Département de génie informatique et de génie logiciel

LOG2810
Structures discrètes

TP2 - Automates

Soumis par :
Chelsy Binet (1850411)
Jean-Simon Marrocco (1785835)
Sébastien Cadorette(1734603)

28 novembre 2017

Introduction:

La mise en situation de ce travail est celle où un étudiant qui effectue un stage auprès d'une start-up de drones nous demande de l'aide. Nous devons l'aider à traiter les requêtes de livraisons par drones ainsi que leur acheminement. Les notions théoriques vues en cours de structures discrètes peuvent être très utiles pour ce genre d'algorithme. En particulier les modules de la théorie des langages et des automates. Il faut être en mesure de valider les codes postaux fournis, d'attribuer les colis à des drones disponibles en respectant leurs caractéristiques et de répartir les drones équitablement dans chaque quartier.

Présentation de nos travaux

Emplacement des composants à implémenter

Tout d'abord, afin de vous simplifier la vie, voici une liste des composants que nous avions à implémenter, ainsi que du fichier dans lequel chacun se retrouve.

Composant	Emplacement (fichier)
creerArbreAdresses()	CreationArbreAdresses.py
equilibrerFlotte()	Flotte.py
assignerLesColis()	AssignationColis.py
traiterLesRequetes()	TraiterRequetes.py

Classe *Drone*, *Flotte* et fonction *equilibrerFlotte()*

En ce qui concerne la flotte et les drones, nous avons opté pour une agrégation. Autrement dit, nous avons une class Drone qui a comme rôle d'enregistrer, ainsi que de gérer les informations d'un drone (ex : statuts, nombre de colis, catégorie, poids, adresse, etc.). Ensuite, nous avons une class Flotte qui a comme rôle de gérer un ensemble de drones. Nous avons choisi, ici, une relation d'agrégation puisque cela nous laisse la liberté d'ajouter un nombre voulu de drones grâce à la méthode *addDrone()*. De plus, c'est dans la classe Flotte qu'on retrouve la

méthode *equilibrerFlotte()*. Premièrement, pour cette méthode, nous donnons à Flotte un dictionnaire de dépôts ayant comme clé les adresses et comme contenu le nombre de drones, ainsi que les drones à cette position.

__listDepots = { adresse: [nbrDrones, Drone1, Drone2]}

Nous divisons, ensuite, *equilibrerFlotte()* en deux boucles. La première permet d'identifier les dépôts où se trouvent plus de drones que le nombre permis et retire les drones d'extra. Le nombre permis de drones est déterminé à l'aide du nombre total de dépôts. Le deuxième répartit également les drones qui n'ont pas d'adresse à un dépôt pouvant en recevoir. À souligner que les drones qui ne sont pas disponibles ne seront pas touchés par l'équilibrage. En résumé, nous faisons en sorte que les drones sont partagés également à travers les dépôts. De plus, nous avons ajouté au début de la méthode *equilibrerFlotte()* un appel à la méthode *__updateStandbyTime()* qui permet de mettre à jour le statut des drones et le nombre de cycles qu'un drone doit encore attendre avant d'être disponible. Comme il est dit dans le mandat, un drone ayant plus d'un colis devra sauter des cycles dépendamment du nombre de colis supplémentaire. Donc, un drone ayant un seul colis pourra être équilibré après l'assignation.

