南开大学

本科生课程设计论文

中文题目：一种基于 Java 语言的在线五子棋游戏软件

外文题目：An Online Gobang Game Software Based on Java Language

学　号：2012039

姓　名：冯朝芃

年　级：2020

专　业：计算机科学与技术

系　别：计算机系

学　院：计算机学院和网络空间安全学院

指导教师：刘嘉欣

完成日期：2021 年 12 月 21 日

关于南开大学本科生毕业论文（设计）的声明

本人郑重声明：所呈交的学位论文，是本人在指导教师指导下，进行研究工作所取得的成果。除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人创作的、已公开发表或没有公开发表的作品内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。本学位论文原创性声明的法律责任由本人承担。

学位论文作者签名：

年　　月　　日

本人声明：该学位论文是本人指导学生完成的研究成果，已经审阅过论文的全部内容，并能够保证题目、关键词、摘要部分中英文内容的一致性和准确性。

学位论文指导教师签名：

年　　月　　日

摘　要

本文介绍了一款基于 Java 语言的，实现了网络通信功能的五子棋游戏程序设计。该程序图形界面基于 Java Swing 开发。程序架构上严格遵循 MCV 设计模式，实现了 Controller 层、Mapper 层、Viewer 层的分离。本程序网络通信部分基于 UDP 协议实现，能够实现即时的网络对战和文字聊天，并且实现了悔棋、认输、保存对战存档、复盘、输赢判断等功能，以及本地对战和其他装饰性功能，如计时器、背景音乐等。本程序还实现了游戏大厅功能，能够由任意多的客户端各自开辟不同数量的游戏房间，并任选游戏房间加入游戏对战。本程序在游戏大厅主从节点通信、一对一对战通信等方面使用了具有创新性的通信方式设计。本程序在设计过程中综合利用了 Java 语言的面向对象等特性，具有创新性、综合性、独创性等特点。

关键词：

Java Swing；UDP 协议；MCV 设计模式；五子棋

Abstract

This article introduces a program design of Gobang game based on Java language that realizes network communication function. The graphical interface of the program is developed based on Java Swing. The program architecture strictly follows the MCV design pattern, and realizes the separation of the Controller layer, the Mapper layer, and the Viewer layer. The network communication part of this program is based on the UDP protocol, which can realize real-time network battles and text chatting, and realize the functions of regretting chess, surrendering, saving the battle archive, replay, as well as local battles and other decorative functions, such as timers and background music. This program also implements a game hall function, which can open up different numbers of game rooms by any number of clients, and for each user can choose the game room to join the game battle. This program implements innovative communication design in the game hall master-slave node communication and one-to-one battle communication. This program comprehensively utilizes the object-oriented characteristics of the Java language in the design process, and has the characteristics of innovation, comprehensiveness, and originality.

Keywords

Java Swing; UDP protocol; MCV design pattern; Gobang

目　录

第一章　项目背景

第一节　Java 语言简介

Java 是一种广泛使用的计算机编程语言。Java 的特性包括且不限于跨平台、面向对象、泛型编程等，被广泛应用于企业级 Web 应用开发和移动应用开发[1]。

Java 最初的目标为应用于家用电器等小型系统，如电视机、电话、闹钟、烤面包机等。最开始，开发 Java 语言的 Sun 公司试图在这些设备上使用 C++语言，结果发现 C++和可用的 API 存在很大问题。嵌入式系统可用的资源极其有限，而 C++太过复杂难以施展。C++也缺少垃圾回收系统，还有可移植的安全性、分布程序设计、和多线程功能。于是，Sun 公司决定开发一种易于移植到各种设备上的平台，Java 也应运而生。

Java 得到广泛应用，互联网的推动功不可没。随着 1990 年代互联网的发展，Java 创始团队敏锐的觉查到 Java 的特性，能够广泛用于网络编程。于是，他们迅速开发了一个能够运行 Java 的浏览器——HotJava。HotJava 拥有了在页面中运行 Java 代码的能力，这一项目成就了后来的 Java Applet。从此网页第一次具有了动态内容，Java 由此迎来了大发展。

第二节　五子棋简介

现代五子棋起源于日本，在日本颇受人们欢迎。1899 年日本人黑岩泪香证明了原始规则的五子棋先下必胜后，五子棋迈入一条不断改良的道路，经过数十年的修改、验证、再修改，最终发展出加入禁手的五子棋，并经过公开征名，称为连珠，因此规则在日本成型，又称为日式规则或连珠规则。原始规则依然有人在玩，也被称为无禁规则、自由规则[2]。

自 1989 年以来，有多个五子棋人工智能程序赛事。人工智能冠军程序弈心是目前最先进的五子棋专家系统，与人类高手有过多次人机比赛。其中，2017 年 7 月，

弈心以 2:0 比分战胜世界冠军 Rudolf Dupszki——这是五子棋人工智能首次战胜人类世界冠军。

第二章　软件架构和业务流程

第一节　MVC 架构

模型-视图-控制器(MVC)架构作为一种软件设计模式，通常用于开发用户界面，将相关的程序逻辑划分为三个相互关联的部分。这样做是为了将信息的内部表示与向用户呈现和接受信息的方式分开。MVC 架构最开始用于桌面图形用户界面，后来流行于设计 Web 应用程序。

MVC架构的组织形式发生过一定的变化。在最开始的架构中，模型（Model）是程序的核心组件。它是应用程序的动态数据结构，独立于用户界面。它直接管理应用程序的数据和业务逻辑。显示（View）负责向用户呈现信息。控制器（Controller）接受输入并将其转换为模型或视图的命令。除了将应用程序划分为这些组件之外，模型-视图-控制器设计还定义了它们之间的交互方式。在后来的架构中，尤其是 Web 应用程序趋于主流后，业务逻辑部分逐渐被从任务较重的模型部分转移到控制器部分，模型也随之退化为负责数据储存的 Mapper。本程序在 MVC 架构设计上更近似于后一种形式。

本游戏大体可以分为两个部分，游戏大厅界面和对战界面。这两者都采用 MVC 架构进行设计。游戏大厅可以保存对战界面 Controller、Viewer 和 Mapper 对象的引用，并对之进行管理。一个游戏大厅可以管理多个对战界面，这些对战界面可以同时运行；不同的游戏大厅可以被部署到同一台或互联网上的多台设备上。

第二节　游戏大厅的架构和业务流程

游戏大厅设计上使用 MVC 架构。分为 Controller 层、Mapper 层和 Viewer 层。Control 层主要负责业务流程的实现，同时保存有 Mapper 层和 Viewer 层的引用。

