Notice d'utilisation de ElevatorProject

Ervan SILVERT Paul-Armand MICHAUD

Introduction

Dans le cadre du projet d'Algorithmique et Programmation de L1S2 CMI-Informatique nous avons été invité à produire une simulation d'un ascenseur dans un building de hauteur variable ; respectant quelques règles :

Nous allons maintenant rappeler le sujet en y plaçant les valeurs et noms de variables de notre projet.

Sujet n^{o}1 - Fonctionnement d'un ascenseur Le but du projet n ^{o} 1 consiste à développer un programme informatique de simulation du fonctionnement d'un ascenseur.

- Elle comporte un nombre d'étages, variable de la simulation.
- Chaque étage est haut de 3 mètres, caractérisé dans le projet par la constante `Defines.FLOOR_HEIGHT_METERS`
- L'ascenseur peut s'arrêter à chaque étage (sauf le 404e, jamais trouvé...) .
- A chaque étage est définie la probabilité qu'au cours d'une période d'une seconde, un futur passager arrive. Elle est définie au moment de la création du type agrégé `ElevatorProject.Floor` par un nombre Random.
- Lorsqu'un futur passager arrive devant la porte de l'ascenseur :
  - Le futur passager appelle d'ascenseur et le bouton de l'étage clignote vert.
  - Le futur passager se met en attente à l'étage.
- Lorsqu'un passager monte dans l'ascenseur :
  - Son étage de destination est tiré au sort parmi tous les numéros d'étage possibles (sauf le sien) et l'ascenseur est programmé pour rejoindre cet étage, le marquant en clignotant rouge. Les caractéristiques de l'ascenseur sont les suivantes :
- Il peut transporter au maximum `ElevatorProject.Elevator.CAPACITY` passagers simultanément, défini au lancement du programme via la console .

— Quand il se déplace, sa vitesse de déplacement est uniforme (pas d'accélération, pas de décélération) et égale à 1 m/s.
— S'il n'y a pas assez de place dans l'ascenseur pour que tous les passagers d'un étage puissent y monter, un appel est automatiquement réalisé pour que les passagers qui restent à l'étage puissent monter au prochain passage.
— L'ascenseur gère la liste des numéros d'étage où il doit se rendre selon les règles suivantes :
  — Si cette liste est vide, il reste immobile à l'étage où il était arrivé en dernier. Sa direction vaut chez nous 0.
  — L'ascenseur se déplace de haut en bas ou de bas en haut en s'arrêtant à tous les étages où il doit aller dans sa direction de déplacement avant de pouvoir changer de direction.

Le simulateur sera implanté sous la forme d'une boucle infinie de gestion de ce qui se passe seconde après seconde. Ainsi, à chaque seconde divers événements peuvent arriver qu'il convient de gérer :

— Arrivées de passagers aux étages.
— Départ de l'ascenseur
— Arrivée de l'ascenseur à un étage où il a été appelé, gestion des montées et descentes
— Arrêt de l'ascenseur suite à l'absence de passager en attente aux étages
— Changement de direction du déplacement de l'ascenseur

1 Utilisation

Au lancement du programme la console vous demande de saisir le nombre d'étage du `Building` inférieur à 100.

La console vous demandera également la capacité de l'ascenseur (Elevator.CAPACITY) .

Une fois ces données saisies, la simulation se lance, le bâtiment évolue et l'ascenseur bouge.

1.1 Contrôle

— Appuyer sur 'c' permet de centrer ou décentrer la vue sur l'ascenseur.
— Appuyer sur 'o' en mode décentré, la vue va monter.
— Appuyer sur 'l' en mode décentré, la vue va descendre.
— Appuyer sur 'p' pour « Plus vite ! » permet de doubler la vitesse d'exécution du programme.

— Appuyer sur 'm' pour « Moins vite ! » permet de réduire de moitié la vitesse d'excution du programme.
— Appuyer sur 'q' pour « Quitter ! » permet au programme de se fermer proprement.

2 Fonctionnement du programme

2.1 Fonctionnement global

La boucle principale est un « Tant que » disant « Tant qu'on ne veut pas quitter ».

Avant la boucle nous initialisons toutes les variable (étoiles du ciel, étages, ascenceur etc) .

Une fois dans la boucle le rythme classique se lance, nous mettons à jour puis nous dessinons le model.

2.2 Model - Moteur physique

La fichier `ElevatorProject.java` est notre moteur physique, il contient le `main` et gère tout à propos du bâtiment.

Dans un premier temps nous gèrons tous ce qui est à propos du bâtiment par `updateBuilding()`, principalement les apparitions dans les étages des nouveaux passagers via `manageApparition()`.

Dans un second temps nous gèrons l'ascenceur qui doit mettre à jour sa liste d'attendre (`Elevator.waitingList`) par la fonction `manageCalls()`. Ensuite il cherche à se déplacer `move`. Trois cas sont possibles :

— L'ascenceur n'a pas d'appel dans sa liste et il reste immobile
— L'ascenceur souhaite bouger et continue alors dans sa direction ou s'il n'y a personne dans la direction inverse, tout est dans `updateDirection()`.
— L'ascenceur souhaite bouger mais il est à un étage, alors il doit d'abord gérer les passagers.
    1. Il cherche d'abord à se décharger d'un passager par la fonction `unstack()`
    2. Il souhaite ensuite se charger d'un nouveau passager jusqu'à être plein ou jusqu'à ce que l'étage soit vide par la fonction `stack()`.
    3. Il attend ensuite la prochaine update pour relancer ce processus et s'il n'y a plus personne, il bouge enfin.

Parmis les bonus proposés, nous avons effectivement opté pour un temps d'attente d'une seconde par passager qui se déplace. Ce temps est sauvegardé dans la constante `Defines.WAITING_TIME_PER_PASSENGER`. Á l'image

de ce dernier points, tous les paramètres de la simulation sont dans ce fichier `Defines.java`, amusez-vous donc à les modifier !

2.3 Vue - Moteur graphique

Dans une idée du model MVC [1], nous avons decidé de contruire une classe `Graphics` dans le fichier `Graphics.java` qui pose un graphisme sur le système précédent sans même le modifier.

Voici l'ordre d'exécution :

1. Nous dessinons le ciel puis le sol, qui sont de l'ordre du détail avec `drawSky()` et `drawGround()`.
2. Nous dessinons ensuite le building (`drawBuilding()`) qui dessine lui-même chaque étage avec `drawFloors()` qui appelle elle-même le dessin des passagers `drawPassengersInFloor()`.

   Dans le même temps, `drawBuilding()` appelle aussi `drawElevator()` qui s'occupe de l'ascenceur (comme son nom l'indique) . Cette fonction a probablement été la plus compliquée de tout le modèle graphique car nous avons souhaité afficher un mouvement fluide. Nous avons donc dû interpolé le mouvement de l'ascenceur entre deux `update()` en utilisant un `dt` qui donne la « distance temporelle » jusqu'à la prochaine `update()`.

2.4 Controle - Moteur graphique

Cette partie a été brièvement programmée. Nous souhaitions une interface bien plus interactive, mais elle sert finalement que de panneau de contrôle pour l'utilisateur. Le temps qui s'est écoulé depuis le début de la simualtion apparaît, relativement au multplicateur de vitesse, lui-même inscrit sur la droite.

Pour une meilleure observation, nous avons inscrit le nombre de personne qui souhaite descendre et monter au prochain étage, de manière à pouvoir contrôler le temps d'arrêt.

1. Model - Vue - Contrôleur ; séparation du graphisme et des données