电梯模拟

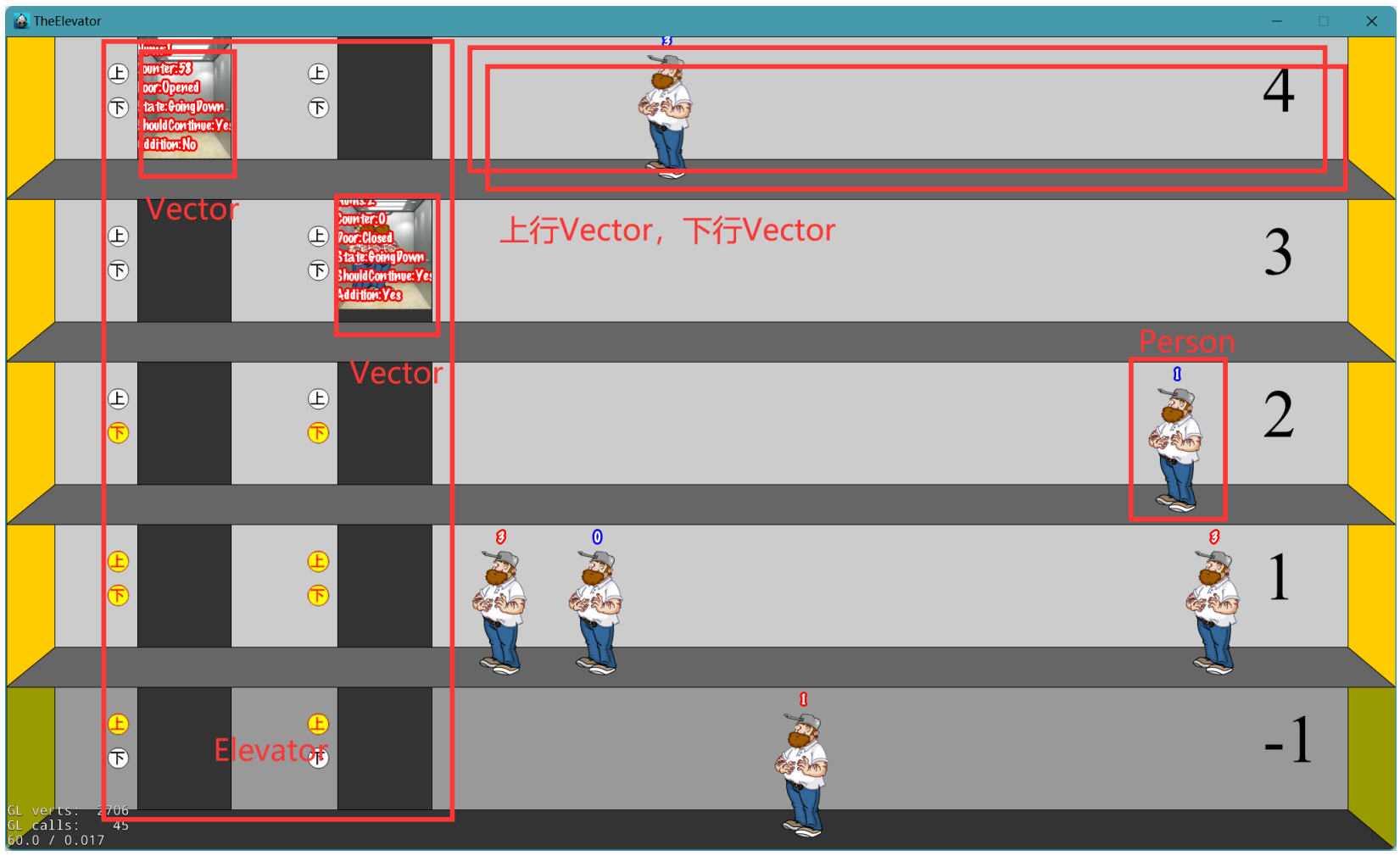
PB20111686　黄瑞轩

实验要求

设计一个电梯模拟系统——一个离散的模拟程序。电梯系统是由乘客和电梯等“活动体”构成的集合，虽然他们彼此交互作用，但他们的行为是基本独立的。在离散的模拟中，以模拟时钟决定每个活动体的动作发生的时刻和顺序，系统在某个模拟瞬间处理有待完成的各种事情，然后把模拟时钟推进到某个动作预定要发生的下一个时刻。

