Intersection intelligente

Luc Heydel – Guillaume Muller – Aurélien Saunier – Victor Sonza

# Problématique

Le but du projet est de développer une simulation d’intersection intelligente, dictant le passage de voitures à un carrefour. Le développement de l’application a été effectué en SARL, en utilisant JAVA AWT pour l’interface graphique.

Nous utilisons la bibliothèque mathématique Arakhne pour définir la trajectoire des voitures lors de leur passage dans l’intersection.

# Architecture du projet

Ci-dessous, le diagramme « UML » de l’architecture de notre projet :

Une image contenant texte, carte

Description générée automatiquement

## Le fichier Car.sarl

**Car** contient l’agent qui définit le comportement d’une voiture au sein d’une intersection. Il est représenté par son **CarBody**, qui est la représentation physique de l’agent dans la simulation. Le CarBody fait partie d’une liste (waitingList or leavingList) d’une **RoadSection** d’une intersection et est également contenu dans une **Map** dans l’environnement.

## Le fichier Environnement.sarl

## Le fichier EventsAndSharedData.sarl

EventsAndShareData permet de définir les events utilisés dans le programme, ainsi que les classes de type « Body » permettant d’instancier des objets contenant les données des corps physiques des agents. Ces objets sont liés aux agents via leur UUID.

## Le fichier Intersection.sarl

## Le fichier IntersectionCenter.sarl

## Le fichier IntersectionSimulationLauncher.sarl

Ce fichier est l’agent que nous utilisons pour lancer la simulation. Il contient notamment la fonction launchEnvironment, qui lance l’environnement (lancement du noyau, lancement du contexte par défaut, …)

## Le fichier Population.sarl

Une population permet de caractériser les voitures auxquelles elle est affectée. Cela se traduit concrètement par une vitesse maximale et une accélération maximale que toutes les voitures qui y sont liées auront en commun.

## Le fichier RoadSection.sarl

## Le fichier Settings.sarl

Le fichier settings permet de définir des constantes globales pour le programme, telles que la taille de l’environnement, le nombre d’intersection, le temps de pause entre chaque instant de la simulation, etc…

# Affichage

L’affichage est géré par la classe EnvironmentGui. Celle-ci est inspirée de la classe créée par M. Nicolas Gaud dans son exemple de simulation de Boids, et adaptée à nos besoins. Cette classe peint les bâtiments, les lignes des routes, ainsi que les voitures (représentées sous formes de triangles noirs). Nous disposons d’un timer en haut à gauche de l’interface qui permet de voir le temps moyen passé par une voiture dans l’intersection.

La fonction paint de la GUI est appelée par l’environnement une fois que tous les agents présents dans la simulation ont envoyé leur demande d’action.

La création de la GUI est entièrement codée en brut, les positions sont définies de manières absolues. Il pourrait être intéressant de modifier ce point afin de pouvoir créer un enchaînement d’intersection.

# Evènements

# Points à améliorer

## La création de la GUI

La création de la GUI est codée en dur dans l’application, et est codée en utilisant AWT. Il serait intéressant de rendre la création de l’interface de manière dynamique, pour permettre la création d’enchainement d’intersections.

Qui plus est, il serait également intéressant de changer de technologie pour la programmation de l’interface, pourquoi pas en utilisant JavaFX, qui est beaucoup plus récent.

## La gestion de la multi intersection