编号 _____

版本 第二次迭代

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　杨佳乐　　2020 年 5 月 5 日

校对　毛文月　　2020 年 5 月 6 日

审核　李泽雨　　2020 年 5 月 6 日

批准　章博文　　2020 年 5 月 7 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二次迭代的设计文档，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《第二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. 软件国标 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，本次迭代中的任务如表 2.1 所示：

表 2.1 第二轮迭代任务

序号	任务描述
1	GUI 部分、游戏逻辑部分、AI 策略部分继续完善。
2	各个部分和平台逻辑部分统一接口。
3	平台逻辑部分实现用户信息数据库的建立，并且通过 RMI 的方法实现用户远程连接服务器，可以注册和登录。

顶层类图反映了整体架构的设计，而图 2.2 的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。

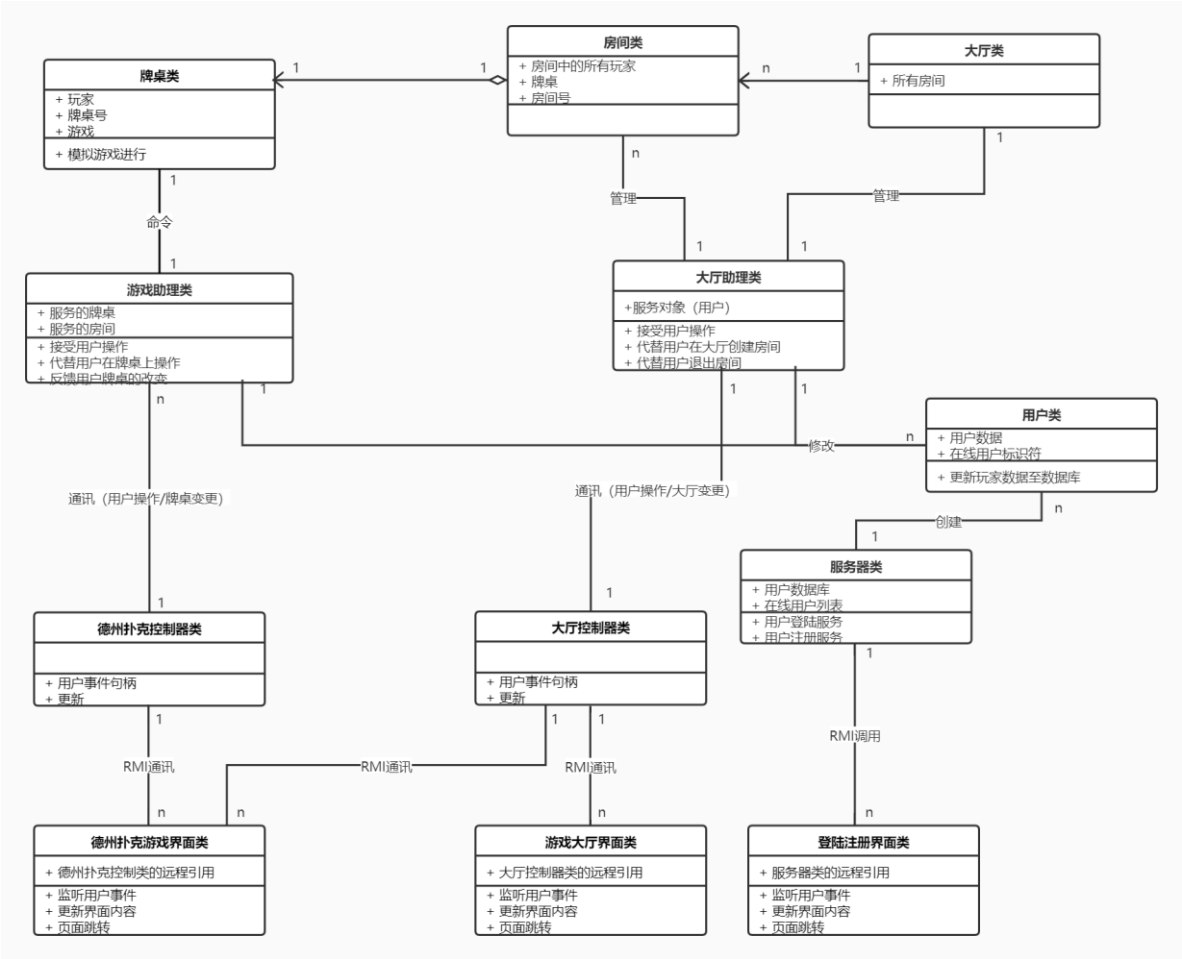
牌桌类
+ 玩家
+ 牌桌号
+ 游戏
+ 模拟游戏进行
房间类
+ 房间中的所有玩家
+ 牌桌
+ 房间号
大厅类
+ 所有房间
命令
管理
管理
游戏助理类
+ 服务的牌桌
+ 服务的房间
+ 接受用户操作
+ 代替用户在牌桌上操作
+ 反馈用户牌桌的改变
大厅助理类
+服务对象（用户）
+ 接受用户操作
+ 代替用户在大厅创建房间
+ 代替用户退出房间
用户类
+ 用户数据
+ 在线用户标识符
+ 更新玩家数据至数据库
修改
通讯（用户操作/牌桌变更）
通讯（用户操作/大厅变更）
创建
服务器类
+ 用户数据库
+ 在线用户列表
+ 用户登陆服务
+ 用户注册服务
德州扑克控制器类
+ 用户事件句柄
+ 更新
大厅控制器类
+ 用户事件句柄
+ 更新
RMI调用
RMI通讯
RMI通讯
RMI通讯
德州扑克游戏界面类
+ 德州扑克控制类的远程引用
+ 监听用户事件
+ 更新界面内容
+ 页面跳转
游戏大厅界面类
+ 大厅控制器类的远程引用
+ 监听用户事件
+ 更新界面内容
+ 页面跳转
登陆注册界面类
+ 服务器类的远程引用
+ 监听用户事件
+ 更新界面内容
+ 页面跳转

