**《JAVA语言程序设计》**

**课程设计报告**

　　　　　　　　　　（2016—2017学年第二学期）

题 目 人机五子棋

学生姓名 刘 壮 飞

专业班级 软件15-1

学生学号 311509060128

教师姓名 智 慧 来

成 绩**：**

评 语：

教师签名：

日期：

**河南理工大学**

**JAVA语言课程设计评阅书**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 题目名称 | | 人机五子棋 | | | | | | | |
| 项目 | 分值 | 考核内容 | 优秀 | 良好 | 中等 | 及格 | 不及格 | 分值 |
| 选题 | 25 | 选题符合专业培养目标，体现综合训练基本要求；有一定的学术性和应用价值 | ≥22 | ≥20 | ≥18 | ≥15 | ＜15 |  |
| 成果与  能力 | 30 | 成果有创新性或独到见解；综合运用所学知识分析与解决问题，动手能力强 | ≥27 | ≥24 | ≥21 | ≥18 | ＜18 |  |
| 工作量 | 25 | 按任务书要求完成规定的工作量，达到选题要求 | ≥22 | ≥20 | ≥18 | ≥15 | ＜15 |  |
| 规范  程度 | 20 | 结构规范，逻辑性强，思路清晰，语言准确，文字流畅，图纸、表格、插图等绘制规范 | ≥18 | ≥16 | ≥14 | ≥12 | ＜12 |  |
| 评阅成绩（百分制）：  评阅教师签名：  年 月 日 | | | | | | | | |

目录

[1.任务设计书 1](#_Toc485909554)

[2.类与对象的设计 1](#_Toc485909555)

[2.1类 1](#_Toc485909556)

[2.1.1位置实体类Location 1](#_Toc485909557)

[2.1.2自定义棋盘类ChessPanel 2](#_Toc485909558)

[2.1.3 控制器类Chess 3](#_Toc485909559)

[2.1.4 视图类View 5](#_Toc485909560)

[2.1.5 测试类Test 5](#_Toc485909561)

[2.2对象 5](#_Toc485909562)

[2.3对象之间的关系 6](#_Toc485909563)

[3.主要方法的实现 6](#_Toc485909564)

[3.1实现原理 6](#_Toc485909565)

[3.1.1评分表 7](#_Toc485909566)

[3.1.2评估函数 8](#_Toc485909567)

[3.1.3算法效果 9](#_Toc485909568)

[3.1.4 AI落子过程分析 9](#_Toc485909569)

[3.1.5算法改进后效果 11](#_Toc485909570)

[3.2关键功能实现 11](#_Toc485909571)

[3.2.1 窗口布局 11](#_Toc485909572)

[3.2.2控制器的实现 13](#_Toc485909573)

[3.2.3棋盘的实现 14](#_Toc485909574)

[3.2.4位置实体类实现 14](#_Toc485909575)

[4.运行结果 14](#_Toc485909576)

[4.1效果图 14](#_Toc485909577)

[4.2问题分析 14](#_Toc485909578)

[5.总结与致谢 15](#_Toc485909579)

[6.参考文献 16](#_Toc485909580)

[7.附录：全部代码 17](#_Toc485909581)

[7.1 Location 17](#_Toc485909582)

[7.2 ChessPanel 18](#_Toc485909583)

[7.3 Chess 21](#_Toc485909584)

[7.4 View 34](#_Toc485909585)

[7.5 Test测试类 38](#_Toc485909586)

# 1.任务设计书

本项目要实现的是五子棋人机版，通过制定棋型的评分表使机器能够对棋盘局势评估。五子棋玩家双方分别称为“人”、“机器” ，当人落子后，机器对棋盘扫描获取可行棋的位置集合，然后遍历该集合，利用评估函数对每个空位依次估分，得分最高的位置即为机器要落子的位置，在使用评估函数对空位打分时，为了避免机器只攻不守，需要使用“换位思考”的思想，也就是说打分时不仅考虑自身，还要考虑对方。

# 2.类与对象的设计

**2.1类**

**2.1.1位置实体类Location**

Location类封装棋盘上的一个位置，AI对局势分析时会对位置打分，所以位置实体类应该有个字段保存位置分数，Location类的设计如图1所示。

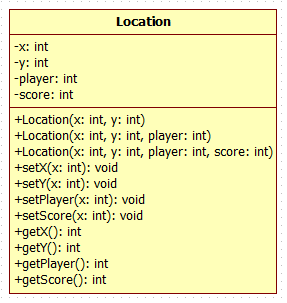


图1.位置实体类类图

1）public Location(int x, int y)

构造函数。x：横坐标，y：纵坐标

2）public Location(int x, int y, int player)

构造函数。x：横坐标，y：纵坐标，player：位置所有者

3）public Location(int x, int y, int player, int score)

构造函数。x：横坐标，y：纵坐标，player：位置所有者，score：位置分数

4）public void setX(int x)

设置横坐标的值

5）public void setY(int y)

设置纵坐标的值

6）public void setScore(int score)

设置位置分数

7）public void setPlayer(int player)

设置该位置由玩家player落子，player可取：Chess.PLAYER、Chess.AI

8）public int getX()

获取对象的横坐标

9）public int getY()

获取对象的纵坐标

10）public int getPlayer()

获取该位置是由哪位玩家所有

11）public int getScore()

获取该位置的分数

**2.1.2自定义棋盘类ChessPanel**

ChessPanel类负责视图上的事情，如棋盘以及棋子的绘制、棋盘状态的保存、落子、清空等事件，ChessPanel类的设计如图2所示。

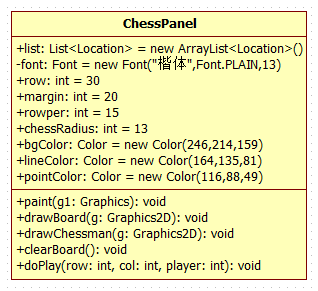


图2.棋盘类类图

1）public void paint(Graphics g1)

重写该方法，绘制棋盘、棋子

2）public void drawBoard(Graphics2D g)

绘制棋盘

3）public void drawChessman(Graphics2D g)

绘制棋子

4）public void clearBoard()

清空棋盘

5）public void doPlay(int row, int col, int player)

