PROJET:

SIMULATEUR D'UNE SMART CITY EN C++ (POO & Raylib)

INTRODUCTION:

Ce code met en place une simulation interactive de trafic routier avec Raylib, où l'utilisateur peut visualiser plusieurs cartes de routes, des intersections et des véhicules se déplaçant automatiquement selon des trajets générés sur un graphe routier. Il propose un système complet allant de la modélisation du réseau (routes, virages, carrefours) jusqu'à l'animation en temps réel des voitures, bus et autres types de véhicules.

Dessin les routes verticales, horizontales et les virages:

enum TurnType

- Rôle : Enumération qui décrit le type de virage:
- TL = Turn Left en haut (top-left)
- BL = bottom-left
- TR = top-right
- BR = bottom-right.
- Utilisation : Sert à dire dans quel coin se trouve le virage pour calculer le centre, les angles de l'arc, etc.

Classe Road

- Rôle : Classe de base générique pour toutes les routes.
- Contenu :
- Attributs protégés : width, height.
- Constructeur qui initialise largeur et hauteur.
- Destructeur virtuel pour permettre l'héritage.
- Utilisation : VerticalRoad, HorizontalRoad et TurnRoad héritent de Road pour partager la taille de la route.

Classe VerticalRoad

- Rôle : Représente un tronçon de route vertical avec sa bande blanche centrale.
- Attributs :
- width, height hérités de Road.
- nbrRoads : nombre de segments verticaux à dessiner dans les fonctions DrawRoad1/2/3.
- Méthodes :
- VerticalRoad(float width, float height, int nbr): initialise la route.
- Draw(float xPos, float yPos):
- Dessine un rectangle vertical gris (la route).
- Dessine au centre une ligne blanche (marquage axial).
- Getters : getHeight(), getWidth(), getNbrRoads().
- Lien avec le reste : utilisé dans DrawRoad1, DrawRoad2, DrawRoad3 pour créer les colonnes de routes.

Classe HorizontalRoad

- Rôle : Représente un tronçon de route horizontal avec sa bande blanche centrale.
- Attributs :
- Hérite width, height de Road.
- nbrRoads : nombre de segments horizontaux à dessiner.
- Méthodes :
- HorizontalRoad(float width, float height, int nbr): initialise la route.
- Draw(float xPos, float yPos) :
- Dessine un rectangle horizontal gris.
- Dessine une ligne blanche horizontale au milieu.
- Getters : getHeight(), getWidth(), getNbrRoads().
- Lien avec le reste : utilisé pour les axes horizontaux de chaque plan de route.

Classe TurnRoad

- Rôle : Représente un virage (quart de cercle) entre routes, avec les marquages discontinus.
- Attributs principaux :
- TurnType type : coin du virage (TL, BL, TR, BR).
- startAngle, endAngle : angles du quart de cercle en degrés.
- segments : précision de dessin du cercle raylib.
- innerRadius, outerRadius : rayon intérieur et extérieur de la route en arc.
- center : centre du cercle utilisé par DrawRing.
- nbrRoads (déclaré mais non exploité dans Draw).
- Constructeur :
- TurnRoad(float width, int segments, TurnType type, int nbr) :
- Initialise la largeur de la route circulaire.
- Fixe un innerRadius (100) et calcule outerRadius.
- Selon type, choisit startAngle / endAngle pour dessiner le bon quart de cercle.
- Méthode Draw :
- Calcule center en fonction du type et de la position (xPos, yPos).
- Dessine la chaussée courbe via DrawRing (anneau entre innerRadius et outerRadius).
- Dessine ensuite plusieurs petits arcs blancs (bandes discontinues) en jouant sur les angles.
- Getters : rayon intérieur, extérieur, largeur, angles, etc.
- Lien avec le reste : permet de connecter les VerticalRoad et HorizontalRoad dans les intersections et virages des plans 2 et 3.

Classe *Vehicle:*

1. Introduction:

- Cette classe Vehicle permet de **représenter différents types de véhicules** (voiture, bus, moto, ambulance) dans un programme graphique.
- Chaque véhicule peut **se déplacer automatiquement** sur un chemin prédéfini et **tourner pour suivre la direction** du mouvement.