图 2.1 顶层类图

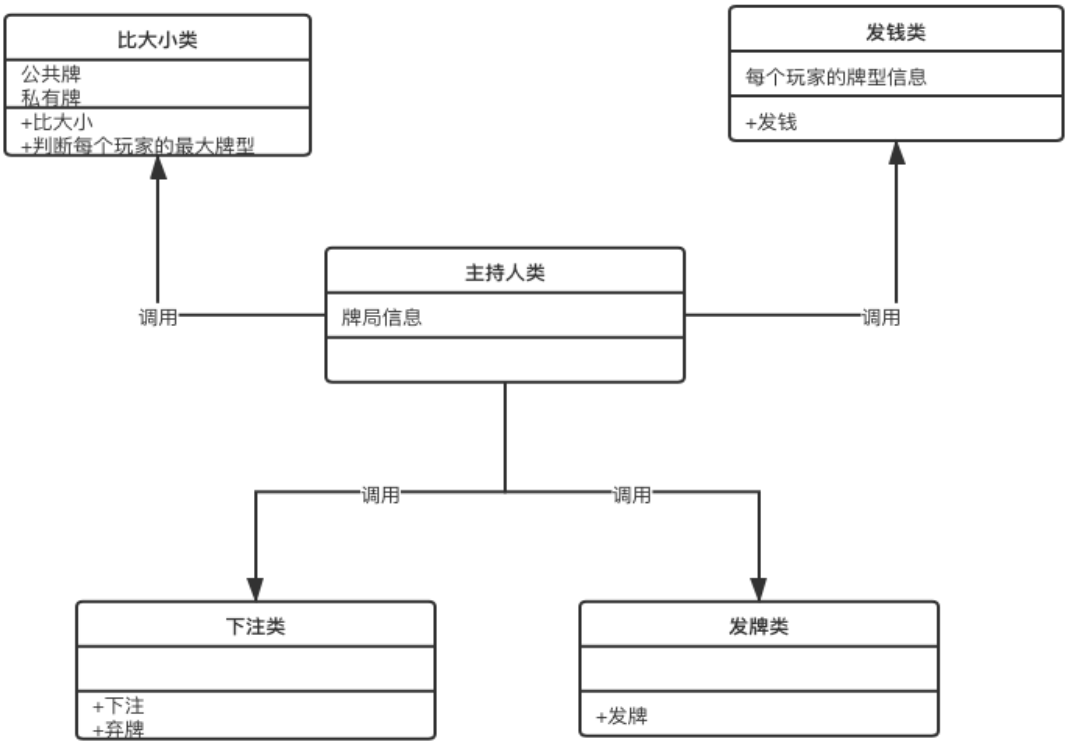
比大小类
公共牌
私有牌
+比大小
+判断每个玩家的最大牌型
发钱类
每个玩家的牌型信息
+发钱
主持人类
牌局信息
调用
调用
调用
调用
下注类
+下注
+弃牌
发牌类
+发牌

