Table des matières

1 Introduction

Dans le cadre de l'UV IA54 (Systèmes multi agents et résolution distribuée de problèmes), nous avons dû réaliser un simulateur de traffic, à l'aide des bibliothèques Janus et Jaak. Différents objectifs étaient proposés au cours de la réalisation de ce projet :

- réaliser le simulateur de traffic en langage Java,
- étendre la bibliothèque Jaak afin de lui permettre d'être utilisée avec certains langages de scripts,
- implanter dans le simulateur des agents développés dans un langage de script choisi.

Ce rapport présente dans une première partie les outils utilisés au cours du développement de ce projet. Dans une seconde partie, nous expliquons la manière dont nous avons réalisé le simulateur de traffic. Nous proposons dans une troisième partie une extension de la bibliothèque Jaak afin d'y intégrer les langages de script. Enfin nous concluons sur les résultats obtenus ainsi que les améliorations possibles à apporter au projet.

2
Outils utilisés

Notre simulateur de traffic doit être implémenté à l'aide de deux bibliothèques : Janus et Jaak.

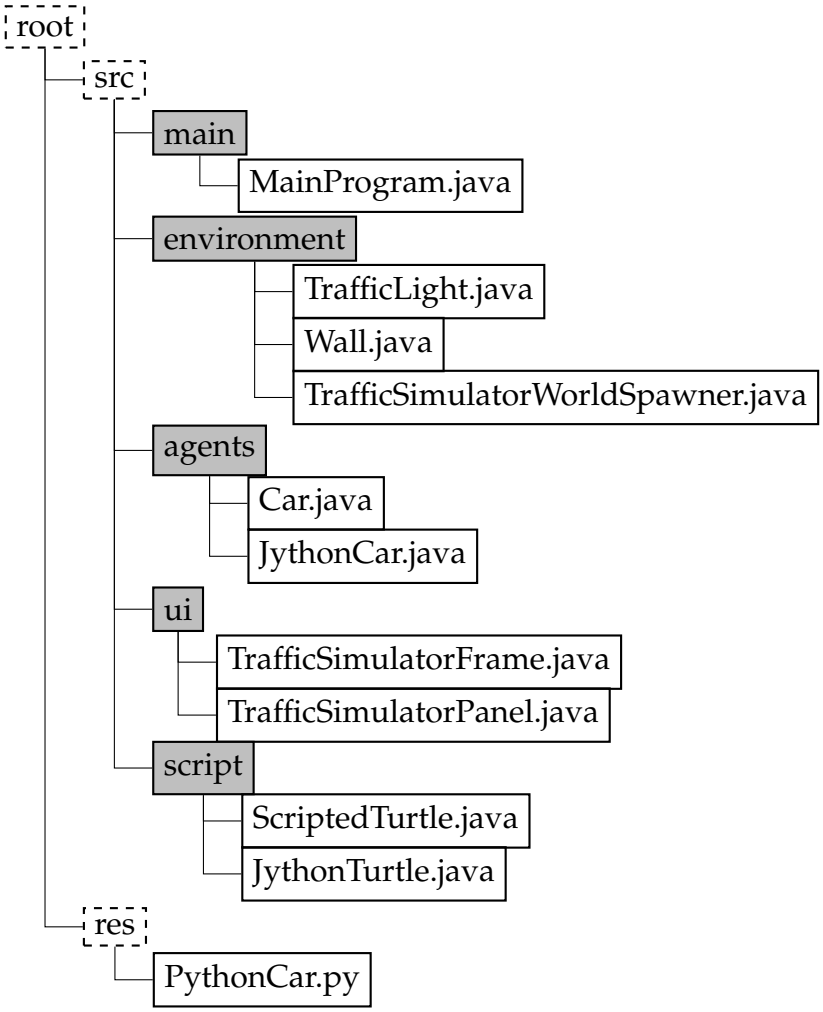
Janus est une plateforme multi-agent open source implémentée en langage Java (version 1.6). Elle fournie des fonctionnalités permettant de développer, lancer, afficher et suivre des applications basées multi-agent. Janus peut être utilisée comme plateforme agent-orientée, plateforme organisationnelle, ou plateforme holonique. Elle est construite selon les normes du modèle organisationnel CRIO. Enfin Janus est libre pour une utilisation non commerciale et distribuée sous les termes de la licence GPLv3.

