斗地主AI算法——第五章の总值计算

原文地址：<https://blog.csdn.net/sm9sun/article/details/70821106#comments>

本章算是比较重点的一章，前一章已经对各个牌型做出了价值定义，本章主要实现计算手牌总价值模块函数。

根据之前的思路，我们设定一下输入输出：

输入：手牌数据类（主要用手牌个数nHandCardCount以及手牌状态数组clsHandCardData.value\_aHandCardList）

输出：手牌价值结构 HandCardValue

先处理剪枝部分，如果剩下的手牌是一手牌，我们即直接返回该牌型的价值，这个下一章会写出，我们先假设存在这个函数ins\_SurCardsType。

其返回值设计：若是一手牌，返回牌型，若不是一手牌，返回错误牌型。

**[cpp]** [view plain](https://blog.csdn.net/sm9sun/article/details/70821106) [copy](https://blog.csdn.net/sm9sun/article/details/70821106)

CardGroupData uctCardGroupData = ins\_SurCardsType(clsHandCardData.value\_aHandCardList);

//如果能一次性出去且没有炸弹，因为有炸弹的话权值可能会更大

**if** (uctCardGroupData.cgType != cgERROR&& !HasBoom(clsHandCardData.value\_aHandCardList))

{

    uctHandCardValue.SumValue = uctCardGroupData.nValue;

    uctHandCardValue.NeedRound = 1;

**return** uctHandCardValue;

}

若不是一手牌的情况，我们通过get\_PutCardList函数（后续几章会写出实现方法）获取最佳的出牌策略 进行出牌，

**[cpp]** [view plain](https://blog.csdn.net/sm9sun/article/details/70821106) [copy](https://blog.csdn.net/sm9sun/article/details/70821106)

/\*只是获取出牌的序列，即clsHandCardData.value\_nPutCardList及clsHandCardData.uctPutCardType

    其他成员均无改变，也不会调用出牌函数，get\_PutCardList返回最优方案\*/

    get\_PutCardList\_2(clsHandCardData);

    //要保存当前的clsHandCardData.value\_nPutCardList及clsHandCardData.uctPutCardType用于回溯

    CardGroupData NowPutCardType = clsHandCardData.uctPutCardType;

    vector<**int**> NowPutCardList = clsHandCardData.value\_nPutCardList;

**if** (clsHandCardData.uctPutCardType.cgType == cgERROR)

    {

        cout << "PutCardType ERROR!" << endl;

    }

获取完出牌序列后，打出这些牌（只改变value\_aHandCardList数组），再计算剩余的牌总价值。

**[cpp]** [view plain](https://blog.csdn.net/sm9sun/article/details/70821106) [copy](https://blog.csdn.net/sm9sun/article/details/70821106)

**for** (vector<**int**>::iterator iter = NowPutCardList.begin();

        iter != NowPutCardList.end(); iter++)

    {

        clsHandCardData.value\_aHandCardList[\*iter]--;

    }

    clsHandCardData.nHandCardCount -= NowPutCardType.nCount;

    //---回溯↑

    HandCardValue tmp\_SurValue = get\_HandCardValue(clsHandCardData);//递归剩余牌价值

    //---回溯↓

**for** (vector<**int**>::iterator iter = NowPutCardList.begin();

        iter != NowPutCardList.end(); iter++)

    {

        clsHandCardData.value\_aHandCardList[\*iter]++;

    }

    clsHandCardData.nHandCardCount += NowPutCardType.nCount;

    //---回溯↑

    uctHandCardValue.SumValue = NowPutCardType.nValue + tmp\_SurValue.SumValue;

    uctHandCardValue.NeedRound = tmp\_SurValue.NeedRound + 1;

最后将返回的价值与当前打出的牌价值相加即时该阶段手牌总价值。

下面给出完整代码

**[cpp]** [view plain](https://blog.csdn.net/sm9sun/article/details/70821106) [copy](https://blog.csdn.net/sm9sun/article/details/70821106)

