RAPPORT
PROJET CPP – Tower Control

Sommaire

TASK 0 : Se familiariser avec l'existant

A- Exécution

Compilez et lancez le programme.

Allez dans le fichier `tower_sim.cpp` et recherchez la fonction responsable de gérer les inputs du programme. La méthode est create_keystrokes.
Sur quelle touche faut-il appuyer pour ajouter un avion ? La touche c.
Comment faire pour quitter le programme ? Les touches q ou x.
A quoi sert la touche 'F' ? Active/Désactive le fullscreen.

Ajoutez un avion à la simulation et attendez.
Que est le comportement de l'avion ? animation d'atterrissage puis attente puis animation de décollage: il atterrit, il rend un service, le termine et décolle.
Quelles informations s'affichent dans la console ?
On peut récupérer les actions précédé par l'ID de l'avion :
DL9742 is now landing...
now servicing DL9742...
done servicing DL9742
DL9742 lift off

Ajoutez maintenant quatre avions d'un coup dans la simulation.
Que fait chacun des avions ? Ils tournent en attendant de pouvoir atterrir via un terminal. (max 3 dans l'aéroport). Puis redécollent pour attendre d'atterrir.

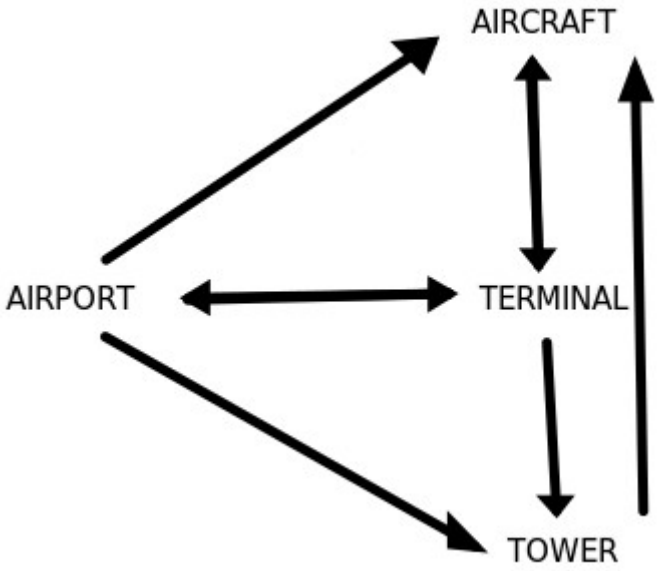
B- Analyse du code

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.
Commencant avec le repo GL/. Nous avons donc :
- Displayable - c'est une classe abstrait qui forme la base pour tout les choses qui peuvent être dessiné sur l'ecran. La classe contient une coordonné z qui permet de trier les objets de cette classe.
- DynamicObjject - une autre classe abstrait qui forme la base pour tout les choses qui peuvent "bouger" (en sense large, par example, un Terminal est aus si un DynamicObject, son movement est le débarquement des avions) opengl-interface - pas de classe ici, juste quelques fonctions necessaires pour interagir avec OpenGL; on remarque, par contre, la fonction timer dans laquelle tout les objets dans la move_queue ont leur move() appelé!
- Texture2D - une texture qui contient un pointeur vers un img::Image (qui contient les octets vrac de l'image); la texture peut être affichée avec Texture2D::draw Dans img/ on trouve les suivantes: -
- Image - une classe qui gère des octets d'un image dans la memoire pour être utilisé avec Texture2D
- MediaPath - une classe qui gère l'acces aux PNGs qui vont avec le code stb_image - pas de classe ici, c'est en fait une bibliotheque C qui sait lire les PNGs correctement
Finalement, les classes important pour la logique du projet:
- Point2D / Point3D - classes qui gèrent des maths entre points dans l'espace 2D et 3D Waypoint - un point sur un chemin d'un avion, c'est juste un Point3D avec l'information si ce point se trouve au

