Projet V2V – Étape 1 : Visualisation interactive du territoire

Auteur : Filip Barbutov
Université de Haute-Alsace – Université de Strasbourg
Licence 3 Informatique

Objectif du projet

Cette première étape du projet **V2V (Vehicle-to-Vehicle)** a pour but de concevoir une interface graphique permettant de **visualiser un territoire réel** sur une carte interactive. Elle sert de base pour les étapes ultérieures : construction du graphe routier, simulation des déplacements des véhicules et visualisation des connexions dynamiques.

Objectifs techniques :

- Charger une carte réelle à partir de tuiles en ligne (MapTiler).
- Afficher et naviguer sur la carte (zoom et déplacement).
- Préparer la structure pour superposer le graphe et les véhicules.

Outils et technologies

Langage principal	C++17
Framework GUI	Qt 5 (QtWidgets, QPainter, QNetworkAccessManager)
Gestion de projet	CMake
Source de cartes	MapTiler (API REST, tuiles raster 256×256)
Système de test	Ubuntu 24.04 (VirtualBox)

Architecture du code

```
projet_v2v/
 include/
    map_view.h        → Déclaration de la classe MapView
 src/
    map_view.cpp      → Implémentation complète de MapView
    main.cpp          → Fenêtre principale + intégration MapTiler
 data/
    strasbourg.png    → Image locale (fallback)
 CMakeLists.txt
```

Structure principale

- **Classe `MapView`** : hérite de `QWidget` et gère le rendu cartographique, la navigation et l'affichage du HUD (zoom, coordonnées, échelle).

— `main.cpp` : crée la fenêtre principale Qt, initialise la carte MapTiler, et gère la clé d'API via les variables d'environnement.

Mode en ligne (MapTiler)

Clé API et configuration

1. Créer un compte gratuit sur cloud.maptiler.com.
2. Copier la clé API (exemple : `UJmRLIswAr960kmciD8v`).
3. L'exporter dans le terminal :

```
export MAPTILER_KEY=UJmRLIswAr960kmciD8v
```

URL des tuiles utilisées

```
https://api.maptiler.com/maps/streets/256/{z}/{x}/{y}.png?key=MAPTILER_KEY
```

Coordonnées par défaut

```
view->setCenterLonLat(7.7521, 48.5734, 13); // Strasbourg
```

Fonctionnement interne

1. Projection et rendu

Les coordonnées géographiques (longitude, latitude) sont converties en coordonnées pixel selon la projection **Mercator**. Chaque niveau de zoom double la résolution ($2^z \times 2^z$ tuiles).

2. Chargement des tuiles

— Calcul des indices de tuiles visibles (z, x, y).
— Téléchargement asynchrone via `QNetworkAccessManager`.
— Mise en cache mémoire (`QCache`) pour réduire les requêtes.

3. Navigation

Action	Méthode
Zoom avant / arrière	Molette / touches + et -
Déplacement	Clic gauche + glisser / flèches clavier
Recentrage	Redémarrage (Strasbourg par défaut)

Compilation et exécution

Dépendances (Ubuntu)

```
sudo apt update
sudo apt install -y qtbase5-dev libqt5network5 libssl-dev cmake g++
```

Compilation

```
cd ~/Desktop/Reseau/projet_v2v
cmake -S . -B build
cmake --build build -j
```

Exécution

```
export MAPTILER_KEY=UJmRLIswAr960kmciD8v
./build/ConnectedVehicles
```

Résultat attendu

— Fenêtre interactive Qt.
— Carte réelle de Strasbourg affichée.
— HUD : niveau de zoom, coordonnées, barre d'échelle.
— Zoom fluide, déplacement continu à la souris et au clavier.