Controller 层使用单例模式，确保 Mapper 层和 Viewer 层能够准确有效地获得 Controller 层。

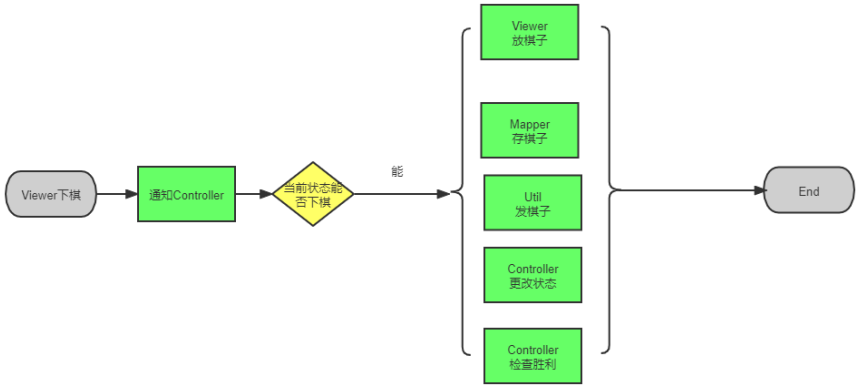

图 2.1 MVC 架构协同运作举例

Controller 保存有默认的服务器地址与通信端口号、本地地址和本地通信端口号等，同时存有 Mapper 层和 Viewer 层的对象引用。游戏大厅有“主服务端”和“从服务端”两种启动模式，以下阐述其启动流程。

游戏大厅启动后，首先检查是否有已经运行的主服务端，如果检查到主服务端正在运行，则以从服务端模式启动。在以从服务端模式启动时，首先将根据接收的服务器响应信息，更新本地用于与服务器通信的端口。之后游戏大厅将启动新的线程，根据接收的服务器响应信息更新房间列表，并根据此决定是否开启新的战局。之后 Controller 通知 Viewer 以从服务端模式显示界面，至此，以从服务端模式启动流程结束。游戏大厅启动时，若没有检查到存在已经运行的主服务端，服务器将以主服务端模式启动。在以主服务端模式启动时，将先后启动以下两个线程。首先开启一个线程在默认端口上开启服务器运行广播，之后开启另一个线程负责监听房间列表更新和向各个服务端同步新列表，这将在网络设计部分展开。最后以主服务端模式启动 Viewer 层，至此，以主服务端模式启动流程结束。若满足对战窗口启动的条件，如，同一房间中对战双方到齐，或用户点击“本地游戏”或用户选择存档进行复盘等，将导致对战窗口的启动。对战窗口的启动在在线游戏、本地游戏和复盘模式上各有不同，笼统的说，可以分为以下几步：第一，新建对战窗口的

Controller、Mapper 和 Viewer 的三者的新对象，并将其引用保存，如果是在线对战还要分配房间的 Key 值。第二，为之分配对战信息和对战玩家信息。第三，通知新建的 Controller 启动。

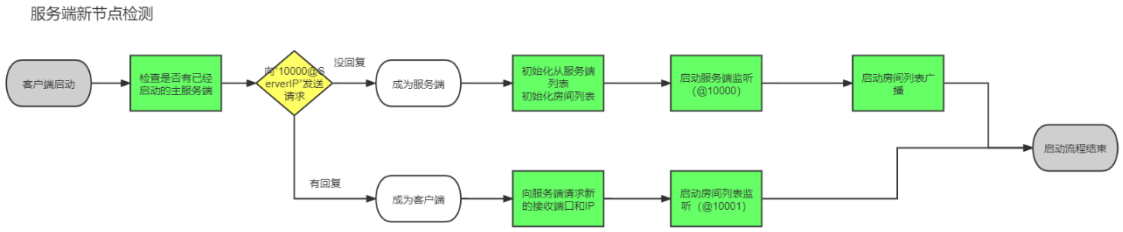

图 2.2　游戏大厅启动流程

为了方便主服务端对其他被管理的从服务端进行管理，本程序设计了端口重定向机制。利用主服务端的端口重定向函数，为新的从服务端分配用于与主服务端交流的端口号。这一点同样将在后面的网络部分详细展开。

为了能够使主服务端对所有的游戏房间进行妥善的管理，在游戏大厅 Controller 层上，Key 值分配函数将房间序号和房间创立者的 IP 地址进行绑定，共同作为房间唯一的 Key 值，即作为程序运行时这一房间的唯一标识符，用于房间在数据结构中的保存。

Mapper 层上，本程序主服务端同时维护四张列表：本地对战表、房间管理列表、远程客户端列表和客户端-房间列表。本地对战表由可以用 Key 值查找的三张子表组成，三张子表分别保存本地启动的不同对战面板的 Controller 层、Mapper 层和 Viewer 层的对象引用。房间管理列表将房间的 Key 值和房间的游戏信息数据进行绑定；远程客户端列表保存了所有从服务端的端口和 IP 地址；客户端-房间列表将主服务端分配给从服务端的端口号和其所开辟的房间的序号进行绑定，即可以通过某一个特定的从服务端服务端口号可以找到这个从服务端建立了哪些房间。

游戏大厅的主界面由一系列房间子面板和一些功能按钮组成。Viewer 层负责，对这些对这个面板进行初始化，提供一些方法如“更新房间列表”等接受 Controller 的调用，并通过调用 Mapper 层的函数更新 Viewer 层显示信息。

第三节　对战页面的架构和业务流程

对战界面可分为三种：网络对战模式、本地模式、复盘模式。对战界面同样采用 MVC 架构进行设计，对于不同模式的对战界面，它们具有各自不同的 Controller 层、Mapper 层和 Viewer 层。Controller 层存有 Mapper 层、Viewer 层和网络工具类对象的引用，并且保存有对战状态、对战信息、玩家信息、棋盘状态、获胜者等信息，以及是否终止游戏等判断函数和一些对应的 Getter 和 Setter 方法。

以下介绍网络对战启动流程。首先，Controller 向 Mapper 层和 Viewer 层分配从游戏大厅得到的 Key 值，以此在后续操作中通过游戏大厅中的方法找到 Mapper 层和 Viewer 层对象，同时将初始化网络工具类对象。之后将设置对局信息，并将对局信息传递给 Viewer 层进行显示。之后将进入“抢先后手”的环节。首先 Viewer 层弹出提示，弹出提示信息通知玩家进行棋子持方的选择，如图所示。两方的对战者分别选择自己的执棋颜色，如果颜色不同将进行直接进行分配和显示；如果颜色相同，将根据发来数据包中携带的选择时刻，进行持方的分配。“抢先后手”的环节结束后，将打开 Post 线程，用于在对战过程中发送和接收各种消息。

