# 各模块设计说明

### form模块

该模块包括以下函数

const char\* **trans\_city\_back**(int city)

//城市与城市变量之间的转换函数

int **trans\_city**(char\* string)

//城市与城市变量之间的转换函数

const char\* **trans\_transportation\_back**(int transportation)

//交通工具与交通工具变量之间的转换函数

int **trans\_transportation**(char\* string)

//交通工具与交通工具变量之间的转换函数

以上四个函数的主要作用是实现城市、交通工具的代码与文字的转换 使得程序输出有更好的可读性

### get\_table模块

该模块包括以下函数

void **set\_time**()

//获取系统时间

void **change\_time**()

//修改时间

void **transform**(char\* string)

//城市名称的大小写转换

void **cut\_timetable**(int timetable\_temp[130][6])

//该函数用来将得到的总时间表切割为三维数组 对应各个城市

void **get\_timetable**()

//用来从文件中时间表的函数

void **get\_danger**()

//用来从文件中风险情况的函数

以上6个函数实现了读取danger.txt和timetable.txt，并将其转化为便于使用的，不同城市单独时刻表。并且实现了对时间的设定和更改

### mainwindow模块

该模块包括以下函数

int **on\_btntravel\_clicked**();

//选择旅行策略

void **on\_check\_all\_clicked**();

//查询当前所有旅客情况

void **on\_check\_sb\_clicked**();

//查询某个指定旅客

void **on\_start\_clicked**();

//开始模拟

void **show\_time**();

//显示时间

void **delay\_**(int msec);

//sleep函数

void **on\_new\_traveller\_clicked**();

//进入新建旅客状态

void **on\_new\_tr\_isok\_clicked**();

//确定建立旅客

void **on\_new\_tr\_iscan\_clicked**();

//取消建立旅客

void **on\_pause\_clicked**();

//暂停

void **on\_go\_on\_clicked**();

//继续

int **on\_x\_clicked**();

//倍速

void **on\_output\_log\_clicked**();

//输出日志

以上这些函数利用QT的信号与槽实现了UI界面和用户的交互

void **set\_map**();

//设置地图

上面set\_map函数完成了地图的设定以及资源的读取

### passenger模块（核心模块）

以下通过passenger类进行描述

**Passenger**() { m\_status = -1; m\_time = 0; };

//默认构造函数

void **initial\_a\_Passenger**(int day, int leave\_time, int source, int destination, int strategy, int passenger\_num, int max\_time);

//初始化限时模型下的旅客

void **initial\_a\_Passenger**(int day, int leave\_time, int source, int destination, int strategy, int passenger\_num);

//初始化非限时模型下的旅客

~**Passenger**() {};

//析构函数

以上4个函数主要完成passenger类对象的初始化

**核心算法模块**：

**1.最优策略生成模块：**

void **travel\_simulation\_DMS**(int trans, double danger, int temp\_time, Passenger& A);

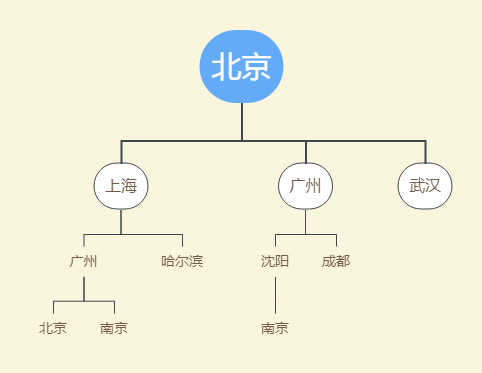
//非限时模型下的旅行模拟，用来求出最优旅行方案

void **travel\_simulation\_PDMS**(int trans, double danger, int temp\_time, int elapsed\_time, Passenger& A);

//限时模型下的旅行模拟，用来求出最优旅行方案

**算法思想：根据数据结构知识，将旅行出发的目的地视为根节点，所有可以乘坐的车次按照时间顺序依次排列为子节点。**

**示例(北京到南京的解空间树)如下：**



....

.........

........

.........

**通过解空间树的先序遍历，并且依靠对转车次数、最大预计风险的估算等条件限制里递归次数，并形成了相应的剪枝函数，能较快的根据旅行策略模拟出一个符合要求的计划。**

**特点：由于回溯法本身是穷举加剪枝，所以本算法基本不会遗漏任何合适的策略**

**该算法是本程序的核心部分，为本程序内除了基础函数的所有函数进行服务**

**且以下函数仅为本部分服务**

**{**

void **initial\_min\_danger\_trans\_and\_stack**();

//初始化旅行模拟辅助数组

void **stack\_back\_all**();

void **stack\_back\_1**(int trans);

void **stack\_back\_2**(int trans);

//退栈函数，用来辅助旅行模拟函数

**}**

**2.旅行、查询模块：**

QString **travel**();

//旅行函数 按照随时间进行旅客移动 返回QString类型以便文字显示

**该函数通过对于最优策略生成模块的获取来驱动passenger类对象的旅行，该函数通过对passenger类成员的数值来进行判断乘客处于何种状态**

**1.旅行状态**

**2.等车状态**

**3.旅行结束状态**

**4.(限时)旅客无法旅行状态**

**并通过相应的情况来进一步执行接下来的步骤**

**同时该函数返回一个QString类型用于与mainwindow模块进行交互，以便于在UI上实时显示日志，旅客运动情况等**

**此外通过实时的旅行情况，该函数也承担了如何使得mainwindow中的动画运动的任务**

**且以下函数仅为本部分服务**

**{**

int **label\_pos\_x**(int i);

//获取label横坐标

int **label\_pos\_y**(int i);

//获取label纵坐标

**}**

QString **inquire\_now**();

//查询当前状态函数

QString **inquire\_plan**();

//查询计划函数

**以上两个函数通过访问passenger类的旅行策略数组来反馈旅客当前的状态或者是旅客总的出行计划**

**两个函数均返回QString类型，目的与Travel函数相同，均用于与mainwindow进行交互，并方便日志输出**

**且以下函数为本部分服务**

**{**

int **get\_status**() { return m\_status; };

//获知旅客状态

void **get\_plan**();

//用于获取模拟的计划

**}**