浙江大学

扑克牌游戏
项目设计说明

小组 6

学生姓名：沈韵泓　　学号：3200104392

学生姓名：王柯棣　　学号：3200105119

学生姓名：魏鼎坤　　学号：3200105652

学生姓名：赵伊蕾　　学号：3200104866

2021-2022 春夏学期　2022 年 06 月

目录

1 需求分析

在本次面向对象程序设计的 Project 中，我们小组根据递交的中期报告完成了对“扑克牌游戏”项目的设计与开发。结合开发的实际情况，我们对 Project 中期报告中提出的待实现功能目标进行了合理的增删。最终，该“扑克牌游戏”项目实现了以下功能：

1.1 友好的图形化用户界面

启动程序后，系统将首先进入标题页，用户可以通过鼠标操作实现不同界面间的切换，并随时通过返回按钮回到前移界面修改已经做出的选项。不同界面间遵从统一的 UI 设计规范，提供一致的视觉效果，通过文字、颜色两种视觉交互方式达成了良好的人机交互效果。

1.2 支持两种扑克牌游戏

程序实现了 21 点（BlackJack）与德州扑克两种扑克牌游戏模式，为用户提供了趣味性与思考性并存的两种不同选择。界面中实时显示玩家当前拥有的筹码数量与庄家当前的明牌，并在对应轮次以图形化形式展现玩家拥有的手牌，以辅助玩家做出决策、提供良好的游戏体验。

1.3 支持 4 名玩家单机参与游戏

本程序支持最多四名玩家在本地同时参与游戏，每名玩家由玩家编号 Player 1 至 Player 4 进行标识。处于活跃状态的玩家在自己的轮次可根据自己拥有的筹

码、手牌与庄家明牌的情况做出对应决策以赢取筹码。处于破产/投降/停牌状态的玩家在对应轮次仅进行对应信息的展示，不能进行相关操作。

1.4 支持少于 4 名玩家参与游戏，不足人数由计算机扮演

当本地玩家不足四名时，将由对玩家数量进行补齐，由编号 AI1 至 AI3 进行标识。处于活跃状态的 AI 将在同一界面展示其在本轮中做出的所有决策、自功更新对应数据并跳转至下一玩家。处于破产/投降/停牌状态的 AI 将仅展示其对应状态及信息，不能进行相关操作。

