**斗地主AI算法——第十一章の被动出牌(5)**

原文地址：<https://blog.csdn.net/sm9sun/article/details/70854470>

本章是被动出牌的最后一章，截止目前，我们已经解决了大部分牌型。只剩下飞机和炸弹了。

飞机无疑是最复杂的类型，他等于顺子和三带的结合体，但又增加了很多难度。

根据上一章的算法，我们可以大概想到，若是带出去一张我就加一个循环，若是带出去两张我就加俩循环，但是这个飞机长度不一致，带出去的牌个数也就不一致，这TM怎么加啊！！我一开始的想法是外置一个全排列函数，给定count个数，然后列举所有的方案去改变value\_aHandCardList数组再筛选出最优解。但这样做有坏处，第一，这个外置函数非常的麻烦，因为他全排列的全集和子集个数都不确定。第二，影响了程序分支一致性，别的模块都差不多，就飞机这里搞特殊化不太好。第三，不一定安全，因为这种做法意味着我又要多了一个可以影响value\_aHandCardList的模块，我并不希望这样。

所以思考了很久，我觉得宁愿多做几个分支，毕竟飞机的个数还是可控的，就2-4。虽然这种代码写起来跟看起来都很傻逼，但是也没有办法。。。

飞机带单：

//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;

//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 / 4;

int tmp\_1 = 0;

int tmp\_2 = 0;

int tmp\_3 = 0;

int tmp\_4 = 0;

//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;

/\*本来想做全排列选取带出的牌然后枚举出最高价值的，但考虑到当飞机长度也就是在2-4之间

所以干脆做三个分支处理算了\*/

//为两连飞机

if (length == 2)

{

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

{

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

{

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

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

{

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

{

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

HandCardValue tmpHandCardValue = get\_HandCardValue(clsHandCardData);

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

//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;

tmp\_1 = j;

tmp\_2 = k;

PutCards = true;

}

}

}

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

}

}

}

//为三连飞机

if (length == 3)

{

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

{

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

{

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

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

{

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

{

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

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

{

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

{

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

HandCardValue tmpHandCardValue = get\_HandCardValue(clsHandCardData);

//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;

tmp\_1 = j;

tmp\_2 = k;

tmp\_3 = l;

PutCards = true;

}

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

}

}

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

}

}

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

}

}

}

//为四连飞机

if (length == 4)

{

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

{

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

{

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

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

{

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

{

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

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

{

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

{

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

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

{

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

{

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

HandCardValue tmpHandCardValue = get\_HandCardValue(clsHandCardData);

//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;

tmp\_1 = j;

tmp\_2 = k;

tmp\_3 = l;

tmp\_4 = m;

PutCards = true;

}

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

}

}

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

}

}

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

}

}

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

}

}

}

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

{

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

}

clsHandCardData.nHandCardCount += clsGameSituation.uctNowCardGroup.nCount;

}

}

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);

}

if (length == 2)

{

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

clsHandCardData.value\_nPutCardList.push\_back(tmp\_2);

}

if (length == 3)

{

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

clsHandCardData.value\_nPutCardList.push\_back(tmp\_2);

clsHandCardData.value\_nPutCardList.push\_back(tmp\_3);

}

if (length == 4)

{

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

clsHandCardData.value\_nPutCardList.push\_back(tmp\_2);

clsHandCardData.value\_nPutCardList.push\_back(tmp\_3);

clsHandCardData.value\_nPutCardList.push\_back(tmp\_4);

}

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

return;

}

大家可以看到我回溯的处理方式和之前的不一样了，因为飞机类型很有可能把对牌当成两个单牌带出，甚至可以拆炸弹。所以每个循环内当确定了一个点就先处理value\_aHandCardList状态，这样也相对安全，上一章中在四带二环节我也有提到过这方面。

飞机带对类似，而且这里是被动出牌，所以不存在4连飞机的情况，因为4连飞机带对的话就有20张牌了。只考虑2连和3连就可以了。

最后再说一下炸弹，这个炸弹就厉害了，我给的策略就是—直接炸丫的！不要怂！！

else if (clsGameSituation.uctNowCardGroup.cgType == cgBOMB\_CARD)

{

//更大的炸弹——这里直接炸，不考虑拆分后果。因为信仰。

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

}

//管不上

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

return;

}

当然也可以改成考虑拆分后果什么的，或者如果你手上有多个炸弹是否对比一下出那个接下来更好 等等逻辑。

不过对于我来说，你都有俩炸弹了，还怕什么，肯定都是要炸的！宁输不拆！就是这么浪！

好了至此被动出牌模块就全部写完了，从下一章开始，我们讲主动出牌。