SimTaxi

Simulation pour l'optimisation des politiques d'utilisation de l'information à l'intérieur d'une compagnie de taxis.

Julien BURDY
Grégory BURRI
Lucien CHABOUDEZ
Alexandre D'AMICO
Vincent DECORGES
Patrice FERROT
Lionel GUÉLAT
Joël JAQUEMET

3 mars 2003

Table des matières

Chapitre 1

Introduction

1.1 Organisation

Ce document est fractionné de la façon suivante :

Introduction présentant le problème lui-même (l'énoncé), l'équipe que nous sommes, le choix des outils utilisés ainsi que le langage de programmation choisi.

Documentation de développement présente le raffinement général du problème en modules, puis décrit le rôle de chaque module ainsi que les méthodes mises en oeuvre pour y arriver. Cette partie de documentation fait un maximum abstraction de l'implémentation. L'implémentation précise est décrite par la documentation technique au format html (cf. chapitre 3)

Annexes Dans les annexes nous trouverons nos documents internes, la documentation utilisateur, le listing des sources, le journal de travail, etc...

De façon à pouvoir uniformiser le travail, nous avons créé nos propres documents internes. Soit :

Documentation qualité pour uniformiser l'écriture du code source (cf. annexe B)

Documentation de rédaction pour uniformiser l'écriture de la documentation (cf. annexe C)

1.2 Énoncé

Il s'agit de développer un programme de simulation visant à optimiser des politiques d'utilisation de l'information à l'intérieur d'une compagnie de taxis.

1.2.1 Description du système

- Un certain nombre de taxis circulent dans la ville, leur activité étant partiellement coordonnée à partir d'un central. Un certain nombre de stations, dans lesquelles les taxis attendent leur prochaine course, sont réparties dans la ville.
- Lorsqu'un taxi a fini sa course, il se dirige vers une station (choisie en fonction d'une certaine politique) dans laquelle il reste au moins une place de libre. Il transmet cette information au central.
- Au moment où le central reçoit un appel d'un client demandant un taxi, il transmet l'ordre de course à un taxi (choisi selon les critères de la politique en cours). Il peut s'agir soit d'un taxi en station (donc libre) ou d'un taxi qui est en train de retourner à une station.
- Le but de la compagnie de taxis est de minimiser le nombre de kilomètres parcourus par les taxis à vide (sans clients).

- Les carrefours sont modélisés par des sommets et les morceaux de rues (entre 2 croisements) sont modélisés par les arêtes d'un graphe orienté et connexe. On supposera que les stations de taxis et les points de départ/fin d'une course se trouvent au milieu des arêtes du graphe. Pour tous les trajets effectués, les taxis emprunteront les chemins les plus courts.

1.2.2 Modélisation

La modélisation réaliste de ce problème est une tâche dépassant largement le cadre du projet. C'est pourquoi nous n'avons pas simulé les fluctuations de la demande et l'encombrement du trafic en fonction de l'heure (heures de pointe, heures creuses). Pour simplifier, nous avons donc supposé que nous nous trouvions dans une situation de charge moyenne. Nous avons néanmoins veillé à respecter certains ordres de grandeurs au niveau :

- du réseau : le graphe doit comporter 800 sommets (carrefours) et 1400 arêtes (morceaux de rues). Le diamètre du graphe doit être de 12 km et le temps maximum pour aller de 2 points se situant à chaque extrémité doit être d'environ 20 min.
- de la demande : globalement, on compte environ 2700 courses à effectuer chaque jour. En moyenne, une course représente environ 8 km (distance de transport d'un client).
- des stations : le graphe comprend 30 stations, une de 20 places, deux de 10 places et le reste avec 5 places.
- des taxis : 180 taxis circulent en permanence et effectuent en moyenne 160 km par jour.

1.3 Équipe

L'équipe de SimTaxi est constituée de huit étudiants de la classe EIA-5 de l'eivd[1]. La répartition du travail à été faite de la manière suivante :

Julien Burdy	Chef de groupe
Grégory Burri	Interface utilisateur
Lucien Chaboudez	Gestionnaires taxis et stations
Alexandre D'Amico	Échéancier et central
Vincent Decorges	Central et politiques
Patrice Ferrot	Initialisations et génération des courses/clients
Lionel Guélat	Algorithmique dans le réseau
Joël Jaquemet	Génération et structure du réseau/graphe

Nous avons eu environ 30h/personne à disposition.

1.4 Choix du langage

Pour réaliser ce programme, nous nous sommes tournés vers le langage Python (interprété, interactif, orienté-objet, portable), permettant une rapidité de développement imbattable à notre connaissance mais au profit d'une perte de performance de la simulation même.

1.5 Outils

Toute la documentation a été créée avec LaTeX[2]. La gestion des différentes versions de chacun des fichiers du programme a été réalisée grâce au CVS[3] de Sourceforge[4]. Pour les schémas UML,

[1] http ://ina.eivd.ch
[2] http ://www.latex-project.org
[3] http ://www.cvshome.org
[4] http ://www.sf.net

nous avons eu recours à Umbrello[5]. L'extraction de documentation technique (depuis les sources) à été faite avec HappyDoc[6].

1.6 Environnement et librairies

L'interface utilisateur est implémentée à l'aide de wxPython[7], l'affichage du réseau avec PyOpenGL[8] et le tout est sur un environement Python[9] récent.

[5] http ://uml.sf.net
[6] http ://happydoc.sf.net/
[7] http ://www.wxpython.org
[8] http ://pyopengl.sf.net
[9] http ://www.python.org

Chapitre 2

Documentation de développement

2.1 Raffinement général

La décomposition du problème s'est faite de façon assez rapide. Nous avons fractionné au maximum pour faciliter le travail en groupe. La figure 2.5 montre en un coup d'oeil cette décomposition.

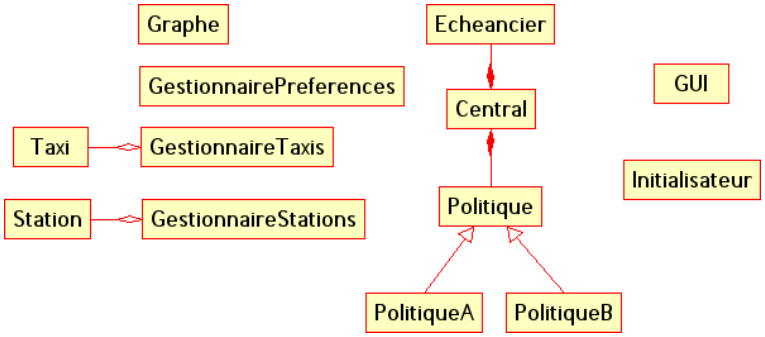

FIG. 2.1 – *Raffinement général des modules*

Nous avons rapidement identifié un graphe, contenant les données cartésiennes des différents carrefours, facilitant ainsi l'affichage et la génération du réseau. C'est aussi à lui qu'on demande les chemins les plus courts d'un point à un autre. Il est unique et accessible depuis n'importe quel autre module de la simulation.

Différents gestionnaires, qui permettent de regrouper et de gérer différents éléments, comme les taxis, les stations, les paramètres, etc... Ils évitent (dans les cas des gestionnaires de taxis et de stations) aux autres modules de devoir itérer toutes les instances pour faire leur sélection («quel est le taxi le plus proche de ?»).

Le module d'initialisation s'occupe de la génération de la demande (les clients) et de la mise en place des stations (aléatoire) et des taxis.

Ensuite, comme toute simulation classique, nous avons un échéancier contenant les événements. Événements qui sont traités depuis le module central (ce qui est assez proche de la vie réèlle).

Les modules politiques définissent le comportement des taxis. Avant chaque décision le central consulte la politique courante (qui pourra être modifiée en cours de simulation) pour savoir quel taxi choisir ou où va se diriger un taxi. La figure **??** image un peu la sélection d'un taxi.

Il sera assez facile d'écrire une nouvelle politique, en choisissant parmi les méthodes fournies par les gestionnaires pour sélectionner un taxi ou un lieu (station).

Actuellement les gestionnaires sont assez pauvres et ne permettent qu'une politique du plus proche.

Le GUI permet de visionner l'état du graphe, l'emplacement des stations, l'apparition et la disparition de clients ainsi que les mouvements des taxis. Il est indépendant de la simulation, il ne fait que consulter l'état et l'emplacement des éléments pour ensuite les afficher.

Ensuite (2ème semestre) il nous permettra de consulter et modifier les paramètres de la simulation et de consulter les statistiques.

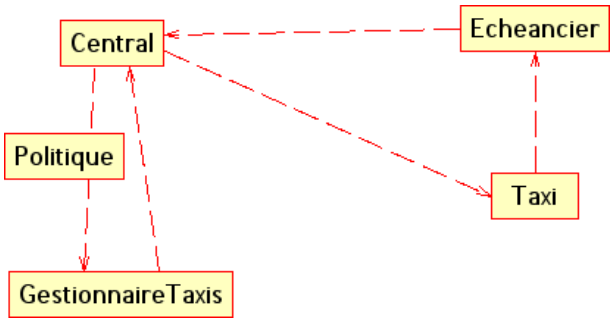

FIG. 2.2 – *Communication au coeur de la simulation*

2.2 Le central

Le central est le coordinateur global, c'est lui qui gère tout. Il comprend :

- **Un échéancier** : contient les événements(Client, Charger un client, Poser un Client et Arriver en station).
- **Un gestionnaire de taxis** : gère les taxis, fourni un taxi libre suivant la politique utilisée.
- **Un gestionnaire de stations** : gère les stations, indique une station pas pleine suivant la politique utilisée.
- **Une politique** : détermine le choix d'un taxi libre plutôt qu'un autre pour un client donné. C'est elle aussi qui détermine le choix de la station à laquelle le taxi se rendra après sa course.

2.2.1 Les événements

Les événements sont au coeur de SimTaxi, c'est eux qui déclenchent les actions à effectuer lors de la simulation.

Ils sont de 4 types :

- **Client** : correspond à une demande d'un client.
- **Charger un client** : un taxi va charger un client.
- **Poser un client** : un taxi dépose le client à sa destination.
- **Arriver station** : un taxi arrive à une station.

Chaque événement dispose d'un champ *temps* qui indique à quel moment ils doivent survenir. Tous les événements sont stockés dans un échéancier (voir 2.2.2) dans leurs ordre chronologique.

Implémentation

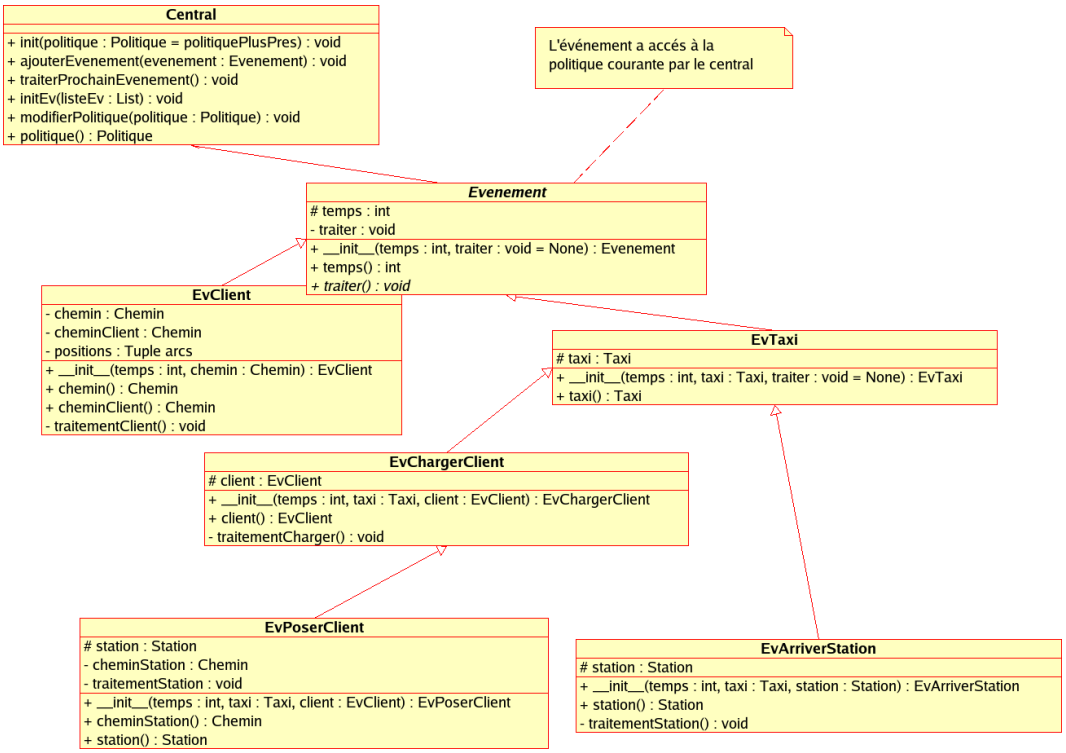

FIG. 2.3 – *Diagramme de classe des événements*

Le diagramme de classe (figure 2.3) montre l'implémentation des différents événements. Tous les événements dérivent de la classe abstraite *Evenement* qui défini leurs structures de base. Le champ *temps* permet de fixer à quel moment l'événement va avoir lieu et le champ *traiter* permet d'associer une action à un événement. Un appel à la méthode *traiter* provoque l'exécution de celle-ci.

Les actions peuvent être définies à l'extérieur de l'événement (sous-programme externe) ou à l'intérieur de celui-ci. Pour SimTaxi tous les événements définissent eux-mêmes leurs actions.

Voici la liste des actions pour chaque événements.

EvClient :

1. Calcul le chemin de la position du client à la destination.
2. Demande à la politique courante (voir 2.6) un taxi.
3. S'envoie au taxi.

EvChargerClient :

1. S'envoie au taxi.

EvPoserClient :

1. Demande à la politique courant une station.
2. S'envoie au taxi.

EvArriverStation :

1. S'envoie au taxi.

Séquence des événements

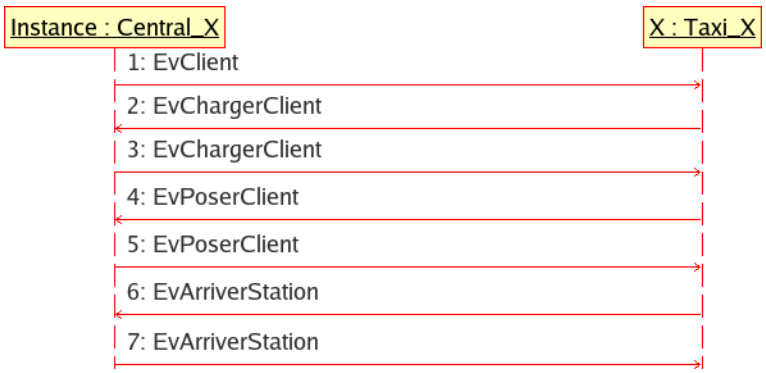

FIG. 2.4 – *Séquences des événements*

La figure 2.4 montre l'enchaînement des évènements pour une course.

On peut la résumer ainsi :

1. Un événement client (*EvClient*) est sortis de l'échéancier du central. Il est envoyé à un taxi par le biais de sa méthode *traiter*.
2. Le taxi reçoit l'*EvClient*. Il calcul à quelle date il va charger le client et peut ainsi créer un événement charger client (*EvChargerClient*) qu'il transmet au central.
3. Le central transmet l'*EvChargerClient* (méthode *traiter*) au taxi lorsque la date est atteinte pour lui dire qu'il a chargé un client.
4. Le taxi calcul la date à laquelle il va déposer le client et envoie l'événement *EvPoserClient* au central.
5. Le central envoie *EvPoserClient* au taxi lorsque la date est atteinte.
6. Le taxi calcul la date d'arrivée à la station est transmet *EvArriverStation* au central.
7. Le central transmet *EvArriverStation* au taxi pour lui dire qu'il est arrivé à la station.