sol, chez un terminal ou dans l'air; ici, on voit aussi qu'un "chemin" (WaypointQueue) est un deque de Waypoints.
- Runway - stocke le debut et le fin d'un piste de décollage Terminal - classe qui gère le débarquement d'un avion; chaque Terminal peut débarquer qu'un seul avion à la fois
- Aircraft - un avion qui peut (1) être déssiner (Displayable) et (2) bouger (DynamicObject); chaque avion peut retourné son "flight number" ansi que sa distance à un point donné
- AircraftType - le type d'un avion stocke des limites de vitesse et acceleration ansi que la texture; il y a 3 types pré-definit
- Airport - gère l'aeroport, contient les terminals et le tower; seulment son friend class Tower peut réserver des terminals et démander un chemin de décollage
- AirportType - contient les coordonnés importants (relatives au centre de chaque aeroport) comme le debut/fin des runways (il peut en avoir plusieurs); chaque AirportType peut génerer des chemins pour atterir et pour décoller
- Tower - classe qui gère la fonctionalité du tour de control; des avions peuvent démander des nouvelles instructions ansi qu'indiquer qu'ils sont arrivés à leur Terminal; Chaque Tower contient une affectation des avions aux terminals. Si un avion X démande d'atterir à un moment quand tout les Terminals de l'aeroport sont affectés, alors le Tower retourne un "cercle" autour de l'aeroport pour que X re-démande quand il a fini son cercle
- 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, etc config - pas de classe ici, mais des constantes qui determinent quelques comportements de la simulation, par example le nombre d'intervals necessaire pour débarquer un avion
- Tower : get_instructions : l'avion récupère les instructions de la tour. arrived_at_terminal : défini si l'avion est arrivé au terminal.
- Aircraft : turn_to_waypoint : se dirige vers le point défini turn : tourne l'avion get_speed_octant : donne la vitesse pendant un huitième de cercle arrive_at_terminal : défini si l'avion est au terminal operate_landing_gears : amorce l'atterissage add_waypoint : ajoute un waypoint move : bouge l'avion display : dessine l'avion dans la fenêtre
- Airport : get_tower : donne la tour de l'aéroport display : affiche la tour dans la fenêtre move : fait bouger les avions dans le terminal
- Terminal : in_use : défini si le terminal est utilisé is_servicing : défini si l'avion est en cours de débarquement assign_craft : associe un avion à un terminal finish_service : défini si l'avion a fini son débarquement move : fais bouger l'avion sur le terminal.

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

Tower -> get_instructions : donne la prochaine instruction à un avion
arrived_at_terminal : signale qu'un avion a attérit
Aircraft -> get_flight_num : récupère le numéro de vol
distance_to : donne la distance entre l'avion et un point
display : affiche un avion
move : déplace l'avion
Airport -> get_tower : récupère la tour de l'aéroport
display : affiche l'aéroport
move : déplace tout les terminaux de l'aéroport
Terminal -> in_use : signal si le terminal est utilisé
is_servicing : signal so le terminal produit un service
assign_craft : assigne un avion
start_service : démarre un service (si avion assez prêt)
finish_service : arrête un service en cour
move : +1 aux progrès de tout les services



Quelles classes et fonctions sont impliquées dans la génération du chemin d'un avion ? la classe tower avec les méthodes get_circle et get_instructions.
Quel conteneur de la librairie standard a été choisi pour représenter le chemin ? Dequeue de waypoint
Expliquez les intérêts de ce choix. Facile d'obtenir le premier éléments et vu que l'on va de point en point, bah c'est bien

C- Bidouillons !

1) **Déterminez à quel endroit du code sont définies les vitesses maximales et accélération de chaque avion.** Dans la classe aircraft_types
Le Concorde est censé pouvoir voler plus vite que les autres avions.
Modifiez le programme pour tenir compte de cela.
aircraft_types[2] = new AircraftType { .02f, .05f * 2., .02f, MediaPath { "concorde_af.png" } };

