ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

HONORIS UNITED UNIVERSITIES

Rapport de projet

3ème année

Ingénierie Informatique et Réseaux

Sous le thème

GESTION DE TRAFIC ROUTIER

Réalisé par :
Ayoub Majjid
Rochdi mohamed Amine

Encadré par :
Tuteur de l'école : Pr. Mariame Amine

ANNEE UNIVERSITAIRE: 2023-2024

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.

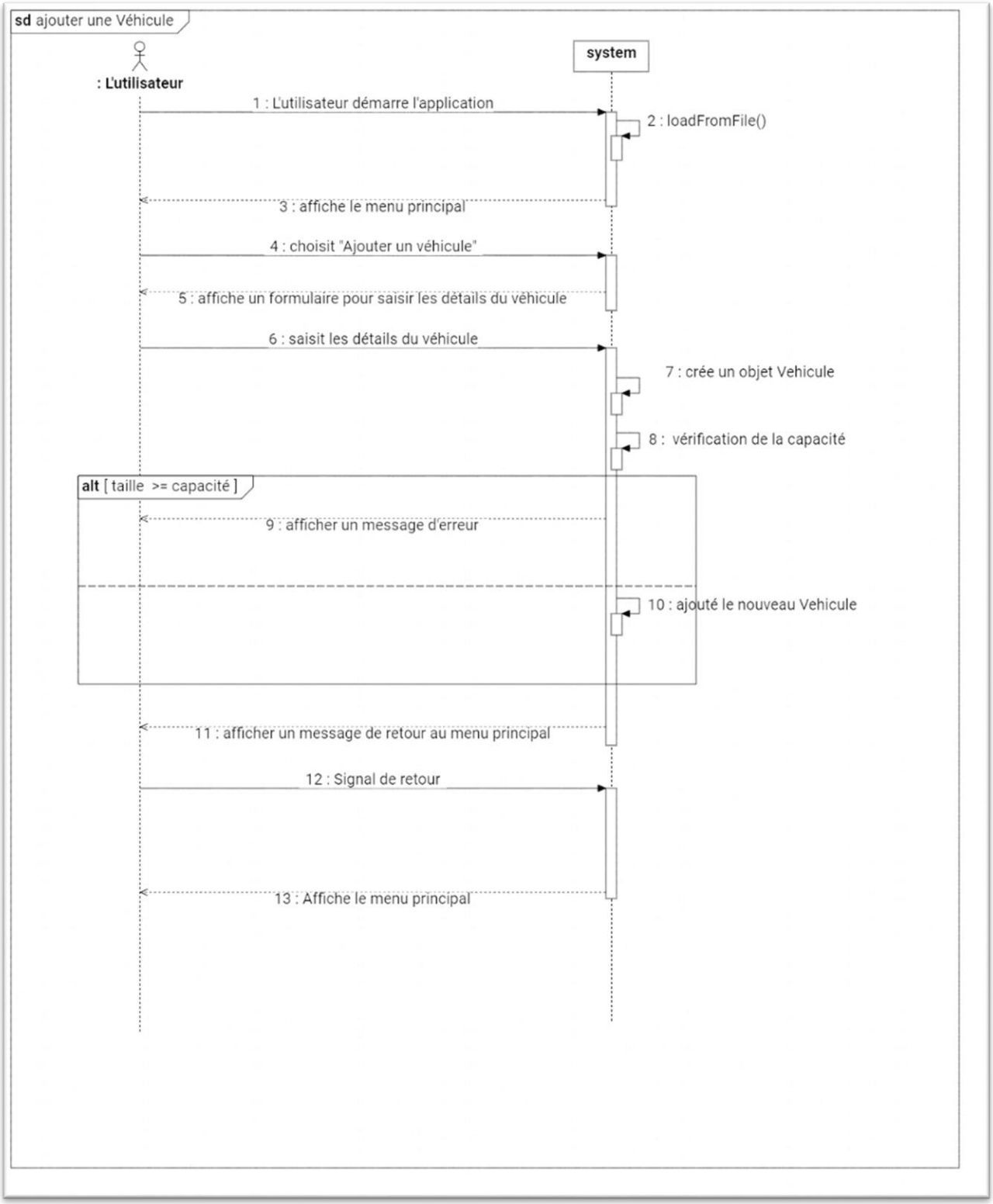
Diagrammes UML

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

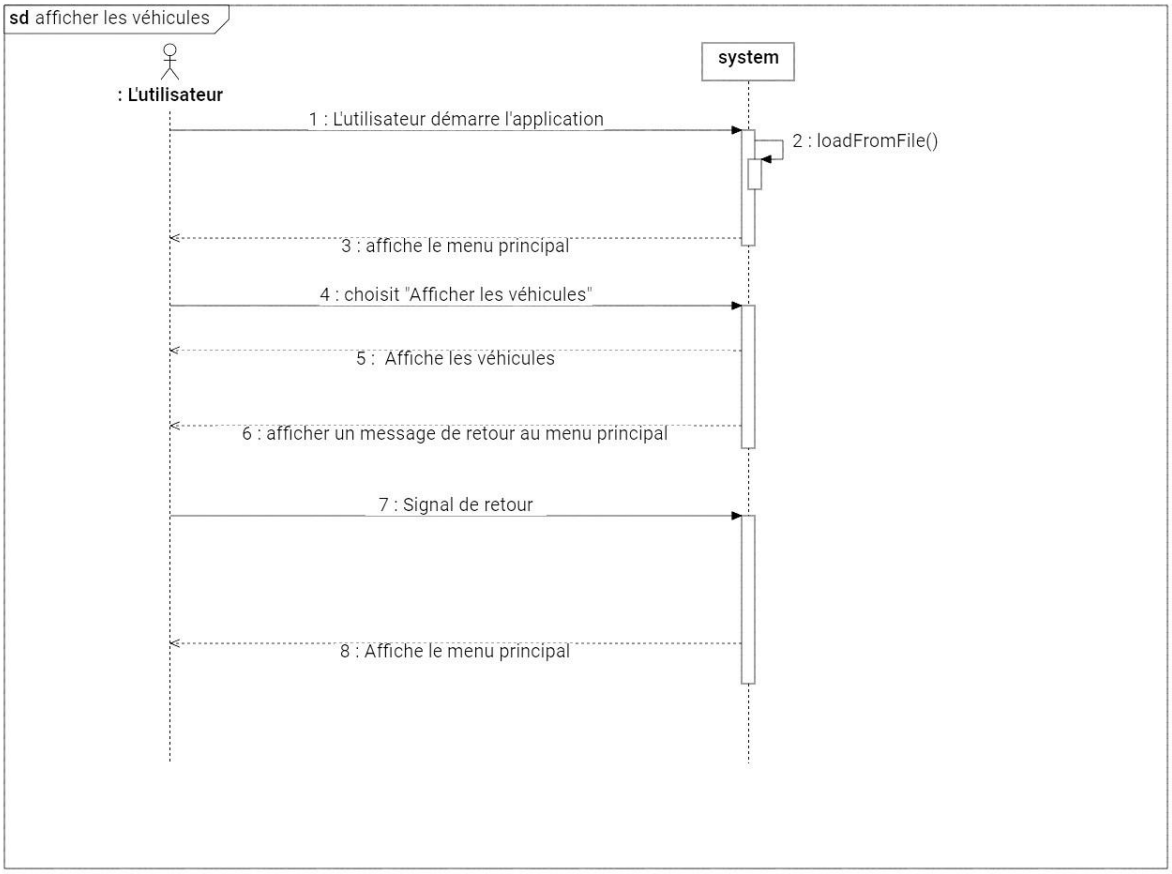
- Un diagramme de cas d'utilisation a été créé pour représenter les interactions entre les acteurs (utilisateurs) et le système de gestion de trafic routier.

