Rapport de Projet

Simulateur de Smart City – Module Smart Parking & EV Charging

- **Année universitaire :** 2025–2026
- **Filière :** Licence IDAI
- **Matière :** Programmation Orientée Objet en C++
- **Encadrante :** Pr. Ikram BEN ABDEL OUAHAB

Membre du groupe :

MOHAMED ZARKIK
IMANE RHANEBOU
ABDALLAOUI ALAOUI MOHAMED

1. Introduction

Dans le cadre du module de Programmation Orientée Objet en C++, nous avons réalisé un projet portant sur la simulation d'une Smart City. Notre travail concerne le sous-projet Smart Parking & EV Charging.
L'objectif de ce projet est de créer une application capable de simuler la gestion des parkings dans une ville, en prenant en compte les véhicules classiques et les véhicules électriques. Ce projet nous a permis d'appliquer les notions vues en cours de POO, telles que les classes, l'héritage et l'organisation du code.

2. Objectifs du sous-projet Smart Parking

*Les objectifs principaux de notre module sont :

- Simuler la gestion intelligente des parkings urbains.
- Gérer la capacité, le taux d'occupation et la saturation des parkings.
- Simuler la recherche de stationnement selon différentes stratégies.
- Intégrer la gestion des véhicules électriques avec recharge.
- Offrir une visualisation claire et interactive de la simulation.

3. Technologies et outils utilisés

- Pour réaliser ce projet, nous avons utilisé les outils suivants :
- C++ comme langage de programmation
- Programmation Orientée Objet (POO)
- Raylib pour l'affichage graphique
- JSON pour la configuration
- Makefile pour la compilation
- Tests unitaires simples pour vérifier certaines fonctionnalités

4. Architecture du projet

L'architecture du projet respecte la structure réelle utilisée dans notre implémentation. Le code est organisé de manière simple et claire, comme suit :

- include/ : contient les fichiers d'en-tête (.hpp) qui déclarent les classes du projet (Game, structures, etc.).
- src/ : contient les fichiers source (.cpp) qui implémentent la logique principale de la simulation.
- demos/ : contient le fichier main.cpp, qui représente le point d'entrée du programme et lance la simulation.
- config/ : contient les fichiers de configuration au format JSON, utilisés pour paramétrer la simulation sans modifier le code.
- images/ : contient les ressources graphiques utilisées par Raylib (véhicules, parkings, arrière-plan, etc.).
- tests/ : contient les fichiers de tests permettant de vérifier le bon fonctionnement de certaines classes.
- Makefile : permet de compiler facilement le projet.

5. Conception orientée objet

La conception du module Smart Parking repose directement sur les classes réellement implémentées dans le projet.

5.1 Classes principales (issues du code)

- Game : classe centrale de la simulation. Elle gère la boucle principale, le chargement de la configuration JSON, l'initialisation de Raylib, la mise à jour des objets (véhicules, parkings) et l'affichage graphique.

- ❖ Vehicle : structure/classe représentant un véhicule générique avec ses attributs essentiels (position, état, déplacement).
- ❖ ElectricVehicle (EV) : extension du comportement des véhicules avec la gestion du niveau de batterie et la nécessité de recharge.
- ❖ Parking : classe responsable de la gestion des parkings (capacité maximale, places disponibles, entrées et sorties des véhicules).
- ❖ ChargingStation : gère la recharge des véhicules électriques, le temps de recharge et les files d'attente.
- ❖

5.2 Principes POO appliqués dans le code

- ❖ Encapsulation : les attributs des classes sont protégés et manipulés via des méthodes.
- ❖ Héritage : les véhicules électriques étendent le comportement des véhicules standards.
- ❖ Polymorphisme : les comportements diffèrent selon le type de véhicule.
- ❖ Responsabilité unique : chaque classe gère un rôle précis de la simulation.

6. Design Patterns utilisés

1. Singleton : la classe Game agit comme point central unique de la simulation.
2. Strategy (simplifié) : la logique de choix de parking varie selon la disponibilité et la saturation.

Ces choix correspondent à l'implémentation réelle du projet et assurent une bonne extensibilité.

7. Fonctionnalités implémentées

- Le module Smart Parking propose les fonctionnalités suivantes :
- Gestion de plusieurs parkings avec des capacités différentes.
- Détection des parkings pleins (saturation).
- Entrée et sortie des véhicules.
- Gestion des véhicules électriques avec recharge.
- Affichage en temps réel de l'état des parkings et des véhicules.
-

8. Visualisation avec Raylib

- La bibliothèque Raylib a été utilisée pour :
- Afficher les parkings et leur état (libre, saturé).
- Représenter les véhicules en mouvement.
- Montrer visuellement la recherche de stationnement.

- Offrir une démonstration interactive et pédagogique.

9. Scénario de démonstration

- Le scénario de démonstration comprend :
- Deux parkings de capacités différentes.
- Un parking volontairement saturé.
- Plusieurs véhicules cherchant une place selon leurs préférences.
- Un véhicule électrique utilisant une station de recharge.

Ce scénario permet d'observer le comportement du système en temps réel.

10. Tests unitaires

- Des tests unitaires ont été réalisés pour vérifier :
- La gestion correcte de la capacité des parkings.
- Les entrées et sorties des véhicules.
- Le bon fonctionnement de la recharge des EV.

Ces tests assurent la fiabilité du système.

11. Conclusion

Ce projet nous a permis de mieux comprendre et appliquer les concepts de la programmation orientée objet en C++. La réalisation du module Smart Parking nous a aidés à travailler sur un projet structuré, proche d'un cas réel.

La simulation est fonctionnelle et respecte les consignes demandées. Ce travail pourra être amélioré dans le futur en intégrant les autres modules de la Smart City afin d'obtenir une simulation plus complète.

Les Outils utilisé :

- **Raylib** est une bibliothèque graphique open-source en C, conçue pour créer des jeux 2D et 3D facilement. Elle est simple à utiliser, légère et idéale pour les débutants, tout en offrant des fonctionnalités puissantes pour les développeurs expérimentés.

- **Visual Studio Code** est un éditeur de code extensible développé par Microsoft pour Windows, Linux et macOS.

- **GitHub** est une plateforme de gestion de code basée sur Git, permettant de collaborer, versionner et héberger des projets. Elle est idéale pour les développeurs et équipes pour partager, réviser et déployer du code.