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 登录界面设计

图 3.1 登录界面

如图 3.1 所示，左上方为标题显示区，玩家先点击左下角的“注册账号”进行注册。然后在中间输入账号密码，最后点击“登录”。

3.1.2 注册界面设计

图 3.2 注册界面

如图 3.2 所示，输入三栏信息之后，点击“注册”后即可完成信息在服务器上面的录入。

3.1.3 消息提醒界面

会出现的消息提醒界面如图 3.3-图 3.5 所示。

图 3.3 注册成功提醒界面

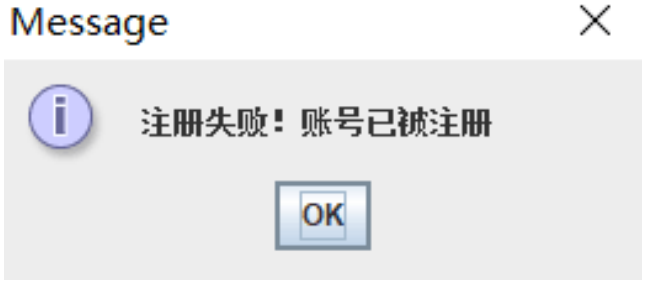

图 3.4 注册失败提醒界面

图 3.5 登录成功提醒界面

3.1.4 牌桌界面设计

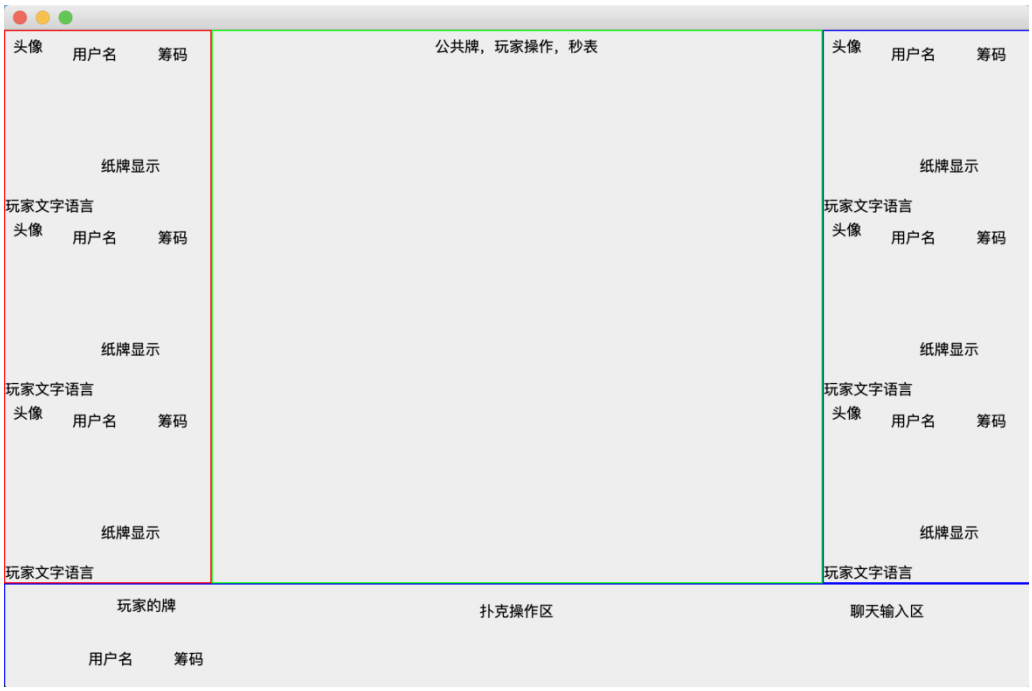

图 3.6 牌桌界面

如图 3.6 所示，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

上一轮的迭代中，游戏逻辑部分已经实现了数据结构设计与核心算法设计。在本轮迭代中，新增加两个算法。

第一个算法的功能是可以判断一个玩家的手牌加上牌桌上的公共牌之后，能够拼凑出来的最大牌型。

第二个算法的功能是比较每一位玩家能拼凑出来的最大牌型，选出其中牌型最大的作为获胜者。

3.2.1 最大牌型判断算法设计

```
boolean fourOfAKind()

int whichFace = 12 //记录是哪个牌面有四张牌
while (whichFace >=0)
  if( facenumber[whichFace] == 4)
    break
  else
    whichFace = whichFace - 1
  end if
end while
if( whichFace == -1)
  return false //不存在四张同牌面的牌，返回 false
end if

int firstSingle = 12 //找到除了四张以外的最大单牌
while(firstSingle >= 0 AND firstSingle >whichFace)
  if( facenumber[firstSingle] >=1)
    break
  else
    firstSingle = firstSing  -  1
  end if
end while

if(firstSingle == whichFace)
  firstSingle = firstSingle  -  1 //跳过凑成了四张的那个牌面
  while(firstSingle >= 0)
    if( facenumber[firstSingle] >=1)
      break
    else
    firstSingle = firstSingle  -  1
    end if
  end while
end if
```

遍历 7 张牌，找到 firstSingle 对应牌面的最大花色（因为这个牌面可能对应 3 张、2 张或者 1 张牌）的牌，也加入 PokerTypeResult.finalCards 中

在 PokerTypeResult 记录最大牌型为 4 张，记录 4 张的牌面与最大单牌的花色+牌面

```
boolean threeOfAKind ()

int whichFace = 12
while (whichFace >=0)
  if( facenumber[whichFace] == 3)
    break
  else
    whichFace = whichFace - 1
  end if
end while
if( whichFace == -1)
  return false //不存在 3 张同牌面的牌，返回 false
end if
把 facenumber 对应的牌面的 3 张牌加入 PokerTypeResult.finalCards 中

拷贝一个数组，facenumberCopy，其内容同 facenumber
facenumberCopy[whichFace] = 0

int firstSingle = 12 //找到除了 3 张以外的最大单牌
while(firstSingle >= 0)
  if( facenumberCopy[firstSingle] >=1)
    facenumberCopy[firstSingle] = 0
    break
  else
    firstSingle = firstSingle  -  1
  end if
end while

int secondSingle = 12 //找到除了 3 张以外的第二大单牌
while(secondSingle >= 0)
  if( facenumberCopy[secondSingle] >=1)
    facenumberCopy[secondSingle] = 0