2. Types de véhicules

enum VehicleType { CAR, BUS, MOTO, EMB };

- CAR : voiture normale
- BUS : autobus
- MOTO : moto
- EMB : ambulance

Chaque type a **une taille différente** (largeur et hauteur).

3. Propriétés principales

- height et width : dimensions du véhicule
- angle : orientation du véhicule (en degrés)
- color : couleur du véhicule
- carBody : rectangle représentant le corps du véhicule
- carHead : rectangle représentant le “capot” ou la partie avant
- path : chemin que le véhicule doit suivre
- autoMode : true si le véhicule se déplace automatiquement
- active : true si le véhicule est actif

4. Dessiner le véhicule

void Draw()

- DrawRectanglePro permet de **dessiner le rectangle du corps et de la tête du véhicule**.
- L’**origine** du rectangle est au centre du bas (carOrigin), pour que la rotation soit réaliste.
- La **rotation** (angle) permet au véhicule de **regarder dans la direction où il se déplace**.

5. Déplacement automatique

void FollowPath(float dt, TrafficLight &light, const vector<Vehicle> &cars)

- Le véhicule suit une série de **points (path)**.
- dir = direction vers le prochain point :

Vector2 dir = Vector2Normalize(Vector2Subtract(next, pos));

- atan2(dir.x, -dir.y) * RAD2DEG calcule **l’angle pour tourner le véhicule** vers la direction du mouvement.
- Le véhicule **avance seulement si le feu de circulation le permet**.
- Quand il atteint la fin du chemin, autoMode devient false.

6. Exemple d’utilisation

Vehicle car1(100, 200, RED, CAR); car1.SetPath({{100, 200}, {300, 400}, {500, 400}}); car1.FollowPath(dt, trafficLight, cars); car1.Draw();

- Crée une voiture rouge.
- Définit son chemin de déplacement.
- Déplace la voiture selon le temps écoulé (dt) et le feu de circulation.

- Dessine la voiture à l'écran.

Classe *Controller :*

Dans ce projet de simulation de traffic , la classe Controller joue un rôle central. Elle est responsable de la gestion des cartes routières, de la construction du réseau logique des routes et de la définition des points d'entrée des véhicules.

Contrairement au dessin des routes, qui est purement visuel, la classe Controller construit une structure logique appelée RoadGraph. Cette structure permet aux véhicules de se déplacer de manière autonome en suivant des chemins prédéfinis.

Ainsi, la simulation repose sur deux parties distinctes :

- Une partie graphique (affichage des routes)
- Une partie logique (déplacements des véhicules à l'aide d'un graphe)

I- Partie graphique

Rôle de la classe Controller

La classe **Controller** a plusieurs rôles importants :

- Gérer le plan routier actif (roadMap 1, 2 ou 3)
- Dessiner les routes à l'écran avec Raylib
- Construire le graphe routier utilisé par les véhicules
- Définir les points d'entrée des véhicules

Elle centralise donc toute la logique liée aux routes.

Attributs principaux

- **screenW, screenH** : largeur et hauteur de la fenêtre.
- **roadMap** : numéro du plan routier sélectionné.
- **graph** : graphe représentant les routes et leurs connexions.
- **entryNodes** : liste des points d'entrée des véhicules.

Méthode DrawRoad1

Rôle

La méthode **DrawRoad1** dessine un plan routier simple. Il est composé de deux routes verticales et d'une route horizontale centrale.

Description

- Deux routes verticales sont dessinées à gauche et à droite de l'écran.
- Une route horizontale traverse le centre de l'écran.
- Les routes sont dessinées en plusieurs segments pour former des axes continus.

Utilité

Ce plan est utilisé pour une simulation simple avec des intersections droites.

Méthode DrawRoad2

Rôle

La méthode **DrawRoad2** dessine un plan routier plus complexe avec des virages.

Description

- Une route verticale part du bas de l'écran.
- Un virage en haut à gauche (TL) relie la route verticale à une route horizontale.
- Un second virage en haut à droite (TR) permet de redescendre avec une autre route verticale.
- Une route horizontale inférieure complète le plan.