2.2.2 L'échéancier

Description

L'échéancier a pour but le stockage des événements à venir. Ils sont ordonnés dans un ordre croissant, en fonction de la date à laquelle aura lieu l'événement.

Implémentation

L'échéancier n'est autre qu'une queue de priorité, avec une insertion dichotomique.

2.2.3 Etat actuel

En l'état le central coordonne tout. Il traite les différents événements en conséquence. La suppression d'un événement a été implémentée pour la raison suivante. Si l'on désire détourner un taxi se dirigeant vers une station (il est donc vide) car il est plus proche du client que les taxis se trouvant en station, il faudra donc supprimer son événement d'arrivée en station ainsi que sa réservation. La suppression n'a pas encore été utilisée du fait de la politique employée jusqu'à présent choisi le taxi se trouvant dans une station la plus proche du client.

2.3 Le réseau (graphe)

Le rôle du réseau et de modéliser la carte routière sur laquelle nous voulons faire une simulation.

Nous avons donc utilisé un graphe pour représenter le réseau. Ce graphe est orienté pour permettre de définir des sens uniques. Un dictionnaire est utilisé pour implémenter le graphe avec comme clés, les sommets. A chaque clé et associée une liste contenant en première position les attributs du sommet et ensuite les sommets qui sont reliés depuis le sommet décrit par la clé. On a donc pour chaque entrée du dictionnaire : un sommet, ses attributs et ses arcs sortant. Nous avons implémenté toutes les méthodes "usuelles" permettant de manipuler un graphe et deux méthodes permettant d'enregistrer et de récupérer un graphe dans, et respectivement à partir d'un fichier.

2.3.1 Génération d'un reseau

Il nous a fallu créer une méthode permettant de générer un réseau avec les contraintes suivantes : nombre de carrefours, nombre de portions de rues (orientées) et distance maximale entre 2 carrefours). Nous ne voulions pas avoir un réseau "américain" (rues perpendiculaires), nous avons donc imaginé un algorithme qui donne un résultat proche des plans de villes réels.

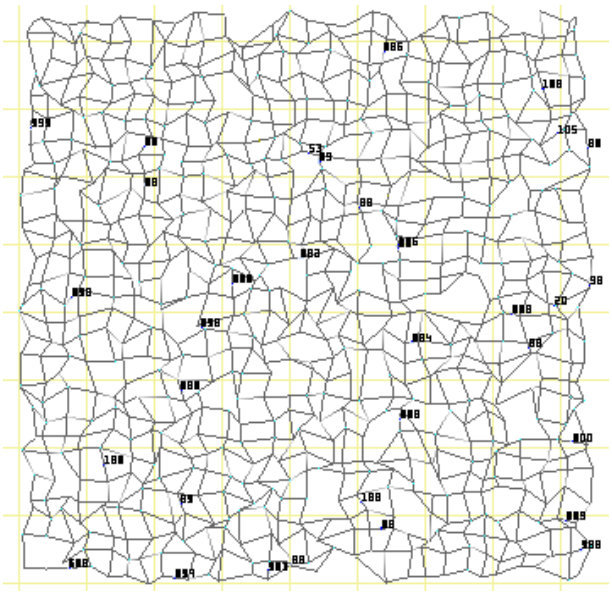

FIG. 2.5 – *Un graphe résultant de la génération*

Pour ce faire, nous formons dans un premier temps un réseau "américain" contenant le nombre suivant de carrefours :

$$\lfloor \sqrt{nbCarrefoursDemands} \rfloor^2$$

Chaque carrefour est cependant placé aléatoirement dans un périmètre qui lui est reservé (les périmètres ne se chevauchant pas) pour supprimer la forme "américaine" du réseau ! Ensuite nous créons aléatoirement une rue (bi-directionnelle) qui passe par tous les carrefours pour assurer la connexité du réseau. Il ne nous reste plus qu'a supprimer aléatoirement des carrefours (en garantissant la connexité) jusqu'à en avoir le nombre demandé et qu'a ajouter (toujours aléatoirement) le nombre de rues manquantes.

2.3.2 Recherche du plus court chemin (PCC)

Pour l'acheminement des clients et pour les décisions du système, il est nécessaire de connaître la distance et le chemin le plus court d'un endroit à un autre du graphe. D'après la donnée du problème, nous avons des stations et des clients situés au milieu des arcs. Il s'agit donc de trouver le chemin le plus court d'un arc à un autre. Le chemin renvoyé contient tous les sommets/arcs du chemin ainsi que la distance totale. Pour pouvoir afficher un taxi en chemin à un moment donné, il faut connaître sa position dans ce chemin.

PCC entre deux arcs

Pour trouver un PCC entre deux arcs, on aurait pu imaginer de compléter le graphe avec des sommets au centre de chaque arc. Mais pour minimiser la taille du graphe, nous avons choisi plutôt de rechercher et de comparer les chemins depuis les sommets qui composent les extrémités de chacun des deux arcs. Nous avons donc quatre chemins à calculer pour un PCC entre deux arcs bidirectionnels.

Arbre des PCC

Pour trouver les PCC entre deux sommets nous utilisons l'algorithme de Dijkstra qui va en fait nous donner l'arbre des PCC d'un sommet de départ. Pour "amortir" ces informations calculées et non utilisées, nous allons les stocker dans une donnée membre du graphe. Ceci nous permettra d'éviter le recalcul du même arbre lors d'appels ultérieurs et donc d'économiser du temps processeur. Ainsi, pour trouver un chemin le plus court d'un sommet à un autre, on va d'abord vérifier pour les deux sommets si on a déjà l'arbre des PCC calculé. Sinon, on le construit avec Dijkstra.

PCC entre deux sommets

L'algorithme de Dijkstra parcourt tout le graphe depuis un sommet de départ et nous donne son arbre des PCC. Pour stocker et pour pouvoir construire le chemin qui nous intéresse, on va enregistrer, pour chaque sommet, le sommet qui permet de l'atteindre et la distance qui le sépare du sommet de départ. Ainsi nous pouvons retrouver le chemin en partant du sommet à atteindre et en remontant les parents jusqu'au départ.

Performances

Le stockage des arbres des PCC permet de gagner beaucoup de temps de calcul. Après quelques centaines d'appels sur le graphe de taille réelle, le temps de réponse moyen est divisé par 8. L'utilisation de la mémoire approche les 19 Mo pour ce graphe quand tous les sommets ont été demandés au moins une fois.

2.4 Les gestionnaires

Le rôle d'un gestionnaire est de regrouper des instances de différents objets. Il dérive d'un dictionnaire et 2 méthodes ont été définies pour permettre de le gérer un minimum.

2.4.1 Gestionnaire des stations

Le rôle du gestionnaire de stations est de regrouper les stations qui sont employées pour la simulation. Il doit aussi permettre d'accéder à une station en donnant son numero et de trouver la station la plus proche d'une position donnée.

Quand on demande le taxi suivant à une station, celle-ci renvoie celui qui est en tête de la liste de taxis. La politique de gestion de cette liste de taxi est FIFO.

Une méthode permet de trouver la station la plus proche d'une position donnée. Cette méthode parcourt les stations qui sont dans le gestionnaire et renvoie la plus proche. Si aucune station n'existe, une exception est propagée. La complexité de cette méthode est de O(s * PCC), "s" étant le nombre de stations.

2.4.2 Gestionnaire des taxis

A chaque fois qu'on ajoute un taxi dans le gestionnaire de taxis, une instance de taxi est créée et elle est ajoutée au gestionnaire. A ce moment, on affecte egalement le taxi directement à une station dont le no a été passé lors de la demande d'ajout d'un taxi.

Une méthode permet de trouver le taxi le plus proche d'une position donnée. Cette méthode cherche la station la plus proche du client et qui contient au moins un taxi. Si aucune station ne contient de taxi ou que le gestionnaire de taxi ne contient aucun taxi, une exception est levée. La complexité de cette méthode est en O(s * PCC), "s" étant le nombre de stations.

2.4.3 Gestionnaire de préférences

Tout comme le graphe, le gestionnaire de préférence est unique (singleton). Il est consultable et modifiable depuis n'importe où et tout au long de la simulation.

Au démarrage il charge les options contenues dans le fichier texte $config.txt$ mais celles-ci peuvent être modifiées lors de l'exécution. Aucune modification n'est faite au fichier, seul l'édition manuelle le modifie de manière permanente. Ce qui ne sera plus le cas lorsque nous aurons un interface utilisateur qui permettra de modifier les valeurs.

2.5 Initialisation

Ce module a pour but de générer les demandes des clients, les emplacements des stations de taxis ainsi que les taxis eux-mêmes, et ceci avant le début de la simulation à proprement parler. Pour ce faire, il dispose de trois méthodes dont nous allons expliquer le fonctionnement ci-dessous.

2.5.1 Génération de la demande

Afin de générer correctement l'ensemble des demandes, il est nécessaire de connaître certaines informations :

- Le graphe associé à la simulation.
- Les dates de la première et de la dernière demande.
- Le nombre de clients désirés.
- La distance moyenne des trajets des courses.

Partant de ces informations, on va générer une course en choisissant aléatoirement un arc de départ et d'arrivée sur le graphe (un client se déplace toujours du centre d'un arc au centre d'un autre arc), en prenant bien sûr garde à ce qu'ils ne soient pas identiques. Ensuite, on associera à chaque course le chemin que devra emprunter le taxi afin de conduire le client de son point de départ à son point d'arrivée. Afin de respecter la moyenne des distances des courses, on procède de la manière suivante : après avoir généré un certain pourcentage du nombre de courses demandé, on compare la moyenne des courses déjà générées avec la moyenne désirée. Si la moyenne actuelle est supérieure à la moyenne désirée et que la distance de la course qui vient d'être générée est également supérieure à cette moyenne, alors on rejette cette course. On procède de manière analogue avec la cas symétrique (moyenne actuelle et distance de la dernière course générée inférieures à la moyenne désirée). Cependant, afin d'éviter que la demande de génération des courses n'aboutisse pas en raison de l'incohérence de la moyenne demandée

(par exemple 20, alors que l'envergure du graphe n'est que de 10), on autorise un nombre de rejets maximum, après quoi on acceptera toute course, même si elle n'adapte pas la moyenne. Finalement, il faut affecter une heure à chaque client : pour le premier, ce sera l'heure de la première demande, pour le second l'heure de la dernière demande et pour tous les autres, une heure aléatoire se trouvant entre les heures de début et de fin.

Suite à apporter

L'algorithme présenté ci-dessus réalise bien le travail demandé. Cependant, cette méthode va nécessiter bon nombre d'améliorations, notamment au niveau de l'adaptation de la moyenne. En effet, l'algorithme utilisé actuellement est très coûteux : on rejette un nombre important de courses et chaque nouvelle course générée fait appel à la méthode du plus court chemin (PCC), qui est très gourmande en temps CPU, notamment pour un graphe de la taille du nôtre. Une idée pourrait être de "tronquer" une course qu'on désirerait plus courte, mais cela ne résout pas le problème d'une course qu'on désirerait plus longue. A étudier...

2.5.2 Mise en place des stations

Cette opération nécessite de connaître un certain nombre d'informations :

- Le graphe associé à la simulation.
- Le nombre de stations désirées ainsi que leur nombre respectif de places (représenté par une liste d'entiers).

Pour placer une station, nous allons tout d'abord choisir le nombre de places qu'elle devra comporter. Ensuite, on génère aléatoirement sa position sur le graphe (les stations sont situées au centre d'un arc), en prenant néanmoins garde à ce qu'une station ne soit pas déjà présente au même endroit. De plus afin d'améliorer quelque peu leur répartition, deux stations ne pourront se trouver sur deux arcs consécutifs. Finalement, nous allons indiquer au gestionnaire de stations qu'il peut ajouter une station comportant le nombre de places choisi à l'endroit précisé. Cette procédure sera répétée tant qu'il reste des stations à placer.

Suite a apporter

Cette façon de procéder conduit à des résultats corrects. Il arrive pourtant que plusieurs stations se trouvent regroupées dans une région du graphe alors qu'une autre partie de ce graphe s'en trouve dépourvues. Il serait donc utile de réfléchir à un algorithme permettant d'améliorer le positionnement des stations.

2.5.3 Mise en place des taxis

Signalons pour commencer que cette opération n'est réalisable que si le graphe associé comporte déjà des stations, faute de quoi la méthode se terminera sans avoir placé un seul taxi. Mis à part le graphe, il n'est nécessaire de connaître que le nombre de taxis à placer. Avant de commencer à placer les taxis, on récupère la liste des stations associées au graphe. Ensuite, pour placer un taxi, on choisit une station aléatoirement dans la liste et on lui affecte un taxi. Après cette affectation, si la station ne comporte plus de place, on la supprime de la liste des stations. On répète l'opération jusqu'à ce que le nombre désirés de taxis ait été affecté ou alors jusqu'à ce qu'il n'y ait plus de station dans la liste.

Suite à apporter

Cette méthode fonctionne correctement. On peut néanmoins se demander s'il ne serait pas préférable de lever une exception dans le cas où on ne peut placer le nombre de taxis désirés en raison de la capacité insuffisante des stations, plutôt que de terminer la méthode sans erreur, mais en n'ayant pas respecté la demande de l'appelant.

2.5.4 Conclusion

Le choix de générer l'ensemble des demandes avant le début de la simulation peut surprendre : il aurait peut-être semblé plus naturel de les générer durant la simulation, en utilisant par exemple une loi de Poisson pour générer la date de la prochaine demande. Cependant, nous avons fait ce choix notamment pour résoudre le problème lié à la distance moyenne des courses. D'autre part, cette façon de procéder à l'avantage de décharger quelque peu le processeur durant l'exécution de la simulation.

2.6 Les politiques

2.6.1 Description

SimTaxi permet de définir différentes politiques de gestion des courses. Les politiques permettent de répondre aux deux questions suivantes : *Quel taxi va prendre en charge un client ?* et *A quel station un taxi va se rendre une fois sa course terminée ?*.
SimTaxi fournit les primitives pour permettre de répondre à ces deux questions. Une politique consiste à assembler ces différentes briques.

2.6.2 Implémentation

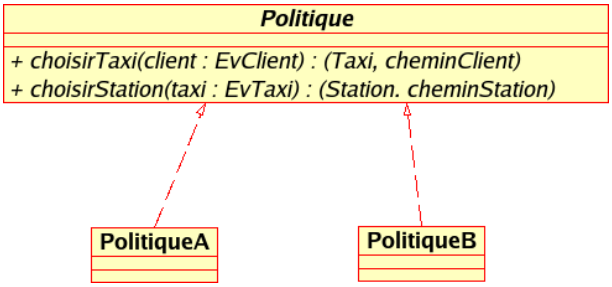

FIG. 2.6 – *Diagramme de classe des politiques*

La classe abstraite *Politique* (voir figure 2.6) fournit l'interface permettant de réaliser une politique. Une politique doit redéfinir les deux méthodes abstraites *choisirTaxi* et *choisirStation* qui permettent réciproquement de choisir le taxi à affecter un client et de choisir une station pour un taxi.

C'est les gestionnaires de stations et de taxis (voir 2.4) qui fournissent les briques de base à la réalisation de la politique en mettant à disposition des sous-programmes permettant de retourner le taxi le plus proche d'un client, ou un taxi d'une station particulière etc... . Il en est de même pour les stations.

Il est possible de changer à tous moment de politique (même en cours de simulation) grâce à la méthode *modifierPolitique* du central.

2.6.3 État actuel

Pour l'instant seul la politique du "plus proche" a été implémentée (voir *PolitiquePlusPres.py*). Cette politique affecte à un client le taxi le plus proche de lui qui n'est pas en train de se rendre à une station ou d'effectuer une course. Une fois que le taxi a posé son client, il rentre à la station la plus proche de lui. Cette politique est celle prise par le central par défaut.

