“熊猫斗地主”说明文档

Source Code: https://github.com/Li-Jia-Jun/MyGames/tree/master/PandaLordCard
Demonstration Video: https://www.youtube.com/watch?v=skJvBKR3DMc

● 规则说明

此款游戏的玩法基于传统三人斗地主的经典模式。

流程：选地主--->地主首出牌--->三人轮出牌--->某玩家出完所有牌--->结束。

牌型：单张、对子、三带一、三带对、四带二、炸弹、王炸（又称火箭）单顺、双顺、三顺、三带单、三三带单单等将近四十种组合方式。单张牌的牌型由小到大为 3、4、……、10、J、Q、K、A、2、王，花色不影响牌型，红王大与黑王。三带一的权值由三张决定，三三带一一由大的三张决定，其余情况类似。

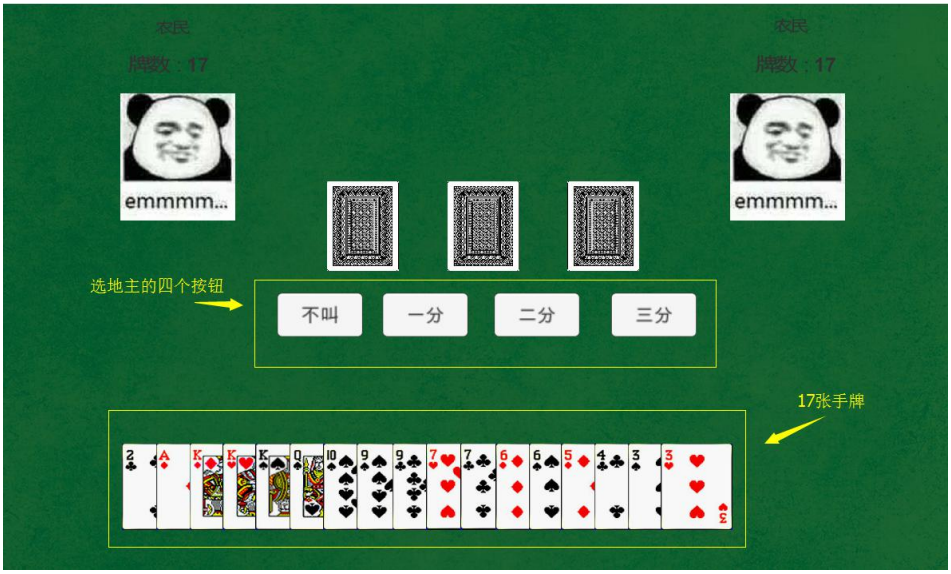
出牌：自由出牌时（上一轮没人管上自己）时只要出牌符合某一牌型即可；管上出牌时要求和上一轮出的牌同牌型且权值更大。炸弹（四张）和王炸（双王） 特殊，王炸大于所有牌型，四张大于除王炸外其他所有牌型，但会被权值更大的炸弹管上。

更多规则细节请参考：
https://baike.baidu.com/item/%E4%B8%89%E4%BA%BA%E6%96%97%E5%9C%B0%E4%B8%BB/9429860?fr=aladdin

