**Projet multi-agents**

*Dial-A-Ride Multi-Agent Based Simulation*

# Ludovic CLARISSOU & Robin VANET

## Choix d’implémentations

Pour ce projet, nous avons décidé de réaliser 3 types d’agents :

* Des Sources,
* Des Taxis,
* Des Clients

L’environnement sera une grille bidirectionnelle de 50\*50 dans lequel évolueront les agents. Chaque agent aura ses propres croyances et ses propres actions pour accomplir ses buts.

Voici le détail de chaque agent :

|  |  |
| --- | --- |
| Taxi | |
| Buts | * Transporter des clients vers la destination qu’ils souhaitent |
| **Actions** | * S’il n’y a pas de clients, il se déplace aléatoirement dans l’espace. * Si un client lance un appel aux taxis : il regarde s’il peut le prendre (siège bébé pour un client qui a un bébé) et s’il est le plus près. Si ces conditions sont réunies : il dit aux autre qu’il le prend. * Lorsqu’il est vers un client, il le « prend » dans son taxi et l’emmène vers la destination qu’il souhaite |
| **Croyances** | * Sa position dans l’espace * S’il est occupé ou non * S’il dispose d’un siège bébé. |
|  |  |

|  |  |
| --- | --- |
| Sources | |
| Buts | * Émettre des clients aléatoirement dans le temps |
| **Actions** | * Lance un compteur lorsque le client est émis et le décompte à chaque tick. * Si le compteur est à 0, faire apparaitre un nouveau client. |
| **Croyances** | * Sa position dans l’espace * Son identifiant * Le nombre de clients provenant de la source qui sont satisfaits * Le nombre de clients provenant de la source qui sont insatisfaits |
|  |  |

|  |  |  |
| --- | --- | --- |
| Clients | | |
| Buts | * Monter dans un taxi. | |
| **Actions** | * Appeler un taxi en criant sa position (et dire s’il voyage avec un bébé) à l’ensemble des taxis. * Disparaitre s’il est insatisfait (temps d’attente trop long). | |
| **Croyances** | * Sa position dans l’espace. * Sa patience. * S’il voyage avec un bébé ou non. | |
|  | |  |

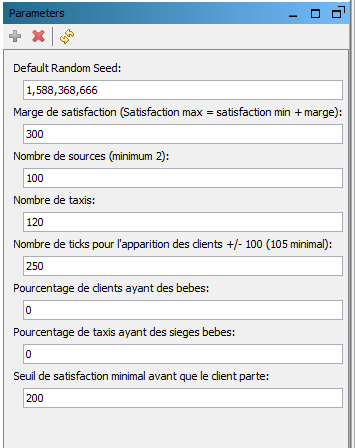
Nous avons donc choisi de créer 3 différents agents pour simuler ce problème. Cette simulation est totalement décentralisée : Il n’y a pas de centrale qui gère les appels. La communication entre les agents peut se faire :

* Entre un client et les taxis (Demande de taxi).
* Entre les taxis entre eux (étape de négociation de qui va aller chercher le client).

## Fonctionnalités

Comme ce projet attrait à la simulation, nous avons essayé de rendre certains paramètres (temps d’attentes des clients, nombre de clients ayant un siège bébé, nombre de taxis ayant un bébé, …) le plus configurable par l’utilisateur tout en gardant une notion d’aléatoire.

Voici les paramètres configurables lors du lancement de l’application (**Figure 1**) :



Sur la **Figure 1**, nous avons, dans l’ordre :

* **Marge de satisfaction** : La satisfaction d’un client est calculé entre une borne minimale et d’une marge de satisfaction. Le programme calcule alors la satisfaction du client comme ceci : b
* **Nombre de sources :** Le nombre de source qui apparaitra (avec des coordonnées aléatoires) dans l’environnement de simulation. Comme un client va d’une source à une autre, il faut au moins 2 sources pour lancer la simulation.
* **Nombre de taxis :** Le nombre de véhicules qui se déplaceront dans l’environnement. Ils seront, au lancement de l’application, placés aléatoirement dans l’environnement.
* **Nombre de ticks pour l’apparition des clients :** Les sources vont émettre des clients aléatoirement dans le temps. Cependant, on peut configurer une base fixe pour l’apparition des clients avec un nombre aléatoire de plus ou moins 100. Par exemple, si on laisse le paramètre par défaut 250. Les clients peuvent au maximum apparaitre après 150 ou 350 ticks après la création du client précédent.
* **Pourcentage de clients ayant des bébés :** On peut rentrer le pourcentage de clients ayant un bébé. Un taxi qui ne dispose pas de sièges bébés ne pourra pas prendre un client qui a un bébé.
* **Pourcentage de taxis qui ont un siège bébé dans leur voiture :** On choisit le pourcentage des taxis qui ont un siège bébé dans leur voiture.
* **Seuil de satisfaction minimal avant que le client parte :** On choisit un seuil minimal au-delà duquel le client insatisfait par. On rappelle que le seuil est calculé avec la formule : b

## Programmation

Notre projet utilise 7 classes différentes : Agent.java, AgentStyle2D.java, ContextCreator.java, Coordonnees.java, Customer.java, Source.java et Taxi.java.

