

**Sous le thème**

**Rapport de projet**

**3ème année**

**Ingénierie Informatique et Réseaux**

Annee UNIVERSITAIRE: 2023-2024

GESTION DE TRAFIC ROUTIER

**Réalisé par :**

Ayoub Majjid

Rochdi mohamed Amine

**Encadré par :**

Tuteur de l’école : Pr. Mariame Amine

**Introduction :**

Le projet de gestion de trafic routier en C++ a été réalisé dans le cadre du cours de Programmation Orientée Objet 2, sous la supervision du Dr. Mariame AMINE, pour l'année universitaire 2023/2024. L'objectif principal de ce projet était de développer une plateforme de gestion du trafic routier en utilisant les principes de la programmation orientée objet, en mettant particulièrement l'accent sur l'héritage et le polymorphisme.

**Objectif :**

Le principal objectif du projet était de créer un système de gestion du trafic routier permettant de surveiller et de gérer la circulation des véhicules sur les routes. Le système devait permettre l'ajout, la visualisation et le contrôle des véhicules en circulation, tout en évitant les collisions potentielles.

**Fonctionnalités Implémentées :**

**1. Classe de base Vehicule:**

- La classe de base Vehicule a été implémentée avec les attributs marque, modèle et année de fabrication, ainsi que la méthode virtuelle afficher().

**2. Classes dérivées:**

- La classe Voiture a été dérivée de la classe Vehicule avec l'ajout de l'attribut couleur et la redéfinition de la méthode afficher().

- La classe Camion a été dérivée de la classe Vehicule avec l'ajout de l'attribut capacité de charge et la redéfinition de la méthode afficher().

**3. Nouveaux Types de Véhicules:**

- Deux nouveaux types de véhicules ont été ajoutés: moto, bus

- Chaque type de véhicule a des attributs spécifiques et la méthode afficher() et collision() a été redéfinie en conséquence.

**4. Classe GestionTrafic:**

- La classe GestionTrafic a été implémentée pour gérer les véhicules en circulation en utilisant un tableau de pointeurs.

- Les méthodes ajouterVehicule(), afficherVehicules() et verifierCollisions() ont été implémentées conformément aux spécifications.

**5. Gestion de la Mémoire:**

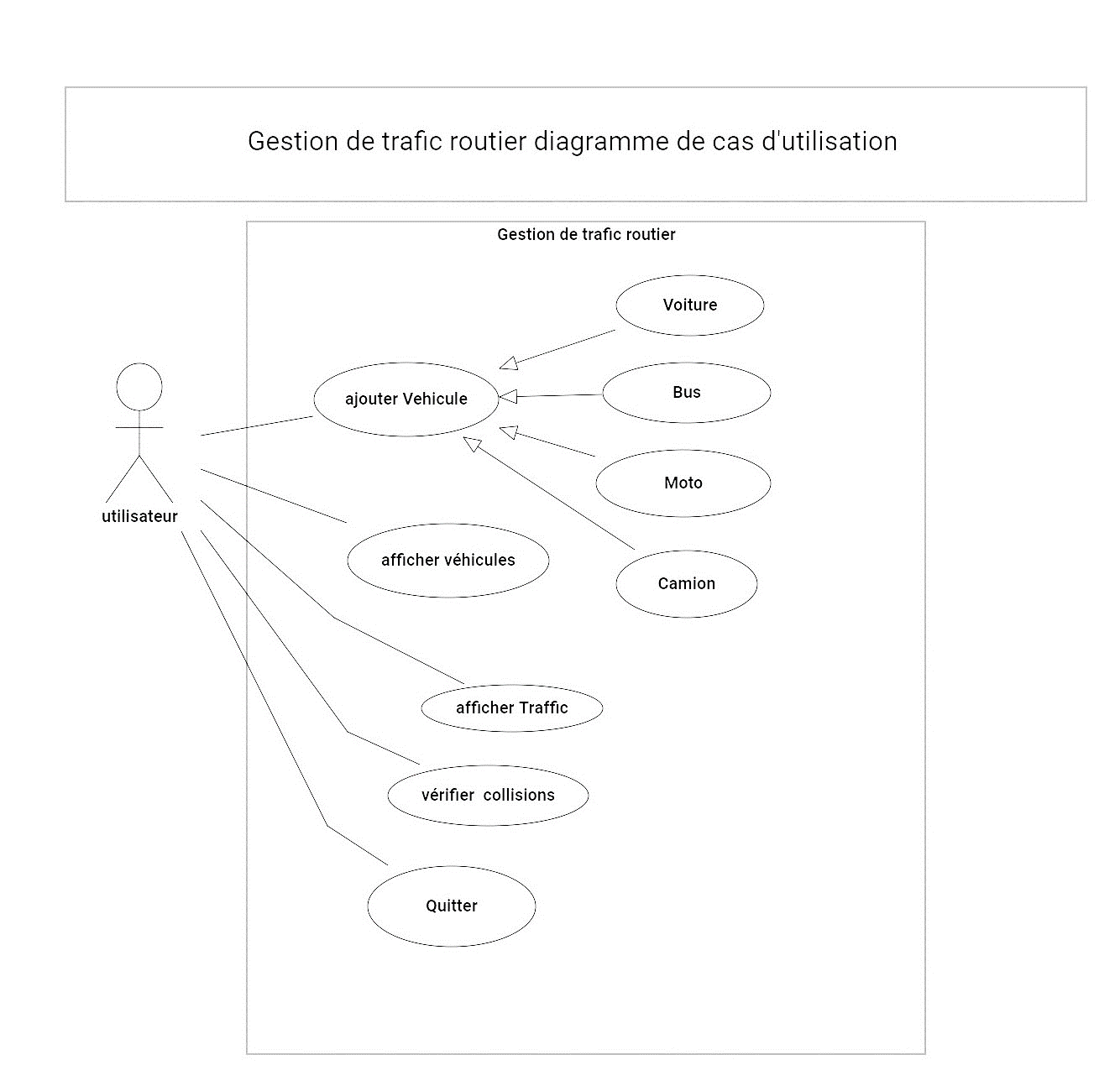
- Une gestion appropriée de la mémoire allouée pour les véhicules a été assurée afin d'éviter les fuites de mémoire à la fin du programme.

**Tests et Validation :**

Le système a été testé en ajoutant plusieurs véhicules à la plateforme et en vérifiant que les détails de tous les véhicules étaient correctement affichés. De plus, la méthode verifierCollisions() a été testée pour garantir l'évitement des collisions entre les véhicules.