2.6.4 Suite a apporter

Au cours du deuxième semestre plusieurs autres politiques seront implémentées. Notamment une politique du plus proche qui affecte à un client le taxi le plus proche de lui en prenant en compte les taxis rentrant à une stations. Pour la réalisation de ces différentes politiques, il sera nécessaire de rajouter des primitives dans les gestionnaires.

2.7 Interface graphique

Le but de l'interface graphique est de fournir un support visuel au déroulement de la simulation, elle permet, en temps réel, de voir l'évolution des taxis, l'état des stations ainsi que d'autres données moins rapidement cernable au travers des chiffres.

Ceci est important afin de rapidement se rendre compte du bon fonctionnement des différents algorithmes, le comportement des taxis peut être analysé en un clin d'oeil afin d'en vérifier la cohérence.

Il est primordial d'avoir une vision graphique du problème, ceci afin de faciliter sa compréhension et de pouvoir plus rapidement imaginer de nouveaux algorithmes.

2.7.1 Interface utilisateur

L'interface de SimTaxi est réalisée avec wxWindows pour Python (wxPython), le choix s'est naturellement orienté vers cet outil car il possède beaucoup de points forts comme sa facilité d'utilisation, son grand choix de composants (menu, bouton, etc..) et bien sûr sa gratuité.

État actuel

Seulement de petits essais ont été effectués au niveau de l'interface graphique, il n'y a actuellement que l'affichage qui est fait.

Suite à apporter

Il faut encore réaliser toute l'interface utilisateur. On peut diviser l'interface en trois parties :

- Une partie contrôlant le déroulement de la simulation. Il faut pouvoir changer le temps : accélérer ou ralentir la simulation, mettre en pause ou recommencer une simulation.
- Une autre partie concernant les paramètres de la simulation comme le nombre de taxi, le nombre de station etc.
- Finalement, une partie affichant différents statistiques et résultats de la simulation.

2.7.2 Interface OpenGL

Au regard du nombre de taxis, routes et stations à afficher en temps réel il est nécessaire d'utiliser une bibliothèque graphique de bas niveau contrairement aux outils de dessins fournis par wxWindows. Il faut donc un affichage rapide et pouvant permettre l'affichage d'animation (double tampon s'affichage), le choix de OpenGL s'est alors naturellement fait.

OpenGL est à la base destiné à afficher des scènes en trois dimensions, dans notre cas il n'est pas nécessaire d'avoir une telle possibilité. La caméra est donc placée sur l'axe des Z et une projection orthogonale est utilisée pour avoir une vue en deux dimensions.

Les objets de la scène (taxis, stations etc.) sont placés sur différentes couches qui sont définies par une valeur en Z. On peut donc facilement définir quel objet est au dessus d'un autre. Par exemple les taxis sont au dessus des routes.

Chaque objet est composé à l'aide de polygones, il est important de réduire au maximum le nombre de polygone des objets, par exemple un taxi n'est formé que d'un seul polygone et une route de deux polygones, afin de diminuer le temps de calcul de l'affichage.

Etat actuel

L'affichage est actuellement terminé. Il affiche les taxis avec leur état et numéro associé, les routes et les stations avec leur nombre de places totales et leur nombre de places occupées. Le temps en heures/minutes/secondes est également affiché directement en OpenGL.

Le déplacement dans la ville se fait à l'aide de la souris en laissant appuyer le bouton gauche , il est possible de faire des zoom avant et arrière à l'aide du bouton droit.

Suite à apporter

Il est aisé, par la suite, d'ajouter facilement d'autres objets graphiques ou informations à l'affichage.

2.7.3 Conclusion

Le plus gros du travail reste dans l'interface utilisateur, ce qui sous-entend un apprentissage assez approfondi de wxWindows. Il n'y a quasiment plus rien à faire au niveau de l'affichage OpenGL.

Chapitre 3

Documentation technique

La documentation technique a été directement extraite de nos codes source à l'aide d'HappyDoc. Le format étant le HTML et ce genre de documentation n'étant pas vraiment à lire d'une seule traite, mais plutôt par partie, elle n'a pas été imprimée. Mais elle est consultable dans le dossier $web/index.html$ de SimTaxi ou directement sur le site $http : //SimTaxi.sf.net$. A noter que HappyDoc nous génère aussi notre site web.

Chapitre 4

Conclusion

Dans l'ensemble, le projet s'est bien déroulé, les problèmes de communication ont été minimes, il y a eu très peu de ligne écrite pour rien. Nos documents internes même s'ils sont minimes et peu rigides, nous ont permis de travailler de manière relativement uniforme, sans avoir à tout contrôler en permanence.

Nous avons peu de chose à remettre en cause, mais beaucoup de choses à améliorer et à créer.

4.1 État actuel du projet

Le projet actuel est plus ludique que pratique, à part regarder les clients apparaître et les taxis bouger, il n'y a encore peu de résultat concret.

Il reste des erreurs au niveau du comportement des taxis, ils ne se dirigent pas vers la station la plus proche, mais la plus proche non pleine (non respect de la donnée). Ils ne savent encore pas interrompre leur retour en station pour prendre un nouveau client qui serait apparu non loin.

Nous sommes handicapés par nos problèmes de performance, la simulation est lourde, les chargements des données persistantes ne sont pas vraiment tolérables (pourtant nous utilisons la librairie standard de Python)

4.2 Suite à apporter

En plus des éventuelles nouvelles indications, nous allons principalement nous occuper de la création du modules statistique. Dans le même temps, certains problèmes devront être réglés, et d'autres modules devraient être optimisés. Le comportement des taxis n'est pas encore totalement correct. Il y aura aussi la création définitive de l'interface utilisateur ainsi qu'un système agréable pour consulter et stocker les statistiques.

Annexe A

Journal de projet

2002-10-23 Formation du groupe. Choix du projet.

2002-10-30 Reçu énoncé du projet par ETR. Discussion générale.

2002-11-06 Reçu transparents de simulation par ETR. Proposition d'un design. Discussions puis acceptation de celui-ci. Annulation de ce design réorientation (ETR).

2002-11-13 Présentation d'un autre design orienté par ETR. Séparation des modules. Répartition des tâches.

2002-11-20 Discussions plus détaillées sur les interfaces. Conception du centrale/échéancier

2002-11-27 Présentation et démo du CVS. Présentation d'un prototype d'affichage du réseau. Discussion générale (Design Patterns).

2002-12-02 SimBouffe - Filet Wellington avec ses petits légumes (souper chez JB).

2002-12-04 Discussion avec ETR de la génération des clients et du graphe ; Discussion avec ETR de ce qu'est une politique ; Discussion générale.

2002-12-11 Décision de créer un gestionnaire de préférences.

2002-12-18 Réorganisation complète du système d'extraction de documentation. Utilisation d'HappyDoc.

2002-12-22 Fin des modifs des commentaires, génération de la doc technique. naissance de http ://SimTaxi.sf.net.

2002-12-28 Fin de la nouvelle documentation qualité ; Mise en place de la structure LaTeX qui permettra d'écrire la doc. Problème avec l'algo du chemin le plus court.. trop lent.

2003-01-08 Instruction pour l'écriture de la documentation.

2003-01-15 Intégration des modules, écriture du programme principal.

2003-01-22 La simulation tourne de A à Z.

2003-01-29 Démonstration de la simulation. Discussion des problèmes d'optimisation.

2003-02-05 Sérialisation de la demande. Amélioration de PCC (les arbres calculés sont stockés et sérialisés).

2003-02-12 Préparation de la présentation. Entrecôte de boeuf aux morilles (dîner cafète eivd).

2003-01-19 Présentation (Organisation interne, choix techniques, etc.).

2003-01-26 Relecture et mise au point de la documentation. Préparation de la démonstration.

2003-03-03 Rendu du projet avec sa documentation.

2003-03-05 Démonstration du logiciel.

Annexe B

Documentation qualité

B.1 Remarque préliminaire

Ce document a été écrit dans le but d'uniformiser l'écriture du code source de SimTaxi. Ceci est nécessaire pour les raisons suivantes :

- Chaque développeur a ses propres règles pour écrire du code. Pour cette raison il est parfois difficile de relire le code de quelqu'un d'autre. Utiliser les mêmes conventions assurera une compréhension plus facile du code.
- Les standards de qualité permettent d'obtenir du code plus robuste qui facilite la détection des bugs.
- Il est plus facile de reprendre un projet où des standards de qualité qui ont été définis.

B.2 Règles de qualités

Cette section décrit les différents standards de qualité à appliquer au projet SimTaxi.

B.2.1 Noms des fichiers

Les noms de fichiers Python (module) devront respecter le nom de la classe principale qu'ils contiennent afin d'éviter les problèmes de portabilité entre les différents systèmes d'exploitation. Le nom des autres fichiers sont en minuscule ainsi que les extensions.

Rappelons qu'en Python un fichier est un module (contenant des classes, des fonctions, ...) et qu'un dossier est un paquetage.

B.2.2 Tabulations

Les tabulations sont de longueur 4 (utiliser des espaces). La longueur d'une ligne ne doit pas excéder 78 caractères.

B.2.3 Expressions

Chaque opérateur mathématique et l'opérateur d'affectation doivent être précédés et suivis d'un espace. Une virgule (',') est toujours suivie d'un espace.

B.2.4 Noms

Les noms ne doivent pas comporter de souligné ou de caractères spéciaux. La séparation des mots dans un identificateur est indiqué grâce à une majuscule.

Un nom de classe commence toujours par une majuscule. Tous les autres identificateurs commencent par une minuscule.

Exemples :

```
1  # Définition d'une classe
2  class SimTaxiApp :
3
4      # Déclarations de fonctions et de variables
5      def maFonction(unEntier, ...):
6          maVariableEntiere = unEntier
```

B.2.5 En-têtes

Chaque fichier doit comporter l'en-tête suivante :

```
1  #!/usr/bin/env python
2  """
3  Description succincte du module (1 ligne maximum) terminée par un point.
4
5  Description plus complète de ce module :
6  -A quoi sert-il?
7  -Les classes qu'il contient
8  -Principe de fonctionnement
9  -Comment l'utiliser
10
11 $Id: st-qualite.tex,v 1.2 2003/03/02 13:59:53 erreur Exp $
12 """
13
14 __version__ = "$Revision: 1.2 $"
15 __author__ = "EI5a, eivd, SimTaxi (Groupe Burdy)"
16 __date__ = "2002-10-30"
```

Les identifiants mis entre $ seront automatiquement mis à jour par le CVS lors des commit. Il ne faut plus les toucher après la création du fichier.

Les classes devront comporter l'en-tête suivante :

```
"""
Description succincte de la classe (1 ligne maximum) terminée par un point.

Description plus complète de la classe :
-A quoi sert-elle?
-Principe de fonctionnement
-Comment l'utiliser
"""
```

Cette en-tête se place sous la ligne de déclaration de la classe.
Exemple :

```
class MaClasse
    """
    Une classe d'exemple.

    Cette classe ne sert que pour exemple. Elle n'apporte aucune fonctionnalité
    et n'est pas utilisable.
    """
```

Enfin les en-têtes de méthodes se présentent sous cette forme :

```
def methode(self, param):
    """
    Courte description sur une ligne finie par un point.

    Description plus complète si nécessaire.

    nom (type) -- Description du paramètre

    retourne (type) -- Description de l'objet retourné par la fonction
                       (si elle en retourne un)

    - depuis - n°version : Depuis quelle version cette méthode existe

    - autheur - Nom
    """
```

Exemple :

```
def methodeExemple(self, param1, param2):
    """
    Cette méthode n'a pas d'action.

    param1 (tuple(int, list)) -- Tuple (nombre de bagages, clients)

    param2 (Taxi) -- le taxi pour ...

    retourne (bool) -- si le taxi peut le faire...

    - depuis - 1.0

    - depuis - Vincent Decorges
    """
```

B.2.6 Description des types

Dans le cas où un type n'est pas connu, il n'est pas utile de le préciser.

Les types les plus courants en Python sont : `tuple, list, int, float, dict, str, ...` pour connaître le type d'un objet il existe toujours une fonction `type`.
Exemple :

```
>>> type("blabla")
<type 'str'>
```

Le type booléen n'existe pas. Il est représenté par des entiers. Mais nous le considèreront quand même comme un type. Rappelons que 0 = faux, tout le reste = vrai.

Tout comme l'exemple de la rubrique En-Têtes il est commode de préciser les types contenus dans un tuple.

B.2.7 Commentaires

Les commentaires peuvent contenir des caractères accentués.

Les commentaires entre """ décrivant nos En-Têtes, sont utilisés pour générer la documentation technique. La phase de génération comporte un formatage de texte. Ce qui explique pourquoi il faut mettre une ligne vide entre deux phrases pour passer à la ligne (ce qui peut paraître lourd dans le code). Les détails de ce formatage ne sont pas décrit dans ce document, mais vous pouvez déjà visionner (dans la doc technique) le résultat d'un -- en milieu de ligne ou d'un - en début de ligne.

Annexe C

Documentation de rédaction

C.1 Remarque préliminaire

Ce document a été écrit dans le but d'informer sur les techniques d'écriture de la documentation de SimTaxi et de permettre à ceux ne connaissant pas LaTeX d'écrire leur partie de documentation.

La documentation SimTaxi comportera les parties suivantes :

La documentation utilisateur Description des paramètres et manipulation pour l'utilisation de SimTaxi. Utilisation de l'interface utilisateur. Interprétation des résultats de simulation. Définition d'une politique.

La documentation développement Description des diverses parties, des choix de conceptions (ainsi que les schémas), des algorithmes, etc... Cette partie est la plus lourde et sera fractionnée en plusieurs fichiers (voir section C.3).

La documentation technique La documentation technique sera entièrement générée depuis les commentaires du code source.

Pour faciliter l'écriture de cette documentation nous allons utiliser LaTeX . En deux mots, une documentation LaTeX s'écrit dans un simple fichier texte (extension .tex), qui se «compile» pour générer un document au format ps, pdf, html, etc... Lors de l'écriture d'un document LaTeX on ne se préoccupe ni de la disposition du texte, ni des marges, ni de la césure des mots, ni du format de sortie. On ne fait qu'écrire le texte ainsi que quelques instructions pour le formatage final.

Le fait que les fichiers sources soient au format texte nous permet d'utiliser le CVS et donc de travailler à plusieurs sur la documentation. Comment travailler à 8 sur un «.doc» ?

C.1.1 Environnement requis

Pas besoin d'installer la structure LaTeX sur vos systèmes, elle fait partie des distributions Linux, mais est assez lourde à installer sur du Win32.

Des documents intermédiaires au format pdf seront disponibles régulièrement pour que l'on puisse se rendre compte de l'état du travail.

C.2 Syntaxe minimum

L'écriture d'un document LaTeX est donc très simple, il suffit d'écrire le texte accompagné de quelques instructions/commandes qui permettront au «compilateur» de gérer la structure du document.

La documentation de SimTaxi sera fractionnée en plusieurs fichiers sources .tex, ce qui va alléger un maximum les instructions des fichiers «feuilles» (la structure étant une sorte d'arbre). Et minimisera le nombre de personne travaillant sur un même fichier (idéalement nous aurons chacun nos fichiers).

C.2.1 Caractères spéciaux

Un commentaire en LaTeX est précédé du caractère %.

C.2.2 Paragraphes

Voici un (mauvais) exemple de source .tex

```
Les noms de fichiers Python (module) devront respecter le nom de la classe principale
qu'ils        contiennent afin           d'éviter les problèmes
de % ceci est un commentaire
portabilité entre les différents systèmes
 d'exploitation.

Le nom des autres fichiers sont en minuscules
ainsi que les extensions    .
\paragraph{}
Rappelons qu'en Python un fichier est un module (contenant des classes,
des fonctions, ...) et qu'un dossier est un paquetage.
```

Ce qui donnera :

Les noms de fichiers Python (module) devront respecter le nom de la classe principale qu'ils contiennent afin d'éviter les problèmes de portabilité entre les différents systèmes d'exploitation.
Le nom des autres fichiers sont en minuscules ainsi que les extensions.