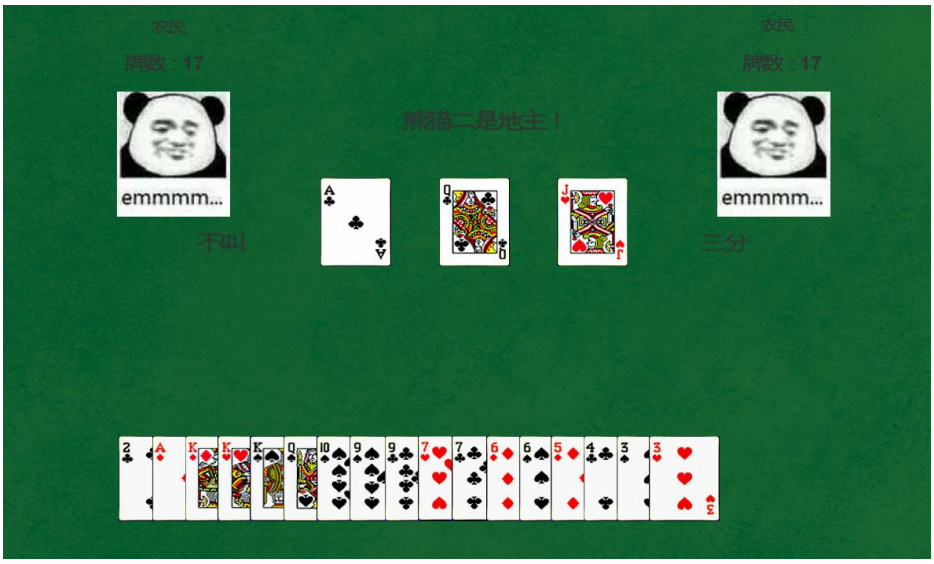
● 游戏元素

一、开始界面
开门见山，只有开始按钮（开始表演），点击则开始游戏

二、游戏进行界面
选地主 UI：①四个按钮，与选地主规则相对应。
②此轮玩家的 17 张手牌（地主得到额外三张），玩家将根据此来权衡是否要争地主。



（与斗地主规则相符，地主的额外三张牌会公布）



游戏界面：①轮到玩家出牌时激活 17 张手牌的 UI，使手牌可点击。

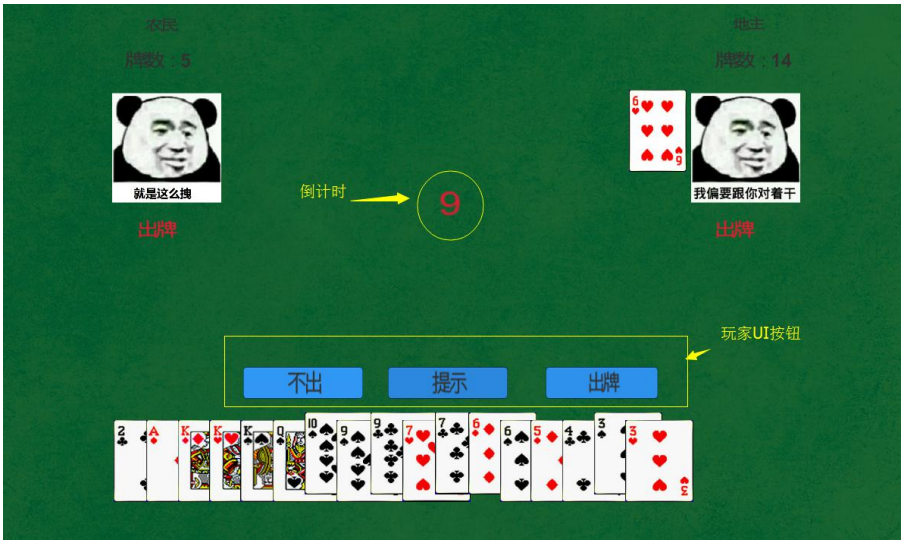
②三个按钮。不出：手牌复位，轮到下一个参与者出牌（顺时针）；提示：上出牌时手牌中有可以管上的牌时则可点击，否则不可点击。自由出牌时不可点击。点击后会根据贪心算法选出当前可以管上的最小牌型（牌弹起）。出牌：玩家选出的牌型符合规则则可点击，否则不可点击。

