Software Engineering Project Report

Belal Hmedan & Deng Jianning
University of Burgundy

Thursday 12th December, 2019

Contents

Introduction

BUILDING an Offline map software is a challenging project, because it requires a good knowledge in Geography, Mathematics, Graph Theory, and Computer Science.

At the very beginning of such project many questions must be asked, such as: What places should the map covers? For which application the map will be used? Which Information is necessary, and which information is useless so it can be filtered, so total amount of Data will be reduced.

This Project is a try to answer some of many questions that any computer science student will ask while starting such kind of projects. Maybe we didn't answer all The questions because of time limit, but we hope that this project will be useful For Other students to get some answers , and to get the fastest way between two Locations, That's why we selected 20 locations carefully to satisfy the daily needs Of students like universities, cool restaurants, pharmacy, hospital, shopping centers, students residences, etc.

This project with all cool features and nice looking objects, is nothing more than "Hello Map", because Cartography is a great science developing everyday for both civil and military use, and everyday there is changes on our planet leads to remarkable changes on the Map, starting from environmental changes, and not ending by Human-Made changes.

Chapter 1

Project Description

1.1 Goal of the Project

The goal of this project is to develop a software to locate (and rate) various buildings, such as schools, university buildings, major offices, hospitals, various shops, cool restaurants, streets, roads, parks, and other interesting points in Le-Creusot. The key idea is to develop a software to visualize Le-Creusot, its streets, some buildings, roads, and parks, in which the user can perform several actions, such as asking for an itinerary between two (or passing through more) points.

1.2 Completeness of the Project

1.2.1 Place Locating

Twenty locations were provided on this project, and there is ability to add many other places due to requirements, but user can't add places, only programmer. User's privilege on this stage of the project is limited to chose only from a list of those twenty places.

1.2.2 Place Rating

This Feature isn't added yet, but it maybe available in next releases.

1.2.3 Routing

Routing in this stage only by foot, expanding travel Methods to car, bicycle, and Train is possible in the following releases.

1.3 Overview of the Project

The goal of this project is to develop a software to show the whole picture of Le Creusot city. Apart from that, it also locates various amenities, such as schools, university buildings, major offices, hospitals, various shops, cool restaurants, streets, roads, parks, and other interesting points in Le-Creusot. With this map, user can search, rate and comment for the places of interests and do routing to get there. I the follow section, a more clear picture of the goals and achievements of this picture will be shown.

This project would be a great opportunity for us to practice and get a little bit more familiar with the C++. Besides, it's also a chance for us to project management. This two parts of contents will also be explain in the following.

1.4 Dataset Introduction

In this project, we decide to use OSM(open-street-map) data as the base data. OSM data provides huge amount of detailed but abstract details for users. We choose this data-set because the following advantages:

1. It provide detailed data base on nodes, ways and relations. Which will be more convenient to implement the Dijkstra's algorithms for routing.
2. With the standard format of database, this application can be applicable for other cities or even countries out of box.
3. It is a little bit more complicated, which will help us get more practice.

Of course, choosing this way to develop the application means more complexity, more time consuming and more difficult to manage the project. But we still go for it because no pay no gain :D

1.5 Completeness of the Project

For this release of the application

1. File I/O
   - ☑ Loaded by the user
   - ☒ Modified and saved by the user
2. Map Displaying
   - ☑ Graphically display the map
3. Available Location
   - ☑ Locations required in the reference is available
   - ☑ Almost all locations contained in OSM data is available
4. Roads and routing
   - ☑ Presenting roads
   - ☑ Different kinds of roads are distinguished graphically.
   - ☑ Basic routing with 20 places as beginning or end
   - ☒ Different travel mode
5. Offline
   - ☑ All the features is based on offline OSM data

1.6 Dependencies

1. **Boost**
   Need to be installed if you don't have it. Follow the installation guide below.