2 总体设计

2.1 页面架构

本程序共包含标题页、游戏选择页、21 点游戏页、德州扑克游戏页四个页面。为保证页面刷新的相应速度，所有页面均在同一窗口中实现，以图层进行区分。

2.1.1 标题页

本页向玩家展示程序 logo，并支持用户自定义用户名。

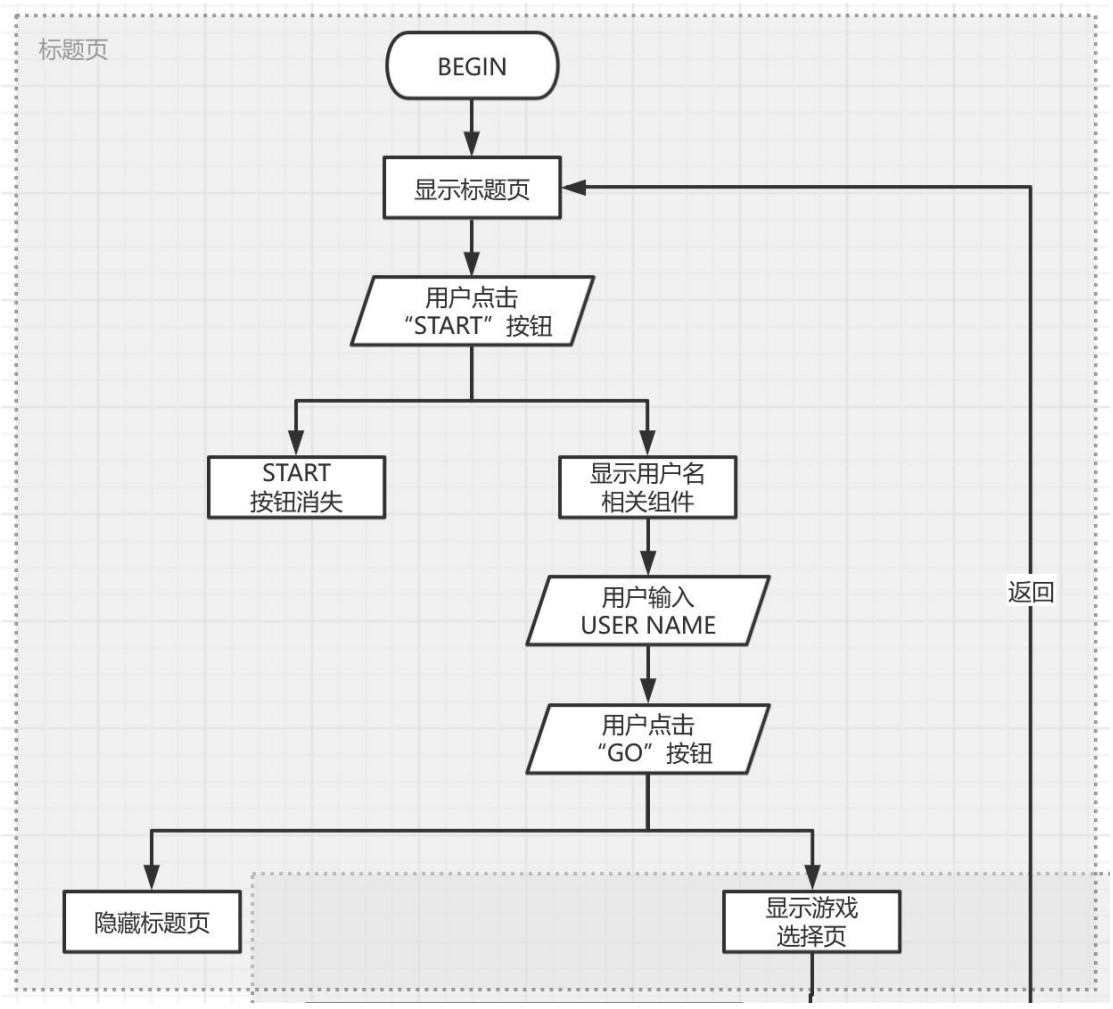


图 2.1 标题页流程图

2.1.2 游戏选择页

本页支持玩家在两种支持的游戏（21 点/德州扑克）与游戏模式（本地/联机）中做出选择。本地游玩时需指定玩家量。

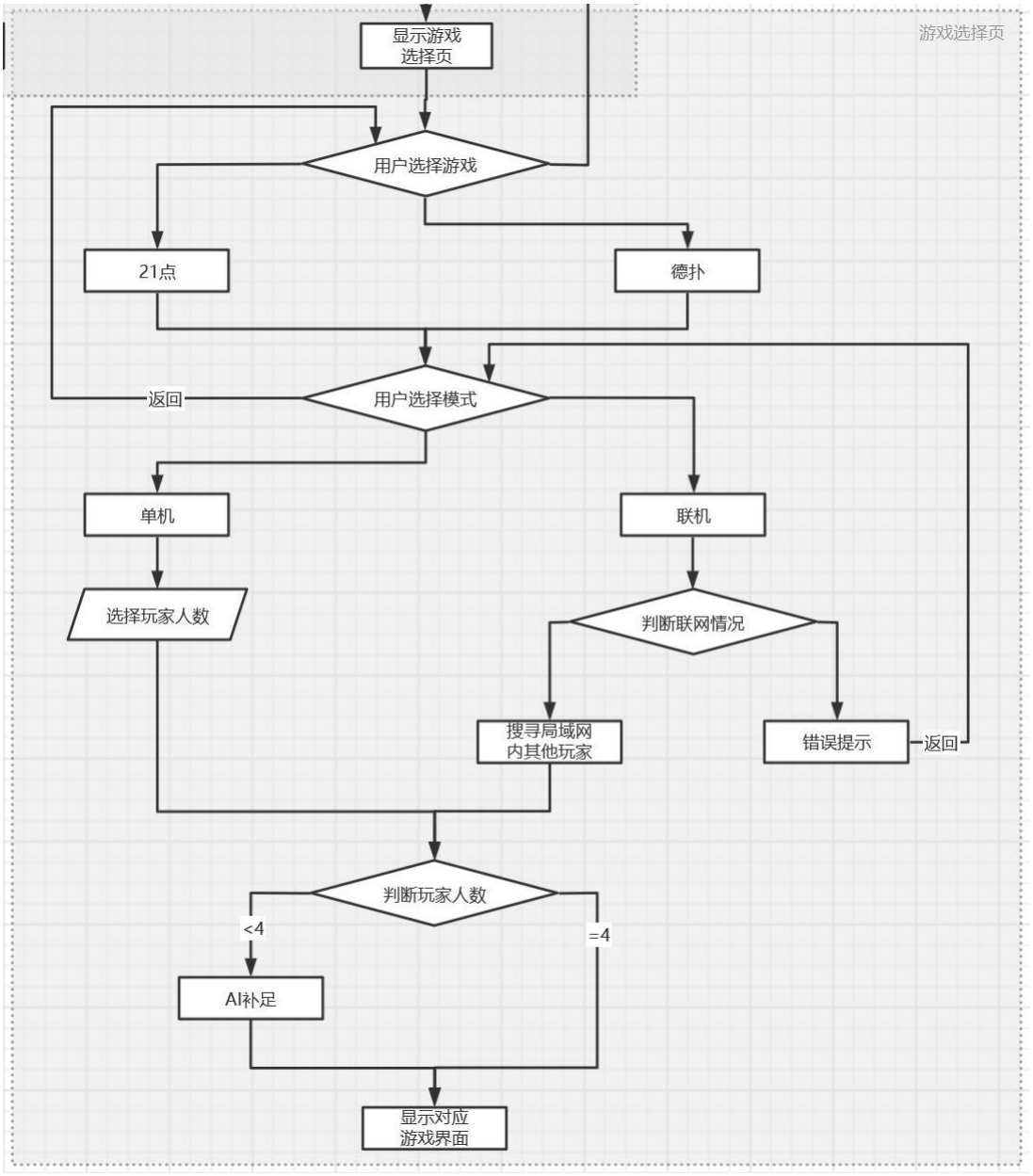


图 2.2 游戏选择页流程图

2.1.3 21 点游戏页

本页面支持用户进行 21 点游戏。

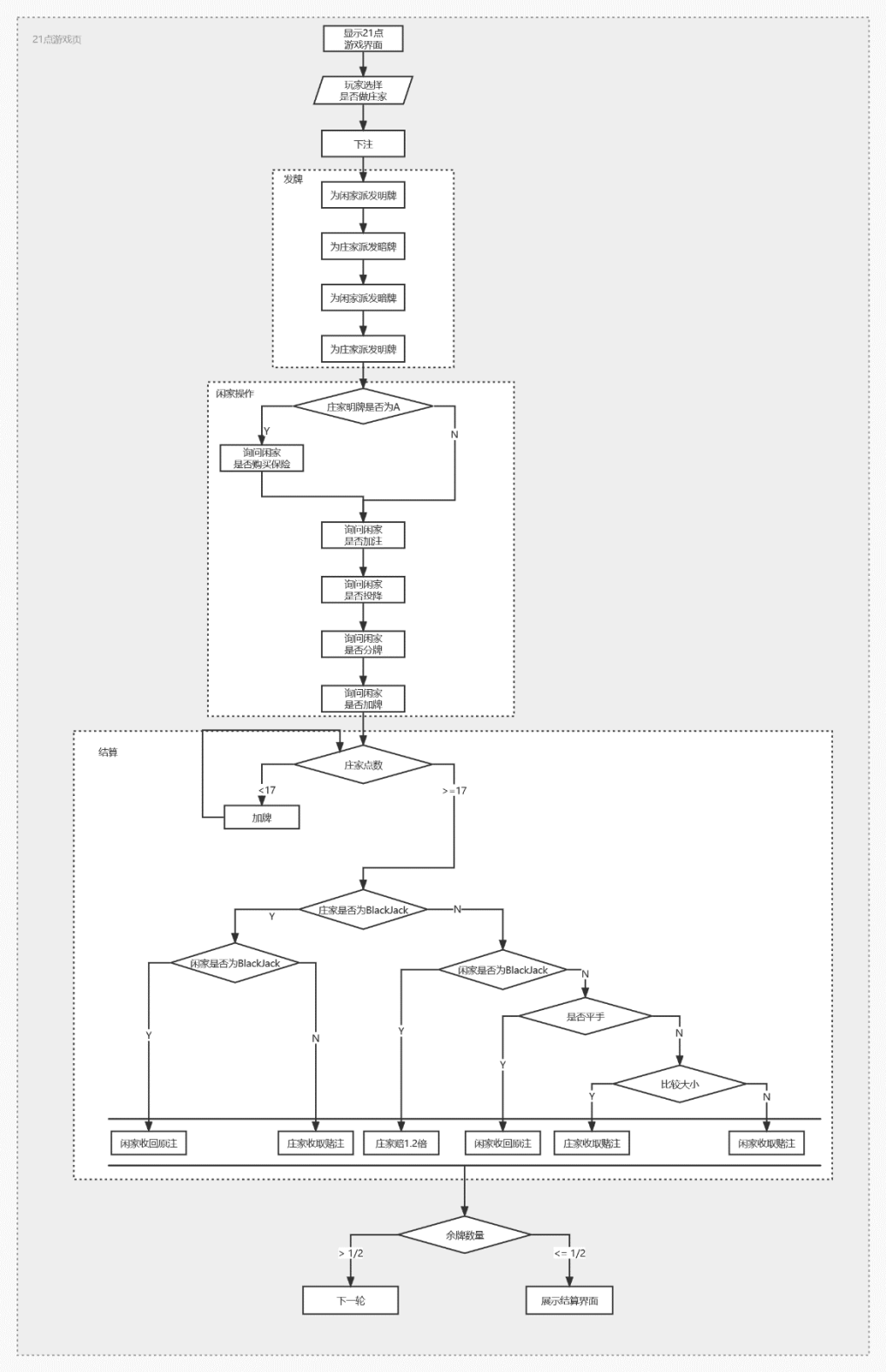


图 2.3 21 点游戏页流程图

2.1.4 德州扑克游戏页

本界面支持用户进行德州扑克游戏。

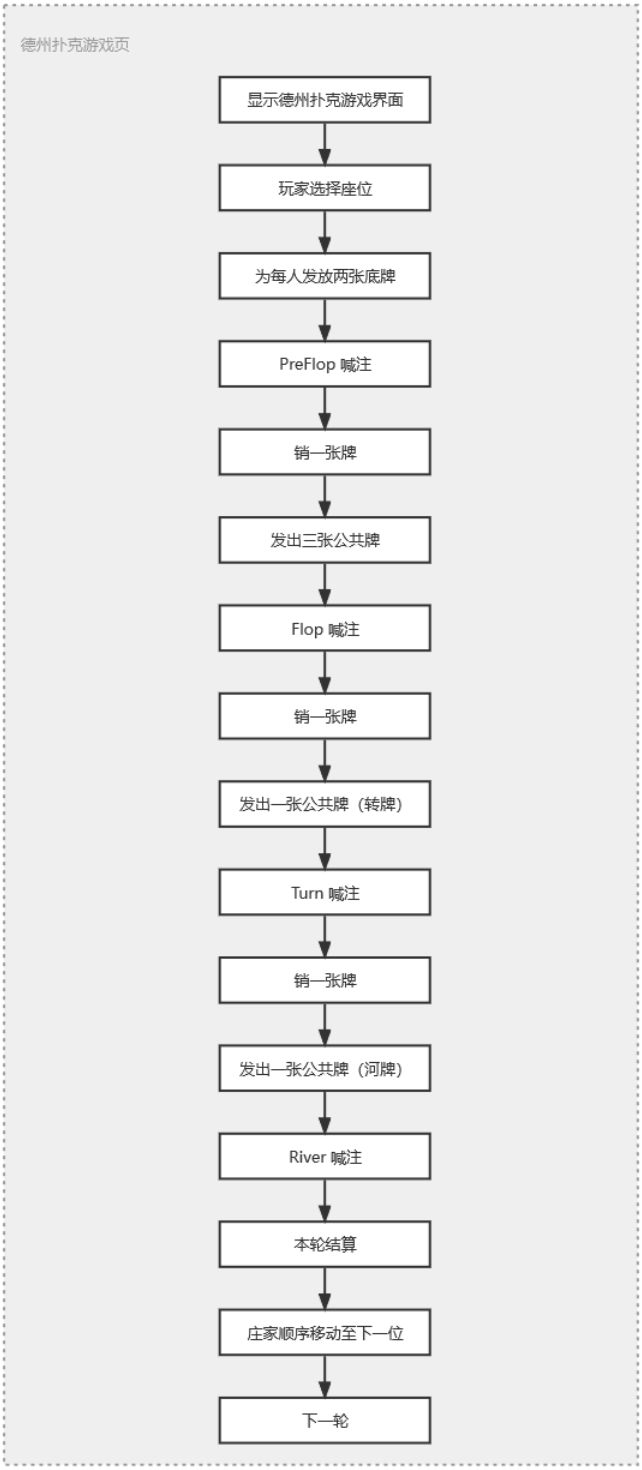


图 2.4 德州扑克游戏页流程图

3 系统模块设计

3.1 21 点（BlackJack）

3.1.1 Player 类

Player 类的基类当中要求实现以下函数：

```
get_status();
//获取当前玩家状态
get_card();
//拿牌
stop_card();
//停牌
bet(int n);
//下注
surrender();
//投降
judge();
//判断手牌价值
```

上述函数能够给出一个玩家的所有操作，但本身玩家的属性还需要进行维护，包括以下部分：

```
int status;
int money;
int bet_money;
vector<Card> Com_Card;
```

在具体实现中还需要注意几个细节：

- 创建玩家类的时候需要传入至少一个参数：玩家当前资金
- 当玩家的资金为负的时候只能投降
- 当玩家的手牌已经爆掉之后，不允许拿牌

接口层代码展示：

```
class player
{
public:
    int status;
    int money;
    int bet_money;
    vector<Card> Com_Card;
    player(int n)// initial the player with his monney
    {
        money = n;
        status = 0;
        bet_money = 0;
    }
    int get_status()//return status which means this player surrender or
not
    {
        return status;
    }
    void get_card()// get a card from cards
    {
        if (status != 0)
        {
            return;
        }
        Com_Card.push_back(liscening());
    }
    void stop_card()// no more get card
    {
        status = 1;
    }
    void bet(int num)// bet money
    {
        if (status != 0)
        {
            return;
        }
        bet_money += num;
    }
    void surrender()// surrender and status change
    {
        status = 2;
    }
    void start()
    {
```

```
        get_card();
        get_card();
    }
    int judge(void);
};
int player::judge(void) // return the score of cards
{
    int sum = 0;
    int down=0;
    vector<Card>::iterator it;
    for (it = Com_Card.begin(); it != Com_Card.end(); it++)
    {
        int a=(*it).num>10?10:(*it).num;
        if(a==1)
        {
            a=11;
        }
        sum +=a;
        if(a==11)
        {
            down-=10;
        }
    }
    if(sum>21)
    {
        while(sum>21 && down!=0)
        {
            sum-=10;
            down+=10;
        }
    }
    if(sum>21 || status==2)
    {
        sum=0;
    }
    return sum;
}
```

3.1.2 Card 类

- Card 类为牌堆，支持发牌、洗牌等功能

- Card 类需要为其他所有类的友元类，方便调用

洗牌的实现：

- 洗牌的时候先将所有牌按照次序放置，也就是 1、2、3、4、5、6……

- 进行洗牌的操作如下：随机得到两个小于 54 的数，然后交换两张牌的位置

- 重复上述交换牌的操作，直到整个牌堆洗清

```
void Card::shuffing()// shuffing the cards
{
    unsigned seed=time(0);
    srand(seed);
    for (int i = 0; i < 8; ++i) // set the num of cards
    {
        for (int j = 1; j <= 13; ++j)
        {
            card[j + i * 13].num = j;
        }
    }
    for (int i = 0; i < 8; ++i) //set the type of cards
    {
        for (int j = 1; j <= 13; ++j)
        {
            card[j + i * 13].type = i%4;
        }
    }
    for (int ii = 0; ii < 500; ++ii) // rand two numbers and change the
this two cards
    {
        int r1 = rand()%52 + 1;
        int r2 = rand()%52 + 1;
        int t = card[r1].num, w = card[r1].type;
        card[r1].num = card[r2].num;
        card[r1].type = card[r2].type;
        card[r2].num = t;
        card[r2].type = w;
    }
}
```

发牌的实现：

- 发牌时需要随机获取一个数，并将该位置的牌 return

- 需要维护一个已发的牌的总数，当总数超过 54 时及时报错终止程序

```
card Card::liscening()// get card
{
    final++; // do check on card that has been got
    int a=card[final].num;
    if(a>=10)
    {
        count--;
    }
    if(a<7)
    {
        count++;
    }
    return card[final];
}
```