对战界面的 Controller 层，可以根据其所处于的不同状态，对于不同的请求进行就进行区别的响应。本程序对战界面的 Controller 层中设置了四种状态：空状态、下棋状态、等待状态和结束状态。以等待状态为例，在等待状态中 Controller 只对对方传来的消息进行响应，而对本方棋盘的点击事件不做出任何响应。在下棋状态中，Controller 只对本方 Viewer 层的棋盘的点击事件进行响应，并在响应后保存落子位置，通知 Post 进行网络发送，并且立刻判断输赢。而在这个过程中，对于对方发来的消息，如果是进行落子的，则不进行响应。这样就共同保证了不会出现抢先落子或两方落子不同步的情况。

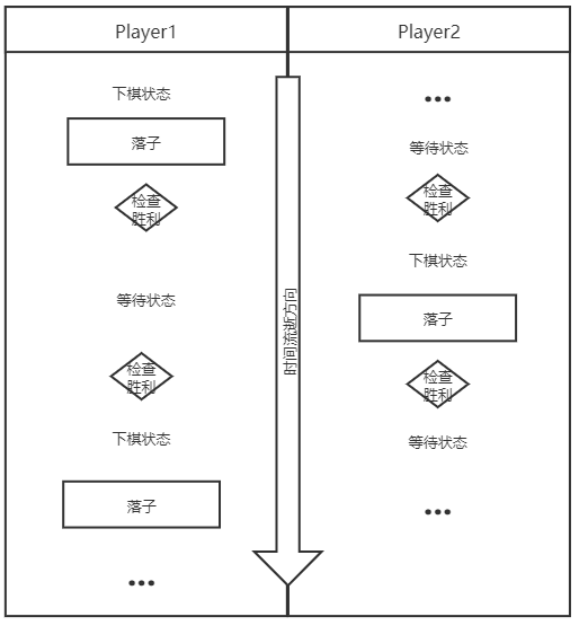

图 2.3　对战过程示意图

实现悔棋功能通过以下方式实现。在收到悔棋消息后，Viewer 层将对应棋子设为不可见；Mapper 层会正常保存悔棋信息，以便复盘使用；这样便实现了悔棋效果。

为实现投降功能，用户点击投降按钮后，将发送对应的投降信息。Controller 层在收到投降信息后，立刻修改对战输赢状态，并通知 Viewer 层显示相应的提示信息。

保存存档功能，用户可通过点击“Save to File”按钮来使用。该操作将保存对战双方的 Player 信息对象和保存有下棋棋子顺序的链表对象到 TXT 文件中。复盘时可通过此文件恢复两个对象，以重现下棋棋局。

对于本地下棋和复盘两种模式，在三个不同模块层面都各自进行了的简化。如在本地对战模式时，不会启动网络发送端，并且不会出现聊天框和对方信息等内容，并且在落子的判断上也进行了相应的简化。在复盘模式下显示对战双方的信息，但不会显示聊天框，只显示上一步下一步两个按键，以此简化操作。

第三章　GUI 设计

第一节　Java Swing 简介

Swing 是 Java 的 GUI 部件工具包，是 Java 基础类(JFC)的一部分，是一种用于为 Java 程序提供图形用户界面(GUI)的 API。

Swing 的开发目的是提供一组比早期抽象窗口工具包(AWT)更复杂的 GUI 组件。Swing 提供了一种跨平台的观感，也就是观感与底层平台无关。Swing 具有比 AWT 更强大、更灵活的组件，除了按钮、复选框和标签等常见的组件外，Swing 还提供了几个高级组件，例如选项卡式面板、滚动窗格、树、表格和列表。Swing 与 AWT 组件不同，Swing 组件不是基于特定平台的代码实现的。Swing 完全用 Java 编写，因此与平台无关[3]。

第二节　游戏大厅界面整体布局

游戏大厅界面采用 Swing Layout Manager 中的 Border Layout 进行布局。中央部分显示的是可用的各个房间，该部分由一个个小的房间面板组成，可以用于创建新的房间，也可以加入已有房间。上方显示“本地游戏”按钮，点击可以开始进行本地游戏。左侧“来点音乐”按钮，点击可以选择背景音乐。下方“本地复盘”按钮，点击可以选择存档文件进行本地复盘。

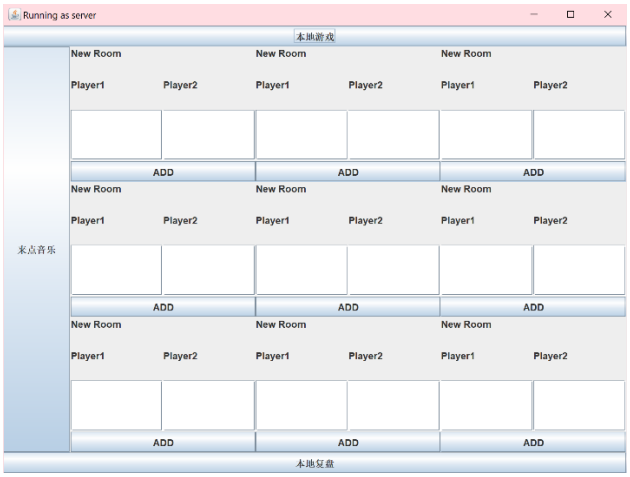

图 3.1　游戏大厅界面

房间面板由以下部分组成，从上至下分别是：房间名、玩家一和玩家二的名称信息（昵称输入文本框）、功能键。房间面板有三种变化模式。第一种是空房间状态，这时可以在任意一个文本框内输入玩家昵称，并点击“ADD”按钮进行房间的添加。之后房间面板将变成待加入状态，空缺出来的昵称输入底色将变红，房间名称也将做出相应的改变，改变为游戏大厅主服务端分配的名称。另外一个玩家可以在另外的客户端上，在变红色的文本框中填入自己的昵称信息，并点击“JOIN!”按钮开启新的对局。对局开始以后，功能键将变成灰色，表示不可加入，并且正常地显示房间名称、对战者的昵称等。图 3.2 从左至右分别为，空房间状态的房间面板、待加入状态的房间面板和对局开始后的房间面板状态。

