

TRAFFIC CORE

Simulation d'une Smart City en C++ (POO & Raylib)

MODULE

POO en C++ (LST : IDAI)

ANNÉE UNIVERSITAIRE
2025-2026

SOUS L'ENCADREMENT DE

Pr. BEN ABDEL OUAHAB Ikram

RÉALISÉ PAR

EL BOURMAKI Salim • EL HAJIOUI Houssam • SADIKI Maroua • DANY Homam

Le Défi : Maîtriser la Complexité du Trafic Urbain

Dans un contexte d'urbanisation croissante, la gestion des flux de circulation est devenue un enjeu majeur pour les villes intelligentes ("Smart Cities"). La simulation est un outil essentiel pour modéliser, visualiser et anticiper les dynamiques complexes du trafic routier avant de déployer des solutions coûteuses.



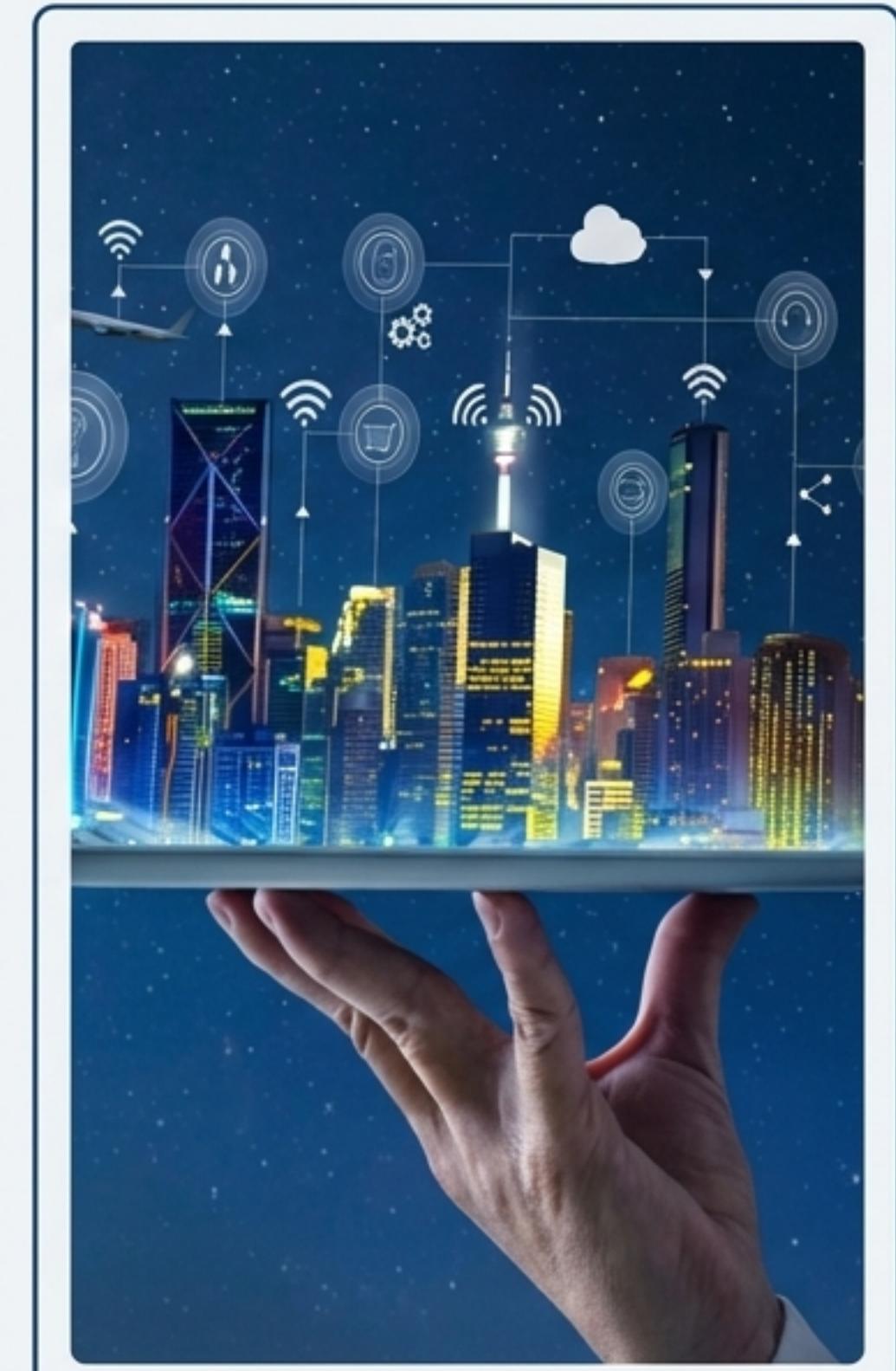
Modéliser : Créer une représentation fidèle d'un réseau routier.



