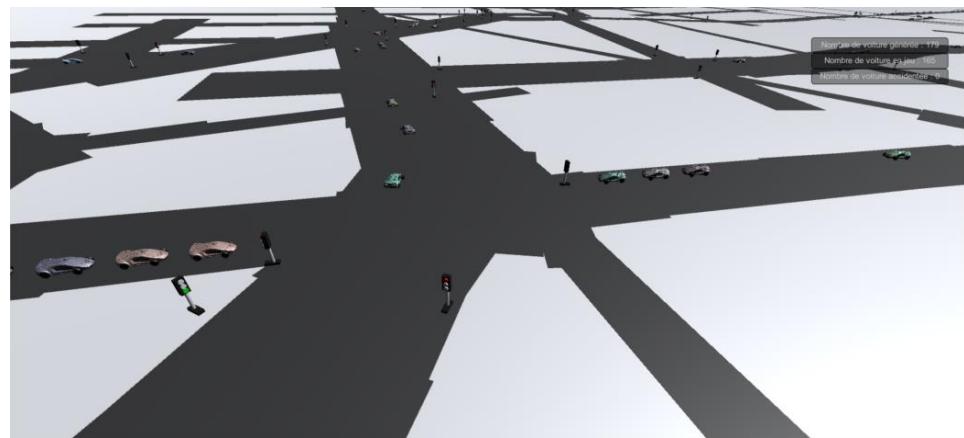


# « Simulation de mobilité urbaine dans Unity 3D »



Thèse de Bachelor présentée par

**Monsieur Nicolas VAN DOOREN**

pour l'obtention du titre Bachelor of Science HES-SO en

**Ingénierie des technologies de l'information avec orientation en  
Logiciels et Systèmes complexes**

**Septembre 2016**

Professeur HES responsable TB

**Paul ALBUQUERQUE**

Mandant

**Olivier DONZE**

**INGÉNIERIE DES TECHNOLOGIES DE L'INFORMATION**  
**ORIENTATION – LOGICIELS ET SYSTEMES COMPLEXES**  
**SIMULATION DE MOBILITÉ URBAINE AVEC UNITY 3D****Descriptif :**

L'utilisation d'outils de modélisation et de simulation dans le processus d'élaboration d'aménagements urbains est une tendance actuelle qui va en se renforçant. En effet, l'augmentation des performances des moteurs de jeu 3D tels que Unity, conjuguée à la mise à disposition en open data de données géographiques du territoire, crée un contexte favorable au développement de telles applications. Des projets d'aménagement liés à la mobilité urbaine pourraient bénéficier d'une simulation 3D pour identifier de potentiels impacts. En outre, une visualisation 3D permettrait aux politiques et au public de mieux appréhender les projets. Plus particulièrement, la circulation automobile autour des futures gares du CEVA (Cornavin - Eaux-Vives - Annemasse) à Genève constitue un cas d'étude.

Dans ce projet de Bachelor, on propose de simuler la circulation dans un quartier de Genève avec le moteur de jeu Unity 3D. Les données géographiques pour réguler les déplacements des véhicules seront importées depuis le SITG (système d'information du territoire genevois). On pourra ainsi prendre en compte les règles de circulation, les signaux lumineux ainsi que les flux automobiles.

**Travail demandé :**

En fonction du temps à disposition, le candidat effectuera les tâches suivantes :

1. Prise en main des technologies (C#, Unity 3D, logiciels pour SIG tels que QGIS)
2. Définition de l'architecture générale de l'application
3. Établissement de cas d'utilisation et description de la chaîne de traitement des données
4. Définition des fonctionnalités de l'application
5. Constitution à partir du SITG d'une base de données avec les informations routières pour un quartier
6. Modélisation d'un carrefour et de l'objet routier pour un quartier
7. Modélisation des véhicules et de leur comportement
8. Importation et construction dans Unity 3D de l'objet routier
9. Simulation et visualisation des déplacements de véhicules dans Unity 3D
10. Mise en place de l'architecture complète de l'application

Candidat :

**M. van Dooren Nicolas**

Filière d'études : ITI

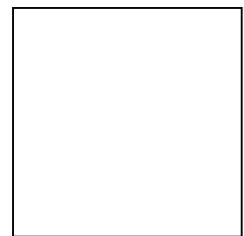
Professeur(s) responsable(s) :

**ALBUQUERQUE Paul**

En collaboration avec :

Prof. Olivier Donzé, MIP, inPACT, hepia  
Travail de bachelor soumis à une convention  
de stage en entreprise : **non**  
Travail de bachelor soumis à un contrat de  
confidentialité : **non**

Timbre de la direction



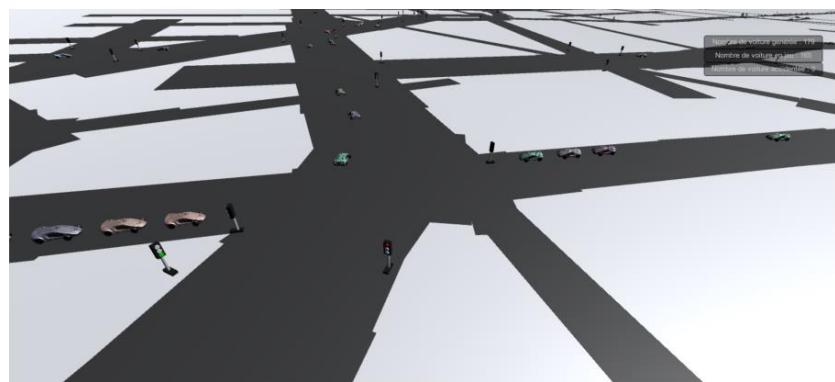
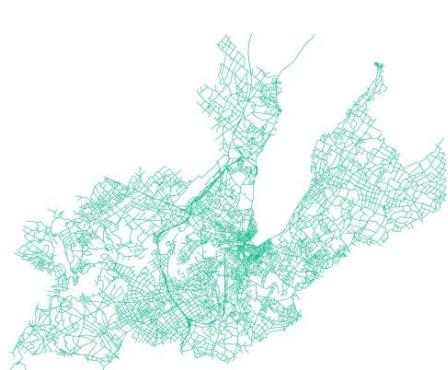
**Résumé :**

La simulation 3D de mobilité urbaine est, dans la gestion d'aménagement urbain, une tendance grandissant dans le monde du génie civile. Dans le monde de l'animation 3D, différents outils devenus très performants peuvent être sollicités à la création de telles simulations. Ceci permettra de visualiser les impacts qu'auront des aménagements sur le trafic routier avant leur réalisation. Cela pourra ainsi permettre aux ingénieurs et au grands public de mieux visualiser et mieux comprendre les conséquences d'une modification de carrefour.

L'objectif de ce projet de Bachelor consiste à réaliser une simulation de trafic urbain à partir de données provenant d'une base de données SIG afin de créer un simulateur basé sur des données réelles de qualité professionnelle.

La *simulation de mobilité urbaine* est créée à l'aide du moteur de jeu Unity 3D. Une analyse est faite sur les données provenant du catalogue du SITG (système d'information du territoire à Genève) afin de mettre en avant les données utilisables, ainsi que les éventuelles données manquantes, nécessaires à la création d'une simulation réelle.

L'application proposée pour simuler est adaptable au format des données provenant de bases de données SIG, afin que l'application soit utilisable par différentes plateformes, pas seulement celle du SITG. Une architecture adéquate a été mise en place afin de réaliser cette adaptation. L'utilisation de ces données permet de créer les composants routiers d'un carrefour. Les automobilistes du simulateur gèrent eux-mêmes les priorités, distance de sécurité, angle mort, etc. Différents profils de conducteurs peuvent ainsi être créés. Ces éléments devraient permettre d'offrir une simulation au plus proche de la réalité.



Données venant d'une base de données SIG et visualisation d'une simulation.

Candidat :

**M. VAN DOOREN NICOLAS**

Filière d'études : ITI

Professeur(s) responsable(s) :

**Albuquerque Paul**

En collaboration avec :

Prof. Olivier Donzé, MIP, inPACT, hepia  
Travail de bachelor soumis à une convention

de stage en entreprise : NON

Travail de bachelor soumis à un contrat de  
confidentialité : non

---

## Avant propos

Ce rapport décrit mon travail de Bachelor, réalisé sur une période de 12 semaines au sein de l'HEPIA. Le sujet de ce travail de Bachelor, a été proposé par M. OLIVIER DONZÉ professeur à l'HEPIA. L'objectif de ce projet consiste à réaliser une simulation d'un trafic urbain à partir de donnée provenant d'une base de donnée SIG afin de créer un simulateur basé sur des données réelles.

## Informations

**van Dooren Nicolas :** vandooren.nicolas@gmail.com

Etudiant ITI travaillant sur le projet *Simulation de mobilité urbaine 3D*.

**Albuquerque Paul :** paul.albuquerque@hesge.ch

Professeur à l'HEPIA dans la filière ITI.

**Donzé Olivier :** olivier.donze@hesge.ch

Professeur Olivier Donzé ayant proposé le sujet.

**António Domingos Ana Sofia :** ana-sofia.antonio-domingos@hesge.ch

Assistant à l'HEPIA dans la filière ITI ayant suivis ce travail

**Polla Michael :** michael.poll@hesge.ch

Assistant à l'HEPIA dans la filière ITI ayant suivis ce travail

## Suivi

Un rendez-vous hebdomadaire a été fixé afin d'avoir un suivi et un compte rendu de l'avancement du projet chaque semaine. Ce suivi a été réalisé par Prof. Alburquerque, M. Polla et Mme António Domingos qui ont apporté leurs avis et conseils tout au long du projet.

## Dépôt du travail

Le travail a été mis à disposition sur GitHub.

<https://github.com/nicolasvandooren/SimulationUrbaine3D>

Le contenu des dossiers :

- C# - L'application du traitement de données.
- Data - Données SITG utilisées, ainsi que la base de données mise en place.
- Unity - L'application simulateur réalisée avec l'outil Unity.
- Doc - Le mémoire du travail de Bachelor.

## Remerciements

Je tiens à remercier toutes les personnes qui m'ont aidé à la réalisation de ce travail de Bachelor. Plus particulièrement :

M. PAUL ALBUQUERQUE, MME. ANA SOFIA ANTÓNIO DOMINGOS et M. MICHAËL POLLÀ, professeur et assistants à l'HEPIA de la filière ITI, pour m'avoir conseillé et aidé tout au long de la réalisation du projet.

---

M. OLIVIER DONZÉ, professeur à la filière Architecture du paysage, pour m'avoir aidé à utiliser correctement les données provenant d'une base de donnée SIG.

M. ALAIN DUBOIS, professeur à la filière Architecture du paysage, pour m'avoir fournis diverses données manquantes et donné son avis sur la meilleure utilisation de données SIG.

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>10</b>
1.1	Contexte . . . . .	10
1.2	Le projet "Simulation de mobilité urbaine 3D" . . . . .	10
1.3	Architecture globale . . . . .	11
<b>2</b>	<b>Technologies et outils</b>	<b>13</b>
2.1	SITG - Système d'information du territoire à Genève . . . . .	13
2.2	QGIS . . . . .	16
2.3	Base de données . . . . .	19
2.4	C# . . . . .	22
2.5	JSON . . . . .	23
2.6	Unity . . . . .	26
<b>3</b>	<b>Implémentation</b>	<b>29</b>
3.1	Organisation et utilisation des données du SITG . . . . .	30
3.2	Lien SITG-UNITY . . . . .	34
3.3	Modélisation 3D . . . . .	37
3.4	Comportement des Objet 3D . . . . .	42
3.5	Visualisation . . . . .	56
<b>4</b>	<b>Tests</b>	<b>60</b>
4.1	Tests unitaires . . . . .	60
4.2	Tests d'intégration . . . . .	61
4.3	Tests utilisateur . . . . .	63
<b>5</b>	<b>Discussions</b>	<b>65</b>
5.1	Problématiques . . . . .	65
5.2	Alternatives . . . . .	68
<b>6</b>	<b>Conclusion</b>	<b>69</b>
6.1	Bilan . . . . .	69
6.2	Améliorations . . . . .	70
6.3	Perspectives . . . . .	71
	<b>Annexes</b>	<b>72</b>
	<b>A Glossaire</b>	<b>72</b>

---

TABLE DES MATIÈRES

---

<b>B Journal de bord</b>	<b>74</b>
<b>C Planning</b>	<b>79</b>
<b>Bibliographie</b>	<b>81</b>

# Table des figures

1.1 Représentation de l'architecture globale du simulateur de mobilité urbaine . . . . .	11
1.2 Représentation de l'architecture globale . . . . .	12
2.1 Flux : A gauche données SITG inexploitables, à droite données fournies par une personne du domaine . . . . .	14
2.2 En vert zone de modération de vitesse (20/30 km/h) . . . . .	15
2.3 Représentation du schéma liant le SITG à QGIS . . . . .	17
2.4 Visualisation de données avec QGIS . . . . .	18
2.5 Visualisation graphe routier avant et après extraction . . . . .	18
2.6 Représentation du schéma liant SITG et la simulation . . . . .	19
2.7 Représentation du schéma liant SITG à une base de données puis à la simulation	20
2.8 Modèle relationnel de la base de données . . . . .	21
2.9 Graphe d'utilisation des différents langages proposé par Unity . . . . .	22
2.10 Visualisation pour l'implémentation d'une Form C# . . . . .	23
2.11 Représentation du schéma liant la base de données à JSON . . . . .	25
2.12 Représentation des zones de collision . . . . .	27
2.13 Marché des moteurs de jeu [8] . . . . .	28
3.1 Répartition des sections dans l'architecture . . . . .	29
3.2 Exemple de la lecture d'une donnée d'un fichier Shape avec la librairie <i>CartoShapefile</i> . . . . .	31
3.3 Représentation de la zone de simulation sur le graphe routier . . . . .	32
3.4 L'application de chargement de données Shape . . . . .	33
3.5 L'application d'édition de la base de données . . . . .	34
3.6 Communication et structure des fichiers JSON . . . . .	35
3.7 Diagramme des objets avec leur fichier JSON correspondant . . . . .	36
3.8 Les modèles utilisés . . . . .	38
3.9 Route simulateur, route utilisée du SITG . . . . .	38
3.10 Voiture zone non <i>trigger("solide")</i> et zone <i>trigger("non-solide")</i> . . . . .	39
3.11 Schéma représentant les étapes des objets à initialiser . . . . .	40
3.12 Visualisation de la hiérarchie de l'objet <b>Graph(Clone)</b> et des informations 4681(arête) associé. La voie en 3D . . . . .	40
3.13 Illustration pour le calcul vectoriel . . . . .	41
3.14 Diagramme d'état "voiture" . . . . .	44
3.15 Distance de sécurité à 10km/h et à 50km/h . . . . .	45
3.16 Zone de détection priorité . . . . .	46
3.17 Priorité de droite . . . . .	46

---

## TABLE DES FIGURES

---

3.18 Exemple calcul de direction entre $\vec{V1}$ et $\vec{V2}$ . . . . .	46
3.19 Situation bloquée dans un carrefour . . . . .	47
3.20 Toutes les voitures se détectent, puis la voiture du bas avance en ayant réduit sa zone de détection . . . . .	47
3.21 Exemple d'accident avec son impact sur la circulation . . . . .	48
3.22 L'angle mort de la voiture . . . . .	48
3.23 Points d'arêtes . . . . .	49
3.24 Parcours d'une voiture . . . . .	50
3.25 Exemple de positionnement de voiture sur une voie . . . . .	51
3.26 Illustration pour le calcul de position d'une voiture . . . . .	51
3.27 Exemple de parcours sans traitement et avec traitement . . . . .	53
3.28 Exemple de présélection gauche . . . . .	54
3.29 Exemples de feux au rouge puis au vert . . . . .	55
3.30 Comportement d'une voiture à un stop . . . . .	56
3.31 Exemple de vue d'ensemble . . . . .	57
3.32 Exemple de vue depuis une voiture . . . . .	58
3.33 Exemple de vue d'un compteur . . . . .	59
4.1 Exemple de test d'arrêt de voiture avec une voiture test . . . . .	62
4.2 Exemple de test de carrefour . . . . .	62
4.3 Exemple de test de présélection . . . . .	63
5.1 Illustration problème des feux . . . . .	66
5.2 Illustration du problème des présélections . . . . .	67

# Chapitre 1

## Introduction

### 1.1 Contexte

Il existe de nombreux simulateurs pour la mobilité urbaine, qu'ils soient en 2D ou 3D. On peut citer par exemple quelques simulateurs connus dans le domaine comme PTV Vissim<sup>1</sup> ou Aimsun<sup>2</sup> pour une visualisation d'une simulation 2D ou 3D.

Cependant, ces simulateurs présentent des conducteurs idéaux qui ne provoquent pas d'accidents. De plus, les conflits sont réglés de manière non naturelle et très rapidement. Ainsi, lors de l'aménagement d'un carrefour, il est difficile de visualiser les conflits réels qui surviendront. Ce projet de Bachelor *Simulation de mobilité urbaine 3D* propose donc d'utiliser les données fournies par les SITG (système d'information du territoire genevois), afin de créer une simulation 3D ainsi qu'une visualisation des comportements et objets routiers.

Lors du déroulement de la simulation, les différents objets routiers auront leur propre comportement. Les automobilistes agiront de manière à suivre les règles du code de la route selon leur profil de conducteur. Ils devront respecter les priorités, stop, feux de circulations, limitation de vitesse, ainsi que s'arranger lors d'une situation de blocage dans un carrefour.

L'augmentation des performances des ordinateurs et des moteurs de jeux permet actuellement de mettre en œuvre un simulateur 3D. En particulier, avec le moteur de jeu Unity3D, il est possible de gérer les comportements des objets routiers.

Grâce à une simulation 3D, on pourra mieux appréhender les impacts des aménagements sur la mobilité urbaine. De plus, cela en permettra une meilleure compréhension et analyse pour le grand public.

### 1.2 Le projet “Simulation de mobilité urbaine 3D”

Le projet *Simulation de mobilité urbaine 3D* utilise des outils tels que QGIS pour visualiser, sélectionner et extraire les informations routières provenant du SITG (Système d'Information du Territoire à Genève). Ces informations doivent ensuite être stockées dans une base de données.

---

1. <http://vision-traffic.ptvgroup.com/fr/produits/ptv-vissim/>  
2. <https://www.aimsun.com/>

Le populaire moteur de jeu Unity permet alors de créer un environnement de simulation 3D dans lequel les différents comportements routiers sont visualisables et analysables.

Le but de ce projet étant de réaliser une simulation 3D réaliste d'un carrefour ou d'un quartier choisi au préalable, la simulation se base sur des données statiques fournies par le SITG. Cependant, le logiciel de simulation n'est pas lié au SITG. Les données peuvent venir d'un autre canton ou d'un autre pays, tant que les informations requises sont présentes.

Voici un schéma démontrant la séparation entre les données SITG et la partie du simulateur 3D. Le simulateur dépend de la bonne mise en forme des données des fichiers à charger.

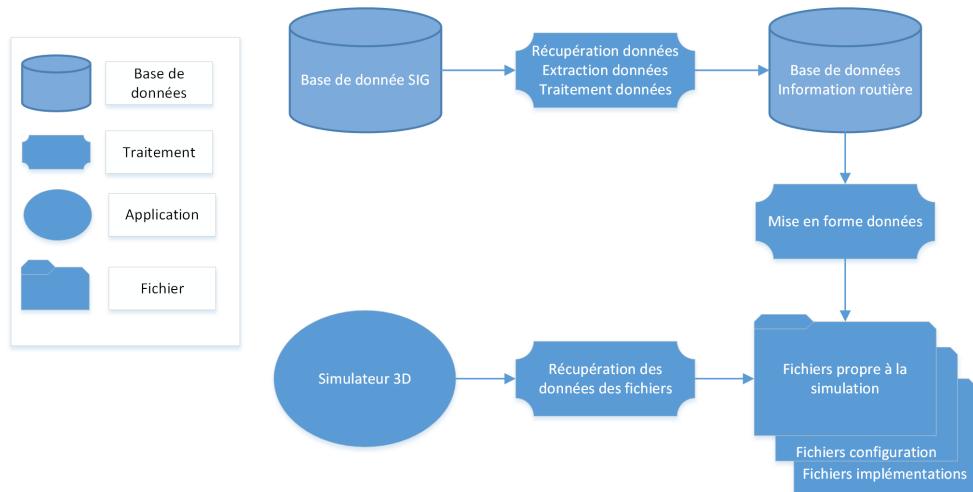


FIGURE 1.1: Représentation de l'architecture globale du simulateur de mobilité urbaine

Comme mentionné précédemment la plupart des simulateurs disponibles gèrent des cas idéaux, c'est-à-dire qu'il n'y a pas d'accidents ou des voitures qui force le passage. Tous les comportements de voiture sont quasiment les mêmes. Or, ce projet propose des comportements indépendant pour chaque voiture, ainsi qu'une gestion de collision proche du réel. Ainsi en cas d'inter-blocage, les voitures ne seront pas directement débloquées. Il faudra un temps d'attente afin que les automobilistes se mettent d'accord sur qui va passer en premier.

Chacun des automobilistes aura un point de départ et un point d'arrivée. Grâce au graphe routier du SITG, les voitures calculeront le parcours le plus court pour arriver à destination. Il existe des accidents comme dans la vie réelle. Ces accidents permettront de montrer les voies susceptibles d'en causer fréquemment. En cas d'accident, il faudra attendre un moment avant que les voitures accidentées soient dégagées de la voie. Ceci devrait faire apparaître les axes sensibles.

### 1.3 Architecture globale

L'établissement de l'architecture globale du projet a été l'une des premières étapes. Cette architecture permet de mieux structurer l'application que l'on souhaite obtenir, ainsi que d'avoir une meilleure vue de l'ensemble des étapes à réaliser pour finaliser le projet.

La représentation 1.2 montre toutes les étapes pour arriver à la simulation en séparant la partie Unity de celle du SITG. A ce stade, elle ne montre pas la structure de classes de la simulation.

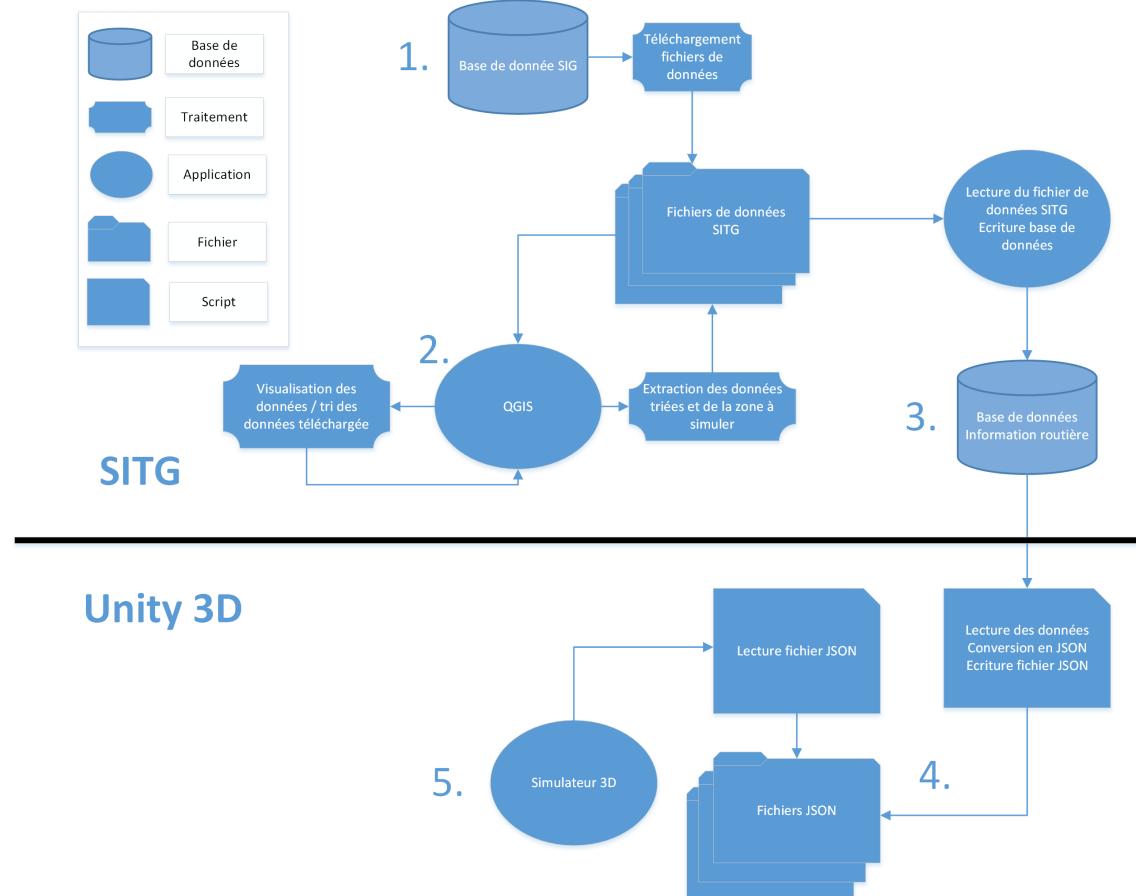


FIGURE 1.2: Représentation de l'architecture globale

Les grandes étapes de traitement de données illustrées dans la figure 1.2 :

1. BASE DE DONNÉES SIG. Récupération des données via le catalogue de données du SITG <sup>3</sup>.
2. QGIS. Visualiser, sélectionner et exporter les données téléchargée du SITG, afin de mettre à jour les FICHIERS DE DONNÉES SITG.
3. BASE DE DONNÉES INFORMATION ROUTIÈRE. Alimentation à travers une application de la base de données depuis les fichiers SITG/QGIS.
4. SIMULATEUR 3D. Conversion des informations de la base de données en FICHIER JSON.
5. SIMULATEUR 3D. Récupération des données et exploitation des données pour le lancement de la simulation.

