Task 0

Obectif A – Exécution

La fonction ***create_keystrokes*** est responsable de gérer les inputs du programme, il faut appuyer sur la touche “C”.

Pour quitter le programme, il faut appuyer sur la touche “q” ou “x”.

En appuyant sur la touche “C” , un nouvel avion est ajouté, l’avion s’attire directement à l’aéroport. Les informations d’atterissage, de service de l’avion au terminal, et les informations de décollage.

Si on ajoute quatre avions au même temps, le trois terminals seront réservés, le dernier avion reste en attente pour réserver un terminal en bouclant sur l’aéroport.

Obectif B- Analyse du code

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

Waypoint : qui permet la génération du chemin de l’avion

Tower : permet reserver le terminal et fournir le nouveau chemin

Aircraft : permet de stocker les chemins de l’avion

Pour stocker les chemins, on utilise un deque<Waypoint> qui nous permet une insertion et une suppression rapide des Waypoint .

Obectif C- Bidouillons

1- Les vitesses maximales et l’accélération sont définis dans aircraft_types, plus précisément dans init_aircraft_types(), qui stoque 3 types de aircraft différnent dans l’array aircraft_types.

On peut changer la vitesse du Concorde en changeant les valeurs d’initialisation

```
new AircraftType { .02f, .08f, .03f, 3 ,MediaPath { "concorde_af.png" } }
```

2- la variable ticks_per_sec dans opengl_Interface.hpp contrôle le framerate de la simulation.

Pour controller le framerate du programme, on utilise les touches “a” pour ralentir le framerate, et “z” pour augmenter le framerate.

Si on essaye de mettre en pause le programme en manipulant le framerate, le programme reste bloqué aprés qu’on le met en “pause”

3- le temps de débarquement est SERVICE_CYCLES = 40u;

4- L'endroit le plus approprié pour retirer l'avion, c'est lorsque :

- l'attérissage a déjà eu lieu => on ajoute un attribut
- l'avion a terminé sa course de décollage => waypoints.empty()

Pour déterminer si l’avion doit être modifier, la fonction DynamicObject::update renvoie un boolean, on ne peut pas supprimer l’avion dans le update, comme la fonction sera appelé dans plusieurs endroits sur différents contenaire.

On supprime l’avion plutôt dans les fonctions qui font appel à la méthode update()

5- Pour gérer l’ajout et la suppression de display_queue avec la création et la destruction d’un object Displayable, on peut rendre display_queue comme un champ static, ce qui nous permet d'accéder à display_queue dans les autres classes.

On ajoute un object Displayable dans display_queue lors de la création de l’objet, dans le distructeur de l’objet, on parcours display_queue pour supprimer l’objet de la liste

6- on peut remplacer std::vector<std::pair<const Aircraft*, size_t>> par std::map<const Aircraft*, size_t> auquel on affecte a chaque aircraft l’index de terminal, l'accès est donc en O(1)

Task 1

Gestion des ressources

Pour accéder à un avion avec un numéro de vol précis, on doit parcourir la liste des avions en comparant à chaque fois le numéro de l’avion au numéro cherché

Objectif 1 - Référencement des avions

A - Choisir l'architecture

La création d’une nouvelle classe AircraftManager nous peremt de gérer les aircraft depuis un seul endroit dans le code en donnant cette responsabilité à la classe, cepandant on doit respecter cette l’implémentation dans les différents partie du code.

Si on donne ce rôle à une classe existatne, cette classe n’aura plus une responsabilité unique

B - Déterminer le propriétaire de chaque avion

1- La fontion timer a la responsabilité de détruire les avions

2 – les deux structures GL::display_queue et GL::move_queue font référence au avions

3- Comme les aircrafts sont référencés par un unique_ptr, il faut utiliser la méthode erase

4- La classe aircraft doit dans ce cas extend AircraftManager afin de créer des conteneur de AircraftManager, on risque de perdre la responsabilité unique de la classe.

Objectif 2 - Usine à avions

A - Création d'une factory

On peut instancier la classe dans AircraftFactory dans towerSim comme on aura besoin de créer de aircraft dans cette classe

Pour refactoriser le code, on peut déplacer init_aircraft_types, create_aircraft et create_random_aircraft dans AircraftFactory.

B – Conflits

Lors de la génération de flight number, on peut vérifier si le numéro de vol existe déjà, on répéte la génération autant que le numéro est présent dans le conteneur.

