

iTravel开发者文档

阅读本文档前，可先阅读用户手册以快速了解本软件。

目录

iTravel开发者文档

目录

1.概述

1.1设计任务

1.2功能需求说明及分析

1.3名词解释

2总体方案设计说明

2.1软件开发环境

2.2总体结构和模块划分

3数据结构说明和数据字典

3.1窗体类

3.2城市文件信息格式

3.3交通工具班次文件信息格式

3.4时间结构体

3.5交通工具班次结构体

3.6城市结构体

3.7用户需求结构体

3.8直达路线结构体

3.9旅行路线结构体

4各模块设计说明

4.1初始化模块

4.1.1主初始化函数 `init()`;

4.1.2读入交通工具函数 `void read_vehicle(ifstream & in_data, int type)`

4.1.3获取城市对应编号函数 `int get_city_no_vehicle_data(string city_name)`

4.2路线查找模块

4.2.1核心算法

4.2.2触发函数 `void MainWindow::on_findRoute_pushButton_clicked(bool checked)`

4.2.3查找路线函数 `void MainWindow::find_route()`;

4.3模拟旅行模块

4.3.1核心算法

4.3.2触发函数 `void MainWindow::on_simulation_pushButton_clicked(bool checked)`

4.3.3时间改变函数 `void MainWindow::changeTime()`

4.3.4交通工具状态改变函数 `void MainWindow::changeVehicleStatus()`

4.3.5改变计划函数 `void MainWindow::on_stopSimulation_pushBotton_clicked(bool checked)`

4.4日志输出模块

4.5时间模块

4.5.1获取当前时间函数 `void get_my_time(my_time & cur_time)`

4.5.2时间相加函数 `void add_time(my_time & _start, int _add)`

5附加功能说明

5.1模拟期间计划变更

5.2用户自定义模拟速度

5.3用户自定义准备出发时间

6范例执行结果与测试情况

6.1鲁棒性设置

6.2范例测试

- 6.2.1 总述
- 6.2.2 测试两种风险策略和自定义准备出发时间
- 6.2.3 测试模拟旅行
- 6.2.4 测试改变计划
- 6.2.5 测试模拟速度
- 6.2.6 测试日志文件

7 评价和改进意见

- 7.1 精确时间到分钟甚至秒
- 7.2 设置**SLOWER**和**FASTER**为滑块
- 7.3 设置更为清楚的提示让用户知道可以改变准备出发时间
- 7.4 改进旅行模拟中交通工具状态改变算法

1.概述

1.1设计任务

在当前COVID-19疫情肆虐的环境下，如何低风险地达到一个目的城市成为一个重要命题。各个城市的风险程度不一样，分为低风险、中风险和高风险三种；各种交通工具的风险也不同。要求设计一个系统，能根据风险评估，为旅客设计一条符合风险策略（最少风险策略和限时最少风险策略）的旅行线路；能查询当前时刻旅客所处的地点和状态（停留城市/所在交通工具）。

1.2功能需求说明及分析

根据用户给定的出发城市、目的城市、风险策略，在自定义地图模型中实现两种旅行线路的设计，包括最少风险策略和限时最少风险策略，并且能够实现旅行的模拟。具体要求：

1. 实现超过十个城市，不同城市设置不同的单位时间风险值：低风险城市为0.2；中风险城市为0.5；高风险城市为0.9。各种不同的风险城市分布要比较均匀，个数均不得小于3个。旅客在某城市停留风险计算公式为：旅客在某城市停留的风险=该城市单位时间风险值*停留时间
2. 建立汽车、火车和飞机的时刻表（航班表），假设各种交通工具均为起点到终点的直达，中途无经停。城市之间不能总只是1班车次；整个系统中航班数不得超过10个，火车不得超过30列次；汽车班次无限制。旅客的请求包括：起点、终点和选择的低风险策略。其中，低风险风险策略包括：
 1. 最少风险策略：无时间限制，风险最少
 2. 限时最少风险策略：在规定的时间内风险最少
3. 旅行模拟系统以时间为轴向前推移，每10 秒左右向前推进1个小时(非查询状态的请求不计时，即有鼠标和键盘输入时系统不计时)，系统时间精确到小时
4. 不考虑城市内换乘交通工具所需时间
5. 建立日志文件，对旅客状态变化和键入等信息进行记录

1.3名词解释

对本开发者文档中的名词做如下定义。

名词	解释
直达路线	一个交通工具班次对应的路线，该定义包含交通工具、出发城市、目的城市、出发时间、历时
旅行路线	出发城市到目的城市过程中所经过的所有直达路线，“路线”不特别说明则指旅行路线
窗体	Qt的概念，既是软件图形界面的主窗口的C++类，也是软件图形界面的主窗口
出发城市-目的城市	对应于旅行路线
当前城市-到达城市	对应于直达路线
准备出发时间	指用户计划的开始旅行时间，在此基础上用户需要在出发城市等待一定的时间才能乘坐交通工具出发，故称“准备出发时间”。默认为当前时间，本软件中用户可自定义修改。

2总体方案设计说明

分析功能需求，可将一次响应用户需求分为以下步骤：

梳理功能需求，得到程序的处理过程如下：

1. 用户输入出发城市，目的城市，风险策略，日志文件记录输入信息；
2. 根据风险策略，使用算法计算得到旅行路线，方案内容包括路线信息（从出发城市到目的城市途径的城市）、时间信息（处于每个地点的开始时间和结束时间），总时间和总风险值，日志文件记录最终计算结果；
3. 查询结束后根据用户需求模拟旅行，每10秒推进一个小时，日志文件记录位置和状态。

2.1软件开发环境

Windows 10操作系统，使用C++语言，使用Desktop Qt 5.9开发图形界面，使用MinGW 32bit为编译工具。

2.2总体结构和模块划分

由于使用Qt开发图形界面，main函数中只调用窗体，软件主体在mainwindow.cpp中，体现为mainwindow受到用户的输入触发相应功能的执行。

模块划分为：**初始化模块**，**路线查找模块**，**模拟旅行模块**，**日志输出模块**，窗体对象调用以上模块实现软件功能。

总体结构为：

1. 窗体构造时进行调用**初始化模块** (`void init();`) 读入城市和交通工具班次信息；
2. 用户输入信息后对信息做检查并报告错误；
3. 用户要求查询路线后，调用**日志输出模块**记录用户需求、最佳路线、模拟情况，调用**路线查找模块** (`void find_route(const request &user_request, route_info &best_route);`)；
4. 用户要求模拟，调用**模拟旅行模块**(`void MainWindow::on_simulation_pushButton_clicked(bool checked)`)

