北京邮电大学课程设计报告

<table>
<tr><td>课程设计名称</td><td colspan="2">数据结构</td><td>学院</td><td>计算机</td><td>指导教师</td><td>张海旸</td></tr>
<tr><td>班级</td><td>班内序号</td><td colspan="2">学号</td><td>学生姓名</td><td colspan="2">成绩</td></tr>
<tr><td>2018211305</td><td>21</td><td colspan="2">2018211281</td><td>李林</td><td colspan="2"></td></tr>
<tr><td>2018211306</td><td></td><td colspan="2">2018211303</td><td>张海东</td><td colspan="2"></td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td><td colspan="2"></td></tr>
<tr><td>课程设计内容</td><td colspan="6">一所大学的校园中有众多的建筑物和道路，该大学还有多个（至少 2 个）校区分布在一个城市的不同地方。校园导览系统具备导航功能，当用户于某一时刻请求导航时，校园导览系统将根据用户的具体要求为其设计一条线路并输出；校园导览系统能提供查询服务，内容包括当前时刻用户所处的地点，周围的教学楼、宿舍楼、餐饮、后勤服务、操场等信息。</td></tr>
<tr><td>学生课程设计报告 (附页)</td><td colspan="6"></td></tr>
<tr><td>课程设计成绩评定</td><td colspan="6">遵照实践教学大纲并根据以下四方面综合评定成绩:
1、课程设计目的任务明确，选题符合教学要求，份量及难易程度
2、团队分工是否恰当与合理
3、综合运用所学知识，提高分析问题、解决问题及实践动手能力的效果
4、是否认真、独立完成属于自己的课程设计内容，课程设计报告是否思路清晰、文字通顺、书写规范
评语:

成绩:

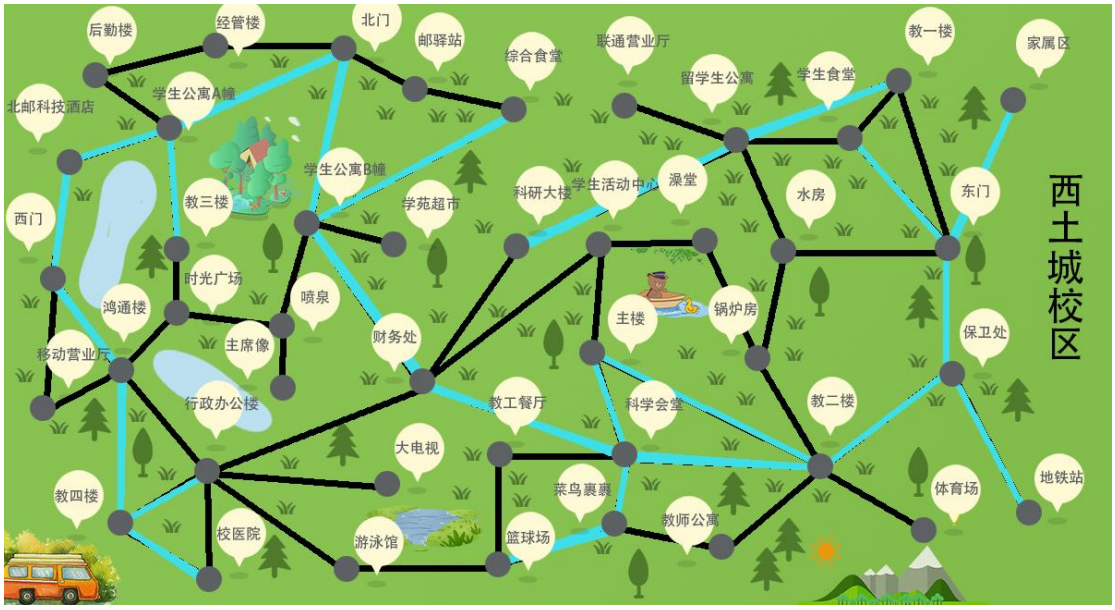
指导教师签名:
年 月 日</td></tr>
</table>

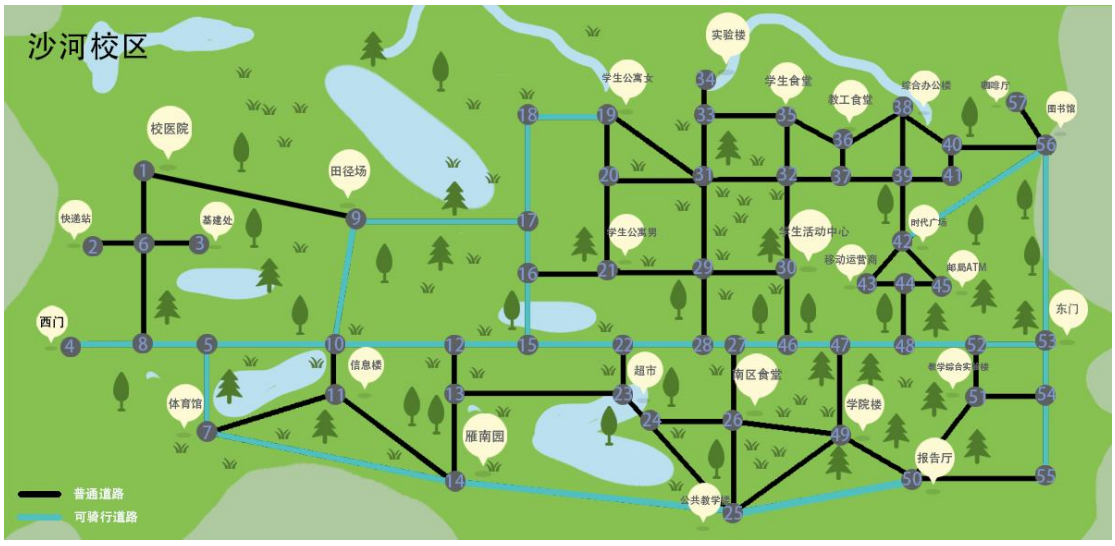
注：评语要体现每个学生的工作情况，可以加页。

校园导览系统目录

一、问题描述

一所大学的校园中有众多的建筑物和道路，该大学还有多个（至少 2 个）校区分布在一个城市的不同地方。校园导览系统具备导航功能，当用户于某一时刻请求导航时，校园导览系统将根据用户的具体要求为其设计一条线路并输出；校园导览系统能提供查询服务，内容包括当前时刻用户所处的地点，周围的教学楼、宿舍楼、餐饮、后勤服务、操场等信息。





二、系统功能需求

1.系统建立校园内部道路图，包括各种建筑物、服务设施等。要求校园内建筑物不少于20个，其他服务设施不少于5种，共20个。

2.系统提供用户注册登录功能，以日志的形式记录每个用户的键入命令和操作信息，日志使用数据库保存。

3.系统能够分析用户键入命令，并做一定的调度；

4.系统应提供四种导航行进策略，前3种策略默认在校区内步行，第4种可在校区内选择交通工具：

a) 最短距离策略：给出两点间距离最短的路线；

b) 最短时间策略：给出两点间时间最短的路线；

c) 最短距离策略（途径某地）：途径某些地点的最短距离；

d) 最短时间策略（校内可选交通工具）：校区内可选自行车，且自行车在校区内任何地点都有。

5.系统提供模拟用户移动的导航功能，即在用户选定路线后系统按照所制定的路线，随着时间以设定的行进速度前进。

6.系统能模拟时间流逝，当用户在不同状态下系统流逝时间不同，以系统时间与真实时间的比值为分三类标准：系统待机（系统时间/真实时间=1:1)、用户校区内行进（系统时间/真实时间=10:1)、用户校区间行进（系统时间/真实时间=300:1)。

7.系统提供查询功能，用户可以实时查询自身所处的位置，系统给出该位置周边x米内的建筑物，以及到该建筑物的最短路线距离；查询时若选定某一目标，系统会将其设为新的目的地以最短距离策略制定路线。