3. [http://ge.ch/sitg/sitg\\_catalog/sitg donnees](http://ge.ch/sitg/sitg_catalog/sitg donnees)

## Chapitre 2

# Technologies et outils

Ce chapitre décrit toutes les technologies ainsi que les outils nécessaires pour aboutir à la réalisation du projet *Simulation de mobilité urbaine 3D*.

## 2.1 SITG - Système d'information du territoire à Genève

Cette section va décrire les données utiles du SITG, ainsi que fournir les explications sur les données proposées et utilisées dans le cadre de ce projet.

### 2.1.1 Description

Cette section est directement tirée du site officiel du SITG, le système d'informations du territoire à Genève[6]. "Le SITG est un organisme fondé sur un réseau de partenaires publics ayant pour but de coordonner, centraliser et diffuser largement les données relatives au territoire genevois. Les partenaires partageant entre eux et avec un large public les données géographiques produites dans le cadre de leurs missions.

Il comporte l'ensemble des données intervenant dans l'organisation du territoire et les outils associés qui en permettent la gestion, la consultation et la restitution multiformes. Une large partie de ces géodonnées sont ouvertes pour une libre réutilisation afin de favoriser l'innovation, le dynamisme et la création de services à valeurs ajoutées, pour les entreprises et pour le public.

Il constitue en même temps :

- une mémoire du territoire ;
- un outil de recherche, de traitement et de diffusion des informations géolocalisées ;
- un outil de communication entre les partenaires de la gestion du territoire ;
- un outil d'aide à la décision fournissant une image flexible et dynamique du territoire et de ses contraintes ;
- un vecteur d'innovation et de dynamisme autour du territoire.

Un grand nombre de données sont mises à disposition en *Open Data* dans leur catalogue. Une sélection est à réaliser sur les données nécessaires à récupérer pour alimenter notre projet de simulation. Le site est convivial, la prise en main facile car il est intuitif. Il est riche en informations concernant les domaines du territoire genevois."

### 2.1.2 Données

Il y a différents types de données mises à disposition par le SITG. Voici les formats d'extraction proposés concernant les données vectorielles :

- SHAPE ;
- GEODATABASE-FILE - Présentation visualisation ;
- GML,
- KML,
- CSV.

Certains types de données ne sont pas disponibles selon le format choisi. Certains formats deviennent inutilisables au-delà d'un certain volume de données ou sont inexploitables car ils ne contiennent parfois pas les coordonnées géométriques des objets, qui correspond à la position x,y de l'objet. C'est par exemple le cas pour le format CSV.

Après réflexion, le format Shape a été choisi. Ce format permet de récupérer l'ensemble des informations nécessaires : géométrie des lignes, points et polygones. De plus, de nombreuses librairies sont disponibles pour la lecture de ce format.

Certaines données sont manquantes ou inexploitables pour ce projet. Néanmoins, il faut savoir que le catalogue ne fournit pas toutes les données en "Open Data" pour des questions de non publication de données, de libre accès ou de données trop volumineuses. Cependant il est toujours possible d'obtenir certains types de données en faisant une demande aux personnes compétentes.



FIGURE 2.1: Flux : A gauche données SITG inexploitables, à droite données fournies par une personne du domaine

L'illustration 2.1 montre un exemple de données inexploitables dans le cadre de ce projet. Ces données représentent la charge de trafic : à gauche le graphe n'est pas superposé avec le graphe routier qui de plus ne possède aucun lien avec ce graphe (nom de rue, identifiant, etc.). Ceci est une illustration du cas où il a fallu faire une demande à un expert (M. Alain Dubois) afin de récupérer des données exploitables (à droite).

### 2.1.3 Données utilisées

Les données utilisées pour la simulation sont celles du graphe routier, des flux automobiles, des limitations de vitesses et des feux de circulation.

### Graphe routier

Pour la simulation, il est nécessaire d'avoir un graphe routier représentant les routes du quartier :

- SOMMETS ET ARÈTES,
- NOMBRE DE VOIES ET LEUR DIRECTION,
- VITESSES.

Le graphe routier sera utilisé pour calculer le plus court chemin d'un point d'entrée à sa destination ainsi que pour définir les voies empruntées par une voiture.

Le fichier GMO\_GRAPHE\_ROUTIER contient les coordonnées des sommets et arêtes ainsi que le nombre et le sens des voies de circulation. Les informations routières concernant les limitations de vitesses sur les arêtes ne sont pas présentes dans ce fichier : elles sont traitées à partir d'un autre fichier (voir la section suivante "Limitation de vitesse").

### Limitation de vitesse

Pour compléter le graphe routier, le fichier OTC\_ZONE\_MODERATION\_TRAFIC permettra de fixer les limitations de vitesse sur les arêtes. Ce fichier contient les zones 20 et 30 km/h définies par des polygones.

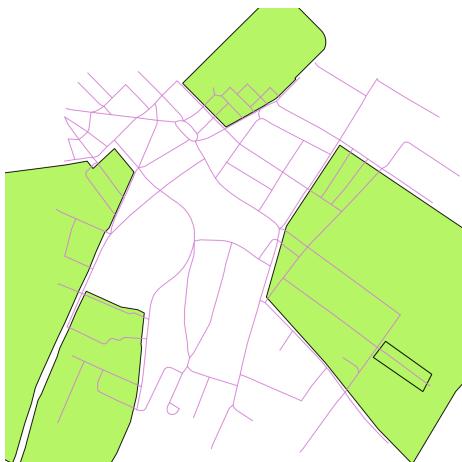


FIGURE 2.2: En vert zone de modération de vitesse (20/30 km/h)

Toutes les voies en dehors de ces polygones sont limitées par défaut à une vitesse de 50 km/h.

### Flux automobiles

Les données des flux automobiles sont importantes pour une simulation réaliste. Le flux indique le débit de voitures entrant et sortant du quartier sur une arête.

Le flux sortant va permettre de définir la répartition des points de destination des voitures. Une règle a été établie pour que chaque voiture entrante ait une destination sortante, ceci afin d'éviter qu'un automobiliste "tourne en rond" dans un quartier. On considère donc essentiellement un trafic de transit.

Le fichier PLANDECHARGE2010 contient beaucoup d'informations dont seules les informations de comptage routier seront retenues. Ce fichier propose des charges moyennes pour les créneaux horaires :

- 7H - 8H,
- 8H - 9H,
- 16H - 17H,
- 17H - 18H,
- 22H - 6H.

Seule la tranche horaire 8h-9h sera simulée, car elle est la plus représentative des problèmes de circulation à Genève.

Ce fichier de flux ne provient pas du système du SITG, mais a été fourni par M. Alain Dubois (voir illustration 2.1).

### Feux de circulation

Pour tout ce qui concerne les signaux lumineux tels que les feux de circulation, les informations sont stockées dans le fichier OTC\_SL\_EQUIPEMENT\_BOITEFEUX. Ce fichier fournit les positions de tous les signaux lumineux présents sur un carrefour, que ce soit pour les piétons, les transports publics, les cyclistes ou les véhicules privés.

Ce fichier n'est pas lié au graphe routier. Il ne permet pas de lier un feu de circulation avec la voie de circulation correspondante et il ne contient pas la gestion des signaux des feux (rouge, jaune, vert).

Ce fichier est volumineux ; ceci est dû aux informations concernant l'ensemble des feux des piétons, des cyclistes et des transports en commun. Ces informations seront supprimées, car on se concentrera sur les feux pour les voitures.

## 2.2 QGIS

Cette section explique ce que l'outil QGIS apporte pour la conception de ce projet. Sachant que QGIS est un Système d'Information Géographique (SIG), nous allons voir les avantages d'utiliser un tel logiciel.

### 2.2.1 Description

QGIS est un outil utilisé pour créer, éditer, visualiser, analyser et publier des informations géographiques. Il est compatible avec différents systèmes d'exploitation. Cet outil est complètement open source et téléchargeable sur leur site [www.qgis.org](http://www.qgis.org)

QGIS propose un grand nombre de fonctionnalités et des extensions. L'utilisation de QGIS dans le cadre de ce projet n'a pas nécessité l'installation d'une extension. Rien qu'avec les fonctionnalités de base il est possible de visualiser, d'analyser et d'extraire une sélection des données géographiques voulues.

### 2.2.2 Utilisation des données géographiques

QGIS a été utilisé essentiellement pour la visualisation et l'extraction des données du SITG.

La fonctionnalité d'extraction des données permet d'avoir un fichier contenant les informations souhaitées sur une zone définie.

Le schéma 2.3 représente le traitement de données avec l'utilisation de QGIS. Il montre le chargement des données du SITG dans QGIS, une visualisation ainsi que le tri des données à extraire. Pour finir, QGIS modifie les fichiers téléchargés de la base de données SIG par des fichiers triés et extraits sur une zone prédéfinie. Ceci permettra pour la suite d'avoir des fichiers contenant seulement les informations routières nécessaires à la simulation.

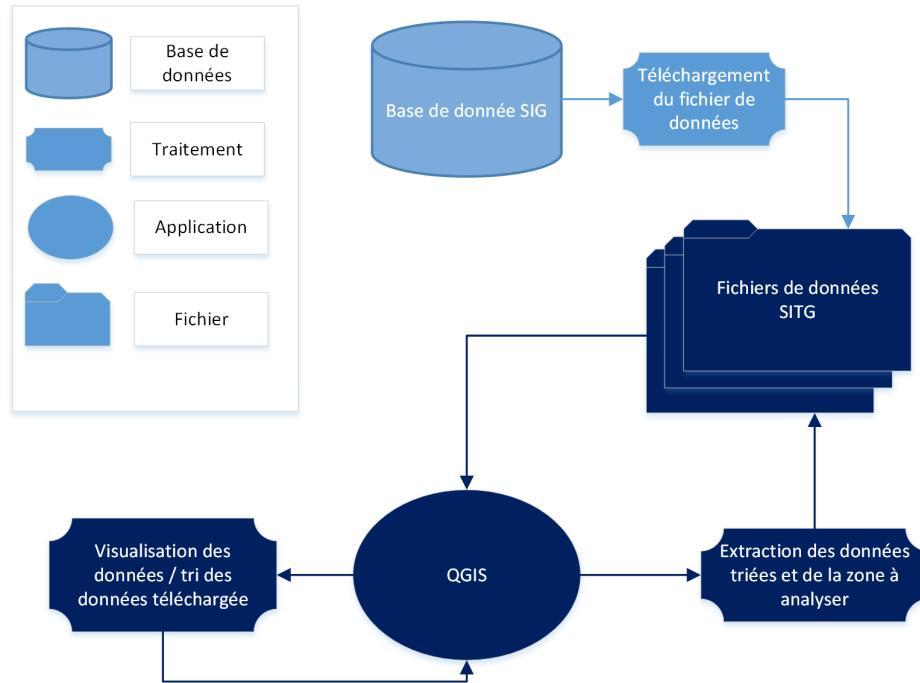


FIGURE 2.3: Représentation du schéma liant le SITG à QGIS

La visualisation a permis de faire un tri entre les données qui sont pertinentes pour le bon fonctionnement du projet et celles qui ne le sont pas. Cela permet de mieux réaliser comment utiliser ces données avec ces attributs et de connaître les informations présentes ou manquantes pour le déroulement correct de la simulation.

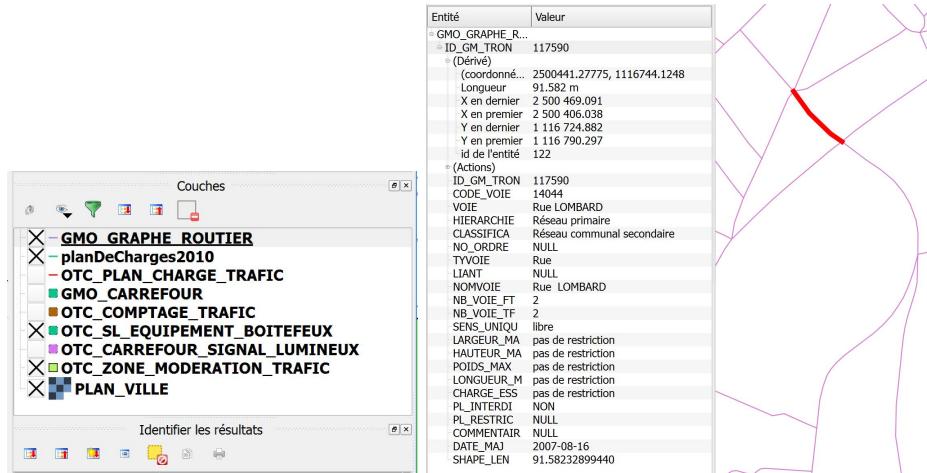


FIGURE 2.4: Visualisation de données avec QGIS

Les données sont affichées en couches. Ceci permet de mieux organiser le projet lors de la phase de tri de données. Par exemple, on peut mettre en avant certaines données. Par la suite, on pourra visualiser ce que stockent les différentes couches. Dans la figure 2.4, il est possible de voir toutes les informations que contient un segment du fichier `GMO_GRAPHE_ROUTIER`.

L'outil QGIS permet de définir une zone à extraire. Cette extraction est donc restreinte à la zone qu'on souhaite utiliser par la suite. Après la définition de la zone et son extraction, un nouveau fichier de données du SITG est fourni. Ce fichier contient seulement la zone extraite, ce qui facilite la visualisation ainsi que la lecture des données.



FIGURE 2.5: Visualisation graphe routier avant et après extraction

Certaines données venant du catalogue des SITG représentent l'ensemble du canton. Afin d'éviter un tri depuis une application tierce, QGIS propose la possibilité d'extraire la zone que l'on désire.

## 2.3 Base de données

La base de données permet de stocker et structurer des données afin d'en faciliter l'accès. Cette section permet d'analyser en quoi la base de données nous a apporté une meilleure architecture du projet.

### 2.3.1 Description

La base de données sera utilisée pour stocker toutes les données reçues du SITG, afin de séparer le SITG de Unity. Il est intéressant de casser le lien direct entre le SITG et Unity, afin que le programme de simulation soit complètement indépendant du SITG. Ceci permettra par la suite de pouvoir remplacer les données du SITG par un autre jeu de données.

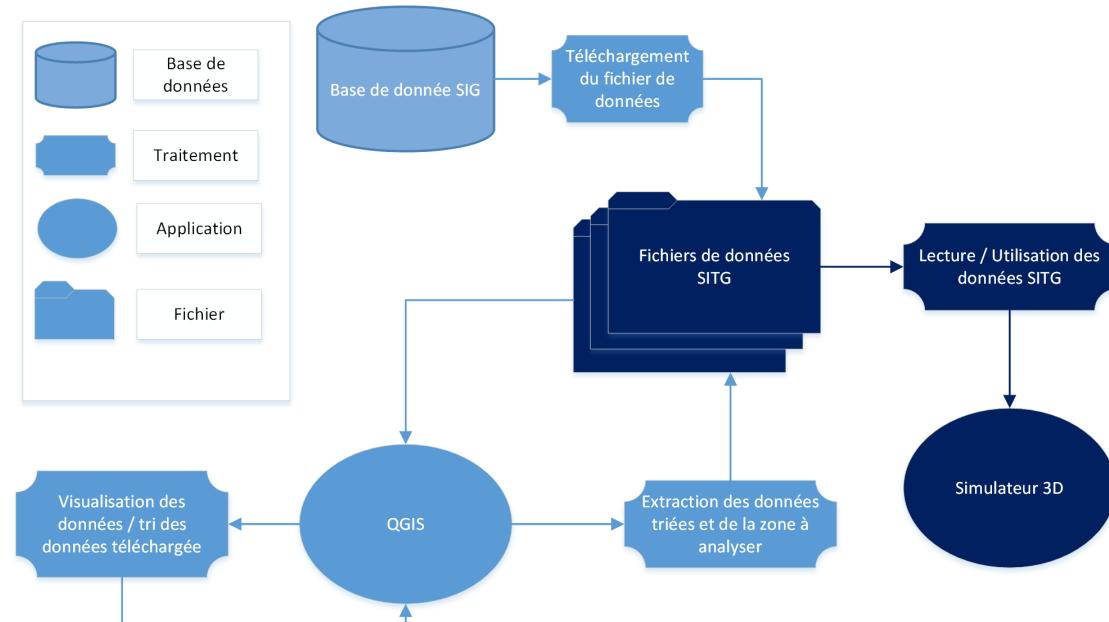


FIGURE 2.6: Représentation du schéma liant SITG et la simulation

Le schéma 2.6 montre un lien direct entre les données du SITG et la simulation 3D. Comme le schéma le montre, le simulateur 3D utilise les données du SITG. Il faut relever que ces données sous le format Shape contiennent des identifiants qui ne sont pas forcément communs. Par exemple, les données du graphe routier de Genève contiennent `nb_voie_ft` comme identifiant pour le nombre de voies dans un sens. Cependant, si nous voulons faire une simulation de la ville de Zurich, le nombre de voies ne contiendra pas cet identifiant. Donc afin d'éviter de modifier toute la structure de données du simulateur dans le cas d'un changement de ville, il est préférable d'avoir une séparation entre la base de données des SIG et le simulateur. Seul le traitement d'injection de données dans la base de données sera à adapter. Le simulateur ne nécessitera aucune modification.

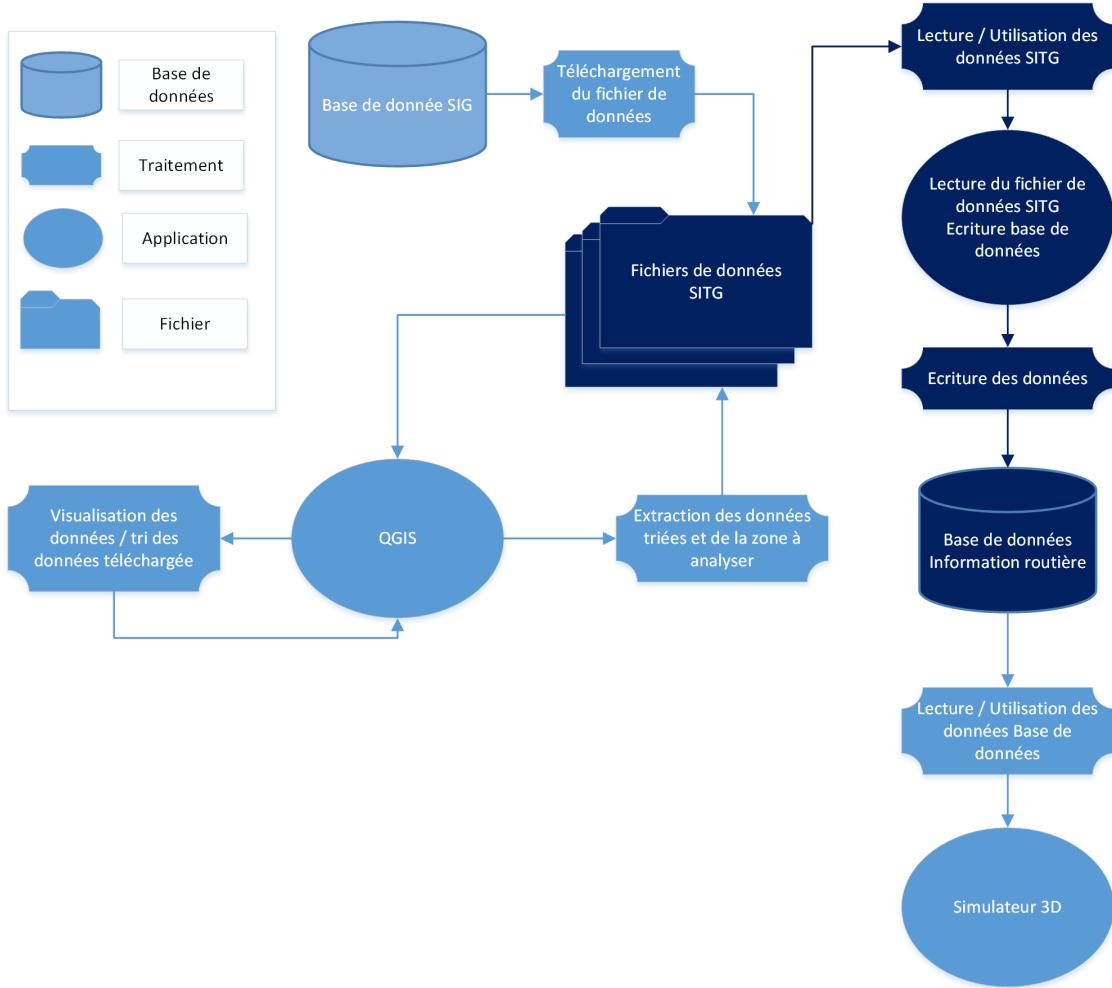


FIGURE 2.7: Représentation du schéma liant SITG à une base de données puis à la simulation

Le choix de la base de données permet de garantir une intégrité des données au travers de liens relationnels. Les données seront stockées selon le bon format afin de réaliser moins de traitement du côté applicatif.

Un des avantages est que la base de données pourrait être stockée sur un serveur distant, si on veut faire du multi-utilisateur. On pourra donc y sécuriser et récupérer un historique des données.

### 2.3.2 Modèle relationnel

La figure 2.8 illustre le modèle de la base de données pour stocker les informations routières.

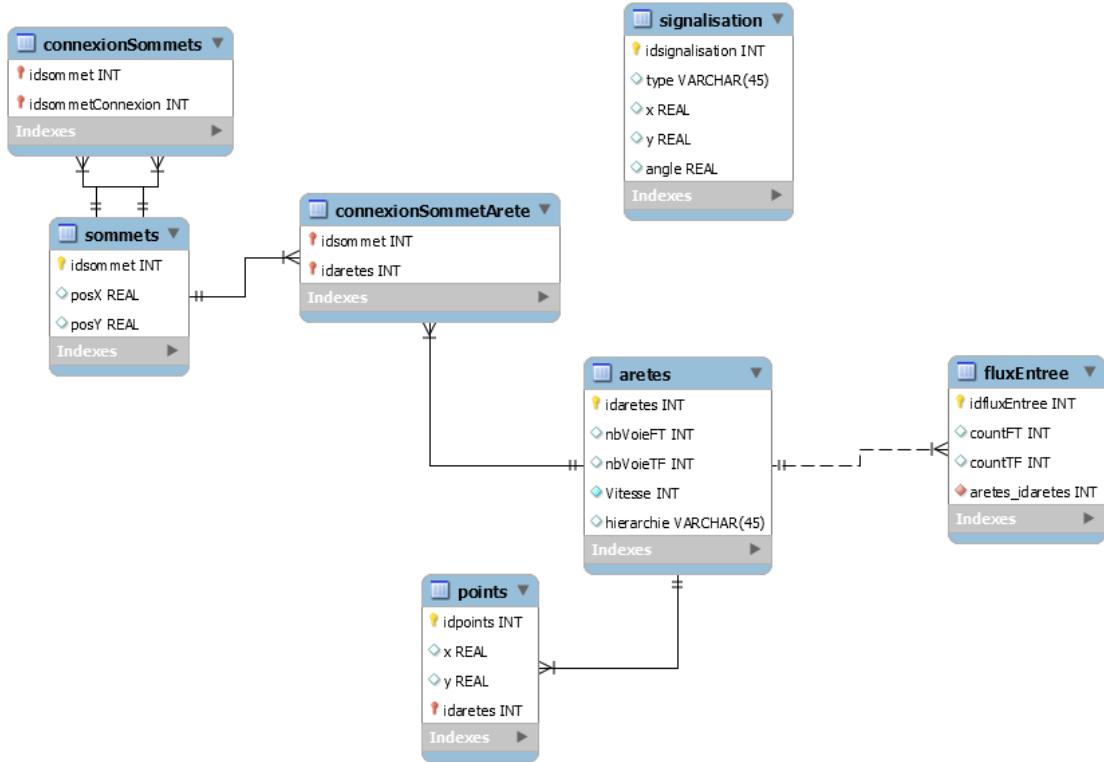


FIGURE 2.8: Modèle relationnel de la base de données

### 2.3.3 SQLite

Comme l'application est destinée à s'exécuter sur une machine en local, une base de données SQLite (local) convient. L'installation d'une telle base de données est très simple et permet un gain de temps. Pour le moment, il n'y a pas d'intérêt à définir une base de données sur un serveur distant car il pourrait potentiellement avoir de mauvaises performances d'accès et une installation plus complexe.

