Université de technologie de Belfort-Montbéliard

LO43 Flotte de véhicule autonomes

Buri Theo Florian Lacour

Belfort, Décembre 2014

Table des matières

Introduction

Dans le cadre de l'UV LO43 " Bases fondamentales de la programmation orientée objet", il nous a été demandé de réaliser un projet de groupe, afin de mettre en pratique les connaissances acquises lors des cours et TDs du semestre.
Trois sujet nous on été présentés. Nous avons fait le choix de traiter le sujet de la "Flotte de véhicules autonomes", et ceci pour plusieurs raisons :

- (A TROUVER)
- N'étant que deux, sur un maximum de quatre étudiants par groupe autorisés, les autres sujets ne nous ont pas parus réalisables en temps et en heures et sans bugs majeurs...
- (A TROUVER)

Nous présenterons tout d'abord le sujet, ses contraintes, et les libertés prises par rapport à celles-ci. Par la suite, nous parlerons des différents diagrammes UML, et les expliquerons. Enfin, nous terminerons par l'implémentation en Java et l'interface graphique.

Première partie

Présentation du sujet

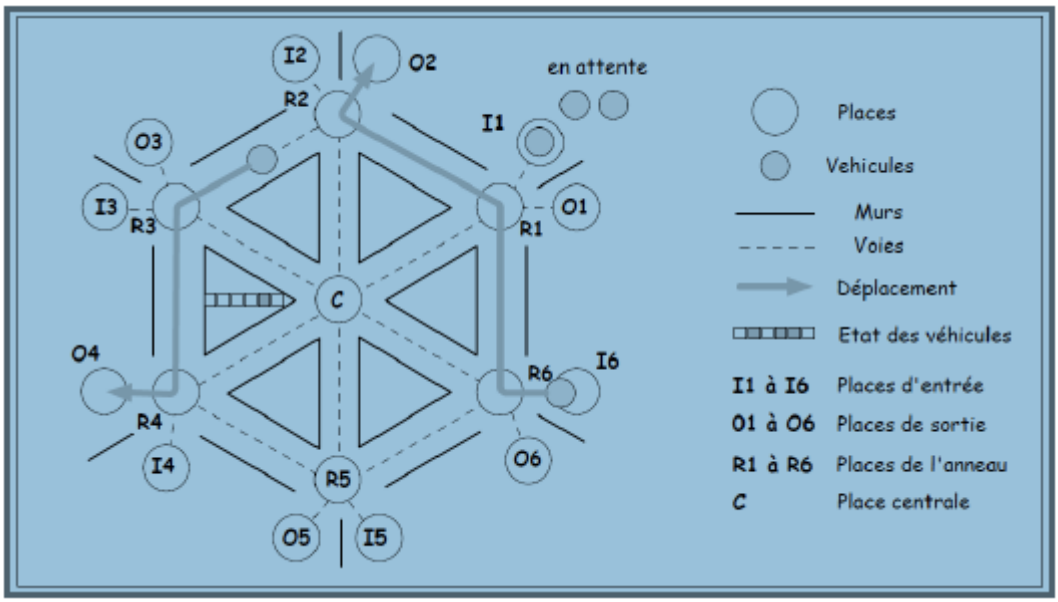
1.2 Objectif

Le programme à réaliser consistait en la modélisation d'une flotte de véhicules évoluant dans une infrastructure de circulation partagée. Pour cela, il a tout d'abord fallu modéliser la partie calculatoire à l'aide du langage UML, puis ensuite l'implémenter en Java et lui donner une interface graphique

1.3 Reformulation du sujet

1.3.1 Plateau

Le plateau donné par le sujet est le suivant :



Chaque place, exceptée celle du centre, est rattachée à une place de départ et une place de sortie. Les voitures disposant d'un passager partent depuis une place d'entrée (que celui-ci leur donne) et, à la fin de celle-ci, rejoignent une place de sortie (même chose).
La place centrale n'est utilisée que lors d'un trajet reliant deux places opposées...