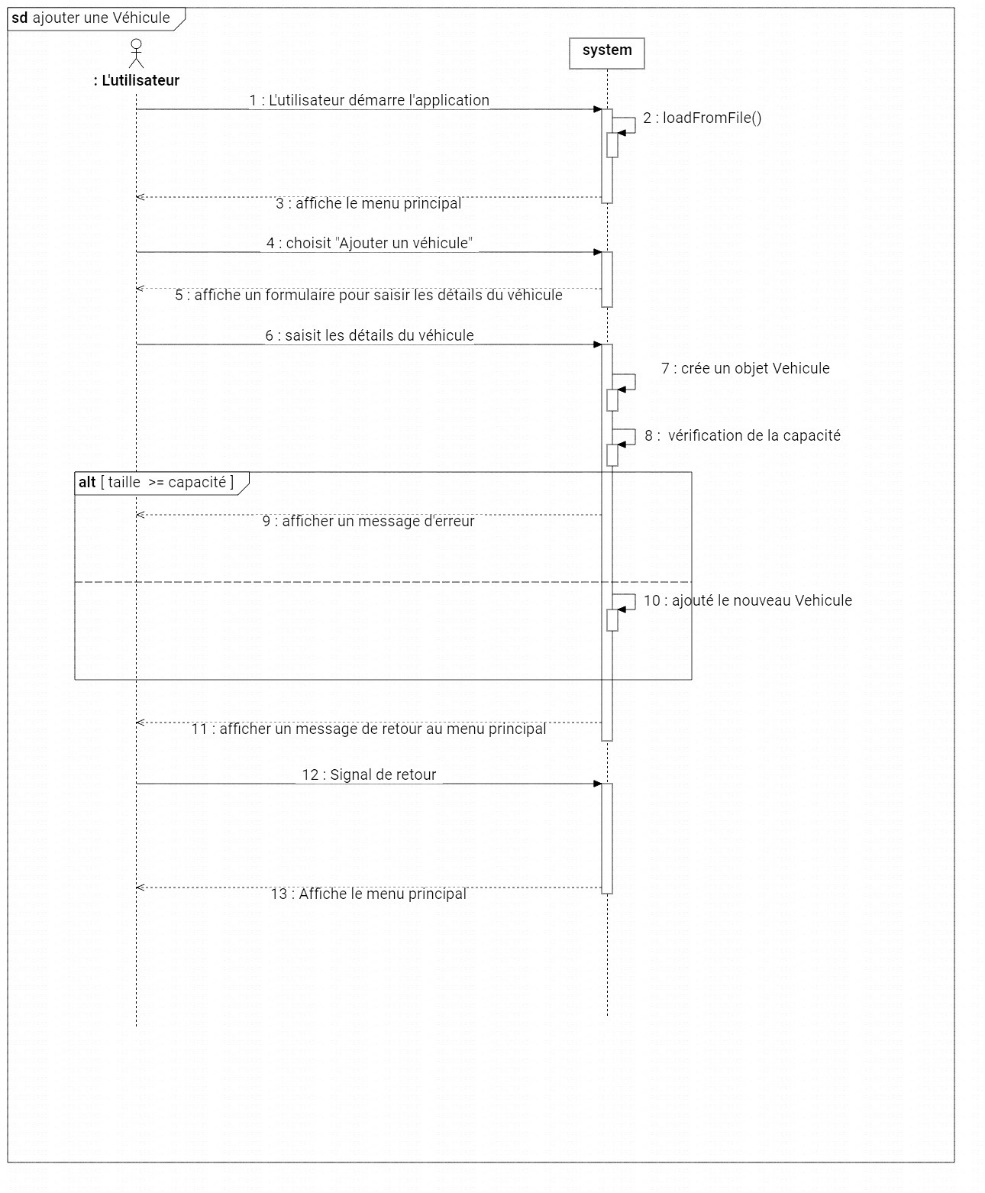
**Diagrammes UML**

1. **Diagramme de Cas d'Utilisation:**

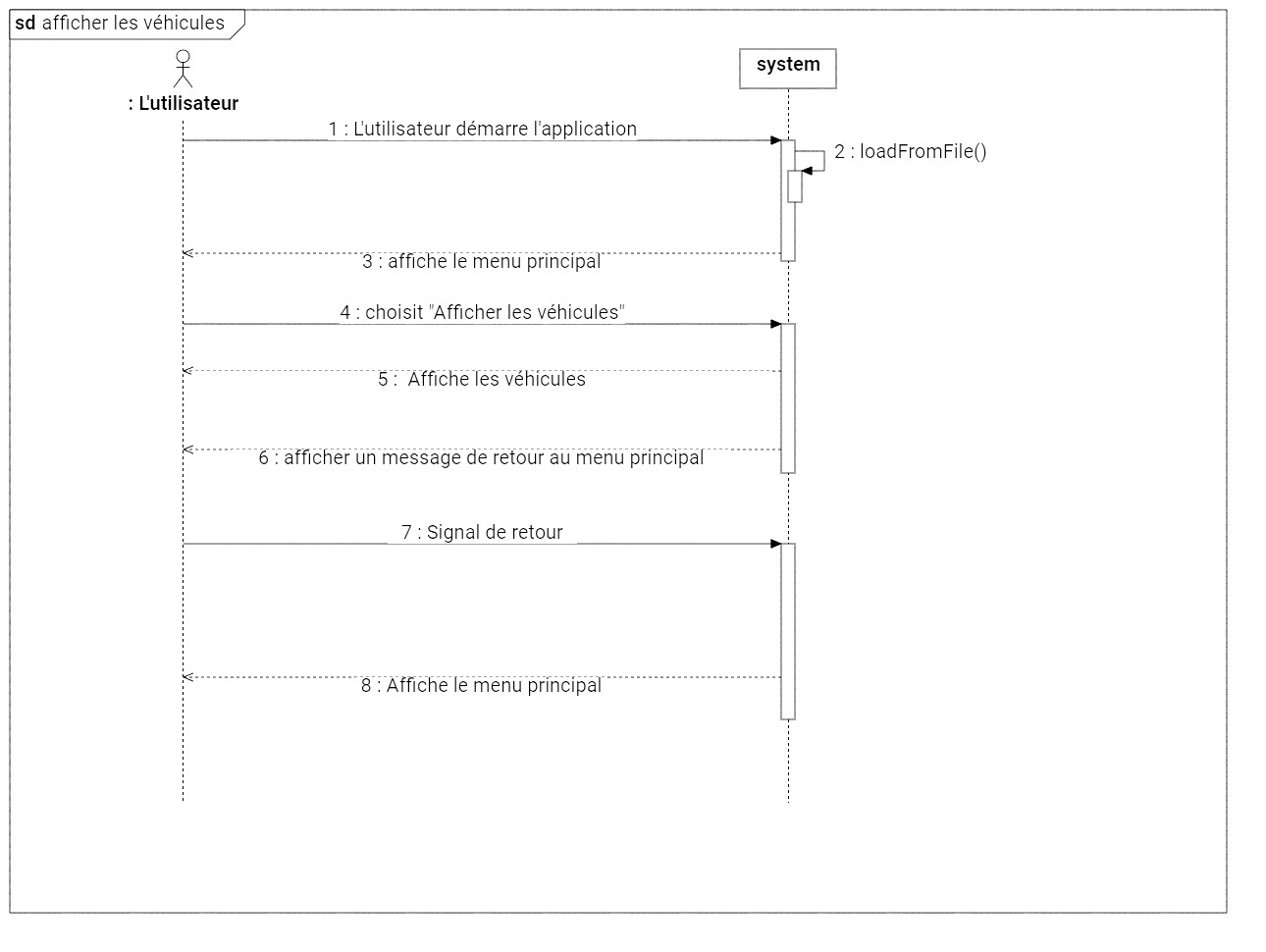


- Un diagramme de cas d'utilisation a été créé pour représenter les interactions entre les acteurs (utilisateurs) et le système de gestion de trafic routier.