   - **Windows**
     Using vcpkg from Microsoft (https://github.com/microsoft/vcpkg) you can install Boost libraries through command line window as following:
     vcpkg install boost
   - **Ubuntu**
     If it is not pre-install in your Ubuntu, a single command should help you do this (for Ubuntu 18.04):

     ```
     $ apt-get install libboost-all-dev
     ```

2. **Libosmium**
   This is a header-only library, no need for installation. Just includ the header files in the project is OK to go. This is already put in the project repository.

3. **Protozero**
   This is a header-only library, no need for installation. Just includ the header files in the project is OK to go. This is already put in the project repository.

4. **Zlib**
   Need to be installed if you don't have it. Follow the installation guide below.

   - **Windows**
     Using vcpkg from Microsoft (https://github.com/microsoft/vcpkg) you can install Zlib library through command line window as following:
     vcpkg install zlib
   - **Ubuntu**
     If it is not pre-install in your Ubuntu, a single command should help you do this (for Ubuntu 18.04):

     ```
     $ apt-get insatll zlib1g-dev
     ```

5. **Expat**
   Need to be installed if you don't have it. Follow the installation guide below.

   - **Windows**
     Using vcpkg from Microsoft (https://github.com/microsoft/vcpkg) you can install Expat library through command line window as following:
     vcpkg install expat
   - **Ubuntu**
     If it is not pre-install in your Ubuntu, a single command should help you do this (for Ubuntu 18.04):

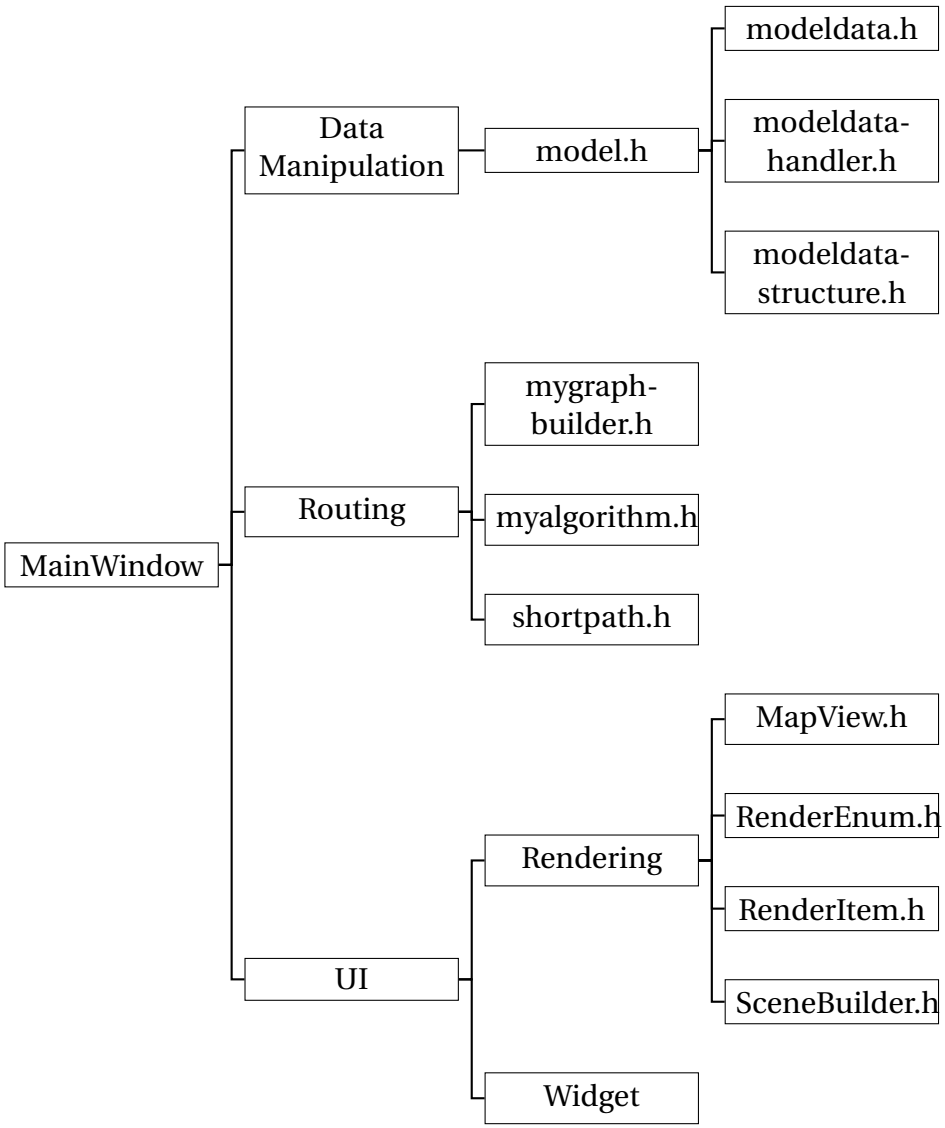
     ```
     $ apt-get insatll expat
     ```