8.系统提供行进途中暂停、临时修改目的地、临时修改行进策略的操作。

9.系统在教学楼和公寓提供精准到具体楼层和房间号的导航。

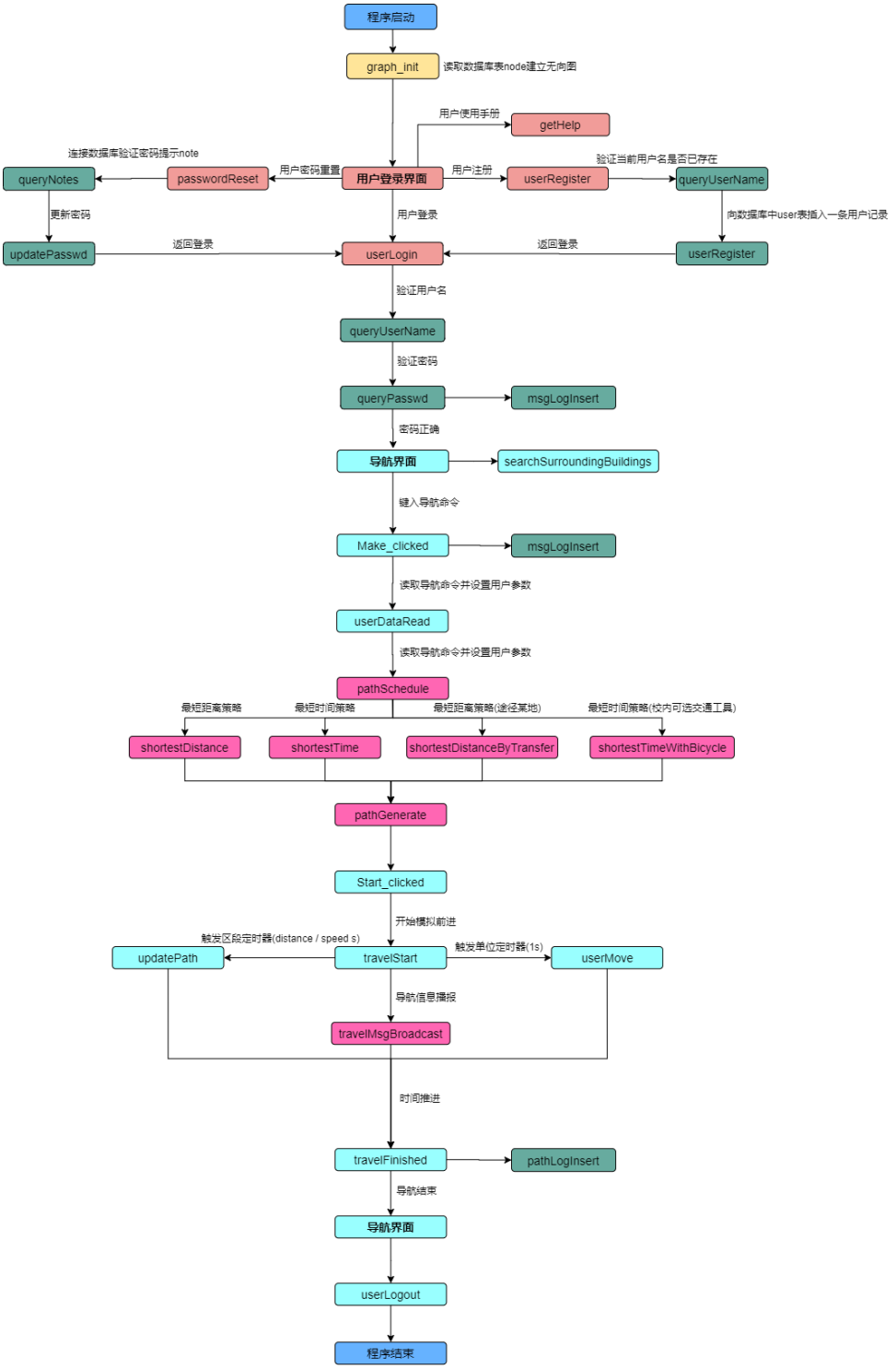
10.系统能够模拟人流变化，即每x分钟根据随机数对建筑物内的人数做模拟变化，以颜色表现各建筑物内的人数多少：绿色（0~99)、蓝色（100~199)、红色（200+)。

11.系统提供语音实时播放导航信息的功能。

三、总体设计方案说明

1.开发环境：Windows 10 操作系统，Qt Creator 4.8。

2.总体结构流程图：



3.成员工作分工

成员	主要复杂的工作
李林	①数据库管理； ②用户模拟移动函数的相关设计； ③扩展功能函数的设计； ④扩展 floyd 算法的设计； ⑤日志模块设计。
张海东	①静态数据设计，如地图节点的坐标、地图绘制； ②基础 floyd 算法的设计与测试。

4.模块划分

模块名	模块功能
导航主界面 campusGuide	①命令分析功能的实现； ②模拟时间推进； ③根据制定的路线，模拟用户移动； ④图形化地图的展示； ⑤其他必要的参数设置、数据读取、状态变化函数。
功能算法实现 guide	①路线规划函数的实现； ②查询周边建筑物等扩展功能的实现。
数据库管理 database	①连接数据库读取用户名、密码； ②连接数据库写入键入命令日志 msgLog 与路径日志 pathLog。
登录注册界面 manage	①用户登录、注册、重置密码； ②系统使用帮助手册、功能说明。
公有设计 public	①声明结构、全局变量与宏； ②读取数据库中存储的静态数据建立无向图及邻接矩阵； ③提供一些常用的函数，如计算两点间距离等函数的声明与实现。

四、数据结构说明和数据字典

1.结构体定义

①地图的节点结构定义 node

```
typedef struct node{
    unsigned seq;   // 结点编号
    QString name;  // 结点名称
    QPoint COI;      // 中心点坐标
    QString campus;    // 结点所属校区
    unsigned symbol;  // 结点标记，如食堂、教学楼、公寓等
    unsigned people;  //  结点人数
}Node, *Np;
```

解释说明：读取数据库表 node，将各项数据存入序号对应节点的成员变量中，主要用于根据序号迅速查询到对应的建筑物节点。

②路径中区段结构定义 section

```
typedef struct section{
    unsigned from, to;
    bool bicycle;              // 用户在当前区段骑自行车
    double curSpeed;         // 用户在区段[from,to]上的行进速度
    unsigned sectionCongestion;    // 当前区段的拥挤度
    unsigned remainingDistance;   // 距离下一节点的剩余距离
    unsigned sectionTime; //  到达下一节点的总时间
}Route, *Rp;
```

解释说明：以区段形式存储路径，如 x1->x2->x3 被存储为 x1->x2, x2->x3 两段点到点的区段路径；同时还存有当前区段是否可使用交通工具、行进速度、时间与距离等参数。

③用户结构定义 user

```
typedef struct user{
    QString userName, userState;   // 用户名、用户状态
    QString strategy, campus;     // 行进策略、用户当前所处的校区
    QString start_time, end_time; // 记录一次导航的开始与结束时间，主要用于日志记录
    Node src, dst, transPoint;     // 起点、终点、中转点
    Node srcAccurate, dstAccurate, transAccurate; // 可能存在的精确制导点
    unsigned timecost, walked;  // 走过的总时间和距离
    QPoint location;       // 用户的当前位置

    QList<Rp>   Path;  // 总的行进路线
    Rp curNode;         // 行进路线中的当前节点
}User, *Up;
```

解释说明：存储用户设定的出发地、目的地、行进策略等参数，以及一些系统使用的用于模拟导航的参数。

④班车记录结构 busRecord

```
typedef struct busRecord{
    QString date;  // 日期
    QString origin, destination; // 出发地、目的地
    QString start_time, arrive_time; // 开始时间、结束时间
    QString type;  // 交通方式，"校车"、"班车"
}bR, *bRP;
```

解释说明：读取数据库表 bus_timetable 建立的校车交通时刻表，需要时遍历容器查找等待时间最短的某一列校车。

2.全局变量及宏定义

```
#define  MAX_NUM   256  // 最大节点数
#define  INF   0x3f3f3f3f  // 不可达距离
#define  coordinateCorrect QPoint(20,25)  // 使用坐标导航时排除用户标签本身的大小
#define  Search_Ridus   500 // 查询功能的查找半径为中心点向外500m
#define  TIME_FLAG_TRUE      1     // 等待状态，系统时间1s=真实时间1s
#define  TIME_FLAG_CAMPUS  10   // 校区内移动，系统时间1s=真实时间10s
#define  TIME_FLAG_WAITBUS 60  // 等车，系统时间1s=真实时间1min
#define  TIME_FLAG_CITY      300 // 校区间移动，系统时间1s=真实时间5min
```

```
extern int walkSpeed, bicycleSpeed, busSpeed;     // 步行、骑行、校车的理想速度
extern int Graph[MAX_NUM][MAX_NUM];               // 连通矩阵，Graph[i][j]的值为两节点的连通距离
extern double Congestion[MAX_NUM][MAX_NUM]; // 拥挤度矩阵，Congestion[i][j]的值为两节点间的拥挤度
extern bool Bicycle[MAX_NUM][MAX_NUM];  // 邻接矩阵，表示两节点间是否可以骑自行车
extern QMap<unsigned, Np> graph_nodes;  // 地图节点映射，将节点序号与节点结构相关联
extern QVector<bRP> busTable;         // 交通时刻表
extern User uuser;          // 全局用户
extern QDateTime datetime; // 系统时间
extern QMap<QString, QString> mappingRelation; // 自定义的逻辑地址向物理地址的映射
```

解释说明：

①无向图距离矩阵 Graph，是读取数据库表 node 建立的点与点之间的关系矩阵，其值两点间距离，不为 INF 则表示两点连通。

②无向图邻接矩阵 Congestion，其值为两点间的拥挤度；无向图邻接矩阵 Bicycle，其值为两点间是否可骑自行车。

③时间推进标志 timeFlag，有 3 类标准：系统待机状态(1:1)、校区内移动状态(10:1)、校区间移动(300:1)。与定时器 sysTimer 相关联，通过超时信号 timeout()触发属于系统的时间更新函数，datetime 用于存储系统时间。

④mappingRelation 为用户自定义的逻辑地址向物理地址的映射表，一个逻辑地址对应一个唯一的物理地址。

3.导航主界面中的私有变量

变量声明	解释说明
QLabel *permanent;	置于主界面右下状态栏中显示时间的标签
QTimer *sysTimer;	用于更新系统时间的定时器
QTextToSpeech *tts;	语音播报对象
bool ttsFlag=false;	标志当前是否启用了语音播报功能
bool accurateFlag=false;	标志当前是否启用了精确导航功能
QString msgCache;	导航信息缓存
bool movePause=false, pauseFlag;	前者记录当前是否处于暂停状态，后者记录上一区段路程是否暂停过
Dialog *load;	校区转换时的中转加载框
QList<QString> commonPath;	存储路径相关的偏好设置信息
QStringListModel pathModel;	将路径偏好用视图展示出来
QStandardItemModel *_matchModel, *_searchModel;	前者用于将编辑时按关键字匹配的搜索结果展示出来，后者用于展示周边建筑物查询结果。
QString whichMap;	记录当前使用的导航平台图的名字
QTimer *userTimer;	用于模拟移动过程中记录用户移动的区段计时器
QTimer *moveTimer;	用于控制用户移动的单位计时器
QTimer *randTimer;	用于更新建筑物内人数的随机数计时器

