编号 _____

版本 第一次迭代

详 细 设 计 说 明 书

项目名称　　智能扑克游戏平台

项目负责人　章博文

编写　李泽雨　　2020 年 4 月 27 日

校对　罗溥晗　　2020 年 4 月 28 日

审核　章博文　　2020 年 4 月 28 日

批准　章博文　　2020 年 4 月 28 日

单位：武汉大学弘毅学堂 17 级计算机班

目录

1 引言

1.1 编写目的

本文档为智能扑克游戏平台项目第一次迭代的设计文档，主要针对本项目的需求规格文档，进行进一步的详细设计。通过这份文档，希望能够反映目前主要的设计思想，同时帮助组员在实现过程中有更好的理解。

1.2 背景

本项目的承办单位：2017HYAP01 小组

待开发系统软件的名称：智能扑克游戏平台

1.3 参考资料

a. 2017HYAP01 小组《第一次迭代需求规格文档》，2020

b. Stephen R.Schach《软件工程 面向对象和传统的方法 原书第 8 版 中文版》[M]. 第 8 版. 北京: 机械工业出版社, 2011.

c. 国标 GB8657——88 设计详细说明书

1.4 具体分工

表 1.1 任务分工明细表

模块设计	负责人
图形界面设计	金千千、李泽雨
游戏逻辑设计	罗溥晗、莫会民
AI 策略设计	李泽雨
平台逻辑设计	毛文月、杨佳乐、章博文

2 程序系统的结构

2.1 类图设计

如图 2.1，为顶层类图，反映了在本次迭代中的主要设计，本次迭代中的任务是将在本地环境下完成用户在命令行中的基本游戏行为。

顶层类图反映了整体架构的设计，而游戏逻辑部分类图反映了在纯模拟环境下德州扑克游戏的设计逻辑。

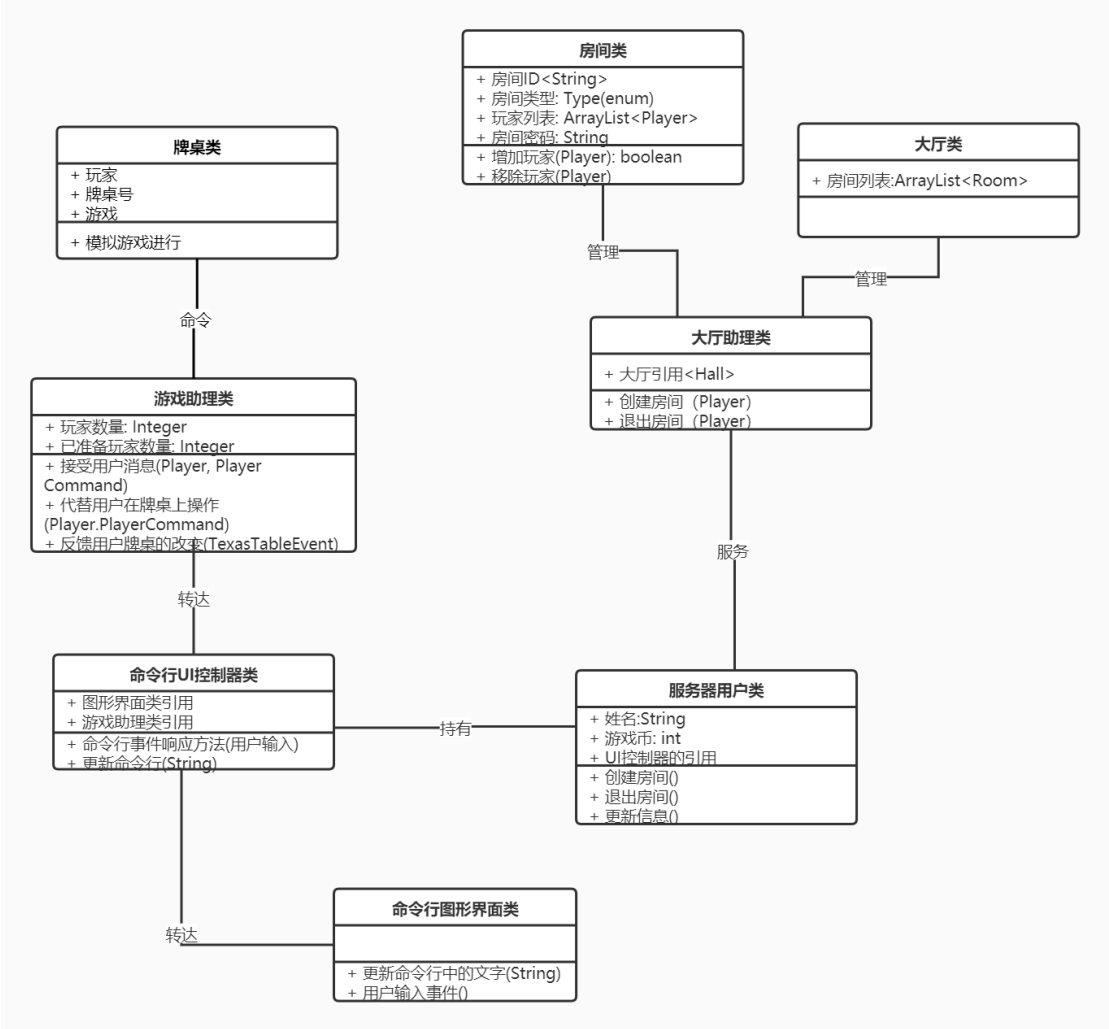

图 2.1 顶层类图

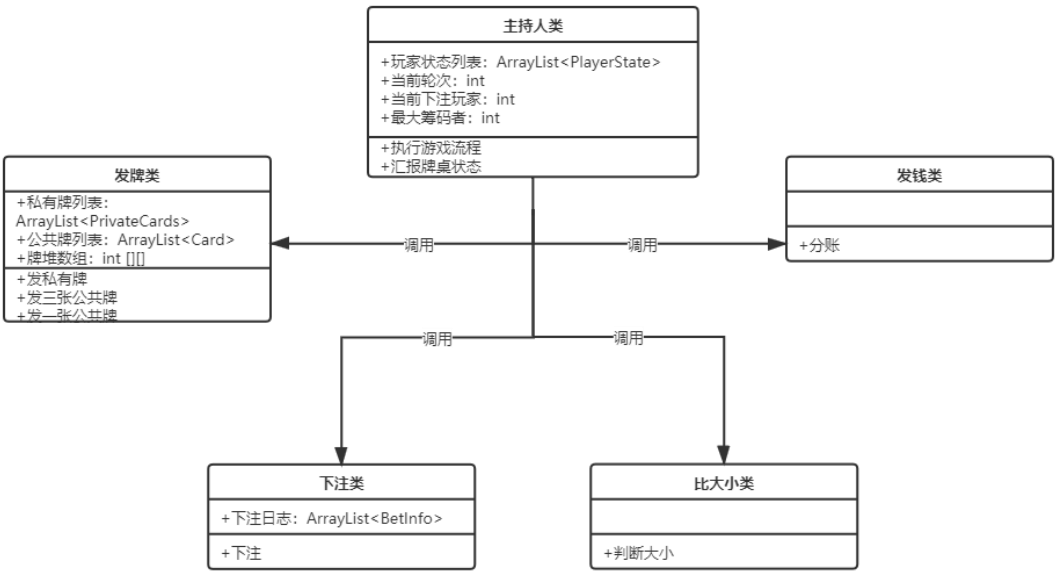

图 2.2 游戏逻辑类图

3．程序详细设计

3.1 图形界面设计

3.1.1 登陆界面线框设计

图 3.1 登陆界面

最上方为标题显示区，中间为玩家信息输入区，最下方为开始按钮。

3.1.2 结果界面线框设计

图 3.2 结果显示界面

上方为结果显示区，下方三个按钮玩家可选择退出、重新开始、下一轮游戏。

3.1.3 牌桌界面线框设计

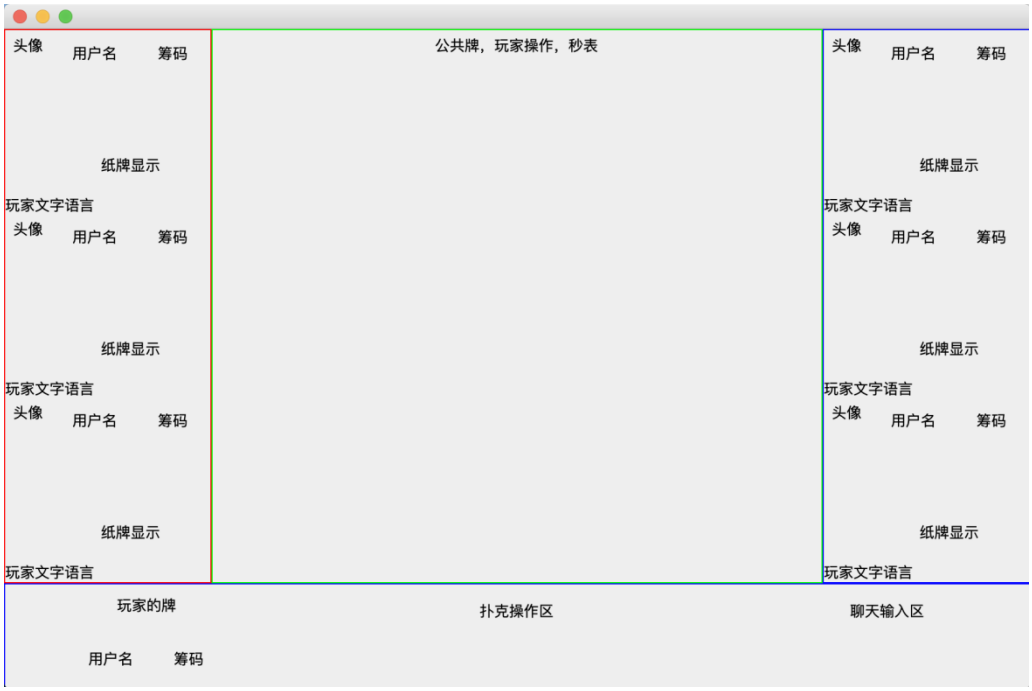

图 3.3 牌桌界面

整张牌桌中间显示各个玩家的操作，押注的筹码，公共牌，计时秒表等公共信息，最下方显示玩家自己的信息、牌，扑克操作区可进行游戏操作，聊天输入区可输入信息，其他玩家的各个信息以及输入的文字语言在牌桌左右两侧显示。

3.2 游戏逻辑设计

3.2.1 数据结构设计

总牌堆设计：

一个全局变量二维数组 int pile[4][13]，其大小为 4 x 13。

其中，第一维表示花色，第二维表示牌面。

