燕园导航PKU-MAP

2025春程序设计实习大作业报告

一、程序功能介绍

我们组具体实现了一个燕园校区的地图导航功能，主要服务于对学校不熟悉的同学以及前来参观的游客。

实现的主要功能包括：

- 标注自身所在的位置
- 搜索目标位置
- 任选两个位置返回最优路线

在主要功能的基础上，出于用户友好的考虑，我们还实现了以下辅助功能：

- 通过鼠标滚轮或右侧缩放按钮实现地图的缩放
- 通过鼠标拖拽实现地图的拖动
- 关键字不明确时的目标位置搜索
- 一键返回当前位置
- 在地图对应的建筑上标注其名称
- 根据对应建筑的不同类型用不同的图案标注

总体而言，我们实现了一个功能较为齐全完善的地图导航措施，也进行了一系列出于用户服务的优化。

二、项目各模块与类的设计细节与小组分工

从总体上我们将项目分成了三个模块，分别是算法设计（包括路径搜索的dijkstra算法以及搜索位置时的模糊处理），地图数据处理（将xml格式的地图osm文件解析为python对象以供后续调用）以及地图渲染（展示地图）。这三个模块同样对应了我们组三位成员的分工。分别是张之钰：地图渲染，范博文：算法设计，方俊：地图数据处理。在各个模块之间我们秉持高内聚低耦合的核心思想，由张之钰负责耦合另外两位同学实现的内容并整合主程序入口。

主程序 `main.py`

整合所有模块，应用程序的执行入口，使用了面向对象编程的单例模式。

- `app` **类**
  - **用途**：整合和统一运行所有模块。
  - **实现**：
    - `__init__` **方法**：初始化窗口，读取地图文件，将槽函数和事件绑定。
    - `changeSearchStatus` **方法**：按下切换按钮时，改变地点搜索或路径搜索的状态。
    - `setdest` **方法**：按下设为起点或设为终点按钮时，将选定地点设置为起点或终点。
    - `initnavigation` **方法**：初始化时调用，启动器实时更新我的位置。
    - `navigateSelf` **方法**：按下定位按钮时，定位到我的位置。
    - `selectSpot` **方法**：通过点击地点名称选中地点的处理函数。
    - `focuspresearch` **方法**：在搜索框获得焦点时执行的预搜索。
    - `presearch` **方法**：输入框内容变化是执行的预搜索。

- `search` **方法**：按下搜索按钮时执行的搜索。
- `searchSpot` **方法**：地点搜索。
- `searchWay` **方法**：路径搜索。
- `searchClickSpot` **方法**：从模糊地点搜索备选列表中选中地点。

常量模块 `constants.py`

设置程序中的所有重要常量，以便跨文件统一修改和引用。

地图渲染模块 `ui.py`

继承并重写Python QT中的许多类，实现美观和个性化的的渲染

- `window` **类**
  - **用途**：主窗口类，继承自`QMainWindow`，用于窗口的绘制
  - **实现**：
    - `__init__` **方法**：基本的窗体初始化内容，设置图标和标题，并为地图页面创建GraphicsView以动态添加建筑元素。
    - `resizeEvent` **方法**：继承自Qt的方法，用于在窗口大小变化时重定位按钮保持相对位置不变。
    - `initcontrol` **方法**：初始化时调用的方法，创建主窗口中所有按钮和输入框。
    - `initmap` **方法**：初始化时调用的方法，创建地图View中的位置标定元素。
    - `setSearchlist` **方法**：设置模糊搜索时的地点备选方案的字符串列表。
    - `centeraArea` **方法**：使指定地点居中。
    - `changeselectbutton` **方法**：在地点搜索和路径搜索的按钮图标中切换。
    - `drawMap` **方法**：按顺序调用初始化方法，使道路和地点的层级关系合理。
    - `drawArea` **方法**：初始化时调用的方法，绘制所有地点，包含建筑、水域、绿地和场地，并绘制名称标签。
    - `drawRoute` **方法**：初始化时调用的方法，绘制所有道路，包含步道、自行车道和各级机动车道。
    - `updateAreas` **方法**：在地图缩放时，更改地点名称标签的可见性，避免互相交错。
    - `pinText` **方法**：在地点上添加选中标记。
    - `startText` **方法**：在地点上添加起点标记。
    - `endText` **方法**：在地点上添加终点标记。
    - `startSelf` **方法**：在我的位置上添加起点标记。
    - `endSelf` **方法**：在我的位置上添加终点标记。
    - `setSelfPos` **方法**：设置我的位置的坐标。
    - `drawway` **方法**：绘制起点到终点之间的最短路径。
    - `clearway` **方法**：清除起点到终点之间的最短路径。
- `mapView` **类**：地图视图类，继承自`QGraphicsView`，用于地图的动态绘制。
- `selfGraphicsItem` **类**：我的位置类，继承自`QGraphicsItem`，用于绘制我的位置。
- `areaItem` **类**：地点类，继承自`QGraphicsPolygonItem`，用于绘制地点。
- `textItem` **类**：标签类，继承自`QGraphicsTextItem`，用于显示地点的名称。