玩家在视图上落子

**2.1.3 控制器类Chess**

Chess类主要负责逻辑上的各个事件，如逻辑上落子、AI局势分析、胜负判定等，Chess类的设计如图3所示。

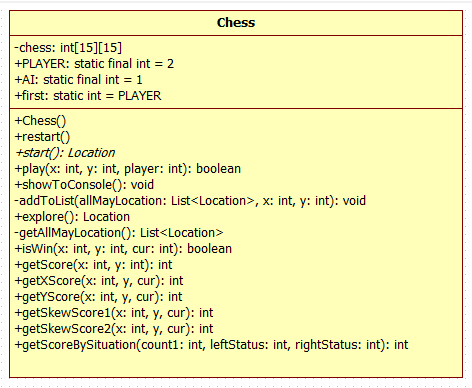


图3.控制器类图

1）public void setFirst(int first)

设置先手的玩家

2）public Chess()

构造函数，进行棋局的初始化

3）public void restart()

棋局初始化

4）public Location start()

AI先手时调用，决策第一手棋位置

5）public boolean play(int x, int y, int player)

玩家落子。x，y是落子坐标，player的取值：Chess.AI、Chess.PLAYER，返回值表示是否落子成功

6）public void showToConsole()

显示棋盘信息到控制台

7）private void addToList(List<Location> allMayLocation, int x, int y)

添加位置到可行棋的位置集合，过滤重复的位置

8）public Location explore()

返回分数最高的位置

9）private List<Location> getAllMayLocation()

得到可行的位置集合

10）public boolean isWin(int x, int y, int cur)

判断胜负，x、y是落子位置，cur是Chess.AI、Chess.PLAYER

11）public int getScore(int x, int y)

局势评估函数，计算总得分

12） private int getScoreBySituation(int count1, int leftStatus, int rightStatus)

根据棋型计算空位得分，count1为相连棋子数，leftStatus、rightStatus为1或2，1代表为空，2为墙或者对方棋子

13）public int getXScore(int x, int y, int cur)

横向扫描计算得分

14）public int getYScore(int x, int y, int cur)

纵向扫描计算得分

15）public int getSkewScore1(int x, int y, int cur)

正斜向扫描计算得分

16）public int getSkewScore2(int x, int y, int cur)

反斜向扫描计算得分

**2.1.4 视图类View**

View类主要负责游戏窗口的显示以及对Chess、ChessPanel进行调度，通过调用控制器的方法来控制逻辑上的棋局，通过调用ChessPanel的方法来控制视图上的棋局，View类的设计如图4所示。

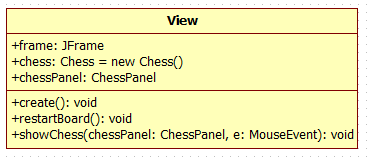


图4.视图类类图

1) public void create()

创建视图

2) public void restartBoard()

棋局重开

3) public void showChess(ChessPanel chessPanel, MouseEvent e)

鼠标点击落子事件处理，chessPanel是已实例化的棋盘面板对象，e是鼠标事件

**2.1.5 测试类Test**

Test类是程序的入口，提供了主函数，其主要负责实例化视图类View对象，Test类的设计如图5所示。

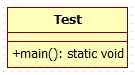


图5.测试类 类图

**2.2对象**

以四个对象为例，画出对象图，每个图中第一格是对象名：类名，第二格是对象拥有的属性，对象的设计如图6所示。

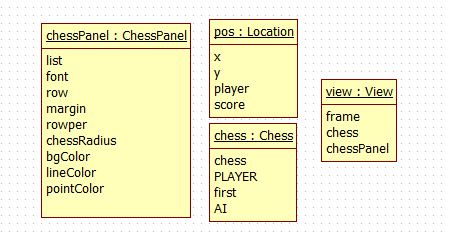


图6.四个对象图

**2.3对象之间的关系**

view对象是视图类的实例，chessPanel是棋盘类的实例，pos是位置实体类的实例，chess是控制器类的实例。可以说view是游戏的控制中枢，负责调度chessPanel和chess；chessPanel负责游戏视图；chess是负责游戏逻辑；pos表示一个位置，可以使用Location的集合来保存棋盘的状态。

**3.主要方法的实现**

**3.1实现原理**

我们时常会想，五子棋AI是怎么确定下在哪个位置上的，以下分析一下实现的原理。玩家落子后会形成一个局势，在该局势下AI依次为每个空位打分，得分最高的位置就是AI落子的位置。

问题一：空位那么多，遍历加上计算的开销可不小，那么怎么提高效率？

问题二：怎么为空位打分？

问题一解决：按照原理，每当玩家落子后AI需要遍历所有空位为其打分，其实有很多空位是可以直接排除掉的，每次都要计算一些无用的空位，所以在时间上就造成了浪费。玩过五子棋的人都知道，除第一手棋外，我们的棋子一般下到其他棋子的附近，也就是说那些远离棋子的空位是不需要考虑的，只有那些在非空位附近的位置才是需要考虑的，那么怎么定义“附近”这个概念呢?

设有某非空位置pos1，以pos1为中心呈米字型所能覆盖到的每一个空位pos2都可以称为是pos1附近的位置，整个棋盘所有非空位附近的位置构成一个无重复的集合称为可行位置集合。

表1.搜索空位示意图

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | △ | △ | ● | △ |  | △ | △ | △ |
| △ | △ | △ |  | △ | ● | △ | △ |  | △ | ○ | △ |
| △ | ● | △ |  | △ | ○ | △ |  |  | △ | △ | △ |
| △ | △ | △ |  | △ | △ | △ | △ | △ |  |  |  |
|  |  |  |  |  |  | △ | ● | △ |  |  |  |

●黑方棋子 ○白方棋子 △AI要计算的空位

从图中可以看到，AI只需要搜索与非空位附近的位置，减少了很多不必要的搜索、计算，虽然随着棋局的深入这种方法依然需要搜索非常多的位置，但是在这个过程中已经排除掉了不少的位置，运算效率已经有了提升。

问题二解决：为空位打分我们需要定义一张评分表作为评分的标准，假设为某空位pos打分，首先AI在这个位置试探性的放一枚己方棋子，AI为白方。

**3.1.1评分表**

1）五子情况

表2.五子棋型

|  |  |
| --- | --- |
| □○○○○○□ | 200000 |
| □○○○○○● |
| ●○○○○○□ |
| ●○○○○○● |

2）四子情况

表3.四子棋型

|  |  |
| --- | --- |
| □○○○○□ | 50000 |
| □○○○○● | 3000 |
| ●○○○○□ | 3000 |
| ●○○○○● | 1000 |

3）三子情况

表4.三子棋型

|  |  |
| --- | --- |
| □○○○□ | 3000 |
| ●○○○□ | 1000 |
| □○○○● | 1000 |
| ●○○○● | 500 |