第一维的取值范围为 1-4,依次代表黑桃 红桃 方片 梅花,第二维的取值范围为 1-13，依次代表 2 3 4 5 6 7 8 9 10 J Q K A，当某一位置的值为 0，表示该牌还在牌堆里，为 1 时表示已被发牌发走了。

```
    public class BetInfo{
         //单条下注信息，包含用户名和下注金额以及是在第几轮下的注，是否弃牌
        public String username; //下注人的用户名
        public int money; //下注金额
        public int round; //下注轮次
        public boolean quit; //是否弃牌（0=不弃牌，1=弃牌）

    public BetInfo(String username, int money, int round, Boolean.
quit){                          //实例化方法
            this.username = username;
            this.money = money;
            this.round = round;
            this.quit = quit;
        }
    }


    public class Card{//单张卡，包含花色（suit）和牌面（face）
        public int suit; //1-4 依次代表 Spades, hearts, diamonds , clubs
        public int face; //1-13 依次代表 2 3 4 5 6 7 8 9 10 J Q K A

    public Card(String suit, int face){ //实例化方法
            this.suit = suit;
            this face = face;
        }

    public void assginCard(int suit, int face) { //修改卡信息
            this.suit = suit;
            this.face = face;
        }
    }
    public class PrivateCard{//玩家手牌，包含用户名和以单张卡为元素的
arraylist
        public String username; //玩家的用户名
        public ArrayList<Card> privateCards; //玩家的手牌

        public PrivateCard(String username, Card card1, Card card2){
             //实例化方法
            this.username = username;
            this.privateCard.add(card1);
