计算机系列教材

Java五子棋游戏制作

宁淑荣 杨国兴 主编

清华大学出版社

计算机系列教材

宁淑荣 杨国兴 主编

Java五子棋游戏制作

清华大学出版社
北京

内 容 简 介

本书以五子棋游戏制作为例，介绍 Java 在开发应用软件中的各种技术，并体现面向对象的设计思想。内容包括单机版五子棋、下网络五子棋、下棋数据的保存以及人机对战等。书中对于 Java 中的输入输出、数据库、异常处理、网络编程以及界面编程等进行了比较深入的探讨。

本书可作为计算机相关专业 Java 课程设计、Java 实训等课程的教材，也可作为学生毕业设计以及 Java 程序设计爱好者的参考书。

图书在版编目(CIP)数据

Java 五子棋游戏制作/宁淑荣，杨国兴主编. —北京：清华大学出版社，2017
(计算机系列教材)
ISBN 978-7-302-46563-8

Ⅰ. ①J…　Ⅱ. ①宁… ②杨…　Ⅲ. ①JAVA 语言－程序设计－教材　Ⅳ. ①TP312.8

中国版本图书馆 CIP 数据核字(2017)第 030137 号

责任编辑：白立军　张爱华
封面设计：常雪影
责任校对：李建庄
责任印制：宋　林

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795954
印 装 者：清华大学印刷厂
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：10　　**字　　数**：243 千字
版　　次：2017 年 5 月第 1 版　　**印　　次**：2017 年 5 月第1次印刷
印　　数：1～2000
定　　价：29.00 元

产品编号：072065-01

Java 是目前使用最广泛的语言之一，对于软件开发人员，掌握 Java 语言基础以及拥有使用 Java 进行软件开发的能力是非常重要的，因此大多数与计算机相关的专业都开设了 Java 程序设计课程。

Java 程序设计是一门实践性很强的课程，仅仅掌握 Java 的基本语法知识，与利用 Java 进行软件开发还有很大的差距。掌握 Java 基本知识后，应该通过大量的编程实践，逐步提高利用 Java 进行软件开发的能力。

本书以五子棋游戏制作为例，介绍利用 Java 进行软件开发的技术。五子棋游戏比较简单，是大家比较熟悉的游戏之一，因此选择五子棋游戏为例，有助于提高学习者的兴趣，易于跟着书中介绍的步骤，一步步将五子棋游戏制作出来。为了简单起见，本书中的五子棋程序不考虑禁手。

本书由 4 章内容组成，包括单机版五子棋、网络五子棋、下棋数据的保存，以及人机对战；涉及的主要知识有异常处理、输入输出流、数据库编程、多线程和网络编程等。

本书可作为计算机相关专业 Java 课程设计、Java 实训等课程的教材，也可作为学生毕业设计和 Java 游戏程序爱好者的参考书。

本书的所有程序都由作者亲自编写，并在 JDK1.6 环境下调试通过，数据库使用的是 MySQL 数据库。

为了方便教师教学与学生学习，本书为使用本教材的教师提供 PowerPoint 电子教案，方便教师根据具体情况进行必要的修改，相关资源可以从清华大学出版社网站 www.tup.com.cn 下载。

