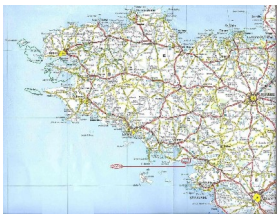
Mini Projet C++ 2025

ISEN CIR2 Brest



Calcul d'itinéraire

Ce projet a pour objectif la réalisation en langage C++ d'une application graphique de calcul d'itinéraire routier.

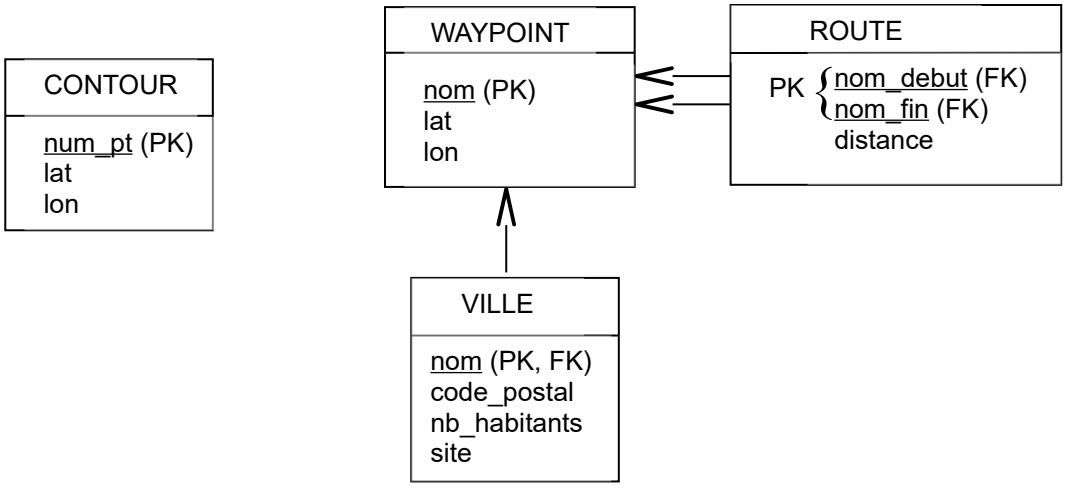
Le but est de coder un algorithme de recherche du plus court chemin entre deux nœuds d'un graphe, ici un réseau routier, et de représenter graphiquement le chemin trouvé (à l'aide de la librairie *Qt*, cf. TP N°8).

Les informations fournies sont : un contour géographique (trait de côte et limite Est), les « *waypoints* » (nœuds du réseau routier = sommets du graphe), les routes entre ces « *waypoints* » (réseau routier = arcs du graphe). Certains de ces « *waypoints* » sont des villes, mais pas tous : il y a aussi des ponts, des échangeurs, des intersections routières,... Seules les villes peuvent être des points de départ ou d'arrivée pour le calcul d'itinéraire.

Tous ces éléments sont localisés en coordonnées géographiques : latitude-longitude en degrés et décimales de degrés. Les latitudes Nord sont positives, les longitudes Ouest négatives et Est positives. Les distances des routes sont données en km.

Les informations sont stockées dans une base de données *MySQL* (base de données *itineraires*). Le fichier *itineraires.sql* fourni permet de créer les tables et d'y charger des données. Les données peuvent être extraites de cette base de données à l'aide de fonctions C++ (nécessite *libmysqlcppconn*, cf. TP N°6).

Modèle physique de la base de données *itineraires* :

