摘 要

在人们文化与物质生活逐渐丰富的今天，老百姓对于工作学习之余的娱乐需求越来越高。五子棋作为起源于中国民间，如今仍然妙趣横生的智力运动，老少皆宜，很适合作为人们闲暇时间游乐的活动。笔者也是选择了五子棋作为本次课程设计的项目。

文章探讨了笔者对于这款Java网络五子棋的设计和期望，阐述了程序的功能分块、运行逻辑、各功能块之间的关系、各功能块的实现细节和开发中遇到的问题及解决方案。并在文章最后评价程序，提出了对程序未来的展望。

关键字：Java；五子棋；实现细节；解决方案

Abstract

With the gradual enrichment of people's cultural and material life, our demand for entertainment is getting higher and higher. Gobang, an intellectual sport originated from Chinese folk life, which is still interesting today, is suitable for all ages. It is a good choice for people to spare their leisure time. The author also chose Gobang as the project of this course project.

This paper discusses the author's design and expectation for this Java network Gobang, and expounds the function block, the logical contents of the program, the relationship between each function block, the implementation details of each function block, the problems and solutions encountered in the development. Finally, the paper evaluates the program and puts forward the prospect of the program in the future.

Key words: Java; Gobang; Implementation details; Solutions

目 录

第一章 引言

五子棋是一项为广大人民群众所喜爱的智力运动，也是人们闲暇时刻交友娱乐的一个良好媒介。可是五子棋双方必须面对面才能游玩，棋具不便携也不便宜等问题，为这项娱乐带来了一定的门槛。为了解决这些问题，作者用Java语言实现了一款可联机的网络五子棋，既方便人们随时随地游玩，也磨练了自己的代码水平。

本文描述了这款五子棋游戏的特性和功能，记录了游戏的分工方式、运行逻辑和实现方法，并展示了如何使用Java的特性来方便游戏制作。过程中包含很多开发时遇到的bug以及解决方案，灵感来源。本五子棋游戏基于Eclipse实现。

第二章 功能设计

2.1 基本功能

实现网络双方的五子棋对弈，具有基本的重复落子检测，落子顺序检测，五子连珠（胜利）检测、棋局重开和输赢提示。

2.2 额外特性

本网络五子棋在五子棋基本功能实现的基础上添加了游戏主界面，支持棋手在游戏结束后查看已保存的历史对局，对棋局的每一步落子进行复盘。在对局中添加聊天，悔棋、和棋和认输功能，给每位棋手增加了落子时间控制（倒计时），并支持棋手在游戏结束后再来一局。

2.3 预期效果

游戏能够稳定运行，没有损害体验的游戏bug。棋手可以在游戏中磨练棋艺，网络交流；在棋局后多次复盘，永久保存。实现一个可游玩的，有一定游玩体验的五子棋小游戏。

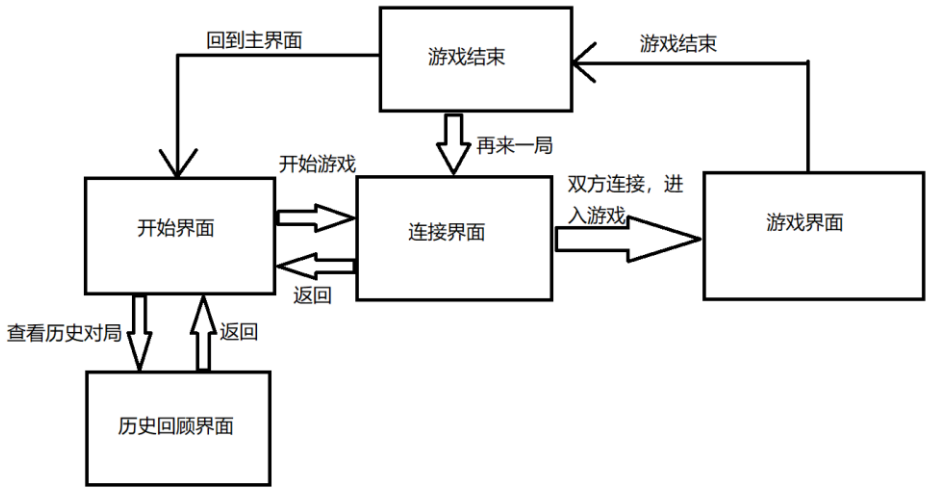
第三章 游戏流程设计

3.1 游戏界面

为了实现游戏功能，本游戏总共包含开始界面、连接界面、游戏界面和历史回顾界面四个内容板块，根据玩家的不同需求而进入不同板块。

3.2 游戏流程逻辑

在进入游戏后，玩家首先进入开始界面，并可以选择开始游戏、查看历史棋局或退出游戏三个选项。若玩家选择查看历史棋局，将进入历史回顾界面，界面内玩家可以查看自己下的所有棋局以及棋局中的每一步棋子，再点击返回即可回到主界面。若玩家选择开始游戏，玩家将进入连接界面，选择开房间/进入房间。等待对手连接后即可进入到游戏界面，正式开始下棋。一局棋局结束后，玩家可以选择回到主界面或再来一局。游戏流程如下图所示。



3.3 界面的切换方式

游戏包含一个JFrame窗口作为顶层容器，下设开始界面、连接界面、游戏界面和历史回顾界面四个JPanel，借助CardLayout布局和PanelSwitcher类的辅助来切

