Han KE
Yu CHEN
Zhenliang MU
Tuteur: Emmanuel DAUCÉ

PROJET SISN S8:
Simulateur à événement discrèts

03/2017 - 05/2017

1. Présentation du projet

1.1 Introduction à la simulation à événements discrets

La simulation à événements discrets est une technique utilisée dans le cadre de l'étude de la dynamique des systèmes. Une simulation à événements discrets est une modélisation informatique où le changement de l'état d'un système, au cours du temps, est une suite d'événements discrets. Chaque événement arrive à un instant donné et modifie l'état du système.

De nos jours, cette technique est couramment utilisée tant par les industries et les entreprises de services afin de concevoir, optimiser et valider leurs organisations que par les centres de recherche dans l'optique d'étudier les systèmes complexes non-linéaires.

1.2 Sujet et objectif de projet

Et ici pour notre projet, le sujet est décrit comme ci-dessous:

On dispose de k ascenseur permettant d'accéder aux étages d'un immeuble de N étages. Des utilisateurs se présentent aléatoirement à chaque étage i afin de se rendre à l'étage j. On cherche à coordonner les réponses des ascenseurs afin de minimiser le temps d'attente des utilisateurs et le temps de transport.

Notre objectif global de projet est de trouver et tester différentes algorithmes possibles pour réaliser un système ascenseur qui satisfait les demandes de notre sujet, à l'aide d'un simulateur à événements discrets(package Simpy sous langage python)

1.2. L'équipe et son fonctionnement

Après avoir clairement et ardemment identifié notre problématique et la portée qu'on voulait donner à notre travail, nous avons répartit la charge de travail comme suit :

Chacun endossait un rôle dans la gestion de projet en plus de se charger d'une partie de l'exposé finale.

Yu CHEN	Responsable planning
Zhenliang MU	Responsable de programmation
Han KE	Responsable du développement UI

2. Codage et algorithme

2.1 Premier arrivé, premier servi

Tout d'abord, on a essayé une édition qui réalise l'algorithme de premier arrivé, premier servi. C'est-à-dire que des utilisateurs sont livrés par l'ordre d'apparition. Il y a deux parties dans cette édition, une partie de génération des utilisateurs et l'autre partie de transport des utilisateurs.

On a appliqué le transport des utilisateurs simplement à l'aide de package Simpy qui possède une fonction *request()* nous permet de réaliser directement. Cette fonction contrôle un événement selon le temps ou les autres conditions, et ici c'est le temps en fonction de la distance entre l'étage de départ et arrivée. Or, on ne peut pas préciser la position des ascenseurs car cette fonction de *request()* ne nous la donne pas. Donc on obtient seulement les étages de départ et d'arrivée et plus leurs temps respectivement.

La partie de génération des utilisateurs est réalisée par deux fonctions du package *random*, *random.sample(l'intervalle, nombre de variables)* et *random.expovariate(λ)*. La première fonction nous permette de générer deux valeurs aléatoires dans l'intervalle défini préalablement . En fait c'est les étages de départ et d'arrivée. La deuxième retourne une valeur aléatoire positive suivant la loi exponentielle dont l'espérance est 1/λ. Cette valeur est la distance d'apparition entre deux utilisateurs voisins. Elle peut varier de 0 à infinie mais son espérance est fixée.

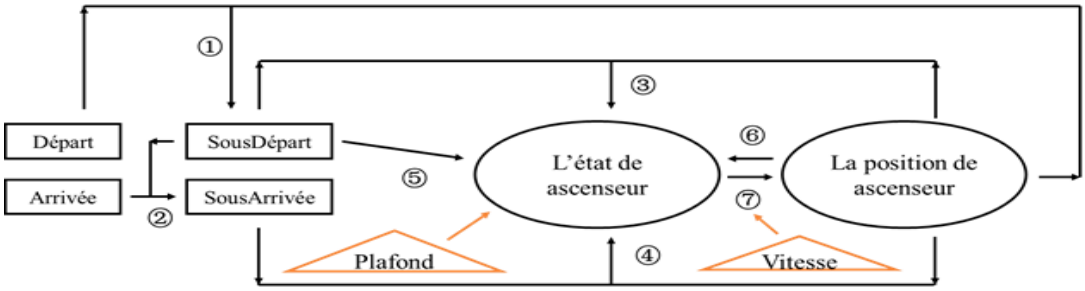
Cette édition est simple et inefficace, mais il a totalement réalisé cette simulation. Pour aller plus loin, on va ajouter la position des ascenseurs et améliorer l'algorithme pour le rendre plus efficace.