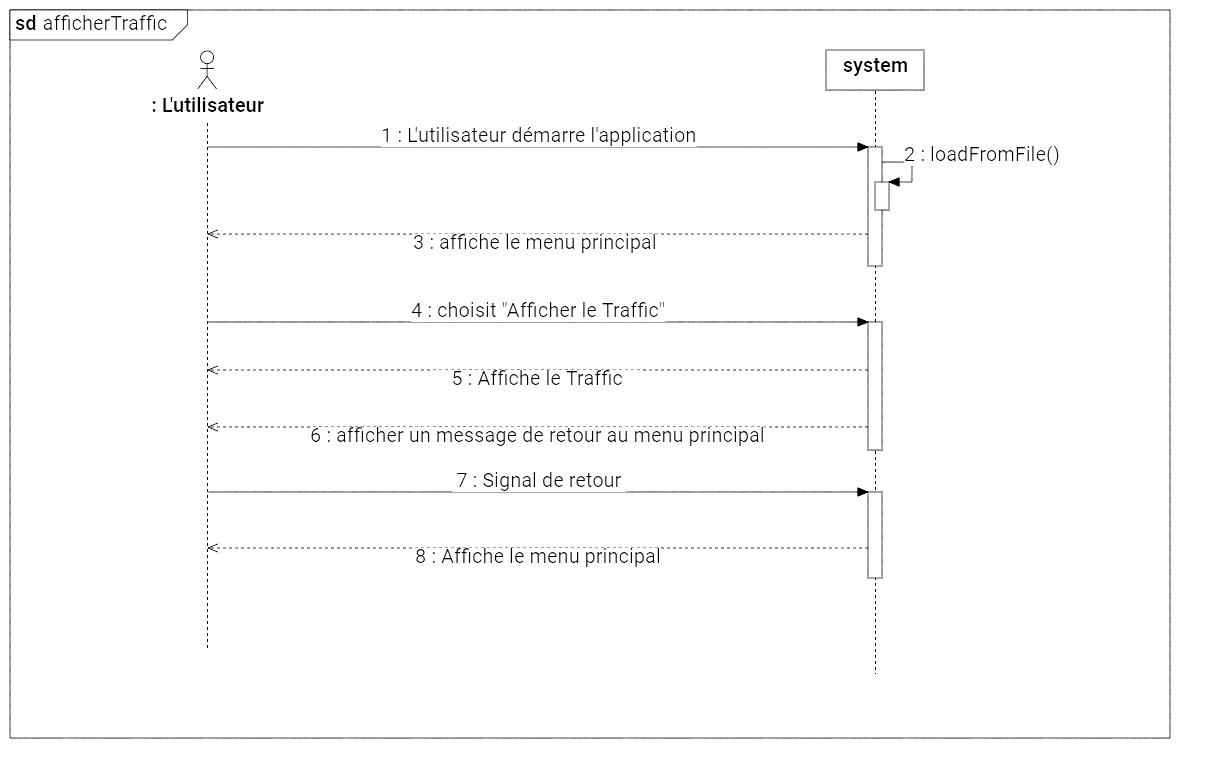
1. **Diagramme de Séquence:**
2. ajouter une Véhicule :



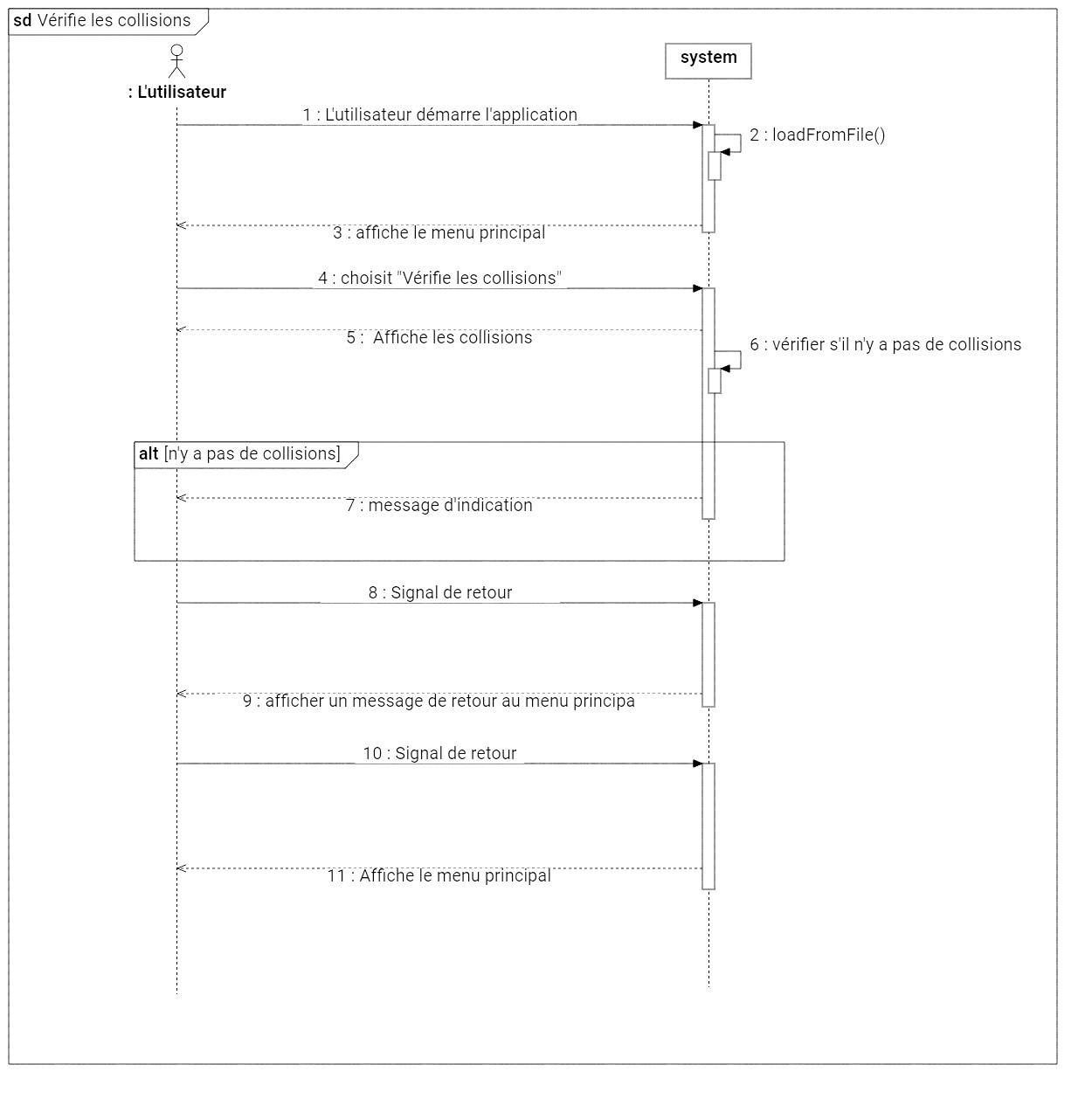
1. afficher les véhicules :



