电脑象棋循序渐进

象棋百科全书网 (webmaster@xqbase.com)　2008年4月

本连载将分六个阶段来阐述一个电脑象棋程序从无到有、从弱智到聪明的过程，对应“象棋小巫师”示范程序的各个版本：

0.1版：介绍制作象棋图形界面的方法；

0.2版：介绍中国象棋规则的实现；

0.3版：介绍以Alpha-Beta搜索为基础的一些计算机博弈基本智能算法；

0.4版：介绍Zobrist校验码、重复局面判断以及消除水平线效应的各种技术，进一步提高程序的智能；

0.5版：介绍置换表技术和走法顺序优化技巧，使一个象棋程序的技术架构趋于完整；

0.6版：介绍克服搜索不稳定性的方法、开局库、走子随机性等锦上添花的技术。

本连载花费的文字不多，关键的技术只是点到为止，而在示范程序里则有详尽的注释。关于一些技术细节，可参阅象棋百科全书网计算机博弈专栏关于国际象棋程序设计的系列译文：

http://www.xqbase.com/computer.htm

(一) 从图形界面做起

与本文配套的示范程序是“象棋小巫师”0.1版，程序清单是：

(1) XQWL01.CPP——C++源程序；

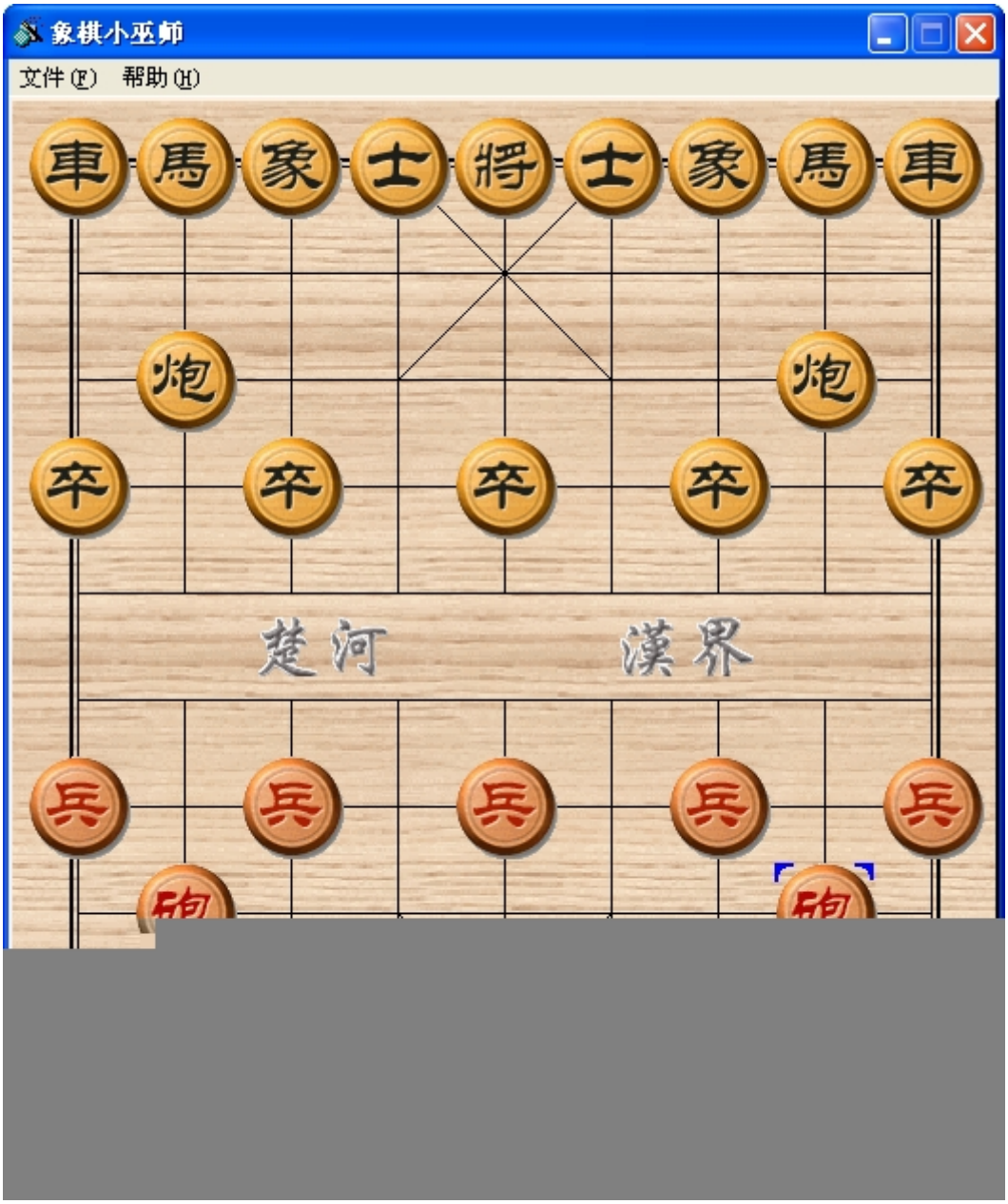
(2) XQWLIGHT.RC——资源描述文件；

(3) RESOURCE.H——资源符号定义文件；

(4) RES目录——图标、图片、声音等资源。

为了使更多的程序设计师对这个话题感兴趣，我们选用了最常用的程序开发工具——Microsoft Visual C++，它适合编写Windows下的任何应用程序。为了让程序尽可能简洁，我们不使用任何第三方的控件和库，取而代之的是大量的Win32 API函数。