4）二子情况

表5.二子棋型

|  |  |
| --- | --- |
| □○○□ | 500 |
| □○○● | 200 |
| ●○○□ | 200 |
| ●○○● | 100 |

5）一子情况

表6.一子棋型

|  |  |
| --- | --- |
| □○□ | 100 |
| □○● | 50 |
| ●○□ | 50 |
| ●○● | 30 |

五子棋中有连、长连、五连、成五、四、活四、冲四、死四等诸多概念，可以看到在评分表中有几种没有进行考虑，比如跳活三之类的情况暂时不进行判定。至此，我们有了评分表，接下来我们需要定义评估空位的方法。

**3.1.2评估函数**

评估函数是一个对单个可行位置评分的方法，比如它对某个可行位置pos进行评估，评估步骤如下：

1）横向扫描

A.以可行位置pos的左侧为中心，向左扫描

如果遇到空格，记录下左侧为空格，停止向左扫描

如果遇到己方棋子，棋子个数加1，继续向左扫描

如果遇到对方棋子，记录下左侧为对方棋子，停止向左扫描

如果已到达最左侧，记录下左侧为墙，停止向左扫描

B.以可行位置pos为中心，向右扫描

如果遇到空格，记录下右侧为空格，停止向右扫描

如果遇到己方棋子，棋子个数加1，继续向右扫描

如果遇到对方棋子，记录下右侧为对方棋子，停止向右扫描

如果已到达最右侧，记录下右侧为墙，停止向右扫描

C.根据棋子个数、评分表为该位置打分，该空位得分score1

2）纵向扫描

扫描纵向上的相连己方棋子个数，根据棋子个数、评分表为该位置打分，该空位得分score2

3）正斜向扫描

扫描正斜向上的相连己方棋子个数，根据棋子个数、评分表为该位置打分，该空位得分score3

4）反斜向扫描

扫描反斜向上的相连己方棋子个数，根据棋子个数、评分表为该位置打分，该空位得分score4

5）对以上四个方位之后可以得到score=score1+score2+score3+score4，我们将总得分作为该空位的评分，这样就能使AI考虑到周围四个方向。

**3.1.3算法效果**

AI下的位置就是在所有可行位置中评分结果最高的位置，评分结果最高意味着AI觉得这个位置对自己是最有利的，这个算法效果怎么样呢？ AI是白棋，玩家先手，由图，AI的第6手是个败笔，AI第6手应该封堵黑棋，但是它并没有这个意识，可知此时的AI太脆弱，不具备”防”的意识，只有“攻”的意识。原始算法的效果如图7所示。

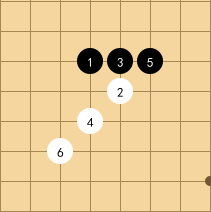


图7.初始算法效果图

**3.1.4 AI落子过程分析**

以下通过模拟AI分析棋局的方式分析AI第6手为什么不封堵黑棋，并找到改善的方案。

1）玩家执黑，AI执白，首先根据局势找出AI所有可行棋位置

表7.局势的可行位置分析

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | △ | △ | △ | △ | △ |  |  |
|  | △ | ● | ● | ● | △ |  |  |
|  | △ | △ | ○ | △ | △ |  |  |
|  | △ | ○ | △ | △ |  |  |  |
|  | △ | △ | △ |  |  |  |  |
|  |  |  |  |  |  |  |  |

●黑方棋子 ○白方棋子 △AI要计算的空位

2）对上步分析到的可行位置依次计算空位得分，评分结果如下

表8.局势可行位置评分

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 350 | 300 | 250 | 250 | 300 |  |  |
|  | 350 | ● | ● | ● | 350 |  |  |
|  | 750 | 850 | ○ | 700 | 350 |  |  |
|  | 800 | ○ | 900 | 500 |  |  |  |
|  | **1300** | 800 | 800 |  |  |  |  |
|  |  |  |  |  |  |  |  |

●黑方棋子 ○白方棋子

单元格中的分数代表该空位的评分，第五行第三列的空位评分最高，所以AI应该将棋子下到该位置处。通过模拟这个过程可以知道，评估函数在为空位打分时只考虑到了己方的棋子，在比对评分表时并没有考虑对方的局势，所以AI决策出的棋子位置只注重了“攻”，而忽视了“防”。

那么怎么才能平衡两者呢？我想到的方法是“换位思考”，所谓“换位思考”，就是当评估函数对空位打分时，不仅要考虑自己，还要站在对方的角度来看待局势。原算法对空位评分时是以AI白棋的角度评定该空位四个方向上形成的棋型，根据相应棋型给出相应得分aiScore，改进的算法还要以玩家黑棋的角度对空位评分，计算出得分playerScore，该空位的得分为score=aiScore+playerScore。

**3.1.5算法改进后效果**

假设刚才的棋局进行到了第6手，轮到AI执棋，我们用改进的算法分析一下AI应该落子的位置

表9.改进后局势分析

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  | 850 | 1400 | 1550 | 1450 | 800 |  |  |
|  | **50650** | ● | ● | ● | **50650** |  |  |
|  | 1500 | 1700 | ○ | 1850 | 1150 |  |  |
|  | 1150 | ○ | 1200 | 850 |  |  |  |
|  | 1650 | 1150 | 1150 |  |  |  |  |
|  |  |  |  |  |  |  |  |

可知，此时第3行3列和3行7列的单元格评分最高，所以AI会在两个位置上随机选取一个，不管AI选择哪个位置，都将有效的阻止黑棋的进攻，所以说，目前AI已经具备了攻击和防守的能力。

**3.2关键功能实现**

**3.2.1 窗口布局**

窗口布局由View类来控制，主要分为两部分：顶部的工具条、中央的棋盘面板。工具条有两个Action：重开一局、玩家先手，分别设置棋盘初始化以及哪位玩家先手，窗口布局如图8所示。

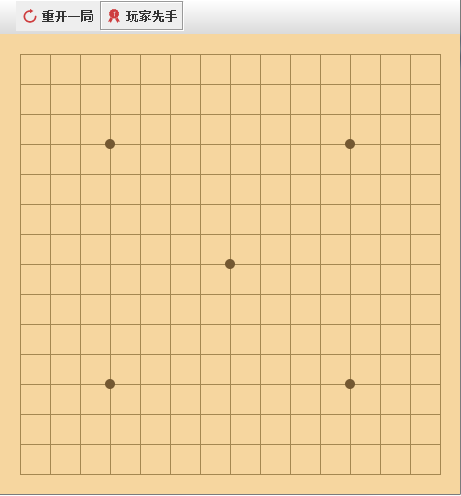


图8.窗口布局图

**1）添加工具条以及棋盘**

//初始化棋盘面板以及添加

chessPanel = new ChessPanel();

frame.add(chessPanel);