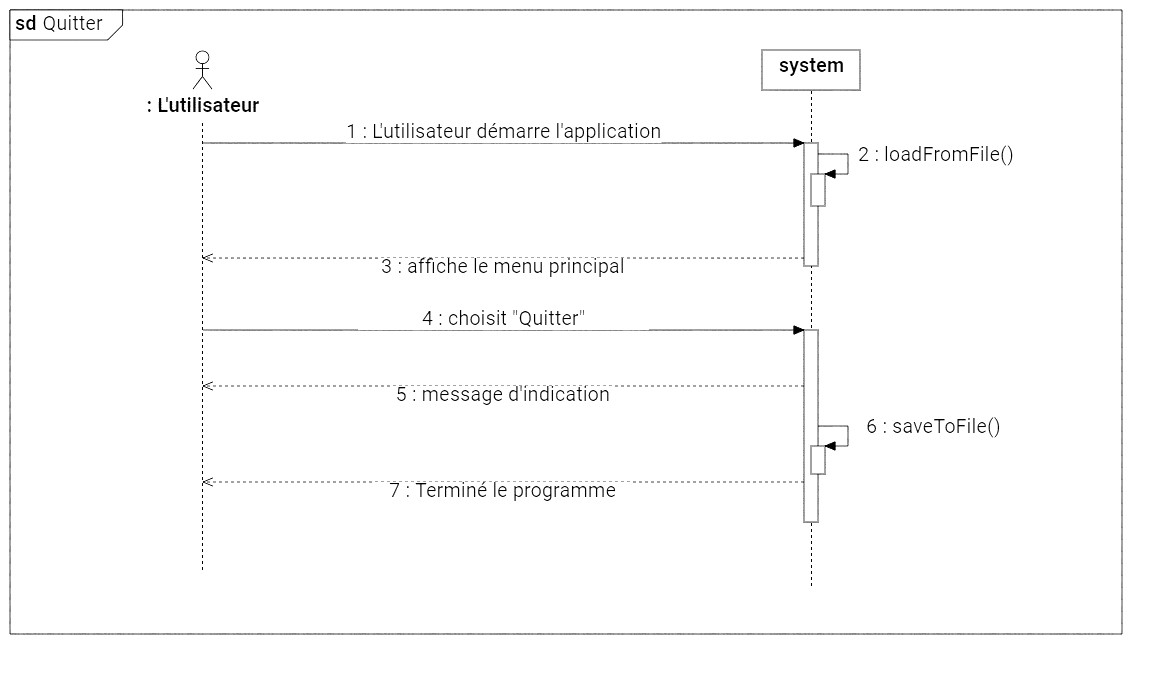
1. Afficher le Traffic :