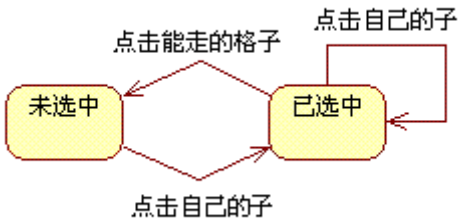
象棋程序是让人跟电脑下棋的，所以图形界面必不可少。我们选取了开源程序《象棋巫师》中的素材——一张520x576大小的棋盘图片和几张56x56大小的棋子图片，拼凑在一起便可组成一副不太难看的棋具。



仅仅把棋盘和棋子画到窗口上是不够的，图形界面的另一个作用是让用户走棋。走棋的操作应该是非常简单的——用鼠标点击一枚棋子，该棋子就被选中(上面有个标记)，用户想让这枚棋子走到哪个位置，就用鼠标点击哪个位置。因此，操作过程可分为两状态：

A. 让用户选子的状态：只能点击自己的子；

B. 让用户落子的状态：如果点击自己的子，那么被选中的子改变了；如果点击对方的子或空格处，那么可以走这步棋。



这样，鼠标点击事件(WM_LBUTTONDOWN消息)的处理过程就可以写成：

```
if (点击自己的子) {
    把点击的子选中; // 进入状态B(已选中)
```

```
} else if (已经有子选中) {
    可以走这步棋(刚才选中位置->现在点击的位置); // 进入状态A(未选中)
}
```

那么，如何把一枚棋子画在棋盘上呢？实际上只要用BitBlt和TransparentBlt两个函数就够了，过程如下：

(1) 把原来位置的棋子用棋盘图案覆盖掉(用BitBlt函数)；
(2) 在该位置贴上棋子的图案，由于棋子图案是透明的，所以要用TransparentBlt函数；
(3) 如果这枚棋子是选中的，那么再在该位置贴上选中的图案(再用一次TransparentBlt函数)。

在象棋小巫师中，所有在棋盘上的棋子都放在数组 ucpcSquares[256] 中(长度256的好处将在后面介绍)，数组指标代表格子的编号，匈牙利标记 uc 表示每个元素占用一个字节，pc 表示棋子标识。棋子标识的含义如下：

A. 0表示空格(没有棋子)；
B. 8~14依次表示红方的帅、仕、相、马、车、炮和兵；
C. 16~22依次表示黑方的将、士、象、马、车、炮和卒。

这样做的好处是判断棋子的颜色非常简单——(pc & 8) != 0 表示红方的棋子，(pc & 16) != 0 表示黑方的棋子。

在象棋小巫师中，选中的棋子用变量 sqSelected 表示，sq 代表格子的编号。判断棋子 ucpcSquares[sq] 是否被选中，只需要判断 sq 与 sqSelected 是否相等即可。sqSelected == 0 表示没有棋子被选中。

在象棋小巫师中，一个走法只用一个数子表示，即 mv = sqSrc + sqDst * 256，mv 代表走法，mv % 256 就是起始格子的编号，mv / 256 就是目标格子的编号。走完一步棋后，通常会把该走法赋值给变量 mvLast，并把 mvLast % 256 和 mvLast / 256 这两个格子都做上标记，这样就能清晰地看到用户或电脑刚才走的一步棋了。

- 上一篇
- 下一篇 电脑象棋循序渐进(二)：掌握象棋规则
- 返　回 象棋百科全书——计算机博弈

电脑象棋循序渐进

象棋百科全书网 (webmaster@xqbase.com)　2008年4月

(二) 掌握象棋规则

与本文配套的示范程序是“象棋小巫师”0.2版，程序清单是：
(1) XQWL02.CPP——C++源程序；
(2) XQWLIGHT.RC——资源描述文件；
(3) RESOURCE.H——资源符号定义文件；
(4) RES目录——图标、图片、声音等资源。

在阅读本章前，建议读者先阅读象棋百科全书网计算机博弈专栏的以下几篇译文：
(1) 国际象棋程序设计(一)：引言(François Dominic Laramée)；
(2) 国际象棋程序设计(二)：数据结构(François Dominic Laramée)；
(3) 国际象棋程序设计(三)：着法的产生(François Dominic Laramée)；
(4) 数据结构——简介(Bruce Moreland)；
(5) 数据结构——0x88着法产生方法(Bruce Moreland)。

