斗地主AI算法——第八章の被动出牌(2)

原文地址：https://blog.csdn.net/sm9sun/article/details/70844544

上一章我们已经搭好了被动出牌的基本架子，本章我们主要说明一下被动出牌算法的基本步骤。

我把出牌逻辑分为四个阶段，也就是策略的优先级。分别是：【直接打光手牌】→【同类型牌压制】→【炸弹王炸压制】→【不出】

第一阶段【直接打光手牌】就是说如果我们可以一次性把手牌打出，那就不用考虑接下来价值之类的问题了，因为已经赢了。这种情况可能是对方打出的牌型和你一样且你比他大，或者你剩的牌是炸弹王炸。

我们先以单牌为例：

**[cpp]** [view plain](https://blog.csdn.net/sm9sun/article/details/70844544) [copy](https://blog.csdn.net/sm9sun/article/details/70844544)

//剪枝：如果能出去最后一手牌直接出

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

**if** (SurCardGroupData.cgType != cgERROR)

        {

**if** (SurCardGroupData.cgType == cgSINGLE&&SurCardGroupData.nMaxCard>clsGameSituation.uctNowCardGroup.nMaxCard)

            {

                Put\_All\_SurCards(clsGameSituation, clsHandCardData, SurCardGroupData);

**return**;

            }

**else** **if** (SurCardGroupData.cgType == cgBOMB\_CARD|| SurCardGroupData.cgType == cgKING\_CARD)

            {

                Put\_All\_SurCards(clsGameSituation, clsHandCardData, SurCardGroupData);

**return**;

            }

        }

也就是通过调用第六章实现的判定是否为一手牌函数，如果返回的类型与当前牌型相同且nMaxCard值更大一些，或者是炸弹王炸。那么直接打光所有手牌。

Put\_All\_SurCards打光所有手牌实现方法：

**[cpp]** [view plain](https://blog.csdn.net/sm9sun/article/details/70844544) [copy](https://blog.csdn.net/sm9sun/article/details/70844544)

/\*封装好的  将所有的牌都打出\*/

**void** Put\_All\_SurCards(GameSituation &clsGameSituation, HandCardData &clsHandCardData, CardGroupData SurCardGroupData)

{

    /\*全部出完\*/

**for** (**int** i = 0; i < 18; i++)

**for** (**int** j = 0; j< clsHandCardData.value\_aHandCardList[i]; j++)

            clsHandCardData.value\_nPutCardList.push\_back(i);

    /\*\*\*\*\*\*\*\*\*\*/

    clsHandCardData.uctPutCardType = clsGameSituation.uctNowCardGroup = SurCardGroupData;

**return**;

}

**void** Put\_All\_SurCards( HandCardData &clsHandCardData, CardGroupData SurCardGroupData)

{

    /\*全部出完\*/

**for** (**int** i = 0; i < 18; i++)

**for** (**int** j = 0; j< clsHandCardData.value\_aHandCardList[i]; j++)

            clsHandCardData.value\_nPutCardList.push\_back(i);

    /\*\*\*\*\*\*\*\*\*\*/

    clsHandCardData.uctPutCardType  = SurCardGroupData;

**return**;

}

第二阶段【同类型牌压制】就是需要遍历当前手牌满足可以管上的组合，然后选出最优解。我们先做一些准备工作，因为要考虑出牌和不出牌收益情况，所以我们先计算出当前手牌的价值，之所以把原始牌型轮数+1也是为了在这里若能抢占一轮尽量出牌管上。当然，若管完之后的剩余价值损失的太大就只能算了。

还需要设置暂存最佳牌号的变量、是否出牌的标志。

**[cpp]** [view plain](https://blog.csdn.net/sm9sun/article/details/70844544) [copy](https://blog.csdn.net/sm9sun/article/details/70844544)

//暂存最佳的价值

        HandCardValue BestHandCardValue = get\_HandCardValue(clsHandCardData);

        //我们认为不出牌的话会让对手一个轮次，即加一轮（权值减少7）便于后续的对比参考。

        BestHandCardValue.NeedRound += 1;

        //暂存最佳的牌号

**int** BestMaxCard=0;

        //是否出牌的标志

**bool** PutCards = **false**;

然后就是循环遍历满足条件的若干个选择，选出最优的解决方案。还是以单牌为例：

**[cpp]** [view plain](https://blog.csdn.net/sm9sun/article/details/70844544) [copy](https://blog.csdn.net/sm9sun/article/details/70844544)

**for** (**int** i = clsGameSituation.uctNowCardGroup.nMaxCard + 1; i < 18; i++)

{

**if** (clsHandCardData.value\_aHandCardList[i] > 0)

    {

        //尝试打出一张牌，估算剩余手牌价值

        clsHandCardData.value\_aHandCardList[i]--;

        clsHandCardData.nHandCardCount--;

