Compte Rendu Projet C++
RAVENDRAN Preanthy
Groupe 2

Table des matières

TASK_0

A- Exécution

Allez dans le fichier tower_sim.cpp et rechercher la fonction responsable des inputs du programme.

La méthode responsable des inputs est void TowerSimulation::create_keystrokes() const.

Sur quelle touche faut-il appuyer pour ajouter un avion ?

Pour ajouter un avion, il faut appuyer sur la touche 'C'.

Comment faire pour quitter le programme ?

Pour quitter le programme, il faut appuyer sur la touche 'X' ou ‘Q’.

A quoi sert la touche 'F' ?

La touche 'F' sert à mettre ou à enlever la fenêtre du mode plein écran.

Ajoutez un avion à la simulation et attendez.
Que est le comportement de l'avion ?

L'avion atterrit, il va sur la case blanche, attend et redécolle puis il vole en attendant d'effectuer une nouvelle fois ses opérations en boucle.

Quelles informations s'affichent dans la console ?

Dans la console, on voit le nom de l'avion qui atterrit, lorsque qu'il arrive à sa base, lorsqu'il est près à repartir et lorsqu'il a décollé.

Ajoutez maintenant quatre avions d'un coup dans la simulation.
Que fait chacun des avions ?

Lorsque l'on lance quatre avions, on voit qu'ils ne peuvent aller que trois par trois sur le plateau vu qu'il n'y a que trois terminaux. En attendant qu'il y ait de la place, il continue de tourner autour du plateau.

B- Analyse du code

Listez les classes du programme à la racine du dossier src/.
Pour chacune d'entre elle, expliquez ce qu'elle représente et son rôle dans le programme.
AircraftType → Classe représentant les différents types d’avions et leur caractéristiques.
Aircraft → Classe qui représente un avion. Il pourra être afficher et il pourra bouger.
Airport → Classe qui gère l'aéroport avec son AirportType, ses Terminals et sa Tower.
AirportType → Classe qui contient toute les information sur un aéroport.
Config → Classe qui contient toute les configurations du programme.
Geometry → Classe représentant les calculs géométriques nécessaires pour le bon fonctionnement du programme.

Runway → Classe représentant les pistes de décollage et d'atterrissage.
Terminal → Classe qui gère le terminal, stocke l'avion courant qu'il a et le supprime lorsqu'il s'en va, il peut contenir un avion à la fois maximum.
TowerSimuation → Classe qui s'occupe de simuler notre programme avec la création des aéroports, des avions…
Tower → Classe qui représente le comportement de la tour de contrôle. Gérer si un avion est atterrit, s'il peut atterrir.
Waypoint → Classe qui représente les points de cheminements. Permet de savoir si un avion est dans les airs ou dans un terminal.

Pour les classes `Tower`, `Aircaft`, `Airport` et `Terminal`, listez leurs fonctions-membre publiques et expliquez précisément à quoi elles servent.

Class Tower:

WaypointQueue get_instructions(Aircraft& aircraft);
Donne les instructions en fonctions de ou est l'avion et de ce qu'il veut faire.
void arrived_at_terminal(const Aircraft& aircraft);
Permet de lancer le service lorsque l'avion arrive sur un terminal.

Class Aircraft:

const std::string& get_flight_num() const;
Permet de récupérer le numéro de vol de l'avion.
float distance_to(const Point3D& p) const;
Permet de calculer la distance entre le point p en paramètre et la position de l'avion.
void display() const override;
Permet d'afficher l'avion.
void move() override;
Permet de s'occuper du déplacement en fonction de ce qu'il fait ou ce qu'il peut faire.