            this.privateCard.add(card2);
        }
    }
```

```
    public class TexasTableEvent {//游戏逻辑返回给平台逻辑的返回值
        //枚举值
        public static int TABLESTATE_DELIVER_HAND_CARD = 1; //发手牌
        public static int TABLESTATE_DELIVER_THREE_PUBLIC_CARD = 2; //发公共
牌
        public static int TABLESTATE_GAME_OVER = 3; //一局游戏结束
        public static int TABLESTATE_DELIVER_FIRST_EXTRA_PUBLIC_CARD = 4; //
发了第一张额外的公共牌
        public static int TABLESTATE_DELIVER_SECOND_EXTRA_PUBLIC_CARD =5; //
发了第二张额外公共牌
        public static int TABLESTATE_NOTHING = 6;//没有发生上述事件
        public int whatHappen; //当前发生了什么，是上面个中的一个
        public int whoseTurn;  //轮到谁了，牌桌上第几个人
    }


    public class TexasCommand {//平台逻辑传递给游戏逻辑的参数
        //枚举值
        public static int HAPPEN_EVERYONE_READY = 1; //每个人都准备好了
        public static int HAPPEN_SOMEONE_BET = 2; //有人下注
        public static int ACTION_ADD_BET = 1; //加注
        public static int ACTION_FOLLOW_BET = 2; //跟注
        public static int ACTION_ABORT_BET = 3; //弃牌
        public static int ACTION_CHECK = 4; //让牌
        public int whatHappen; //发生了什么，是上面的一个值
        public int who; //谁
        public int action; //动作
        public int chips;  //筹码数
        public ArrayList<String> nameList = new ArrayList<String>();
    }
```