3.1.3 Bnaker 类

Banker 类需要包含以下成员：

- 明牌

- 暗牌

- 发牌结算

- 资金清算

在实现时候需要注意以下细节：

- 发牌清算的时候，庄家需要一直摸牌到手牌大小超过 17

- 资金清算时若自身破产，则需要按照赔付的比例将剩余资金进行赔付（有多少赔多少）

接口层代码展示：

```
class Banker
```

```
{
public:
    vector<Card> com_card;
    Card under_card;
    void get_card()// get card from cards
    {
        com_card.push_back(liscening());
    }
    void start()// initial to get two card and one as the under card
    {
        under_card=liscening();
        get_card();
    }
    int judge(void);
};
int Banker::judge(void)// caculate the scores
{
    int sum = 0,down=0;
    int a=under_card.num>10?10:under_card.num;
    if(a==1)
    {
        a=11;
    }
    if(a==11)
    {
        down-=10;
    }
    vector<Card>::iterator it;
    for (it = com_card.begin(); it != com_card.end(); it++)
    {
        a=(*it).num>10?10:(*it).num;
        if(a==1)
        {
            a=11;
        }
        sum +=a;
        if(a==11)
        {
            down-=10;
        }
    }
    if(sum>21)
    {
        while(sum>21 && down!=0)
```

```
        {
            sum-=10;
            down+=10;
        }
    }
    if(sum>21)
    {
        sum=0;
    }
    return sum;
}
```

3.1.4 界面设计

图 3.1 21 点游戏背景

本模块的界面设置主要分为三类：玩家界面、过渡界面、庄家界面，其中玩家界面又可分为操作类与过场类。以上界面均通过 frame 组件进行伪图层实现，操作流程与界面跳转使用 button 的 click 事件与 timer 进行实现。

3.1.3.1 玩家界面

玩家界面分为正常（可操作）与异常（仅展示）两类。

- 正常操作界面

页面按近似于 1：1 的比例划分为上下两个部分，上侧显示庄家情况（明牌及总点数），下侧以卡片形式显示玩家现状。

当玩家点数小于 21 点且未选择投降或停牌时，显示正常操作界面。右侧相关数据通过 label 控件进行实现，当玩家执行相应操作时应及时进行数据更新。

玩家手牌同样通过 frame 实现，在更换玩家时预先将该玩家的手牌信息加载到窗口上，并在玩家选择加牌操作后对新抽取的手牌进行追加显示。

玩家通过点击卡片中部的 SKIP/DO IT 按钮推进游戏流程。

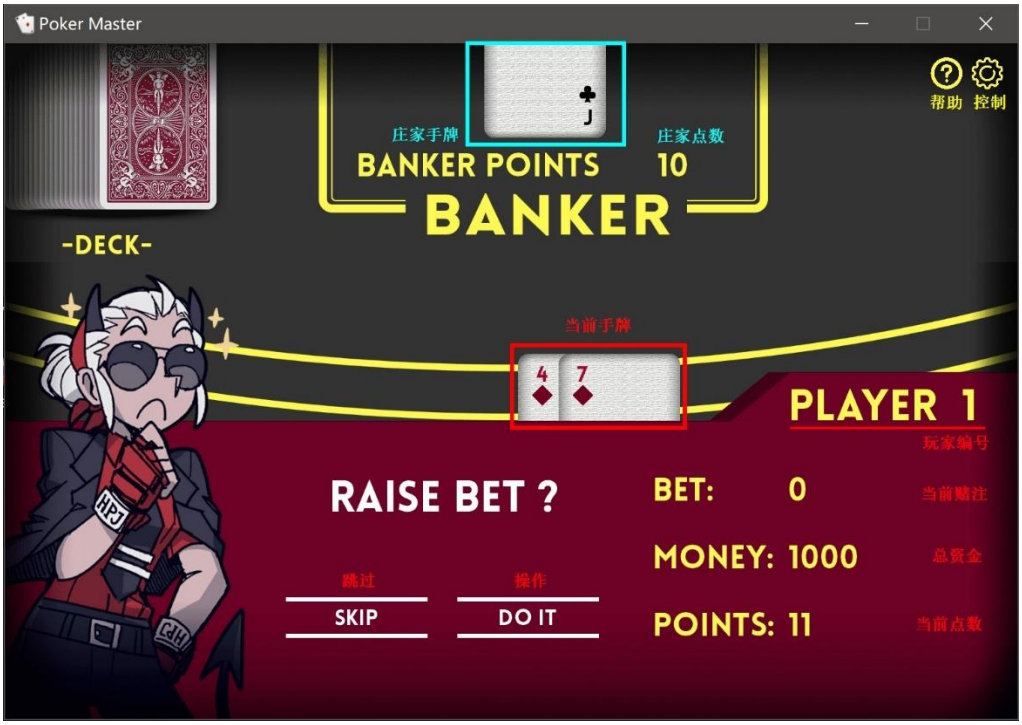


图 3.2 正常玩家界面

加注界面额外包含一个检查输入值合法性的文本框与 ADD 按钮。确保玩家仅输入非空、在整形范围内，且不超过玩家现有资金总量。

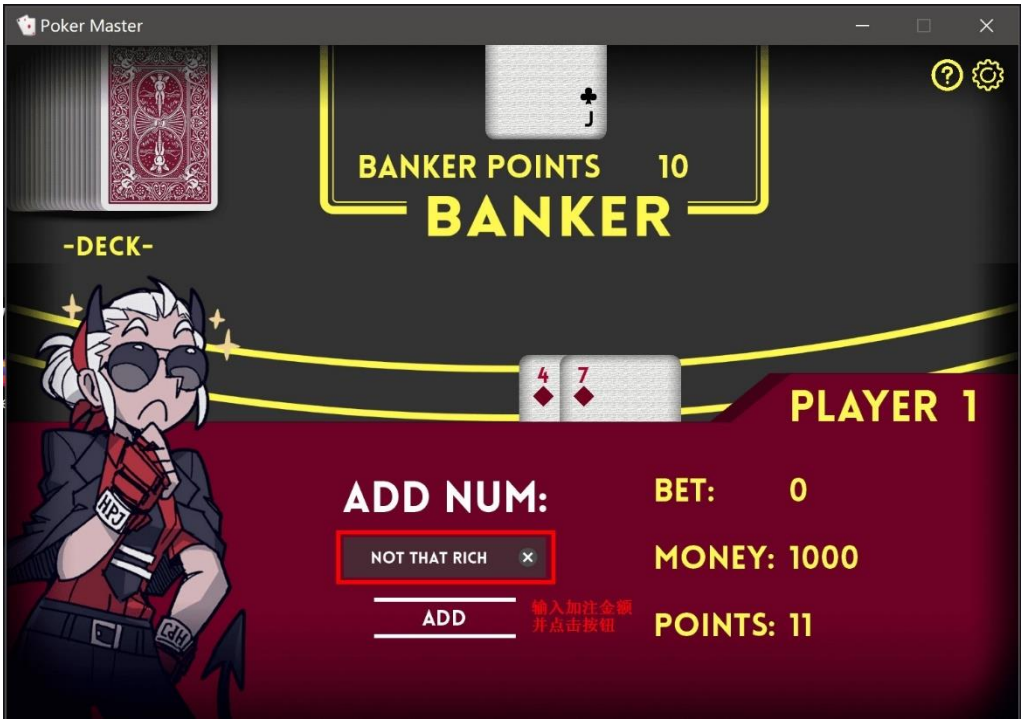


图 3.3 加注界面

- 异常展示界面

当玩家点数大于 21 点或选择停牌或投降时，显示玩家异常界面。中央的标签显示了玩家所处异常状态的类型：BURSTED（点数超过 21）/ SURRENDER（投降）/ STAND（停牌）。右侧状态栏与中央的手牌显示逻辑与正常玩家界面一致。

由于异常界面不允许玩家进行操作，我们在此处采用 timer 推进游戏流程。载入该异常玩家时，发出 timer->start 信号，并在 timeout 时调用函数显示下一界面。

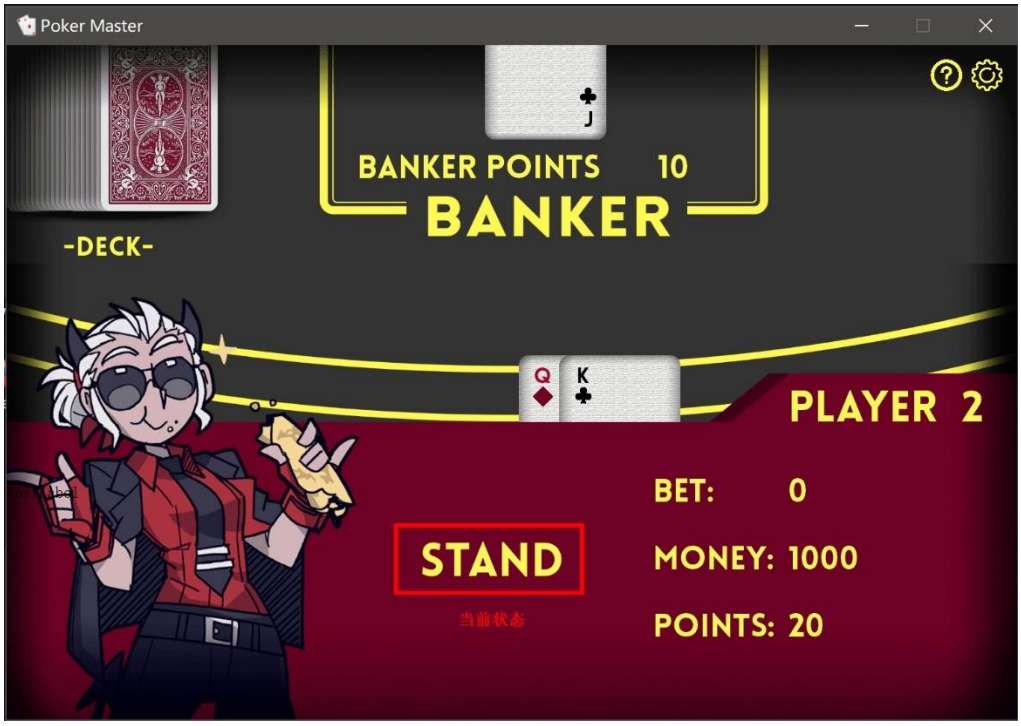


图 3.4 异常玩家界面

3.1.3.2 过渡界面

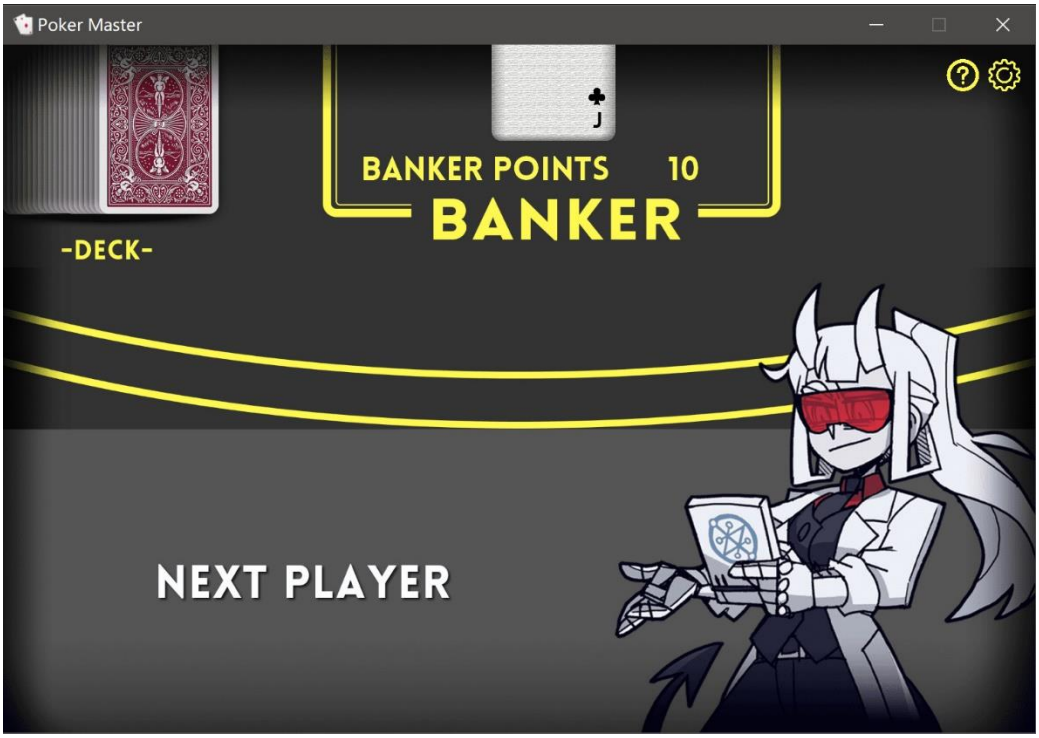


图 3.5 过渡界面

3.1.3.3 庄家界面

在检测到四位玩家（包括 AI 玩家）均处于异常状态时，调用函数进入庄家结算页。在庄家点数超过 17 点前，持续调用函数令庄家抽卡，并实时将抽卡情况反馈在页面中部。

