# 斗地主AI算法——第二章の数据结构

原文：<https://blog.csdn.net/sm9sun/article/details/70804909>

上一章我们已经确立了基本的业务逻辑处理流程。从本章开始，我们便进入开发阶段。

首先就是明确我们都需要哪些数据，且它们以怎样的形式存储。

首先从上一章反复提到的手牌权值结构说起，也就是F（）的返回值，他包含了两个成员，①手牌总价值②需要打几手牌。

[cpp] [view plain](https://blog.csdn.net/sm9sun/article/details/70804909) [copy](https://blog.csdn.net/sm9sun/article/details/70804909)

//手牌权值结构

**struct** HandCardValue

{

**int** SumValue;        //手牌总价值

**int** NeedRound;       // 需要打几手牌

};

这个不难理解吧，我们出牌的时候总是希望把牌划分成手数少且相对价值又大的组合方式，比如说AKQJ10，虽然都挺大，但你总不见得故意拆开出吧。

接下来就是组合牌型的数据结构，就算是上一章G（）的返回值吧，因为G（）主要返回的就是一个组合牌结构及出牌的序列

[cpp] [view plain](https://blog.csdn.net/sm9sun/article/details/70804909) [copy](https://blog.csdn.net/sm9sun/article/details/70804909)

//牌型组合数据结构

**struct** CardGroupData

{

    //枚举类型

    CardGroupType cgType=cgERROR;

    //该牌的价值

**int**  nValue=0;

    //含牌的个数

**int**  nCount=0;

    //牌中决定大小的牌值，用于对比

**int** nMaxCard=0;

};

说明一下nMaxCard，比如说99，他的nMaxCard就是9，比如说33366，他的nMaxCard就是3。被动出牌规则便是通过这个值进行判断是否可以出牌。

cgType是所有牌型的枚举，具体定义为：

[cpp] [view plain](https://blog.csdn.net/sm9sun/article/details/70804909) [copy](https://blog.csdn.net/sm9sun/article/details/70804909)

//手牌组合枚举

**enum** CardGroupType

{

    cgERROR = -1,                                   //错误类型

    cgZERO = 0,                                     //不出类型

    cgSINGLE = 1,                                   //单牌类型

    cgDOUBLE = 2,                                   //对牌类型

    cgTHREE = 3,                                    //三条类型

    cgSINGLE\_LINE = 4,                              //单连类型

    cgDOUBLE\_LINE = 5,                              //对连类型

    cgTHREE\_LINE = 6,                               //三连类型

    cgTHREE\_TAKE\_ONE = 7,                           //三带一单

    cgTHREE\_TAKE\_TWO = 8,                           //三带一对

    cgTHREE\_TAKE\_ONE\_LINE = 9,                      //三带一单连

    cgTHREE\_TAKE\_TWO\_LINE = 10,                     //三带一对连

    cgFOUR\_TAKE\_ONE = 11,                           //四带两单

    cgFOUR\_TAKE\_TWO = 12,                           //四带两对

    cgBOMB\_CARD = 13,                               //炸弹类型

    cgKING\_CARD = 14                                //王炸类型

};

接下来便是两个主要的类，首先是游戏全局类，主要用于储存当前游戏的发展情况

[cpp] [view plain](https://blog.csdn.net/sm9sun/article/details/70804909) [copy](https://blog.csdn.net/sm9sun/article/details/70804909)

//游戏全局类

**class** GameSituation

{

**public**:

    //构造函数

    GameSituation::GameSituation()

    {

    }

    //析构函数

**virtual** GameSituation::~GameSituation()

    {

    }

**public**:

    //地主玩家

**int** nDiZhuID = -1;

    //本局叫分

**int** nLandScore = 0;

    //当前地主玩家——还未确定

**int** nNowDiZhuID = -1;

    //当前本局叫分——还未确定

**int** nNowLandScore = 0;

    //三张底牌

**int** DiPai[3] = { 0 };

    //已经打出的牌——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\_aAllOutCardList[18] = { 0 };

    //三名玩家已经打出的手牌记录

**int** value\_aUnitOutCardList[3][18] = { 0 };

    //三名玩家已经剩余手牌个数

**int** value\_aUnitHandCardCount[3] = { 0 };

    //本局当前底分倍数

**int** nMultiple = 0;

    //当前控手对象（用于区分是否可以自身任意出牌以及是否地主已经放弃出牌从而不去管队友）

**int** nCardDroit = 0;

    //当前打出牌的类型数据，被动出牌时玩家根据这里做出筛选

    CardGroupData uctNowCardGroup;

    //本局游戏是否结束

**bool** Over = **false**;

};

然后是自身手牌的数据类，这里重点说明的是手牌序列相关，大家可以看到，类成员里定了很多，感觉蛮晕的。

首先，在我们出牌逻辑中，是没有花色的概念的，即我们后续所有的逻辑计算只需要考虑当前手牌权值部分即可，所以定义了vector <int> value\_nHandCardList手牌序列

当我们要反馈出牌情况时，再根据返回的无花色出牌序列映射到自己有花色的手牌序列中，再返回有花色的出牌序列。这些下一章会给出实现方法。

而无花色value手牌序列为了便于计算又设立了状态记录的数组int value\_aHandCardList[18]。因为后续算法肯定需要大量通过回溯法深度遍历出牌策略的操作。

这个在后续的算法里会看到，也就是说，当程序在做出牌逻辑计算时，受影响的是int value\_aHandCardList[18]，当然最终都会回溯到原点。当确定好了出牌的序列，返回无花色出牌序列vector <int> value\_nPutCardList。最后通过处理无花色出牌序列，改变其他数组的值。

[cpp] [view plain](https://blog.csdn.net/sm9sun/article/details/70804909) [copy](https://blog.csdn.net/sm9sun/article/details/70804909)

//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;

};

最后定义一些极限值

[cpp] [view plain](https://blog.csdn.net/sm9sun/article/details/70804909) [copy](https://blog.csdn.net/sm9sun/article/details/70804909)

//最多手牌

#define HandCardMaxLen 20

//价值最小值

#define MinCardsValue -25

//价值最大值

#define MaxCardsValue 106

那么目前所需要的数据基本制定完毕，接下来是手牌中类成员函数的实现方法。