// 顶部工具栏

JToolBar bar = new JToolBar();//创建工具栏

frame.add(bar, BorderLayout.NORTH);//添加

bar.setBorderPainted(false);//设置工具栏不画边框

**2）添加Action至工具条**

Icon icon = new ImageIcon(View.class.getResource("/image/restart.png"));//Icon

JButton restartAction = new JButton("重开一局", icon);//Action

restartAction.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

restartBoard();//重开棋局

}

});

bar.add(restartAction);//添加Action

**3）鼠标事件**

//为棋盘面板设置鼠标监听事件

chessPanel.addMouseListener(new MouseAdapter() {

@Override

public void mouseClicked(MouseEvent e) {

showChess(chessPanel, e);

}

});

在showChess方法中我们要执行的是以下几件事情：

根据鼠标点击位置计算出棋盘上的行列值

调用chess对象使玩家在逻辑上落子，调用chessPanel使玩家在视图上落子

chess对棋盘扫描判定玩家是否获胜，如果获胜则提示并重开棋局，如果未获胜，继续执行

chess对棋盘扫描并对每个可行的空位评分，选取一个得分最高的位置，调用chess对象使AI在逻辑上落子，调用chessPanel使AI在视图上落子

chess对棋盘扫描判定AI是否获胜，如果获胜则提示并重开棋局，如果未获胜，继续执行

之后继续由玩家点击鼠标，回到第一步。

**3.2.2控制器的实现**

控制器是整个程序最为核心的一个类，它有多达15个方法，实现了对整个棋局逻辑的控制以及分析。

以3.1的实现原理为依据，AI分析局势先调用getAllMayLocation()方法获取可行的位置集合，然后对每个空位调用getScore(int x, int y)方法打分，打分的时候需要考虑四个方向以及自己、对方两个角色，打分的步骤是先模拟落子，然后以该点为中心进行四个方向上的扫描，获取每个方向上的棋型，然后调用

getScoreBySituation(int count1, int leftStatus, int rightStatus)方法打分，这个方法就是评分表，它对应着3.1中实现原理的评分表。评分表的好与坏直接关系着AI的智能性，这里我只是大概给了评分表的值，细节没有再调。

以下给出一个局势评估的函数，更多的代码见附录。

//局势评估函数，评估该点的得分

public int getScore(int x, int y) {

//使用换位思考思想

//以己方棋子和对方棋子模拟落子计算分数和

int xScore = getXScore(x, y, 1) + getXScore(x, y, 2);

int yScore = getYScore(x, y, 1) + getYScore(x, y, 2);

int skewScore1 = getSkewScore1(x, y, 1) + getSkewScore1(x, y, 2);

int skewScore2 = getSkewScore2(x, y, 1) + getSkewScore2(x, y, 2);

return xScore + yScore + skewScore1 + skewScore2;

}

**3.2.3棋盘的实现**

棋盘的绘制与棋子的绘制都比较简单，为了实现落子操作，我们需要使用一个集合保存棋盘的状态，用Loction保存每个位置的信息，重绘时遍历集合绘制棋子。比较麻烦的一点是绘制棋子上边的数字，这个数字是用来记录棋子是第几手下的，棋盘上有黑棋、白棋，那么棋子上的数字颜色也应该是两种，以下代码中string就是要画的字符串，核心代码如下：

FontMetrics metrics=g.getFontMetrics();

int ascent = metrics.getAscent();

int descent = metrics.getDescent();

if(location.getPlayer()==Chess.first) g.setColor(Color.white);

else g.setColor(Color.black); //设置棋子颜色

g.drawString(string,margin + location.getY() \* row-metrics.stringWidth(string)/2,margin + location.getX() \* row-(ascent+descent)/2+ascent);

**3.2.4位置实体类实现**

这个类是个实体类，只有四个属性以及setter、getter方法以及两参、三参、四参构造方法，封装的是一个位置实体类。

**4.运行结果**

**4.1效果图**

人获胜如图9所示，机器获胜如图10所示。

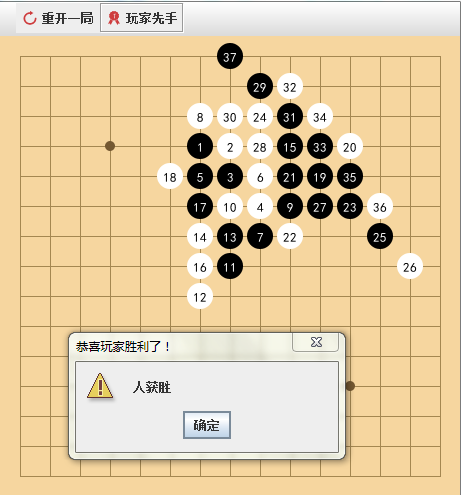
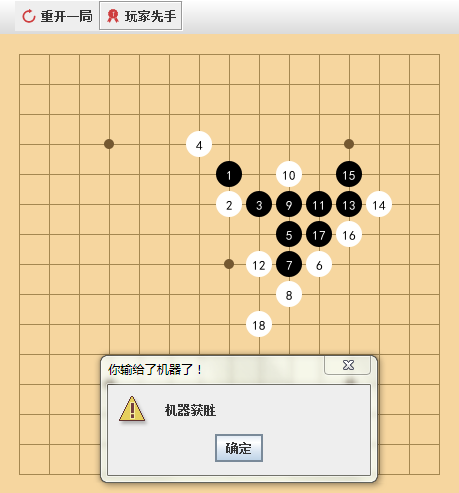
 

图9.人获胜 图10.机器获胜

**4.2问题分析**

出现的问题主要有三个：

问题一：评分表的值怎么设定

问题二：机器先手这个功能怎么实现

问题三：棋子上的数字字符串怎么居中

对于问题一：我觉得这个是靠经验吧，我先根据各个棋型的重要性给一个大致的分数，然后再慢慢调试，这样的结果可能不是最优的，但是基本可以用了。

对于问题二：机器先手是指机器需要先决策第一手棋的位置，所以Chess需要提供一个start()方法返回第一个位置，因为此时棋盘上没有棋子，所以AI分析后也不会有结果，我就直接把中心的点给返回了作为AI第一步棋的位置。

