

# “理治棋壮”中国象棋计算机博弈引擎

## 基本数据结构与模块设计说明

“理治棋壮”中国象棋计算机博弈引擎开发小组

本文简述“理治棋壮”中国象棋计算机博弈引擎在编程实现时使用的基本数据结构以及模块设计方案。设计基于《程序架构设计与主要算法概述》一文，请参考阅读。对于上文中已经说明的内容不再赘述。

对于模块设计，只给出 C++ 头文件中的函数接口定义和重要的数据成员定义，略去具体实现及一些次要的变量、辅助函数定义。

本文档与 BitStronger 0.06 (build 0801) 版本保持一致。不排除在今后编码过程中，对相关设计有所增删和修改的情况。

### 一、基本数据结构

#### 1、基本常量定义：

```
///棋手（红方 / 黑方）
typedef unsigned char PLAYER;
///棋子
typedef unsigned char CHESSMAN;
///棋盘坐标（16*16表示法）
typedef unsigned char BDPOINT;
///哈希数（64位）
typedef long long HASHNUM;

///搜索棋局深度（采用迭代深化搜索后此常量暂不使用）
const int SEARCH_DEPTH = 4;
///最大搜索棋局深度（用于迭代深化搜索）
const int MAX_SEARCH_DEPTH = 16;
///静态搜索时最大延伸搜索棋局深度
const int MAX QUIES_DEPTH = 16;
///每步最长搜索时间（以秒计，用于迭代深化搜索）
const int LONGEST_SEARCH_TIME = 5;

///搜索时每层最多保存棋局个数
const int SEARCH_WIDTH = 80;
///表示当前局面是死局面的评估值，此值应小于我方失去所有棋子的分数
const int NO_BEST_MOVE = -20000;
///表示每步搜索时间超时的评估值（用于迭代深化搜索），此值应小于我方失去所有棋子的分数
const int TIME_OVER = -65432;
///表示当前局面在置换表中不存在的评估值，此值应小于我方失去所有棋子的分数
const int NOT_IN TT = -60000;
///局面正常情况（不失将帅）一方可能的最高评估值
const int MAX_VALUE_OF_BOARD = 2000;

///棋手编码
enum Player
{
    RED = 0, BLACK = 1
};

///棋子编码。注意：用unsigned char 类型的position 数组存储棋子编码，
```

```

    /*! 负数将存为其位二进制补码，如：-2 -> 254。
    NONE 表示无棋子，OUT 表示棋盘之外的位置。
    */
enum Chessman
{
    NONE = 0, OUT = 99,
    R_KING = 1, R_ADVISOR = 2, R_BISHOP = 3, R_KNIGHT = 4, R_ROOK = 5, R_CANNON = 6, R_PAWN = 7,
    B_KING = 255, B_ADVISOR = 254, B_BISHOP = 253, B_KNIGHT = 252, B_ROOK = 251, B_CANNON = 250, B_PAWN = 249
};

///局势编码（开局 / 中局 / 残局）
enum BoardState
{
    START_STATE = 0, MID_STATE = 1, END_STATE = 2
};

```

在棋盘表示类中，用上述用无符号单字节整数表示有无棋子，是什么棋子。用 255~249 表示黑方棋子，原因是这些数恰为 -1~-7 的 8 位二进制补码，与红方棋子一一对应，方便程序对棋子进行判断。

## 2、辅助宏定义：

```

///是否为红棋
#define IsRed(x) (!((x) & 0xf0) && (x))
///是否为黑棋
#define IsBlack(x) (!(~(x) & 0xf0))
///是否为红棋或无子（用于黑方判断可行着法）
#define IsRedOrNone(x) (!((x) & 0xf0))
///是否为黑棋或无子（用于红方判断可行着法）
#define IsBlackOrNone(x) (!(~(x) & 0xf0) || !(x))
///判断是红棋还是黑棋
/*! 注意这条只有在有棋子时能返回正确结果，NONE和OUT时返回结果无意义。
*/
#define GetColor(x) (!(~(x) & 0xf0))

```

这些宏用来判断棋盘上某一个位置有无棋子，是什么棋子。用位运算替代关系运算，用宏替代函数，可以提高执行效率。

## 3、棋盘表示类：

```

class Board
{
public:
    /*-----成员变量-----*/
    ///这一局面轮到谁走棋了？
    PLAYER player;
    ///走棋一方各种棋子可行的走法数目
    /*! 第一个元素无意义，后面的按棋子代码顺序排列，黑方也记绝对值1~7
    仅在执行MoveMaker::Make 后有意义
    */
    int numOfMove[8];
    ///当前局面
    /*! 使用16*16表示法，利于快速生成着法，同时防止棋子走出棋盘边界。
    参考：http://www.elephantbase.net/computer/eleeye_struct.htm
    */
    CHESSMAN position[256];
    ///当前局面的Zobrist 数
    HASHNUM zobrist;
    ///当前局面棋子个数，用于判断是开局 / 中局 / 残局
    int numOfChess;
};