2.1 走法生成器

走法生成器是象棋程序中的一个重要组成部分，它可以解决几乎所有象棋规则的问题。
假设我们的棋盘使用9x10的数组，按照常规的做法，找到一个马的所有走法，这将是一件非常痛苦的事：

```
// 判断马的下面一格有没有子
int yDst = ySrc + 2;
if (yDst <= Y_BOTTOM && ucpcSquares[xSrc][ySrc + 1] == 0) {
  int xDst = xSrc + 1;
  if (xDst <= X_RIGHT && !SELF_PIECE(ucpcSquares[xDst][yDst])) {
    ADD_MOVE(xSrc, ySrc, xDest, yDest);
  }
  xDst = xSrc - 1;
  if (xDst >= X_LEFT && !SELF_PIECE(ucpcSquares[xDst][yDst])) {
    ADD_MOVE(xSrc, ySrc, xDest, yDest);
  }
}
// 判断马的上面一格有没有子
......
```

不仅代码数量庞大，运行速度缓慢，而且一不小心就容易写错。
好在我们的棋盘是一个大小为16x16的二维数组，只不过写在程序里的是 ucpcSquares[256] 而已。

00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
10	11	12	13	14	15	16	17	18	19	1a	1b	1c	1d	1e	1f
20	21	22	23	24	25	26	27	28	29	2a	2b	2c	2d	2e	2f

30	31	32	33	34	35	36	37	38	39	3a	3b	3c	3d	3e	3f
40	41	42	43	44	45	46	47	48	49	4a	4b	4c	4d	4e	4f
50	51	52	53	54	55	56	57	58	59	5a	5b	5c	5d	5e	5f
60	61	62	63	64	65	66	67	68	69	6a	6b	6c	6d	6e	6f
70	71	72	73	74	75	76	77	78	79	7a	7b	7c	7d	7e	7f
80	81	82	83	84	85	86	87	88	89	8a	8b	8c	8d	8e	8f
90	91	92	93	94	95	96	97	98	99	9a	9b	9c	9d	9e	9f
a0	a1	a2	a3	a4	a5	a6	a7	a8	a9	aa	ab	ac	ad	ae	af
b0	b1	b2	b3	b4	b5	b6	b7	b8	b9	ba	bb	bc	bd	be	bf
c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	ca	cb	cc	cd	ce	cf
d0	d1	d2	d3	d4	d5	d6	d7	d8	d9	da	db	dc	dd	de	df
e0	e1	e2	e3	e4	e5	e6	e7	e8	e9	ea	eb	ec	ed	ee	ef
f0	f1	f2	f3	f4	f5	f6	f7	f8	f9	fa	fb	fc	fd	fe	ff

上表就是9x10的象棋棋盘在16x16的数组中的位置，我们将在这个棋盘上演绎马是如何走棋的。

首先，我们预置一个常量数组 ccInBoard[256]，表示哪些格子在棋盘外(紫色格子，填0)，哪些格子在棋盘内(浅色格子，填1)，所以就没有必要使用 x >= X_LEFT && x <= X_RIGHT && y >= Y_TOP && y <= Y_BOTTOM 之类的语句了，取而代之的是 ccInBoard[sq] != 0。

其次，一维数组的好就是上下左右关系非常简明——上面一格是 sq - 16，下面一格是 sq + 16，左面一格是 sq - 1，右面一格是 sq + 1。马可以跳的点只有8个，终点相对起点的偏移值是固定的：

```
const char ccKnightDelta[4][2] = {{-33, -31}, {-18, 14}, {-14, 18}, {31, 33}};
```

而对应的马腿的偏移值是：

```
const char ccKingDelta[4] = {-16, -1, 1, 16};
```

这个数组之所以命名为 ccKingDelta，是因为它也是帅(将)的偏移值。

这样，找到一个马的所有走法就容易很多了。首先判断某个方向上的马腿是否有子，然后判断该方向上的两个走法是否能走：

```
for (i = 0; i < 4; i ++) {
  sqPin = sqSrc + ccKingDelta[i];
  if (IN_BOARD(sqPin) && ucpcSquares[sqPin] == 0) {
    for (j = 0; j < 2; j ++) {
      sqDst = sqSrc + ccKnightDelta[i][j];
      if (IN_BOARD(sqDst) && !SELF_PIECE(ucpcSquares[sqDst])) {
        ADD_MOVE(sqSrc, sqDst);
      }
    }
  }
}
```

用类似的办法就可以产生其他棋子的所有走法。

2.2 判断走法是否符合规则

尽管我们已经使用了一些炫技，让走法生成器尽可能地小巧，但它仍然是象棋程序中最耗费时间的运算模块。有时候走法生成器真是大材小用了，比如用户点击鼠标走一步棋的时候，判断这步棋是否符合走法规则，就有几种不同的考虑：

A. 用走法生成器产生全部走法，看看这些走法中有没有用户刚才走出的那步棋，如果没有就说明用户在乱走；

B. 前一种做法中，大部分工作都是白费的，因为用户只是走了一个棋子，走法生成器没必要生成其他棋子的走法；

C. 用户只走了一步棋，而走法生成器会生成一个棋子的所有走法，是不是太浪费了呢？

判断一个走法是否合理，有更简单的方法。依然以马为例，假设用户的鼠标动作肯定在棋盘内的，那么判断过程如下：

(1) 马是否走了马步，即位移是否符合 ccKinghtDelta 中的值；

(2) 根据马步，找到对应的马腿位置，判断马腿的格子上是否有棋子。

在象棋小巫师中，我们用了一个 KNIGHT_PIN(sqSrc, sqDst) 的函数来获取马腿的位置(如果函数返回 sqSrc，则说明不是马步)。这样，判断马的某个走法是否符合规则，只需要很简单的两句话：

```
sqPin = KNIGHT_PIN(sqSrc, sqDst);
return sqPin != sqSrc && ucpcSquares[sqPin] == 0;
```