当庄家点数超过 17 点时进行结算，并将结算结果反馈至页面中部的标签。

玩家此后可点击 CONTINUE 进入下一局势游戏，或点击 MENU 返回主菜单对游戏模式/玩家人数等设置进行修改。

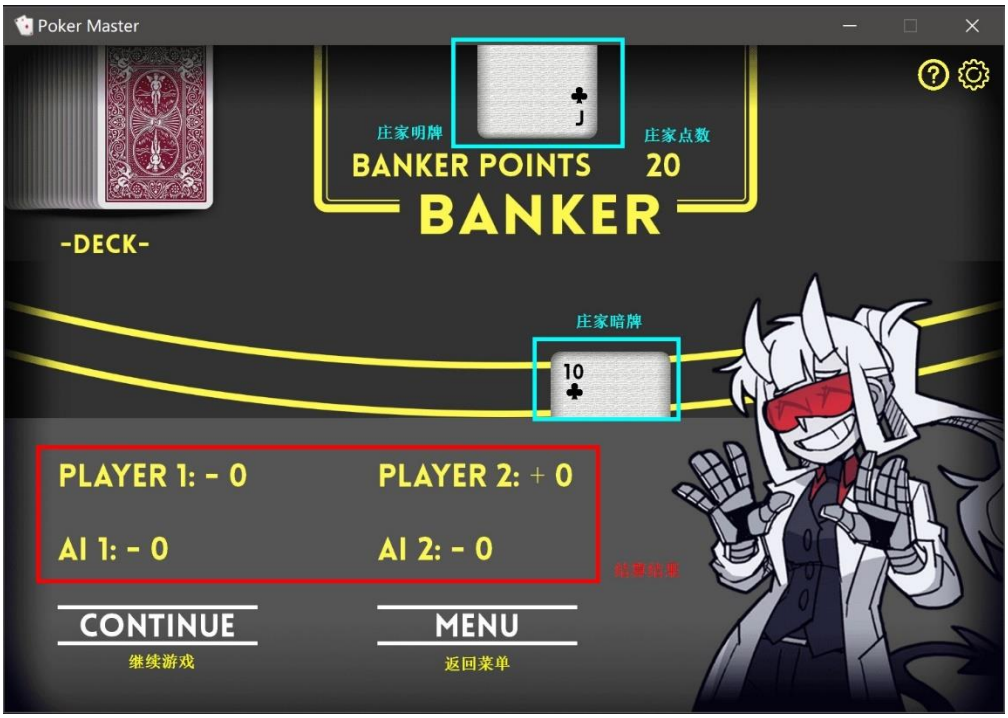
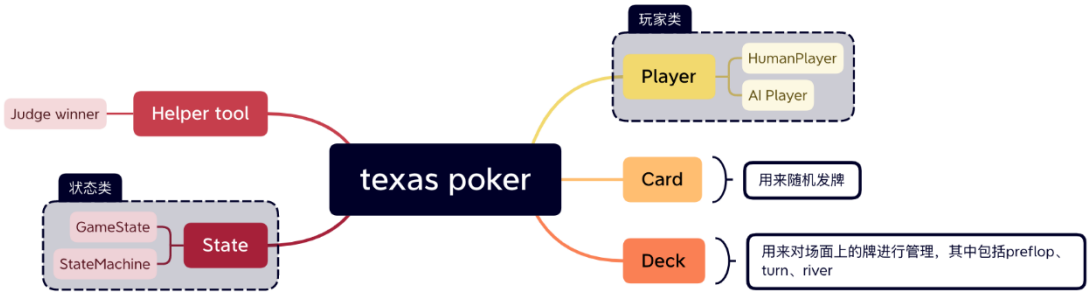


图 3.6 庄家界面

3.2 德州扑克

3.2.1 类的架构

3.2.1.1 类的架构与继承关系



3.2.1.2 Player 类

```
4  class Player
5  {
6  public:
7      Player(size_t id, size_t money) : id(id), money(money), bet(0),
   folded(false),
8                                        lastCommand(NONE){}
9      /* Returns the type of action taken.(raise, call or fold*/
10     virtual Command playTurn() = 0;
11     /* Give the two hand cards to player. */
12     void setHand(texas_Card* card1, texas_Card* card2)
13     {
14         hand.clear();
15         hand.push_back(card1);
16         hand.push_back(card2);
17     }
18     /* folded: return false; */
19     bool isActive() const {return !folded;}
20
21     /* Set the player's bet.*/
22     void setBet(size_t bet);
23
24     virtual string getName();
25     size_t getBet() const {return bet;}
```

```
26      size_t getMoney(){return money;}
27    size_t getId() const {return id;}
28    vector<texas_Card*> getHand() const {return hand;}
29    /* Return the cards in hand and the cards on the table. */
30    vector<texas_Card*> getCards() const;
31      Command getLastCommand() const{return lastCommand;}
32
33      /* Resets players's last command to NONE.*/
34      void resetLastCommand()  {lastCommand = NONE;}
35      void resetFold(){folded = false;}
36    void setLastCommand(Command c){lastCommand = c;}
37      void giveMoney(size_t amount){money += amount;}
38
39    void raise(); /* Raise / Bet*/
40    void call();  /* Call / Check */
41    void fold();  /* Fold */
42
43  protected:
44      const size_t id;
45      vector<texas_Card*> hand;
46      size_t money;
47      size_t bet;
48      bool folded;
49      bool callWasCheck;
50      Command lastCommand;
51  };
```

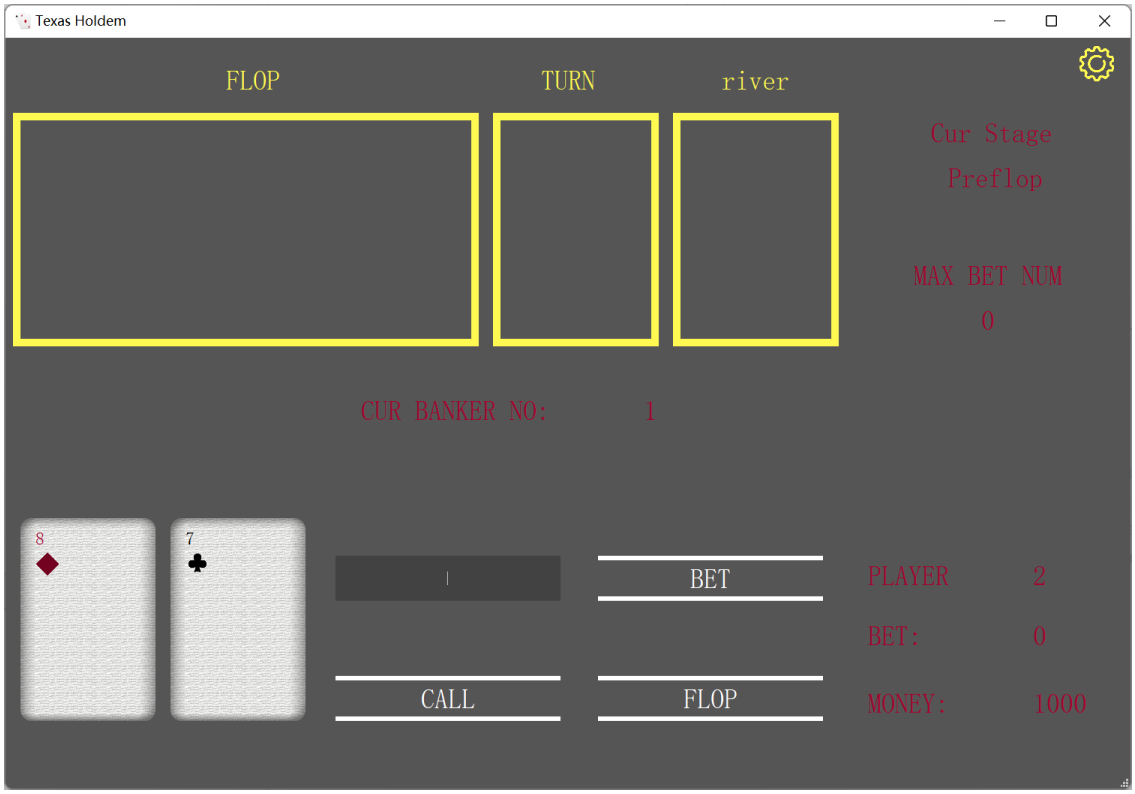
3.2.1.3 HumanPlayer 类

```
4      class HumanPlayer : public Player
5      {
6      public:
7          HumanPlayer(size_t id, size_t money): Player(id, money) {}
8          Command playTurn();
9          string getName(){return "Human Player " +
       to_string(getId());}
10     };
```

因为玩家的种类有 Human 玩家与 AI 两种，所以在逻辑实现上采用了子类父类继承的方式。游戏规则中是 AI 将自动补齐人类玩家不足的人数，保证对局的完整性。`Player` 父类中有多个虚函数，以便子类 `override`，其中包括对牌的

操作，还有用户编号等信息的返回。

3.2.2 界面设计



游戏的整体界面如图所示，左上角是公共牌区域，在相应轮次开始之后会被亮出来。在其右方是当前轮次的状态以及最大赌注的提示，当玩家 CALL 时追平的便是此处提示的最高赌注。而游戏界面的中间是当前的庄家 ID。