2.2 Scan-look

L'algorithme de Scan, aussi appelé l'algorithme d'ascenseur, est un algorithme d'ordonnancement de disque pour déterminer le mouvement du bras du disque dans le service de lire et d'écrire des demandes. Cet algorithme est nommé d'après le comportement d'un ascenseur du bâtiment, où l'ascenseur continue de se déplacer dans sa direction actuelle, vers le haut ou vers le bas, jusqu'à non utilisateur effectué.

2.2.1 Transport

En réalisant cet algorithme, on jette le package de Simpy pour améliorer l'efficacité. Par ailleurs, on crée une fonction de transport qui sert contrôler le comportement des ascenseurs.



1 : on trouve un ascenseur le plus proche à l'étage de départ d'un demande dont la direction de mouvement actuel est le même à celle de demande d'utilisateur, et après le demande est distribué à une liste de *SousDépart* de cet ascenseur.

2 : Les étages d'arrivée sont distribués à une liste de *SousArrivée* de cet ascenseur selon la relation entre *Départ* et *SousDépart*.

3 : L'ascenseur prend un utilisateur si son demande est dans *SousDépart* de cet ascenseur et la position de l'ascenseur est identique à l'étage de départ.

4 : L'utilisateur sort de l'ascenseur si la position de l'ascenseur est identique à l'étage d'arrivée.

5 : S'il y a des demandes dans la liste de SousDépart, l'ascenseur va effectuer ces demandes par modifier l'état à monte ou descend. De plus, s'il n'a y plus de demande active, l'état se transforme à l'arrêt.

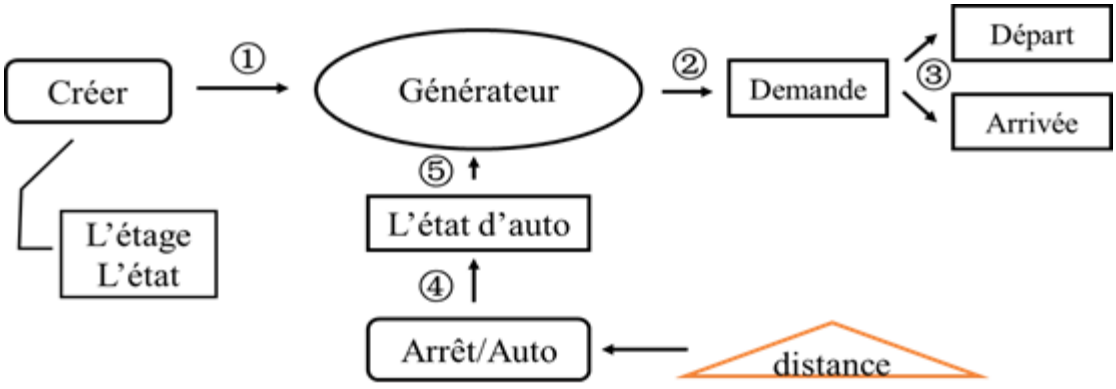
6 : L'ascenseur change sa position selon l'état, soit un pas en haut à l'état de *monte*, soit un pas vers le bas à l'état de *descend*. Sa position ne change pas si l'état est *arrêt*.

7 : Si l'ascenseur est arrivé au fond ou à l'étage sur le toit, son état se transforme à l'arrêt afin d'éviter des bugs de la position.

Il y a une valeur du plafond modifiable qui limite le nombre de l'utilisateur dans un ascenseur. De plus le paramètre de la vitesse contrôle la vitesse de mouvement des ascenseurs.

2.2.2 Génération des utilisateurs

Succédant à la première édition, le générateur des utilisateurs est contrôlé par deux commandes. Pour simplifier l'opération de demande, l'étage d'arrivée est toujours aléatoire. Il se compose par des éléments suivants :



1 : Le bouton de *Créer* qui porte deux paramètres, l'étage de départ et l'état de la direction demande au générateur de créer un utilisateur.