3数据结构说明和数据字典

软件中用到的跟功能相关的数据结构和宏定义位于**struct.h**中，与Qt使用相关的头文件位于**mainwindow.h**中。

3.1窗体类

```
1      Ui::MainWindow *ui;  
2      request user_request; //记录用户需求  
3      route_info best_route; //记录最佳路径  
4      QGraphicsScene *scene;  
5      my_time simu_time; //记录当前模拟时间  
6      //图片相关  
7      QGraphicsPixmapItem *vehicle_item;  
8      QGraphicsPixmapItem * waiting_item;  
9      QGraphicsPixmapItem *finish_item;  
10     QPixmap map_png;  
11     QPixmap waiting_png;  
12     QPixmap plane_png;  
13     QPixmap train_png;  
14     QPixmap car_png;  
15     QPixmap vehicle_png;  
16     QPixmap finish_png;  
17     route *now_route; //记录当前模拟路径  
18     int wait_time_last; //记录当前剩余的等待时间  
19     int vehicle_time_last; //记录乘坐当前交通工具的剩余时间  
20     QTimer *time_timer; //定时器，管理模拟时间的变化  
21     QTimer *status_timer; //定时器，管理交通工具显示的变化  
22     double height_diff; //当前点到点路线中每个TIME_SCALE需要增加的高度  
23     double width_diff; //当前点到点路线中每个TIME_SCALE需要增加的宽度  
24     int cur_progress; //当前点到点路线中交通工具的进度  
25     string vehicle_name; //当前交通工具名称  
26     int SECPERHOUR; //模拟一小时对应的真实秒数
```

3.2城市文件信息格式

格式：[城市名称] [城市风险编号] [城市在地图中的位置x] [城市在地图中的位置y];
其中，风险编号0为高风险，1为中风险，2为低风险。

城市的风险编号对应的风险值定义位于**struct.h**中，为 `const double city_risk_no2value[3]={0.9, 0.5, 0.2};`。

3.3交通工具班次文件信息格式

格式：[出发城市名称] [目的城市名称] [出发时间 hh:mm] [历时 hh:mm]。

交通工具的编号对应的风险值定义位于**struct.h**中，为 `const int vehicle_risk_no2value[3]={9, 5, 2};`。

3.4时间结构体

```

1 struct my_time{
2     int year;
3     int month;
4     int day;
5     int hour;
6     int minute;
7 };
8
9 struct my_clock{
10    int hour;
11    int minute;
12 };

```

3.5交通工具班次结构体

```

1 struct vehicle{
2     int type;//交通工具类型，0：飞机，1：火车，2：汽车
3     int dst_city_no;//目的城市编号
4     my_clock start_time;//交通工具出发时间
5     my_clock time_spent;//交通工具到达时间
6 };

```

3.6城市结构体

```

1 struct city{
2     int city_risk_no;//城市风险编号
3     string city_name;//城市名称
4     vector <vehicle> my_vehicle;//城市的交通工具向量，包含城市的所有交通工具班次
5 };

```

3.7用户需求结构体

```

1 struct request{
2     string id;
3     int dept_city_no;//用户需求的出发城市
4     int dst_city_no;//用户需求的目的地城市
5     int travel_type;//0:最小风险策略； 1:限时最小风险策略
6     int limited_time;//限时的时间，单位：小时
7 };

```

3.8直达路线结构体

```

1 struct route{
2     my_time arrival_time;//到达当前城市的时间
3     int city_no;//当前城市的编号
4     int wait_time;//等待下一交通工具出发的等待时间
5     int vehicle;//下一交通工具的类型
6     int vehicle_time;//在下一交通工具上花费的时间
7     route * next_ptr;//下一个直达路线信息
8 };

```

3.9旅行路线结构体

```
1 struct route_info
2 {
3     double total_risk;//该旅行路线的总风险
4     int total_time;//该旅行路线的总时间
5     route_ptr detail_route;//旅行路线，由直达路线结构体的链表构成
6 };
```

4各模块设计说明

4.1初始化模块

模块在init.cpp文件中，包括读入城市信息和交通工具的班次信息，读入后各个城市以city结构体存于cities[]数组中。每个交通工具将以vehicle结构体按读入顺序接入出发城市的交通工具链表中。

本模块包含以下三个函数：主初始化函数init()，读入交通工具函数void read_vehicle(ifstream & in_data, int type)，获取城市对应编号函数int get_city_no_vehicle_data(string c)，以下具体说明这三个函数

4.1.1主初始化函数init();

负责从city.data文件中读入城市信息，读入顺序即城市编号（从0开始编号），也即城市对应的数组下标；同时调用读入交通工具函数void read_vehicle(ifstream & in_data, int type)依次读入飞机航班信息，火车车次信息，汽车车次信息。

4.1.2读入交通工具函数void read_vehicle(ifstream & in_data, int type)

由于共有三种交通工具信息，并位于不同文件夹，为简化代码，构造该函数。

虽然系统时钟仅精确到小时，但读入的时候将完整读入班次的出发时间和路途时间，这是为了方便日后软件拓展。

4.1.3获取城市对应编号函数int get_city_no_vehicle_data(string city_name)

由于交通工具班次的信息是由城市名称构成而非城市编号，存储于出发城市的交通工具链表时需要找到出发城市对应的编号。参数city_name为城市名称，返回城市编号。

注意：如果此时发现交通工具的出发城市名称不在已读入的城市数组中，将在Qt控制台报告错误，但不在软件界面报告错误，并且不终止软件运行。这是为了保证软件的使用体验并考虑到出现其余城市或拼写错误不影响软件的正常运行。

4.2路线查找模块

4.2.1核心算法

路线查找使用的算法是深度优先遍历+分支界限法（或称回溯法）。

由出发城市开始，假设第一个交通工具班次可行，到达目的城市后，假设该城市的第一个交通工具班次可行，继续深度优先搜索，如果发现班次的目的城市已经在前面的路线中，或发现当前总时间超过了用户要求（若用户的需求是限时最小风险策略）或当前总风险超过了当前最佳路线的总风险，则放弃该直达路线，转而使用当前城市的下一个交通工具班次，继续使用深度遍历。当得到一条从出发城市到目的城市的路线，且路线比当前最佳路线的总风险少，则将该路线替换为该最佳路线，否则忽略该路线。无论是否替换，取消该路线的最后一条直达路线，转而遍历新的当前城市的下一个交通工具班次，直至该城市的所有线路遍历完毕，再向上回溯。