2. Diagramme de Séquence:

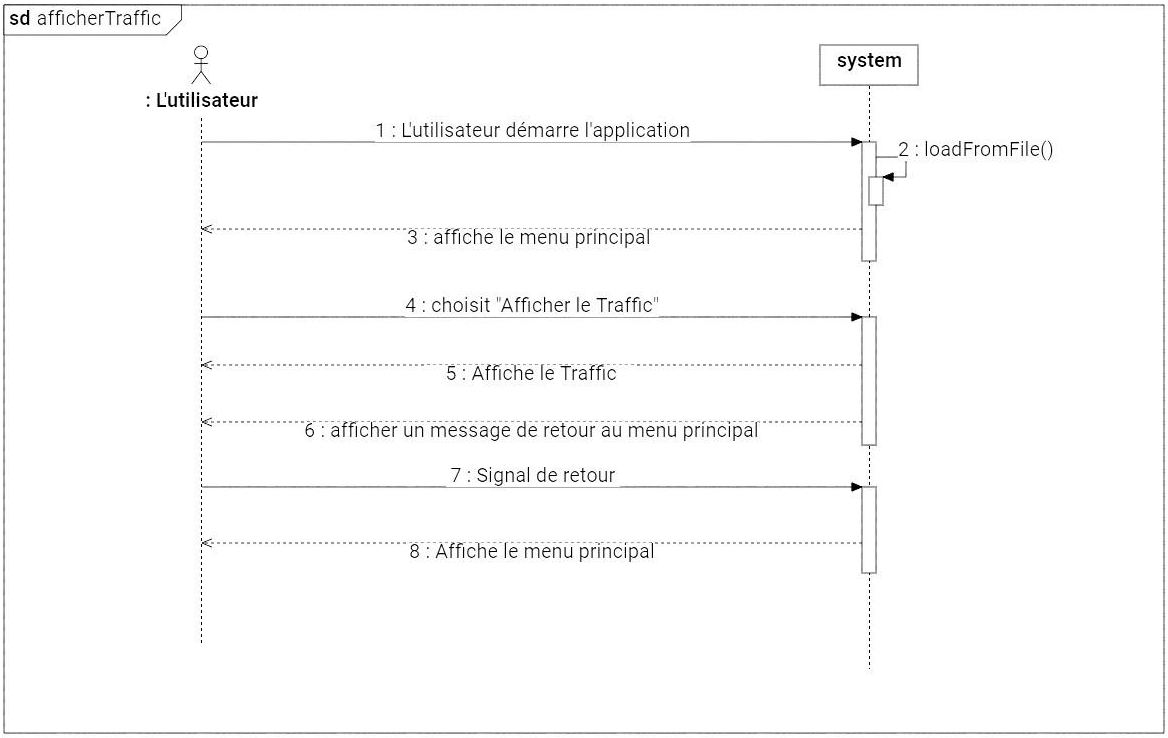
1- <u>ajouter une Véhicule :</u>



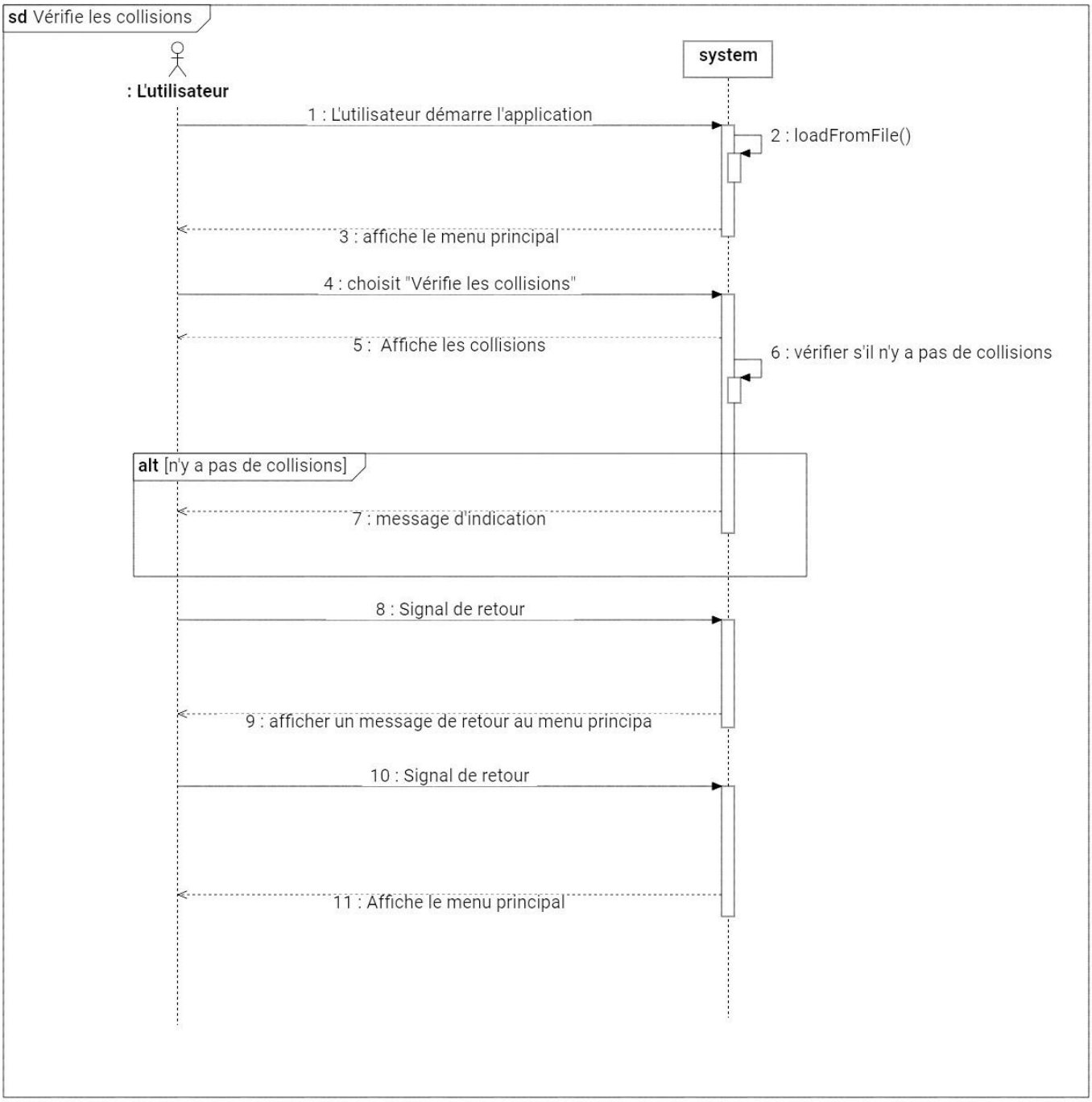