Suite au chargement et à la lecture des données provenant du SITG, un traitement alimentera la base de données qui contient les informations routières nécessaires au bon fonctionnement de l'application.

L'utilisation d'une base de données PostGreSQL était envisageable, de par son extension PostGIS qui est une base de données spatiale. Il ajoute le support pour les objets géographiques permettant des requêtes de localisation exécutées en SQL. Cependant, l'utilisation de PostGIS n'aurait été utile que dans un cas identifié : déterminer si une arête est dans un polygone, pour le traitement des limitations de vitesse (voir section 2.1.3) ; l'intérêt est donc faible.

## 2.4 C#

### 2.4.1 Description

Le C# est un langage de programmation orienté objet, ayant une forte ressemblance avec Java. Il est régulièrement utilisé pour développer des applications de bureau, web ou des widgets.

Ce langage a été développé par Microsoft et a un accès à l'API .NET. De plus, plusieurs packages sont mis à disposition afin de réaliser les fonctionnalités désirées.

La totalité des scripts écrits dans le cadre du simulateur l'ont été en C#.

Etant donné que Unity utilise soit UnityScript (langage similaire au JavaScript), soit C#, le choix s'est tout de suite porté sur le C# pour profiter de toutes les fonctionnalités mises à disposition, notamment son accès à l'ensemble de la librairie .NET.

Naturellement la création d'une application de récupération des données du SITG et son traitement, ainsi que l'insertion et l'édition de la base de données furent réalisée à l'aide d'une interface utilisateur C#.

### 2.4.2 Utilité

Unity a la particularité de supporter trois sortes de langages :

- C#,
- UNITYSCRIPT, langage proche du JavaScript,
- Boo.

Etant donné que UnityScript est un langage propre à Unity, mais ressemblant au JavaScript et Boo est très peu utilisé dans le monde du développement, il est fortement conseillé de créer les scripts en C#. De plus, une grande communauté gravite autour de ce langage dans le monde Unity. Par ailleurs, ceci permet d'avoir une homogénéité du langage de programmation. Cela rend aussi plus probable la réutilisation par d'autres développeurs.

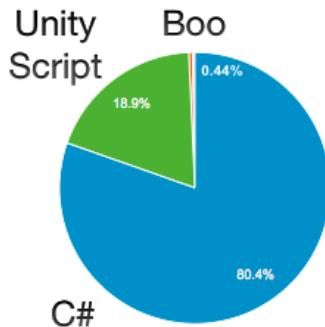


FIGURE 2.9: Graphe d'utilisation des différents langages proposé par Unity

Le fait que C# du côté Unity propose l'accès à l'API Microsoft .NET, signifie aussi un accès à des types d'objets plus variés et plus intéressants. De plus la communauté Unity propose beaucoup de solutions, documentation autour de ce langage, contrairement à Boo ou à UnityScript qui sont moins répandus.

Pour le développement d'une application indépendante à Unity, il est possible d'utiliser l'environnement de développement Visual Studio pour Windows. Cet environnement propose un ensemble d'outils afin de développer une application en C#. L'avantage d'utiliser un tel environnement est la simplicité de mise en place d'une application.

L'illustration 2.10 montre le développement de l'application C# du chargement des données du SITG dans la base de données.

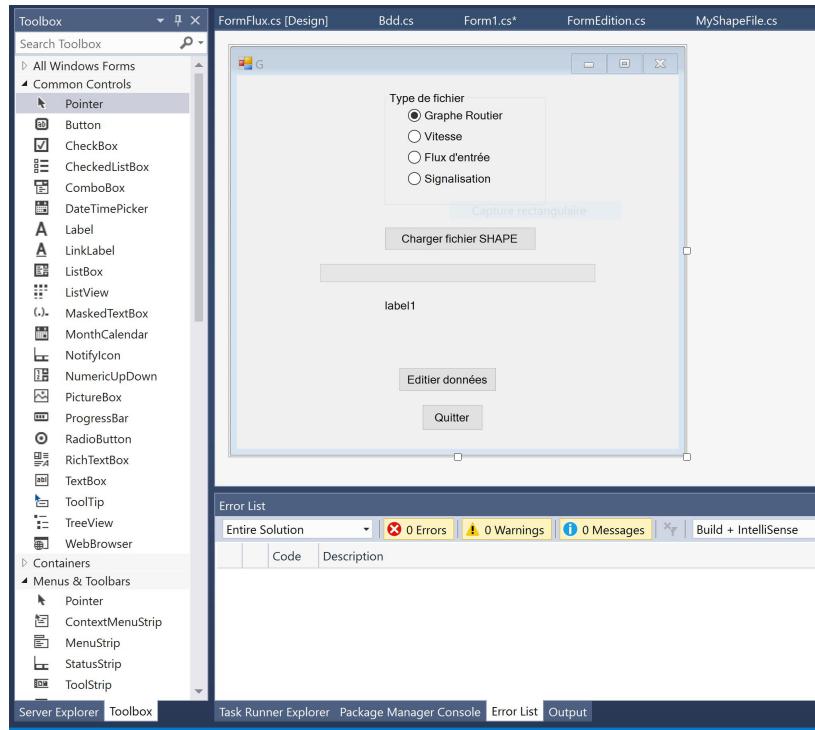


FIGURE 2.10: Visualisation pour l'implémentation d'une Form C#

Les outils pour l'implémentation d'une application sont disponibles et facilement réalisables (barre d'outil de gauche). Ultérieurement, il est très aisément d'interagir avec les différents contrôles mis en place sur l'application depuis l'éditeur Visual Studio.

## 2.5 JSON

### 2.5.1 Description

Le format JSON vient du monde JavaScript et s'apparente au XML. Ce format n'a aucune dépendance avec JavaScript, donc il est pris en charge par beaucoup de langages tels que le JavaScript, Python, Java, C# et d'autres. Le JSON possède sa propre syntaxe très simple, qui est compréhensible par tous.

- { ... } : les accolades définissent un objet ;
- "CLÉ" : VALEUR : ce format définit un membre (Clé/Valeur) ;
- [...] : les crochets définissent un tableau ;

— , : les virgules permettent de séparer les membres.

Le format JSON supporte différents types de données.

— CHAÎNES DE CARACTÈRES ;

— NOMBRES ;

— BOOLEAN (VRAI / FAUX).

### 2.5.2 Utilité

Rappelons que l'utilité principale est d'avoir un intermédiaire séparant la partie de la base de données de celle de la simulation. Ceci permet par la suite de s'abstraire du type de base de données (SIG, SQLite, PostGreSQL). Le simulateur accèdera à des données préparées au préalable et contenues dans les fichiers JSON.

Le JSON a la particularité d'avoir une structure élémentaire. Ceci permet au simulateur d'avoir sa propre structure de données avec une normalisation simple et compréhensible.

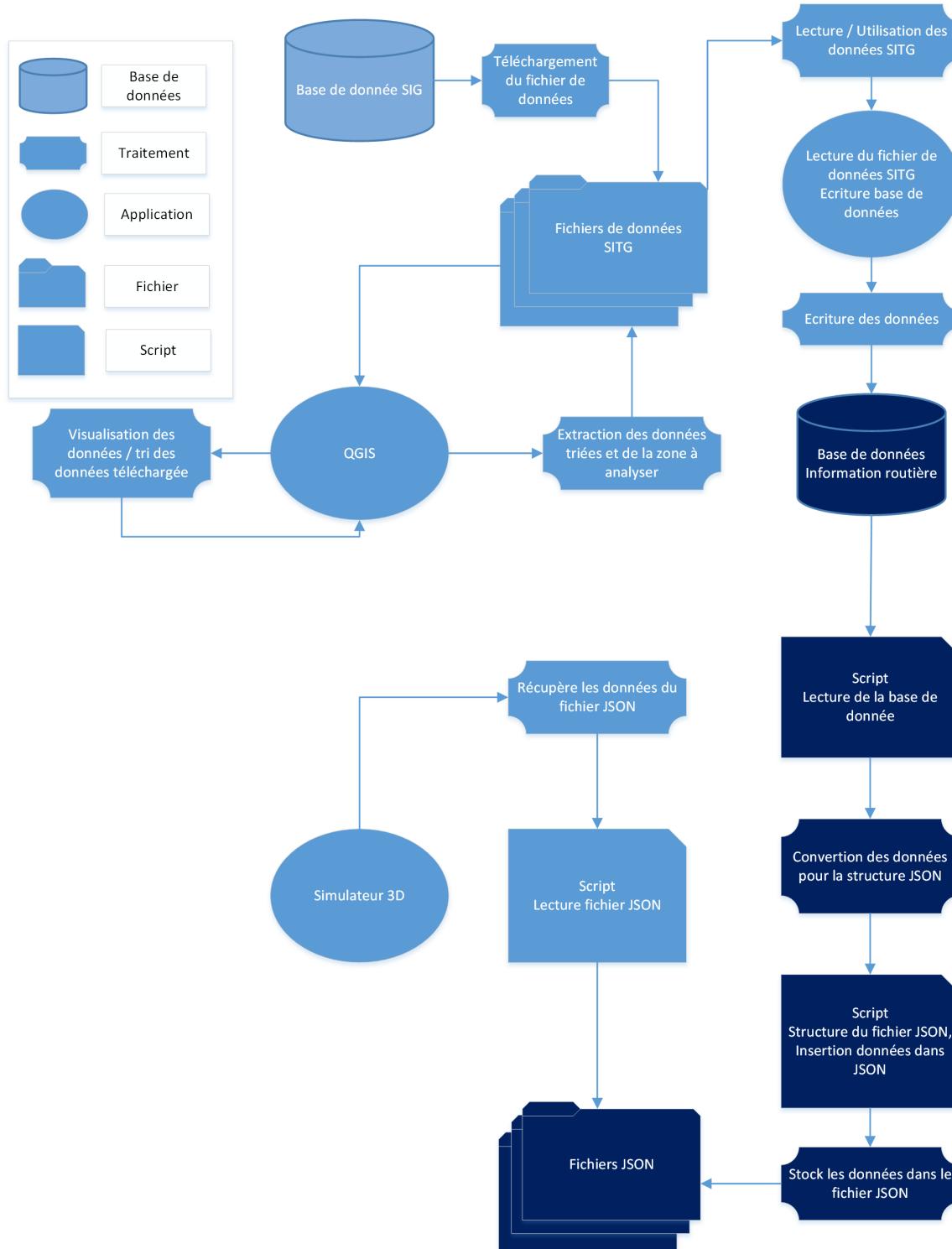


FIGURE 2.11: Représentation du schéma liant la base de données à JSON

Grâce à un fichier JSON normalisé, il suffira de remplir les champs requis pour avoir un fichier fonctionnel avec la base de données.

### 2.5.3 Avantage et désavantage

Si la normalisation est bien réalisée et facile à prendre en main, c'est un véritable atout pour une nouvelle implémentation d'une autre structure de stockage de données et un réel gain de temps. Par contre, si elle est complexe et difficile à comprendre, cela pose plus de problèmes qu'elle n'en résout.

Il est important de prendre du temps afin de définir les structures des fichiers JSON, afin de réaliser un gain de temps lors de l'implémentation dans un système de stockage.

Comme dit précédemment, la structure d'un fichier JSON peut être lue par le commun des mortels grâce à sa structure simple. Ceci est un avantage majeur pour la compréhension de la structure proposée pour le simulateur.

## 2.6 Unity

### 2.6.1 Description

Unity est un moteur de jeu compatible avec différentes plateformes. Il est fortement utilisé dans l'univers du jeu vidéo 2D ou 3D et compte un grand nombre d'adeptes professionnels ou privés, ce qui a permis d'avoir une grande communauté autour de cet outil. Unity propose aussi une documentation en ligne très complète qui explique chaque fonctionnalité de l'outil.

Unity est régulièrement utilisé comme éditeur de jeu. Cependant il existe bon nombre d'applications développées à partir d'Unity qui n'ont aucun lien avec des jeux, tels que des éditeurs de carte 2D/3D, simulation de fluide<sup>1</sup>.

Unity offre une version gratuite ainsi qu'une version payante. La version payante propose plus de fonctionnalités que la version gratuite. Cependant cette version gratuite est déjà largement dotée pour développer des applications complètes.

### 2.6.2 Utilité

Unity est un éditeur de jeu proposant beaucoup de fonctionnalités très utiles pour un simulateur. Dans une simulation, les voitures ont leur propre "intelligence artificielle", tout comme dans le monde des jeux vidéos avec des personnages par exemple.

La gestion de la physique existant dans Unity est extrêmement performante et complète, ce qui permet d'avoir des comportements plus réalistes.

Pour une simulation réaliste, il faut une gestion des collisions, de la physique et une gestion de vecteurs. Unity possède toutes ces fonctionnalités. L'utilité de cette bonne physique est d'avoir des voitures qui réagissent selon les lois de la physique sur la Terre, c'est-à-dire par exemple qu'elles ne s'envolent pas lors d'un choc avec un autre objet.

La gestion des collisions permet de créer des éléments solides. Unity propose diverses sortes de collisions : les collisions entre solides et une zone de détection non solide. Ces fonctionnalités sont déjà présentes et très complètes dans Unity.

---

1. <http://madewith.unity.com/>

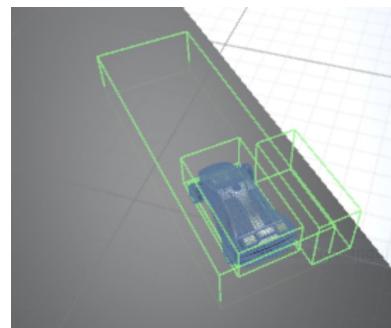


FIGURE 2.12: Représentation des zones de collision

Dans l'exemple 2.12, il est possible de voir que l'élément possède plusieurs zones de collision. La zone de collision la plus proche de l'objet est dite "solide", les objets ne passent pas à travers cette zone. Si deux éléments dits "solides" se touchent, cela aura un impact sur la physique de ces deux objets, comme une collision entre 2 billes qui s'entrechoquent.

Cependant la zone de collision sur la droite et la grande zone entourant l'objet sont des zones de collision dites "non solide". Il y aura un traitement si un élément entre dans cette zone mais il n'y aura pas d'impact physique sur l'objet.

Ceci est un exemple parmi tant d'autres des avantages qu'un moteur de jeu tel qu'Unity apporte pour le développement d'une telle simulation.

### 2.6.3 Comparatif d'Unity et d'autres moteurs de jeux

Il existe bon nombre d'éditeurs de jeu tout aussi performant qu'Unity.

Unity a déjà un énorme avantage comparé à tous les autres moteurs de jeux, c'est la communauté autour de ce moteur de jeu ainsi qu'une documentation complète. Unity possède 45% du marché des moteurs de jeu.<sup>2</sup>

Cependant tous ont des qualités et des faiblesses. Par exemple Blender propose aussi un moteur de jeu, mais pas aussi performant ni autant adaptable qu'Unity, car la fonction première de Blender est la modélisation. Le moteur de jeu de Blender est orienté pour faire des tests de physique sur un objet 3D. Cependant, les modèles réalisés sur Blender sont facilement importables dans Unity.

Il existe un autre moteur de jeu qui fait sa place dans ce monde, il s'agit d'Unreal Engine.

Parmi les leaders du marché, Unity possède une avance sur les autres, car il reste le moteur de jeu favori des indépendants. Cependant, Unreal Engine possède une large communauté autour de lui qui propose diverses documentations, ainsi que des tutoriels.

Voici une répartition du taux d'utilisation de différents moteurs de jeu (2.13) selon : <https://unity3d.com/public-relations>.

---

2. <https://unity3d.com/public-relations>

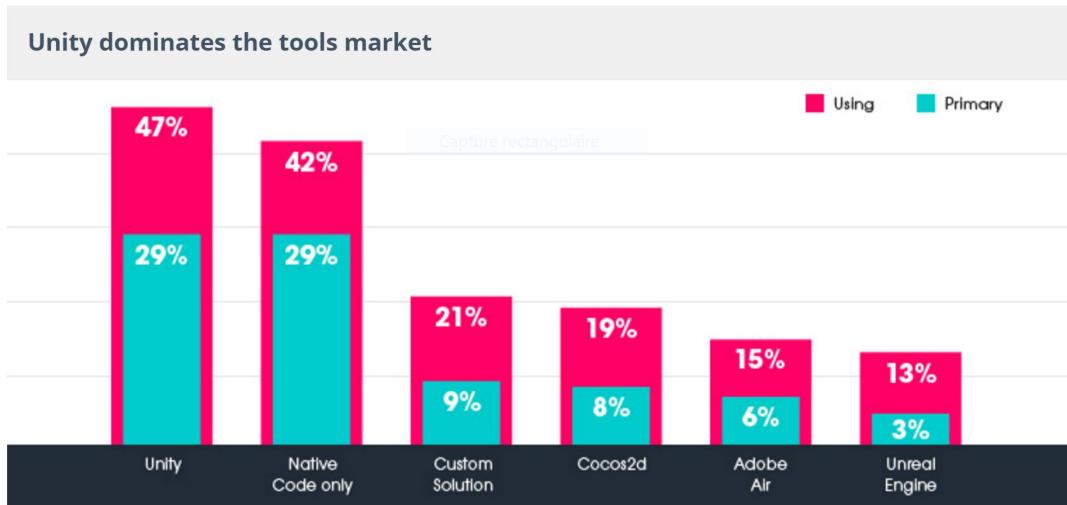


FIGURE 2.13: Marché des moteurs de jeu [8]

Unreal Engine propose en cas de commercialisation d'un projet un reverssement de 5% de la recette en plus de 3000 dollars pour chaque trimestre. Au contraire, Unity propose une publication gratuite aux entreprises ou aux développeurs individuels dont le chiffre d'affaire n'excède pas les 100'000 dollars par an. La seule contrainte étant que lors du lancement de l'application un message montre qu'elle a été réalisée par Unity 3D.

Ces deux modèles économiques sont bien différents mais Unity reste plus séduisant pour les particuliers.

Même si il existe une multitude de moteurs de jeu 3D, les leaders du marché restent Unity et Unreal Engine.

# Chapitre 3

# Implémentation

Ce chapitre explique comment les fonctionnalités ont été mises en place.  
Le schéma 3.1 fait apparaître les thèmes des sections abordées dans ce chapitre.

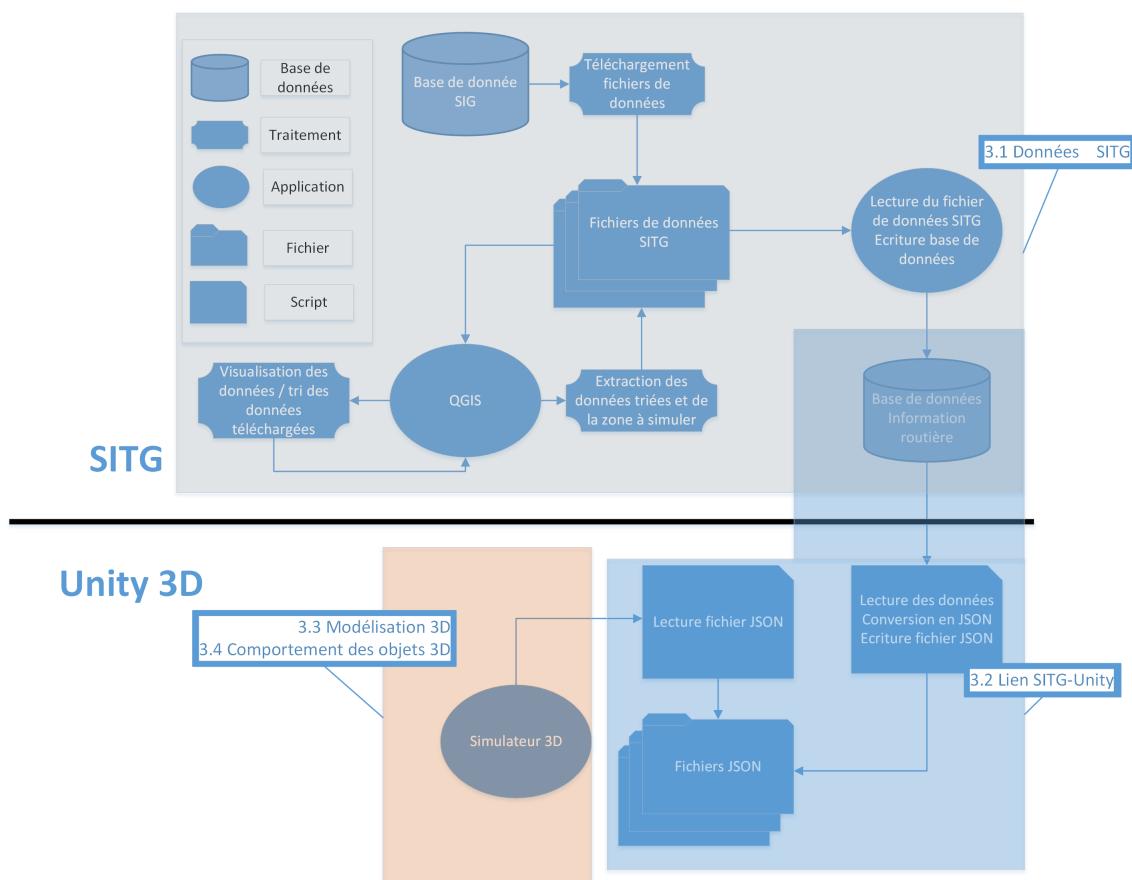


FIGURE 3.1: Répartition des sections dans l'architecture

En particulier, la partie de l'application concernant le simulateur 3D lui-même sera détaillée

dans les sections (3.3 et 3.4).

Pour rappel tout ce qui a été implémenté dans le cadre de ce projet se trouve à l'adresse suivante :

<https://github.com/nicolasvandooren/SimulationUrbaine3D>

## 3.1 Organisation et utilisation des données du SITG

Le traitement des données téléchargées de la base de données du SITG est un élément important du projet.

Après téléchargement, visualisation, tri et extraction à l'aide de l'outil QGIS, les données sont prêtes à être traitée afin d'alimenter la base de données avec les informations routières d'un quartier (le modèle relationnel de la base de données a été donné à la section 2.8).

### 3.1.1 Les données du SITG

Les données utilisées ont été expliquées au chapitre *Technologies et outils* (voir section 2.1.3).

Pour rappel, les données SITG extraites sont les informations concernant le graphe routier, les zones 20 et 30 km/h, les flux automobiles et les feux de circulation.

Maintenant il faut établir des liens entre ces données pour créer les liens relationnels de la base de données. L'application devra proposer de trier les différentes sortes de données.

Pour la création du graphe routier, il faut définir un système d'arêtes et de sommets. Les sommets seront reliés pour donner les arêtes et correspondent ainsi aux extrémités des arêtes du graphe. Ils auront chacun un identifiant unique.

Les vitesses ne sont pas stockées dans le graphe routier, mais associées à un polygone (zone de même limitation de vitesse). Pour savoir si une arête du graphe fait partie d'un polygone, il suffit d'utiliser l'algorithme de Winding Number [4]. Cet algorithme détermine si une arête est à l'intérieur d'un polygone.

Ensuite, il faut lier les flux automobiles au graphe routier. Etant donné que celui-ci et le graphe des flux automobiles sont superposés, une simple comparaison d'arêtes sera nécessaire afin de savoir si l'arête du graphe correspond à l'arête de flux.