路线查找模块共包含四个函数：

触发函数 `void MainWindow::on_findRoute_pushButton_clicked(bool checked);`，查找路线函数 `void MainWindow::find_route();`，以上两个函数实现为窗体类的成员函数，在 **MainWindow.cpp** 中；深度优先遍历+分支界限法查找函数 `void trace_back(route_ptr cur_ptr, const request & user_request, double risk_v, int time_v, route_info & best_route, route_ptr & head);`，赋值当前最佳路线函数 `void duplicate_best_route(route_info & best_route, route_ptr & temp_route);`，以上两个函数实现在 **find_route.cpp** 中。

4.2.2 触发函数 `void`

`MainWindow::on_findRoute_pushButton_clicked(bool checked)`

当用户点击 **SEARCH!** 按钮后触发查找路线，该函数负责：

1. 检查当前用户的输入是否符合规范，如若发生未选择出发城市、未选择目的城市、选择限时但未输入限时时间、出发城市和目的城市相同，则报告错误信息；
2. 调用查找路线函数 `void MainWindow::find_route(const request &user_request, route_info &best_route);` 查找最佳路线，路线情况存储于 `route_info` 结构体中；
3. 调用日志输出模块输出用户请求和查询结果，并将模拟信息打印到日志中；
4. 在窗体中显示最佳路线的具体信息






4.2.3 查找路线函数 `void MainWindow::find_route();`

该函数实现查找路线的初始化工作，设定旅行的初始时间（本软件的附加功能：可实现用户自定义出发时间），设定最佳路线的头指针和临时路线的头指针。

注意：为了方便设计算法，最佳路线实现为带头指针的链表，即 `best_route.detail_route` 是头指针，`best_route.detail_route->next_ptr` 才是路线的第一个直达路线信息。而临时路线不设置头指针，`head` 即为临时路线的第一个直达路线信息。

4.3 模拟旅行模块

在地图上模拟整个旅行过程，主要体现为三个过程：

1. 等待状态，地图将在等待的城市坐标上显示时间漏斗  表示等待；
2. 乘坐交通工具状态，根据不同交通工具，地图将按乘坐时间从出发城市到下一站直达城市均匀移动交通工具图标，图标为飞机 ，火车 ，汽车 ，由于在班次设定中大致考虑了三种交通工具的速度，所以可以在模拟中比较明显得看到三者的时速不同。
3. 到达状态，将在目的城市坐标上显示完成图标  表示到达。

4.3.1 核心算法

使用定时器控制地图改变的时间，含两个定时器：时间定时器 `time_timer` 和状态定时器

`status_timer`，前者是关于时间改变的定时器，后者是地图上的图标改变定时器。当开始一次旅行或开启一次直达路线，则设定状态定时器为等待时间（系统不考虑换乘时间但为符合实际，在出发城市每座中转城市的等待时间都大于0小时）；当等待时间结束，则状态定时器到时，交通工具开始运行，设定状态改变定时器为时间精度 `TIME_SCALE`（指交通工具状态每隔 `TIME_SCALE` 即可改变一次，该定义不为用户自定义，但在代码中实现为宏定义，可以比较容易地改变），状态定时器到时更新当前的交通工具位置。

开启新的一次直达路线，将更新 `vehicle_time_last` 和 `wait_time_last`，分别表示乘坐交通工具的剩余时间和等待的剩余时间。由这两个变量控制当前的旅行状态——若 `wait_time_last>0` 则当前处于等待状态；若 `wait_time_last==0 && vehicle_time_last==now_route->vehicle_time` 则为从等待状态转为乘坐交通工具状态；若 `wait_time_last==0` 而 `vehicle_time_last>0`，则处于乘坐交通工具状态；若 `vehicle_time_last==0` 则说明当前直达路线已经完成。

使用“微分法”实现交通工具的移动，`TIME_SCALE` 默认为50ms，即每秒沿着直达路线移动交通工具20次。每次开启新一次的直达路线，将计算每个 `TIME_SCALE` 需要移动的距离（高度和宽度），在软件中定义为 `height_diff` 和 `width_diff`。计算方法为直达路线当前城市与到达城市的距离作差再除以该次交通工具行驶的总 `TIME_SCALE` 次数，以 `height_diff` 为例，代码为 `height_diff=(double(cities[next_city_no].posy-cities[now_route->city_no].posy))/now_route->vehicle_time/(SECPERHOUR*1000/TIME_SCALE)`，`now_route->vehicle_time` 即本直达路线的交通工具行驶时间。

设置 `cur_progress` 变量，用于每次状态改变时累加一，则每次重新设定交通工具的坐标时为 `vehicle_item->setOffset(cities[now_route->city_no].posx-vehicle_png.width()/2+width_diff*cur_progress,cities[now_route->city_no].posy-vehicle_png.height()/2+height_diff*cur_progress)`；在当前城市的坐标的基础上减去交通工具图片的宽度（高度）的一半，即为将交通工具的坐标放置在出发城市的坐标的中心，再加上进度相关的宽度（高度）改变值，即得该状态下交通工具的坐标。

在整个模拟过程中，`time_timer` 始终设定为模拟时间一小时对应的真实时长，在开始模拟即启动，每次结束后再重新启动以保证时间持续前推，直至该模拟结束，将暂停 `time_timer` 和 `status_timer`。

在整个模拟状态过程中，除了地图上的展示外，状态栏将展示当前的旅行状态。有两种旅行状态信息：

1. 等待状态：提示当前城市和剩余的等待时间；
2. 乘坐交通工具状态：提示当前城市和到达城市，乘坐的交通工具和剩余的乘坐时间

由于模拟旅行跟窗体联系密切，所以主要实现在窗体类实现函数 `MainWindow.cpp` 中。

该模块包含以下函数：触发函数 `void MainWindow::on_simulation_pushButton_clicked(bool checked)`，时间改变函数 `void MainWindow::changeTime()`，交通工具状态改变函数 `void MainWindow::changeVehicleStatus()` 和改变计划函数 `void MainWindow::on_stopSimulation_pushButton_clicked(bool checked)`。

4.3.2 触发函数 `void`

`MainWindow::on_simulation_pushButton_clicked(bool checked)`

当用户点击 **SIMULATION** 按钮后开始模拟旅行，会处理如下工作：

1. 重新设定窗体各个控件的状态；
2. 清除地图之外的图标；
3. 设定相关第一次直达路线的相关信息，包括 `now_route`，`wait_time_last` 和 `vehicle_time_last`，并且展示等待图标；
4. 设置两个定时器

4.3.3 时间改变函数 `void MainWindow::changeTime()`

当 `time_timer` 到时，将调用该函数。该函数负责判断状态改变，并设置相关信息。包括：

1. 从等待到交通工具出发：设定交通工具图片、名称，绘制初始交通工具图标，重置 `cur_progress`，计算 `height_diff` 和 `width_diff`，定时 `status_timer` 的时间为 `TIME_SCALE`，改变模拟时间