③倒计时。玩家在规定时间内不做出选中则默认不出。

④AI：手牌数，身份（农民或地主）、在不同情形下的表情，倒计时，消息

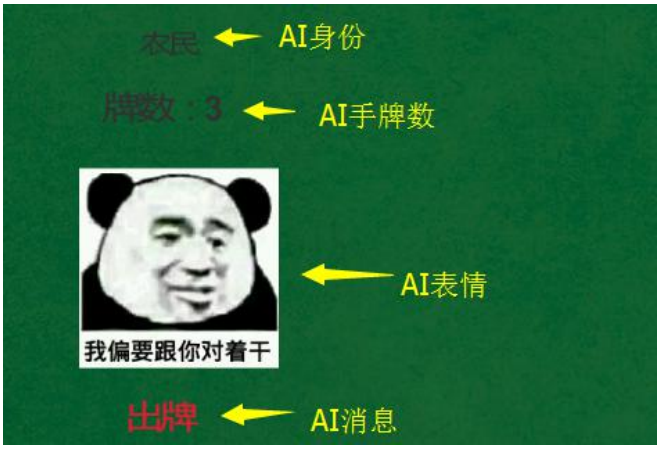
⑤一轮游戏结束后弹出重新开始（重新表演）和退出游戏（放弃表演）按钮，并显示结果（胜利或失败）

（玩家 UI 和倒计时）



（要不起时的图片提示）

（AI 元素）



（胜利）

（失败）

● 游戏实现

一、用到的类或结构

PokerNameSpace 中：Poker：扑克牌类(struct)，存放单张扑克牌的牌号、权值、花色等信息。

PokerGroup：牌组类(struct)，存放扑克牌组的信息。

PKType：扑克牌牌型枚举类(enum)，枚举出所有可行的牌组合形式。

Pos：二维坐标类(struct)。

PokerManager：扑克牌管理类(class)，主要用途是将一组牌归类为 PokerGroup。

RoleNameSpace 中：Role：参与者基类(class)，含有 AI 和玩家共用的函数。

PlayerRole：玩家类(class)，继承于 Role，额外包含有 UI 的相关内容。

AIRole：AI 类(class)，继承与 Role，额外包含了 AI 出牌决策的相关内容。

Unity3D 环境中：GameController：游戏管理类(class)，用于对发牌、轮转、开始结束等游戏流程的管控。

HandPokerController：手牌控制类(class)，挂载在玩家手牌上的脚本类，实现点击回调和牌的上下移动。

PedalController：选地主 UI 控制类(class)，挂载了选地主 UI 的按钮和三张牌。

二、具体功能实现：

此款游戏的实现难点主要有两个：第一，将玩家选择的牌归纳成近四十种牌型的一种。第二，从手牌中找出所有可能的牌型，并根据情形选择可行的应付策略。一般的问题则包括游戏轮转，AI 出牌策略等。下面是我的应对策略

①将牌归类：

首先，PokerController 中 getPKXXXX(Poker[])系列函数将传入的牌数组归纳出指定的一类，归纳失败则标记为 PK_NAT（还不是类）。将需要数组长度相同的委托放入委托数组中，方便统一调用。

（getPKXXX()系列函数）

```
PokerGroup getDAN(Poker[] p)  //获取 单张
PokerGroup getDUI(Poker[] p)  //获取 对子、王炸
PokerGroup getSAN(Poker[] p)  //获取 三张
PokerGroup getBOMB(Poker[] p)  //获取 炸弹

PokerGroup getSHUN(Poker[] p)  //获取 单顺（5张~12张）
PokerGroup getSHUN2(Poker[] p)  //获取 双顺（6~20张）
PokerGroup getSHUN3(Poker[] p)  //获取 三顺（9~18张）

PokerGroup getPK_3_DAI_DAN(Poker[] p)  //获取 三带单、三三带单单、三三三带单单单.....
PokerGroup getPK_3_DAI_DUI(Poker[] p)  //获取 三带对、三三带对对、三三三带对对对.....
PokerGroup getPK_4_2(Poker[] p)  //获取 四带对、四带二单
PokerGroup getPK_4_DUI_DUI(Poker[] p)  //获取 四带对

PokerGroup getPK_NAT(Poker[] p)  //获取 非类
```