算法 3.1 牌桌数据结构定义

3.2.2 核心算法设计

```
void bet(String username, int money){
    将 username 和 money 装入一个 BetInfo;
    将该 BetInfo 装入 BetLog;
}
```

算法 3.2 下注类算法

```
    void intiPrivate(int playernumber){
    用 dealPrivate 方法生成一个玩家的私有牌;
    向 this.privateCardList 中加入生成的 PrivateCard;
    }

    PrivateCard dealPrivate(String username){
    随机生成两张仍在总牌堆里的 Card，分别是 Card1，Card2;
    利用 PrivateCard 的实例化方法，传入 username 与 Card1，Card2;
    将生成的 PrivateCard 加入 this.privateCardList;
    return 上面生成的 PrivateCard;
    }

    void dealPublic(){
    利用 Card 实例化方法，生成仍在总牌堆里的三张 Card，分别是 Card1，Card2，
Card3;
    将这三张 Card 加入 this. publicCards;
    }

    void dealOneRound(){
    利用 Card 实例化方法生成仍在总牌堆里的 1 张 Card;
    将这张 Card 加入 this.publicCards;
    }
```

算法 3.3 发牌类算法

```
    public TexasTableEvent call(TexasCommand command) {
        if(收到准备完毕命令){
                初始化
                返回发私有牌成功和下一个下注的玩家号
        }
        else if(收到某人下注命令){
                while(true){
                    计算出下一个操作的人 x
                    if(x 还在场上){
                            break;
                    }
                }
                if(有人加注){
                    修改玩家列表
                    调用下注函数
                    修改上一次加注玩家的序号
                }
                else if(有人跟注){
                    修改玩家列表
                    调用下注函数;
                }
```

```
                else if(有人弃牌){
                    修改玩家列表
                    调用下注函数，注明弃牌
                }
                else if(有人让牌){
                    调用下注函数
                }
                if(一轮下注结束){
                    if(是第一轮) {
                        轮数+1
                        发三张公共牌
                        返回发公共牌和下一个下注的玩家号
                    }
                    else if(是第二轮){
                        轮数+1
                        发一张公共牌
                        返回发公共牌和下一个下注的玩家号
                    }
                    else if(是第三轮){
                        轮数+1
                        发一张公共牌
                        返回发公共牌和下一个下注的玩家号
                    }
                    else if(是第四轮){
                        结算
                        返回游戏结束
                    }
                }
                返回没有事件发生和下一个下注的玩家号
        }
    }
```

算法 3.4 牌桌类算法

3.3 AI 策略设计

3.3.1 程序描述

在用户选择单机模式进行游戏时，系统将会根据用户选择游戏人数与 AI 难度，相应的实例化 AI 类加入用户牌桌。在游戏过程中，AI 会根据玩家选择的难度，采取不同的策略与玩家进行对战。