本书由宁淑荣、杨国兴主编，参加编写工作的还有庄凤娟、王国芳等。

由于作者水平有限，书中难免有不妥之处，恳请广大读者批评指正。

作 者

2017 年 3 月

第 1 章　单机版五子棋游戏

单机版五子棋程序，需要提供一个棋盘，让两个用户用鼠标轮流下棋，系统负责判断下棋的位置是否合法以及判断输赢。五子棋游戏的界面如图 1.1 所示。

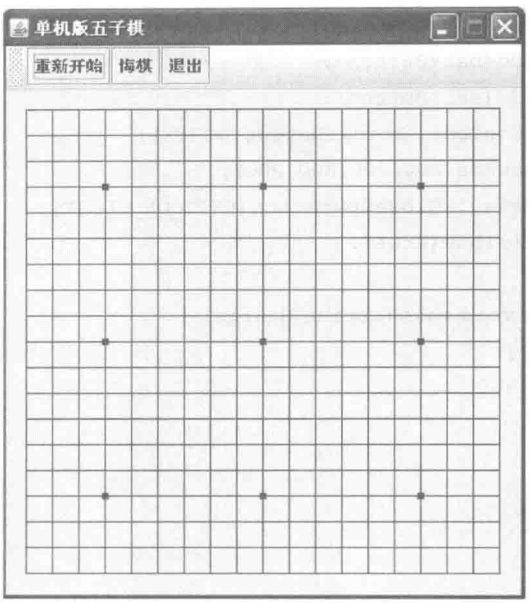

图 1.1　五子棋游戏的界面

本章将分 7 节分别完成以下步骤：创建窗口、创建棋盘类、创建棋子类、实现单击鼠标下棋、判断输赢、实现工具栏上的功能，以及改变鼠标的形状。

在实际的五子棋比赛中，为了公平，先行的黑棋是有禁手的，即某些棋型黑棋是不可以下的。为简单起见，本书中的五子棋程序不考虑禁手。

1.1　五子棋游戏窗口制作

五子棋窗口类从 JFrame 继承，窗口上面包含一个工具栏，工具栏上有 3 个按钮，如图 1.2 所示。

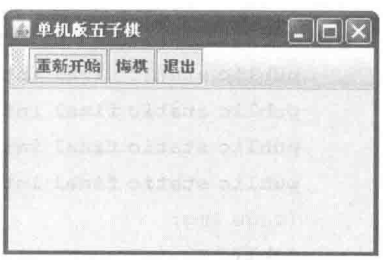

图 1.2　五子棋窗口

启动 Eclipse 后，创建一个名字为 Five 的 Java project，然后创建类 Five(从 JFrame 继承)，在类中添加一个工具栏和 3 个按钮，在构造方法中创建这些组件对象，并将这些组件组织到窗口中，在主方法中创建一个 Five 类对象。代码如下：

```
public class Five extends JFrame {
    private JToolBar toolbar;
    private JButton startButton,backButton,exitButton;
    public Five(){
        super("单机版五子棋");
        toolbar=new JToolBar();
        startButton=new JButton("重新开始");
        backButton=new JButton("悔棋");
        exitButton=new JButton("退出");
        toolbar.add(startButton);
        toolbar.add(backButton);
        toolbar.add(exitButton);
        this.add(toolbar, BorderLayout.NORTH);
        this.setBounds(200,200,300,200);
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        this.setVisible(true);
    }
    public static void main(String[] args) {
        new Five();
    }
}
```

1.2 创建棋盘类

1.2.1 准备图片

棋盘由一张背景图片和 19 条横线与 19 条竖线组成，棋盘界面如图 1.1 所示。

准备一张图片作为棋盘的背景，将图片的名字改为 board.jpg，在 project 文件夹中建立一个子文件夹 img，将背景图片文件放在子文件夹 img 中。

1.2.2 棋盘类的创建

创建棋盘类 ChessBoard，该类从 JPanel 继承。代码如下：

```
public class ChessBoard extends JPanel {
    public static final int MARGIN=15;                    //边距
    public static final int SPAN=20;                      //网格宽度
    public static final int ROWS=18;                      //棋盘行数
    public static final int COLS=18;                      //棋盘列数
    Image img;
    public ChessBoard(){
        img=Toolkit.getDefaultToolkit().getImage("img/board.jpg");
```

```
    }
    //画棋盘
    public void paintComponent(Graphics g){
        super.paintComponent(g);
        g.drawImage(img, 0, 0, this);
        for(int i=0;i<=ROWS;i++){                          //画横线
            g.drawLine(MARGIN,MARGIN+i * SPAN,
                       MARGIN+COLS * SPAN,MARGIN+i * SPAN);
        }
        for(int i=0;i<=COLS;i++){                          //画竖线
            g.drawLine(MARGIN+i * SPAN,MARGIN,
                       MARGIN+i * SPAN,MARGIN+ROWS * SPAN);
        }
        g.fillRect(MARGIN+3 * SPAN-2, MARGIN+3 * SPAN-2, 5, 5);
        g.fillRect(MARGIN+(COLS/2) * SPAN-2, MARGIN+3 * SPAN-2, 5, 5);
        g.fillRect(MARGIN+(COLS-3) * SPAN-2, MARGIN+3 * SPAN-2, 5, 5);
        g.fillRect(MARGIN+3 * SPAN-2, MARGIN+(ROWS/2) * SPAN-2, 5, 5);
        g.fillRect(MARGIN+(COLS/2) * SPAN-2, MARGIN+(ROWS/2) * SPAN-2, 5, 5);
        g.fillRect(MARGIN+(COLS-3) * SPAN-2, MARGIN+(ROWS/2) * SPAN-2, 5, 5);
        g.fillRect(MARGIN+3 * SPAN-2, MARGIN+(ROWS-3) * SPAN-2, 5, 5);
        g.fillRect(MARGIN+(COLS/2) * SPAN-2, MARGIN+(ROWS-3) * SPAN-2, 5, 5);
        g.fillRect(MARGIN+(COLS-3) * SPAN-2, MARGIN+(ROWS-3) * SPAN-2, 5, 5);
    }
    public Dimension getPreferredSize(){
        return new Dimension(MARGIN * 2+SPAN * COLS, MARGIN * 2+SPAN * ROWS);
    }
}
```