换JFrame的内容面板，依此实现不同界面之间的切换。这样做避免了大窗口的切换，方便最终关闭游戏并占用了更少的系统资源。

第四章 游戏框架设计

4.1 基本思想

Java五子棋作为一个较大的编程项目， 编写时如果不加注意很可能造成写出的代码耦合度很高，可读性、扩展性差。为了解决这个问题，游戏逻辑设计借鉴大型公司的管理办法，采取“专人专职”的策略，为每个分工模块编写专门的代码。这样做虽然可能牺牲了少量程序的性能，但是使得程序的可读性、扩展性、可维护性和测试性都大大提高，减少了编写程序的难度，使得程序逻辑清晰明了。

4.2 程序模块

根据上述分工思想，作者在编写程序时设定了如下模块，每个模块负责特定的功能，自己负责的功能之外一律不管，交给别的模块解决。接下来以下棋模块为例对程序分块。

4.2.1 游戏模型模块

游戏模型模块负责五子棋游戏基本逻辑的实现，即下棋、悔棋、保存当前棋盘上的棋子和判断输赢等功能。游戏模型用静态常量记录棋盘宽度和棋盘上的黑子、白子和空白三种状态，并在其内置的19 * 19数组中抽象记录了当前棋盘。下图为程序内置的静态变量。

```
public static final int BLACK = 1;
public static final int WHITE = -1;
public static final int SPACE = 0;
public static final int WIDTH = 19;
```

4.2.2 棋盘模块

棋盘模块继承自JPanel，负责棋局显示、棋手聊天和可视化下棋命令（判断用户所点击的棋盘位置或和棋、悔棋等等操作按钮）的获取。

4.2.3 网络助手模块

作为一款联机五子棋，网络连接是必不可少的。网络助手专门负责网络连接，内置多个方法，实现了和网络相关的各种操作，例如主机和房客的连接、下棋和悔棋等命令的发送等等功能均由网络助手来实现。

4.2.4 主控模块

上述三个模块中主要是编写了各种各样的方法，将底层实现抽象在了public的方法中，而这些方法均交给主控模块来调用。主控模块是控制程序运行的核心，知道程序应该怎样运行，并指挥程序在合适的时机作出合适的反应。模块内部代码量不大，通过接收来自网络助手或其他模块的消息后去调用对应模块中的方法工作，是程序的核心模块。

4.3 游戏部分总体实现

程序的实现以及运行依托于以上几个部分组成的控制体系，主控模块抽象地调用其他模块中地功能，其他模块负责具体实现。类似地，这套体系还同样应用于历史对战记录查看模块，模块的具体剖析在下一章给出。

第五章 游戏模块功能

5.1 模型模块

五子棋模型的实现是游戏的第一步，是游戏抽象的内核。建立五子棋模型时，我们只关心在内部实现游戏应有的元素、规则和操作，如棋盘、棋子的建立和保存、输赢的判断、下棋和悔棋操作的实现等等，然后为检测棋盘和玩家胜负等外部模块需要的信息保留接口即可。上一章已经说到，模块内定义了int常量，我们用1表示黑子，用-1表示白子，用0表示空格，以下直接用中文代替int值进行叙述。

为了实现下棋操作，作者编写了一个叫做InputIsLegal的方法来检验输入合法性：若输入坐标在棋盘外/对应位置上已经有棋子则返回false，否则返回true。接下来编写了一个WinOrNot方法，这个方法的作用是对特定坐标的棋子进行纵向、横向和斜向的检测，看是否完成了五子连珠。若完成，返回连珠颜色，否则返回空白。至此，只要在每次下棋时依次调用这两个方法，就可以完成下棋操作并检出胜负。

接下来是悔棋操作，考虑到游戏流程中添加了计时器元素（某一方玩家如果超过一分钟没有下棋，则视为放弃本次落子，将棋权交给对方），悔棋不能是简单的悔一步棋或检测颜色。由此，作者设计了一个递归调用自身的悔棋方法，方法如下：

```
public void Retrack(int RetrackColor) {
    int i = Vars.historypanel.Gettop().size() - 1;
    if(i == -1){
        return;
    }
    ChessPoint temp = Vars.historypanel.Gettop().remove(i);
    if(temp.color != RetrackColor){
        Retrack(RetrackColor);
    }
    ChessBoard[temp.row][temp.col] = SPACE;
}
```

这个方法会拿走上一步下的棋子（无论是己方还是对方），接着检测这个拿走的棋子是否为要悔棋的颜色，若不是则递归调用自身，直到要悔棋的颜色

悔棋成功或棋盘上没有棋子为止。

棋局的保存使用了一个LinkedList来实现，也只有LinkedList可以实现方便的存盘和读取操作。

功能完备之后，我们可以打印这个抽象的棋盘来验证其功能，图中&代表白棋，#代表黑棋，-代表空格。在Eclipse的Console窗口输出如下（注：此处仅为了测试，将棋盘大小设置为15*15，真实棋盘大小应为19*19）。

```
轮到白方下棋，请输入下棋坐标x y:
1 5
- - - - - - - - - - - - - - -
- - - - - & - - - - - - - - -
- - - - - - - - - - - - - - -
- - # - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -
```