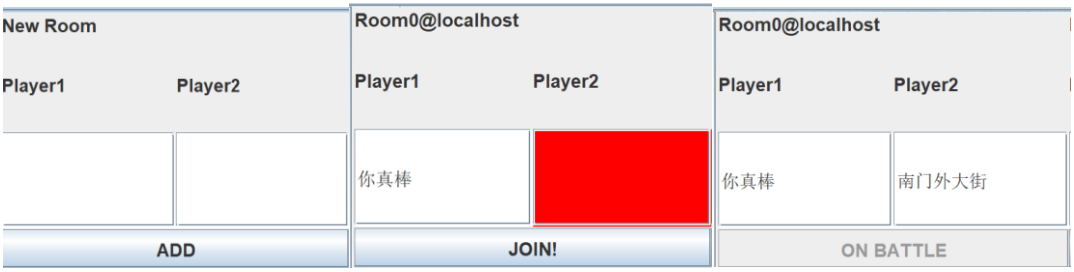

图 3.2　对战面板及其状态变化

其他按钮功能介绍。点击“本地游戏”按钮可以开始本地对局。点击“本地复盘”按钮，可以在文件选择器中选择之前保存的对局存档进行复盘。点击“来点音乐”按钮，可以通过文件选择器选择想要播放的背景音乐。

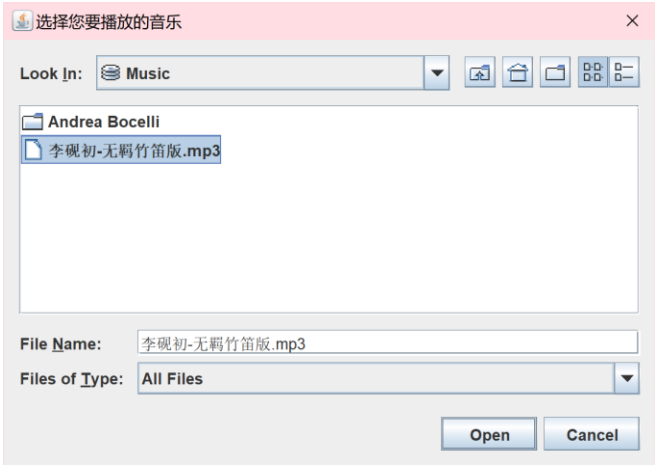

图 3.3　背景音乐选择界面

第三节　对战面板的设计

对战界面主要由棋盘、玩家信息、功能键组、对话面板、倒计时牌等部分组成。对战面板为一个具有棋盘图片作为背景的 JPanel，和其上划分为 19*19 个正方形的透明 JPanel。在玩家进行点击后，Viewer 层将进行根据玩家持方进行判断，将对应位置的透明 JPanel 的背景图设置为黑色棋子或白色棋子，以此达到显示落子的效果。右侧的功能键分别为悔棋（Withdraw）、退出（Quit）、保存游戏存档（Save to File）以及上一步（Previous）和下一步（Next）（上一步、下一步在网络对战时不可用）。玩家信息分别显示两个玩家的昵称、对战端口号和 IP 地址以及持方。聊天面板主要分为两个部分，一个是显示之前聊天记录的 Chat（JTextArea）和用于输入新消息的 MessageBox（JTextField）和发送（Send）按钮，用于发送新的消息。倒计时牌可以显示当前时间和以五分钟为限进行倒计时。

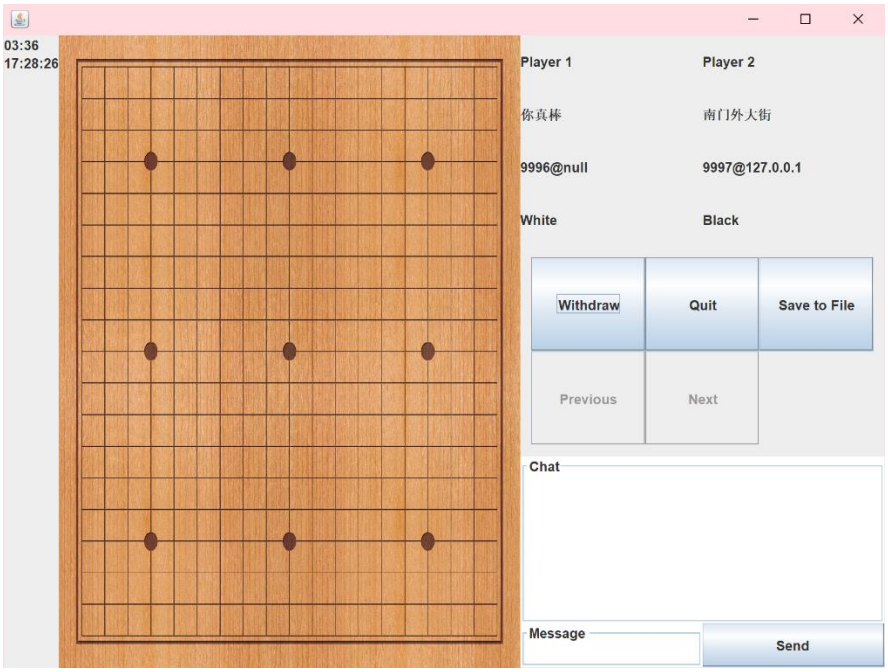

图 3.4　对战面板设计

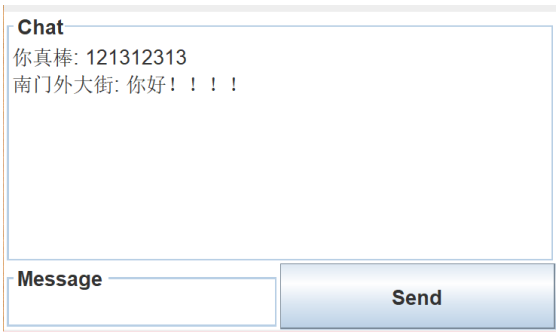

图 3.5　即时聊天面板

本地对战面板较网络对战面板，进行了简化，仅保留了棋盘功能键和倒计时牌，如图所示。

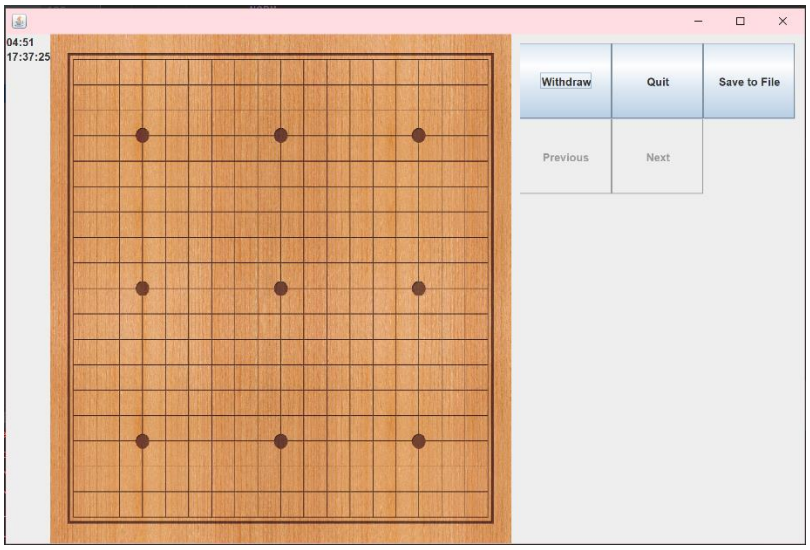

图 3.6　本地对战面板设计

复盘对战面板在网络对战面板的基础上进行了简化，仅保留的棋盘功能键、玩家信息和倒计时牌。复盘者可以通过文件选择器进行复盘存档的选择，并使用 Previous 和 Next 的按键进行每一步的查看。