```

```

    ///记录将（帅）位置
    BDPOINT kingpos[2];
    ///记录得到此局面的上一个着法
    /*! 只在执行Board::AddMovement 之后有意义，且不可由Board::DelMovement 回溯。
    */
    Movement lastmove;
    ///双方没有吃子的走棋步数（半回合数）
    int non_eat_steps;
    ///当前的回合数
    int bouts;
    /*-----成员函数-----*/
    ///默认构造函数——暂不使用
    Board();
    ///由FEN串生成棋盘的构造函数
    Board(const char * fen, int nMoveNum, long * lpdwCoordList);
    ///由FEN串生成棋盘
    /*!FEN串定义参见: http://www.elphantbase.net/protocol/cchess\_fen.htm
    */
    void FromFen(const char * fen, int nMoveNum, long * lpdwCoordList);
    ///添加一个着法后改变局面
    void AddMovement(const Movement & move);
    ///删除一个着法后改变局面
    void DelMovement(const Movement & move);
    ///验证一个字节着法字符串（如“h0i2”）对于本局面是否可行
    /*! 只验证起止点棋子合法性，不验证棋规合法性。
    坐标格式参见: http://www.elphantbase.net/protocol/cchess\_move.htm。
    */
    Bool CanGo(const char * str);
private:
    /*-----私有函数-----*/
    ///清空棋盘
    void ClearBoard();
};

```

棋盘表示类中只有“当面局面”和“这一局面轮到谁走棋了？”是必要信息，别的数据成员全都是为了方便其它模块的快速计算而设置。比如记录将（帅）位置可以在着法生成模块中快速得知某一着法是否造成将帅照面；记录当前局面的 Zobrist 数（即棋盘的哈希表示）用以在开局库中快速检索这一局面。

#### 4、着法表示类：

```

class Movement
{
public:
    /*-----成员变量-----*/
    ///起始位置（用16*16表示法）
    BDPOINT from;
    ///落子位置（用16*16表示法）
    BDPOINT to;
    ///被吃的子（用enum Chessman 表示）
    CHESSMAN ate;
    ///是否构成将军
    Bool check;
    /*-----成员函数-----*/
    ///默认构造函数——暂不使用
    Movement();
    ///使用已知信息的构造函数
    Movement(BDPOINT f, BDPOINT t, CHESSMAN a, Bool c);
    ///把着法转换成界面接收的字符串
    char * Move2Str(char * str);
    ///重载赋值运算符
    Movement & operator = (const Movement & obj);
};

```

着法表示记录着诸如“马五进三”这样的棋子相对位置变化信息。其中有必要记录被吃的子，因为在博弈树搜索中涉及着法回溯，必须使一个局面能够完整地还原回上一局面。

#### 5、置换表项目类：

```
///置换表项目类型
enum EntryType
{
    EXACT, L_BOUND, U_BOUND
};

class TTItem
{
public:
    ///该局面的哈希数
    /*!尽管用哈希数的后位作为存储下标，但为防止冲突情况，仍应记录哈希数并检验之。*/
    HASHNUM zobrist;
    ///该局面所在的博弈树最大搜索层数
    int max_depth;
    ///该局面所在的博弈树深度
    int depth;
    ///置换表项目类型
    EntryType entry;
    ///局面评估值
    int value;
};
```

对于博弈树中的重复局面（如先进左马后进右马与先进右马后进左马形成同样的“屏风马”局面），可以用置换表技术避免重复评估，直接利用第一次计算的结果代入之后出现的重复节点，节约运算时间。