可见棋盘被成功创建，类似地在其中也可检测如悔棋、胜负判断等功能，这些功能在完整程序后也可检测，故此处不予列出。

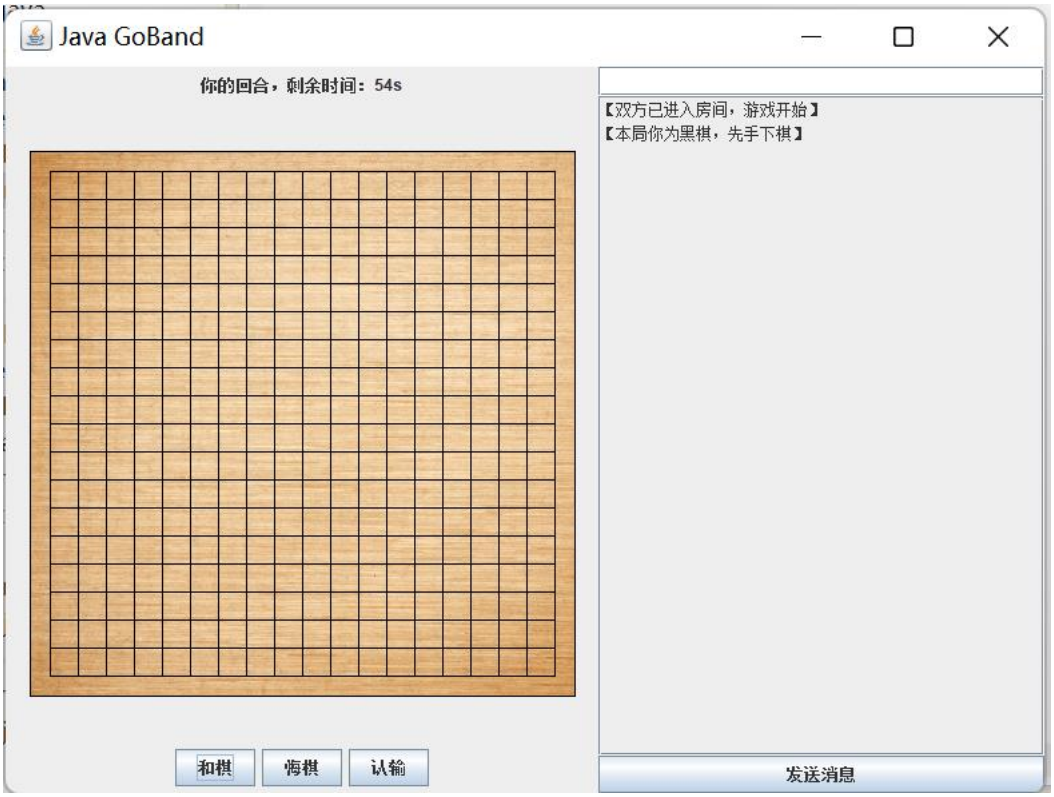
5.2 棋盘模块

作为一款可视化的五子棋游戏，棋盘的实现就是可视化过程最重要的一步。棋盘类继承自JPanel，和“开始面板”，“网络连接面板”一样是顶层容器CardLayout中的一个面板。

考虑棋盘的布局。棋盘模块要包含以下部分：

1. 棋盘显示
2. 聊天框和输入框
3. 认输、和棋、悔棋按钮
4. 计时器

布局设计如下：



5.2.1 棋盘面板

棋盘面板的布局是棋盘类编写中比较困难的一步。作者在编写程序时想通过代码去模拟一个真的棋盘，且不希望这个棋盘布满整个屏幕。于是定义如下变量：

```
public class ChessPanel extends JPanel{
    int sx=30,
        sy=30,
        unit=50,
        gap=30,
        edge = 3*unit/4;
```

sx和sy代表棋盘左上角的坐标，unit代表棋盘中每条线的间隔，gap代表棋盘边缘和屏幕边界的最小距离，edge代表棋盘周围一圈没有横竖交错线的区域的宽度。这样一来棋盘就可以在小窗口中被完整的画出。但是当我们调整屏幕大小时，会发现棋盘还是会不可避免地被遮住，于是我们采取另一个方法，为窗口大小增加Listener，在窗口大小被改变时通过计算调整上述参数，保证棋盘被完整地画出。此处放监听器中componengtResized的代码：

```
public void componentResized(ComponentEvent e) {
    super.componentResized(e);
    int wid = getWidth();
    int hei = getHeight();
    int min = wid<hei?wid:hei;
    unit = (min-gap*2)/(GameModel.WIDTH-1);
    sx = (wid-unit*(GameModel.WIDTH-1))/2;
    sy = (hei-unit*(GameModel.WIDTH-1))/2;
    edge = 3*unit/4;
    repaint();
}
```

代码首先求出屏幕大小变化后的宽高最小值，以此为依据求出居中棋盘的宽度高度，重绘页面，解决问题。尺寸问题解决后，我们来看看棋盘的绘制过程。

棋盘的背景使用了一张jpg图片，绘画方法是将其转为一个Image对象后，使用Graphics的drawImage方法绘制到背景中。棋盘的线条直接在for循环中嵌套Graphics的drawLine方法来绘制，四条边缘线也直接在棋盘之外照背景图片的尺寸画出即可。棋子绘画时，只要检测到模型模块中对应位置有棋子，就根据棋子颜色使用Graphics的fillOval方法绘制，其它同理。值得一提的是，在画棋子时可以在棋子左上角1/5处画一个半径为棋1/6的小圆圈以模拟棋子的反光，使画面更为美观。