Utilité

Ce plan permet de tester les virages et les déplacements non linéaires des véhicules.

Méthode DrawRoad3

Rôle

La méthode DrawRoad3 dessine le plan routier le plus complexe du projet.

Description

- Deux routes horizontales (gauche et droite).
- Deux routes verticales au centre.
- Quatre virages (TL, TR, BR, BL) forment une boucle fermée.

L'ensemble ressemble à un carré ou à un rond-point rectangulaire.

Utilité

Ce plan est idéal pour tester les déplacements circulaires et un trafic plus dense.

Conclusion

La classe Controller est essentielle dans le projet Smart City. Elle permet de créer différents plans routiers avec des niveaux de complexité croissants. Les
méthodes DrawRoad1, DrawRoad2 et DrawRoad3 utilisent les classes de routes pour construire des réseaux cohérents, utilisés ensuite par les véhicules pour se déplacer automatiquement.

Cette organisation rend le projet clair, structuré et facile à faire évoluer.

II- Partie logique

Principe du RoadGraph

Le *RoadGraph* est un graphe composé de **nœuds** et de **connexions**.

Chaque nœud représente :

- Une position précise sur la route (coordonnées X et Y)
- Une liste de nœuds suivants possibles

Les connexions entre les nœuds définissent les **directions que les véhicules peuvent emprunter**. Lorsqu'un véhicule apparaît dans la simulation, un chemin est généré automatiquement à partir d'un nœud d'entrée. Le véhicule suit ensuite ce chemin point par point.

Ce système permet :

- Des trajets variés
- Des changements de direction
- Une circulation réaliste sans trajectoires fixes

MapRoad1 : Carrefour simple

La carte **MapRoad1** représente un carrefour classique composé de :

- Une route horizontale à double sens
- Deux routes verticales
- Plusieurs points d'entrée et de sortie

a. Route horizontale

La route horizontale est divisée en deux voies :

- Une voie supérieure (circulation de droite à gauche)

- Une voie inférieure (circulation de gauche à droite)

Chaque voie est construite à l'aide de plusieurs nœuds connectés entre eux, ce qui permet aux véhicules d'avancer progressivement sur la route.

b. Routes verticales

Deux routes verticales sont placées :

- Une au quart de la largeur de l'écran
- Une aux trois quarts de la largeur de l'écran

Ces routes permettent aux véhicules :

- De traverser le carrefour
- De tourner vers la route horizontale
- De quitter la simulation

c. Points d'entrée

Les points d'entrée sont définis aux extrémités des routes :

- À gauche et à droite de la route horizontale
- En haut et en bas des routes verticales

Ces points permettent l'apparition dynamique des véhicules dans la simulation.

MapRoad2 : Route avec virages

La carte **MapRoad2** introduit un élément plus avancé : les **virages courbes**.

a. Principe des virages

Les virages ne sont pas des courbes continues, mais une **succession de petits segments**. Ces segments sont calculés à partir de formules mathématiques utilisant les fonctions *cosinus* et *sinus*.

Chaque virage est donc représenté par plusieurs nœuds positionnés le long d'un arc de cercle.

b. Structure de la carte

La carte est composée de :

- Une route verticale en bas

- Un virage vers la droite
- Une longue route horizontale
- Une route verticale de sortie
- Un retour par la voie supérieure

Ce système permet une circulation fluide avec des changements de direction naturels.

c. Sens de circulation

Deux arcs sont utilisés :

- Un pour la voie inférieure
- Un pour la voie supérieure

Cela permet de simuler une circulation à double sens, même dans les virages.