2) **Identifiez quelle variable contrôle le framerate de la simulation.** Dans le fichier de config :
constexpr unsigned int DEFAULT_TICKS_PER_SEC = 16u;
Ajoutez deux nouveaux inputs au programme permettant d'augmenter ou de diminuer cette valeur.
GL::keystrokes.emplace('$', []() { GL::change_ticks(1); });
GL::keystrokes.emplace('*', []() { GL::change_ticks(-1); });
et dans opengl_interface on créer la fonction changeant la variable de tick_speed : inline unsigned int ticks_per_sec = DEFAULT_TICKS_PER_SEC;
(on empêche de passer en dessous de 0)

```
void change_ticks(int newTick)
{
    if (ticks_per_sec + newTick > 120 || ticks_per_sec + newTick < 1)
    {
        return;
    }
    ticks_per_sec += newTick;
}
```

Essayez maintenant de mettre en pause le programme en manipulant ce framerate. Que se passe-t-il ? Fixez le problème.
Fallait pas passer dans les négatifs (ou même 0)... mais du coup j'ai déjà fait !

3) **Identifiez quelle variable contrôle le temps de débarquement des avions et doublez-le.**
Il s'agit de cette variable : constexpr unsigned int SERVICE_CYCLES = 20u;

4) **Lorsqu'un avion décolle, celui-ci n'est pas retiré du programme.**
Faites en sorte qu'il le soit.
Dans aircraft il faut supprimer l'avion des listes (move_queue et display_queue).

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.
Que pourriez-vous faire afin que l'ajout et la suppression de la liste soit "automatiquement gérée" lorsqu'un `Displayable` est créé ou détruit ?
Faites de même pour `DynamicObject`.
Il faut ajouter le code dans les constructeur/destructeur des classes `Displayable`.

Pensez-vous qu'il soit pertinent d'en faire de même pour `DynamicObject` ?

D- Théorie

1) **Comment a-t-on fait pour que seule la classe `Tower` puisse réserver un terminal de l'aéroport ?**
Toutes les fonctions de airport sont private (donc inutilisable par les autres classes), mais : friend class Tower;

2) **En regardant le contenu de la fonction `void Aircraft::turn(Point3D direction)`, pourquoi selon-vous ne sommes-nous pas passer par une réference ?** pour pas modifier le paramètre direction (et vu que direction est modifié dans la fonction donc on peut pas passer const).
Pensez-vous qu'il soit possible d'éviter la copie du `Point3D` passé en paramètre ? oui on peut mais c'est mal, on ne pourra pas mettre le paramètre const (vu qu'on appelle des fonctions de la classe), donc juste par référence ainsi, on peut appeler turn comme cela :
target = target - pos - speed;
turn (target);

TASK 1 : Gestion des ressources

**La création des avions est aujourd'hui gérée par les fonctions `TowerSimulation::create_aircraft` et `TowerSimulation::create_random_aircraft`.
Chaque avion créé est ensuite placé dans les files `GL::display_queue` et `GL::move_queue`.**

Si à un moment quelconque du programme, vous souhaitiez accéder à l'avion ayant le numéro de vol "AF1250", que devriez-vous faire ? Je suppose que l'on doit parcourir l'une des 2 files précédentes pour trouver "AF1250".

Objectif 1 - Référencement des avions

A - Choisir l'architecture

Pour trouver un avion particulier dans le programme, ce serait pratique d'avoir une classe qui référence tous les avions et qui peut donc nous renvoyer celui qui nous intéresse.

Vous avez 2 choix possibles :

- créer une nouvelle classe, AircraftManager, qui assumera ce rôle, pros : laisse l'architecture simple (et c'est simple à implémenter), cons : ajouter une nouvelle classe (plus lourd et long)
- donner ce rôle à une classe existante. pros : c'est rapide, cons : complexifie le programme en ajoutant un lien d'appartenance, structurellement c'est pas ouf

Réfléchissez aux pour et contre de chacune de ces options.

Pour le restant de l'exercice, vous partirez sur le premier choix.

