UNIVERSITÉ DE
SHERBROOKE

# Traffic Simulator

Travail présenté à monsieur Froduald Kabanza & Gabriel Girard

Dans le cadre des cours
**IFT 615 – Intelligence Artificielle &**
**IFT 630 – Processus concurrent et parallélisme**


Par mm.

| | |
|---|---|
| Alex ST-LAURENT | 05 640 167 |
| Charles VILLEMURE | 05 645 509 |
| Jean-François ROUSSEAU ROSS | 05 648 739 |
| Pierre-Denis NOËL | 00 912 329 |

Département d'informatique
Université de Sherbrooke


Déposé le 11 avril 2008

## Description du projet :

Dans le cadre de ce projet, nous avons développé une application permettant de faire une simulation routière. Lors du développement de ce projet, nous avons rencontré plusieurs problématiques :

- Affichage graphique

  Afin d'être en mesure d'afficher une carte routière ainsi que les voitures en mouvement, nous avons utilisé la librairie *Advanced Window Toolkit (AWT)* de Java. Pour ne pas surcharger la tâche d'affichage, nous avons utilisé des « BufferedImage » nous permettant de ne pas redessiner la carte routière à chaque affichage du mouvement des voitures.

- Gestion des intersections

  Afin d'avoir un résultat réaliste, nous avons fait en sorte que lorsqu'une voiture arrive à une intersection, elle doit s'assurer que la route qu'elle tente d'accéder soit libre, c'est-à-dire qu'elle n'est pas fermée ni bloquée par un autre véhicule.

- Gestion des collisions

  Afin d'éviter les collisions, avant de faire un mouvement, chaque voiture doit s'assurer qu'elle laisse un espace suffisant, égalant la moitié de sa propre longueur, entre elle et le véhicule qui la précède. Par conséquent, si le véhicule qui la précède est arrêté, la voiture cessera d'avancer avant qu'il y ait une collision. Aussi, si le véhicule qui la précède avançait plus lentement qu'elle, la voiture ralentirait elle aussi.

- Gestion de la vitesse

  Afin d'avoir une simulation plus réaliste, chaque segment de route possède sa propre limite de vitesse. Grâce à cela, nous avons pu simuler des conducteurs qui désirent aller plus vite à leur destination ainsi que des conducteurs qui désirent s'y rendre avec la plus courte distance possible. Par contre, contrairement à ce qui se passe présentement sur nos routes, toutes nos voitures respectent les limites de vitesse.

- Création et chargement de carte routière

  Afin d'éviter de n'avoir qu'une seule carte routière fixe et afin de simplifier la création et le chargement de nouvelle carte routière, nous avons créé un format

standard et simple utilisant un fichier texte. Ainsi, il est facile de créer de nouvelle carte routière afin de tester différentes configurations.

- Gestion de la direction de la circulation

  Toujours afin d'atteindre un certain niveau de réalisme, il est possible de créer des segments de route à sens unique. Ceux-ci, tout comme les segments de route bidirectionnels, seront pris en compte lors du choix du parcours.

- Parallélisme
  - Voir la section *Processus concurrent et parallélisme* plus bas.
- Accès concurrentiel à certaines données en mémoire
  - Voir la section *Processus concurrent et parallélisme* plus bas.
- Algorithme de décision du parcours
  - Voir la section *Intelligence artificielle* plus bas.

## Intelligence artificielle :

Afin que les véhicules sélectionnent un parcours optimal, nous avons implémenté l'algorithme *AStar (A*)*. De plus étant donné que nous voulions que l'algorithme soit flexible pouvant permettre de trouver un chemin plus rapide ou plus court, nous avons été contraints de mettre une euristique de 0 pour tous les nœuds, n'ayant pas réussi à trouver une euristique qui ne surestimait pas la valeur des nœuds pour trouver le chemin plus rapide étant donné qu'il est impossible de prévoir la limite de vitesse sur le chemin à venir jusqu'à la destination. Il est possible que nous ayons pu trouver une formule complexe permettant de prévoir ces vitesses, mais le temps et les connaissances nous manquaient.

## Processus concurrent et parallélisme :

Le projet est constitué à la base d'un processus maitre et de 3 threads principaux permettant de gérer l'affichage, de gérer la fermeture des routes et un autre pour la gestion du déplacement des véhicules. Le thread d'affichage est rafraichi de manière constante à toutes les 20 millisecondes pour le déplacement des véhicules et à toutes les 500 millisecondes pour l'affichage de la console qui permet d'afficher le déroulement de la simulation. Le thread de gestion des routes lui, ferme un nombre de routes fixé à son instanciation toutes les 4 secondes et les ouvre à nouveau après le délai de 4 secondes où il en ferme d'autres. La carte routière est donc redessinée toutes les 4 secondes, le reste du temps la « BufferedImage » est utilisée pour l'affichage. Le thread principal de gestion du mouvement des véhicules est

initialisé au départ de l'application avec une liste de véhicules pour chacun desquels il démarre un nouveau thread s'occupant de la gestion de ses déplacements. Ce dernier thread recalcule à toutes les 100 millisecondes l'avancement du véhicule en s'assurant qu'il n'y ait pas de collisions. Étant donné la quantité de thread présents dans l'application, plusieurs méthodes et accès à des variables ont dû être synchronisés afin de permettre le bon déroulement de notre simulateur.

## Déroulement du projet :

Lors du développement de ce projet, nous avons travaillé en utilisant la méthode « Extreme Programming » en deux équipes de deux. Ainsi, nous avons pu développer efficacement et rapidement. De plus, nous avons utilisé *Java 1.6* ainsi qu'*Eclipse* qui nous permettent aussi de développer rapidement. Nous avons donc réussi à réaliser ce projet en 4 jours à temps complet, ce qui représente 32 heures par personne.

## Diffusion publique :

La distribution est libre tant que le crédit est donné aux programmeurs de l'application.

## Compilation et exécution de l'application :

Pour compiler le projet, il suffit d'utiliser la commande javac ou compiler avec Eclipse.
Pour lancer l'application sans passer par le .jar il faut se placer dans le répertoire parent de Project.class et exécuter la commande suivante :

```
java project.Project <option>
```

Pour lancer l'application en passant par le .jar, il faut exécuter la commande suivante :

```
java -jar project.jar <option>
```

Ou option est une valeur entre 0 et 4. Ces dernières changent le nombre de voiture en action ainsi que la map utilisée.
À noter que lorsque l'application démarre, la simulation elle-même attend qu'une personne appuie sur le bouton Pause.