Plug-in multi-agent pour une plate-forme de simulation de mobilité urbaine

Étudiants :
Barberot Mathieu
Racenet Joan

Tuteurs :
Lang Christophe
Marilleau Nicolas

28 janvier 2013
UFR ST - Besançon

Table des matières

1 Simulation Urbaine **3**

2 La plate-forme : GAMA **4**

3 Présentation du plug-in **5**
3.1 La modélisation d'une ville avec GAMA 5
3.2 Contenu du plugin 6

4 Mise en place et premiers contacts **7**
4.1 Téléchargement et exécution de GAMA 7
4.2 Mise en place du plug-in 7

5 Sens uniques **8**
5.1 Les fichiers SIG 8
5.2 Implémentation 8
5.3 Différentes approches 8
5.3.1 Première approche : une sorte de mapping relationnel 9
5.3.2 Seconde approche : le sens de digitalisation 9
5.4 Première approche : le mapping relationnel 9
5.4.1 Pourquoi ? 9
5.4.2 Côté SIG 9
5.4.3 Côté plug-in 9
5.4.4 Côté GAML 10
5.4.5 Problème 11
5.5 Deuxième approche : à partir du sens de digitalisation des points 11
5.5.1 Avantages de la méthode 11
5.5.2 Problème posé et solutions envisagées 12
5.5.3 Côté SIG 12
5.5.4 Côté plug-in 13
5.5.5 Côté GAML 13
5.5.6 Et la création d'un statement ? 13

6 Limitations de vitesse **14**
6.1 Objectif 14
6.2 Organisation du développement 14
6.3 Première étape : Affecter la vitesse maximum 14
6.4 Seconde étape : Le nombre d'agent sur le tronçon 15
6.4.1 La méthode active 15
6.4.2 La méthode passive 15
6.5 Troisième étape : Généralisation de la formule 15

7 Le plug-in final **16**

Chapitre 1

Simulation Urbaine

Un système multi-agents est, comme son nom l'indique, un ensemble d'agents (entités autonomes dotées d'une intelligence artificielle constituée de comportements prédéfinis) capables d'interactions aussi bien entre eux, qu'avec leur environnement et capables de s'adapter en fonction de l'état de celui-ci.
Les SMA (pour systèmes multi-agents) proposent donc une approche assez intuitive pour simuler des comportements « de masse »- les premiers exemples qui nous viendraient rapidement en tête seraient les jeux vidéos ou le cinéma, où des effets spéciaux mettant en scène une foule d'entités peuvent être simulés par un SMA de manière crédible, évitant ainsi aux graphistes la tâche colossale d'animer et de coordonner des centaines, voire des milliers de modèles 3D.
Mais outre son application à des systèmes purement virtuels, on s'aperçoit rapidement que la philosophie des SMA est particulièrement adapté pour des simulations ayant pour contexte les sciences humaines. On s'intéressera plus particulièrement ici à une application au domaine de la simulation urbaine. Concrètement, il serait intéressant de pouvoir observer le comportement d'êtres humains dans une ville et de pouvoir mesurer l'impact de certaines modifications de l'environnement sur ces derniers (par exemple, dans quelle mesure serait influencé le trafic routier si j'entreprends des travaux sur tel ou tel axe).
C'est dans cette optique qu'a été développé le projet MIRO qui consiste en une analyse des flux d'activités quotidiennes (comme par exemple le trafic routier ou les mouvement des habitants en fonction de l'heure de la journée) de la ville de Dijon.Le projet tuteuré, dont nous allons ici détailler les tenants et aboutissants, s'inscrit dans cette thématique, étant donné que son objectif est justement de fournir des outils pour simplifier au maximum l'écriture d'une simulation de mobilité urbaine.

