DOI:10.16184/j.cnki.comprg.2013.09.017

Java 版井字棋的设计与实现

仇 宾

摘　要：井字棋是大家所熟知的一个小游戏，虽然简单，但其中包含了一些编程的基本技巧和基本算法，在 Eclipse 环境下用 Java 语言编写一个可以人人、人机对弈的井字棋游戏。

关键词：Java 语言；井字棋；游戏编程

1　引言

井字棋，即棋盘是一个井字，是一种在 3X3 格子上进行的连珠游戏，和五子棋比较类似，由于棋盘一般不画边框，格线排成井字而得名。游戏规则很简单，游戏双方一方为“X”，一方为“O”，哪方率先实现 3 子相连即为胜者。如图 1 所示。

图 1　正在进行中的井字棋游戏

现在来对井字棋游戏的代码实现做一个探讨，首先介绍人人对弈方式的实现过程，然后在此基础上介绍人机对弈井字棋的实现。

2　人人对弈

2.1　难点释疑

人人对弈实现起来较为简单，游戏双方交替在棋盘上落下棋子“X”或“O”即可，最大问题就是如何判定胜负。从棋盘可以看出，获胜（即任一方出现三连子）一共有 8 种情况：3 连横、3 连竖以及 2 个斜对角，如果我们给每个落子点从 0 到 8 编号，如图 2 所示。

那么，这 8 种获胜情况我们可以用一个二维数组来表示：

```
static final int[][] WIN_STATUS = {
        {0, 1, 2},
        {3, 4, 5},
        {6, 7, 8},
        {0, 3, 6},
        {1, 4, 7},
        {2, 5, 8},
        {0, 4, 8},
        {2, 4, 6}
};
```

图 2　棋盘落子点编号

这样，再定义一个一维数组，每走一步棋就对上面的二维数组进行遍历——从二维数组中依次取出 8 种情况放入一维数组，然后查看一维数组中的 3 个棋子是否相同，如果相同可以判定获胜。

2.2　设计与实现

第一步：写一个类继承自 JFrame，然后定义几个必要的变量和常量，如下：

```
public class Tic extends JFrame {
    JButton[] jb = new JButton[9];// 按钮数组构成棋盘的 8
//个落子点
    static final char empty = 32; // 代表空格
    static int clicknum = 0;      // 记录单击次数,决定是走 X
//还是走 O
    static final int INFINITY = 100;// 带标无穷值
    static final int WIN = +INFINITY;     // O 获胜
    static final int LOSE = -INFINITY;    // X 获胜
```

```
    static final int DRAW = 0; // 平局
    // 获胜棋盘状态
    static final int[][] WIN_STATUS = {
            {0, 1, 2},
            {3, 4, 5},
            {6, 7, 8},
            {0, 3, 6},
            {1, 4, 7},
            {2, 5, 8},
            {0, 4, 8},
            {2, 4, 6}
    };
}
```

第二步：构建棋盘。

在 Tic 类的构造方法中，JFrame 的布局方式设置为 GridLayout，然后每个格子里放置一个按钮即可，代码如下：

```
    public Tic(){
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setSize(400, 400);       //棋盘大小
        this.setLayout(new GridLayout(3,3));
        this.setTitle("井字棋");
        // 让窗口居中显示
        Dimension screen = Toolkit.getDefaultToolkit().
    getScreenSize();     // 获取屏幕尺寸封装到 screen 中
    this.setLocation((screen.height – this.getHeight())/2,
        (screen.width – this.getWidth())/2);  // 窗口居中
        // 把按钮加入窗体
        for(int i=0; i<9; i++){
            jb[i] = new JButton("");
            jb[i].setFont(new Font("Arial",Font.BOLD,30));
        jb[i].addActionListener(new JBClick());// 事件监听
            this.add(jb[i]);
        }
        this.setVisible(true);
    }
```

第三步：经过前两步，现在加上 main 方法，就可以执行显示棋盘了，代码如下：

```
    public static void main(String[] args) {
        new Tic();
JOptionPane.showMessageDialog (null,"O 代表玩家 1,X 代表玩家 2,玩家 1 先走","提示",JOptionPane.DEFAULT_OPTION);
    }
```