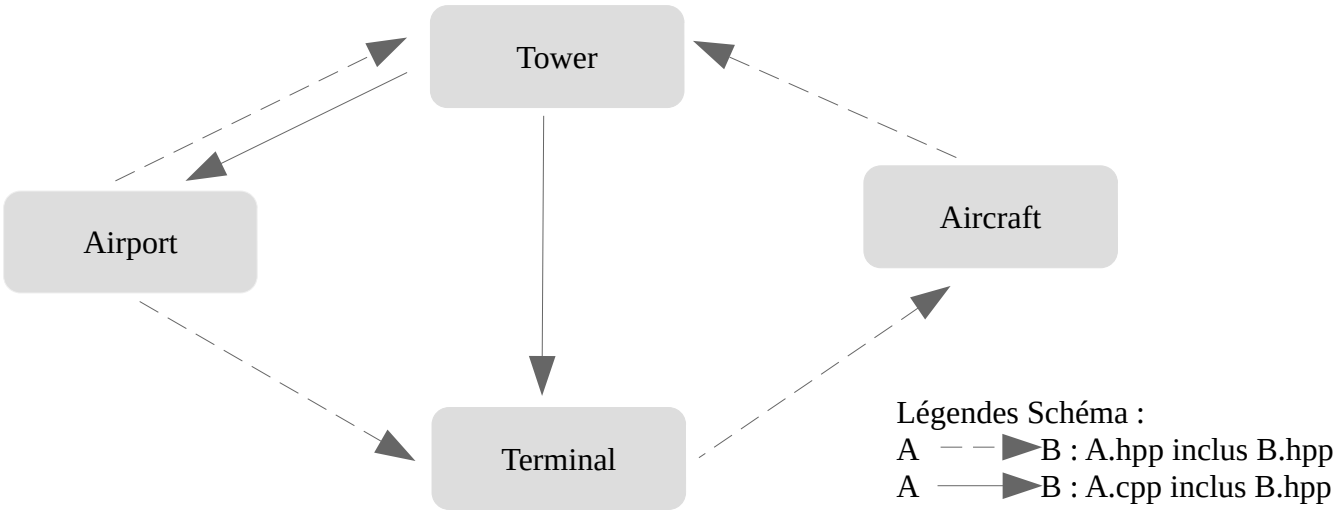
Class Airport:

Tower& get_tower();
Permet de récupérer la tour qui s'occupe de l'aéroport.
void display() const override;
Permet d'afficher l'aéroport.
void move() override;
Appel la méthode move pour chacun des terminal de l'aéroport.

Class Terminal:

bool in_use() const;
Permet de voir si il y a un avion sur le terminal ou si il est libre.
bool is_servicing() const;
Permet de savoir si un terminal est en service.
void assign_craft(const Aircraft& aircraft);
Permet d'assigner un avion à un terminal.
void start_service(const Aircraft& aircraft);
méthode qui indique le démarrage du service d'un avion et initialise la progression du service à 0.
void finish_service();
Permet de libérer a place s'il il n'y a plus d'avion dans le terminal et d'annoncer avec un message que le service est fini pour l'avion qui vient de partir.
void move() override;
Cette méthode avance le service de l'avion.

Réalisez ensuite un schéma entre ces différentes classes pour illustrer comment elles interagissent ensemble.



Graphe d'inclusions étendus entre les fichiers Tower, Aircraft, Terminal, Airport

Quelles classes et fonctions sont impliquées dans la génération du chemin d'un avion ?

Les classes impliqués dans la génération du chemin d'avion sont Aircraft, Tower et Waypoint. La méthode qui gère le chemin de l'avion est Tower::get_instructions(Aircraft& aircraft), donc toute les fonctions appelé dedans gère le déplacement d'un avion.

Quel conteneur de la librairie standard a été choisi pour représenter le chemin ? Expliquez pourquoi ce choix a été fait.

using WaypointQueue = std::deque<Waypoint>;
Le conteneur utilisé pour représenter les chemins est std::deque car les opérations utilisées sont en complexités constantes.

C- Bidouillons !

1. ***Déterminez à quel endroit du code sont définies les vitesses maximales et accélération de chaque avion. Le Concorde est censé pouvoir voler plus vite que les autres avions. Modifiez le programme pour tenir compte de cela.***

   Les vitesses maximales d'accélération de chaque avion sont défini dans **aircraft_types.hpp**
   On modifie les valeurs à l'initialisation du Concorde pour qu'il aille plus vite que les autres avions :

   ```
   aircraft_types[2] = new AircraftType {.05f, .10f, .05f, MediaPath { "concorde_af.png" } };
   ```

2. ***Identifiez quelle variable contrôle le framerate de la simulation.***

   La variable qui gère le framerate de la simulation est la variable **ticks_per_sec** de la classe opengl_interface.hpp.

Ajoutez deux nouveaux inputs au programme permettant d'augmenter ou de diminuer cette valeur. Essayez maintenant de mettre en pause le programme en manipulant ce framerate. Que se passe-t-il ?

Pour l'augmenter, il faut appuyer sur la touche 'a' et pour le diminuer sur la touche 'z'. Lorsqu'on essaie de mettre en pause le programme en manipulant ce framerate, on observe qu'il reste bloqué.

Ajoutez une nouvelle fonctionnalité au programme pour mettre le programme en pause, et qui ne passe pas par le framerate.

On ne peux pas mettre la vitesse du framerate. Pour stopper le programme, on peux utiliser un booléen qui met sur pause 'is_paused '.

