UNIVERSITE DU QUEBEC EN OUTAOUAIS

Département de l'informatique

COURS: INF1573-Programmation II

RAPPORT DE PROJET – Système de Navigation Routière Dynamique

Professeure: Benyahia Ilham

Membre du groupe 6:

Anis, Davidson

Tambat, Tresor Megane

Kouyaté, Yasmine Jawad

Mbemba, Mechack Biyudi

Djieunang Noumbo, Nelie Mabelle

Nyama Koumba, Aude Guysliah Maelisse

Date de remise: 25 avril 2025

Table des Matières

Liste des illustrations

Figures

Tableau

Introduction

Objectif général du projet

Le système à développer est une application GPS intelligente qui permet de guider dynamiquement un véhicule à travers un réseau routier simulé. L'application calcule le chemin optimal à parcourir entre un point de départ et une destination, en prenant en compte l'état actuel du réseau, incluant la présence éventuelle d'obstacles tels que des congestions ou des accidents.

Le système repose sur une programmation orientée objet et par événements. Il modélise un graphe constitué d'intersections et de routes pondérées, dont les poids sont influencés dynamiquement par des événements de circulation. Lorsqu'un événement survient, le système est en mesure de recalculer le trajet en temps réel et d'afficher de nouvelles recommandations de direction au conducteur.

Contexte du GPS intelligent et fonctionnement général:

Le système s'inspire des systèmes de navigation embarqués modernes utilisés dans les véhicules connectés. Ces GPS intelligents, équipés d'une interface graphique, permettent non seulement de calculer le plus court chemin à un instant donné, mais aussi de s'adapter à l'évolution du trafic. Lorsqu'un événement comme un accident ou un ralentissement est détecté, le système recalcule automatiquement le trajet en tenant compte des nouvelles conditions de circulation. Le conducteur est informé via l'écran du GPS des nouvelles directions à suivre, que ce soit tourner à gauche, à droite ou continuer tout droit à l'intersection suivante.

Fonctionnalités principales

Affichage du réseau routier:

Une carte du réseau routier est affichée à l'écran sous forme de graphe (noeuds = intersections, arcs= routes).

La voiture (représentée par un point) est visible sur la carte, avec un point de départ et une destination définie.

Entrée utilisateur – itinéraire initial :

L'utilisateur choisit un point de départ et un point d'arrivée.

Le système calcule automatiquement l'itinéraire optimal (le plus court en distance).

Le système affiche :

- La route à suivre.
- Le temps estimé de parcours.

- La distance
- Les directions à suivre (via le module GPS)

Simulation en temps réel du déplacement du véhicule :

Le véhicule est déplacé progressivement sur l'itinéraire, et sa position est mise à jour en temps réel.

Le déplacement est géré par des modules de javaFX qui met à jour la position du véhicule à intervalles réguliers.

Simulation d'un accident ou d'une congestion :

L'utilisateur peut indiquer une intersection pour simuler un accident ou une congestion.

L'état de cette route passe à "accidenté" ou "congestionné".

Si cette route fait partie de l'itinéraire courant :

Le système met à jour dynamiquement le graphe.

Il recalcule automatiquement un nouveau chemin optimal.

Il affiche les nouvelles directions et le temps mis à jour.

Composantes du système:

Composantes statiques (fixes après initialisation)

- Le graphe représentant la ville (intersections + routes)
- Les positions initiales de départ et d'arrivée

Composantes dynamiques (modifiables)

- L'état des routes (fluide / congestionné / accidenté)
- L'itinéraire calculé (recalculé si accident ou congestion)
- La direction à suivre à chaque intersection
- Position du véhicule en temps réel (affichée dynamiquement pendant le déplacement)

Exemple de cas d'utilisation:

Utilisateur	Système
1. Ce cas d'utilisation débute lorsque l'utilisateur à lancer le programme.	2. Démarrer le système et afficher la carte du réseau routier avec la voiture à un point quelconque.
3. Choisir un point de départ et une destination.	4. Recevoir les points de départ et d'arrivée.

	5. Calculer le chemin optimal et afficher : chemin, distance, temps estimé, direction à prendre à la prochaine intersection.
	6. Déplacer la voiture en temps réel, mettre à jour la position et afficher la direction à chaque étape.
7.Choisir le nœud pour simuler l'accident ou pour simuler la congestion	8. Simuler un accident sur la route sélectionnée ou rendre la route "congestionnée".
	9. Si la route est sur l'itinéraire, recalculer le chemin le plus court, mettre à jour l'itinéraire, la direction et le temps restant.
	10. Afficher la nouvelle direction à prendre à la prochaine intersection.
	11. Arriver à destination, afficher la distance parcourue et le temps du trajet.
	12.Ccliquer sur réinitialiser pour faire un nouveau trajet ou quitter le programme.
	13. Attendre la réponse de l'utilisateur.
14. réinitialiser	15. Recommencer à l'étape 2.

Tableau 1 : Cas d'utilisation du système

Scénario alternatif :

ID	Condition	Comportement du système
A1	Le point de départ ou d'arrivée est invalide	Affiche un message d'erreur et attend une nouvelle saisie.
A2	Aucun chemin n'existe entre le départ et l'arrivée	Affiche "Aucun chemin trouvé" sans démarrer l'animation.
A3	L'accident est simulé sur une route hors itinéraire	L'itinéraire n'est pas modifié, le trajet se poursuit normalement.
A4	La route indiquée pour l'accident n'existe pas	Affiche un message d'erreur "Aucune route directe trouvée".

Tableau 2: Scénario alternatif du cas d'utilisation

1-Spécification de l'interface du système

- **Zone de sélection du trajet (Haut gauche)**

  - **Composants JavaFX** : Deux TextField (departField, arriveeField) pour saisir les noms des intersections, un Button « Calculer trajet ».

- **Fonction** : Permet de choisir les intersections de départ et d'arrivée par leur nom, appelle calculerTrajet() qui utilise GPS.calculerCheminOptimal() pour générer le chemin.
- **Classes liées** : GPS, CarteVille, Intersection, Trajet.

- **Zone de simulation d'événement (Haut centre/droite)**

  - **Composants JavaFX** : Un TextField (eventField) pour indiquer l'intersection ciblée, deux Button « Simuler accident » et « Simuler congestion », un Slider (congestionSlider) de niveau 0 à 2.
  - **Fonction :** Simule un accident ou une congestion à une intersection précise, modifie dynamiquement l'état du graphe (via Route.appliquerImpact()), appelle simulerAccident() ou simulerCongestion() pour créer un Evenement et l'ajouter via GPS.declarerEvenement().
  - **Classes liées :** Accident, Congestion, Evenement, GPS, CarteVille, Route.