1.3.2 Passagerss et contrôleur

Chaque passager doit être transporté d'une place à une autre. Pour ce faire, ils envoient une requête au module qui se charge du contrôle du plateau : le contrôleur. Par la suite, celui-ci assignera cette mission à un des véhicules de sa flotte (sous la forme d'un passager, ayant un départ et une arrivée). Celui-ci calculera ensuite le trajet qu'il suivra pour la mener à bien.
C'est alors le véhicule qui décide de partir : pour cela il doit envoyer une requête au contrôleur pour savoir si son chemin est déjà réservé. Si ce n'est pas le cas, le contrôleur l'autorise, et le véhicule part. La requête se fait sous la forme d'une Request Map, constituée de booléens indiquant l'intention du véhicule de réserver une place ou non.
Pour exemple, lorsqu'une voiture désire aller de la place I1 à O2, elle aura besoin de réserver I1, R1, R2 et O2. La Request Map seras alors la suivante :

I1	I2	I3	I4	I5	I6	R1	R2	R3	R4	R5	R6	O1	O2	O3	O4	O5	O6	C
T	F	F	F	F	F	T	T	F	F	F	F	F	T	F	F	F	F	F

Afin d'éviter toute erreur lors de la réservation, nous avons défini la priorité des places qui suit :

$$I1<I2<I3<I4<I5<I6<R1<R2<R3<R4<R5<R6<O1<O2<O3<O4<O5<O6<C$$

Ceci permettra d'éviter que deux véhicules tentent de réserver la même place à deux moments différents, et que le contrôleur les valide...

1.3.3 Contraintes et libertés prises sur le sujet

S'ajoutent alors quelques contraintes, qui régissent le programme :

- Les véhicules ne peuvent utiliser la place centrale qu'à la seule condition qu'ils doivent se rendre à la place en face de la leur.
- Une place ou une route (reliant deux places) ne peut être utilisée que par un seul et unique véhicule.
- Un véhicule qui souhaite emprunter un chemin occupé doit attendre que celui-ci se libère.

Ainsi, par rapport à ces différentes consignes, nous avons choisi de prendre certaines libertés :

- Un véhicule ne peut pas contourner une place occupée pour mener sa mission à bien.
-

Deuxième partie

Conception UML

Chapitre 1

Les Diagramme UML

1.1 Diagramme des cas d'utilisation (annexe A)

Le diagramme des cas d'utilisation permet de représenter les actions que l'utilisateur peut faire. Dans notre projet, en lançant le programme, l'utilisateur a plusieurs solutions :

- Voir les crédits (c'est à dire les auteurs du programme, nous)
- Choisir de jouer en mode manuel, dans ce cas il devras par la suite choisir lui-même les missions attribuées aux véhicules.
- Jouer en simulation, dans ce mode toutes les requête sont générées par le programme : l'utilisateur se contente de regarde la simulation s'opérer.
- Avoir accès aux options : une fois dans ce menu, il peut choisir de changer la rapidité des voitures.

Nous avons choisi de diviser l'utilisateur principal en deux autres : un utilisateur qui utiliserait le mode manuel, et un autre qui utiliserait le mode automatique. Ceci étant dû au fait que l'utilisateur en mode manuel pourra accéder à des fonctions interdites au mode automatique (la création d'un passager).
En outre, une fois sur l'écran de "jeu" (où se déplacent les véhicules), l'utilisateur (et quel que soit son mode) pourra mettre la simulation en pause et, si nécessaire, réinitialiser celle-ci.

1.2 Diagramme de séquence(annexe B)

Le diagramme de séquence permet de représenter les interactions entre les différentes classes selon un ordre chronologique.

1.3 Diagramme de classes(annexe C)

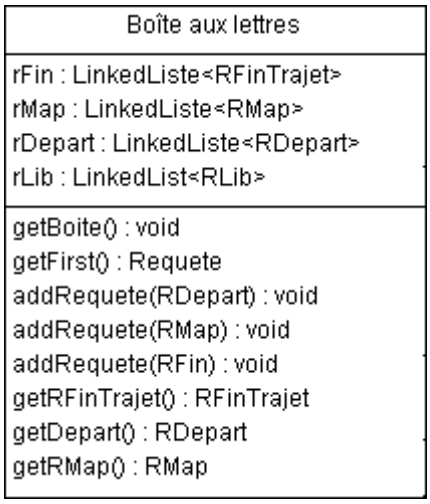
Nous verrons en détail, une description de chaque classe.

Chapitre 2

Description des classes

2.1 Classe BoiteAuxLettre

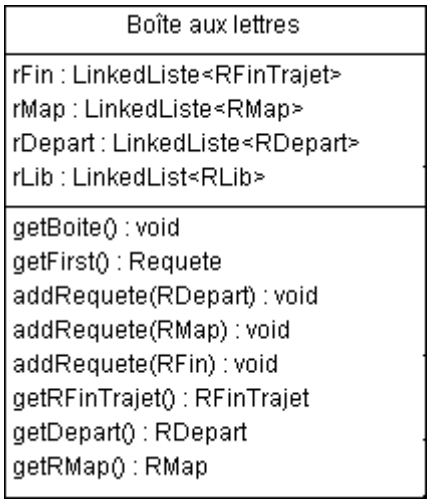
Il s'agit de la classe recevant toute les requêtes, qu'elles soient envoyées depuis la partie graphique (c'est à dire des requêtes pour la création d'une nouvelle mission, et donc d'un nouveau passager), ou du modèle. Il y a 4 requêtes différentes, stockées chacunes dans une liste différente :