2. 当前交通工具到达直达路线的到达城市：

1. 若不是目的城市：改变 `now_route`，`wait_time_last` 和 `vehicle_time_last`，展示等待图标；
2. 若是目的城市：停止所有定时器，展示完成图标，设定窗体的控件的状态；

并改变模拟时间；

3. 当前状态和上一小时状态相同：打印状态信息，改变模拟时间。

4.3.4 交通工具状态改变函数 `void MainWindow::changeVehicleStatus()`

由于大量准备工作在 `changeTime()` 函数中完成，交通工具状态改变函数较为简单，其工作为在当前 `vehicle_time_last > 0` 的情况下，递增 `cur_progress`，重新展示交通工具，并定时 `status_timer` 为 `TIME_SCALE`，以自动启动下一次交通工具状态改变。

4.3.5 改变计划函数 `void`

`MainWindow::on_stopSimulation_pushButton_clicked(bool checked)`

改变计划函数在用户点击 **CHANGE PLAN** 时调用，可以在模拟期间的任何时刻暂停，系统将暂停画面，并设定出发城市的下拉框为当前城市。当前城市的定义为：若当前处于等待状态，则当前城市为当前正处于的城市；若当前处于乘坐交通工具状态，则当前城市为直达路线的到达城市。用户可以在此基础上改变目的城市，也可以改变出发城市。

注意：改变城市后需要重新点击 **SEARCH!** 查询路线；暂停模拟后点击 **SIMULATION!** 将重新模拟。

4.4 日志输出模块

日志输出模块实现在 `write_log.cpp` 中，包含用户需求日志函数 `void log_input(const request & user_request)`，最佳路线输出日志函数 `void log_output(const route_info & best_route, int dst_city_no)`，模拟状态日志函数 `void log_simulation(const route_info & best_route, int dst_city_no)`。

实现较为简单，主要通过指针控制进行输出路线。

4.5 时间模块

时间模块实现在 `time.cpp` 中，包含获取当前时间函数 `void get_my_time(my_time & cur_time)`，时间相加函数 `void add_time(my_time & _start, int _add)`。

4.5.1 获取当前时间函数 `void get_my_time(my_time & cur_time)`

在日志模块记录用户需求的时候调用。为了简化窗体类，日志模块并不实现为窗体的成员函数，故在日志模块记录用户需求时，直接使用 C++ 的库函数获取时间。

4.5.2 时间相加函数 `void add_time(my_time & _start, int _add)`

多处调用，实现 `_start` 往后推 `_add` 小时并将结果存于 `_start` 中。该函数处理了日月前推的情况，但没有处理跨年的情况。

5 附加功能说明

本软件实现为有图形界面的含交通工具风险的模拟旅行软件。在此基础上为方便使用，增设如下功能。

5.1 模拟期间计划变更

在旅行模拟期间，用户可在任意时刻暂停模拟，系统将暂停画面，用户重新设定旅行计划。

本软件为用户提供的便利是自动更新出发城市：设定出发城市的下拉框为当前等待位于的城市或当前直达路线的到达城市。

实现在改变计划函数 `void MainWindow::on_stopSimulation_pushButton_clicked(bool checked)` 中。

5.2 用户自定义模拟速度

用户可自定义模拟速度，即设置模拟一小时所用真实时间的秒数。默认为2s，用户可点击**FASTER**减少1s，可用**SLOWER**增加1s。当设置为0s时，将报错并改为1s。

具体实现为 `void MainWindow::on_slower_pushButton_clicked()` 和 `void MainWindow::on_faster_pushButton_clicked()` 两个接受到按钮点击后启动的函数。软件中设置 `SECPERHOUR` 为窗体的成员变量，意义是模拟一个小时对应的真实秒数，以上两个函数即对 `SECPERHOUR` 进行处理。

5.3 用户自定义准备出发时间

用户可以自定义准备出发时间，在非模拟状态下，通过改变模拟时间 `Simulation Time` 即可，查找路线将会获取模拟时间作为准备出发时间。

模拟时间改变方式可以是控件的上下按钮，也可直接输入数字。

具体实现为 `void MainWindow::find_route()` 函数，该函数中获取窗体显示的模拟时间并以此为出发城市的到达时间，即准备出发时间。查找路线模块以该准备出发时间为起点后推时间，模拟时间显示以当前窗体的模拟时间为基础增加一个小时，以上两点保证了程序的正确性。

6 范例执行结果与测试情况

6.1 鲁棒性设置

为了增强鲁棒性，本软件设置多处限制和错误提示：

1. 用户需求限制：设置了未输入出发城市、未输入目的城市、出发城市和目的城市相同、选择限时最小风险策略到未输入限时时长的错误信息，并且**限制了限时时长的大小为十进制9位数**，这是因为读入该信息后将转化为整型，可有效防止程序出错；
2. 查找结果错误提示：在限时最小风险策略下，如果在限时内找不到任何一条线路，则将报告出错；
3. 确定的控件使用权：
 1. 当不选择限时最小风险策略时，不允许输入限时时长；
 2. 当未开始模拟时，不允许暂停模拟；
 3. 当开始模拟之后，不允许改变模拟时间、ID、出发城市、目的城市、限时策略、限时时长，不允许点击**SEARCH!**、**SLOWER**、**FASTER**、**SIMULATION**，只允许改变计划
 4. 点击改变计划后，恢复到预设置的情况
 5. 不允许设置模拟速度为用真实时间0s模拟一小时，此时会报告错误并将速度重置为用真实时间1s模拟一小时。

