Projet SIM (Système et sIMulation)
Ascenseur (version 2014)

1 Objectifs

L'objectif principal consiste à présenter la technique de **simulation guidée par les événements**.

Avec l'expérience des années qui précèdent, le projet de SIM est une très bonne occasion d'approfondir les connaissances des étudiants en matière de **programmation par objets**.

2 Le sujet

Il s'agit d'écrire un simulateur d'ascenseur dont le but consiste à évaluer l'intérêt d'avoir deux boutons sur les paliers des étages de l'immeuble. En effet, lorsqu'il y a deux boutons à un palier, un passager (respectueux des autres passagers) n'enfonce qu'un seul bouton, selon qu'il souhaite monter ou descendre. Ainsi, grâce à cette information, la cabine peut décider de ne pas stopper devant un étage donné ; par exemple, dans le cas où le passager qui souhaite entrer dans la cabine veut descendre alors que la cabine est en priorité montante.

Afin de tester l'intérêt d'avoir deux boutons, le simulateur pourra fonctionner selon deux modes : le mode *parfait* dans lequel tous les passagers sont respectueux du bon usage des deux boutons, et le mode *barbare* dans lequel tous les passagers sont des égoïstes. Un passager égoïste est un passager qui enfonce systématiquement les deux touches afin d'entrer le plus vite possible dans la cabine ce qui la ralentit et la remplit inutilement.

Grâce aux générateurs aléatoires à germe constant qui sont fournis, il est possible de rejouer une simulation avec les mêmes arrivées de personnes sur les paliers. Ainsi, il sera possible de calculer le temps total de transport cumulé pour tous les passagers dans les deux cas de figure (mode parfait et mode barbare).

Pour aller du simple vers le compliqué, on va demander de commencer par programmer uniquement le mode parfait dans une simulation sans les évènements OPC et FPC [1] ; ceci pour comprendre le fonctionnement d'un ascenseur, notamment sa statégie pour changer de mode. Quand ce projet est terminé, il est assez simple d'ajouter les deux nouveaux évènements.

1. Les classes compilées java sont sur
`http://www.loria.fr/~jfmari/Cours/ascenseur_simple.zip`

3 Réalisme de la simulation

Autant que possible, il faut veiller à faire un simulateur assez réaliste. Par exemple, comme il n'y a généralement pas de capteur permettant à l'ascenseur de savoir si la cabine est pleine ou non, il serait non réaliste que les algorithmes de simulation en tiennent compte ! En effet, le programme informatique dispose de ces informations, mais pas les capteurs d'un ascenseur réel. De même, autre exemple, le parcours de la cabine ne doit pas être influencé par la longueur des files d'attente aux différents paliers. Je ne connais pas d'ascenseur ayant des dispositifs capables de mesurer le nombre de personnes en attente à un étage donné.

Pour cette simulation, l'unité de temps sera le dixième de seconde et une date sera représentée avec un `int`. Ainsi, un événement prévu pour se produire à la date 600 est en fait un événement survenant après 1 minute de simulation.

4 Liste des événements

La liste des événements est donnée aux étudiants. Elle contient deux événements de plus que le TP donné dans l'archive. Ces deux événements permettent de traiter le retard que prend la cabine quand elle s'arrête devant un palier quand un "barbare" situé sur un palier demande l'ascenseur dans les deux directions (cas du mode barbare). Il y a quatre sortes d'événements codés [2] respectivement : OPC, FPC, PCP et APP. Pour chaque code d'événement, une description de la nature de l'événement ainsi que certaines précisions concernant l'algorithme de traitement correspondant est donnée :

OPC : Ouverture Porte Cabine. Se produit au moment très précis où les portes de la cabine viennent juste de terminer leur ouverture. Cet événement est généré par le traitement d'un événement PPC (voir plus bas) quand la cabine doit s'arrêter. Il faut 5 dixièmes de seconde pour arrêter la cabine et ouvrir les portes. Une personne met exactement 4 dixièmes de seconde pour entrer ou sortir. Ceci permet de prédire l'événement FPC en fonction du nombre de personnes qui entrent et sortent.

FPC : Fermeture Porte Cabine. Se produit au moment précis où les portes de la cabine viennent juste de terminer leur fermeture. Sachant également que le temps mis par la cabine pour aller d'un étage à un

2. Au niveau de l'affichage et aussi dans ce document et non pas comme des nomns de classes !

autre est constant et vaut exactement 14 dixièmes de seconde, l'algorithme de traitement de FPC ajoute le prochain PPC à la bonne date. Attention au cas particulier ou l'immeuble serait vide (rare mais possible !).

PCP : Passage Cabine Palier. Se produit quand la cabine est juste en face d'un palier donné. Selon le cas, si la cabine doit faire un arrêt, il est possible de calculer et de générer un OPC sachant que le temps de freinage et d'ouverture des portes est de 5 dixièmes de seconde, comme dit plus haut.

APP : Arrivée Passager Palier. Se produit quand un passager commence son voyage à partir d'un palier donné. Si le passager a beaucoup de chance, alors la cabine est juste sur le bon palier, les portes ouvertes, avec encore une place. La plupart du temps, l'algorithme de traitement d'un APP se contente d'ajouter le nouveau passager dans la file d'attente du palier correspondant. Dans tous les cas, l'algorithme de traitement d'un APP génère l'APP suivant grâce au générateur de la loi de Poisson qui est lui aussi associé au palier correspondant. A tout moment, l'échéancier contient autant d'APP que d'étages. Notons également que le traitement d'un événement APP doit prendre en compte le cas rare de la cabine en mode priorité - il faut dans ce cas redémarrer la cabine en créant soit un FPC, soit un PCP.