6. **PThread** (for Linux)
   Use the following command to install the library sudo apt-get install libpthread-stubs0-dev

   ```
   $ apt-get insatll expat
   ```

Chapter 2

Code Design and Structure



2.1 Data Manipulation

2.1.1 Overview

This part of code aims to handle the OSM data. We use *Libosmium* [1] to load the file and apply basic filtering to the raw data with the code in **modelDataHandler.h**. That is basically a modification from the example of the library. After that, all the data will be loaded on heap using *std::map* in the **modelData.h**. The file **modelDataStructure.h** defines the basic type of the OSM element. And **model.h** is the top level package of all the information we read from the file and provide interface for other part of the code to access those data. In the following subsections, I will give detailed explanations for some of those code.

2.1.2 ModelData Class

This class actually contains all available information we get from the OSM file. It is basically a container for several std::map for different kind of data(nodes, ways, relations, etc.) and other supported std::map for place searching. The reason why we load all file in the heap using map is that the interface to access data in *Libosmium* is not so convenient. By using the std::map, we can easily access all available with the id of the data. Although it has a drawback that when we load a large file, it would takes a large mount of ram. But since we are now only focus on the city level map, this is not an issue. It also declares to friend **modelDataHandler** so that the OSM file data can be written in the private member of **ModelData**. And the handler will only be called when we load a new file, which ensures the raw data will not be modified again when the loading is finished.

2.1.3 ModelDataHandler Class

This class is modified from an example of *Libosmium*. It is responsible for extracting data from OSM file while filtering different types of roads and buildings. This class will only be called when the user load a new file.

2.2 Algorithm and Routing

This part of code organized in three different classes, each of them contains header, and source File:

2.2.1 MyGraphBuilder Class:

This class mainly handles data provided from our data structure represented by the Model which depends on the OSM file in its compressed format PBF. This class mainly takes the nodes included inside ways only from the Model as an input, then builds vertices and edges between vertices to construct the graph, which in turn will be as an input to the next stage which is nothing except Dijkstra algorithm. The graph in my approach is **adjacency list** proposed by Boost Graph Library. The need for this procedure comes from the fact that: finding shortest path will be implemented through an algorithm, and the algorithm deals with a graph, not a map, so I wrote this class to build the desired graph from the data structure we imported from OSM file. The input for this class is the Model (data structure) extracted from OSM file, and the output is a bidirectional weighted graph. bidirectional: because itâĂŹs easy to implement, in case we want improve it to directional graph, we must do a complicated constrains, and check each tag, to see the type of transportation, and if the way was one direction, or bidirectional.

Weighted graph: because we are calculating Euclidean distance between the nodes depending on longitude, and latitude.

2.2.2 MyAlgorithm Class:

This class takes the graph produced on the past class MyGraphBuilder, and the source node which we will start from, and runs Dijkstra algorithm proposed by Boost Graph Library to Find the shortest path between the source node, and all other nodes in The graph. Special function in this class called: getShortPath takes the destination node, then returns the shortest path between the source and destination only. In case that there was no path between nodes, the path (vector) will contain only one default node, so we donâĂŹt face issues due to empty vector.