MapRoad3 : Carrefour circulaire (rond-point)

La carte **MapRoad3** est la plus complexe du projet. Elle représente un **rond-point**, avec plusieurs routes qui y sont connectées.

a. Structure générale

La carte comprend :

- Une route horizontale à gauche
- Une route horizontale à droite
- Une route verticale en haut
- Une route verticale en bas
- Une voie circulaire centrale (rond-point)

b. Construction du rond-point

Le rond-point est construit à l'aide de **quatre arcs de cercle** :

- Bas gauche
- Bas droit
- Haut droit
- Haut gauche

Chaque arc est composé de plusieurs nœuds, ce qui permet aux véhicules de tourner progressivement autour du centre.

c. Connexions et circulation

Les routes extérieures sont connectées à la voie circulaire. Les véhicules peuvent :

- Entrer dans le rond-point
- Circuler autour du centre
- Sortir par une autre route

Ce comportement est obtenu uniquement grâce au graphe, sans règles complexes.

Points d'entrée des véhicules

Pour chaque carte, la classe *Controller* définit une liste de **points d'entrée**. Ces points correspondent aux endroits où les véhicules peuvent apparaître.

Cela permet :

- Une génération aléatoire des véhicules
- Une circulation variée
- Une simulation plus réaliste

Conclusion sur la classe Controller

La classe **Controller** constitue le cœur logique du système de circulation. Elle permet de créer plusieurs cartes routières différentes en utilisant une seule structure de graphe.

Grâce à cette approche :

- Le système est modulaire
- Les cartes sont facilement modifiables
- Le comportement des véhicules reste cohérent

Description détaillée de la fonction int main()

Introduction

La fonction main() constitue le point d'entrée principal du programme et représente le cœur de l'application de simulation de trafic urbain. Elle assure l'initialisation de l'environnement graphique, la configuration des paramètres de la simulation, la gestion du menu de démarrage, le contrôle des véhicules et du trafic, la gestion de l'interface utilisateur ainsi que le rendu graphique en temps réel. L'ensemble du comportement dynamique de la simulation est orchestré au sein de cette fonction.

1. Initialisation

La première étape de la fonction main() consiste à initialiser la fenêtre graphique à l'aide de la bibliothèque Raylib. Les dimensions de la fenêtre sont définies à 1800×1200 pixels et un titre est attribué afin d'identifier clairement l'application. La cadence d'exécution est fixée à 60 images par seconde, garantissant ainsi une animation fluide et une expérience visuelle réaliste. Cette étape est indispensable, car elle permet l'affichage de tous les éléments graphiques de la simulation.

2. Création du contrôleur et sélection de la carte

Un objet contrôleur est ensuite instancié. Ce contrôleur joue un rôle central, car il est chargé de dessiner les routes, de gérer les différentes cartes routières et de générer les graphes de circulation. Une variable interne permet d'identifier la carte routière active, ce qui autorise la sélection dynamique de plusieurs configurations de routes prédéfinies.

3. Construction du graphe de circulation

Après la création du contrôleur, le graphe de circulation est construit. L'ancien graphe est d'abord supprimé afin d'éviter toute incohérence. Ensuite, un nouveau graphe est généré en fonction de la carte routière sélectionnée. Dans ce graphe, chaque nœud représente un point de la route et chaque arête correspond à un chemin possible entre deux points. Ce graphe constitue la base logique utilisée pour le calcul des trajectoires des véhicules.

4. Récupération du graphe et des points d'entrée

La fonction récupère ensuite une référence vers le graphe de circulation ainsi que vers les points d'entrée. Les points d'entrée représentent les emplacements à partir desquels les véhicules peuvent apparaître, généralement situés aux bords de la carte. Tous les véhicules générés dans la simulation débutent leur trajet à partir de ces points, ce qui permet une distribution réaliste du trafic.

5. Initialisation des feux de circulation

Deux feux de circulation sont créés et intégrés à la scène. Chaque feu possède un état (rouge, orange ou vert) et change automatiquement de couleur en fonction du temps écoulé. Ces feux influencent directement le comportement des véhicules, qui s'arrêtent lorsque le feu est rouge, contribuant ainsi à la simulation d'un trafic urbain réaliste.

6. Gestion dynamique des véhicules

Les véhicules sont stockés dans une structure de données dynamique contenant l'ensemble des voitures présentes dans la simulation. Les véhicules sont ajoutés automatiquement au cours de l'exécution et sont supprimés lorsqu'ils atteignent la fin de leur trajet. Cette gestion dynamique permet de maintenir un nombre cohérent de véhicules dans la simulation.

