疫情下的低风险旅行模拟系统

目录

1. 任务描述

1.1. 用户需求分析

在 COVID-19 疫情的影响下，人们乘坐交通工具出行的健康风险大大增加。这些风险来自不同的交通工具和城市环境，并且不同的交通工具和城市都会有不同的风险程度。对于人们来说，出行时综合考虑时间，城市风险，交通工具风险，以及交通时刻表是一个很复杂的问题。“COVID-19 疫情下的低风险旅行模拟系统”是针对这个问题的解决方案。

1.2. 基础功能描述

城市之间有各种交通工具（汽车、火车和飞机）相连，有些城市之间无法直达，需要途径中转城市。某旅客于某一时刻向系统提出旅行要求。考虑在当前 COVID-19 疫情环境下，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。系统根据风险评估，为该旅客设计一条符合旅行策略的旅行线路并输出；系统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）。

2. 功能需求说明及分析

- 城市总数不少于 10 个，为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。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。

- 不能太简单，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；

- 旅客的要求包括：起点、终点和选择的低风险旅行策略。其中，低风险旅行策略包括：
- 最少风险策略：无时间限制，风险最少即可
- 限时最少风险策略：在规定的时间内风险最少
- 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)；
- 不考虑城市内换乘交通工具所需时间
- 系统时间精确到小时
- 建立日志文件，对旅客状态变化和键入等信息进行记录
- 选做一：用图形绘制地图，并在地图上实时反映出旅客的旅行过程。
- 选做二：为不同交通工具设置不同单位时间风险值，交通工具单位时间风险值分别为：
- 汽车=2；火车=5；飞机=9。
- 将乘坐交通工具的风险考虑进来，实现前述最少风险策略和限时风险最少策略。

