# 1．用户界面设计

## 1.1 图形界面

图1：

****

此为程序开始最初界面，有“手动输入”和“文件配置输入”两个选择，手动输入会自动配置好最佳参数选择，可以通过箭头达到修改的目的，文件配置以达到读取参数的目的。下面分别介绍两个选项界面。

图2：



图3



此图为手动输入界面，首先是“蛇形队列最大等待行数”有四个挡位分别为 1 3 5 7，我们认为一次增开两列缓冲区便于乘客行走才符合现实逻辑。其次是“最大允许等待长度”用于判断是否增开队列，同时我们设置了不能超过最大上限和不能超过“安检口队伍最大长度”的设定，然后便是两个“最大最小”，在这里我们设定了“最小”不能超过“最大”的规定，当“最小”增大到超过“最大”的时候，“最大”也随之增大，保持“最小”不会超过“最大” ， 反之，“最大”减小到要小于“最小”时，“最小”也会随规定减小。再然后便是下班时间，通过设置可以预先决定下班时间（同时程序内也能手动下班）。关于乘客的两个参数，“自动产生乘客”会由正态分布随机乘客，“不自动产生乘客”会有相应的按钮手动输入（此按钮在自动产生时同样有效），乘客速度有五档，1—5分别代表不同的速度。可以预先设定，也可以在程序内手动调节，此为程序一大特色。选择完毕以后点击确认键便可转到运行界面。（程序支持混合输入）