设计思路

将真实场景中的各对象进行抽象，分成类：`Vector`（容器），`Person`（人），`Elevator`（电梯），`TheWorld`（世界），`Board`（活动记录面板）。给这些类设置适宜的描述变量和方法，使其能够交互运作，一个示例如下图所示。



本实验使用cocos2d-x-v4来实现图形化。因为要显示模块图像，`Person`、`Elevator`和`TheWorld`类继承`cocos2d::Node`类。利用cocos2d-x的时序刷新模块`scheduleUpdate()`来实现每一帧的刷新，在`TheWorld`的`update(float dt)`方法中写每一帧应对各人、各电梯、各容器所做的操作（`ElevatorOperate`和`PersonOperate`）。`Board`作为活动记录面板，当检测到鼠标点击事件时呼出，再次点击即消失。`Vector`相当于一个链表类，`Person`相当于一个链表节点类。下面是数据结构头文件，其中一些无关紧要的变量被隐去。

```
class Vector {
public:
    // 电梯轿厢和等候队列共用区
    Person* FirstPerson = nullptr;     // 容器中第一个人的指针
    int PersonNums = 0;                // 容器中人数

    // 轿厢特有的成员
    int BoxElevNo = -1;                // 轿厢编号
    bool ShouldRefresh = false;        // 是否应该根据PersonNums刷新图形化

    // 成员函数部分
    void PushPerson(Person* p);              // 将p所指的人放到此容器末尾
    void DeleteFromLine(Person* p);          // 将p所指的人从此容器中删除，但是不释放内存
```

```
    void DeleteFromLineCleanUp(Person* p);    // 将p所指的人从此容器中删除，并且从程序中卸载
    Person* GetPrevPerson(Person* p);         // 获得p所指的人的前一个人的指针，没有返回nullptr
    Person* GetLastPerson();                  // 获得容器内最后一个人的指针
    Person* GetFirstUper();                   // 获得容器内第一个要上楼的人
    Person* GetFirstDowner();                 // 获得容器内第一个要下楼的人

    Vector();
};
```

```
class Person : public cocos2d::Node {
public:
    cocos2d::Sprite* PersonSprite;      // 人的图像
    Person* NextPerson = nullptr;       // 下一个人指针
    int TolerateTimeNums = -1;          // 容忍时间计数器
    int TolerateHistory = -1;           // 容忍时间
    int FloorNow = -1;                  // 现在所处的楼层（即出生楼层）
    int FloorPurpose = -1;              // 希望去的楼层
    int tag = 0;                        // 状态：0表示在电梯外等候，1表示在电梯内乘坐
    int ToUp = 1;                       // 0表示要向下，1表示要向上，在构造时确定
    int ElevNo = -1;                    // 现在处于几号电梯，在外等候时为-1

    Person(int BirthFloor);
};
```

```
class Elevator : public cocos2d::Node {
public:
    int ElevatorNo = -1;          // 电梯序号
    Vector* Box;                  // 电梯轿厢
    bool CallUp[5] = { false };   // 外侧楼层按钮
    bool CallDown[5] = { false };
    bool CallCar[5] = { false };  // 内部楼层按钮

    int ElevatorState = Idle;     // 电梯状态
    int DoorState = Closed;       // 电梯门状态
    int Floor = 1;                // 电梯当前所处楼层
    int Counter = 0;              // 倒计时计数器
    bool FloorAddition = false;   // 是否在楼层之间

    const int WaitForInPeopleTimeNum = 70;            // 开门后等人进来的时间间隔数

    cocos2d::Sprite* ElevSprite;                      // 电梯图像
    cocos2d::Sprite* PersonInTheBox[MaxPersonNum];    // 电梯中的人的图像

    Elevator(int i);

    int ShouldContinue();                             // 判断是否应该
    void DeleteEveryPersonEquals(int Purpose);
    void refresh();
    void MoveOneFloor(int UpDown);
    bool CheckOutside(int UpDown, int WantTo);
};
```

```
class TheWorld : public cocos2d::Node {
public:
    float Time = 0.0f;                // 时钟
    float Interval = 0.1f;            // 时间间隔
    float AdventInterval = 0.0f;      // 这一个人出现到下一个人出现的时间间隔
    float Counter = 0.0f;             // 计时器

    cocos2d::Sprite* Boundry;         // 世界图像

    Elevator* elev[ELEVATOR_NUM];     // 电梯
    Vector* line[FLOOR_NUM];          // 等待序列
    Board* board = new Board(false); // 活动记录面板
```