但是按钮还没有事件响应，现在单击按钮没有任何反应。下面给按钮添加事件响应。每次单击按钮需要完成如下任务：因为是两个玩家进行游戏，所以按钮要轮流显示“X”和“O”，表示游戏双方交替走棋。每次单击走棋后还要判定是否有获胜方，有则提示哪方获胜，游戏结束，无则继续走棋。代码如下：

```
    private class JBClick implements ActionListener{
        // 当单击按钮时
        public void actionPerformed(ActionEvent e) {
            for(int i=0; i<9; i++){
        if(e.getSource() == jb[i]){// 哪个按钮被单击
            if(++clicknum % 2 == 0){//偶数次是 X
                    jb[i].setText("X");
                }else{
                    jb[i].setText("O");//奇数次是 O
                }
        jb[i].setEnabled(false);//被单击过的按钮不可用
                }
            }
    int gamestate = gameState(jb);// 调方法获取棋盘状态
            // 输出棋局胜负
            switch (gamestate) {
            case WIN:
JOptionPane.showMessageDialog(null, "O 方获胜", "提示",
JOptionPane.DEFAULT_OPTION);
                break;
            case LOSE:
JOptionPane.showMessageDialog(null, "X 方获胜", "提示",
JOptionPane.DEFAULT_OPTION);
                break;
            case DRAW:
JOptionPane.showMessageDialog (null, "平局", "提示",
JOptionPane.DEFAULT_OPTION);
                break;
            }
            // 如果结束,则提示
if (gamestate == WIN || gamestate == LOSE || gamestate
== DRAW) {
                int over = JOptionPane
        .showConfirmDialog(null, "是否再来一局？", "提示",

JOptionPane.YES_NO_OPTION,JOptionPane.
QUESTION_MESSAGE);
                if (over == JOptionPane.YES_OPTION)
{
                    for (int i = 0; i < 9; i++) {
                        jb[i].setText(" ");
                        jb[i].setEnabled(true);
                    }
                } else {
                    System.exit(0); // 退出游戏
                }
            }
        }
    }
```

其中代码：int gamestate = gameState (jb) ;表示调用 gameState 方法来判定是否有获胜方。如何判定呢？首先，遍历整个棋盘，看棋盘是否已经满，如果未满，就直接返回，继续走棋，如果满了，则判断是否符合 8 种数组中的一种，符合则分出胜负，不符合则说明平局。代码如下：

```
public int  gameState(JButton[] jb){
        int result = 1; // 棋局状态初始值
        boolean isFull = true;        // 棋盘是否已满
        // 判断棋盘是否已满
        for (int pos = 0; pos < 9; pos++) {
            char chess = jb[pos].getText().charAt(0);
            if (empty == chess) {
                isFull = false;
                return result; // 未满则返回,继续走棋
            }
        }
        // 棋局已满,判定胜负
for(int[] status:WIN_STATUS){//遍历 8 中棋局获胜状态
            //得到某个获胜棋局状态的第一个索引的字符
            char chess = jb[status[0]].getText().charAt(0);
// 如果为空,说明此处未下棋子,跳出循环,找下一个状态
            if(chess==empty){
                continue;
            }
            int i ;
            //不为空,则看其余两子是否与其相同
            for(i=1; i<status.length; i++){
          if(jb[status[i]].getText().charAt(0)! =chess){
            break;      //不同则跳出循环
              }
            }
            if(i==status.length){  // 三子连线
                result = chess== 'O' ? WIN : LOSE;
                break;
            }
        }
        if (result ! = WIN & result ! = LOSE & isFull)
        result = DRAW;       //不输不赢且棋盘满则为平
        return result;
    }
```

调用该方法后，如果返回值 result 等于 1，说明棋局没有下满，继续走棋；等于 WIN，说明走“O”的一方获胜；等于 LOSE，说明“X”方获胜；等于 DRAW，则平局。