图4：



此图为选择文件配置以后的界面，我们将输出文件配置参数，有五秒时间供用户阅读。随后便转到运行界面。

图5：

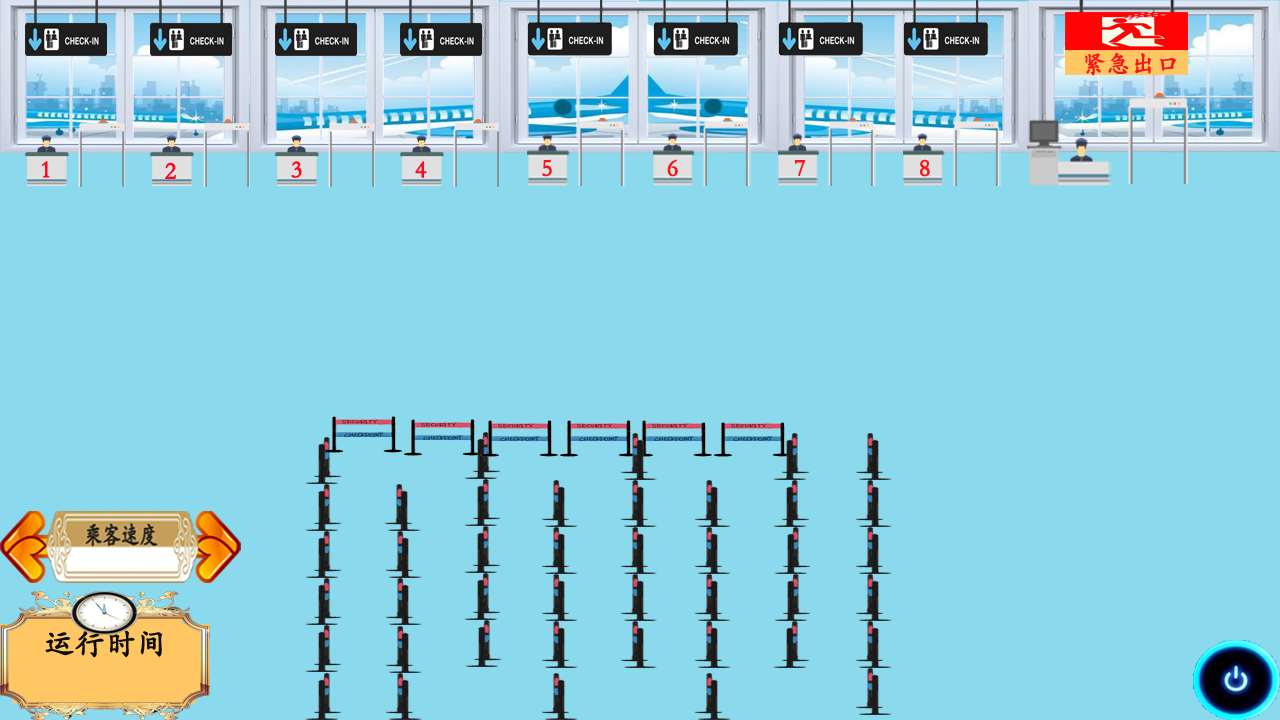
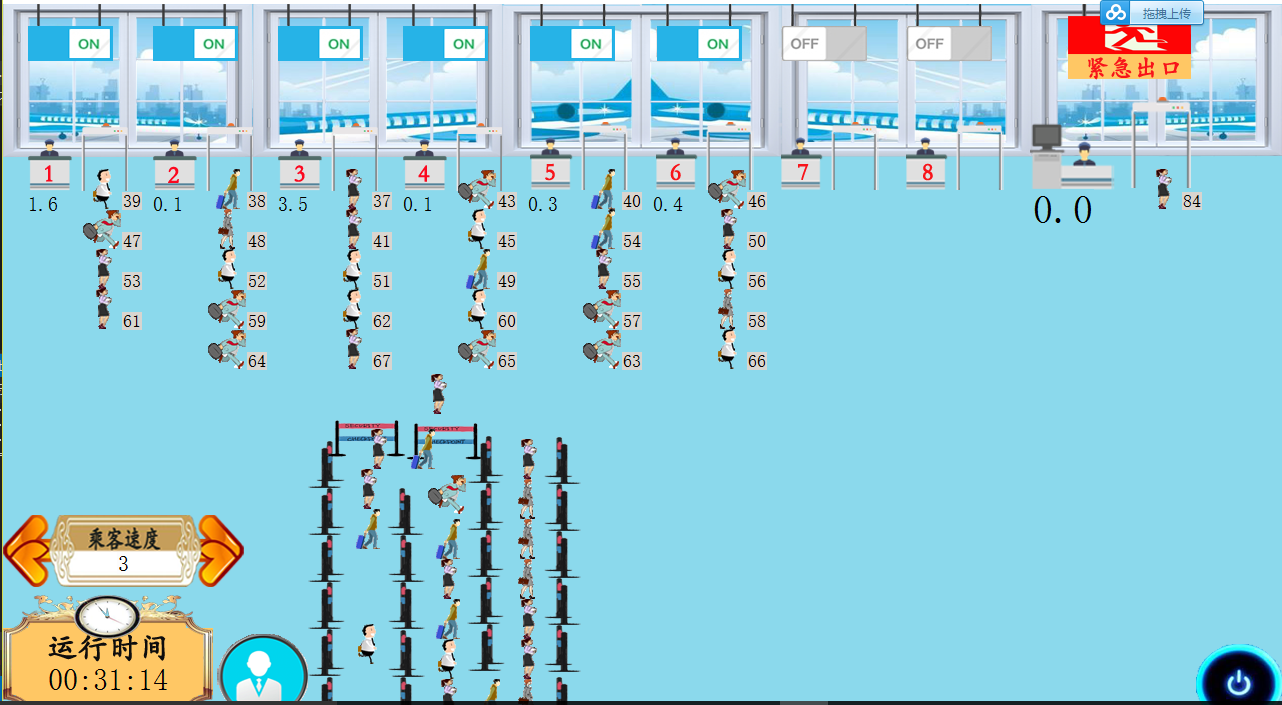


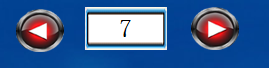
图6：



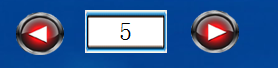
此二图为运行界面展示，界面上方从左至右八个数字标明了一至八号安检口，八个安检口上方的八个挂牌将标识安检口的状态；稍后展示运行图片，右上方较大的柜台代表紧急安检口，紧急乘客从界面右边产生并移动到紧急安检口；界面下半部分以下几点介绍：1.左下角的钟表牌用于输出程序运行时间2.其上方为“乘客速度调节按钮”调节左右箭头可调节乘客速度。3.其右方蓝色按钮为增加乘客按钮。4.中间部分为缓冲区，乘客在其中移动。5.最右方为关机下班按钮。