界面下方是玩家的相应操作区域，右侧是当前玩家的 ID、赌注以及剩余金额，左侧则是该玩家所拥有的底牌。界面中间是该玩家可以进行的操作，分别是 `BET`、`CALL` 和 `FLOP`，相应作用分别如下：

- BET：

  当前玩家可以输入一定的金额，并且将相应金额投注到奖池当中。每一局游戏之中，第一个非投降的玩家必须投入一定数量的金额，用以让后续的玩家跟注。之后每个玩家的 BET 都必须投注比之前最高金额更高的数额。

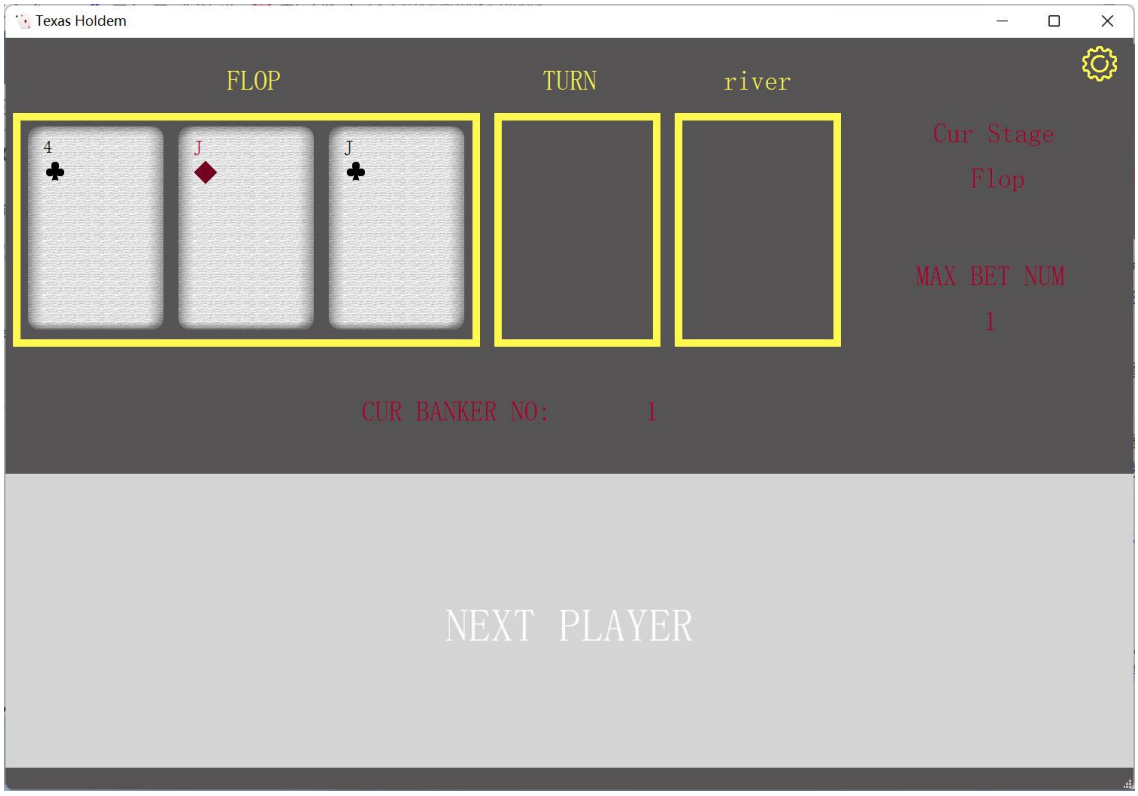
- CALL:

  当玩家选择 CALL 时，玩家会投注与上一投注玩家相等的金额。当一轮中每个非投降玩家都投注相等的金额时，游戏才能进入下一轮。

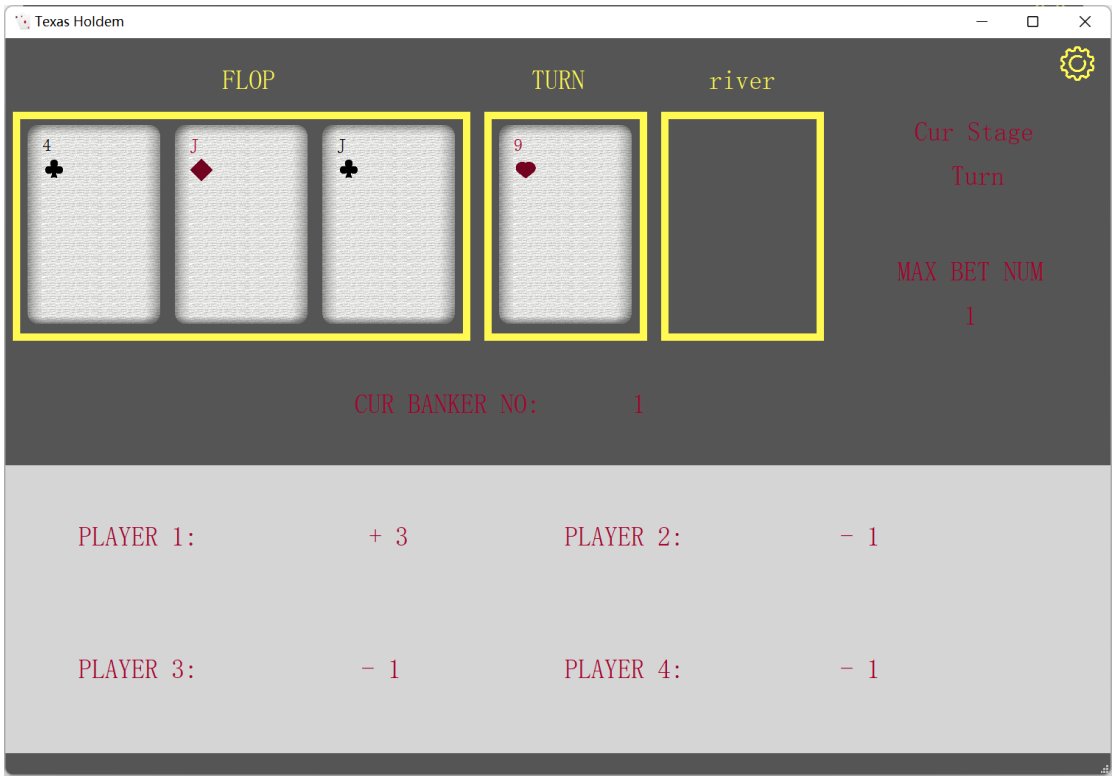
- FLOP:

  玩家投降，放弃本局投注的所有金额，退出游戏。

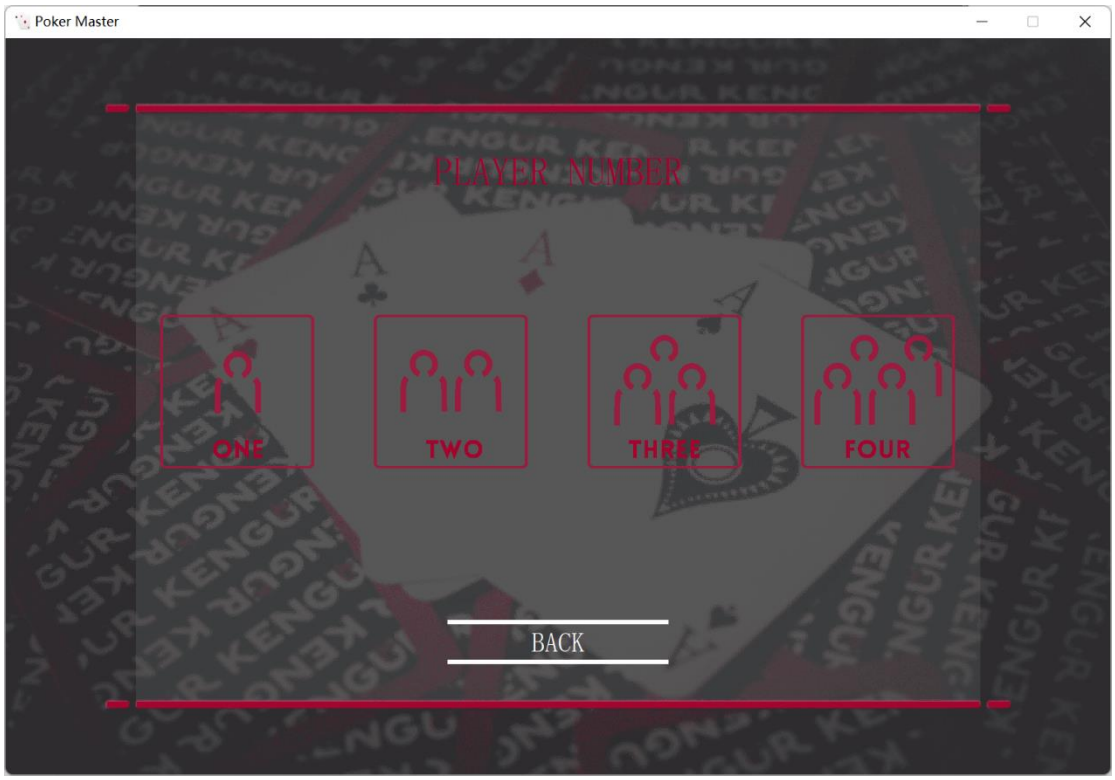
由于我们的游戏是单机游戏，当某个玩家选择操作时，游戏下方会出现 NEXT PLAYER 的提示，用于提供时间来更换玩家，防止当前玩家看到下一个玩家的牌。当然，至于玩家会不会真的更换，全靠自觉。



- 当遇到 AI 时，玩家不需要手动操作，只需等待 AI 自动操作即可。
- 当游戏结束之后，有游戏结果的窗口弹出，提示当局游戏结果。



如果你想退出游戏，点击右上角的“⚙”按钮或是直接关闭窗口即可，游戏会回到选择玩家个数的窗口。



4 系统设计难点及其解决

1. 伪多图层显示与扑克牌绘制

由于 Qt 本身不支持分图层显示，且采用多窗口切换会造成严重闪烁，严重伤害用户使用体验。本程序采用对组件进行多层组合的方式提高控件的复用率，并缩短程序的使用时间。以游戏选择模块为例，游戏种类/游戏形式/玩家数量三个界面实际上共享同一个 BACK 按键，每次通过对 MainWindow 的私有成员变量进行操作以标注当前图层，并选择对应的函数进行显示。

由于页面背景均由导入图片产生，且存在多个页面共享同一背景的情况，所有背景图均在打开程序时加载到界面上，通过控制其可见性确保与前方小控件的内容相匹配。

由于扑克牌的底纹与花色同样通过导入图片实现的，而在每一个可能的手牌位上均预先加载 52 张扑克牌显然会导致极大的开销，且 52 张不同的图片素材也会令文件体积显著增加。本程序最终决定将一个标注点数的 label 与五个用于显示底纹和花色的 label 组合成 frame 用于卡牌显示，每次通过玩家持有的手牌数量与 card 类返回的点数和花色信息完成对牌组的显示，使得在避免实时读入图片素材的同时，每个可能位置上需要预加载的图片数量减至 4 张。