五、各模块详细设计说明

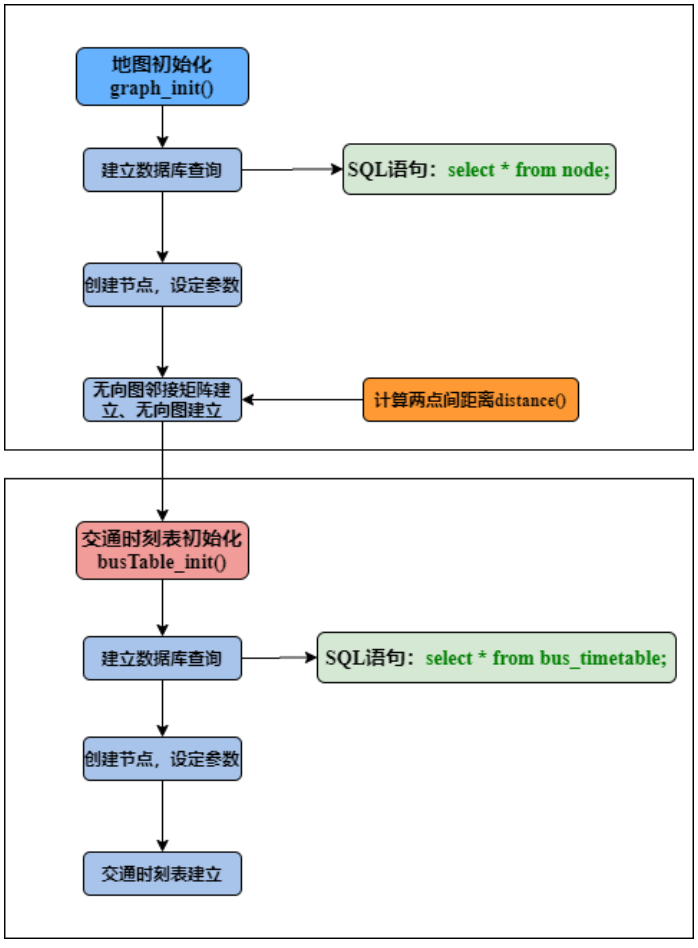
系统基本上是串行的，故以初始用户的角度出发描述该系统各模块的功能。

1.公有设计模块 public

①系统启动，开始初始化。首先连接数据库，从数据库表 node 中读取地图节点数据建立无向图，并设置各邻接矩阵。

②同样地，连接数据库从库表 bus_timetable 中读取数据建立交通时刻表。

函数原型	函数功能
void graph_init()	初始化无向图
int distance(unsigned A, unsigned B)	返回[A, B]间的距离
int distance(QPoint &A, QPoint &B)	重载：直接计算两坐标间距离
void busTable_init()	初始化交通时刻表
void mappingRelation_init()	读取逻辑-物理的地址映射关系



2.数据库管理模块 database

该模块只定义了一个 database 类及类成员函数，没有实际的动作。因此仅给出其中定义的函数接口，具体操作等后续引用时描述。

函数原型	函数功能
static database *getDatabase()	若数据库实例已存在则返回，不存在则创建一个实例再返回。
bool userRegister(const QString &userName, const QString &passwd, const QString &notes)	输入用户名、密码、提示来尝试注册一个用户。注册成功则向数据库表 user 中插入一条记录同时返回 true，失败则返回 false 与消息框。
bool queryUserName(const QString &userName)	登录或注册时用于检测数据库表 user 中是否存在该用户，存在 true，否则 false。
bool queryPasswd(const QString &userName, const QString &passwd)	登录时验证密码是否正确，是则 true，否则 false。
bool queryNotes(const QString &userName, const QString &notes)	重置密码时需要验证该用户名对应的关键字是否正确，是则 true，否则 false。
void updatePasswd(const QString &userName, const QString &passwd, const QString &notes)	重置密码时更新数据库表 user 中的用户密码
static bool msgLogInsert(const QString logTime, const QString userName, const QString userState, const QString logMsg)	用户操作时，向数据库表 msg_log 中插入一条用户操作记录(logTime, username, userState, logMsg)，插入成功 true，失败 false。
static bool pathLogInsert(const QString userName, const QString timeStart, const QString from, const QString trans, const QString to, const QString strategy, const QString pathMsg, const QString timecost)	导航完成时，向数据库表 path_log 中插入一条路径日志记录(userName, timeStart, from, trans, to, strategy, pathMsg, timecost)，插入成功 true，失败 false。
static void getPreferredPath(const QString userName, QList<QString> &commonPath)	从数据库表 user 中获取该用户对应的常用路径，用作偏好设置。
static void updateCommonPath(const QString userName)	导航完成时，更新一次常用路径。
static void updateLastPath(const QString userName, QString src, QString trans, QString dst, QString strategy)	导航完成时，更新一次最近使用路径。

3.用户登录界面 manage

①首先注册用户，需要输入用户名、密码以及用于重置密码的提示，系统会对用户名进行查重，若当前用户名未被注册，则向数据库表 user 中插入一条用户记录。

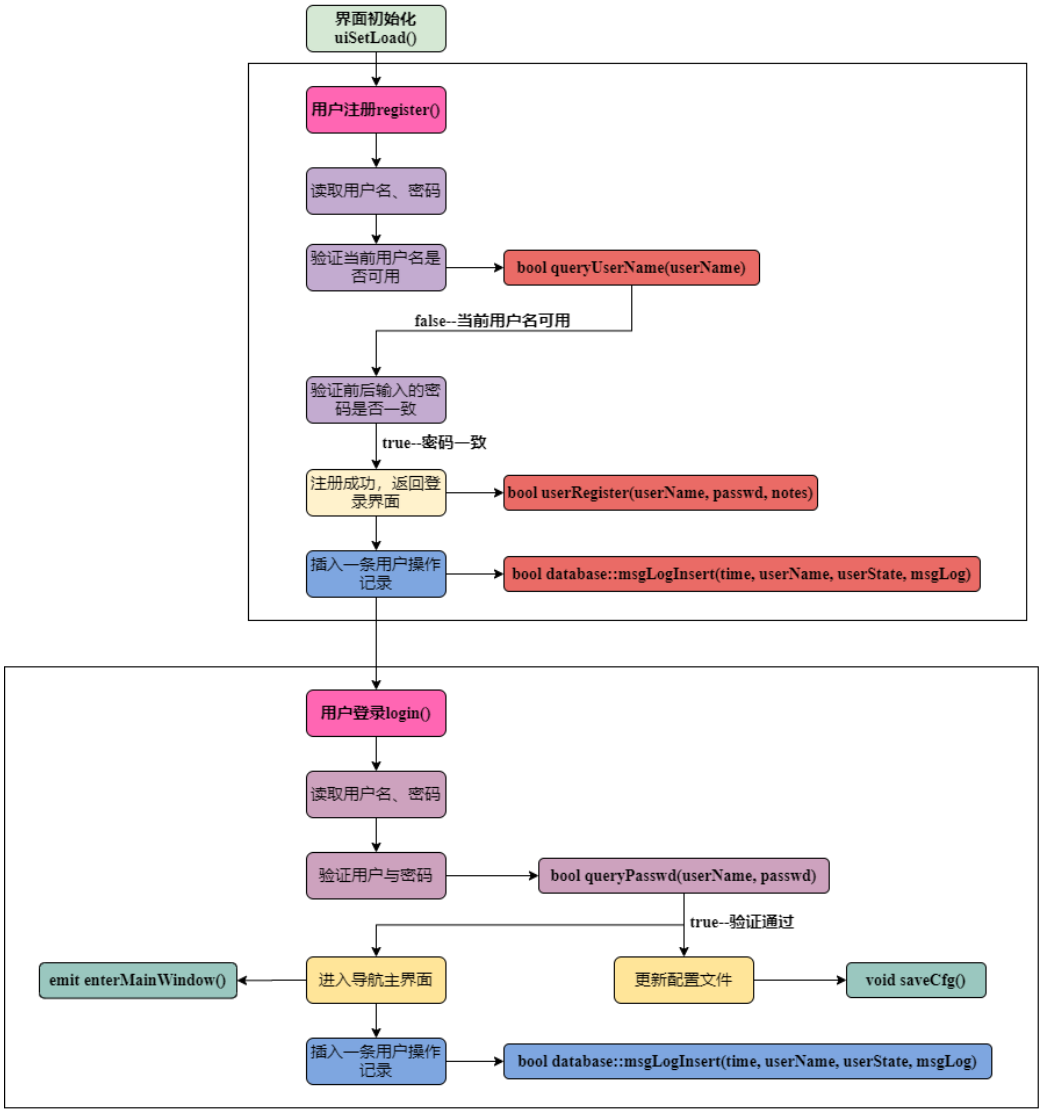
②登录时输入用户名、密码（可以选择记住密码和自动登录），系统验证用户名及密码通过后弹出导航主界面，用户名与密码不匹配时弹出错误对话框。