3. ***Identifiez quelle variable contrôle le temps de débarquement des avions et doublez-le.***

Dans la classe config.hpp:, c'est la variable const expr unsigned int **SERVICE_CYCLES** = 40u; qui contrôle le temps.

4. ***Lorsqu'un avion a décollé, il réatterrit peu de temps après. Faites en sorte qu'à la place, il soit retiré du programme. Indices : A quel endroit pouvez-vous savoir que l'avion doit être supprimé ? Pourquoi n'est-il pas sûr de procéder au retrait de l'avion dans cette fonction ? A quel endroit de la callstack pourriez-vous le faire à la place ? Que devez-vous modifier pour transmettre l'information de la première à la seconde fonction ?***

On sait lorsque l'avion doit être supprimé grâce à la méthode move, qui sait quand on atteint le dernier Waypoint du chemin en retournant True. On modifie les fonctions qui utilise la méthode move().

5. ***Lorsqu'un objet de type Displayable est créé, il faut ajouter celui-ci manuellement dans la liste des objets à afficher. Il faut également penser à le supprimer de cette liste avant de le détruire. Faites en sorte que l'ajout et la suppression de display_queue soit "automatiquement gérée" lorsqu'un Displayable est créé ou détruit. Pourquoi n'est-il pas spécialement pertinent d'en faire de même pour DynamicObject ?***

En ajoutant le champ dispay_queue statique dans la classe Displayable, nous pouvons accéder à ce champs depuis le constructeur et le destructeur sans avoir à les définir en dehors de la classe. Il n'est donc pas spécialement pertinent d'en faire de même pour DynamicObject si le champ dispay_queue est statique.

6. ***La tour de contrôle a besoin de stocker pour tout Aircraft le Terminal qui lui est actuellement attribué, afin de pouvoir le libérer une fois que l'avion décolle. Cette information est actuellement enregistrée dans un std::vector<std::pair<const Aircraft*, size_t>> (size_t représentant l'indice du terminal). Cela fait que la recherche du terminal associé à un avion est réalisée en temps linéaire, par rapport au nombre total de terminaux. Cela n'est pas grave tant que ce nombre est petit, mais pour préparer l'avenir, on aimerait bien remplacer le vector par un conteneur qui garantira des opérations efficaces, même s'il y a beaucoup de terminaux.***
Modifiez le code afin d'utiliser un conteneur STL plus adapté. Normalement, à la fin, la fonction find_craft_and_terminal(const Aicraft&) ne devrait plus être nécessaire.

On peut utiliser une map ou une unordered map, afin de récupérer l'index du terminal de Aircraft dont l'accès est de O(1).

D- Théorie

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

   Pour que seule la classe Tower puisse réserver un terminal de l'aéroport car c'est la seule classe qui fourni les instructions nécessaires aux avions et donc de réserver des terminaux aux avions.

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

   Nous ne sommes pas passé par une référence car plusieurs avions peuvent avoir les mêmes Waypoint et que la méthode `turn` modifie les valeurs du point, on ne veut donc pas que les modifications se propagent.

3. ***Pensez-vous qu'il soit possible d'éviter la copie du Point3D passé en paramètre ?***

   La copie du Point3D n'est pas possible car la mise à jour d'un champ Point3D crée dans une classe Aircraft sera plus difficile à manipuler que de passer par une copie ce qui est plus simple.

TASK_1

Gestion mémoire

Analyse de la gestion des avions

Si à un moment quelconque du programme, vous souhaitiez accéder à l'avion ayant le numéro de vol "AF1250", que devriez-vous faire ?

Pour accéder aux avions déjà lancés dans le programme, nous devons créer une classe dans lequel nous aurons un champ qui représente la liste des avions existant.

Objectif 1 - Référencement des avions

- *A - Choisir l'architecture*
  - *Créer une nouvelle classe AircraftManager*

    Pour :
    - Les classes sont mieux organisé en ayant une classe AircraftManager uniquement consacrée à la gestion des avions. Toute classe doit posséder une seule responsabilité.

    Contre :
    - Pour l'utilisation de cette classe dans une autre, nous devons passer par une référence pour y avoir accès

  - *Donner ce rôle à une classe existante.*

    Pour :
    - Moins de classe à gérer en évitant la création d'une classe en plus.

    Contre :
    - Si l'on veut utiliser seulement la gestion d'un avion dans une autre classe en la passant en référence, la moitié de cette classe référencé sera inutile.
    - La classe sera surchargé et ne sera pas explicite.