Après ces traitements il est possible de stocker les données de manière relationnelle.

### 3.1.2 Traitement

Le traitement est séparé en quatre parties distinctes : le graphe routier, les vitesses, les flux et les feux de signalisations.

Rappelons que le format Shape est celui qui a été choisi. L'avantage de ce format de fichier est que toutes les données sont stockées à l'aide d'un identifiant et que pour chaque donnée le type de géométrie y est stocké (point, ligne, polygone).

Une librairie en C# a été utilisée afin de réaliser la lecture des informations stockées. Il s'agit de *Catfood.Shapefile*. Cette librairie a permis une lecture complète d'un fichier Shape.

<pre> Shape 1, Type PolyLine Metadata: nb_voie_ft=1 (DBTYPE_WVARCHAR) largeur_ma=pas de restriction (DBTYPE_WVARCHAR) charge_ess=pas de restriction (DBTYPE_WVARCHAR) hierarchie=R\seau de quartier (DBTYPE_WVARCHAR) nomvoie=Avenue de Beau-S\jour (DBTYPE_WVARCHAR) commentair= (DBTYPE_WVARCHAR) classifica=R\seau communal secondaire (DBTYPE_WVARCHAR) code_voie=01600 (DBTYPE_WVARCHAR) poids_max=pas de restriction (DBTYPE_WVARCHAR) longueur_m=pas de restriction (DBTYPE_WVARCHAR) shape_len=73.1233646102 (DBTYPE_R8) tyvoie=Avenue (DBTYPE_WVARCHAR) sens_unique=dans le sens (DBTYPE_WVARCHAR) pl_restric= (DBTYPE_WVARCHAR) date_maj=12.11.2015 00:00:00 (DBTYPE_DATE) voie=Avenue de Beau-S\jour (DBTYPE_WVARCHAR) pl_interdi=NON (DBTYPE_WVARCHAR) hauteur_ma=pas de restriction (DBTYPE_WVARCHAR) id_gm_tron=113426 (DBTYPE_WVARCHAR) nb_voie_tf=0 (DBTYPE_WVARCHAR) no_ordre= (DBTYPE_WVARCHAR) liant=de (DBTYPE_WVARCHAR) </pre>	<pre> PolyLine Point: Number of points : 10 2500495.7902, 1116111.3762 2500490.1099, 1116100.3667 2500488.4591, 1116090.5267 2500490.5185, 1116083.5568 2500494.6177, 1116079.0473 2500502.4068, 1116076.9874 2500509.3764, 1116077.8174 2500515.9463, 1116081.5073 2500519.6363, 1116085.6075 2500524.087, 1116093.7373 </pre>
---	---

FIGURE 3.2: Exemple de la lecture d'une donnée d'un fichier Shape avec la librairie *Cat-food.Shapefile*

La figure 3.2 montre la lecture d'une donnée provenant du graphe routier. Cet exemple montre la phase d'analyse complète avant l'insertion dans la base de données, il est possible de visualiser les identifiants des données voulues ainsi que les points des arêtes.

Après la récupération de données nécessaires et leur traitement, il faut les stocker en base de données. Pour cela, on utilise la librairie *System.Data.SQLite* [13].

Par la suite, la création d'une classe **Bdd** est réalisée qui contient toutes les requêtes de lecture, insertion, édition et suppression des données de la base de données SQLite. Cette classe contient plusieurs objets afin d'en faciliter l'utilisation.

Voici un exemple d'objet **Edge** structuré avec les données extraites.

```

1   class Edge
2   {
3       public int id;
4       public int nbVoieFT;
5       public int nbVoieTF;
6       public int vitesse;
7       public string hierarchie;
8       public string nomRue;
9       public List<Point> points;
10
11      public void setEdge(int idArete, int nbVoieFT, int nbVoieTF, int
12                           vitesse, string hierarchie, string nomRue)
13      {
14          this.id = (idArete);
15          this.nbVoieFT = (nbVoieFT);
16          this.nbVoieTF = (nbVoieTF);
17          this.vitesse = (vitesse);
18          this.hierarchie = hierarchie;
19          this.nomRue = nomRue;
20      }
}

```

L'exemple ci-dessous montre la création d'un nouvel objet depuis une lecture de la base de données.

```

1  public List<Edge> getAllEdges(){
2      List<Edge> edges = new List<Edge>();
3      string sql = "SELECT * FROM aretes ";
4      command = new SQLiteCommand(sql, dbConnection);
5      SQLiteDataReader reader = command.ExecuteReader();
6
7      while (reader.Read())
8      {
9          Edge newEdge = new Edge();
10         newEdge.setEdge(Convert.ToInt32(reader["idaretes"]),
11                         Convert.ToInt32(reader["nbVoieFT"]),
12                         Convert.ToInt32(reader["nbVoieTF"]),
13                         Convert.ToInt32(reader["Vitesse"]),
14                         Convert.ToString(reader["hierarchie"]),
15                         Convert.ToString(reader["nomRue"]));
16         edges.Add(newEdge);
17     }
18     return edges;
19 }
```

Les flux automobiles constituent une partie à part. En effet, il a fallu définir les flux en entrée de la zone de simulation.

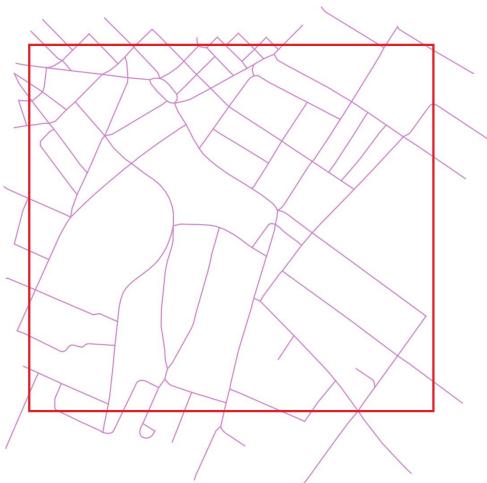


FIGURE 3.3: Représentation de la zone de simulation sur le graphe routier

Les flux utilisés sont seulement ceux sortant de la zone de simulation, comme représenté dans la figure 3.3 par un rectangle rouge.

Cette zone est définie par les coordonnées géographiques correspondant aux coins supérieurs gauche et inférieurs droits d'une zone rectangulaire.

Suite à cela, il suffit d'identifier quelles sont les arêtes sortantes de ce rectangle. Une simple comparaison avec les coordonnées des sommets d'arêtes et les coordonnées de la zone fait ressortir les flux entrants et sortants.

Les fichiers peuvent ensuite être chargés de manière correcte dans la base de données.

### 3.1.3 Application pour l'utilisation des données

Afin d'éviter aux utilisateurs qui souhaitent générer leur propre simulation de modifier le code complexe présenté dans la présente section, une application est proposée pour l'utilisation des fichiers fournis par le SITG.

Cette application contient un affichage simple et intuitif, qui permet à l'utilisateur de charger facilement ces données. Elle a accès à la base de données SQLite. Lors du démarrage de l'application, si la base de données est remplie, l'application demande à l'utilisateur s'il désire la vider complètement pour charger de nouvelles données.

Suite à cela, il est possible de choisir entre quatre types de données à charger :

- GRAPHE ROUTIER,
- VITESSE,
- FLUX D'ENTRÉE,
- SIGNALISATION.

Après avoir choisi le type, il suffit de cliquer sur "Charger fichier SHAPE" qui ouvrira une fenêtre pour sélectionner le fichier à charger.

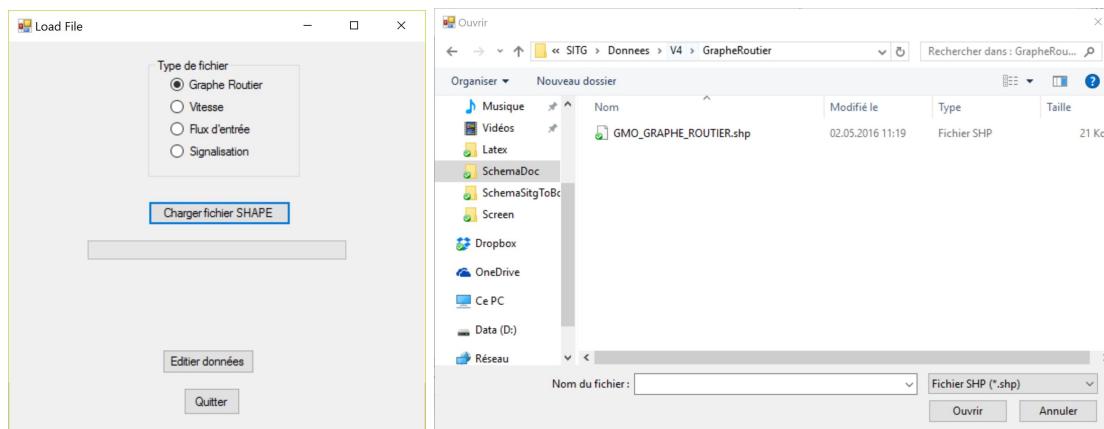


FIGURE 3.4: L'application de chargement de données Shape

Pendant le chargement les données seront traitées et stockées. Il est possible de voir l'avancement du traitement des données avec la barre de chargement. Il est ensuite possible d'éditer des données en cliquant sur "Editer données", voir figure 3.5.

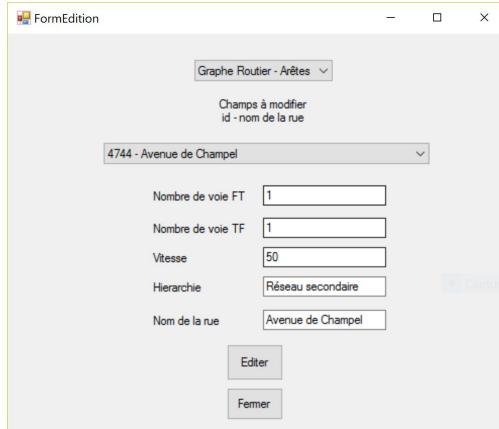


FIGURE 3.5: L'application d'édition de la base de données

Deux types de données peuvent être édités : les informations du graphe routier et les feux de signalisation.

Il suffit de modifier les champs que l'on désire, puis de cliquer sur "Editer" afin que les champs soient modifiés dans la base de données.

## 3.2 Lien SITG-UNITY

Cette section explique les traitements effectués sur les données du SITG et dans Unity pour fournir les fichiers JSON à l'application Unity. Le premier traitement consiste à lire les données de la base de données afin de les extraire au format JSON préalablement défini. Ensuite les objets Unity sont créés à partir des données contenues dans le fichier JSON.

### 3.2.1 Structure JSON

La structure d'un fichier JSON est très basique, comme vu au chapitre précédent (voir section 2.5)

Il y a quatre fichiers JSON :

- EDGES.JSON,
- VERTEX.JSON,
- FLUX.JSON,
- ROADSIGN.JSON.

Un lien existe entre chacun de ces fichiers pour permettre une bonne communication entre eux. L'illustration 3.6 montre la structure de ces fichiers JSON, ainsi que leurs liens. *Vertex* correspond au sommet du graphe, *edge* aux arêtes et *roadsign* aux panneaux de circulation.

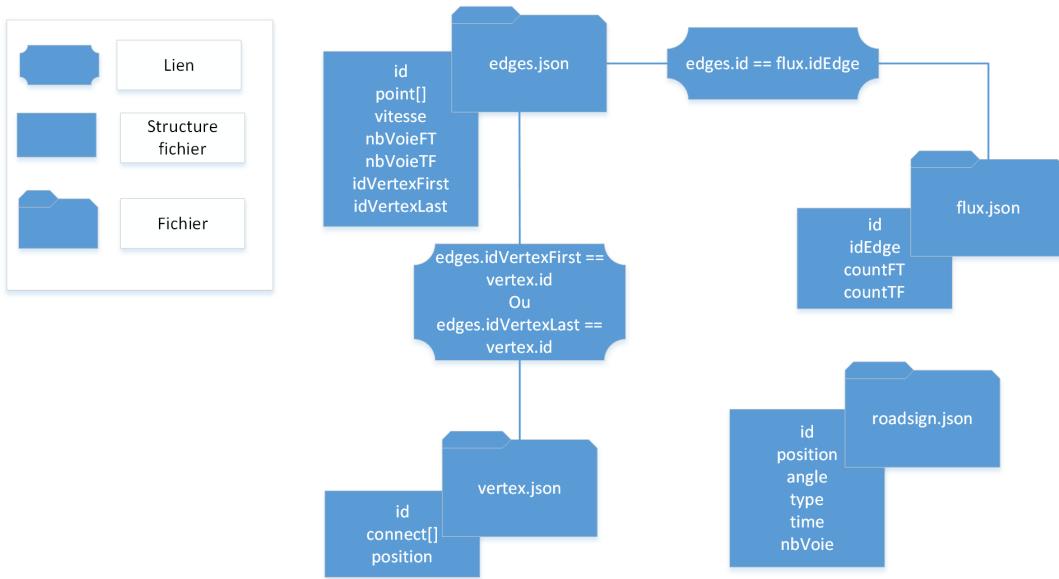


FIGURE 3.6: Communication et structure des fichiers JSON

Le SITG ne fournit pas les informations pour ROADSIGN.JSON (feux tricolores et stops). Des tests ont été effectués avec cette structure en attendant des données exploitables.

Voici un exemple d'un objet stocké dans le fichier EDGES.JSON :

```

1 {"edges": [
2     {"id": "4681",
3      "point": [
4          {"x": "2500495.7902", "y": "0.75", "z": "1116111.3762"},
5          {"x": "2500490.1099", "y": "0.75", "z": "1116100.3667"},
6          {"x": "2500488.4591", "y": "0.75", "z": "1116090.5267"},
7          {"x": "2500524.087", "y": "0.75", "z": "1116093.7373"}],
8      "vitesse": "50",
9      "nbVoieFT": "1",
10     "nbVoieTF": "0",
11     "idVertexFirst": "7191",
12     "idVertexLast": "7192"
13 }
14 
```

### 3.2.2 Conversion de la base de données vers du JSON

En premier lieu, il y a la phase d'extraction des données avant leurs conversions. Pour cela le simulateur accède à la base de données et convertit ces données en objets JSON pour la création des fichiers correspondants. Voici le diagramme 3.7 qui illustre l'association des objets aux fichiers JSON.

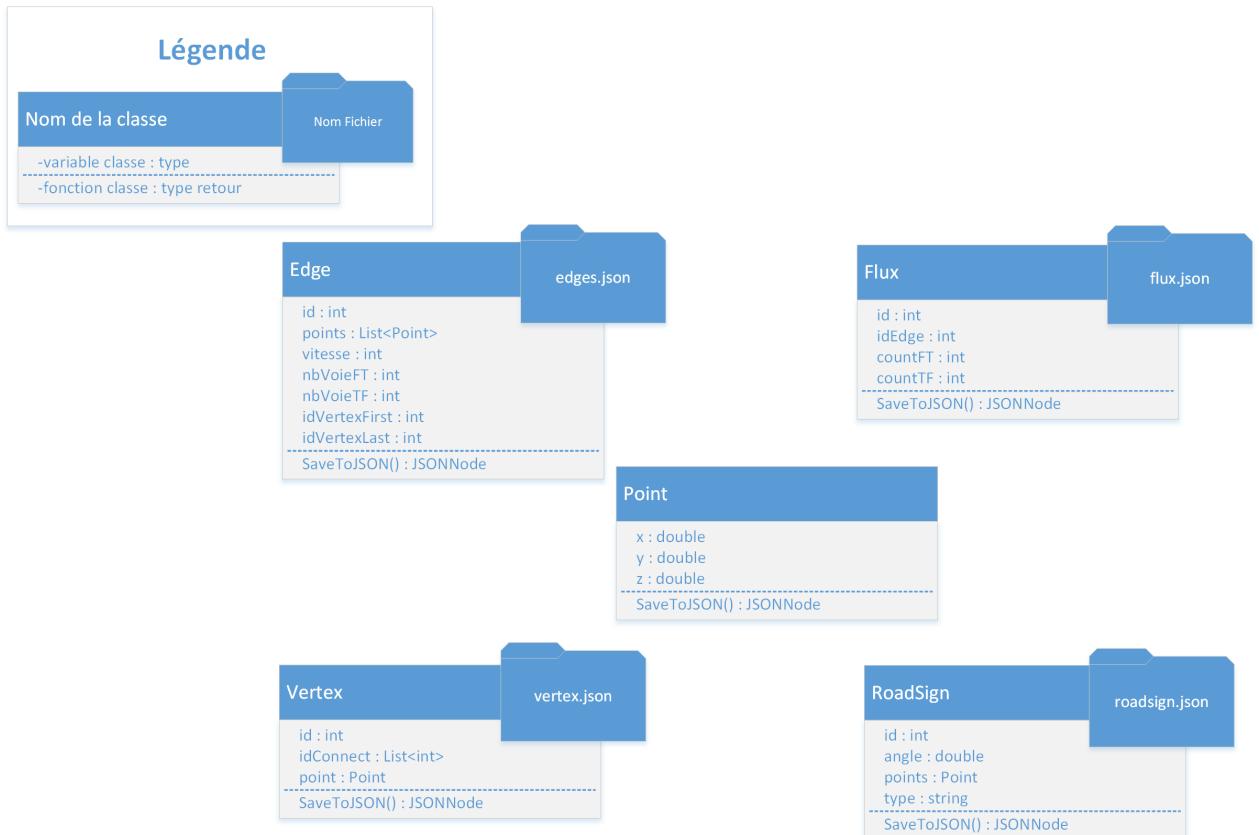


FIGURE 3.7: Diagramme des objets avec leur fichier JSON correspondant

Le diagramme 3.7 montre les liens entre les fichiers et les objets. Les objets sont composés de différents champs qui doivent être remplis. L'ensemble des objets doit être récupéré de la base et créé dans une liste avant la création du fichier (pour voir le modèle relationnel de la base de données voir 2.8).

Voici les étapes de la création des fichiers JSON :

1. ACCÉDER À LA BASE DE DONNÉES.
2. LIRE ET FILTRER LES DONNÉES ET CRÉER LES OBJETS RESPECTIFS DANS UNE LISTE.
3. ÉCRIRE LES OBJETS DE LA LISTE DANS LES FICHIERS JSON CORRESPONDANTS.

Voici un exemple de code qui récupère toutes les arêtes du graphe routier de la base de données.

```

1   public List<Edge> getAllEdges() {
2       IDbCommand dbcmd = dbconn.CreateCommand();
3       List<Edge> edges = new List<Edge>();
4       string sqlQuery = "SELECT * FROM aretes ";
5       ...
6       while (reader.Read())
7       {
8           int id = reader.GetInt32(0);
9           int vitesse = reader.GetInt32(3);
10          ...

```

```

11         Edge newEdge = new Edge(id, vitesse, points, nbVoieFT,
12             nbVoieTF, getVertexFirstByIdEdge(id),
13             getVertexLastByIdEdge(id));
14         edges.Add(newEdge);
15     }
16     ...
17     return edges;
}

```

Cette fonction utilise l'objet `Edge` pour le fichier JSON. Elle en fait une liste qui va être remplie au fur et à mesure des éléments lus depuis la base de données. Une fois tous les éléments de la base de données parcourus, la liste contenant les objets `Edge` sera retournée.

La logique de cette fonction est identique pour la création des autres objets : création de la liste d'objets, lecture de la base de données, insertion des objets dans la liste au fur et à mesure de la lecture, puis retourner la liste.

Il faut ensuite écrire ces objets dans le fichier leurs correspondant. L'objet `Edge`, par exemple, contient la fonction `SaveToJson()` qui le formate en JSON, c'est-à-dire mettre l'objet avec la structure JSON correspondante.

Voici le code qui crée le fichier `edges.json` avec une liste d'objets `Edge`.

```

1  public void insertEdge(List<Edge> edge) {
2      string pathEdge = path + "/edges.json";
3
4      StreamWriter sw = new StreamWriter(pathEdge);
5      insertDataEdge(sw, edge);
6      sw.Close();
7  }
8
9  private void insertDataEdge(StreamWriter sw, List<Edge> edge) {
10     JSONNode nodeTitle = new JSONClass();
11     JSONArray arrayEdge = new JSONArray();
12     foreach (Edge e in edge)
13         arrayEdge.Add(e.SaveToJson());
14     nodeTitle["edges"] = arrayEdge;
15     string json = nodeTitle.ToString();
16     sw.WriteLine(json);
17 }

```

Ceci est le code de lancement pour la création du fichier `edges.json`, qui fait appel aux fonctions vues précédemment.

```

1  List<Edge> edges = new List<Edge>();
2
3  myBdd = new Bdd(Application.dataPath + Config.pathBDD);
4  edges = myBdd.getAllEdges();
5  myBdd.closeConnection();
6
7  DataJson json = new DataJson(Application.dataPath + "/JSONFile");
8  json.insertEdge(edges);

```

### 3.3 Modélisation 3D

Cette section explique quels modèles sont utilisés et comment ces derniers sont initialisés.

### 3.3.1 Modèles

Les modèles 3D utilisés sont les éléments qui permettent une identification des objets composant la simulation, ainsi que leur comportement. Leurs comportements sont les scripts liés aux modèles. Par exemple, le script de la gestion des feux est associé au modèle 3D du feu.

Dans le présent simulateur, trois objets principaux le composent :

- LES FEUX TRICOLORES,
- LES STOPS,
- LES VOITURES.

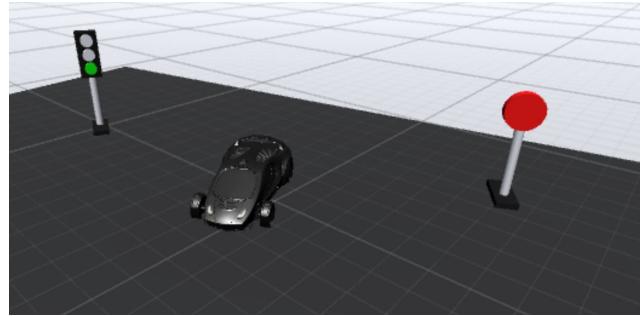


FIGURE 3.8: Les modèles utilisés

Après la création des fichiers JSON, la phase suivante est linstanciation des objets 3D à leur position définie avec les coordonnées du SITG.

Voici un exemple qui représente la route instanciée après linitialisation de celle-ci. Le graphe routier de la simulation est similaire au graphe routier des données du SITG (voir section 3.9).



FIGURE 3.9: Route simulateur, route utilisée du SITG

Certains modèles sont composés déléments non visibles lors de la simulation. Ils sont néanmoins importants, par exemple les *box colliders*. Les *box colliders* sont utilisés afin de gérer toute

sorte de collisions. Le *box collider* peut être sous deux formes distinctes : *trigger* ou non *trigger*.

Pour faire simple un élément *trigger* est considéré comme une zone de détection, une zone "non-solide", tandis que l'élément *non trigger* est considéré comme un "solide". Lorsque deux *box colliders* non *trigger* se touchent, il y aura une collision. Tandis que si un élément *trigger* entre en contact avec un élément non *trigger*, il n'y aura pas de collision.

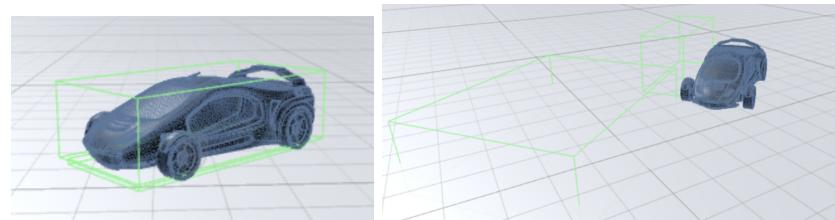


FIGURE 3.10: Voiture zone non *trigger*("solide") et zone *trigger*("non-solide")

Dans la simulation, les *box colliders trigger* servent de zone de détection. Cette zone gérera les comportements des objets impliqués.

### 3.3.2 Initialisation des modèles

L'initialisation des modèles des objets routiers aux bonnes positions est essentielle au bon fonctionnement de la simulation.

Celle-ci est divisée en quatre parties distinctes.

- PANNEAUX DE CIRCULATION ;
- ROUTE ;
- GRAPHE ;
- VOITURES.

Chacune de ces initialisations utilisent les fichiers JSON correspondants. Les panneaux de circulation, la route ainsi que le graphe routier seront instanciés avant le début de la visualisation de la simulation. Les voitures seront instanciées dans la simulation "au fil de l'eau", afin de créer un flux de voiture.