1. Vérifie les collisions :

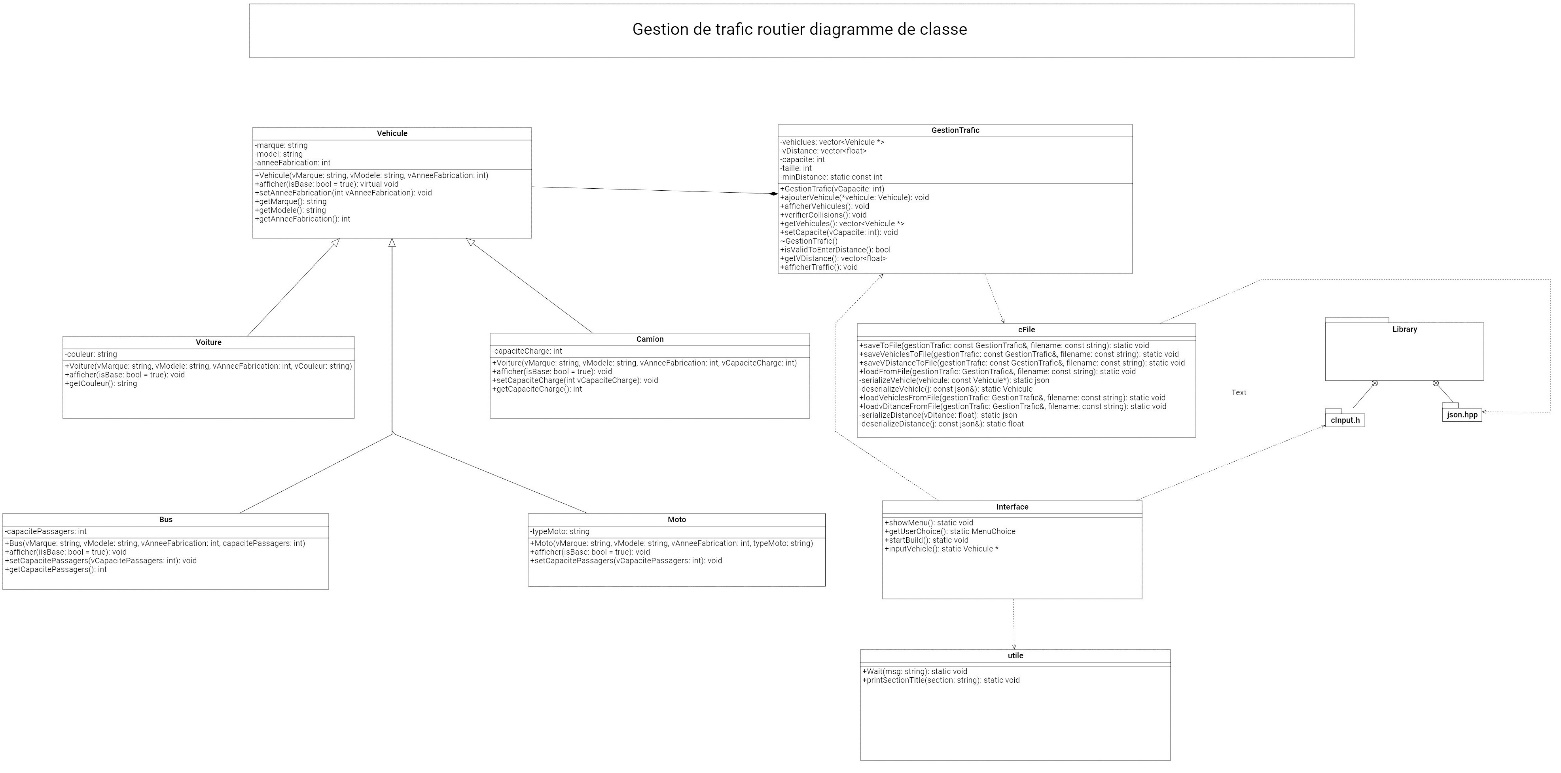


1. Quitter :

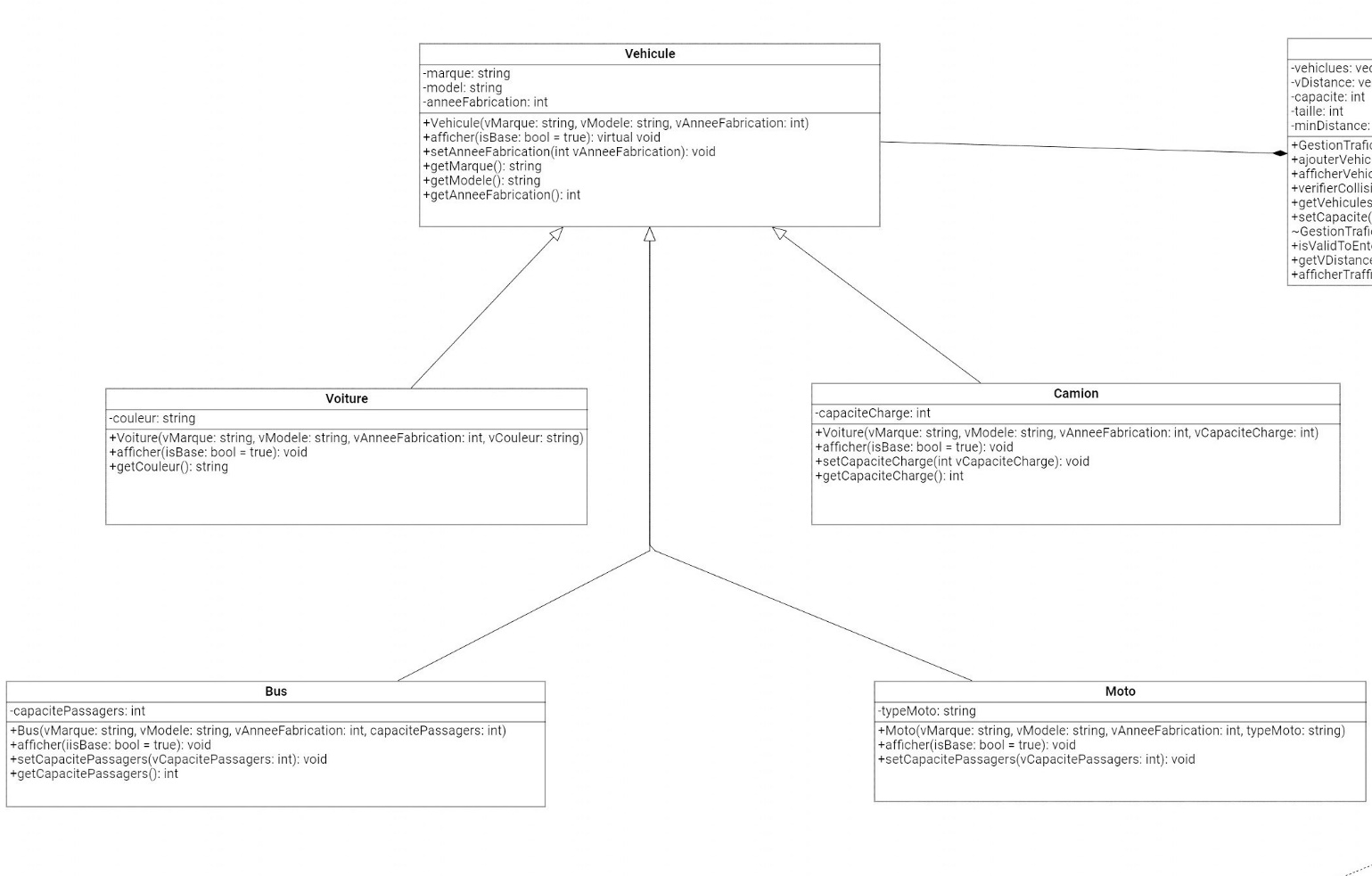


- Un diagramme de séquence a été conçu pour illustrer la séquence des actions entre les objets dans le système lors de l'ajout de véhicules, l'affichage des détails et la vérification des collisions.

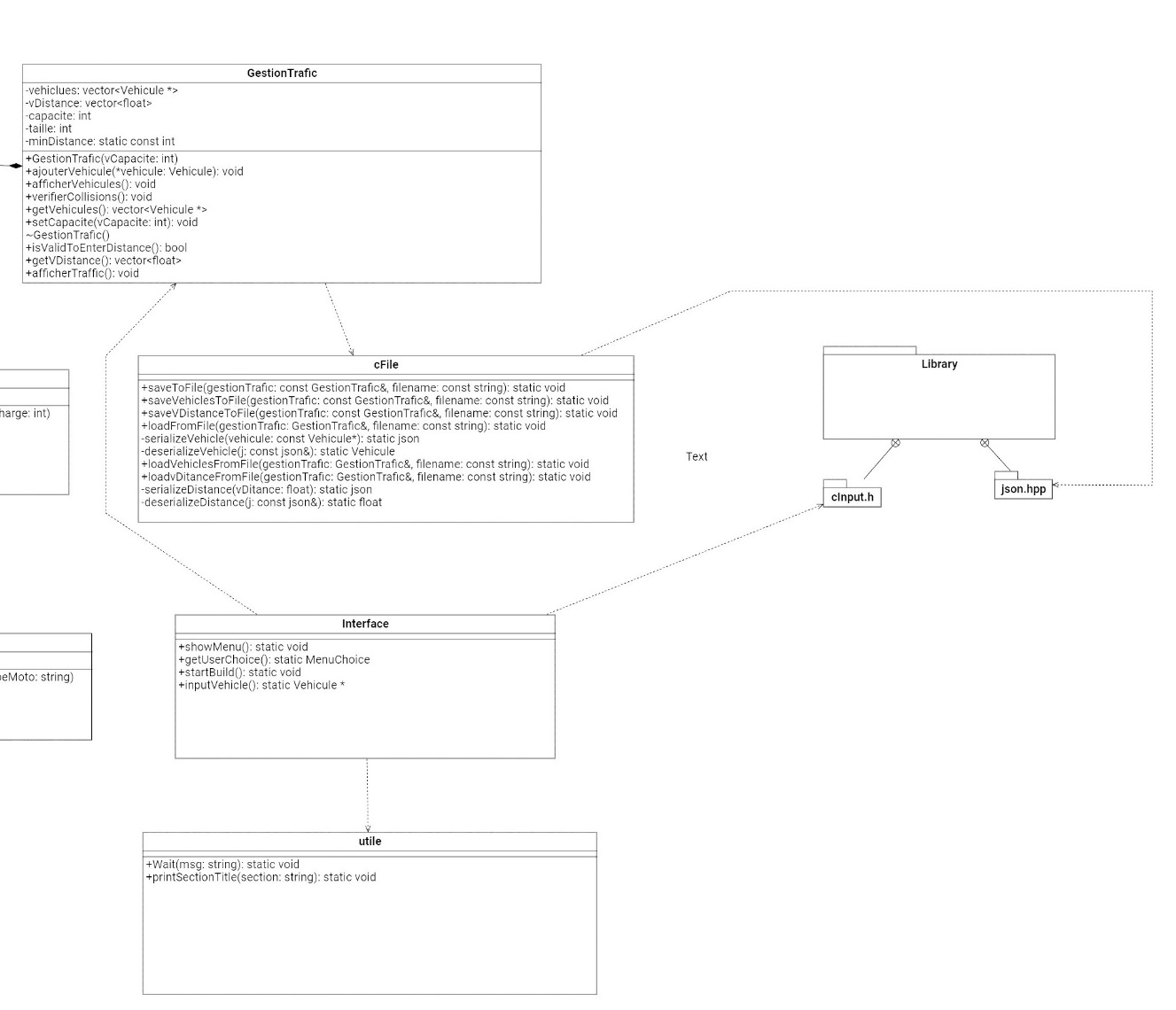
1. **Diagramme de class:**



**2-a ) Les classes de l’héritage :**

****

**2-b) Les classes de Gestion :**



-Un diagramme de séquence a été conçu pour illustrer la séquence des actions entre les objets du système lors de l'ajout de véhicules, l'affichage des détails et la vérification des collisions