2.2.3 ShortPath Class:

This class is just to reduce amount of code in the main window, it doesnâĂŹt do much, it just takes source node, destination node, and Model, then it builds the graph from the Model using MyGraphBuilder class, and runs MyAlgorithm class, and its special function getShortPath, finally the output is our shortest path.

2.2.4 Routing

Routing on this approach is just on foot, it's possible to add more transport methods later using tag filtering, building different transport maps, vertices, and edges, but unfortunately time was short for this project !

Isolated nodes which doesn't belong to any way are not included in building the Graph, because our purpose is to find a way between two nodes, so I didn't build the graph with all nodes, only connected nodes which belong to way. Euclidean distance between the nodes is our way to assign weight to edges between nodes, and of course the way or path is nothing except group of sequential edges between connected nodes.

2.2.5 Difficulties during implementation:

There is two kind of Difficulties here:

Difficulties we got rid of it:

1. **First difficulty was CMake didnâĂŹt find some libraries:**
   specially Boost Library, later we used qmake.

2. **Second difficulty was that edges canâĂŹt be built using idType nodes:**
   the solution for that was mapping the idType Nodes to unsigned int indexes through a map, so each node has ID, and vertex number.

3. **Third one is redundancy of nodes in different Ways:**
   the solution was to compare each node of way to the map we built as solution to problem1.2, so if the node already found, we donâĂŹt add it to our map, instead we just call it and get node index.

Difficulties we can solve, but it needs more time:

1. **Supporting different travel method.**
2. **Building directional graph instead of bidirectional one.**

Unexpected Difficulties

Losing member of the team at critical time of the project doubled the responsibilities to do more than what we have already planned to do, and forced us to work under pressure in some parts of the project which we didn't have ideas about it before enough time.

2.3 GUI

2.3.1 Overview

The GUI is composed by two part of the code. One uses a class promoted from **QGraphicsView** for displaying the map and interact with the users. The other one is for menu bars and routing selection as a complementary interaction entry. The allowed interactions are managed by a *Finite State Machine* in the class **MapView**. Those details will be well explained in the following of the report.

2.3.2 Rendering

MapView Class

This class is a class promoted from **QGraphicsView** which can be considered as the top level class for the rendering. It provides the users a graphical view of the map together with several ways of interactions. I decided to use the **QGraphics** framework because I've seen another open source rendering library *Marble* [2] with this framework. And after google a bit about this, I get to know this frame work can provide smooth graphics dealing with thousands of objects. So I decided to use it and can also get hints from *Marble*.

Figure 2.1 shows the graphical view of a map.

The following functions of **QGraphicsView** is overridden to provide the interactions.

1. **wheelEvent(QWheelEvent *);**

   This function is overridden to provide scaling of the map with mouse wheel, which is more convenient to the user. The default scaling factor is set to 0.001, ranging from 0.06 to 0.0001 with step at 1.2.

2. **mousePressEvent(QMouseEvent *);**

   This function enables users to drag the map and to select buildings directly on the map. At the very beginning of this design is thought as an entry for building details, ratings, comments and choosing routing places. But since we don't have time to implement the user rating system and routing algorithms will cause crashes when it failed to find a path, this entry now is hook to an empty operation with a right click menu.

3. **contextMenuEvent(QContextMenuEvent *);**

   This function will provide users an access to a right click menu, which is controlled by an embedded *finite state machine*. The state and transition of the *finite state machine* is explained in figure 2.2.

