Compte Rendu de TP

C++

- TP6 DM -

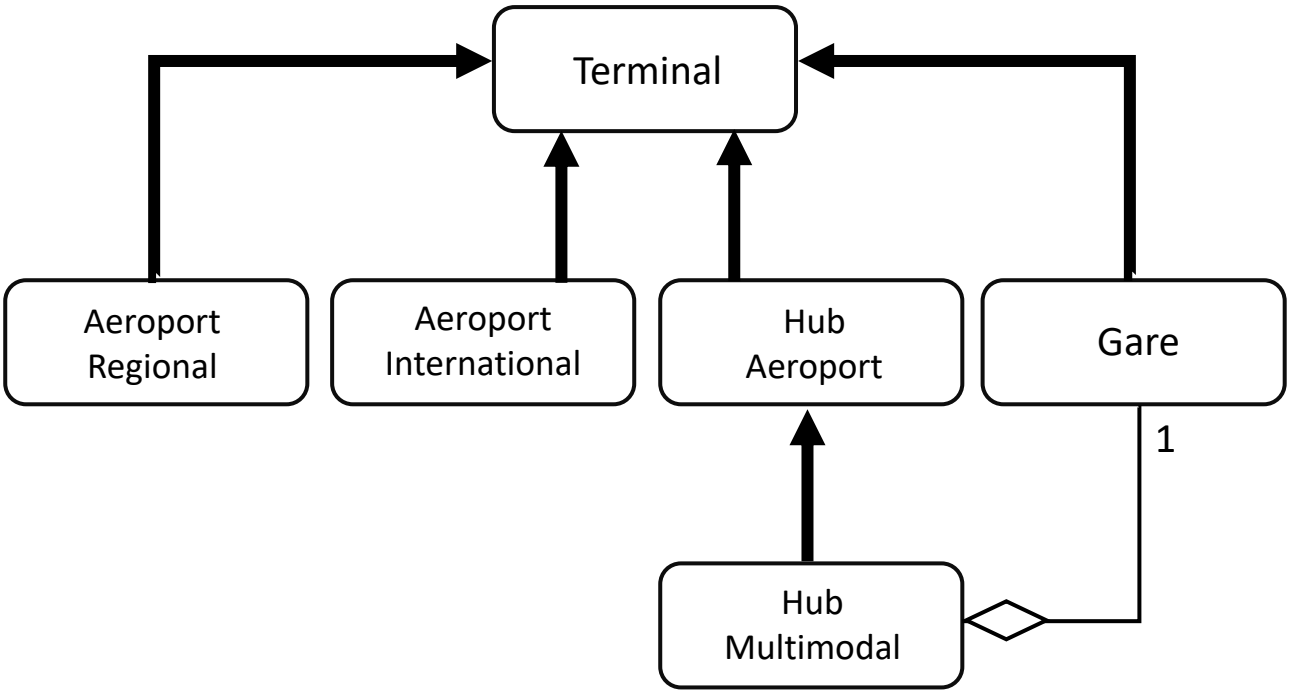
M. Romain KERVARC

2017

AUGER Romain & RIO Charles

Ingénieurs Sup Galilée 2ème année

Description : choix d'implémentation



La classe Terminal est abstraite et contient en méthodes virtuelles toutes les méthodes de ses classes dérivées. Cela est nécessaire car nous travaillons par la suite sur des Terminaux qui se distingueront par leur type dynamique.

La différence entre les classes dérivées de Terminal (hormis la gare) est le nombre de liaisons qu'elles peuvent établir. La méthode get_maxAeroport() permet d'obtenir cet attribut.

On a choisi de représenter les liaisons entre Terminaux comme étant des Lignes, et non de simples pointeurs vers d'autres Terminaux, pour faire immédiatement la liaison entre les parties A et B.
Ainsi, un Terminal est doté de 3 vecteurs pour stocker les liaisons par train, par avion et avion électrique.

Un Voyage est, de la même façon que le Terminal, doté de trois vecteurs pour stocker les lignes qui le composent. Il y a ensuite un quatrième vecteur qui nous sert à savoir quel type de ligne est la suivante. L'avancement dans chacun des trois types de ligne est enregistré dans une variable indiceType.

Les scenarii sont implémentés dans la classe Monde qui est dotée de terminaux et de voyages.

Utilisation

Ajout de liaison

L'ajout de liaison entre les terminaux se fait avec les méthodes add_liaison_type() (où type est train ou avion) depuis les classes dérivées de Terminal. Ces méthodes appellent ensuite les méthodes add_ligne_type() de la classe Terminal.
L'ajout se fait sur le terminal d'origine de la ligne, et une ligne symétrique est automatiquement créée et ajoutée. Dans le cas de l'ajout d'une ligne d'avion, la ligne d'avion électrique est ajoutée (dans les deux sens) automatiquement.
Les aeroports ne peuvent ajouter que des avions, et les gares uniquement des trains.