- **Zone graphique de simulation (Centre - Canvas principal)**

  - **Composants JavaFX :** Pane contenant des Circle pour les intersections, Line pour les routes, Circle ou ImageView mobile pour le véhicule (Vehicule).
  - **Fonction :** Représente visuellement la carte, affiche le trajet optimal en rouge (calculé via Dijkstra), met en évidence les événements (rouge/orange), déclenche l'animation du véhicule avec TrajetAnimator.animer().
  - **Classes liées :** Vehicule, Trajet, CarteVille, Route, Intersection, TrajetAnimator.

- **Bouton de réinitialisation (Haut droite)**

  - **Composants JavaFX :** Un Button « Réinitialiser ».
  - **Fonction :** Efface tous les événements, trajets, et remet le réseau à l'état initial, via resetReseau() qui appelle GPS.supprimerTousEvenements() et Route.reinitialiserPoids().
  - **Classes liées :** GPS, CarteVille, Route.

- **Zone d'alerte et d'information (Bas gauche ou sous la carte)**

  - **Composants JavaFX :** Label ou TextArea stylisée (infoLabel).
  - **Fonction :** Informe l'utilisateur des événements simulés (ex : « Accident détecté à G »), utilise afficherAlerte(String message).
  - **Classes liées :** Evenement, GPSInterface.

Schéma conceptuel du système

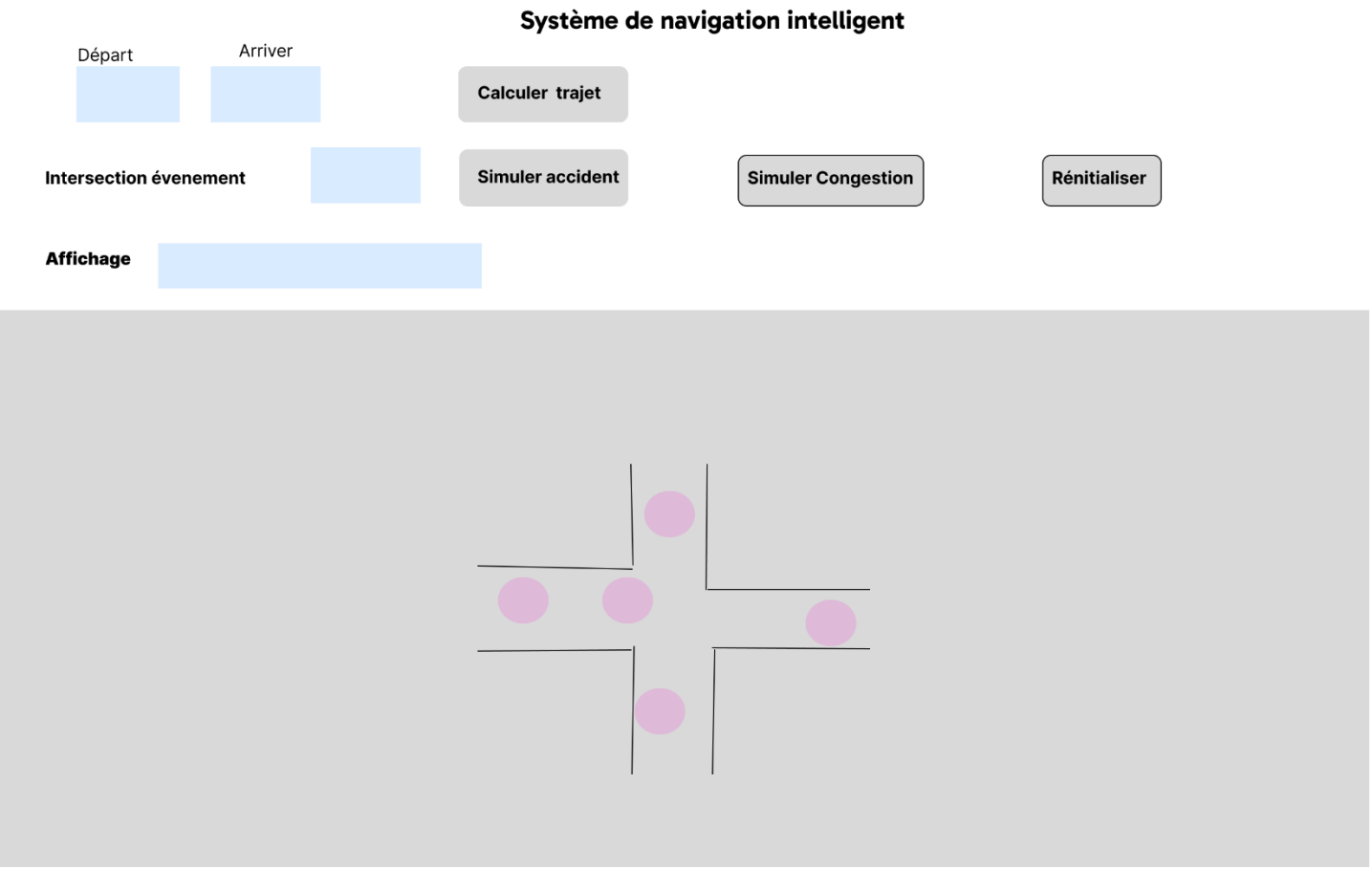

Figure 1: Maquette du système à développer.

Le schéma ci-dessus présente l'interface principale qui sera développée. Le système affichera un graphe représentant un réseau routier composé de 26 intersections nommées de A à Z, connectées par des routes.

Description des éléments de l'interface

L'interface utilisateur en haut de l'écran comprend :

- Des champs de saisie de type TextField pour indiquer l'intersection de départ et d'arrivée ;
- Un bouton « Calculer trajet » pour lancer le calcul du chemin optimal entre deux intersections ;
- Un champ de saisie de type TextField pour spécifier une intersection où simuler un événement;
- Un bouton « Simuler accident » pour générer un accident à une intersection donnée ;
- Un bouton « Simuler congestion » pour déclencher une congestion selon le niveau choisi ;

- Un bouton « Réinitialiser » pour restaurer l'état initial du graphe.

Interaction avec le graphe

Le graphe visuel implémenté permettra à l'utilisateur de suivre le trajet du véhicule calculé dynamiquement et d'observer l'impact des événements. Cette interface nous permettra de mieux contextualiser notre système de guidage à travers la programmation par événements et la représentation graphique d'un réseau routier intelligent dans le cadre de ce projet.

Révision critique de la spécification proposée

Étape 1 – Source d'information, preuves et critères de qualité

La spécification initiale du devoir décrivait globalement un système de navigation intelligent réagissant à des événements dynamiques comme les accidents ou la congestion. Bien que l'intention fonctionnelle soit claire, plusieurs éléments techniques nécessaires à la conception objet n'étaient définis ni illustrés.

Pour établir des critères de qualité d'une spécification, nous nous sommes référés :

- Aux principes vus en cours (clarté, exhaustivité, cohérence technique) ;
- À des normes de spécification logicielle (IEEE, bonnes pratiques d'analyse orientée objet) ;
- À notre expérience de développement (compréhension du besoin, transformation en conception UML).