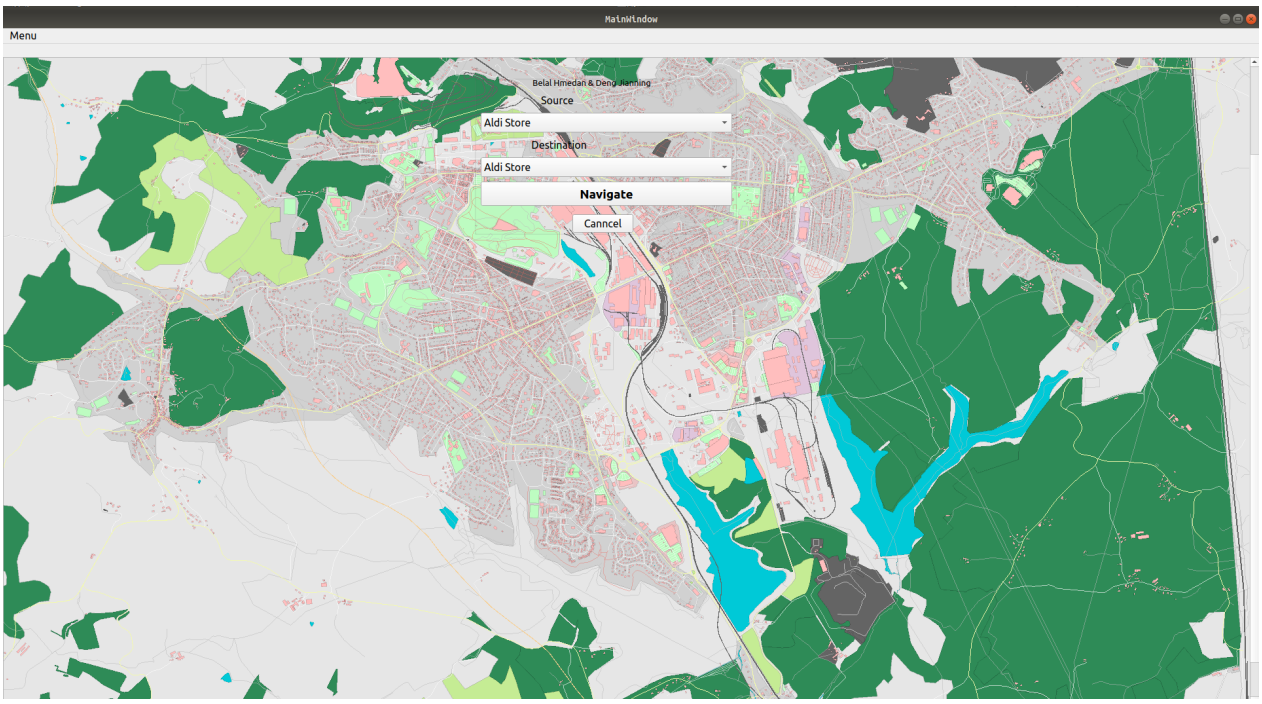


Figure 2.1: Graphical View

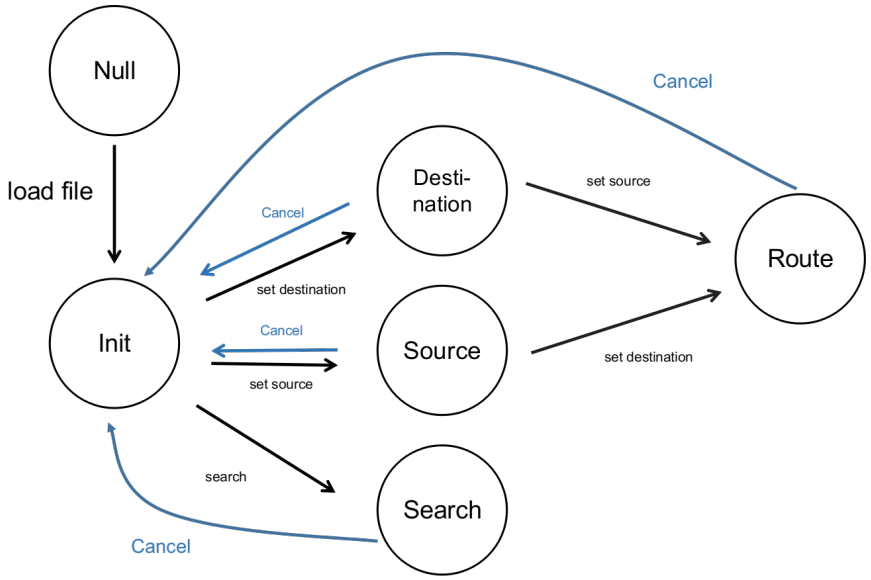


Figure 2.2: FSM

The user state was set to *Null* before loading the data file. In this state all the interaction will be disable to avoid segmentation fault. Once the data file is loaded, the state will change to *Init* to enable all the interactions. And the state will get changed as the user choose different options in the right click menu. And the option *cancel* will reset the state to *Init*. This help sort the logic for development. And the visual feedback is shown in figure 2.3.