功能方面，棋盘面板应该可以检测用户鼠标点击位置，根据用户鼠标的点击位置确定用户想在棋盘中下棋的位置。首先添加鼠标监听者，在其内的MouseAdapter中重写mouseReleased方法。由用户点击的位置判断下棋位置的公式如下：

```
int row = (e.getY() - sy + unit/2)/unit;
int col = (e.getX() - sx + unit/2)/unit;
```

考虑到棋盘的下棋是在横竖线交叉点完成而非横竖线构成的框内，所以算法中的

$$+\frac{unit}{2}$$

可以理解为“将检测区域整体向左上方平移半格”，这样就可以将棋盘与下棋检测的位置错开，使得下棋位置被正确检测。

5.2.2 聊天面板

聊天面板整体分为上中下三部分，分别为消息输入栏（JTextField）、历史消息表（TextArea）和消息发送按钮（JButton）。

聊天面板的实现很简单：消息输入栏允许编辑，输入要发送的内容；发送按钮添加监听者，点击后通知网络助手发送当前内容并清空消息输入栏；历史消息用于

记录自己发出的消息、对方发出的消息以及系统提示。当网络助手接收到对方发来的信息时，会调用一个public接口将信息传递给聊天面板，聊天面板再将信息放到历史消息表中以达到显示对方信息的目的。

5.2.3 计时器

本游戏的Java计时器以JLabel的文本方式显示于棋盘上方，自己下棋时会显示一个六十秒的倒计时，在对方下棋时显示“对手下棋中”。计时器的实现依托于Java自带的Timer类，使用了类内的schedule方法开启任务。

首先我们定义一个继承自TimerTask的计时类如下：

```
class TimingTask extends TimerTask{
    int time = 60;
    public void run() {
        // TODO Auto-generated method stub
        time--;
        Vars.northpanel.TimeLeast.setText("你的回合，剩余时间：" + time + "s");
        Vars.northpanel.repaint();
    }
}
```

类内有一个值为60的int变量time表示剩余时间，完成的功能是每次执行这个任务（即运行一次run方法）就将时间减1，同时更新屏幕上的标签。使用这个类进行计时的多线程代码如下：

```
new Thread(){
    public void run(){
        timer = new Timer();
        timer.schedule(new TimingTask(), 1000, 1000);
        try {
            Thread.sleep(60000);
        } catch (InterruptedException e) {
            // TODO: handle exception
            e.printStackTrace();
        }
        timer.cancel();
        timer.purge();
        Timing = false;
        TimeLeast.setText("对手下棋中");
        Vars.control.GiveUpChessing();
    }
}.start();
```

此时我们新开一个线程（否则整个游戏都将停止运行），在线程内用schedule定义一个计时任务，后面的参数表示这个任务一秒钟后开始，每秒进行一次。结合上文的任务定义即可知这个计时器每秒将时间减一，并更新屏幕上的标签。如果此时棋手在棋盘上下了棋，这个线程将被停止方法终止，计时也随之停止。下图为停

止方法（由控制模块调用）：

```
public void StopTiming() {
    Timing = false;
    // TODO Auto-generated method stub
    if(timer!=null){
        timer.cancel();
        timer.purge();
    }
    TimeLeast.setText("对手下棋中");
}
```

当然，如果时间用尽而棋手并未下棋，计时器会告诉控制模块本轮己方放弃下棋，然后停止计时。

5.2.4 总体布局

在整个棋盘模块的布局中，棋盘面板和聊天框分别使用了BorderLayout的CENTER和EAST部分，而棋盘面板部分重新添加一个BorderLayout，将计时器、棋盘本身和棋局操作按钮分别置于NORTH、CENTER和SOUTH部分即可。

5.3 网络助手模块

5.3.1 网络连接

网络助手的第一大功能就是实现房主和房客的网络连接。考虑到网络游戏所需要的网络稳定性，本游戏的网络连接采取TCP协议连接主客机。

对于主机的连接，玩家在网络连接界面中输入端口并点击连接按钮后开启主机监听：在新线程中初始化ServerSocket，用语句：

$$Socket\ s = ss.accept();$$

来监听连接上的客机，客机连接上前此线程在此停止。客机连接上后线程继续运行，初始化输入输出流即可对客机开启读线程。读线程是一个在线程内死循环读取网络对方发来的消息的线程。为了易读易用，作者将“开启读线程”操作抽象为了一个专门的方法，方法体如下：

```
protected void startReadThread() {
    new Thread(){
        public void run() {
            while(s != null){
                try {
                    String line = in.readLine();
                    if(line==null){
                        Vars.control.NetAdmitDefeat();
                        return;
                    }
                    parseMessage(line);
                } catch (IOException e) {
                    e.printStackTrace();
                } catch (UnsupportedAudioFileException e) {
                    e.printStackTrace();
                } catch (LineUnavailableException e) {
                    ExitGame();
                    e.printStackTrace();
                }
            }
        }
    }.start();
}
```