6.2 范例测试

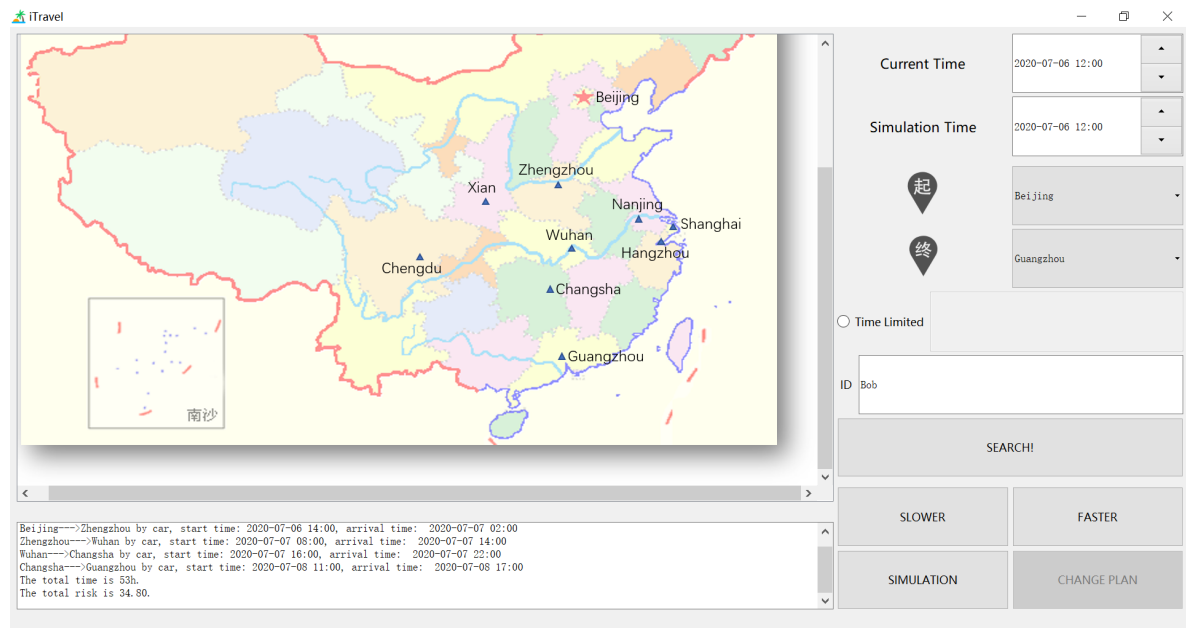
6.2.1 综述

软件测试是一个非常重要的过程，本软件一共经过三种测试：

1. 代码书写过程中的模块测试；
完成每个模块会进行模块测试：初始化模块中读入城市和交通工具信息后输出；查找路线模块中输出所有考虑的路线情况；日志输出模块中直接查看日志检查程序是否正确；模拟模块直接查看地图中交通工具移动是否正确。
2. 鲁棒性设置测试；
对于上述鲁棒性设置，在完成后均进行测试。
3. 综合范例测试；
以下面的测试过程为例，进行测试，实际测试过程遍历所有城市。

6.2.2测试两种风险策略和自定义准备出发时间

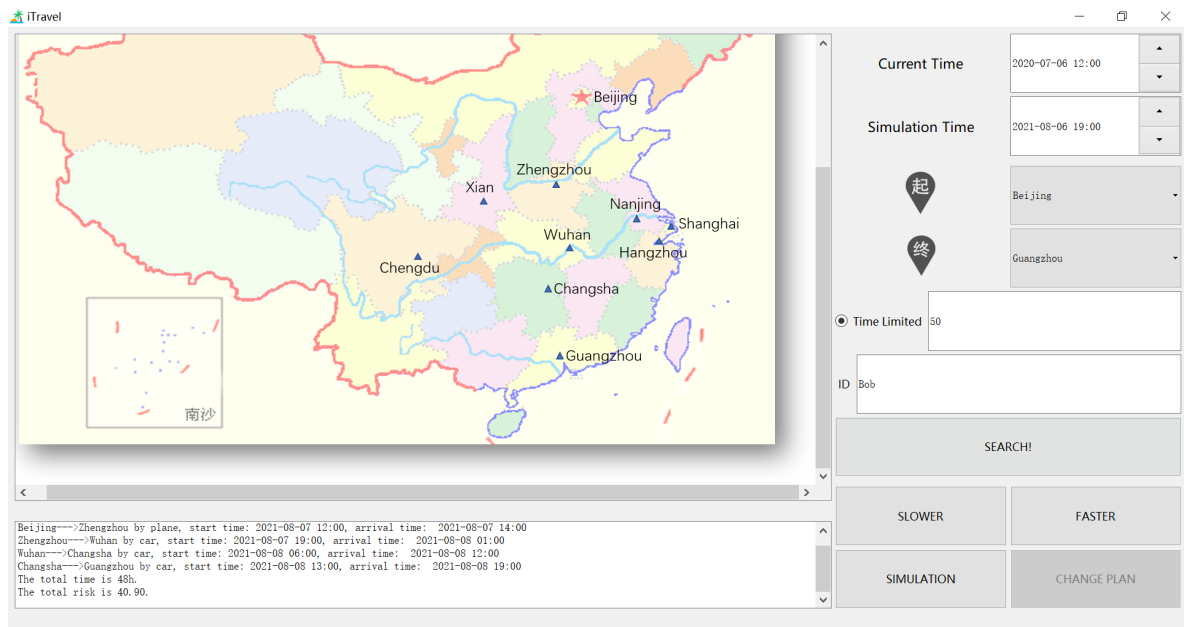
以北京到广州为例，不设置限时，得到最佳路径：



Your best route is

Beijing--->Zhengzhou by car, start time: 2020-07-06 14:00, arrival time: 2020-07-07 02:00
Zhengzhou--->Wuhan by car, start time: 2020-07-07 08:00, arrival time: 2020-07-07 14:00
Wuhan--->Changsha by car, start time: 2020-07-07 16:00, arrival time: 2020-07-07 22:00
Changsha--->Guangzhou by car, start time: 2020-07-08 11:00, arrival time: 2020-07-08 17:00
The total time is 53h.
The total risk is 34.80.

改为设置时间限制为50，并设置模拟时间为2021-08-06 19: 00，重新搜索，得到新的最佳路径：



Your best route is

Beijing--->Zhengzhou by plane, start time: 2021-08-07 12:00, arrival time: 2021-08-07 14:00

Zhengzhou--->Wuhan by car, start time: 2021-08-07 19:00, arrival time: 2021-08-08 01:00

Wuhan--->Changsha by car, start time: 2021-08-08 06:00, arrival time: 2021-08-08 12:00

Changsha--->Guangzhou by car, start time: 2021-08-08 13:00, arrival time: 2021-08-08 19:00