**Réalisation :**

1. **Bibliothèque :**

****

ce code semble préparer l'environnement de développement pour un programme C++ qui manipulera des fichiers JSON et utilisera des fonctionnalités de lecture/écriture de fichiers, ainsi que des opérations sur les chaînes de caractères et les vecteurs.

1. **Classe utile :**

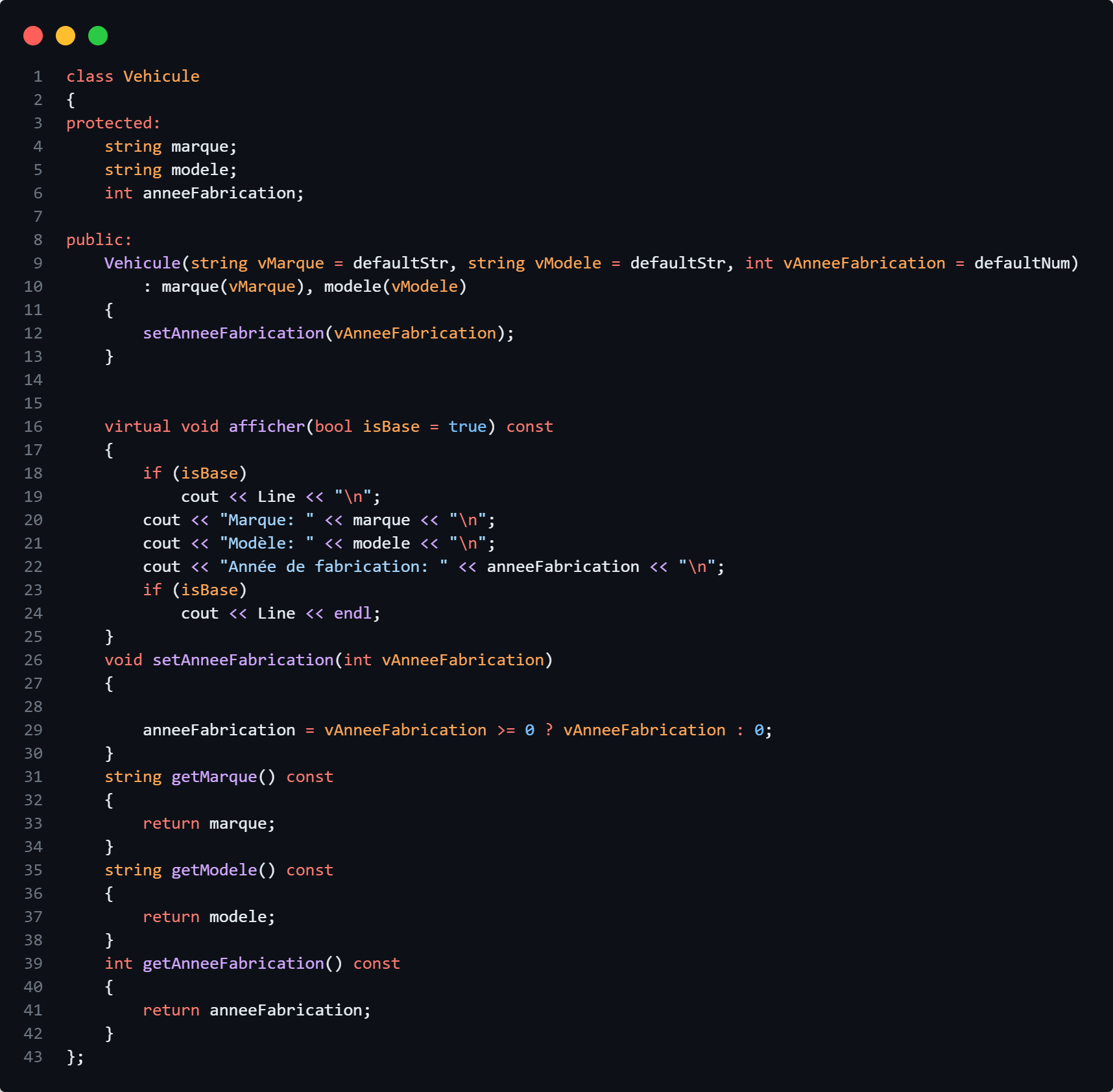
****

 La classe **utile** comporte des méthodes statiques utilisées pour des opérations générales.

 La méthode **Wait** affiche un message à la sortie standard et attend que l'utilisateur appuie sur une touche avant de poursuivre.

 La méthode **printSectionTitle** affiche un titre de section encadré par des lignes de tirets pour la mise en forme.

1. **Class véhicule :**

****

1. **Classe voiture :**

****

- La classe de base `Vehicule` représente un véhicule avec des attributs tels que la marque, le modèle et l'année de fabrication. Elle possède des méthodes pour définir et obtenir ces attributs, ainsi qu'une méthode `afficher` pour afficher les détails du véhicule.

- Les classes dérivées `Voiture`, `Camion`, `Bus` et `Moto` héritent de la classe de base `Vehicule` et ajoutent des attributs spécifiques à chaque type de véhicule, tels que la couleur pour les voitures, la capacité de charge pour les camions, la capacité de passagers pour les bus, et le type de moto pour les motos.

- Chaque classe dérivée redéfinit la méthode `afficher` de la classe de base pour afficher les détails spécifiques au type de véhicule en plus des détails de base.

- Les méthodes `set` et `get` sont utilisées pour définir et obtenir les valeurs des attributs spécifiques à chaque classe dérivée.

1. **Classe GestionTraffic :**

****

classe `GestionTrafic` gère le trafic des véhicules enregistrés dans un système. Voici une description générale de ses fonctionnalités :

- La classe contient des vecteurs de pointeurs vers des objets de type `Vehicule` et de distances entre les véhicules.

- Elle a des attributs pour la capacité maximale du trafic et la taille actuelle du trafic.

- Le constructeur initialise la capacité du trafic et la taille actuelle à partir d'un paramètre donné.