执行过程如图 3 所示。

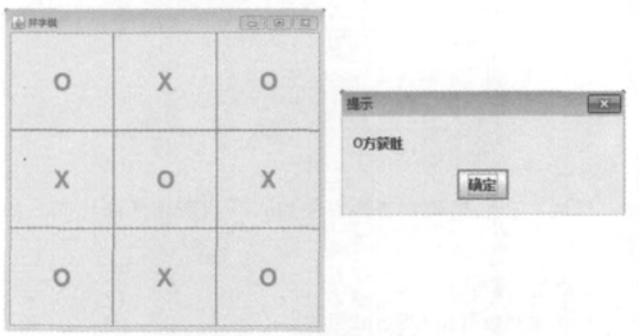

图 3　O 方获胜

2.3　编程技巧

尽管很简单的一个小程序，但是其中却包含一些编程技巧，比如使用一个二维数组来记录棋局获胜的情况，如果不用数组，则需要使用 8 个 if 语句来描述 8 种获胜情况。再比如程序中用到的 for each 语句，很轻松地做到了用一个一维数组去遍历二维数组。如果使用 for 语句，那么这个循环将会很难写。

3　人机对弈

3.1　难点释疑

人机对弈的难点在于当人走一步棋之后，计算机如何走下一步，即计算机如何找出最合适的位置去走棋。这就需要一定的算法，或者叫做计算机的 AI。对于井字棋、五子棋等两方较量的游戏来说，Minimax 算法（极小极大算法）是最基本也是最常用的。算法的原理不在这里解释了，直接看该算法在井字棋中的应用。

井字棋中，假设使用“X”的是人，使用“O”的是计算机。“X”方先走，设定 X 方的最大利益为正无穷（程序使用常量+INFINITY 表示），O 方的最大利益为负无穷（程序中使用–INFINITY 表示），即 X 方和 O 方走的每步棋都要力图使自己的利益最大化，而使对方的利益最小化。这样称 X 方为 MAX（因为它总是追求更大的值），O 方为 MIN（它总是追求更小的值），各自都为争取自己的最大获益而努力。现在举例说明，比如图 4 所示的棋局树。

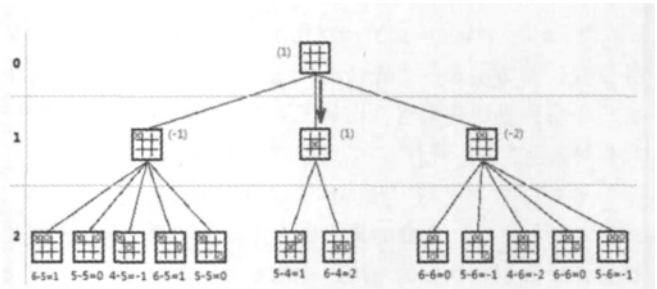

图 4　棋局形成的树

X 方先走，有 3 种选择，如图 4 中第二层所示。假设 X 方选择最左边的走法，那么 O 方接下来将有 5 种走法，O 方会选择最小化的走法，即值为–1 的走法，因为它的最大利益是负无穷；同理，X 方的另外两种走法会分别得到 O 方的最小值 1 和–2。这样，对于 X 方来说，3 种走法会导致 O 方最小化值分别为–1、1、–2，X 方的最佳策略则是选择其中最大的，即第二层中间的走法，因为它的最大利益是正无穷，这就是极小极大算法的体现——X 方的选择总是极大化，O 方的选择总是极小化。

对于其中那些值是如何计算的举例说明，比如对于第 3 层最左边的棋局，在这种状态下，如果把棋局空白处都填上 X，则 X 共有 6 中 3 连子情况，即获胜情况；如果把空白处都填上