Le schéma 3.11 montre les différentes étapes ainsi que les utilisations des fichiers JSON pour instancier tous les objets de la simulation.

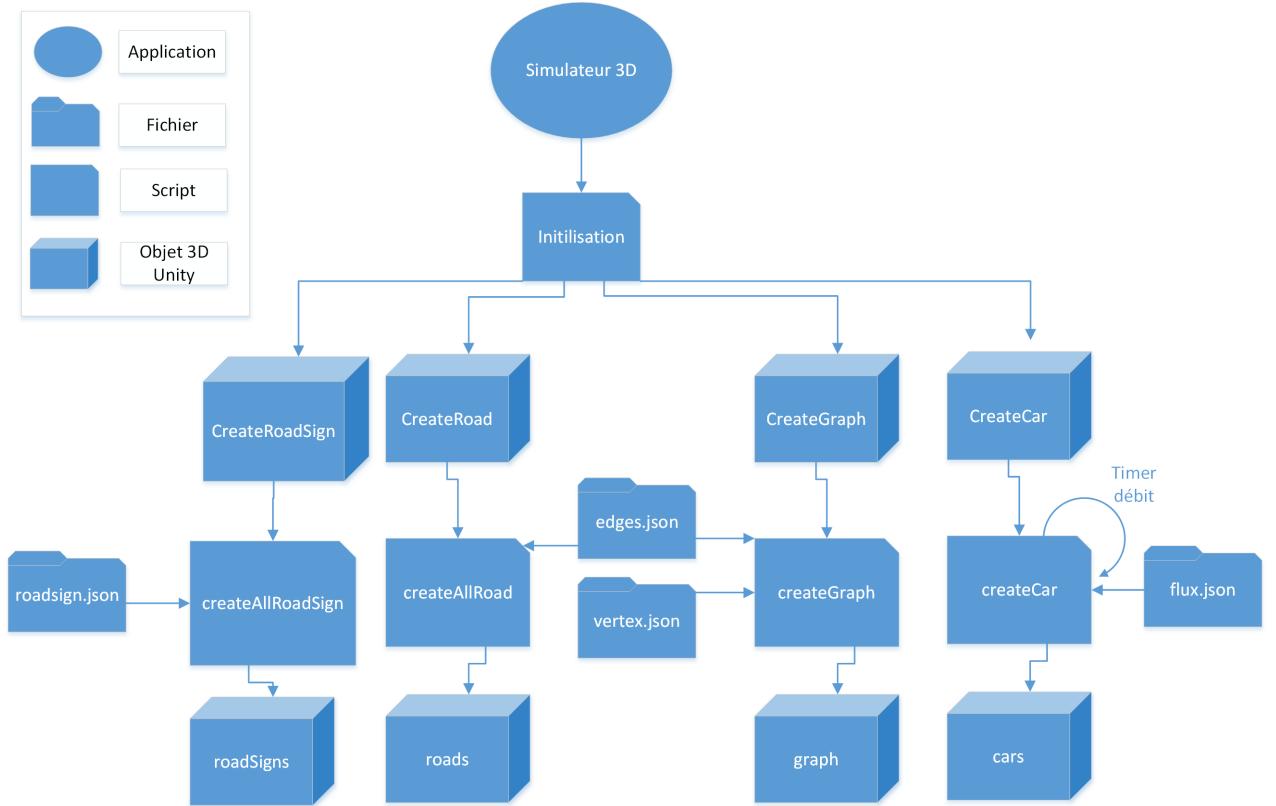
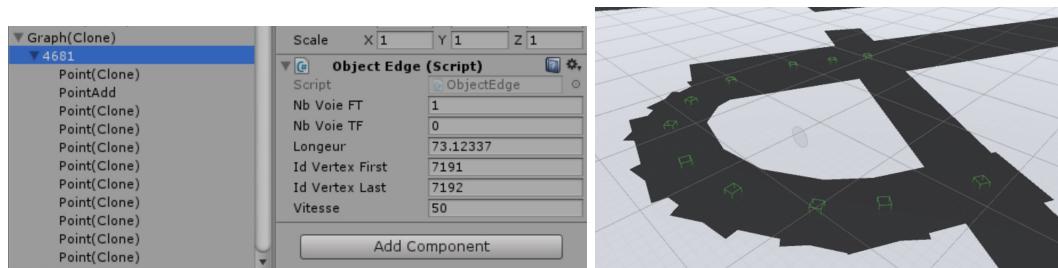


FIGURE 3.11: Schéma représentant les étapes des objets à initialiser

Les panneaux de circulation ainsi que les routes communiqueront avec les objets de la simulation uniquement via leur *box collider*.

La partie du graphe routier a une hiérarchie (parent - enfant) qui a une importance pour son utilisation. Par exemple, la hiérarchie du graphe permet un regroupement logique des éléments du graphe. Ceci en permet une meilleure lecture.

La structure de l'objet du graphe routier ainsi que la façon dont il est instancié dans la simulation est présenté par l'illustration 3.12.


 FIGURE 3.12: Visualisation de la hiérarchie de l'objet **Graph(Clone)** et des informations associées. La voie en 3D

La figure 3.12 montre que l'objet "Graph(Clone)" est le parent, il contient toutes les informations concernant les connexions d'arêtes. Le parent contient différents enfants. Ces enfants sont les identifiants des arêtes du graphe routier qui contient toutes les informations routières de ce tronçon de la route : la vitesse, le nombre de voies, les sommets connectés et la longueur. Ces identifiants contiennent eux aussi des enfants, ce sont les points (illustration 3.12 de droite). Ces points permettent de positionner le système de routes. Les voitures vont les suivre sur leur trajet.

Différents points sont rajoutés par la suite afin d'avoir des parcours de voitures plus fluides. Le simulateur rajoute des points si la distance entre le premier point et le suivant ou le dernier point et le précédent est supérieure à 10 mètres. Puis il le rajoute à une distance fixée, comme illustré sur la figure 3.13.

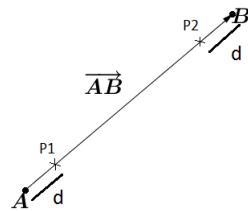


FIGURE 3.13: Illustration pour le calcul vectoriel

L'ajout des points entre  $A$  et  $B$  à une distance  $d$  se fait par calcul vectoriel :

$$\vec{AB} = B - A$$

Formule d'ajout d'un point au début d'un vecteur :

$$P1 = A + \vec{AB} * (d/\|\vec{AB}\|)$$

Formule d'ajout d'un point à la fin d'un vecteur :

$$P2 = B - \vec{AB} * (d/\|\vec{AB}\|)$$

### 3.3.3 Initialisation des voitures

Cette section va expliquer comment est généré le flux de voitures dans le simulateur. C'est une partie importante et complexe car le simulateur n'initialise pas toutes les voitures en même temps, mais il crée un flux de voitures.

Pour générer ce flux, il faut tout d'abord lire le fichier JSON `flux.json`, puis établir les tableaux de flux entrant et sortant.

Ces tableaux seront composés d'objets de type `flux` avec un champ `count` qui définit le nombre de voiture par heure :

```

1  public class DataFlux {
2      public int idVertex;
3      public int idEdge;
4      public int count;
5
6      public DataFlux(int idVertex, int idEdge, int count)
7  {

```

```

8     this.idVertex = idVertex;
9     this.idEdge = idEdge;
10    this.count = count;
11 }
12 }
```

Grâce au graphe routier initialisé au préalable, il est possible de connaître quelle arête est un flux entrant ou/et sortant grâce au sens de la voie.

Par la suite, il faut faire une répartition de ces flux sur un nombre voulu de voiture à générer.

Après la création des tableaux concernant les flux entrant/sortant, il est possible de constituer un tableau contenant tous les parcours possibles selon ces différents flux.

Un algorithme du plus court chemin sera appliqué entre chaque entrée et sortie différente. Deux algorithmes standards ont été implémentés : Floyd[3] et Dijkstra[2]. Les plus courts chemins ont été calculés avec comme poids la longueur des arêtes.

Au moment où le simulateur possède toutes les informations concernant les données de flux et les plus courts chemins entre chacune des entrées et sorties, il est possible de créer le flux de voiture en parcourant le tableau des flux d'entrée (avec un débit de flux d'entrée par minute).

Voici les étapes expliquées ci-dessus en code.

```

1  public void createCar(){
2      JSONNode jsonFlux = JSON.Parse(ReadFile(Config.pathFlux));
3      createFlux(jsonFlux);
4      TabFlux.avgFlux();
5      //Dijkstra algo = new Dijkstra();
6      Floyd algo = new Floyd();
7      foreach (DataFlux fEntrante in TabFlux.tabFluxEntrant){
8          foreach(DataFlux fSortante in TabFlux.tabFluxSortant){
9              if (fEntrante.idEdge != fSortante.idEdge){
10                  //algo.dijkstra(fEntrante.idVertex, fSortante.idVertex);
11                  algo.floyd(fEntrante.idVertex, fSortante.idVertex);
12              }
13          }
14      }
15      foreach(DataFlux fEntrante in TabFlux.tabFluxEntrant){
16          StartCoroutine(createFluxCar(fEntrante));
17      }
18  }
```

## 3.4 Comportement des Objet 3D

### 3.4.1 Voiture

Un conducteur de voiture doit toujours interagir avec l'environnement qui l'entoure. Il doit toujours garder un œil sur les autres automobilistes, les panneaux de circulation ou être attentif aux règles de circulation. Cet œil qu'un automobiliste doit garder sur la route est représenté par une zone de détection propre à une voiture dans le simulateur.

Par la suite, les conducteurs doivent se placer dans le bon sens de la voie, éviter de se retrouver à contre sens ou même au milieu d'une voie. Ainsi lorsqu'un conducteur sait qu'il devra tourner sur la droite ou sur la gauche, il se positionnera sur la bonne voie en fonction de sa pré-sélection. Le simulateur va gérer tous ces cas en utilisant la trigonométrie et des calculs vectoriels sur le

parcours défini pour la voiture.

Sur la route, il existe différents styles de conduites. Beaucoup de personnes adoptent une conduite standard, mais il y en a toujours qui ne respectent pas la limitation de vitesse ou d'autres qui ne regardent jamais leur angle mort. Le simulateur propose de mettre en place différents sortes de profil pour simuler ceci.

Voici le diagramme d'état pour une voiture

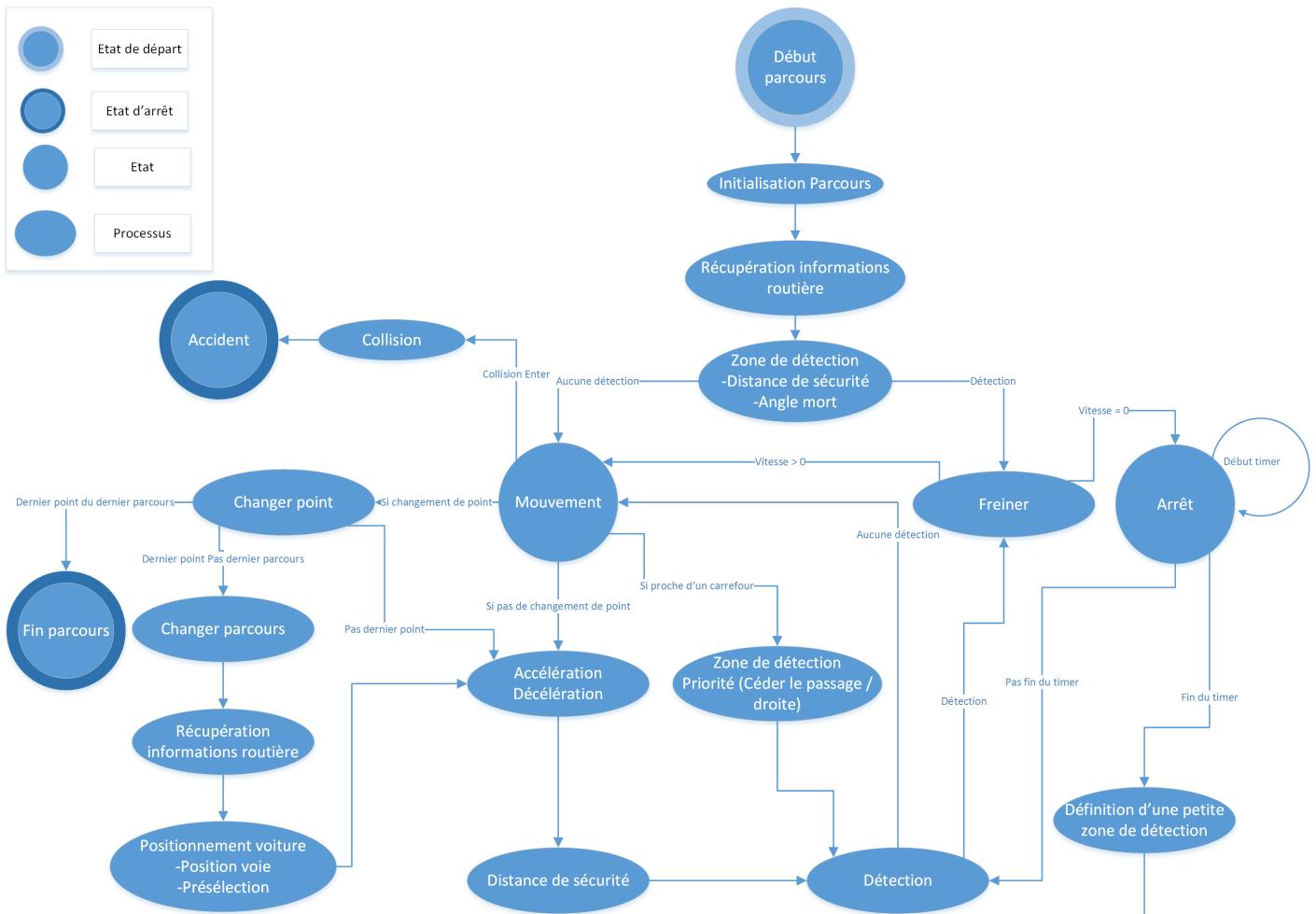


FIGURE 3.14: Diagramme d'état "voiture"

### Zone de détection

Une voiture possède des zones de détection :

- LA DISTANCE DE SÉCURITÉ ;
- LES PRIORITÉS ;
- L'ANGLE MORT.

La distance de sécurité est calculée par rapport à la vitesse de la voiture. Voici le calcul fait pour la distance de sécurité :

$$5 / 9 * \text{vitesse} + 4$$

Le calcul de distance de sécurité provient de la norme française [https://fr.wikipedia.org/wiki/Distance\\_de\\_s%C3%A9curit%C3%A9\\_en\\_France](https://fr.wikipedia.org/wiki/Distance_de_s%C3%A9curit%C3%A9_en_France), car il n'y pas de calcul simple pour la distance de sécurité en Suisse.

On additionne 4, par rapport à la taille de la voiture. Voici l'illustration 3.15 pour la distance de sécurité d'une voiture roulant à 10km/h et une voiture roulant à 50km/h.

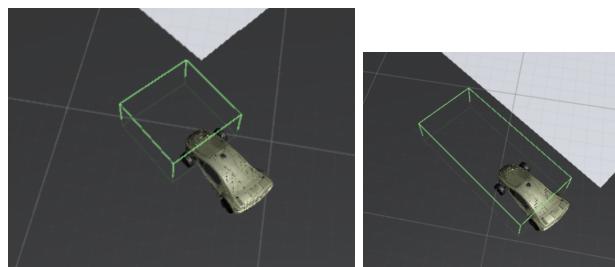


FIGURE 3.15: Distance de sécurité à 10km/h et à 50km/h

La distance de sécurité est utilisée pour détecter les différents éléments rencontrés sur son chemin. Cette zone détecte les voitures, les feux de circulation, ainsi que les stops.

Les priorités sont gérées à partir des zones de détection. Cependant cette zone se crée que pendant une certaine distance, elle est générée uniquement lorsque la voiture est à 20 mètres du centre du carrefour et disparaît lorsque que la voiture sort du centre du carrefour.

L'illustration 3.16 ci-dessous représente une voiture qui souhaite tourner sur la gauche. Comme dans le code de la route, lorsqu'une voiture coupe la voie face à lui, il doit céder la priorité à la voiture venant en face. C'est pour cela que la zone va jusqu'à la voie d'en face, ainsi que sur la voie de droite.

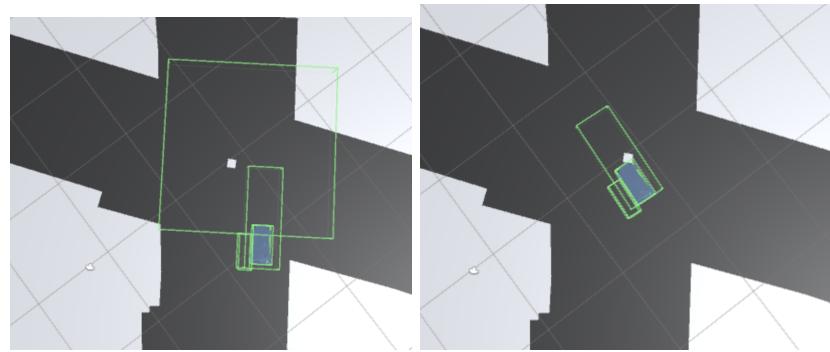


FIGURE 3.16: Zone de détection priorité

Lorsqu'une voiture va tout droit et croise une voie, il doit céder la priorité à la voiture venant de sa droite. Une voiture qui va tout droit, doit donc regarder sur la droite et voir si elle doit s'arrêter, exemple avec la figure 3.17.

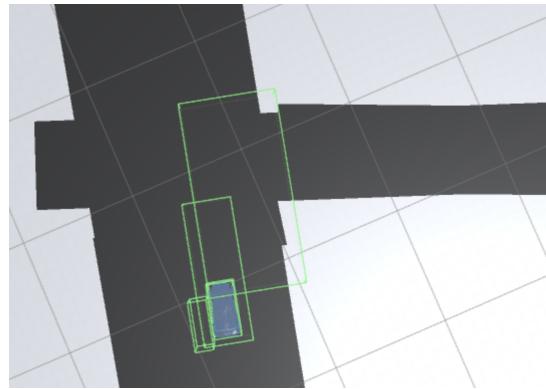
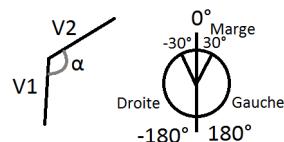


FIGURE 3.17: Priorité de droite

Le calcul réalisé pour savoir si la voiture tourne sur la gauche ou pas est un calcul d'angle entre 2 vecteurs.


 FIGURE 3.18: Exemple calcul de direction entre  $\vec{V}_1$  et  $\vec{V}_2$ 

Voici la manière pour déterminer si la voiture tourne sur sa droite ou sur sa gauche. En admettant une marge de 30 degrés pour quand elle va tout droit.

Suite à la gestion des priorités, il a fallu prévoir le cas où les voitures se retrouvent dans une situation bloquée, c'est-à-dire lorsque les automobilistes doivent faire un signe de la main pour laisser la priorité. Voici un exemple 3.19 à un carrefour dans le simulateur.

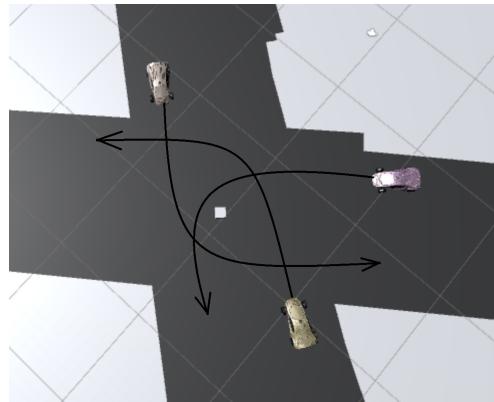


FIGURE 3.19: Situation bloquée dans un carrefour

Cette illustration 3.19 montre le cas où les règles de priorités ne peuvent pas déterminer qui passe en premier.

En réalité dans le simulateur, chacune des voitures détectent une autre voiture. Cela crée un blocage et du coup elles ne vont pas avancer. Dans ce cas-là, chacune des voitures va lancer un *timer* sur un temps tiré aléatoirement dans une fourchette (5 secondes - 15 secondes).

La première voiture à arriver à la fin du *timer* indiquera aux autres voitures qu'elle passe puis réduira sa zone de détection pour pouvoir avancer.

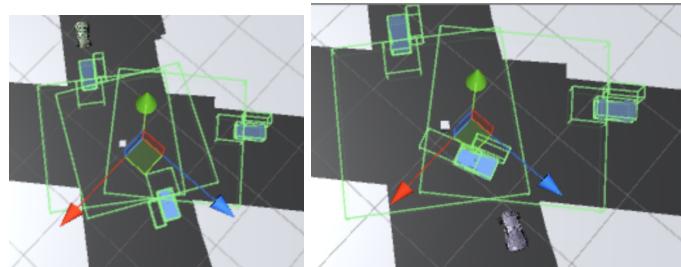


FIGURE 3.20: Toutes les voitures se détectent, puis la voiture du bas avance en ayant réduit sa zone de détection

Cependant malgré toutes ces zones de détections, il y a tout de même des accidents, mais sans danger réel. Dans le cas d'un accident les voitures accidentées deviennent rouges. Ceci permet une identification facile de ces voitures lors de la visualisation.

Pour donner un côté réaliste lors d'un accident la voiture ne disparaît pas directement. Elle disparaît après un certain temps tiré aléatoirement dans une fourchette entre 10 et 20 secondes. Ce temps d'attente permet de créer une sorte de "bouchon".

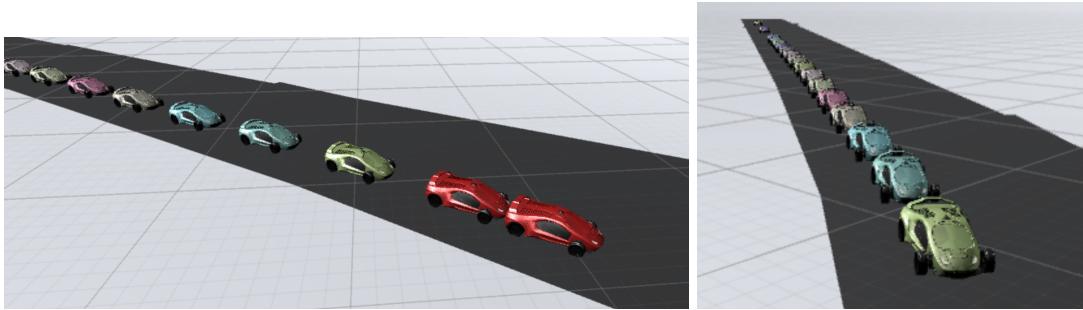


FIGURE 3.21: Exemple d'accident avec son impact sur la circulation

De plus la voiture possède une autre zone de détection. Celle de l'angle mort, qui sera détaillée par la suite.

Voici l'illustration 3.22 de l'angle mort d'une voiture. Elle se situe toujours sur sa gauche lors de son lancement.



FIGURE 3.22: L'angle mort de la voiture

Il est ensuite expliqué comment se passe le changement d'angle mort dans la partie "présélection" de cette section 3.4.1.

La gestion d'une voiture de façon automatique est une gestion extrêmement complexe, puisque même la voiture Tesla a des accidents.

## Parcours

Une voiture se déplace selon les points des arêtes du graphe routier. A chaque point atteint, elle passe au suivant. Puis à chaque point avant le sommet de l'arête elle passe à l'arête suivante et ainsi de suite.

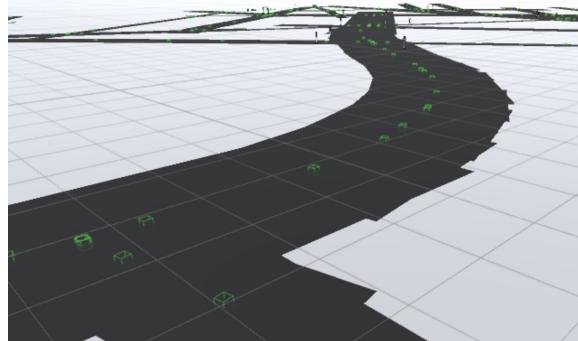


FIGURE 3.23: Points d'arêtes

L'illustration 3.23 est un exemple de points des arêtes qu'une voiture peut parcourir. Elle passera d'un point à un autre jusqu'à destination.

Chaque objet voiture possède son propre parcours. Il est défini dans un tableau qui contient tous les identifiants d'arêtes à parcourir. Ceci permet à la voiture d'avoir un point de départ et un point d'arrivée afin d'éviter que des voitures "tournent en rond" dans le quartier.