B - Déterminer le propriétaire de chaque avion

**Vous allez introduire une nouvelle liste de références sur les avions du programme.
Il serait donc bon de savoir qui est censé détruire les avions du programme, afin de déterminer comment vous allez pouvoir mettre à jour votre gestionnaire d'avions lorsque l'un d'entre eux disparaît.**

Répondez aux questions suivantes :
1. Qui est responsable de détruire les avions du programme ? (si vous ne trouvez pas, faites/continuez la question 4 dans TASK_0) la fonction timer de opengl_interface.
2. Quelles autres structures contiennent une référence sur un avion au moment où il doit être détruit ? tower & terminal & les queue
3. Comment fait-on pour supprimer la référence sur un avion qui va être détruit dans ces structures ? dans tower et terminal l'avion n'existe plus, et move & dislpay queue : on retire des conteneurs (avec timer enlevant de la move_queue)
4. Pourquoi n'est-il pas très judicieux d'essayer d'appliquer la même chose pour votre `AircraftManager` ? car c'est le owner (responsable de destruction donc il doit se débrouiller)

Pour simplifier le problème, vous allez déplacer l'ownership des avions dans la classe `AircraftManager`.
Vous allez également faire en sorte que ce soit cette classe qui s'occupe de déplacer les avions, et non plus la fonction `timer`.
Il va falloir utiliser un unique_ptr.

Objectif 2 - Usine à avions

A - Création d'une factory

La création des avions est faite à partir des composants suivants :
- `create_aircraft`
- `create_random_aircraft`
- `airlines`
- `aircraft_types`.

Pour éviter l'usage de variables globales, vous allez créer une classe `AircraftFactory` dont le rôle est de créer des avions.

Définissez cette classe, instanciez-la à l'endroit qui vous paraît le plus approprié, et refactorisez le code pour l'utiliser.
Vous devriez du coup pouvoir supprimer les variables globales `airlines` et `aircraft_types`.

On créé une classe factory où l'on recopie les fonctions create_aircraft et create_random_aircraft. On aura aussi besoin d'un getter pour rettrouver le numéro de ligne d'un avion vu que la variable globale airlines devient un membre privé de la factory.
Dans tower_sim on ajoute maintenant un champ de type AircraftFactory ainsi l'on peut créer des avions avec. On garde les fonctions membres create_aircraft et create_random_aircraft (de tower_sim) pour faire appel à la factory.

B - Conflits

Il est rare, mais possible, que deux avions soient créés avec le même numéro de vol.
Ajoutez un conteneur dans votre classe `AircraftFactory` contenant tous les numéros de vol déjà utilisés.
Faites maintenant en sorte qu'il ne soit plus possible de créer deux fois un avion avec le même numéro de vol.

On ajoute un champ current_airlines, de type unordered_set de string, ainsi l'on se protège d'un potentiel doublons.

TASK2 : Algorithmes

Objectif 1 - Refactorisation de l'existant

A - Structured Bindings

`TowerSimulation::display_help()` est chargé de l'affichage des touches disponibles.
Dans sa boucle, remplacez `const auto& ks_pair` par un structured binding adapté.
Cela donne : for (const auto& [key_pair, value_pair] : GL::keystrokes) { … }

B - Algorithmes divers

1. `AircraftManager::move()` (ou bien `update()`) supprime les avions de la `move_queue` dès qu'ils sont "hors jeux".
En pratique, il y a des opportunités pour des pièges ici. Pour les éviter, `<algorithm>` met à disposition la fonction `std::remove_if`.
Remplacez votre boucle avec un appel à `std::remove_if`.
On utilise la fonction erase, on utilise ensuite un try catch pour tester si un avion c'est crashé ou non.

2. Pour des raisons de statistiques, on aimerait bien être capable de compter tous les avions de chaque airline. A cette fin, rajoutez des callbakcs sur les touches `0`..`7` de manière à ce que le nombre d'avions appartenant à `airlines[x]` soit affiché en appuyant sur `x`.
Rendez-vous compte de quelle classe peut acquérir cet information. Utilisez la bonne fonction de `<algorithm>` pour obtenir le résultat.
On procède ainsi : pour tout les caractères de 0 à 8, on ajoute les callbacks correspondants en affichant via une lambda le nombre d'avions sur l'airline correspondante (en appelant une fonction qui compte dans la factory).