3.3.2 功能

1.根据场上其他玩家的行为与自己底牌信息等进行行为决策。

2.在决策后，能够模拟真实玩家进行跟注、加注、弃牌等行为。

3.3.3 算法

```
public class AI{
    public int pool;// 表示 AI 当前的积分
    public String type; //表示 AI 的难度类别
    public void Strategy_control(){
    // 根据 AI 的难度，选择不同的行为策略
      switch(this.type){
        case "easy":
          easy_action();
          break;
        case "medium":
          medium_action();
          break;
        case "hard":
          hard_action();
          break;
      }
    }
```

算法 3.5 AI 类控制算法

控制算法是根据 AI 的难度类型，进一步控制 AI 再游戏中的行为策略。

```
    public void easy_action(){
      // 简单模式下的行为策略
      初始化一个行为策略矩阵 A// A[i][j]表示在状态 i 下，采取行为 j 的概
率。
      计算当前可组成最大牌牌型 i；// 最大牌型作为矩阵的状态索引。
      随机产生一个大于 0 小于 1 的数 pro//作为概率索引。

      if（判断 pro 和此牌型状态下的行为概率大小）
      产生相应的行为 j；
    }
```

算法 3.6 简单模式 AI 策略算法

```
public void medium_action(){
// 中等模式下的行为策略
    if(this.turn==1){
        计算当前最大牌;
        if (牌型较大)
            raise () ;
        else if (牌型较小) {
            double prob=calaulate_prob(); //计算当前获胜的概率
            if (prob>0.5)
                call () ;
            else
                fold () ;
        }
    }
    else{
        计算当前最大牌;
        if (牌型较大)
            raise () ;
        else if(牌型较小){
            double c_prob=model_competitor();
            if(c_prob<0,5)
                fold();
            else
                产生随机数选择 call () 或者 raise () ;
        }
    }
}
```

算法 3.7 中等模式 AI 策略算法

中等模式的 AI 策略相比于简单策略，加入了对自己手牌变化的预期，以及对对手上一轮行为所产生结果的估计。

```
private double calculate_prob(){
//计算自己获胜概率，对自己在下一轮获得更大牌型概率的预测
    double max_pro0;
    for（第 n 种牌型，由大到小）{
        if（当前牌中存在这种牌型的子集）{
            int diff=这种牌型与自己牌型差距的牌的数目;
            double prob= (1-n/总牌型数)*(ratio^diff)
            // ratio 为衰减参数，在后期实验时进一步确定，故没有在此声明
            max_prob=max_prob>prob>max_prob:prob;
        }
    }
}

private double modeling_competior(){
//计算对手获胜的概率，考虑对对手之前的下注操作，对获胜概率进行更新
    if(上一轮中选择加注的玩家>总玩家数*0.5){
        double prob=calculate_prob()*alpha;//alpha 为衰减参数，暂未确定
        return prob;
    }
}
```

算法 3.8 获胜概率计算算法

3.3.3 尚未解决的问题

中等模式下的 AI 策略虽然考虑了未来自己手牌的变化与上一轮中对手下注的情况。但式模型较为简单，暂未考虑诈唬与诈唬等行为，并且参数的值暂未确定。在后期不影响困难模式策略的设计情况下，如果有充裕时间，则会做进一步优化。

困难模式下 AI 的行为策略，目前计划使用强化学习训练出一个策略，也就是一个矩阵。但由于对目前大部分非完全信息博弈强化学习的认识还处于双人博弈阶段，对于多人博弈认识较少，在后续会深入研究并设计出适合该应用的算法。

3.4 平台逻辑设计

3.4.1 总体设计

本节将解释平台逻辑的总体设计，在本阶段，我们完全在本地模拟用户在德州扑克以及游戏大厅中的行为。

在设计上，我们将整个平台逻辑部分分成了三种类型的事务，分别是用户、牌桌和大厅。

用户事务的主要工作有两大块，分别是接受真实用户的行为，并且模拟它，模拟的方式主要是和大厅事务和牌桌事务模块各自的“助理”进行通讯，发布自己的动作。同时，用户事务还负责对于模拟用户接收到的牌桌、平台信息更新，它需要将其反馈到真实用户那里，也就是通过操纵图形界面。

牌桌事务主要的工作是接受模拟用户的命令，将其通知牌桌游戏，同时，当接收到牌桌中的改变时，牌桌事务模块将会将内容反馈给牌桌中的各个用户。

大厅事务主要的工作和牌桌事务类似，它主要负责关于房间的创建、加入、退出等操作。

3.4.2 用户事务设计

```
public class Player {
    public TexasTerminalController controller; // UI 控制器
    public TexasGameHelper gameHelper; // 游戏助理
    public HallHelper hallHelper; // 大厅助理类
}
```

