Simulation de la circulation des taxis et des navettes dans une ville

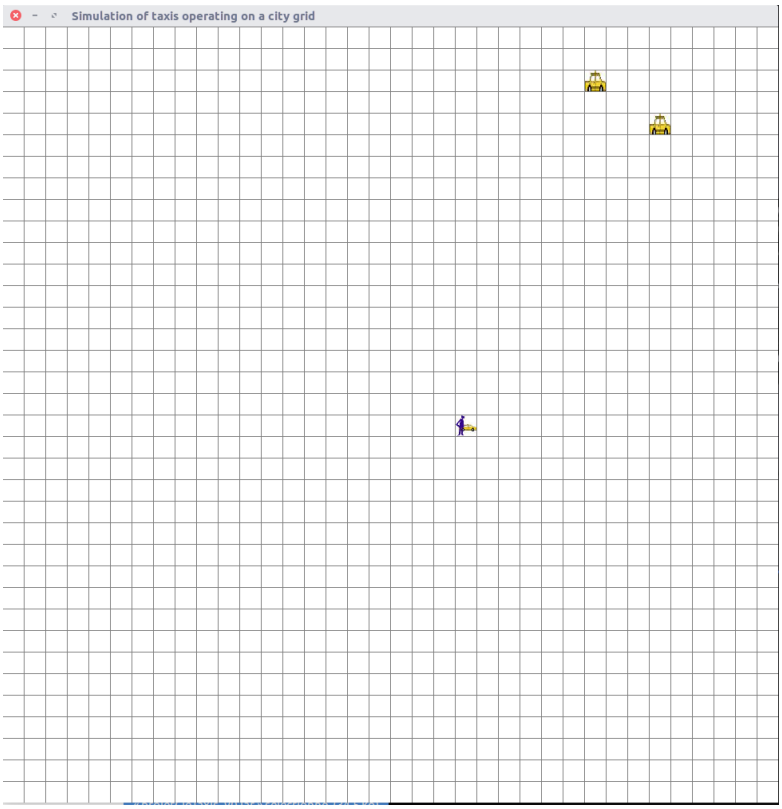
L'3 Programmation Java Année 2016-2017

Le sommaire

I Introduction

Un template nous a été fourni au pour ce projet. Au début, ce template représente une ville dans laquelle trois taxis existent. Des passagers apparaissent aléatoirement sur la grille de la ville, et un taxi quelconque dans la ville va se déplacer pour collecter un passager tant qu'il est libre. Une fois que le taxi a collecté le client, il va ensuite vers la destination. Une fois qu'il est arrivé à la destination du client, le client descend et le taxi redevient libre.

Le but de ce projet est de d'abord comprendre les classes fournies puis d'ajouter des fonctions à cette GUI petit à petit, et de développer une simulation aussi riche que possible d'une manière progressive. Dans les classes fournis, on a constaté que le modèle MVC est utilisé, donc la compréhension de ce modèle est aussi très importante avant le lancement du projet.



II Prérequis : le modèle MVC

MVC est une abréviation de Modèle-Vue-Contrôleur. C'est un modèle qui sert à développer des GUI (Graphical User Interface).

Le modèle :

C'est l'élément qui contient les données ainsi que de la logique en rapport avec les données: validation, lecture et enregistrement.

La vue :