5 Affichage du programme

Afin de faciliter la mise au point des programmes, la comparaison des résultats entre les groupes ainsi que la communication avec les enseignants, l'affichage du programme est imposé. Il est par ailleurs interdit de faire un simulateur graphique sauf groupe exceptionnel. Il y a largement assez de travail comme cela.

Affichage des passagers

Qu'il soit dans la cabine ou sur un des paliers d'un immeuble, un passager est toujours affiché de la même manière [3], avec quatre numéros qui sont respectivement, le numéro du passager, l'étage de départ, l'étage de destination ainsi que la date d'apparition de ce passager sur son étage de départ. Pour repérer plus facilement ceux qui montent et ceux qui descendent, on

3. Il faut bien entendu, dans le code Java correspondant utiliser la même et unique méthode.

ajoute soit le caractère `^` soit le caractère `v`. Par exemple, voici l'affichage du passager numéro 2222 qui a commencé son voyage en partant de l'étage 0 en destination de l'étage 4 au temps 3333 :

```
#2222:0^4:3333
```

Le passager 444 qui va du 5 ième étage au 2 ième étage a commencé son voyage au temps 66 :

```
#444:5v2:66
```

Le début du voyage correspond à l'arrivée de la personne sur un palier (et non pas l'instant où la personne entre dans la cabine). La fin du voyage correspond à l'instant où la personne sort de la cabine sur le palier de destination.

Notons au passage que la structure des données pour la classe `Passager` est évidente : trois attributs de type `int`. Rien de plus (il ne faut pas mémoriser ni le `^` ni le `v`). D'autre part, toujours pour la classe `Passager`, la numérotation des instances doit être faite automatiquement, à l'intérieur du constructeur, et ceci via une variable de classe privée.

Affichage de l'état du simulateur

Après un ou plusieurs pas de simulation, il est demandé d'afficher l'état du simulateur de la manière suivante. A priori, nous travaillerons tous avec un immeuble ou l'étage le plus haut est le 6ième et ou l'étage le plus bas est le -1. Voici un affichage possible de l'état du simulateur :

```
  6    [] : #45:6v0:26
  5    [] :
  4    [] :
  3 C [  ]:
  2    [] : #47:2v0:26, #52:2v0:28
  1    [] :
  0    [] :
 -1    [] :

Contenu de la cabine: #43:4v2:21, #44:3v0:23
Priorite de la cabine: v
Cumul des temps de transport: 9919
Nombre total de passagers sortis: 47
```

```
Echeancier: [55,FPC], [72,APP], [88,APP], [88,APP], [90,APP], [100,APP],
[110,APP], [120,APP], [160,APP]
```

Sur l'affichage qui précède, la cabine `C` est en face du palier numéro 3 et les portes du palier correspondant sont actuellement ouvertes. Une seule personne est en attente au niveau du 6ième et deux personnes au niveau 2. Toujours sur l'affichage qui précède, il y a deux passagers dans la cabine et celle-ci est actuellement en priorité `v`, c'est à dire descendante. Notons qu'à un instant donné, la cabine est soit en priorité `^` (montante), `v` (descendante) ou bien `-` (stoppée). A priori, la cabine passe en priorité `-`, la nuit quand personne ne circule dans le bâtiment.
L'affichage suivant est celui qui correspond au pas suivant de simulation :

```
  6    [] : #45:6v0:26
  5    [] :
  4    [] :
  3 C  [] :
  2    [] : #47:2v0:26, #52:2v0:28
  1    [] :
  0    [] :
 -1    [] :

Contenu de la cabine: #43:4v2:21, #44:3v0:23
Priorite de la cabine: v
Cumul des temps de transport: 9919
Nombre total de passagers sortis: 47
Echeancier: [70,PCP,2], [72,APP], [88,APP], [88,APP], [90,APP], [100,APP],
[110,APP], [120,APP], [160,APP]
```

Notez que c'est l'algorithme déroulé par l'événement `[55,FPC]` qui a fermé les portes et a ajouté `[70,PCP,2]`.

6 Structure de données

Pour presque toutes les classes, la structure de données est imposée. Le squelette des classes est donné ainsi que le `Main.java`. Il est interdit d'avoir la moindre redondance de données dans les attributs (c'est une bonne habitude à donner).

7 Réalisme de la simulation (bis)

On suppose que le constructeur d'ascenseur est assez raisonnable et que l'électronique de l'ascenseur recalcule le `status` de la cabine *avant* de finir l'ouverture des portes.

8 Dernières remarques ... en vrac

En fin de simulation, donner le temps cumulé de transport de tous les passagers.

Toute redondance de données est strictement interdite en particulier dans la représentation de l'immeuble.

Il doit être possible de changer très facilement la taille de l'immeuble en ne modifiant qu'une ou deux constantes.

Une erreur classique que j'ai souvent rencontrée réside dans la confusion entre le moment de la création d'un événement et le moment de son traitement... Ainsi, il est faux de créer l'objet `Passager` dans le constructeur de `APP`. Il convient de créer les objets `Passager` dans l'algorithme de traitement des événements `APP`.

Sinon, il faut absolument que la comparaison `==` soit expliquée et acquise durant ce projet.