类中定义的常量分别是棋盘边框的宽度，网格的宽度以及网格的行数和列数。注意：网格的行数比横线少1，网格的列数比竖线少1，将网格的行列数都定义为18，则棋盘的横线和竖线都是19条。

当一个组件需要重画时，会调用到自身的paintComponent()方法，因此我们在这个方法中绘制棋盘。首先将背景图片显示出来，然后第一个循环画出横线，第二个循环画出竖线，最后9行代码画出棋盘上的9个小的黑色正方形。

方法getPreferredSize()返回棋盘最适合的尺寸，在后面的程序中，确定主框架窗口大小时会用到这个方法。

1.2.3 显示棋盘

在Five类中添加ChessBoard类的属性成员。

```
private ChessBoard boardPanel;
```

在构造方法中创建boardPanel对象，并将其添加到窗口框架中。

```
boardPanel=new ChessBoard();
this.add(boardPanel, BorderLayout.CENTER);
```

将原来设置窗口位置和大小的语句替换成下列 3 行代码：

```
this.setLocation(200,200);
this.pack();
this.setResizable(false);
```

setLocation()方法将窗口的左上角设置到(200,200)，调用方法 pack()，使窗口的大小根据窗口中组件的大小自动调整，因此需要知道棋盘的大小。为此，在棋盘类中添加了一个方法 getPreferredSize()。

调用方法 setResizable(false)，使窗口的大小不可改变。

1.3 创建棋子类

1.3.1 棋子类

棋子类的属性包括棋子的直径、颜色(黑或白)、在棋盘中的行列号，以及棋子所在的棋盘。棋子的直径是一个不变的值，因此定义为常量，将其值设置为比单元格宽度稍小一点。

棋子类的方法包括构造方法、属性的 get()方法，最主要的一个方法是将棋子画出来的方法 draw()。

程序代码如下：

```
public class Chess {
    public static final int DIAMETER=ChessBoard.SPAN-2;
    private int col;                                    //棋子在棋盘中的 x 索引
    private int row;                                    //棋子在棋盘中的 y 索引
    private Color color;                                //颜色
    ChessBoard cb;
    public Chess(ChessBoard cb,int col,int row,Color color){
        this.cb=cb;
        this.col=col;
        this.row=row;
        this.color=color;
    }
    public int getCol() {
        return col;
    }
    public int getRow() {
        return row;
    }
    public Color getColor() {
        return color;
```

```
    }
    public void draw(Graphics g){
        int xPos=col * cb.SPAN+cb.MARGIN;
        int yPos=row * cb.SPAN+cb.MARGIN;
        Graphics2D  g2d=(Graphics2D) g;
        RadialGradientPaint paint=null;
        int x=xPos+DIAMETER/4;
        int y=yPos-DIAMETER/4;
        float[] f={0f, 1f};
        Color[] c={Color.WHITE, Color.BLACK};
        if(color==Color.black){
            paint=new RadialGradientPaint(x,y, DIAMETER, f, c);
        }
        else if(color==Color.white){
            paint=new RadialGradientPaint(x, y, DIAMETER * 2, f, c);
        }
        g2d.setPaint(paint);
        //以下两行使边界更均匀
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                             RenderingHints.VALUE_ANTIALIAS_ON);
        g2d.setRenderingHint(RenderingHints.KEY_ALPHA_INTERPOLATION,
                           RenderingHints.VALUE_ALPHA_INTERPOLATION_DEFAULT);
        Ellipse2D e=new Ellipse2D.Float(xPos-DIAMETER/2, yPos-DIAMETER/2,
                                        DIAMETER,DIAMETER);
        g2d.fill(e);
    }
}
```