## 操作元素和操作效果

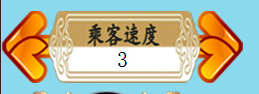
１.手动参数调节按钮(初始化界面)

（示例）

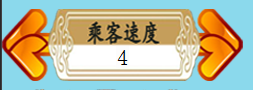
通过左右箭头调节参数，如调节左边有下图：



1. 乘客速度操作按钮



通过左右按键调节大小，比如点击右边按钮之后



1. 安检口状态调节按钮

状态2状态3

通过点击开关可以手动向对应安检口发出休息或开启指令。

４.添加乘客按钮



手动点击添加乘客

５.下班按钮



点击之后程序进入准备下班状态，停止接收新乘客进入缓冲区，待当前安检口及缓冲区清空后即下班，如图。

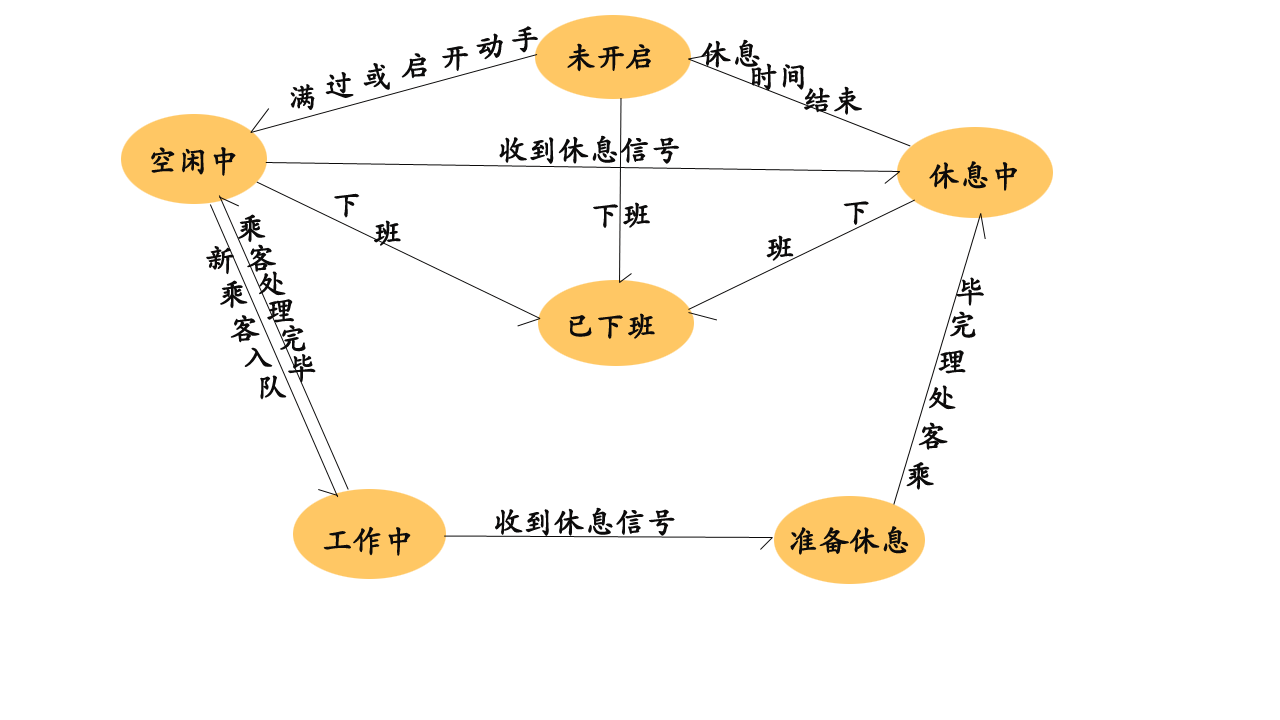


（注意缓冲区入口已封闭）然后程序自动退出。

# 2 有限状态自动机状态转换图

**2.1状态图**

**2.11安检口状态自动机**



**2.12系统上下班状态自动机**

****

**2.2状态说明**

安检口部分有六个状态，分别是1.空闲中 2.工作中 3.准备休息 4.休息中 5.未开启 6.已下班。转换如下，①当工作中遇到休息信号时，安检口转到准备休息状态，不再接收乘客；当工作中所有乘客处理完毕，转到空闲中状态；②当在空闲状态接收到新乘客即转为工作中状态；当在空闲状态且系统已下班时即转为已下班状态；③当准备休息状态处理完已有乘客即转到休息状态；④当休息时间已过后即安检口转到未开启状态；当在休息时且系统已下班，即转为已下班状态；⑤当在未开启状态接收到开启信号即转为空闲状态；若系统已下班，转为已下班状态

系统运行部分有以下三个状态 1.工作中 2.准备下班 3.已下班。具体转换情况如下①在系统工作时如接收到手动下班信号或下班时间到时转为准备下班状态；②在准备下班状态且所有乘客处理完毕后，系统正式进入下班状态。

# 3 高层数据结构设计

**3.1全局常量/变量定义：**

**3.1.1全局常量定义：**

#define SPARE 0 //安检口空闲状态；

#define WORKING 1 //安检口工作中状态，即窗口有人；

#define WAITING 2 //安检口过渡态，即安检口发出休息申请但窗口仍有人,此时不再接收乘客；

#define RESTING 3 //安检口休息状态，即窗口没人，也不接收乘客；

#define NOTOPEN 4 //安检口待开放状态；

#define CLOSED 5 //安检口已下班状态；

#define BEONDUTY 0 //机场上班状态；

#define TOOFFDUTY 1 //机场准备下班，清空缓存区，处理完安检区进入下班；

#define BEOFFDUTY 2 //机场下班状态；

#define UP 0 //乘客向上移动；

#define DOWN 1 //乘客向下移动；