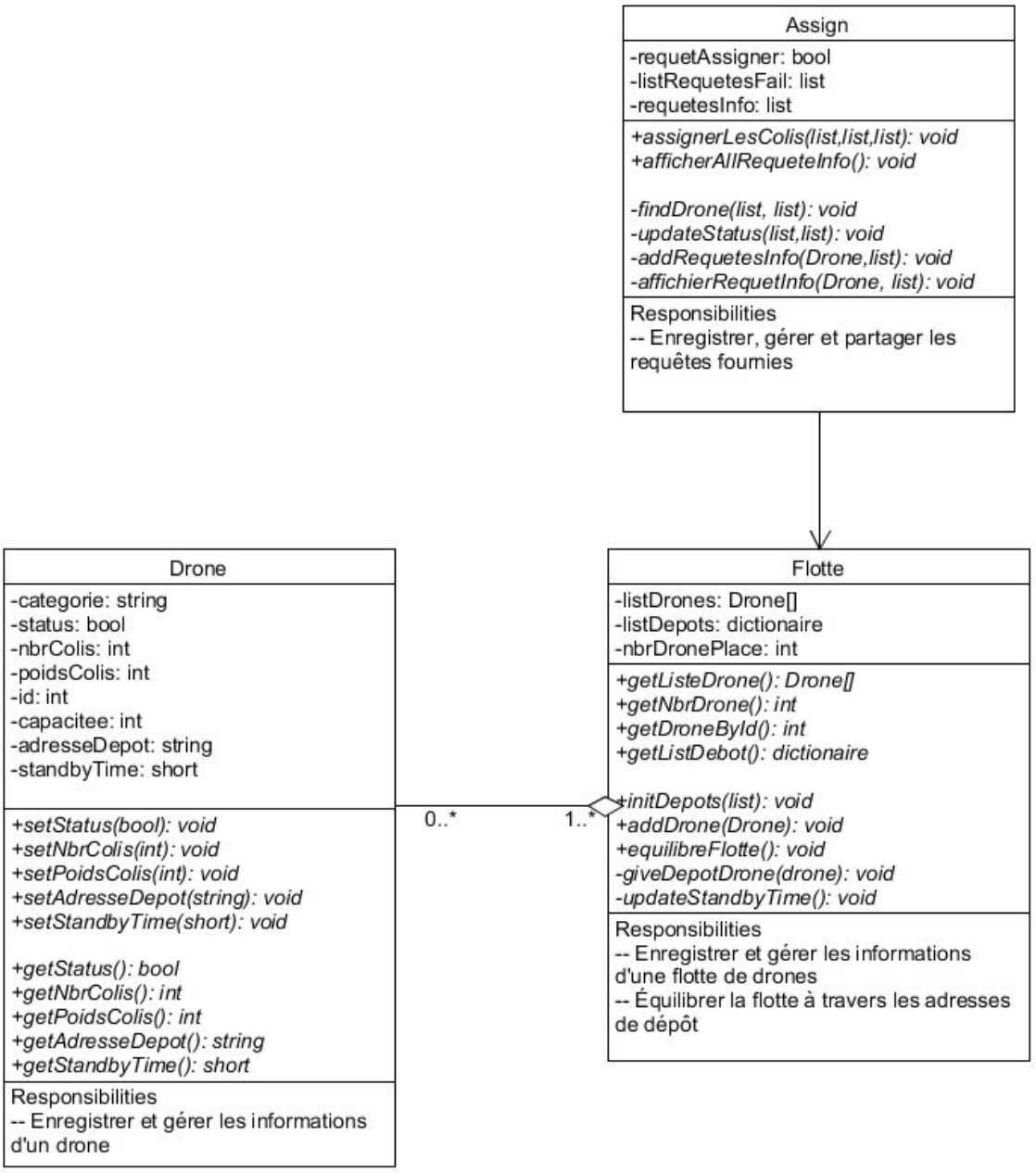

Diagramme de classes 1 : Classe *Drone, Flotte* et *Assign*.

Fonction *assignerLesColis()*

Du côté de l'assignation des colis, nous utilisons une classe *Assign*. Cette class est composée de la méthode *assignerLesColis()* et d'autres méthodes qui permettent de réduire la répétition, ainsi qu'afficher l'information. Donc, on retrouve dans la méthode *assignerLesColis()* tous les critères et conditions pour faire en sorte que les

drones sont bien assignés. Voici les étapes suivies par une requête lors de l'exécution de la méthode :

1. Nous vérifions si la requête livre à elle-même pour éviter d'utiliser un drone inutilement.
2. Nous trions les requêtes qui n'ont pas de drone à leur position de départ pour pouvoir traiter les requêtes pouvant être réglées immédiatement.
3. Si le colis n'a pas pu être assigné avec un drone à la position de départ il est ajouté la liste de requêtes à régler.
4. Après que tous les colis pouvant être assignés immédiatement ont été assignés, nous passons à la liste de requêtes restantes et essayons de trouver un drone disponible pour livrer le colis.
5. Si aucun drone n'a pu être trouvé, alors un message est donné pour aviser l'utilisateur et la requête en question est mise en attente et sera assignée au prochain cycle..
6. À la fin de la méthode, nous vérifions que tout a bien été réparti.

Classe *ArbreAdresses*

Cette classe est celle qui s'occupe de la gestion des adresses. Elle a comme attribut un automate, qui à son tour contient plusieurs noeuds. Chaque noeud est une position dans l'adresse. Alors, dans les faits, l'automate contient 6 noeuds, respectivement aux 6 caractères d'une adresse postale. Chacun de ces noeuds a un noeud de référence permettant de produire un enchaînement entre les 6 noeuds, construisant ainsi un automate. En plus de contenir l'automate, cet ensemble de classes permet de valider des adresses selon les critères fournis dans l'énoncé. C'est toutefois lorsqu'on veut vérifier si une adresse est contenue dans notre banque que l'automate se met à l'oeuvre. En appelant la fonction *contientAdressePostale(*uneAdressePostale*)*, il est possible de savoir si cette adresse est contenue ou non dans l'automate. Chaque caractère est alors vérifié selon sa position dans l'adresse et il est comparé avec toutes les possibilités répertoriées dans l'automate.