O，则 O 共有 5 种 3 连子情况，所以结果是二者相减等于 1。

在具体走棋过程中，MAX 面对 MIN 最大获利中的最小值时，会选择其中最大的，比如图 4 第二层小括号内的值都是第三层中能使 MIN 最大获利的最小值，这时候 MAX 选择其中最大的，这对 MAX 最为有利，所以 MAX 方选择图 4 第二层中间的走法最好。同样道理，MIN 也会一样，选择对自己最有利的，即 MAX 有可能获得最大值。这时候，MIN 在走棋时会考虑 MAX 方占据哪个位置对 MAX 最有利，然后 MIN 把这个位置先占了。有点难理解，其实就是抢先把对对手有利的位置抢占了。

简单说，X 方或者 MAX 方的走棋是由人来控制的。对于 O 方或者 MIN 方，它走棋时要考虑哪个位置对 X 方最有利，然后把该位置占据，即 O 的最佳走棋就是 X 的最佳走棋。所以 O 在走棋之前，先站在 X 的角度寻找最佳走棋位置。后文中 minimax 方法就是站在 X 角度来考虑极小极大算法，找到 X 的最佳走棋位置，然后由 O 方来占据该位置。

3.2 极小极大算法

整个算法包括如下几个部分：

首先要有一个评估方法 gameState，对每走一步棋后的棋局进行评估，估值为 WIN 常量说明 X 方，即 MAX 方获胜；估值为 LOSE 则 O 方，即 MIN 方获胜；估值 DRAW 为平局；估值为 INPROGRESS，说明棋未走完；估值为 DOUBLE_LINK，说明棋局中有两连子情况。

然后用一个 minimax 方法寻找在当前棋局状态下 X 方的最佳位置， X 方的最佳位置就是当 X 走该位置后，O 方所有走法中最小值里的最大值，比如图 4 中第二层 X 的位置选择。当找到该位置后，由 O 方来抢先占据该位置。

最后用两个递归方法 MIN 和 MAX 来遍历所有的棋局。MIN 方法负责找出 O 方的最小值，比如图 4 第二层最左边的棋局会导致 5 中 O 方的走法，MIN 方法就是找出这 5 种走法中的最小值。同理，MAX 方法负责找出 X 方的最大值，比如图 4 第二层 3 种棋局中的中间棋局。

3.3 代码实现

在刚才那个人人对弈井字棋的基础上，进行一些修改来实现人机对弈。首先单击事件加入调用方法让计算机走棋的代码，如下：

```
// 按钮的监听事件
private class JBClick implements ActionListener {
    // 当单击按钮时
    public void actionPerformed(ActionEvent e) {
        for (int i = 0; i < 9; i++) {
            if (e.getSource() == jb[i]) {
                jb[i].setText("X"); // 被单击的按钮走“X”
                jb[i].setEnabled(false); //置为不可用
                }
            int gamestate = gameState(jb); // 获取棋盘状态
                // 如果棋局未结束,则计算机走下一步
if (! (gamestate == WIN || gamestate == LOSE || gamestate == DRAW)) {
                int nextpos = getNextMove(jb); // 获取下一步//走棋位置
                jb[nextpos].setText("O"); // 走棋“O”
                jb[nextpos].setEnabled(false);
    gamestate = gameState(jb); // 获取最新的棋盘状态
                }
                // 输出棋局胜负
                switch (gamestate) {
                case WIN:
 JOptionPane.showMessageDialog(null, "X 方获胜", "提示", JOptionPane.DEFAULT_OPTION);
                  break;
                case LOSE:
                    JOptionPane.showMessageDialog (null, "O 方获胜", "提示",JOptionPane.DEFAULT_OPTION);
                    break;
                case DRAW:
     JOptionPane.showMessageDialog(null, "平局", "提示", JOptionPane.DEFAULT_OPTION);
                    break;
                }

                // 如果结束,则提示
    if (gamestate == WIN || gamestate == LOSE || gamestate == DRAW) {
                int over = JOptionPane
    .showConfirmDialog(null, "是否再来一局？", "提示",
     JOptionPane.YES_NO_OPTION,
          JOptionPane.QUESTION_MESSAGE);
    if (over == JOptionPane.YES_OPTION) {// 再来一局
                for (int i = 0; i < 9; i++) {
                    jb[i].setText(" ");
                    jb[i].setEnabled(true);
                }
            } else {
                System.exit(0); // 退出游戏
            }
            }
    }
}
```