Manipulation des scenarii

La classe Monde est dotée d'une méthode scenario(unsigned int s) qui permet d'initialiser les Terminaux du Monde et leurs liaisons selon les descriptions du scenario s de l'énoncé.
Cette méthode appelle la méthode init_flux() qui initialise les valeurs des flux vers les différentes destinations.

On a défini des constantes pour référer aux 5 villes/terminaux :
Paris : PAR = 0
Rome : ROM = 1
Lyon : LYO = 2
Bruxelles : BRU = 3
Naples : NAP = 4

Commentaires

Variables flottantes

Nous avons remplacé les variables double par des float car nous n'avons pas besoin de valeurs très hautes et cela nous permet d'accroitre la précision.

Temps d'attente

Le temps d'attente moyen dans un Terminal est stocké dans la Ligne car il doit y en avoir un par destination. Lors du calcul du temps d'un voyage, le temps correspondant est rajouté à chaque étape. Cette variable est nulle dans notre modélisation, on considère qu'elle doit être initialisée à la bonne valeur par l'utilisateur (gérant de la compagnie de transports…).

Fréquence

On considère la fréquence comme illimitée pour créer les voyages nécessaires en optimisant sur les critères de temps et d'empreinte carbone. Ces résultats pourront permettre ensuite de déterminer la fréquence qu'il faut fixer sur une ligne donnée, donc combien de vols quotidiens sont à prévoir.

Capacité

Un voyage prend en paramètre un nombre de passagers. Si la capacité du transport est dépassée par ce nombre, on effectue des trajets supplémentaires, ce qui affecte l'empreinte carbone et la fréquence à fixer. Le temps reste identique car on considère que cela est géré dans le temps d'attente moyen.

Distance

Le calcul de la distance avec les coordonnées de longitude et latitude est très approximatif.
Nous utilisons en effet une valeur approchée du rayon de la Terre et travaillons de plus dans l'approximation que la Terre est une boule.

Méthodes virtuelles

La classe Terminale est abstraite et contient plusieurs méthodes virtuelles pures pour ses classes dérivées. Ces méthodes sont systématiquement définies dans toutes les classes dérivées, même si une méthode n'est pas à leur intention. En effet, dans le cas contraire, la classe dérivée ne pourrait pas être instanciée.
Par exemple, dans le cas d'une gare, il n'y a pas de limite de liaisons. La méthode get_maxAeroport() y est tout de même définie afin que la classe Gare ne soit pas abstraite et ainsi pouvoir instancier des objets de classe Gare.

Voyages

On considèrera qu'un Voyage ne regroupe que des gens ayant la même destination. Cela implique que deux voyages ayant la même origine et, par exemple, la même première correspondance, vont utiliser des transports distincts et donc ne pas forcément les utiliser au maximum de leur capacité. Cependant, une fois la liste des voyages établie en fonction des conditions de flux, on pourrait faire une nouvelle étude pour les comparer et repérer les lignes communes sur lesquelles une optimisation est possible.

Etude de cas

Le but est d'acheminer les passagers à destination en minimisant l'empreinte carbone et en respectant un temps maximum.
En comparant les caractéristiques des trois transports disponibles, on remarque que le train a à la fois la plus grande capacité et la plus petite empreinte carbone. C'est donc le transport à privilégier lorsque le temps le permet.
Viens ensuite l'avion électrique, qui compense sa capacité inférieure à celle de l'avion avec une empreinte carbone 3 fois moindre.
Enfin, l'avion classique devra être utilisé lorsque le temps devient une contrainte importante.

Tests

Voici un premier test pour les Parties A et B et leur fonctionnement ensemble :

```cpp
int main()
{
    AeroportInternational* Paris = new AeroportInternational("Paris", 2.333333, 48.866667);
    AeroportRegional* Berlin = new AeroportRegional("Berlin", 13.4105300, 52.5243700);
    Ligne<Avion>* T = new Ligne<Avion>(Paris, Berlin);

    Paris->add_liaison_avion(T);

    std::cout << "Ligne Parisienne\n";
    std::cout << "\tOrigine : " << Paris->get_liaison_avion_elec(0)->get_origine()->get_nom() << std::endl;
    std::cout << "\tDestination : " << Paris->get_liaison_avion_elec(0)->get_destination()->get_nom() << std::endl;

    std::cout << "Ligne Berlinoise\n";
    std::cout << "\tOrigine : " << Berlin->get_liaison_avion_elec(0)->get_origine()->get_nom() << std::endl;
    std::cout << "\tDestination : " << Berlin->get_liaison_avion_elec(0)->get_destination()->get_nom() << std::endl;

    std::cout << "\nDistance : ";
    std::cout << Paris->distance(Berlin->get_longitude(), Berlin->get_latitude()) << std::endl;

    Voyage* V = new Voyage(Paris, Paris);
    V->ajouter_ligne(Paris->get_liaison_avion_elec(0));
    V->ajouter_ligne(Berlin->get_liaison_avion_elec(0));
    std::cout << "\nVoyage pour 1 passager\n";
    V->calcul(1);
    std::cout << "\nVoyage pour 101 passagers\n";
    V->calcul(101);

    return 0;
}
```

```
Creation AeroportInternational de Paris
Creation AeroportInternational de Berlin
Ligne Parisienne
        Origine : Paris
        Destination : Berlin
Ligne Berlinoise
        Origine : Berlin
        Destination : Paris

Distance : 1298.56

Voyage pour 1 passager
Temps : 6.4928
Empreinte carbone : 15582.7

Voyage pour 101 passagers
Temps : 6.4928
Empreinte carbone : 31165.5

Process returned 0 (0x0)   execution time : 4.454 s
Press any key to continue.
```