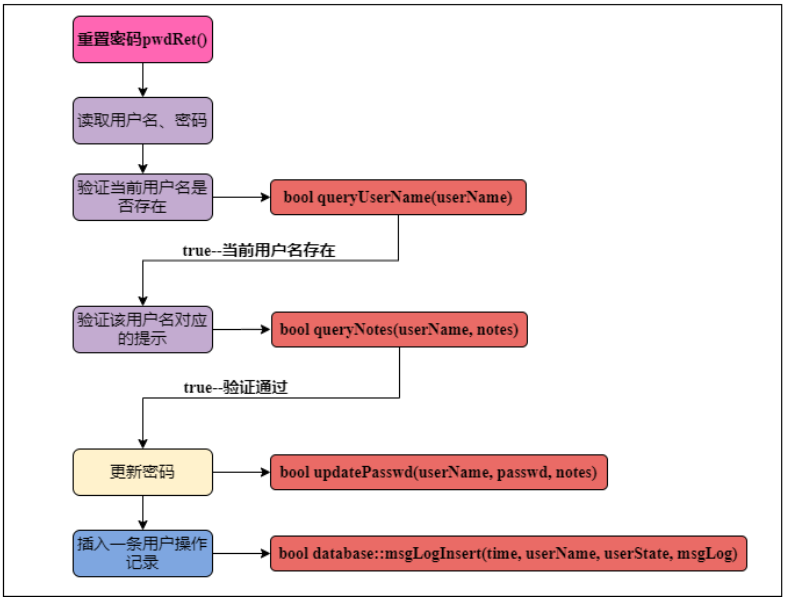
③忘记密码时可重置密码，只需要输入要重置的用户以及对应的提示就可以重置密码。

④系统提供用户使用手册帮助使用。

函数原型	函数功能
void uiSetLoad()	设定界面参数
void on_RP_Register_clicked()	【按钮槽】注册用户时，验证用户名、密码，通过则向数据库表 user 中插入一条记录同时返回登录界面，失败则清空输入并弹出失败对话框。
void enterMainWindow()	【信号】登录验证通过进入主窗口
void on_login_clicked()	【按钮槽】验证输入的用户名、密码是否存在、是否与数据库中保存的一致，是则发出 enterMainWindow()信号，否则弹出错误提示框并清空编辑栏。
void on_FP_reset_clicked()	【按钮槽】重置密码时验证用户名与提示，通过则将新密码更新为密码。
void on_FP_back_clicked();	【按钮槽】返回登录界面。
void on_help_clicked()	【按钮槽】弹出帮助手册。
void switchPage()	【按钮槽】帮助手册的翻页函数。
QWizardPage *createPage1()	帮助手册第一页
QWizardPage *createPage2()	帮助手册第二页
QWizardPage *createPage3()	帮助手册第三页
void on_autoLogin_stateChanged(int arg1)	【按钮槽】当自动登录的选框状态改变时，修改配置文件对应参数，同时设置记住密码框的选中状态；选中自动登录时默认选中记住密码，当取消记住密码时自动登录也随之取消。
void on_remPasswd_stateChanged(int arg1)	【按钮槽】当记住密码的选框状态改变时，修改配置文件对应参数。
void saveCfg()	将记住密码、自动登录的状态变化写回文件
void loadCfg()	读取配置文件用以设置记住密码、自动登录参数
void autoLogin()	【信号】自动登录参数有效，发出信号触发登录按钮

1.所有的按钮槽函数名以功能简写，如on_pushButton_login_clicked() --> login();
2.数据库操作的bool返回值如无特殊说明，都是SQL语句的执行结果，true执行成功，false反之。

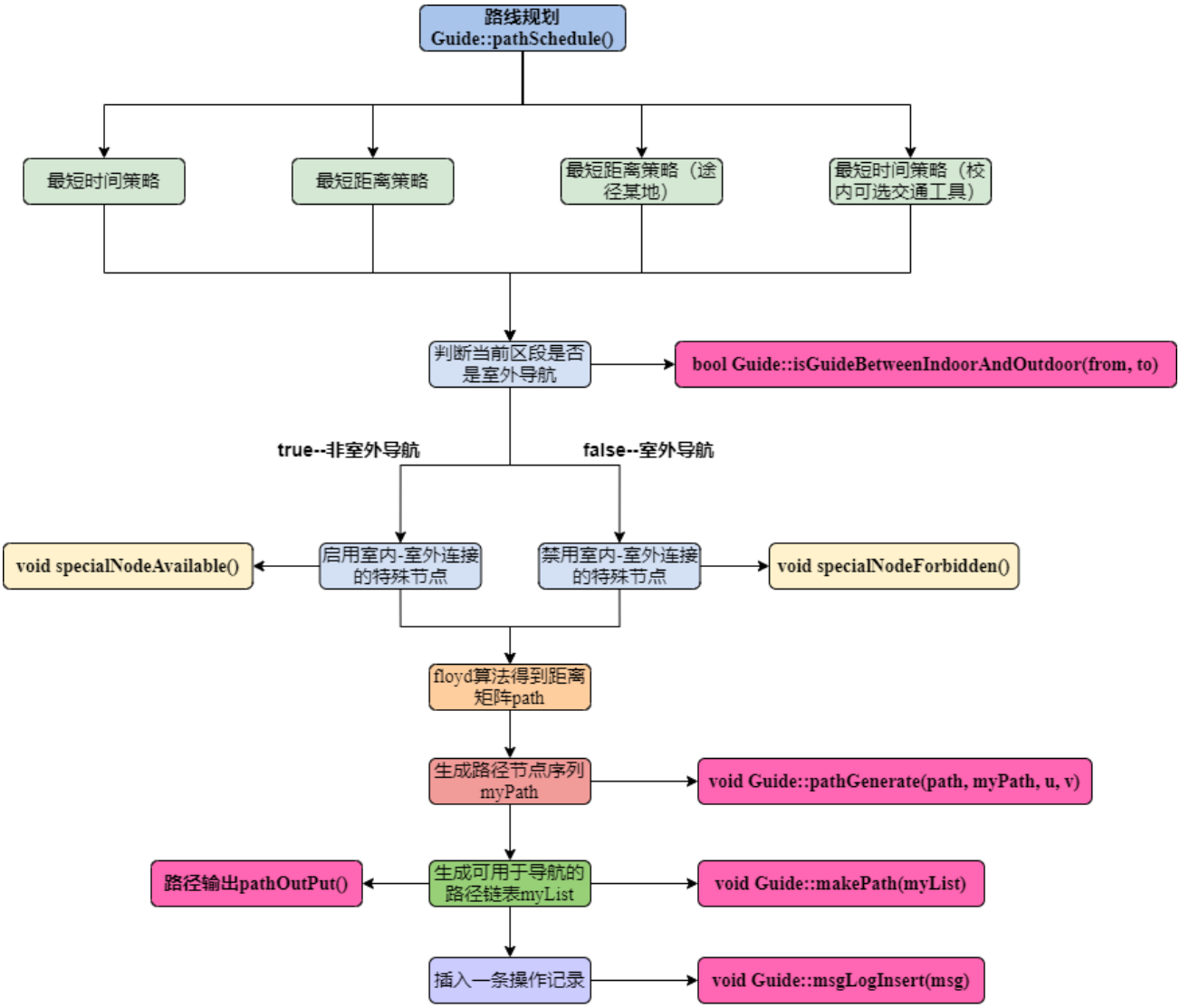




4.功能算法设计 guide

该模块中大部分函数都是用来实现路线规划、模拟导航的，因此以一个路线导航实例[from, to]来说明一次导航路线的规划是如何完成的：①首先，系统读取用户在行编辑器内设定的导航参数，选择调用不同的路径规划算法； ②其次，路径规划算法计算得到 path 矩阵，并根据选定的起点和终点来生成路径节点序列； ③最后，根据得到的路径节点序列，生成可用于系统模拟导航的路径链表。

需要特别说明的两点：(1)在室外导航时不能使用室内的节点，因此需要禁用；同理在室内导航时禁用室外节点。(2)所有策略的路线规划算法其核心都是 floyd 算法，其正确性已于概要设计中证明，文末再附证明。



该类主要用于实现一些功能供导航主界面调用，因此定义了大量的静态函数作类外调用，这里给出该类中完整的函数接口。

函数原型	函数功能说明
static void pathSchedule()	根据用户参数进行路径规划

static bool **isEnterIndoor**(int seq)	判断当前精确导航节点是否有效，有效 true，无效 false
static void **travelMsgConcat**(QString &msg)	生成导航信息实时播报并返回
static QString **timeTransfer**(unsigned time)	将毫秒时间转化为(hh:mm:ss)形式的时间
static void **shortestDistance**(QList<unsigned> &myPath, unsigned from, unsigned to)	【路线规划】最短距离策略的路径规划算法，生成[from, to]的路径节点序列并放入链表中。
static void **pathGenerate**(int path[][MAX_NUM], QList<unsigned> &myPath, unsigned u, unsigned v)	【路线生成】根据路线规划算法得到的 path 矩阵生成[u, v]的路径节点序列
static void **makePath**(QList<unsigned> &myList)	【路线生成】根据 myList 提供的节点序列制定可用于导航的路径链表
static QString **pathOutput**()	【路线输出】输出用户当前的路线
static void **shortestTime**(QList<unsigned> &myPath, unsigned from, unsigned to)	【路线规划】最短时间策略的路径规划算法，生成[from, to]的路径节点序列并放入链表中。
static void **shortestDistanceByTransfer**(QList<unsigned> &myPath, unsigned from, unsigned to)	【路线规划】途径某地的最短距离策略的路径规划算法，生成[from, to]的路径节点序列并放入链表中。
static void **shortestTimeWithBicycle**(QList<unsigned> &myPath, unsigned from, unsigned to)	【路线规划】可选交通工具的最短时间策略的路径规划算法，生成[from, to]的路径节点序列并放入链表中。
static void **searchSurroundingBuildings**(QStandardItemModel *_model, QString name, unsigned ridus)	【查询】查询周边 ridus 米内的建筑物信息，以链表_strList 返回。
static bool **isTransferArrived**()	判断当前时刻是否已经通过中转点，主要用于暂停与恢复。
static bool **isCampusMoving**()	判断当前区段是否是在校区间移动，true 是，false 否。
static bool **isGuideBetweenIndoorAndOutdoor**(unsigned from, unsigned to)	判断[from, to]是否是在室外导航，true 是，false 否。
static void **addListItem**(QStandardItemModel* _model, const QString _str, const unsigned people)	向_model 中加入一个内容为_str 的子项目，其颜色根据 people 改变。
static int **findSuitableBus**(QTime start_time, QDate date , unsigned from, unsigned to, unsigned type, int &waitTime)	在交通时刻表中查找距离当前时间最短的校车/班车，返回该车次在容器中的序号以及等待时间。
static void **getSitePara**(QString str, QString &campus, QString &siteName, unsigned &roomId)	读取偏好设置路径中的路径并将其设置为当前的导航参数
static void **msgLogInsert**(QString msg)	【日志】向数据库表 msg_log 中插入一条操作记录日志
static void **pathLogInsert**(QString msg)	【日志】向数据库表 path_log 中插入一条路径记录日志