然后，获取棋局状态的方法加入寻找两连子的代码，如下：

```
// 获取棋盘当前状态
public int gameState(JButton[] jb) {
    int result = INPROGRESS;
    boolean isFull = true;
```

```
        // 判断棋盘是否已满
        for (int pos = 0; pos < 9; pos++) {
            char chess = jb[pos].getText().charAt(0);
            if (empty == chess) {
                isFull = false;
            }
        }
        // 寻找三连子情况
for (int[] status : WIN_STATUS) {// 遍历 8 中棋局获胜状态
        // 得到某个获胜棋局状态的第一个索引的字符
            char chess = jb[status[0]].getText().charAt(0);
        // 如果为空,说明此处未下棋子,跳出循环,找下一个状态
            if (chess == empty) {
                continue;
            }
            int i;
        for (i = 1; i < status.length; i++) {// 查看其余两个字符
if (jb[status[i]].getText().charAt(0) ! = chess) { // 不与第一个索
//引字符一致
                    break; // 表明未三子连线,跳出
                }
            }
            if (i == status.length) { // 三子连线
                result = chess == 'X' ? WIN : LOSE;
                break;
            }
        }
        // 寻找两连子情况
        if (result ! = WIN & result ! = LOSE) {

            if (isFull) {
            result = DRAW;        //不输不赢且棋盘满则为平
            } else {
            int[] finds = new int[2];// 存放 X 或 O 的两连子情况
                for (int[] status : WIN_STATUS) {
                    char chess = empty;
                    boolean hasEmpty = false;
                    int count = 0;    // 计数
                    for (int i = 0; i < status.length; i++) {
                if (jb[status[i]].getText().charAt(0) == empty) {
                    hasEmpty = true;      // 该处没有棋子
                } else {
                if (chess == empty) {// 有棋子
                    chess = jb[status[i]].getText().charAt(0);
                }
                if (jb[status[i]].getText().charAt(0) == chess) {
                    count++;  //且棋子相同则加 1
                }
            }
        }
        if (hasEmpty && count > 1) {
            if (chess == 'X') {
                finds[0]++;
            } else {
                finds[1]++;
            }
        }
}
        // 两连子情况
        if (finds[1] > 0) {// O 的两连子
            result = -DOUBLE_LINK;
        } else if (finds[0] > 0) {      // X 的两连子
            result = DOUBLE_LINK;
        }
}
}
return result;    // 记录了胜负平或者两连子情况
}
```

O 方走棋时，要得到走棋位置，用一个方法来获取该位置，如下：

```
public int getNextMove(JButton[] board) {
    int nextPos = minimax(board, 3);
    return nextPos;
}
```

上面方法中调用了极小极大算法 minimax，如下：

```
//以 'X' 的角度来考虑的极小极大算法
public int minimax(JButton[] board, int depth) {
    int[] bestMoves = new int[9];//存放最佳走棋位置
    int index = 0;
    int bestValue = -INFINITY;
    // 搜索所有空位,试探填上 X,然后选其中最小值的
    for (int pos = 0; pos < 9; pos++) {
        if (board[pos].getText().charAt(0) == empty) {
            board[pos].setText("X");
            int value = min(board, depth);// 得到最小值
            if (value > bestValue) {// 选择最小值里最大的
                bestValue = value;
                index = 0;
                bestMoves[index] = pos;
            } else if (value == bestValue) {
                index++;
                bestMoves[index] = pos;
            }
            board[pos].setText(" ");
        }
    }
return bestMoves[index];
}
```