（委托数组声明）

```
delegate PokerGroup getPKType(Poker[] p);
getPKType[] getPKTypeArray;
```

（在构造函数中初始化委托数组）

```
public PokerManager()
{
    getPKTypeArray = new getPKType[20];
    //getPKTypeArray[0]代表牌数为一张的getPKType，以此类推
    getPKTypeArray[0] = new getPKType(getDAN);
    getPKTypeArray[1]  = new getPKType(getDUI);
    getPKTypeArray[2] = new getPKType(getSAN);
    getPKTypeArray[3]  = new getPKType(getBOMB);
    getPKTypeArray[3] += getPK_3_DAI_DAN;
    getPKTypeArray[4]  = new getPKType(getSHUN);
    getPKTypeArray[4] += getPK_3_DAI_DUI;
    getPKTypeArray[5]  = new getPKType(getSHUN);
    getPKTypeArray[5] += getSHUN2;
    getPKTypeArray[5] += getPK_4_2;
    getPKTypeArray[6]  = new getPKType(getSHUN);
    getPKTypeArray[7]  = new getPKType(getSHUN);
    getPKTypeArray[7] += getSHUN2;
    getPKTypeArray[7] += getPK_3_DAI_DAN;
    getPKTypeArray[7] += getPK_4_DUI_DUI;
    getPKTypeArray[8]  = new getPKType(getSHUN);
    getPKTypeArray[8] += getSHUN3;
    getPKTypeArray[9]  = new getPKType(getSHUN);
    getPKTypeArray[9] += getSHUN2;
    getPKTypeArray[9] += getPK_3_DAI_DUI;
    getPKTypeArray[10]  = new getPKType(getSHUN);
    getPKTypeArray[11]  = new getPKType(getSHUN);
    getPKTypeArray[11] += getSHUN2;
    getPKTypeArray[11] += getSHUN3;
    getPKTypeArray[11] += getPK_3_DAI_DAN;
    getPKTypeArray[12]  = new getPKType(getPK_NAT);
    getPKTypeArray[13]  = new getPKType(getSHUN2);
    getPKTypeArray[14] = new getPKType(getSHUN3);
    getPKTypeArray[14] += getPK_3_DAI_DUI;
    getPKTypeArray[15] = new getPKType(getSHUN2);
    getPKTypeArray[15] += getPK_3_DAI_DAN;
    getPKTypeArray[16]  = new getPKType(getPK_NAT);
    getPKTypeArray[17]  = new getPKType(getSHUN2);
    getPKTypeArray[17] += getSHUN3;
    getPKTypeArray[18]  = new getPKType(getPK_NAT);
    getPKTypeArray[19] = new getPKType(getSHUN2);
    getPKTypeArray[19] += getPK_3_DAI_DAN;
    getPKTypeArray[19] += getPK_3_DAI_DUI;
}
```

PokerManager 中的公有函数 getPokerType(Poker[])是统一入口，函数根据数组的长度调用委托数组中的相应委托。这里委托数组中的某个委托是指是一个委托列表，如 getPKTypeArray[3]对应的是四张牌的委托列表，其中包括 getBomb()和 getPK_3_DAI_DAN()。

（getPokerType()函数）

```
//把p归纳成某种类型的牌型并将其返回
public PokerGroup getPokerType(Poker[] pokers)
{
    int length = pokers.Length;
    if (length <= 0 || length >= 20)
        return new PokerGroup(PKType.PK_NAT, PokerGroup.NAT_VALUE, null);

    foreach (getPKType d in getPKTypeArray[length-1].GetInvocationList()) {
        PokerGroup pg = d(pokers);
        if (pg.type != PKType.PK_NAT)
            return pg;
    }
    return new PokerGroup(PKType.PK_NAT, PokerGroup.NAT_VALUE, null);
}
}
```