在路径规划算法中调用了四个定义在 public 中的函数：

函数原型	函数功能说明
void **specialNodeAvaliable**()	启用室内-室外特殊连接节点
void **specialNodeForbidden**()	禁用室内-室外特殊连接节点
unsigned **getIndexOfGraph**(QString nodeName)	根据节点名查找其对应的地图节点序号并返回
unsigned **getIndexOfGraph**(QString nodeName, QString campus)	根据节点名与校区查找其对应的地图节点序号并返回

5.导航主界面 campusGuide

该模块主要用于实现分析命令、模拟用户移动，都是系统的核心部分，由于主观设计部分较多，下面以代码+注释的形式逐行解释。

(1) 命令分析

①信号关联

```
    // 当编辑栏中文本改变时自动调用命令分析槽，检索出图中包含该关键字的节点
    connect(ui->lineEdit_src, SIGNAL(textChanged(QString)), this, SLOT(commandParse()));
    connect(ui->lineEdit_dst, SIGNAL(textChanged(QString)), this, SLOT(commandParse()));
    connect(ui->lineEdit_trans, SIGNAL(textChanged(QString)), this, SLOT(commandParse()));
```

②命令分析槽的实现

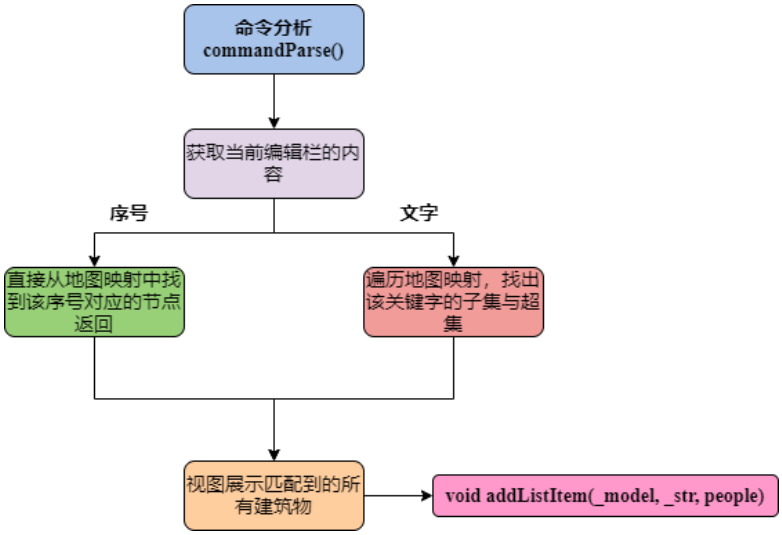
视图展示当前命令关键字的子集与超集。如输入公寓，会得到诸如“学生公寓”、“教师公寓”等结果。

```
// 命令分析，返回所有包含该关键字的建筑物
void campusGuide::commandParse()
{
    QLineEdit *lineEdit = qobject_cast<QLineEdit*>(sender());//得到按下的按钮的指针
    QString command = lineEdit->text();

    bool seqFlag;
    unsigned seq = command.toInt(&seqFlag);
    _matchModel = new QStandardItemModel(this);

    if(seqFlag)   // 当前输入为节点编号则直接在容器中查询该节点后返回
    {
        if(seq<=0 || seq>100)
        {
            lineEdit->clear();
            QMessageBox::information(nullptr, "输入错误", "超出节点编号范围，普通节点(1~100)", QMessageBox::Ok);
            return;
        }
        QString str = QString("[%1] %2").arg(graph_nodes[seq]->campus).arg(graph_nodes[seq]->name);
        addListItem(_matchModel, str, graph_nodes[seq]->people);
    }
    else
    {
        for(auto it=graph_nodes.begin(); it!=graph_nodes.end(); it++)
        {
            if(command.contains((*it)->name, Qt::CaseSensitive)     // 查询关键字command的子集与超集
                    || (*it)->name.contains(command, Qt::CaseSensitive))
            {
                QString str = QString("[%1] %2").arg((*it)->campus).arg((*it)->name);
                addListItem(_matchModel, str, (*it)->people);  // 将当前建筑物加入匹配得到的视图
            }
        }// end of for
    }// end of else

    ui->siteView->setModel(_matchModel);
    ui->textBrowser_msg->hide();
    ui->siteView->show();
```



命令分析演示 1—公寓：

命令分析演示 2—9：

(2) 模拟移动函数的实现

模拟用户移动主要通过定时器实现，每触发一次单位定时器，就根据流逝的时间乘以速度将用户作移动。而每触发一次区段定时器，就代表当前区段已经完成，切换到下一个区段继续移动，直到终点。

其中最主要的难点不在于定时器的设置，而是如何让程序在定时器计时器间等待同时保持主窗口响应，最终采用 Qt 的局部事件循环 QEventLoop 让程序在该时间内局部休眠，成功模拟了用户移动过程。

简单流程如下：

①开始移动前的参数设置：设定时间推移标志为校区内移动、用户状态、出发时间、当前位置、移动用户到起始位置、设定当前显示的地图。

```
timeFlag = TIME_FLAG_CAMPUS;    // 校区内行进，时间加速推进
uuser.userState = "行进中";
uuser.walked = 0;
uuser.campus = graph_nodes[uuser.Path.first()->from]->campus;
uuser.start_time = QDateTime::currentDateTime().toString(tr("yyyy-MM-dd hh:mm:ss"));
ui->userLabel->move(graph_nodes[uuser.Path.first()->from]->COI);
uuser.location = ui->userLabel->pos();
whichMap = uuser.campus;
ui->mapChange->click();
```

②开始模拟移动，启动单位定时器 moveTimer 和区段定时器 userTimer：前者每(1000/timeFlag)触发一次超时信号，更新一次用户当前的位置；后者以区段时间(1000/timeFlag * (*it)->sectionTime + 4*timeFlag)作单次计时，当该定时器超时，即代表该区段已经完成。
（加了 4*timeFlag 的修正时间，因为循环中其他语句有一定执行时间，会导致原定的计时时间被占用一部分，因此需要额外加时间作修正。）

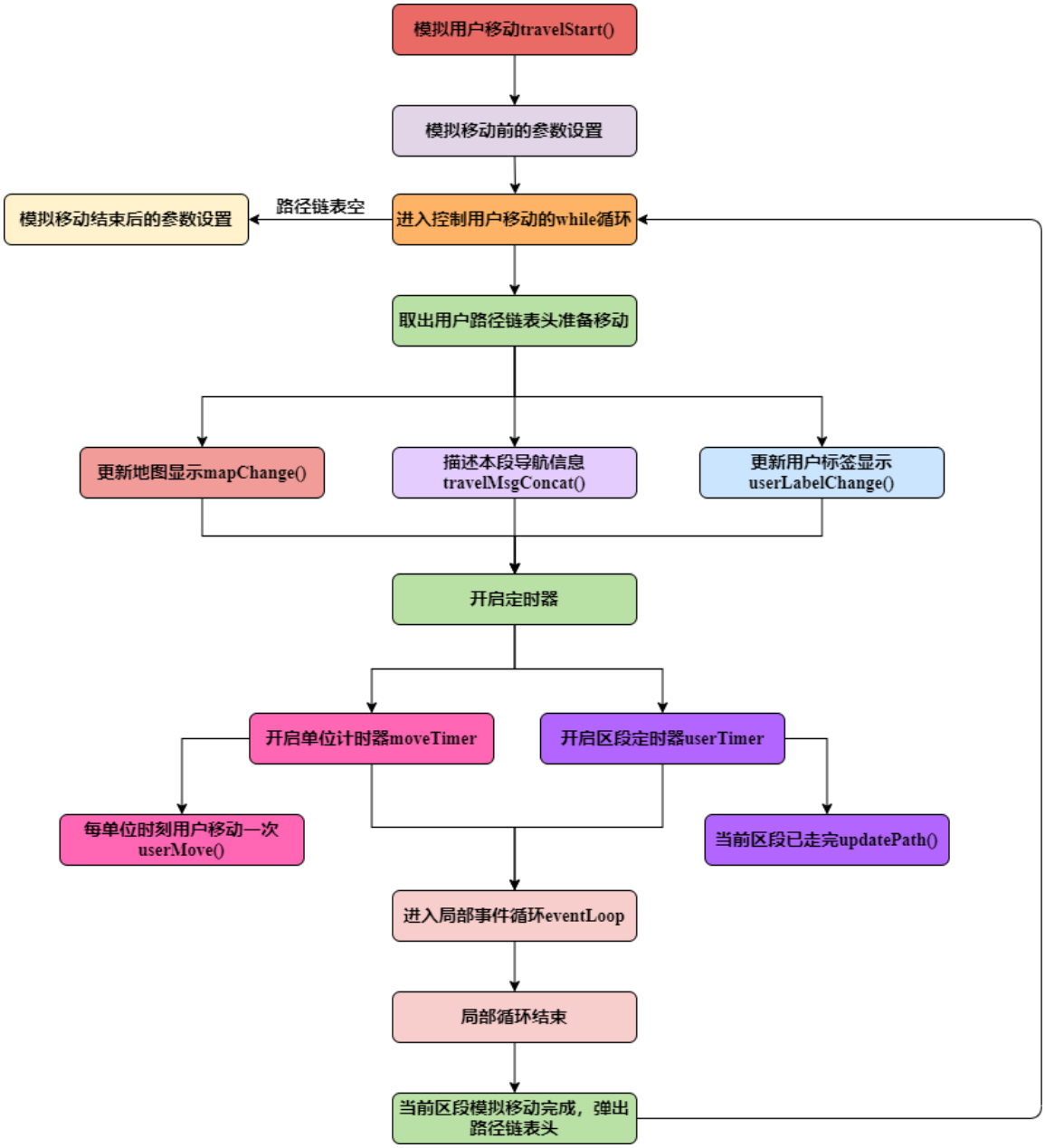
```
moveTimer->start(1000);
connect(userTimer, SIGNAL(timeout()), this, SLOT(updatePath()));
userTimer->start(1000/timeFlag * (*it)->sectionTime + 4*timeFlag); // 系统时间为10倍真实时间推进
// 额外加了4倍timeFlag的修正时间，用于校正usetTimer与sysTimer间的开启计时误差(主要针对校区间移动，校区内几乎可忽略)
```