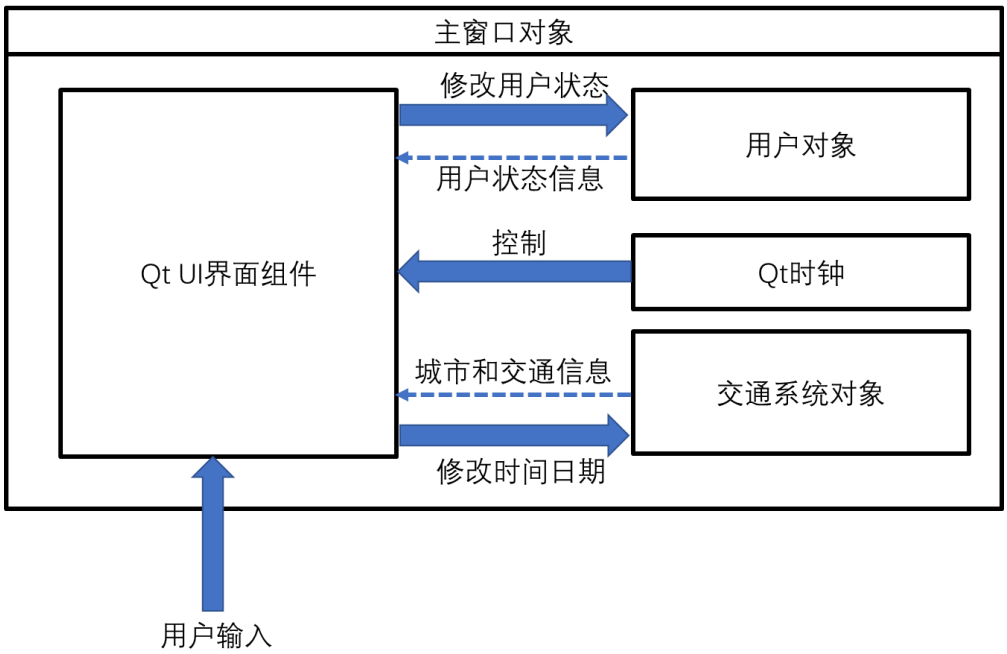
3. 总体方案设计说明

3.1. 软件开发环境

Qt 版本号 5.12.8 msvc2017 x64
Visual studio 2019 C++
Windows10

3.2. 总体结构

3.2.1. 总体结构图

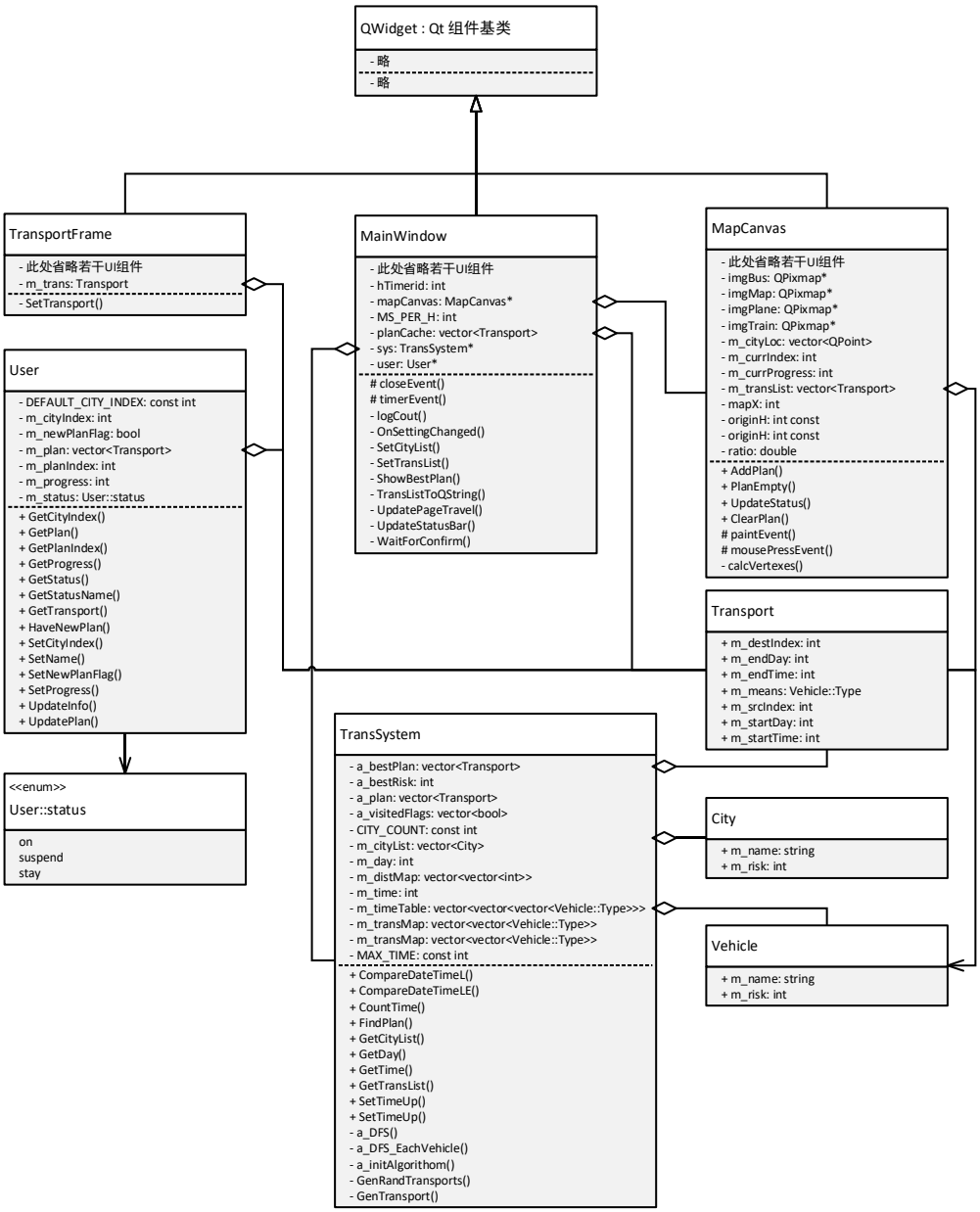


3.2.2. 总体结构说明

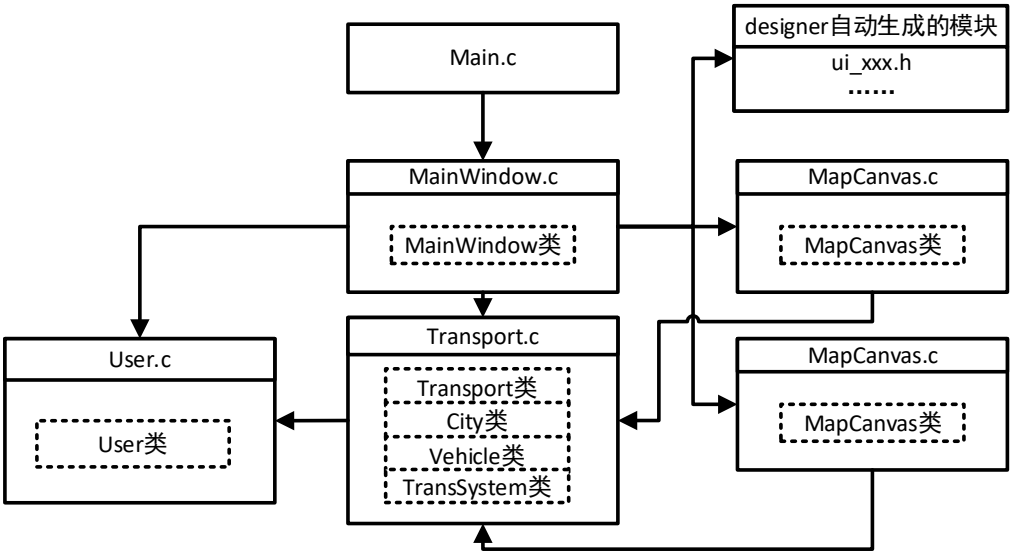
程序结构使用面向对象的思想设计，整个程序是一个窗口对象，该对象主要包括 UI 界面，用户对象，交通系统对象，以及时钟四个部分。这四个部分中只有 UI 界面是用户可以直接操作的。系统接收两个输入，一个是来自用户的输入，由 UI 组件接收；一个是程序内部时钟信号的输入。用户对象和交通系统对象为前端 UI 界面组件提供各种信息，UI 组件接收到输入后对用户和交通系统进行修改。

