TASK 0 :

A - Execution :

il faut appuyer 'c', pour quitter le programme 'x' et 'q', le touche 'f' permet de mettre en grand écran.
L'avion, apparait, descend sur l'aéroport, se gare, et puis part et après il revient, atteri et recommence.
Du coup pour chaque action, ils affichent se qu'il se passe avec le nom de l'avion. Ils disent quand il commence son service, quand il le finis, quand il atteris (is now landing), et quand il decolle (lift off)
On remarque qu'il y a que 3 places sur l'aéroport, donc impossible que + de 3 avions atteris. Donc le 4eme atterit pas maintenant car il y a plus de place, il atterrira après. Il y a des avions qui disparaissent des fois de la fenetre.

B - Analyse de code :

AirCraft : represente les avions.
const std::string& get_flight_num() const : renvoit juste le numero de l'avion
float distance_to(const Point3D& p) const : retourne la distance entre l'avion et le point en parametre.
void display() const override : affiche l'avion
void move() override : s'occupe des mouvements de l'avion en fonction de ou il se situe, si il est sur l'aérport, dans un terminal ou dans les airs.

AirCraftType : represente les caracteristiques d'un avion

AirPort : represente les aéroports.
Tower& get_tower() : renvoi la tour de controle lié à l'aéroport
void display() const override : affiche l'aéroport
void move() override : pour chaque terminal occupé (dans la liste), on va move appelé move sur chaque terminal.

AirPortType : represente les caracteristiques d'un avion

Point2D : represente un Point en 2D avec 2 coordonnées

Point3D : represente un Point en 3D avec les 3 coordonnées

Runway : represente la piste (position de celle en fonction de la position de l'aéroport).

Terminal : represente le terminal, il voit les différents avions qui vienne et partent (debut de service et fin de service).
bool in_use() const : regarde si l'avion est utilisé (donc l'avion utilisé n'est pas nulle)
bool is_servicing() const : regarde si l'avion est en service, tant que son service_progress est plus petit que le SERVICE_CYCLES.
void assign_craft(const Aircraft& aircraft) : l'avion courant representera l'avion en parametre.
void start_service(const Aircraft& aircraft) : commence le service pour un avion (avion vole)
void finish_service() : finis le service pour un avion (avions atteris et se gare)
void move() override : augmente le service_progress

Tower : represente la tour de controle, celle qui va donner les instructions aux avions.
WaypointQueue Tower::get_instructions(Aircraft& aircraft) : va donner les instructions à l'avion en fonction de ou il est, il va voir si l'avion n'est pas deja dans le terminal, puis il va regarder sa distance avec l'aérport, si elle est plus petite que 5, elle va regarder si il y a de la place dans le terminale, puis ajouter ou pas un nouvelle avion, sinon il va voler (faire un cercle dnas

les airs).
void Tower::arrived_at_terminal(const Aircraft& aircraft) : va juste prendre l'avion et dire qu'il est arrivé au terminal, et le faire repartir avec start_service.

TowerSimulation : Simule une animation avec l'aéroport et les avions.

WaypointType : represente un enum qui possède des caracteritiques qui represente ou est l'avion, dans les airs, dans un des terminaux ou sur le sol de l'&éroport.

Waypoint : rergarde en fonction des points ou est l'avion en fonction de sa position.

Tower travaille sur les aircraft avec aircraft get_instruction et arrived_at_terminal mais prend aussi un Airport en parametre qu'en en crée un Tower et sur les terminaux
Aircraft possède un Tower& qui represente sa tour de controle auquel il est associé, qui est remplis quand on crée un Aircraft avec son contructeur
Airport possède une fonction qui crée des terminaux
Terminal travail sur les Aircraft, et possède un champ qui represente l'avion qui est garé dans un terminal. On a aussi quelques fonctions comme in_use, assign_aircraft, start_service
et finish_service qui travaille sur le current aircfrat ou assigne un avion à celui ci.

Les classes et fonctions qui sont impliquées dans la génération du chemin d'un avion sont :
tower :
    WaypointQueue get_instructions(Aircraft& aircraft)

aircraft:
    void move()

Le conteneur de la librairie standard qui a été choisi pour représenter le chemin est std::deque<Waypoint> pour le conteneur, on a choisi deque car pour une queue, on aurait pu utiliser std::queue, mais on a choisi std::deque parce qu'on veut rajouter des Waypoints à la fin et au debut de la queue.

C - Bidouillons

1. Les vitesses maximales et accélarations sont définies dans la classe AircraftType avec le max_air_speed qui represente la vitesse de l'avion dans l'air, max_accel qui represente l'acceleration maximum d'un avion et on a aussi max_ground_speed qui represente la vitesse sur l'aéroport. On a 3 types d'avion, l'avion concorde est la plus rapide donc on change sa vitesse, je rend l'avion concorde 2 fois plus rapide que les autres.

2. la variable est ticks_per_sec, dans la classe GL dans le dossier GL. On remarque que si on met le framerate à 0, le programme s'arrete. Pour mettre pause, on fais un booléen et fais un if/else dessus, s'il il est vrai, on arrete le programme sinon on le continue. J'ai juste chercher dans le timer dans la classe GL, qui enfait faisait la boucle pour pas que le programme s'arrete avec une recursion, faites avec l'appel de timer dans glutTimerFunc qui est une fonction de glut qui appel un timer, du temps (framerate) et une valeur. D'un autre coté il appel les différents "move" qu'il y a dans les autres classes et glutPostRedisplay qui affiche en continue le programme.

3. Après recherche, j'ai remarqué que c'était la variable SERVICE_CYCLES qui fait en sorte que les avions restent dans le terminal plus longtemps, donc on a juste à le multiplier par 2.

4. On va faire un booléen dans la classe Aircraft qui va etre faux et passera à vrai avant de s'envoler une deuxieme fois. Puis on checkera dans timer si le booléen du dynamic object renvoi true, on detruit l'object et l'enlève de la move_queue
sinon on continue, pour pouvoir récupérer le booléen, on crée une fonction virtual dand DynamicObject qui renvoit un booléen (le getter du booléen), on le met à false par défaut et dans la classe Aircraft on va faire
un getter avec un override qui lui changera la valeur du booléen. Dans le move de Aircrfat on fera un return quand tout les points que l'avion doit parcourir sont parcourus et que le booléen est vraie pour l'arreter.

5. Ce n'est pas très pertinent de faire la meme pour le set de DynamicObject car celle ci est un set tout simplement, donc on va changer l'emplacement de notre vector de Displayable, et le mettre en static inline dans
notre classe Displayable pour en avoir accès partout et la modifier dans les endroits ou on en a besoin. Maintenant on peut en avoir accès dans notre constructeur et destructeur donc on va pouvoir du coup ajouter les
éléments directement dedans avec emplace_back de this sur le vector mais pour le suprrimer (erase), c'est plus technique, il faut erase avec un find qui cherche l'élément à retirer et erase le retire
(on a vu ca dans le cours 6 de cette fin de semaine (20/02)).

6. Uiliser une map, l'avion en tant que clés car on veut récupérer dans quel terminal on à l'avion. Ca se resume à dire qu'on trouve le terminal à partir de l'avion.

D - Théorie

1. On utilise une map privé dans la classe tower qui s'occupera directement d'associer des avions et des terminaux, vu qu'elle est private, seule Tower peut l'utiliser.

2. Parce qu'on ne veut pas que la direction change au cours du programme, donc on ne passe pas par référence pour pas avoir de soucis de modification. En plus la fonction est privé de plus pour éviter les incohérences et
effet indésirables.