean pause et la fonction pause_key() pour pouvoir changer le boolean.

3. SERVICE_CYCLE contrôle le temps de débarquement des avions.

4. L'avion pourrait être supprimé dans sa fonction move mais celle-ci ne devrait pas etre retiré dans la fonction move() car seul l'avion sait si elle est a la limite ou non. On devrait donc modifier dans la fonction timer() mais pas directement dans le foreach puisque il est interdit de modifier le contenu de la liste passé dans le foreach.

5. DynamicObject est un set donc il n'y a pas besoin de modifier celle-ci tandis que Displayable est accessible et modifiable où l'on veut si on ajoute le mot clé « inline »

D- Théorie

1. On a fait en sorte que seule la classe Tower interagit entre Terminal et Aircraft.

2. On n'est pas passé par une référence car on n'a pas l'intention de modifier la valeur passé en argument, on veut seulement ajouter la valeur à une variable.

TASK 1

Analyse de la gestion d'avion :

Pour accéder à un avion précis, il faut rechercher dans la map de Aircraft puis trouver le numéro de vol précis voulu.

Objectif 1- Référencement des avions

A- Choisir l'architecture

L'avantage d'un AircraftManager est que c'est plus lisible et propre, elle permet de mieux organiser les programmes en laissant un unique rôle pour toute les classes existantes. Cependant, il faudrait créer une ou plusieurs classe en plus, ce qui peut être embêtant car il faudrait créer plus d'objet et les maintenir fonctionnel.

B- Déterminer le propriétaire de chaque avion

1. delete(aircraft) dans la fonction timer dans opengl_interface.cpp

2. move_queue, display_queue

3. auto aircraft = *it; delete(aircraft);

4. La création et la destruction d'un aircraft doit seulement se dérouler dans la classe Aircraft, AircraftManager a uniquement pour rôle de gérer les Aircrafts.

Objectif 2 – Usine à avions

Il faut définir ContextInitializer avant la classe TowerSimulation et l'attribut context de type ContextInitializer avant l'attribut de la factory.

TASK 2

Objectif 1 – Refactorisation de l'existant

A- Structured Bindings

const auto& ks_pair a été remplacé par const auto& [x, y].

B- Algorithmes divers

1.

```
std::remove_if(aircrafts.begin(), aircrafts.end(),
        [this](std::unique_ptr<Aircraft>& aircraft) {
                try { return aircraft->move() ; }
                catch (const AircraftCrash& crash) { …..... ; return true ; }
        }) ;
```

sera contenu dans une variable et sera donné dans la fonction erase en tant que paramètre pour effacer si le prédicat a renvoyé true.

2. AircraftManager ::count_airline() et AircraftFactory ::get_airline() ont été ajouté pour compter les avions. La fonction « count_if » provenant de <algorithm> est utilisé pour compter uniquement si le prédicat donné renvoie true.

C- Relooking de Point3d

1. et 2. La fonction « std::transform » modifie chaque valeur de l'array donné en fonction de la fonction donné en 4ème paramètre en partant du 1er paramètre donné et terminant au 2ème paramètre donné.

3. La fonction « std::inner_product » ajoutera juste dans un accumulateur toute les valeurs données.

Objectif 2 – Rupture de kérosène

A- Consomation d'essence

Pour initialiser le champ fuel, on a utilisé la fonction « std::rand » pour qu'à l'initialisation de celui-ci, le carburant sera aléatoirement initialisé entre 150 et 3000. Dans le cas où le fuel tombe a 0, le champ « crash » est mis à true et la fonction renverra le type d'erreur AircraftCrash pour signaler qu'il y a un crash.

B- Un terminal s'il vous plaît

La fonction move() d'aircraft sera modifié en ajoutant une condition qui regardera si l'avion est toujours en train de voler autour de l'aéroport (si il est en attente dans les airs), si le résultat de « reserve_terminal() » n'est pas vide, on ajoutera l'avion concerné dans la queue avec std::move() car Aircraft est un unique_ptr.

