# 斗地主AI算法——第三章の数据处理

原文地址 ：<https://blog.csdn.net/sm9sun/article/details/70807809>

上一章我们定义了基本的数据结构，相信大家看到手牌类里面那么多出牌序列时一定会比较愤慨。。。其实一开始写的时候我也是觉得很脑残，不过后续开发证明了这样的结构还是可以的，因为只要我封装了一层数据转换，接下来所有的算法策略都只用到2个成员变量，状态数据及手牌数量。特别便于调试、管理。那么接下来就写出类成员函数的实现方法

[cpp] [view plain](https://blog.csdn.net/sm9sun/article/details/70807809) [copy](https://blog.csdn.net/sm9sun/article/details/70807809)

//手牌数据类

**class** HandCardData

{

**public**:

    //构造函数

    HandCardData::HandCardData()

    {

    }

    //析构函数

**virtual** HandCardData::~HandCardData()

    {

    }

**public**:

      //手牌序列——无花色，值域3~17

    vector <**int**> value\_nHandCardList;

      //手牌序列——状态记录，便于一些计算，值域为该index牌对应的数量0~4

**int** value\_aHandCardList[18] = { 0 };

      //手牌序列——有花色，按照从大到小的排列  56，52：大王小王……4~0：红3黑3方3花3

    vector <**int**> color\_nHandCardList;

      //手牌个数

**int** nHandCardCount = 17 ;

      //玩家角色地位       0：地主    1：农民——地主下家   2：农民——地主上家

**int** nGameRole = -1;

      //玩家座位ID

**int** nOwnIndex = -1;

    //玩家要打出去的牌类型

    CardGroupData uctPutCardType;

    //要打出去的牌——无花色

    vector <**int**> value\_nPutCardList;

    //要打出去的牌——有花色

    vector <**int**> color\_nPutCardList;

    HandCardValue uctHandCardValue;

**public**:

    //要打出的牌序列清空

**void** ClearPutCardList();

    //手牌排序，大牌靠前

**void** SortAsList(vector <**int**> &arr);

    //出一张牌，返回操作是否合法

**bool** PutOneCard(**int** value\_nCard, **int** &clear\_nCard);

    //出一组牌，返回操作是否合法

**bool** PutCards();

    //通过有花色手牌获取无花色手牌（包含两种结构）

**void** get\_valueHandCardList();

    //初始化

**void** Init();

    //输出所有成员变量，用于测试

**void** PrintAll();

 };

void HandCardData::ClearPutCardList() 是把要出的牌打入出牌序列前清空现列表的操作，含有花色和无花色，顺便把之前出牌类型的值初始化一下

[cpp] [view plain](https://blog.csdn.net/sm9sun/article/details/70807809) [copy](https://blog.csdn.net/sm9sun/article/details/70807809)

**void** HandCardData::ClearPutCardList()

{

    color\_nPutCardList.clear();

    value\_nPutCardList.clear();

    uctPutCardType.cgType = cgERROR;

    uctPutCardType.nCount = 0;

    uctPutCardType.nMaxCard = -1;

    uctPutCardType.nValue = 0;

**return**;

}

void HandCardData::SortAsList(vector <int> & arr )简单的排序，这个就不说了

[cpp] [view plain](https://blog.csdn.net/sm9sun/article/details/70807809) [copy](https://blog.csdn.net/sm9sun/article/details/70807809)

/\*降序排序对比\*/

**int** cmp(**int** a, **int** b) { **return** a > b ? 1 : 0; }

**void** HandCardData::SortAsList(vector <**int**> & arr )

{

    sort(arr.begin(), arr.end(), cmp);

**return**;

}

void HandCardData::get\_valueHandCardList()根据获取的有花色手牌序列转换成无花色手牌序列

我们的花色定义是按照从大到小的排列  56，52：大王小王……4~0：红3黑3方3花3  所以花色值/4再加上最小的牌3就是我们要的无花色权值

注：2对应的值是15 A对应的值是14

[cpp] [view plain](https://blog.csdn.net/sm9sun/article/details/70807809) [copy](https://blog.csdn.net/sm9sun/article/details/70807809)

**void** HandCardData::get\_valueHandCardList()

{

    //清零

    value\_nHandCardList.clear();

    memset(value\_aHandCardList, 0,**sizeof**(value\_aHandCardList));

**for** (vector<**int**>::iterator iter = color\_nHandCardList.begin(); iter != color\_nHandCardList.end(); iter++)

    {

        value\_nHandCardList.push\_back((\*iter / 4)+3);

        value\_aHandCardList[(\*iter / 4) + 3]++;