    break
  else
    secondSingle = secondSingle  -  1
  end if
end while
把 firstSingle 与 secondSingle 对应的两张牌加入 PokerTypeResult.finalCards
中

在 PokerTypeResult 记录最大牌型为 3 张，记录 3 张的牌面与最大、次大单牌的花
色+牌面
```

```
boolean twoPair()
拷贝一个数组，facenumberCopy，其内容同 facenumber

int firstPair = 12
while (firstPair >=0)
  if( facenumberCopy[firstPair] == 2)
    facenumberCopy[firstPair] = 0 //清空，便于找下一对
    break
  else
    firstPair = firstPair- 1
  end if
end while
if( whichFace == -1)
  return false //不存在 2 张同牌面的牌，返回 false
end if

int secondPair = 12
while (secondPair >=0)
  if( facenumberCopy[secondPair] == 2)
    facenumberCopy[secondPair] = 0 //清空，便于找最大单牌
    break
  else
    secondPair = seocndPair- 1
  end if
end while
if( whichFace == -1)
  return false //不存在 2 对同牌面的牌，返回 false
end if

int firstSingle = 12 //找到除了 2 对以外的最大单牌
while(firstSingle >= 0)
  if( facenumberCopy[firstSingle] >=1)
    facenumberCopy[firstSingle] = 0
    break
  else
    firstSingle = firstSingle  -  1
  end if
end while
在 PokerTypeResult 记录最大牌型为 2 对，记录 2 对的牌面与最大单牌的花色+牌面
```

```
boolean onePair()
拷贝一个数组，facenumberCopy，其内容同 facenumber
int firstPair = 12
while (firstPair >=0)
  if( facenumberCopy[firstPair] == 2)
    facenumberCopy[firstPair] = 0 //清空，便于找单牌
    break
  else
    firstPair = firstPair- 1
  end if
end while
if( whichFace == -1)
  return false //不存在 2 张同牌面的牌，返回 false
end if
int firstSingle = 12 //找到除了 1 对以外的最大单牌
while(firstSingle >= 0)
  if( facenumberCopy[firstSingle] >=1)
    facenumberCopy[firstSingle] = 0
    break
  else
    firstSingle = firstSingle  -  1
  end if
end while

int secondSingle = 12 //找到除了 3 张以外的第二大单牌
while(secondSingle >= 0)
  if( facenumberCopy[secondSingle] >=1)
    facenumberCopy[secondSingle] = 0
    break
  else
    secondSingle = secondSingle  -  1
  end if
end while

int thirdSingle = 12 //找到除了 3 张以外的第二大单牌
while(thirdSingle >= 0)
  if( facenumberCopy[thirdSingle] >=1)
    facenumberCopy[thirdSingle] = 0
    break
  else
    thirdSingle = thirdSingle  -  1
  end if
end while
在 PokerTypeResult 记录最大牌型为 1 对，记录 2 对的牌面与最大、次大、第三大
```

```
单牌的花色+牌面
    boolean highCard()

    拷贝一个数组，facenumberCopy，其内容同 facenumber
    从第 12 位开始，到 0 位截止，依次找五张最大单牌
```