Chapitre 2

La plate-forme : GAMA

FIGURE 2.1 – GAMA

Les outils dont nous venons justement de parler ont pour visée d'être intégrés au sein de la plate-forme de développement de systèmes multi-agents nommée GAMA.
GAMA est un IDE (environnement de développement) open-source basé sur Eclipse (donc codé dans le langage Java) et développé par l'UMMISCO (Unité de Modélisation Mathématique et Informatique de Systèmes COmplexes) depuis 2007.
Cette plate-forme est très nettement orientée géomatique, puisqu'elle propose nativement des outils permettant d'exploiter des données complexes issues de Systèmes d'Informations Géographiques (SIG ou GIS en anglais) afin de définir l'environnement des simulations (on pourra par exemple récupérer rapidement le tracé des routes d'une ville et ses bâtiments).
Pour l'écriture des simulations, GAMA propose son propre langage nommé GAML (pour GAMA Language), qui permet de manipuler facilement des agents, leur environnement ainsi que divers types complexes classiques (listes dynamiques, graphes . . .).
Les fonctionnalités proposées par GAMA sont implémentées au sein de plug-ins Eclipse venant se greffer autour du noyau de l'application. Afin de fournir nos propres fonctions, nous passerons nous-mêmes par le développement d'un greffon en se basant sur la version développeur de GAMA, librement téléchargeable. Cette méthode a l'avantage d'éviter aux éventuels développeurs de modifier la base de l'application.

Chapitre 3

Présentation du plug-in

3.1 La modélisation d'une ville avec GAMA

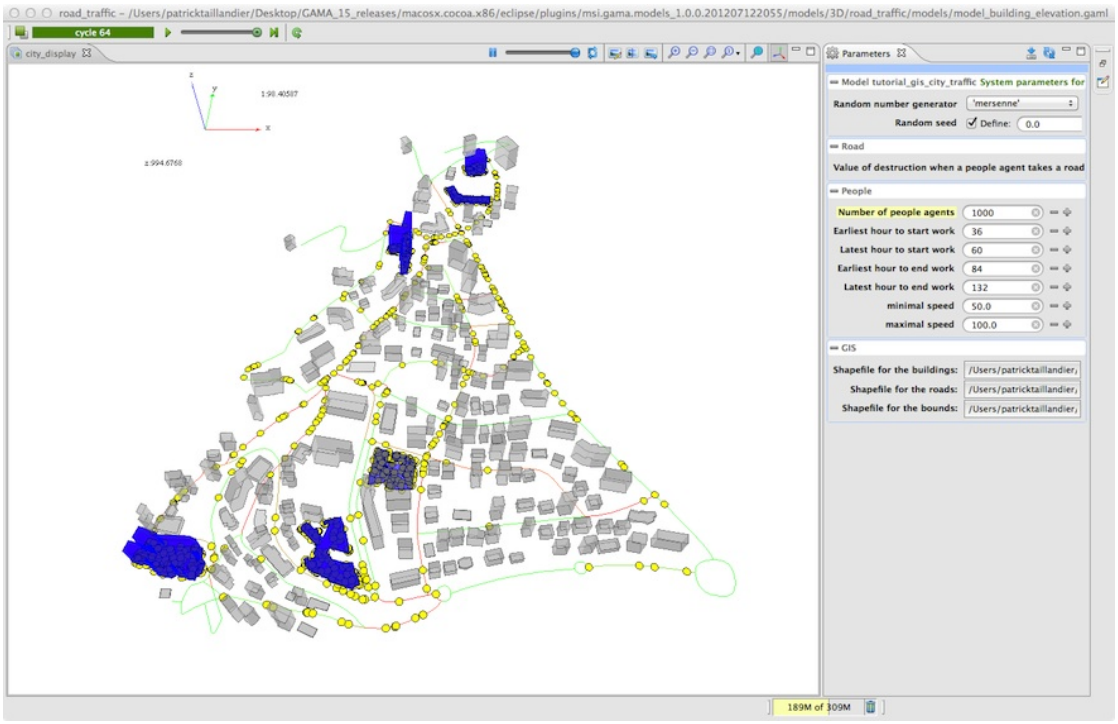

FIGURE 3.1 – Modélisation d'une ville

Comme on l'a vu, importer le tracé d'une ville, ses routes et ses bâtiments, est assez aisé dans GAMA, puisqu'on peut les importer directement depuis un SIG. De même, on peut rapidement exploiter ces données, en les assignant à des types d'agents (dans notre cas, on pourra créer un agent pour les routes, un autre pour les bâtiments et enfin