3.3. 模块划分

3.3.1. 类设计


QWidget : Qt 组件基类
- 略
- 略
TransportFrame
- 此处省略若干UI组件
- m_trans: Transport
- SetTransport()
MainWindow
- 此处省略若干UI组件
- hTimerid: int
- mapCanvas: MapCanvas*
- MS_PER_H: int
- planCache: vector<Transport>
- sys: TransSystem*
- user: User*
closeEvent()
timerEvent()
- logCout()
- OnSettingChanged()
- SetCityList()
- SetTransList()
- ShowBestPlan()
- TransListToQString()
- UpdatePageTravel()
- UpdateStatusBar()
- WaitForConfirm()
MapCanvas
- 此处省略若干UI组件
- imgBus: QPixmap*
- imgMap: QPixmap*
- imgPlane: QPixmap*
- imgTrain: QPixmap*
- m_cityLoc: vector<QPoint>
- m_currIndex: int
- m_currProgress: int
- m_transList: vector<Transport>
- mapX: int
- originH: int const
- originH: int const
- ratio: double
+ AddPlan()
+ PlanEmpty()
+ UpdateStatus()
+ ClearPlan()
# paintEvent()
# mousePressEvent()
- calcVertexes()
User
- DEFAULT_CITY_INDEX: const int
- m_cityIndex: int
- m_newPlanFlag: bool
- m_plan: vector<Transport>
- m_planIndex: int
- m_progress: int
- m_status: User::status
+ GetCityIndex()
+ GetPlan()
+ GetPlanIndex()
+ GetProgress()
+ GetStatus()
+ GetStatusName()
+ GetTransport()
+ HaveNewPlan()
+ SetCityIndex()
+ SetName()
+ SetNewPlanFlag()
+ SetProgress()
+ UpdateInfo()
+ UpdatePlan()
Transport
+ m_destIndex: int
+ m_endDay: int
+ m_endTime: int
+ m_means: Vehicle::Type
+ m_srcIndex: int
+ m_startDay: int
+ m_startTime: int
<<enum>>
User::status
on
suspend
stay
TransSystem
- a_bestPlan: vector<Transport>
- a_bestRisk: int
- a_plan: vector<Transport>
- a_visitedFlags: vector<bool>
- CITY_COUNT: const int
- m_cityList: vector<City>
- m_day: int
- m_distMap: vector<vector<int>>
- m_time: int
- m_timeTable: vector<vector<vector<Vehicle::Type>>>
- m_transMap: vector<vector<Vehicle::Type>>
- m_transMap: vector<vector<Vehicle::Type>>
- MAX_TIME: const int
+ CompareDateTimeL()
+ CompareDateTimeLE()
+ CountTime()
+ FindPlan()
+ GetCityList()
+ GetDay()
+ GetTime()
+ GetTransList()
+ SetTimeUp()
+ SetTimeUp()
- a_DFS()
- a_DFS_EachVehicle()
- a_initAlgorithom()
- GenRandTransports()
- GenTransport()
City
+ m_name: string
+ m_risk: int
Vehicle
+ m_name: string
+ m_risk: int

3.3.2. 文件模块与包含关系



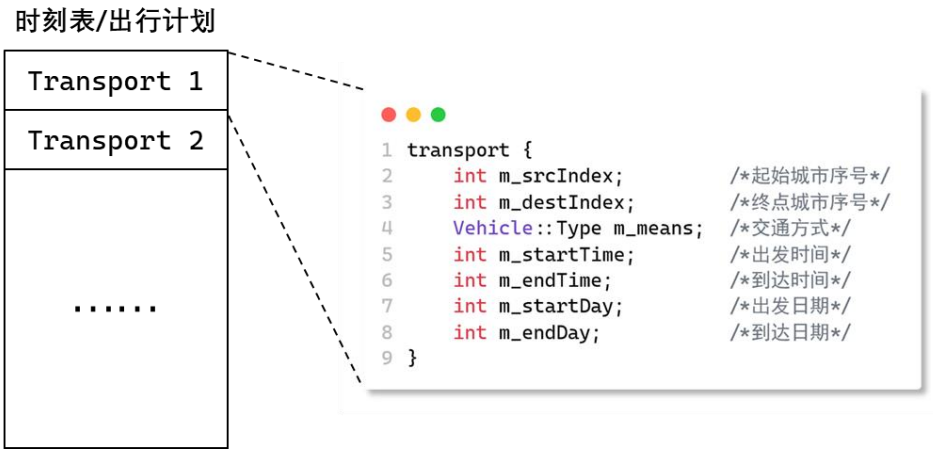
4. 数据结构说明和数据字典

4.1. 交通相关的数据结构说明和字典

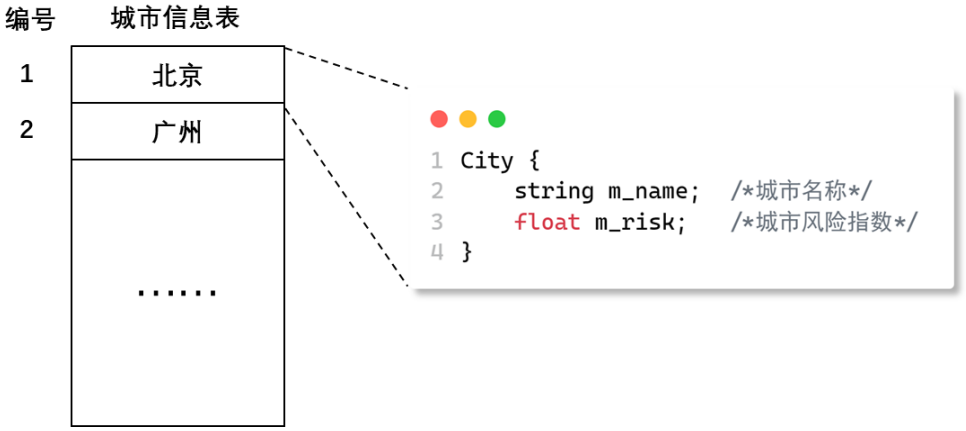
此处的设计考虑多个方面的因素，在保证时刻表的复杂性，合理性的前提下，根据现实情况进行适当的简化以提高灵活度和减少手工工作量。在此模拟系统中，每 24 小时的时刻表都相同，该时刻表基于给定的某些参数随机生成（详情见 5.2.3.3），实际运行中通过一些巧妙的算法同步时间日期，最终使模拟系统可以以无限的时间日期运行。