每个 getPKXXXX()都有各自的算法，下面以 getPK_3_DAI_DAN()为例进行说明。
该函数用于归纳三带单形式的所有牌型，即三带单、三三带单单、三三三带单单单、三三三三带单单单单等。首先，将传入的 Poker 数组对应到 Dictionary<int,int>，该字典用来记录单张牌号和其数量的键值对。临时变量 num3 和 num1 记录了达到三张和一张的牌号的数量,需要注意的是 num3 加 1 时 num1 要相应减 1。markValue 标记了该牌组作为三带一系列的权值，其值取决于最大三张的牌号的权值。函数最后根据 num3 和 num1 的值将牌归类，如果不属于某一类则创建非类 PokerGroup 返回(PK_NAT)。

（getPK_3_DAI_DAN()函数）

```
PokerGroup getPK_3_DAI_DAN(Poker[] p)
{
    int length = p.Length;
    //dictionary保存各出值与其数量的键值对，markValue标记了数量为3的牌的最大权值
    //num1和num3分别表示数量为1和数量为3的牌的个数
    Dictionary<int, int> d = new Dictionary<int, int>(); int markValue = -1;
    int num3 = 0, num1 = 0;

    foreach (Poker poker in p) {
        if (d.ContainsKey(poker.Id)) {
            if(++d[poker.Id] == 3) {
                num3++;
                num1--;
                if (markValue < poker.Value)
                    markValue = poker.Value;
            }
        }
        else {
            num1++;
            d.Add(poker.Id, 1);
        }
    }

    if (num3 == 1 && num1 == 1 && length == 4) {
        return new PokerGroup(PKType.PK_3_DAN, markValue,p);
    }
    else if (num3 == 2 && num1 == 2 && length == 8) {
        return new PokerGroup(PKType.PK_33_DAN_DAN, markValue,p);
    }
    else if (num3 == 3 && num1 == 3 && length == 12) {
        return new PokerGroup(PKType.PK_333_DAN_DAN_DAN,markValue,p);
    }
    else if (num3 == 4 && num1 == 4 && length == 16) {
        return new PokerGroup(PKType.PK_3333_DAN_DAN_DAN_DAN, markValue,p);
    }
    else if (num3 == 5 && num1 == 5 && length == 20) {//能拿到这种牌你也是牛啊
        return new PokerGroup(PKType.PK_33333_DAN_DAN_DAN_DAN_DAN,markValue,p);
    }
    else {
        return new PokerGroup(PKType.PK_NAT, PokerGroup.NAT_VALUE,null);
    }
```

其他一些函数如 getPK_4_DAI_2()、getPK_4_DAI_DUI()、getPK_3_DAI_DUI()均采用上述思想，另一些函数如 getSHUN()、getSHUN2()、getSHUN3()采用的是另一类思想，具体体现在源码注释中，这里不再赘述。

②从手牌中找出某一种管上牌型：

为方便寻找牌型，这里将 Role 类中的手牌存入 Dictionary<int,List<Poker>>容器，其中 int 表示 14 种牌的牌值，即 1~14（14 表示小王大王）。

另一个辅助容器时 List<Pos> hints,它记录了找到一种牌型后组成这种牌型每张牌在 Dictionary<int,List<Poker>>中的位置。

（手牌容器和策略容器）

```
protected Dictionary<int, List<Poker>> pokers;          //手牌（int 表示List<Poker>的牌值，14表示王）
protected List<Pos> hints;                              //出牌策略（记录可以出牌的位置）
```

一系列 findRuleXXXX()函数用来从手牌中寻找一种管上的牌型，如果能找到，就将 hints 清空并装入此次的寻找结果，返回 true，找不到则返回 false，不操作 hints。这类函数既用于 PlayerRole 中提示按钮的功能实现，也提供了 AIRole 出牌策略的基础支持，因此定义在它们共有的基类 Role 中，并用 protected 修饰。