un dernier pour la population de la ville). Par la suite, nous pourrons créer un graphe en utilisant les routes en tant qu'arêtes et les « services » (objectifs pour la population) en tant que nœuds afin de représenter la ville en mémoire. La ville pourra être ensuite librement parcourue par les agents, ce que GAMA traduit par un parcours de graphe et l'application d'algorithmes de recherches des plus courts chemins (Dijkstra) dans le cas où l'on cherche à faire atteindre un objectif à un agent.
Modéliser la ville en elle-même est donc relativement aisée. En revanche, les choses se compliquent rapidement dès que l'on veut donner des comportements réalistes aux agents de la population, comme leur faire respecter le code de la route (par exemple, respecter les sens uniques ou les limitations de vitesse) ou leur faire subir les contraintes de l'environnement (réduction de la vitesse en fonction du nombre d'autres automobilistes sur un tronçon). Cela reste possible, mais moyennant une certaine quantité de code GAML supplémentaire assez complexe.
Le but de notre plug-in est donc de rendre plus accessibles l'écriture de simulation utilisant ce genre de comportements, en proposant nativement ceux-ci dans le logiciel au développeur GAML. Dans le cadre de notre travail, nous nous sommes concentrés justement sur le respect des sens uniques et des limitations de vitesse.

3.2 Contenu du plugin

La solution que nous avons trouvée pour faciliter cette modélisation est d'embarquer un agent prédéfini représentant une rue, afin d'éviter aux utilisateurs de redéfinir lui-même un tel type à chaque écriture d'une nouvelle simulation. Pour simplifier leur utilisation, ces agents peuvent être instanciés à l'importation d'un fichier SIG comme il était déjà possible de le faire auparavant (une rue dans le fichier SIG correspond à un agent dans GAMA).
Ces agents permettent alors de définir des sens-uniques ou encore une vitesse maximum autorisée, en embarquant des attributs correspondants à ces informations.
En plus des rues, nous avons également pensé mettre en place un agent prédéfini représentant une intersection, permettant ainsi la gestion des priorités ou encore des feux de signalisation et qui peuvent de même servir d'objectifs aux agents représentant les véhicules lors de leurs déplacements.
Enfin, le respect du code de la route passe également par la définition d'actions prédéfinies pour les agents automobilistes lors de leurs déplacement (par exemple : aller à un certain point de la ville). Ces comportements prendront eux-mêmes automatiquement en compte les règles de circulation.

Chapitre 4

Mise en place et premiers contacts

La première étape du projet a consisté à prendre en main la plateforme GAMA, comprendre son fonctionnement (notamment l'écriture de simulations en GAML) et faire nos premiers pas dans la création d'un plug-in.

4.1 Téléchargement et exécution de GAMA

La plateforme GAMA est librement téléchargeable en version stable pour le public et en version développement pour ceux qui, comme nous, désirent en étendre les capacités.
Nous avons téléchargé la première version pour nous familiariser avec la plateforme et son interface ainsi qu'essayer et analyser le code des nombreux exemples fournis dans le programme et par nos tuteurs, comme le projet MIRO (simulation de la ville de Dijon). Nous avons également téléchargé la seconde, directement depuis le serveur SVN du projet GAMA, puis nous l'avons ouvert et configurer son build dans Eclipse sans problèmes majeurs , grâce à la documentation disponible sur le site du projet.

4.2 Mise en place du plug-in

Toujours en suivant la documentation du site, nous avons créé un plug-in « Hello World » pour nous familiariser avec ce nouvel environnement. Notre plug-in permettait ainsi l'affichage du message « Hello World » par les agents. Ce premier essai était nécessaire pour se familiariser avec le jargon propre à GAMA (facets, skills ...) et le langage GAML. Dans ce premier plug-in, nous avons donc créé un skill, qui représente un ensemble de fonctions directement utilisables par un agent (la plateforme contient par exemple un skill « moving » qui permet aux agents de se déplacer au hasard sur un graphe, d'aller à un point donné, de suivre un certain tracé, etc...).
Après cette « mise en bouche », nous avons suivi la même procédure de création de plug-in, mais cette fois, pour y implémenter nos fonctionnalités de mobilité urbaine.

Chapitre 5

Sens uniques

La majeure partie de notre temps de développement fut portée sur l'écriture du comportement permettant le respect des sens-uniques. Une fois que ceux-ci furent respectés par les agents, nous nous sommes penchés sur les limitations de vitesse.

5.1 Les fichiers SIG

Ces fichiers contiennent les informations sur le tracé comme les coordonnées des points ou encore le sens de digitalisation, c'est à dire le sens dans lequel le tracé a été créé dans le logiciel géographique. Ces fichiers peuvent également embarquer une base de données, qui permet en plus de donner des informations plus précises à nos tracés (dans notre cas, on pourra assigner à chaque route une information de type booléenne indiquant si la route est en sens unique ou pas).