4.1.1. 航班/车次 Transport

是计划表/时刻表中最基本的单位，表示两个城市之间直达的一次航班/车次。
时刻表/计划表：一个元素为 Transport 的数组：vector<Transport>。



4.1.2. 城市 City



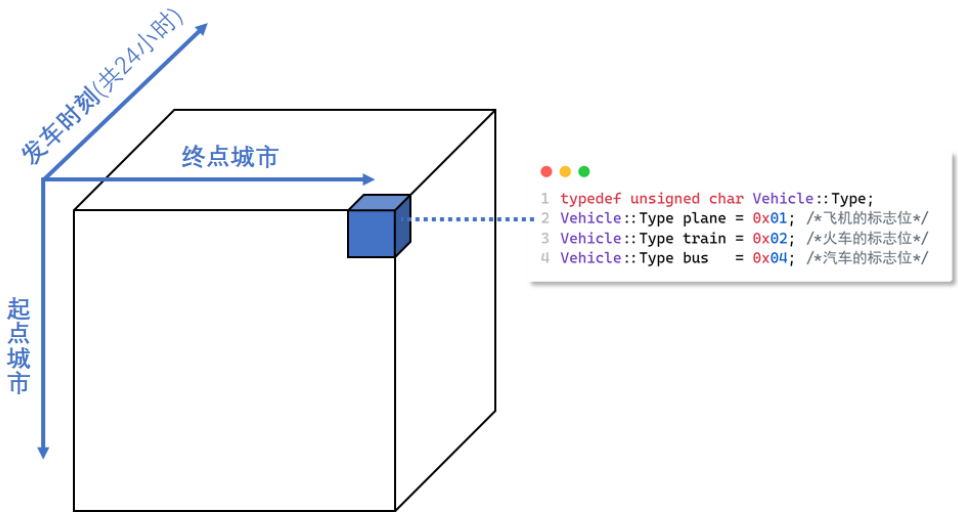
系统中的城市信息表只有一个，作为 TransSystem 类的私有成员变量 m_cityList，在该类的构造函数中初始化，可以使用该类封装的 GetCityList()方法获，不可修改。在系统中，城市都用唯一的编号表示，需要获取信息时才从城市信息表中获取。

4.1.3. 城市距离表：二维数组：vector<vector<int>>

系统中的城市距离图只有一个，作为 TransSystem 类的私有成员变量 m_distMap,在该类的构造函数中初始化，没有提供外部接口，在该类中直接使用。使用 m_distMap[srcIndex][destIndex]获取编号为 srcIndex 和 destIndex 的城市之间的距离。

4.1.4. *时刻表：三维数组：vector<vector<vector<Vehicle::Type>>

Vehicle::Type 本质上是一个 unsigned char 类型,即 8bit 的空间。每一个交通方式使用一个标志位表示，飞机，火车，汽车的标志位分别为低 1、2、3 位，其值作为静态常量存储在 Vehicle 类中。使用 m_timeTable [srcIndex][destIndex][time]可以获取从编号为 srcIndex 到 destIndex 的城市之间第 time 小时发车的所有直达交通工具。判断是否有某种交通工具的方法是与运算：if (m_timeTable[i][j][t] & Vehicle::plane)。




```cpp
typedef unsigned char Vehicle::Type;
Vehicle::Type plane = 0x01; /*飞机的标志位*/
Vehicle::Type train = 0x02; /*火车的标志位*/
Vehicle::Type bus   = 0x04; /*汽车的标志位*/
```

4.1.5. 交通系统数据说明

```cpp
TransSystem {
private:
    vector<vector<vector<Vehicle::Type>>>   m_timeTable;    /*每日时刻表*/
    vector<vector<Vehicle::Type>>           m_transMap;     /*交通方式表,表示两点之间存在的交通方式*/
    vector<vector<int>>                     m_distMap;      /*距离表,表示城市之间的距离*/
    vector<City>                            m_cityList;     /*城市信息表*/
    int                                     m_time;         /*系统当前时间*/
    int                                     m_day;          /*系统当前日期*/

    static const int                        MAX_TIME = 24;  /*每日时间为24小时*/
    static const int                        CITY_COUNT = 12;/*一共有12个城市*/

    vector<Transport>                       a_plan;         /*算法模块:全局变量,当前遍历的计划*/
    vector<Transport>                       a_bestPlan;     /*算法模块:最优计划*/
    vector<bool>                            a_visitedFlags; /*算法模块:城市是否遍历过的*/
    int                                     a_bestRisk;     /*算法模块:当前最优解的累计风险,-1表示无解*/
}
```

4.2. 用户数据说明

```cpp
User{
public:
    enum class          status                  { on = 0, suspend = 1, stay = 2 };
private:
    status              m_status;               /*当前状态*/
    int                 m_cityIndex;            /*当前所在城市*/
    int                 m_planIndex;            /*当前进行的行程在计划中的位置*/
    vector<Transport>   m_plan;                 /*当前正在执行的出行计划*/
    bool                m_newPlanFlag;          /*当前是否申请了新的计划*/
    int                 m_progress;             /*当前进度*100*/
};
```