Rappelons qu'en Python un fichier est un module (contenant des classes, des fonctions, ...) et qu'un dossier est un paquetage.

On peut voir qu'un retour à la ligne se fait avec une ligne vide séparant les deux lignes[1] et qu'un nouveau paragraphe s'introduit grâce à `\paragraph{}`.

C.2.3 Sections

Les sections sont très importantes, c'est elles qui permettent de structurer le document, de générer la table des matières, etc... Il y a trois niveaux d'imbrication : `\section{sonNom}`, `\subsection{sonNom}`, `\subsubsection{sonNom}`.

Exemple des sections de ce document :

```
\section{Syntaxe minimum}
L'écriture d'un document \LaTeX\ (.tex) est....

   \subsection{Listes}
   Voici un ...
```

Pas besoin de s'amuser avec l'indentation.

[1] comme pour HappyDoc

C.2.4 Listes

Il y a plusieurs façons de faire des listes, voici des exemples :

```
\begin{description}
\item[La documentation utilisateur] Description des paramètres...
\item[La documentation développement] Description des...
\item[La documentation technique] La document...
\end{description}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\begin{itemize}
\item blabla
\item Les standards...
\item Il est plus...
\end{itemize}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\begin{enumerate}
\item blabla
\item Les standards...
\item Il est plus...
\end{enumerate}
```

La documentation utilisateur Description des paramètres...

La documentation développement Description des...

La documentation technique La document...

- blabla
- Les standards...
- Il est plus...

1. blabla
2. Les standards...
3. Il est plus...

C.2.5 Styles

```
Voici \textit{italique}, \textbf{gras} et \underline{souligné}.
Ou alors le mode $math$ pour écrire des expressions $a_z+bx^2=0$.
```

Voici *italique*, **gras** et souligné. Ou alors le mode $math$ pour écrire des expressions $a_z + bx^2 = 0$.

C.2.6 Notes de bas de pages

Les notes de bas de pages sont assez agréables à condition de ne pas en abuser.

```
Après ce mot\footnote{ici la note de bas de page} une note de bas de page.
```

Après ce mot[2] une note de bas de page.

[2] ici la note de bas de page

FIG. C.1 – Description de l'image XY

C.2.7 Stopper le formatage

L'environnement «verbatim» permet de stopper le formatage de texte, c'est lui qui a été utilisé pour présenter les exemples de code dans ce document.

```
begin{verbatim}
   Il manque le \ avant le begin et le end...
end{verbatim}
```

C.2.8 Les étiquettes

Les étiquettes sont très pratiques dans les grands documents. Elles permettent de faire référence à une autre partie du même document sans connaître à l'avance la numérotation qu'elle aura, ni la page où elle se trouvera. Il suffit de placer un «label» à l'emplacement auquel nous voulons faire référence.

```
\label{xy} Comme nous pouvons le voir au point \ref{xy} de la page \pageref{xy}...
```

Comme nous pouvons le voir au point C.2.8 de la page 29...

C.2.9 Schémas, figures, images

Tous nos graphiques seront dans un bloc «figure» ce qui permet la numérotation du graphique. Les figures ne sont pas spécialement insérées à l'endroit de leur description (LaTeX s'occupe de gérer la répartition pour ne pas alourdir le document), il faut donc y faire référence lorsqu'on en parle.

```
\begin{figure}
% ici viendra le schéma UML de....
\caption{\label{shemaXY}Description de l'image XY}
\end{figure}
... comme nous pouvons le voir sur la figure \ref{shemaXY} à la page \pageref{shemaXY}
```

... comme nous pouvons le voir sur la figure C.1 à la page 29

C.2.10 Math

LaTeX est pratiquement fait pour écrire des maths, mais il y a trop à dire...

```
$$ \sum_{i=0}^{\infty} x_i = 1$$
```

$$\sum_{i=0}^{\infty} x_i = 1$$

C.3 Sectionnement de la documentation

Voici une 1ère proposition de fractionnement. Dans un 1er temps elle ne concerne que les modules. Il y aura aussi toute la partie documentation utilisateur, introduction, etc...

st-gui.tex description de l'affichage ainsi que de l'interface utilisateur.

st-graphe.tex description générale du graphe, de la structure de donnée.

st-graphe-generation.tex méthode pour la génération du graphe.

st-graphe-PCC.tex méthode du plus court chemin.

st-initialisateur.tex description des techniques d'initialisations des divers parties.

st-central.tex description du rôle du central, avec qui il communique.

st-central-evenements.tex description de la hiérarchie des événements.

st-central-echeancier.tex description de l'échéancier.

st-politiques.tex description des politiques, sous quelle formes elles sont représentées. Celle que nous avons implémentée.

st-gestionnaires.tex description du rôles des gestionnaires.

st-gest-stations.tex description du gestionnaire des stations mais aussi de ce qu'est une station.

st-gest-taxis.tex description du gestionnaire des taxis ainsi que du comportement d'un taxi, les étapes d'une course.

st-gest-preferences.tex description de la structure des préférences.

C.4 Conclusion

Il y a peu de chance qu'après lecture de ce document vous ayez les connaissances suffisantes pour écrire ce que vous voulez. N'hésitez pas à me demander[3] ce qu'il vous manque.

Sur le CVS vous pouvez trouver le source .tex de notre documentation qualité, qui est un bon exemple (ce document ne l'est pas vraiment).

[3]par email de préférence -> *jb at urbanet.ch*

Annexe D

Documentation Utilisateur

D.1 Introduction

Dans cette 1ère partie de développement nous nous sommes principalement attardés sur la simulation ainsi que l'affichage de celle-ci. De ce fait, il n'y a pas vraiment d'interface utilisateur.

D.2 Pré-requis

L'exécution de SimTaxi requiert les éléments suivant :

- Python $\geq$ 2.2.1
- wxPython $\geq$ 2.3.3
- PyOpenGL $\geq$ 2

Dans le cas où l'une ces librairies ne serait pas présente sur votre système, startSimTaxi vous l'indiquera ainsi que le lien pour l'obtenir.

Tous les tests ainsi que les validations du logiciel ont été fait sur une Mandrake(9) Linux(2.4.19). Certains développeurs étant sous win32, nous avons pu constater que le logiciel y fonctionnait aussi.

Nous n'avons pas fait de tests sur des versions antérieures des librairies.

D.3 Installation

SimTaxi ne requiert pas d'installation particulière. Il suffit de posséder le dossier des sources que l'ont peut obtenir à l'aide de la commande[1] :

```
cvs -z3 -d:pserver:anonymous@cvs.sf.net:/cvsroot/simtaxi co dev
```

ou depuis le site $http://SimTaxi.sf.net$

[1] requière un client CVS

D.4 Configuration

Actuellement, SimTaxi se configure à l'aide d'un fichier texte ($config.txt$). On y trouve les options suivantes (les options commentées ne sont pas encore prises en compte) :

```
# Fichier de préférences de SimTaxi (http://SimTaxi.sf.net)

#################
# Interface utilisateur
gui             : True # utilisation ou non de l'affichage
# ATTENTION dans le cas de la non utilisation du gui, la simulation
# est difficile à stopper

# uniquement si gui
pseudoContinu   : False # crée un intervalles de temps artificiel entre les év.

#uniquement si pseudoContinu
dureeSec        : 0.05  # la durée d'un sec dans la simulation


#################
# Initialisation
nbTaxi          : 180
vitesseTaxiKMH  : 50

ndStation       : 30
tailleStation   : [20, 10, 10, 5] # 1x20 2x10 et les autres 5


##########
# Demandes
nbCoursesJour   : 2700 # nb de clients pour 1 "journée"
hPremiereCourse : 0    # date du 1er client
hDerniereCourse : 3600*24 # durée en seconde d'une "journée" (d'une simulation)
moyenneCourseKM : 0 # 8, 0 = ne pas en tenir compte
#germeDemandes : 1


########
# Graphe

# sep permet de séparer les dossiers
# curdir est le dossier courant
fichierGraphe   : curdir+sep+'graphe'+sep+'graphe.gr'
#nbCarrefour     : 80
#nbRue           : 140
#diametreKM      : 12 # 20 min en temps
#germeGraphe     : 1
```

D.5 Utilisation

Le lancement de SimTaxi se fait à l'aide de $startSimTaxi.pyw$:

```
./startSimTaxi.pyw
```

L'extension pyw indique sur certains systèmes, que nous avons à faire un à GUI.

Un démarrage correct doit afficher les lignes suivantes :

```
# Id: SimTaxi.py,v 1.25 2003/03/02 19:53:36 erreur Exp #

Chargement des chemins déjà calculés...
800 arbres existants.
Initialisation stations...
Initialisation taxis...
Recup de la demande...
2700 clients
==== Début de la simulation
0.0 %
0.0150462962963 %
0.0162037037037 %
```

SimTaxi commence par charger[2] tous les arbres qui sont déjà calculés (fichier $dumpChemins$). Ensuite la mise en place des stations et des taxis est faite. Le tout est suivi du chargement de la demande (fichier texte $dumpClients$). Puis la simulation commence avec le % du déroulement total de celle-ci.

D.5.1 les dumps

Pour améliorer la performance de la simulation, les chemins ainsi que les clients sont sauvés dans des fichiers pour ne plus être calculés lors de l'exécution. Et ce, pour plusieurs raisons :

dump des chemins Si nous ne stockions pas les arbres retournés par l'algorithme de Dijkstra, ils seraient calculés un nombre important de fois. Leur stockage sous forme de fichier est raisonnable (10MB pour 800 arbres).

dump des clients Le respect de la moyenne de longueur des chemins lors de la génération de la demande étant un point lourd ce qui nous pose certains problèmes. Nous arrivons à la respecter mais de façon trop lourde pour pouvoir le faire en cours de simulation. Nous l'avons donc séparée de la simulation. La demande étant très légère à stocker (fichier texte contenant les dates, lieu de départ et lieu d'arrivée des clients).

D.6 Interprétation de l'affichage

Les carrés rouges sont les carrefours, reliés par les morceaux de routes. Les clients sont des carrés bleus clignotants. Les taxis sont les triangles jaunes accompagnés de leur numéro et d'un symbole. Quand ce symbole est une flèche bleue ils vont chercher un client, quand il est un rond rouge ils transportent un client et quand il est une flèche orange ils rentrent en station. Quand un taxi pose un client, celui-ci est toujours bleu mais disparait progressivement. Les stations sont les polochons bruns sur le bord des routes. Elle sont accompagnées de leur taux de remplissage.

[2] Pour des raisons inconnues le chargement est très long (2min pour un 1,6GHz). Et tant que la simulation n'a pas tourné assez longtemps pour pouvoir calculer tous les arbres, l'arrêt de celle-ci est tout aussi long (enregistrement des nouveaux arbres).

Annexe E

Listing des sources

Pour ne pas alourdir le document, uniquement une sélection des fichiers sources jugé important ont été imprimés.

E.1 Central

E.1.1 Central.py

```
#!/usr/bin/env python
"""
Module contenant la classe du central.

$Id: Central.py,v 1.17 2003/01/26 10:30:53 vega01 Exp $
"""
__version__ = '$Revision: 1.17 $'
__author__ = 'EI5A, eivd, SimTaxi (Groupe Burdy)'
__date__ = '2002-12-01'

from Echeancier import *
from GestionnaireTaxis import *
from GestionnaireStations import *
from Evenement import *
from Singleton import *
import PolitiquePlusPres


# pour les tests

def foncComp(evenement1, evenement2):
    """
    Fonction de comparaison des evenements pour l'echeancier.

    evenement1, evenement2 (Evenement) -- les evenements a comparer

    retourne (Integer) --  0 si les evenements ont lieu en meme temps egaux
                           1 si le premier evenement a lieu apres le deuxieme
                           -1 sinon

    - depuis - 1.0

    - auteur - Alexandre D'Amico
    """

    return evenement1.temps() < evenement2.temps()



class Central(Singleton):
    """
    Implemente le central.

    Cette classe fournit un central qui traite les evenements de l'echeancier.
    """


    def init(self, politique = PolitiquePlusPres.PolitiquePlusPres()):
        """
        Constructeur.

        Permet de créer un objet de la classe Central.

        - depuis - 1.0

        - auteur - Alexandre D'Amico
        """

        # INITIALISATIONS

        # initialisation de l'echeancier avec sa fonction de comparaison
        self._echeancier = Echeancier(foncComp)

        # creation des gestionnaires
        self._gestionnaireTaxis = GestionnaireTaxis()
        self._gestionnaireStations = GestionnaireStations()

        #creation de la politique
        self._politique = politique
```

```
70
71      def initEv(self, listeEv):
72          """
73          Initialise l'echeancier avec des evenements.
74
75          listeEv (Liste) -- les evenements pour initialiser l'echeancier
76
77          - depuis - 1.0
78
79          - auteur - Alexandre D'Amico
80          """
81          self._echeancier.initEcheancier(listeEv)
82
83
84      def ajouterEvenement(self, evenement):
85          """
86          Ajoute un evenement dans l'echeancier du Central.
87
88          evenement (Evenement) -- l'evenement a inserer dans l'echeancier
89
90          - depuis - 1.0
91
92          - auteur - Alexandre D'Amico
93          """
94          self._echeancier.deposer(evenement)
95
96
97      def traiterProchainEvenement(self):
98          """
99          Supprimer le premier element de l'echeancier et le traiter en
100         fonction de son type. Puis le retourne.
101
102         retourne (Evenement) -- L'événement qui a été traité.
103
104         - depuis - 1.0
105
106         - auteur - Alexandre D'Amico
107         """
108         # utiliser la methode traiter propre a chaque evenement
109         ev = self._echeancier.prelever()
110         ev.traiter()
111         return ev
112
113
114     def modifierPolitique(self, politique):
115         """
116         Modifie la politique de traitement des evenements.
117
118         Modifie la politique pour le choix du taxi en fonction de la
119         position d'un client et le choix d'une station en fonction de la
120         position du taxi.
121
122         politique (Politique) -- la nouvelle politique de traitement des
123                                  evenements
124
125         - depuis - 1.1
126
127         - auteur - Alexandre D'Amico
128         """
129         self._politique = politique
130
131
132     def politique(self):
133         """
134         Retourne la politique actuelle de traitement des evenements.
135
136         retourne (Politique) -- la politique actuelle
137
138         - depuis - 1.5
139
140         - auteur - Alexandre D'Amico
141         """
142         return self._politique
143
```

```
144
145     def supprimerEvArriverStation(self, taxi):
146         """
147         Supprime l'evenement de l'arrivee d'un taxi en station.
148
149         taxi (Taxi) : le taxi qui devait arriver en station
150
151         - depuis - 1.5
152
153         - auteur - Alexandre D'Amico
154         """
155         for i in self._echeancier:
156         # recherche de l'evenement a supprimer
157             if (i.im_class == EvArriverStation and i.taxi() == taxi):
158             # c'est un evenemet d'arrivee en station et il concerne ce taxi
159                 self._echeancier.remove(i)  # supression de l'evenement
160                 # annulation de la reservation du taxi
161                 i.station().annulerReservation()
162                 break
163
164
165     def evenement(self):
166         """
167         Permet de savoir s'il y a encore au moins un evenement.
168
169         retourne (Bool) -- Vrai s'il y a au moins un element
170
171         - depuis - 1.1
172
173         - auteur - Alexandre D'Amico
174         """
175         return not self._echeancier.vide()
176
177
178     def intervalleProchainEvement(self, mnt=0):
179         """
180         Retourne le temps auquel aura lieu le prochain evenement.
181
182         mnt (Temps) -- Le temps actuel.
183
184         retourne (Temps) -- L'intervalle de temps entre mnt et le prochain evenement.
185
186         - depuis - 1.16
187
188         - auteur - Julien Burdy
189         """
190         return self._echeancier.tempsProchainEv() - mnt
```

