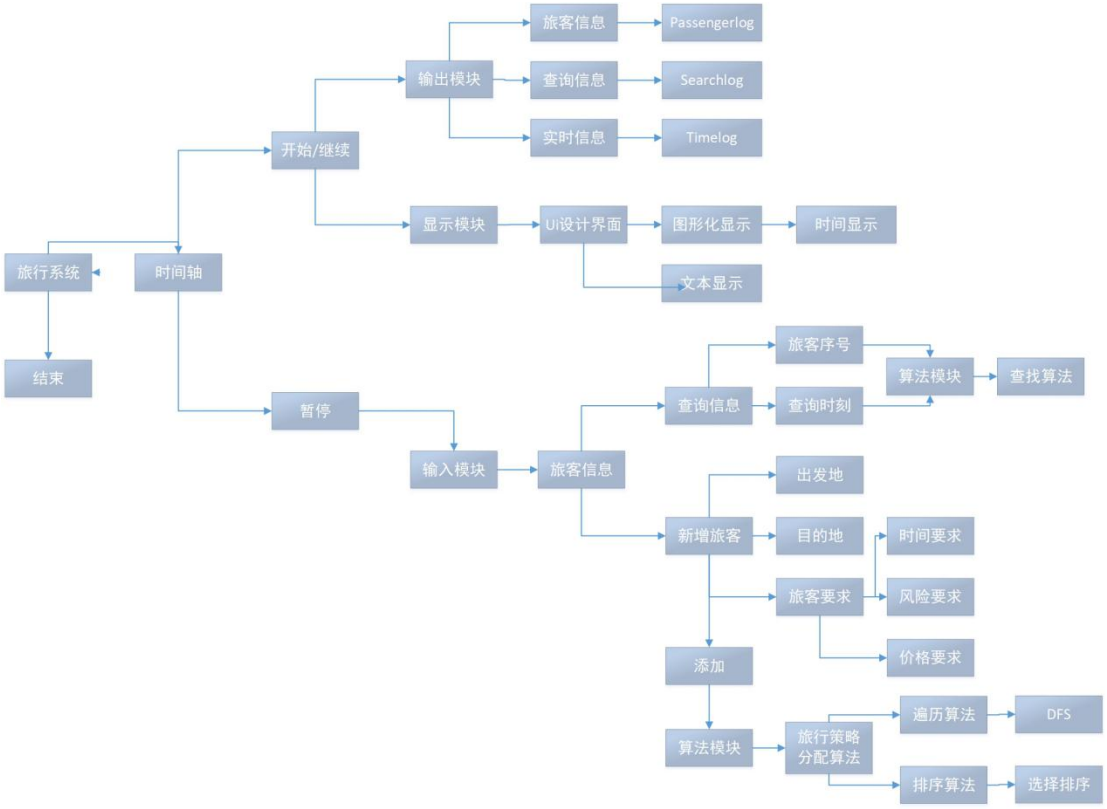
旅行模拟系统— — 各模块设计说明

Travel_System

一、程序整体设计简述

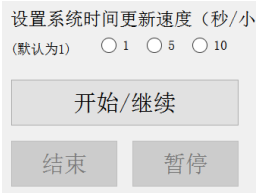


上图为程序的完整模块划分关系图，可以得出程序主要分为时间轴模块，输入模块，输出模块，显示模块以及算法模块。

二、各模块设计说明

（一）时间轴模块

本模块由两个 QPushButton 按钮控制，“开始/继续”按钮控制时间的开始推进与继续推进操作，“暂停”按钮可使时间暂停推进，从而进行增加旅客或查询等操作。



① 算法

本模块利用 QTimer 设定一个定时器，每次定时器超时的时候就对时间值进行加一操作。从而推进时间的进行，

暂停操作会让定时器停止。

② 特点

可控性强，修改一个参数即可改变定时器的值，从而改变使得时间值增加的时间间隔，可随时通过点击“开始/继续”，“暂停”按钮来控制时间的推进。

③ 与其他模块的关系

本模块是显示模块的基础，因为显示模块的刷新时通过时间值的更新实现的，每次显示模块更新显示时，都需要对时间值进行读取，再找到相应的旅客状态进行显示。

本模块与输出模块联系也十分密切，输出信息中时间信息是必不可少的，关于本模块在输出模块的应用，到输出模块处进行介绍。

（二）显示模块

本程序的显示模块主要依靠 Qt 集成开发环境的 ui 界面设计完成。（如下图）

其中文本输入框均由 QTextEdit 控件完成，文本输出框均由 QTextBrowser 控件完成，图形化以及文字输出界面由 QLabel 构成，按钮均为 QPushButton 控件，时间由 QLCDNumber 控件实现。