La classe Coordonnees sert à représenter coordonnées de chaque agent sur l’espace (Taxis, Clients et Sources), ainsi que leur destination, leur position de départ… Elle ne comporte que deux champs, x et y qui représentent les deux dimensions de l’espace dans lequel les agents évoluent. Elle contient aussi des getters et des setters, une méthode toString() et une méthode permettant de calculer la distance entre deux instances de Coordonnees.

La classe ContextCreator permet de créer les espaces ou la simulation va se passer, de récupérer les paramètres que l’utilisateurs vient de récupérer, puis à créer et placer tous les agents au lancement de la simulation en fonction des paramètres qu’elle vient de récupérer. Tout cela s’effectue dans sa fonction build.

La classe Agent est la classe mère de tous nos types d’agents. Elle ne contient que les coordonnées de l’agent et les espaces (espace continu et espace grille) ou les agents se déplacent.

La classe Source est dérivée de la classe Agent. Elle possède en plus les variables pourcentageBaby pour savoir quel pourcentage de ses clients doivent posséder un siège bébé, l’ID de la source elle-même, des variables permettant de savoir l’ID du prochain client créé. La fréquence à laquelle ses clients doivent apparaître, les variables représentant la satisfaction qu’auront ses clients, et deux variables d’entiers représentant le nombre de clients satisfaits et non satisfaits par source. La seule méthode présente (en plus des getters, setters et constructeur) est la fonction compute() qui s’active à chaque tick.