2- <u>afficher les véhicules :</u>



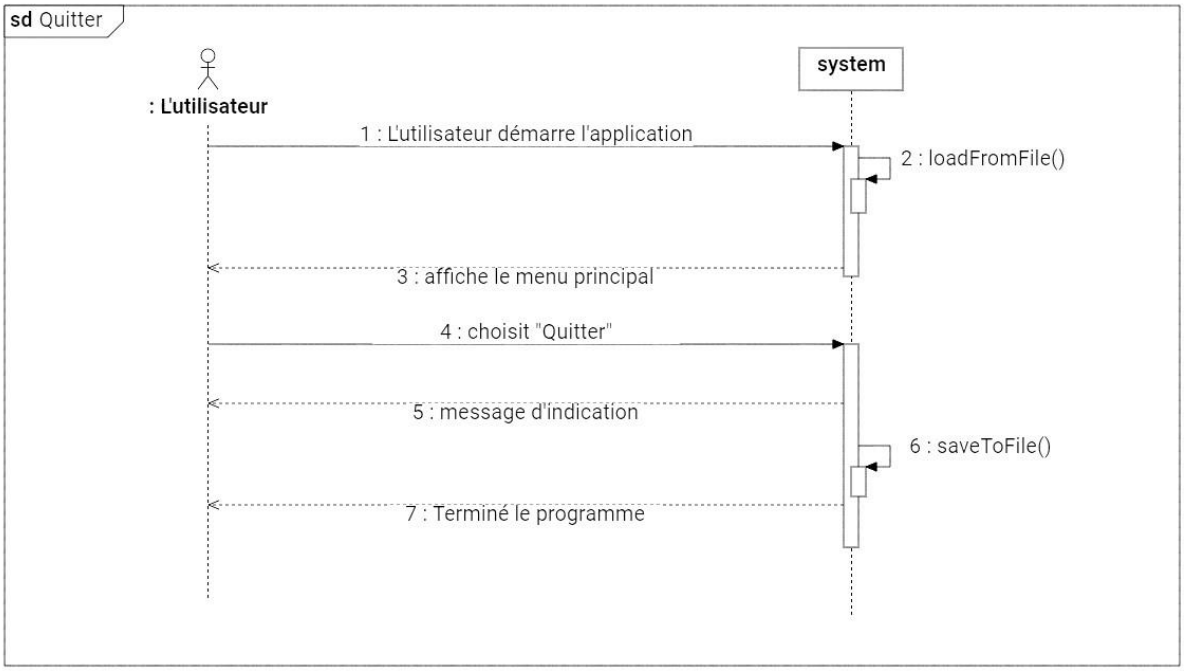
3- Afficher le Traffic :