对于问题三：首先在绘制棋子的时候我们可以知道棋子要绘制在第几行第几列，然后根据这个行列值计算出棋子所在的圆心位置，字符串的横坐标=圆心的横坐标-字符串的宽度/2，字符串的高度=圆心的纵坐标-字符串的高度/2。

横坐标=圆心横坐标-metrics.stringWidth(string)/2

纵坐标=圆心纵坐标-(ascent+descent)/2+ascent

**5.总结与致谢**

到目前，课程设计彻底实现完成，在论文中，因为篇幅原因并没有写清楚功能的实现代码，但是原理都说清楚了，全部的代码见附录。上一年就想实现这个五子棋，但是尝试了一段时间不知道怎么让机器能够分析，以前我是通过一种情况一种情况的判断，最后写了六七百行连封堵的能力都没达到，这次使用的是评分的方法，比较简单并且棋力还算可以。

网上关于五子棋AI的资料我也看的不少，基本都是生成一棵固定深度的博弈树，博弈树的偶数层代表先手的玩家的决策行为，奇数层代表另一方玩家的决策行为，使用极大极小搜索算法对几步之内的局势遍历，每条路径都是一个局势，每个结点都会有一个评分，不管对手怎么下棋，AI只需要尽可能的往得分高的那条路径上就可以了，但是棋盘上有225个交叉点，仅仅第二层就会有225个分叉，第三层有225\*2个分叉，不得不说计算量太大了，对AI来说，仅仅预测三步的计算量都是非常大的。所以要用到alpha-β剪枝提高搜索效率。

我用到的不是这个方法，本论文提到的方式只是对空位评分，仅仅思考一步，并且因为在分析局势的时候并没有对哪一方有偏袒（双方的权值都为1），所以AI挑选的位置可能对自己来说不是最优的，但对双方来说都是最优的。如果想让AI棋力增加，可以让它多思考一会，当AI决策出的位置有多个时，不要随机选择一个，可以让AI模拟对手落子，然后AI再模拟落子，最后计算几步之后的总得分，得分高者说明几步之后这个位置还是有优势的。

最后还要谢谢智慧来老师，教的很认真，并且能够严格的要求我们，这对我们以后是有帮助的，谢谢老师。

**6.参考文献**

[1] 李刚，疯狂Java讲义第3版，北京：电子工业出版社，2014.7。

[2] 郑莉，Java语言程序设计第2版，北京：清华大学出版社，2011.6。

**7.附录：全部代码**

**7.1 Location**

package chessAI;

import java.io.Serializable;

/\*\*

\* 棋子位置实体类

\* 坐标、玩家或者AI、位置分数

\* @author Administrator

\*

\*/

public class Location implements Serializable{

//坐标

private int x;

private int y;

//落子所有者

private int player;

//分数

private int score;

//以下是setter、getter方法

public void setScore(int score) {

this.score = score;

}

public int getScore() {

return score;

}

public int getX() {

return x;

}

public void setX(int x) {

this.x = x;

}

public int getY() {

return y;

}

public void setY(int y) {

this.y = y;

}

public int getPlayer() {

return player;

}

public void setPlayer(int player) {

this.player = player;

}

//以下是2参、3参、4参构造方法

public Location(int x, int y) {

super();

this.x = x;

this.y = y;

}

public Location(int x, int y, int player) {

super();

this.x = x;

this.y = y;

this.player = player;

}

public Location(int x, int y, int player, int score) {

super();

this.x = x;

this.y = y;

this.player = player;

this.score = score;

}

}

**7.2 ChessPanel**

package chessAI;

import java.awt.\*;

import java.util.ArrayList;

import java.util.List;

import javax.swing.JPanel;

/\*\*

\* 棋盘面板

\* 负责棋盘显示、交互、视图上落子

\* @author Administrator

\*/

public class ChessPanel extends JPanel {

private static final long *serialVersionUID* = 1L;

//棋子位置集合

public List<Location> list = new ArrayList<Location>();

//字体

private Font font=new Font("楷体",Font.*PLAIN*,13);

//棋盘单元格的长度

int row = 30;

//棋盘每行第一个点与左侧的距离

int margin=20;

//每行15个落子点

int rowper=15;

//棋子半径

int chessRadius=13;

//棋盘背景颜色

Color bgColor=new Color(246,214,159);

//棋盘线颜色

Color lineColor=new Color(164,135,81);

//星的颜色

Color pointColor=new Color(116,88,49);

@Override

public void paint(Graphics g1) {

super.paint(g1);

Graphics2D g=(Graphics2D) g1;

g.setFont(font);

//设置抗锯齿

g.setRenderingHint(RenderingHints.*KEY\_ANTIALIASING* ,RenderingHints.*VALUE\_ANTIALIAS\_ON*);

drawBoard(g);//画棋盘

drawChessman(g);//画棋子

}

/\*\*

\* 画棋盘

\*/

public void drawBoard(Graphics2D g){

//设置背景颜色以及画背景

g.setColor(bgColor);

g.fillRect(0, 0, this.getWidth(), this.getHeight());

//设置线条颜色

g.setColor(lineColor);

//画棋盘线

for (int i = 0; i < rowper; i++) {

g.drawLine(margin, margin + row \* i, this.getWidth() - margin, margin + row \* i);

g.drawLine(margin + row \* i, margin, margin + row \* i, this.getHeight() - margin);

}

//设置颜色以及画棋盘上的点

//10是圆的半径

g.setColor(pointColor);

g.fillOval(margin-5 + 3 \* row, margin-5 + 3 \* row, 10, 10);

g.fillOval(margin-5 + 7 \* row, margin-5 + 7 \* row, 10, 10);

g.fillOval(margin-5 + 3 \* row, margin-5 + 11 \* row, 10, 10);

g.fillOval(margin-5 + 11 \* row, margin-5 + 3 \* row, 10, 10);

g.fillOval(margin-5 + 11 \* row, margin-5 + 11 \* row, 10, 10);

}

/\*\*

\* 画棋子

\* @param g

\*/

public void drawChessman(Graphics2D g){

int i=1;//标志当前进行的手数

//得到FontMetrics对象

//主要为了设置字体居中

FontMetrics metrics=g.getFontMetrics();

int ascent = metrics.getAscent();

int descent = metrics.getDescent();

//遍历棋局绘制棋子

for (Location location : list) {

if (location.getPlayer() == Chess.*first*)

g.setColor(Color.*black*);//设置先手为黑色

else

g.setColor(Color.*white*);//设置后手为白色

//画棋子

g.fillOval(margin-13 + location.getY() \* row, margin-chessRadius + location.getX() \* row, chessRadius\*2, chessRadius\*2);