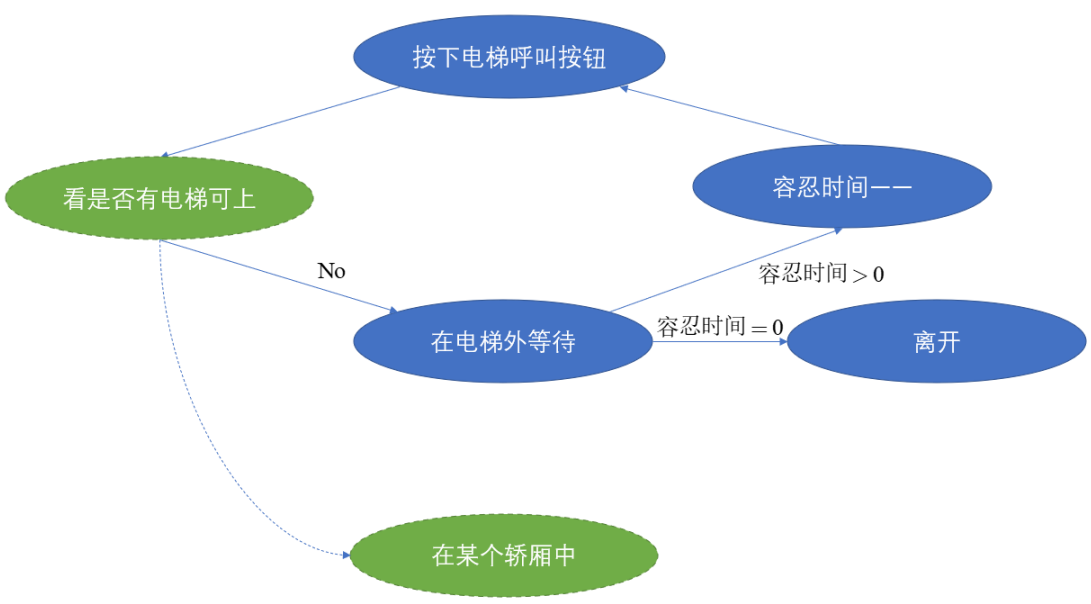
```
    TheWorld();
    virtual void update(float dt);

    // 人每帧应当做的操作
    void PersonOperate(Person* p);
    // 电梯每帧应当做的操作
    void ElevatorOperate(Elevator* v);
    // 将Floor楼所有Up的键熄灭，比如Up==0时将该层楼所有电梯的CallDown置false
    void DisableCallAllElevators(int Floor, int Up);
    // 检查电梯的楼层状态，这个函数设置Elevator::Floor和Elevator::FloorAddition
    void CheckAddition(Elevator* e);
    // 检查是否有停在Floor楼的向上的门开的电梯
    int CheckUpingElevators(int Floor);
    int CheckDownElevators(int Floor);
};
```

本程序还加入了自定义参数设置功能，测试者可以自行设置不同的参数搭配，以观察不同情况下的模拟结果，支持的变量如下图所示。这里的Assessment Pattern是一个延申部分，即电梯在接受乘客时是优先接受距离短的使得测试者可以观察在不同优先策略下的情况，具体代码体现在Vector类中。

关键代码讲解

TheWorld中的PersonOperate和ElevatorOperate是核心代码。

PersonOperate的状态图如下所示，绿色虚线部分的内容实际上是由ElevatorOperate来完成的，这里为了理解方便而加上了。



实际代码结构如下，和上面的状态图一致，就不详细赘述了。

```
        if (p->tag == 0) {
            // p->tag == 0表示是在外面等待的人
            p->TolerateTimeNums--;
            if (p->ToUp) {
                //要上楼，把每一架电梯的上按了
                for (int i = 0; i < ELEVATOR_NUM; i++) {
                    //这里省略操作按钮的代码
                }
            }
            else {
                //要下楼，把每一架电梯的下按了，和上面类似，省略
            }
        }
        //这人等不下去了，跑了，从所在容器中删除此人
        if (p->tag == 0 && p->TolerateTimeNums < 0) {
            //在所在楼层容器中将此人移除，代码省略
        }
```