5.2 Implémentation

Pour simplifier au maximum l'utilisation de notre plug-in nous avons opté pour l'utilisation de deux types d'agents prédéfinis : l'un pour les routes et l'autre pour les services, c'est à dire un objectif comme un bâtiment, une intersection, ou tout lieu spécifique (piscine, travail, parc, ...).
Les méthodes permettant aux agents de se mouvoir seront incluses dans un « skill ». Les skills définissent des actions réalisables par les agents, comme se déplacer dans notre cas, et sont directement appelables dans le code GAML.

5.3 Différentes approches

Pour résoudre ce problème de respect des sens-uniques, nous avons dès le départ anticipé deux approches :

5.3.1 Première approche : une sorte de mapping relationnel

Au sein du SIG, on attribue à chaque service un identifiant unique. Pour chaque route, on donne une propriété indiquant si elle est à sens unique ou pas ainsi qu'un identifiant (selon le principe des clés étrangères en base de données relationnelles) pour le service de départ et le service d'arrivée, ce qui nous permet d'obtenir un sens de circulation.

5.3.2 Seconde approche : le sens de digitalisation

Comme vu précédemment, le sens de digitalisation des points dans le SIG est l'ordre dans lequel sont tracés les points. Il pourrait donc être utilisé pour déterminer du même coup le sens de circulation (le sens dans lequel on trace une route correspondrait au sens dans lequel cette route doit être empruntée). On attribue de même à chaque route un attribut pour spécifier si la route est à sens unique ou non et, éventuellement, si le sens de digitalisation doit être inversé (pour éviter de devoir retracer une route en cas d'erreur sur la direction).
Évidemment, dans les deux cas, cela impose une certaine convention aux utilisateurs au niveau de la création de leurs fichiers SIG, afin que notre plug-in puisse les traiter correctement.

5.4 Première approche : le mapping relationnel

5.4.1 Pourquoi ?

Tout d'abord car cette approche est la plus simple conceptuellement et nous semblait la plus « propre » sur le papier car on a le contrôle total sur les sens de circulation, contrairement au sens de digitalisation qui nous oblige à tracer les routes tels qu'elle ont été tracées dans le logiciel, ce qui ne correspond pas forcément au sens dans lequel les automobiliste roulent. De même, il nous semblait plus simple ensuite, dans le plug-in, de reconstituer le graphe orienté de la ville en associant les arêtes (les routes) aux bons noeuds (les services) en fonction de leurs identifiants.

5.4.2 Côté SIG

Nous avons travaillé sur un petit tracé comportant quelques routes et intersections. Nous avons créés pour chaque type d'élément, une table d'attributs contenant les identifiants et un attribut indiquant si la rue est sens unique ou pas.

5.4.3 Côté plug-in

Nous avons créé deux classes Java représentant les deux agents spécifiques : la rue (classe StreetAgent) et l'intersection (classe IntersectionAgent) . Chacune de ces classes

comportent des attributs qui seront mappés avec ceux des représentations dans le fichier SIG (un booléen pour signifier le sens unique de la route et des entiers pour les identifiants). Des annotations spécifiques à GAMA sont ajoutés pour avoir accès aux attributs directement dans le code GAML.

5.4.4 Côté GAML

On peut alors utiliser nos agents embarqués directement dans le code GAML. Il est possible de les créer à l'importation du fichier SIG, et de leur donner les valeurs contenues dans ce dernier (identifiant des bâtiments, information du sens unique), comme vous pouvez le voir à la figure 5.1.

```
global
{
    // Importation du fichier GIS
    var roads_file type:string init:"../includes/city_roads.shp";

    graph city_graph;
    init
    {
        // Création des rues depuis le fichier importé
        create roads
            from:roads_file
            with: [waytype::read("type"), max_speed::read("speed")];

        // Création du graphe avec les rues
        set city_graph <- directed(as_edge_graph(list(roads)));
    }
}
```