4- <u>Vérifie les collisions :</u>

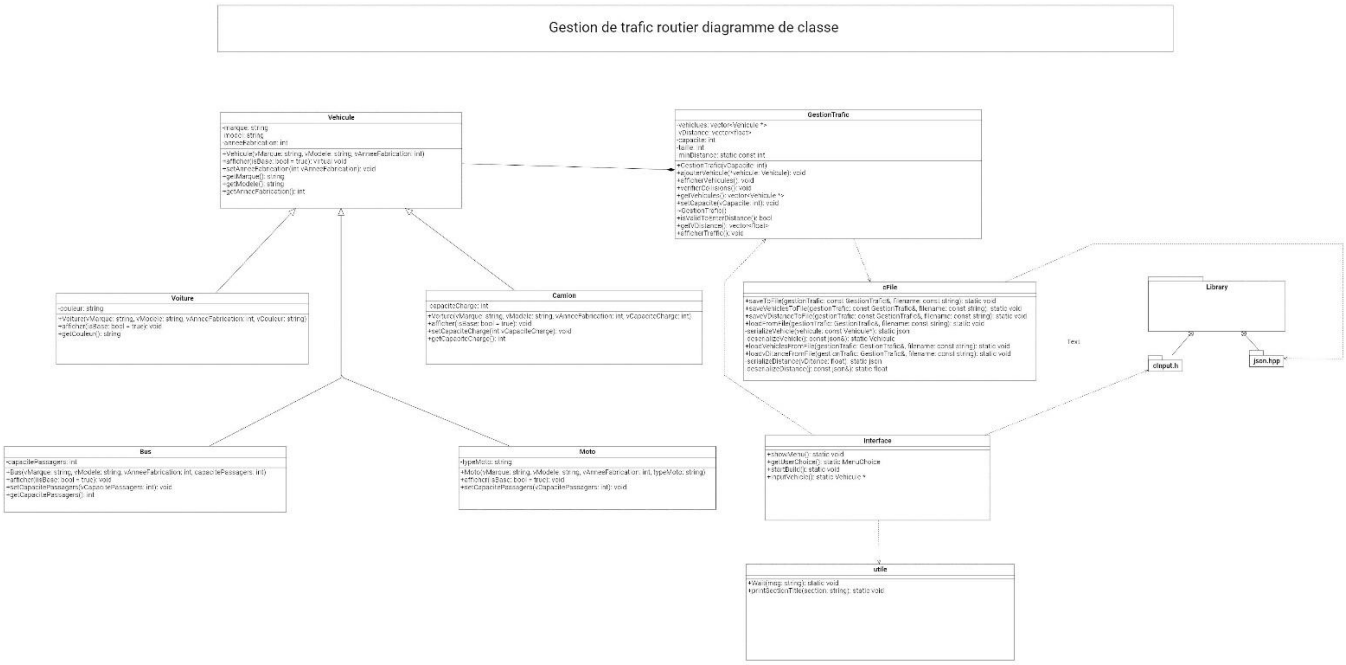


5- Quitter :