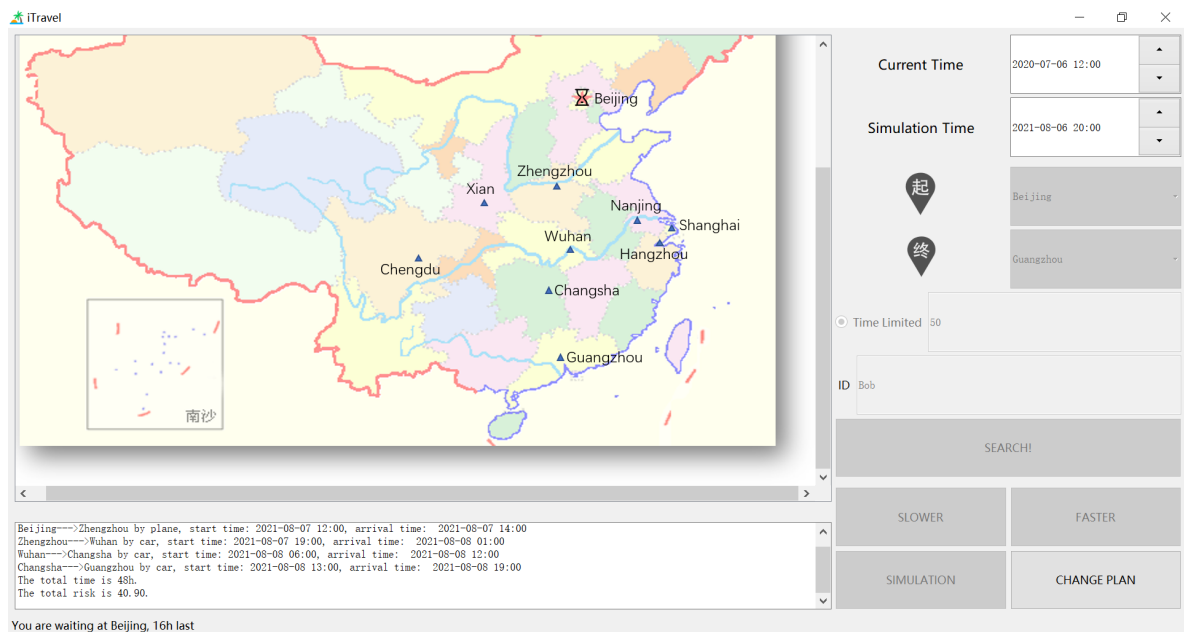
The total time is 48h.

The total risk is 40.90.

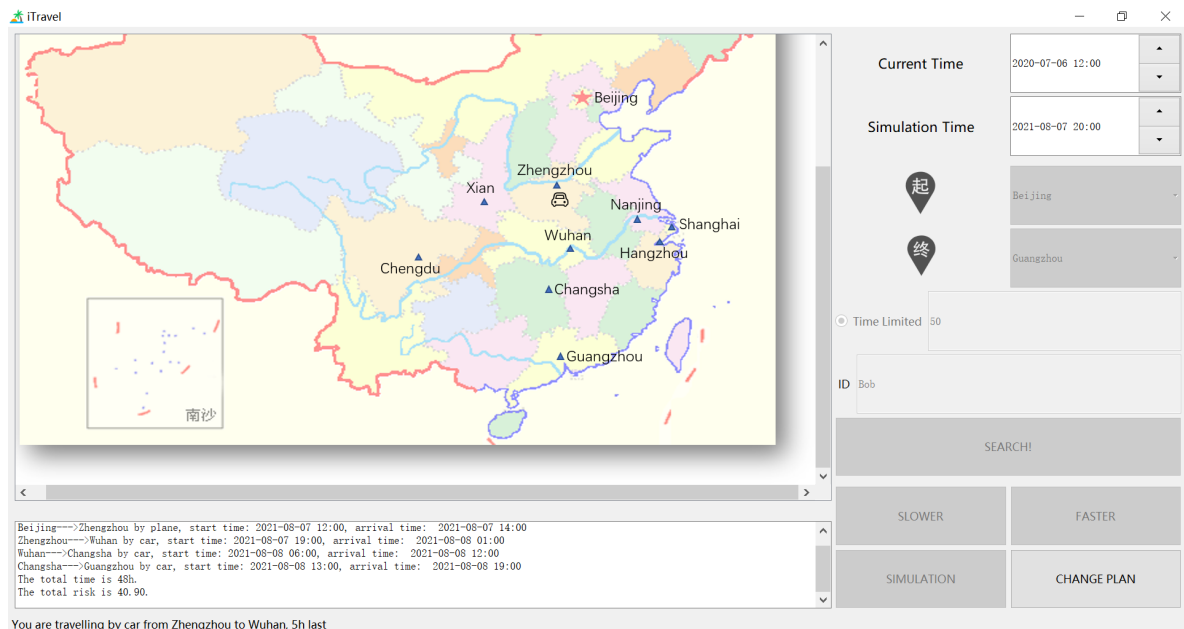
可以发现最佳路径的时间改成48h，其中北京到郑州的路线交通工具改为飞机，但总风险变高。

6.2.3测试模拟旅行

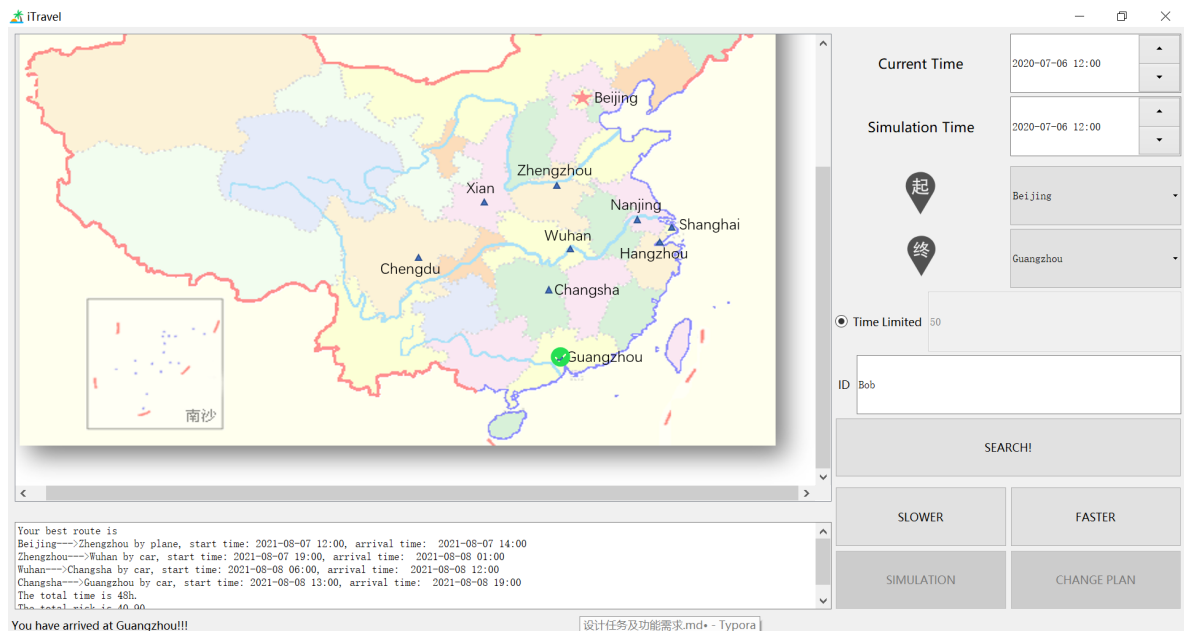
点击**SIMULATION**后，开始模拟，除了**CHANGE PLAN**外其他均设置为只读（模拟时间）和不可改动（其余）。此时处于等待状态，状态栏可见提示。