在本“读线程”方法中，我们在死循环内不停地读取对方发来的内容，并交给专门的文字处理模块去处理。并添加了对于对手掉线的检测，捕捉异常并进入退出方法。

完成了主机的连接，客机的连接就简单了很多。玩家在网络连接面板中输入网络地址和服务端口号后，网络助手在新线程中用其初始化ip和port，接着初始化输入输出流即可开启读线程，读线程与主机一致，调用相同方法即可，此处不加赘述。值得一提的是，为了解决客机连接时主机没开始服务等问题，我们可以在catch块中捕捉网络连接异常并刷新界面，等待客机的下一次连接。

5.3.2 消息处理

首先考虑一个问题，假如我们只有一条网络连线，却要同时实现聊天、下棋信息传递和双方诸如悔棋的各种信息交互时，应该怎么办？这个问题的答案是增加单次传递的信息量，在原始信息之上增加额外信息，以确保对方收到这条信息时可以判断这条信息包含的内容应该交给哪个模块来处理。本游戏选择的信息增加方式是在原始信息前加上前缀，即传递的每一条信息均为

信息前缀 + *原始信息*

的格式，其中信息前缀表明信息类型（聊天信息，下棋信息等），原始信息表明要传递的内容（可以为空）。所以网络助手收到一条网络信息之后会去检查信息前缀，然后通知对应模块作出反应。此处列举部分：

```
protected void parseMessage(String line)
    if(line.startsWith("PutChess:")){
        parsePutChess(line);
    }
    else if(line.startsWith("Chat:")){
        parseChat(line);
    }
    else if(line.startsWith("DrawAsk")){
        Vars.control.NetDrawRoundAsk();
    }
    else if(line.startsWith("Retract")){
        Vars.control.NetRetractRequest();
    }
```

如上图所示，传递的信息中诸如“收消息”就由信息前缀和原始信息组成，信息前缀就是“Chat:”，原始信息是想要发送的内容。网络助手接收到这个信息后会通知专门处理聊天文字的parseChat来处理这个信息，parseChat方法会去除这条消息的“Chat:”前缀，然后将原始信息传递给聊天面板。而如桶Retract等信息就只有信息前缀“Retract”而没有原始信息。当网络助手识别到对方的悔棋信息后，网络助手会通知控制模块对方要悔棋，把对方的悔棋申请交由控制模块来处理。

5.3.3 网络连接稳定性

很多时候我们希望获得一个稳定的网络连接，不稳定的网络连接带来的不只是游戏体验的下滑，有时候会直接导致程序无法继续运行且无法正常退出。关于如何增加信息传递的的稳定性，作者从一些市面上的聊天软件中获得了灵感。

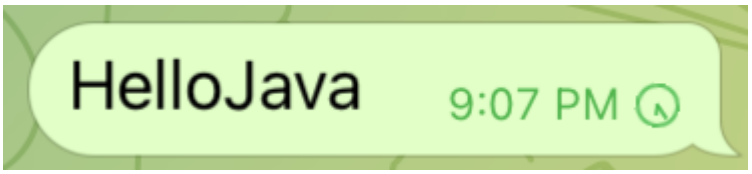


图1

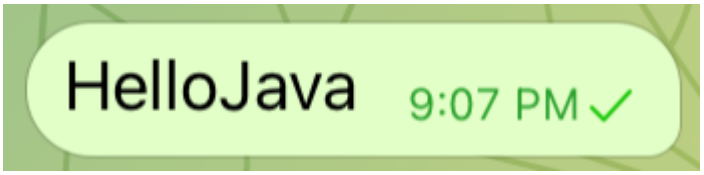


图2

图1和图2展示了某聊天软件中发出的同一条消息的不同状态。可以看到在图1的消息旁有一个时钟标识，这个标识代表消息正在发送，服务器正在接收和传递消息，我们称此时消息在第一个状态；而在图2的消息旁这个标识变成了一个勾，表示消息已经成功发送，我们称此时消息在第二个状态。那么我的设备是如何确定消息是否发送成功的呢？既然我的设备端消息出现了状态变化，那么必然有信息从另一端（此时是服务器）发来。我的手机在发出消息后把消息置为第一个状态，此时消息正在发往服务器；当服务器完成信息接收后，就会立即网络通知我的手机消息已经被收到，从而我的手机上消息被置为第二个状态。对聊天的另一方也同理。事实上，由于网速很快，一条消息发出后第一个状态持续时间很短，马上就会变成第二个状态。这个确认收到步骤的添加并不会对实际使用造成大的影响，但是可以处理很多异常情况下的网络问题。所以在双方进行信息交互时候 不妨添加一个“确认收到”步骤，以此保证连接的稳定性。

```
private void parseChat(String line) {
    line = line.substring(5);
    Vars.control.NetGetMessage(line);
    out.println("Copy");
}
```

在本五子棋程序中，以局内聊天为例，当一方收到聊天消息后，会向对方发送一个“Copy”表示收到消息；而对方也会看到提示，从而达成一个有确认关系的、更安全的网络连接。