- La méthode `ajouterVehicule` permet d'ajouter un véhicule au trafic si la capacité n'est pas dépassée. La classe `cFile` fournit des méthodes statiques pour sauvegarder et charger des données liées au trafic depuis et vers des fichiers JSON. Voici une explication générale de ses fonctionnalités :

- Les méthodes `saveVehiclesToFile` et `saveVDistanceToFile` sont utilisées pour sauvegarder les véhicules et les distances entre eux dans des fichiers JSON.

- Les méthodes `loadVehiclesFromFile` et `loadvDistanceFromFile` sont utilisées pour charger les données de véhicules et de distances à partir de fichiers JSON.

- La méthode `saveToFile` permet de sauvegarder à la fois les véhicules et les distances dans des fichiers spécifiés.

- La méthode `loadFromFile` permet de charger à la fois les données de véhicules et de distances à partir de fichiers spécifiés.

- Les méthodes `serializeDistance`, `serializeVehicle`, `deserializeDistance` et `deserializeVehicle` sont utilisées pour sérialiser et désérialiser les données de distance et de véhicule entre les objets JSON et les objets de classe.

En résumé, cette classe fournit une interface pratique pour sauvegarder et charger les données de trafic depuis et vers des fichiers JSON, en utilisant les fonctionnalités de sérialisation et de désérialisation de la bibliothèque JSON.

- `ajouterDistance` permet d'ajouter la distance entre les véhicules.

- `afficherVehicules` affiche les détails de tous les véhicules dans le trafic.

- `afficherTraffic` affiche les véhicules avec leurs distances respectives entre eux.

- `verifierCollisions` vérifie s'il y a des collisions entre les véhicules en fonction de leurs distances.

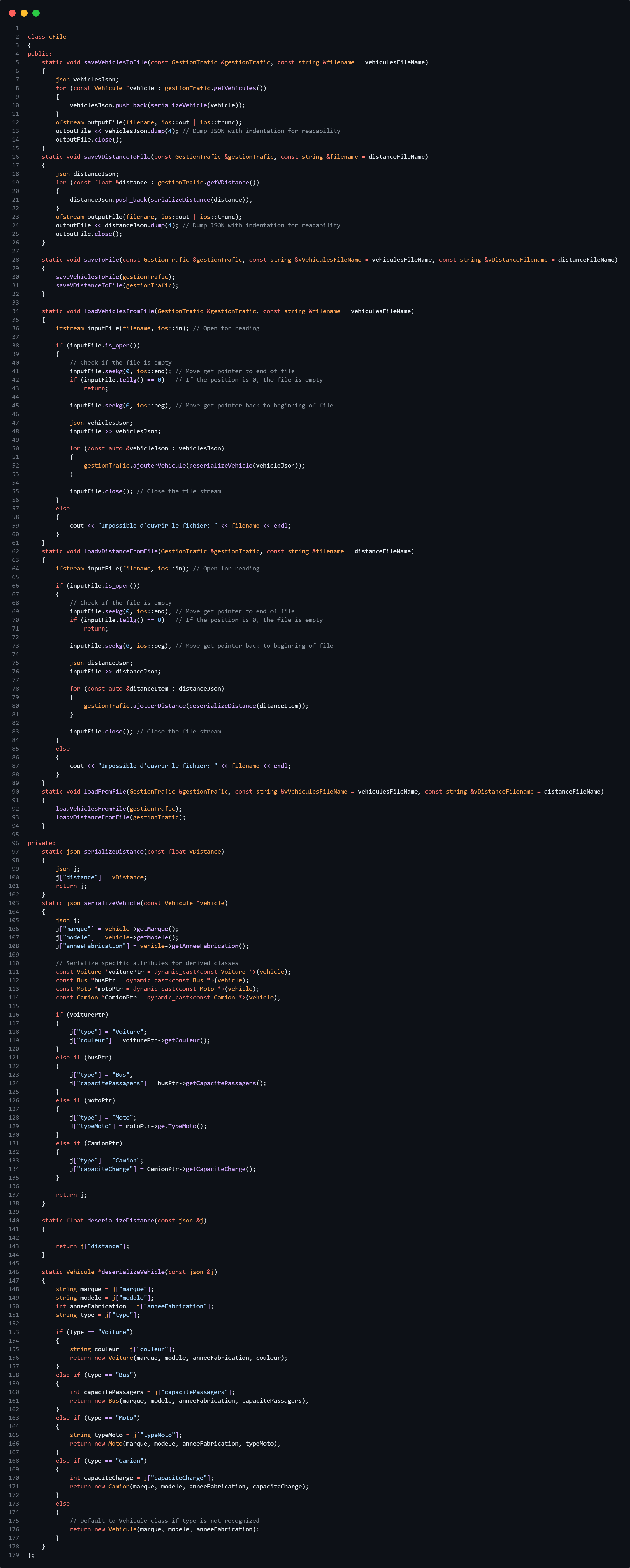
- Il y a des méthodes `get` pour obtenir les vecteurs de véhicules et de distances.

- Il y a une méthode `setCapacite` pour définir la capacité du trafic.

- Le destructeur libère la mémoire allouée dynamiquement pour les objets `Vehicule`.

La constante `minDistance` représente la distance minimale entre les véhicules pour éviter les collisions.

1. **classe cFile :**

**a**

La classe `cFile` fournit des méthodes statiques pour sauvegarder et charger des données liées au trafic depuis et vers des fichiers JSON. Voici une explication générale de ses fonctionnalités :

- Les méthodes `saveVehiclesToFile` et `saveVDistanceToFile` sont utilisées pour sauvegarder les véhicules et les distances entre eux dans des fichiers JSON.

- Les méthodes `loadVehiclesFromFile` et `loadvDistanceFromFile` sont utilisées pour charger les données de véhicules et de distances à partir de fichiers JSON.