SceneBuilder

This part of code responsible for creating a proper scene for **MapView** to display. It also provides an interface for drawing routes and pins. This class extracts information form the **modelData** to form multi polygons for builds and poly lines for roads and path.

(a) Place Searching

(b) Select route source.

(c) Select route destination.

Figure 2.3: Visual feedback for right click menu

RenderItem

This part of the code is to construct several basic item of the map rendering using **QGraphicsItem**. And in those items, we use enumerators to help set the z-value. The category logic uses ideas from *IO2D demo: Maps* [3], but not exactly the same. There're two types of enumerator for polygons and roads, which are specified in **renderItemEnum.h** file. Also, they are rendering styles are based on its type.

2.3.3 Difficulties for rendering

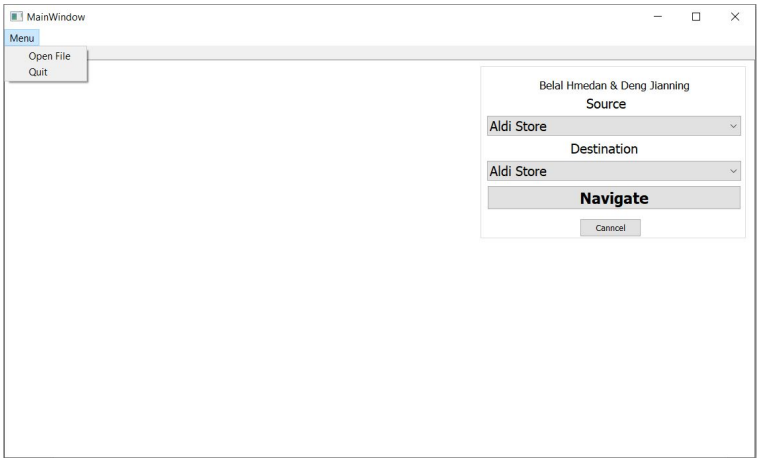
When developing this part of code, you need to have a global view of the whole project. Because this part is the bridge between database, graphical view and user interface. Which means your are dealing all the information you can get in the project. So, if we divide this part to several key points, those are the problems we solved:

1. filter and categorized the proper element from OSM data for rendering
2. properly draw buildings and roads in the *QGraphics* framework
3. locate the object in the view
4. build a right click menu for users
5. interaction logic control
6. interface for routing algorithm
7. visualize the place searching result and place selection

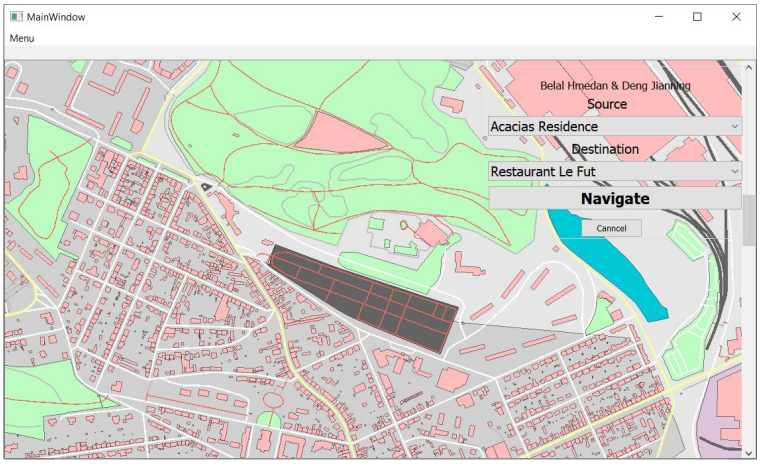
To achieve those above requires a certain knowledge on OMS database and Qt graphical development. Since both of them are pretty new to us, we need to spend more time to read and learn. And that results in a more difficult project management. Because you just don't have enough knowledge to estimate how much time you need to spend on this.