Les éléments suivants manquent ou sont peu explicités :

- L'identification des classes objets nécessaires (par exemple : GPS, Trajet, Véhicule, Intersection, Route, Évènement) et leurs relations ;
- Les composants graphiques à utiliser pour l'interface (par exemple les champs de saisie, les boutons, la carte interactive, le curseur) ne sont pas précisés ;
- Les effets concrets des événements (comme : multiplicateurs de poids sur les routes, réinitialisation, animation de recalcul) ne sont pas formalisés dans un modèle dynamique.

Nous avons donc vérifié :

- La présence ou non d'un découpage fonctionnel précis ;
- L'identification des classes, attributs, et méthodes dès la spécification ;
- L'adéquation entre les exigences et leur potentiel de mise en œuvre graphique (Swing ou JavaFX).

Étape 2 – Évaluation technique de la spécification (paramètres et logiques de traitement cohérent avec les méthodes à coder)

La spécification initiale ne décrivait pas les signatures de méthodes ni les structures de données à utiliser. Nous avons donc évalué sa faisabilité à partir des critères suivants :

- Respect des paramètres et des types dans les fonctions : les fonctions de simulation, de calcul de trajet et de gestion du graphe doivent recevoir des types précis (String, Intersection, int). L'absence de ces précisions dans la spécification imposait une interprétation libre.

- Boucles et traitements réactifs : le système étant fondé sur la programmation par événements, il nécessite de surveiller certains états de manière continue (ex : arrivée à une intersection). Cela a demandé une conception robuste pour éviter les boucles infinies et assurer la fin des transitions (PauseTransition, SequentialTransition).

- Choix des noms et documentation : la spécification ne fournissait aucune convention de nommage ni documentation. Nous avons dû renforcer la lisibilité par des noms explicites (calculerCheminOptimal, appliquerEffet, reinitialiserPoids) et des commentaires clairs.

- Choix de structures de données : le besoin de représenter dynamiquement un graphe routier nous a amenés à utiliser des structures comme Map, List, et des objets personnalisés comme Intersection, Route, etc. Ces choix sont absents de la spécification initiale.

- Gestion des événements non formalisée : les effets des accidents ou congestions n'étaient pas précisés, ni leur propagation sur les routes adjacentes.

- Tentative initiale avec Swing : la première version du projet a été esquissée en Swing. Toutefois, l'intégration d'un graphe dynamique avec interaction en temps réel s'est révélée trop complexe.
  Le projet a été basculé vers JavaFX, plus adapté à la manipulation d'éléments visuels vectoriels (formes, animation, scène graphique, transitions).

Validation, adaptation ou correction

Notre équipe a validé l'intention fonctionnelle générale de la spécification proposée. Le concept d'un système GPS dynamique basé sur un graphe routier, affecté par des événements, est bien aligné avec les objectifs pédagogiques du devoir et les exigences du contexte.

Cependant, pour répondre aux contraintes de conception orientée objet, nous avons apporté les adaptations suivantes :

- Nous avons clarifié la structure de l'interface graphique, en choisissant explicitement des composants de JavaFX : TextField pour les saisies, Button pour les actions, Slider pour la congestion, et Group pour l'affichage dynamique du graphe.

- Nous avons défini précisément les classes nécessaires, leur rôle, leurs attributs, et leurs méthodes dans un diagramme UML pour encadrer la phase de codage.

- Nous avons introduit un système d'animation du trajet, permettant de visualiser le déplacement du véhicule étape par étape, avec détection des événements en temps réel.

- Nous avons intégré des conditions d'arrêt claires dans les transitions (verifierEvenements() dans TrajetAnimator).

- Nous avons formalisé l'impact numérique des événements :
    - ❑ Un accident multiplie les poids des routes connectées par 50 ;
    - ❑ Une congestion applique un multiplicateur de 1, 3 ou 30 selon le niveau (0 à 2).

Ces adaptations garantissent une meilleure cohérence avec les attentes pédagogiques du cours et assurent la faisabilité technique de l'implémentation.

Étape 3 – Tableau de critères, observations et actions qualité

Critère qualité	**Observation sur la spécification initiale**	**Action qualité prise**
Définition des classes et responsabilités	Aucune classe n'est identifiée ni décrite	Classes formalisées dans un UML complet
Structure de l'interface utilisateur	Aucune indication des composants à utiliser	Interface conçue avec TextField, Button, Slider, etc.
Modélisation du graphe routier	Réseau routier mentionné mais non défini ni structuré	Création de Intersection, Route, CarteVille
Intégration de la programmation par événements	Présence implicite mais aucune formalisation des mécanismes réactifs	Conception de TrajetAnimator + événements dynamiques
Gestion des événements	Pas de détail sur leur effet ou impact dans le système	Création d'une hiérarchie Evenement, Accident, Congestion

Choix du langage et de la technologie graphique	Swing suggéré mais non adapté pour ce type de système	Basculement vers JavaFX
Documentation et nommage	Aucun standard proposé	Application de bonnes pratiques Java (noms explicites + javadoc)
Adaptabilité à l'évolution du graphe	Spécification rigide, peu évolutive	Modularisation via carte, événements, véhicule
Réutilisabilité du code	Spécification monolithique	Architecture modulaire et orientée objet

Tableau 3 : Tableau de critères, observations et actions qualité

Étape 4 – Liste des corrections appliquées

- Remplacement de Swing par JavaFX:
  - ❑ La tentative initiale avec Swing s'est avérée complexe en raison du manque de flexibilité pour afficher dynamiquement un graphe routier avec animation. JavaFX a été choisi pour sa gestion fluide des formes (Line, Circle), de la scène (Group, Scene) et de l'animation (PathTransition, SequentialTransition).

- Création explicite de toutes les classes nécessaires
  - ❑ Les classes GPS, CarteVille, Intersection, Route, Trajet, Vehicule, TrajetAnimator, Evenement, Accident, Congestion, etc. ont été identifiées dès la conception. Elles sont toutes documentées dans le diagramme UML et associées à des responsabilités claires.

- Ajout de composants JavaFX clairs dans l'interface
  Tous les composants de saisie, action et affichage sont spécifiés :
  - ❑ TextField pour les entrées ;
  - ❑ Button pour déclencher les événements ;
  - ❑ Slider pour la congestion ;
  - ❑ Label pour les retours utilisateur ;
  - ❑ Group pour l'affichage du graphe.

- Formalisation des effets des événements
  - ❑ Accident → multiplicateur de poids : ×50
  - ❑ Congestion → multiplicateur selon niveau : ×1, ×3, ou ×30

- Ajout d’un système d’animation événementiel
  La classe TrajetAnimator contrôle le déplacement et déclenche des actions automatiques lors de l’arrivée à une intersection affectée par un événement.

- Amélioration du nommage et de la documentation
  Chaque méthode porte un nom explicite et est documentée dans le code pour faciliter la compréhension.