继续模拟，可以看到从郑州出发到武汉的小汽车：

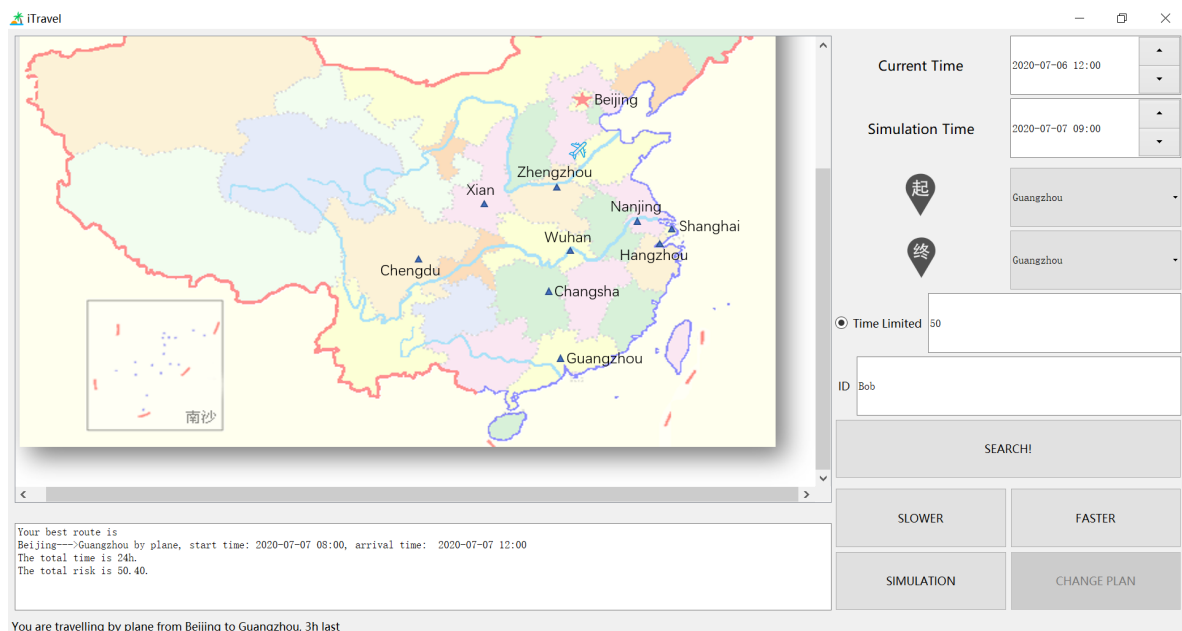


结束后，可见在终点城市显示完成标志：



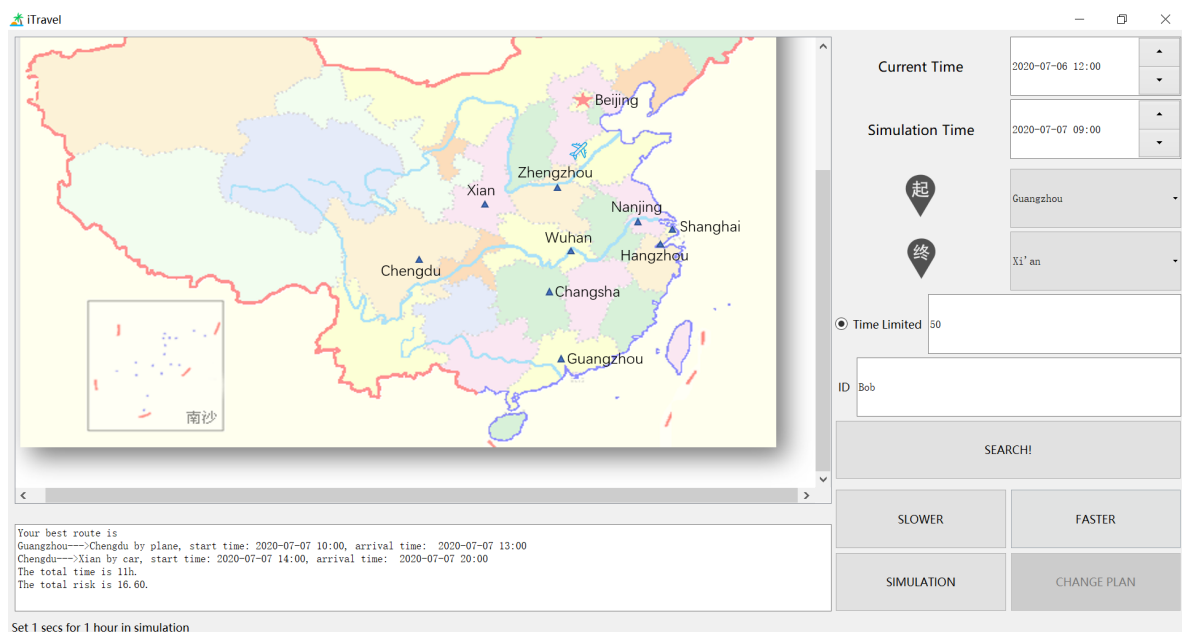
6.2.4测试改变计划

在正在旅行过程中，点击**CHANGE PLAN**之后，画面暂停，可见画面上的飞机。出发城市改为广州，即当前飞机飞往的城市。结果正确。



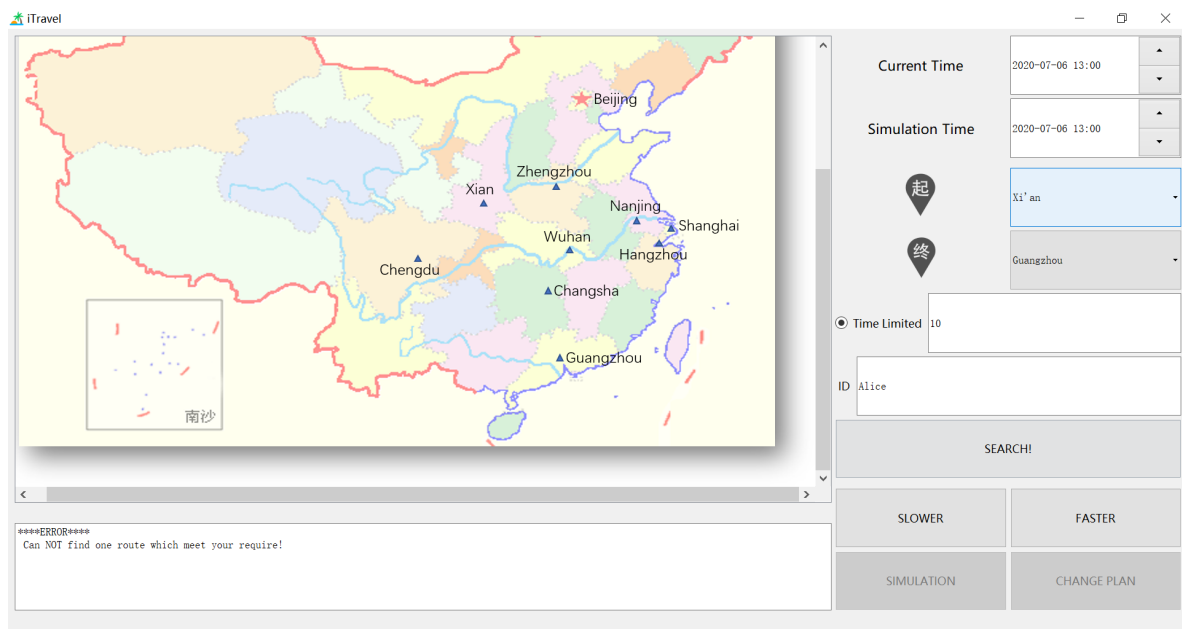
6.2.5测试模拟速度

点击 **SLOWER** 和 **FASTER** 改变事件，可见状态栏有提示。改变后可以发现交通工具的运行事件的变化。结果正确。

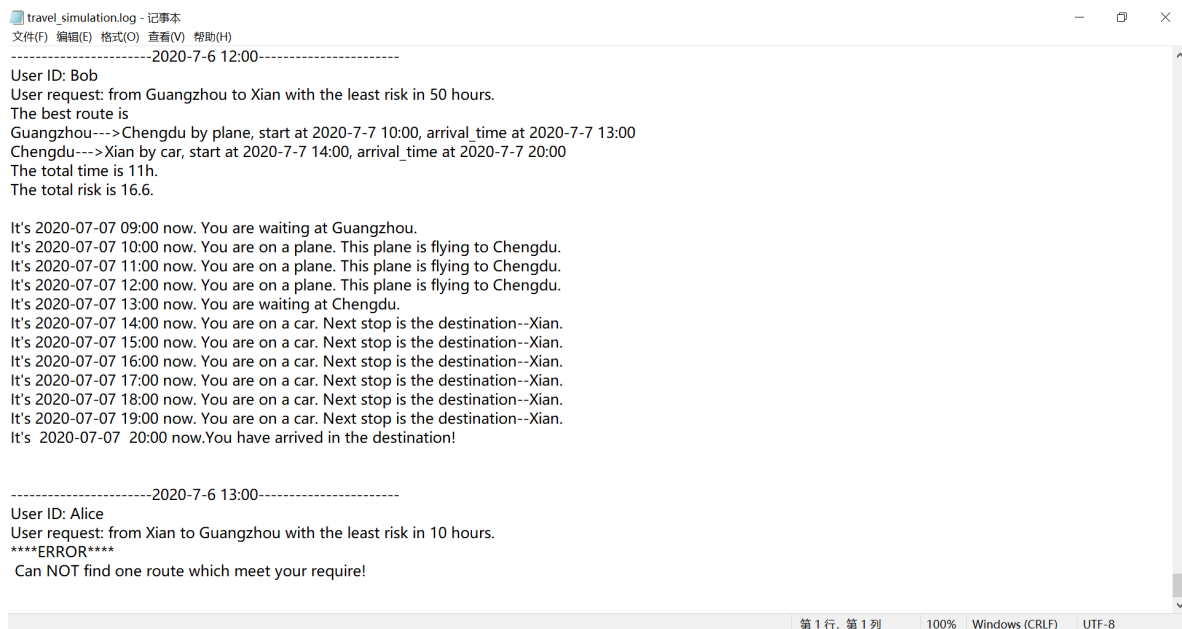


6.2.6测试日志文件

进行两次查询：一次如上图（测试模拟速度中的图片Bob查询Guangzhou-Xi'an限时50小时），一次如下图（Alice查询Xi'an-Guangzhou 限时10小时）中不能得到最佳路径的情况：



可以得到日志文件中最新添加的一部分，可以看到日志文件中当前时间、用户信息、最佳路线、模拟旅行的信息都已写入；如果没有找到最佳路径但用户输入信息正确，也会记录进入日志。



7评价和改进意见

本软件基本完成了一个旅行路线查询和模拟软件的基本功能和自定义的三个附加功能：

1. 实现了11个城市的最小风险策略和限时最小风险策略的旅行路线规划，其中风险包括**交通工具的风险**；
2. 实现了图形界面模拟旅行；
3. 实现了用户自定义准备出发时间；
4. 实现了用户自定义模拟速度；
5. 实现了模拟期间暂停并改变计划；