//画棋子上的数字

if(location.getPlayer()==Chess.*first*) g.setColor(Color.*white*);

else g.setColor(Color.*black*);

String string=i+"";

//计算字符串应在的坐标

g.drawString(string,margin + location.getY() \* row-metrics.stringWidth(string)/2,margin + location.getX() \* row-(ascent+descent)/2+ascent);

i++;

}

}

/\*\*

\* 清除棋盘

\*/

public void clearBoard() {

list.clear();

repaint();

}

/\*\*

\* 视图上落子

\*/

public void doPlay(int row, int col, int player) {

list.add(new Location(row, col, player));

repaint();

}

}

**7.3 Chess**

package chessAI;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* 棋局控制器

\* 负责玩家双方逻辑上落子、判断胜负、AI分析

\* @author Administrator

\*

\*/

public class Chess {

// 棋盘

private int[][] chess;

// 玩家

public static final int PLAYER = 2;

// AI

public static final int AI = 1;

// 先手

public static int first = PLAYER;

/\*\*

\* 设置先手

\*

\* @param first

\*/

public void setFirst(int first) {

this.first = first;

}

/\*\*

\* 构造方法，初始化棋盘

\*/

public Chess() {

chess = new int[15][15];

restart();//初始化

}

/\*\*

\* 棋局初始化

\*/

public void restart() {

for (int i = 0; i < chess.length; i++)

for (int j = 0; j < chess.length; j++)

chess[i][j] = 0;

}

/\*\*

\* AI先手时获取第一手落子位置

\*/

public Location start() {

chess[7][7] = first;

return new Location(7, 7);

}

/\*\*

\* 玩家落子(AI、人落子都是该方法)

\* @param x 横坐标 0-14

\* @param y 纵坐标 0-14

\* @param player 落子所有者，静态常量

\* @return 是否落子成功

\*/

public boolean play(int x, int y, int player) {

//边界判断

if (x < 0 || x >= chess.length)

return false;

if (y < 0 || y >= chess.length)

return false;

//非空判断，0为空位

if (chess[x][y] != 0)

return false;

//校验通过，落子

chess[x][y] = player;

return true;

}

/\*\*

\* 将棋盘结果显示在控制台上

\*/

public void showToConsole() {

for (int i = 0; i < chess.length; i++) {

for (int j = 0; j < chess.length; j++)

System.out.print(chess[i][j] + " ");

System.out.println();

}

}

/\*\*

\* 将点添加到集合中，过滤掉重复的位置

\* @param allMayLocation 目标集合

\* @param x 坐标x

\* @param y 坐标y

\*/

private void addToList(List<Location> allMayLocation, int x, int y) {

int sign = 0;

for (Location location : allMayLocation)

if (location.getX() == x && location.getY() == y) {

sign = 1;

break;

}

if (sign == 0) allMayLocation.add(new Location(x, y));

}

/\*\*

\* AI对棋盘分析，控制器的核心方法

\* 本算法计算出的位置对双方都是有益的

\* @return

\*/

public Location explore() {

// 得到可行位置的集合

List<Location> allMayLocation = getAllMayLocation();

// 所有得分最大且相同的位置

//打分时可能存在分数相同的位置，将这个位置保存起来随机落子

List<Location> allMaxLocation = new ArrayList<Location>();

// 对每个可落子的空位计算分数

int max = 0;//最大分数

//遍历可行位置集合

for (Location location : allMayLocation) {

//计算位置得分并设置位置分数

int score = getScore(location.getX(), location.getY());

location.setScore(score);

//判断

if (score > max) max = score;

//如果socre是当前最大值且不是0

//如果allMaxLocation集合中第一个元素值小于max

//先清空，然后再添加该位置

//否则，直接添加该位置

if (max != 0 && score == max) {

if (allMaxLocation.size() > 0)

if (allMaxLocation.get(0).getScore() < max)

allMaxLocation.clear();

allMaxLocation.add(location);

}

System.out.println("x=" + location.getX() + " y=" + location.getY() + " score=" + score);

}

//从最高分集合中随机抽取一个位置

Location pos = allMaxLocation.get((int) (Math.random() \* allMaxLocation.size()));

System.out.println("机器落子:行：" + (pos.getX() + 1) + " 列:" + (pos.getY() + 1));

//返回分析的位置

return new Location(pos.getX(), pos.getY());

}

/\*\*

\* 获取可行位置集合

\* 对每个非空位置，将其四周的位置添加到集合中

\* 注意去掉重复的位置

\* @return 位置集合

\*/

private List<Location> getAllMayLocation() {

List<Location> allMayLocation = new ArrayList<Location>();

// 搜索棋盘获取可行棋的点

for (int i = 0; i < chess.length; i++)

for (int j = 0; j < chess.length; j++) {

if (chess[i][j] != 0) {

if (j != 0 && chess[i][j - 1] == 0)

addToList(allMayLocation, i, j - 1);

if (j != (chess.length - 1) && chess[i][j + 1] == 0)

addToList(allMayLocation, i, j + 1);

if (i != 0 && j != 0 && chess[i - 1][j - 1] == 0)

addToList(allMayLocation, i - 1, j - 1);

if (i != 0 && chess[i - 1][j] == 0)

addToList(allMayLocation, i - 1, j);

if (i != 0 && j != (chess.length - 1) && chess[i - 1][j + 1] == 0)

addToList(allMayLocation, i - 1, j + 1);

if (i != (chess.length - 1) && j != 0 && chess[i + 1][j - 1] == 0)

addToList(allMayLocation, i + 1, j - 1);

if (i != (chess.length - 1) && chess[i + 1][j] == 0)

addToList(allMayLocation, i + 1, j);

if (i != (chess.length - 1) && j != (chess.length - 1) && chess[i + 1][j + 1] == 0)

addToList(allMayLocation, i + 1, j + 1);

}

}

return allMayLocation;

}

/\*\*

\* 判断胜负，对棋盘四个方向扫描连子数

\* @param x 坐标x

\* @param y 坐标y

\* @param cur 判断玩家cur的胜负情况

\* @return 是否胜利

\*/

public boolean isWin(int x, int y, int cur) {

// 四个方向上的连子数

int count1 = 0, count2 = 0, count3 = 0, count4 = 0;

int i, j;

//横向扫描

for (j = y; j < chess.length; j++) {

if (chess[x][j] == cur)

count1++;

else

break;