（一系列 findRuleXXXX()函数）

```
protected bool findRuleDAN()...//找出 单张
protected bool findRuleDUI()...//找出 对子
protected bool findRuleWANG_ZHA()...//找出 王炸
protected bool findRuleSAN()...//找出 三张
protected bool findRuleBOMB()...//找出 炸弹
protected bool findRule3_DAN()...//找出 3带1
protected bool findRule3_DUI()...//找出 3带对
protected bool findRule4_DAN_DAN()...//找到 4带2单
protected bool findRule4_DUI()...//找到 4带对
protected bool findRuleSHUN()...//找到单顺
protected bool findRuleSHUN2()...//找到双顺
protected bool findRuleSHUN3()...//找到三顺
```

下面以 findRule3_DAN()为例做介绍。

这个函数用于从手牌中找出三带一。有了前面的容器基础，找到三张和单张的操作就简化为在 pokers 中找到 pokers[i].Count 大于 3 或 1 的位置。变量 pos3 和 pos1 记录三张和一张的位置，这里采用了简单的贪心策略，即从满足条件的情况中找到最小的三张和最小的单张。最后结果用 hints 记录。

与之前介绍的 getPK_3_DAI_DAN()不同，这个函数仅能找到三带一。如果需要找三三带单单，三三三带单单单则需要另写函数，略显冗余。这也是当初设计的一个缺陷。但是由于三三带单单、三三三带单单单这类牌

型出现的几率较少，也为了保持代码的简洁性，在没有想到更好的算法之前，这里暂时不考虑这些推广的情况。也就是说玩家出牌能支持三三带单单的推广情况，但提示按钮不会帮玩家找出这类情况。

（findRule3_DAN()函数）

```
protected bool findRule3_DAN()
{
    int value = gameController.lastGroup.value;
    if (gameController.whoisrule == gameController.whoseturn) value = -2;
    int pos3 = -1; int pos1 = -1;//记录三张和一张的位置

    for (int i = 1; i <= 13; i++) {

        if (pokers[i].Count >= 3) {
            //优先选取最小的作为三张
            if ((pos3 == -1 && pokers[i][0].Value > value) || (pos3 != -1 && pokers[pos3][0].Value > pokers[i][0].Value && pokers[i][0].Value > value)) {
                pos3 = i;
                continue;
            }
        }

        if (pokers[i].Count >= 1) {
            //优先选取最小的作为单张
            if ((pos1 == -1) || (pos1 != -1 && pokers[pos1][0].Value > pokers[i][0].Value))
                pos1 = i;
        }
    }

    if (pos3 != -1 && pos1 != -1) {
        hints.Clear();
        hints.Add(new Pos(pos3, 0)); hints.Add(new Pos(pos3, 1));
        hints.Add(new Pos(pos3, 2)); hints.Add(new Pos(pos1, 0));
        return true;
    }
    else {
        return false;
    }
```

③游戏轮转

在 GameController 中保存有三个参与者的数组 Role[] roles，这里规定 roles[0]为玩家，roles[1]、roles[2]分别为 AI_1 和 AI_2。用成员变量 whoisrule 标记数组中上一次出牌(即等待被管)的 Role 的下标，whoseturn 标记正在出牌，boor1 和 boor2 标记农民 1、农民 2，lord 标记地主。lastGroup 保存上一名玩家的出牌，即 whoisrule 的出牌。

选完地主后，根据结果先给以上介绍的变量赋相应的值，然后在函数 Run()中调用地主的出牌函数，即 Role 的纯虚函数 discard()（该函数是参与者出牌的入口函数，由 AIRole 和 PlayerRole 分别实现)。如果下一个参与者出牌，则调用 GameController 的 ruleCall()进行轮转，即更新 whoisrule 和 whoseturn 和调用下下一个参与者的 discard()。如果不出牌则调用 dontRuleCall()进行轮转，即保持 whoisrule 不变，更新 whoseturn 和调用下下一个参与者的 discard()。