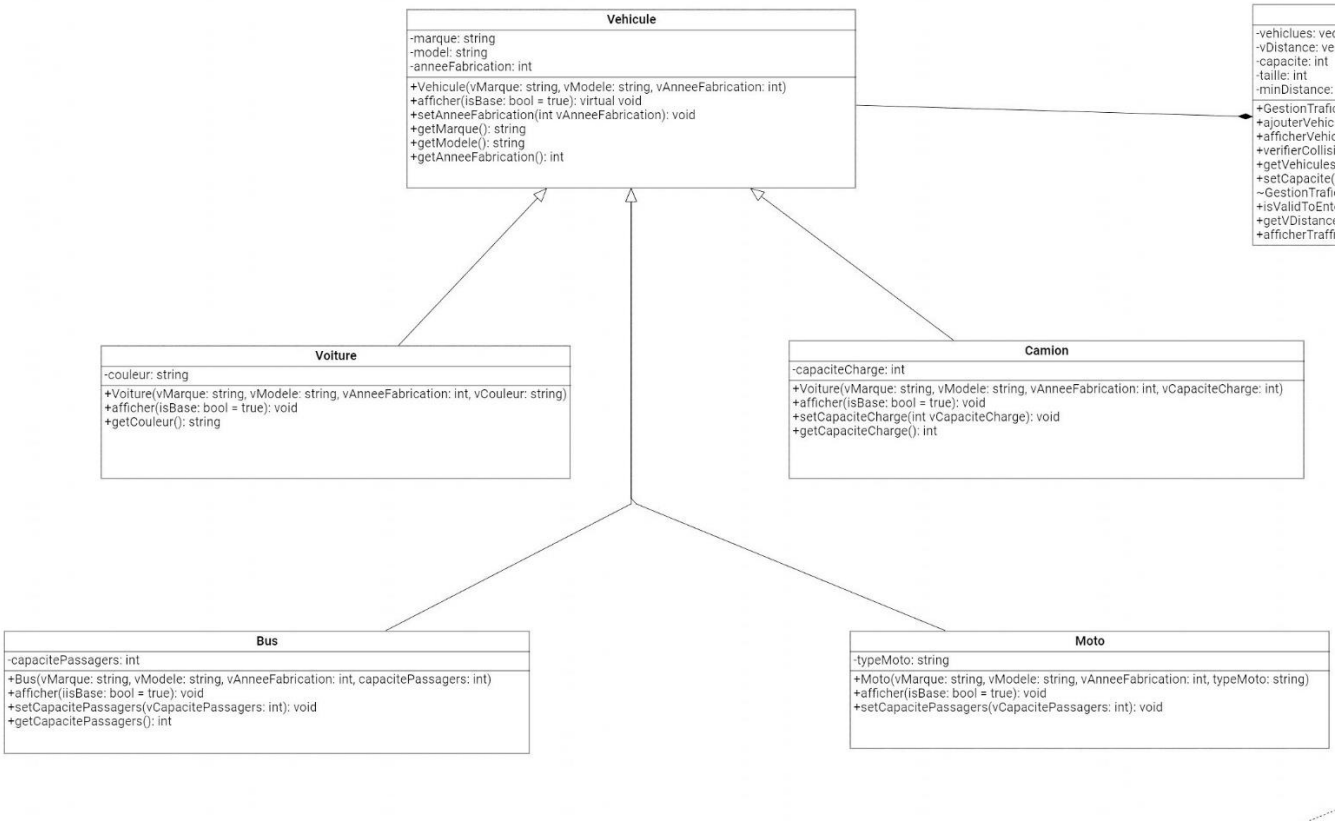


- Un diagramme de séquence a été conçu pour illustrer la séquence des actions entre les objets dans le système lors de l'ajout de véhicules, l'affichage des détails et la vérification des collisions.

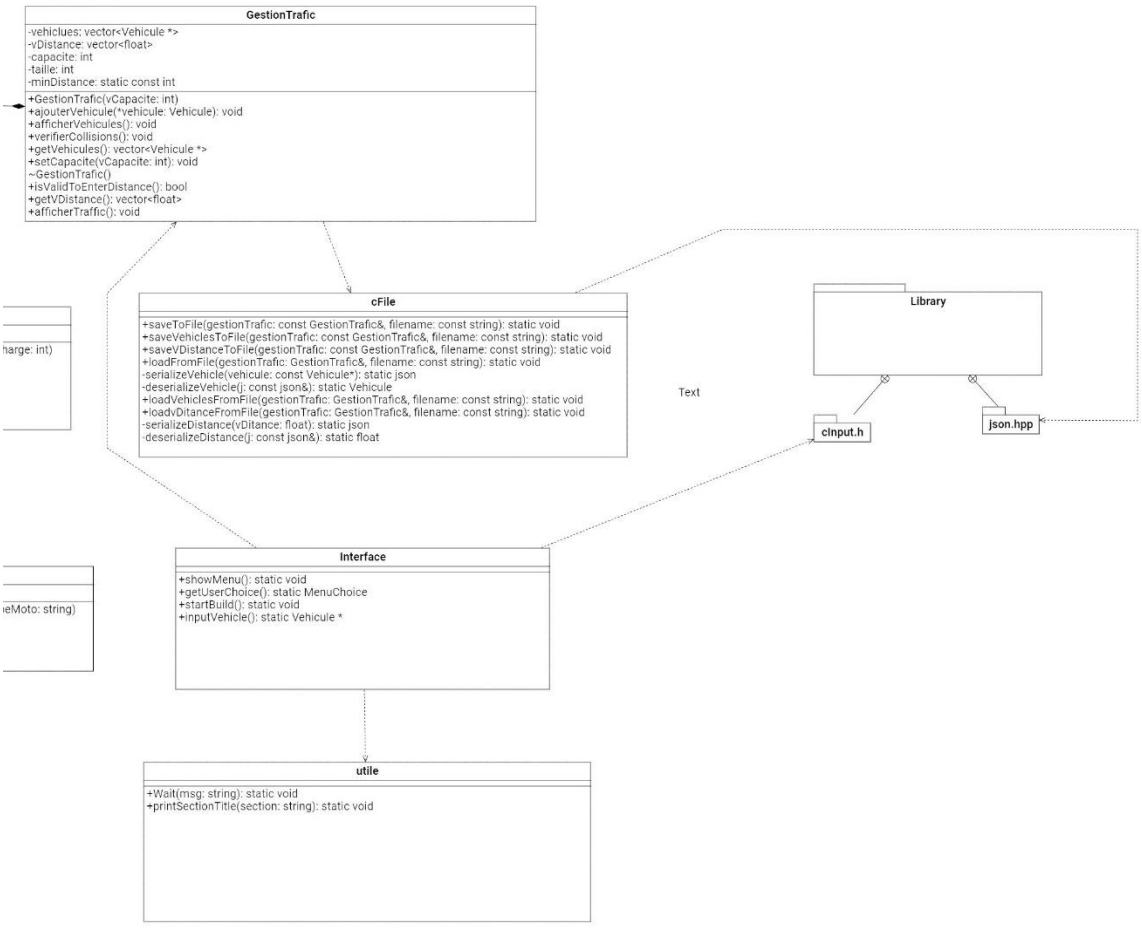
2- Diagramme de class:



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 :

```cpp
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
#include <iomanip>
#include "library/cInput.h"
#include "library/json.hpp"
using json = nlohmann::json;
using namespace std;

const string Line = "_____________________________________";
const string defaultStr = "unknown";
const int defaultNum = 0;
const string vehiculesFileName = "file/vehicles.json";
const string distanceFileName = "file/vDistance.json";
```

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.