## 二、模块设计说明

#### 1、主控模块：

```
int main(int argc, char *argv[])
{
    ///UCCI指令类型
    UcciCommEnum uce;
    ///UCCI指令内容
    UcciCommStruct ucs;
    ///当前局面
    Board curboard;

    getpath();
    Hashnum::InitHash();
    OpenBook::InitBook();

    if (BootLine() == UCCI_COMM_UCCI)
    {
        printf("id name BitStronger\n");
        fflush(stdout);
        printf("id copyright 2006 B.I.T.\n");
        fflush(stdout);
        printf("id author BitStronger Authors\n");
        fflush(stdout);
        printf("id user Our Users\n");
        fflush(stdout);
        printf("ucci ok\n");
        fflush(stdout);
    }
```

```

// 以下是接收指令和提供对策的循环体
uce = UCCI_COMM_NONE;
while (uce != UCCI_COMM_QUIT)
{
    uce = IdleLine(ucs, TRUE);
    switch (uce)
    {
        case UCCI_COMM_ISREADY:
            printf("readyok\n");
            fflush(stdout);
            break;
        case UCCI_COMM_STOP:
            printf("nobestmove\n");
            fflush(stdout);
            break;
        case UCCI_COMM_POSITION:
            curboard.FromFen(ucs.Position.szFenStr, ucs.Position.nMoveNum,
ucs.Position.lpdwCoordList);
            break;
        case UCCI_COMM_BANMOVES:
            //Nothing
            break;
        case UCCI_COMM_SETOPTION:
            //Nothing
            break;
        case UCCI_COMM_GO:
        case UCCI_COMM_GOPONDER:
            Searcher::GetBestmove(curboard, curboard.player);
            break;
    }
}
printf("bye\n");
fflush(stdout);

OpenBook::ClearBook();

return 0;
}

```

主控模块维护着一个全局唯一的当前局面实例。它的核心是消息循环处理机制，通过调用 UCCI 通讯模块实时地监听界面程序的各种指令，依指令要求调用引擎各个模块协调工作，然后将引擎的反馈输出给界面程序。

## 2、UCCI 通讯模块：

我们直接使用 UCCI 开发人员提供的通用通讯模块（遵循 GPL 许可），具体实现和功能说明参见：[http://www.elephantbase.net/protocol/cchess\\_ucci.htm](http://www.elephantbase.net/protocol/cchess_ucci.htm)。

## 3、着法生成模块：

```

///各种棋子可行着法数组这里从略

class MoveMaker
{
public:
    ///从当前局面得到所有可行着法，返回可行的着法种数
    static int Make(Board & cur, PLAYER who, Movement move[]);
private:
    ///对MoveMaker::Make 得出可行的着法再次判断合理性，排除主动送将和不会应将的着法
    /*!同时判断有无将军的情况，有则置check 为TRUE。<-暂未做这个判断!

```

```

*/
static Bool PreAdd(Board & cur, PLAYER who, BDPOINT from, BDPOINT to, Bool & check);
};

```

对不同的子，用不同的方法尽可能快速地找到所有可行的着法。比如士（仕）的可行位置有限，直接枚举；马可以用相对位置模板匹配。同时直接排除主动送将和不会应将的着法。

#### 4、博弈树搜索模块：

```

///搜索层数
static int searchDepth;

///可行着法集合
static Movement allmoves[MAX_SEARCH_DEPTH + 1][SEARCH_WIDTHH];
///静态搜索的可行着法集合
static Movement qsmoves[MAX_QUIES_DEPTH + 1][SEARCH_WIDTHH];

///当前最佳着法
static Movement best;
///上次搜索最佳着法
static Movement lastbest;
///当前次佳着法（保存前三位，避免循环局面）
static Movement better;
///上次搜索次佳着法（保存前三位，避免循环局面）
static Movement lastbetter;
///当前第三佳着法（保存前三位，避免循环局面）
static Movement good;
///上次搜索第三佳着法（保存前三位，避免循环局面）
static Movement lastgood;

///搜索开始时间
static time_t startTime;

///历史着法记录
/*! 记录最后四着，也就是可能出现的两个循环。
   因为走棋出现循环反复三次者要依棋规判负或判和。
   初始值置为不可能的着法，避免对开局的四着特殊处理。
*/
static long history[2][4] = {{0, 1, 2, 3}, {0, 1, 2, 3}};

///哈希树历史局面记录
static HASHNUM hashLink[MAX_SEARCH_DEPTH];

///静态搜索的哈希树历史局面记录
static HASHNUM qshashLink[MAX_QUIES_DEPTH];

///得到一个最终局面的路径
static Movement trace[MAX_SEARCH_DEPTH];
///静态搜索得到一个最终局面的路径
static Movement qstrace[MAX_QUIES_DEPTH];

class Searcher
{
public:
    ///得到一个最佳着法，向界面输出
    static void GetBestmove(Board & cur, PLAYER who);
    ///判断棋局是否决出胜负
    inline static Bool GameOver(Board & cur);
private:
    ///用PVS搜索查找最佳着法
    static int PVSearch(Board & cur, int depth, PLAYER who, int alpha, int beta);
    ///静态搜索（须与其它搜索配合使用，在其它搜索的叶子节点处调用）
    static int QuiesSearch(Board & cur, int depth, PLAYER who, int alpha, int beta);
};