Jaak est un outil qui complète la bibliothèque Janus. Il fournit un environnement 2D discret ainsi que des interactions simplifiées basées sur les primitives du langages Logo. Cet outil est particulièrement adapté à la modélisation de systèmes complexes se développant au cours du temps.

3
Simulateur de traffic

3.1 Architeture générale

L'architecture de notre application de simulation de traffic est divisée en 5 packages :



Le lanceur *MainProgram.java* est à l'origine de la création de l'interface ainsi que de l'environnement.

L'environnement est composé de différents objets tels que les murs et les feux

tricolores. Il contient aussi le spawner qui sera responsable de la création des agents dans l'application.

Dans notre application, les agents sont des voitures. Il existe une version de l'agent écrite en Java (à des fins de test de l'application), ainsi qu'une version écrite dans le langage de script de notre choix sur laquelle nous reviendrons dans la suite de ce rapport.

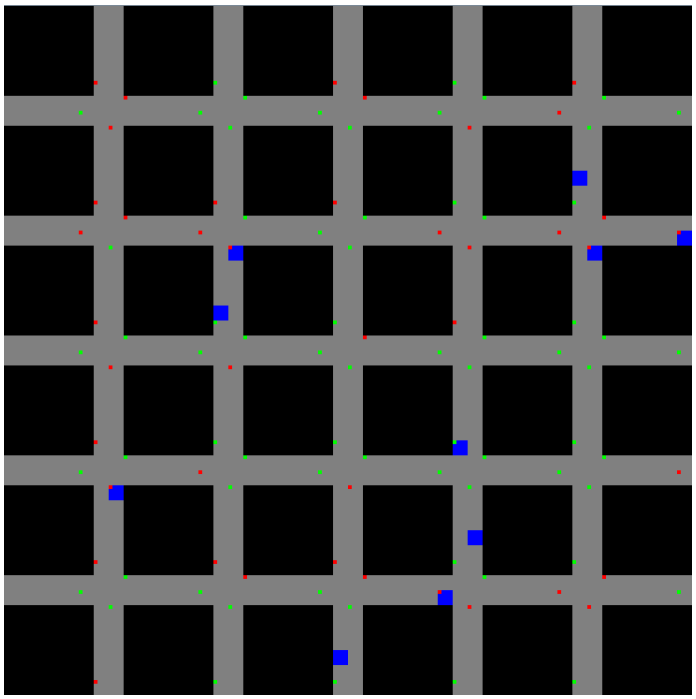
L'interface est uniquement composée d'une fenêtre ainsi que d'un panel, basés sur la bibliothèque Swing.

Un dernier package, *script*, contient les classes ayant permis d'étendre Jaak à l'utilisation de langages de scripts.

Enfin les scripts sont stockés dans un dossier de ressources.

3.2 Environnement

Notre environnement est un espace discret 2D représenté par une grille qui dans chaque casela composant, peut contenir différents objets et/ou agents.



Parmi ces objets, notre environnement possède des murs, qui sont des obstacles : cela signifie qu'aucun autre élément ne peut se trouver dans une case où se trouve déjà un mur. Les routes apparaissent donc aux endroits où il n'y a pas de murs de placés.

Un second objet compose notre environnement : il s'agit des feux de circulation. Cette catégorie d'objet ne correspond pas un obstacle. Il permet donc aux agents de se déplacer sur une case comportant un feu de circulation. Néanmoins ces feux agissent sur le dépacement des agents : lorsque le feu est rouge, les agents ne peuvent pas progresser au delà de celui-ci. Lorsque le feu est vert, ils le peuvent.

La création de l'environnement se déroule dans la fonction *createEnvironment* de la classe *MainProgram*. Pour chaque case de la grille on lui applique le ou les objets que l'on souhaite aux coordonnées correspondantes.

La création des agents est géré au sein de la classe *TrafficSimulatorWorldSpawner*. Cette classe possède deux arguments importants :

- le nombre d'agents dont la simulation disposera,
- l'environnement créé précedemment.

Ainsi tant que le nombre d'agent dans la simulation n'est pas atteint, le spawner créé un nouvel agent qu'il positionne sur une case de la grille ne contenant pas de murs.

3.3 Agent

3.4 Interface

4

Extension aux langages de scripts

4.1 Extension de Jaak

4.2 Agent Python

5
Conclusion