Simuler : Gérer des flottes de véhicules hétérogènes avec des comportements réalistes.



Visualiser : Offrir un retour visuel en temps réel pour l'analyse et la prise de décision.



Notre Solution : TRAFFIC CORE, un Moteur de Simulation Modulaire

Nous avons développé "TRAFFIC CORE", un moteur de simulation de trafic en C++ et Raylib, conçu comme la colonne vertébrale d'un futur écosystème de Smart City.



Modélisation Topologique :

Représenter un réseau routier (routes, intersections) sous une forme mathématique exploitable par des agents autonomes.



Gestion des Agents :

Simuler le déplacement de véhicules variés (voitures, bus, motos) respectant des règles de circulation et dotés de comportements uniques.



Visualisation Temps Réel :

Assurer un rendu 3D fluide et interactif de la simulation via la bibliothèque Raylib.



Architecture Flexible :

Fournir une base de code extensible pour intégrer de futurs modules (gestion de feux, routage dynamique, etc.).

Une Fondation Technique Moderne et Performante

Le choix des outils a été guidé par la recherche de performance, de contrôle et de portabilité.



C++ & Raylib

Pour un contrôle "bas niveau" du rendu 3D via OpenGL et une performance maximale, sans la lourdeur d'un moteur de jeu complet.

CMake

CMake

Pour un système de construction multiplateforme, automatisant la compilation et la gestion des dépendances de manière standardisée.



Visual Studio Code

Pour un environnement de développement léger, extensible et doté d'outils de débogage C++ performants.