5. 各模块(类)设计说明

5.1. User.c (User 类)

用户类，包含用户数据以及对用户数据的操作。

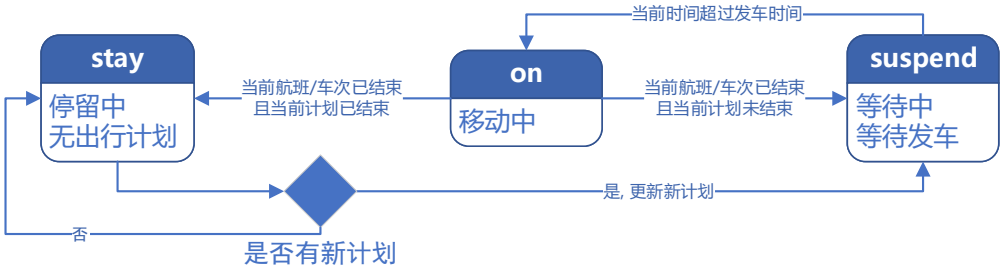
5.1.1. 成员变量介绍 (见 4.2)

5.1.2. 成员函数介绍

```
User{
    const string                GetStatusName ()const;      /*获取状态名称*/
    const User::status          GetStatus() const;          /*获取状态*/
    const Transport&            GetTransport() const;       /*获取当前正在执行的航班*/
    const vector<Transport>&    GetPlan() const;            /*获取当前计划*/
    int                         GetProgress() const;        /*获取当前进度*/
    int                         GetCityIndex() const;       /*获取当前城市或正在离开的城市*/
    int                         GetPlanIndex() const;       /*获取当前行程在计划中的编号*/
    const                       bool HaveNewPlan() const;   /*查询用户是否有新的计划待接收*/

    void        SetCityIndex(int cityIndex);                /*修改用户当前的城市*/
    void        SetNewPlanFlag(bool flag);                  /*修改新计划的标志*/
    void        SetProgress(int progress);                  /*修改当前航班/车次进度*/
    void        UpdatePlan(const vector<Transport>& plan);  /*为用户添加新计划*/
    void        UpdateInfo(int time, int day);              /*根据时间,日期,状态,更新用户信息*/
};
```

5.1.3. 用户状态机



5.2. Transport.c

5.2.1. Vehicle 类

该类只提供静态方法和静态常量，并不用于构造对象。该类的作用是提供关于各种交通工具的类型和信息。车辆的属性包括名字，固定的发车间隔，单位距离需要的时间，每小时风险值。

```cpp
class Vehicle {
public:
    typedef unsigned char Type;      /*定义交通工具类型*/
    struct Attribute {               /*定义交通工具属性类型*/
        string  name;                /*交通工具名字*/
        int m_interval;              /*两次航班的固定间隔*/
        int m_distTimes;             /*单位距离需要的时间*/
        int m_risk;                  /*每小时风险值*/
    };
    static const Type                bus    = 0x01;              /*定义汽车*/
    static const Type                train  = 0x02;              /*定义火车*/
    static const Type                plane  = 0x04;              /*定义飞机*/
    static const Type                all    = bus | train | plane;/*定义所有交通工具*/
    static const Vehicle::Attribute  GetAttribute(Type);         /*获取交通工具属性*/
}
```

5.2.2. City 类: 见 4.1.2

5.2.3. *TransSystem 类

交通系统类，包含整个系统的客观环境信息，包括城市列表，距离表，时刻表，时间，日期等信息。内置算法模块提供计算最佳路径的接口，提供同步时间日期的接口。

5.2.3.1. 成员变量介绍(见 4.1.5)

5.2.3.2. 成员函数介绍

```cpp
class TransSystem {
public:
    int                     GetTime() const;                                        /*获取当前时间*/
    int                     GetDay() const;                                         /*获取当前日期*/
    const vector<City>&     GetCityList() const;                                    /*获取城市列表*/
    const vector<Transport> GetTransList(int srcIndex, int destIndex, Vehicle::Type means)const;/*获取两个城市间的时刻表*/
    void                    SetTimeUp();                                            /*时间增加一小时*/
    static const int        CountTime(int startTime, int endTime, int startDay, int endDay);    /*计算两个时间日期之间间隔的总时间*/
    static const bool       CompareDateTimeLE(int timeA, int timeB, int dayA, int dayB);        /*比较时间日期大小,返回A≥B*/
    static const bool       CompareDateTimeL(int timeA, int timeB, int dayA, int dayB);         /*比较时间日期大小,返回A>B*/
    const vector<Transport> FindPlan(                                               /*根据不同的策略寻找最小风险计划*/
                                int srcIndex, int destIndex,
                                int startDay, int endDay,
                                int startTime, int endTime,
                                bool repeat, bool limTime, bool enableTransRisk
                            );
private:
    void                    GenRandTransports(Vehicle::Type means, int srcIndex, int destIndex);/*根据交通工具的属性和距离表,
                                                                                                  生成给定城市间的时刻表*/
    const Transport         GenTransport(                                           /*封装一个transport对象并返回*/
                                int startTime, Vehicle::Type means,
                                int srcIndex, int destIndex, int startDay = 0
                            ) const;
    /*以下为算法模块*/
    void                    a_InitAlgorithm(int srcIndex);                          /*根据起始城市初始化算法相关变量*/
    void                    a_DFS(                                                  /*算法核心:DFS递归函数*/
                                int srcIndex, int destIndex,
                                int startDay, int endDay,
                                int startTime, int endTime,
                                int riskBefore,
                                bool enableRepeat, bool enableLimTime, bool enableTransRisk
                            );
    bool                    a_DFS_EachVehicle(                                      /*a_DFS的一个组件,详情见代码*/
                                int srcIndex, int destIndex,
                                int startDay, int endDay,
                                int startTime, int endTime,
                                int riskBefore,
                                bool enableRepeat, bool enableLimTime,bool enableTransRisk,
                                int time, int cityi, int riskAfter, Vehicle::Type v);
};
```