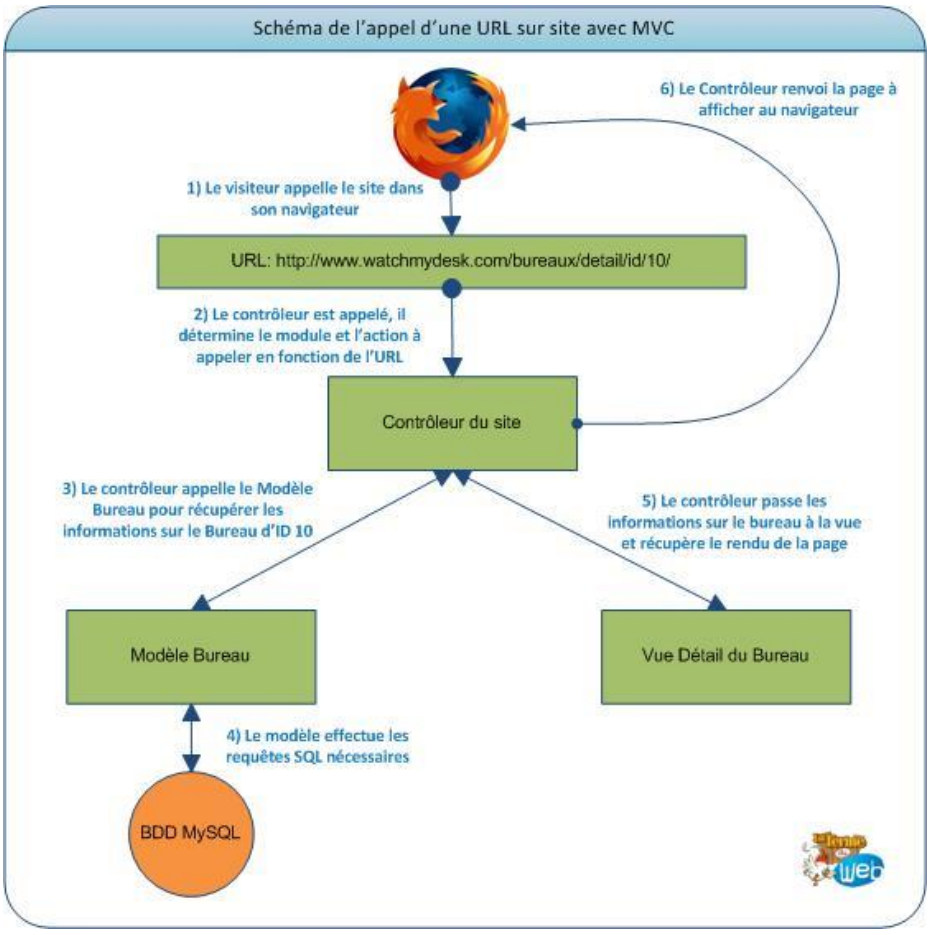
C'est la partie visible d'une interface graphique. La vue se sert du modèle, et peut être un diagramme, un formulaire, des boutons, etc.

Le contrôleur :

C'est le module qui traite les actions de l'utilisateur, modifie les données du modèle et de la vue.

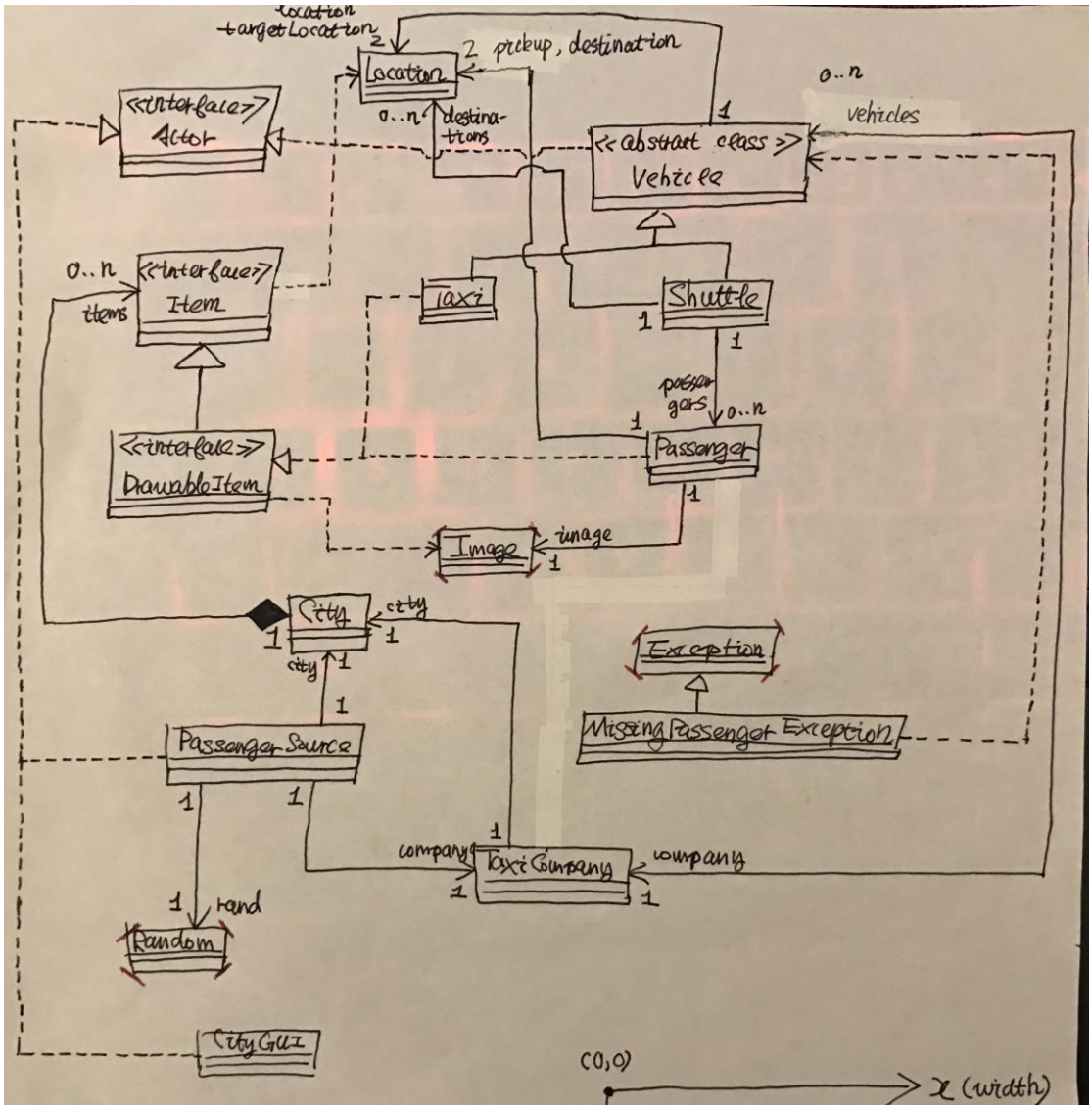
Exemple :