- `searchItem` **类**：搜索候选项类，继承自`QListWidgetItem`，用于显示模糊搜索的备选项。
- `SearchLineEdit` **类**：搜索框类，继承自`QLineEdit`，用于搜索时触发事件。

算法设计模块 `algorithm.py`

- `graph` **类**：
  - **用途**：实现与 `map.py` 的耦合，完成了最优路径的查找等主要功能。
  - **实现**：
    - `__init__` **方法**：读取了`map.py`中的`node` 类和`route` 类，并且依据传入的 `route` 数据创建地图邻接表 `near_map`。
    - `get_distance` **方法**：用于计算两个点之间的欧几里得距离。
    - `find_nearest_node` **方法**：用于当选择的`node`不在`route`上时，找到在路径上的最近点，方便渲染道路。
    - `_dijkstra` **方法**：使用 `dijkstra`算法找到从起始点到目标点的最近道路，返回一系列route上对应节点的唯一标识符。
- **模糊搜索函数**`searchstr`： 对于传入的`lst`，以及目标字符串，遍历`lst`中的每个`name`，计算它与目标`name`的`Levenshtein距离`，如果达到阈值就存入`ans`中。
  - **相似度阈值** `similar_rate`：作为一个临界的判断，如果`Levenshtein距离`大于`similar_rate`就认为是模糊搜索的一个潜在目标。
  - **ans列表的最大长度** `list_num`：为了避免列表过长，设定的输出列表的最大长度。如果有超过`list_num`个满足要求，则按照`similar_rate`从高向低返回。

地图数据处理模块 `map.py`

我们定义了多个类，用于解析OpenStreetMap（OSM）的XML文件，并将其转换为地图数据结构，同时提供了一系列方法来处理和查询这些地图数据，包括路径搜索、区域搜索、位置查找等功能。

- `node` **类**：
  - **用途**：表示地图上的一个位置节点，包含节点的唯一标识符、经纬度信息。
  - **实现**：通过 `__init__` 方法初始化节点的属性，并提供 `getpos` 方法获取节点的坐标。
- `route` **类**：
  - **用途**：表示地图上的道路路径，包含路径的唯一标识符、节点列表、道路类型和标签信息。
  - **实现**：通过 `__init__` 方法初始化路径的属性，并提供 `getpos`、`gettype` 和 `gettag` 方法分别获取路径的坐标列表、道路类型和标签信息。
- `area` **类**：
  - **用途**：表示地图上的闭合区域，如建筑、绿地、水域等，包含区域的唯一标识符、节点列表、区域类型和标签信息。
  - **实现**：通过 `__init__` 方法初始化区域的属性，并提供 `getpos`、`gettype` 和 `gettag` 方法分别获取区域的坐标列表、区域类型和标签信息。
- `nav` **类（主要部分）**：

    - **用途**：整合地图信息，实现与 `algorithm.py` 的耦合，提供地图数据的读取、解析和查询功能。
    - **实现**：
        - `__init__` **方法**：读取指定路径的OSM文件，解析XML数据，初始化地图的节点列表、路径列表和区域列表。
        - `getroute` **方法**：根据指定的条件筛选道路路径，并返回满足条件的路径列表。
        - `getarea` **方法**：根据指定的条件筛选区域路径，并返回满足条件的区域列表。
        - `findpos` **方法**：使用 `algorithm` 模块中的 `graph` 类查找距离指定位置最近的路径上的节点。
        - `searchspot` **方法**：使用 `algorithm` 模块中的 `searchstr` 函数模糊搜索地点，并返回包含 `area` 和 `route` 的列表。
        - `calcroute` **方法**：使用 `algorithm` 模块中的 `graph` 类进行路径搜索，返回 `route` 的列表。

- **XML解析**

  在 `nav` 类的 `__init__` 方法中，使用 `xml.etree.ElementTree` 模块解析OSM文件的XML数据。遍历XML树的每个元素，根据元素的标签类型（`node`、`way` 或 `relation`）进行不同的处理：

    - `node` **标签**：提取节点的ID、经纬度信息，并创建 `node` 对象。如果节点带有 `name` 属性，则将其转换为 `area` 对象。
    - `way` **标签**：提取路径的ID、节点列表和标签信息，根据标签信息判断路径是道路还是区域，并创建相应的 `route` 或 `area` 对象。
    - `relation` **标签**：处理关系标签，提取关系的ID、标签信息和成员路径，根据成员路径的角色创建相应的 `area` 对象。

三、项目总结与反思

对于本次合作的项目，一方面我们认为项目的完成度是比较高的，基本上充分实现了一个导航系统应该有的内容。另一方面我们也同样意识到，对于我们的项目而言，完成基本的内容之外我们还有很多可以进一步完善的地方。我们小组认为后续还可以添加的功能包括但不限于：为路径添加出行方式（如步行，自行车，开车等方式），再根据出行方式的不同返回不同的道路推荐、根据实时使用人数的多少来预测不同路段的拥堵情况等等。同时参考到路演中与我们类似项目内容的小组，我们还可以出于用户考虑添加一些地图导航之外的元素。如学校不同食堂的开放时间，以及一个公开匿名的校园论坛，实现功能的复合化，更能满足用户的多维度需求。