算法 3.1 最大牌型判断算法

3.2.2 比较大小算法

```
    public boolean straightFlush()
        {
            遍历花色数组判断有无同花
            if(没有同花)
            {
                return false
            }

            挑出同花牌
            用挑出的同花牌组建新的牌组

            if(新的牌组里有 10，J，Q，K，A 这五张牌)
            {
                pokerTypeResult.type = 同花顺
                pokerTypeResult.straightMaxNumber = 14
                pokerTypeResult.finalCards 新增这五张牌
                return true
            }
            for(i 从牌 K 向下遍历至牌 5)
            {
                if(存在连续的五张牌)
                {
                    pokerTypeResult.type = 同花顺
                    pokerTypeResult.straightMaxNumber = i
                    pokerTypeResult.finalCards 新增这五张牌
                    return true
                }
            }
            return false;
        }
```

```
public boolean flush() {
    遍历花色数组判断有无同花
    if(无同花)
    {
        return false
    }
    else 有同花
    {
        挑出同花牌
        用挑出的同花牌组建新的牌组
        if(新的牌组里有 A)
        {
            将 A 加入被选中的牌组
            再从剩下的牌中从大到小选出四张
        }
        else
        {
            从牌组中从大到小选出五张
        }
        pokerTypeResult.type = 同花
        pokerTypeResult.firstSingle = 被选中牌组中第一张的牌面
        if(pokerTypeResult.firstSingle == 0)
        {
            pokerTypeResult.firstSingle = 14;
        }
        pokerTypeResult.secondSingle = 被选中牌组中第二张的牌面
        pokerTypeResult.thirdSingle = 被选中牌组中第三张的牌面
        pokerTypeResult.fourthSingle = 被选中牌组中第四张的牌面
        pokerTypeResult.fifthSingle = 被选中牌组中第五张的牌面
        return true
    }
}

public boolean straight() {
    if(存在 10, J, Q, K, A 五张牌)
    {
        pokerTypeResult.type = 顺子
        pokerTypeResult.straightMaxNumber = 14
        从牌组中选出这五张牌加入被选中的牌组中
        return true
    }
    for(i 从牌 K 向下遍历至牌 5)
    {
        if(存在连续的五张牌)
```

```
            {
                从牌组中选出这五张牌加入被选中的牌组中
                pokerTypeResult.type = 顺子
                pokerTypeResult.straightMaxNumber = i
                return true
            }
        }
        return false;
    }

    public boolean fullHouse() {

        if(有三张 A)
        {
            pokerTypeResult.threeNumber = 0;
        }
        else
        {
            for(i 从牌 K 向下遍历至牌 2)
            {
                if(有三张相同数字的牌)
                {
                    whichFace1 = i;
                    pokerTypeResult.threeNumber = i;
                    break;
                }
            }
            if(whichFace1 没修改过)
            {
                return false;
            }
        }


        if(有三张 A)
        {
            for(i 从牌 K 向下遍历至牌 2)
            {
                if(有两张相同的牌)
                {
                    whichFace2 = i;
                    pokerTypeResult.twoNumber = i;
                    break;
                }
```

```
            }
        }
        else
        {
            for(i 从牌 K 向下遍历至牌 A)
            {
                if(i == whichFace1)
                {
                    continue;
                }
                if(有两张相同的牌)
                {
                    if(i == 0)
                    {
                        whichFace2 = i;
                        pokerTypeResult.twoNumber = 0;
                        break;
                    }
                    else
                    {
                        whichFace2 = i;
                        pokerTypeResult.twoNumber = i;
                        break;
                    }

                }
            }
        }
        if(whichFace2 没修改过)
        {
            return false;
        }

        选出这五张牌加入到被选中的牌组中

        if(pokerTypeResult.threeNumber == 0)
        {
            pokerTypeResult.threeNumber = 14;
        }



        if(pokerTypeResult.twoNumber == 0)
        {
            pokerTypeResult.twoNumber = 14;
```

```
            }

            pokerTypeResult.type = 葫芦
            return true;
        }
```