为了使棋子有立体效果，可使用一种叫做圆形辐射颜色渐变模式填充方式。下面结合图 1.3 介绍黑棋棋子的绘制过程。

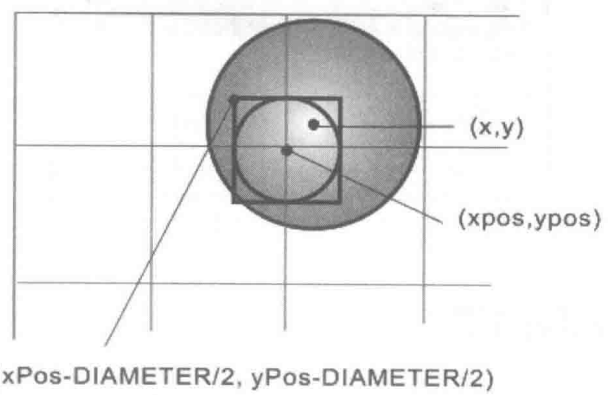

图 1.3 绘制棋子

图中的小圆表示要绘制的黑棋棋子，其直径是 DIAMETER，首先获取棋子的中心坐

标(xPos, yPos)。图中的大圆用于填充,其直径为 2DIAMETER,中心为棋子的高光点,位于棋子右上方 1/4 处。

RadialGradientPaint 类提供了圆形辐射颜色渐变模式填充方式,在程序中使用 5 个参数的构造方法构造一个 RadialGradientPaint 对象:前两个参数为填充的中心坐标;第 3 个参数是渐变范围圆的半径;第 4 个参数是实型数组,在我们的程序中是{0,1};第 5 个参数是 Color 型数组,在我们的程序中是{Color. WHITE, Color. BLACK}。第 4 个参数和第 5 个参数要配合使用,元素的个数要相同,0 表示渐变的中心位置,对应颜色 Color. WHITE,1 表示圆周上,对应的颜色是 Color. BLACK,中间则是两种颜色的逐渐过渡,当然这两个数组也可以有多个元素,实现多种颜色的渐变。

创建 RadialGradientPaint 对象后,调用 setPaint()方法将其选到 g2d 中,最后创建棋子椭圆并填充。注意 Ellipse2D 是抽象类,Ellipse2D. Float 是 Ellipse2D 的一个内部子类,其构造方法的参数分别是椭圆外切矩形的左上角坐标和外切矩形的长与宽。

调用 setRenderingHint()方法的两行代码的作用是使棋子的边界绘制得更平滑。

白棋的绘制与此类似,可以仔细分析程序中创建 RadialGradientPaint 对象的代码,理解白棋棋子的绘制。

1.3.2 在棋盘上画出棋子

为了测试棋子类,在棋盘类的 paintComponent()方法的最后加上以下 4 行代码:

```
Chess c1=new Chess(this,2,1,Color.BLACK);
Chess c2=new Chess(this,5,2,Color.WHITE);
c1.draw(g);
c2.draw(g);
```

上述代码创建两个棋子 c1(2 列、1 行、黑棋)和 c2(5 列、2 行、白棋),然后将两个棋子绘制出来。效果如图 1.4 所示。

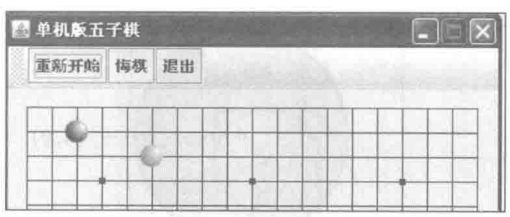

图 1.4 棋盘上的两个棋子

1.4 实现单击鼠标下棋

前面已在棋盘类中的 paintComponent()方法中创建两个棋子并显示出来,而在实际的对局中,是通过单击鼠标进行下棋的,下面的程序实现单击鼠标下棋。

为了记录下棋的过程,在棋盘类中增加以下 3 个属性。

```
Chess[] chessList;                    //记录已经下在棋盘上的棋子的数组
int chessCount;                       //当前棋盘上棋子的个数
boolean isBlack=true;                 //下一步轮到哪一方下棋,默认开始是黑棋先
```