2.3.4 Widget

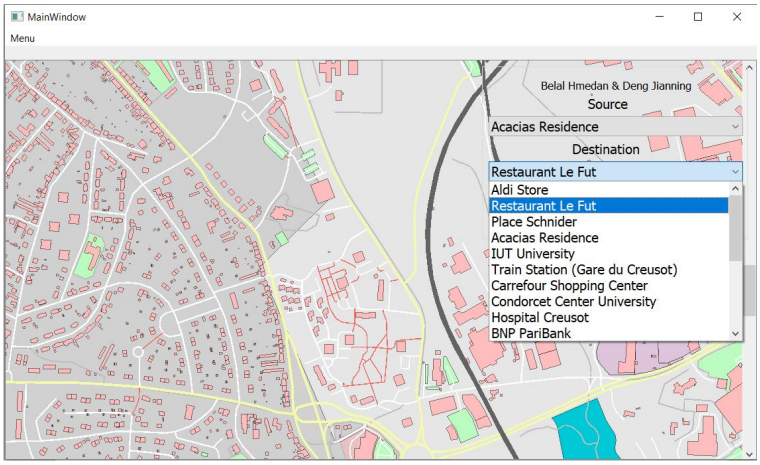
Running the program that will show a window with a Menu and four buttons source, destination, navigate, and cancel.



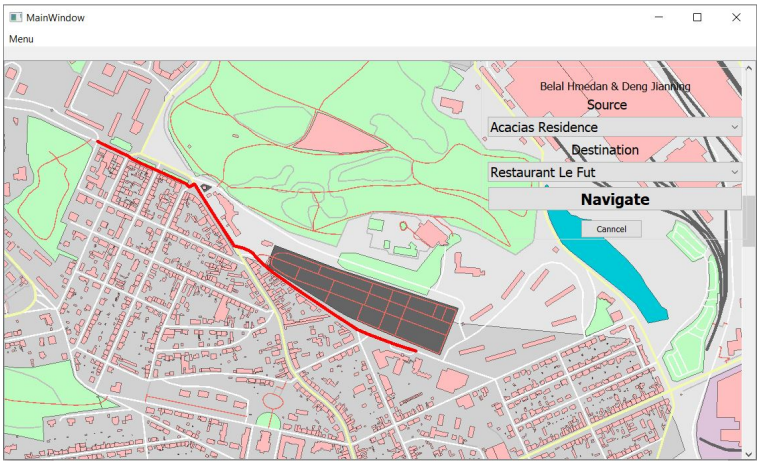
To start working, you should select (Menu - Open) to open the map file, which should be with (.pbf) extension. After you select your map, the software loads the map file showing the map.



The Drop-down lists source, and destination contains 20 places for each of them, all on Le Creusot city, where you can select your start point, and destination.



by clicking on navigate button, the shortest path will be drawn by red , so you can see the way you should follow.



Cancel button is to hide that path, and you can choose again your start point and target, so your path again will be drawn for you. To finish, you can select (Menu - Quit) to exit the software.

Chapter 3

Project Management

3.1 Overview of the Project Planning

We received the project at the end of September and the deadline is in Dec. 14th. So, we have about two and a half months' time for this project. Then after we teamed up with 3 students, we decided to divided the project in 3 different periods. First is the research period. We would spend about a month's time to study about this project. After having a basic directions, we shall then move to the second period development, which will also last for about a month. And at last we plan to release the application for test, so that we can fix some bugs and prepare our documentation. All the details about the research period is logged in the discussion log of this repository.
At about the beginning of October, when we have a basic direction, we decided to divide this project into three main functional part:

1. **Data Manipulation**

   This part will handle the OSM data input, building friendly interface for rendering and routing. So, it requires an in-depth understanding on the OSM data. Apart from that, it will also help build the user rating and comment database.