E.1.2 Evenement.py

Les classes des événements.

```
1   #!/usr/bin/env python
2   """
3   Module contenant les classes événements.
4
5   $Id: Evenement.py,v 1.19 2003/02/09 12:17:38 vega01 Exp $
6   """
7   __version__ = '$Revision: 1.19 $'
8   __author__ = 'El5a, eivd, SimTaxi (Groupe Burdy)'
9   __date__ = '2002-12-01'
10
11  from Taxi import *
12  import GrapheXY
13  import Central
14
15  class ErreurTraiter(Exception):
16      """
17      Exception lors du traitement d'un evenement.
18      """
19      pass
20
```

```
21
22
23  class Evenement:
24      """
25      Classe racine de tout les evenements pour SimTaxi.
26      """
27
28
29      def __init__(self, temps, traiter = None):
30          """
31          Constructeur
32
33          temps (Int) -- Le moment ou l'evenement aura lieu
34
35          traiter (Function) -- Le code a executer lorsque l'evenement a lieu
36
37          retourne (Evenement) -- Un objet evenement
38
39          - depuis - 1.0
40
41          - auteur - Vincent Decorges
42          """
43          self._temps = temps
44          self._traiter = traiter
45
46
47      def traiter(self):
48          """
49          Traitement associe à l'evenement.
50
51          Par défaut aucun traitement associe. Peut déclencher l'exception
52          Erreur_Traiter.
53
54          - depuis - 1.0
55
56          - auteur - Vincent Decorges
57          """
58          self._traiter()
59
60
61      def temps(self):
62          """
63          Donne le temps auquel l'evenement doit avoir lieu.
64
65          - depuis - 1.0
66
67          - auteur - Vincent Decorges
68          """
69          return self._temps
70
71      def __repr__(self):
72          """
73          Donne des infos sur l'événement
74
75          - depuis - 1.0
76
77          - auteur - Vincent Decorges
78          """
79          return self.__class__.__name__ + "\ntemps : " + str(self._temps)
80
81
82
83  class EvClient(Evenement):
84      """
85      Evenement client.
86
87      Permet de savoir qu'un client a appelle un taxi est à quel moment.
88      """
89
90
91      def __init__(self, temps, positions):
92          """
93          Constructeur
94
```

```
95          temps (Int) -- Le moment ou l'evenement aura lieu
96
97          positions (Tuple) -- Tuple contenant la position + la destination
98
99          retourne (EvClient) -- Un objet EvClient
100
101         - depuis - 1.0
102
103         - auteur - Vincent Decorges
104         """
105         Evenement.__init__(self, temps, self.__traitementClient)
106         self.__positions = positions
107         self.__chemin = None
108         self.__cheminClient = None
109
110
111     def chemin(self):
112         """
113         Selecteur.
114
115         retourne (Chemin) -- Chemin pour aller de la position intial a
116                                 la destination.
117
118         - depuis - 1.1
119
120         - auteur - Vincent Decorges
121         """
122         return self.__chemin
123
124
125     def cheminClient(self):
126         """
127         Selecteur.
128
129         retourne (Chemin) -- Chemin que le taxi doit faire pour aller
130                                 chercher le client.
131
132         - depuis - 1.3
133
134         - auteur - Vincent Decorges
135         """
136         return self.__cheminClient
137
138     def positions(self):
139         """
140         Selecteur.
141
142         retourne (Tuple arcs) -- Position du client et destination
143
144
145         - depuis - 1.3
146
147         - auteur - Vincent Decorges
148         """
149         return self.__positions
150
151
152
153     def __traitementClient(self):
154         """
155         Appeler par traiter.
156
157         Choisi un taxi d'après la politique.
158
159         - depuis - 1.0
160
161         - auteur - Vincent Decorges
162         """
163
164         graphe = GrapheXY.GrapheXY()
165         #Calcul du chemin à parcourir utiliser par le taxi
166         self.__chemin = graphe.cheminPlusCourt(self.__positions[0], self.__positions[1])
167
168         #On recupere la politique courante
```

```
169         politique = Central.Central().politique()
170         taxi, self.__cheminClient = politique.choisirTaxi(self)
171 
172         if taxi == None:
173             raise ErreurTraiter
174 
175         taxi.traiterEvenement(self)
176 
177     def __repr__(self):
178         """
179         Donne des infos sur l'événement
180 
181         - depuis - 1.0
182 
183         - auteur - Vincent Decorges
184         """
185         return Evenement.__repr__(self) + "\nchemin : " + str(self.__chemin) \
186         + "\nchemin client : " + str(self.__cheminClient)
187 
188 class EvTaxi(Evenement):
189     """
190     Evenement Taxi.
191 
192     Classe racine des evenements que peut lancer un taxi.
193     """
194 
195 
196     def __init__(self, temps, taxi, traiter = None):
197         """
198         Constructeur
199 
200         temps (Int) -- Le moment ou l'evenement aura lieu
201 
202         taxi (Taxi) -- Le taxi qui cree l'evenement
203 
204         traitement (Function) -- Traitement a effectue
205 
206         retourne (EvTaxi) -- Un objet EvTaxi
207 
208         - depuis - 1.0
209 
210         - auteur - Vincent Decorges
211         """
212         Evenement.__init__(self, temps, traiter)
213         self._taxi = taxi
214 
215 
216     def taxi(self):
217         """
218         Retourne le taxi qui a cree l'evenement
219 
220         retourne (Taxi) -- Un objet Taxi
221 
222         - depuis - 1.0
223 
224         - auteur - Vincent Decorges
225         """
226         return self._taxi
227 
228     def __repr__(self):
229         """
230         Donne des infos sur l'événement
231 
232         - depuis - 1.0
233 
234         - auteur - Vincent Decorges
235         """
236         return Evenement.__repr__(self) + "\ntaxi : " + str(self._taxi.getNo())
237 
238 
239 class EvChargerClient(EvTaxi):
240     """
241     Lancer par le taxi quand il charge un client.
242     """
```

```
243
244
245     def __init__(self, temps, taxi, client):
246         """
247         Constructeur.
248
249         temps (Int) -- Le moment ou l'evenement aura lieu
250
251         taxi (Taxi) -- Le taxi qui cree l'evenement
252
253         client (EvClient) -- Le client que le taxi charge
254
255         retourne (EvChargerClient) -- Un objet EvChargerClient
256
257         - depuis - 1.0
258
259         - auteur - Vincent Decorges
260         """
261         EvTaxi.__init__(self, temps, taxi, self.__traitementCharger)
262         self._client = client
263
264
265     def client(self):
266         """
267         Retourne l'evenement client.
268
269         retourne (EvClient) -- Un objet EvClient
270
271         - depuis - 1.0
272
273         - auteur - Vincent Decorges
274         """
275         return self._client
276
277
278     def __traitementCharger(self):
279         """
280         Appeler par traiter.
281
282         S'envoye au taxi.
283
284         - depuis - 1.3
285
286         - auteur - Vincent Decorges
287         """
288         self._taxi.traiterEvenement(self)
289
290     def __repr__(self):
291         """
292         Donne des infos sur l'événement
293
294         - depuis - 1.0
295
296         - auteur - Vincent Decorges
297         """
298         return EvTaxi.__repr__(self) + "\nclient : " + self._client.__repr__()
299
300 class EvPoserClient(EvChargerClient):
301     """
302     Lancer par le taxi quand il pose un client.
303     """
304
305
306     def __init__(self, temps, taxi, client):
307         """
308         Constructeur.
309
310         temps (Int) -- Le moment ou l'evenement aura lieu
311
312         taxi (Taxi) -- Le taxi qui cree l'evenement
313
314         retourne (EvTaxi) -- Un objet EvTaxi
315
316         - depuis - 1.3
```

```
317
318             - auteur - Vincent Decorges
319             """
320             EvTaxi.__init__(self, temps, taxi, self.__traitementStation)
321             self._client = client
322             self.__cheminStation = None
323
324
325         def cheminStation(self):
326             """
327             Selecteur.
328
329             retourne (Chemin) -- Chemin que le taxi doit faire pour aller
330                             chercher le client.
331             - depuis - 1.3
332
333             - auteur - Vincent Decorges
334             """
335             return self.__cheminStation
336
337
338         def station(self):
339             """
340             la station ou est le taxi.
341
342             retourne (Station) -- Un objet Station
343
344             - depuis - 1.6
345
346             - auteur - Vincent Decorges
347             """
348             return self._station
349
350
351         def __traitementStation(self):
352             """
353             Appeler par traiter.
354
355             D'apres la politique redonne une station.
356
357             - depuis - 1.4
358
359             - auteur - Vincent Decorges
360             """
361             #On recupere la central pour avoir la politique
362             politique = Central.Central().politique()
363
364
365             self._station, self.__cheminStation = politique.choisirStation(self)
366
367             if self._station == None:
368                 raise ErreurTraiter
369
370             self._taxi.traiterEvenement(self)
371
372         def __repr__(self):
373             """
374             Donne des infos sur l'événement
375
376             - depuis - 1.0
377
378             - auteur - Vincent Decorges
379             """
380             return EvChargerClient.__repr__(self) + "\nclient : " + self._client.__repr__() \
381             + "\nchemin station : " + str(self.__cheminStation) + "\nstation : " + str(self.
                    _station.getNo())
382
383 class EvArriverStation(EvTaxi):
384     """
385     A lieu quand un taxi arrive à une station.
386     """
387
388
389     def __init__(self, temps, taxi, station):
```

```
390         """
391         Constructeur.
392 
393         temps (Int) -- Le moment ou l'evenement aura lieu
394 
395         taxi (Taxi) -- Le taxi qui cree l'evenement
396 
397         station (Station) -- La station ou le taxi arrive
398 
399         retourne (EvArriverStation) -- EvArriverStation
400 
401         - depuis - 1.0
402 
403         - auteur - Vincent Decorges
404         """
405         EvTaxi.__init__(self, temps, taxi, self.__traitementStation)
406         self._station = station
407 
408 
409     def __traitementStation(self):
410         """
411         Appeler par traiter.
412 
413         Doit transmettre des infos à la station.
414 
415         - depuis - 1.0
416 
417         - auteur - Vincent Decorges
418         """
419         self._taxi.traiterEvenement(self)
420 
421     def __repr__(self):
422         """
423         Donne des infos sur l'événement
424 
425         - depuis - 1.0
426 
427         - auteur - Vincent Decorges
428         """
429         return EvTaxi.__repr__(self) + "\nstation : " + str(self._station.getNo())
```

E.1.3 Politique.py

La classe politique mère (classe abstraite)

```
#!/usr/bin/env python
"""
Module contenant la classe abstraite des politiques.

$Id: Politique.py,v 1.3 2003/01/11 11:32:00 vega01 Exp $
"""
__version__ = '$Revision: 1.3 $'
__author__ = 'EI5A, eivd, SimTaxi (Groupe Burdy)'
__date__ = '2002-12-22'



class ErreurAbstraite(Exception):
    """
    Exception pour les classes abstraites.
    """
    pass



class Politique:

    """
    Classe abstraite pour l'implémentation des politiques.
    """


    def choisirTaxi(self, client):
        """
        Retourne un taxi pour prendre en charge un client
        d'après la politique courante.

        client (EvClient) -- Le client qui veut faire la course

        retourne (Tuple(Taxi, Chemin)) -- Le taxi qui va prendre en charge la course

        - depuis - 1.0

        - auteur - Vincent Decorges
        """
        raise ErreurAbstraite


    def choisirStation(self, taxi):
        """
        Retourne une station d'après la politique courante.

        taxi (EvTaxi) -- Le taxi qui va à une station

        retourne (Station, Chemin) -- La station

        - depuis - 1.0

        - auteur - Vincent Decorges
        """
        raise ErreurAbstraite
```

PolitiquePlusPres.py

Notre unique politique actuele.

```
#!/usr/bin/env python
"""
Module contenant la politique du plus près.

$Id: PolitiquePlusPres.py,v 1.5 2003/01/26 10:37:55 vega01 Exp $
"""
__version__ = '$Revision: 1.5 $'
__author__ = 'EI5a, eivd, SimTaxi (Groupe Burdy)'
__date__ = '2002-12-22'



from Politique import Politique
import GestionnaireTaxis
import GestionnaireStations

class PolitiquePlusPres (Politique):
    """
    Implémante la politique du taxi le plus près et de
    la station la plus proche.
    """


    def choisirTaxi(self, client):
        """
        Retourne un taxi pour prendre en charge un client
        d'après la politique courante.

        client (EvClient) -- Le client qui veut faire la course

        retourne (Tuple(Taxi, Chemin)) -- Le taxi qui va prendre en charge la course

        - depuis - 1.0

        - auteur - Vincent Decorges
        """
        return GestionnaireTaxis.GestionnaireTaxis().plusProcheDe(client)


    def choisirStation(self, taxi):
        """
        Retourne une station d'après la politique courante.

        taxi (EvTaxi) -- Le taxi qui va à une station

        retourne (Tuple(Station, Chemin)) -- La station et

        - depuis - 1.0

        - auteur - Vincent Decorges
        """
        return GestionnaireStations.GestionnaireStations().plusProcheDe(taxi)
```