L'utilisateur vois une fenêtre d'une application (Vue) sur l'écran, il clique sur un bouton de la fenêtre



II Prérequis : la compréhension totale du template fourni

Il faut noter que la compréhension du rôle de chaque classe est très importante pour le projet, cela sert des briques de base pour toutes les autres phases du projet. Si la compréhension complète du projet n'est pas faite, on aura souvent besoin de revenir au début pour comprendre le fonctionnement de chaque partie et cela sera très inefficace.



III Travail à réaliser

Phase 1 :

1. Dans la classe Location, implémenter le corps de la méthode nextLocation()
2. Attacher à chaque Vehicle un attribut de type String représentant son ID.
3. Compléter la classe Vehicle et la méthode appropriée de la classe TaxiCompany pour que chaque véhicule créé reçoive automatiquement un identifiant qui lui soit spécifique. Cet identifiant, une fois attribué, ne pourra plus être modifié.
4. Dans la classe TaxiCompany, modifier la méthode appropriée pour que ce soit le taxi libre le plus proche du demandeur qui lui soit envoyé.

Phase 2 :

Compléter la classe CityGUI pour afficher des informations reliées à la simulation sur la même fenêtre :

a) La dimension de la grille de ville
b) Le nombre de taxis dans la ville
c) Le nombre de taxis disponibles dans la ville
d) Le nombre de passagers en attente d'un taxi
e) Des « missed pickup »
f) Le temps qu'un véhicule ne bouge pas (pas de demande du client).

Phase 3 :

1. Créer une classe ConfigReader permettant à l'utilisateur de saisir, dans une seule fenêtre graphique autonome, l'ensemble des paramètres de configuration de la simulation : nombre de taxis de la compagnie, nombre de navettes de la compagnie, dimensions (largeur, hauteur) de la grille de la ville, voire autres paramètres.
2. Dans la classe CityGUI, optimiser la taille de la fenêtre graphique initiale de façon qu'elle soit aussi grande que possible (quelle que soit la taille de l'écran) mais conserve un ratio hauteur / largeur égal à celui caractérisant la ville
3. Imposer aussi une taille minimale à la fenêtre graphique de façon que l'utilisateur ne puisse pas réduire exagérément la taille de fenêtre d'affichage

Phase 4 :