在某一位置下棋之前,应该检测该位置是否已经有棋子,如果已经有了棋子,则不可以在该位置再下棋,因此为棋盘类添加一个方法 hasChess()。代码如下:

```
private boolean hasChess(int col,int row){
    for(int i=0; i<chessCount; i++){
        Chess ch=chessList[i];
        if(ch!=null&&ch.getCol()==col&&ch.getRow()==row)
            return true;
    }
    return false;
}
```

在棋子数组中逐个判断棋子是否在指定的位置,如果找到一个棋子在该位置,返回 true,否则返回 false。

为了响应鼠标消息,给棋盘类添加一个内部类 MouseMonitor,从 MouseAdapter 类继承,重写函数 mousePressed(),当鼠标按下时会调用此方法。代码如下:

```
class MouseMonitor extends MouseAdapter{
    public void mousePressed(MouseEvent e){
        //将鼠标单击的像素坐标转换成网格索引
        int col=(e.getX()-MARGIN+SPAN/2)/SPAN;
        int row=(e.getY()-MARGIN+SPAN/2)/SPAN;
        //落在棋盘外不能下棋
        if(col<0||col>COLS||row<0||row>ROWS)
            return;
        //如果 x,y 位置已经有棋子存在,不能下棋
        if(hasChess(col, row))
            return;
        Chess ch=new Chess(ChessBoard.this, col, row, isBlack?Color.black:
                Color.white);
        chessList[chessCount++]=ch;
        repaint();                    //通知系统重新绘制
        isBlack=!isBlack;
    }
}
```

首先将鼠标点的像素坐标转换为棋盘坐标,在图 1.5 所示的"矩形 1"区域内按下鼠标,其棋盘坐标都应该是(2,1)。先将按下鼠标点的行、列坐标都加上单元格宽度的一半,这样的坐标范围相当于"矩形 2"的区域,然后将其整除单元格的宽度,就得到需要的棋盘坐标。

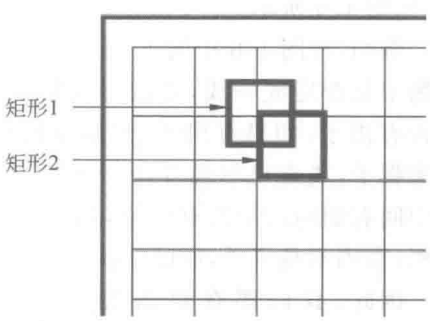

图 1.5 像素坐标转换为棋盘坐标

然后判断该位置是否可以下棋。如果鼠标位于棋盘之外，或该位置已经有了棋子，则不能下棋，直接返回。如果可以下棋，则创建一个棋子对象，棋子的颜色由属性 isBlack 当前的值确定，如果 isBlack 为真，则创建黑子，否则创建白子。将棋子添加到棋子数组中，再将棋盘上的棋子数增加 1。

最后调用 repaint()方法，重绘棋盘，再将 isBlack 的值翻转一次，实现黑白两方轮流下棋。

将 paintComponent()方法的最后 4 行用下面的程序替换。通过循环将棋盘上的所有棋子绘制出来。为了标识最后一步棋的位置，在最后的棋子上画一个红色矩形。代码如下：

```
for(int i=0;i<chessCount;i++){
    chessList[i].draw(g);
    if(i==chessCount-1){                    //如果是最后一个棋子
        int xPos=chessList[i].getCol() * SPAN+MARGIN;
        int yPos=chessList[i].getRow() * SPAN+MARGIN;
        g.setColor(Color.red);
        g.drawRect(xPos-Chess.DIAMETER/2, yPos-Chess.DIAMETER/2,
                   Chess.DIAMETER, Chess.DIAMETER);
    }
}
```