E.2 Gestionnaires

E.2.1 GestionnaireTaxis.py

C'est lui qui contiendra les méthodes disponibles pour écrire une politique.

```
1  #!/usr/bin/env python
2  """
3  Module du gestionnaire des taxis.
4
5  $Id: GestionnaireTaxis.py,v 1.28 2003/02/13 19:23:48 lulutchab Exp $
6  """
7  __version__ = '$Revision: 1.28 $'
8  __author__ = 'EI5a, eivd, SimTaxi (Groupe Burdy)'
9  __date__ = '2002-11-20'
10
11 from Singleton import Singleton
12 import Taxi
13 from Gestionnaire import Gestionnaire
14 from GestionnaireStations import GestionnaireStations
15 from GrapheXY import GrapheXY
16
17
18
19 class ErreurAucunTaxi(Exception):
20     """
21     Exception levée quand on cherche le taxi le plus proche et
22     qu'il n'y a aucun taxi dans la liste.
23
24     """
25     pass
26
27
28 class ErreurAucunTaxiLibre(Exception):
29     """
30     Exception levée quand on demande quel est le taxi le plus
31     proche et qu'il n'en reste plus aucun de libre
32     """
33     pass
34
35
36 class GestionnaireTaxis(Singleton):
37     """
38     Implemente un gestionnaire de Taxis.
39
40     Cette classe fournit un gestionnaire de taxis.
41     """
42
43
44     def init(self):
45         """
46         Constructeur.
47
48         Permet de creer un objet de la classe.
49
50         - depuis - 1.0
51
52         - auteur - Lucien Chaboudez
53         """
54         # Creation du gestionnaire
55         self.__gestionnaire = Gestionnaire()
56
57
58
59     def addTaxi(self, noStation):
60         """
61         Ajouter un Taxi.
62
63         Permet d'ajouter un taxi au gestionnaire.
64
65         noStation (int) -- Le no de la station dans laquelle le taxi se trouve.
66
67         - depuis - 1.0
```

```
68
69          - auteur - Lucien Chaboudez
70          """
71          lesStations = GestionnaireStations()
72
73          # Recherche du no du nouveau taxi
74          no = self.getNbTaxis() + 1
75
76          # Creation du nouveau taxi
77          newTaxi = Taxi.Taxi(no, noStation)
78
79          # Ajout du taxi dans le gestionnaire
80          self.__gestionnaire.addElement(no,newTaxi)
81
82          # Ajout du taxi dans la station
83          lesStations.affecterTaxi(noStation,no)
84
85
86      def getNbTaxis(self):
87          """
88          Nombre de taxis qui sont dans le gestionnaire.
89
90          Renvoie le nombre de taxis du gestionnaire.
91
92          retourne (int) -- Le nombre de taxis du gestionnaire.
93
94          - depuis - 1.0
95
96          - auteur -
97          """
98          # Renvoie le nombre d'elements
99          return self.__gestionnaire.nbElements
100
101
102
103
104
105     def plusProcheDe(self, client):
106         """
107         (TODO : add description)
108
109          client (EvClient) -- un evenement client.
110
111          retourne tuple(Taxi, Chemin) -- retourne le taxi le plus
112          proche ainsi que le chemin pour aller jusqu'au client.
113
114         - depuis - 1.0
115
116         - auteur -
117         """
118         #Reference sur le graphe.
119         graphe = GrapheXY()
120
121         #Récupération des stations
122         listeStations = GestionnaireStations().getListeStations()
123
124         #recherche de la position du client en prenant les 2 premiers sommets
125         #du chemin qu'il faudra prendre pour le conduire a destination
126         posClient = client.chemin().posDepart()
127
128         indexStation = -1
129
130         #Recherche d'une station contenant un taxi
131         for stationCour in listeStations :
132
133             # si il y a un taxi de libre dans la station,
134             if stationCour.getNbTaxis() > 0 :
135
136                 #arc de la 1ere station
137                 posStation = stationCour.arc()
138                 #récupération du no du taxi
139                 taxiPlusProche = stationCour.getTaxiSuivant()
140                 #recherche du chemin le plus court jusqu'au taxi
141                 cheminLePlusCourt =  graphe.cheminPlusCourt(posStation,posClient)
```

```
142                 #calcul de la taille du chemin
143                 distancePlusCourte = cheminLePlusCourt.distTotalPos()
144 
145                 #recherche de la position de la station dans la liste
146                 indexStation = listeStations.index(stationCour)
147                 break
148 
149         #Si on ne trouve pas de taxi,
150         if indexStation == -1:
151             raise ErreurAucunTaxiLibre
152 
153         #suppression des stations qu'on a déjà visité,
154         listeStations = listeStations[indexStation:]
155 
156 
157         #Recherche d'une station contenant un taxi
158         for stationCour in listeStations:
159 
160             # si il y a un taxi de libre dans la station,
161             if stationCour.getNbTaxis() > 0:
162 
163                 #arc de la 1ere station
164                 posStation = stationCour.arc()
165 
166                 #recherche du chemin le plus court jusqu'au taxi
167                 cheminCourant =  graphe.cheminPlusCourt(posStation,posClient)
168                 distanceCourante = cheminCourant.distTotalPos()
169 
170                 # Si on trouve un chemin plus court,
171                 if distanceCourante < distancePlusCourte:
172 
173                     #récupération du no du taxi
174                     taxiPlusProche = stationCour.getTaxiSuivant()
175                     #mise à jour du chemin
176                     cheminLePlusCourt = cheminCourant
177                     #mise à jour de la distance
178                     distancePlusCourte = distanceCourante
179 
180 
181         #Recuperation du taxi en fonction de son numero
182         taxiPlusProche = self.getTaxi(taxiPlusProche)
183 
184         return (taxiPlusProche,cheminLePlusCourt)
185 
186 
187     def delContenu(self):
188         """
189         Efface les taxis.
190 
191         Vide le gestionnaire contenant les taxis.
192 
193         - depuis - 1.3
194 
195         - auteur - Lucien Chaboudez
196         """
197 
198         self.__gestionnaire.delContenu()
199 
200 
201     def getListe(self):
202         """
203         Renvoie une liste des taxis.
204 
205         Permet de mettre les taxis dans une liste et de la renvoyer.
206 
207         retourne (List) -- Une liste de taxis
208 
209         - depuis - 1.9
210 
211         - auteur - Lucien Chaboudez
212         """
213 
214         # retour de la liste
215         return self.__gestionnaire.values()
```

```
216
217
218     def getTaxi(self, noTaxi):
219         """
220         Renvoie le taxi correspondant au no.
221
222         Permet de renvoyer le taxi qui correspond au numero passé.
223
224         noTaxi int -- le no du taxi
225         retourne Taxi -- Le taxi
226
227         - depuis - 1.24
228
229         - auteur - Lucien Chaboudez
230         """
231         return self.__gestionnaire[str(noTaxi)]
```

E.2.2 GestionnaireStations.py

```
1   #!/usr/bin/env python
2   """
3   Module du gestionnaire des stations.
4
5   $Id: GestionnaireStations.py,v 1.19 2003/01/29 16:15:17 lulutchab Exp $
6   """
7   __version__ = '$Revision: 1.19 $'
8   __author__ = 'EI5a, eivd, SimTaxi (Groupe Burdy)'
9   __date__ = '2002-11-20'
10
11  from Singleton import Singleton
12  from Station import Station
13  from Gestionnaire import Gestionnaire
14  from GrapheXY import GrapheXY
15  from Chemin import Chemin
16
17
18  class ErreurEvenementIncorrect(Exception):
19      """
20      Exception quand on passe un évenement incorrect
21      à la méthode plusProcheDe()
22      """
23      pass
24
25
26  class ErreurAucuneStation(Exception):
27      """
28      Exception quand on cherche la station la + proche
29      et qu'aucune station n'existe
30      """
31      pass
32
33
34  class ErreurAucuneStationAvecPlaceLibre(Exception):
35      """
36      Exception quand on demande la station la + proche et qu'il n'en
37      reste aucune avec une place de libre
38      """
39      pass
40
41
42  class GestionnaireStations(Singleton):
43      """
44      Implemente un gestionnaire de stations.
45
46      Cette classe fournit un gestionnaire de stations.
47      """
48
49
50      def init(self):
51          """
52          Constructeur.
```

```
53
54          Permet de creer un objet de la classe.
55
56          - depuis - 1.0
57
58          - auteur -  Lucien Chaboudez
59          """
60          # Creation du gestionnaire
61          self.__gestionnaire = Gestionnaire()
62
63
64
65      def addStation(self, nbPlaces, sommet1, sommet2):
66          """
67          Ajouter une station.
68
69          Permet d'ajouter une station dans le gestionnaire.
70
71          nbPlaces (int) -- le nombre de places de la station
72
73          sommet1,sommet2(tuple(nomSommet, Point)) -- les sommets entre lesquels
74                                      se trouve la station
75
76          retourne (int) -- le no de la station ajoutee
77
78          - depuis - 1.0
79
80          - auteur - Lucien Chaboudez
81          """
82          # Recherche du no de la nouvelle station
83          no = self.getNbStations() + 1
84
85          # Creation de la nouvelle station
86          newStation = Station(nbPlaces, no , sommet1, sommet2)
87
88          # Ajout de la station dans le gestionnaire
89          return self.__gestionnaire.addElement(no,newStation)
90
91
92
93      def getNbStations(self):
94          """
95          Nombre de stations qui sont dans le gestionnaire.
96
97          Renvoie le nombre de stations du gestionnaire.
98
99          retourne (int) -- Le nombre de stations du gestionnaire.
100
101         - depuis - 1.0
102
103         - auteur - Lucien Chaboudez
104         """
105         # Renvoie le nombre d'elements
106         return self.__gestionnaire.nbElements
107
108
109
110     def plusProcheDe(self, evPoserClient):
111         """
112         Renvois la station la plus proche de la position.
113
114         position (EvPoserClient) -- un évenement poser client
115
116         retourne (tuple(Station,Chemin)) -- la station la plus proche et
117         le chemin pour s'y rendre.
118
119         - depuis - 1.0
120
121         - auteur - Lucien Chaboudez
122         """
123
124         #Si ce n'est pas le bon évenement,
125         if evPoserClient.__class__.__name__ != 'EvPoserClient' :
126             raise ErreurEvenementIncorrect
```

```
127
128         # Creation d'une liste de taxis.
129         listeStations = self.__gestionnaire.values()
130
131         # si il n'y pas de stations,
132         if len(listeStations) == 0 :
133
134             raise ErreurAucuneStation
135
136
137         #Reference sur le graphe
138         graphe = GrapheXY()
139
140         #recherche de la position du taxi
141         position = evPoserClient.taxi().arc()
142         position = position[0]
143
144         posStation = -1
145
146         for stationPlusProche in listeStations :
147
148             # si il reste des places dans la station,
149             if stationPlusProche.getNbPlacesLibres() > 0 :
150                 # enregistrement de la position de la station dans la liste
151                 posStation = listeStations.index(stationPlusProche)
152                 break
153
154         # Si il n'y a plus de place dans aucune des stations,
155         if posStation == -1 :
156             # on propage une exception
157             raise ErreurAucuneStationAvecPlaceLibre
158
159         # recherche de l'arc sur lequel la station se trouve
160         arc = stationPlusProche.arc()
161
162         #Recherche du chemin le plus court jusqu'a la 1ere station
163         cheminLePlusCourt =  graphe.cheminPlusCourt(position, arc)
164
165         # Recherche de la taille du chemin
166         distancePlusCourte = cheminLePlusCourt.distTotalPos()
167
168         # Suppression de la station
169         listeStations = listeStations[posStation:]
170
171         # parcour des autres taxis
172         for stationCourante in listeStations :
173
174             # si il reste des places de libre dans la station,
175             if stationCourante.getNbPlacesLibres() > 0 :
176
177                 # recherche de l'arc sur lequel la station se trouve
178                 arc = stationCourante.arc()
179
180                 #recherche du chemin le + court jusqu'a la station courante.
181                 cheminCourant =  graphe.cheminPlusCourt(position, arc)
182                 distanceCourante = cheminCourant.distTotalPos()
183
184                 # Si le chemin courant est plus court,
185                 if distanceCourante < distancePlusCourte :
186
187                     #mise a jour des infos
188                     distancePlusCourte = distanceCourante
189                     cheminLePlusCourt = cheminCourant
190                     stationPlusProche = stationCourante
191
192
193         #retour de l'optimal
194         return (stationPlusProche,cheminLePlusCourt)
195
196
197     def getListeStations(self):
198         """
199         Renvoie une liste des stations.
200
```

```
201         Permet de mettre les stations dans une liste et de la renvoyer.
202
203         retourne (List) -- Une liste de stations
204
205         - depuis - 1.2
206
207         - auteur - Lucien Chaboudez
208         """
209         # retour de la liste
210         return self.__gestionnaire.values()
211
212
213
214     def affecterTaxi(self, noStation, noTaxi):
215         """
216         Affecte un taxi a la station dont le no est passe.
217
218         Permet d'affecter un taxi a la station dont le no est passe. Sera
219         appelee a l'initialisation du programme.
220
221         noStation (int) -- le no de la station a laquelle le taxi est affecte.
222
223         noTaxi (int) -- Le no du taxi a ajouter
224
225         - depuis - 1.2
226
227         - auteur - Lucien Chaboudez
228         """
229         # Ajout du taxi a la station
230         self.__gestionnaire[str(noStation)].affecterTaxi(noTaxi)
231
232
233
234     def delContenu(self):
235         """
236         Efface les stations.
237
238         Vide le gestionnaire contenant les stations.
239
240         - depuis - 1.3
241
242         - auteur - Lucien Chaboudez
243         """
244         self.__gestionnaire.delContenu()
245
246
247
248     def getPosition(self, noStation):
249         """
250         Donne la position d'une station.
251
252         Permet de connaitre la position de la station, dont le no est passe,
253         sur le graphe en fonction des informations contenues dans les sommets.
254
255         noStation (int) : le no de la station dont on desire la position.
256
257         retourne (Tuple(Tuple(float,float),Tuple(float,float)) --
258         un tuple contenant 2 tuples.
259         1 avec la position (x,y) et le 2e avec vecteur d'orientation.
260
261         - depuis - 1.4
262
263         - auteur - Lucien Chaboudez
264         """
265         #appelle de la fonction getPosition de la station correspondante.
266         return self.__gestionnaire[str(noStation)].getPosition()
267
268
269     def arc(self, noStation):
270         """
271         renvoie l'arc sur lequel la station se trouve.
272
273         Permet de connaitre l'arc sur laquelle la station se trouve.
274
```

```
275         noStation (int) -- le no de la station dont on veut l'arc
276
277         retourne (Tuple(Sommet, Sommet)) -- un tuple contenant les sommets
278         entre lesquels la station se trouve.
279
280         - depuis - 1.4
281
282         - auteur - Lucien Chaboudez
283         """
284
285         #appel de la fonction arc de la station correspondante
286         return self.__gestionnaire[str(noStation)].arc()
287
288
289
290     def getNbPlacesLibres(self, noStation):
291         """
292         renvoie le nb de places libres dans la station.
293
294         Permet de connaitre le nombre de places qui sont libres dans la station
295         dont le no est passé.
296
297         noStation (int) -- le no de la station dont on veut le nombre de places libres.
298
299         retourne (int) -- le nombre de places libres.
300
301         - depuis - 1.7
302
303         - auteur - Lucien Chaboudez
304         """
305
306         #appel de la fonction getNbPlacesLibres de la station correspondante
307         return self.__gestionnaire[str(noStation)].getNbPlacesLibres()
308
309
310     def getStation(self, noStation):
311         """
312         renvoie la station correspondant au no.
313
314         Permet d'avoir accès à la station dont le no est passé en paramètre.
315
316         noStation (int) -- le no de la station.
317
318         retourne (int) -- la station.
319
320         - depuis - 1.17
321
322         - auteur - Lucien Chaboudez
323         """
324         # retour de la station
325         return self.__gestionnaire[str(noStation)]
```