Une voiture possède 2 tableaux : l'un contenant tous les identifiants d'arêtes qu'elle doit parcourir, et l'autre ayant les points de l'arête qu'il est en train de parcourir.

Le système de segments sera expliqué dans la partie suivante. C'est la position de la voiture sur une voie.

L'illustration 3.24 permet de comprendre le système de positionnement de la voiture sur la voie. Le point de l'arête à parcourir est toujours sur la gauche de la voiture. L'illustration concerne uniquement un système de deux voies dans les deux sens.

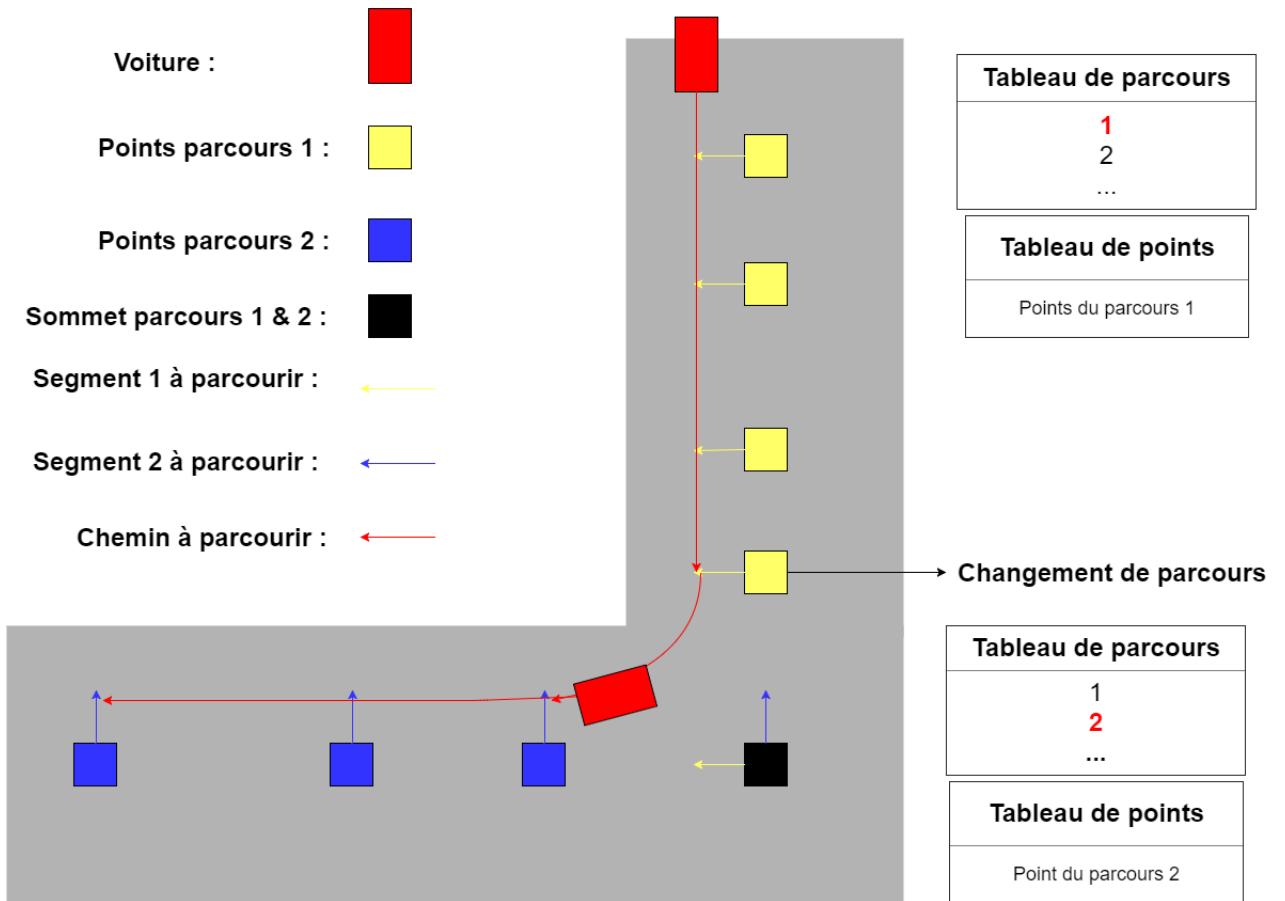


FIGURE 3.24: Parcours d'une voiture

L'illustration 3.24 montre une voiture allant du parcours 1 au parcours 2. Il est possible de constater que la voiture possède deux tableaux qui évoluent lors du changement de parcours.

Lors du changement de parcours :

- Le tableau de parcours passe au parcours suivant.
- Le tableau de points est vidé et rechargé avec les points du parcours correspondant.

#### Positionnement de la voiture

Le graphe proposé par les données du SITG ne contient pas le système de voies. Il donne juste l'information du nombre voies dans un sens et dans l'autre. Pour que les voitures évitent de se retrouver face à face, il a fallu mettre en place un système de voies.



FIGURE 3.25: Exemple de positionnement de voiture sur une voie

Dans l'illustration 3.25, on voit que les voitures ne sont pas positionnées sur l'arête du graphe, mais sur sa droite. Comme on peut le voir il y a deux voitures, chacune sur une voie différente.

Pour réussir à positionner la voiture correctement, il faut définir son décalage  $x$  et  $z$  par rapport au point de l'arête. Ce décalage doit être perpendiculaire à l'arête. Ceci fait appel aux règles trigonométriques.

L'illustration 3.26 montre quelles sont les coordonnées utiles pour le calcul.

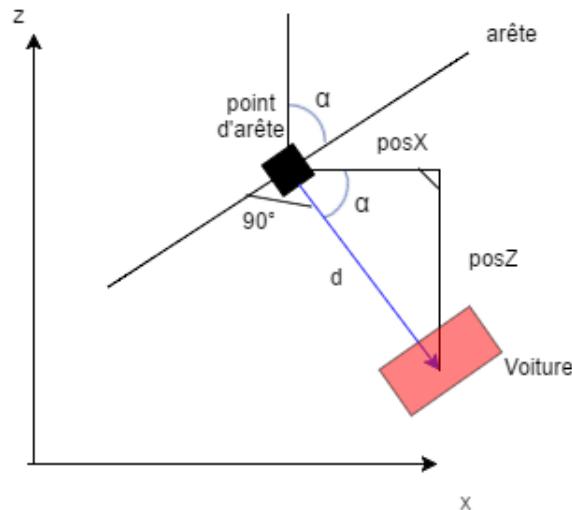


FIGURE 3.26: Illustration pour le calcul de position d'une voiture

L'angle  $\alpha$  est récupéré par une fonction standard d'Unity de l'angle de l'arête. Cet angle est en radian.

Voici les calculs de décalages pour une distance  $d$  :

$$\begin{aligned} posX &= \cos(\alpha) \cdot d \\ posZ &= \sin(\alpha) \cdot d \end{aligned}$$

La position de la voiture :

$$\begin{aligned} x_{voiture} &:= x_{voiture} + posX \\ z_{voiture} &:= z_{voiture} + posZ \end{aligned}$$

Cependant lorsque la voiture parcourt tous ces points, elle commence à avoir un comportement peu naturel. En effet, en utilisant ce système trigonométrique, la voiture fera peut-être des "aller-retours".

Afin que la voiture ait un comportement le plus naturel possible, il faut qu'elle ignore le premier et dernier point de son tableau de points, autrement dit les sommets des arêtes. Ceci permet d'avoir un comportement plus naturel et d'éviter les "aller-retour". L'ajout des points (voir section 3.3.2) lors de l'initialisation permet d'éviter le cas où l'arête ne contient que deux sommets.

La figure 3.27 montre un cas de "aller-retour" et la solution apportée.

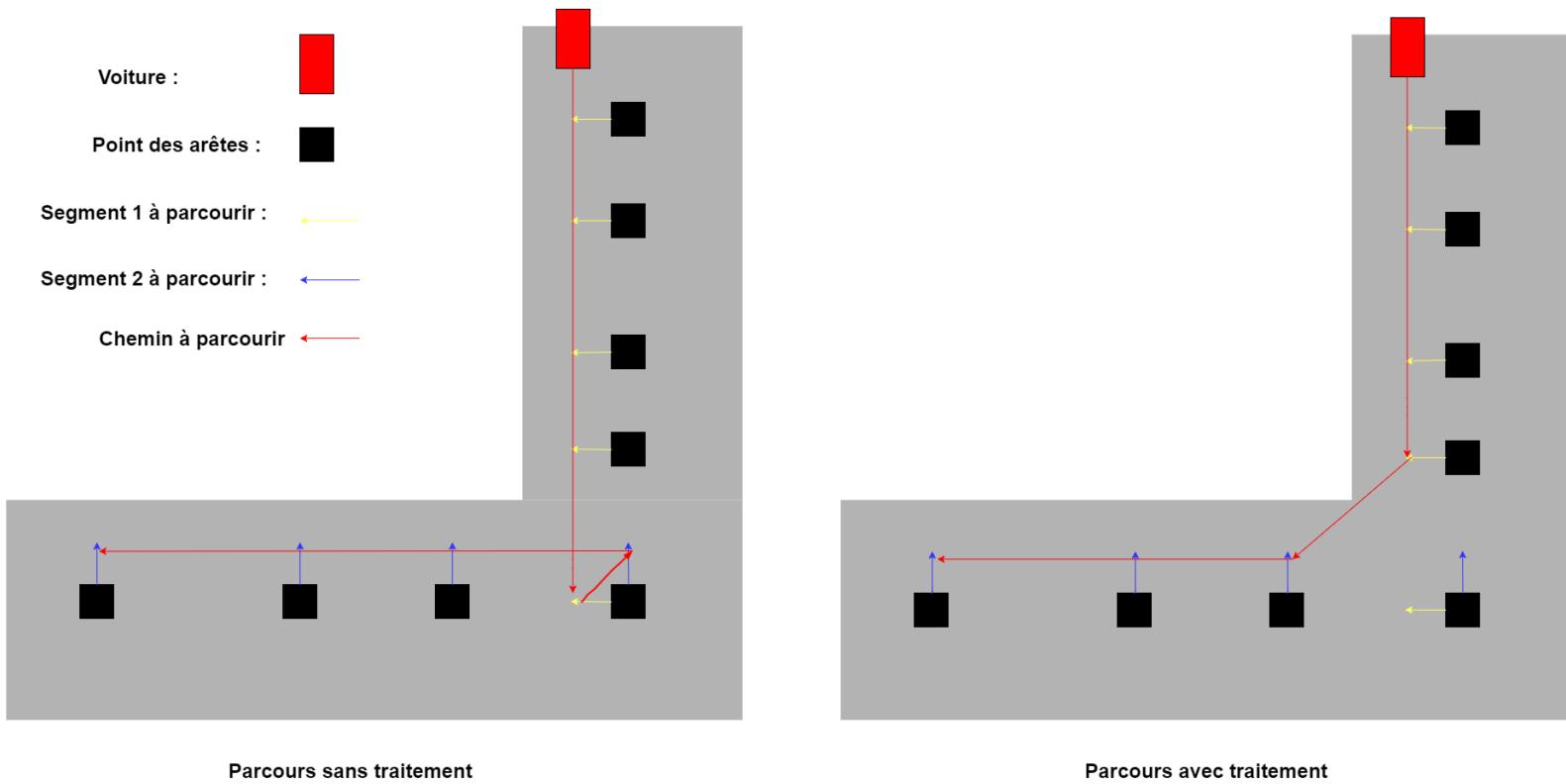


FIGURE 3.27: Exemple de parcours sans traitement et avec traitement

La voiture se présélectionne lors d'un changement d'arête du graphe. Un calcul est réalisé pour savoir si la voiture tourne à droite ou à gauche (Voir 3.4.1) à la fin de l'arête pour attribuer la voie à utiliser.

Pour le cas de la présélection, il suffit d'utiliser l'angle entre ces deux arêtes et définir si la prochaine destination de la voiture se trouve sur sa droite, gauche ou tout droit. Ensuite, il suffit de positionner la voiture sur la voie de présélection.

Si la voiture se trouve sur une route à une voie, ce traitement n'a pas lieu.

Comme vu précédemment, les voitures possèdent une zone de détection dite d'angle mort (figure 3.22). Cet angle mort change de position selon la présélection. Ainsi si la voiture prend la présélection de droite, l'angle mort sera à sa droite et inversement si la présélection est à gauche.

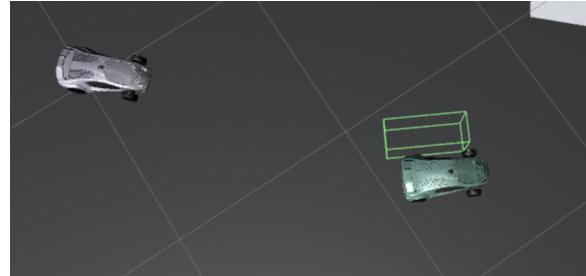


FIGURE 3.28: Exemple de présélection gauche

L'illustration 3.28 ci-dessus montre une voiture allant sur la présélection de gauche. Donc la zone de détection se situera sur sa gauche. Pour éviter des accidents lors d'un changement de voie dû à une présélection.

## Profil

Les voitures peuvent avoir différents comportements selon leur profil défini. Chacune des voitures possède un objet **Profil** :

```

1  public Profil(int acceleration, int accelerationPourcentage, int distance,
2      float timeStop, float timeWait)
3  {
4      this.acceleration = acceleration;
5      this.accelerationPourcentage = accelerationPourcentage;
6      this.distance = distance;
7      this.timeStop = timeStop;
8      this.timeWait = timeWait;
}

```

Grâce à cet objet **Profil**, il est possible de définir des comportements d'automobilistes lors de la simulation.

Voici l'explication des variables de l'objet :

- **acceleration** et **accelerationPourcentage** sont liés lors de l'utilisation de l'objet **Profil**. La variable **acceleration** va définir à combien de km/h au maximum la voiture roulera au-dessus de la limite. La variable **accelerationPourcentage** indique le pourcentage de chance que la voiture roule au-dessus de la limitation de vitesse. Ces variables permettent déjà d'avoir différentes sortes de conducteurs. On peut avoir un profil qui ne dépasse jamais les limitations de vitesse (**acceleration = 0**), un qui rarement les dépasse de quelques

km/h et un conducteur fou du volant. Si la variable `acceleration` est négative, le conducteur pourra rouler sous la limite de la vitesse maximale.

- `distance` permet de modifier la distance de sécurité, afin de représenter un automobiliste prudent en ayant une valeur positive (allongement de la zone de sécurité) ou imprudent en ayant une valeur négative (rétrécissement de la zone de sécurité). Cette valeur sera additionnée au calcul de la distance de sécurité.
- `timeStop` et `timeWait` vont interagir lorsque la voiture a un temps d'arrêt. La variable `timeStop` fixe le temps d'arrêt à un stop. Il peut être de 0 seconde comme de 10 secondes, tout dépend des conducteurs. `timeWait` est le temps d'attente lors d'un conflit entre conducteurs, par exemple lorsque différents automobilistes se retrouvent dans une situation bloquée (voir section 3.19). Ce temps correspond à des conducteurs impatients ou patients en agissant sur le temps d'attente avant de forcer le passage.

Ces profils ont pu être testés. Leur implémentation n'est pas complète dans le sens où il manque la structure de stockage des données de profil.

### 3.4.2 Panneau de circulation

Dans le simulateur, il y a deux types de panneaux de circulations les feux de signalisation et les stops. Tous deux ont des comportements bien distincts, comme dans le code de la route, ils n'ont pas les mêmes règles.

La voiture doit s'arrêter au feu s'il est rouge et peut continuer son chemin s'il est au vert ou au jaune. La voiture va donc devoir détecter si le feu est au rouge ou pas.

La règle du stop est simple. La voiture doit s'arrêter complètement à la ligne du stop, puis laisser la priorité aux voitures déjà engagées sur la voie.

#### Feu de circulation

Le comportement du feu est assez basique. Il réalise une certaine séquence pendant un temps donné. Voici un exemple de séquence possible :

- 120 secondes : 5s Rouge, 1s Jaune, 3s Vert, 1s Jaune.
- 80 secondes : 10s Rouge, 1s Jaune, 5s Vert, 1s Jaune.

A la fin de chaque séquence, il passe à la suivante ou recommence la même séquence si c'est la dernière. Le feu est réglé de manière cyclique.

Le feu possède un *box collider* afin que la voiture sache si elle peut passer ou non. Voici un exemple sur la figure 3.29. On peut voir que le *box collider* "disparaît" lorsque le feu passe au vert. En réalité le *box collider* du feu se situe sous le terrain quand il est au vert.

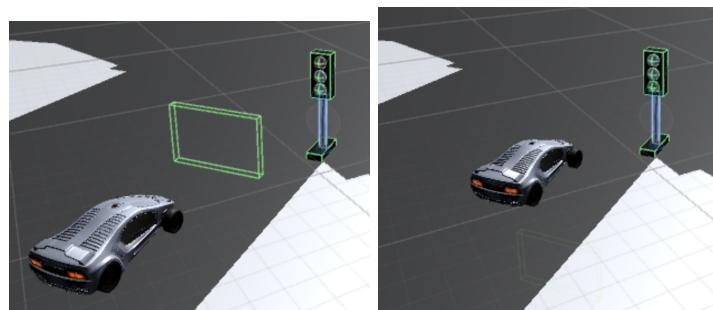


FIGURE 3.29: Exemples de feux au rouge puis au vert

L'illustration 3.29 montre comment une voiture détecte devant elle que le feu est au rouge ou au vert/jaune.

### Stop

Le stop contrairement aux feux de circulation consiste seulement en une ligne d'arrêt pour les véhicules. Ce sera à l'automobiliste d'adapter son comportement lors d'une rencontre avec un stop.

L'automobiliste va s'arrêter au stop, puis après un certain temps d'arrêt, il va continuer sa route si le champ est libre.

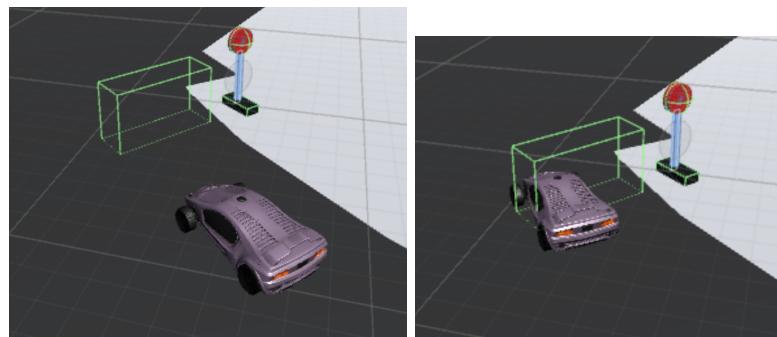


FIGURE 3.30: Comportement d'une voiture à un stop

## 3.5 Visualisation

La visualisation permet de voir la simulation pendant qu'elle tourne. Cette simulation propose trois types de visualisations :

- VUE D'ENSEMBLE
- VUE VOITURE
- VUE COMPTEUR

### 3.5.1 Vue d'ensemble

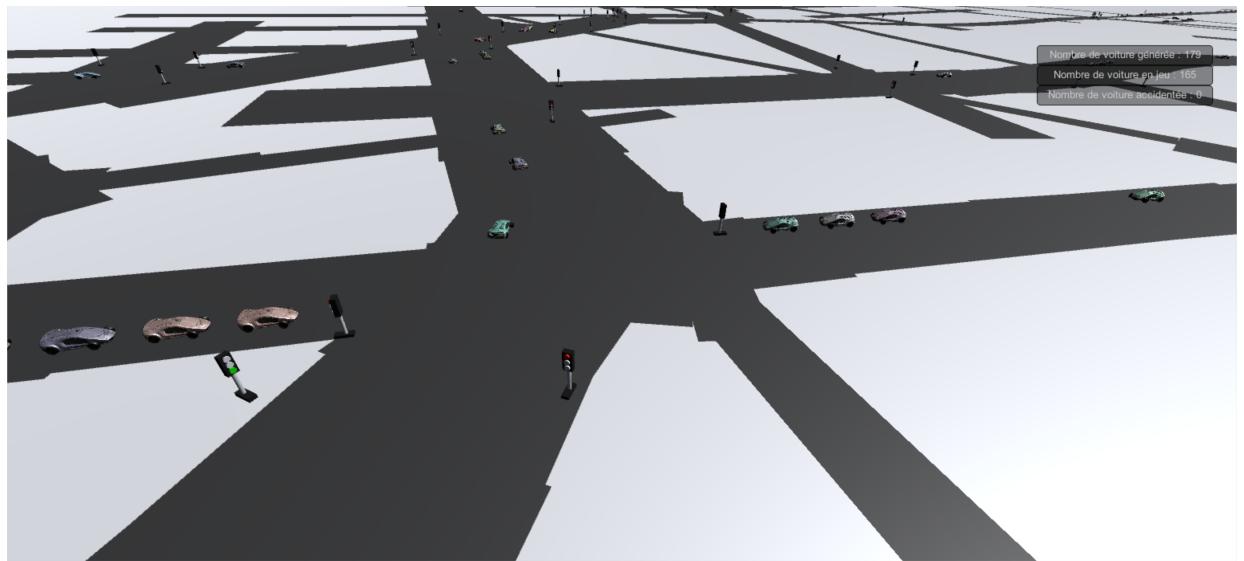


FIGURE 3.31: Exemple de vue d'ensemble

La vue d'ensemble est la vue principale. C'est sur cette vue que la simulation démarre. Avec cette vue, il est possible de se déplacer dans la simulation à l'aide des touches suivantes :

- FLÈCHES DIRECTIONNELLES entraîne un déplacement de la vue dans la simulation. Flèche droite à droite, flèche gauche à gauche, flèche haut en avant et flèche bas en arrière.
- CLIC DROIT SOURIS de la souris gère la vue, mais sans déplacement. C'est le même effet que lorsqu'on tourne la tête.
- ESPACE permet d'avoir une vue plus aérienne de la simulation.
- SHIFT sert à avoir une vue qui se rapproche du sol.

Grâce à ces contrôles, la navigation peut être assimilée à une visualisation depuis un drone.

### 3.5.2 Vue voiture

Pour accéder à la vue depuis une voiture, il suffit de faire un clic droit avec la souris sur la voiture que l'on souhaite visualiser depuis la vue d'ensemble.



FIGURE 3.32: Exemple de vue depuis une voiture

Lors de la visualisation depuis une voiture , voir 3.32, un complément d'information concernant la voiture sont affichées :

- la vitesse à laquelle elle roule,
- la vitesse de la voie,
- le nombre de voies dans un sens,
- le nombre de voie dans l'autre sens,
- la longueur de la voie parcourue.

Pour retourner sur la vue d'ensemble, il faut cliquer sur la touche "Espace".

### 3.5.3 Vue compteur

Pour accéder à la vue d'un compteur, il suffit de faire un clic sur un cube représentant le compteur depuis la vue d'ensemble.

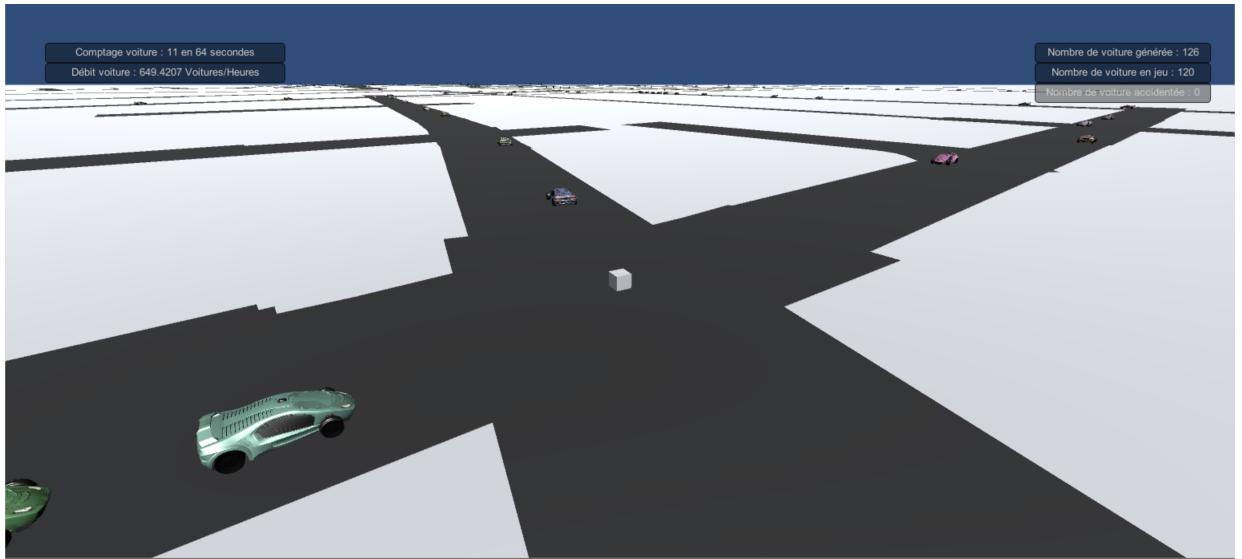


FIGURE 3.33: Exemple de vue d'un compteur

Depuis cette vue 3.33, il est possible de visualiser un exemple de compteur. Ce compteur compte le nombre de voitures passant à une certaine distance de lui. Après 60 secondes, il calcule le débit de voitures dans cette zone. Ce compteur fait office d'exemple. D'autres calculs peuvent être effectués ou d'autres styles de compteurs implémentés tels que le nombre d'accidents dans le carrefour, la pollution, ...