FIGURE 5.1 – Initialisation des rues

Pour donner un aspect visuel à ces agents, il suffit à l'utilisateur d'en créer un nouveau type, héritant des nôtres, et de lui définir ces propriétés (forme, couleur, taille), comme indiqué dans la figure 5.2.

```
species roads parent:Street
{
    rgb color <- rgb('red') ;
    aspect base {
        draw shape: circle size:15 color: color ;
    }
}
```

FIGURE 5.2 – Héritage de l'agent Street et redéfinition de l'aspect

5.4.5 Problème

Les mouvements des agents sur le tracé est un parcours de graphe et l'algorithme de plus court chemin est implémenté dans une librairie externe à GAMA. Se pose alors le problème de pouvoir reconstituer en mémoire un graphe à partir de nos informations relationnelles, alors que GAMA se base uniquement sur des coordonnées des points pour gérer ses graphes.
On s'aperçoit alors de l'ampleur du travail pour permettre à cette approche de se concrétiser, surtout étant donné le fait que le code Java de l'application est peu (voire pas) commenté ou documenté. Récupérer les éléments de l'API permettant l'implémentation de notre solution impliquerait donc un temps encore plus important, sans compter le supplément à ajouter pour se familiariser à des nouvelles librairies pour l'exécution du parcours (style GraphStream).
Parallèlement, alors que nous commencions sérieusement à bloquer sur le problème, M. Nicolas MARILLEAU nous a envoyé un exemple de simulation où les sens uniques sont gérés à l'aide du sens de digitalisation des points. Le traitement est effectué entièrement en code GAML, ce qui implique l'existence de fonctions dans le code Java pour cela, fonctions que l'on pourra réutiliser dans notre plug-in. Cet exemple nous a de plus clarifier le concept de sens de digitalisation (qui était assez nébuleux pour nous à la base) et le fonctionnement de GAMA pour ses parcours de graphe.

5.5 Deuxième approche : à partir du sens de digitalisation des points

5.5.1 Avantages de la méthode

En effet, l'algorithme de parcours des graphes utilisé dans GAMA se base sur ce sens de digitalisation. Un test en GAML nous l'a confirmé : il existe une commande permettant de déclarer le graphe comme étant orienté. Nous avons donc pu voir que les agents, après avoir utilisé cette commande, parcouraient le graphe dans le sens de digitalisation (nous avons effectué ce test avec un tracé créé par nos soins dans un SIG et importé dans GAMA), alors qu'ils allaient indifféremment dans les deux directions

auparavant.
Le comportement de déplacement inclus dans le skill « moving » (dont on a déjà parlé plus haut) prend donc déjà en compte les graphes orientés, ce qui implique un gain de temps considérable pour nous :

- Nous n'aurons pas besoin d'écrire l'algorithme de parcours de graphe, mais simplement à modifier la structure de ce dernier.
- De ce fait, on pourra gérer le problème directement dans nos agents prédéfinis et nous pourrons donc nous passer de l'écriture d'un « skill » particulier pour gérer leurs mouvements.

5.5.2 Problème posé et solutions envisagées

Nouveau problème : en déclarant le graphe comme étant orienté, les voies à double sens sont alors également considérées comme des sens uniques. Il nous faudra donc, dans le cas des double-sens, dupliquer l'agent représentant la route, inverser son sens de digitalisation et le rajouter à la population des routes (ce qui revient donc, en mémoire, à dupliquer une arête du graphe et de changer son orientation).
Cette opération de duplication pouvait être implémentée de plusieurs manières : soit elle se faisait directement à la construction des agents routes que nous avons fourni dans le plug-in (au moment de l'instanciation, on vérifie si l'agent est une route à double sens-unique, et le cas échéant, on le duplique) soit à l'appel d'un « statement » (mot-clé du langage). Dans ce cas, l'idée aurait été de créer un nouveau « statement », qui prendrait la liste des agents routes en paramètre et qui retournerait cette liste enrichie des duplicats nécessaires au traitement des routes à double-sens.
Même si cette dernière solution aurait été plus propre (l'utilisateur n'ayant pas forcément envie de faire une simulation impliquant une problématique de sens uniques pourrait se passer de la duplication forcée des routes à double-sens) , nous avons préféré implémenter la première solution, d'une part pour une question de facilité au niveau programmation (accès direct aux propriétés de l'agent et à sa population pour y ajouter le duplicat) et d'autre part, à cause d'un cruel manque de documentation à ce sujet dans la documentation développeur de GAMA.