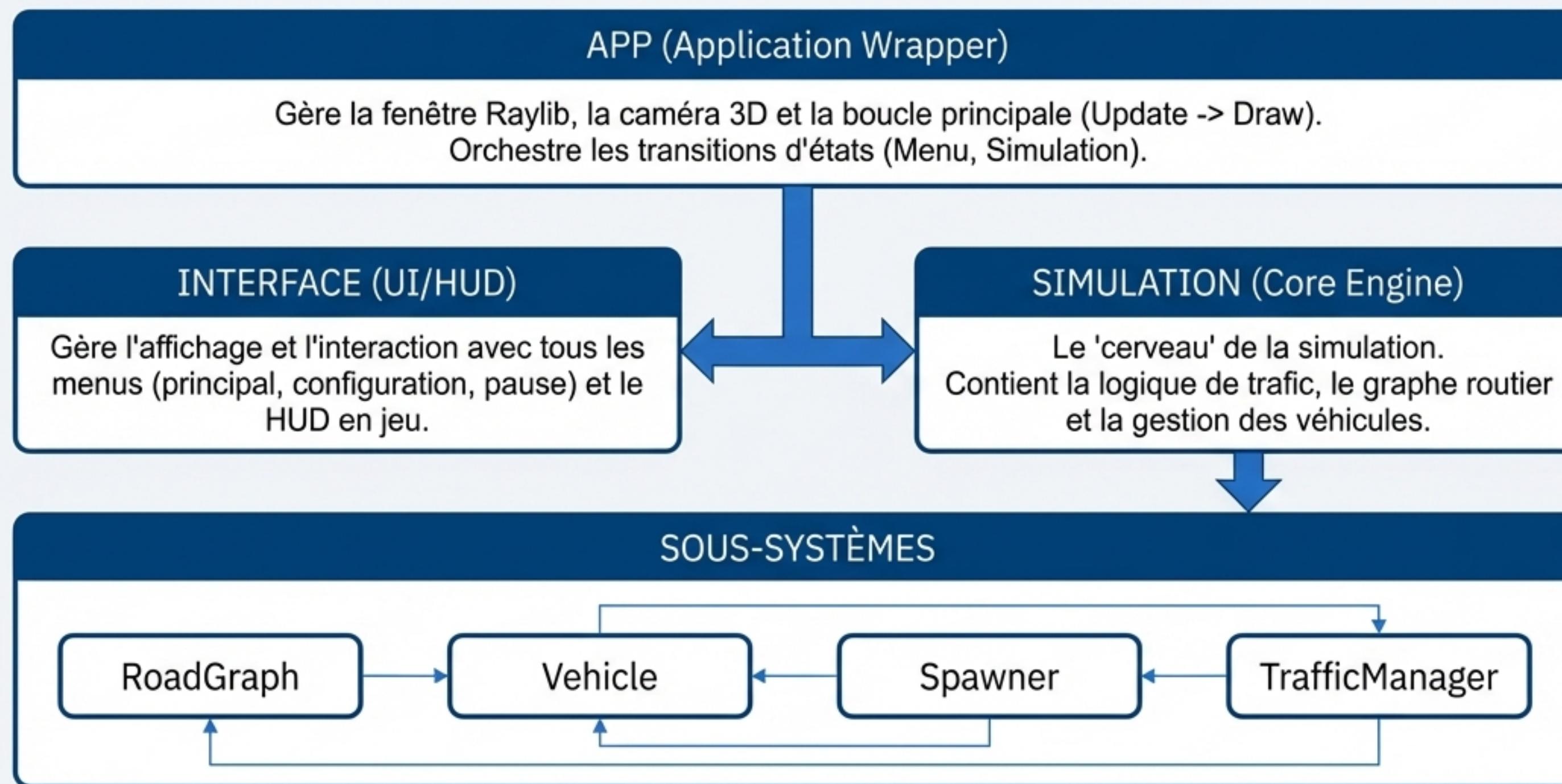


GitHub

Pour une gestion de versions rigoureuse, garantissant la traçabilité et la sécurité du code source tout au long du développement.

Une Architecture Modulaire Pensée pour l'Évolution

Le système est organisé en couches distinctes avec une séparation claire des responsabilités, garantissant la maintenabilité et l'extensibilité du projet.



Les composants spécialisés qui implémentent la logique de navigation, le comportement des agents et les règles de circulation.

Au Cœur de la Simulation : Le Flux de Données par Trame

Chaque trame de la simulation suit un cycle logique précis pour garantir une exécution cohérente et réactive.

1. Phase d'Entrée (INPUT)

- 💻 • Capture des actions de l'utilisateur : clics sur les menus, pressions sur les touches (P, ESC, N, WASD).
- 🖱 • Mise à jour de l'état de l'interface.



2. Phase de Mise à Jour (UPDATE)

(Si la simulation n'est pas en pause)

`Spawner.Update()`: Création de nouveaux véhicules.

`TrafficManager.Update()`: Application des règles de distance et d'évitement.

`Vehicle.Update()`: Mise à jour de la position et de l'orientation de chaque véhicule sur le `RoadGraph`.



3. Phase de Rendu (RENDER)

- 3D `BeginMode3D()`: Rendu de la carte et de chaque véhicule (`vehicle->draw()`).
- 2D Affichage des overlays 2D (HUD, menus).

La Puissance de la POO : Au Service d'une Simulation Robuste

L'architecture repose sur des principes de Programmation Orientée Objet pour gérer la complexité et garantir la flexibilité du code.

Héritage et Polymorphisme **(Le point fort du projet)**

Une classe de base abstraite 'Vehicle' définit un contrat commun ('update', 'draw'). Des classes dérivées ('Motorcycle', 'Taxi', 'PoliceCar') surchargent ces méthodes pour implémenter des comportements visuels et logiques uniques.

****Avantage**** : Ajouter un nouveau type de véhicule (ex: Ambulance) ne requiert aucune modification du moteur de simulation principal (Principe Ouvert/Fermé).