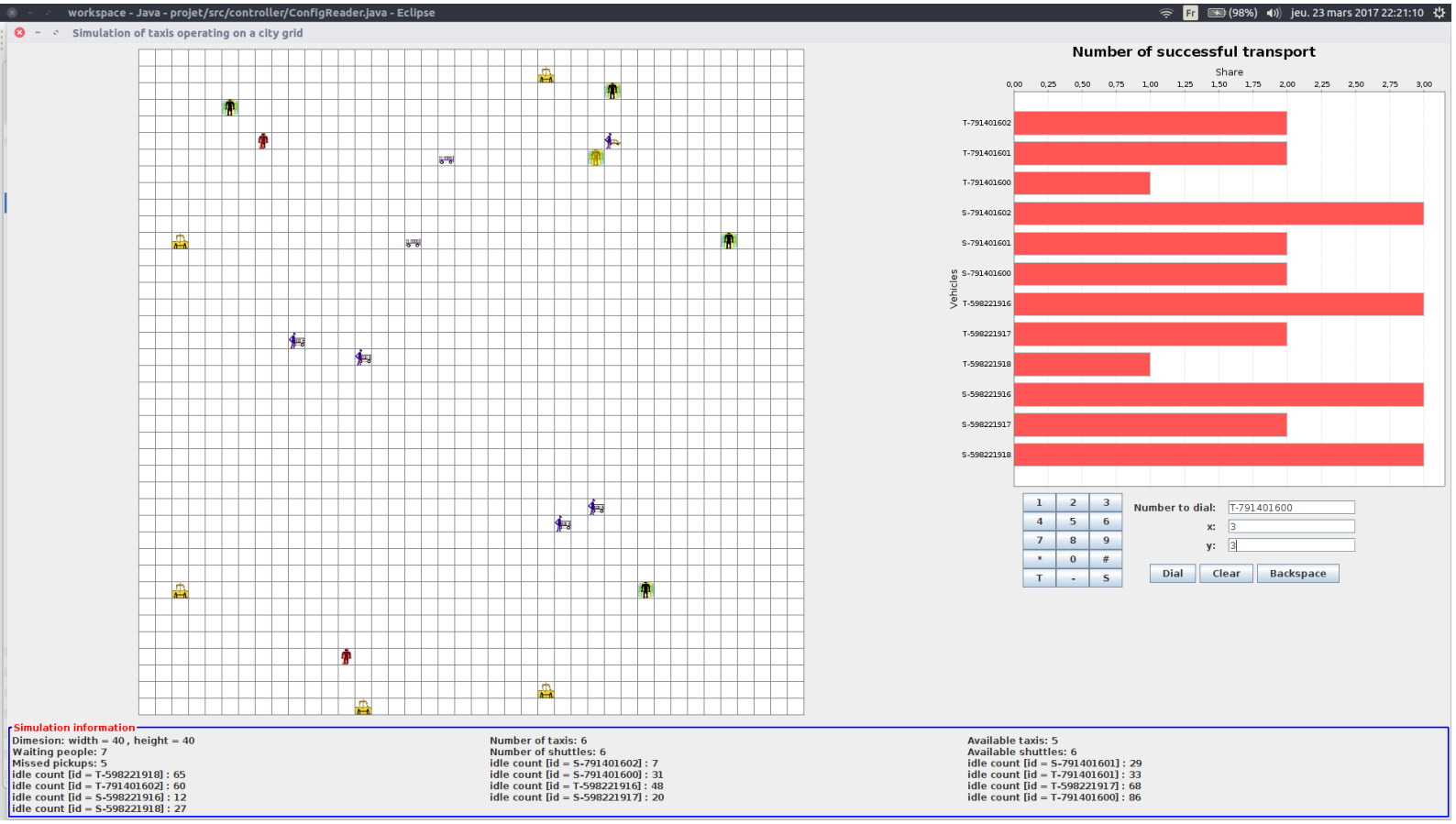
Implémenter la classe Shuttle (navette) et compléter les autres classes pour permettre, en sus et sur le même panneau graphique que celui déjà géré, une simulation de navettes opérant sur la grille de la ville.

Phase 5 :

1. Implémenter : si un passager attend trop longtemps et il n'y a pas de véhicule qui vient le collecter, il disparaît sur la carte.
2. Fournir une IHM clavier pour simuler un appel pour demander un taxi.
3. Afficher un histogramme JFreeChart le nombre de transports réalisés par chaque véhicule.
4. Gérer plusieurs compagnies de taxis.
5. Définir une zone piétonne interdite aux véhicules.
6. Sauvegarder l'information de la simulation dans un fichier.

IV Description explicite du projet effectué

Après avoir fini le projet, le produit final est :



IV Description explicite du projet effectué

Description des joueurs principaux du projet

Passenger :

1. Un Passenger dispose toujours d'un point de départ (pickup) et une destination
2. Un Passenger peux choisir de prendre soit un Taxi soit un Shuttle (navette), s'il a décidé de prendre un taxi, son image sur la fenêtre est noire ; s'il son choix est un Shuttle, alors que son image est rouge sur la fenêtre.
3. Le choix du Passenger est aléatoire (50% pour Taxi, 50% pour Shuttle) lors de la création d'un Passenger
4. Un Passenger s'énerve s'il le temps d'énervement est dépassé, à ce moment-là, si le Passenger est encore sur le map (pas dans un véhicule), alors que l'image de ce Passenger devient jaune pour montre qu'il est énervé.
5. Une fois qu'un Passenger a réussi à faire une demande de « pickup » auprès d'un véhicule, un couple (Passenger, Vehicle) est enregistré chez la compagnie de taxis, et donc le Passenger ne peut plus faire la demande de « pickup » auprès d'autres Vehicles (Véhicules), du coup d'autre Vehicles ne peuvent pas collecter ce Passenger car ce Passenger n'existe pas dans sa liste de demande.
6. Après qu'un Passenger devient énervé, pas longtemps après, la limite d'attente de ce Passenger est aussi atteinte, à ce moment-là,
    a. Si le Passenger est encore sur le map (pas encore été collecté par un véhicule), et
        i. S'il sait que le véhicule qu'il attends (le véhicule auprès lequel il a déjà fait une demande) est en train de diriger vers lui, alors qu'il ne disparaît pas, et il attend jusqu'au moment où son véhicule arrive ;
        ii. Sinon, il disparaît directement sur le map
    b. Si à ce moment-là le Passenger est dans un véhicule (soit un Taxi, soit un Shuttle),
        i. S'il sait que le véhicule est en train de diriger vers sa destination, alors il ne disparaît pas, et attend jusqu'à quand son véhicule arrive sur sa destination.
        ii. Sinon, il disparaît directement sur le map.