- Modification de l’algorithme de Dijkstra
  L’algorithme initial a été adapté pour :
    - ❑ Prendre en compte les poids modifiés en temps réel par les événements ;
    - ❑ Permettre le recalcul dynamique à partir d’un point intermédiaire (et non uniquement du point de départ) ;
    - ❑ Intégrer un mécanisme de réinitialisation automatique des poids avant chaque nouveau calcul.

- Modélisation du graphe routier
  Le réseau a été modélisé comme un graphe orienté pondéré. Chaque route possède un poids modifiable, et chaque intersection contient la liste des routes qui y sont connectées.

Critères de qualité d’une bonne spécification

Pour juger de la qualité d’une spécification logicielle, nous avons retenu les critères suivants, inspirés des bonnes pratiques vues en cours :

Clarté : les objectifs fonctionnels doivent être exprimés de manière compréhensible, sans ambiguïté;

Exhaustivité : tous les cas d’usage, composants d’interface et éléments du modèle doivent être mentionnés ;

Spécificité technique : la description doit être assez précise pour être transformée directement en conception technique (notamment pour le diagramme de classes) ;

Cohérence avec les contraintes du projet : en l’occurrence, l’utilisation de JavaFX, la programmation par événements, et la manipulation d’un graphe routier ;

Visualisation : l’intégration d’un schéma ou d’une maquette améliore la compréhension de l’interface et du comportement attendu.

Nous avons construit notre spécification révisée selon ces critères, afin de garantir une continuité fluide entre les phases d’analyse, de conception et d’implémentation.

2- Conception de la solution

2.1 Décomposition du problème et architecture du système

Notre système de navigation GPS implémente une solution complète de guidage routier dynamique en utilisant les principes de la programmation orientée objet et de la programmation par événements. Il est capable de modéliser un réseau routier, de calculer un itinéraire optimal, de simuler des perturbations (accidents, congestions), et de réagir en temps réel en adaptant la trajectoire du véhicule.

L'objectif est de fournir au conducteur une interface intuitive qui affiche les recommandations de trajet en tenant compte de l'état actuel du réseau.

Architecture globale

L'architecture repose sur quatre modules principaux, chacun regroupant des classes aux responsabilités cohérentes :

1. Modèle Réseau

- **Classes** : CarteVille, Intersection, Route
- **Rôle** : Représente la structure du réseau routier sous forme de graphe orienté pondéré.
- **Relations** :
    - CarteVille est associée à plusieurs Intersection et Route.
    - Route relie deux Intersection et peut être affectée par des événements.

2. Navigation

- **Classes** : GPS, Trajet, Vehicule, TrajetAnimator
- **Rôle** : Gère le calcul des trajets, le déplacement du véhicule et l'animation dynamique du guidage.
- **Relations** :
    - GPS est associé à une Vehicule.
    - TrajetAnimator anime le véhicule sur le Trajet calculé par GPS.

3. Événements

- **Classes** : Evenement , Accident, Congestion
- **Rôle** : Modélise les incidents pouvant affecter le réseau routier en modifiant les poids des routes.
- **Relations** :
    - Chaque Evenement est associé à une Intersection.
    - Les Route peuvent être impactées par plusieurs événements.

4. Interface Utilisateur

- **Classe** : GPSInterface
- **Rôle** : Permet à l'utilisateur d'interagir avec le système via JavaFX. Elle affiche le réseau, permet de saisir un trajet, de simuler des événements, et de visualiser l'itinéraire.

Description synthétique des classes

Classe	Rôle principal	Méthodes clés (extraits)	Relations notables
CarteVille	Gère le réseau routier (graphe)	chargerReseauDepuisFichier( ), getRouteEntre()	Association avec Intersection et Route
Intersection	Point du graphe avec ID, nom, coordonnées	ajouterRoute(), getRoutes()	Associée à plusieurs Route
Route	Relie deux intersections, avec poids variable et représentation graphique	appliquerImpact(), reinitialiserPoids()	Dépend de Evenement
GPS	Moteur du système : calcule les itinéraires, applique les événements	calculerCheminOptimal(), declarerEvenement()	Dépend de CarteVille, Route, Vehicule
Trajet	Stocke une séquence d'intersections représentant un chemin optimal	ajouterIntersection(), imprimerTrajet()	Utilisé par TrajetAnimator
Vehicule	Représente graphiquement un véhicule sur la carte	deplacerInstant()	Dépend de Intersection, Route, GPS
TrajetAnimator	Anime le trajet du véhicule, détecte les événements en temps réel	animer(), arreter(), reinitialiser()	Utilise Trajet, Vehicule, GPS
Evenement	Classe abstraite pour les incidents affectant le réseau	appliquerEffet()	Héritée par Accident, Congestion
Accident	Augmente fortement les poids des routes adjacentes à l'intersection concernée	appliquerEffet()	Hérite de Evenement
Congestion	Applique un multiplicateur de poids selon le niveau de congestion	appliquerEffet()	Hérite de Evenement

GPSInterface	Gère l'interface utilisateur avec JavaFX : saisie, affichage, animation, simulation	dessinerReseau(), recalculerTrajetDepuisPositionActuelle()	Dépend de GPS

Tableau 4 : Description synthétique des classes

5. Architecture logicielle

Le système GPS intelligent a été conçu selon une architecture en couches, fondée sur la séparation claire des responsabilités. Cette structure facilite la maintenabilité, la lisibilité et l'évolutivité de l'application. Les trois couches sont les suivantes :

- Couche interface utilisateur : Elle est assurée par JavaFX à travers la classe GPSInterface. Elle comprend les champs de saisie, les boutons de contrôle, le slider de niveau de congestion, ainsi que la zone graphique affichant dynamiquement le réseau routier, les itinéraires et l'animation du véhicule.

- Couche métier (logique) : Elle regroupe les traitements fonctionnels essentiels comme le calcul du plus court chemin (GPS), l'application des événements (Evenement, Accident, Congestion), la simulation du déplacement (TrajetAnimator), et l'adaptation en temps réel du trajet.

- Couche données et modélisation : Elle structure le graphe du réseau via les classes CarteVille, Intersection et Route. Elle gère également les trajets (Trajet) et les éléments visuels associés aux véhicules (Vehicule).

Cette architecture respecte les principes **SOLID**, la réutilisabilité des objets, et permet une séparation forte entre la logique, les données et l'affichage.

2.2 Modélisation orientée objet

Le système repose sur une modélisation rigoureuse des entités sous forme de classes. Voici les principales classes identifiées :

Classes de modélisation :

- CarteVille : contient toutes les intersections et routes. Permet de charger le réseau depuis un fichier.
- Intersection : représente un sommet du graphe. Stocke son ID, nom, position et routes associées.
- Route : relie deux intersections. Contient le poids initial et modifiable, et sa représentation graphique.