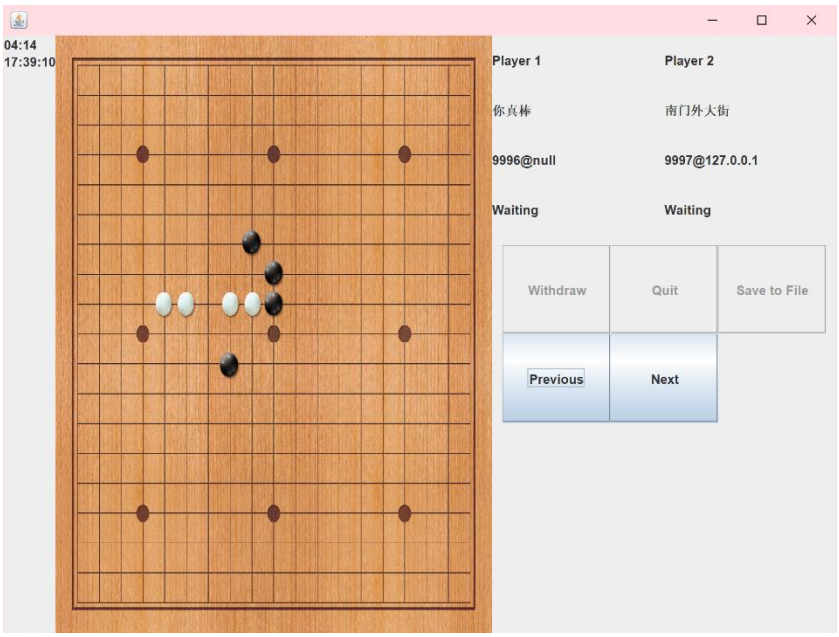

图 3.7　复盘面板设计

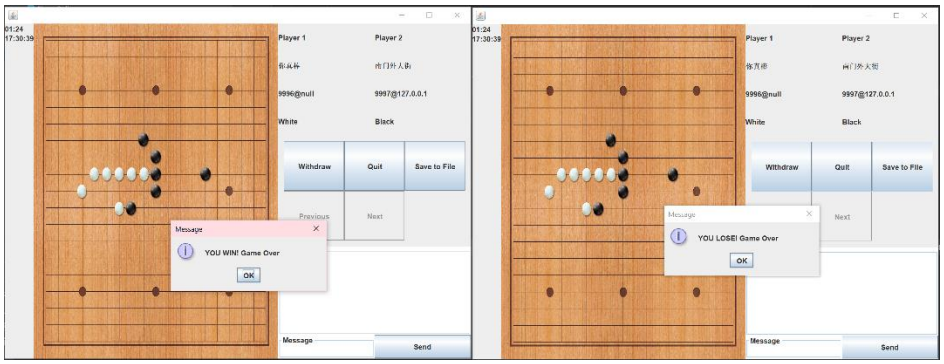

图 3.8　终局输赢判断

第四章　网络通信设计

第一节　UDP 通信简介

用户数据报协议(UDP)是 Internet 协议套件的核心成员之一。通过使用 UDP，应用程序可以在不需要预先通信来建立通信通道或数据路径的情况下，向网络上的其他主机发送消息，即数据报。UDP 使用具有最少协议机制的简单无连接通信模型，提供数据完整性校验和，以及用于寻址数据报源和目的地的不同功能的端口号。因为 UDP 通信没有握手对话，因此会将用户的程序暴露在底层网络的任何不可靠性之下，因而不保证交付和订阅，也无法过滤重复信息[4]。

UDP 适用于对错误不敏感的场景。时间敏感的应用程序通常使用 UDP，因为此时丢弃数据包比等待由于重传而延迟的数据包更可取。

本程序使用 UDP 协议进行通信，主要是考虑到 UDP 协议不需要维持连接，因而在程序实现上更加灵活简单。

第二节　网络通信实现

一、通信分离机制

这一部分介绍主服务端节点向从服务端分配端口的过程。服务器为每一个客户端分配特定的通信端口是一种常规做法。如果使用服务端在一个固定的端口上接收来自所有从服务端的消息，那么在过滤上就会产生较大的困难，时间开销增大和信息丢失的问题也会凸显。如果所有的对战中的信息都通过主服务端进行转发，那么就会导致大量的并发访问，这样同样都不利于本程序稳定性的提高。针对以上问题，我们设计了两种端口分配机制，一是在主服务端节点上对于各个从服务端分配固定的监听端口，主服务端与任意一个从服务端通信时都使用本机的不同端口。二是在对每一场对战中，进行对战的双方进行端口分配，使之实现点对点的在线对战。以下进行详细介绍。

图 4.1　主从服务端关系示意图

（一）主从节点通信端口分配

每一个服务端启动时，都会向主服务端默认的端口，发送检测消息，检测主服务端是否存在。如果主服务端已经存在，那么主服务端在默认端口上的监听进程，就向新启动的从服务端响应一个新分配的接口。确定新分配的接口之后，主从服务端之间就会通过这个接口来进行通信，而不占用之前的启动检查时的默认接口，以实现通信检查接口的稳定，和对每一个从服务端有针对性的信息传输。

（二）端口再分配

为了减轻主服务端节点的压力和维持程序运行流程清晰流畅，我们在网络通信上实现了端口再分配和端口分离的机制。具体的机制有如下两点，第一，主从节点之间：主服务端上与从服务端通信的端口各自分离，主服务端与每一个从服务端通信都使用一个特定的端口。第二，对战端口分配：对战过程中所使用的端口相互分离；对战过程中由服务端之间，使用主服务端为其分配的端口，以点对点的方式通信；由服务端开辟线程，以点对点的方式完成对战，实现了对战端口与服务端间通信端口口的分离。