/\*

通过回溯dp的方式获取手牌价值

与get\_PutCardList作为交替递归调用

返回：价值结构体HandCardValue

权值的计算规则参考头文件评分逻辑思维

\*/

HandCardValue get\_HandCardValue(HandCardData &clsHandCardData)

{

    //首先清空出牌队列，因为剪枝时是不调用get\_PutCardList的

    clsHandCardData.ClearPutCardList();

    HandCardValue uctHandCardValue;

    //出完牌了，其实这种情况只限于手中剩下四带二且被动出牌的情况，因为四带二剪枝做了特殊处理。

**if** (clsHandCardData.nHandCardCount == 0)

    {

        uctHandCardValue.SumValue = 0;

        uctHandCardValue.NeedRound = 0;

**return** uctHandCardValue;

    }

    //————以下为剪枝：判断是否可以一手出完牌

    CardGroupData uctCardGroupData = ins\_SurCardsType(clsHandCardData.value\_aHandCardList);

    //————不到万不得已我们都不会出四带二，都尽量保炸弹

**if** (uctCardGroupData.cgType != cgERROR&&uctCardGroupData.cgType != cgFOUR\_TAKE\_ONE&&uctCardGroupData.cgType != cgFOUR\_TAKE\_TWO)

    {

        uctHandCardValue.SumValue = uctCardGroupData.nValue;

        uctHandCardValue.NeedRound = 1;

**return** uctHandCardValue;

    }

    //非剪枝操作，即非一手能出完的牌

    /\*只是获取出牌的序列，即clsHandCardData.value\_nPutCardList及clsHandCardData.uctPutCardType

    其他成员均无改变，也不会调用出牌函数，get\_PutCardList返回最优方案\*/

    get\_PutCardList\_2(clsHandCardData);

    //要保存当前的clsHandCardData.value\_nPutCardList及clsHandCardData.uctPutCardType用于回溯

    CardGroupData NowPutCardType = clsHandCardData.uctPutCardType;

    vector<**int**> NowPutCardList = clsHandCardData.value\_nPutCardList;

**if** (clsHandCardData.uctPutCardType.cgType == cgERROR)

    {

        cout << "PutCardType ERROR!" << endl;

    }

      //---回溯↓

**for** (vector<**int**>::iterator iter = NowPutCardList.begin();

        iter != NowPutCardList.end(); iter++)

    {

        clsHandCardData.value\_aHandCardList[\*iter]--;

    }

    clsHandCardData.nHandCardCount -= NowPutCardType.nCount;

    //---回溯↑

    HandCardValue tmp\_SurValue = get\_HandCardValue(clsHandCardData);//递归剩余牌价值

    //---回溯↓

**for** (vector<**int**>::iterator iter = NowPutCardList.begin();

        iter != NowPutCardList.end(); iter++)

    {

        clsHandCardData.value\_aHandCardList[\*iter]++;

    }

    clsHandCardData.nHandCardCount += NowPutCardType.nCount;

    //---回溯↑

    uctHandCardValue.SumValue = NowPutCardType.nValue + tmp\_SurValue.SumValue;

    uctHandCardValue.NeedRound = tmp\_SurValue.NeedRound + 1;

**return** uctHandCardValue;

  }

注：当前出牌策略是2.0版本，所以是get\_PutCardList\_2，后续还会写出更多的版本，敬请期待。

如果你之前了解回溯算法的话，那么应该很好理解上述代码，若没有接触过回溯，可以看我以前的博客<http://blog.csdn.net/sm9sun/article/details/53244484>

总值计算的模块就算完成了，其实并没有太多东西，除了一个回溯递归以外就是里面调用了两个函数ins\_SurCardsType判断是否是一手牌以及get\_PutCardList出牌逻辑。那么下一章我们便要实现这个ins\_SurCardsType函数。