5.4 主控模块

主控模块是整个程序的核心部分，位于中央位置调动其他模块，指挥程序运行，是程序进程的把控者。

5.4.1 交互方式

如果我们仔细观察程序，会发现很多“浪费”效率的敌方。比如网络助手收到对方的下棋信息，其实可以直接通知模型来下棋，可为什么要经过控制模块的中转，使得程序效率降低？事实上，如果把人脑看作一台计算机，那应该是一台GPU无限强大，而内存极小的计算机。人对于一个问题的理解、抽象等能力无疑是十分强大的，但是人脑并不能同时处理太多的“进程”，这就导致人们同时处理多个问题的能力很低下。但我们做的看起来“浪费”效率的事情事实上对程序的工作进行了工程化的分类，这让我们可以将问题拆分成很多个小问题来处理，从而更加专注，做的更好。就像刘老师在课上所说：“让你的大脑尽量简单。”

基于这个思想，主控模块做的更多是抽象的事情，其大部分方法都是在调用前几个模块的各种各样的方法，但是有顺序，有指挥。这也是如此一个较大的项目，我们可以有条不紊地顺序编写的诀窍。

5.4.2 下棋控制

主控模块对于下棋的控制体现在下棋顺序、放弃下棋和游戏结束判断的检测上。

对于下棋顺序的把控，主控模块内置四个变量如下：

```
private boolean OpenDoor = false;
private boolean OpenDoorStateWhenRetrack = false;
public int localColor;
public int otherColor;
```

其中OpenDoor表示此时是否可以下棋，若为false则忽视棋盘模块发来的用户点击信息（即棋盘模块哪怕不能下棋还是会检测用户的鼠标点击情况），若为true则执行下棋流程。本地下棋方法如下：

```
public void LocalPutChess(int row,int col) throws IOException, U
    if(!OpenDoor){
        return;
    }
    boolean success = Vars.model.putChess(row, col, localColor);
    if(success){
        Vars.assistent.sendChess(row,col);
        //Vars.chessingsound.Sound();
        //new ChessingSound();
        Vars.northpanel.StopTiming();
        OpenDoor = false;
        Vars.chesspanel.repaint();
        int winner = Vars.model.WhoWin();
        if(winner==GameModel.BLACK){
            JOptionPane.showMessageDialog(null, "黑子胜！");
            Vars.northpanel.StopTiming();
            EndGame(winner);
        }
        else if(winner==GameModel.WHITE){
            JOptionPane.showMessageDialog(null, "白子胜！");
            Vars.northpanel.StopTiming();
            EndGame(winner);
        }
    }
}
```

如果没有胜利，控制模块要做的事情就是成功下棋，网络发出下棋位置，停止计时并停止本地下棋。如果胜利，添加一个选项框即可。网络下棋和本地下棋类似，且不需要检测位置合法性（对方检查过了）也不需要再网络发出下棋指令。

至于下棋时间控制，作者在主控模块中编写了GiveUpChessing方法如下：

```
public void GiveUpChessing() {
    // TODO Auto-generated method stub

    Vars.assistent.OverTime();
    OpenDoor = false;
    Vars.chatpanel.addChatMessage("【你已超时，丧失本轮棋权，请积极游戏】");
}
```

这个方法会在己方下棋且计时器时间结束时调用，并通知网络模块发出己方放弃下棋的信息，对方接收信息后到后调用NetOverTime方法如下：

```
public void NetOverTime() {
    // TODO Auto-generated method stub
    OpenDoor = true;
    Vars.northpanel.BeginTiming();
}
```

这样就可以实现超时的放弃下棋。

5.4.3 游戏开始控制

当主机开始监听而客机连接成功后，网络连接面板将调用控制面板中的StartGame方法来开始游戏，方法体如下：

```
public void StartGame() {
    Vars.chatpanel.addChatMessage("【双方已进入房间，游戏开始】");
    if(localColor == GameModel.BLACK){
        Vars.chatpanel.addChatMessage("【本局你为黑棋，先手下棋】");
        Vars.northpanel.BeginTiming();
    }
    else{
        Vars.chatpanel.addChatMessage("【本局你为白棋，后手下棋】");
    }
    Vars.historypanel.Add();
    Vars.switcher.ToGameWindow();
}
```

方法首先检测下棋方的颜色，决定下棋次序后，开始下棋。

5.4.4 游戏结束控制

当某一步落子之后系统发现五子连珠产生，则提示框后调用游戏结束方法。游戏结束方法中使用了JOptionPane中的showOptionDialog来显示一个对话框，用户选择退出游戏或者再来一局。

```
String options[] = {"回到主界面", "再来一局"};
int choice = 1;
choice = JOptionPane.showOptionDialog(null, "游戏结束",
        WinMsg, JOptionPane.OK_CANCEL_OPTION,
        JOptionPane.QUESTION_MESSAGE, null, options, "退出房间");
switch(choice){
case 0://选择回到主界面
    Vars.chatpanel.Clear();
    QuitRoomMain();
    Vars.model.ClearChess();
    break;
case 1://选择再来一局
    Vars.chatpanel.Clear();
    QuitRoom();
    Vars.model.ClearChess();
    break;
```

值得一提的是，这里弹出的对话框是可以关闭的，若用户关闭则choice不会被赋值。为了解决这个问题，我们对choice赋初始值1，这样用户哪怕直接关闭对话框程序也可以正常运行。

5.5 音乐模块

为了使游戏的游玩更加有趣，我为游戏添加了音乐模块，以播放背景音乐。音乐模块的实现依托于javax中的sound类，其中的Sound方法如下：

```
public void Sound() throws IOException{
    while (sumByteRead != -1) {
        sumByteRead = am.read(b, 0, b.length);
        if(sumByteRead >= 0 ) {
            sd.write(b, 0, b.length);
        }
    }
    sd.drain();
    Sound();
}
```