2. 德州扑克设计难点

设计的主要难点在于使用 c++进行编程时该如何提升代码的封装性。在开始写代码前，还需要对类进行分析设计，设置什么变量、什么函数、返回什么值都需要精心设计，这一部分非常难一下子思考清楚，往往是后面写的时候发现问题了，再来修改前面的内容。另外，程序封装得太好也会导致写代码的难度增加。

许多时候要调用函数实现某个功能，则需要连续向下调用好几次才能够实现。而且在与前端对接的时候，由于 qt 功能限制问题，后端代码在原版的基础之上修改了许多，德扑部分的 AI 机器人也没能够实现。总体开发难度相较于 C 大程而言有所提升，但有经验所以整体开发的周期没有拉的很长，能够较快地完成进度。

3. 初次上手利用 Qt 控件搭建图形化界面

需要大量阅读 Qt 本身的文档，并广泛查阅相关资料，掌握各种组件的功能区别，此外还需注意不同控件固有属性的区别，并根据最终的实现需求进行选择。此外，由于 Qt 的串行运行逻辑不可在函数中进行打断，对原本能在命令行中正常运行的后端代码改造较为困难，需要前端实现者重新对功能函数进行拆分和拼装。最终通过模仿官方示例中的信号绑定机制，并积极与实现后端功能的同学进行沟通解决了问题。

4. 对于 21 点的 AI 设计

最开始有一个失败的方案，就是用 pytorch 训练一个模型，但由于打标签困难，最终我使用无监督学习得到的模型有一个严重问题，就是在 bet 的决策上只会选择 all in 或者 不加注。这样的 AI 显然存在问题，最终我们想到了手动设置决策边界的方法。

本次设计中，根据有关研究得出的决策边界，我们设计出了以下决策边界表：

玩家手牌之和/庄家明牌	2	3	4	5	6	7	8	9	10	A
<=11	T	T	T	T	T	T	T	F	F	F
12	T	T	F	F	F	F	F	F	F	F
13	T	F	F	F	F	F	F	F	F	F
14	T	T	T	T	F	F	F	F	F	F

15	T	F	F	F	F	F	F	F	F	F
16	T	T	F	F	F	F	F	F	F	F
17	T	T	T	T	T	F	F	F	F	F
18	T	T	T	T	T	F	F	F	F	F
19	T	T	T	T	T	T	T	F	F	F
20	F	T	T	T	T	F	F	F	F	F
21	F	T	F	F	F	T	T	F	F	F

但仅仅用上述决策边界显然是不够的。我们加入了决策边界动态化的想法，其决策边界的变化主要体现在两方面：一是考虑到全场所有牌的分值情况。根据 21 点的计算策略，当场上整体的明牌中，大牌较多的情况下，庄家具有较大的爆牌可能性。同时，在小牌较多的情况下，AI 也应当权衡考虑去抽牌来加大自身的分数。因此，全场大小牌数是第一个决策边界动态因素。二是为了使结果更加丰富，并让 AI 表现与真人更加一致，我们引入了运气评判机制。每一次抽牌都会带来当前手牌的价值变化，这种变化的方向和程度能够说明运气。我们将每次运气变化保留，并求二次导数，获得运气的变化方向，根据变化方向来更改决策边界。

5 总结

沈韵泓

开发经验:

在整个扑克牌游戏的项目开发中，我主要负责了总体架构设计、“开始”界面的设计与功能实现及 21 点游戏界面的实现。因此，如何在不同的机器上实现同一的 UI 风格成为了困惑我良久的问题。

原本计划所有 button 的弹起/悬停态均通过图片实现，然而在实际过程中发现: 由于需要的图片较多, 即便对图片进行压缩, 此举也将大幅增加文件的体积。最终决定使用同一的 png 格式图片作为按钮背景、使用系统预装字体作为文本字体，从而在大幅缩减文件体积的同时兼顾界面美观度与字体风格的统一。

不足:

受 Qt 框架及本人技术水平限制，原本计划的发牌动画（平移、翻转）最终未能实现，最终以静态展示的方式进行呈现。界面切换时的淡入淡出效果也直接被简单的 show/hide 指令取代，使得游戏界面呈现有一种 PPT 的僵硬感。未能实现这些效果导致了巨大的沉没成本, 说明了设计初期预先对相关技术实现难度及可能性进行了解的重要性。

此外，由于 Qt 在临时读取图片资源/切换窗口时会产生明显的卡顿感，本程序牺牲了载入时间, 预先将所有图片资源载入窗口并设置其可见性以向用户展示不同的页面效果。虽然取得了较为顺畅的视觉效果，但随着游戏规模的增大，进入游戏前的漫长等待必定会超出玩家的预期，导致用户流失。需要在后续迭代中寻找有效的解决方案。

王柯棣

本学期是我第一次接触 C++，也是第一次接触 QT，上来就要写这种大程确实难顶了一些。从最开始连德扑的规则是啥都不知道，到后来学 QT 学到头皮发麻，再到后来拿到命令行的代码一头雾水，整个大程还是挑战十足的，全靠硬着头皮硬生生一个字符一个字符地敲出来。不过写完了大程收获还是挺大的，对C++的理解进一步加深了，也学到了前端的各种奇奇怪怪的知识，也在 QT 的各种乱七八糟的 bug 面前好好磨练了一把性子。当然，我自己的速成能力还是要再进一步加强，对于各种开发工具的应用也有待进一步的学习。当然最后还是一定要感谢无敌的队友们直接把我带飞，否则大程怕是要烂了。

魏鼎坤

在本次实验中，学习和实践并行的开发过程让对整个学期 OOP 知识进行了复习，说来惭愧，我平时上课没有记笔记的习惯，但是为了开发项目复习 OOP 知识的时候，却记了接近一万字的笔记。同时，很多看起来很简单的知识在实际应用中也存在很多细节和注意的地方，在不断犯错和纠正的过程中，我也无形中完成了对本学期课程知识的复习和巩固。当初我们小组选择设计游戏的时候，考虑了大家对游戏的兴趣可能会更加浓厚，而当实际上手之后，却发现设计游戏和开发游戏与自己之前想的很简单的玩游戏差别很大，很多地方需要很严谨的逻辑，游戏里很简单的一些交互可能背后有很复杂的设计。总体上我们的心情经历了从兴奋到沮丧，再到坦然，最后又充满激情的状态。最后成功的时候，还是充满成就感的。

赵伊蕾

在本次实验中，我主要负责了德州扑克游戏中类的设计与游戏逻辑的实现，设计的主要难点在于如何提升代码的封装性。实际动工前，需要预先对过程中可能需要的类进行分析，设置什么变量、什么函数、返回什么值都需要精心设计，这一部分很难一下子思考清楚、考虑全面，往往是在编写时出现问题后，再对成员变量及成员函数的返回值类型、参数列表等进行修改。另外，程序封装得太好也会导致实现难度的增加。许多时候要调用函数实现某个功能，则需要连续向下调用好几次才能够实现。而且在与前端对接的时候，由于 qt 功能限制问题，后端代码在原版的基础之上修改了许多，德州扑克部分的 AI 机器人也没能够最终实现。总体开发难度相较于 C 大程而言有所提升，但由于有先前的积淀的开发经验，整体开发周期并没有拉的很长，最终成功以较快的进度顺利完成了开发工作。

6 系统使用说明

- 准备工作

1. 进入程序

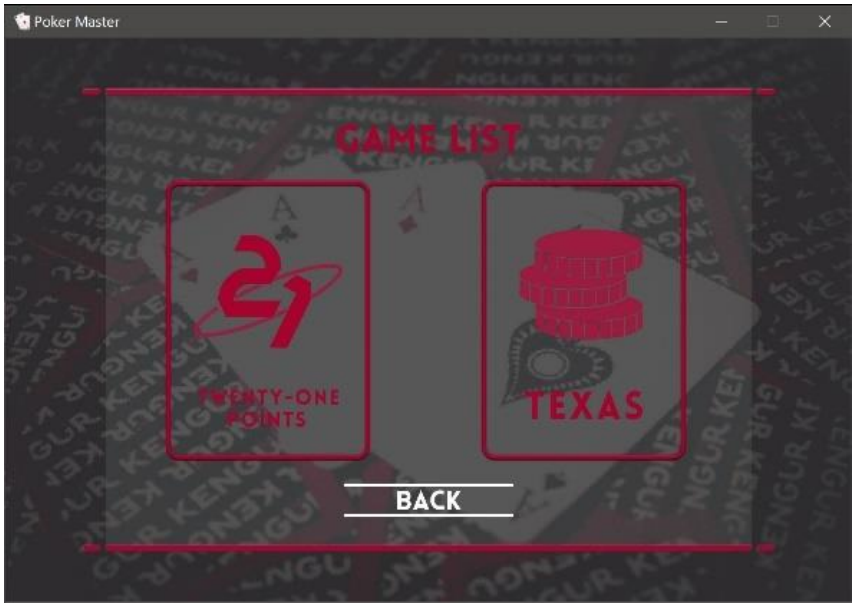
运行 exe 文件后，您将看到 PokerMaster 的标题页，请点击 START 按钮以进行后续选择。