2.3 判断将军

到现在为止，我们剩下一件事没有做了，那就是判断胜负。中国象棋的胜负标准就是帅(将)有没有被将死或困毙，我们的做法很简单——生成所有走法，如果走任意一步都会被将军，那么该局面就是将死或困毙的局面，棋局到此结束。

那么如何来判断是否被将军呢？我们有两种做法：

A. 让对方生成全部走法，看看其中有没有走法可以吃掉自己的帅(将)；

B. 按照判断走法是否符合规则的思路，采用更简单的做法。

第一种做法没有什么不对的，但电脑象棋程序每秒种需要分析上万个局面，对每个局面都去生成全部走法显然太花时间了，所以我们要尝试第二种做法。其实判断帅(将)是否被将军的过程并不复杂：

(1) 假设帅(将)是车，判断它是否能吃到对方的车和将(帅)(中国象棋中有将帅不能对脸的规则)；

(2) 假设帅(将)是炮，判断它是否能吃到对方的炮；

(3) 假设帅(将)是马，判断它是否能吃到对方的马，需要注意的是，帅(将)的马腿用的数组是 ccAdvisorDelta，而不是 ccKingDelta；

(4) 假设帅(将)是过河的兵(卒)，判断它是否能吃到对方的卒(兵)。

这样，一个复杂的走法生成过程(方案A)就被简化成几个简单的走法判断过程(方案B)。

- 上一篇 电脑象棋循序渐进(一)：从图形界面做起
- 下一篇 电脑象棋循序渐进(三)：最初级的人工智能
- 返　回 象棋百科全书——计算机博弈

电脑象棋循序渐进

象棋百科全书网 (webmaster@xqbase.com)　2008年4月

(三) 最初级的人工智能

与本文配套的示范程序是“象棋小巫师”0.3版，程序清单是：
(1) XQWL03.CPP——C++源程序；
(2) XQWLIGHT.RC——资源描述文件；
(3) RESOURCE.H——资源符号定义文件；
(4) RES目录——图标、图片、声音等资源。

在阅读本章前，建议读者先阅读象棋百科全书网计算机博弈专栏的以下几篇译文：
(1) 国际象棋程序设计(四)：基本搜索方法(François Dominic Laramée)；
(2) 国际象棋程序设计(六)：局面评估函数(François Dominic Laramée)；
(3) 基本搜索方法——简介(一)(David Eppstein)；
(4) 基本搜索方法——简介(二)(David Eppstein)；
(5) 基本搜索方法——最小-最大搜索(Bruce Moreland)；
(6) 基本搜索方法——Alpha-Beta搜索(Bruce Moreland)；
(7) 基本搜索方法——迭代加深(Bruce Moreland)；
(8) 高级搜索方法——简介(二)(David Eppstein)；
(9) 其他策略——胜利局面(David Eppstein)；
(10) 局面评估函数——简介(一)(David Eppstein)。

3.1 局面评价函数

根据国际象棋程序的经验，局面评价函数中最关键的因素是子力价值(后9车5象马3兵1)。这个经验同样也适合于中国象棋，并且适当调整就可以得到更好的效果——子力价值是跟它的绝对位置相关的。最明显的例子是中国象棋中的兵(卒)，过河前我们给它很低的分数，过河后分数大涨，越接近九宫分数就越高，九宫中心甚至接近一个马或炮的值。

如此一来，每个兵种就都会有一个与绝对位置相关的价值数组，因此我们的程序里有一个常量数组 cucvlPiecePos[7][256]，它是从开源的象棋程序 ElephantEye 中照搬过来的。

现在要开始进行局面评价了，我们是不是应该这样做：

```
vlEvaluate = 0; // 相对于红方来说的局面评价值
for (sq = 0; sq < 256; sq ++) {
  pc = ucpcSquares[sq];
  if (IS_RED(pc)) {
    vlEvaluate += cucvlPiecePos[PIECE_TYPE(pc)][sq];
  } else if (IS_BLACK(pc)) {
    vlEvaluate -= cucvlPiecePos[PIECE_TYPE(pc)][SQUARE_FLIP(sq)];
  }
}
```

这样做太浪费时间了，因为根本没有必要每次都把棋盘扫描一遍。在我们的程序里，每走一步棋都会调用两到三次 AddPiece (放一枚棋子)和 DelPiece (取走一枚棋子)，可以趁这个机会更新 vlEvaluate(将上面代码中红色的部分放到 AddPiece 和 DelPiece 中)。