ElavatorOperate的状态图则比较复杂，不方便绘出，涉及到电梯坐标（是否在楼层之间）、电梯门是否打开、电梯正在执行的任务（向上还是向下）、电梯的状态是否应该继续这四个变量。其中，电梯的状态是否应该继续（ShouldContinue）是一个辅助变量，它是为了避免过于复杂的条件判断嵌套而引入的。这里我先讲述ShouldContinue的处理逻辑。

ShouldContinue()返回一个枚举值，有YesCompletely,YesFindTheHigherDowner,YesFindTheLowerUper,No四种返回值。它先查看当前世界下该电梯的所有信息，然后返回一个信息告知电梯是否应该变更自己当前的状态以及如何变更。

（**1**）电梯当前状态为GoingUp，其先查看电梯内部是否还有人，如果还有人表示上升过程未结束，返回YesCompletely；如果电梯内部没人了，其查看上方是否有需要下行的人，如果有，返回YesFindTheLowerUper，告知电梯应当继续向上，找一个楼层最低的上行者；如果没有，说明电梯状态应当改变，返回No；

（**2**）电梯当前状态为GoingDown，与上面情况是对称的，不再赘述。

（**3**）电梯当前状态为Idle，其查看三种情况：上方是否有人呼叫、下方是否有人呼叫、该层楼是否有人呼叫，在这里不处理优先级的问题，仅告知电梯是否需要改变状态，返回YesCompletely或者No。

现在根据上面的内容，讲述ElevatorOperate的处理逻辑。

（**1**）把电梯状态设置成Idle，当四个条件都满足：不在楼层中间，也就是停靠在某一层、门是关着的、当前不是Idle状态、不应该再保持先前的状态。

（**2**）查看电梯是否正在移动，即检查电梯坐标是否是停止坐标（即每一层楼的地板坐标）。

- 如果是，则代表任务没有完成，电梯向移动方向继续移动一个单位；
- 如果不是，则代表电梯来到了某一层，现在需要做的是确认电梯是否应该开门。如果当前楼层没有“顺路”的，并且也没有人要在这一层楼里下的，就不开门，反之则开门。

> 注：检查是否有人“顺路”实际上是根据各楼层的呼叫按钮来确定的，和各*Vector*的实际情况无关，事实上在现实生活中，*Elevator*只能被动的接受*Person*的信息，当*Person*离开后，不会对*Elevator*的信息输入做主动改变。例如人如果不能忍受而离开后，他不会去取消已呼叫的按钮。

（**3**）检查电梯门的状态，有打开和关闭两个状态分支。

- 当前电梯门打开，则有一个等待时间计数器（Counter~WaitForInPeopleTimeNum）。

  当此计数器为正值时，电梯会不断地判断是否应该让该楼层的每个人都上来，考虑的因素有：当前电梯是否满员；此人是否顺路。一旦有新人上电梯来，则会刷新此计数器。

  当此计数器为零时，电梯门被设置为关闭状态。
- 当前电梯门关闭，电梯的ShouldContinue告知电梯是否应该改变状态，电梯再根据具体的情况（上方是否有人呼叫、下方是否有人呼叫、该层楼是否有人呼叫）改变至具体的状态。

> 注：这里很重要的一点是设置了两个电梯的优先级策略，是为了避免发生当两个电梯都*Idle*时，有人出现后两个电梯一起运动的情况。

调试分析

时间复杂度

（1）**Board**类

- `update`方法每次刷新容器中所有变量值，时间复杂度是$O(n)$，n是Board类变量数。
- `AddNumberToAverage`方法用立即数计算每层楼的平均时间，时间复杂度是$O(1)$。