Gestion de la Mémoire (RAII) **(Pointeurs Intelligents)**

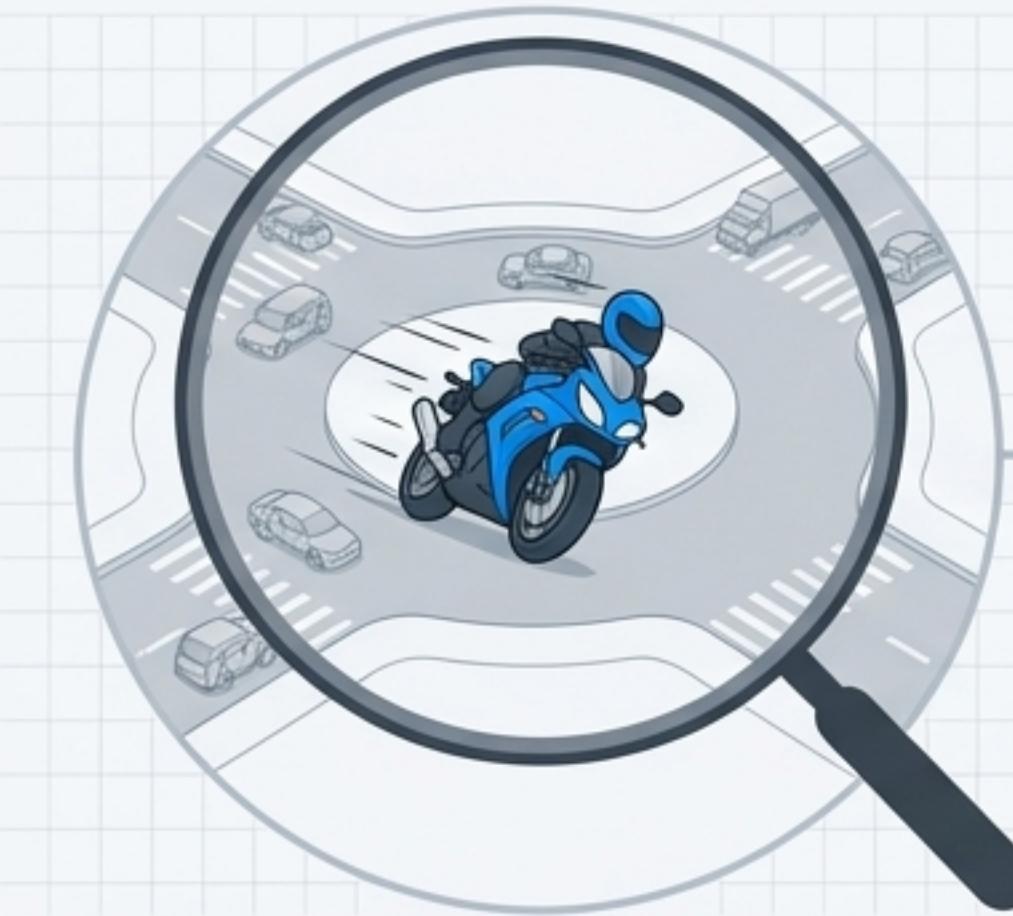
Utilisation de `std::vector<std::unique_ptr<Vehicle>>`. Garantit la libération automatique de la mémoire lorsqu'un véhicule est détruit, éliminant les risques de fuites de mémoire.

Encapsulation **(Intégrité des Données)**

Protection des attributs internes ('private', 'protected') pour garantir l'intégrité de l'état de chaque véhicule.

Le Polymorphisme en Action : Des Comportements Uniques

Grâce à la surcharge des méthodes, la boucle de simulation traite une liste de **Vehicle*** génériques, mais chaque objet exécute son comportement spécialisé.



Motorcycle

Surcharge `draw()` pour calculer un `tiltAngle` et s'incliner dynamiquement dans les virages, simulant la force centrifuge.



Taxi

Surcharge `update()` pour gérer un `signBlinkTimer` indépendant, faisant clignoter son enseigne lumineuse.



PoliceCar

Intègre une logique de `gyrophares alternés`, gérée au sein de sa propre classe.