① 算法

显示界面主要分为图形化显示和文本消息显示两类，这两类均在程序运行界面显示信息。

图形化界面是程序运行界面左上角可视的部分中国地图，地图中标有三种不同风险值的 12 所城市。图形化界面在时间推进的同时，每次定时器超时进行一次更新显示，通过算法模块的查找算法遍历每个旅客在对应时间值的状态信息并进行显示。

图形化界面上遍布着 QLabel 控件，用来显示处于城市以及处于城市之间的

交通工具上的旅客状态，QLabel 控件上以及通过 QPainter 画好了任意邻接城市的线路，开始时所有控件 setVisible（false），也即是不可见。当读取到旅客状态时，则将对应的 QLabel 控件设置为 setVisivle（true），由此实现了图形化显示实时旅客状态信息。

文本消息显示在程序运行界面有三个控件，分别为查询显示，实时消息显示和策略方案显示。

查询显示仅在使用查询功能时才进行显示，策略方案显示在执行新增旅客功能时（点击添加后）进行显示，实时消息的显示机制类似于图形化界面的显示，均是通过读取到的时间值显示旅客的状态信息，有所不同的是，实时消息显示 界面还会显示增添旅客信息和报错信息。

② 特点

主要显示模块均与时间相关联，随着时间的推进不断更新。

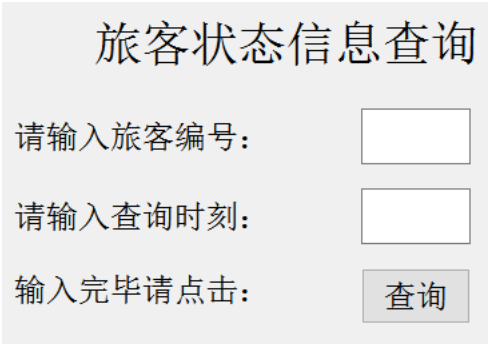
③ 与其他模块的关系

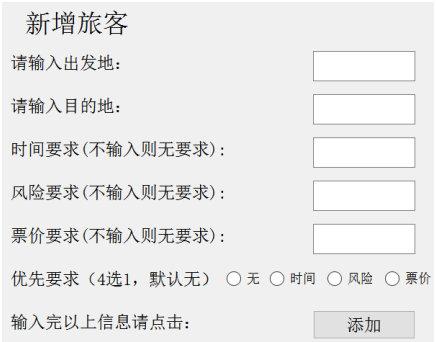
图形化显示界面和实时消息显示界面需借助算法模块的查询功能中的查找算法读取旅客在对应时间的信息。

本模块与时间轴模块关系最为紧密，本模块除了查询显示模块均由时间轴模块驱动。

（三）输入模块

本模块由新增旅客模块和查询模块组成。（如下图）





本模块在用户输入完相应信息后点击“查询”，“添加”按钮即可完成对信息的输入。

① 算法

本模块并未使用算法，只是将输入的 QString 类型变量转换成对应数据类型变量。

在本模块设置了检错机制，所有用户引起的错误均会在此模块被检测出来，如用户将出发地，目的地设为同一城市或输入了非法数据等。程序均会检测到并在实时消息界面显示报错信息以提醒用户检查并修正输入。