2 : Le générateur crée un utilisateur dont l'étage de départ et la direction de mouvement sont donnés par les entrées. L'étage d'arrivée est une valeur aléatoire dépendant aux entrées.

3 : Une demande d'un utilisateur distribue ses informations à deux listes, *Départ* et *Arrivée*.

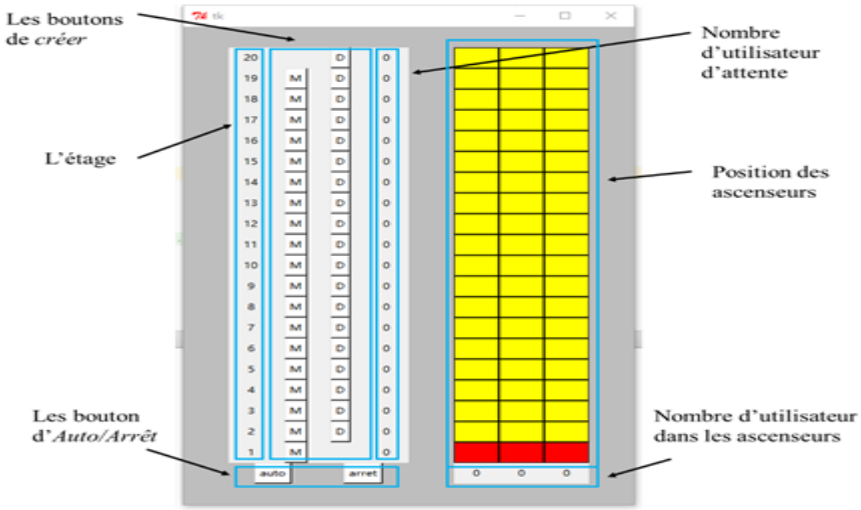
4 : Deux boutons qui contrôlent la fonction d'auto-génération modifient la clé, *L'état d'auto*, soit active, soit inactive.

5 : Si *L'état d'auto* est actif, il donne deux valeurs aléatoires, respectivement l'étage de départ et la direction de mouvement. Le générateur prend les valeurs comme les entrées dans procédure 1.

La distance contrôle le temps d'attente entre deux apparitions d'utilisateur. Elle est le paramètre de loi exponentielle aussi qui est comme la distance dans chapitre 2.1.

2.2.3 Interface par tkinter

On introduit une interface à l'aide du package tkinter. A part des fonctions dessus, on ajoute quelques labels qui affichent le nombre d'utilisateurs d'attente dans chaque étage, le nombre d'utilisateurs dans chaque ascenseur. La figure est un exemple de 20 étages totaux et de 3 ascenseurs équipés.



Les étages sont affichés par les chiffres de 1 à *nom_etage*, un paramètre qui détermine le nombre total d'étage. Les boutons de *créer* se séparent par deux classes, *M* et *D* qui représentent l'état de montée et de descend respectivement. Les boutons d'*Auto/Arrêt* modifient la clé de *L'état d'auto* contrôlant la fonction d'auto-génération. Le nombre d'utilisateur d'attente est à côté des boutons de *créer*. A droite, il y a des chiffres au fond qui précisent le nombre d'utilisateur dans les ascenseurs. D'ailleurs, des ascenseurs sont présentés par les rectangles rouges.

3. Conclusion

Les ascenseurs sont très présents dans notre vie quotidienne. On voit donc tout l'intérêt de la réalisation de la programmation de cet automate en utilisant la simulation à événement discrets . Cela nous a permis en effet de programmer et de comprendre un exemple que l'on rencontre fréquemment dans notre vie quotidienne et que l'on pourrait peut être avoir à faire dans notre vie professionnelle.

Après avoir essayé et comparé de différents algorithmes, on a finalement réalisé notre objectif du projet par la programmation basée sur la méthode SCAN-LOOK. Notre algorithme a facilité l'étape de distribution des utilisateur aux ascenseur pour minimiser le temps de transport et le temps d'attente.

De plus, pour aller plus loin dans notre projet, il nous reste encore beaucoup de questions intéressantes à réfléchir. On peut en fait améliorer notre programme en ajoutant d'autres fonctions, comme des boutons secours d'appel, un parking souterrain dans le bâtiment,etc.