图 4.2　端口分配示意图

（三）新设备上线时的端口分配

主服务端与各个从服务端通信使用的端口分离，通过以下过程实现。从服务端在启动时通过默认的端口与检查主服务端是否存在。在主服务端侧，收到从服务端启动后发来的请求时，将立刻为新上线从服务端分配新端口，并通过原有默认端口向新上线从服务端相应。从服务端接收到该端口号后，立刻改变自己与主服务端通信的目标端口号。主服务端同时也做出相应的调整，主服务端会把新上线的从服务端端口号和地址存在本地的数据结构中。这样双方都实现了端口调整之后，主服务端就可以和每一个从服务端以特定端口分离的进行通信，以免造成过多流量挤占同一端口和过滤不同来源信息的不便。

（四）对战的点对点模式

考虑到同一时间活动的对战房间数量可能较多，若使所有对战内容经过主服务端节点进行转发并不现实等因素，本程序将对战过程与主从服务端通信过程分离。对战过程开始时游戏大厅启动的新的对战界面线程负责通信和维护用户界面，因而

对战双方是采用点对点的通讯方式，通过主服务端分配的端口号进行一对一的通信，这样就避免了对战信息经过主服务端转发，从而减轻主服务端节点压力，并且使整个通信过程更为清晰和明确，减少干扰。

二、房间列表的更新与同步

该部分介绍游戏大厅主从节点之间的房间列表同步更新的实现。房间列表同步和各房间的状态更新是本程序需要解决的重点问题。如果房间列表得不到及时的更新，那么各个玩家的游戏体验就会大打折扣。为了保证每一个节点上建立的新房间，都能被每一个游戏大厅节点及时捕获并通知给玩家，以及每一次玩家加入房间和房间对战启动的过程能够顺利流畅，我们设计了以下房间更新解决机制。在主服务端侧，我们建立了一个监听线程，该线程将轮询每一个从服务端端口列表中的端口，每一个从服务端监听超时时间为 50 毫秒。完成轮询后，该线程将作出判断是否已经收到新的更新信息。如果收到了这样的更新信息，那么接下来将根据信息内容判断，其中包含的房间信息是建立一个新的普通房间，还是有新的对战者加入一个房间。如果是加入一个已有的房间，那么我们就将分配新的对战端口号，并通知对战双方，用以实现双方的点对点对战，如果这个房间正好由在本地服务端节点创建，那么要立刻在本地开始对战。如果该房间信息是开辟的新房间，那么我们直接进行下一步，因为不管是加入房间，还是开辟新房间，程序都需要更新本地房间信息。对于新房间，程序还要分配房间号，之后主服务端将对每一个从服务端发送更新的房间信息。如果本地缓存中有待发送的数据，那么意味着本地需要开创或加入已有房间那么。那么，我们还将向每一个从服务端发送缓存中的信息。最终，这个流程结束，并将自动开始下一轮的监听过程。

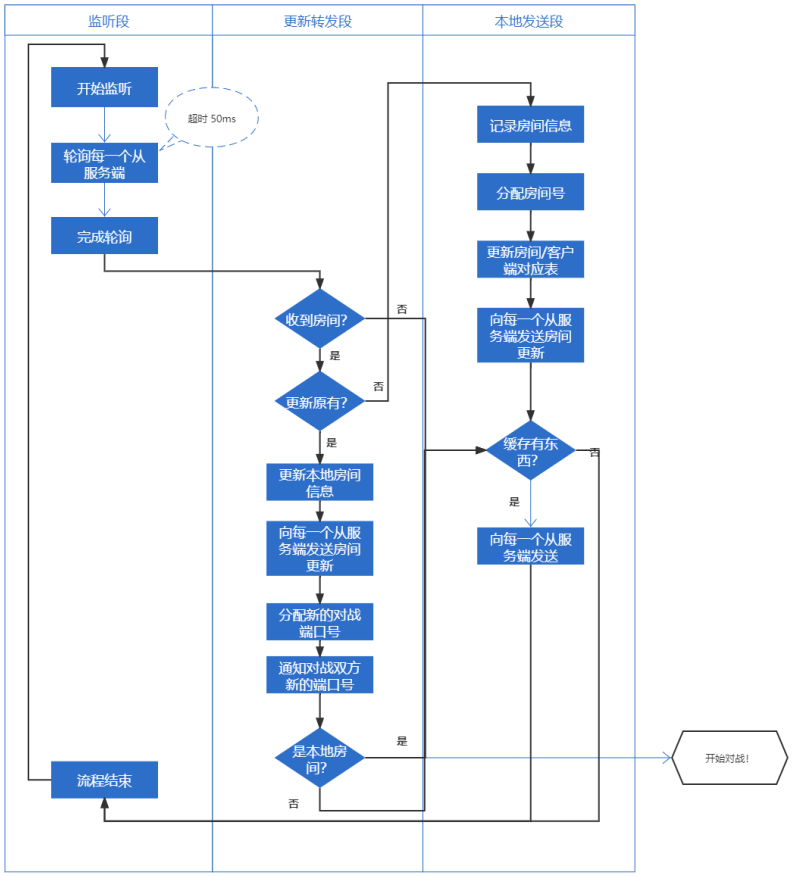

图 4.3　主服务端侧的房间信息更新转发流程

在从服务端节点上，我们对接受转发流程进行了相应的简化。接收线程开始监听后超时 1000 毫秒。如果收到了房间信息，这意味着两种情况，一是主节点发来的创建新房间的信息，那么我们直接更新本地房间信息。如果这间房间为本地持有的房间，也就是说，本地服务端发来的是有新的对战者加入本地开辟房间的信息，那么在这种情况下，我们将直接开始对战并更新本地房间记录。上述流程结束后，如果本程序本地缓存中有待发送的信息，这意味着本地创建了新的房间，或是本地需要加入已有房间，那么我们把缓存中的信息向主节点进行汇报。汇报完成后，本程序清除缓存，并开始下一轮的监听过程。