最后，两个递归方法 min 和 max 如下：

```
//对于 'O',估值越小对其越有利
public int min(JButton[] board, int depth) {
    int evalValue = gameState(board);
boolean isGameOver =  (evalValue == WIN || evalValue ==
```

```
LOSE || evalValue == DRAW);
            if (depth == 0 || isGameOver) {
                  return evalValue;
            }
            int bestValue = INFINITY;
            for (int pos = 0; pos < 9; pos++) {
                  if (board[pos].getText().charAt(0) == empty) {
                        board[pos].setText("O");
                        // 选择最小值
                  bestValue = Math.min (bestValue, max(board,
depth – 1));
                        board[pos].setText(" ");
                  }
      }
      return evalValue;
}
//对于 ′X′,估值越大对其越有利
public int max(JButton[] board, int depth) {
      int evalValue = gameState(board);
      boolean isGameOver =  (evalValue == WIN || evalValue
== LOSE || evalValue == DRAW);
            if (depth == 0 || isGameOver) {
                  return evalValue;
            }
            int bestValue = –INFINITY;
            for (int pos = 0; pos < 9; pos++) {
                  if (board[pos].getText().charAt(0) == empty) {
                        board[pos].setText("X");
                        // 选择最大值
                  bestValue = Math.max (bestValue, min (board,
depth – 1))board[pos].setText(" ");
                  }
      }
      return evalValue;
}
```

4 结语

人人对弈较为简单，把对胜负判定的代码写好就行了，人机对弈需要考虑算法，让计算机来计算下一步走棋的位置，这比较复杂一些，尽管用了很大的篇幅来讲解极小极大算法，可能还是不太好理解，需要大家对照代码进一步仔细思考了。

还有一点在文中没有交代，就是搜索棋局过程中，限定了搜索的深度，即在 min 和max 方法中通过 depth 变量来控制的，程序中限定 depth 是 3，即搜索 3 层，因为每增加一层，棋局状态都会成几何指数的增长，层数太多会加大计算机的计算时间。这里不再详细探讨了，有兴趣的可以查看相关资料。

这里介绍的算法是最基本的，还有在此基础上的剪枝算法、负极小极大算法等，大家可以深入地去进一步学习。

（收稿日期：2013–02–18）

网游巨头云聚分享传统游戏转型之路

一份由艾瑞咨询提供，2012 年第一季度中国手机网游市场规模突破 10 亿元，同比增长超过五成，手机网游用户规模超过 1 亿，同比增长 34.8%。另外 CNNIC 也显示，2012 年我国手机网民数量为 4.2 亿，年增长率达 18.1%，远超网民整体增幅。此外，网民中使用手机上网的比例也继续提升，由 69.3%上升至 74.5%。

2012 年传统网络的发展依旧光鲜亮丽，但是增速下降以及总量增加比不高说明了目前传统网络发展已经从爆发期向缓和期过渡。在整体网民增速放缓的时代，更多的电脑用户在已经使用手机上网来取代原有上网的模式。依附于传统互联网的网游面临增量放缓的大环境，如何寻求突破成为 2013 年的关键点。

面对手机游戏这块还未完全开发的蛋糕，以及不断增长的用户付费数都吸引不同厂商加入这一领域。作为传统互联网“吸金大户”的网游巨头同样不想错过，如何利用自己原有的资源，搭上移动互联网的快车成为他们考虑的重要问题。面对着 2013 中国移动广告将达 16 亿美元的诱惑，金山软件副总裁兼金山游戏总裁西山居游戏 CEO 邹涛、昆仑万维创始人兼 CEO 周亚辉、光宇游戏总裁宋洋三家传统网游巨头云聚 GMGC 第二届全球移动游戏大会，并围绕“传统厂商的转型之路”主题与众多移动互联网业内人士，及开发者、运营商、终端厂商等各家关键合作伙伴共同畅谈讨论。

第二届 GMGC 全球移动游戏大会将于 5 月 6 日在北京国家会议中心召开，大会由全球移动游戏联盟 GMGC 主办，全面邀请来自全球移动游戏产业生态链的相关者参与，会场将全球移动游戏开发者训练营品及国际推广与合作专场、创业世界杯–全球移动游戏应用评选大赛等多个板块，同时全球移动游戏联盟将通过组织 GMGC 全球移动游戏大会、沙龙、比赛等多种形式的交流和推广活动，把游戏玩家、开发商、运营商、平台商等齐聚一堂，共商合作大计，为移动游戏产业共建和谐健康的生态链！