- Classes de logique :
- GPS : moteur central du système. Gère le graphe, les événements, et les calculs de chemin.
- Trajet : contient la séquence ordonnée des intersections formant un itinéraire.
- Evenement (abstraite) : représente tout changement affectant une intersection.
- Accident : augmente fortement les poids des routes.
- Congestion : augmente le poids selon un niveau d'intensité.
- Classes de simulation dynamique :
- Vehicule : objet graphique représentant le véhicule dans l'interface.
- TrajetAnimator : anime le déplacement et déclenche le recalcul si un événement est détecté.

2.3 Diagramme UML des classes

Un diagramme UML a été généré pour illustrer :

- Les classes mentionnées ;
- Leurs attributs et méthodes ;
- Leurs relations.

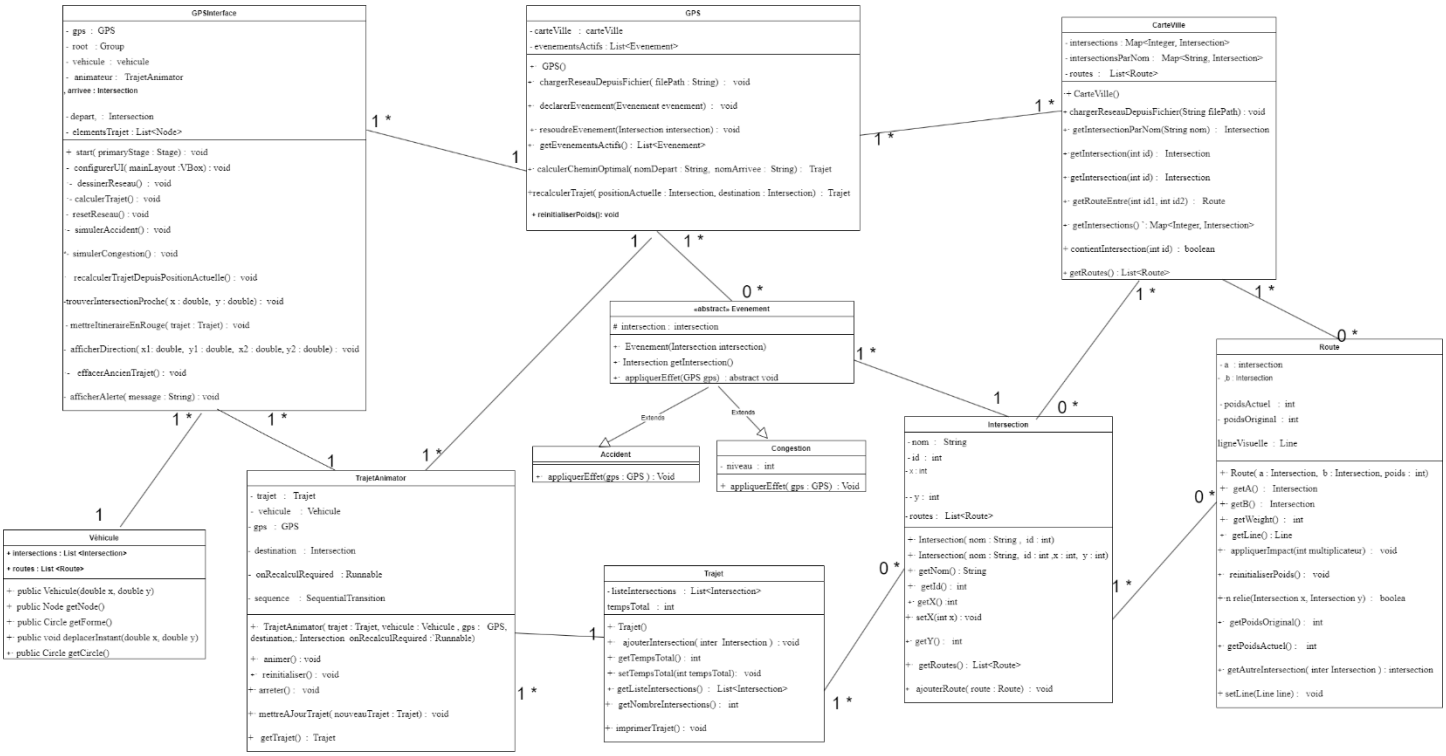

Figure 2 : Diagramme UML

2.4 Algorithme du plus court chemin

Le système utilise une version adaptée de l'algorithme de Dijkstra, optimisée pour tenir compte des événements en temps réel.

Adaptations principales :

À chaque nouveau calcul, tous les poids des routes sont réinitialisés à leur valeur d'origine.

Les événements actifs sont appliqués juste avant le calcul :

- Accident applique un multiplicateur ×50 ;
- Congestion applique un multiplicateur variable (1, 3 ou 30).

L'algorithme accepte un point de départ dynamique : il peut être exécuté non seulement depuis l'origine du trajet, mais aussi depuis la position courante du véhicule (recalcul partiel).

Il retourne un objet Trajet contenant la séquence d'intersections à parcourir.

2.5 Gestion des événements dynamiques

Les événements sont gérés par la classe Evenement et ses sous-classes. Chaque événement cible une intersection et affecte dynamiquement toutes les routes adjacentes.

Les événements sont stockés dans une liste evenementsActifs dans GPS. Lorsqu'un événement est simulé :

- Il est ajouté à la liste.
- Ses effets sont appliqués.
- Le trajet est recalculé si un véhicule est en mouvement.

2.6 Révision de la conception proposée

Une version partielle de la conception avait été esquissée dans le devoir, mais elle nécessitait une transformation complète pour répondre aux exigences techniques :

Élément	Observations sur la conception initiale	Action de correction
Classes absentes	Pas de modélisation des objets du graphe	Définition complète dans un UML
Interface graphique	Non spécifiée	Création en JavaFX, avec choix des composants
Algorithme flou	Dijkstra mentionné sans adaptation dynamique	Implémentation sur mesure avec recalcul partiel
Simulation d'événement	Non formalisée	Création de Evenement, Accident, Congestion
Animation du véhicule	Non prévue	Ajout de Vehicule et TrajetAnimator avec transitions

Tableau 5 : Révision de la conception proposée

Nous avons ainsi corrigé et complété l'ensemble de la conception, en veillant à respecter les principes de la programmation orientée objet et les exigences spécifiques du devoir.

3- Implémentation (codage)

3.1 Présentation générale de l'implémentation

Le système a été développé en Java 17 avec l'outil JavaFX pour la partie interface graphique. L'ensemble du projet est structuré selon une architecture orientée objet claire, et respecte les principes de modularité, de réutilisabilité et de séparation des responsabilités.

La solution comprend 11 classes principales, organisées autour 4 grands modules :

- Modèle de données : CarteVille, Intersection, Route.
- Navigation: Trajet, GPS, Vehicule, TrajetAnimator
- Evénementiels : Evenement, Accident, Congestion
- Interface et visualisation : GPSInterface

Le projet peut être lancé directement à partir de la classe GPSInterface, qui hérite de Application de JavaFX.