2. **Map Rendering**

   This part aims to render the OSM data, provide a graphical view of the map. It will also help build some common interactions like zooming, dragging and etc. Besides, it also need to provide the entry for user to rate and comment a place.

3. **Routing**

   This part is trying to implement routing algorithms on the OSM data.

So far, everything seems to be nice and neat. However, at Nov. 11th, one of our member choose to leave the team because of the different implementation ideas. So, we need to change our plan to guarantee a basic functional version can be done on time. So, the following will be the new work assignment to the two of us.

Belal

1. Routing algorithm implementation
2. Basic GUI widget

Deng

1. Basic data manipulation(reading OSM file only)
2. Map rendering

With those work, we can at least purpose a functional map application. And user rating and commenting will be considered as a advanced features, which means it has the lowest priority in this project.

3.2 Work schedules

This section contains a detailed schedule for each member in the team.

3.2.1 Belal schedule

1. October

Independent Research on, from 1st of October to 31 October
Task (1) : Duration 20 days.

(a) Get familiar with OSM data structure.

(b) Basic Math for path planning on map:

- Graph Theory review.
- Data Structures review.
- Algorithms review.

Task (2) : Duration 11 days.

(a) General implementation methods on map application

- Writing The pseudo code of the algorithm
- Specifying Data Requirements :
  Inputs - Outputs - Data Structure to be used.

(b) Check available libraries

2. November

Implementation, and Integration, from 1st of November to 30 November
Task (1) : Duration 22 days.

(a) Understanding standard templates library, boost graph library, and implementing them

- Understanding vector and map objects.
- Understanding how to build a graph using boost graph library
- Interfacing Dijkstra algorithm from boost graph library with the Model (data structure of the Project.

Task (2) : Duration 8 days.
Building friendly user interface to interact with user.

3. December

   Writing documentation, report, and preparing the presentation, from 1st of December to 14 December

   (a) Documentation writing.
   (b) Report writing.
   (c) Preparing defense presentation.

3.2.2 Deng schedule

Research Period

From Oct. 06 to Nov. 06: The main task in this period is to gather enough information about this project and form a big picture of it.

1. Get familiar with OSM data
2. Research on libraries about rendering
   Cartotype, Marble, IO2D, mapnik
3. Read source codes in *Marble* and *IO2D*
4. Read source codes and examples of *Libosmium*
5. Learn how to draw polygon using *QPainter*
6. Estimate if it is possible to build our own rendering library
   Documents for *Marble* is not so detailed and the source code is too complicated for us.
7. Think how to construct a easy-to-use database

Development Period

From Nov. 06 to Nov.17:

1. Try to run and modified example code of *Libosmium*
2. Implement the basic database using C++ STL.
3. Learn how to use *QGraphics* framework and its interaction mechanism
4. Try to dray simple things with *QGraphics* framework
5. Try to use data from database to draw buildings

From Nov. 17 to Nov. 30:

1. Sort useful data from database for rendering(buildings and roads)
2. Tweak the style and z-value for the *QGraphicsItem*
3. Update interface for routing
4. Implement item retrieving for the rendering
5. Design user interaction logic using *finite state machine*
6. Get familiar and implement with signal and slot for interaction
7. Implement place searching and visualize the result

Document Period

From Dec. 01 to the deadline of this project:

1. Fix bugs in place searching to avoid crashes
   Segmentation fault for those object without tag: name

2. Fix bugs in widgets to avoid crashes
   Segmentation fault when clicking on the button before loading the file

3. Fix bugs in FSM
   Segmentation fault because of incomplete reset to state *init*

Chapter 4

Summary

Bibliography

[1] github: https://github.com/osmcode/libosmium

[2] website: https://marble.kde.org/index.php

[3] website: https://kazakov.life/2018/06/07/io2d-demo-maps/

[4] Jeremy G. Siek, Lie-Quan Lee, Andrew Lumsdaine. *The Boost graph library : user guide and reference manual.* , “Book," *BGL*, pp. 161-322,Pearson Education, Inc., Reading, Massachusetts, 2002.