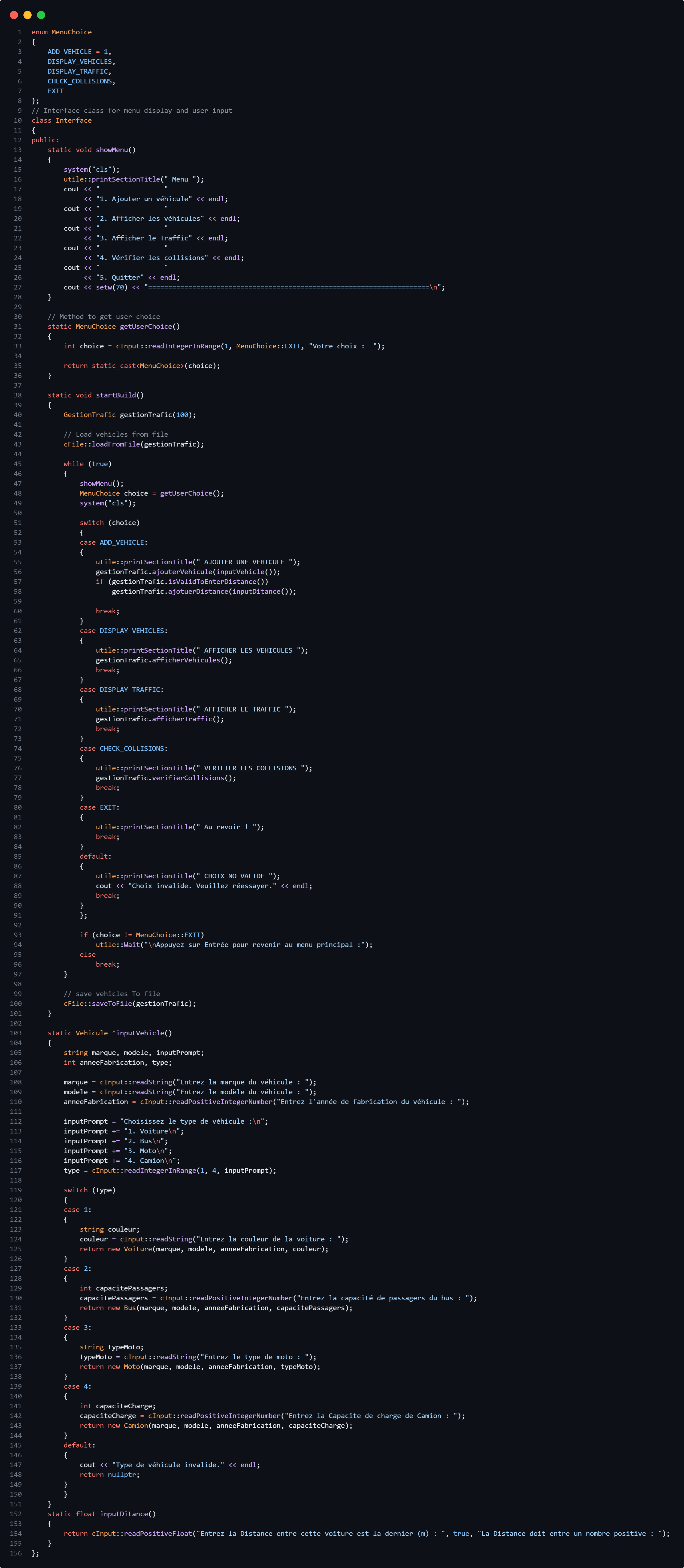
- La méthode `saveToFile` permet de sauvegarder à la fois les véhicules et les distances dans des fichiers spécifiés.

- La méthode `loadFromFile` permet de charger à la fois les données de véhicules et de distances à partir de fichiers spécifiés.

- Les méthodes `serializeDistance`, `serializeVehicle`, `deserializeDistance` et `deserializeVehicle` sont utilisées pour sérialiser et désérialiser les données de distance et de véhicule entre les objets JSON et les objets de classe.

En résumé, cette classe fournit une interface pratique pour sauvegarder et charger les données de trafic depuis et vers des fichiers JSON, en utilisant les fonctionnalités de sérialisation et de désérialisation de la bibliothèque JSON.

1. **Classe interface :**

****

Cette classe `Interface` gère l'interface utilisateur pour le système de gestion du trafic. Voici une explication générale de ses fonctionnalités :

- La méthode `showMenu` affiche un menu à l'utilisateur avec différentes options, telles que l'ajout de véhicules, l'affichage des véhicules, l'affichage du trafic, la vérification des collisions et la sortie du programme.

- La méthode `getUserChoice` permet à l'utilisateur de choisir une option à partir du menu affiché et retourne cette valeur.

- La méthode `startBuild` initialise le système de gestion du trafic, charge les véhicules à partir d'un fichier, puis affiche le menu et gère les actions de l'utilisateur en fonction de ses choix.

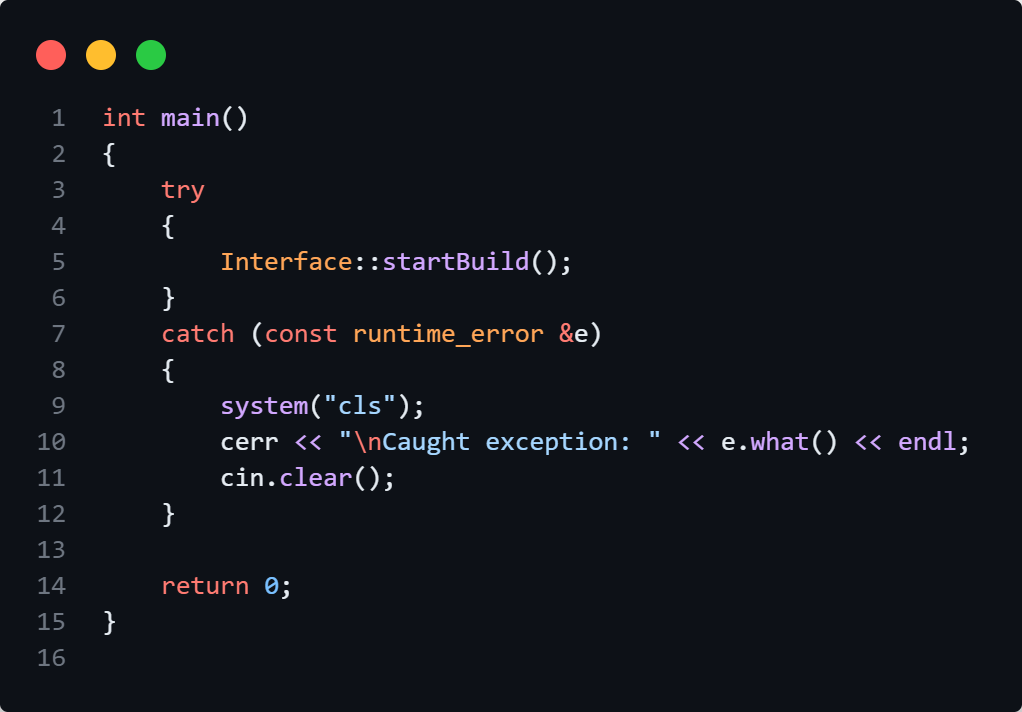
- Les méthodes `inputVehicle` et `inputDitance` sont utilisées pour saisir les détails d'un nouveau véhicule et la distance entre les véhicules, respectivement.

- Le choix de l'utilisateur est traité dans une boucle `switch` et les actions correspondantes sont effectuées, comme l'ajout de véhicules, l'affichage des véhicules ou la vérification des collisions.

- Une fois que l'utilisateur choisit de quitter le programme, les véhicules sont sauvegardés dans un fichier.

En résumé, cette classe fournit une interface utilisateur conviviale pour interagir avec le système de gestion du trafic, en permettant à l'utilisateur d'effectuer différentes opérations et en fournissant des instructions claires et des contrôles pour faciliter la saisie des données et la navigation dans le système.

1. **Main :**

****

**Conclusion :**

En conclusion, le projet de gestion de trafic routier en C++ a été mené à bien, avec toutes les fonctionnalités spécifiées implémentées et testées. Le respect des délais et la qualité du code ont été des points essentiels dans la réalisation de ce projet. L'ajout des diagrammes UML a permis de mieux comprendre et visualiser le fonctionnement du système.

Ce rapport étendu inclut maintenant une mention des nouveaux types de véhicules (moto, bus) et l'introduction des diagrammes de cas d'utilisation et de séquence pour illustrer l'interaction avec le système.