2- Classe utile :

```cpp
class utile
{
public:
    static void Wait(string msg)
    {

        cout << msg;
        system("pause>0");
    }
    // Method to display the menu
    static void printSectionTitle(string section)
    {
        cout << setw(70) << "\n======================================================================\n";

        cout << "                          " << section << "                          \n";
        cout << setw(70) << "======================================================================\n";
    }
};

```

- 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.

3- Class véhicule :

```cpp
class Vehicule
{
protected:
    string marque;
    string modele;
    int anneeFabrication;

public:
    Vehicule(string vMarque = defaultStr, string vModele = defaultStr, int vAnneeFabrication = defaultNum)
        : marque(vMarque), modele(vModele)
    {
        setAnneeFabrication(vAnneeFabrication);
    }


    virtual void afficher(bool isBase = true) const
    {
        if (isBase)
            cout << Line << "\n";
        cout << "Marque: " << marque << "\n";
        cout << "Modèle: " << modele << "\n";
        cout << "Année de fabrication: " << anneeFabrication << "\n";
        if (isBase)
            cout << Line << endl;
    }
    void setAnneeFabrication(int vAnneeFabrication)
    {

        anneeFabrication = vAnneeFabrication >= 0 ? vAnneeFabrication : 0;
    }
    string getMarque() const
    {
        return marque;
    }
    string getModele() const
    {
        return modele;
    }
    int getAnneeFabrication() const
    {
        return anneeFabrication;
    }
};
```

4- Classe voiture :

```cpp
// Derived class representing a car
class Voiture : public Vehicule
{
private:
    string couleur;

public:
    Voiture(string vMarque = defaultStr, string vModele = defaultStr, int vAnneeFabrication = defaultNum, string vCouleur = defaultStr)
        : Vehicule(vMarque, vModele, vAnneeFabrication), couleur(vCouleur) {}

    void afficher(bool isBase = true) const override
    {
        if (isBase)
            cout << Line << "\n";
        cout << "--- Voiture  ---\n";
        Vehicule::afficher(false);
        cout << "Couleur: " << couleur << endl;
        if (isBase)
            cout << Line << "\n";
    }

    string getCouleur() const
    {
        return couleur;
    }
};
```

- 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.

5- Classe GestionTraffic :

```cpp
class GestionTrafic
{
private:
    vector<Vehicule *> vehicules;
    vector<float> vDistance;
    static const int minDistance;
    int capacite;
    int taille;

public:
    GestionTrafic(int vCapacite) : taille(0)
    {
        setCapacite(vCapacite);
    }

    void ajouterVehicule(Vehicule *vehicule)
    {
        if (vehicules.size() < capacite)
        {
            vehicules.push_back(vehicule);
            taille++;
        }
        else
        {
            cout << "Le trafic est plein, impossible d'ajouter plus de véhicules." << endl;
        }
    }
    bool isValidToEnterDistance() const
    {
        return vDistance.size() < vehicules.size() && vehicules.size() >= 2;
    }
    void ajotuerDistance(float distanceBetweenPrevVeh = minDistance)
    {

        if (isValidToEnterDistance())
        {
            vDistance.push_back(distanceBetweenPrevVeh);
        }
    }

    void afficherVehicules() const
    {
        int index = 0;
        for (const Vehicule *v : vehicules)
        {
            cout << "\n v[" << ++index << "]"
                 << "\n";
            v->afficher();
        }
    }
    void afficherTraffic() const
    {
        if (!isValidToEnterDistance())
        {
            cout << "l n'existe pas un nombre suffisant de voitures pour l'affichage :\n";
        }
        else
        {
            for (size_t i = 0; i < vDistance.size(); i++)
            {
                cout << " ==>  v[ " << setw(20) << left << vehicules[i]->getMarque() << " ]   "
                     << "   --- [ " << setw(8) << vDistance[i] << " m ]"
                     << " ---     v[ " << setw(20) << left << vehicules[i + 1]->getMarque() << " ] "
                     << "\n";
            }
            cout << endl;
        }
    }

    void verifierCollisions() const
    {
        bool isCollisions = false;
        for (size_t i = 0; i < vDistance.size(); i++)
        {

            if (vDistance[i] < minDistance)
            {
                isCollisions = true;
                cout << "\n"
                     << Line << Line << "\n";
                cout << "Collision détectée entre :\n ";
                vehicules[i]->afficher(false);
                cout << "\n---------- et---------- Distance : [ " << vDistance[i] << " m ] \n\n ";
                vehicules[i + 1]->afficher(false);
                cout << Line << Line << endl;
            }
        }
        if (!isCollisions)
        {

            utile::Wait("Il n'y a pas de collections dans le système: ");
        }
    }
    vector<Vehicule *> getVehicules() const
    {
        return vehicules;
    }

    vector<float> getVDistance() const
    {
        return vDistance;
    }

    void setCapacite(int vCapacite)
    {
        capacite = vCapacite >= 0 ? vCapacite : 0;
    }

    ~GestionTrafic()
    {
        for (Vehicule *v : vehicules)
        {
            delete v;
        }
    }
};

const int GestionTrafic::minDistance = 5;
```

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.