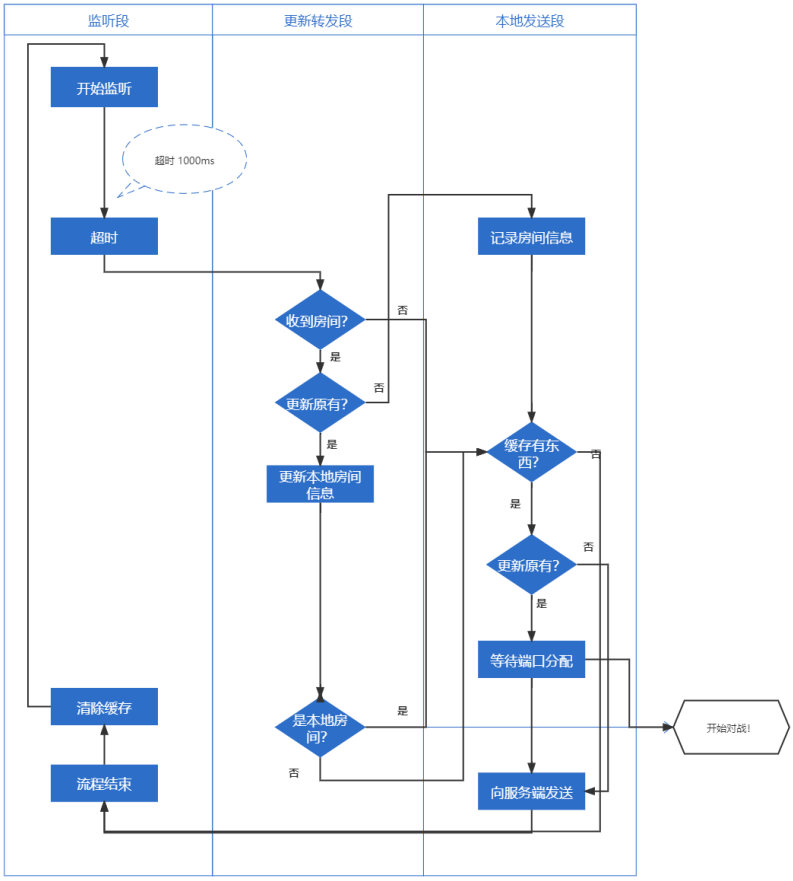

图 4.4　从服务端侧的房间信息更新转发流程

三、点对点对战过程中的通信

为了实现点对点对战通信中的数据接收与发送我们设计了一个用于对战中数据收发的邮局（Post）线程。首先在先前指定的对战通信端口进行监听，超时之后判断是否收到对战相关消息，若收到消息则通知 Controller。Controller 将根据当前棋局状态和消息内容类型进行处理。如果是落子信息，且本方为等待对方落子状态，则通知 Viewer 层进行落子；如果是悔棋、放弃等信息，则通知 Viewer 层进行相应处理；如果是文字信息，则通知 Viewer 层，将其展示在文本框中。之后邮局将检查本方缓存中是否有待发送的数据，如果有的话将其发送给对手并清除缓存，之后将开始下一轮的监听等待。

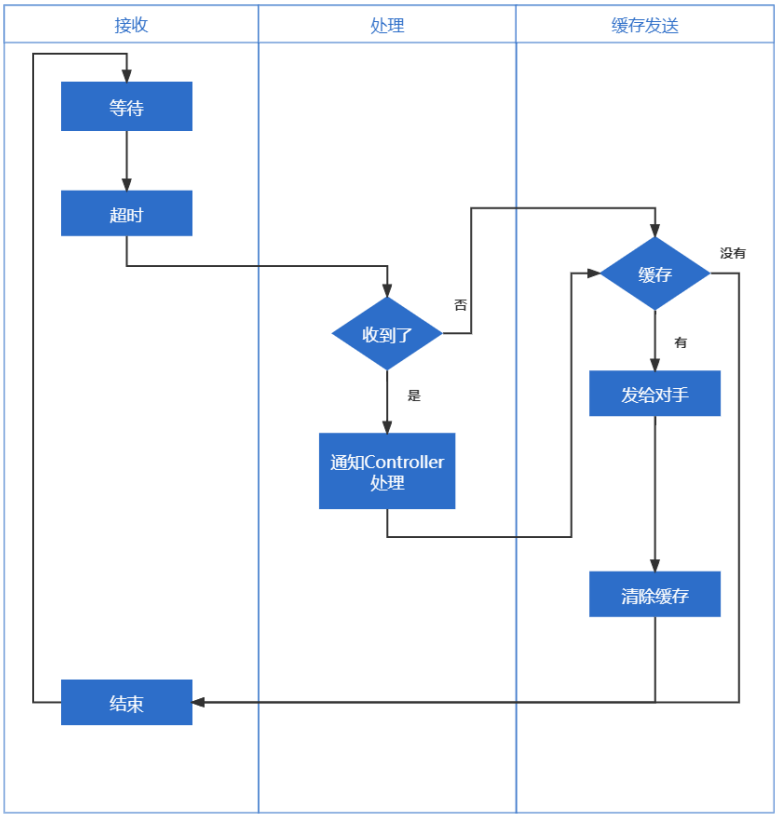

图 4.5　邮局线程工作流程

为了行之有效的发送数据，我们将游戏中可能发送的各类信息都打包成“棋子”对象。一个棋子对象首先包括一个类型信息，该类型信息可以用于判断该棋子对象所保存的内容类型，如正常下棋的位置、颜色信息，悔棋的位置信息，投降信息，聊天文本消息等。

四、抢夺先后手环节

为了增加对战过程中的趣味性，我们还设计了争夺先后手环节。为确定对阵双方的下棋次序，对战面板一旦打开，首先将弹出提示信息，提示玩家选择先手后手。玩家选择后将自动记录玩家选择的持方和选择时间，并将选择的持方和时间信息封装进一个对象中进行发送。对方接收到该对象后，通过比照本方选择时间和对方选择时间确定谁先抢到，以便在抢夺持方相同的情况下，判断持方的归属。

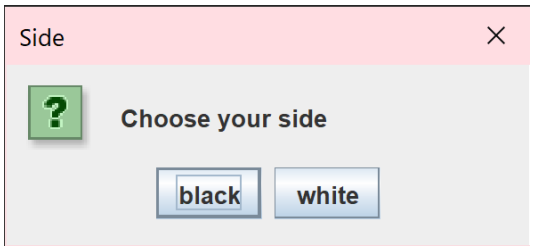

图 4.6　选择持方提示信息

第五章　Maven 依赖导入和背景音乐功能

Maven 是一个主要用于 Java 项目的构建自动化工具。Maven 项目由 Apache 软件基金会托管，它解决了构建软件的两个方面：软件是如何构建的，以及它的依赖关系。Maven 使用 XML 文件描述了正在构建的软件项目、它对其他外部模块和组件的依赖关系、构建顺序、目录和所需的插件。它带有预定义的目标，用于执行某些明确定义的任务，例如代码编译及其打包。Maven 从一个或多个存储库动态下载 Java 库和 Maven 插件，并将它们存储在本地缓存中[5]。