接下来，您将看到一下界面。在文本框中输入用户名并点击 GO 按钮即可进行游戏相关设置。

2. 进行游戏相关设置

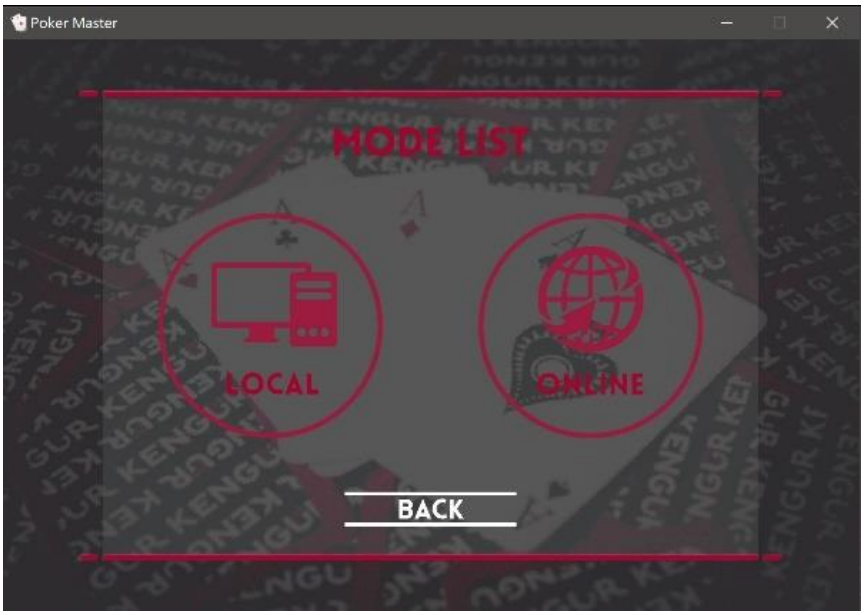
2.1 游戏模式的选择

在本页面中，您可以点击左侧按钮选择 21 点游戏模式，或点击右侧按钮选择德州扑克模式。点击 BACK 按钮将返回自定义用户名页面。



2.2 游玩模式的选择

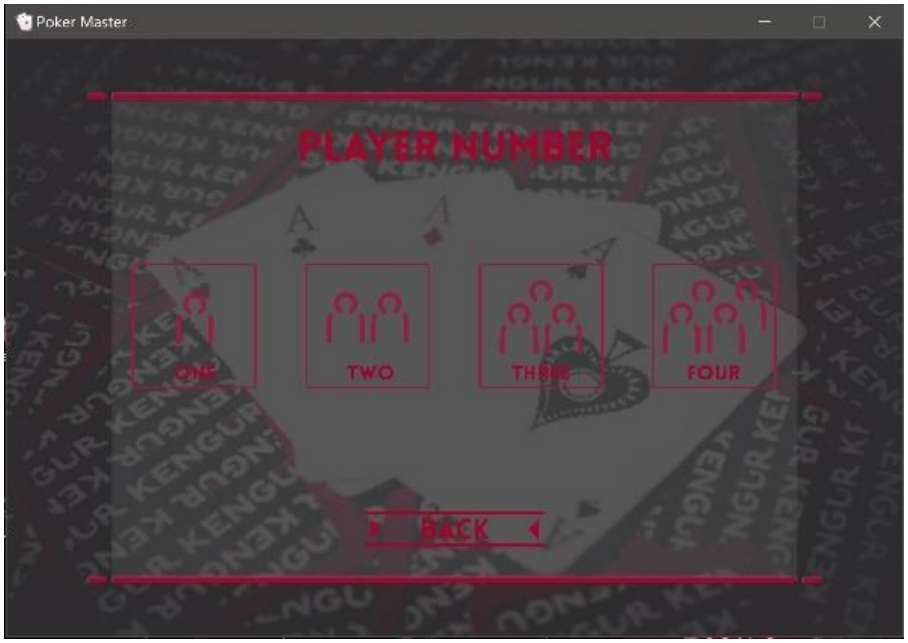
在本页面中，您可以点击左侧按钮选择本地游玩模式，后点击右侧按钮选择在线匹配模式（暂不支持）。点击 BACK 按钮将返回游戏模式选择页面，