Depuis ce compteur, il est possible de faire un tour autour de celui-ci en appuyant sur la touche "b" pour tourner à gauche et "n" à droite.

# Chapitre 4

## Tests

Ce chapitre présente les tests principaux effectués pour s'assurer du bon fonctionnement des applications mises en place, ainsi que le traitement de données. Naturellement, des tests ont été réalisés au fur et à mesure de l'avancement du projet et lors de l'implémentation des fonctionnalités.

### 4.1 Tests unitaires

#### 4.1.1 Base de données

Durant la visualisation et l'analyse des données depuis QGIS, il est possible de connaître, en allant dans l'onglet "Métatdonnées" de la couche, le nombre de données.

Ensuite, après avoir noté le nombre d'éléments qu'on va insérer dans la base de données, il suffit de faire une requête SQL qui compte le nombre d'éléments insérés dans la table correspondante.

Le graphe routier utilisé pour la simulation contient 180 données. Ensuite il faut vérifier dans la table **aretes** de la base de données le nombre d'éléments qui ont été insérés avec une simple requête SQL comme celle-ci :

```
SELECT COUNT(*) FROM aretes
```

qui retourne le nombre de composants dans la table **aretes** laquelle possède toutes les informations routières.

#### 4.1.2 Fichier JSON

Suite aux différents tests réalisés sur la base de données, il est possible de savoir le nombre d'éléments à instancier dans la simulation. Ce nombre peut être utilisé afin de savoir si le nombre d'éléments à créer est correctement stocké sur les fichiers JSON. Étant donné que les fichiers JSON contiennent les informations concernant les objets de la simulation, c'est avec ces fichiers qu'il va falloir comparer les éléments.

En comptant le nombre d'éléments qui seront instanciés dans ces fichiers, il est possible de savoir si le nombre d'objets routiers est correctement stocké. On peut ainsi avoir une idée de si la phase de conversion des extractions de la base de données vers ces fichiers s'est bien déroulée. Voici le code permettant de faire cette comparaison de comptage.

```

1  public bool checkNbElement( int cmp_cpt, string fileName, string objectJson )
2  {
3      JSONNode json = JSON.Parse(ReadFile("/JSONFile/" + fileName));
4      for (int j = 0; j < json[objectJson].Count; j++) {
5          if (j == cmp_cpt) return true;
6      }
7  }
8
9 }
```

### 4.1.3 Éléments routiers

Pour vérifier si les éléments routiers sont bien instanciés aux positions correspondantes aux données provenant du SIG, une comparaison doit être faite entre les positions dans la simulation et la position stockée.

Voici un script qui retourne vrai, si les éléments ont été instanciés au bon endroit dans la simulation en comparant les positions.

```

1  public bool createRoad( string nameObjectUnity, string fileName, string
2  nameObjectJson )
3  {
4      bool good = false;
5      JSONNode json = JSON.Parse(ReadFile("/JSONFile/" + fileName));
6      GameObject[] gos;
7      gos = GameObject.FindGameObjectsWithTag(nameObject);
8      for (int i = 0; i < json[nameObjectJson].Count; i++){
9          double x = Convert.ToDouble(json[nameObjectJson][i]["position"][0]) -
10             Config.x_0 ;
11         double y = Convert.ToDouble(json[nameObjectJson][i]["position"][1]);
12         double z = Convert.ToDouble(json[nameObjectJson][i]["position"][2]) -
13             Config.z_0;
14         Vector3 posCmp = new Vector3(x,y,z);
15         foreach (GameObject go in gos){
16             if (posCmp == go.transform.position) {
17                 good = true;
18                 break;
19             }
20             if (!good)
21                 return false;
22         }
23     }
24 }
```

## 4.2 Tests d'intégration

### 4.2.1 Voiture "test"

L'utilisation d'une voiture test pendant la simulation a permis de tester et de vérifier les comportements des voitures. La voiture test n'a pas été instanciée par le simulateur.

C'est une voiture qui n'a aucun comportement et peut être placée n'importe où dans le simulateur.

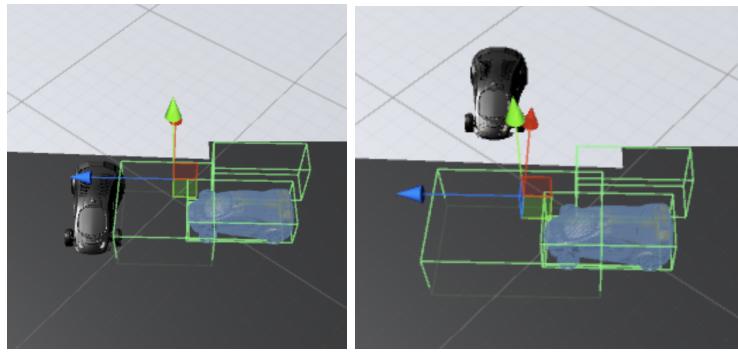


FIGURE 4.1: Exemple de test d'arrêt de voiture avec une voiture test

Les tests tels que la gestion des collisions et la gestion de conflit ont pu être réalisés en positionnant la voiture de manière à générer ces cas comme illustré dans la figure 4.1.

#### 4.2.2 Comportement voiture

Les tests des comportements des voitures sont surtout visuels. Une modification de certaines données du simulateur permet d'analyser si le comportement est correct.

Les tests de la gestion des priorités dans un carrefour ont pu être réalisés en modifiant les données de flux dans le fichier `flux.json`.

En modifiant les données de flux, il est possible de créer des flux d'entrées à des endroits spécifiques afin d'avoir une concentration de voitures dans un carrefour. Ceci a permis d'analyser les comportements des voitures dans un carrefour sans signalisation.

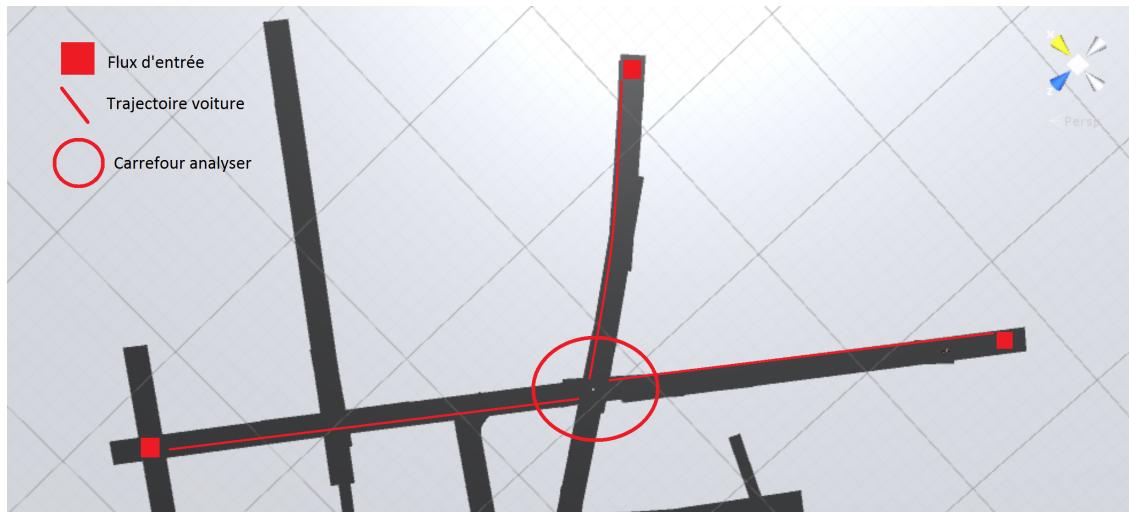


FIGURE 4.2: Exemple de test de carrefour

L'illustration 4.2 montre comment le fichier `flux.json` a été modifié afin d'avoir le plus de conflits possibles à un carrefour choisi au préalable.

Les tests du choix de la présélection des voitures furent réalisés selon le même fonctionnement, en modifiant les données de `flux.json`.

Il suffit tout simplement de créer un flux d'entrée sur un tronçon de route contenant plus d'une voie, puis des flux de sortie afin de créer différents parcours pour analyser le choix des présélections effectuées par les voitures.

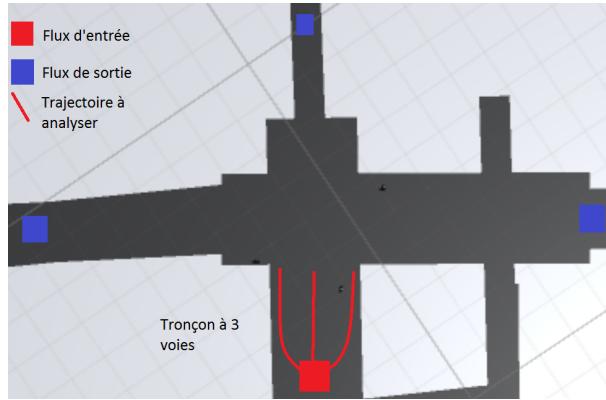


FIGURE 4.3: Exemple de test de présélection

Avec des parcours définis comme dans l'illustration 4.3, il est possible de visualiser si la voiture s'est correctement présélectionnée sur le tronçon.

Le test du bon positionnement de la voiture par rapport à l'arête (voir section 3.26) a été effectué de façon unitaire et manuelle sur les différents cas.

La vérification du bon fonctionnement du calcul du chemin le plus court a été réalisée en comparant les résultats des algorithmes de Dijkstra et de Floyd. De plus des analyses de parcours ont été effectués pour vérifier que le parcours sélectionné était bien le plus court, et que la distance correspondait au calcul. En changeant la distance de certaines arêtes, le parcours optimal est modifié.

## 4.3 Tests utilisateur

Les tests utilisateurs n'ont pas pu être réalisés, mais voici quelques idées sur la démarche à suivre.

### 4.3.1 Nouvelle base de données SIG

Le test afin de savoir la complexité de l'intégration d'une nouvelle base de données SIG doit être réalisée par un panel d'informaticiens.

Il leur sera demandé de comprendre la structure du projet.

Puis il leur faudra adapter leur base de données avec les objets proposés par l'application.

Par la suite il sera noté les difficultés rencontrées, les incompréhensions, les zones d'ombres ainsi que les points forts.

### 4.3.2 Utilisation des différentes applications

Ce test consiste à demander à différents ingénieurs en mobilité d'utiliser les applications mise en place afin de réaliser la simulation.

Différentes données seront mise à disposition afin qu'ils puissent les charger à travers l'application, puis à la fin du remplissage de la base de données, ils devront lancer la simulation. Lors de la simulation, il leur sera demander d'analyser les différents comportements à des carrefours spécifiques afin de voir si la visualisation est intuitive.

Un questionnaire à remplir leur serait demandé à la fin de la simulation, afin de noter les difficultés et facilités rencontrées tout au long des tests.

### 4.3.3 Utilisation par des personnes lambdas

Un panel de personnes venant de tout horizon sera sélectionné afin d'effectuer des tests de visualisation d'une simulation donnée. Il leur sera demandé de se balader dans la simulation avec les différentes touches.

A la fin de la simulation, un sondage serait effectué pour l'interface et l'utilisation afin de noter les difficultés rencontrées.

# Chapitre 5

## Discussions

### 5.1 Problématiques

Au fur et à mesure de l'avancement de ce projet plusieurs contraintes se sont présentées. Les contraintes principales étaient dues aux données fournies par le SITG.

Lors de la phase d'identification des données du SITG utiles à la réalisation, il est apparu que des données manquaient ou étaient inexploitables dans le cadre de ce projet.

#### 5.1.1 Flux de voitures

Les données concernant les flux ont posé des problèmes, notamment les deux fichiers suivants :

- OTC\_COMPTAGE\_TRAFFIC
- OTC\_PLAN\_CHARGE\_TRAFFIC

Ces deux fichiers contenaient un comptage routier fort intéressant pour la gestion des flux de voitures. Cependant le fichier OTC\_COMPTAGE\_TRAFFIC contient des informations de comptage des jours ouvrables (lundi - vendredi) ou une moyenne sur toute la semaine associé à un point du graphe. Ces informations ne sont pas liées aux sommets ou aux arêtes du graphe routier. D'une part, ce fichier ne fournissait pas assez d'informations pour qu'il soit exploitable, et d'autre part il contenait trop peu de points de comptage sur le graphe.

La première piste abordée était de se baser sur les flux des grands axes où des points de comptage étaient positionnés. On aurait pu essayer d'estimer la répartition des flux sur les axes connectés ou proches.

Le second fichier de données OTC\_PLAN\_CHARGE\_TRAFFIC est représenté sous forme de graphe. Malheureusement ce graphe n'était pas superposé et n'avait aucun lien avec le graphe routier de base, voir illustration 2.1 de gauche. La piste abordée pour résoudre ce problème était de mapper les deux graphes afin d'obtenir des flux sur le graphe routier.

La troisième piste envisagée était d'utiliser l'application réalisée pour l'insertion et l'édition des données dans la base de données. Ceci consistait à ajouter manuellement les différents flux à partir de cette application, en proposant les arêtes du graphe des SITG sortant de la zone à simuler (voir figure 3.3).

Pour finir aucune des trois pistes n'a permis de résoudre le problème de façon satisfaisante. M. Dubois a eu la gentillesse de fournir un graphe de flux superposé au graphe routier, ce qui a permis de débloquer la situation et d'avoir les informations de flux exploitables pour la suite du projet.

### 5.1.2 Panneaux de signalisation

Plusieurs fichiers concernant les panneaux de signalisation ont été traités. Cependant dans chacun des fichiers analysés, il manquait des informations. Soit les informations n'étaient pas assez précises, soit il n'y avait aucune coordonnée des positions réelles des feux.

Le fichier OTC\_SL\_EQUIPEMENT\_BOITEFEUX a été utilisé pour la réalisation de ce projet, car ce fichier fournit des détails concernant les feux, tels que la position et la direction du feu.

Cependant lors de l'intégration des feux de circulation, l'information manquante concernant le lien entre la voie et le feu fut problématique.

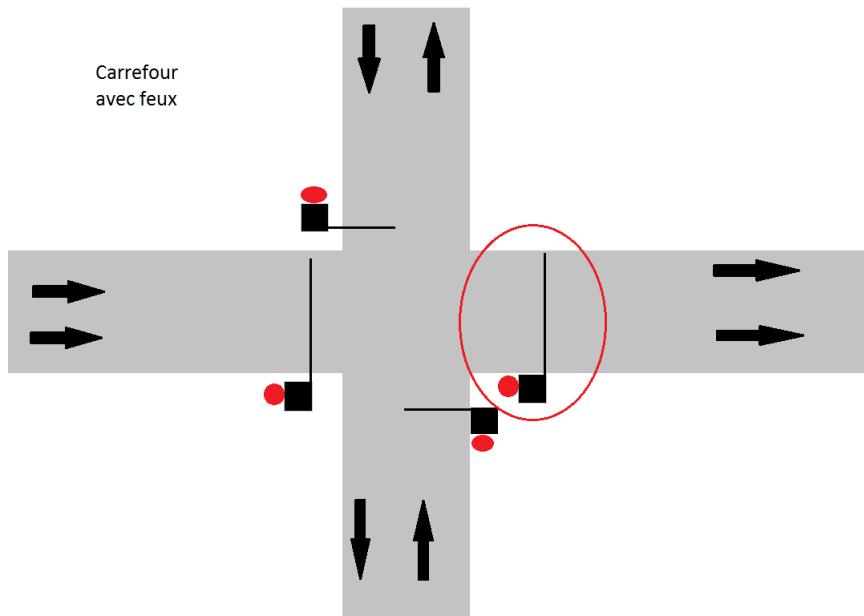


FIGURE 5.1: Illustration problème des feux

L'illustration 5.1 montre des feux qui posent leur ligne d'arrêt à leur droite selon le nombre de voies. Cependant avec cette méthode, certains feux n'auront pas leur ligne d'arrêt sur la bonne voie. Par exemple le feu entouré en rouge devrait avoir sa ligne d'arrêt avec le feu en face de lui. La seule solution pour le moment est de positionner la ligne d'arrêt manuellement.

Une solution est de supprimer les feux gênant ou les doublons dans un carrefour en utilisant leur angle. Il est possible de définir dans un carrefour un regroupement de feux ayant le même angle et supprimer les feux les plus distants de l'arête liée au feu.

Si les données de feux étaient liées à l'arête correspondante, cela permettrait de régler le problème.

Par la suite aucune donnée proposée par le SITG ne contient les séquences des feux. Donc à moins de rentrer manuellement tous les feux, il faudrait créer un algorithme qui calcule les séquences des feux, afin d'éviter que deux feux d'un carrefour se retrouvent au vert en même temps. A moins que les données correspondant à la séquence ou type de feux puissent être récupérées.

Le problème des données manquantes ou inexploitables concerne les panneaux de circulation aussi ("céder le passage" et "stop"). Dès que les informations seront mises à disposition selon un format exploitable, comme les feux liés à l'arête correspondante, la gestion des stops et des céder de passage sera fonctionnelle. La gestion des stops est en effet déjà implémentée et le céder le passage est une simple gestion de zone de collision à faire évoluer.

### 5.1.3 Graphe routier

Le graphe routier est presque complet. Cependant il ne donne aucune indication concernant les voies de présélection et les directions possibles depuis une voie. Voici une illustration 5.2 du problème des présélections.

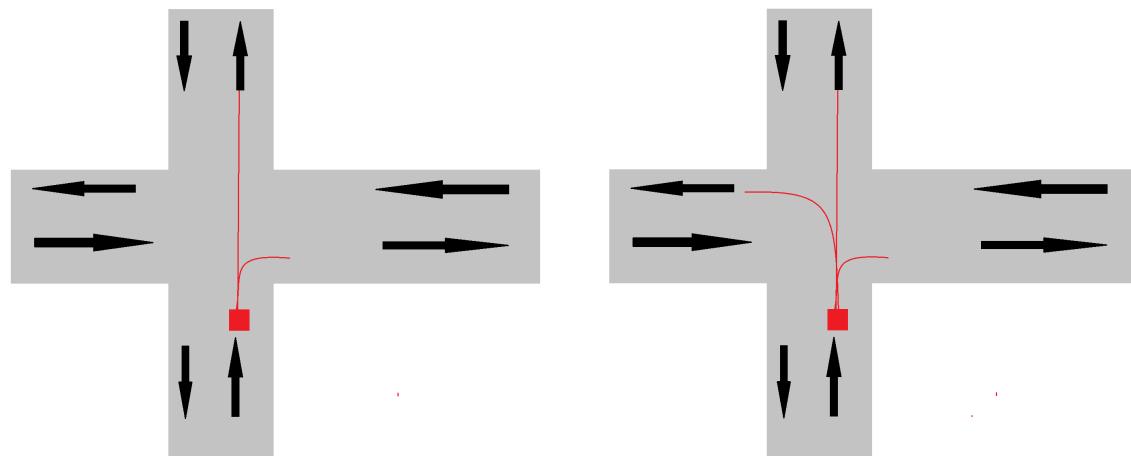


FIGURE 5.2: Illustration du problème des présélections

Le cas 5.2 montre à gauche un carrefour où les directions que la voiture est autorisée à prendre dans la vraie vie, et à droite les directions que la voiture est autorisée à prendre dans le graphe routier par manque d'informations.

Les voies de présélection ne sont pas présentes dans les données du graphe routier. Une voie pour tourner seulement à gauche peut être utilisée par une voiture voulant aller tout droit par exemple. Pour résoudre ce problème, le système de présélection a été mis en place.

De plus, le graphe ne présente pas chacune des voies, mais seulement une arête en indiquant le nombre de voies. Donc pour chaque point la voiture doit calculer la position à avoir afin de se placer sur la bonne voie. Ceci peut devenir lourd en termes de calcul et dégrader la performance

s'il y a un nombre important de voitures générées. Il serait intéressant d'avoir un graphe contenant les différentes voies, afin de ne pas à avoir à calculer la position de la voie.

## 5.2 Alternatives

D'autres alternatives auraient pu être envisagées au départ pour aboutir au résultat attendu de ce projet. Cette section va en présenter quelques-unes.

### 5.2.1 Base de données PostGreSQL

Lors de la réalisation du projet, une base de données SQLite a été utilisée de par sa simplicité d'installation et ses performances. L'avantage majeur d'une base de données sous ce format est qu'elle est directement intégrée au programme.

Cependant la question de l'utilisation d'une base de données PostGreSQL s'est posée. L'extension PostGIS qui utilise la notion de géométrie, aurait pu simplifier la manipulation des données venant d'une base de données SIG.

La manipulation des données géographiques aurait été intéressante s'il y avait des traitements de données tels que des calculs liés aux coordonnées. Cependant ces calculs n'apparaissent pas assez souvent pour voir un réel avantage d'utiliser une telle base de données. Ainsi, la base de données SQLite a été choisie étant donné que celle-ci sert plutôt d'intermédiaire pour le stockage de données.

### 5.2.2 Format de données JSON

Le format JSON fut intéressant à utiliser car sa compréhension et sa gestion sont simples.

Le format XML a été envisagé. Cela permettrait d'avoir certainement le même résultat qu'avec un fichier JSON. Le format XML est aussi plus volumineux. Le XML propose plus d'outils que le JSON, qui n'ont pas de réelle utilité.

XML aurait été intéressant à utiliser dans le cas où il aurait fallu faire de réels traitements. Cependant, il s'est avéré que cette structure serait utilisée seulement pour stocker et récupérer les données géographiques. Donc il fut plus simple et économique en termes de mémoire d'utiliser du JSON.

# Chapitre 6

# Conclusion

## 6.1 Bilan

L'objectif du projet *Simulation de mobilité urbaine 3D avec Unity* est de réaliser une simulation de trafic avec des données provenant d'une base de données SIG et avec le moteur de jeu Unity 3D.

Le riche catalogue du SITG a permis de récupérer un certain nombre de données pour la réalisation de la simulation. Cependant malgré la richesse des informations un certain nombre de données n'ont pas pu être exploitées car il manquait des liens entre les données.

Suite à l'analyse et à la collecte de ces données, une base de données ainsi que son traitement à travers une application tierce pour l'alimenter, a été mise en place afin de découpler la partie des données avec l'application.

Ainsi l'application réalisée avec Unity 3D possède toutes les informations afin de réaliser une simulation de mobilité urbaine avec une gestion réaliste des comportements d'un trafic routier.

Voici les différents éléments que propose l'application finale :

- Une analyse du catalogue de données SITG à l'aide de l'outil QGIS a été réalisée ainsi qu'une récupération de différentes données exploitables. Lors de cette analyse une identification des données inexploitables ou manquantes pour la simulation a été effectuée.
- Une architecture a été définie pour que l'application soit indépendante des données afin de permettre de séparer tout lien direct entre des données et la simulation.
- Une base de données a été créée contenant les différentes informations routières afin de découpler l'application Unity avec les données SITG.
- Une application tierce a été réalisée pour permettre d'alimenter et d'éditer de manière simple et intuitive la base de données.
- Une structure de fichier JSON a été définie pour le stockage de données propre à l'application de simulation.
- Des objets routiers composant un carrefour ont été modélisés en 3D (routes, feux, stop, voitures).
- Des visualisations sont proposées par le simulateur : vue dirigée par l'utilisateur, vue depuis une voiture choisie, vue d'un compteur. Des informations sont à disposition depuis ces vues.