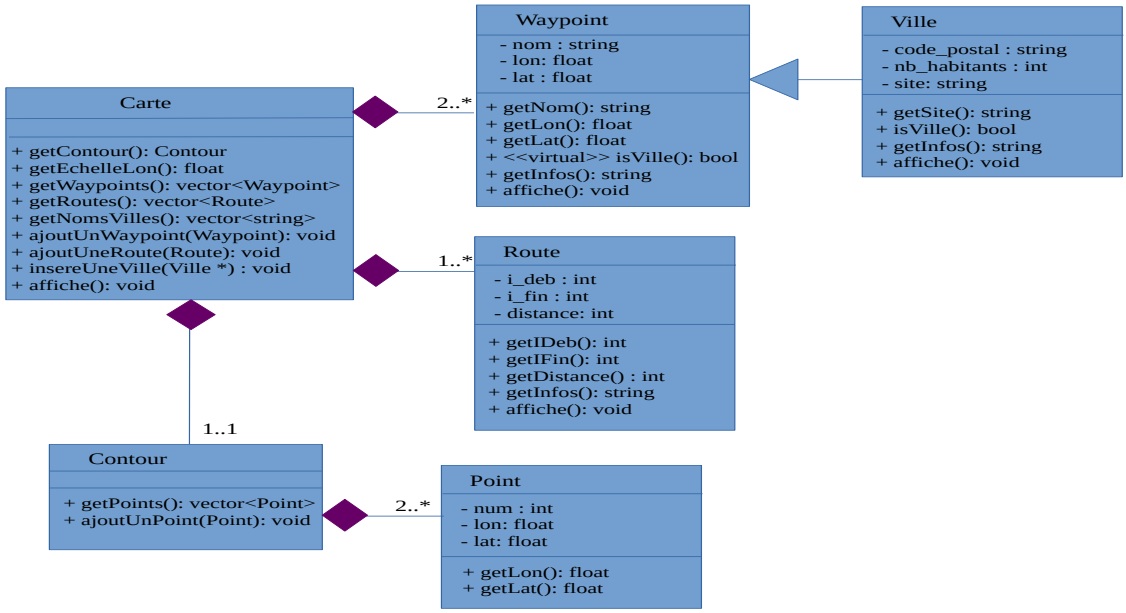


Quelques explications sur ce modèle :

- la table **Contour** contient tous les points de contour. Elle est indépendante du reste du modèle,
- la table **Waypoint** contient tous les « *waypoints* » du réseau routier.,
- la table **Route** contient toutes les routes du réseau routier. Une ligne de la table fait référence à deux *waypoints* (début et fin),
- la table **Ville** contient toutes les villes du réseau routier. Une ligne de la table fait référence à un *waypoint* de même nom. Pour avoir la localisation d'une ville, il faut donc accéder à celle du *waypoint* associé.

Classes et fonctions du Modèle (M de MVC)

Contrairement au TP N°8, le code qui concerne le Modèle n'est pas fourni, c'est-à-dire les fonctions d'extraction de la base de données et les classes et méthodes du Modèle. Vous pouvez vous inspirer du code fourni pour le TP N°8 et du modèle UML simplifié ci-dessous (pour information, vous n'êtes pas obligés de le suivre).



Remarques sur ce modèle :

- la classe ***Carte*** sert à regrouper les accès à toutes les données du modèle, à travers deux tableaux : *waypoints* (*vector<Waypoint *>*) et *routes* (*vector<Route>*), et un attribut *contour* (de classe *Contour*). Ces trois attributs ne sont pas indiqués sur le diagramme car ils se déduisent des liens de composition entre classes. Le tableau *waypoints* est un tableau de pointeurs, cela permet d'y stocker des objets *Waypoint* ou des objets *Ville* (polymorphisme, cf. ci-dessous).
- la classe ***Waypoint*** est la racine d'une hiérarchie polymorphe : elle possède une méthode virtuelle `isVille()` qui retourne *false*.
- la classe ***Ville*** est une classe dérivée de *Waypoint* : la méthode virtuelle `isVille()` y est redéfinie et retourne *true*.
- la classe ***Route*** permet de relier deux *waypoints* mais elle n'a pas de lien direct avec la classe *Waypoint :* en effet les deux *waypoints* reliés sont repérés par leur indice dans le tableau *waypoints* de la *Carte* (*i_deb* et *i_fin*).

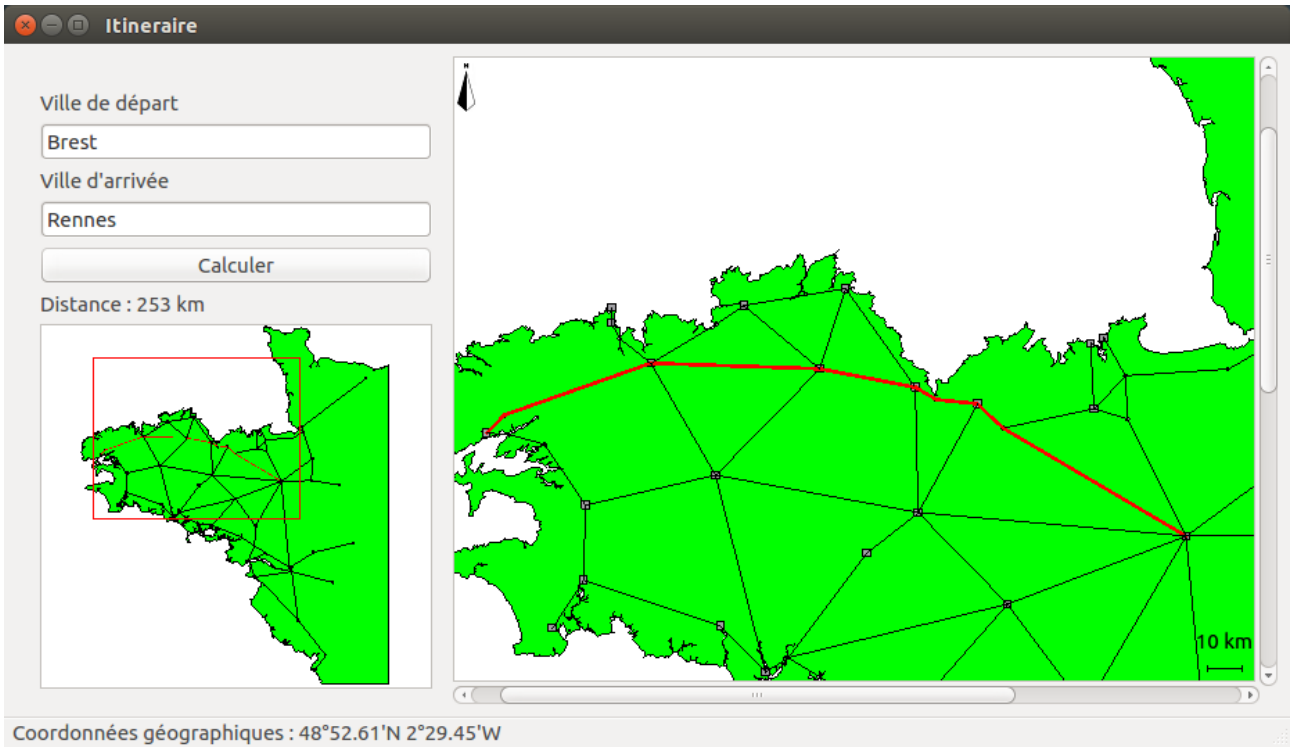
Remarque importante : il faut ajouter à ce diagramme une classe permettant de stocker le graphe (sous forme de tableau d'adjacence –à 2 dimensions– ou de liste d'adjacence, à votre choix). Cette **classe *Graphe*** doit avoir un accès aux tableaux *waypoints* et *routes* de la *Carte*. Les méthodes nécessaires pour la classe *Graphe* sont :