```

在试验和调试过程中，可以编写多种不同的搜索算法，在 GetBestmove 函数中分别调用即可。

#### 5、局面评估模块：

```
///各类评估表这里从略
class Evaluation
{
public:
    ///评估当前局面价值
    static int Eval(Board & cur, PLAYER who);
private:
    ///棋子价值评估与子力位置评估
    static int ValueAndPosition(Board & cur, PLAYER who);
    ///棋子灵活性评估
    static int Agility(Board & cur, PLAYER who);
    ///棋子威胁、保护评估
    static int AttackAndProtect(Board & cur, PLAYER who);
    ///棋子配合度评估
    static int Cooperation(Board & cur, PLAYER who);
    ///得到棋子的威胁、保护评估价值
    inline static int GetAnPScore(BDPOINT point, CHESSMAN chess);
};
```

外部程序只调用 Eval 函数，Eval 函数内部调用各种子评估函数，加权求和返回给外部程序。

#### 6、开局库模块：

##### (1) 哈希表处理模块：

```
///Zobrist 数组
static HASHNUM hash[14][256];

class Hashnum
{
public:
    ///初始化Zobrist 数组和置换表
    static void InitHash();
    ///清除置换表
    static void ClearHash();
    ///得到某个棋子在某个位置对应的Zobrist 数
    static HASHNUM GetHash(CHESSMAN chess, BDPOINT point);
    ///在置换表中查找已存项目
    static int SearchTT(HASHNUM zobrist, int max_depth, int depth, PLAYER who, int alpha, int beta);
    ///将项目保存到置换表
    static void SaveTT(HASHNUM zobrist, int max_depth, int depth, PLAYER who, EntryType entry, int value);
};
```

采用博弈算法中经典的 Zobrist 键值作为哈希数表示一个局面，即对每个棋子位于每个位置的情况用一个 64 位数（8 个字节）表示，一个棋盘上所有棋子的 Zobrist 键值之异或可以表示一种局面。（编写小程序随机生成一组 64 位数，存在一个文件中，在 InitHash 函数中读取。理论表明这种异或后的随机数冲突概率极低，不用考虑冲突处理）

##### (2) 开局库检索模块：

```

///开局库条目结构
typedef struct BookItem BookItem;

///内存中记录开局库的向量
static vector<BookItem> * books;

///开局库条目结构
/*!
    开局库条目记录一个局面对应的多个着法，用long型存储起点到终点的个字节（如“h0i2”）。
*/
struct BookItem
{
    ///局面的Zobrist 数
    HASHNUM zobrist;
    ///这一局面轮到谁走棋了？
    PLAYER player;
    ///该局面对应的多个着法
    vector<long> * tricks;
};

class OpenBook
{
public:
    ///初始化开局库
    static void InitBook();
    ///清除开局库
    static void ClearBook();
    ///在开局库中搜索某一局面，返回对应着法数目，随机选择一个着法写入str
    static int Search(const Board & cur, char * str);
};

```

程序在开局或一些典型的中局时，先不进行博弈树搜索，而是在开局库中检索典型的应对着法。这样使避免了博弈树搜索方法在开局时因为局部很小的利益而忽视全局的发展，走出贻笑大方的局面的情况。便于我方快速出动子力，占据有利位置。开局库的棋谱数据源自《象棋百科全书》网站上收录的 8000 多局对局，使用该网站提供的小程序将棋谱中的经典开局提取，然后使用自己编写的小程序将上面总结出来的文本格式的开局信息转为我们程序中专用的二进制格式开局库，使用 Zobrist 键值加速检索。

#### 参考文献：

- [1] 王小春，PC 游戏编程（人机博弈），重庆大学出版社，2002
- [2] 黄晨，解剖大象的眼睛——中国象棋程序设计探索，象棋百科全书网站（<http://www.elphantbase.net>），2005

---

北京理工大学“理治棋壮”中国象棋计算机博弈引擎开发小组版权所有（2006.7~2006.8）

项目主管：	林 健（北京理工大学计算机科学技术学院）
指导教师：	黄 鸿（北京理工大学信息科学技术学院自动控制系系统工程研究所）
技术顾问：	赵陈翔（北京理工大学软件学院）
开发人员：	林 健（北京理工大学计算机科学技术学院）
	高 然（北京理工大学计算机科学技术学院）
	应张彬（北京理工大学软件学院）
	武 斌（北京理工大学软件学院）
联系方式：	<a href="mailto:lj@linjian.cn">lj@linjian.cn</a> （林健）
	<a href="mailto:honghuang@bit.edu.cn">honghuang@bit.edu.cn</a> （黄鸿）