E.3 Initialisateur

E.3.1 Initialisateur.py

Génération de la demande client. Mise en place de stations et taxis.

```
#!/usr/bin/env python
"""
Module d'initialisation.

$Id: Initialisateur.py,v 1.15 2003/02/09 10:05:04 erreur Exp $
"""
__version__ = '$Revision: 1.15 $'
__author__ = 'EI5a, eivd, SimTaxi (Groupe Burdy)'
__date__ = '2002-12-04'

from GrapheXY import *
from GestionnaireTaxis import *
from Evenement import EvClient
from Chemin import *
from random import *

class Initialisateur:
    """
    Initialisations de SimTaxi.

    Cette classe permet de generer les courses des clients et d'initialiser
    les taxis en les positionnant dans une station.
    """


    def __init__(self):
        """
        Constructeur.

        Permet de créer un objet de la classe.

        - depuis - 1.0

        - auteur - Patrice Ferrot
        """
        # Les courses qui vont etre generees, sous forme de
        # liste d'evenements clients.
        self.__courses = []
        # Le graphe de travail.
        #self.__graphe = GrapheXY('graphe.gr')
        self.__graphe = GrapheXY()

    def genererCourses(self, nbCourses, heurePremiereCourse,
                       heureDerniereCourse, distanceMoyenne = 0, germe = None):
        """
        Generateur de courses.

        Genere la totalite des courses qui seront effectuees.

        nbCourses -- le nombre de courses a generer

        heurePremiereCourse -- l'heure du premier appel d'un client

        heureDerniereCourse -- l'heure du dernier appel d'un client

        distanceMoyenne -- la distance moyenne des courses, 0 pour
                           des distances quelconques

        germe -- le germe utilise pour les generations aleatoires. Si pas
                 de germe specifie, se base sur l'heure actuelle.

        retourne (List) -- les courses generees, liste d'evenements clients.

        - depuis - 1.0

        - auteur - Patrice Ferrot
        """
```

```
68          # Si pas de course a generer.
69          if (nbCourses <= 0):
70              return self.__courses
71
72          # Le generateur de nombre aleatoires.
73          generateur = Random()
74          # La distance moyenne des courses.
75          moyenneTemp = 0
76          # Avoir tous les sommets du graphe.
77          listeSommets = self.__graphe.listeSommets()
78          # Le nombre de sommets du graphe.
79          nbSommets =  len(listeSommets)
80          # Les courses.
81          coursesTemp = []
82
83          # Utiliser le germe desire.
84          generateur.seed(germe)
85
86          # Le nombre de courses crees.
87          i = 0
88          # Le nombre de passage dans la boucle.
89          j = 0
90          # Pour generer le bon nombre de courses.
91          #for i in range(nbCourses):
92          while i < nbCourses:
93              # Choisir le premier sommet de depart aleatoirement.
94              #print listeSommets
95              sommetDepart1 = listeSommets[int(self.__nbAleatoire(generateur) *
96                              nbSommets)]
97              # Choisir l'autre sommet de depart aleatoirement dans la
98              # liste des sommets voisins.
99              listeSommetsVus = self.__graphe.sommetsVus(sommetDepart1)
100             sommetDepart2 = listeSommetsVus[int(self.__nbAleatoire(generateur) *
101                             len(listeSommetsVus))]
102
103             # Pour ne pas faire un trajet nul, supprimer les deux sommets
104             # de depart.
105             listeSommets.remove(sommetDepart1)
106             listeSommets.remove(sommetDepart2)
107             nbSommets = nbSommets - 2
108             # Choisir le premier sommet d'arrivee aleatoirement.
109             sommetArrivee1 = listeSommets[int(self.__nbAleatoire(generateur) *
110                              nbSommets)]
111             # Choisir l'autre sommet d'arrivee aleatoirement dans la liste des
112             # sommets voisins.
113             listeSommetsVus = self.__graphe.sommetsVus(sommetArrivee1)
114             sommetArrivee2 = listeSommetsVus[int(
115                              self.__nbAleatoire(generateur)*
116                              len(listeSommetsVus))]
117             # Remettre les sommets qui avaient ete ecartes.
118             listeSommets.append(sommetDepart1)
119             listeSommets.append(sommetDepart2)
120             nbSommets = nbSommets + 2
121
122             # Trajet pour effectuer la course.
123             cheminCourse = self.__graphe.cheminPlusCourt(
124                            (sommetDepart1, sommetDepart2),
125                            (sommetArrivee1, sommetArrivee2))
126             coursesTemp.append((cheminCourse,))
127             distanceCourse = coursesTemp[i][0].distTotalPos()
128
129             # S'il faut supprimer cette derniere course pour adapter
130             # la moyenne.
131             if not (distanceMoyenne == 0) and (
132                (i > (nbCourses/3) and j < (10*nbCourses)) and (
133                ((moyenneTemp > distanceMoyenne) and
134                 (distanceCourse > distanceMoyenne)) or
135                ((moyenneTemp < distanceMoyenne) and
136                 (distanceCourse < distanceMoyenne))
137                )) :
138                 # La supprimer.
139                 coursesTemp.remove(coursesTemp[i])
140             # Sinon, on garde cette course.
141             else:
```

```
142                 # Mise a jour de la moyenne.
143                 moyenneTemp = ((moyenneTemp*(i)) + distanceCourse) / (i+1)
144                 # Generation des heures des courses.
145                 if (i == 0):
146                     coursesTemp[i] = (heurePremiereCourse,) + coursesTemp[i]
147                 elif (i == 1):
148                     coursesTemp[i] = (heureDerniereCourse,) + coursesTemp[i]
149                 else:
150                     heure = (int(self.__nbAleatoire(generateur) *
151                             (heureDerniereCourse-heurePremiereCourse)) +
152                             heurePremiereCourse)
153                     coursesTemp[i] = (heure,) + coursesTemp[i]
154 
155                 # Une course de plus.
156                 i = i + 1
157 
158             # Un passage de plus dans la boucle.
159             j = j + 1
160 
161         # Trier les courese selon leur ordre chronologique.
162         coursesTemp.sort()
163 
164         # Affiche la moyenne des distances des courses et les courses.
165         #print coursesTemp
166         #print "Moyenne des distances des courses : ", moyenneTemp
167 
168         # Creer une liste d'evenements clients.
169         while len(coursesTemp) > 0:
170             pseudoChemin = (coursesTemp[0][1].posDepart(),coursesTemp[0][1].posArrivee())
171             self.__courses.append(EvClient(coursesTemp[0][0],
172                                    pseudoChemin))
173             coursesTemp.remove(coursesTemp[0])
174 
175         # Retourner la liste d'evenements clients.
176         return self.__courses
177 
178 
179     def genererStations(self, nbStations, nbPlaces, germe = None):
180         """
181         Initialise les stations.
182 
183         Place le nombre voulu de stations dans le graphe.
184 
185         nbStations -- le nombre de taxis a placer.
186 
187         nbPlaces -- liste indiquant le nombre de place que doivent contenir
188                     les stations. Si moins d'elements dans la liste que de
189                     stations, utilise le dernier pour toutes les suivantes,
190                     si plus d'elements dans la liste, ne tient pas compte
191                     des supplementaires.
192 
193         germe -- le germe utilise pour les generations aleatoires. Si pas
194                  de germe specifie, se base sur l'heure actuelle.
195 
196         - depuis - 1.5
197         - auteur - Patrice Ferrot
198         """
199         # Si pas de station a creer.
200         if nbStations < 1:
201             return
202 
203         # Le generateur de nombre aleatoires.
204         generateur = Random()
205         # Utiliser le germe desire.
206         generateur.seed(germe)
207 
208         # Avoir tous les sommets du graphe.
209         listeSommets = self.__graphe.listeSommets()
210         # Le nombre de sommets du graphe.
211         nbSommets =  len(listeSommets)
212         # Le gestionnaire de stations.
213         gestStations = GestionnaireStations()
214         # La liste des sommets supprimes.
215         listeSommetsSupprimes = []
```

```
216
217         # Creer le nombre de stations voulues.
218         for i in range(nbStations):
219             # Le bon nombre de places pour cette station.
220             if i < len(nbPlaces):
221                 places = nbPlaces[i]
222             # Choisir le premier sommet de la station.
223             sommet1 = listeSommets[int(self.__nbAleatoire(generateur) *
224                       nbSommets)]
225
226             """
227             print "Liste sommets :  ", listeSommets
228             print "Liste sommets supprimes : ", listeSommetsSupprimes
229             """
230
231             # La liste des sommets voisins.
232             listeSommetsVus = self.__graphe.sommetsVus(sommet1)
233
234             """
235             print "Liste sommets vus avant : ", listeSommetsVus
236             """
237
238             # Supprimer les sommets deja utilises pour une
239             # station, si possible.
240             for j in range(len(listeSommetsSupprimes)):
241                 if len(listeSommetsVus) > 1:
242                     try:
243                         listeSommetsVus.remove(listeSommetsSupprimes[j])
244                     except:
245                         pass
246
247             """
248             print "Liste sommets vus apres : ", listeSommetsVus
249             """
250
251             # Choisir le deuxieme sommet de la station.
252             sommet2 = listeSommetsVus[int(self.__nbAleatoire(generateur) *
253                       len(listeSommetsVus))]
254
255             """
256             print "Sommet1 : ", sommet1
257             print "Sommet2 : ", sommet2
258             print
259             """
260
261             # Pour ne pas placer deux stations au meme endroit.
262             # Les tests assurent qu'il n'y aie pas d'erreur, mais possibilité
263             # de plusieurs stations au meme endroit...
264             if len(listeSommets) > 1:
265                 listeSommets.remove(sommet1)
266                 nbSommets = nbSommets - 1
267                 listeSommetsSupprimes.append(sommet1)
268             if len(listeSommets) > 1:
269                 try:
270                     listeSommets.remove(sommet2)
271                     nbSommets = nbSommets - 1
272                     listeSommetsSupprimes.append(sommet2)
273                 except:
274                     pass
275
276             # Ajouter la station.
277             gestStations.addStation(places, \
278              (sommet1, self.__graphe.attributsSommet(sommet1)), \
279              (sommet2, self.__graphe.attributsSommet(sommet2)) )
280
281
282     def initialiserTaxis(self, nbTaxis, germe = None):
283         """
284         Initialise les taxis.
285
286         Place le nombre voulu de taxis dans les stations.
287
288         nbTaxis -- le nombre de taxis a placer.
289
```

```
290         germe -- le germe utilise pour les generations aleatoires. Si pas
291                  de germe specifie, se base sur l'heure actuelle.
292 
293         - depuis - 1.0
294         - auteur - Patrice Ferrot
295         """
296 
297         # Si pas de taxi a placer.
298         if nbTaxis < 1:
299             return
300 
301         # Le generateur de nombre aleatoires.
302         generateur = Random()
303         # Utiliser le germe desire.
304         generateur.seed(germe)
305 
306         # Le gestionnaire utilise.
307         gestStations = GestionnaireStations()
308         # Les taxis
309         gestTaxis = GestionnaireTaxis()
310 
311         # Le nombre de stations et leur liste.
312         nbStations = gestStations.getNbStations()
313         listeStations = range(1, nbStations + 1)
314 
315         # Pour tous les taxis demandes.
316         for i in range(nbTaxis):
317             # Si plus de place, sortir.
318             if nbStations == 0:
319                 break
320             # Choisir la station ou placer le taxi.
321             noStation =  listeStations[int(self.__nbAleatoire(generateur) *
322                          nbStations)]
323             # Placer le taxi.
324             #gestStations.affecterTaxi(noStation, i)
325             gestTaxis.addTaxi(noStation)
326             # Si la station est pleine, la supprimer des stations possibles.
327             if gestStations.getNbPlacesLibres(noStation) == 0:
328                 listeStations.remove(noStation)
329                 nbStations = nbStations - 1
330 
331 
332     def __nbAleatoire(self, gene):
333         """
334         Generateur aleatoire.
335 
336         Retourne un nombre aleatoire de l'intervalle [0,1[.
337 
338         gene -- le generateur utilise.
339 
340         retourne (Float) -- le nombre aleatoire genere.
341 
342         - depuis - 1.0
343 
344         - auteur - Patrice Ferrot
345         """
346 
347         nb = gene.random()
348         if nb == 1.0:
349            nb = nb - 0.01
350         return nb
```

E.4 Graphe (réseau)