3.2 Organisation des fichiers et classes

Le projet a été développé en suivant la convention Maven, ce qui permet une organisation standardisée du code, facilitant la compilation, la gestion des dépendances (notamment JavaFX) et l'exécution.

gps-management/

├── pom.xml ← Fichier de configuration Maven

└── src/

└── main/

└── java/

└── labo/ ← Package Java contenant tout le code source

├── GPSInterface.java ← Interface graphique principale (JavaFX)

├── GPS.java ← Moteur de calcul des trajets

├── CarteVille.java ← Graphe du réseau routier

├── Intersection.java ← Nœud du graphe

├── Route.java ← Liaison entre intersections

├── *Trajet.java* ← *Chemin calculé (liste d'intersections)*

├── *Vehicule.java* ← *Représentation visuelle du véhicule*

├── *TrajetAnimator.java* ← *Animation du déplacement*

├── *Evenement.java* ← *Classe abstraite d'événement*

├── *Accident.java* ← *Événement de type accident (impact fort)*

└── *Congestion.java* ← *Événement de type congestion (impact variable)*

Point d'entrée :

Le point d'entrée de l'application est la classe GPSInterface.java, qui lance l'interface graphique et gère tous les événements utilisateur.

Cette structure respecte les conventions de développement modernes en Java, garantit une bonne séparation des responsabilités, et rend le projet facilement extensible et maintenable.

3.4 Qualité du code

a) Respect des bonnes pratiques

- ✓ Les noms de classes et méthodes sont explicites (comme: calculerCheminOptimal, appliquerImpact, simulerCongestion).
- ✓ La documentation interne a été rédigée dans toutes les classes à l'aide de commentaires.
- ✓ L'encapsulation a été respectée : les attributs sont privés, et l'accès se fait par getters/setters si nécessaire.

b) Réutilisation de code et adaptation

Nous avons réutilisé une version de base de l'algorithme de Dijkstra, que nous avons modifiée pour :

- ✓ Intégrer les événements dynamiques ;
- ✓ Permettre une exécution à partir d'un point intermédiaire (pour le recalcul) ;
- ✓ Être compatible avec un graphe à poids variables.

3.5 Particularités techniques rencontrées

a) Changement de technologie

Une première version a été esquissée avec Swing, mais la représentation dynamique du graphe, les animations, et l'ergonomie ont été difficilement réalisables. Nous avons donc migré vers JavaFX, qui offre des outils puissants pour :

- ◊ Dessiner et manipuler graphiquement des objets (ex : Line, Circle) ;
- ◊ Gérer les événements utilisateur (Button, TextField, Slider) ;

- ◊ Animer des objets avec PathTransition, PauseTransition, etc.

b) Animation et recalcul en temps réel

La complexité de ce projet réside dans la programmation par événements. Il a fallu déclencher automatiquement des traitements (recalcul de trajet, mise à jour des poids, redessiner le chemin) dès qu'un événement est détecté pendant l'animation. La classe TrajetAnimator centralise cette logique.

3.6 Exemple d'animation déclenchée par un événement

Lorsqu'un accident est simulé pendant que le véhicule se déplace :

- ❑ Le bouton "Simuler accident" déclenche l'ajout de l'événement dans la liste.
- ❑ Le Vehicule atteint une intersection concernée.
- ❑ Le TrajetAnimator vérifie la présence d'un événement via verifierEvenements().
- ❑ L'événement est appliqué (appliquerEffet()), modifiant les poids.
- ❑ Le système appelle recalculerTrajet(), et l'animation redémarre sur le nouveau trajet.

4- Révision du travail et qualité

4.1 Jeux de tests fonctionnels

Pour valider notre système de guidage GPS, nous avons mené plusieurs tests dynamiques, visant à évaluer le comportement de l'application dans des situations variées de circulation. Chaque test ci-dessous est accompagné d'une description, d'un scénario, et de captures d'écran dans le rapport final.

Test 1 – Itinéraire simple sans événement

Départ : Intersection A

Arrivée : Intersection F

Événement : Aucun

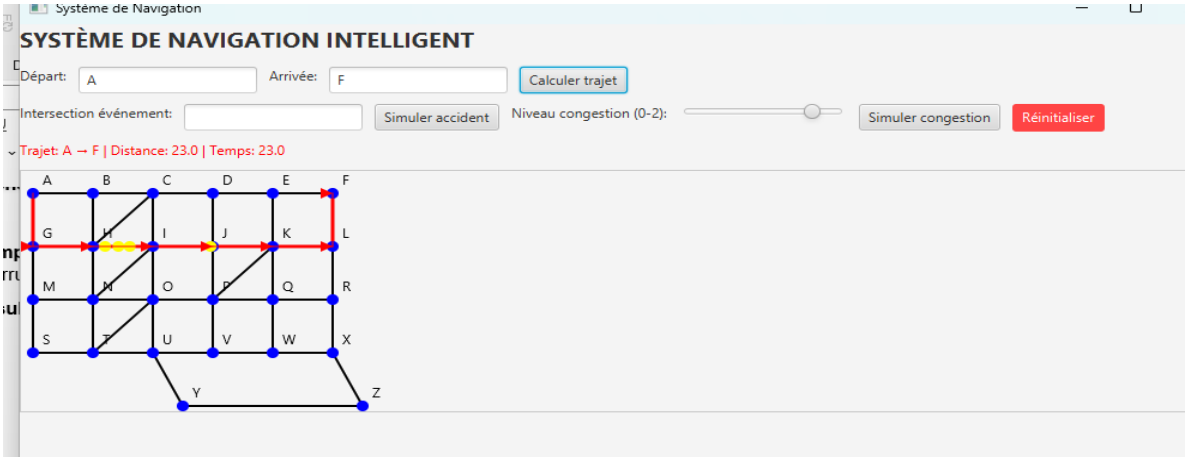

Figure 3: Simulation d'un trajet entre les points A et F

Comportement attendu : Calcul du plus court chemin initial, déplacement fluide du véhicule, aucune interruption.

Résultat observé : Le trajet optimal a été affiché, l'animation s'est déroulée sans recalcul. Bien que visuellement le chemin le plus court semble être la ligne droite, l'itinéraire suivi est en réalité plus court, car il dépend du poids attribué à chaque arrêt.

Test 2 – Congestion modérée sur une route intermédiaire

Départ : Intersection C

Arrivée : Intersection M

Événement : Congestion de niveau 1 (×3) sur l'intersection N

Comportement attendu : Au moment où le véhicule atteint ou approche N, un recalcul du trajet doit être déclenché pour éviter la zone congestionnée.

Résultat observé : Le trajet initial a été annulé dès la détection de congestion, un nouveau chemin a été calculé automatiquement, avec affichage des flèches directionnelles mises à jour.