#define LEFT 2 //乘客向左移动；

#define RIGHT 3 //乘客向右移动；

#define STILL 4 //乘客停止移动；

#define SCR\_WIDTH 1280 //动画窗口宽度；

#define SCR\_HEIGHT 720 //动画窗口高度；

#define PASSENGER\_HEIGHT 40 //乘客贴图高度；

#define PASSENGER\_WIDTH 40 //乘客贴图宽度；

define ENTRANCE\_X 345 //入口处X坐标值；

#define ENTRANCE\_Y 675 //入口处Y坐标值；

#define BUFFER\_UP\_Y 425 //缓冲区上侧边界Y坐标；

#define BUFFER\_DOWN\_Y 675 //缓冲区下侧边界Y坐标；

#define BUFFER\_LINE\_WIDTH 76//缓冲区队列宽度；

#define CHECK\_LINE\_Y 165//安检口Y坐标；

#define PI 3.141592654

**3.1.2全局变量定义：**

int MaxCustSingleLine; //缓冲区单队列最大乘客数；

int MaxLine; //蛇形缓冲区最多有多少行；

int MaxSeqLen; //最大允许等待长度；

int MaxCustCheck; //安检口队伍最大长度；

int MinCheckTime; //一次安检最短时长；

int MaxCheckTime; //一次安检最大时长；

int MinRestTimeLen; //安检口一次休息最短时长；

int MaxRestTimeLen; //安检口一次休息最长时长；

int OffdutyTime; //下班时间；

bool IsAutoGenerate; //是否自动产生乘客；

int Speed; //乘客移动速度；

int SystemState; //记录当前系统运行状态；

double CurrentTime; //记录当前时间；

vector<Passenger> buffer; //利用vector动态数组模拟缓冲区队列；

CheckPoint checkpoint[9]; //安检口数组，共九个，0号为紧急安检口；

**3.1.3数据结构定义：**

（1）乘客数据结构：

typedef struct

{

int ID; //乘客编号；

PIMAGE pic; //乘客对应贴图指针;

int x, y; //乘客坐标；

int dir; //乘客运动方向；

int LastDir; //乘客静止前最后一次运动方向,方便转向处理；

double CheckTime; //检查时间；

double LastLineX; //记录在上一个缓冲区队列的X坐标，方便横向移动处理；

}Passenger;