6- classe cFile :

```cpp

class cFile
{
public:
    static void saveVehiclesToFile(const GestionTrafic &gestionTrafic, const string &filename = vehiculesFileName)
    {
        json vehiclesJson;
        for (const Vehicule *vehicle : gestionTrafic.getVehicules())
        {
            vehiclesJson.push_back(serializeVehicle(vehicle));
        }
        ofstream outputFile(filename, ios::out | ios::trunc);
        outputFile << vehiclesJson.dump(4); // Dump JSON with indentation for readability
        outputFile.close();
    }
    static void saveVDistanceToFile(const GestionTrafic &gestionTrafic, const string &filename = distanceFileName)
    {
        json distanceJson;
        for (const float &distance : gestionTrafic.getVDistance())
        {
            distanceJson.push_back(serializeDistance(distance));
        }
        ofstream outputFile(filename, ios::out | ios::trunc);
        outputFile << distanceJson.dump(4); // Dump JSON with indentation for readability
        outputFile.close();
    }

    static void saveToFile(const GestionTrafic &gestionTrafic, const string &vVehiculesFileName = vehiculesFileName, const string &vDistanceFilename = distanceFileName)
    {
        saveVehiclesToFile(gestionTrafic);
        saveVDistanceToFile(gestionTrafic);
    }

    static void loadVehiclesFromFile(GestionTrafic &gestionTrafic, const string &filename = vehiculesFileName)
    {
        ifstream inputFile(filename, ios::in); // Open for reading

        if (inputFile.is_open())
        {
            // Check if the file is empty
            inputFile.seekg(0, ios::end); // Move get pointer to end of file
            if (inputFile.tellg() == 0)   // If the position is 0, the file is empty
                return;

            inputFile.seekg(0, ios::beg); // Move get pointer back to beginning of file

            json vehiclesJson;
            inputFile >> vehiclesJson;

            for (const auto &vehicleJson : vehiclesJson)
            {
                gestionTrafic.ajouterVehicule(deserializeVehicle(vehicleJson));
            }

            inputFile.close(); // Close the file stream
        }
        else
        {
            cout << "Impossible d'ouvrir le fichier: " << filename << endl;
        }
    }
    static void loadvDistanceFromFile(GestionTrafic &gestionTrafic, const string &filename = distanceFileName)
    {
        ifstream inputFile(filename, ios::in); // Open for reading

        if (inputFile.is_open())
        {
            // Check if the file is empty
            inputFile.seekg(0, ios::end); // Move get pointer to end of file
            if (inputFile.tellg() == 0)   // If the position is 0, the file is empty
                return;

            inputFile.seekg(0, ios::beg); // Move get pointer back to beginning of file

            json distanceJson;
            inputFile >> distanceJson;

            for (const auto &ditanceItem : distanceJson)
            {
                gestionTrafic.ajoturDistance(deserializeDistance(ditanceItem));
            }

            inputFile.close(); // Close the file stream
        }
        else
        {
            cout << "Impossible d'ouvrir le fichier: " << filename << endl;
        }
    }
    static void loadFromFile(GestionTrafic &gestionTrafic, const string &vVehiculesFileName = vehiculesFileName, const string &vDistanceFilename = distanceFileName)
    {
        loadVehiclesFromFile(gestionTrafic);
        loadvDistanceFromFile(gestionTrafic);
    }

private:
    static json serializeDistance(const float vDistance)
    {
        json j;
        j["distance"] = vDistance;
        return j;
    }
    static json serializeVehicle(const Vehicule *vehicle)
    {
        json j;
        j["marque"] = vehicle->getMarque();
        j["modele"] = vehicle->getModele();
        j["anneeFabrication"] = vehicle->getAnneeFabrication();

        // Serialize specific attributes for derived classes
        const Voiture *voiturePtr = dynamic_cast<const Voiture *>(vehicle);
        const Bus *busPtr = dynamic_cast<const Bus *>(vehicle);
        const Moto *motoPtr = dynamic_cast<const Moto *>(vehicle);
        const Camion *CamionPtr = dynamic_cast<const Camion *>(vehicle);

        if (voiturePtr)
        {
            j["type"] = "Voiture";
            j["couleur"] = voiturePtr->getCouleur();
        }
        else if (busPtr)
        {
            j["type"] = "Bus";
            j["capacitePassagers"] = busPtr->getCapacitePassagers();
        }
        else if (motoPtr)
        {
            j["type"] = "Moto";
            j["typeMoto"] = motoPtr->getTypeMoto();
        }
        else if (CamionPtr)
        {
            j["type"] = "Camion";
            j["capaciteCharge"] = CamionPtr->getCapaciteCharge();
        }

        return j;
    }

    static float deserializeDistance(const json &j)
    {

        return j["distance"];
    }

    static Vehicule *deserializeVehicle(const json &j)
    {
        string marque = j["marque"];
        string modele = j["modele"];
        int anneeFabrication = j["anneeFabrication"];
        string type = j["type"];

        if (type == "Voiture")
        {
            string couleur = j["couleur"];
            return new Voiture(marque, modele, anneeFabrication, couleur);
        }
        else if (type == "Bus")
        {
            int capacitePassagers = j["capacitePassagers"];
            return new Bus(marque, modele, anneeFabrication, capacitePassagers);
        }
        else if (type == "Moto")
        {
            string typeMoto = j["typeMoto"];
            return new Moto(marque, modele, anneeFabrication, typeMoto);
        }
        else if (type == "Camion")
        {
            int capaciteCharge = j["capaciteCharge"];
            return new Camion(marque, modele, anneeFabrication, capaciteCharge);
        }
        else
        {
            // Default to Vehicule class if type is not recognized
            return new Vehicule(marque, modele, anneeFabrication);
        }
    }
};
```

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.