算法 3.2 比较大小算法

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 3.3 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 3.4 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 3.5 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 3.6 获胜概率计算算法

3.3.4 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

平台逻辑部分在前一次迭代的基础上，继续完成两部分的工作。首先是使用 RMI 的方法将用户和服务器分离开。其次是服务器运行程序后建立用户数据库。最后和 GUI 部分结合，初步实现用户在本地通过 GUI 连接服务器，在服务器上注册账户并且成功登录的功能。

3.4.1 远程连接

RMI 原理介绍：

RMI 是远程方法调用，能够使在一个 JVM 上运行的程序远程调用另一个 JVM 上运行的程序的方法。我们称前者为顾客，后者为服务者。对于前者，它只需要知道接口的定义，对于后者，它将提供一个类实现对应的接口。服务者在实例化该类后，将其通过 RMI 技术注册（register）到自己的某一端口并指定服务名称，便可以提供远程方法调用的服务；然后顾客便可以通过 RMI 技术查询（lookup）该服务，只需要找到对应端口的某个服务，便可以获取一个本地代理，通过代理远程调用那些方法。

本项目中使用 RMI 的例子

在我们的项目中，我们在登陆与注册上使用了 RMI 技术。服务器是服务者，提供登陆和注册服务，而各个用户是顾客，需要使用登录和注册服务。

服务器类需要实现图 3.7 所示的接口,分别代表登录和注册。

```
public interface IServer{
    String service_login(String id, String passWord);
    boolean service_register(String name, String id, String passWord);
}
public Server implements IServer{}
```

图 3.7 登录和注册接口

随后通过 Java 自带的 RMI 库注册 RMI 服务至本地端口,在图 3.8 所示的例子中，我们将服务注册到本地的 12200 端口，并将服务命名为 Server.

```
LocateRegistry.createRegistry(12200);
Naming.bind("//localhost:12200/Server", this);
```

图 3.8 端口举例

而对于用户，只需要使用 RMI 的查询，便可以获得一个 IServer 的远程接口，并可以通过这个远程接口调用所需要的方法。在图 3.9 所示的例子中，我们本地的用户通过本地的 12200 端口的 Server 查询到了对应的服务，并将返回的接口绑定到 s，并调用了登录方法。

```
IServer s = Naming.lookup(“//localhost:12200/Server”)
s.login(“xiaoming”, “123456”)
```

图 3.9 查询登录举例

3.4.2 建立数据库

本次迭代中建立用户信息数据库使用 SQLite 和 JDBC。SQLite 是一个软件库，实现了自给自足的、无服务器的、零配置的、事务性的 SQL 数据库引擎。JDBC 全称 Java Database Connectivity，是 Java 语言用来访问数据库的应用程序接口。

在 Server 类中定义了一些操作数据库的方法：

表 3.1 服务器类访问数据库方法

方法名	描述
Service_deploy()	建立一个 PLAYERS 表，包含属性：ID、NAME、PASSWORD、MONEY、WIN、LOSE。
Service_login()	在数据库中判断账户是否存在，然后返回一个加密之后的字符串。
Service_update_record()	每一局游戏结束后用来更新玩家的胜负场数（WIN 和 LOSE）。
Service_update_money()	每一局游戏结束后用来更新玩家剩余的筹码数。
Service_register()	注册账户，首先判断输入的 ID 是否已经存在，然后在 PLAYERS 表里面插入一个元组。
Encipher()	Server 类维持一个哈希表，索引为加密后的字符串（当前系统时间+用户名），值为玩家 ID。该函数输入玩家基本信息后返回加密字符串。
Decipher()	解析字符串。输入为加密字符串，查找哈希表中的玩家 ID 并且返回。