为了增加扩展性和增进更多的功能。本项目开发过程中加入了一个 Maven Project，本项目利用它导入了 Google Soundlibs 包，通过它实现了背景音乐播放的功能。

第六章　总结与展望

第一节　引入 Kafka 架构与旁观模式

本程序没有试图加入观战模式，主要原因在于目前的对战环节通信过程比较复杂，且流程中信息丢失概率大；在房间加入过程中，信息传递也相对环节较多，主服务端、从服务端之间的通信过程比较复杂。在这种情况下，直接加入观战列表及一系列观战从服务端节点是非常困难的。

基于这些原因，本程序计划使用 Kafka 架构作为主服务端和从服务端之间的信息传输方式，以优化目前的信息传输过程。Kafka 软件总线流处理框架由 Apache 软件基金会开发。Kafka 旨在提供一个统一的、高吞吐量、低延迟的平台来处理实时数据反馈[6]。Kafka 实现了一种名为发布/订阅模式的消费模式，即利用 topic 存储消息，消息生产者将消息发布到 topic 中，同时有多个消费者订阅此 topic，消费者可以从中消费消息，这些消息不会被立刻清除，而是默认被 Kafka 保留一段时间。这就和本程序的应用场景非常相符，游戏中的各类消息并非所有的客户端节点都需要接收，而是可以分成不同的类，每个类对应一个 topic。这些信息也往往并非需要永久保存，更需要及时分发，这样就可以很好地利用 Kafka 架构特性来实现我们的要求。

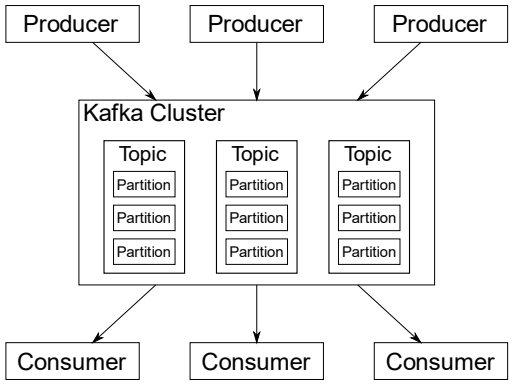

图 6.1　Kafka 架构示意图

Kafka 架构会建立一系列的 topic，对于每个 topic 下的订阅者，我们可以对其定向的传送数据。这样一来，我们可以将所有的客户端节点放入一个 topic 下，使

其同步所有的房间更新信息。另一方面，对于每一个房间的对战双方和观战方，我们可以把它放入其他的 topic 下，只对他们发送对当前战局的更新信息，在这个由对战者和观战者构成的列表中进行同步。这样一来，就解决了原始程序没有观战模式，和相关通信困难的问题。

第二节　主服务端和从服务端的分离

在原始程序中，我们的游戏大厅的程序可以既充当主服务端，也可以充当从服务端。这样一来，服务端所要实现的业务逻辑十分繁杂，并且要进行维护非常困难。主节点本身要维护自己的房间列表，并开始战局，这实际上并不符合在真正生产场景中，主服务端所处的位置和作用。如果我们把主服务端放入服务器中，那么事实上，主服务端只需要维护客户端之间的通信就可以了。本程序的下一步优化计划，就是专门制作服务端程序和客户端程序，服务端程序只负责客户端程序之间的通信和数据同步；客户端程序被部署到客户机上，用户用客户端程序来进行对战，这样一来就可以使代码，尤其是房间同步的监听和转发过程，大大简化，降低了复杂度，提高了可扩展性。

第三节　通信线程的优化

目前本程序的通信线程，不论是主从节点之间，还是一对一对战过程中，都使用先轮询或超时，后发送的通信模式。在这样的程序设计下，如果对方在本方发送过程中发来消息，很有可能造成接收不到的情况。这样就给程序的运行埋下了隐患，在测试过程中也偶然出现接收不到的情况。为此，本程序下一步应使用多线程的方式对该通信过程进行优化。平时，监听线程应该独立运行，并始终保持监听状态。需要发送的时，将监听线程结束并立刻发送，之后再重新启动监听线程。原有的发送方式每经过一轮轮询之后，都会检查缓存并决定是否发送，至现在只有在需要时才发送。在这样的情况下，很难遇到双方同时发送的情况，这样就大大降低了信息丢失的概率。

致　谢

在本程序完成之际，我想首先感谢 Java 语言最初的开发者们。他们精妙的构思成就了 Java 这门神奇精彩的语言体系。并且是他们独具慧眼，发现了 Java 语言在 Web 时代的独特应用，才使得 Java 没有销声匿迹，而成为一种广泛广泛为人们接受和使用的优秀语言。

之后尤其感谢 Java Swing 的开发者们。Java Swing 的体系架构非常完美，可以说是一个让人钦佩不已的艺术品。虽然说在今天，Java Swing 的运用在减少，但是通过在它的基础上开发程序的过程，使我充分感受到了程序架构的精美、完善的艺术性，可以说是对我的工程能力和专业素养的极大提升。

然后我还要感谢优秀的 IDE——IDEA。没有它的帮助，我不可能顺利的完成本程序。同时我还要感谢 Github Copilot——虽然它不能替我完成业务逻辑，但是对于一些套路性的代码，它显得尤其实用，并且在我写程序“断片儿”的时候，它也总是能提供及时的帮助。

最后的最后，衷心感谢刘嘉欣老师及其团队为我们准备了如此精美绝伦的、丰富完善的 Java 课程，没有他的指导，我不可能有今天对于 Java 语言掌握到一个初步熟练的程度。非常感谢他们能够在百忙之中评阅我的论文和聆听我的答辩，谢谢！

参考文献

[1] 维基百科. Java[EB/OL]. [2021-12-21]. https://zh.wikipedia.org/wiki/Java.

[2] 维基百科. 五子棋[EB/OL]. [2021-12-21].
https://zh.wikipedia.org/wiki/%E4%BA%94%E5%AD%90%E6%A3%8B.

[3] Wikipedia. Swing(Java)[EB/OL]. [2021-12-21].
https://en.wikipedia.org/wiki/Swing_(Java).

[4] Wikipedia. User Datagram Protocol[EB/OL]. [2021-12-21].
https://en.wikipedia.org/wiki/User_Datagram_Protocol.

[5] Wikipedia. Apache Maven[EB/OL]. [2021-12-21].
https://en.wikipedia.org/wiki/Apache_Maven.

[6] Wikipedia. Apache Kafka[EB/OL]. [2021-12-21].
https://en.wikipedia.org/wiki/Apache_Kafka.