若您看到以下提示，可以点击 BACK 按钮返回游玩模式选择界面。请在检查自身网络连接情况后重新做出选择。

2.3 玩家人数的选择

在本页面中，请根据真人玩家数量做出相应选择。少于四人进行游戏时，剩余人数将由 AI 进行补全。点击 BACK 按钮将返回游玩模式选择页面。



- 21 点游戏模式

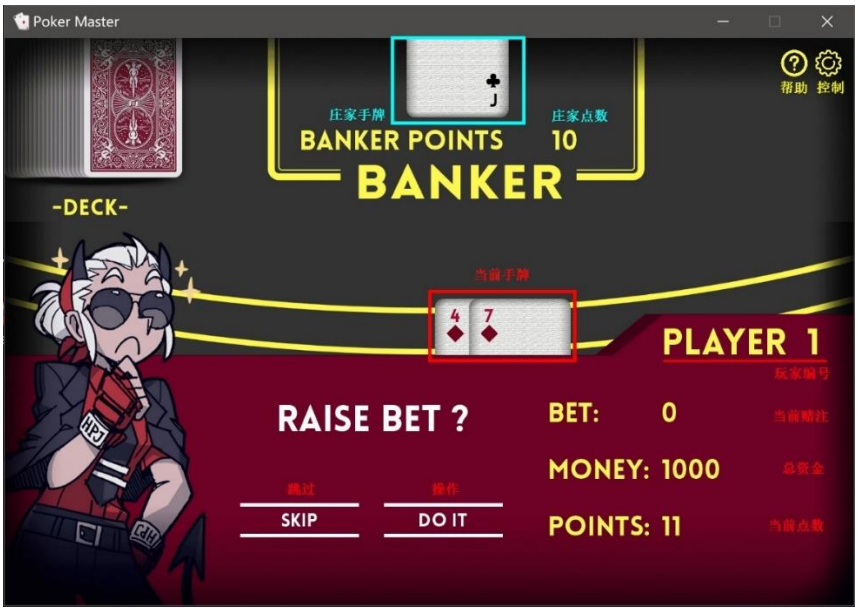
1. 预操作阶段（自动）

本阶段您将看到以下画面，在此过程中，庄家将先给所有玩家发一张手牌，为自己发一张明牌，再为所有玩家发一张手牌。

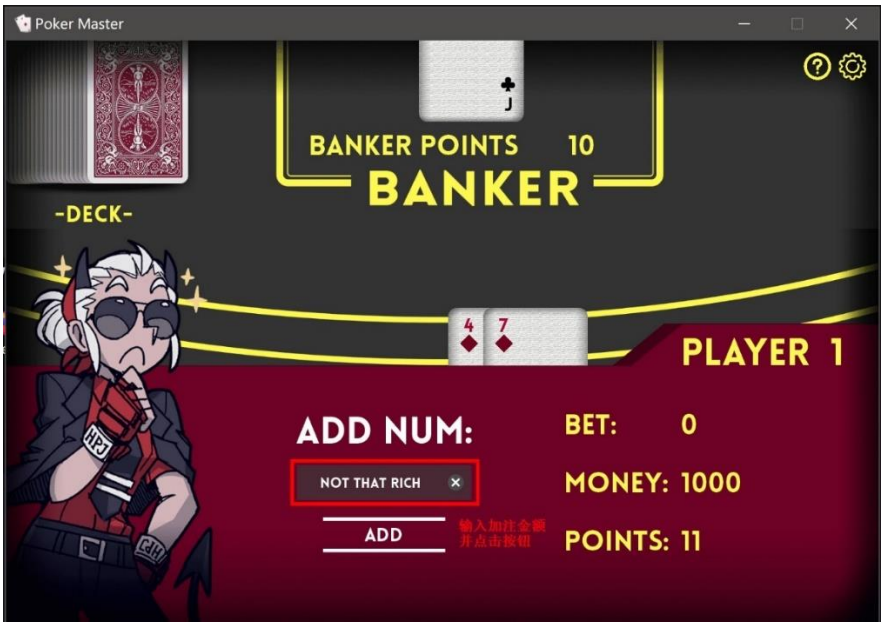
2. 玩家操作阶段

2.1 加注

玩家可点击 DO IT 进入加注页面，或点击 SKIP 跳过本阶段。

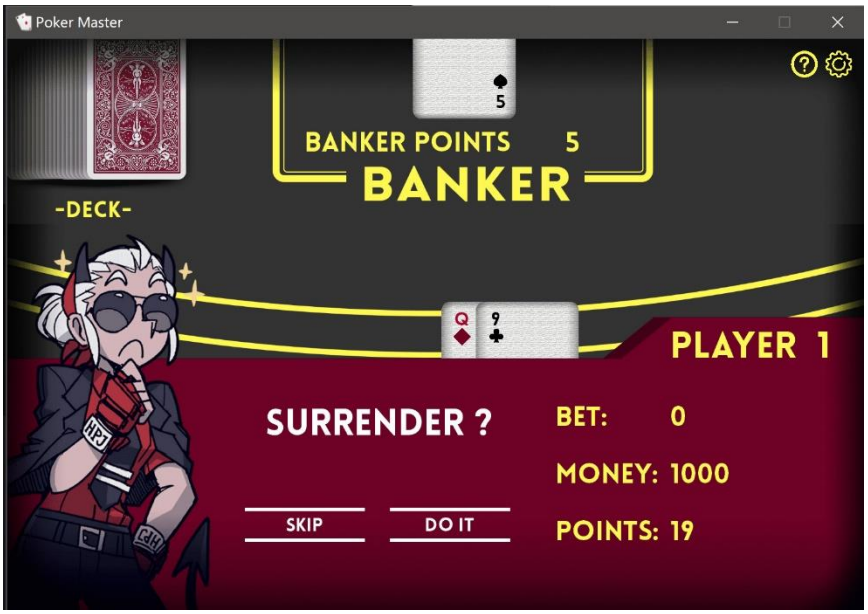


玩家可在文本框中输入小于自己当前资金总量的数额并点击 ADD 按钮完成加注，BET 与 MONET 将实时进行更新。



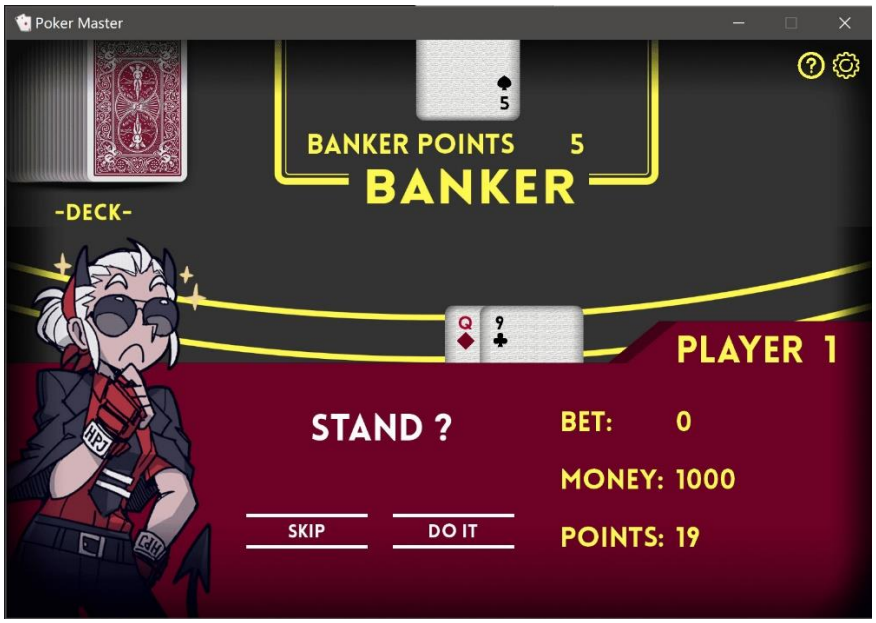
2.2 投降

玩家可以点击 DO IT 完成投降操作（此后将不能进行人工操作），或点击 SKIP 跳过该阶段。

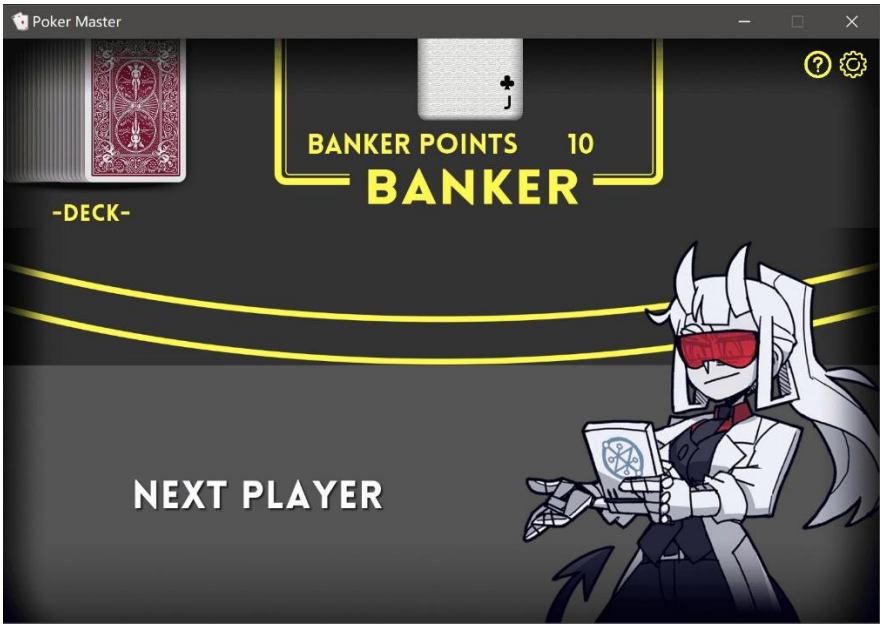


2.3 停牌/摸牌

玩家可以点击 DO IT 完成停牌操作（此后将不能进行人工操作），或点击 SKIP 进行摸牌操作（一轮仅限一张）。

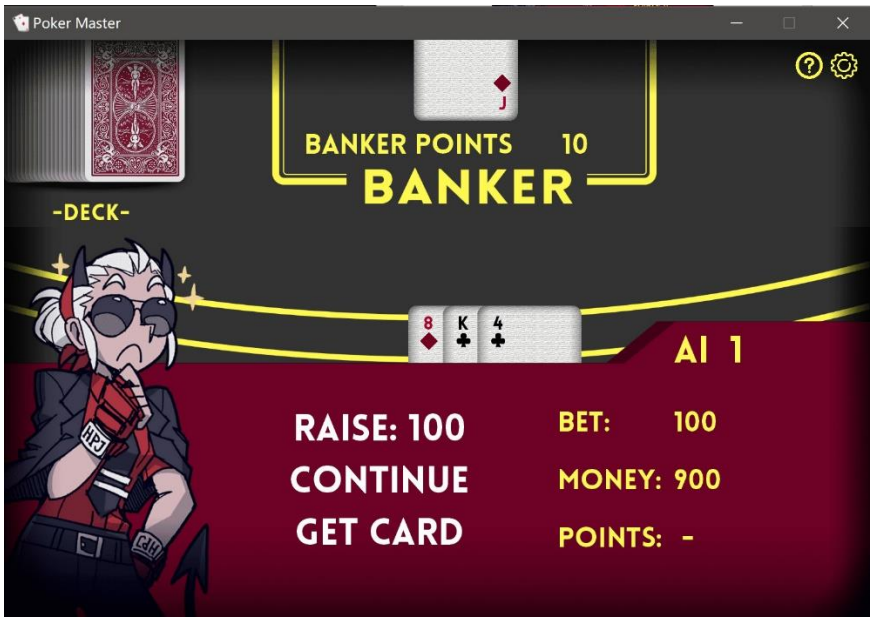


此后将进行玩家更换，您将看到以下界面：



3. AI 操作阶段（自动）

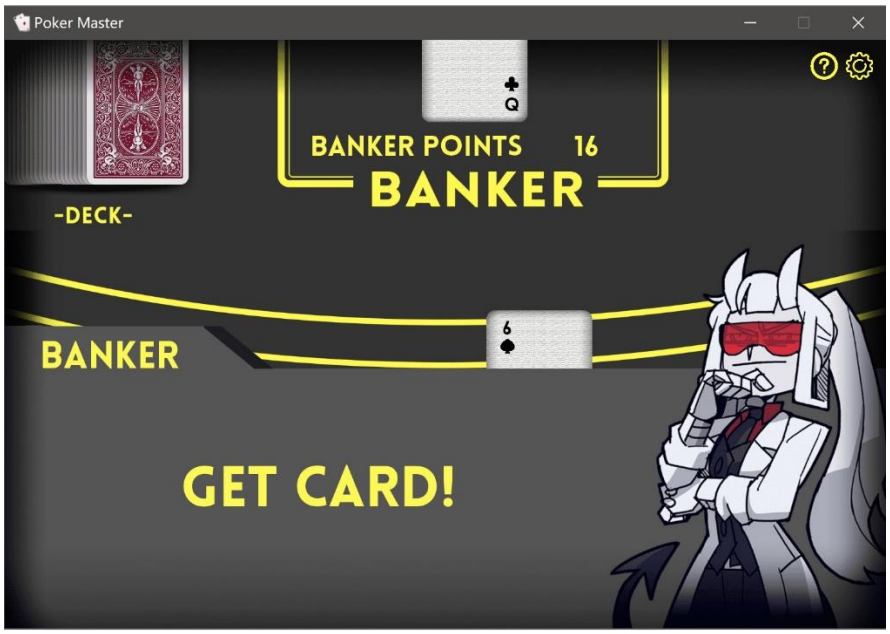
   AI 操作与人工操作流程一致，界面显示如下：



   您将在 2s 后自动跳转至下一界面。

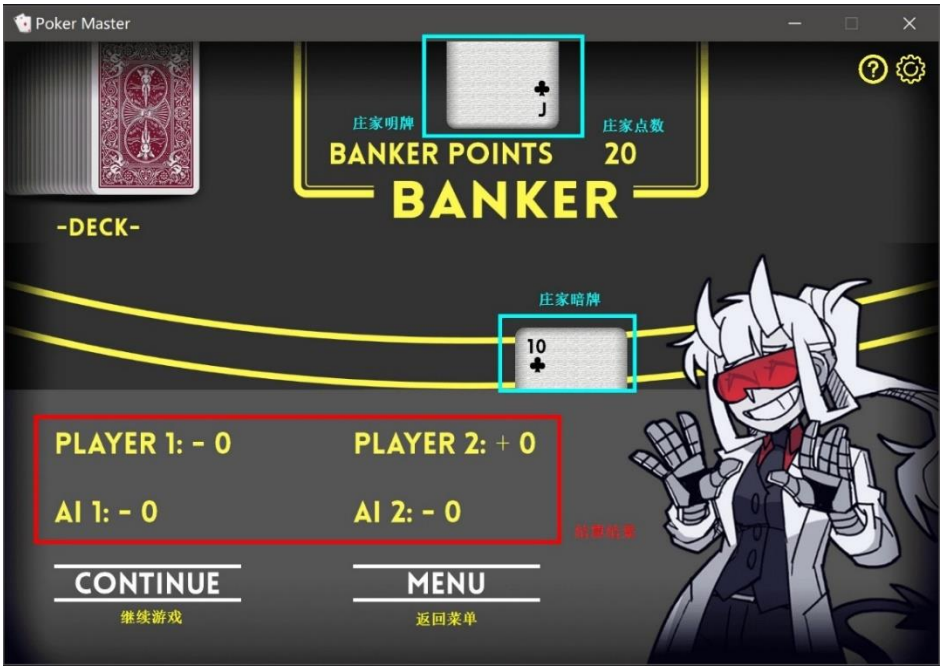
4. 庄家操作阶段（自动）

   庄家在超过 17 点钱将持续摸牌，您将看到以下界面：



5. 结算（自动）

您可以点击右上角的 help 按钮查看具体的结算规则，点击 setting 暂停/继续游戏，或回到主界面。结算完毕后，您可以点击 CONTINUE 继续进行下一轮游戏，您的资金将会被继承，或点击 MENU 返回游戏选择页面。



- 德州扑克游戏模式

7 系统开发日志

2022/04/28 - 2022/05/01

- 对各选题的相关资料进行收集、整理
- 对可用开发框架信息进行收集、整理
- 进行第一次小组会议，确定选题（扑克牌游戏）与开发框架（Qt）

2022/05/02 - 2022/05/08

- 制定编码规范
- 学习 Qt 编程技术
- 熟悉两种扑克牌游戏的规则
- 绘制程序整体流程图

2022/05/30 - 2022/06/05

- 完成 21 点与德州扑克两个模式的前后端联调
- 完成游戏与准备界面的连接

2022/06/06 - 2022/06/12

- 完成开发文档与个人总结撰写

7.1 前端进度

2022/05/02 - 2022/05/08

- 完成 UI 设计与交互逻辑设计
- 完成类及函数设计
- 收集图片素材

2022/05/09 - 2022/05/15

- 通过 PhotoShop 完成对图片素材的处理
- 实现标题页

  解决了单页面多图层实现。
- 实现游戏选择页

  解决了对按钮 hover / 弹起状态切换的问题及样式重设问题。

2022/05/16 - 2022/05/22

- 完成 21 点游戏页面

  解决扑克牌绘制问题。

  解决 sleep 停顿时间与标注时间不一致的问题。

  解决 timer 并发运行问题目。
- 完成德州扑克游戏页面

2022/05/23 - 2022/05/29

- 完成 21 点游戏 AI 接入

7.2 后端进度

2022/05/02 - 2022/05/08

- 完成对类之间继承关系的设计
- 完成对各类成员函数的设计

2022/05/09 - 2022/05/15

- 定义供前端调用的接口
- 完成各类中较为独立的函数

2022/05/16 - 2022/05/22

- 实现类中的其余函数
- 规范现有代码的格式，并进行注释

2022/05/23 - 2022/05/29

- 完成 21 点游戏 AI 后端实现

附录

中期会议记录

会议时间：2022 年 5 月 10 日

会议地点：碧峰二幢

出席人员：沈韵泓、王柯棣、魏鼎坤、赵伊蕾

会议内容：

1. 讨论确定选题

经过讨论和分析，我们认为扑克牌游戏在实现上具有一定挑战性，可以加强我们的代码能力，并且其功能的实现具有较强逻辑，更加适合我们。以扑克牌游戏为目标，我们可以丰富自己的专业知识，学习更丰富的本领和技能。

- 选题内容：扑克牌游戏
- 选题要求：
  - ✧ 友好的图形用户界面
  - ✧ 支持至少 MAX(2，组员人数-2) 种扑克牌游戏
  - ✧ 支持四玩家参与游戏（单机参与）
  - ✧ 支持少于四玩家参与游戏，不足人数由计算机扮演
  - ✧ 支持不同玩家局域网内联机参与游戏

2. 讨论确定任务分工

经过各位组员的自我展示和相关特长分析，讨论决定本次将实现任务分为两个部分，一个负责二十一点游戏，一个负责德州扑克游戏，游戏开始界面等公共界面共同开发。具体人员分工见上。

3. 讨论确定了实现细节

- 经过讨论分析，决定使用 VS2017 作为 IDE，并使用 QT 框架辅助开发。
- 讨论确定了页面风格，确定了前端样式。
- 讨论确定了高级功能的实现
- 讨论了类的设计和架构