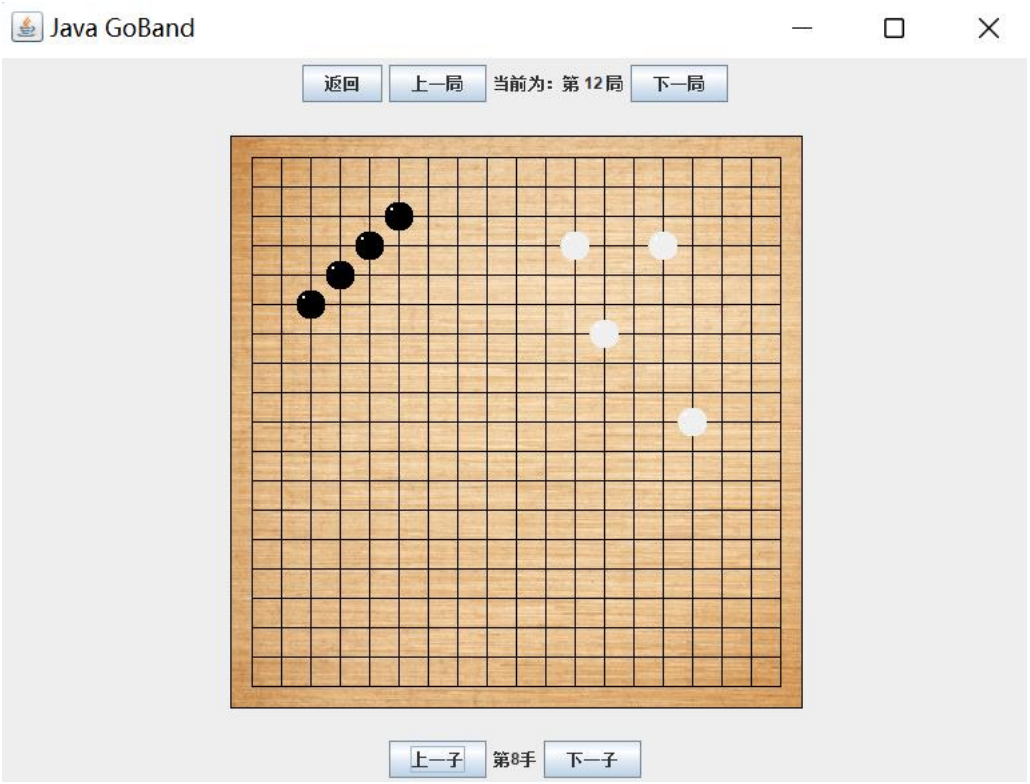
图中的am是取自背景音乐（wav格式）的AudioInputStream，sd是用于播放音乐的SourceDataLine，sunByteRead是一个标志音乐播放是否完毕的int变量。在这个方法中我们向数组b（搬运作用）读取一段am的字节（音频字节），后将这段字节放入sd的缓冲区中，循环此过程，让sd根据时间播放音乐，直到读完整首曲子。sd.drain方法用于主动输出sd的缓冲区（播放音乐）。最后在整个方法完成后递归调用自身，达到循环播放的效果。

第六章 历史回顾模块功能

历史回顾是本Java五子棋的一大特性，其支持玩家赛后对自己进行的每一盘棋局的每一颗落子进行复盘。本章内容即为历史回顾的不同模块及其实现方法。

6.1 历史回顾面板

在历史回顾面板中，页顶有三个按钮，分别用于返回主界面、切换至上一局和切换至下一局。页底有两个按钮，用于切换每一盘棋局中的下一子和上一子。历史回顾面板的布局图如下：



历史回顾面板通过读取历史棋局存档获得棋局数据，并可视化地将其显示在屏幕上。那么我们的历史棋局是如何存储的？历史回顾面板又是如何具体实现？

6.2 对战棋局存储

游戏中的棋局如果离散的存取很复杂。假如我们去观察很多大型游戏，我们会发现这些游戏的游戏进度都是以一个完整存档的形式储存的。在需要存档时整体加载，整体保存。参照这个思路，作者在实现五子棋时将所有的棋局打包成一个LinkedList，将其作为一个整体“存档”进行读取，也同样作为一个“整体”保存。这样做的好处是不需要频繁地对硬盘进行读取和修改操作，且读和写都十分便利。

ChessPoint是作者自己定义的一个类，其实现了序列化接口Serializable，用于存储一个棋子的坐标和颜色。在游戏中，我们用一个ArrayList<ChessPoint>保存一整局棋，在进行下棋和悔棋等操作时对该ArrayList进行增删。对于所有棋局，我们用一个链表将整体存档为LinkedList<ArrayList<ChessPoint>>格式，这样就可以清晰地存取每一局棋。我们设定了一个特定的时间点（比如本程序中设定的时间点为每一局棋局结束时）自动保存存档，在不需要手动保存的情况下保证了不会错过每一局完整棋局的保存。值得一提的是，此处保存所有棋局时不能用ArrayList来实现，一定要用LinkedList，这可能和Java的引用特性有关。