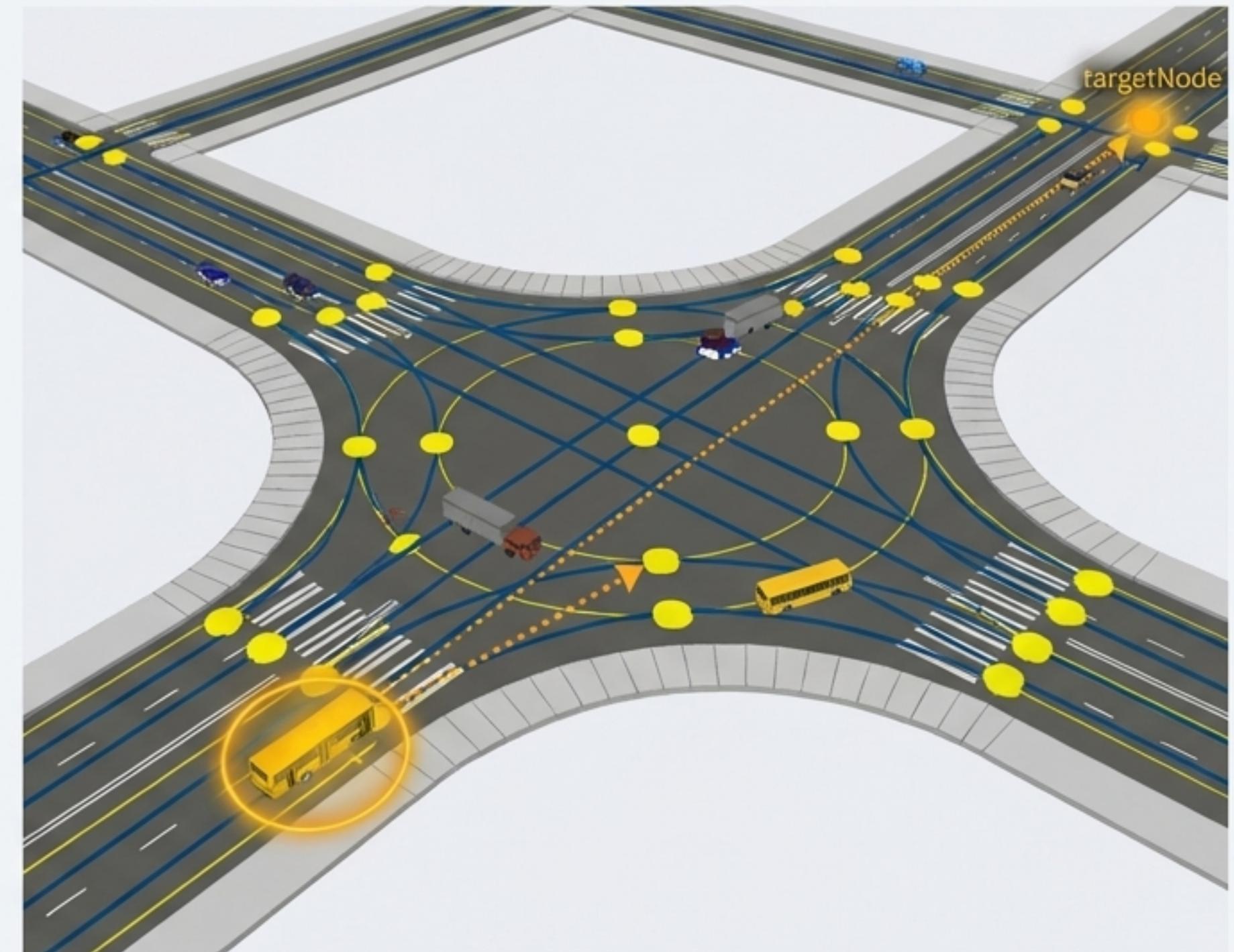
Navigation Autonome : L'Intelligence du Graphe Routier

Les véhicules ne suivent pas des chemins prédéfinis. Leur navigation est gérée par un graphe orienté qui représente le réseau routier.

Structure: La carte est modélisée par une classe RoadGraph contenant des nœuds (intersections, points de contrôle) et des arêtes (routes).

Navigation: Chaque véhicule connaît son nœud cible (targetNode). Il calcule en temps réel sa direction pour l'atteindre.

Décision: Lorsqu'un nœud est atteint, le véhicule interroge le graphe pour connaître les sorties possibles et choisit sa prochaine destination, permettant une circulation infinie et non-répétitive.



Une Expérience Utilisateur Complète et Intuitive

L'interface a été conçue pour offrir un contrôle total sur la simulation, de la configuration initiale aux ajustements en temps réel.

- **Accessibilité:** Prise en main rapide sans formation.
- **Flexibilité:** Contrôle granulaire sur les paramètres de simulation.
- **Immersion:** Expérience visuelle engageante et informative.



Préparez votre Scénario : Le Menu de Configuration

Avant de lancer la simulation, l'utilisateur peut définir précisément la composition et la dynamique du trafic.

Limite Maximale de Véhicules :