5.5.3 Côté SIG

Contrairement à la méthode relationnelle, il n'existe plus le besoin d'avoir une information concernant les sens de direction dans une table, vu que ceux-ci sont les sens de digitalisation des tracés. En revanche, il est toujours nécessaire d'indiquer pour chaque route si elle est à sens-unique (ou pas) et si le sens de digitalisation doit être inversé (en cas d'erreur ou de changement de sens de circulation). Concrètement, cela se traduit par l'ajout d'un attribut « waytype » dans la base de données du fichier SIG, pouvant prendre la valeur 0 (double-sens), 1 (sens unique) ou 2 (sens unique avec inversion du sens de digitalisation). Cette convention de nommage doit être respectée pour que notre plug-in puisse traiter les informations correctement.

5.5.4 Côté plug-in

Nous avons pu garder la classe représentant les routes (StreetAgent) développée lors de nos essais avec la méthode relationnelle. En revanche, nous avons pu nous passer de la classe des agents représentant les services, étant donné que son seul rôle était de leur fournir un identifiant, ce que nous n'utilisons plus ici.
C'est donc dans la classe StreetAgent que s'effectue la duplication nécessaire au traitement des routes au double sens. En effet, au moment où l'on donne une valeur à l'attribut du type de route (le « waytype »), nous vérifions désormais la valeur de celui-ci et agissons en conséquent.

- Si sa valeur vaut 1 : on ne fait rien, l'agent est une route à sens unique et sera traité correctement dans l'algorithme de parcours de graphe de GAMA.
- Si sa valeur vaut 2 : on récupère la géométrie (qui correspond à une liste ordonnée de points) de l'agent, et on l'inverse (l'ordre de la liste correspondant au sens de digitalisation du tracé dans le SIG).
- Si sa valeur vaut 0 : l'idée est de faire un clone de l'agent courant et d'en inverser la géométrie. Nous nous sommes frottés ici à un obstacle : les objets « Agent » étant déclarés comme immuables, nous n'avons pas pu créer une véritable copie, puisqu'en Java, on ne peut que dupliquer la référence sur un objet immuable et non l'objet en lui même. De fait, en voulant inverser la géométrie de la copie, nous inversions aussi la géométrie de l'objet courant. Pour pallier à ce problème, nous créons donc un nouvel agent, auquel on donne les mêmes propriétés que l'agent courant (sauf, évidemment, sa géométrie que l'on inverse) et l'ajoutons dans la même population que lui.

5.5.5 Côté GAML

Grâce à cette méthode d'implémentation, le tout est resté transparent pour l'utilisateur qui écrit sa simulation en GAML. En effet, pour que le sens de circulation soit bien pris en compte, il lui suffit juste d'utiliser nos agents prédéfinis pour créer ses routes et de déclarer le graphe de la ville comme orienté, la prise en compte du type de route s'effectuant à l'importation du fichier SIG.

5.5.6 Et la création d'un statement ?

Comme nous l'avons signalé au début de cette partie, nous avons songé à créer un mot clé dans le langage (un "statement") qui permettrait de traiter la duplication des routes à double sens dans un autre endroit qu'à l'instanciation de celles-ci.
Lorsque nous avons commencé à nous pencher sur l'approche par sens de digitalisation, nous avons tout de même tenté cette méthode, mais avons rapidement été bloqué par des problèmes triviaux (comment renvoyer un type de donnée dans le code GAML ou encore comment accéder au contenu d'une liste passée en paramètre dans le code GAML dans le code Java). Autrement dit, cette méthode s'est rapidement soldée par un échec et par une perte de temps assez conséquente.

Chapitre 6

Limitations de vitesse

Deuxième étape de nos développement, nous nous sommes penchés sur les limitations de vitesse et de la vitesse d'un agent sur un tronçon, afin de modéliser les engorgements.

6.1 Objectif

Nous voulions permettre aux utilisateurs du plug-in de définir une vitesse maximum à laquelle les agents rouleront sur un tronçon. Ensuite, en fonction du nombre d'agent circulant dans le même sens sur ce tronçon, la vitesse effective de chaque agent diminuerait en fonction d'une formule donnée.