③在循环体 while 中，先启动上述定时器，然后进入局部事件循环让主线程等待，但仍保持响应，以免循环体导致定时器无法正常运行。

```
// do nothing but wait
QEventLoop eventLoop;    // 使用局部事件循环让主线程等待，但仍保持响应。
QTimer::singleShot(1000/timeFlag * (*it)->sectionTime + 4*timeFlag, &eventLoop, SLOT(quit()));
eventLoop.exec();
```

④区段移动完成时，将该区段的剩余距离清零，同时弹出链表头为下次循环做准备。

程序执行顺序图如下：



该模块完整的接口如下：

函数原型	函数功能说明
void uiSetLoad()	加载界面的初始设置
void objcetAndSignalsInit()	实例初始化、关联控件与信号
void buttonStateChange(int arg)	对各个状态下的按钮状态做限制防止误操作
void commandParse()	【命令分析】命令分析调度
bool userDataRead()	读取用户设定的导航参数，如果输入非法则 false
void userLabelChange()	【模拟移动】根据行进方向及交通方式修改标签图片
void travelStart()	【模拟移动】开始模拟移动
void travelFinished()	【模拟移动】导航结束
void accurateIsOk(unsigned seq, unsigned lineEdit)	判断当前序号对应的建筑物是否是可以精确导航

	的“教学楼”、“公寓”。
void **updateTime**()	【时间推进】更新系统时间
void **updatePath**()	【模拟移动】模拟移动时更新区段路径
void **userMove**()	【模拟移动】模拟移动时移动用户标签
void **updatePeople**()	每 x 分钟取随机数更新一次建筑物内的人数
void **editFinished**()	编辑完成，命令分析结束
void **mapChange**()	【模拟移动】切换显示的地图
void ***closeEvent***(QCloseEvent *event)	当程序异常结束时，用户登出
void **on_logout_clicked**()	【按钮槽】用户登出
void **on_checkBox_talk_stateChanged**(int arg1)	【按钮槽】语音播报功能的单选框
void **on_start_clicked**()	【按钮槽】开始模拟移动
void **on_make_clicked**()	【按钮槽】开始制定路线
void **on_pause_clicked**()	【按钮槽】用户暂停移动
void **backToLogin**()	【信号】回到登录界面
void **on_campusGuide_destroyed**();	【按钮槽】窗口关闭时登出用户
void **on_dstChange_clicked**()	【按钮槽】行进途中改变目的地
void **on_strategyChange_clicked**()	【按钮槽】行进途中改变行进策略
void **on_search_clicked**()	【按钮槽】查询周边建筑物，以视图的形式展示
void **on_searchList_doubleClicked**(const QModelIndex &index)	【按钮槽】双击视图中某一项时
void **on_checkBox_accurate_stateChanged**(int arg1)	【按钮槽】精确导航功能的单选框
void **on_siteView_clicked**(const QModelIndex &index)	【按钮槽】命令分析对应的可选建筑物展示
void **on_preferredView_doubleClicked**(const QModelIndex &index)	【按钮槽】展示用户常用路径
void **signal_travelPause**()	【信号】暂停时发出暂停信号
void **signal_travelContinue**()	【信号】暂停恢复时发出继续信号
void **removeStagnation**()	当本次导航结束下次导航开始前，移除上一次导航过程中因暂停产生的驻点
void **slot_travelPause**();	【信号槽】使用户暂停的一系列操作
void **slot_travelContinue**()	【信号槽】使用户继续移动的一系列操作
void **preferredPathShow**()	【信号槽】每当用户完成一次导航，则更新一次偏好路径显示

六、设计中的重要设计与分析

1.路线规划算法的设计及最优性证明

仔细分析四种策略下的路线规划算法容易发现，本质上它们都是带权无向图的多源最短路径问题，不同的策略差异只表现在无向图的权值上。因此，它们可以使用一个核心的路线规划算法来解决。在综合考量 dijkstra 和 floyd 算法的实用性和可推广性，最后决定采用 floyd 算法来实现路线规划。

下面给出最短距离策略的 floyd 算法作示例：

```
void campusGuide::shortestDistance()
{
    int n=graph_nodes.size(); // 顶点个数
    int path[MAX_NUM][MAX_NUM];  // path[i][j]记录从i→j经过了哪个点
    memset(path, -1, sizeof path);

    int graphTemp[n+1][n+1];
    for(int i=1;i<=n;i++)
        for(int j=1; j<=n;j++)
            graphTemp[i][j] = Graph[i][j];

    // Floyd算法
    for(int k=1;k<=n;k++){ //选中的中间值
        for(int i=1;i<=n;i++){ //数组横坐标
            for(int j=1;j<=n;j++){ //数组纵坐标
                if(graphTemp[i][j]>graphTemp[i][k]+graphTemp[k][j]){     //如果以k中间点为中间点检测到路径更短
                   graphTemp[i][j]=graphTemp[i][k]+graphTemp[k][j];       //更新路径值
                   path[i][j]=k;    //更新要经过的中间点
                }
            }// end of for j
        }// end of for i
    }//end of for k


    qDebug() <<"路径长度: " << graphTemp[uuser.src.seq][uuser.dst.seq];
    QList<unsigned> myPath;

    pathGenerate(path, myPath, uuser.src.seq, uuser.dst.seq); // 根据得到的floyd结果矩阵输出节点序列
    myPath.append(uuser.src.seq);
    qDebug() <<uuser.src.seq;
    makePath(myPath);    // 由输出的路线指定用于导航的路径path
}
```

下面简单证明一下途径中转点的最短距离问题是可以通过起点到中转点，再由中转点到终点的两次 floyd 算法得到解决的。首先需要明确的是，不同于 dijkstra 算法是基于贪心思想，floyd 算法是动态规划的。而要证明 floyd 中转点问题可以由两次 floyd 算法解决，传统意义上应该证明该问题同时具备最优子结构性质与子问题重叠性质。但我们注意到，floyd 算法的本质就是通过一个中转点对节点间的连通性作传递，因此算法本身就是一个必经中转点的算法，只是这个中转点从 1~N 被遍历了 N 次。

我们这里从有 N 个节点的无向图入手进行证明。若图中两个节点不连通，那么经过算法计算后仍为不连通。若它们连通就必然存在一条最短路径。假设任意 2 节点之间存在一条最短路径，该路径上有 M 个节点（M<N）。floyd 算法为三重循环，我们重点考虑最外层循环，我们将最外层循环此时所在的节点称为“桥点”。桥点遍历到了某一节点时，算法就会计算任意一点 A 到桥点的距离加上任意一点 B 到桥点距离的和，并将这个和与矩阵中存储的 A 到 B 的路径的距离进行比较，将短的存入矩阵中相应的位置。

2.路线规划算法的缺陷--输出路径时出现跳点

跳点：如 2-3-4-5-6 的直线上，要从 6 到 3,理论上输出的路径是 6->5->4->3，但实际输出为 6->5->3。即因为连通性造成了输出跳点。

测试时发现，从大序节点到小序节点的导航路线设计完全正确，但反过来就会错误。仔细分析，可能是无向图造成的。查阅诸多资料发现，很多人使用 floyd 算法计算无向图的最短路径时，都是将其当做有向图计算的，这样在基于连通性更新 path 矩阵时不会出现意外的跳点。

解决：做判断，直接置换两节点的起始地位。因为路径是一样的，区别只是方向而已。（也可以通过在从 path 矩阵生成路径时，增加条件分支做更精细的判断来控制）

```
void Guide::pathGenerate(int path[][MAX_NUM], QList<unsigned> &myPath, unsigned u, unsigned v)
{
    bool reverse=false;
    if(u<v)
    {
        reverse = true;
        swap(u,v);
    }

    QStack<unsigned> S;
    S.push(u);

    while(path[u][v] != -1)
    {
        unsigned temp = path[u][v];
        if(Graph[temp][u] != INF) // Graph[v][temp] 当big→small √ ; but small→big × 大概率是无向图造成的
        {
            S.push(temp);
            u = temp;
        }
        else
        {
            unsigned mid = path[u][temp];
            while(path[u][mid] != -1)
            //while(Graph[u][mid]!=INF)
            {
                if(Graph[u][mid] != INF)
                {
                    u = path[u][mid];
                    S.push(u);
                }
                else
                {
                    mid = path[u][mid];
                }
            }
            S.push(mid);
            u = mid;
        }// end of else
    }// end of while
    S.push(v);

    // 将栈中的内容弹出到链表中
    if(reverse) // 如果u、v翻转过则直接转化为正序列表，否则逆序弹出
    {
        myPath.append(S.toList());
    }
    else
    {
        while(!S.empty())   myPath.append(S.pop());
    }
}
```