② 特点

所有数据均以 QString 类型读入，在程序内部转化为对应数据类型。

鲁棒性，检错性强，检验输入的错误。

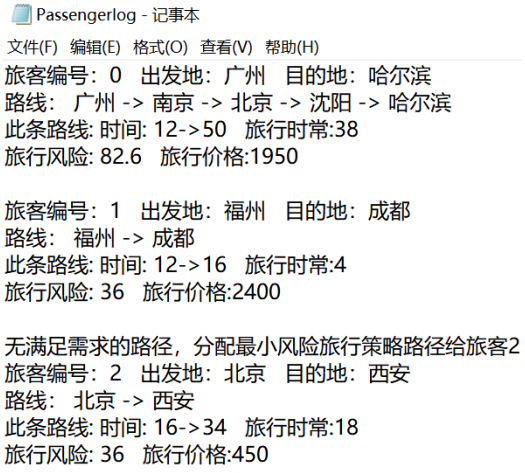
③ 与其他模块的关联

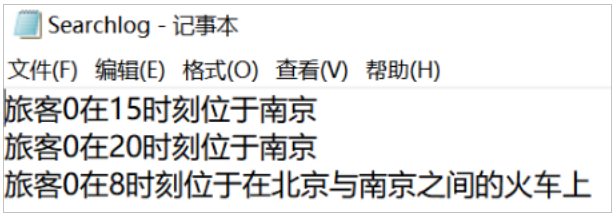
本模块通过给算法模块相关联，为算法模块提供旅客的数据，从而进行后续的旅行策略分配等算法。

本模块输入的数据会被除时间轴模块以外所有的模块利用，因此必须确保本模块的数据是可靠的。

（四）输出模块

本模块输出三个日志文件“Passengerlog.txt”，“Timelog.txt”，“Searchlog.txt”到工程文件目录，分别记录旅客的增添信息，实时消息信息，查询信息。下面分别为三个日志文件记录信息的示例：





三个日志文件通过 Qt 集成开发环境中的 QFile 功能进行操作，每次程序运行清空日志文件，在程序运行过程中不断对日志文件进行 append 操作。

```
QFile filepas("Passengerlog.txt");
QTextStream outp(&filepas);

QFile filetime("Timelog.txt");
QTextStream outt(&filetime);

QFile filesearch("Searchlog.txt");
QTextStream outs(&filesearch);
```

```
//每次打开程序清空log文件
filepas.open(QIODevice::WriteOnly);
filepas.close();
filetime.open(QIODevice::WriteOnly);
filetime.close();
filesearch.open(QIODevice::WriteOnly);
filesearch.close();
```

（声明和清空操作）

```
filetime.open(QIODevice::Append);
outt << nowtime << Timer_Time << lvkexuhao << temp->Seq << nowstate << d << endl;
filetime.close();
```

（写操作示例）

① 算法

本模块并无算法。

② 特点

记录所有有用信息，供用户进行查看。

③ 与其他模块的关联

本模块被嵌入在输入模块中，在输入完成，由算法模块分配的旅行策略产生后，旅客信息就被输出到 Passengerlog.txt 中；在查询完成后，查询输入的信息以及由查找算法返回的旅客状态信息会被写入 Searchlog.txt 中。