（2）安检口数据结构：

typedef struct

{

int x, y; //安检口坐标；

int state; //安检口状态；

double RestTime; //安检口休息时间；

vector<Passenger> CheckQueue; //安检口对应乘客队列；

}CheckPoint;

**3.2.1 控制模块常量与变量定义**

int BufferLineNum = 1; //记录当前蛇形队列行数；

int EXIT\_X[] = { 345,505,670,830 }; //缓冲区出口X坐标；

int PassengerNum = 1; //乘客数量，用于乘客编号，从1开始；

# 4 系统模块划分

## 4.1 系统模块划分

main.cpp

SystemControl.cpp

Input.cpp

Output.cpp

Init.cpp

void CheckPointControl()

void BufferControl()

**1．模块名称：main.cpp**

模块功能简要描述：主函数，主要是执行初始化操作，在主循环中按逻辑调用各模块功能函数，通过ege库提供的delay\_fps函数实现稳定帧数效果并判断系统进入下班状态后自动结束等。

**2．模块名称：Init.cpp**

模块功能简要描述：初始化模块，绘制起始界面和用户进行交互完成重要全局变量的初始化，以及进行程序运行过程中所需数据的初赋值等。

**3．模块名称Input.cpp**

模块功能简要描述：完成程序运行中动态获取用户输入的工作，并根据输入更改相应参数。

**4. 模块名称SystemControl.cpp**

模块功能简要描述：核心控制模块，分为两个主要功能子函数：BufferControl和CheckpointControl

BufferControl函数实现乘客出、入缓冲区及控制乘客在缓冲区内的运动，包括转向、停止，重新启动，并控制缓冲区蛇形队列的动态增开；

CheckPointControl函数实现根据运行状态实现安检口各个状态之间的转换，包括人数过多时动态开启新安检口等。同时控制乘客出、入安检口及控制乘客在安检口队列的移动，以及新开安检口后邻近安检口乘客的动态调整等。

SystemControl模块还实现对程序运行过程中的周期性文件输出，事件即时记录以及系统上下班状态的转换。

**5. 模块名称 Output.cpp**

模块功能简要描述： 输出模块，主要通过在SystemControl模块中进行控制修改的参数来在相应位置绘制贴图，实现动画效果。包括程序运行背景图，乘客贴图，安检口状态贴图等。·；