在棋盘类的构造方法中注册鼠标监听，完成通过鼠标下棋的功能。

```
this.addMouseListener(new MouseMonitor());
```

以上程序已经实现了黑白两方轮流下棋的功能，在 1.5 节中将实现判断输赢的功能。

1.5 判断赢棋

每下一个棋子都要判断是否有 5 个连续的棋子连成一线，一旦 5 个连续棋子连成一线就赢棋，棋局结束。

在判断 5 子连成一线时，要考虑 4 个方向：水平、垂直、从左上角到右下角，以及从左下角到右上角，如图 1.6 所示。

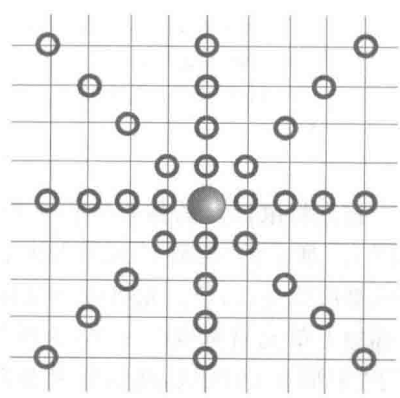

图 1.6 4 个方向判断 5 子连成一线

例如，在图 1.6 中间下了一个黑子，判断水平方向黑子是否连成一线，方法是先检查中间左侧的点是否有黑子，如果有黑子，则继续向左判断，直到不是黑棋子，或者已经够 5 个子了则结束。否则再判断中间右侧的点是否有黑子，如果有黑子，继续向右判断，直到不是黑子，或已经够 5 个黑子则结束。

因此，我们要在棋盘类中再添加一个方法 hasChess()[与 1.4 节的 hasChess()方法是重载关

系]，判断某个点有没有黑子，或有没有白子。代码如下：

```
private boolean hasChess(int col, int row, Color color){
    for(int i=0; i<chessCount; i++){
        Chess ch=chessList[i];
        if(ch!=null&&ch.getCol()==col&&ch.getRow()==row &&ch.getColor()==color)
            return true;
    }
    return false;
}
```

函数通过一个循环，逐个判断棋盘上的棋子是不是参数指定位置和相应颜色的棋子，如果是，返回 true，如果循环结束仍然没有找到参数指定的棋子，则返回 false。

判断胜负方法的代码如下：

```
private boolean isWin(int col, int row){
    int continueCount=1;                    //连续棋子的个数
    Color c=isBlack?Color.black:Color.white;
    //横向向左寻找
    for(int x=col-1;x>=0;x--){
        if(hasChess(x,row,c))
            continueCount++;
        else
            break;
    }
    //横向向右寻找
    for(int x=col+1;x<=COLS;x++){
        if(hasChess(x,row,c))
            continueCount++;
        else
            break;
    }
    if(continueCount>=5)
        return true;
    else
        continueCount=1;
    //继续另一种搜索;纵向
    //向上搜索
    for(int y=row-1;y>=0;y--){
        if(hasChess(col,y,c))
            continueCount++;
        else
            break;
    }
    //纵向向下寻找
```

```
for(int y=row+1;y<=ROWS;y++){
    if(hasChess(col,y,c))
        continueCount++;
    else
        break;
}
if(continueCount>=5)
    return true;
else
    continueCount=1;
//继续另一种情况的搜索:右上到左下
//向右上寻找
for(int x=col+1,y=row-1; y>=0&&x<=COLS; x++,y--){
    if(hasChess(x,y,c))
        continueCount++;
    else
        break;
}
//向左下寻找
for(int x=col-1,y=row+1; x>=0&&y<=ROWS; x--,y++){
    if(hasChess(x,y,c))
        continueCount++;
    else
        break;
}
if(continueCount>=5)
    return true;
else
    continueCount=1;
//继续另一种情况的搜索:左上到右下
//向左上寻找
for(int x=col-1,y=row-1; x>=0&&y>=0; x--,y--){
    if(hasChess(x,y,c))
        continueCount++;
    else
            break;
}
//右下寻找
for(int x=col+1,y=row+1; x<=COLS&&y<=ROWS; x++,y++){
    if(hasChess(x,y,c))
        continueCount++;
    else
        break;
}
```