本模块被键入在显示模块中，随着时间值的增加，每次计时器产生超时信号就会将实时消息显示模块中的旅客信息记录到 Timelog.txt 中。

（五）算法模块

此模块是本程序中最主要的模块，它是实现程序功能的基础。

本程序的算法主要可分为旅行策略分配算法以及查找算法。

① 算法

旅行策略分配算法：

旅行策略分配算法是一个基于深度优先遍历的优化算法。

时间复杂度为 O（n+e），e 为邻接城市数。大致思路如下：

首先利用 DFS 遍历并记录用户设定的出发地与目的地间全部的路径。并记录每一条路径的长度信息（共经过了多少城市）。

其次通过筛选函数筛选掉途经城市比 5 多的路径，保存剩余路径准备进行第二轮筛选。

第二轮根据用户键入的要求将剩余的路径进行筛选，将满足用户要求的路径保存准备进入排序，如无满足用户要求的路径，直接将未经第二轮筛选前的路径进入排序。

在排序过程中找到风险值最小的路径，保存到旅客的信息中。

DFS 查找可能路径伪代码：

```
void Get_All_Path(C* Depart, C* Dest, int City_Num, M* map) {
    for (i = 0; i < Depart->Count_Adj_City; i++) {
        if (Depart->Adj_City[i]->City_State == UNPASSED_CITY) {
            Depart->Adj_City[i]->City_State = PASSED_CITY;
            Depart->next = Depart->Adj_City[i];
            Get_All_Path(Depart->Adj_City[i], Dest, City_Num, map);//递归
            Depart->next = NULL;
        }
    }
    //用 record 数组记录所有可能路径，此数组用于后续筛选与排序
```

```
    for (i = 0; i < Depart->Count_Adj_City; i++) {
        if (Depart->Adj_City[i]->City_State == END_CITY) {
            if (Depart->City_State == START_CITY) {
                record[a][0] = Depart->City_Name;
                record[a][1] = Dest->City_Name;
                count_record_num[a] = 2;
                a++;
                return;
            }
            else {
                Depart->next = Depart->Adj_City[i];
                for (j = 0; j < City_Num; j++) {
                    if (map->Map_City[j]->City_State == START_CITY) {
                        Start = map->Map_City[j];
                    }
                    if (map->Map_City[j]->City_State == END_CITY) {
                        End = map->Map_City[j];
                    }
                }
                Record_All_Path(Start, End);
                Depart->City_State = UNPASSED_CITY;
                Depart->next = NULL;
                return;
            }
        }
    }
    //还原
    for (i = 0; i < Depart->Count_Adj_City; i++) {
        if (Depart->Adj_City[i]->City_State == PASSED_CITY) {
            Depart->City_State = UNPASSED_CITY;
            return;
        }
    }

    for (i = 0; i < Depart->Count_Adj_City; i++) {
        if (Depart->Adj_City[i]->City_State == START_CITY) {
            Depart->City_State = UNPASSED_CITY;
        }
    }
}
```

由于后续筛选是针对数组的操作，只是简单的优化过程，在此不再展示伪代码。

查找算法是基于旅客链表的遍历查找算法。

遍历链表，找到对应的旅客，将旅客数据域的信息

下面给出伪代码

```
QString Search_Pas_State(int Seq, int Time){
//遍历找到对应旅客节点
    while(Pas_ptr->Seq != Seq){
        Pas_ptr = Pas_ptr->next;
    }

    a.append(Pas_ptr->Depart_Place);
    b.append(Pas_ptr->Dest_Place);
//以下代码为根据已经存好的时间间隔数组，和交通工具数组，接合成需要返回的内容
    if(Time < Pas_ptr->Plan.Plan_Time[0]){
        s = "";
        s.append(Pas_ptr->Plan.Plan_City[0]);
        markstate = 1;
        markcity_index = 0;
    }
    else{
        for(i=0; ttime[i] != 0; i++){
            if(Time >= ttime[i] && Time < ttime[i+1]){//现在时间落在此时间间隔内
                if(i % 2 == 1){//旅客状态为在某城市
                    s = "";
                    s.append(city[(i+1)/2]);
                    markstate = 1;
                    markcity_index = (i+1)/2;
                    break;
                }
                else if (i % 2 == 0){//旅客状态为在某两城市间的交通工具上
                    s = "";
                    markstate = 0;
                    s.append("在");
                    s.append(city[i/2]);
                    markajcity_index = i/2;
                    s.append("与");
                    s.append(city[(i/2)+1]);
                    s.append("之间的");
                    if(vehi[i/2] == 1){
                        s.append("飞机");
```

```
                    markvehic_index = i/2;
                }
                else if(vehi[i/2] == 2){
                    s.append("汽车");
                    markvehic_index = i/2;
                }
                else if(vehi[i/2] == 3){
                    s.append("火车");
                    markvehic_index = i/2;
                }

                s.append("上");
                break;
            }
        }
        else if(Time >= ttime[i] && ttime[i+1] == 0){//旅客已经结束行程
            s = "";
            s.append(b);//返回值为旅行行程终点
            markstate = 1;
            markcity_index = (i+1)/2;
            break;
        }
    }
}

while(Pas_ptr->Seq != 0)
    Pas_ptr = Pas_ptr->prev;

return s;
}
```

② 特点

在存在满足用户需求的路径时求满足用户需求的最优解（风险最小），在不存在时替用户选择风险最小的解。

③ 与其他模块的关联

在输入模块中的添加旅客模块调用了本模块的旅行策略分配算法，查询模块调用了查找算法。

在显示模块和输出模块均调用了查询算法。