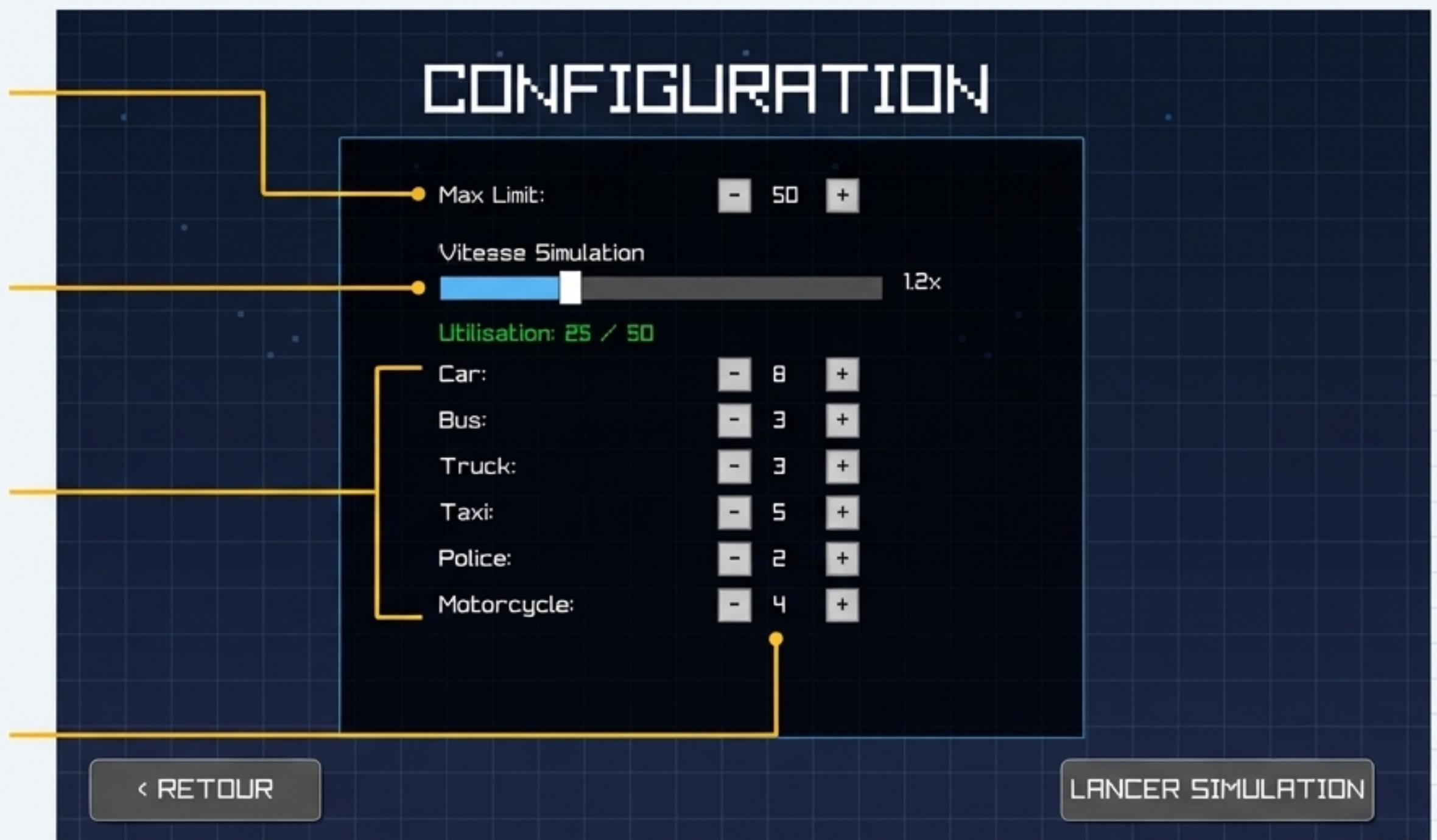
Définit le nombre total d'entités simultanées (plafond à 100).

Vitesse de Simulation : Curseur pour ajuster le 'time scale' de 0.5x (ralenti) à 3.0x (accéléré).

Composition de la Flotte :

Contrôle individuel pour six types de véhicules (Car, Bus, Truck, Taxi, Police, Motorcycle).

Feedback Visuel : Une barre d'utilisation (Utilisation : 25 / 50) indique en temps réel la capacité restante.



Traffic Core Simulator

- [P] : Settings
- [ESC] : Main Menu
- [N] : Show Nodes
- [WASD] : Move Camera
- Click Car : Force Move
- Vehicles: 52

Au Cœur de l'Action : L'Environnement de Simulation Interactif

Une fois lancée, la simulation offre un environnement 3D entièrement explorable, complété par un HUD informatif.

Interface en Jeu (HUD)

Commandes Clavier



Déplacer la caméra librement.



Accéder au menu de pause et aux paramètres dynamiques.



Afficher/Cacher les nœuds du graphe pour le débogage.



Retourner au menu principal.

