C++

M1 Informatique

Tower Control Simulation : Rapport

Luis Daniel Medina Zuluaga

Avril 2022

1 Les questions non-code

Les réponses aux questions « non-code » (et aussi à celles de « code ») se trouvent dans les fichiers `Task_0.md`, `Task_1.md`, `Task_2.md`, `Task_3.md` et `Task_4.md`.

2 Les choix d'architecture du programme

Étant donné que le point de départ était un projet fonctionnel, et que les questions posées dans les « *Task* » nous dictaient la façon dont on devait procéder en termes d'architecture ou amenaient à des changements d'implémentation sans introduire des besoins implicites de changement sctructurel, je trouve que la plupart des choix d'architecture étaient déjà faites, comprenant :

- des classes ayant certaines responsabilités
- une interface avec `OpenGL`
- centralisation de la configuration pour éviter les *magic numbers*
- une hiérarchie de classes commençant par `Displayable` et `Dynamic Object` permettant de séparer la logique d'affichage de celle liée à la simulation en tant que telle
- une modélisation concrète de tout le scénario simulé.
- le choix d'une *Factory* pour les avions.
- l'introduction de certains attributs/fonctions-membre

Cela dit, voici quelques choix de structures de données et de fonctions de la librairie standard que j'ai dû faire pour l'implémentation de certaines tâches :

Task 1

Objectif 1 - `AircraftManager`

Pour stocker des pointeurs vers les avions créés, j'ai choisi d'utiliser `std::vector` en me disant que peut-être il y aurait besoin de les avoir dans un ordre spéci-

fique plus tard. Et j'avais raison, nous en avons besoin pour l'objectif 2 de la *Task 2*.

Objectif 2 - `AircraftFactory`

J'ai choisi de rendre la méthode de création des avions `static`, suivant le patron *factory*, vu que dans cette instance, on n'a pas besoin d'avoir un état pour la *factory*.

Task 2

Objectif 1 - B

J'ai choisi d'utiliser, en plus de `std::remove_if`, la fonction-membre `erase` de `std::vector`, cela fait un code compact et compréhensible.

Objectif 1 - C

J'ai choisi d'utiliser `std::transform`, cela permet de faire la même opération sur les éléments un rang de valeurs d'une structure itérable d'un coup, et même d'en faire des opérations binaires élément par élément avec une autre structure, donc cela m'a semblé le plus adapté pour ce qui est demandé.

Objectif 2 - A

J'ai choisi d'utiliser `std::random_device` et `std::uniform_int_distribution`, car combinées, ces structures me semblaient produire le code le plus lisible parmi toutes les options que j'ai trouvé.

Objectif 2 - B

J'ai choisi d'utiliser `std::copy` avec `std::back_inserter` parce que cela permet de mettre tous les éléments du chemin donné à la fin de la queue d'une façon très propre et lisible. On insère tous les éléments de `path_to_terminal` à la fin (donc **back**-inserter) de `waypoints`.

Objectif 2 - C

J'ai choisi de trier les avions en utilisant `std::sort` au lieu d'implémenter mon propre algorithme de tri.

Objectif 2 - D

Pour calculer l'essence manquante, j'ai utilisé `std::accumulate`, qui permet de parcourir toute la structure itérable en accumulant un résultat avec un opérateur qui prend la valeur cumulée et l'élément courant.

3 Questions difficiles

La seule question où j'ai vraiment eu du mal à m'en sortir avec ce que j'avais vu en cours et appris dans la doc, c'était la toute dernière : **Task 4 - Objectif 1 - Point 6**, où il fallait utiliser des notions que je n'avais pas suffisamment saisies, comme les *variadic-templates* et le *perfect-forwarding*.

Finalement, avec un peu de recherche sur *stack overflow* j'ai pu trouver le bon chemin. La solution était de faire un `std::cast` dans `std::forward` avant de faire la *pack expansion* pour arriver à *forward* au constructeur de `std::array` les paramètres du bon type.

4 Appréciations

Ce que j'ai aimé

J'aime bien le principe de partir d'un projet déjà fait parce que cela nous prépare en tant que développeurs à ce qui va sûrement faire partie de la plupart de nos tâches : décrypter du code que l'on ne trouve pas parfait, et qui a été fait pour quelqu'un d'autre pour ensuite en faire des améliorations ou des changements.

Ce que j'ai détesté

J'ai moins bien apprécié, d'une part, le déroulement des tâches dans le projet, dans lequel je me retrouvais à faire des changements qui ne donnaient pas l'impression d'avancer, ni dans le projet, ni dans l'apprentissage du langage. Le changement le plus visible que l'on nous a demandé de faire, c'était d'ajouter des touches pour afficher des messages dans la console. Si l'on exécute le programme avant et après de compléter les tâches du projet, on a donc l'impression que rien n'a changé. J'aurais plutôt préféré d'ajouter des nouvelles *features*.

D'une autre part, le fait de devoir, sous indication de l'énoncé, faire une tâche d'une certaine manière pour ensuite devoir revenir en arrière et le réimplémenter de la « bonne façon » est assez décourageant, car on a l'impression de devoir désapprendre des choses que l'on vient de découvrir, et cela cause plutôt de la confusion.