（2）**Vector**类

- `GetLastPerson`方法遍历链表获得最后一个元素的指针，时间复杂度是$O(n)$，n是容器中人数。
- `GetPrevPerson`方法获得指定元素的前一个元素的指针，平均时间复杂度是$O(n)$，n是容器中人数。
- `PushPerson`方法调用`GetLastPerson`方法获得位置，再插入，时间复杂度是$O(n+1)=O(n)$，n是容器中人数。
- `DeleteFromLine`方法调用`GetPrevPerson`方法获得指针，再删除，平均时间复杂度是$O(n+1)=O(n)$，n是容器中人数。
- `DeleteFromLineCleanUp`方法调用`DeleteFromLine`方法，再释放内存，平均时间复杂度是$O(n)$，n是容器中人数。
- `GetFirstPerson`方法是一个修补方法，时间复杂度是$O(1)$。

（3）**Person**类

- `Rand`方法是C++11标准库函数，对本项目而言时间复杂度是$O(1)$。

（4）**Elevator**类

- `MoveOneFloor`方法对电梯坐标进行一个单元的移动操作，对本项目而言时间复杂度是$O(1)$。
- `CheckOutside`方法按给定参数检查楼层按钮，时间复杂度是$O(5*2)=O(1)$。
- `ShouldContinue`方法调用`CheckOutside`方法的可能次数为0、1、4次，其实时间复杂度也是$O(1)$。
- `DeleteEveryPersonEquals`方法需要遍历轿厢`Vector`中每个元素，时间复杂度是$O(n)$，n是容器中人数。
- `GetLowUpFloor`、`GetHighDownFloor`方法的操作循环数都不超过5，时间复杂度是$O(1)$。

（5）**TheWorld**类

- `PersonOperate`方法对于单人而言时间复杂度是$O(1)$。
- `CheckUpingElevators`、`CheckDownElevators`的时间复杂度都是$O(1)$。
- `ElevatorOperate`尽管复杂，但是是多个条件分支，每个分支的时间复杂度都是直接调用上面各类方法，平均时间复杂度是$O(n)$，n是容器中平均人数。
- `DisableCallAllElevators`方法是一个修补方法，时间复杂度是$O(1)$。
- `CheckAddition`方法是一个修补方法，时间复杂度是$O(1)$。

空间复杂度

（1）**Board**类

- 整个空间复杂度对本项目而言是$O(1)$。

（2）**Vector**类

- 类的空间复杂度是$O(n)$，n是内部人数。

（3）**Person**类

- 这个类的空间复杂度是$O(1)$。

（4）**Elevator**类

- `PersonInTheBox`的空间复杂度是$O(n)$，n是电梯最多能承受的人的个数，其他空间复杂度是$O(1)$，整个类的空间复杂度是是$O(n)$。

（5）**TheWorld**类

- 包含ELEVATOR_NUM个电梯，FLOOR_NUM个上楼容器，FLOOR_NUM个下楼容器，ELEVATOR_NUM个电梯Node对象，整个类的空间复杂度是$O(m+n)$，m是ELEVATOR_NUM，n是FLOOR_NUM；展开来说，整个类的空间复杂度是$O(mi+nj)$，m是ELEVATOR_NUM，n是FLOOR_NUM，i是容器内部人数，j是电梯最多能承受的人的个数。

实验中遇到的问题

（1）选用数据结构的问题

由于设计的数据范围非常小：一个电梯最多10人，只有5层楼，因此在设计数据结构上使用链表或者使用数组都可以，并且两者的性能在这种情况下区别不大，出于灵活性的考虑我使用了链表结构。

