Rapport du sous projet :
Emergency & Priority Management
POO en C++

Realisé par groupe N° 15 :
Kerouad Loubna , Chentouf Ayoub,
Lmasoudi Aya, Charaf eddine Ifrinchaou,
Hmamouchi Ayoub

Description générale du projet

Notre projet consiste à réaliser un **simulateur d'une Smart City en C++**, en utilisant les principes de la **programmation orientée objet (POO)** ainsi que la bibliothèque graphique **Raylib** pour l'affichage et la simulation en temps réel.
L'objectif global du projet est de simuler le fonctionnement d'une ville intelligente, notamment la circulation des véhicules, les routes, les intersections et les comportements dynamiques des différents éléments de la ville.

Sous-projet 5 : Emergency & Priority Management

Dans le cadre de ce projet, notre groupe a été assigné au **Sous-projet 5 : Emergency & Priority Management**.
Ce sous-projet a pour objectif principal de **gérer le comportement des véhicules normaux face aux véhicules d'urgence**, comme les ambulances.
Lorsque qu'un véhicule d'urgence circule avec sa sirène activée, les autres véhicules doivent :

- détecter sa présence,
- lui céder le passage,
- modifier temporairement leur position ou leur comportement afin de faciliter sa progression.

Cela permet de simuler un comportement réaliste de priorité dans une Smart City.

Organisation du travail

Le travail a été divisé en plusieurs classes afin de respecter les principes de la **POO** et de rendre le code plus clair et réutilisable.
Parmi les classes principales utilisées dans notre sous-projet, on trouve :

- **CityMap** : gère la carte de la ville et le calcul des chemins.
- **Vehicle** : représente les véhicules normaux et leur déplacement.
- **EmergencyVehicle** : hérite de la classe Vehicle et ajoute la gestion de la sirène et de la priorité.
- **TrafficLights** : gère le comportement des feux de circulation.

Chaque classe a une responsabilité bien définie, ce qui facilite la maintenance et l'évolution du projet.

Outils et méthodes utilisés

Pour le développement, nous avons utilisé :

- **Visual Studio Code (VS Code)** comme environnement de développement,
- **C++** comme langage principal,
- **Raylib** pour la visualisation 3D et l'animation de la simulation.

Nous avons appliqué les notions principales de la **programmation orientée objet**, telles que :

- l'encapsulation,
- l'héritage,
- le polymorphisme,
- la séparation des responsabilités entre les classes.

La Carte (Map) :

--->Fichier GameCommon.h
Ce fichier regroupe toutes les définitions communes au projet :
L'énumération TileType (routes, herbe, intersections, parkings, etc.)
L'énumération FacilityType (hôpital, police, école, restaurant…)
L'énumération FacingDirection pour l'orientation des bâtiments
Des constantes globales comme la taille d'une tuile (TILE = 4.0f)
Il permet d'éviter la duplication du code et garantit la cohérence globale du projet.
--->Classe CityMap ---- Rôle
La classe CityMap représente l'environnement urbain 3D. Elle est responsable de :
La gestion de la grille de la ville
Le dessin des routes et du sol
La création et l'affichage des bâtiments
La conversion entre coordonnées monde et coordonnées grille
Le calcul des chemins pour les véhicules
------Représentation de la Carte
La carte est représentée par deux matrices 10x10 :
tileMap : décrit le type de terrain (route, herbe, intersection…)

facilityMap : indique les bâtiments présents sur chaque tuile
Cette séparation facilite la gestion de la logique de navigation et du rendu graphique.
-----Système de Rendu
Chaque tuile est dessinée à l'aide d'un modèle cube texturé :
L'herbe est toujours dessinée comme base
Les routes sont dessinées par-dessus selon le type de tuile
Les bâtiments sont dessinés séparément à l'aide de deux méthodes :
DrawBoxBuilding() pour les bâtiments simples
DrawTallBuilding() pour les bâtiments à plusieurs étages
Les textures sont chargées une seule fois afin d'optimiser les performances.
Boucle Principale (main.cpp)

La fonction main() :

Initialise la fenêtre et la caméra

Crée les objets CityMap et Simulation

Gère les déplacements de la caméra

Exécute la boucle principale de rendu et de mise à jour

Le rendu 3D est réalisé à l'aide de BeginMode3D() et EndMode3D().