C- Minimiser les crash

Le trie des avions se fera au début du lancement de AircraftManager::move(). On trie dans l'ordre croissant de fuel en fonction des avions qui ont réservé un terminal ou non.

D- Réapprovisionnement

2. J'ai utilisé std::accumulate de <numeric> pour pour parcourir les valeurs de aircrafts. La valeur commence a 0 et s'incrémente de la valeur du prédicat donné. Pour le prédicat, j'ai vérifié si l'avion est sur le sol et qu'il est bas en fuel pour savoir la quantité de fuel requis.

3. fuel_stock et ordered_fuel sont de type int car le fuel a été mis au type int. Pour que Airport ait accès à AircraftManager, je l'ai ajouté en tant que champs privé de Airport.

4. Sachant que fuel_stock ne doit pas être négatif, après avoir fait tout les calculs qu'il faut pour « remplir », j'ai décidé d'ajouter des assert pour éviter que fuel soit égale à 0 ou négatif et fuel_stock soit négatif.

5. J'ai ajouté la condition sur in_use car dans le cas où current_aircraft était nullptr, le programme se stoppais avant toute action car on essayait d'accédé à un nullptr. refill_aircraft_if_needed est appelé dans le move de Airport juste avant le déplacement de l'avion.

6. J'ai suivi l'algorithme donné dans le sujet pour mettre à jour le move de airport. std::min() a été utilisé pour prendre le minimum entre les 2 valeurs au lieu de faire une boucle if else.

TASK 3

Objectif 1 – Crash des avions

J'ai ajouté un champ privé nbCrashed initialisé à 0 et incrémenté de 1 lorsqu'il y a un crash d'avion. Cependant, étant donné que j'avais déjà utilisé AircraftCrash et tout ce qu'il va avec, le 1. et .3 n'ont pas été traité car déjà fait pendant la task 2.

La question Bonus n'a pas été traité.

Objectif 2 – Détecter les erreurs de programmation

Les assert ont été ajouté dans :

- Airport ::move sur ordered_fuel et fuel_stock
- Aircraft::refill sur fuel_stock et fuel
- AircraftManager::add_aircraft sur aircraft
- Tower::arrived_at_terminal sur &aircraft
- Tower::reserve_terminal sur &aircraft

TASK 4

Objectif 1 – Devant ou derrière

2. J'ai créé un template sur la fonction add_waypoint qui prendra un boolean constant en paramètre et add_waypoint sera applé avec false comme auparavant.

Par soucis de temps, l'objectif 2 n'a pas été traité.

Question supplémentaire :

Sur mes choix réalisé sur ce projet, certaines ont été explicitement expliqué dans les questions. Pour les raisons, c'est soit que j'ai suivi certaines indication du professeur, soit j'ai suivi mon instinct si on peut le dire. Dans le cas où il y a de meilleurs raison, c'est que je n'y avais pas pensé.

J'avais bloqué à la question C4 de la task 0 (la suppression des avions), à la question C5 de la task 0 (la suppression des sprites des avions).

Ma motivation pour ce projet étant quasi inexistante, je ne peux pas dire que j'ai aimé ce projet. Bien que l'idée du programme était marrante, le fait de partir sur un projet déjà fait mais avec des imperfections me dérange. Personnellement, je n'aime pas l'idée de partir sur un projet construit pour l'améliorer. Je ne sais pas du tout si j'ai bien compris tel fonction, si tel comportement était bon pour ce que j'avais fait, j'avais l'impression que plus j'avançais, plus le projet devenait flou. Je penses qu'un projet où nous partons d'un projet déjà existant pour l'améliorer n'est pas forcément le plus intéressant car nous voyons moins une vraie évolution du programme visuellement alors qu'en soit, le programme est plus optimisé.

A part savoir comment appliquer les outils appris en cours, je ne penses pas avoir appris des choses du projet car il y avait déjà des cours avant pour nous l'enseigner, ce projet ne faisait que appliquer la théorie que nous avons appris avant.