（GameController 中用于的轮转的成员变量）

```
public Role[] roles;
public int lord,boor1, boor2;          //地主、农民1、农民2
public int whoseturn;                  //轮到谁
public int whoisrule;                  //谁的牌没人管
public PokerGroup lastGroup;           //上一轮打出的牌
```

（run()函数，它为轮转的触发点）

```
private void run(object[] objs)
{
    //地主拿到额外三张牌
    lord = (int)objs[0];
    whoisrule = whoseturn = lord;
    boor1 = (lord + 1) % 3;
    boor2 = (lord + 2) % 3;
    roles[lord].isLandLord = true;
    roles[lord].getExtra3((int)objs[1],(int)objs[2],(int)objs[3]);
    pedal.SetActive(false);
    //地主出牌
    roles[whoseturn].SendMessage("discard");
}
```

（ruleCall()函数，由参与者选择出牌后调用）

```
private void ruleCall()
{
    whoisrule = whoseturn;

    if (roles[whoisrule].cardNum == 0) {
        endGame();
        return;
    }

    //下一个玩家开始决策
    whoseturn = (whoseturn + 1) % 3;
    roles[whoseturn].SendMessage("discard");
}
```

（dontRuleCall()函数，由参与者选择不出牌后调用）

```
private void dontRuleCall()
{
    whoseturn = (whoseturn + 1) % 3;
    //下一个玩家开始决策
    roles[whoseturn].SendMessage("discard");
}
```

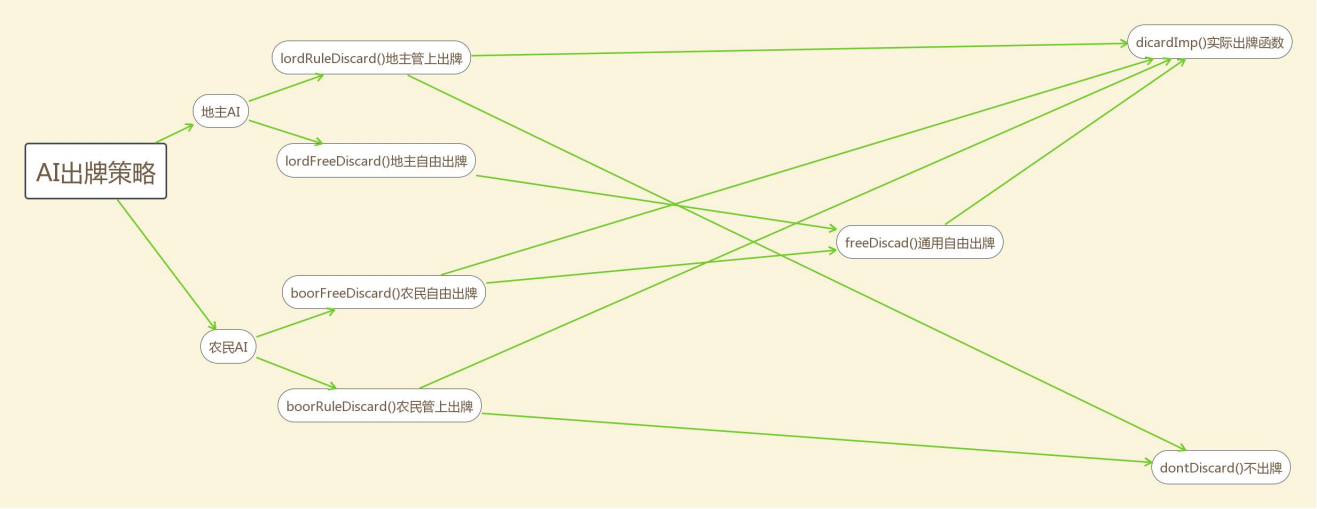
④AI 出牌策略：

AI 出牌分为自由出牌和管上出牌两者情况，而地主和农民出牌策略也不尽相同，在 AIRole 中使用四个函数来实现具体的四种出牌策略情况。

(AI 出牌的相关函数函数)

```
private void freeDiscard()...
// ...
private void boorFreeDiscard()...
// ...
private void lordFreeDiscard()...
// ...
private void boorRuleDiscard()...
// ...
private void lordRuleDiscard()...
// ...
private void dontDiscard()...
// ...
private void markCards()...
// ...
private void discardImp()...
```