对于象棋小巫师来说，这样的局面评价函数已经足够好了。

3.2 Alpha-Beta搜索复杂吗?

我们可以很容易地写出一个Alpha-Beta搜索函数：

```
int AlphaBeta(int vlAlpha, int vlBeta, int nDepth) {
  if (nDepth == 0) {
    return 局面评价函数;
  }
  生成全部走法;
  排序全部走法;
  for (每个生成的走法) {
    走这个走法;
    int vl = -AlphaBeta(-vlBeta, -vlAlpha, nDepth - 1);
    撤消这个走法;
    if (vl >= vlBeta) {
      return vlBeta;
    }
    if (vl > vlAlpha) {
      vlAlpha = vl;
    }
  }
  return vlAlpha;
}
```

但是，这样的程序根本走不出棋来，因为它返回的是一个分数而不是一个走法。另外，我们还发现几个问题：

(1) 排序的依据是什么?
(2) 是不是每个生成的走法都可以走?
(3) 如果什么走法都走不出来，那么返回vlAlpha合理吗?

针对以上几个问题，我们对程序做如下改进：
(0) 如果函数在根节点处被调用，就把最佳走法作为电脑要走的棋;
(1) 国际象棋程序的经验证明，历史表是很好的走法排序依据;
(2) 由于我们的走法生成器并没有考虑自杀(被将军)的情况，因此走完一步后要检查是否被将军了，被将军时应立即退回来;
(3) 如果没有走出过任何走法，说明当前局面是杀棋或困毙局面，应该返回杀棋的分数。
下面是改进过的程序，改进的地方用红色标出：

```
int AlphaBeta(int vlAlpha, int vlBeta, int nDepth) {
  if (nDepth == 0) {
    return 局面评价函数;
  }
  生成全部走法;
  按历史表排序全部走法;
  for (每个生成的走法) {
    走这个走法;
    if (被将军) {
      撤消这个走法;
```

```
        } else {
            int vl = -AlphaBeta(-vlBeta, -vlAlpha, nDepth - 1);
            撤消这个走法;
            if (vl >= vlBeta) {
                将这个走法记录到历史表中;
                return vlBeta;
            }
            if (vl > vlAlpha) {
                设置最佳走法;
                vlAlpha = vl;
            }
        }
    }
    if (没有走过任何走法) {
        return 杀棋的分数;
    }
    将最佳走法记录到历史表中;
    if (根节点) {
        最佳走法就是电脑要走的棋;
    }
    return vlAlpha;
}
```

3.3 杀棋的分数

遇到将死或困毙的局面时，应该返回 nDistance - INFINITY，这样程序就能找到最短的搜索路线。nDistance 是当前节点距离根节点的步数，每走一个走法，nDistance 就增加1，每撤消一个走法，nDistance 就减少1。

如果程序中使用了置换表，这个问题将变得更加复杂，我们以后再讨论。

3.4 历史表

国际象棋程序的经验证明，历史表是很好的走法排序依据。那么，什么样的走法要记录到历史表中去呢？象棋小巫师选择了以下两类走法：

A. 产生Beta截断的走法；

B. 不能产生Beta截断，但它是所有PV走法(vl > vlAlpha)中最好的走法。

象棋小巫师的历史表是一个大小为65536的数组，正好能将走法的数值(mv)作为指标，因此根据走法取得历史表的值非常容易，即nHistoryTable[mv]。那么，一个走法记录到历史表，究竟该给 nHistoryTable 中的这个元素加多少分的值呢？我们仍旧沿用国际的经验——深度的平方。所以，更新历史表的代码非常简单：

```
nHistoryTable[mv] += nDepth * nDepth;
```

3.5 迭代加深和时间控制

迭代加深具有一石多鸟的功效，目前最明显的供效是充分发挥历史表的作用——浅一层搜索结束后，历史表中积累了大量非常宝贵的数据，这将大幅度减少深一层搜索的时间。

在迭代加深的基础上实现时间控制，这将是非常简单的：

```
for (i = 1; i < MAX_DEPTH; i ++) {
  AlphaBeta(-INFINITY, INFINITY, i);
  if (超过最短搜索时间) {
    break;
  }
}
```

当然，我们还可以加入其他结束迭代加深的条件，例如当程序算出了杀棋(分值接近INFINITY或-INFINITY)时，就没有必要进行更深的搜索了。

- 上一篇 电脑象棋循序渐进(二)：掌握象棋规则
- 下一篇 电脑象棋循序渐进(四)：稍微聪明些了
- 返　回 象棋百科全书——计算机博弈

电脑象棋循序渐进

象棋百科全书网 (webmaster@xqbase.com)　2008年4月

(四) 稍微聪明些了

与本文配套的示范程序是“象棋小巫师”0.4版，程序清单是：
(1) XQWL04.CPP——C++源程序；
(2) XQWLIGHT.RC——资源描述文件；
(3) RESOURCE.H——资源符号定义文件；
(4) RES目录——图标、图片、声音等资源。

在阅读本章前，建议读者先阅读象棋百科全书网计算机博弈专栏的以下几篇译文：
(1) 国际象棋程序设计(五)：高级搜索方法(François Dominic Laramée)；
(2) 数据结构——Zobrist键值(Bruce Moreland)；
(3) 基本搜索方法——简介(二)(David Eppstein)；
(4) 高级搜索方法——简介(一)(David Eppstein)；
(5) 高级搜索方法——静态搜索(Bruce Moreland)；
(6) 高级搜索方法——空着裁剪(Bruce Moreland)；
(7) 其他策略——重复检测(Bruce Moreland)。