3.模拟用户移动的设计

(1) 模拟用户移动的实现

模拟用户移动本身并不难，只需要每隔一段时间更新一次用户位置即可。但具体实现时却遇到了问题，用户没有移动的那段时间怎么让程序也停在该段而不继续往前执行？即延时控制的实现。

错误思路：起初，打算使用一个带标志位的空循环 while 来使程序停下，当定时器触发时翻转标志位来退出循环，使程序继续向前执行。但实际运行时发现，整个程序都被阻塞了？？？即程序完全陷在了空循环中，定时器的槽函数根本无法触发翻转标志位，导致循环也无法退出，出现死锁。同理，容易想到 sleep in while 同样也会阻塞整个程序。

解决 1：将计时器移入子线程，从而避免被主线程的 while 阻塞。但考虑到并不打算处理多用户的并发操作，因此该方法仅做扩展学习。

解决 2：使用 QEventLoop 局部事件循环，它让主线程等待但仍保持响应。选择该方案除了因为它实现简单易控制外，还因为在实际测试过程，它使得定时器可以在 debug 模式下也能正常计时，触发槽函数，这无疑是程序调试一个极大助力，值得深入学习。

(2) 用户移动过程的精确控制，如暂停与恢复的实现

想要用户在移动过程中暂停，就要关闭其对应的移动定时器，这很容易做到。但如何恢复？有两种思路，其一是关闭定时器前记录下该定时器的剩余时间，之后再用一个新的定时器，将其剩余时间走完，如何再开启定时器，但这样需要重写 QTimer 类且测试过于麻烦；

其二便是原地处理，设计之初是用的延时暂停的手段，即在当前区段收到了暂停信号，但必须在当前区段走完后才停下，它必定停在地图节点上且剩下的路径恢复时还可以直接使用，这样就不用考虑“野外”暂停出现。但将暂停延时显得有些不符合实际，因为实际场景中必定是立马停在原地而不会继续向前越走越远。

对于该暂停机制的优化很简单，每次暂停时以用户当前位置为基础，建立一个称为“驻点”的新地图节点插入地图映射中，之后恢复时直接以当前“驻点”为起点重新规划路径即可。这样做，不仅能完美实现暂停与恢复，还能契合其他扩展功能，如临时修改目的地、导航策略等，最重要的是符合实际需求。

4.室内导航的实现

系统给出了一个单选框用以选择是否启用精确导航。仔细分析，室内导航除了不考虑拥挤度外，本质上跟室外导航完全一致。因此可以使用相同的路径规划算法，唯一的不同就是室内导航使用的是室内节点，室外导航使用的是室外节点。

为了最大程度与室外导航兼容，同时减少重复代码使用，可以将室内室外节点都放到同一个映射 graph_nodes 中，以 campus 字段作区分。同时建立室内室外连接的特殊节点，

如“教学楼”与室外连接的节点为“楼梯口”，“公寓”与室外的连接节点为“电梯口”。这样就使得原本的路径规划算法可以进一步扩展到室内导航。

但这样设计的问题在于，寻路时出现“抄近路”的情况。如下例，从学生公寓女到学生公寓男显然是有一定距离的。但如果设定了室内室外的特殊连接节点，此时我们认为“学生公寓女”到“电梯口”在理论上是重合的（因为实际环境中进公寓到电梯口/楼梯口的距离相对于全程就是可以忽略不计），那么同理也得到“学会公寓男”到“电梯口”也重合。此时通过传递，意外的使原本有一定距离的两公寓“重合”。具体表现为，正常情况下，此时的路径应该为【学生公寓女->20 号节点->学生公寓男】，而启用了室内室外特殊节点时得到的路径为【学生公寓女->电梯口->学生公寓男】，显然是因为在室外导航时使用了室内节点导致的。

因此需要在不同的情况下，启用或禁用室内室外的特殊连接节点，同时设置该节点到室外的距离为 1 而不是完全重合的 0。每次调用路径规划函数时，额外传入两个参数 from、to 用以判断当前路径是否是纯室外导航。（只要有一个室内节点就不是纯室外导航）

```
bool Guide::isGuideBetweenIndoorAndOutdoor(unsigned from, unsigned to)
{
    if(graph_nodes[from]->campus!="教学楼" || graph_nodes[from]->campus!="公寓"
       ||graph_nodes[to]->campus!="教学楼" || graph_nodes[to]->campus!="公寓")
       return true;
    return false;
}
```

在每个路径规划算法，给临时无向图赋值前，先根据起点与终点启用或禁用特殊连接节点，从而使得在室内导航与室外导航相对独立。

```
if(isGuideBetweenIndoorAndOutdoor(from, to))
    specialNodeAvaliable();
else
    specialNodeForbidden();
```

5.精确制导的进一步思考

给起点新增一个精确导航的楼层房间号，相当于在原先的路线上加了一段起点到该房间的路径。同理对中转点和终点的精确导航也是如此。仔细分析发现，这本质上与途径某地的路径规划算法一致，因此可以如下设计路径调度函数：

①不途径中转点

```
if(uuser.strategy == "最短距离")
{
    if(uuser.dstAccurate.seq)
    {
        shortestDistance(myPath, uuser.dst.seq, uuser.dstAccurate.seq);
        myPath.removeLast();
    }
    shortestDistance(myPath, uuser.src.seq, uuser.dst.seq);
    if(uuser.srcAccurate.seq)
    {
        myPath.removeLast();
        shortestDistance(myPath, uuser.srcAccurate.seq, uuser.src.seq);
    }
}// end of 最短距离
```

②途径中转点时，需要做更为精细的设计以避免重复节点的出现。（多重条件判断）

```
    else if(uuser.strategy == "最短距离(途径某地)")
    {
        if(isEnterIndoor(uuser.dstAccurate.seq))
        {
            shortestDistanceByTransfer(myPath, uuser.dst.seq, uuser.dstAccurate.seq);
            myPath.removeLast();
        }
        if(isEnterIndoor(uuser.transAccurate.seq))
        {
            shortestDistanceByTransfer(myPath, uuser.transPoint.seq, uuser.dst.seq);
            myPath.removeLast();
            shortestDistanceByTransfer(myPath, uuser.transAccurate.seq, uuser.transPoint.seq);
            myPath.removeLast();
            shortestDistanceByTransfer(myPath, uuser.transPoint.seq, uuser.transAccurate.seq);
            myPath.removeLast();

            if(uuser.srcAccurate.seq)
            {
                shortestDistanceByTransfer(myPath, uuser.src.seq, uuser.transPoint.seq);
                myPath.removeLast();
                shortestDistanceByTransfer(myPath, uuser.srcAccurate.seq, uuser.src.seq);
            }
            else
            {
                shortestDistanceByTransfer(myPath, uuser.src.seq, uuser.transPoint.seq);
            }
        }
        else if(uuser.transPoint.seq)
        {
            shortestDistanceByTransfer(myPath, uuser.transPoint.seq, uuser.dst.seq);
            myPath.removeLast();

            if(uuser.srcAccurate.seq)
            {
                shortestDistanceByTransfer(myPath, uuser.src.seq, uuser.transPoint.seq);
                myPath.removeLast();
                shortestDistanceByTransfer(myPath, uuser.srcAccurate.seq, uuser.src.seq);
            }
            else
            {
                shortestDistanceByTransfer(myPath, uuser.src.seq, uuser.transPoint.seq);
            }
        }
        else
        {
            if(isEnterIndoor(uuser.srcAccurate.seq))
            {
                shortestDistanceByTransfer(myPath, uuser.src.seq, uuser.dst.seq);
                myPath.removeLast();
                shortestDistanceByTransfer(myPath, uuser.srcAccurate.seq, uuser.src.seq);
            }
            else
            {
                shortestDistanceByTransfer(myPath, uuser.src.seq, uuser.dst.seq);
            }
        }
    }// end of 最短距离(途径某地)
    else if(uuser.strategy == "最短时间(校内可选交通工具)")
    {
        if(isEnterIndoor(uuser.dstAccurate.seq))
        {
```