Classe Vehicle

La classe Vehicle représente un véhicule normal qui se déplace dans la ville en suivant un chemin. Elle est utilisée comme base pour les autres types de véhicules.

Attributs

- position : position actuelle du véhicule.
- destination : point à atteindre.
- speed : vitesse du véhicule.
- baseSpeed : vitesse initiale.
- color : couleur du véhicule.
- path : liste des points du chemin.
- currentWp : point actuel du chemin.
- rotation : orientation du véhicule pour l'affichage.

Constructeur

Le constructeur initialise la position de départ, la destination, la vitesse et la couleur du véhicule.

Méthode setPath

Cette méthode permet d'affecter un nouveau chemin au véhicule et de réinitialiser le point courant.

Méthode update

La méthode update gère le déplacement du véhicule :

- le véhicule se dirige vers le waypoint courant,
- lorsqu'il est assez proche, il passe au waypoint suivant,
- la rotation est calculée pour que le véhicule regarde dans la bonne direction,
- la position est mise à jour en fonction de la vitesse et du temps.

Méthode draw

Cette méthode affiche le véhicule :

- une matrice de transformation est utilisée pour la position et la rotation,
- le véhicule est représenté par un cube,
- un contour est dessiné pour améliorer la visibilité.

Méthode yieldTo

Cette méthode permet au véhicule de **céder le passage** à un véhicule d'urgence :

- si un véhicule d'urgence est proche et devant,
- le véhicule se décale légèrement sur le côté,
- cela simule un comportement réaliste de priorité.

Classe EmergencyVehicle

La classe EmergencyVehicle hérite de la classe Vehicle. Elle représente les véhicules d'urgence comme les ambulances.

Attributs spécifiques

- type : type du véhicule d'urgence.
- isSirenActive : état de la sirène.
- sirenMultiplier : multiplicateur de vitesse avec sirène.
- normalMultiplier : multiplicateur de vitesse normal.

Constructeur

Le constructeur initialise le véhicule d'urgence et définit les multiplicateurs de vitesse.

Méthode setSirenActive

Cette méthode active ou désactive la sirène :

- quand la sirène est activée, la vitesse augmente,
- sinon, le véhicule revient à sa vitesse normale.

Méthode assignIncident

Cette méthode permet d'envoyer le véhicule vers un incident :

- la sirène est activée,
- le chemin le plus rapide est calculé à l'aide de CityMap,
- le chemin est ensuite assigné au véhicule.

Méthode update

La méthode update réutilise simplement le comportement de la classe Vehicle.

Méthode draw

Cette méthode affiche le véhicule d'urgence :

- le véhicule de base est dessiné,
- si la sirène est active, un cube clignotant rouge/bleu est affiché sur le toit.

Méthode getForwardDir

Cette méthode calcule la direction vers laquelle le véhicule est orienté.

Conclusion

Les classes Vehicle et EmergencyVehicle permettent de gérer le déplacement des véhicules dans la ville.
L'héritage permet de réutiliser le code du véhicule de base tout en ajoutant un comportement spécifique pour les véhicules d'urgence, comme la sirène, la vitesse prioritaire et la priorité sur les autres véhicules.

Classe Simulation

La classe **Simulation** est responsable de la gestion globale de la simulation du trafic dans la Smart City.
Elle sert de lien entre la carte de la ville (CityMap), les véhicules normaux (Vehicle) et le véhicule d'urgence (EmergencyVehicle).

Son rôle principal est d'initialiser les véhicules, définir leurs trajectoires et contrôler l'état général de la simulation (déplacement ou pause).

Attributs principaux

- **EmergencyVehicle* ambulance** : représente le véhicule d'urgence principal de la simulation (ambulance).
- **std::vector<Vehicle*> normalTraffic** : contient l'ensemble des véhicules normaux qui circulent dans la ville.
- **CityMap* cityMap** : pointeur vers la carte de la ville utilisée pour le calcul des chemins.
- **bool isMoving** : indique si la simulation est en cours de mouvement ou en pause.
- **std::vector<Vector3> previewPath** : permet de stocker un chemin temporaire (par exemple pour l'affichage).
- **bool ambulanceMoving** : indique si l'ambulance est en déplacement.