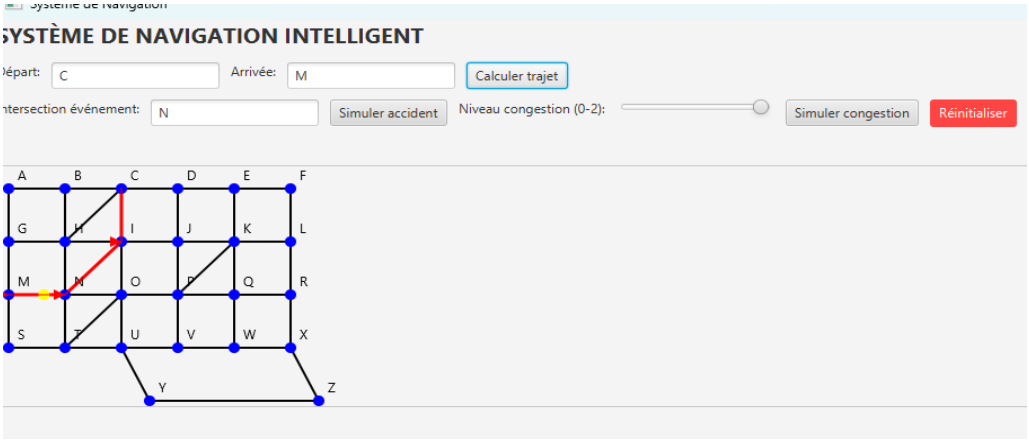

Figure 4: Simulation de trajet entre les points C et M

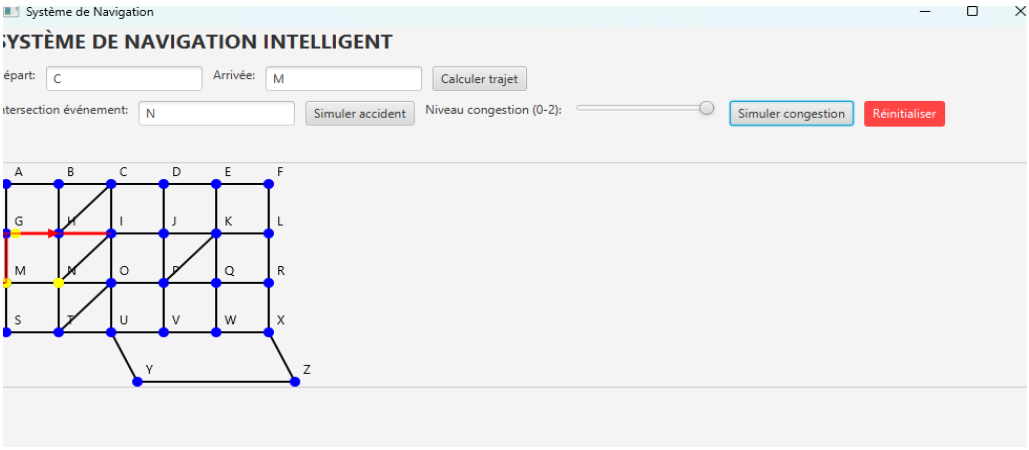

Figure 5: Après la simulation de l'accident au point N

Test 3 – Accident sur le chemin en cours

Départ : Intersection D

Arrivée : Intersection S

Événement : Simulation d'un accident sur l'intersection G

Avant la simulation de l'accident

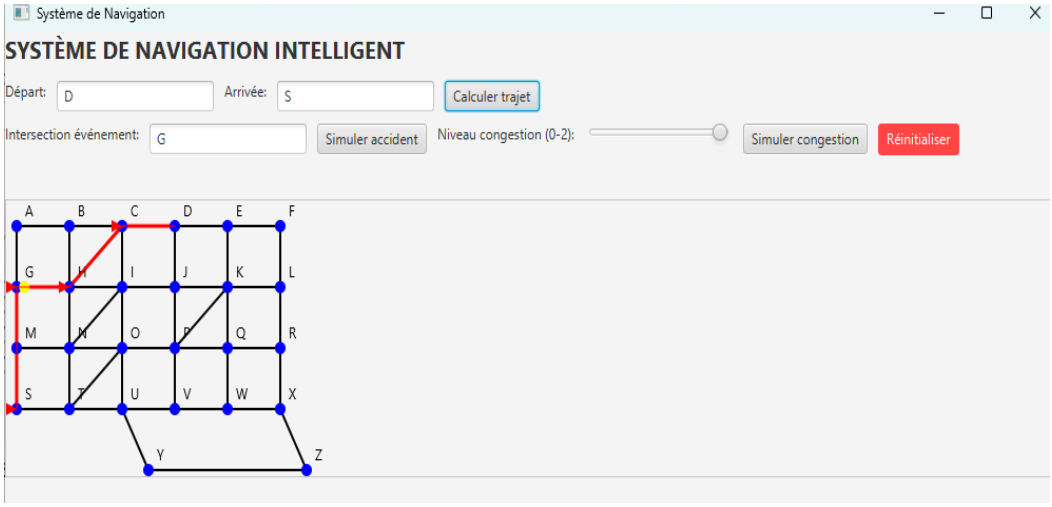

Figure 6: Trajet entre les points D et S avant la simulation de l'accident

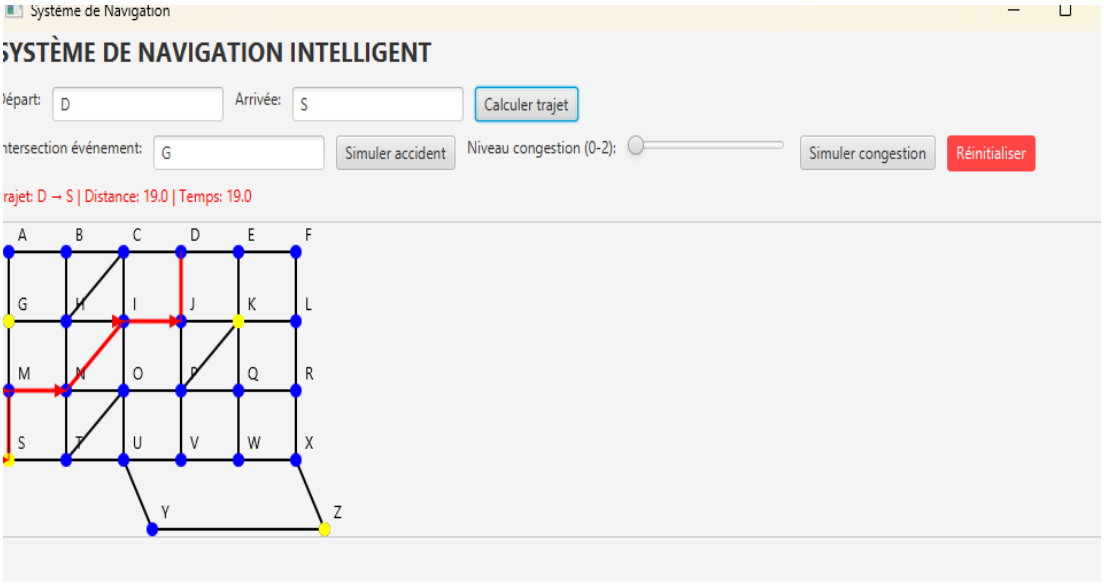

Figure 7: Trajet après la simulation de l'accident au point G

Comportement attendu : L'algorithme doit éviter la zone impactée (multiplicateur de poids élevé) et proposer un détour.

