1. 用户界面设计
   1. 文件方式

* 文件名称：init、input、output、log；
* 文件格式：txt；
* 使用说明：
* init文件：进行火车配置信息的初始化，具体初始化数据如下：

[A Speed] //A运行速度

[A StartTime] //启动时间，0表示即刻启动

[A StartPoint] //A启动位置（例如将A轨道分为0-12点刻度，启动位置在0点）

[A InPoint] //A进入公共轨道探测点的位置

[A OutPoint] //A驶出公共轨道探测点位置

[A Type] //火车类型，1快车，2慢车

按照以上数据初始化方式，再分别初始化其他小火车的配置信息。

注：在图形化界面中的初始化数据中可以加入[ width],[ hight]参数，表示不同火车运行轨道的高度和宽度。

* input文件：输入要求:

T=<请求发生时间>

EVENT=<事件|事件>；

请求发生时间：按程序运行的系统时钟时间，单位秒。

事件：火车暂停或继续运行，事件之间以|分割。例如：PA：代表A车暂停，CA代表A车继续。

* output文件：当火车和轨道状态有变化时进行记录。

Train：表示发生轨道状态的火车；

T：火车轨道状态发生变化的时间；

Trans：火车轨道变化的具体信息；

PauseTime：火车暂停的时间

例：T=10；Train A；Trans：A->A-C

表示第10秒，火车A由轨道A进入公共轨道A-C

* log: 周期性记录火车和轨道的状态。

T：表示时间；

分别输出各小火车的信息，

State：小火车所在的轨道；运行方向：

例，runwest（表示小火车在轨道上向西行驶）；

Position：小火车的位置；

公共轨道的占用情况：是否被占用，被哪辆火车占用；

* 1. 动画方式

（在第二阶段完成）

1. 自动机模型
   1. 一级状态图

控制

计时

结束

状态切换

输出

输入

说明：

输入：获取火车控制事件

控制：根据控制策略计算公共轨道可使用情况

状态：计算此刻轨道、各辆火车的状态

输出：输出此刻各火车、轨道的状态

计时：时间片推进一个

* 1. 二级状态图

停止

顺时针行驶

公共口暂停

任意点暂停

逆时针行驶

顺时针启动

逆时针启动

结束

结束

公共轨道无车

公共轨道有车

且在公共口

用户暂停

用户恢复

用户恢复

用户暂停

在公共轨道口

用户无输入

公共轨道

有车

公共轨道无车

公共轨道有车

且在公共口

1. 高层数据结构定义
   1. 常量定义

#define STOP 1

#define RUN 2

#define PAUSE\_COMMOM 3

#define PAUSE\_ANY 4

#define NORMAL 5

#define REVERSE 6

#define IDLE 7

#define BUSY 8

#define EAST 9

#define WEST 10

#define SOUTH 11

#define NORTH 12

* 1. 全局变量定义

struct train

{

double speed;

int starttime;

int startpoint;

int inpoint;

int outpoint;

char direction[10];

int type;//火车类型，1快车，2慢车

};

struct block

{

int commom;//0非公共，1公共

int direction;//1左2下3右4上

}

1. 系统模块划分

main.c

input.c

control.c

state\_trans.c

output.c

* 1. 系统模块划分

1. 模块名称：input.c

模块功能简要描述：初始化和接受用户输入

1. 模块名称：control.c

模块功能简要描述：计算火车的状态转换

1. 模块名称：state\_trans.c

模块功能简要描述：实现火车的状态转换

1. 模块名称：output.c

模块功能简要描述：输出火车的状态及其转换

1. 模块名称：main.c

模块功能简要描述：实现各模块的调用

* 1. 各模块函数说明

|  |  |  |  |
| --- | --- | --- | --- |
| 模块文件 | 模块说明 | 模块包含的函数名 | 函数功能 |
| input.c | 火车配置的初始化信息 | void init() | 完成火车配置信息的初始化 |
| control.c | 火车命令请求的读取 | void getReq() | 火车命令请求的读取 |
| state\_trans.c | 完成火车状态的转换，计算出火车状态转换时的数据 | int pausetime() | 计算火车暂停或停靠的时间 |
| void trans() | 完成火车状态的转换 |
| int direction() | 计算得出火车的运行方向 |
| void time\_count() | 使时间片推进一个 |
| output.c | 输出并记录火车的相关信息 | void outPut() | 周期性输出并记录火车和轨道的状态 |
| void trainLog() | 输出并记录火车和轨道状态发生变化时的信息 |

* 1. 函数说明

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 函数原型 | 功能 | 参数 | 返回值 |
| 1 | void init() | 完成火车配置信息的初始化 | void | void |
| 2 | void getReq() | 火车命令请求的读取 | void | void |
| 3 | void trans() | 完成火车状态的转换 | void | void |
| 4 | int pausetime() | 计算火车暂停或停靠的时间 | void | 火车暂停或停靠的时间 |
| 5 | int direction() | 计算得出火车的运行方向 | void | 返回火车的运行方向，根据全局变量的定义可知，返回值为9，运行方向为向东；返回值为10，运行方向为向西；返回值为11，运行方向为向南；返回值为12，运行方向为向北 |
| 6 | void time\_count() | 使时间片推进一个 | void | void |
| 7 | void outPut() | 周期性记录输出火车和轨道 | void | void |
| 8 | void trainLog() | 输出并记录火车和轨道状态发生变化时的信息 | void | void |

1. 核心算法设计

* 完成火车状态的转换：

void trans()

{

switch 火车状态

{

case 停止:

if 顺时针启动

火车状态=顺时针行驶；

if 逆时针启动

火车状态=逆时针行驶；

break;

case 顺时针行驶:

if 公共轨道有车&&火车在公共轨道入口

火车状态=公共口暂停；

if 用户暂停

火车状态=任意点暂停；

if 结束信号

exit(1)；

break;

case 逆时针行驶:

if 公共轨道有车&&火车在公共轨道入口

火车状态=公共口暂停；

if 用户暂停

火车状态=任意点暂停；

if 结束信号

exit(1)；

break;

case 公共口暂停:

if 公共口有车

break；

if 公共轨道无车&&火车顺时针运行

火车状态=顺时针行驶；

if 公共轨道无车&&火车逆时针运行

火车状态=逆时针运行；

break；

case 任意点暂停:

if 在公共轨道口

火车状态=公共口暂停；

if 用户恢复&&火车顺时针运行

火车状态=顺时针行驶；

if 用户恢复&&火车逆时针运行

火车状态=逆时针行驶；

break；

}

}

* + 计算火车暂停或停靠的时间

int pausetime() {

int time = 0;

接收到暂停或停靠信号;

while (火车暂停或停靠) {

if (时间片推进一个)

time++;

}

return time;

}

* + 计算得出火车的运行方向

void direction()

{

if （火车在左边）

{

if （火车运行方向为顺时针） 火车方向设定为北；

else 火车方向设定为南；

}

else if （火车在右边）

{

if （火车运行方向为顺时针） 火车方向设定为南；

else 火车方向设定为北；

}

else if （火车在下边）

{

if （火车运行方向为顺时针） 火车方向设定为西；

else 火车方向设定为东；

}

else if （火车在上边）

{

if （火车运行方向为顺时针） 火车方向设定为东；

else 火车方向设定为西；

}

}