5.2.3.3. *时刻表生成算法介绍

生成 4.1.4 中所述的时刻表，需要用到的参数有：交通工具及其属性(5.2.1)，城市距离表(4.1.3)，以及一个增加一个辅助的二维数组 m_transMap,元素类型为 Vehicle::Type,即交通工具类型。m_transMap[srcIndex][destIndex]表示从城市 srcIndex 到 destIndex 是否存在直达航班。

在介绍该算法前，先声明几个假设条件：

1. 每天(24 小时)的时刻表完全相同。
2. 每种交通工具拥有固定的发车间隔。
3. 对于每个城市来说，只有拥有某一种交通工具，那么他们每天的发车间隔、发车次数相同，只有时间不同，该时间由模拟程序随机生成。

算法描述: （伪代码）

```
for (src, dest) {   /*遍历所有起始终点城市的组合*/
    for (vehicle) { /*遍历所有交通方式*/
        if (m_transMap[src][dest] & vehicle) { /*如果这两个城市之间有该交通方式*/
            startTime = rand() % vehicle.intervel; /*随机计算第一班车次的发车时间*/
            while (startTime < 24) { /*如果发车时间在0-23小时*/
                m_timeTable[src][dest][startTime] ⊨ vehicle; /*将该车次添加到时刻表中*/
                startTime += vehicle.intervel;  /*计算下一个车次的发车时间*/
            }
        }
    }
}
```

5.2.3.4. *寻找最低风险算法介绍

该算法使用 DFS 算法，目的是找到一个最小风险的 transport 序列。设停留在城市 C 的单位时间的风险为 R[c]，交通工具 v 的单位时间风险是 R[v]，第 i 个 transport 为 T[i]

选择第 i 个 transport 的总风险函数如下:

1. 不考虑交通工具风险。

$$R(i) = R(i-1) + (T[i].startTime - T[i-1].endTime) * R[c]$$

2. 考虑交通工具风险。

$$R(i) = R(i-1) + (T[i].startTime - T[i-1].endTime) * R[c] + T[i].time * R[v]$$

以上公式构成了 DFS 的基本框架，接下来介绍剪枝策略：

1. 由于每 24 小时的车次相同，所以当 $(T[i].startTime - T[i-1].endTime) > 23$ 时不再往下搜索。
2. 累积风险超过当前最优解的风险时，不再往下搜索。

3. 当用户选择限时策略时，累计时间超过用户期望到达时间则不再往下搜索。
4. 当用户选择不允许重复访问策略时，如果当前 transport 的终点已访问过，则不再往下搜索。

算法伪代码和描述如下：以“限时最小风险，允许重复访问”策略为例

```
DFS(src, dest, startTime, endTime, riskBefore) {
    if (dest == src) return;                    /*起点和终点相同,返回*/
    for (time = 0; time < 24; ++time){          /*中转停留时间从0到23*/
        risk = riskBefore + time * dest.risk;   /*添加停留风险*/
        if(risk ≥ bestRisk) return;             /*如果风险≥最优风险,返回*/
        for (city, vehicle) {                   /*遍历所有城市和交通工具*/
            if(timeTable[src][city][startTime+time] & vehicle){     /*如果有从该时刻开始的车次*/

                tStartTime = startTime + time;                      /*计算发车时间*/
                tEndTime = tStartTime + v.speed*distMap[src][dest]; /*计算到达时间*/
                if(tEndTime > endTime) continue;                    /*加入该transport后超时则不加入*/
                transport = {tstartTime, tEndTime, src, city}       /*生成transport*/
                plan.add(transport);                                /*添加transport*/

                if(city == destIndex) {                             /*如果加入后到达目的地*/
                    bestPlan = plan;                                /*更新最佳计划*/
                    bestRisk = risk;                                /*更新最佳风险*/
                    plan.pop();                                     /*弹出该方案*/
                    return;                                         /*不再搜索从src出发的任何车次*/
                }else{                                              /*否则*/
                    DFS(city, dest, tEndTime, endTime, risk);       /*往深处搜索*/
                }
            }
        }
    }
}
```

详细代码和注释见 TransSystem 中 a_开头的成员。

5.3. UI 模块。

UI 模块包括用代码写的 MainWindow.h 主窗口模块，MapCanvas.h 地图动画模块，TransportFrame.h 车票模块，用 Qt designer 设计生成的 MainWindow.ui 主窗口模块，SettingWindow.ui 设置窗口模块，TransportFrame.ui 车票模块，MapCanvas 地图模块。由于时间原因，繁琐的 ui 界面难以逐一介绍，用户交互逻辑和界面布局请查阅下文的用户使用说明，成员变量和成员函数请查阅 3.31 的 UML 类图。详细代码和注释请查阅源文件。