7. Paramétrage des couleurs et des types de véhicules

L'interface utilisateur permet de configurer les couleurs et les types de véhicules autorisés dans la simulation. Plusieurs couleurs sont disponibles, telles que le rouge, le bleu, le vert, le gris, le rose et le jaune. De même, différents types de véhicules peuvent être activés ou désactivés, notamment les voitures, les motos, les bus et les véhicules d'urgence. Ces paramètres influencent directement la génération aléatoire des véhicules.

8. Paramètres de génération du trafic

La fonction définit ensuite les paramètres de génération du trafic, tels que le nombre actuel de véhicules, le nombre maximal autorisé et l'intervalle de création entre deux véhicules. L'intervalle est fixé à une seconde, ce qui permet de contrôler la densité du trafic et d'éviter une surcharge de la simulation.

9. Menu de démarrage de la simulation

Avant le lancement de la simulation principale, un menu de démarrage est affiché. Ce menu présente un écran d'accueil contenant le titre de la simulation ainsi qu'un bouton « START ». Tant que l'utilisateur n'a pas cliqué sur ce bouton, la simulation reste inactive. Une fois le bouton activé, la simulation démarre et le programme entre dans la boucle principale.

10. Boucle principale de la simulation

La boucle principale s'exécute en continu tant que la fenêtre graphique reste ouverte. Elle constitue le cœur dynamique de la simulation et assure la mise à jour permanente des éléments du système.

11. Gestion du changement de carte routière

Au cours de l'exécution, l'utilisateur peut changer la carte routière. Dans ce cas, le graphe de circulation est reconstruit, toutes les voitures existantes sont supprimées et la nouvelle carte est chargée afin d'assurer une cohérence totale de la simulation.

12. Génération automatique des véhicules

À intervalles réguliers, de nouveaux véhicules sont générés si la simulation n'est pas en pause et si le nombre maximal de véhicules n'est pas atteint. Chaque véhicule choisit un point d'entrée aléatoire, génère un chemin à partir du graphe et reçoit une couleur ainsi qu'un type autorisés par les paramètres définis.

13. Déplacement et comportement des véhicules

Chaque véhicule suit automatiquement le chemin qui lui a été attribué. Il s'oriente correctement, respecte les feux de circulation et progresse jusqu'à atteindre la fin de son trajet. Lorsqu'un véhicule termine son parcours, il est supprimé de la simulation.

14. Mise à jour des feux de circulation

Les feux de circulation sont mis à jour à chaque itération de la boucle principale. Leur état change en fonction du temps écoulé, ce qui permet de réguler le trafic de manière dynamique et réaliste.

15. Rendu graphique et affichage

À chaque image, l'écran est nettoyé puis l'ensemble des éléments graphiques est dessiné. Cela inclut les routes, les véhicules, les feux de circulation ainsi que les éléments de l'interface utilisateur, tels que les boutons et les menus.

16. Gestion de la pause de la simulation

Un bouton de pause permet à l'utilisateur d'arrêter temporairement la simulation. Lorsque la simulation est en pause, les véhicules cessent de se déplacer. Le bouton change dynamiquement de texte entre « Pause » et « Continue », indiquant clairement l'état actuel du système.

17. Menu latéral de configuration

Un menu latéral peut être affiché afin de modifier les paramètres de la simulation en temps réel. Ce menu permet notamment d'ajuster le nombre maximal de véhicules, d'activer ou de désactiver certaines couleurs et certains types de véhicules, ainsi que de changer la carte routière. Un bouton de validation permet de fermer le menu et de reprendre la simulation.

18. Fermeture propre du programme

Lorsque l’utilisateur ferme la fenêtre, la fonction main() procède à une fermeture propre du programme. La fenêtre graphique est fermée, les ressources sont libérées correctement et la fonction se termine en retournant la valeur zéro, indiquant une exécution réussie.

Realiser par: Hamza Soussane Alyaacoubi

Yassine Tahiri

Adam Idzdi

Chemseddine Ayoub

Ilyas Gourou