6. 实现了日志文件记录用户查询和模拟信息。

正确性方面，做到了通过开发者设置的各类测试；易用性方面，软件图标和文字说明清晰，在软件设计过程中给予足够提示，并撰写了用户手册进行说明。

下面详细描述改进方案，目前考虑到的改进点共有以下四点。

7.1精确时间到分钟甚至秒

本软件给此拓展预留了设置，各类结构体定义中设置了minute变量，并在处理时间时也对分钟进行处理，需要做的改变是：

1. 将软件中设置minute为0的代码进行修改，涉及到查找路线模块和时间模块；
2. 将软件中时间模块的两个函数进行修改，考虑分钟。

7.2设置SLOWER和FASTER为滑块

点击的用户体验一般，更改速度慢，改为滑块后可设置为1-100秒的范围让用户滑动滑块进行选择。

7.3设置更为清楚的提示让用户知道可以改变准备出发时间

本软件界面中对用户自定义准备出发时间功能体现不够清晰，在阅读用户手册之前难以发现该功能。可以设置为在模拟旅行之前，通过修改左侧文字说明，让用户得知可以修改；在模拟旅行期间，通过修改左侧文字说明，让用户得知不可修改。

7.4改进旅行模拟中交通工具状态改变算法

本软件的交通工具状态改变算法正确，但效果仍值得改进。该算法的核心在于计算好每次更新间隔需要改变的距离，进而在每次更新间隔来临时根据距离进行改变。理论上正确，但在实际中，由于更新间隔很短（默认每隔50ms更新一次），此时计算机的计算时间也不得不进行考虑，实际中的更新间隔要高于50ms，所以会导致最终到达目的城市时交通工具的位置和目的城市的坐标有微小差别。

目前考虑到的改进方案是将系统的时间精确到分钟后，可以以当前系统时间和交通工具到达的时间之差和当前交通工具的坐标和交通工具到达城市的坐标之差为计算依据每次重新计算坐标，不再使用先前固定每次更新的距离的方法。