PassengerGroup :

1. PassengerGroup est une classe fille de la classe Passenger, et comme ça PassengerGroup héritent quasiment toutes les comportements d'un Passenger, sauf qu'il a redéfini (override) une ou deux méthodes qui lui sont propres.
2. Comme un Passenger, un PassengerGroup s'énerve aussi, et son image sur le map devient aussi jaune s'il est énervé.
3. On peut considérer qu'un PassengerGroup est un Passenger particulier, ce groupe dispose d'un nombre de personnes (un attribut) qui est généré aléatoirement entre 2 et 10 lors de la création d'une instance de la classe PassengerGroup.

Taxi :

1. Un Taxi prend un seul passager à la fois. Donc il ne prend que des Passenger et pas des PassengerGroup.
2. Une fois qu'il est en chemin (sois en train d'aller collecter un passager, soit en train de diriger vers la destination du Passenger), il ne peut plus recevoir des demandes des clients.
3. Un taxi devient libre quand il dépose son client (Passenger).

Shuttle :

1. Un Shuttle (navette) peut prendre soit des clients uniques (Passenger), soit des clients en groupes (PassengerGroup).
2. Un Shuttle dispose de deux listes
    a. Une liste qui représente tous les Passengers ou PassengerGroups dans le Shuttle
    b. Une autre liste qui représente tous les Passengers ou PassengerGroups qui demandent une collecte (pickup).
3. Un Shuttle fais son choix de destination en deux moments :
    a. Le moment où il collecte (pickup) un Passenger ou PassengerGroup
    b. Le moment où il dépose un Passenger ou PassengerGroup
4. Comment un Shuttle choisit sa prochaine destination : pour les deux moments lister ci-dessus, le Shuttle regarde dans sa liste de demandes et sa liste de Passenger et PassengerGroup, lequel est le plus proche, c'est cet endroit le plus proche qu'il va prendre comme sa prochaine destination
5. Un Shuttle ne fais pas de choix de destination s'il est en route. (ni le moment où il collecte un Passenger ou PassengerGroup, ni le moment qu'il dépose un Passenger ou PassengerGroup).

PassengerSource

Son rôle est de générer aléatoirement des Passengers ou PassengerGroups sur le map. A l'intérieur elle transmet des demandes de collecte (pickup) of des Passengers ou PassengerGroups à une des compagnies de taxis existantes (dans notre simulation, plusieurs entreprises existent)(le choix de compagnie est aléatoire), si la compagnie aléatoirement sélectionnée par le PassengerSource peut fournir le service (c'est-à-dire cette entreprise dispose au moins un véhicule qui est capable de prendre la demande du Passenger ou PassengerGroup anciennement généré). Pour le client (Passenger ou PassengerGroup), alors que les demandes sont enregistrées dans la compagnie. Si la compagnie aléatoirement sélectionnée ne peut pas fournir le service, alors que ce Passenger ou Passenger n'apparaissent pas sur le map et un missed pickup sera enregistré dans PassengerSource.

TaxiCompany

Un TaxiCompany contrôle tous les Vehicles qu'il disposent (soit Taxi, soit Shuttle).

V Conclusion

Deux points à remarquer pendent le projet :

1. Au moment où on est en train de travailler sur le projet, il faut **absolument** mettre son téléphone loin de chez lui, car pour pouvoir travailler d'une manière très efficace, l'attention totale est nécessaire.
2. La phase de prérequis n'est **absolument** non négligeable, car « good begun is half done », ces prérequis-là sont des briques de base pour le projet, si on les fait pas, on pourra jamais aller très loin dans le projet. Cela nous permet de ne pas gaspiller du temps au déroulement du développement du projet.