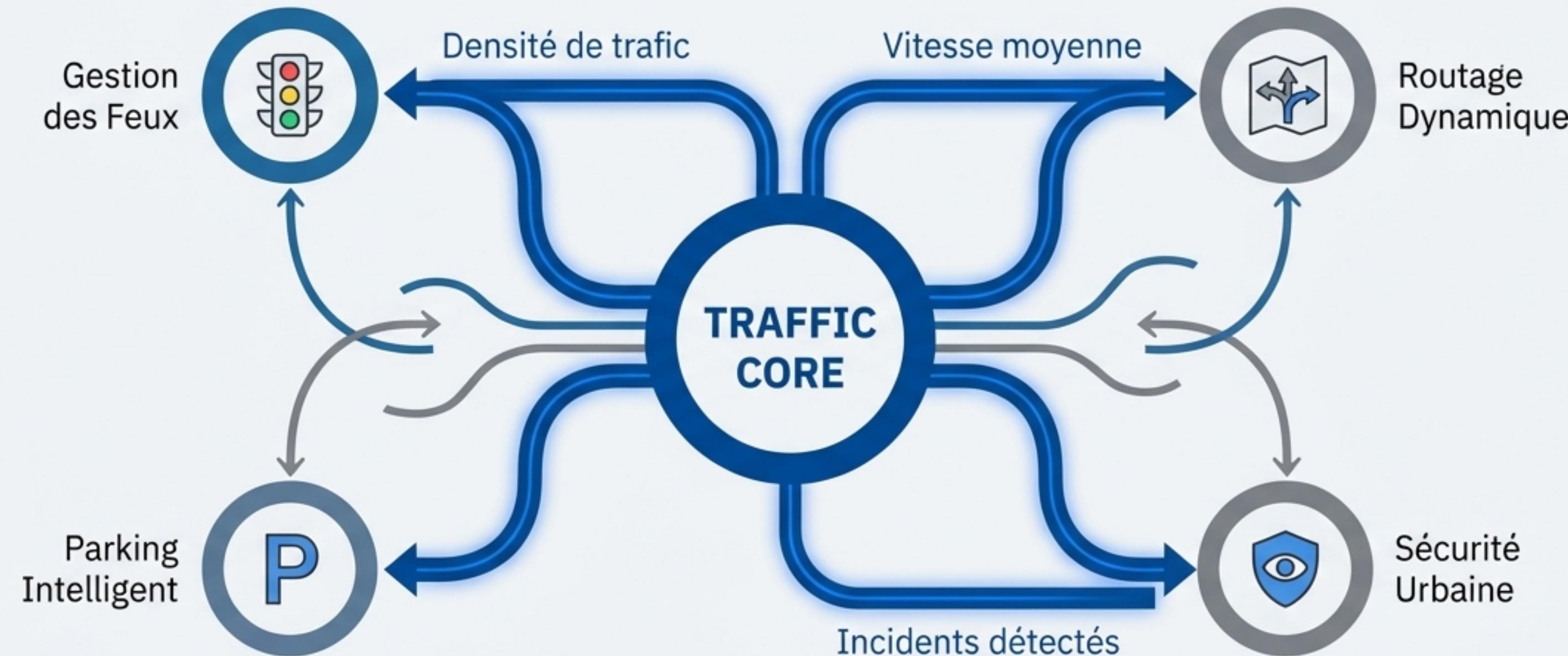
Statistiques en Temps Réel

Vehicles: 52 a dynamic number, e.g.



Au-delà de la Simulation : Vers un Écosystème Connecté

TRAFFIC CORE n'est pas une fin en soi. Il a été conçu comme le socle fondamental, le générateur de "vérité terrain" ("Ground Truth"), pour un écosystème de gestion urbaine active et collaborative.



La Vision: Passer d'un système de monitoring passif à un réseau de modules intelligents qui interagissent pour optimiser la ville en temps réel.

Une Architecture de Collaboration pour l'Avenir

Chaque donnée générée par TRAFFIC CORE peut alimenter des modules spécialisés pour créer un système de gestion prédictif.

Synergies Potentielles

TRAFFIC CORE → Adaptive Signal Control

Les données de densité et de file d'attente permettent d'ajuster dynamiquement les cycles des feux de circulation pour fluidifier le trafic.



TRAFFIC CORE → Routage Dynamique

La détection d'anomalies (ralentissements, blocages) déclenche un recalcul des itinéraires pour les véhicules en amont.

