Projet - Informatique

Document de synthèse

10 mai 2019

Arthur Dujardin

Elie-Alban Lescout

Samuel Mermet

1. Travail en équipe

1.1. Organisation

Jusqu'au 11 avril, nous avons consacré notre temps sur le projet à l'élaboration des diagrammes de classe et d'activités, en identifiant en particulier trois sous-parties principales à notre application : MovingParts et StructuresParts (correspondant respectivement aux objets mobiles et aux éléments structuraux), ainsi que SimulationState (pour gérer les stocker les étapes de la simulation).

Dans un premier temps dédié au développement, nous avons distribué les tâches de développements en fonction de ces trois parties. Cette distribution initiale a été revue en fonction des avancées de chacun des membres de l'équipe, jusqu'à compléter l'ensemble des classes utiles au développement de l'application. Notre diagramme de classe initial a logiquement évolué avec l'avancée dans le développement.

Notre objectif était de réaliser dans un premier temps une application simple et fonctionnelle, qui serait augmentée dans un second temps de spécifications plus complexes. Toutefois nous avons rencontré beaucoup de difficultés dans la manipulation de java2D, au point de décaler notre premier affichage fonctionnel à la semaine du 6 au 10 mai.

A partir de là, les principales difficultés ont consisté à gérer les fusions entre branches et les incohérences générées. En effet, malgré nos échanges réguliers, nous avons identifié tardivement des différences dans les conventions de codage.

1.2. Réalisation

Carmageddon Simulator est une simulation d'un trafic routier, autour d'un carrefour central. Une fois l'application lancée, l'utilisateur choisi la taille de la simulation. Une nouvelle fenêtre s'ouvre, comportant la simulation désirée. Un bouton Start permet de démarrer la simulation sans interruption. Un nouvel appui sur ce bouton permet de stopper la simulation, et ainsi de suite. Deux autres boutons permettent d'afficher un état de la simulation, avant ou après. Ensuite des boutons de flux permettent de générer le flux des piétons et voitures entrantes en temps réel. Pendant ce temps, un outil de statistiques a été mis en place afin de connaitre le temps moyen que met un piéton ou une voiture pour traverser le carrefour ou son temps moyen d'arrêt.

Les piétons se dirigent dans une direction, et dévie légèrement pour un souci esthétique. Ils traversent au carrefour lorsque le feu est vert, et on un parcours vraisemblable : les piétons ne tournent pas en rond autour du carrefour, ni reviennent de là où ils sont partis.

Les voitures circulent sur des voies, au sein d'une route. Elles roulent à une certaine vitesse et accélère selon le profil du conducteur. Les voitures possèdent une vision qui leur permettent de détecter des obstacles : des piétons, d'autres voitures, ou encore un feu tricolore. Ainsi, la voiture est en mesure de s'arrêter à un feu orange ou rouge, et de laisser passer des piétons.

1.3. Travail en cours

Malgré nos efforts, nous n'avons pas pu porter Carmageddon Simulator aussi loin que nous le souhaitions dans le temps imparti. Si nous nous satisfaisons de la conception graphique rendue, nous aurions apprécié présenter quelques fonctionnalités supplémentaires. A titre d'exemple, nous n'avons pas intégré le modèle d'accélération et de ralentissement des voitures envisagé.

En lutte avec des voitures et piétons récalcitrants, nous avons trop tardé à produire le rapport statistique si utile aux urbanistes et gestionnaires de voirie. En effet, si notre simulateur de carrefour routier génère bien les données de congestions routières et piétonnes générées par les durées des feux, nous aurions souhaité faciliter la lecture de ces statistiques par des graphes et indicateurs aisément comparables.

A noter que dans sa version actuelle, l'application stocke les différentes étapes de la simulation. Si cette gestion intégrée répond largement aux besoins de l'utilisateur en matière d'historique de navigation, elle atteint rapidement des limites pour générer des statistiques sur des simulations plus longues. Pour augmenter la durée de vie des simulations, nous envisagions de ne conserver qu'un nombre limité de simulations (de l'ordre de la centaine, selon l'expérience utilisateur) et de stocker dans un fichier les statistiques produites. Bien sûr, une base de données est également envisageable, mais non nécessaire à notre avis.

1.4. Apprentissage

Ce projet nous a permis de mettre en pratique différents enseignements suivis au cours de cette année, notamment ceux liés à l'informatique tels que l'algorithmique pour l'optimisation et, dans une moindre mesure, Python et la programmation scientifique en tant que premiers projets de développement.

Surtout, l'objet principal de ce projet étant de s'initier à la programmation orientée objet, nous nous sommes appuyés sur nos enseignements en analyse informatique pour concevoir les diagrammes d'activités, qui ont facilité la rédaction du code informatique liés aux piétons et aux voitures, et le diagramme de classe, préalable nécessaire à la gestation de notre code informatique.

Ainsi, nous avons implémenté ce diagramme en Java afin de produire un code modulable et évolutif susceptible d'intégrer plus de fonctionnalités que celles que nous avons eu le temps de développer dans ce projet. Ce dernier fait d'ores-et-déjà appel aux trois principes de la programmation orientée objet que nous avons vu en cours : l'encapsulation, l'héritage et le polymorphisme. Il nous a notamment permis d'expérimenter sous Java différents types d'objets, la gestion de la documentation, ainsi qu'un très efficace traitement anticapillaire avec l'utilisation de Java2D !

Enfin, ce projet complexe a constitué une expérience collective très enrichissante. Nous avons pu confronter nos idées, de la conception du diagramme de classes aux algorithmes des méthodes liées à ces dernières. Nous avons pu mesurer la nécessité d'échanger régulièrement et de communiquer sur les choix opérés en amont et en cours de développement.

2. Technique

2.1. Organisation des développements

Le développement s'est organisé autour de différents packages.

- **mobile** régie les éléments amenés à se déplacer sur la simulation, en l'occurence les voitures (Car) et les piétons (Pedestrian)
- **immobile** regroupe tous les éléments structurant la simulation. Une distinction est faite entre les différents feux de circulation (regroupés et coordonnés dans **immobile.lights**), susceptibles d'être modifiés en cours de simulation, et les composantes de la route (**immobile.structure**) où circulent les voitures et piétons.
- **enumeration** regroupe les différentes énumérations appelées.
- **engine** et **model** contiennent le cœur de la simulation et ses paramètres.
- **display** gère l'affichage de la simulation.

2.2. Exécution du programme

L'application Carmageddon Simulator a été compilée sous forme de fichier .jar permettant son exécution automatique sur tout ordinateur doté de Java. Elle a été testée avec les versions 8, 11 et 12 du langage.

A l'ouverture du logiciel, une fenêtre propose à l'utilisateur de configurer la simulation à venir. Ces paramètres structurels permettent de définir la taille de la grille de simulation, le nombre de voies des deux routes composant l'intersection. L'utilisateur peut également faire appel à des valeurs par défaut en cliquant directement sur Start (ou barre d'espace).

Une nouvelle fenêtre s'ouvre alors, composée de plusieurs panneaux dont le principal représente graphiquement la simulation. On peut naviguer dans les différentes étapes de simulation avec les flèches droite (avancer dans le temps) et gauche (revenir en arrière). A droite du panneau principal, les options de gestion du trafic routier et piétons, ainsi que la durée des feux, permettent de modifier ces paramètres et d'en observer visuellement les conséquences, que ce soit sur la fenêtre graphique ou les statistiques affichées.