14/04/2022

Alexandre Jaquier et Jonathan Friedli

Rapport Laboratoire 2 : Squadron

# Introduction

Dans le cadre de ce laboratoire nous devons implémenter une classe Squadron permettant de gérer une escadrille de vaisseaux spatiaux. Pour ce faire, nous devons également implémenter certaines classes qui modélisent des vaisseaux spatiaux. Il a y 2 types de vaisseau différents, les vaisseaux « simple » et les vaisseaux pouvant transporter des cargaisons.

Chaque escadrille (« Squadron ») a un nom et peut ajouter et supprimer des vaisseaux. Une escadrille peut également avoir un leader. Ce leader doit faire partie de l’escadrille avant d’être promu. Si le leader est démis de ses fonctions, il continue de faire partie de l’escadrille. Une escadrille ne peut se déplacer qu’a une vitesse pouvant être atteint par tous les vaisseaux qui font partie de ladite escadrille. Nous pouvons également calculer le carburant que va consommer cette escadrille en fonction de la distance et de sa vitesse. Il doit être possible d’afficher les détails d’une escadrille (infos de l’escadrille et les vaisseaux qui la composent).

# Diagramme UML

Une image contenant texte

Description générée automatiquement

# Choix de modélisation et d’implémentation

## Vaisseau

Comme montré dans l’UML, nous avons fait sur un héritage afin de factoriser le code des vaisseaux. Les vaisseaux simples (« TieHunter » et « TieInterceptor ») ont une vitesse maximale, un poids, un modèle et un identifiant. Les vaisseaux transportant des cargaisons (« Shuttle » et « Dreadnought ») ont, en plus des informations citées précédemment, un poids maximal pour la cargaison et le poids de la cargaison qu’il transportent actuellement.

Toutes ces informations étant propre au type de vaisseau, nous avons d’abord pensé les stocker sous forme de constantes statiques. Cependant cela nous posait quelques problèmes de factorisation, notamment au niveau du poids de la cargaison. Nous avons donc fait le choix, de créer une classe « ShipCharacteristic » et respectivement « TransporterCharacteristic » afin de stocker ces valeurs. Nous créons ensuite dans les vaisseaux (« TieHuter », « Shuttle ») des variables statiques de type « ShipCharacteristic » ou « TransporterCharacteristic ». Dans « Transporter », nous stockons un pointeur sur le « TransporterCharacteristic » et dans Ship, un pointeur sur le « ShipCharacteristic ».

## Squadron

Afin de pouvoir gérer les vaisseaux faisant partie de l’escadrille, nous avons créé notre propre structure de données. Nous avons décidé de créer notre variante de la liste chainée, avec une struct « Maillon » représentant un vaisseau. Chaque « Maillon » a un pointeur sur « Ship » et un pointeur sur le prochain « Maillon ».

Afin d’ajouter et de supprimer des vaisseaux de l’escadrille, nous avons surchargé les opérateurs « + », « += », « - » et « -= ».

# Tests effectués

|  |  |
| --- | --- |
| **Test** | **Résultat attendu** |
| La création d’une matrice de taille 0x0 doit être possible. Si une des composantes de la taille est négative, une exception se lance. | OK |
| La création d’une matrice avec un modulo valant 0 ne doit pas être possible | OK |
| Les opérations entre deux matrices sont possibles uniquement si les deux matrices ont le même modulo. | OK |
| Les opérations entre deux matrices de taille différentes sont possibles si leur modulo est pareil. | OK |
| Les valeurs de la matrice ne sont jamais négatives. En particulier après une soustractions entre deux matrices. | OK |
| Il est possible d’écrire le contenu d’une matrice grâce à l’opérateur d’écriture de flux << | OK |
| Il est possible d’affecter une matrice à une autre matrice via l’opérateur d’affectation = | OK |
| Il est possible de créer copie d’une matrice en passant cette dernière comme paramètre. | OK |