6. 范例执行结果及测试情况说明

6.1. 当前软件内置的城市列表数组如下：

城市	成都	西安	太原	北京	沈阳	长沙
编号	0	1	2	3	4	5
城市	武汉	合肥	济南	广州	福州	杭州
编号	6	7	8	9	10	11

6.2. 当前软件内置的城市距离图如下：

	成都	西安	太原	北京	沈阳	长沙	武汉	合肥	济南	广州	福州	杭州
成都	0	2	4	5	7	3	3	4	5	5	6	6
西安	2	0	2	3	5	3	2	3	4	5	5	5
太原	4	2	0	2	3	4	3	3	2	6	5	4
北京	5	3	2	0	2	5	4	3	2	6	5	4
沈阳	7	5	3	2	0	6	5	4	2	7	6	5
长沙	3	3	4	5	6	0	1	4	4	2	2	3
武汉	3	2	3	4	5	1	0	1	3	2	2	2
合肥	4	3	3	3	4	4	1	0	2	3	2	1
济南	5	4	2	2	2	4	3	2	0	5	4	3
广州	5	5	6	6	7	2	2	3	5	0	2	3
福州	6	5	5	5	6	2	2	2	4	2	0	1
杭州	6	5	4	4	5	3	2	1	3	3	1	0

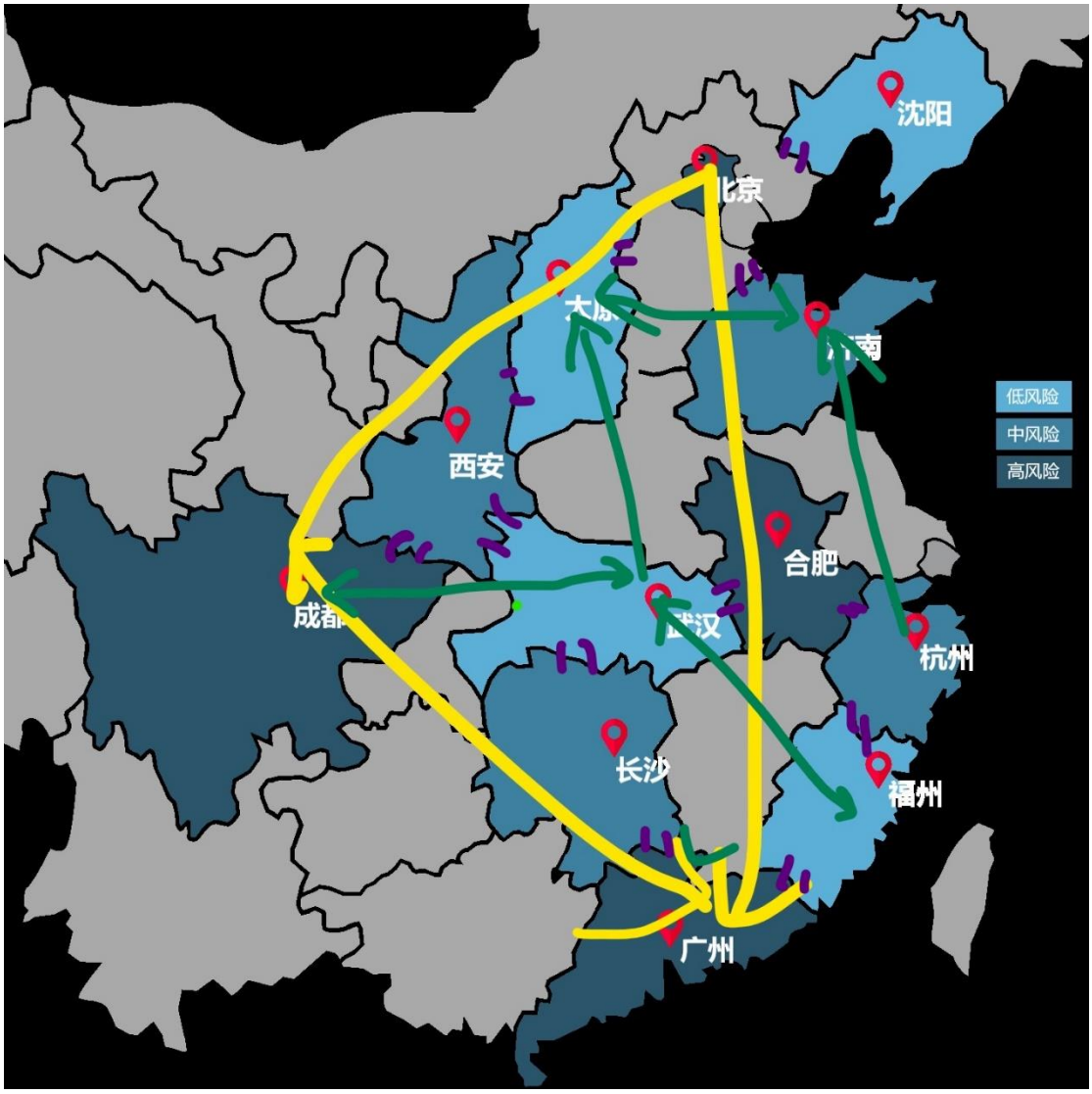
6.3. 当前交通工具属性如下：

名字	代码	发车间隔	单位里程的时间	风险值
飞机	0x04	12	1	9
火车	0x02	8	2	5
汽车	0x01	6	4	2

6.4. 当前 m_transMap(表示两座城市间存在的交通方式的标志位)如下：

	成都	西安	太原	北京	沈阳	长沙	武汉	合肥	济南	广州	福州	杭州
成都	0	1	0	4	0	0	2	0	0	4	0	0
西安	1	0	1	0	0	0	1	0	0	0	0	0
太原	0	1	0	1	0	0	2	0	2	0	0	0
北京	0	0	1	0	1	0	0	0	1	4	0	0
沈阳	0	0	0	1	0	0	0	0	0	0	0	0
长沙	0	0	0	0	0	0	1	0	0	1	0	0
武汉	2	1	2	0	0	1	0	1	0	0	2	0
合肥	0	0	0	0	0	0	1	0	0	0	0	1
济南	0	0	2	1	0	0	0	0	0	0	0	2
广州	4	0	0	4	0	1	0	0	0	0	1	0
福州	0	0	0	0	0	0	2	0	0	1	0	1
杭州	0	0	0	0	0	0	0	1	2	0	1	0

