

28/11/2017

Rapport TP2 LOG2810

(STRUCTURES DISCRETES)

**Auteurs :**

Etienne Laval 1916458

Constantin Bouis 1783438

Xavier Ralay 1786307

# 1.Introduction

Dans le but de venir à nouveau en aide au même étudiant du département du génie informatique et logiciel, nous avons mis au point un projet répondant à ses objectifs.

Ce dernier, étant toujours en stage dans la start-up de drones, est cette fois ci en charge de traiter les requêtes de livraison d’un point de vue logistique. En effet, il doit notamment s’occuper de la partie du logiciel gérant les requêtes et leur acheminement. Cependant, étant en charge d’autres projet, il n’aura pas le temps de tout finir à temps. De plus, cette partie du logiciel utilise des notions du cours de LOG2810, plus précisément sur la théorie des langages. C’est pourquoi il a encore besoin de notre aide. Nous allons donc l’aider a implémenter les composantes de son nouveau logiciel.

Dans ce rapport, nous allons commencer par vous présenter et expliquer les moyens auxquels nous avons eu recours pour mener a bien notre solution afin d’aider cet étudiant. Ensuite, nous allons vous faire part des difficultés rencontrées lors de l’élaboration de notre solution.

# 2. Automates

Il s’agit alors de créer un automate de reconnaissance de langage et de l’utiliser dans un programme qui s’inscrit dans la lignée du TP1. Le module développé s'articule autour de deux principaux sous modules: l'automate de reconnaissance de langage et le contrôleur de drones.

## 2.1 AUTOMATE DE RECONNAISSANCE DE LANGAGE

Un automate (*classe StateEngine*) comprend :

- un état de début

- un état courant

Un Etat (*classe State*) a :

- un nom

- un booléen indiquant s’il est final

- une liste de transitions

Une transition (*classe transition*) comprend :

-un état suivant

-une fonction contrôlant le passage à l'état suivant, en fonction d'un input donné

Fonctionnement :

Avec cela, pour faire fonctionner un automate, lorsqu'on a un input donné, on regarde parmi les transitions de l'état actuel si une de ses transitions permet un passage à un autre état. Le cas échéant, on passe l'état suivant en état courant et on regarde si cet état est final.

Pour construire la machine on opère le même processus mais on ajoute une transition lorsqu'elle n'existe pas et un nouvel état au bout de celle-ci.

Pour le cas de la reconnaissance de langage demandée, on crée une classe héritée (*RecogniserStateEngine*) de celle des automates qui a la particularité de faire des fonctions de passage de transition qui valident une égalité de String.

## CONTROLEUR DE DRONES ET STATIONS

Un contrôleur (*classe StationController*) a :

- un automate de reconnaissance de langage

-une liste de stations

-une liste de drones lourds, une autre de drones légers et une troisième qui est la concaténation des deux premières

-un certain nombre de compteurs pour les statistiques

Un drone (*classe Drone*) a :

- Une station de position et une de destination.

- Un nombre de colis

- Une capacité (un entier représentant des grammes) et une capacité maximale (sous le même format)

Deux classes héritent de drone : les drones lourds (Drone2) et les drones légers (Drone1), chacun ne variant de leur classe parent que par leur capacité maximale.

Une station (*classe Station*) a :

- un nom

- une liste de drones l'occupant

- une liste de requêtes attendant d'être exécutées

Une requête (*classe request*) a :

- un nom de station de départ

- un nom de station d'arrivée

- un poids (en grammes)

- un type de drone ayant priorité sur elle (Drone1-léger ou Drone 2-lourd)

Fonctionnement :

La fonction de création de *StationController* va chercher les noms de station et créé un automate avec, elle crée aussi les listes de stations correspondantes et les 15 drones à vide.

La fonction *TraiterLesRequetes()* successivement lit les requêtes et les crée dans les stations concernées, puis elle équilibre les drones vides, elle les charge, les déplace, puis les décharge d’un de leurs colis et les ramène immédiatement au dépôt le cas échéant, pour qu’ils soient prêt au round suivant.

La fonction d’équilibrage regarde quels sont les besoins de chaque type de station et tel ou tel type de drone (suivant le poids des requêtes et le nombre de celles-ci pouvant aller au même endroit). Puis elle y envoie les drones en commençant par les lourds disponibles pour les station les requérants, puis les légers, et enfin, les drones restants sont indifféremment envoyés aux demandes restantes.

La fonction de chargement charge les drones qui sont vides et sur des stations en besoin. Ils prennent le premier colis qu’ils peuvent porter, puis autant d’autres ayant la même destination que possible.

Remarque

1. Le système fonctionne, on lui avait même adjoint une marque de type de drone prioritaire sur les requêtes concernées pour vider les stations le plus vite possible, le problème étant que si on demandait à un drone léger d’ignorer la première requête car elle était adaptée à un drone plus gros, on risquait de violer la règle du Fifo. On s’est donc juste contenté d’envoyer en priorité les porteurs lourds, puis les légers de sorte que puisque les drones chargent dans leur ordre d’arrivée, les gros puissent emporter ce qu’ils peuvent. Il n’empêche que si on ne fait pas des cycles à vide mais uniquement un enchaînement des requêtes (.txt) données, le système n’arrive pas à désengorger complètement.

2. Il est possible de passer en mode débug en changeant la variable debugMode à True dans misselaneous.py, et ainsi avoir plus d’informations que ce qui se passe dans le code.

C’est donc par tous ces moyens que nous avons pu mener à bien notre solution pour le TP. Cependant, comme mentionnés précédemment, on a rencontré plusieurs difficultés lors de l’élaboration du TP.

# 3.Difficultés rencontrées et solutions

Dans cette partie nous allons donc vous faire part de ces difficultés.

Pour commencer, la principale difficulté de ce TP a été d’appréhender le python qui est un langage très souple et qu’aucun de nous n’avait déjà utilisé (d’où ce choix). Nous avons donc travailler sur la programmation en python, ce qui s’est avéré être moins compliqué que prévu.

La plupart du débogage s’est fait sur des fonctions considérées comme des attributs et donc pas utilisées.

Outre cela, il s’est agi de traiter des problèmes d’optimisation pas toujours solubles.

Enfin, la division des tâches au sein de notre groupe n’était pas toujours évidente, certaines des fonctions demandées n’étaient pas forcement très claires, nous avons donc due concevoir un programme souple pour pouvoir nous adapter aux précisions qui sont venues après le début du projet.

# 4.Conclusion

Finalement, ce laboratoire nous a permis de découvrir un nouveau langage de programmation, le python. Un langage que nous comptons réutiliser étant donnée sa souplesse et sa portabilité.

Aussi, nous avons pu concrétiser nos apprentissages concernant les automates et la reconnaissance de langage.

Par ailleurs, nous avons appris a être plus souple dans notre programmation de manière a faire un code adaptable en fonction de certaines spécifications qui viennent après le début du projet.

Enfin, si on ne comptabilise pas le temps passé sur ce rapport qui a été fait au fur et a mesure que nous avancions le code, nous avons passé une bonne dizaine d’heures sur le code.