        HandCardValue tmpHandCardValue=get\_HandCardValue(clsHandCardData);

        clsHandCardData.value\_aHandCardList[i]++;

        clsHandCardData.nHandCardCount++;

        //选取总权值-轮次\*7值最高的策略  因为我们认为剩余的手牌需要n次控手的机会才能出完，若轮次牌型很大（如炸弹） 则其-7的价值也会为正

**if** ((BestHandCardValue.SumValue-(BestHandCardValue.NeedRound\*7)) <= (tmpHandCardValue.SumValue-(tmpHandCardValue.NeedRound\*7)))

        {

            BestHandCardValue = tmpHandCardValue;

            BestMaxCard = i;

            PutCards = **true**;

        }

    }

}

**if** (PutCards)

{

    clsHandCardData.value\_nPutCardList.push\_back(BestMaxCard);

    clsHandCardData.uctPutCardType = clsGameSituation.uctNowCardGroup = get\_GroupData(cgSINGLE, BestMaxCard, 1);

**return**;

}

按照之前价值的定义，我们对比的公式为总价值-轮数\*7。

第三阶段【炸弹王炸压制】的策略与上文逻辑类似，唯一的区别就是加了一个手牌剩余价值的判定，就是如果我出完炸剩余手牌价值还蛮可观的话，我们就可以任性的炸出，

毕竟此时我们获胜的几率很大。

**[cpp]** [view plain](https://blog.csdn.net/sm9sun/article/details/70844544) [copy](https://blog.csdn.net/sm9sun/article/details/70844544)

**for** (**int** i = 3; i < 16; i++)

        {

**if** (clsHandCardData.value\_aHandCardList[i] == 4)

            {

                //尝试打出炸弹，估算剩余手牌价值,因为炸弹可以参与顺子，不能因为影响顺子而任意出炸

                clsHandCardData.value\_aHandCardList[i] -= 4;

                clsHandCardData.nHandCardCount -= 4;

                HandCardValue tmpHandCardValue = get\_HandCardValue(clsHandCardData);

                clsHandCardData.value\_aHandCardList[i] += 4;

                clsHandCardData.nHandCardCount += 4;

                //选取总权值-轮次\*7值最高的策略  因为我们认为剩余的手牌需要n次控手的机会才能出完，若轮次牌型很大（如炸弹） 则其-7的价值也会为正

**if** ((BestHandCardValue.SumValue - (BestHandCardValue.NeedRound \* 7)) <= (tmpHandCardValue.SumValue - (tmpHandCardValue.NeedRound \* 7))

                    //如果剩余手牌价值为正，证明出去的几率很大， 那么可以用炸获得先手

                    || tmpHandCardValue.SumValue > 0)

                {

                    BestHandCardValue = tmpHandCardValue;

                    BestMaxCard = i;

                    PutCards = **true**;

                }

            }

        }

**if** (PutCards)

        {

            clsHandCardData.value\_nPutCardList.push\_back(BestMaxCard);

            clsHandCardData.value\_nPutCardList.push\_back(BestMaxCard);

            clsHandCardData.value\_nPutCardList.push\_back(BestMaxCard);

            clsHandCardData.value\_nPutCardList.push\_back(BestMaxCard);

            clsHandCardData.uctPutCardType = clsGameSituation.uctNowCardGroup = get\_GroupData(cgBOMB\_CARD, BestMaxCard, 4);

**return**;

        }

        //王炸

**if** (clsHandCardData.value\_aHandCardList[17] > 0 && clsHandCardData.value\_aHandCardList[16] > 0)

        {

            //如果剩余手牌价值为正，证明出去的几率很大，那么可以用炸获得先手，王炸20分

**if** (BestHandCardValue.SumValue > 20)

            {

                clsHandCardData.value\_nPutCardList.push\_back(17);

                clsHandCardData.value\_nPutCardList.push\_back(16);

                clsHandCardData.uctPutCardType = clsGameSituation.uctNowCardGroup = get\_GroupData(cgKING\_CARD, 17, 2);

**return**;

            }

        }

若以上三个阶段都没有return的话，就进入我们第四阶段了。

第四阶段【不出】

**[cpp]** [view plain](https://blog.csdn.net/sm9sun/article/details/70844544) [copy](https://blog.csdn.net/sm9sun/article/details/70844544)

//管不上

    clsHandCardData.uctPutCardType = get\_GroupData(cgZERO, 0, 0);

**return**;

这也就是我们被动出牌算法的基本步骤，本章是以单牌为例，后续文章会进而补充其他的牌型。