该表格可视化草图如下：黄色，绿色，紫色分别表示飞机，火车，汽车。



6.5. 测试方法和测试结果

由于城市数量有限，且每 24 小时的时刻表相同，所以我们可以规律的方法测试完几乎所有情况。对于任意源和目的地的组合(共 12*11 种), 使用所有可选的策略组合，设置起始时间为 0 到 23 的所有时间，对最优解进行搜索。

测试结果：所有样例通过测试，算法可以立刻响应生成结果。

7. 评价和改进意见

个人对这一次的大作业比较满意，代码有着良好的命名，较为良好的面向对象设计，以及很好的扩展性能和易读性。算法设计也兼顾了灵活和效率，拥有良好的鲁棒性，可供用户自由设置。课程设计要求的任务已经全部完成。

所有的不足都来自扩展部分，原本计划完成的一些功能因为时间问题没有完成，以至于该模拟程序离商用程序还有不小的差距。如：没有完成更随机的时刻表生成算法，没有完成软件内部的用户操作教程，没有完成可动态缩放、可后台增删城市的地图。并且，对于设计模式不熟悉导致了对一些类的设计一再改动，这些都是需要改进的地方。

8. 用户使用说明。

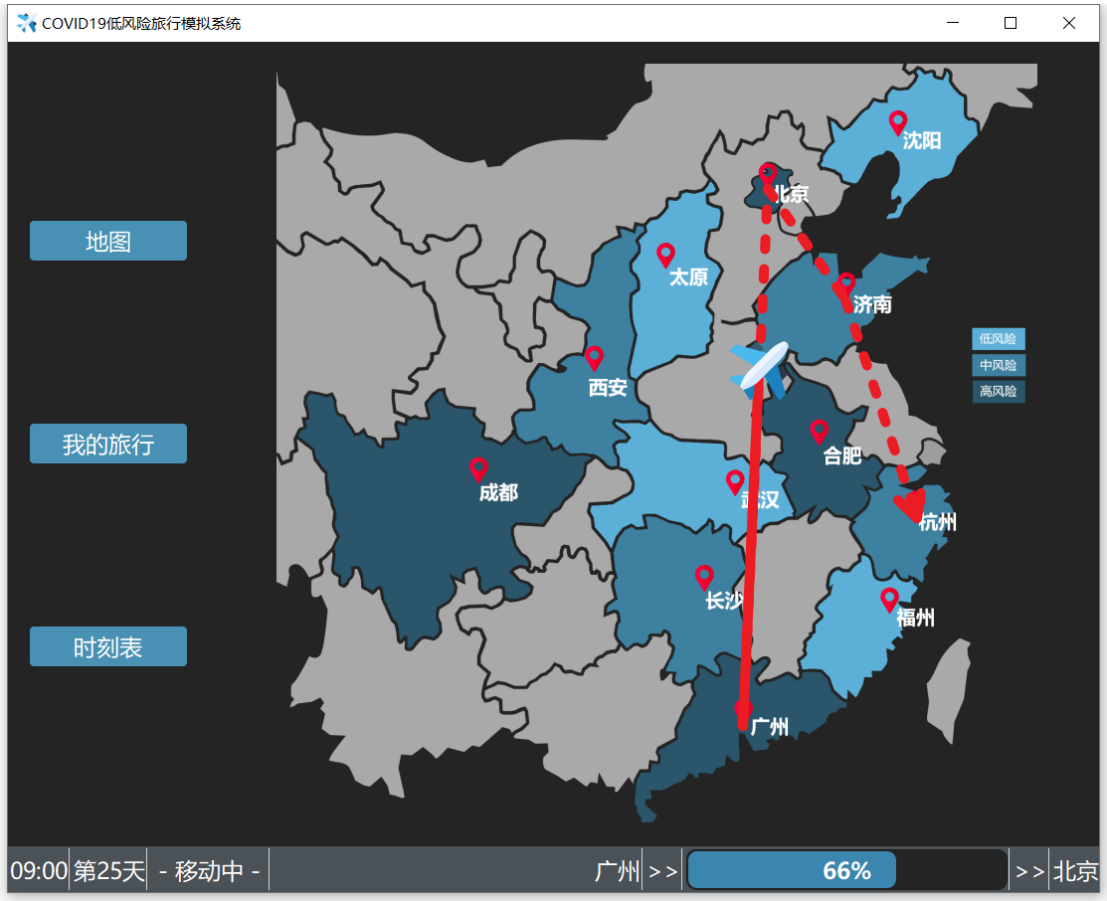
8.1. 进入程序：

8.2. 主界面布局介绍：

8.3. 状态栏介绍：

8.4. “地图”页介绍

在地图页会显示系统中存在的城市，不同区域的风险程度使用不同的色块来表示。带箭头的红色虚线为用户当前计划的路线，该路线上的交通工具图标表示用户所处的位置和乘坐的交通工具。



8.5. “我的旅行”页介绍

8.5.1. 查看我正在进行的计划。

8.5.2. 搜索新计划并添加计划。

注意：用户点击查询计划且查询成功后，系统时间会暂停并等待用户做出决策。在这个过程中，切换页面的按钮被禁用。用户允许的操作包括：添加计划，取消(放弃)计划，重新选择起点、终点、时间、重新查询。

8.6. “时刻表”页介绍