斗地主AI算法——第九章の被动出牌(3)

原文地址：<https://blog.csdn.net/sm9sun/article/details/70851450>

上一章已经说明了被动出牌算法基本的出牌思路，且以单牌为例给出了具体的代码。

对牌、三不带牌型实现方法与单牌基本类似。改动的地方主要是枚举牌类型，出牌时value\_nPutCardList的处理，回溯时value\_aHandCardList和nHandCardCount的变化等几个方面。下面给出完整代码：

/对牌类型

else if (clsGameSituation.uctNowCardGroup.cgType == cgDOUBLE)

{

//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 == cgDOUBLE&&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;

}

}

//---------------------------对牌-------------------------------

//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;

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

{

if (clsHandCardData.value\_aHandCardList[i] > 1)

{

//尝试打出一对牌，估算剩余手牌价值

clsHandCardData.value\_aHandCardList[i]-=2;

clsHandCardData.nHandCardCount-=2;

HandCardValue tmpHandCardValue = get\_HandCardValue(clsHandCardData);

clsHandCardData.value\_aHandCardList[i]+=2;

clsHandCardData.nHandCardCount+=2;

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

clsHandCardData.uctPutCardType = clsGameSituation.uctNowCardGroup = get\_GroupData(cgDOUBLE, BestMaxCard, 2);

return;

}

//-------------------------------------------炸弹-------------------------------------------

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;

}

}

//管不上

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

return;

}

//三牌类型

else if (clsGameSituation.uctNowCardGroup.cgType == cgTHREE)

{

//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 == cgTHREE&&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;

}

}

//-------------------------------------------三牌-------------------------------------------

//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;

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

{

if (clsHandCardData.value\_aHandCardList[i] > 2)

{

//尝试打出一对牌，估算剩余手牌价值

clsHandCardData.value\_aHandCardList[i] -= 3;

clsHandCardData.nHandCardCount -= 3;

HandCardValue tmpHandCardValue = get\_HandCardValue(clsHandCardData);

clsHandCardData.value\_aHandCardList[i] += 3;

clsHandCardData.nHandCardCount += 3;

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

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

clsHandCardData.uctPutCardType = clsGameSituation.uctNowCardGroup = get\_GroupData(cgTHREE, BestMaxCard, 3);

return;

}

//-------------------------------------------炸弹-------------------------------------------

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;

}

}

//管不上

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

return;

}

接下来我们说一下顺子的处理方法。以单顺为例：

首先在第一阶段判断是否同类型牌里要额外增加一个条件，即顺子长度要一致

if (SurCardGroupData.cgType == cgSINGLE\_LINE&&SurCardGroupData.nMaxCard>clsGameSituation.uctNowCardGroup.nMaxCard

&&SurCardGroupData.nCount== clsGameSituation.uctNowCardGroup.nCount)

{

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

return;

}

然后除了设置BestHandCardValue等变量外，我们需要额外设置几个关于顺子的标志

//验证顺子的标志

int prov = 0;

//顺子起点

int start\_i = 0;

//顺子终点

int end\_i = 0;

//顺子长度

int length = clsGameSituation.uctNowCardGroup.nCount;

遍历顺子的方法有点类似于最大子段和问题，大家可以参考下我以前的博客http://blog.csdn.net/sm9sun/article/details/53240992

解决思路就是如果出现某张牌个数为0，那么必然不存在经过他的顺子，此时就把计数器置零，如果计数器长度大于等于length，即可以组成顺子，我们以当前下标i为最高标志构造出(i-length+1)~i的顺子。

举个例子：对方牌型为34567，我从4遍历至8，若满足，此时end\_i=8，即45678，继续走到9，若还满足，end\_i=9。即56789，若没有10，则prov归零，下一次循环若11存在，则prov=1。

for (int i = clsGameSituation.uctNowCardGroup.nMaxCard - length + 2; i < 15; i++)

{

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

{

prov++;

}

else

{

prov = 0;

}

if (prov >= length)

{

end\_i = i;

start\_i = i - length + 1;

for (int j = start\_i; j <= end\_i; j++)

{

clsHandCardData.value\_aHandCardList[j] --;

}

clsHandCardData.nHandCardCount -= clsGameSituation.uctNowCardGroup.nCount;

HandCardValue tmpHandCardValue = get\_HandCardValue(clsHandCardData);

for (int j = start\_i; j <= end\_i; j++)

{

clsHandCardData.value\_aHandCardList[j] ++;

}

clsHandCardData.nHandCardCount += clsGameSituation.uctNowCardGroup.nCount;

//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 = end\_i;

PutCards = true;

}

}

}

最后打出顺子的话在start\_i和 end\_i区间内依次减一即可。

if (PutCards)

{

for (int j = start\_i; j <= end\_i; j++)

{

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

}

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

return;

}

以上就是单顺的处理方法，下一章我们继续填充其他牌型的出牌方法。