同时输出程序运行时间、乘客编号、安检剩余时间以及安检口休息时间等参数。

## 4.2各模块函数说明

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 函数原型 | 功能 | 参数 | 返回值 |
| 1 | void SetData() | 程序初始化时调用，用于设置安检口坐标，贴图指针等变量数据。 | 无 | 无 |
| 2 | void InputConfig() | 程序初始化时调用，用于与用户交互设置程序运行所需参数 | 无 | 无 |
| 3 | void Init() | 初始化图形界面和随机数种子，并调用前两个函数 | 无 | 无 |
| 4 | void Input() | 主循环中调用，在程序运行过程中动态获取用户鼠标点击，进行相应的操作。 | 无 | 无 |
| 5 | double gaussrand() | 模拟随机事件时调用.采用Box-Muller算法，提供符合标准正态分布的随机数. | 无 | 标准正态分布随机数 |
| 6 | int CheckPointNum() | 安检口状态控制函数中调用，返回当前开放的安检口数量，用于判断是否需要新开安检口 | 无 | 当前开放的安检口数量 |
| 7 | int ProperPointID() | 缓冲区出队函数中调用，返回当前开放安检口中人数最少的编号，用于乘客进入安检口判断。 | 无 | 当前乘客人数最少的安检口编号 |
| 8 | bool BufferFull() | 缓冲区控制模块中调用，判断当前缓冲区是否已满。 | 无 | 缓冲区是否已满判断结果 |
| 9 | bool CheckPointFull() | 缓冲区控制模块中调用，判断当前安检口是否已满 | 无 | 安检口是否已满判断结果 |
| 10 | bool CheckPointEmpty() | 系统控制模块中调用，判断当前安检口是否为空，用于系统上下班状态转换 | 无 | 安检口是否为空判断结果 |
| 11 | bool IsInExit(Passenger \* cus) | 缓冲区出队函数中调用，接受一个乘客指针，判断其是否处于当前缓冲区出口，用于缓冲区出队操作。 | 乘客数据结构指针 | 乘客是否处于出口判断结果。 |
| 12 | void CheckPointEnqueue(Passenger \* cus, int num) | 缓冲区出队函数中调用，将乘客入队至相应编号的安检口队列。 | 乘客数据结构指针，安检口编号 | 无 |
| 13 | void BufferDequeue() | 缓冲区控制模块中调用，用于缓冲区出队，如果安检口已满则让乘客停止等待 | 无 | 无 |
| 14 | void MoveInDir(Passenger \* cus) | TransPosInBuffer()中调用，根据乘客运动方向改变其横纵坐标，实现动画效果 | 乘客数据结构指针 | 无 |
| 15 | void TransPosInBuffer() | 缓冲区控制函数中调用，更改乘客坐标，实现乘客在缓冲区内的转向，静止等动画效果。 | 无 | 无 |
| 16 | void TransPosInCheckPoint() | 安检口控制函数中调用，更改乘客坐标，实现乘客分配至安检口及在安检口队列内移动的动画效果 | 无 | 无 |
| 17 | void TransCheckPointState() | 安检口控制函数中调用，根据情况更改安检口状态，实现状态机效果。 | 无 | 无 |
| 18 | void CheckPointDequeue() | 安检口控制函数中调用，用于已安检完毕的乘客出队 | 无 | 无 |
| 19 | void NewOpenAdjust() | 安检口控制函数中调用，用于新开安检口时邻近乘客的动态调整 | 无 | 无 |
| 20 | void BufferControl() | 系统控制模块中调用，控制蛇形队列的动态增开及自动产生乘客，并根据逻辑功能调用上述子函数，实现对缓冲区的总体控制 | 无 | 无 |
| 21 | void CheckPointControl() | 系统控制模块中调用，根据逻辑功能调用上述子函数，实现对安检口的总体控制 | 无 | 无 |
| 22 | void SystemControl() | 主循环中调用，通过对缓冲区和安检口两大控制子函数的调用实现对系统的总体控制，并控制上下班状态转换和进行周期性文件输出 | 无 | 无 |
| 23 | void Output() | 主循环中调用，根据控制模块提供的相应关键参数输出对应一帧的贴图及参数等，通过在每秒60次的主循环中循环调用实现动画效果 | 无 | 无 |

## 4.3 函数调用图示及说明

# 5 高层算法设计

**总体设计及时间控制思路：**

本次程序的总体设计采用离散化原则，将视觉上连续的动画转化为每秒六十帧静态贴图进行处理，以每帧为单位对程序进行控制处理，对程序运行中的关键参数进行操作，如调整乘客贴图的坐标，更新时间参数，更改安检口状态等，然后根据相应参数接口在输出模块中输出对应贴图，数据等，以此来达到流畅的动画效果。

时间控制方面，在控制模块通过ege库中的fclock()函数更新程序运行时间至CurrentTime，并沿承单帧操作的思想，每次将相应的时间参数如安检剩余时间，休息时间每次减去一帧所对应的时长1.0 / 60 s；

**重要的控制策略算法思路：**

以下以伪代码的形式描述项目中的几个核心控制算法：

void TransPosInBuffer() //改变乘客在缓冲区内的位置;

{

for (int i = 0; i <缓冲区人数; i++)

{

switch (缓冲区乘客的方向) //边界转向处理；

{

case 向上运动:

if (乘客y坐标到达缓冲区上界)

乘客运动方向变为向右

break;

case 向下运动:

if (乘客y坐标到达缓冲区下界)

乘客运动方向变为向右;

break;

case 向右运动:

if (如果乘客到达下一行)

{

if (乘客在缓冲区上界)

乘客运动方向变为向下

else

乘客运动方向变为向上

}

default: break;