七、系统范例执行结果及测试情况说明

这里只给出一组用例用以说明系统运行情况，详尽的测试用例在“系统范例执行结果及测试情况说明”文档中给出。

【测试用例】最短距离（途径）& 跨校区 & 精确制导

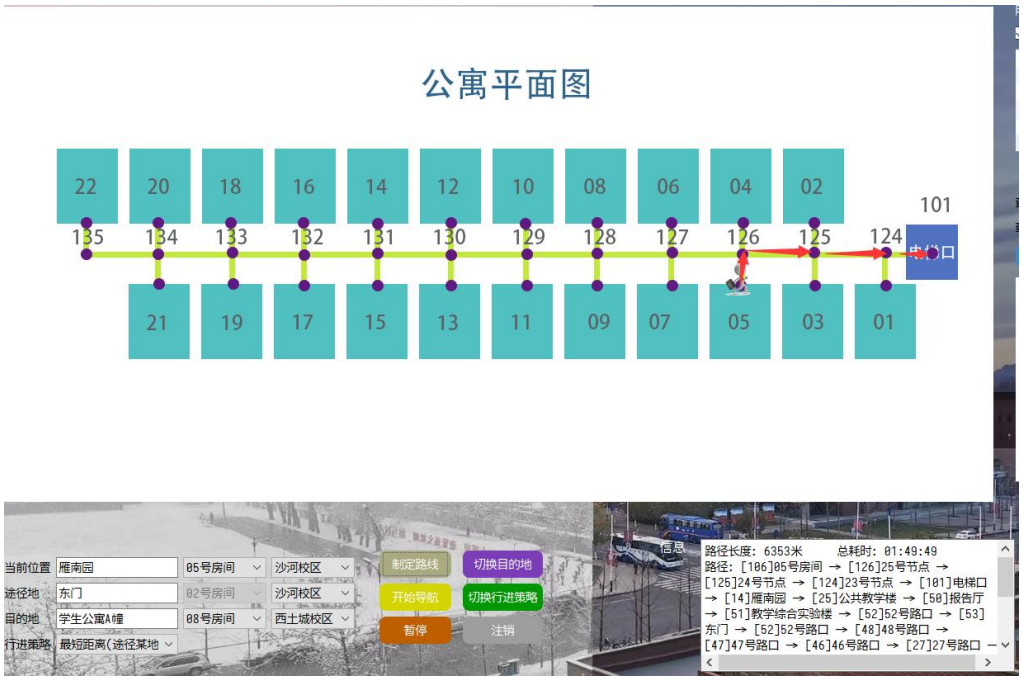
起点：【沙河校区】雁南园-05 号房间

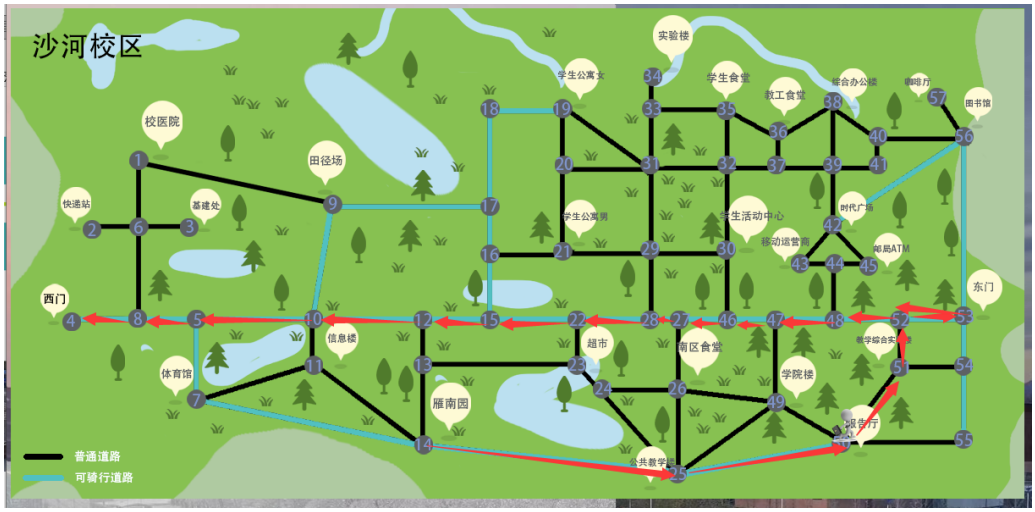
中转点：【沙河校区】东门

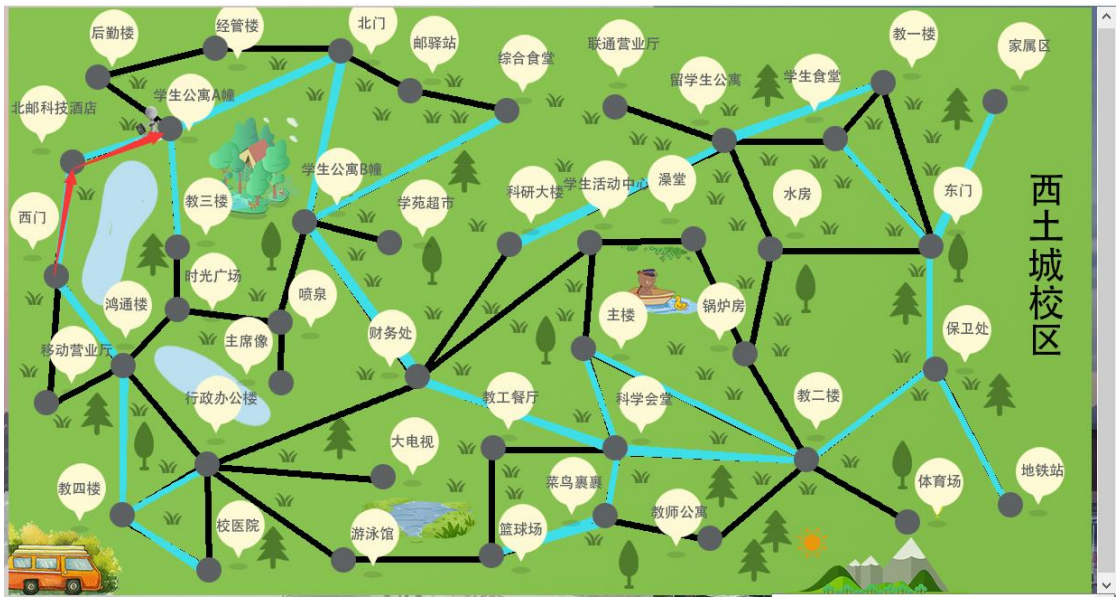
终点：【西土城校区】学生公寓 A 幢-08 号房间

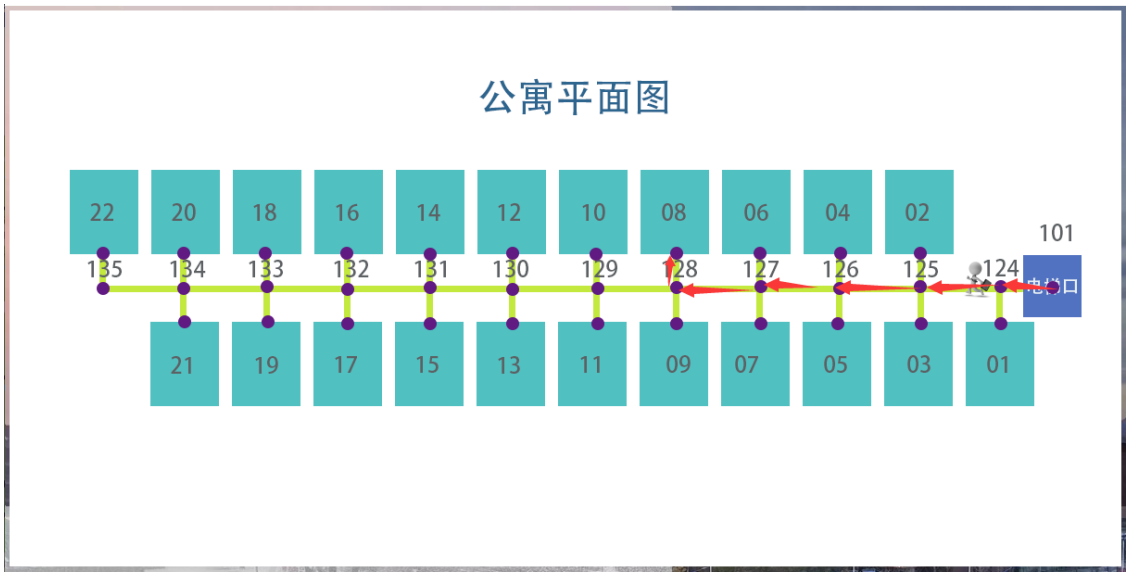
导航策略：最短距离策略（途径某地）

导航路线：









结果分析：

①用户移动轨迹与制定的路线一致；

②模拟移动过程中室内外地图切换正常；

③校区间乘车移动过程正常。

八、评价和改进意见

1.系统评价

首先从用户的角度来看，系统很好的满足了用户的各项基本需求，包括基本导航功能的实现、实时查询周边建筑物、临时修改目的地和导航策略等。除此之外，还考虑到了用户的交互体验，设置了语音播报功能，以及记录最近和最常使用路径等偏好设置，用于快速导航。整体来说，系统的功能相对完备、用户操作简便、用户体验也较好。

而从程序开发人员的角度看，除了以上优点，还有系统各模块间独立性较好，系统维护简单，扩展性强，但本系统仍旧存在一些实现细节上的瑕疵。比如，用户暂停后，系统直接清空了剩余路径使其停下，恢复时是重新规划了路径，并不是传统意义上的暂停与恢复，这导致在同一区段多次暂停时会出现嵌套命名以及路径日志异常的问题。

2.系统改进意见

①界面显示与用户交互优化：如鼠标点选目标；
②优化模拟移动过程中的暂停与恢复机制；
③优化模拟移动过程的实现，采用多线程而不是单线程多事件循环；
④向多进程延伸扩展，支持多用户同时导航。

3.心得体会

本次课程设计总的来说相对顺利，这主要得益于前期系统架构完备、文档与流程图准备充分，使得设计与实现相对独立，即中期编码阶段只是将前期的一些架构设计实现即可，而不是编码与设计并行，这样也使得各模块间独立性较好，也极大简化了后期的系统测试工作。

所以说，通过这次课设经历，让我明白前期的文档、流程图、系统架构远远比编码来得重要，拿到一个设计任务，并不是立马开始着手实现，而是完备分析其结构框架，制定相对合理的设计计划：先设计、再编程、最后才是测试。

除此之外，在开发过程中，遇到过很多课程之外的问题，这些都得靠自己解决，分析产生问题的原因，是否对全局有影响，解决办法是什么等等。这个过程中极大提升了个人发现问题、解决问题的能力，同时也激发了课外学习的动力，并不是用到哪部分就学哪部分，而是要完全掌握知识，洞悉其原理而不只是该怎么使用。

然后是对于团队合作的体会，由于前期设计合理，系统各模块间相对独立，使得组内成员的分工可以更明确，细化到谁谁谁负责哪个模块，提供什么接口等，极大加快的开发速度，同时大量的模块单元测试也一定程度上缩小了系统的整体测试规模，减少了测试压力。这就是团队的力量！