6.2 Organisation du développement

Dans un premier temps, nous avons cherché à affecter une vitesse maximum à un tronçon et de faire rouler les agents à cette vitesse. Cette première étape serait la plus dure car elle nécessitait de mettre une pondération sur un rue, et donc un arc du graphe. Ensuite les agents empruntant cette arc utiliseraient cette pondération automatiquement, grâce aux algorithmes sur les graphes utilisés par GAMA.
La seconde étape consistait à ajouter un nouveau facteur : le nombre d'agent sur le tronçon. Deux approches ont été trouvée : l'une basée sur une attente active et l'autre sur une attente passive. Une fois le nombre d'agent obtenu, il suffisait d'appliquer une formule pour redéfinir la nouvelle vitesse maximum.
Enfin, dernière étape, nous voulions laisser l'utilisateur libre du choix de la formule, qu'il entrerait dans son fichier GAML et qui serait utilisée par notre plug-in.

6.3 Première étape : Affecter la vitesse maximum

Toujours pour simplifier la modélisation de la ville, nous avons choisi de rajouter une nouvelle information à fournir dans le fichier SIG : la vitesse maximum sur la rue.

Ainsi, l'utilisateur ne devra pas manuellement affecter la vitesse à chaque rue. Passé cette formalité, l'agent prédéfini des rues que nous avons développé devait affecter cette valeur à l'arc qu'il représente dans le graphe.
Hélas, deux facteurs nous ont empêchés d'arriver au terme de cette tâche. D'une part, nous n'avons pas réussi à obtenir l'arc du graphe correspondant à une rue a cause du fonctionnement interne complexe de GAMA. D'autre part, le temps imparti aux projet tuteurés arrivait à son terme.

6.4 Seconde étape : Le nombre d'agent sur le tronçon

Malgré la fin du développement, nous avions tout de même quelques idées pour l'implémentation de cette partie : un méthode active et une méthode passive.

6.4.1 La méthode active

La méthode active était fort simple : boucler indéfiniment sur une récupération de tous les agents, puis le calcul de la vitesse et la modification du poids de l'arête. Comme toutes les méthodes actives, elle n'est pas très optimisée puisque le calcul est effectué, qu'il y ait des agents ou non. Il serait plus intéressant de ne faire le calcul qu'à l'entrée ou la sortie d'un agent dans la rue.

6.4.2 La méthode passive

L'idée était donc qu'un agent devrait signaler à la rue qu'il va entrer ou sortir de celle-ci. Ainsi, on peut stocker le nombre d'agents dans la rue et ne recalculer la vitesse que lorsqu'il le faut.

6.5 Troisième étape : Généralisation de la formule

Jusqu'ici nous utilisions une formule, $v = v_{max} * (1 - \frac{n}{k})$, avec, v_{max} la vitesse maximum sur la rue, n le nombre d'agent sur la rue et k le nombre d'agent maximum sur la rue. Celle-ci serait jusque là inscrite en dur dans l'agent prédéfini symbolisant les rues. L'utilisateur ne pourrait donc pas utiliser la sienne, même si elle était plus pertinente.
Nous voulions donc laisser aux utilisateurs la possibilité de renseigner leurs propres formules dans le code GAML, mais n'ayant pas dejà pu mettre en place la formule à temps, nous n'avons pas du tout étudié cette fonctionnalité.

Chapitre 7

Le plug-in final

La dernière version du plug-in à ce jour permet donc de créer un modélisation de base pour une ville. D'un part des rues sur lesquelles les agents roulent et respectent les sens-uniques. Elles peuvent être générées depuis un fichier SIG et on peut leur définir une vitesse maximum mais celle-ci n'est pas utilisée par les agents. Et d'autre part, des services symbolisant les intersections, les points de départ et les points d'arrivées des agents circulant dans la ville.
Notre principale difficulté fut la complexité de la plateforme GAMA. Beaucoup de temps et d'énergie furent dépensés pour mettre en place un simple graphe orienté et quand bien même celui-ci fonctionnât, nous n'avons pas pu l'adapter pour gérer la vitesse des agents.
Un sentiment de frustration générale a donc été ressenti tout au long de notre travail sur ce projet (surtout au niveau du ratio temps de travail / résultats obtenus) même si nous avons pu compter sur l'aide de nos tuteurs pour nous ré-aiguiller vers d'autres pistes pour résoudre nos problèmes. L'aspect positif de ceci (parce qu'il en faut bien un) aura été la constante remise en question de la conception de nos fonctionnalités impliquée.