C - Relooking de Point3D

La classe `Point3D` présente beaucoup d'opportunités d'appliquer des algorithmes.
Particulairement, des formulations de type `x() = ...; y() = ...; z() = ...;` se remplacent par un seul appel à la bonne fonction de la librairie standard.
Remplacez le tableau `Point3D::values` par un `std::array` et puis,
remplacez le code des fonctions suivantes en utilisant des fonctions de `<algorithm>` / `<numeric>`:

1. `Point3D::operator*=(const float scalar)` : on utilise transform
2. `Point3D::operator+=(const Point3D& other)` et `Point3D::operator-=(const Point3D& other)` : on utilise transform
3. `Point3D::length() const` : on utilise reduce

Objectif 2 - Rupture de kérosène

**Vous allez introduire la gestion de l'essence dans votre simulation.\
Comme le but de ce TP est de vous apprendre à manipuler les algorithmes de la STL, avant d'écrire une boucle, demandez-vous du coup s'il n'existe pas une fonction d'`<algorithm>` ou de `<numeric>` qui permet de faire la même chose.**

La notation tiendra compte de votre utilisation judicieuse de la librairie standard.

A - Consommation d'essence

Ajoutez un attribut `fuel` à `Aircraft`, et initialisez-le à la création de chaque avion avec une valeur aléatoire comprise entre `150` et `3'000`. On utilise la fonction rand de la lib std, avec le calcul suivant : %2851 + 150
**Décrémentez cette valeur dans `Aircraft::update` si l'avion est en vol.\
Lorsque cette valeur atteint 0, affichez un message dans la console pour indiquer le crash, et faites en sorte que l'avion soit supprimé du manager.** Dans la fonction update d'Aircraft, on doit throw un AircraftCrash (avec un message adéquat).

N'hésitez pas à adapter la borne `150` - `3'000`, de manière à ce que des avions se crashent de temps en temps.

B - Un terminal s'il vous plaît

**Afin de minimiser les crashs, il va falloir changer la stratégie d'assignation des terminaux aux avions.
Actuellement, chaque avion interroge la tour de contrôle pour réserver un terminal dès qu'il atteint son dernier `Waypoint`.
Si un terminal est libre, la tour lui donne le chemin pour l'atteindre, sinon, elle lui demande de tourner autour de l'aéroport.
Pour pouvoir prioriser les avions avec moins d'essence, il faudrait déjà que les avions tentent de réserver un terminal tant qu'ils n'en n'ont pas (au lieu de ne demander que lorsqu'ils ont terminé leur petit tour).**

1. Introduisez une fonction `bool Aircraft::has_terminal() const` qui indique si un terminal a déjà été réservé pour l'avion (vous pouvez vous servir du type de `waypoints.back()`).
On utilise la fonction is_at_terminal() de waypoint pour savoir s'il est assigné ou non.
Pour circling on utilise has_terminal, le service (fini?) et si l'aircraft est au sol ou non, si non pour tout c'est qu'on doit faire encore un tour.

2. Ajoutez une fonction `bool Aircraft::is_circling() const` qui indique si l'avion attend qu'on lui assigne un terminal pour pouvoir atterir.
3. Introduisez une fonction `WaypointQueue Tower::reserve_terminal(Aircraft& aircraft)` qui essaye de réserver un `Terminal`. Si c'est possible, alors elle retourne un chemin vers ce `Terminal`, et un chemin vide autrement (vous pouvez vous inspirer / réutiliser le code de `Tower::get_instructions`).
Si un avion n'a pas de terminal on l'assigne de la même manière que la fonction get_instruction.