我们的程序终于会走棋了，不过很多时候它很低能。由于水平线效应，任何变化都只搜索固定的深度。还有，有时它会长将。我们能做哪些改进呢？

4.1 克服水平线效应

克服水平线效应的方法有以下几种：
(1) 静态(Quiescence)搜索。进入静态搜索时，要考虑两种情况，一是不被将军的情况，首先尝试不走是否能够截断，然后搜索所有吃子的走法(可以按照MVV/LVA排序)；二是被将军的情况，这时就必须生成所有的走法了(可以按照历史表排序)。
(2) 空步(Null-Move)裁剪。空步裁剪的代码非常简单，但某些条件下并不适用，一是被将军的情况下，二是进入残局时(自己一方的子力总价值小于某个阈值)，三是不要连续做两次空步裁剪，否则会导致搜索的退化。
(3) 将军延伸。

4.2 检查重复局面

在象棋小巫师的前几个版本中，重复局面判断不是必须的，因为任何变化都只搜索固定的深度。但是静态搜索和将军延伸会带来一个问题——遇到“解将还将”的局面，搜索就会无止境地进行下去，直到程序崩溃。
有两个办法可以解决这个问题：
(1) 限制实际搜索深度(通过 nDistance 来限制)；
(2) 自动识别重复局面，遇到这样的局面就根据规则返回和棋或杀棋的分数。
前者实现起来非常简单，我们的程序也这样做了，但仍旧使程序做了很多无用的搜索。在这个版本中，我们重点把力气花在检查重复局面上了。
检查重复局面的办法很简单，每走一个走法就把当前局面的校验码记录下来，再看看前几个局面的校验码是否与当前值相等。当重复局面发生时，就要根据双方的将军情况来判定胜负——单方

面长将者判负(返回杀棋分数而不必要继续搜索了)，双长将或双方都存在非将走法则判和(返回和棋分数)。

象棋小巫师用了一个 RepStatus 函数来检查重复，如果局面存在重复，那么它的返回值将很有意思：

```
return 1 + (bPerpCheck ? 2 : 0) + (bOppPerpCheck ? 4 : 0);
```

起初bPerpCheck(本方长将)和bOppPerpCheck(对方长将)都设为TRUE，当一方存在非将走法时就改为FALSE，这样 RepStatus 的返回值有有这几种可能：

A. 返回0，表示没有重复局面；
B. 返回1，表示存在重复局面，但双方都无长将(判和)；
C. 返回3(=1+2)，表示存在重复局面，本方单方面长将(判本方负)；
D. 返回5(=1+4)，表示存在重复局面，对方单方面长将(判对方负)；
E. 返回7(=1+2+4)，表示存在重复局面，双方长将(判和)。

4.3 Zobrist校验码

我们把Zobrist值作为局面的校验码，好处在于计算迅速。除了检查重复局面外，校验码还有以下作用：

(1) 作为置换表(Hash表)的键值；
(2) 作为开局库的查找依据。

象棋小巫师生成的Zobrist校验码跟开源象棋程序 ElephantEye 是一致的(以空密钥的RC4密码流作为随机序列)，这样就可以使用 ElephantEye 的开局库了。Zobrist值总共96位，放在 dwKey、dwLock0 和 dwLock1 中，其中 dwKey 在检查重复局面时用，也作为置换表的键值，dwLock0 和 dwLock1 用作置换表的校验值，另外，dwLock1 还是查找开局库的依据(后面会提到)。

- 上一篇 电脑象棋循序渐进(三)：最初级的人工智能
- 下一篇 电脑象棋循序渐进(五)：质的飞跃
- 返　回 象棋百科全书——计算机博弈

电脑象棋循序渐进

象棋百科全书网 (webmaster@xqbase.com)　2008年4月

(五) 质的飞跃

与本文配套的示范程序是“象棋小巫师”0.5版，程序清单是：
(1) XQWL05.CPP——C++源程序；
(2) XQWLIGHT.RC——资源描述文件；
(3) RESOURCE.H——资源符号定义文件；
(4) RES目录——图标、图片、声音等资源。

在阅读本章前，建议读者先阅读象棋百科全书网计算机博弈专栏的以下几篇译文：
(1) 基本搜索方法——简介(三)(David Eppstein)；
(2) 基本搜索方法——置换表(Bruce Moreland)；
(3) 其他策略——胜利局面(Bruce Moreland)。

5.1 置换表

没有置换表，就称不上是完整的计算机博弈程序。

象棋小巫师的置换表非常简单，以局面的 Zobrist Key % HASH_SIZE 作为索引值。每个置换表项存储的内容无非就是：A. 深度，B. 标志，C. 分值，D. 最佳走法，E. Zobrist Lock 校验码。置换表的处理函数也很传统——一个 ProbeHash 和一个 RecordHash 就足够了。