Constructeur et destructeur

Le **constructeur Simulation(CityMap* map)** :

- initialise la carte de la ville,
- crée l'ambulance avec une position de départ,
- active la sirène de l'ambulance,
- initialise le trafic normal avec des véhicules prédéfinis,
- met la simulation en pause au démarrage.

Le **destructeur** libère la mémoire en supprimant l'ambulance et tous les véhicules normaux afin d'éviter les fuites mémoire.

Gestion des positions (fonction Tile)

La fonction **Tile(int x, int y)** permet de convertir des coordonnées de grille (x, y) en coordonnées réelles dans la scène 3D.
Elle facilite le placement des véhicules sur la carte et rend le code plus lisible.

Génération des chemins

La classe contient plusieurs fonctions qui génèrent des **trajets prédéfinis** pour les véhicules normaux :

- **GetTopLapPath()** : crée un circuit en haut de la carte.
- **GetBottomLapPath()** : crée un circuit en bas de la carte.
- **GetBigOuterPath()** : crée un grand circuit autour de la ville.

Chaque chemin est construit en combinant plusieurs segments calculés avec la fonction findFastestPath de la classe CityMap.

Initialisation des véhicules normaux

La fonction **SetupLoopingCar(int index)** permet de créer et configurer des véhicules normaux selon un index donné :

- chaque véhicule a une position de départ différente,
- une vitesse différente,
- une couleur différente,
- et un chemin spécifique à suivre en boucle.

Si un véhicule existe déjà à l'index donné, il est remplacé par un nouveau, ce qui permet de réinitialiser facilement le trafic.

Logique générale de la classe

La classe Simulation centralise toute la logique de la simulation :

- elle gère les véhicules normaux et le véhicule d'urgence,
- elle utilise la carte pour le calcul des chemins,
- elle permet de simuler un trafic réaliste avec des trajets continus,

- elle prépare le terrain pour la gestion des priorités, notamment quand l'ambulance est en mouvement.

Classe TrafficLight

La classe **TrafficLight** représente un **feu de circulation** dans la simulation de la Smart City.
Son rôle principal est de gérer l'état du feu (vert, jaune ou rouge), son changement automatique dans le temps et son affichage dans la scène 3D.

Cette classe permet de simuler le fonctionnement réel des feux de circulation dans une ville intelligente.

Attributs de la classe

- **Vector3 position** : indique la position du feu de circulation dans la ville.
- **LightState state** : représente l'état actuel du feu (GREEN, YELLOW ou RED).
- **float timer** : permet de mesurer le temps écoulé depuis le dernier changement d'état.
- **float switchTime** : définit la durée pendant laquelle le feu reste vert ou rouge avant de changer.

Constructeur

Le constructeur **TrafficLight(Vector3 pos, float switchT)** :

- initialise la position du feu,
- définit le temps de changement entre les états,
- met le feu en état **vert** au début de la simulation,
- initialise le compteur de temps à zéro.

Cela permet d'avoir un feu fonctionnel dès le lancement de la simulation.

Logique de mise à jour (update)

La méthode **update(float deltaTime)** gère le cycle normal du feu de circulation :

- **Vert → Jaune → Rouge → Vert**

Le changement d'état se fait en fonction du temps écoulé :

- le feu reste vert pendant un temps donné,

- passe ensuite au jaune pendant une courte durée,
- puis au rouge avant de revenir au vert.

Cette logique simule un comportement réaliste des feux de circulation.

Gestion de la priorité (véhicule d'urgence)

La méthode **forceGreen()** permet de forcer le feu à passer immédiatement au **vert**. Elle est utilisée lorsqu'un véhicule d'urgence, comme une ambulance, approche du carrefour afin de lui donner la priorité et éviter les arrêts inutiles.

Affichage du feu (draw)

La méthode **draw()** s'occupe de l'affichage du feu de circulation :

- un cylindre représente le poteau,
- un cube représente le boîtier du feu,
- une sphère colorée représente la lumière active (rouge, jaune ou verte).

La couleur affichée dépend directement de l'état actuel du feu.

Rôle dans le sous-projet

Dans le cadre du sous-projet **Emergency & Priority Management**, la classe TrafficLight joue un rôle important :

- elle régule la circulation normale,
- elle interagit avec les véhicules d'urgence,
- elle permet de simuler une gestion intelligente des priorités dans une Smart City.