- Un système de séquence de feu permet de gérer ses états (rouge, jaune, vert).
- Un profil d'automobiliste permet de gérer son comportement vis à vis de sa gestion des informations routière, de l'accélération, de la distance de sécurité, des priorités, du chemin le plus court, du positionnement sur la bonne voie, des présélections, des stops, des feux, des situations de blocages entre automobilistes.
- Différents comportements d'automobilistes pour une simulation réelle ont été testés et mis en place : chauffard, conducteur prudent ou trop lent.

Il manque des données afin d'avoir une simulation complètement réaliste. Une partie des informations est exploitable, une autre non.

Malgré les données incomplètes une simulation a quand même pu être faite en réalisant un système de voies, de présélection, de feu, de stop et de comportement d'automobiliste.

La simulation est incomplète à la fin du travail de Bachelor à cause d'un manque de données ou d'informations :

1. Information d'une intersection où le conducteur ne peut pas accéder à une voie est manquante.
2. Les panneaux de circulation sont manquants tels que le stop et le cédez-le-passage.
3. Il manque les ligne d'arrêts des feux, stops et cédez-le-passage.
4. Les séquences des feux ne sont pas mises à dispositions.

## 6.2 Améliorations

Il y a différentes sortes d'améliorations possibles : visuelle, d'optimisation, de comportement, de nouveaux types de véhicules.

### Tests

Une batterie de tests devra être effectuée afin de valider le développement de l'application. Des tests de charges pourront montrer la configuration requise pour utiliser l'application dans les meilleures conditions.

### Visuel

Comme amélioration visuelle, il serait intéressant de modéliser le quartier qu'on simule avec les bâtiments, afin de mieux se retrouver lors de la visualisation de la simulation. Cependant une modélisation de tout le quartier risque de poser de nouveaux problèmes. La simulation risque de devenir plus lourde en rajoutant tout un quartier avec les bâtiments. Il faudra faire en sorte que les bâtiments soient transparents dans certains cas, afin de ne pas gêner la visualisation du trafic routier.

Une meilleure modélisation des voies : pour le moment les lignes blanches ne sont pas présentes et ceci permettra d'avoir une meilleure compréhension des voies. Les routes sont représentées par des zones rectangulaires : il faudrait les "lisser".

Une modélisation de type de véhicules distinct devra être effectuée. Dans la simulation, il n'y a qu'un seul type de voiture présent, seule la couleur change. Il serait intéressant de mettre en place des véhicules de différents types : véhicules longs, courts, larges, des véhicules ayant des accélérations variées, etc.

### Optimisation

Il faudrait avoir différentes sortes de feux, intelligents, statiques ou calculés à partir d'un algorithme. Ceci permettrait d'optimiser le comportement des véhicules et aiderait à analyser la fluidité du trafic.

### Comportement

La mise en place du dépassement des véhicules, lorsque le véhicule de devant roule trop lentement, par un changement de voie puis un dépassement suivi d'un rabattement, serait souhaitable. Ceci pourrait être réalisable puisque que les voitures n'ont pas une voie fixe attribuée. Il suffira de changer la distance à l'arête.

### Différents objets du trafic

Dans une simulation du trafic urbain, il serait intéressant d'incorporer les deux roues. Ceux-ci rouleraient de manière à se faufiler entre les voitures ou à se mettre à plusieurs sur la même voie. Ceci permettrait de comparer le trafic avec ou sans deux roues.

### Éléments d'analyse

Il serait intéressant de mettre en place de nouveaux indicateur pour améliorer l'analyse routière :

- Le temps d'attentes des voitures, le temps cumulé à l'arrêt, avec une moyenne par carrefour indiquera les tronçons pénalisant la fluidité du trafic.
- Le calcul du temps de parcours par voiture et par flux entrant/sortant pourra générer un tableau de données contenant une moyenne de temps entre chaque entrée et sortie selon les créneaux horaires ou selon les séquences de feu.
- Calculer le débit des arêtes, pour obtenir le nombre de voitures ayant parcouru l'arête en cumulé et en moyenne, permettra d'obtenir les arêtes les plus utilisées.

## 6.3 Perspectives

Les comportements automobilistes mis en place permet de se rapprocher du comportement "humain", afin d'avoir une meilleure simulation du réseau routier. Ces profils d'automobilistes permettront d'avoir une meilleure visualisation de l'impact lors d'un aménagement de quartier.

Pour finaliser le projet, il faudrait récupérer les informations manquantes à la simulation auprès de bureaux de génie civile ou faire de la saisie manuelle et réaliser les améliorations identifiées dans le document, afin d'avoir un simulateur optimisé.

La modélisation d'un quartier urbain est un travail à part entière qui pourra être intégrer dans la simulation. Cela donnerait une réelle plus-value à l'application.

Une présentation de l'outil à des utilisateurs finaux pourrait être faite pour identifier les améliorations à apporter dans le but d'être en adéquation avec leurs besoins. La méthodologie itérative permettrait d'avoir une meilleure collaboration avec le client et de mieux répondre à ces attentes.

## Annexe A

# Glossaire

### ***Box collider***

Le *box collider* est une zone de détection de collision.

### **Flux**

Le flux désigne le nombre de voitures sur une voie par unité de temps.

### **FT, TF**

FT et TF est un acronyme pour From To, To From. Dans un graphe orienté, FT parcourt les arêtes du premier au dernier point et TF du dernier au premier.

### **Form C#**

Une Form C# est une fenêtre pour présenter différentes informations à l'utilisateur, ainsi que la possibilité d'interagir avec lui à travers des composants graphiques, tels que des boutons, des listes, etc.

### **Hiérarchie Unity**

Dans Unity 3D, c'est une représentation des objets parents - enfants. Cette représentation permet de regrouper différents objets sous un même parent, afin de faciliter l'accès à un ensemble d'objets.

### **JSON**

"JavaScript Object Notation [...] permet de représenter de l'information structurée." L'information est stockée dans un fichier dont l'extension est .json.

Source Wikipédia : [https://fr.wikipedia.org/wiki/JavaScript\\_Object\\_Notation](https://fr.wikipedia.org/wiki/JavaScript_Object_Notation)

### **Modèle 3D**

Le modèle 3D est une représentation d'objet en 3 dimensions.

### **Shape**

Le shapefile est un fichier venant du monde des SIG.

"Il contient toute l'information liée à la géométrie des objets décrits, qui peuvent être :

- des points
- des lignes,
- des polygones.

"

Source Wikipedia : <https://fr.wikipedia.org/wiki/Shapefile>

### **SIG**

"Un système d'information géographique (SIG) est un système d'information conçu pour recueillir, stocker, traiter, analyser, gérer et présenter tous les types de données spatiales et géographiques."

Source Wikipedia : [https://fr.wikipedia.org/wiki/Syst%C3%A8me\\_d%27information\\_g%C3%A9ographique](https://fr.wikipedia.org/wiki/Syst%C3%A8me_d%27information_g%C3%A9ographique)

### **SITG**

"Le système d'information du territoire à Genève (SITG) met à disposition un vaste choix de données au travers de cartes interactives faciles d'accès et de données" téléchargeables.

Source SITG : <http://ge.ch/sitg/>

### **SQLite**

SQLite est une base de données relationnelle locale qui utilise le langage SQL.

### **Timer**

Le *timer* est un minuteur. Le minuteur permet de déclencher un événement tous les x temps donnés.

### **Trigger**

Le *trigger* est un déclencheur, utilisé dans les *box collider*, qui permet de provoquer des traitements particuliers lorsqu'il est activé.

## Annexe B

### Journal de bord

Un journal de bord a été tenu durant le travail.

Semaine	Jour	Tâches	Problèmes rencontrés	Informations
1	25.04.2016	-Rendez-vous pour le commencement du travail de Bachelor -Se familiariser avec l'outil SITG (Cartes professionnelles/ extraction) -Cherche de données utiles -Créer le planning	Choix du format	
1	26.04.2016	- Se familiariser avec QGIS - Rechercher des informations sur les différents formats proposé par SITG (DXF, Shape) - Script Python lecture Shape (Librairie shapefile)		
1	27.04.2016	- 8h15 - 11h45 : Présentation Rekik (Cours Génie Logiciel) - Analyser différents carrefours (Google maps) - Extraire une zone prédefinie avec QGIS - Rechercher des données du catalogue SITG	Feu de signalisation, différentes sortes de données. Sens de la voie (FT, TF)	
1	28.04.2016	- Définir la structure du projet - Faire un récapitulatif des données Mettre en avant - Rechercher les données manquantes	Données manquantes : Préselection, Panneaux de signalisation (Feu, stop, céder le passage), sens de la voie.	
1	29.04.2016	- Rendez-vous avec Mr. Donzé (SITG / QGIS) - Commencer la réalisation de la documentation		
2	02.05.2016	- Rendez-vous hebdomadaire (Albuquerque, Domingos Polla) - Réfléchir sur la structure des données à stocker dans la BDD - Exploiter les données de flux fournies par SITG	Données de flux fournies par SITG ne sont pas assez fournies, manque d'information.	
2	03.05.2016	- Améliorer le script Python de lecture de données - Gérer les erreurs - Modéliser la base de donnée - Définir le schéma de SITG à BDD	Les flux (ligne) ne sont pas superposé sur le graphe routier. Les flux (point) trop peu de donnée. Utilisation de Python avec Unity ?	
2	04.05.2016	- Installer différents Packages (NetTopologySuite, CatFood) - Tester la lecture en C# sur console - Visualisation de l'application final, Grossièrement	Utilisation des différents packages.	
2	05.05.2016	Ascension		
2	06.05.2016	- Créer une Form C# pour chargement de la BD - Réaliser une Maquette		
3	09.05.2016	- Améliorer la structure de la base de donnée - Installer le package SQLite - Tester la base de données	Réflexion sur quel type d'utilisation de BDD.	
3	10.05.2016	- Améliorer la structure de la base de donnée - Mettre en place les différentes requêtes (SQL) - Mette en place la nouvelle BDD	Erreur pour la connexion entre sommet et arête.	Ajout des points pour la table des sommets.
3	11.05.2016	- Terminer la "classe" pour les requêtes SQL - Intégrer la classe avec le readShape (Console) - Vérifier la comptabilité de la lecture Shape et insertion dans la BDD.		Tests avec les fichiers Shape du graphe routier.
3	12.05.2016	- Vérifier les connexions sommets/arêtes - Mettre en place la gestion des sommets	Graphe routier, ajout de sommet identique	
3	13.05.2016	- Mettre en place l'insertion de la signalisation - Mettre en place l'ajout vitesse - Réfléchir sur la gestion du flux	Vitesse, zone polygonale. Liaison entre arête et cette zone.	Utilisation de l'algorithme Winding Number
4	16.05.2016	Pentecôte		
4	17.05.2016	- Use case, diagramme de séquence - Mettre en place l'algorithme Winding Number - Reflechir sur l'utilisation Spatialite ou PostGre		

4	18.05.2016	- Installer SQLite Spatialite (NetTopologySuite.IO.Spatialite) - Mapper le graphe routier avec le graphe flux - Améliorer et vérifier l'insertion de la notion de vitesse	Mappage des flux avec un seuil engendre trop d'erreurs. (Flux ne mappe pas avec la bonne arête) Utilisation de Spatialite non concluant.	
4	19.05.2016	- Mapper le graphe routier avec le graphe flux - Insérer les données du mappage dans BDD	Problème avec les données de flux, Sens du flux ? Lien avec sommet ? FT - From To, TF - To From à la main?	Connexion flux à l'arête, Edition à la main des données manquantes.
4	20.05.2016	- Mettre à jour la classe requête - Gérer les erreurs (affichage message d'erreur/ d'alerte) - Mettre au propre la form C# d'insertion - Mettre en place l'édition des données		
5	23.05.2016	- Rendez-vous hebdomadaire (Albuquerque, Domingos, Polla) - Insérer les flux à la main - Afficher les différentes rues qui sortent de la zone pour les flux - Réfléchir sur un fichier JSON comme interface entre SITG et Unity	Mappage des flux avec un seuil ne représente pas la réalité, contient trop d'erreurs. Le flux (point) ne comporte pas assez de données pour générer un flux dans le quartier.	Seulement rue sortante de la zone Car pour les flux d'entrée et sortie.
5	24.05.2016	- Créer la structure des fichiers JSON. - Créer le fichier C# lecture SQLite (Unity - Mono.Data.Sqlite) - Réfléchir sur les objets à créer	Fichier JSON facilement adaptable, pas que ce soit seulement optimiser pour la BDD créée	Création de différent objet pour la création du fichier JSON. Objet propre au fichier et non à la BDD
5	25.05.2016	- Créer les objets JSON / fichiers JSON - Insérer les objets JSON dans fichier (pas le flux) - Créer le graphe Unity, 1 arête par voiture (Projet de semestre)	Feux "Inutiles" 3 fois le même feu a des positions différentes. Faire un tri. Créer une arête par voiture -> Trop lourd. Gestion du sens de voies et des voies	Création de tout le graphe pour les parcours des voitures Pas comme pour le projet de semestre.
5	26.05.2016	- Créer le graphe (Unity), rotation des points, positions - Créer la route (Visuel) - Lier les feux au sommet carrefour (Distance entre sommet et feu)		Structure pour les feux : RoadSigns   IDSommet     ObjFeu
5	27.05.2016	- Améliorer les points du graphe - Mettre en place le mouvement des voitures sur le graphe		
6	30.05.2016	- Rendez-vous hebdomadaire (Albuquerque, Domingos, Polla) - Réfléchir sur Algorithme pour la gestion des feux ou temps statique ? - Réfléchir comment lier le feu à la voie ? - Mouvement voiture sur le graphe, FT/TF, connexion arête	Pour le flux comment connaître son sens au démarrage ? Préciser dans l'insertion FT ou TF. Performance : Ouverture / Fermeture du fichier JSON, pour récupérer les connexion.	Temps statique. Position de la ligne d'arrêt du feu sur la voie.
6	31.05.2016	- Rendez-vous Mr. Dubois - Améliorer la création du graphe - Définir l'objet sommet contenant les informations	Ouverture / Fermeture JSON pour récupérer les connexion, trop lourd pour les performances.	Structure du graphe : Graphe   idSommet     Point Mr Dubois m'a envoyé les données de flux qui mappe correctement avec le graphe routier.
6	01.06.2016	- Mettre en place GetConnexion() - Mettre en place l'algorithme Dijkstra (Parcours voiture) - Commencer le schéma de traitement Unity		
6	02.06.2016	- Définir FT/TF au démarrage - Mettre en place la récupération du parcours Dijkstra par la voiture - Améliorer le style de voiture (Trop petite) - Vérifier le résultat de l'algorithme Dijkstra	Dijkstra calcul pour chaque voiture son parcours, ce qui est beaucoup trop lourd... Utiliser Floyd? Problème avec système de voie.	Dijkstra : Création d'un tableau statique contenant les parcours déjà calculé
6	03.06.2016	- Créer le tableau Statique TabParcours, contenant les tab de précédence - Alléger le code pour améliorer les performances, éviter .Find().	Problème mouvement bizarre des voitures au changement de certains point	

7	06.06.2016	<ul style="list-style-type: none"> <li>- Rendez-vous hebdomadaire (Albuquerque, Domingos, Polla)</li> <li>- Calculer tous les parcours possibles (Dijkstra)</li> <li>- Régler le problème du mouvement voiture</li> <li>- Mettre à jour la BDD avec les Flux</li> <li>- Faire évoluer la Form C# : charger une zone pour le flux, définir si flux entrant et sortant</li> <li>- Mettre à jour le traitement pour fichier JSON - flux</li> </ul>		Problème mouvement bizarre : Ignorer premier point du parcours
7	07.06.2016	<ul style="list-style-type: none"> <li>- Mettre à l'échelle les voitures générées.</li> <li>- Alimenter Dijkstra avec nouvelle donnée de flux</li> <li>- Initialiser les voitures en prenant en compte les flux entrants et sortants</li> </ul>		
7	08.06.2016	<ul style="list-style-type: none"> <li>- Vérifier les flux entrants/ flux sortants</li> <li>- Vérifier la phase d'initialisation et du mouvement voiture</li> <li>- Ajouter des points</li> <li>- Définir la zone de détection voiture</li> </ul>	Problème d'angle des points (Globale/Locale), donc ceci pose problème pour les voitures (Pas dans la bonne voie)	
7	09.06.2016	<ul style="list-style-type: none"> <li>- Améliorer le système de changement de parcours</li> <li>- Changer la gestion des voies selon arête (Trigo)</li> </ul>	Nouveau système de voie, pose encore problème position erronée	Problème d'angle des points : Calcul x, y de la position de la voiture en faisant $axe z = \sin(a) * dist$ $axe x = \sin(a) * dist$
7	10.06.2016	<ul style="list-style-type: none"> <li>- Importer voiture Unity (Plus jolie voiture)</li> <li>- Mettre en place les priorités (Droite / Gauche)</li> </ul>	Calcul des priorités trop instable.	Système de voie ok: Mathf.Sin -> radian Calcul de la priorité par rapport à sa rotation.
8	13.06.2016	<ul style="list-style-type: none"> <li>- Rendez-vous hebdomadaire (Albuquerque, Domingos, Polla)</li> <li>- Définir les différent boxCollider (Size, Center)</li> <li>- Définir la distance de sécurité</li> </ul>		
8	14.06.2016	<ul style="list-style-type: none"> <li>- Mettre en place les feux, ligne d'arrêt, séquence (RJV)</li> <li>- Améliorer les détections de priorité</li> </ul>	Problème lors du changement de sizeBox Collider (Priorité)	Problème résolu: Passage de Distance de sécurité à priorité (Droite ou gauche) si voiture est à moins de 15 mètre du carrefour.
8	15.06.2016	<ul style="list-style-type: none"> <li>- Ajouter des objets à la voiture</li> <li>- Mettre en place Angle de vue lors du changement de voie</li> <li>- Gérer le problème d'interblocage</li> </ul>	Lors d'un interblocage que se passe-t-il ? Voiture devient kinematic ? Voiture détruite? Voiture force donc accident ?	Structure voiture : SkyCar (Tag : Car)   Carrosserie (Accident)   BlindSpot (Angle de vue)   SafetyDist (Distance de sécu)
8	16.06.2016	<ul style="list-style-type: none"> <li>- Gérer le problème d'interblocage</li> </ul>	Trop d'accident dû à l'interblocage.	Lors d'un interblocage, La voiture réduit son champ de détection de manière à passer Si aucune voiture n'arrive à passer, alors une est détruite
8	17.06.2016	<ul style="list-style-type: none"> <li>- Mettre en place le système de préselection</li> <li>- Améliorer la gestion de priorité</li> <li>- Mettre en place la gestion des Stops</li> </ul>		Calcul d'angle 15 - 179 - Tourne à gauche -15 - -179 - tourne à droite
9	20.06.2016	<ul style="list-style-type: none"> <li>- Rendez-vous hebdomadaire (Albuquerque, Domingos, Polla)</li> <li>- Définir le profil conducteur</li> <li>- Définir le ralentissement avant carrefour</li> </ul>		
9	21.06.2016	<ul style="list-style-type: none"> <li>- Mettre à jour structure du feu (RJVJ, temps répartie)</li> <li>- Mettre en place le système d'élection - pour les cas de blocage</li> <li>- Ajouter des points (Coefficient)</li> </ul>	Problème du système d'élection, génère plus d'accident. Car force le passage même si voiture face à lui.	Régler problème de détection
9	22.06.2016	<ul style="list-style-type: none"> <li>- Ajouter des points (Distance)</li> <li>- Améliorer le système de parcours voiture</li> </ul>	Problème d'utilisation de la trigonometrie pour ajouter avec une distance	Amélioration du système de parcours pour éviter des "Faces à faces" entre voiture

9	23.06.2016	- Ajouter des point (Distance) - Mettre en place l'algorithme Floyd		Ajout de point avec distance : Calcul vectorielle.
9	24.06.2016	- Vérifier l'algorithme Floyd - Lire les données Floyd - Deployer Floyd sur les voitures		
10	27.06.2016	- Rendez-vous hebdomadaire (Albuquerque, Domingos, Polla) - Mettre en place le compteur - Améliorer les condition changement de parcours - Améliorer la détection		
10	28.06.2016	- Corriger certains bugs		Amélioration des feux/stops
10	29.06.2016	- Corriger certains bugs		Amélioration du déplacement des voitures
10	30.06.2016	- Rediger la documentation		
10	01.07.2016	- Rediger la documentation		
11	04.07.2016	- Rediger la documentation		
11	05.07.2016	- Rediger la documentation		
11	06.07.2016	- Rediger la documentation		
11	07.07.2016	- Rediger la documentation		
11	08.07.2016	- Rediger la documentation		
12	11.07.2016	- Rediger la documentation		
12	12.07.2016	- Rediger la documentation		
12	13.07.2016	- Rediger la documentation		
12	14.07.2016	- Rendu mémoire bachelor		

## Annexe C

# Planning

Un planning initial a été réalisé lors du début du projet de Bachelor. Ce planning a permis d'avoir des délais à respecter. Puis un planning au cours du travail a été réalisé afin de voir l'avancement ou le retard de certaines tâches.

Semaines/Tâches	Semaine 1	Semaine 2	Semaine 3	Semaine 4	Semaine 5	Semaine 6	Semaine 7	Semaine 8	Semaine 9	Semaine 10	Semaine 11	Semaine 12	Status
Compréhension données SITG													
Tries des données													
Récupération des données (SITG - Fichier Shape)													
Création BDD ou Fichiers pour stocker les données													
Création du Graphe (Unity)													
Modélisation du carrefour / Récupération données (Unity)													
Génération voiture avec un plus court chemin (Unity)													
Gestion du flux (Unity / Donnée SITG)													
Détection Voiture, seulement dans une certaine zone (Unity)													
Amélioration des objets Panneaux, Voiture, Voie (Unity)													
Amélioration													
Débogage & Tests													
Documentation													

# Bibliographie

- [1] SITG. Catalogue de données sitg. [http://ge.ch/sitg/sitg\\_catalog/sitg\\_donnees](http://ge.ch/sitg/sitg_catalog/sitg_donnees).
- [2] Wikipédia. Page wikipédia sur l'algorithme de dijkstra. [https://fr.wikipedia.org/wiki/Algorithme\\_de\\_Dijkstra](https://fr.wikipedia.org/wiki/Algorithme_de_Dijkstra).
- [3] Wikipédia. Page wikipédia sur l'algorithme de floyd-warshall. [https://fr.wikipedia.org/wiki/Algorithme\\_de\\_Floyd-Warshall](https://fr.wikipedia.org/wiki/Algorithme_de_Floyd-Warshall).
- [4] Wikipédia. Page wikipédia sur les points dans un polygone. [https://en.wikipedia.org/wiki/Point\\_in\\_polygon](https://en.wikipedia.org/wiki/Point_in_polygon).
- [5] Wikipédia. Page wikipédia sur les sig. [https://fr.wikipedia.org/wiki/Syst%C3%A8me\\_d%27information\\_g%C3%A9ographique](https://fr.wikipedia.org/wiki/Syst%C3%A8me_d%27information_g%C3%A9ographique).
- [6] SITG. Page présentation sitg. <http://ge.ch/sitg/>.
- [7] SITG. Page wikipédia sur le format shape. <https://fr.wikipedia.org/wiki/Shapefile>.
- [8] Unity. Statistique unity dans les relations publics. <https://unity3d.com/public-relations>.
- [9] Julien Bergounhoux. Article sur la comparaison de trois moteurs de jeu populaire. <http://www.usine-digitale.fr/editorial/unreal-unity-valve-trois-moteurs-de-jeux-video-trois-business-models-N317270>, Mars 2015.
- [10] QGIS. Site officiel de l'outil qgis. <http://www.qgis.org/fr/site/>.
- [11] QGIS. Manuel d'utilisation de qgis 2.8. [http://docs.qgis.org/2.8/fr/docs/user\\_manual/](http://docs.qgis.org/2.8/fr/docs/user_manual/).
- [12] NuGet. Manager de paquet pour les outils microsoft inclu .net. <https://www.nuget.org>.
- [13] NuGet. Liens d'installation du package sqlite. <https://www.nuget.org/packages/System.Data.SQLite>.
- [14] Wiki Unity. Liens du plugin unity simplejson. <http://wiki.unity3d.com/index.php/SimpleJSON>.
- [15] Wiki Unity. Liens expliquant le plugin unity sqlite. <http://wiki.unity3d.com/index.php/SQLite>.
- [16] Unity. Documentation d'unity. <https://docs.unity3d.com/Manual/index.html>.
- [17] Unity. Forum d'unity. <http://answers.unity3d.com/>.