- Une liste de requêtes de départ : cette liste contient toutes les demandes de trajet. Ces trajet peuvent être demandés soit par l'utilisateur, soit générés par le modèle.
- Une liste de requêtes de trajet : ce sont les requêtes qu'envoient chacun des véhicules ayant un passager pour demander au contrôleur la permission d'utiliser un chemin (la réponse n'est positive que si le chemin concerné est libre)...
- Une liste de requêtes de fin de trajet : ce sont les requête qu'envoient les véhicules lorsqu'ils ont mené leur mission à bien (et donc qu'ils sont arrivés à leur place d'arrivée). Elles permettent alors de libérer le véhicule.
- Enfin, une liste de requêtes de libération de ressources : ces requêtes sont envoyées par les véhicules lorsqu'il n'ont plus besoin d'une place (lorsqu'ils sont déjà passés de dessus). Le contrôleur peut alors considérer ces places comme disponibles.

2.2 La classes Contrôleur

Il s'agit de la classe principale : c'est elle qui s'occupe de gérer les déplacements des différents véhicules sur la carte. Le contrôleur a directement accès à la boîte aux lettres, et donc aux requêtes qu'elle contient.
C'est également lui qui possède la "carte" (ici, une HashMap) des différentes places du réseau routier : il est le seul à agir dessus. La méthode traiteRequete() se charge de traiter les requêtes selon leur priorité :



- Les requêtes de fin de trajet sont prioritaires sur toutes les autres : rendre un véhicule à nouveau disponible signifie le débarquement d'un passager, et donc la capacité d'en accepter de nouveaux.
- Les requêtes de libération de ressources arrivent en second. En effet, rendre une place disponible le plus vite possible est prioritaire pour pouvoir valider les trajets d'autres véhicules.
- Viennent ensuite les requêtes de départ, permettant aux passagers d'être pris en charge plus vite que les requêtes de trajet, bien plus nombreuses.
- Les requêtes de trajet sont les moins prioritaires, notamment en raison de le nombre : en effet, elles sont générées à intervalle régulier par les véhicules, jusqu'à ce qu'ils aient pu emprunter le chemin qu'ils désirent.

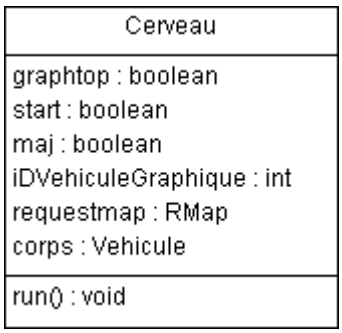
Par la suite, et en fonction des requêtes qu'il reçoit, le contrôleur a plusieurs tâches. Il peut libérer un véhicule en fin de trajet, en réserver

un autre et lui attribuer un passager. Enfin, il valide (ou non) les chemins demandés par les véhicules, et se charge ensuite de réserver les ressources correspondantes.

2.3 Les classes Requêtes

Etant donné que le but de chaque requête a été explicité ci-dessus, et que la structure de celles-ci ne présente pas d'intérêt particulier, nous nous intéresserons à leurs caractéristiques générales.
Chaque type de requêtes a sa propre liste, contenue dans la boîte aux lettres. Un requête, une fois traitée, est immédiatement détruite. Enfin,à chaque liste de requêtes est attribuée une priorité, afin que le contrôleur traite les requêtes les plus importantes en premier.

2.4 La classe Cerveau



Il s'agit du Thread chargé de négocier le départ puis l'arrivée d'un véhicule. Il est ajouté au véhicule lorsque celui-ci reçoit un passager. C'est alors lui qui se charge, après avoir calculé le chemin à emprunter, de faire la demande de réservation auprès du contrôleur et, si celui-ci l'autorise, à faire partir le véhicule. Cette autorisation se fait par le biais d'un booléen, modifié par la flotte de véhicules, à la demande du contrôleur.
Par la suite, il se charge d'acheminer le véhicule jusqu'à sa destination, et signale ensuite au contrôleur la fin du trajet. Il est alors détruit, et le véhicule devient à nouveau disponible. Celui-ci recevra un nouveau "Cerveau" lors de l'embarquement du passager suivant. En parallèle, le cerveau se charge également de synchroniser l'avancée du trajet avec la partie graphique : en effet, afin d'éviter des bugs d'affichage, le modèle doit attendre que le véhicule affiché atteigne la place suivante avant d'envoyer les requêtes de libération et, à la fin du trajet, la requête de fin de trajet.