接下来我们定义了一个专门的ReadAndWriteHistory类，用于读取存档。类内有字符串FileName保存存档地址，和以该地址为参数建立的静态file。其中的write方法如下：

```
private static File file = new File(FileName);
public static void write(Serializable s){
    try {
        if(!file.exists()){
            file.createNewFile();
        }
        ObjectOutputStream objectOutputStream =
                new ObjectOutputStream(
                        new FileOutputStream(FileName));
        objectOutputStream.writeObject(s);
        objectOutputStream.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

方法首先会检查文件是否存在，若不存在则建立该文件。考虑到我们要写入的是一个较大的LinkedList，此处使用ObjectOutputStream来进行写操作。此处需

要注意的就是输出流用完之后记得关闭。

6.3 历史回顾面板模块

历史回顾面板作为可视化的一员，其正常工作同样需要棋盘模块、模型模块和控制模块等模块的辅助。不知大家在看前面的内容时是否留意到，其实History板块的很多元素和正常下棋板块中的内容其实十分相似，所以我们考虑能否借用正常下棋的部分来减轻一些历史回顾模块的工作量。

首先考虑能否直接复用代码。如果我们直接使用下棋部分的代码，无疑会增加下棋部分板块的代码量并造成两者之间的高耦合，由此可能造成维护的极其不便利或增加很多难以解决的bug，此方案不可行。

但是如果我们选择0耦合，完全重写这部分代码，又难以利用原来代码中相似的部分，做很多无用功。

于是，我们考虑一个折中的方案：继承与重写覆盖。以历史模型板块为例，该板块和下棋部分的模型板块十分相似，于是我们继承定义历史模型类，如下：

```
public class HistoryModel extends GameModel{
    public boolean putChess(int row,int col,int color){
        ChessBoard[row][col]=color;
        Winner = WinOrNot(row, col, color);
        return true;
    }
    public void RetrackChess(int row, int col, int color) {
        // TODO Auto-generated method stub
        ChessBoard[row][col] = GameModel.SPACE;
        Winner = GameModel.SPACE;
    }
}
```

这看起来有点难以相信，但以上代码就是HistoryModel类的全部代码。它做的仅仅是继承了GameModel，直接使用那些功能一致的方法，重写少部分功能不同的方法即可完成工作。继承带来的耦合程度是恰到好处的，这既避免了两者之间的高度耦合，又不至于需要完全重写代码的低耦合，是一个优雅的问题解法。

其余模块的实现也很类似，均可认为脱胎于下棋部分的对应模块。并且历史回顾时没有那么多规则限制，代码实际上非常简单，故此处不予一一列出。

第七章 程序整体评价及展望

至此，Java五子棋项目算是画上了一个句号，论文也到了最后一章。论文不长，但回忆起Java课程的学习和整个项目的开发，我感到受益良多。本章我们将对开发出的五子棋项目进行评价，并提出对游戏将来完善方向的部分展望。

7.1 程序评价

程序设计之初所要求的五子棋对弈、悔棋和棋操作、局内聊天和复盘等功能本五子棋已经完全具备。在游戏中运用了诸如图形化界面、IO系统、多线程等Java课程中学习到的新知识和音频播放等在课外了解的知识。本次五子棋也是作者第一次以工程化、分工化的思想去处理一个如此大小的项目，第一次感受到这样处理问题的优势，程序最终也正常运行，结果算是满意。

7.2 程序展望

本程序作为一个游戏程序，交友功能也是其一项重要的特性，不过在本次实践中并未体现。由此，可以引入数据库系统，为每位玩家创建自己的账号；引入数据结构中的稀疏矩阵，为每位玩家建立自己的好友列表。照顾没有网络或者网络不好的玩家，我们还可以引入人工智能模块，提供单人人机对战模式等等。

参考文献

[1]　埃克尔编著,Java编程思想,机械工业出版社,2007.06.01

[2]　Java实现音乐播放.CSDN.2021.02.21

致 谢

这篇论文终是完成了，大二上学期的生活也就此划上句号。谨向本学期在我学习和写作论文时关心、帮助过我的所有师长、朋友致以深深的敬意。

首先，要感谢Java语言与应用课程的刘嘉欣老师。这一个学期下来，我学到了关于Java的诸多知识、很多代码的工程最佳实践以及更强的解决问题的能力。正是这样的一门课程给了我磨练代码能力、抗压能力的机会，并且这个过程中的遇到的学科问题刘老师都会细致地解答，实在让我受益匪浅。

要感谢我的父母，他们养育我长大，在我困难的时候给我以帮助和抚慰，在我失意时给我以激励和动力，是我安心坐在这里写代码敲论文的的坚实后盾。他们并不给我什么压力，告诉我随心而去，做自己真正想做的事情，这也是我将携带一生的财富。

要感谢我的女朋友，在无眠的夜里陪伴我敲代码，在我焦虑时给我安慰，在我没有头绪时和我一起探讨，并愿意听我分享无聊的小bug和小日常。

要感谢在这个过程中一起交流探讨的朋友们，是你们的帮助带我走出了编程时的迷惑和写论文时的困顿。

最后，还要感谢学校和国家，是你们为我创造了这样安然的学习环境，这样优异的学习氛围。在此，向所有关心过帮助过我的人致以最诚挚的感激，你们的支持和关心是我不断前行的动力。