先说 RecordHash，即便采用深度优先的替换策略，RecordHash 也非常简单，在判断深度后，将 Hash 表项中的每个值填上就是了。

再看看 ProbeHash 是如何利用置换表信息的：

(1) 检查局面所对应的置换表项，如果 Zobrist Lock 校验码匹配，那么我们就认为命中(Hit)了；

(2) 是否能直接利用置换表中的结果，取决于两个因素：A. 深度是否达到要求，B. 非PV节点还需要考虑边界。

第二种情况是最好的(完全利用)，ProbeHash 返回一个非 -MATE_VALUE 的值，这样就能不对该节点进行展开了。

如果仅仅符合第一种情况，那么该置换表项的信息仍旧是有意义的——它的最佳走法给了我们一定的启发(部分利用)。

5.2 杀棋分数调整

象棋小巫师从学会走棋开始，就已经考虑了杀棋分数。不过增加了置换表以后，这个分数要进行调整——置换表中的分值不能是距离根节点的杀棋分值，而是距离当前(置换表项)节点的分值。所以当分值接近 INFINITY 或 -INFINITY 时，ProbeHash 和 RecordHash 都要做细微的调整：

(1) 对于RecordHash：置换表项记录的杀棋步数 = 实际杀棋步数 - 置换表项距离根节点的步数；

(2) 对于ProbeHash：实际杀棋步数 = 置换表项记录的杀棋步数 + 置换表项距离根节点的步数。

5.3 杀手(Killer)走法

把这个术语取名为Killer真是有些奇怪，但我们还是沿用这个术语。

杀手走法就是兄弟节点中产生Beta截断的走法。根据国际象棋的经验，杀手走法产生截断的可能性极大，所以我们在中国象棋里吸取了这个经验。很显然，兄弟节点中的走法未必在当前节点下

能走，所以在尝试杀手走法以前先要对它进行走法合理性的判断。我们在0.2版中就写过 LegalMove 这个函数，这里它将大显身手。如果杀手走法确实产生截断了，那么后面耗时更多的 GenerateMove 就可以不用执行了。

如何保存和获取“兄弟节点中产生截断的走法”呢？我们可以把这个问题简单化——距离根节点步数(nDistance)同样多的节点，彼此都称为“兄弟”节点，换句话说，亲兄弟、堂表兄弟以及关系更疏远的兄弟都称为“兄弟”。

我们可以把距离根节点的步数(nDistance)作为索引值，构造一个杀手走法表。象棋小巫师的每个杀手走法表项存有两个杀手走法，走法一比走法二优先：存一个走法时，走法二被走法一替换，走法一被新走法替换；取走法时，先取走法一，后取走法二。

5.4 优化走法顺序

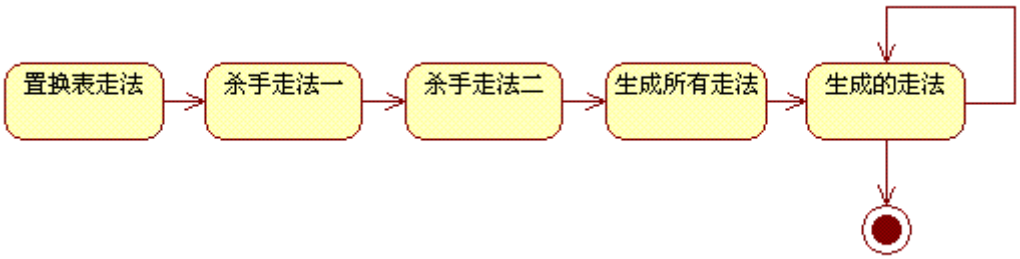
利用各种信息渠道(如置换表、杀手走法、历史表等)来优化走法顺序的手段称为“启发”。象棋小巫师0.5以前，我们只用历史表作启发，但从这个版本开始，我们采用了多种启发方式：

(1) 如果置换表中有过该局面的数据，但无法完全利用，那么多数情况下它是浅一层搜索中产生截断的走法，我们可以首先尝试它；

(2) 然后是两个杀手走法(如果其中某个杀手走法与置换表走法一样，那么可以跳过)；

(3) 然后生成全部走法，按历史表排序，再依次搜索(可以排除置换表走法和两个杀手走法)。

这样，我们就可以构造一个状态机，来描述走法顺序的若干阶段：



我们把状态机写在一个叫 Next 的函数中，那么 Alpha-Beta 的循环体就是：

```
..... // 初始化状态机
while ((mv = Next()) != 0) {
  MakeMove(mv);
  ..... // Alpha-Beta递归调用
  UndoMakeMove(mv);
  ..... // Alpha-Beta边界判断
}
```

在 Next 函数中，我们用了不带break的switch ... case结构：

```
switch (nPhase) {
case PHASE_HASH:
  nPhase = PHASE_KILLER_1;
  ..... // 如果有置换表走法，就可以返回，再次调用就直接跳到 PHASE_KILLER_1
  // 注意：这里没有break!
case PHASE_KILLER_1:
  nPhase = PHASE_KILLER_2;
  .....
}
```