（2）如何实现时钟功能

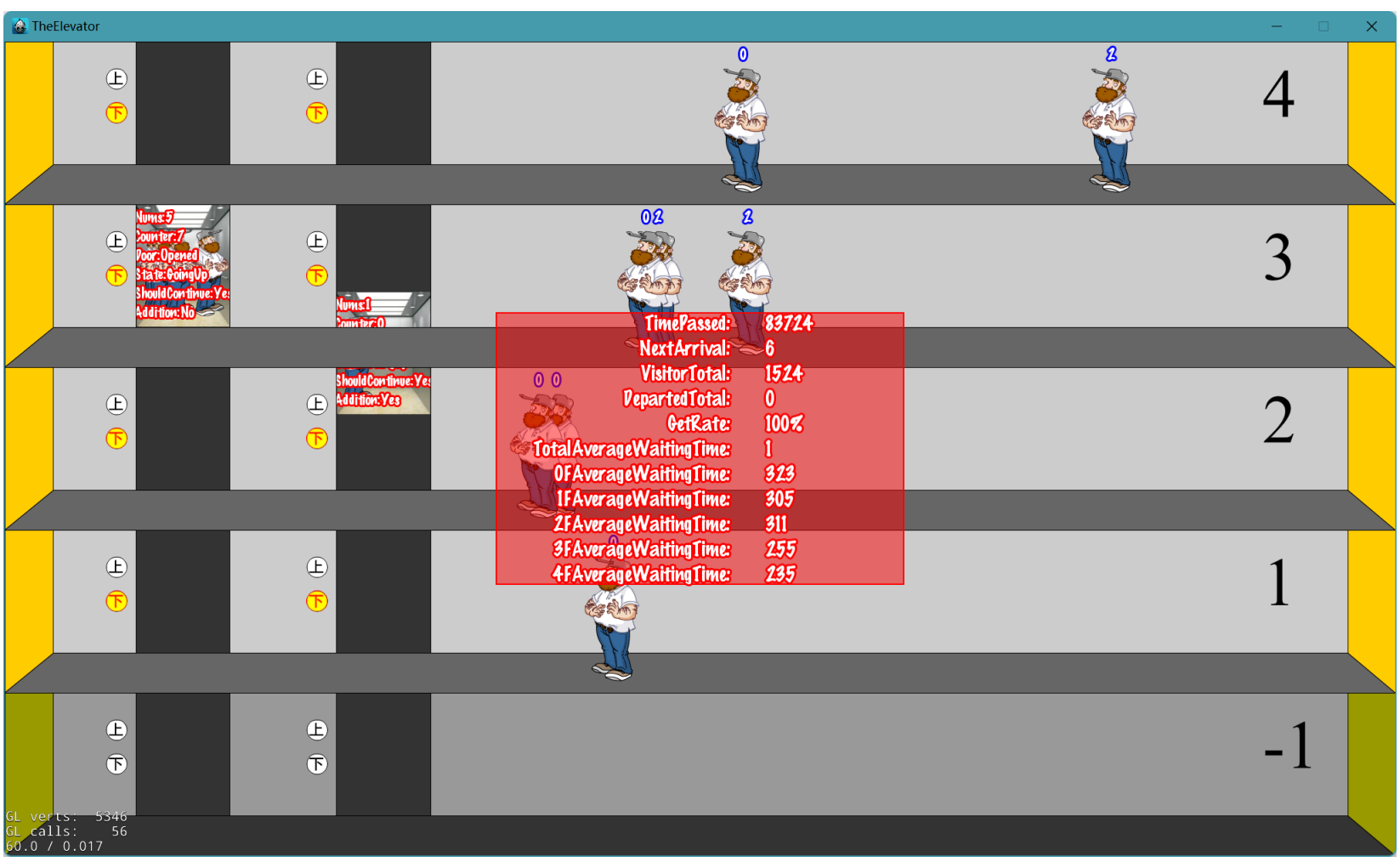
如果不要求图形化，Windows下开发可以使用Windows.h库中的sleep()函数来实现每隔一定时间刷新控制台。我最近正好在学习cocos2dx开发，就用这个引擎制作了图形化界面，利用其自带的update定时器完成了这一功能。对项目设置帧率可以达到间隔不同时间刷新的效果。