    }

}

void HandCardData::Init()手牌的初始化，主要用于根据获取的有花色手牌序列转换成无花色手牌序列，手牌序列排序， 计算出手牌个数。

[cpp] [view plain](https://blog.csdn.net/sm9sun/article/details/70807809) [copy](https://blog.csdn.net/sm9sun/article/details/70807809)

**void** HandCardData::Init()

{

    //根据花色手牌获取权值手牌

    get\_valueHandCardList();

    //手牌 排序

    SortAsList(color\_nHandCardList);

    SortAsList(value\_nHandCardList);

    //当前手牌个数

    nHandCardCount = value\_nHandCardList.size();

}

void HandCardData::PrintAll()就是输出一些类成员变量，测试时使用。

[cpp] [view plain](https://blog.csdn.net/sm9sun/article/details/70807809) [copy](https://blog.csdn.net/sm9sun/article/details/70807809)

**void** HandCardData::PrintAll()

{

    cout << "color\_nHandCardList:" << endl;

**for** (vector<**int**>::iterator iter = color\_nHandCardList.begin(); iter != color\_nHandCardList.end(); iter++)

        cout << get\_CardsName(\*iter) << (iter == color\_nHandCardList.end() - 1 ? '\n' : ',');

    cout << endl;

    /\*

    cout << "value\_nHandCardList：" << endl;

    for (vector<int>::iterator iter = value\_nHandCardList.begin(); iter != value\_nHandCardList.end(); iter++)

        cout << \*iter << (iter == value\_nHandCardList.end() - 1 ? '\n' : ',');

    cout << endl;

    cout << "value\_aHandCardList：" << endl;

    for (int i = 0; i < 18; i++)

    {

        cout << value\_aHandCardList[i] << (i == 17 ? '\n' : ',');

    }

    cout << endl;

    cout << "nHandCardCount:" << nHandCardCount << endl;

    cout << endl;

    cout << "nGameRole:" << nGameRole << endl;

    cout << endl;

    \*/

}

接下来就说出牌的函数了

bool  HandCardData::PutCards()出一组牌，返回操作是否合法

其函数实现为：遍历无花色手牌序列逐一映射到有花色手牌，然后将其加入有花色出牌数组里。说白了PutCards就是循环调用PutOneCard



[cpp] [view plain](https://blog.csdn.net/sm9sun/article/details/70807809) [copy](https://blog.csdn.net/sm9sun/article/details/70807809)

**bool**  HandCardData::PutCards()

{

**for** (vector<**int**>::iterator iter = value\_nPutCardList.begin(); iter != value\_nPutCardList.end(); iter++)

    {

**int** color\_nCard = -1;

**if** (PutOneCard(\*iter, color\_nCard))

        {

            color\_nPutCardList.push\_back(color\_nCard);

        }

**else**

        {

**return** **false**;

        }

    }

    nHandCardCount -= value\_nPutCardList.size();

**return** **true**;

}

重点就是出一张牌的实现方法了，bool PutOneCard(int value\_nCard, int &clear\_nCard);

这里我们需要做的事情可以分成两部分，第一部分，返回一个有花色的手牌以供PutCards加入有花色出牌序列，也就是引用的 int &clear\_nCard

第二个就是处理我们的这几个数组（value状态数组、value列表数组、color列表数组）

[cpp] [view plain](https://blog.csdn.net/sm9sun/article/details/70807809) [copy](https://blog.csdn.net/sm9sun/article/details/70807809)

**bool**  HandCardData::PutOneCard(**int** value\_nCard, **int** &color\_nCard)

{

**bool** ret = **false**;

    //value状态数组处理

    value\_aHandCardList[value\_nCard]--;

**if** (value\_aHandCardList[value\_nCard] < 0)

    {

**return** **false**;

    }

    //value列表数组处理

**for** (vector<**int**>::iterator iter = value\_nHandCardList.begin(); iter != value\_nHandCardList.end(); iter++)

    {

**if** (\*iter == value\_nCard)

        {

            value\_nHandCardList.erase(iter);

            ret = **true**;

**break**;

        }

    }

    // color列表数组处理

**int** k = (value\_nCard - 3) \* 4;      //数值转换

**for** (vector<**int**>::iterator iter = color\_nHandCardList.begin(); iter != color\_nHandCardList.end(); iter++)

    {

**for** (**int** i = k; i < k + 4; i++)

        {

**if** (\*iter == i)

            {

                color\_nCard = i;

                color\_nHandCardList.erase(iter);

**return** ret;

            }

        }

    }

**return** **false**;

}

至此，手牌类成员的数据处理函数就做完了，而全局类并没有什么需要我们处理的，因为那些都应该是我们从服务器获取的信息。

如果说这些都算是准备工作的话，那么接下来便是开始进入AI逻辑环节了，我们先从手牌权值的定义说起。

敬请关注下一章：斗地主AI算法——第四章の权值定义