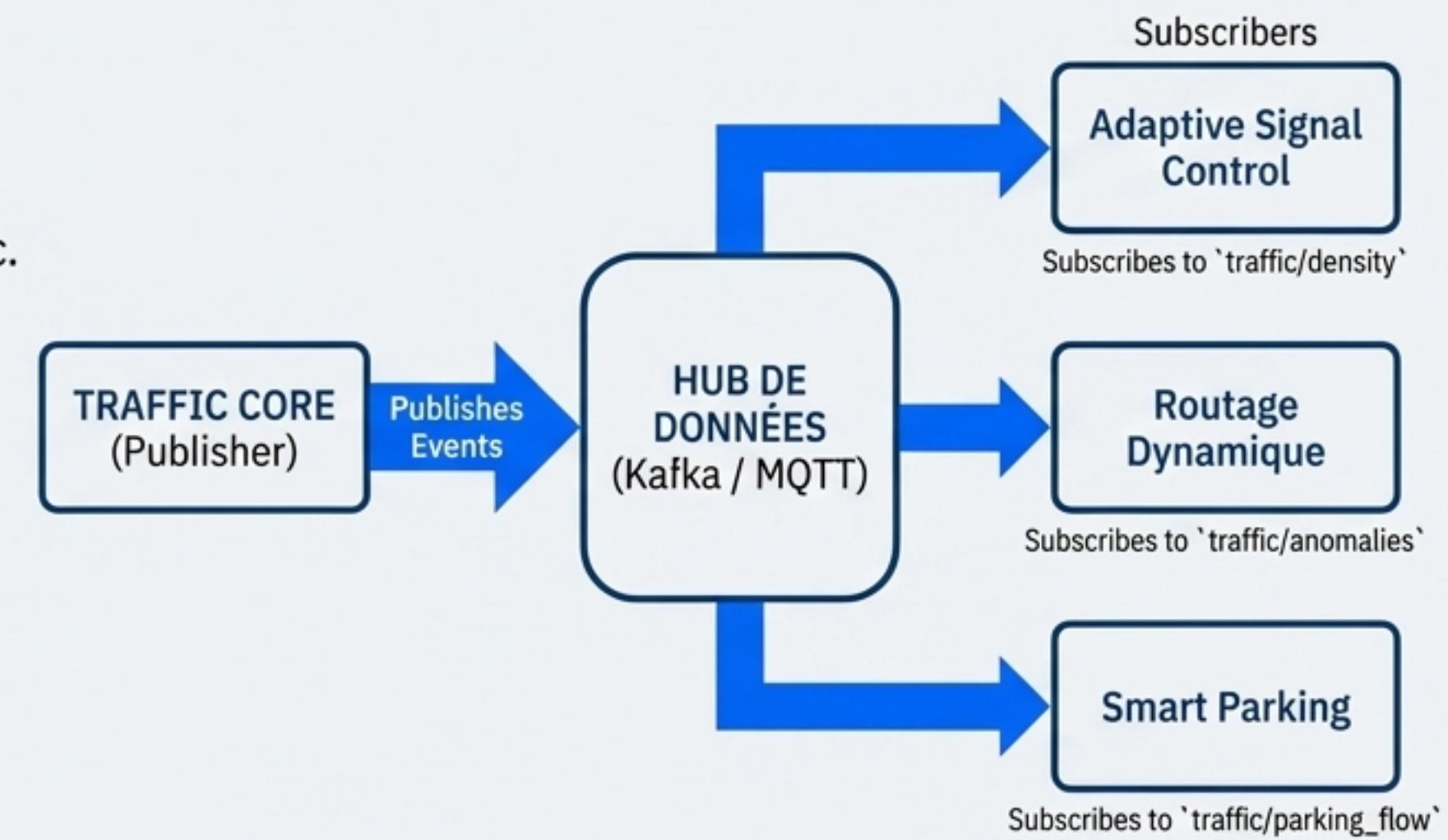


TRAFFIC CORE → Smart Parking & EV Charging

L'analyse des flux entrants vers les parkings permet de prédire la saturation et de guider les conducteurs vers des places disponibles.



Architecture Technique Proposée



Conclusion : Du Code C++ au Système Nerveux d'une Ville Intelligente

Résumé des Acquis

- ✓ Nous avons développé avec succès un moteur de simulation 3D robuste en C++, démontrant une application avancée des principes de la Programmation Orientée Objet (Polymorphisme, RAII).
- ✓ Le système modélise des comportements de trafic complexes de manière autonome, flexible et visuellement engageante.

Déclaration Finale

TRAFFIC CORE pose les fondations techniques d'un système capable non seulement de simuler le trafic, mais de devenir un composant actif dans la réduction de la congestion et l'amélioration de la sécurité. Ce projet est une première étape vers la transformation de nos routes passives en autoroutes intelligentes et connectées.