2.5 La Classe Véhicule

Chaque véhicule possède un identifiant unique donné par la classe "Flotte de Véhicules" et stocké dans un Integer "identifiant". Les véhicules sont instanciés sans trajet ni cerveau. Ceux-ci lui seront attribués plus tard, la de l'embarquement d'un passager. Le Cerveau se charge alors d'appeler les fonctions du véhicule pour calculer le trajet, envoyer des requêtes... A la fin de son trajet, seul le cerveau est détruit. Le véhicule redevient alors libre, dans l'attente d'un nouveau passager.

Troisième partie

Implémentation Java

Chapitre 3

Choix d'architecture

3.1 Le pattern MVC

Afin de permettre une meilleure représenation des différentes parties du programme, le choix a été fait d'utiliser le pattern "Modèle - Vue - Contrôleur". Il est constitué comme suit :

- La vue est la partie visible pour l'utilisateur : elle rassemble les différentes fenêtres du programme, les boutons et autres moyens pour l'utilisateur d'interagir avec celui-ci.
- Le contrôleur (afin d'éviter toute confusion avec le nom de la classe principale du programme, nous l'appellerons Vérificateur)se charge de vérifier que les données venant de la vue (et donc souvent de l'utilisateur), afin d'éviter l'envoi d'arguments non attendus aux fonctions internes. Il se charge par la suite de transmettre ces données au modèle.
- Le modèle est la partie calculatoire du programme. C'est lui qui, dans notre cas, se charge de la totalité de la simulation (voitures, contrôleur, trajets, etc...). Il peut alors envoyer des données à la vue pour qu'elle les affiche (faire bouger un véhicule, par exemple).

Ces trois parties sont mises en relation par le biais d'interfaces "Observer" et "Observable", qui permettent au modèle d'envoyer des informations à la vue sans même avoir conscience de son existence. Cette indépendance permet de modifier la vue à volonté sans avoir à modifier le modèle...

3.2 Des threads pour régir les déplacements

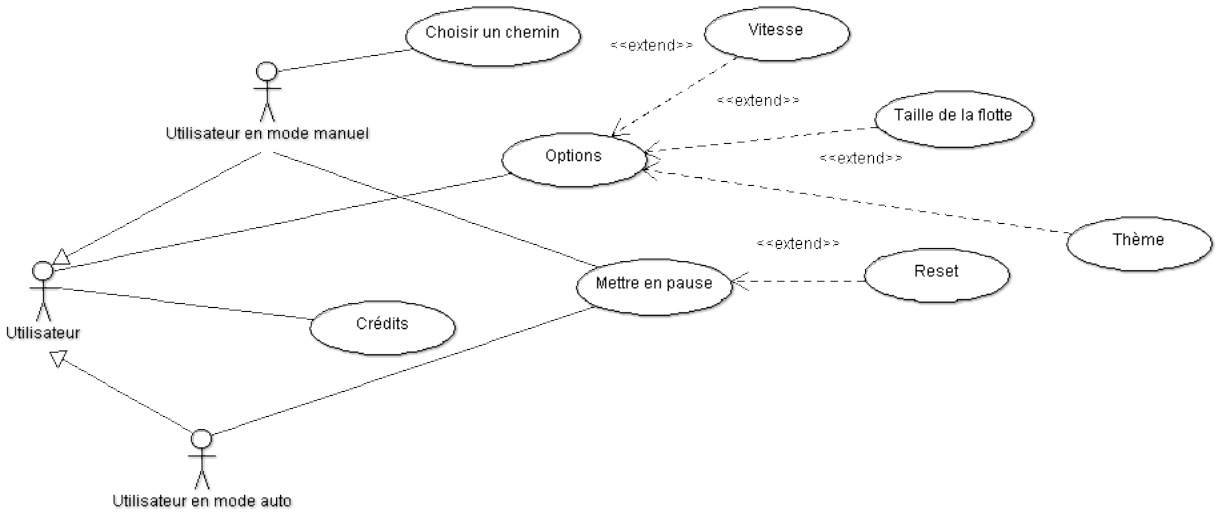
Nous avons fait le choix d'utiliser des Threads pour permettre à chaque véhicule de bouger indépendamment.

Chapitre 4

Difficultés rencontrées

Conclusion

Annexe A


Choisir un chemin
<<extend>>
Vitesse
Utilisateur en mode manuel
<<extend>>
Taille de la flotte
Options
<<extend>>
Thème
<<extend>>
Reset
Mettre en pause
Utilisateur
Crédits
Utilisateur en mode auto

Annexe B