Résultat observé : Le recalcul a bien été déclenché. L'animation du véhicule a été temporairement suspendue, puis relancée avec le nouveau trajet recalculé.

Test 4 – Réinitialisation du réseau

Action : Clic sur le bouton « Réinitialiser »

Comportement attendu : Tous les poids doivent être restaurés à leur valeur initiale, les événements supprimés, le graphe réaffiché en mode neutre.

Résultat observé : Le système a bien effacé tous les événements actifs, réinitialisé les routes et permis un nouveau calcul propre.

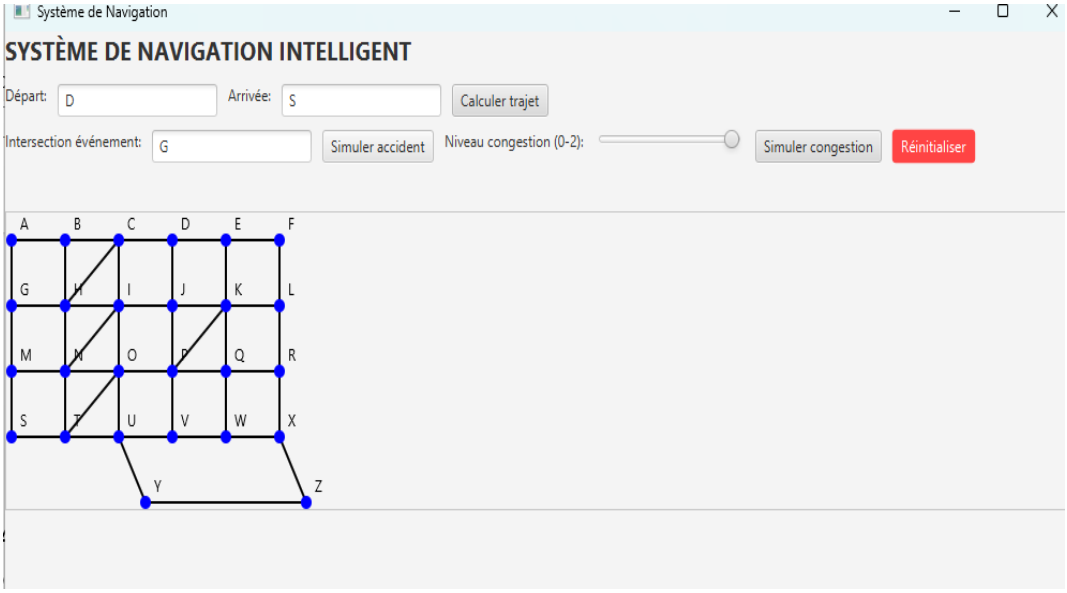

Figure 8 : Le système après avoir cliqué sur réinitialiser

4.2 Fiche de révision technique du code

Pour assurer la qualité du code, nous avons mené un processus itératif de révision et d'amélioration. Voici un tableau synthétique des observations et des actions correctives :

Critère de qualité	Observation	Action prise
Respect des conventions de nommage	Certains noms initiaux étaient ambigus (Node, Point)	Renommage explicite en Intersection, Trajet, etc.
Méthodes trop longues ou imbriquées	calculerCheminOptimal était trop dense	Refactorisé en sous-méthodes (initGraph(), dijkstraLoop())
Manque de commentaires	Classes initiales peu documentées	Ajout de commentaires JavaDoc pour chaque classe et méthode
Portabilité du code	Dépendance au fichier texte de graphe non contrôlée	Ajout de vérification d'existence et de message d'erreur
Multiplication de responsabilités dans GPS	Gestion des événements et du graphe dans la même classe	Délégation partielle vers Evenement et ses sous-classes

Tests manuels seulement	Aucun test unitaire formel	Intégration de tests fonctionnels avec scénario utilisateur

Tableau 6: Fiche de révision technique du code

4.3 Qualité du travail d'équipe

a) Structure de l'équipe

L'équipe était constituée de six étudiants avec une répartition claire des rôles. Un membre jouait le rôle de responsable de communication avec la professeure, assurait la cohérence des livrables et le respect du calendrier.

b) Organisation et gestion du projet

Planification initiale en présentiel

Réunions hebdomadaires sur Teams

Dépôt du code source sur GitHub

Revue croisée du code par binôme avant intégration

c) Répartition du travail

Conception graphique et test : Davidson

Modèle objet et logique : Maelisse et Nelie

Animation et gestion événementielle : Yasmine et Trésor

Codage: Trésor et Mechack

Rédaction du rapport : répartition équitable entre tous les membres

d) Partage de connaissances

Des sessions internes de formation ont été organisées pour :

Apprendre JavaFX (remplaçant Swing),

Maîtriser Draw.io pour la conception UML,

Comprendre l'adaptation de l'algorithme de Dijkstra.

Conclusion

Le développement de ce système de guidage GPS a constitué une occasion concrète de mettre en application les principes fondamentaux de la programmation orientée objet et de la programmation par événements dans un contexte réaliste, inspiré des systèmes de transport intelligents.

À travers les différentes phases du projet, de l'analyse des besoins jusqu'à l'implémentation complète, nous avons conçu une application capable de :

- Modéliser un réseau routier sous forme de graphe pondéré ;
- Calculer dynamiquement le chemin optimal à l'aide d'un algorithme de Dijkstra adapté ;
- Réagir à des événements affectant le trafic (accidents, congestions) ;
- Adapter l'itinéraire d'un véhicule en déplacement en temps réel ;
- Afficher visuellement le trajet, le graphe et l'évolution de l'environnement via une interface JavaFX interactive.

Ce projet a exigé une approche rigoureuse de la conception logicielle, l'utilisation d'UML pour modéliser les classes, une bonne gestion des événements asynchrones, et un travail d'équipe structuré. L'abandon initial de Swing au profit de JavaFX s'est avéré être un choix déterminant pour atteindre les exigences graphiques et interactives du système.

Sur le plan pédagogique, ce travail a renforcé notre capacité à :

- Structurer un projet objet complexe ;
- Développer une interface fonctionnelle et intuitive ;
- Intégrer des traitements asynchrones dans une boucle de simulation ;
- Appliquer des critères de qualité logicielle à toutes les étapes du cycle de vie du logiciel.

En somme, ce projet représente une synthèse réussie des apprentissages du cours Programmation II, et constitue une base solide pour des extensions futures comme l'ajout de nouveaux types d'événements, une intégration réseau ou l'enrichissement des scénarios de simulation.