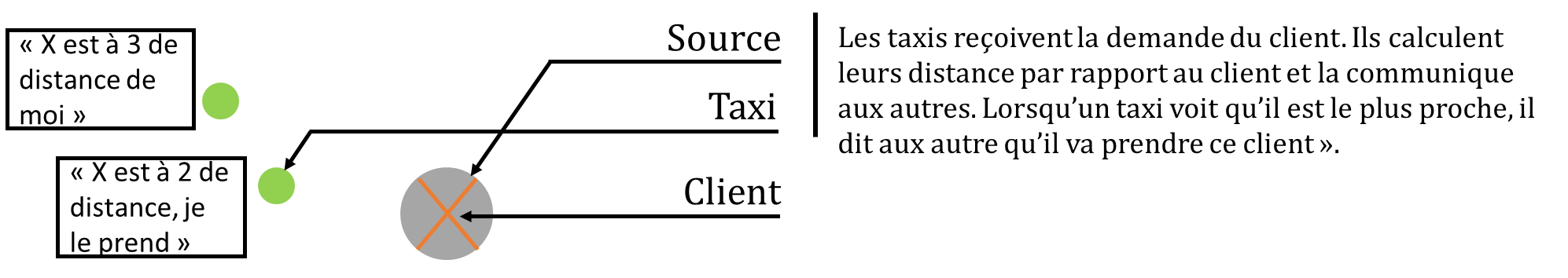
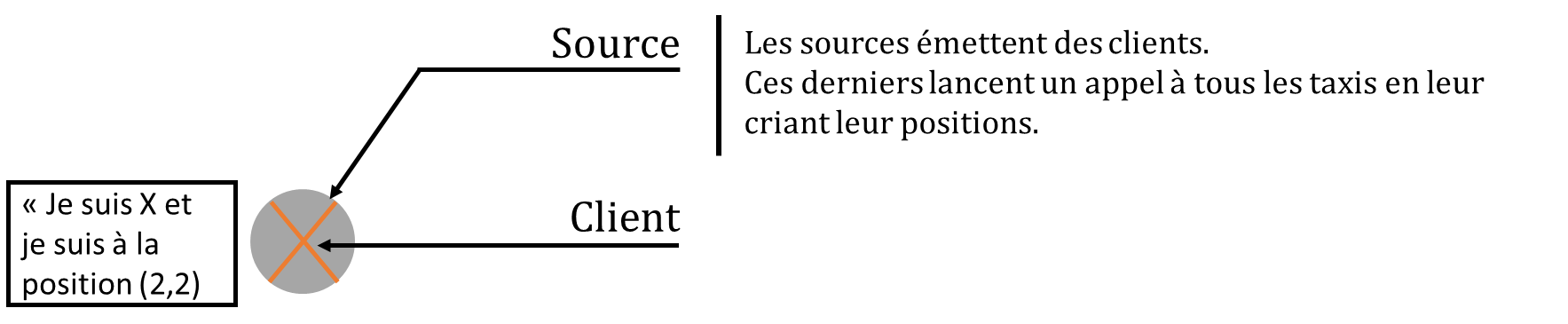
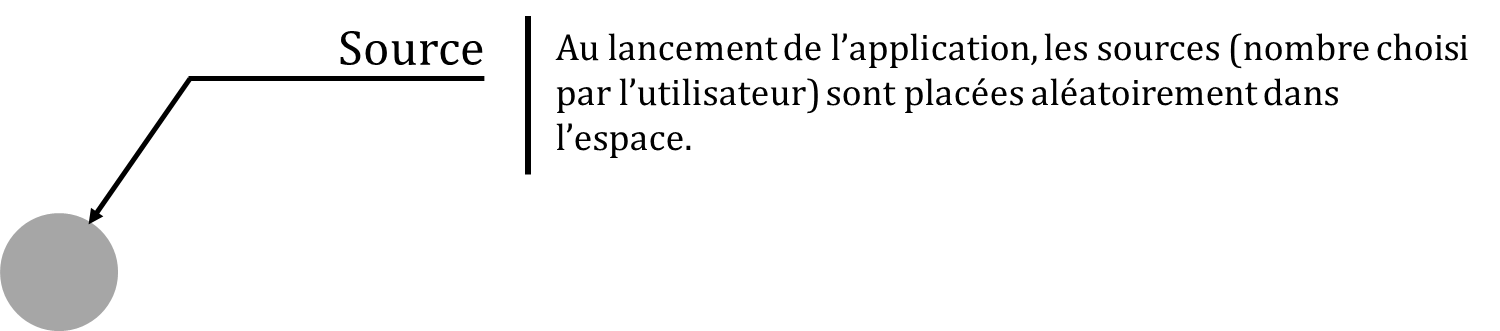
La classe Taxi est elle aussi dérivée d’Agent. Elle en possède en plus un booléen pour savoir si le taxi est libre ou occupé, un booléen pour signifier si le taxi possède un siège bébé, un String pour stocker les messages qu’il a préparé ainsi qu’un booléen permettant d’envoyer ce message. Les parties les plus importantes de cette classe sont les listes clientPossibles, clientSuivi, et minDistReceived, ces listes servent respectivement à : retenir tous les clients dont le taxi à entendu parler, retenir les clients que le taxi considère encore intéressants, et enfin la distance minimale reçue entre le client et les autres taxis. Ces listes vont permettre aux taxis de se mettre d’accord et de se répartir entre les différents clients. Dans leur fonction, les taxis ont la fonction compute, la fonction permettant de choisir le client le plus intéressant vers lequel se diriger, et plusieurs watchers : deux sur les clients (un pour écouter leur requête, un pour les supprimer s’ils décident de partir), et un sur les taxis. Les taxis possèdent aussi une fonction permettant d’effacer leur mémoire pour ne pas trop garder des informations trop vieilles (clients disparus…) Enfin, les deux dernières fonctions sont les fonctions permettant de déplacer le taxi dans l’espace une fois la destination choisie (un client, la destination d’un client, ou un point aléatoire).

La classe Customer est elle aussi dérivée d’Agent. Elle représente les clients dans la simulation. Chaque agent possède une patience, des coordonnées d’origine, une source d’origine et des coordonnées de destination. Elle possède aussi deux booléens permettant d’envoyer deux types de messages : une demande de prise en charge, et une demande de suppression des listes. Le client possède aussi une fonction permettant de dire à sa source d’origine s’il a été satisfait ou non.

La classe AgentStyle2D permet de changer les couleurs des taxis et des clients en fonction de s’ils sont occupés ou libre, s’ils ont des sièges bébés ou non, s’ils ont des bébés avec eux…

## Stratégie

Notre stratégie est simple :



Lorsqu’un client apparait, il « crie » sa position aux véhicules (taxis). Les taxis vont chacun calculer la distance du client par rapport à eux même et s’échangent chacun leur distance d’eux même par rapport au client. Lorsqu’un taxi annonce avoir la plus petite distance, il bloque le client et se dirige vers lui. Les autres taxis ne voient alors plus le client comme disponible.

Lorsqu’une source crée un client, ce dernier lance un appel à tous les taxis de type

« Je suis en (2,2), venez me chercher ».

Les taxis vont alors calculer indépendamment les uns des autres leur distance par rapport au client. Et s’échangent leurs distances les uns des autres. Lorsqu’un taxi voit qu’il est plus près des autres, il dit aux autres « je suis le plus proche, j’y vais ». A ce moment-là, les autres taxis ne considèrent plus ce client comme client potentiel.