}

for (j = y - 1; j >= 0; j--) {

if (chess[x][j] == cur)

count1++;

else

break;

}

if (count1 >= 5)

return true;

//纵向扫描

for (i = x; i < chess.length; i++) {

if (chess[i][y] == cur)

count2++;

else

break;

}

for (i = x - 1; i >= 0; i--) {

if (chess[i][y] == cur)

count2++;

else

break;

}

if (count2 >= 5)

return true;

//正斜向扫描

for (i = x, j = y; i < chess.length && j < chess.length; i++, j++) {

if (chess[i][j] == cur)

count3++;

else

break;

}

for (i = x - 1, j = y - 1; i >= 0 && j >= 0; i--, j--) {

if (chess[i][j] == cur)

count3++;

else

break;

}

if (count3 >= 5)

return true;

//反斜向扫描

for (i = x, j = y; i < chess.length && j >= 0; i++, j--) {

if (chess[i][j] == cur)

count4++;

else

break;

}

for (i = x - 1, j = y + 1; i >= 0 && j < chess.length; i--, j++) {

if (chess[i][j] == cur)

count4++;

else

break;

}

if (count4 >= 5)

return true;

return false;

}

/\*\*

\* 局势评估函数

\* 评估该点的得分

\* @param x 坐标x

\* @param y 坐标y

\* @return 分数

\*/

public int getScore(int x, int y) {

//使用换位思考思想

//以己方棋子和对方棋子模拟落子计算分数和

int xScore = getXScore(x, y, 1) + getXScore(x, y, 2);

int yScore = getYScore(x, y, 1) + getYScore(x, y, 2);

int skewScore1 = getSkewScore1(x, y, 1) + getSkewScore1(x, y, 2);

int skewScore2 = getSkewScore2(x, y, 1) + getSkewScore2(x, y, 2);

// int xScore = getXScore(x, y, 1);

// int yScore = getYScore(x, y, 1);

// int skewScore1 = getSkewScore1(x, y, 1);

// int skewScore2 = getSkewScore2(x, y, 1);

return xScore + yScore + skewScore1 + skewScore2;

}

/\*\*

\* 根据棋型计算得分

\* @param count1 连子个数

\* @param leftStatus 左侧封堵情况 1:空位，2：对方或墙

\* @param rightStatus 右侧封堵情况 1:空位，2：对方或墙

\* @return 分数

\*/

private int getScoreBySituation(int count1, int leftStatus, int rightStatus) {

int score = 0;

// 五子情况

if (count1 >= 5)

score += 200000;// 赢了

// 四子情况

if (count1 == 4) {

if (leftStatus == 1 && rightStatus == 1)

score += 50000;

if ((leftStatus == 2 && rightStatus == 1) || (leftStatus == 1 && rightStatus == 2))

score += 3000;

if (leftStatus == 2 && rightStatus == 2)

score += 1000;

}

//三子情况

if (count1 == 3) {

if (leftStatus == 1 && rightStatus == 1)

score += 3000;

if ((leftStatus == 2 && rightStatus == 1) || (leftStatus == 1 && rightStatus == 2))

score += 1000;

if (leftStatus == 2 && rightStatus == 2)

score += 500;

}

//二子情况

if (count1 == 2) {

if (leftStatus == 1 && rightStatus == 1)

score += 500;

if ((leftStatus == 2 && rightStatus == 1) || (leftStatus == 1 && rightStatus == 2))

score += 200;

if (leftStatus == 2 && rightStatus == 2)

score += 100;

}

//一子情况

if (count1 == 1) {

if (leftStatus == 1 && rightStatus == 1)

score += 100;

if ((leftStatus == 2 && rightStatus == 1) || (leftStatus == 1 && rightStatus == 2))

score += 50;

if (leftStatus == 2 && rightStatus == 2)

score += 30;

}

return score;

}

/\*\*

\* 获取该空位在横向上的得分

\* @param x 位置横坐标

\* @param y 位置纵坐标

\* @return 评分

\*/

public int getXScore(int x, int y, int cur) {

int other;// 对方棋子

if (cur == 1) other = 2;

else other = 1;

// 模拟落子

chess[x][y] = cur;

//左侧、右侧的状态，用来记录棋型

int leftStatus = 0;

int rightStatus = 0;

int j,count1 = 0;//count1是相连棋子个数

//扫描记录棋型

for (j = y; j < chess.length; j++) {

if (chess[x][j] == cur)

count1++;

else {

if (chess[x][j] == 0)

rightStatus = 1;// 右侧为空

if (chess[x][j] == other)

rightStatus = 2;// 右侧被对方堵住

break;

}

}

for (j = y - 1; j >= 0; j--) {

if (chess[x][j] == cur)

count1++;

else {

if (chess[x][j] == 0)

leftStatus = 1;// 左侧为空

if (chess[x][j] == other)

leftStatus = 2;// 左侧被对方堵住

break;

}

}

// 恢复

chess[x][y] = 0;

//根据棋型计算空位分数

return getScoreBySituation(count1, leftStatus, rightStatus);

}

/\*\*

\* 获取该点在纵向上的得分

\* @param x

\* @param y

\*/

public int getYScore(int x, int y, int cur) {

int other;// 对方棋子

if (cur == 1) other = 2;

else other = 1;

// 模拟落子

chess[x][y] = cur;

//左侧、右侧的状态，用来记录棋型

int topStatus = 0;

int bottomStatus = 0;

int i,count1 = 0;

//扫描记录棋型

for (i = x; i < chess.length; i++) {

if (chess[i][y] == cur)

count1++;

else {

if (chess[i][y] == 0)

bottomStatus = 1;// 下侧为空

if (chess[i][y] == other)

bottomStatus = 2;// 下侧被对方堵住

break;

}

}

for (i = x - 1; i >= 0; i--) {

if (chess[i][y] == cur)

count1++;

else {

if (chess[i][y] == 0)

topStatus = 1;// 上侧为空

if (chess[i][y] == other)

topStatus = 2;// 上侧被对方堵住

break;

}

}

// 恢复

chess[x][y] = 0;

return getScoreBySituation(count1, topStatus, bottomStatus);

}

/\*\*

\* 正斜向扫描计算得分

\* @param x

\* @param y

\*/

public int getSkewScore1(int x, int y, int cur) {

int other;// 对方棋子

if (cur == 1) other = 2;

else other = 1;

// 模拟落子

chess[x][y] = cur;