On voit que l'ajout d'une seule Ligne entre Paris et Berlin a bien ajouté la ligne de sens opposé.
De plus, on a jouté une ligne de type Avion, et l'on travaille sur une ligne de type AvionElectrique qui s'est bien créée automatiquement en conséquence de l'ajout de la ligne d'avion.

Le calcul de la distance nous donne ensuite 1298.56 km (on comptera 1300).

On test ensuite deux voyages aller-retour Paris, Berlin, Paris en avion électrique pour 1 puis 101 passagers.
On rappelle les caractéristiques de l'avion électrique :

- Capacité : 100 passagers,
- Empreinte carbone : 6 kg/km,
- Vitesse : 400 km/h.

Le temps de trajet est donc d'environ 2*1300/400, soit environ 6.5h, c'est bien cela.
(On rappelle qu'avec ce modèle, seul le temps de trajet entre en compte car le temps d'attente est nul.)

L'empreinte carbone pour un passager correspond à celle d'un seul avion électrique, soit environ 2*1300*6 = 15600kg.

Et le double pour 101 passagers car on doit utiliser 2 avions.
Les calculs sont donc justes.

On peut ensuite tester la création et l'initialisation d'un monde et d'un scenario :

```
int main()
{
    Monde* M = new Monde();
    M->scenario(1);

    std::cout << "\nFlux Paris -> Rome\n";
    std::cout << M->get_term(PAR)->get_flux(M->get_term(ROM)) << std::endl;
    std::cout << "\nFlux Rome -> Paris\n";
    std::cout << M->get_term(ROM)->get_flux(M->get_term(PAR)) << std::endl;

    return 0;
}
```

```
Creation HubAeroport de Paris
Creation AeroportInternational de Rome
Creation AeroportRegional de Lyon
Creation AeroportRegional de Bruxelles
Creation AeroportRegional de Naples

Flux Paris -> Rome
10000

Flux Rome -> Paris
9000

Process returned 0 (0x0)   execution time : 2.469 s
Press any key to continue.
```

La compilation et l'exécution sans erreur montrent que l'initialisation des terminaux et l'ajout des lignes a bien eu lieu, et ce en tenant compte du type dynamique.
Les flux sont aussi bien de la valeur attendue.

Optimisation de scenario

Scenario 1

Paris et Rome sont des aéroports de transition
Il n'y a ici que des aéroports. On utilisera donc, lorsque le temps maximum le permet, des avions électriques, et lorsqu'un trajet en avion électrique prend trop de temps, on étudiera la possibilité de faire une partie en avion électrique et le reste en avion ou on prendra uniquement l'avion, rapide mais polluant.

Scenario 2

Les liaisons Paris-Bruxelles se font en train, sinon, on pose le même problème qu'au scenario 1.

Scenario 3

Les liaisons Paris-Bruxelles et Rome-Naples (dans les deux sens) se font en train.
Lorsque le délai est assez long, on prendra la ligne de train Paris-Lyon-Rome, sinon on coupera en avion électrique puis en avion par la liaison Paris-Rome.
Les voyages dont l'origine est Lyon ne prendront que le train lorsque la destination est Paris ou Bruxelles. Pour se rendre à Rome (puis à Naples), on peut soit prendre le train directement vers Rome, soit prendre le train vers Paris et y prendre l'avion. Cela fait gagner environ 5 minutes.

Scenario 4

Lorsque le temps maximum permet de faire un trajet direct en avion électrique, ce sera l'option à prendre. Sinon, il faudra comparer le trajet direct en avion, qui respectera le critère de temps mais polluera, avec l'option de faire une ou des correspondances et découper le trajet entre avion électrique et avion.

Scenario 5

Ici, il n'y a pas de question à se poser, tous les passagers prennent le train et suivent la ligne dans l'unique ordre possible.

Scenario 6

On privilégie l'avion électrique et, lorsque le délai est trop court, on prend l'avion. Lorsque la destination n'est pas Paris, on étudiera la possibilité de prendre l'avion électrique sur l'un des deux trajets (avant ou après correspondance à Paris).

Conclusion

1) Lorsque le délai est important, le scenario 5, qui ne propose que des trains, est le meilleur.
2) Lorsque le délai est très court, l'utilisation de l'avion est obligatoire et le scenario 4 devient le plus intéressant en offrant le plus de possibilité de découpage et d'utilisation de l'avion électrique.
3) Le scenario 6 est toujours moins intéressant que le 4.