4. Modifiez la fonction `move()` (ou bien `update()`) de `Aircraft` afin qu'elle appelle `Tower::reserve_terminal` si l'avion est en attente. Si vous ne voyez pas comment faire, vous pouvez essayer d'implémenter ces instructions :
\- si l'avion a terminé son service et sa course, alors on le supprime de l'aéroport (comme avant),
\- si l'avion attend qu'on lui assigne un terminal, on appelle `Tower::reserve_terminal` et on modifie ses `waypoints` si le terminal a effectivement pu être réservé,
\- si l'avion a terminé sa course actuelle, on appelle `Tower::get_instructions` (comme avant).
On se place la où l'on traite le fuel. Si l'avion n'a plus de fuel et était assigné, on doit désassigner le terminal (j'ai donc ajouté une fonction dans tower et dans terminal pour réalisé cette suppression). Ensuite, si l'avion fait des cercles, on va essayé de lui réserver un terminal.

C - Minimiser les crashs

Pas fait par manque de temps.

D - Réapprovisionnement

Pas fait par manque de temps.

TASK 3 : Assertions et exceptions

Objectif 1 - Crash des avions

Actuellement, quand un avion s'écrase, une exception de type `AircraftCrash` (qui est un alias de `std::runtime_error` déclaré dans `config.hpp`) est lancée.

1. Faites en sorte que le programme puisse continuer de s'exécuter après le crash d'un avion. Pour cela, remontez l'erreur jusqu'à un endroit approprié pour procéder à la suppression de cet avion (assurez-vous bien que plus personne ne référence l'avion une fois l'exception traitée). Vous afficherez également le message d'erreur de l'exception dans `cerr`.
Pour cela, il faut modifier dans l'aircraft_manager la fonction d'update, une update d'aircraft doit maintenant être faite dans un try et on doit catch une erreur de type AircraftCrash et afficher son contenue (err.what()).

2. Introduisez un compteur qui est incrémenté chaque fois qu'un avion s'écrase. Choisissez une touche du clavier qui n'a pas encore été utilisée (`m` par exemple ?) et affichez ce nombre dans la console lorsque l'utilisateur appuie dessus.
On ajoute un attribut public au manager que l'on affiche si l'on appui sur 'm'.

3. Si vous avez fini d'implémenter la gestion du kérosène (Task_2 - Objectif 2 - A), lancez une exception de type `AircraftCrash` lorsqu'un avion tombe à court d'esssence. Normalement, cette exception devrait être traitée de la même manière que lorsqu'un avion s'écrase parce qu'il a attérit trop vite.
Ça devrait fonctionner correctement.

Objectif 2 - Détecter les erreurs de programmation

Pour sécuriser votre code, repassez sur les différentes fonctions de votre programme et ajoutez des assertions permettant de vérifier qu'elles sont correctement utilisées.
Voici quelques idées :
- fonctions d'initialisation appelées une seule fois
- état attendu d'un objet lorsqu'une fonction est appelée dessus
- vérification de certains paramètres de fonctions

Dans la classe tower_sim, lorsque l'on créé (aléatoirement ou non) des avions, on peu tester si l'airport n'est pas null.
On peut tester si l'image donné à la construction d'un airport n'est pas nulle.
Lorsque l'on veut récupérer le numéro de ligne, on peut se protéger car on n'a que 8 lignes.
On peut s'assurer que les fonctions de création d'avions de l'usine a avion ne renvoie pas d'avion nulls.
On peut checker si la fonction d'initialisation de l'airport se fait bien qu'une fois.

TASK 4 : Templates

Objectif 1 - Devant ou derrière ?

La fonction `Aircraft::add_waypoint` permet de rajouter une étape au début ou à la fin du parcours de l'avion.
Pour distinguer ces deux cas, elle prend un argument booléen `front` (on parle alors de "flag") qui est évalué à l'exécution.
Votre objectif consistera à modifier cette fonction afin d'économiser cette évaluation.