// 分数

int score = 0;

int topStatus = 0;

int bottomStatus = 0;

int i, j,count1 = 0;

for (i = x, j = y; i < chess.length && j < chess.length; i++, j++) {

if (chess[i][j] == cur)

count1++;

else {

if (chess[i][j] == 0)

bottomStatus = 1;// 下侧为空

if (chess[i][j] == other)

bottomStatus = 2;// 下侧被对方堵住

break;

}

}

for (i = x - 1, j = y - 1; i >= 0 && j >= 0; i--, j--) {

if (chess[i][j] == cur)

count1++;

else {

if (chess[i][j] == 0)

topStatus = 1;// 上侧为空

if (chess[i][j] == other)

topStatus = 2;// 上侧被对方堵住

break;

}

}

// 恢复

chess[x][y] = 0;

return getScoreBySituation(count1, topStatus, bottomStatus);

}

/\*\*

\*

\* 斜线：从右上到左下

\* 反斜向扫描

\* @param x

\* @param y

\*/

public int getSkewScore2(int x, int y, int cur) {

int other;// 对方棋子

if (cur == 1)

other = 2;

else

other = 1;

// 模拟落子

chess[x][y] = cur;

// 分数

int score = 0;

int topStatus = 0;

int bottomStatus = 0;

int i, j;

// 从右上到左下

int count1 = 0;

for (i = x, j = y; i < chess.length && j >= 0; i++, j--) {

if (chess[i][j] == cur)

count1++;

else {

if (chess[i][j] == 0)

bottomStatus = 1;// 下侧为空

if (chess[i][j] == other)

bottomStatus = 2;// 下侧被对方堵住

break;

}

}

for (i = x - 1, j = y + 1; i >= 0 && j < chess.length; i--, j++) {

if (chess[i][j] == cur)

count1++;

else {

if (chess[i][j] == 0)

topStatus = 1;// 上侧为空

if (chess[i][j] == other)

topStatus = 2;// 上侧被对方堵住

break;

}

}

// 恢复

chess[x][y] = 0;

return getScoreBySituation(count1, topStatus, bottomStatus);

}

}

**7.4 View**

package chessAI;

import java.awt.BorderLayout;

import java.awt.event.\*;

import javax.swing.\*;

/\*\*

\* 五子棋视图类

\* 负责棋盘显示以及与用户交互

\* @author Administrator 刘壮飞

\*

\*/

public class View {

//窗口对象

JFrame frame;

//棋盘控制器对象

Chess chess = new Chess();

//棋盘面板对象

ChessPanel chessPanel;

/\*\*

\* 创建窗口

\* 绑定事件监听

\*/

public void create() {

//初始化窗口

frame = new JFrame("五子棋AI测试面板");

//初始化棋盘面板以及添加

chessPanel = new ChessPanel();

frame.add(chessPanel);

// 顶部工具栏

JToolBar bar = new JToolBar();//创建工具栏

frame.add(bar, BorderLayout.*NORTH*);//添加

bar.setBorderPainted(false);//设置工具栏不画边框

//第一个工具：重开一局

Icon icon = new ImageIcon(View.class.getResource("/image/restart.png"));//Icon

JButton restartAction = new JButton("重开一局", icon);//Action

restartAction.setToolTipText("重开一局");

restartAction.setOpaque(true);//透明

restartAction.setBorderPainted(false);//去掉边框

restartAction.setFocusPainted(false);//去掉焦点框

restartAction.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

restartBoard();//重开棋局

}

});

bar.add(restartAction);//添加Action

//第二个工具：玩家先手

Icon shouxianIcon = new ImageIcon(View.class.getResource("/image/shouxian.png"));

final JButton firstAction = new JButton("玩家先手", shouxianIcon);

firstAction.setOpaque(true);

firstAction.setBorderPainted(true);

firstAction.setFocusPainted(false);

firstAction.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

if(firstAction.getText().equals("玩家先手")){

firstAction.setText("机器先手");

//使用棋局控制器设置先手

Chess.*first*=chess.*AI*;

}else{

firstAction.setText("玩家先手");

//使用棋局控制器设置先手

Chess.*first*=chess.*PLAYER*;

}

}

});

bar.add(firstAction);

//为棋盘面板设置鼠标监听事件

chessPanel.addMouseListener(new MouseAdapter() {

@Override

public void mouseClicked(MouseEvent e) {

showChess(chessPanel, e);

}

});

//设置frame的相关属性

frame.setSize(476, 532);

frame.setLocationRelativeTo(null);

frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

frame.setVisible(true);

}

/\*\*

\* 棋局重开事件处理函数

\*/

public void restartBoard(){

chess.restart();//棋盘控制器初始化棋盘

chessPanel.clearBoard();//棋盘view清除棋子重绘

if(Chess.*first*==Chess.*AI*){

//如果AI先手，AI需要先落子

Location location=chess.start();

chess.play(location.getX(), location.getY(),Chess.*AI*);

//棋盘面板控制落子的显示

chessPanel.doPlay(location.getX(),location.getY(), Chess.*AI*);

}

}

/\*\*

\* 棋盘面板的鼠标点击事件

\*/

public void showChess(ChessPanel chessPanel, MouseEvent e) {

//点击的位置坐标

int x = e.getX();

int y = e.getY();

//转化为棋盘上的行列值

int col = (x - 5) / 30;

int row = (y - 5) / 30;

//玩家落子有效

boolean isEnable = chess.play(row, col,Chess.*PLAYER*);

if (isEnable) {

// 棋盘面板绘制棋子

chessPanel.doPlay(row, col, Chess.*PLAYER*);

//玩家胜利

if (chess.isWin(row, col, Chess.*PLAYER*)){

JOptionPane.*showMessageDialog*(frame, "人获胜", "恭喜玩家胜利了！",JOptionPane.*WARNING\_MESSAGE*);

restartBoard();//初始化棋盘

return;

}

//棋局控制器分析后获取落子位置

Location result = chess.explore();

//棋盘控制器控制AI落子

chess.play(result.getX(), result.getY(),Chess.*AI*);

// 棋盘面板绘制棋子

chessPanel.doPlay(result.getX(), result.getY(), Chess.*AI*);

//AI胜利

if (chess.isWin(result.getX(), result.getY(),Chess.*AI*)){

JOptionPane.*showMessageDialog*(frame, "机器获胜", "你输给了机器了！",JOptionPane.*WARNING\_MESSAGE*);

restartBoard();

return;

}

} else System.*out*.println("坐标无效!");

}

}

**7.5 Test测试类**

package chessAI;

/\*\*

\* 测试类

\*/

public class Test {

public static void main(String[] args) {

View view = new View();

view.create();

}

}