**斗地主AI算法——第十章の被动出牌(4)**

原文地址：<https://blog.csdn.net/sm9sun/article/details/70853139>

上一章已经说明了单顺的实现方法，双顺、三不带顺牌型实现方法与单牌基本类似。改动的地方除了上一章说的枚举牌类型，出牌时value\_nPutCardList的处理，回溯时value\_aHandCardList和nHandCardCount的变化等几个方面，还有length设置的变化，因为双顺的length是count的1/2，三顺是1/3。

下面给出完整代码：

//对连类型

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

{

//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\_LINE&&SurCardGroupData.nMaxCard>clsGameSituation.uctNowCardGroup.nMaxCard

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

{

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

return;

}

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

{

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

return;

}

}

//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;

//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/2;

//2与王不参与顺子，从当前已打出的顺子最小牌值+1开始遍历

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

{

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

{

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] -=2;

}

clsHandCardData.nHandCardCount -= clsGameSituation.uctNowCardGroup.nCount;

HandCardValue tmpHandCardValue = get\_HandCardValue(clsHandCardData);

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

{

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

}

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;

}

}

}

if (PutCards)

{

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

{

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

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

}

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

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;

}

//三连类型

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

{

//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\_LINE&&SurCardGroupData.nMaxCard>clsGameSituation.uctNowCardGroup.nMaxCard

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

{

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

return;

}

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

{

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

return;

}

}

//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;

//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 / 3;

//2与王不参与顺子，从当前已打出的顺子最小牌值+1开始遍历

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

{

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

{

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] -= 3;

}

clsHandCardData.nHandCardCount -= clsGameSituation.uctNowCardGroup.nCount;

HandCardValue tmpHandCardValue = get\_HandCardValue(clsHandCardData);

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

{

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

}

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;

}

}

}

if (PutCards)

{

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

{

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

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

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

}

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

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;

}

接下来我们讲三带一的算法，其实也很简单，无非就是在确定了三牌之后再通过循环枚举出能够返回最佳权值的顺出去的那张牌。

三带一单：

//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;

//顺带出去的牌

int tmp\_1 = 0;

//是否出牌的标志

bool PutCards = false;

//三带一

for (int i = clsGameSituation.uctNowCardGroup.nMaxCard + 1; i < 16; i++)

{

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

{

for (int j = 3; j < 18; j++)

{

//选出一张以上的牌且不是选择三张的那个牌

if (clsHandCardData.value\_aHandCardList[j] > 0 && j != i)

{

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

clsHandCardData.value\_aHandCardList[j] -= 1;

clsHandCardData.nHandCardCount -= 4;

HandCardValue tmpHandCardValue = get\_HandCardValue(clsHandCardData);

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

clsHandCardData.value\_aHandCardList[j] += 1;

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)))

{

BestHandCardValue = tmpHandCardValue;

BestMaxCard = i;

tmp\_1 = j;

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(tmp\_1);

clsHandCardData.uctPutCardType = clsGameSituation.uctNowCardGroup = get\_GroupData(cgTHREE\_TAKE\_ONE, BestMaxCard, 4);

return;

}

三带一对：

for (int i = clsGameSituation.uctNowCardGroup.nMaxCard + 1; i < 16; i++)

{

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

{

for (int j = 3; j < 16; j++)

{

//选出一张以上的牌且不是选择三张的那个牌

if (clsHandCardData.value\_aHandCardList[j] > 1 && j != i)

{

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

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

clsHandCardData.nHandCardCount -= 5;

HandCardValue tmpHandCardValue = get\_HandCardValue(clsHandCardData);

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

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

clsHandCardData.nHandCardCount += 5;

//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;

tmp\_1 = j;

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(tmp\_1);

clsHandCardData.value\_nPutCardList.push\_back(tmp\_1);

clsHandCardData.uctPutCardType = clsGameSituation.uctNowCardGroup = get\_GroupData(cgTHREE\_TAKE\_TWO, BestMaxCard, 5);

return;

}

这里我们需要注意的是，因为i和j我们是放在一起减的，所以要保证i和j不能相等。即if (clsHandCardData.value\_aHandCardList[j] > 1 && j != i)

四带二与其类似，无非就是再多一个循环。

四带二单：

for (int i = clsGameSituation.uctNowCardGroup.nMaxCard + 1; i < 16; i++)

{

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

{

for (int j = 3; j < 18; j++)

{

//先选出一张以上的牌且不是选择四张的那个牌

if (clsHandCardData.value\_aHandCardList[j] > 0 && j != i)

{

//再选出一张以上的牌且不是选择四张的那个牌且不是第一次选的两张内个牌（策略里四带二不允许带一对,还不如炸）

for (int k = j + 1; k < 18; k++)

{

if (clsHandCardData.value\_aHandCardList[k] > 0 && k != i)

{

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

clsHandCardData.value\_aHandCardList[j] -= 1;

clsHandCardData.value\_aHandCardList[k] -= 1;

clsHandCardData.nHandCardCount -= 6;

HandCardValue tmpHandCardValue = get\_HandCardValue(clsHandCardData);

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

clsHandCardData.value\_aHandCardList[j] += 1;

clsHandCardData.value\_aHandCardList[k] += 1;

clsHandCardData.nHandCardCount += 6;

//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;

tmp\_1 = j;

tmp\_2 = k;

PutCards = true;

}

}

}

}

}

}

}

四带二单的情况时k时从j+1开始算的，因为我们出牌的策略不允许四带二单带出去一个对，这样还不如直接炸了划算。若此处想修改的话，一定要注意判断i j k不能相等。

或者可以参考下一章处理飞机的方法，内个相对安全一些。

四带二的算法一样，都是修改枚举牌类型，出牌时value\_nPutCardList的处理，回溯时value\_aHandCardList和nHandCardCount这几处。

不过毕竟四带二这种牌型可以拆分出炸弹，所以我在选择四带二出牌前加入了一层判定：

/\*本策略对于这种牌型有炸必炸，四带二相比炸弹来说会多带出两手牌，即最多提高14权值的价值

若当前手牌价值大于14，即认为我们能炸即炸，不必考虑四带二的收益，就是这么任性。\*/

if (BestHandCardValue.SumValue > 14)

{

//炸弹——这里直接炸，不考虑拆分后果。因为信仰。

for (int i = 3; i < 16; i++)

{

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

{

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

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

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

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

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

return;

}

}

//王炸

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

{

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

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

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

return;

}

}

以及最后管不上出炸判断中，直接炸出。

//这里是在拍权值较小的情况，且没有选择出良好的四带二牌型，那么也要炸，因为信仰。

for (int i = 3; i < 16; i++)

{

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

{

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

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

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

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

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

return;

}

}

//王炸

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

{

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

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

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

return;

}

这个纯属个人信仰了，我觉得当对手打出四带二这种牌型，我有炸无论如何都是要炸的。

也就是对于四带二这种牌型我的出牌策略是：先看自己的手牌实力如何，如果较好直接炸。如果不太好，那么再看看以四带二形式管出后实力如何，如果较好，可以四带二管上，如果还是不太好，那么也是要炸，输就输了！！

本章我们把三带一、四带二牌型讲完了，下一章我们再完善剩余的飞机、炸弹牌型，也就是被动出牌最后的一部分了。