2. Modifiez `Aircraft::add_waypoint` afin que l'évaluation du flag ait lieu à la compilation et non à l'exécution.
On met tout le corps de la fonction dans le hpp.
On créé un template<bool Front>, et on écrit un if constexpr (Front). On a besoin maintenant que de l'argument Waypoint.
Que devez-vous changer dans l'appel de la fonction pour que le programme compile ?
L'appel nécessite maintenant que la variable front doit être constexpr et l'appel ressemble à : add_waypoint<front>(wp);

Objectif 2 - Points génériques

1. Reprenez les classes dans `geometry.hpp` et inspirez-vous de `Point2D` et `Point3D` pour définir une unique classe-template `Point` paramétrée par la dimension (nombre de coordonnées) et leur type (entier/float/double).
Pour le moment, ajoutez simplement un constructeur par défaut à votre classe.
Notre classe point sera templaté par un type Type (entier, flottant…) et une taille Size. Le constructeur ressemble simplement à : Point() = default ;
On écrit ensuite toutes les fonctions nécessaires :
- des fonctions d'accès avec operator[] et x(), y() et z()
- des operator x= (+, -, *) et on utilise l'algorithme transform pour le faire proprement
- des operator x (+, -, *) utilisant les x= correspondant
- on recopie les fonctions length, normalize, distance_to et cap_length.

3. Ajoutez le constructeur à 2 paramètres de `Point2D` et le constructeur à 3 paramètres de `Point3D` dans votre classe-template.
Modifiez `Point2D` et `Point3D` afin d'en faire des alias sur des classes générées à partir du template `Point` (respectivement, 2 floats et 3 floats).
Vérifiez que votre programme compile et fonctionne comme avant.
On ajoute 2 allias correspondants au Point2D et 3D :
using Point2D = Point<float, 2> et using Point3D = Point<float, 3>.
Puis on ajoute les 2 constructeurs à 2 et 3 paramètres de manière classique.

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 ?
Ça compile pas !!! On essai de construire un tableau de taille 2 avec 3 éléments. Cette erreur n'apparaît que maintenant car on a nommé les Point2D et Point3D avec une taille fixe donc on ne peu plus faire n'importe quoi, le compilateur nous vois.

5. Que se passe-t-il maintenant si vous essayez d'instancier un `Point3D` avec 2 arguments ?
Ce coup ci ça compile cependant la troisième valeur vaut toujours 0 car elle n'est pas initialisée.
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.
On ajoute les static_assert des constructeurs en regardant si la taille de la struct Point équivaut au nombre d'arguments du constructeur concerné.
Puis les static_assert des y() et z(), on regarde si la taille de la struct Point est suffisante pour faire appel à ces fonctions.

NOTES

Situations bloquantes

J’ai eu des problèmes avec deux bugs en particulier que j’ai rencontré :
- le premier était que mes avions ne se supprimaient pas correctement, le problème venait de la fonction d’update du manager dont je n’avais apparemment pas compris le but au premier abord...
- le second était que mes avions se crashaient en permanence, et l’erreur venait de geometry dans le calcul de distance entre 2 points où j’utilisais mal reduce (il faut d’abord transform toutes les valeurs puis reduce).

Après à part quelque segfault toujours un peu contraignant à corriger ça à été.

Aimé / Détesté

Aimé :

- être diriger tout le long du projet avec les TASK qui se suivent
- le sujet est très fun
- la diversité entre chaque TASK (c’est varié et on fait bien le tour du cours)
- j’adore le langage, il est dur à prendre en main (beaucoup de syntaxes et ça devient vite moche dans les fichiers) mais une fois qu’on prend un peu l’habitude c’est vraiment très plaisant d’écrire du code (propre qui plus est)

Détesté :

- le fait de ne pas avoir développer from scratch, j’aime moins le fait de devoir corriger/débugger/améliorer du code
- je suis débutant en cpp, alors la lecture d’un code n’étant pas le miens dans ce langage rend la difficulté assez importante (et en y ajoutant les TD dans le même format ainsi que les cours non magistraux mais juste en ligne ça n’aide pas)

Ce que j’ai appris

Je pense que j’ai appris à me servir basiquement du c++ (dans les grandes lignes et pas toujours de manière jolie et propre). J’ai néanmoins compris les bases apportées par le cours de ce semestre et avec de l’entraînement, elles deviendront des automatismes que je pourrai exploité aisément.

Maintenant, même si je n’ai pas trop aimé ce point là, j’ai appris à corriger du code, à l’améliorer et trouver des solutions à des problèmes sur de l’existant, chose que l’on avait jamais fait concrètement.