7- Classe interface :

```cpp
enum MenuChoice
{
    ADD_VEHICLE = 1,
    DISPLAY_VEHICLES,
    DISPLAY_TRAFFIC,
    CHECK_COLLISIONS,
    EXIT
};
// Interface class for menu display and user input
class Interface
{
public:
    static void showMenu()
    {
        system("cls");
        utile::printSectionTitle(" Menu ");
        cout << "                    "
             << "1. Ajouter un véhicule" << endl;
        cout << "                    "
             << "2. Afficher les véhicules" << endl;
        cout << "                    "
             << "3. Afficher le Traffic" << endl;
        cout << "                    "
             << "4. Vérifier les collisions" << endl;
        cout << "                    "
             << "5. Quitter" << endl;
        cout << setw(70) << "=====================================================================\n";
    }

    // Method to get user choice
    static MenuChoice getUserChoice()
    {
        int choice = cInput::readIntegerInRange(1, MenuChoice::EXIT, "Votre choix :  ");

        return static_cast<MenuChoice>(choice);
    }

    static void startBuild()
    {
        GestionTrafic gestionTrafic(100);

        // Load vehicles from file
        cFile::loadFromFile(gestionTrafic);

        while (true)
        {
            showMenu();
            MenuChoice choice = getUserChoice();
            system("cls");

            switch (choice)
            {
            case ADD_VEHICLE:
            {
                utile::printSectionTitle(" AJOUTER UNE VEHICULE ");
                gestionTrafic.ajouterVehicule(inputVehicle());
                if (gestionTrafic.isValidToEnterDistance())
                    gestionTrafic.ajotuerDistance(inputDitance());

                break;
            }
            case DISPLAY_VEHICLES:
            {
                utile::printSectionTitle(" AFFICHER LES VEHICULES ");
                gestionTrafic.afficherVehicules();
                break;
            }
            case DISPLAY_TRAFFIC:
            {
                utile::printSectionTitle(" AFFICHER LE TRAFFIC ");
                gestionTrafic.afficherTraffic();
                break;
            }
            case CHECK_COLLISIONS:
            {
                utile::printSectionTitle(" VERIFIER LES COLLISIONS ");
                gestionTrafic.verifierCollisions();
                break;
            }
            case EXIT:
            {
                utile::printSectionTitle(" Au revoir ! ");
                break;
            }
            default:
            {
                utile::printSectionTitle(" CHOIX NO VALIDE ");
                cout << "Choix invalide. Veuillez réessayer." << endl;
                break;
            }
            };

            if (choice != MenuChoice::EXIT)
                utile::Wait("\nAppuyez sur Entrée pour revenir au menu principal :");
            else
                break;
        }

        // save vehicles To file
        cFile::saveToFile(gestionTrafic);
    }

    static Vehicule *inputVehicle()
    {
        string marque, modele, inputPrompt;
        int anneeFabrication, type;

        marque = cInput::readString("Entrez la marque du véhicule : ");
        modele = cInput::readString("Entrez le modèle du véhicule : ");
        anneeFabrication = cInput::readPositiveIntegerNumber("Entrez l'année de fabrication du véhicule : ");

        inputPrompt = "Choisissez le type de véhicule :\n";
        inputPrompt += "1. Voiture\n";
        inputPrompt += "2. Bus\n";
        inputPrompt += "3. Moto\n";
        inputPrompt += "4. Camion\n";
        type = cInput::readIntegerInRange(1, 4, inputPrompt);

        switch (type)
        {
        case 1:
        {
            string couleur;
            couleur = cInput::readString("Entrez la couleur de la voiture : ");
            return new Voiture(marque, modele, anneeFabrication, couleur);
        }
        case 2:
        {
            int capacitePassagers;
            capacitePassagers = cInput::readPositiveIntegerNumber("Entrez la capacité de passagers du bus : ");
            return new Bus(marque, modele, anneeFabrication, capacitePassagers);
        }
        case 3:
        {
            string typeMoto;
            typeMoto = cInput::readString("Entrez le type de moto : ");
            return new Moto(marque, modele, anneeFabrication, typeMoto);
        }
        case 4:
        {
            int capaciteCharge;
            capaciteCharge = cInput::readPositiveIntegerNumber("Entrez la Capacite de charge de Camion : ");
            return new Camion(marque, modele, anneeFabrication, capaciteCharge);
        }
        default:
        {
            cout << "Type de véhicule invalide." << endl;
            return nullptr;
        }
        }
    }
    static float inputDitance()
    {
        return cInput::readPositiveFloat("Entrez la Distance entre cette voiture est la dernier (m) : ", true, "La Distance doit entre un nombre positive : ");
    }
};
```

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.

8- Main :

```cpp
int main()
{
    try
    {
        Interface::startBuild();
    }
    catch (const runtime_error &e)
    {
        system("cls");
        cerr << "\nCaught exception: " << e.what() << endl;
        cin.clear();
    }

    return 0;
}

```

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.