- *B - Déterminer le propriétaire de chaque avion*

1. ***Qui est responsable de détruire les avions du programme ? (si vous ne trouvez pas, faites/continuez la question 4 dans TASK_0)***

   La classe **AircraftManager** est responsable de détruire les avions du programme. Mais aussi la classe **opengl_interface** avec la méthode **timer()** qui s'occupe de supprimer un avion en fonction de la méthode **move()**.

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

Les structures ***display_queue*** et ***move_queue*** font référence aux avions au moment de la destruction.

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

   On utilise la méthode **erase**, pour supprimer la référence sur un avion.

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

   En appliquant la même chose dans la classe AircraftManager, on risque de perdre la responsabilité unique de la classe Aircraft. On doit donc extend la classe AircraftManager dans la class Aircraft.

Objectif 2 - Usine à avions

- *A* - Création d'une factory

On implémente la classe AircraftFactory, qui permettra de gérer de aircrafts, dans tower_sim.

On a déplacé les méthodes init_aircraft_types, create_aircraft et create_random_aircraft dans AircraftFactory

- *B - Conflits*

La classe contient une liste des numéros d’avions qui vérifie si le numéro de vol existe déjà.

TASK_2

Objectifs 1 et 2

Le but de cette Task_2, a été de simplifier notre code en s'aidant des algorithmes vu en cours qui permettent d'écrire en une ligne au lieu de parcourir les éléments qui applique une fonction lambda donné en paramètre. On a eu l'utilisation de std::transform, std::removeif, std::reduce…

Cette Task permet aussi d'implémenter la gestion de l'essence, qui va permettre de gérer les avions qui peuvent ou non Crash. Nous gérons aussi la libération de l'avion s'il a déjà réservé un terminal.

TASK_3

Assertions et exceptions

Objectif 1 - Objectif 2

Dans cette Task_3 nous avons ajouter les exceptions et assertions dans différentes méthodes, surtout avec la gestion des try catch ce qui est plus efficace pour savoir quel est l'erreur rencontrés.

TASK_4

4. ***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 ?***

   Lorsque l'on instancie un Point2D avec trois arguments, le compilateur nous dit qu'il ya trop de paramètres. Cela arrive seulement maintenant car on a utilisé des alias.

5. ***Que se passe-t-il maintenant si vous essayez d'instancier un `Point3D` avec 2 arguments ? Utilisez un `static_assert` afin de vous assurez que personne ne puisse initialiser un `Point3D` avec seulement deux éléments. Faites en de même dans les fonctions `y()` et `z()`, pour vérifier que l'on ne puisse pas les appeler sur des `Point` qui n'ont pas la dimension minimale requise.***

   Si il n'y a pas assez d'arguments, le compilateur renvoie aussi une erreur.

6. ***Plutôt qu'avoir un constructeur pour chaque cas possible (d'ailleurs, vous n'avez pas traité tous les cas possibles, juste 2D et 3D), vous allez utiliser un variadic-template et du perfect-forwarding pour transférer n'importe quel nombre d'arguments de n'importe quel type directement au constructeur de `values`.***

> *Vous conserverez bien entendu le `static_assert` pour vérifier que le nombre d'arguments passés correspond bien à la dimension du `Point`.*
> *En faisant ça, vous aurez peut-être désormais des problèmes avec la copie des `Point`. Que pouvez-vous faire pour supprimer l'ambiguité ?*

Pour supprimer l'ambiguïté, on peut effectuer une assertion statique vérifiant que le nombre de paramètres reçus correspond bien à la dimension donnée pour un Point.

PROBLÈMES RENCONTRÉS

J'ai bloqué principalement sur le début du projet, j'ai mis du temps à comprendre le rôle de chaque classes ce qui m'a fait prendre du retard dès le départ. J'ai été obligé de relire chaque classes tranquillement et de bien trouver où était chaque élément.

J'ai aussi eu du mal à le lancer car le projet ne fonctionnait pas sur mon ordinateur au début.

Ensuite j'ai principalement bloqué sur la création des classes AircraftManager et AircraftFactory dans la Task 1 qui était assez longue et compliquée pour ma part. Mais aussi sur la compréhension de certaines fonctionnalités comme le template.

AVANTAGES/INCONVÉNIENTS DU PROJET

Ce projet m'aura permise de découvrir l'évolution d'un programme en C++ et également de retravailler l'ensemble des notions aborder dans les cours. Reprendre un projet déjà assez complexe a été difficile pour moi surtout pour la compréhension de chaque élément déjà écrit. Faire ce projet à partir de 0 aurait peut-être était plus simple niveau compréhension.

Ce projet m'a également aidé à mieux comprendre l'utilisation des commandes git et du GitHub, notamment les commandes concernant les branches sur lesquelles je n'avais pas beaucoup de notions particulières.