算法 3.9 用户类定义

用户类的定义如上，它始终需要记录为他服务的游戏助理和大厅助理，同时他有一个 UI 控制器，用在游戏助理和 UI 之间的交互，此举的作用是降低用户和游戏之间的耦合度。

```
create_room() {
    调用对应大厅助理的 createRoom 方法，并传入自身的引用
}
exit_room() {
    调用对应大厅助理的 exitRoom 方法，并传入自身的引用
}
update(String s) {
    调用 UI 控制器的 update()方法，传入 s
}
```

算法 3.10 命令行图形界面算法设计

```
用户输入事件处理句柄(用户输入) {
    根据用户输入创建一个用户传给游戏助理类的命令
    调用游戏助理类的 call 方法，并传入用户引用和命令
}

update(String s, boolean isYourTurn) {
    调用命令行界面的 update 方法，并传入 s
    if(isYourTurn) {
        调用命令行界面的用户输入事件方法
    }
}
```

算法 3.11 命令行图形控制器算法设计

3.4.3 牌桌事务设计

游戏助理类设计涉及到的数据结构及定义如下：

```
    public class TexasGameHelper {
        public TexasPokeTable table;
        public Room room;
        public TexasCommand command;
        public TexasTableEvent event;
        public int playerReadyCount;
    public int maxPlayerNumber;
    }
    对应的方法及伪代码如下：
        public void call(Player player, Player2TexasGameHelperCommand
commnad){
            if(command 显示有人准备好){
                准备好人数加一;
                if(所有玩家都准备好){
                    声明一个新的 TexasCommand 类;
                    TexasTableEvent 类调用 TexasCommand 类;
                }
            }
            else if(有人进行出牌动作){
                声明一个新的 TexasCommand 类;
                TexasTableEvent 类调用这个 TexasCommand 类;
                调用 broadcast 方法进行广播;
            }
        }

        public void broadcast(TexasTableEvent event){
            for(房间中的每一个玩家){
                调用 Player 类进行广播;
            }
        }
```

算法 3.12 游戏助理类算法设计

游戏助理命令类设计涉及到的数据结构及定义如下：

```
public class TexasCommand {
public List<String> nameList ;
public int whatHappen; //发生了什么
public int who; //谁，list 里面的顺序
public int action; //动作
public int chips;  //筹码数
}
public void newCommand(int who, Player2TexasGameHelperCommand command){
    显示有进行动作;
    this.who = who;
    this.action = command.action;
    this.chips = command.chips;
}
```

算法 3.13 游戏助理命令算法设计

3.4.4 大厅事务设计

```
public class HallHelper
{
public Hall hall;
}
对应的方法的伪代码如下：
protected TexasGameHelper assignTexasGameHelper(Room room)
{
新建一个游戏助手；
游戏助手负责 room
}

Public void createRoom(Player creator)
{
新建一个房间；
把创建者加入到这个房间；
打印房间创建成功的信息；
调用 assignTexasGameHelper()，创建一个游戏助手负责这个房间；
把大厅助手和游戏助手分配给玩家；
While 房间未满：
    新建一个机器人玩家；
    将机器人玩家添加到房间；
Endwhile;
}

Public void exitRoom(Player p)
{
将玩家从房间中删除；
打印玩家离开的消息；
While 房间未空：
    移除一个机器人；
    打印机器人离开房间的消息；
Endwhile;
将房间从大厅的列表中除去；
}


Public class Room
{
```

```
Public String roomID;
Public String roomPassport;
Public Type roomType;
Public ArrayList<Player>players;
}

对应的方法的伪代码如下：
Public Room(String roomID, int roomType)
{
将 roomID 赋给 this;
将 roomType 赋给 this;
}

Public Boolean addPlayer(Player p)
{
根据房间类型找到房间最大容纳人数;
If 添加一个玩家之后超过最大人数：
    打印房间人数已满的消息;
    Return false;
else
    将玩家添加到 players
Endif
}

Public void removePlayer(Player p)
{
将玩家从房间移除;
}

Public class Hall
{
Public ArrayList<Room> rooms;
}
```

算法 3.14 大厅逻辑算法