}

if (此乘客前一乘客是静止) //移动转向静止判断处理；

{

if (乘客运动方向向上&& 乘客与前一乘客处于同一列 && 此乘客与前一乘客纵坐标即将重合)

{

此乘客方向变为静止并且记录静止前乘客运动方向为向上；

continue;

}

if (乘客运动方向向下 && 乘客与前一乘客处于同一列 && 此乘客与前一乘客纵坐标即将重合)

{

此乘客方向变为静止并且记录静止前乘客运动方向为向下

continue;

}

if (乘客运动方向向右 && 乘客与前一乘客处于同一行 && 此乘客与前一乘客横坐标即将重合)

{

此乘客方向变为静止并且记录静止前乘客运动方向为向右· continue;

}

}

if (乘客方向为静止) //静止转向移动判断处理；

{

if (乘客为队首乘客 && 此乘客未到缓冲区出口)) //队首元素特殊处理

乘客运动方向变为静止前运动方向

if (i > 0 && 乘客静止前方向为向上&& 此乘客与前一乘客在同一列且乘客再运动一次也不会与前一乘客相撞或与前一乘客不在同一列)

乘客方向继续为向上

if (i > 0 && 乘客静止前方向为向下 && 此乘客与前一乘客在同一列且乘客再运动一次也不会与前一乘客相撞或与前一乘客不在同一列)

乘客方向继续为向下

if (i > 0 && 乘客静止前方向为向右 && 此乘客与前一乘客在同一行且乘客再运动一次也不会与前一乘客相撞或与前一乘客不在同一行)

乘客方向继续为向右

}

MoveInDir(&buffer[i]); //根据已改变的运动方向改变乘客的坐标；

}

}

void NewOpenAdjust() //实现新增开安检口时临近安检口乘客队列的动态调整

{

if (第一个安检口为新开) //左侧边界安检口新开时，只需调整右侧安检口队列

{

for (int i = 2 ; i <第二个安检口乘客数量; i++)

将第二号安检口的第i个乘客入队到第一个安检口队列里

for (int i = 二号安检口乘客数减一; i >= 2; i--)

二号安检口删除已出队乘客；

}

if (最后一个安检口为新开) //右侧边界安检口新开时，只需调整左侧安检口队列

{

for (int i = 2; i < 第七个安检口乘客数量; i++)

将第七号安检口的第i个乘客入队到第八个安检口队列里

for (int i = 七号安检口乘客数减一;; i >= 2; i--)

七号安检口删除已出队乘客；

}

for (int i = 2; i <= 7; i++) //调整左侧及右侧安检口队列

{

if (第i号安检口新开)

{

int todel1[5] = { 0 }, todel2[5] = { 0 }; //用于记录将要删除的邻近安检口的乘客

int idx1 = 0, idx2 = 0;

for (int j = 2; j < max(左侧安检口乘客数量, 右侧安检口乘客数量); j++) //两侧安检口对应乘客进入新开安检口

{

若j < 左侧安检口乘客数量 ID1= 左侧i安检口第j个乘客的否则变为无限大;

若j < 右侧安检口乘客数量 ID1= 右侧i安检口第j个乘客的否则变为无限大;

if (ID1 < ID2)

{

将第i-1号安检口的第j个乘客入队到第i个安检口队列里 记录入队乘客号j

}

else

{

将第i+1号安检口的第j个乘客入队到第i个安检口队列里 记录入队乘客号j

}

}

for (int j = idx1 - 1; j >= 0; j--) //删除两侧安检口队列对应乘客

在左侧队列删除已去向新开队列的乘客；

for (int j = idx2 - 1; j >= 0; j--)

在右侧队列删除已去向新开队列的乘客；

}

}

**文件记录生成思路：**

文件记录方面，在初始化模块中，当初始化完成时，会记录程序开始运行的时间及此次运行的配置参数。

在系统控制模块中，利用如下语句

if ((int)(CurrentTime + 1.0 / 60) > (int)CurrentTime)

即检测当前时间个位数进位时，在文件中输出系统的状态记录，以达到每秒周期性输出文件记录的目的。

同时，在相应的功能子函数内会有文件输出语句，以达到对事件的即时记录。**教师评语：**