（出牌函数的调用关系）



markCards()为记牌函数，在 AIRole 的 discard()函数中首先调用，具体内容是记录打出的 7、10（判断外面是否还有顺子）、A、2、王（判断外面剩余的大牌数量）。

discardImp()是执行实际的出牌操作，具体内容为根据 hints 容器中指向的牌显示到出牌处、删除打出的牌，清理容器，调用 GameController 的 ruleCall()来轮转。

freeDiscard()是通用的自由出牌函数，可供地主 AI 和玩家 AI 根据情况调用。策略是优先考虑了组合牌（组合牌牌值不是很大的情况下），然后再考虑对子和单张。其实际出牌仍需调用 discardImp()。

农民 AI：

boorFreeDiscard()为农民 AI 自由出牌函数，策略是：如果下家是手牌较少的农民，则考虑出单张或对子给他过小牌（仍调用 discardImp()）。否则直接调用 freeDiscard()。

boorRuleDiscard()为农民 AI 管上出牌函数，策略是：如果上家是手牌较多的农民则管上，手牌较少则不出，调用 dontDiscard()。是地主则在对方手牌较多时考虑保留手牌，手牌较少则能管就管。

地主 AI：

lordFreeDisard()为地主自由出牌，直接调用 freeDiscard()。

lordRuleDiscard()为地主管上出牌，策略是：农民手牌都较多时保留手牌，较少时则能管就管。

还有许多游戏普遍的功能如图片切换与隐藏、倒计时、按钮回调等就不在此赘述。具体可参考源码。

● 收获

①巧用关系运算符可以写出很简洁的代码。

在 AI 出牌策略的选择中，根据寻找顺序：顺子、三张、二张、单张、炸弹、四张、王炸，可以只用一个 if 语句实现：

```
if ( findRuleSHUN3() || findRuleSHUN2() || findRuleSHUN()
   || findRule3_DUI() || findRule3_DAN() || findRuleDUI() || findRuleDAN()
   || findRule4_DAN_DAN() || findRule4_DUI()
   || findRuleWANG_ZHA() || findRuleBOMB() ) {
```

根据 C#中关系运算符||的检测规则：找到第一个满足条件的操作数就停止向下执行，以上的代码不仅规定了 findRuleXXXX()系列函数的调用顺序，而且确保了实际被执行的函数中只有一个函数能执行返回 true，这样就能保证 hints 中保留了正确的结果。

②AI 与人类相比一个显著的优势就是无限记牌，老道的斗地主玩家可以记住打出的 A、2、王，但人的记忆力毕竟是有限的，而 AI 可以轻易记住打出的所有手牌，再根据算法做出当前最佳的出牌行为。这给人的启迪是巨大的。

③巧用委托可以减少代码量

在 PokerManager 中将一系列函数放入了委托数组，这样每次调用时就省去了大量 switch 语句，使代码更简洁。

● 不足

①代码量将近 1500 行，可能由于前期构思问题还存在一些多余的代码，略显冗余。

②斗地主 AI 是一个很大的主题（实际上 AI 已经成了现代游戏的一个新方向，也有了专门的 AI 接口），网上有许多关于它的复杂算法，而这里由于本人精力有限，只采用了一些简单的算法，留下了一些遗憾。

③模式较为单一，完整的斗地主游戏应该包含其他模式如癞子模式，每次游戏结束后也应该记分，但游戏本人精力有限而没能实现。

④图形布局直接拖拽，固定了各个控件的绝对坐标，而导致不能兼容不同的分辨率，这是游戏制作很大的缺陷。