E.4.1 GrapheXY.py

Notre graphe cartésien, sa génération, l'algorithme du plus court chemin.

```
#!/usr/bin/env python
"""
Module contenant la classe GrapheXY.

$Id: GrapheXY.py,v 1.24 2003/02/20 02:00:40 leyonel Exp $
"""
__version__ = '$Revision: 1.24 $'
__author__ = 'EI5a, eivd, SimTaxi (Groupe Burdy)'
__date__ = '2002-11-10'


from Graphe import *
from Point import *
from QueuePriorite import *
from Chemin import *
from random import *

# les exceptions
erreurTypePoint = "l'attribut du sommet n'est pas un objet Point"
fichierChemins = 'dumpChemins'

class GrapheXY(Graphe):
    """
    Graphe oriente et pondere (pondere par des attributs).

    Les sommets et les arcs ont des attributs de n'importe quel type.
    """

    def init(self, fichierImport = None):
        """
        Cette methode sert a creer un graphe.

        On peut importer un graphe a partir d'un fichier donne.

        fichierImport (String) -- Le nom du fichier a importer

        retourne (Graphe) -- Un objet Graphe

        - depuis - 1.0

        - auteur - Lionel Guélat
        """
        # constructeur parent
        Graphe.init(self, fichierImport)

        import pickle
        try:
            print 'Chargement des chemins déjà calculés...'
            self.__chemins = pickle.load(file(fichierChemins))
            self.__nbCheminsLoad = len(self.__chemins)
            print self.__nbCheminsLoad, 'arbres existants.'
        except :
            self.__nbCheminsLoad = 0
            self.__chemins = {} # pour stoquer les chemins calculés de chaque sommet


    def dump(self):
        """
        Dump (sauvegarde après transformation) de la structure contenant
        les arbres de chemins les plus courts. Le dump est fait uniquement
        s'il y a de nouveaux arbres.

        Le fichier (%s) est sous forme binaire.

        - depuis - 1.20

        - auteur - Julien Burdy
```

```
68          """ % fichierChemins
69          import pickle
70          if self.__nbCheminsLoad < len(self.__chemins):
71              print 'Dump des chemins (%d nouveaux arbres)' % (len(self.__chemins) - self.
                    __nbCheminsLoad)
72              pickle.dump(self.__chemins, file(fichierChemins,'w'), True)
73              self.__nbCheminsLoad = len(self.__chemins)
74              print 'Dump OK'
75
76
77      def insererSommet(self, nomSommet, point):
78          """
79          Cette methode permet d'inserer un sommet avec son point.
80
81          Exception levee si le point n'est pas un objet Point.
82
83          nomSommet -- Le nom du sommet
84
85          Point point -- Le point du sommet
86
87          - depuis - 1.2
88
89          - auteur - Joel Jaquemet
90          """
91          # controle du type Point
92          try: point.getX()
93          except: raise erreurTypePoint
94
95          # inserer le sommet avec son point
96          Graphe.insererSommet(self, nomSommet, point.copy())
97
98
99      def insererArc(self, sommetDep, sommetArr):
100         """
101         Cette methode permet d'inserer un arc dont on calcule sa longueur.
102
103         Exception levee si les sommets ne sont pas definis dans le graphe ou
104         si l'arc est deja defini.
105
106         sommetDep -- Le nom du sommet de depart de l'arc
107
108         sommetArr -- Le nom du sommet d'arrivee de l'arc
109
110         - depuis - 1.0
111
112         - auteur - Joel Jaquemet
113         """
114         # insertion de l'arc avec sa longueur calculee
115         Graphe.insererArc(self, sommetDep, sommetArr,
116                           self.attributsSommet(sommetDep).
117                           distance(self.attributsSommet(sommetArr)))
118
119
120
121     def remplacerAttributsSommet(self, sommet, point):
122         """
123         Cette methode permet de modifier le point d'un sommet.
124
125         On recalcule la logueur des arcs relies au sommet.
126         Exception levee si le point n'est pas un objet Point.
127
128         sommet -- Le nom du sommet
129
130         point (Point) -- Le nouveau point du sommet
131
132         - depuis - 1.2
133
134         - auteur - Joel Jaquemet
135         """
136         # controle du type Point
137         try: point.getX()
138         except: raise erreurTypePoint
139
140         # modifier le point du sommet
```

```
141         Graphe.remplacerAttributsSommet(self, sommet, point.copy())
142         # mettre a jour les nouvelles longueurs des arcs sortants
143         for s in self.sommetsVus(sommet):
144             self.remplacerAttributsArc(sommet, s,
145                                        self.attributsSommet(sommet).
146                                        distance(self.attributsSommet(s)))
147         # mettre a jour les nouvelles longueurs des arcs entrants
148         for s in self.sommetsVoyants(sommet):
149             self.remplacerAttributsArc(s, sommet,
150                                        self.attributsSommet(sommet).
151                                        distance(self.attributsSommet(s)))
152
153
154
155     def cheminPlusCourt(self, arcDepart, arcFin):
156         """
157         Algo du chemin le plus court
158
159         arcDepart -- Arc de départ (tuple de sommets)
160
161         arcFin -- Arc de fin (tuple de sommets)
162
163         retourne (Chemin) -- Le chemin le plus court
164
165         - depuis - 1.5
166
167         - auteur - Lionel Guelat
168         """
169         infini = 999999999999999
170
171         # retourne le chemin le plus court
172         # sans tenir compte des arc bidirectionnels pour le depart et la fin
173         def CPC(arcDepart, arcFin):
174
175             # cas trivial
176             if arcDepart[1] == arcFin[0]:
177               return Chemin([arcDepart[0], arcDepart[1], arcFin[1]] \
178                     , [0, self.attributsArc(arcDepart[0], arcDepart[1]) \
179                       , self.attributsArc(arcDepart[1], arcFin[1])])
180
181             # les deux sommets à relier
182             depart = arcDepart[1]
183             fin = arcFin[0]
184
185             # ATTENTION si des arcs ont été supprimés du graphe!!
186
187             # voir si ce sommet de départ a déjà été demandé
188             if depart in self.__chemins.keys():
189
190                 # récupérer les éléments calculés précédemment
191                 parents = self.__chemins[depart][0]
192                 priorites = self.__chemins[depart][1]
193
194             # sinon chercher le chemin demandé
195             else:
196
197                 # liste des sommets visités
198                 visites = {}
199
200                 # liste des priorités
201                 priorites = {}
202
203                 for sommet in self.listeSommets():
204                     # marquer tous les sommets comme non visites
205                     visites[sommet] = False;
206                     # initialiser les priorités
207                     priorites[sommet] = infini
208
209                 # liste des sommets parents
210                 parents = {}
211                 parents[depart] = arcDepart[0]
212
213                 # pour le tri de la liste (compare les distances)
214                 def comp(a, b):
```

```
215                 return priorites[a] < priorites[b]
216
217             # la liste des sommets triés par priorité
218             liste = QueuePriorite(comp)
219
220             # recherche le chemin depuis la fin
221             priorites[depart] = self.attributsArc(arcDepart[0], arcDepart[1])
222             liste.deposer(depart)
223             # tant que la liste contient des sommets
224             while not liste.vide():
225                 # retirer le sommet prioritaire
226                 extrait = liste.prelever()
227                 # marquer le sommet comme visite
228                 visites[extrait] = True
229                 # pour tous les voisins non visites
230                 for voisin in self.sommetsVus(extrait):
231                     if not visites[voisin]:
232                         # voir s'il est mieux de passer par extrait
233                         if self.attributsArc(extrait, voisin) + priorites[extrait] <
                                priorites[voisin]:
234                             # modifier la priorité
235                             priorites[voisin] = self.attributsArc(extrait, voisin) +
                                    priorites[extrait]
236                             # enregistrer le sommet qui permet de l'atteindre
237                             parents[voisin] = extrait
238                             # ajouter à la liste
239                             liste.deposer(voisin)
240
241             # stoquer l'arbre des chemins de ce sommet
242             self.__chemins[depart] = (parents, priorites)
243
244         # verifier si le chemin désiré existe
245         if not fin in parents.keys():
246            raise Exception, 'Pas de chemin entre ces deux sommets'
247
248         # construire le chemin
249         sommetsChemin = []
250         distancesChemin = []
251         sommet = fin
252         sommetsChemin.insert(0, sommet)
253         distancesChemin.insert(0, priorites[sommet])
254         while not parents[sommet] == depart:
255               sommet = parents[sommet]
256               sommetsChemin.insert(0, sommet)
257               distancesChemin.insert(0, priorites[sommet])
258         sommetsChemin.insert(0, depart)
259         distancesChemin.insert(0, priorites[depart])
260
261         # ajouter les deux sommets des extremites
262         sommetsChemin.insert(0, arcDepart[0])
263         distancesChemin.insert(0, 0)
264         sommetsChemin.append(arcFin[1])
265         distancesChemin.append(priorites[fin] \
266             + self.attributsArc(arcFin[0], arcFin[1]) )
267
268         return Chemin(sommetsChemin, distancesChemin)
269
270
271
272     ##################################################
273     # recherche la meilleure des quatre possibilités
274
275     a = CPC((arcDepart[0], arcDepart[1]), (arcFin[0], arcFin[1]))
276     d = a.distTotalSommets()
277
278     if self.arcDefini(arcDepart[1], arcDepart[0]):
279         b = CPC((arcDepart[1], arcDepart[0]), (arcFin[0], arcFin[1]))
280         if b.distTotalSommets() < d:
281             a = b
282             d = a.distTotalSommets()
283
284     if self.arcDefini(arcFin[1], arcFin[0]):
285         b = CPC((arcDepart[0], arcDepart[1]), (arcFin[1], arcFin[0]))
286         if b.distTotalSommets() < d:
```

```
287                 a = b
288                 d = a.distTotalSommets()
289 
290             if self.arcDefini(arcDepart[1], arcDepart[0]):
291                 b = CPC((arcDepart[1], arcDepart[0]), (arcFin[1], arcFin[0]))
292                 if b.distTotalSommets() < d:
293                     a = b
294 
295         return a
296 
297 
298     def genererGraphe(self, nbSommets, nbArcs, distanceMax, germe = None):
299         """
300         Cette methode permet de generer un graphe.
301 
302         Exception levee si le graphe ne peut pas etre connexe.
303 
304         nbSommets (Int) -- Le nombre de sommets
305 
306         nbArcs (Int) -- Le nombre d'arcs
307 
308         distanceMax (Float) -- La distance maximale entre 2 sommets
309 
310         germe (Int) -- Le germe de la fonction aleatoire
311 
312         - depuis - 1.8
313 
314         - auteur - Joel Jaquemet
315         """
316         # controle des parametres
317         if nbSommets < 8: raise Exception, 'Nombre de sommets trop petit: ' + `nbSommets`
318         if distanceMax <= 0: raise Exception, 'Distance max. invalide: ' + `distanceMax`
319 
320         # calculer le nb de lignes (et de colonnes)
321         nbLignes = int(nbSommets**0.5)
322         if nbLignes**2 < nbSommets: nbLignes = nbLignes + 1
323 
324         # controle du nombre d'arcs donné
325         if 2 * (nbSommets - 1) > nbArcs:
326             raise Exception, "Nombre d'arcs trop petit: " + `nbArcs`
327         if nbArcs > -4 * (nbLignes**2 + nbLignes - 2 * nbSommets):
328             raise Exception, "Nombre d'arcs trop grand: " + `nbArcs`
329 
330         # initialisation de la fonction aleatoire
331         rand = Random(germe)
332 
333         # vider le graphe
334         Graphe.initialiser(self)
335 
336         # palcement des sommets de maniere optimale dans l'espace disponible
337 
338         # calculer la longueur des cotes du graphe
339         longCote = (2**0.5 * distanceMax) / 2.0
340 
341         # calculer l'espace minimum entre les sommets
342         distanceMin = (longCote / (nbLignes - 1)) / 20.0
343 
344         # calculer l'espace dans lequelle peut se trouver un sommet
345         espaceSommet = (longCote - (nbLignes - 1) * distanceMin) / nbLignes
346 
347         # placement des sommets
348         sommetCourant = 1  # numero du sommet courant
349         coord = Point() # coordonnees du sommets courant
350 
351         yTemp = 0
352         while yTemp < longCote:
353             xTemp = 0
354             while xTemp < longCote:
355                 coord.setXY(xTemp + rand.randrange(int(espaceSommet) + 1),\
356                             yTemp + rand.randrange(int(espaceSommet) + 1))
357                 self.insererSommet(sommetCourant, coord)
358                 sommetCourant = sommetCourant + 1
359                 xTemp = xTemp + espaceSommet + distanceMin
360             yTemp = yTemp + espaceSommet + distanceMin
```

```
361
362         # pour connaitre les sommets qui peuvent etre lies avec celui donne
363         def liaisonsPossibles(sommet):
364             # liste des liaisons possibles
365             sommetsProches = []
366             sommetsProches.append(sommetCourant - nbLignes)
367             sommetsProches.append(sommetCourant + nbLignes)
368             if sommetCourant % nbLignes != 1:
369                 sommetsProches.append(sommetCourant - 1)
370             if sommetCourant % nbLignes != 0:
371                 sommetsProches.append(sommetCourant + 1)
372
373             # retirer les arcs impossibles
374             sommetsProchesTemp = []
375             # copie de la liste
376             for sommetArr in sommetsProches:
377                 sommetsProchesTemp.append(sommetArr)
378
379             # enlever les sommets inexistants ou deja lies
380             for sommetArr in sommetsProchesTemp:
381                 if not self.sommetDefini(sommetArr):
382                     sommetsProches.remove(sommetArr)
383                 elif self.arcDefini(sommetCourant, sommetArr):
384                     sommetsProches.remove(sommetArr)
385
386             return sommetsProches
387
388         # generer la connexite du graphe
389         sommetsNonRelies = self.listeSommets()
390         chemin = []
391
392         # choisir un sommet de depart
393         sommetCourant = sommetsNonRelies[rand.randrange(len(sommetsNonRelies))]
394         # le sommet courant va etre relie
395         sommetsNonRelies.remove(sommetCourant)
396         chemin.append(sommetCourant)
397
398         while len(sommetsNonRelies) > 0:
399             # liste des liaisons possibles
400             listeSommets = liaisonsPossibles(sommetCourant)
401
402             # retirer les sommets deja visites
403             listeSommetsTemp = []
404             for sommetSuivant in listeSommets: # copie de la liste
405                 listeSommetsTemp.append(sommetSuivant)
406
407             for sommetSuivant in listeSommetsTemp:
408                 if not sommetSuivant in sommetsNonRelies:
409                     listeSommets.remove(sommetSuivant)
410
411             if len(listeSommets) > 0:
412             # on peut continuer => choisir le sommet suivant
413                 sommetSuivant = listeSommets[rand.randrange(len(listeSommets))]
414                 # insertion des 2 arcs
415                 self.insererArc(sommetCourant, sommetSuivant)
416                 self.insererArc(sommetSuivant, sommetCourant)
417                 # le sommet suivant a ete relie
418                 sommetsNonRelies.remove(sommetSuivant)
419                 chemin.append(sommetSuivant)
420                 # passer au sommet suivant
421                 sommetCourant = sommetSuivant
422
423             else:
424             # cul de sac => revenire en arriere
425                 chemin.remove(sommetCourant)
426                 sommetCourant = chemin[len(chemin) - 1]
427
428         # reduction du nombre de sommets
429         while self.nbSommets() > nbSommets:
430             # choisir un sommet a supprimer
431             sommetCourant = self.listeSommets()[rand.randrange(len(self.listeSommets()))]
432
433             # pour conserver les liaisons des sommets
434             for voyant in self.sommetsVoyants(sommetCourant):
```

```
435                 for vu in self.sommetsVus(sommetCourant):
436                     if voyant != vu:
437                         if not self.arcDefini(voyant, vu):
438                             self.insererArc(voyant, vu)
439 
440             # suppression du sommet
441             self.supprimerSommet(sommetCourant)
442 
443         # augmentation du nombre d'arcs
444         while self.nbArcs() < nbArcs:
445             # choisir un sommet de depart de l'arc
446             sommetCourant = self.listeSommets()[rand.randrange(len(self.listeSommets()))]
447 
448             # liste des liaisons possibles
449             listeSommets = liaisonsPossibles(sommetCourant)
450 
451             # garde fou pour eviter d'essayer d'ajouter un arc alors qu'on ne peu plus
452             if len(listeSommets) > 0:
453                 # insertion d'un arc
454                 self.insererArc(sommetCourant, listeSommets[rand.randrange(len(
                        listeSommets))])
```

E.5 SimTaxi

E.5.1 SimTaxi.py

Le programme principal. Contenant la boucle de simulation. Il est peut être peu compréhensible car expérimental.

```
#!/usr/bin/env python
"""
Programme principal.

$Id: SimTaxi.py,v 1.25 2003/03/02 19:53:36 erreur Exp $
"""
ID = "$Id: SimTaxi.py,v 1.25 2003/03/02 19:53:36 erreur Exp $"

from threading import Thread
from Central import Central
from GestionnairePreferences import GestionnairePreferences
from GrapheXY import GrapheXY
from Initialisateur import Initialisateur
from time import sleep
import sys
sys.stderr = file('ERREURS.log', 'a')#.write('\n'*5+ID)

fichierClients = 'dumpClients'

print '\n'*3+ID+'\n'


def dumpClients(liste):
    """
    Dump (sauvegarde après transformation) de la demande (liste d'év clients).

    Le fichier (%s) est sous forme texte.

    liste -- la liste d'événements clients.

    - depuis - 1.21

    - auteur - Julien Burdy
    """ % fichierClients
    f = file(fichierClients,'w')
    for i in liste: f.write(str(i.temps())+';'+str(i.positions())+'\n')
    f.close()


def loadClients():
    """
    Chargement du dump de la demande (liste d'év clients).

    Le fichier (%s) est sous forme texte.

    retourne (list) -- liste d'événements clients.

    - depuis - 1.21

    - auteur - Julien Burdy
    """ % fichierClients
    from Evenement import EvClient
    from string import split
    clients = []
    liste = file(fichierClients).readlines()
    for i in liste:
        spl = split(i, ';')
        clients.append(EvClient(eval(spl[0]), eval(spl[1])))
    return clients



if __name__ == '__main__':
    print """Utilisez startSimTaxi.pyw pour lancer le programme
(a lancer depuis la console pour voir le log)"""
    raw_input()
```

```
67  else:
68
69      D = 1 # 1 = DEBUG
70
71      # recuperation des préférences (paramètres + configuration)
72      gp = GestionnairePreferences('config.txt')
73
74      ##### initialisation de l'interface utilisateur, permettant à l'utilisateur
75      ##### de changer certain paramètre avant les initialisations suivantes (donc bloquant
            )
76
77      # création des objets
78      graphe = GrapheXY(gp.valeurDe('fichierGraphe'))
79      central = Central()
80
81      initialisateur = Initialisateur()
82
83      # mise ne place des taxis/stations
84      print 'Initialisation stations...'
85      initialisateur.genererStations(gp.valeurDe('ndStation'),gp.valeurDe('tailleStation'))
86      print 'Initialisation taxis...'
87      initialisateur.initialiserTaxis(gp.valeurDe('nbTaxi'))
88
89
90      # generation des courses et initialisation du central
91      try:
92          print 'Recup de la demande...'
93          clients = loadClients()
94          if len(clients) != gp.valeurDe('nbCoursesJour'): raise
95          print len(clients), 'clients'
96          #for i in clients: graphe.cheminPlusCourt(i.positions()[0],i.positions()[1])
97      except:
98          print 'Génération de la demande (long la 1ère fois)...'
99          clients = initialisateur.genererCourses(gp.valeurDe('nbCoursesJour'),
100                                                  gp.valeurDe('hPremiereCourse'),
101                                                  gp.valeurDe('hDerniereCourse'),
102                                                  gp.valeurDe('moyenneCourseKM'))
103         dumpClients(clients)
104
105     central.initEv(clients)
106
107     if gp.valeurDe('gui'):
108         from SimTaxiGUI import SimTaxiGUI
109         gui = SimTaxiGUI(0)
110
111 class T(Thread):
112     def run(self):
113         # tant qu'il y a des evenements
114         print '==== Début de la simulation'
115         while central.evenement():
116             evenement = central.traiterProchainEvenement() # renvois l'evenement traite (
                    pour le gui)
117             #if D: print str(evenement) + "\n"
118             temps = evenement.temps() # saisie du temps de l'evenement
119             print (temps/(3600.0*24.0))*100.0, '%'
120             #if D: print '=== temps :', temps
121             if gp.valeurDe('gui'): gui.rafraichir(temps, evenement) # le gui raffraichi
                    et renvois vrai s'il veut stopper
122
123             # si le prochain evenement est trop loin dans le temps, on continue de
                    raffraichir avant
124             # de traiter ce prochain evenement
125             while gp.valeurDe('pseudoContinu') and central.evenement() and central.
                    intervalleProchainEvement(temps) > 1 \
126                   and gp.valeurDe('gui'):
127                 temps += 1
128                 #if D: print '=== tempsInter :', temps
129                 if gp.valeurDe('gui'): gui.rafraichir(temps, None)
130                 sleep(gp.valeurDe('dureeSec'))
131         print '==== FIN'
132         graphe.dump()
133
134 def main():
135     t = T()
```

```
136     if gp.valeurDe('gui'): t.setDaemon(True)
137     t.start()
138     if gp.valeurDe('gui'): gui.start()
139     graphe.dump()
```