- *voisins*() qui prend en paramètre l'indice d'un *waypoint* et retourne un tableau (ou vecteur) des indices des *waypoints* voisins,
- *distance*() qui prend en paramètre les indices de deux *waypoints* voisins et retourne la distance entre les deux (en km),
- une fonction de calcul du plus court chemin, qui prend en paramètre les indices de deux *waypoints* quelconques et retourne un tableau (ou vecteur) des routes à suivre entre ces deux *waypoints*.

D'autres méthodes peuvent être rajoutées, selon vos besoins.

Cahier des charges minimal :

- Partie fonctionnelle : le programme doit charger les informations de la base de données *itineraires*. Il doit permettre de saisir une ville de départ et une ville d'arrivée. Après avoir lancé le calcul du chemin le plus court, il doit afficher la distance entre ces deux villes et, sur une carte, l'itinéraire à suivre.
- Affichage graphique : il doit ressembler à la vue suivante (assez proche du « *layout* » du TP N°8). Une grande vue zoomable et une mini-vue fixe (avec tracé du cadre de la grande vue) sont obligatoires, ainsi que les champs de saisie des villes. L'affichage des coordonnées géographiques du curseur, de l'échelle et de la flèche du Nord est facultatif.



Vos ajouts (facultatifs) :

- Vous pouvez ajouter les éléments facultatifs indiqués ci-dessus (coordonnées, échelle, flèche N).
- Toute amélioration de l'interface est la bienvenue (par exemple auto-complétion du nom des villes à partir des premières lettres saisies, ouverture de la page Web d'une ville par un clic droit,...).
- Autres ajouts selon votre imagination, à partir du moment où l'aspect général de l'application est respecté.

Contraintes de programmation :

- Le programme devra s'exécuter sous Linux.
- Le programme devra bien sûr être conçu selon une conception «objet» et les règles de bonne programmation (pas de variables globales, nom des variables explicite,...) devront être respectées.
- Votre code doit être commenté et lisible (indenté, aéré,...).

Livraisons et tests demandés :

- Ce projet sera réalisé en trinôme. Les échanges d'informations (verbales) entre groupes sont autorisés, mais tout échange ou recopie de code est interdit. Je passe des outils de comparaison de code et anti-plagiat/Internet. Vous risquez un 0/20.

- Les sources et fichiers nécessaires à la compilation et l'installation devront être fournis. La qualité du programme source sera notée.
- Une démonstration du programme aura lieu en fin de projet. Elle permettra de produire une partie de la note sur la qualité graphique et le bon fonctionnement du calcul d'itinéraire.

Déroulement :

- pas de créneaux de cours spécifiques en dehors des séances indiquées ci-dessous
- 2 séances encadrées prévues le 19 mars (présentation et démarrage) et le 2 avril, mais vous devrez déjà avoir bien avancé le projet pour que cette dernière séance soit profitable.
- Démonstration et livraison lors d'une séance supplémentaire (après les vacances de Pâques, date à préciser).

Barème :

- le barème précis vous sera donné ultérieurement.
- le principe : le respect du cahier des charges minimal et de la qualité du code est noté sur 20. Vos ajouts sont des bonus. Dans le meilleur des cas vous pouvez donc avoir plus de 20, et cette note peut bonifier votre moyenne de semestre en C++, qui ne pourra cependant pas excéder 20!