（3）如何做到优化

本项目的数据量其实是很少的，如果追求极致的时间性能，还可以牺牲一些空间。比如在Vector中添加最后一个结点域名，就可以免去寻找最后一个结点的时间复杂度，等。

代码测试

图形化统计



长时间的统计结果见图，参数为NextArrival varies in 5 ~ 6，容忍时间很长，最后每层楼的平均等待时间稳定在300时间单位附近。

控制台活动记录

测试时的活动记录截图样例如下图所示。

```
[55.1]Elevator[0] closed its door!
[57.1]Person[Yole Peng][09BEA528] from 4 floor press 0# elevator's 0 button!
[57.1]Person[Yole Peng][09BEA528] from 4 floor press 1# elevator's 0 button!
[63.1]Person[Yole Song][09BE8AB0] from 3 floor press 0# elevator's 0 button!
[63.1]Person[Yole Song][09BE8AB0] from 3 floor press 1# elevator's 0 button!
[63.1]Elevator[0] stop at floor[3] and opened its door!
[63.9]Elevator[1] stop at floor[4] and opened its door!
[69.1]Person[Pascal Gu][09BE9240] from 3 floor press 0# elevator's 0 button!
[69.1]Person[Pascal Gu][09BE9240] from 3 floor press 1# elevator's 0 button!
[70.8]Elevator[1] closed its door!
[70.9]Elevator[1] opened its door and start going down!
[75.1]Person[Smith Liu][09BE9D98] from 3 floor press 0# elevator's 1 button!
[75.1]Person[Smith Liu][09BE9D98] from 3 floor press 1# elevator's 1 button!
[76.1]Elevator[0] closed its door!
[77.9]Elevator[1] closed its door!
[83.1]Elevator[0] closed its door!
[85.1]Person[Rozzon Song][09BE8E78] from 2 floor press 0# elevator's 1 button!
[85.1]Person[Rozzon Song][09BE8E78] from 2 floor press 1# elevator's 1 button!
[90.1]Person[Oscar Gu][09BEA160] from 4 floor press 0# elevator's 0 button!
[90.1]Person[Oscar Gu][09BEA160] from 4 floor press 1# elevator's 0 button!
[91.1]Elevator[0] stop at floor[2] and opened its door!
[91.2]Person[Rozzon Song][09BE8E78] from 2 floor press 0# elevator's 1 button!
[91.2]Person[Rozzon Song][09BE8E78] from 2 floor press 1# elevator's 1 button!
[93.8]Elevator[1] stop at floor[2] and opened its door!
[98.1]Elevator[0] closed its door!
[100.7]Elevator[1] closed its door!
[100.8]Elevator[1] opened its door and start going up!
[100.9]Person[Terry Wen][09BEA8F0] from 2 floor press 0# elevator's 1 button!
[100.9]Person[Terry Wen][09BEA8F0] from 2 floor press 1# elevator's 1 button!
[106.1]Person[Mary Song][09BE8320] from 3 floor press 0# elevator's 0 button!
[106.1]Person[Mary Song][09BE8320] from 3 floor press 1# elevator's 0 button!
[106.1]Elevator[0] stop at floor[1] and opened its door!
[107.9]Elevator[1] closed its door!
[111.1]Person[Mary Gu][09BE9608] from 1 floor press 0# elevator's 1 button!
[111.1]Person[Mary Gu][09BE9608] from 1 floor press 1# elevator's 1 button!
[113.0]Elevator[0] closed its door!
[115.9]Elevator[1] stop at floor[3] and opened its door!
[116.1]Person[Rozzon Fang][09BE7400] from 1 floor press 0# elevator's 0 button!
[116.1]Person[Rozzon Fang][09BE7400] from 1 floor press 1# elevator's 0 button!
[121.0]Elevator[0] stop at floor[0] and opened its door!
```

实验总结

经过本次实验，我学会了对离散世界进行模拟的基本设计方法，强化了为特定问题选择特定数据结构的设计思路。本次实验总体思路是比较简单的，实现起来也并不复杂，主要是注意设计时思维的连续性，对于判断的条件等不要有遗漏或者错误。

我完成了实验的所有要求的基础部分，在发散部分，我做了以下工作：①实现了两个电梯时的情况；②利用cocos2d-x-v4引擎来实现了图形化界面；③加入了世界参数设置面板；④为电梯设置了可选的Assessment Pattern，以便测试不同策略下的情况。

附录：文件清单

```
Board.cpp               // 这些类的用途已在设计思路部分阐明，不再赘述
Board.h
Elevator.cpp
Elevator.h
Person.cpp
Person.h
TheWorld.cpp
TheWorld.h
Vector.cpp
Vector.h
AppDelegate.cpp         // 图形化界面相关设置
AppDelegate.h
HelloWorldScene.cpp     // 图形化入口类（场景）
HelloWorldScene.h
```