这就是“基于置换表的启发式Alpha-Beta搜索”，目前顶尖的电脑(国际)象棋程序都逃脱不了这种架构，只不过它们在置换表和启发算法上更加优化而已。

- 上一篇 电脑象棋循序渐进(四)：稍微聪明些了
- 下一篇 电脑象棋循序渐进(六)：精益求精
- 返　回 象棋百科全书——计算机博弈

www.xqbase.com

电脑象棋循序渐进

象棋百科全书网 (webmaster@xqbase.com) 2008年4月

(六) 精益求精

与本文配套的示范程序是“象棋小巫师”0.6版，程序清单是：
(1) XQWL06.CPP——C++源程序；
(2) XQWLIGHT.RC——资源描述文件；
(3) RESOURCE.H——资源符号定义文件；
(4) RES目录——图标、图片、声音等资源；
(5) BOOK.DAT——开局库文件。

在阅读本章前，建议读者先阅读象棋百科全书网计算机博弈专栏的以下几篇译文：
(1) 高级搜索方法——主要变例搜索(Bruce Moreland)；
(2) 高级搜索方法——搜索的不稳定性(Bruce Moreland)。

尽管我们的程序在架构上已经接近完整，但细节上存在不少问题：
(1) 对于同一个局面，总是走固定的走法；
(2) 搜索算法是否能更优化一些(某些读者听说过PVS、Nega-Scout之类的算法)；
(3) 有些杀棋局面会走出莫名其妙的走法。
本章我们将把这些问题一一解决。

6.1 开局库

开局库几乎是每个象棋程序必备的部件，它的好处是：
(1) 即使再笨的程序，开局库能使得它们在开局阶段看上去不那么业余；
(2) 通过随机选择走法，让开局灵活多变，增加对弈的趣味性。
象棋小巫师使用开源象棋程序 ElephantEye 的开局库，开局库文件 BOOK.DAT 的结构是：

```
struct BookItem {
  DWORD dwLock;
  WORD wmv, wvl;
} BookTable[BOOK_SIZE];
```

其中，dwLock 记录了局面 Zobrist 校验码中的 dwLock1，wmv 是走法，wvl 是权重(随机选择走法的几率，仅当两个相同的 dwLock 有不同的 wmv 时，wvl 的值才有意义)。

搜索一个局面时，首先不做Alpha-Beta搜索，而是查找 BookTable 中有没有对应的项，有的话就直接返回一个走法。由于 ElephantEye 在制作开局库时是按照 dwLock 排序的，因此可以用二分查找。找到一项以后，把它前后 dwLock 相同的所有项都取出，从中随机选择一个 wmv。

ElephantEye 为了压缩开局库的容量，所有对称的局面只用一项，所以当一个局面在 BookTable 中找不到时，还应该试一下它的对称局面是否在 BookTable 中。

6.2 根节点的特殊处理

现在我们的程序一开局不会总是跳正马了，根据 ElephantEye 提供的开局库，它大部分时候走中炮，有时也走仙人指路(进兵)或飞相。可是当它脱离开局库时，仍然摆脱不了思维的单一性，例如我

们第一步走边兵(开局库中当然没有这个局面)，它仍旧只会跳同一边的正马。

一个解决办法是：在根节点处，让一个不是最好的走法也能在一定的几率取代前一个走法。我们的程序是这样写的：

```
if (vl > vlBest) {
  vlBest = vl;
  对vlBest作小范围的随机浮动;
}
```

我们把根节点的搜索函数单独分离，这样做有很多好处：
(1) 处理思考的随机性；
(2) 没有必要尝试 Beta 截断(根节点处 Beta 始终是 +INFINITY)；
(3) 省略了检查重复局面、获取置换表、空步裁剪等步骤。

6.3 PVS

很多计算机博弈的资料都介绍了PVS算法，但它只有当走法顺序充分优化时才能带来明显的好处，因此象棋小巫师直到最后一个版本才用了这种算法。

6.4 长将判负策略

由于单方面长将不变作负的规则，0.6以前的版本如果发生这种情况，想当然地给予-MATE_VALUE的值，再根据杀棋步数作调整。但是由于长将判负并不是对某个单纯局面的评分，而是跟路线有关的，所以使用置换表时就会产生非常严重的后果——某个局面的信息可能来自另一条不同的路线。

象棋小巫师的解决办法就是：获取置换表时把“利用长将判负策略搜索到的局面”过滤掉。为此这个版本中我们把长将判负的局面定为BAN_VALUE(MATE_VALUE - 100)，如果某个局面分值在WIN_VALUE(MATE_VALUE - 200)和BAN_VALUE之间，那么这个局面就是“利用长将判负策略搜索到的局面”。

我们仍旧把部分“利用长将判负策略搜索到的局面”记录到置换表，因为这些局面提供的最佳走法是有启发价值的。反过来说，如果“利用长将判负策略搜索到的局面”没有最佳走法，那么这种局面就没有必要记录到置换表了。

经过这种处理，我们的程序在杀棋阶段不再会走出莫名其妙的走法了，最后一个疑难杂症终于攻克了。

- 上一篇 电脑象棋循序渐进(五)：质的飞跃
- 下一篇
- 返 回 象棋百科全书——计算机博弈