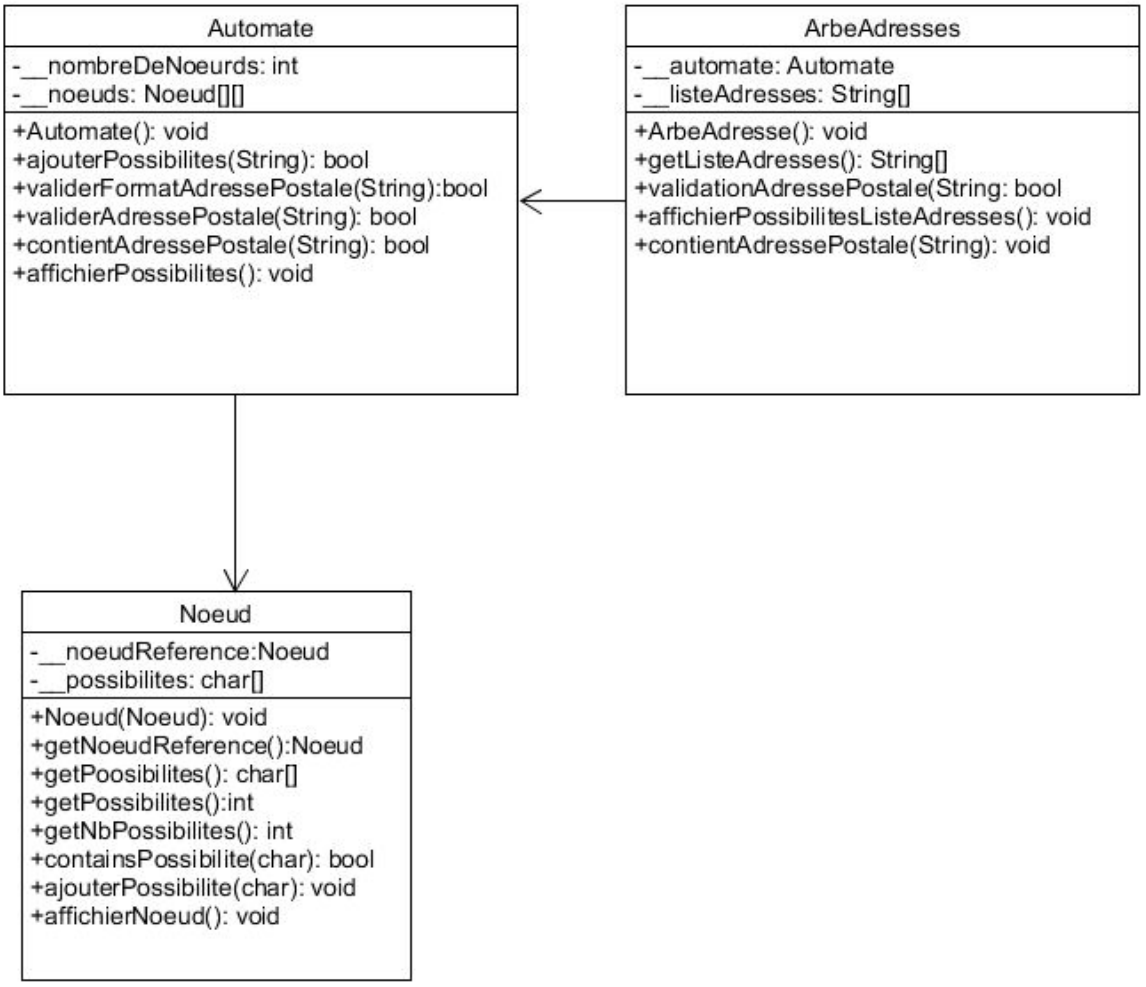

Diagramme de classe 2 : Classe *ArbreAdrsses, Automate* et *Noeud*.

Difficultés rencontrées:

Une des principales difficultés rencontrées fut le fait que souvent, l'énoncé du travail pouvait porter à de multiples interprétations. Ainsi, nous avons souvent fait un premier jet de code quand, par la suite, un courriel était envoyé pour clarifier les requis, ce qui nous amenait à constamment devoir changer ce qui avait déjà été fait. Également, les indications apportées par notre chargé de laboratoire allaient à l'inverse de ce qui était dit par Justine, apportant ainsi encore plus de questionnements. Nous avons donc tenté de décortiquer le tout en ne conservant

que les instructions ayant le plus de sens. Nous avons terminé par prendre la décision de commenter nos choix dans le code, étant donné que les indications fournies pour ce laboratoire allaient dans tous les sens. Une autre difficulté rencontrée était le manque d'expérience de codage dans le langage python, ce qui demandait un peu plus de temps de recherche. Cette difficulté fut cependant surmontée assez aisément grâce à la bonne documentation trouvée en ligne.

Conclusion:

En conclusion, nous sommes très satisfaits du travail accompli dans ce TP. Celui-ci nous a permis d'approfondir les connaissances développées dans le cours de structures discrètes et plus précisément les notions d'automates et de structures de données. En effet, nous avons pu mettre en pratique ces notions théoriques afin d'optimiser la distribution de drones de livraisons répartis par quartiers. Il s'agit là d'une excellente façon de comprendre l'utilité des mathématiques discrètes dans des scénarios d'utilisation concrets.