Task 2

Objectif 1 - Refactorisation de l'existant

A - Structured Bindings

On peut remplacer ks_pair par la structure binding const auto& [key, value] : GL::keystrokes

B - Algorithmes divers

En remplaçant avec remove_if :

```
aircrafts.erase(std::remove_if(aircrafts.begin(), aircrafts.end(), [this](auto& aircraft){
        try{
            return !aircraft->update();
        }catch(AircraftCrash& e){
            this->_crash_conter++;
            std::cerr << e.what() <<std::endl;
            return true;
        }
), aircrafts.end());
```

2- on peut créer une méthode AircraftManager::number_aircraft_by_index(const std::string& airline) qui renvoie le nombre des avions pour un airLine spécifique

on utilise cette méthode dans TowerSimulation::create_keystrokes()

C - Relooking de Point3D

Pour les opérateur *= et +=, on utilise std::transform (cf la classe point)

Pour la méthode length, on utilise la méthode inner_product (cf la classe point)

Objectif 2 - Rupture de kérosène

A - Consommation d'essence

On ajoute le champ fuel dans Aircraft, ainsi que les valeur MAX_FUEL, INITIAL_MIN_FUEL, dans config.

Chaque frame ou l’avion n’est pas dans le terminal, la quantité de l’essance démuni.

B - Un terminal s'il vous plaît

1- has_terminal vérifie si un terminal a été réservé si le waypoints de l’avion n’est pas vide et la derniére waypoint correspond a un terminal

2- is_circling vérifie si l’avion est entrain de faire une tour sur l'aéroport, c’est le cas si l’avion n’a pas de terminal, n’est pas dans un terminal, et que le service n’est pas fait encore

3- on essaie de réserver un terminal, si ok on renvoie chemin vers ce terminal, sinon on renvoie une waypointqueue vide

4- cf Aircraft::update()

C - Minimiser les crashs

Le conteneur de AircraftManager est ordonable : vector de unique_ptr de Aircraft.

on utilise std::sort pour trier les avions

D – Réapprovisionnement

On vérifie si l’avion n’a pas assez d’essance, en vérifiant la quantité actuelle avec
200 unités

On implémente get_required_fuel qui renvoie la somme de l’essnace manquant pour tout les avions

Une fois qu’on a détermininer la quantité d’essance manquant, on commande cette quantité, sur chaque aircraft on fait appel a refill_aircraft_if_needed(int& fuel_stock)

pour restocker l’avion s’il en a besoin.

Dans le cas ou on n’a pas assez d’essance pour un avion dans le stock, on attend un certain temps next_refill_time = 100 pour la prochaine commande de stock.

E - Paramétrage (optionnel)

Pour changer la consomation selon le type d’avion, on peut simplement varier la variable “consumption” dans l’appel au constructeur de “AircraftType”.

On essaie aussi de varier la consomation en fonction de la vitesse de l’avions, de sorte que la consomation soit proportionnelle à la vitesse de l’avion

Task 3

Objectif 1 - Crash des avions

1 - Comme la gestion des aircrafts est la responsabilité de AircraftManager, l'endroit le plus approprié pour gérer les exceptions sur les avions sera dans cette classe. On ajoute donc un bloc try – catch, dans le quelle, si on détécte une erreur on affiche le message d'exception sur le cerr .

2 -Pour déterminier le nombre des crashs qui ont eu lieu, il suffit d'ajouter un compteur, dans le bloc catch, ainsi qu'un getter pour le nombre de crash actuel.

3- On remplace les messages d'erreur par des exceptions de type AircraftCrash

4. BONUS – on introduit la classe AircraftCrash qui hérite de std::runtime_error qui nous permet de nos propre message d'erreur :

Aircraft AF2620 crashed !!

Error : Aircraft AF2620 crashed at coordinates (0.263788,-0.797074,-0.000052) Info : Speed of Aircraft (0.020000) Error message : crashed due to fuel

Objectif 2 - Détecter les erreurs de programmation

On ajoute des asserts pour assurer l'initialisation des variables et que les variables ne soient pas nulle, cela nous permet de mieux sécurisier et maintenir le code .

Task 4

Objectif 1 - Devant ou derrière ?

1- on remplace get_instruction dans la classe Aircraft comme indiqué

2- Pour que l'évaluation du flag se fait lors de la compilation, il faut modifier le passage du flag en tamplate plutôt que en passage en paramétre
template <bool front>

void add_waypoint(const Waypoint& wp);

Objectif 2 - Points génériques

1- On crée une classe Point, qui rassemble le code de Point2D et Point3D

2-