南 开 大 学

JAVA 语 言 与 应 用 实 验 报 告

中文题目： **在线五子棋游戏实验报告**

外文题目： Experiment Report: Online Gomoku Game

学　　号： 2210556

姓　　名： 廖望

年　　级： 2023 级

专　　业： 计算机科学与技术

系　　别： 计算机科学与技术

学　　院： 计算机学院

指导教师： 刘嘉欣

完成日期： 2024 年 12 月

摘 要

本文介绍了一款基于 Java 语言的五子棋游戏程序设计，具备网络通信功能。该程序的图形用户界面（GUI）采用 Java Swing 开发。程序架构严格遵循 MVC 设计模式，实现了 Controller 层、Mapper 层和 Viewer 层的分离。网络通信部分基于 UDP 协议，支持即时的网络对战和文字聊天。程序还实现了悔棋、认输、保存对战记录、复盘、输赢判断等功能，以及本地对战和其他装饰性功能，如计时器和背景音乐等。此外，本程序实现了游戏大厅功能，支持任意数量的客户端开设不同数量的游戏房间，并可自由选择加入游戏对战。游戏大厅主从节点通信、一对一对战通信等方面采用了创新性的通信方式设计。在设计过程中，本程序综合利用了 Java 语言的面向对象等特性，展现了创新性、综合性和独创性等特点。

关键词： Java Swing；UDP 协议；MCV 设计模式；五子棋

Abstract

This paper introduces the design of a Gomoku game program based on the Java language, featuring network communication functionality. The program's graphical user interface (GUI) is developed using Java Swing. The program architecture strictly follows the MVC design pattern, achieving separation of the Controller, Mapper, and Viewer layers. The network communication is based on the UDP protocol, supporting real-time online battles and text chat. The program also implements functions such as move undo, concede, save game records, replay, win/loss judgment, as well as local battles and other decorative features like a timer and background music. Additionally, the program includes a game lobby feature that allows any number of clients to open different game rooms and freely choose to join game battles. The communication design in the game lobby, including master-slave node communication and one-on-one battle communication, adopts innovative communication methods. Throughout the design process, the program makes comprehensive use of Java's object-oriented features, demonstrating innovation, integration, and originality.

Key Words: Java Swing; UDP protocol; MVC design pattern; Gomoku

目 录

第一章　项目背景

第一节　Java 语言简介

1.1.1　什么是 Java？

Java 是一种面向对象的高级编程语言，由 James Gosling 和他的团队在 1995 年由 Sun Microsystems 推出（现为 Oracle 公司所有）。Java 以其“Write Once, Run Anywhere”（一次编写，到处运行）的理念闻名，是一种广泛应用于各类软件开发的语言。

1.1.2　Java 的主要特点

1. **跨平台性** Java 程序通过 Java 虚拟机（JVM，Java Virtual Machine）运行，不依赖操作系统的特定功能，使其能够在不同平台上运行。
2. **面向对象** Java 是一种纯面向对象的语言，核心思想包括封装、继承和多态，使代码更易维护和复用。
3. **丰富的类库** Java 提供了一个庞大的标准库（Java API），涵盖了数据结构、网络编程、数据库连接、GUI 开发、多线程等功能。
4. **安全性** Java 的设计考虑了安全性，例如内存管理由 JVM 自动完成，并且具有严格的权限管理和防御未授权访问的功能。
5. **多线程支持** Java 内置对多线程的支持，允许程序并发执行多个任务，提高运行效率。
6. **垃圾回收机制** Java 提供了自动垃圾回收机制，开发者无需手动管理内存，降低了内存泄漏的风险。
7. **动态性** Java 支持动态加载类和运行时绑定（Late Binding），并能够与动态语言进行良好的集成。

1.1.3　Java 的核心组件

- **JVM（Java 虚拟机）**负责解释和运行 Java 字节码，使得 Java 程序能够跨平台运行。
- **JDK（Java Development Kit）** Java 开发工具包，包含编译器（javac）、运行时环境（JRE）和相关开发工具。
- **JRE（Java Runtime Environment）** Java 运行时环境，提供运行 Java 应用

所需的 JVM 和类库。

1.1.4 Java 的主要应用

- **桌面应用程序**利用 Java 的 GUI 框架（如 Swing 和 JavaFX）开发桌面软件。
- **Web 开发** Java 提供了强大的 Web 开发框架（如 Spring、Struts），适合构建企业级 Web 应用。
- **移动开发** Android 应用开发的主要语言之一是 Java（基于 Android SDK）。
- **企业级应用** Java 的 EE（Enterprise Edition）平台广泛用于开发分布式、大型企业应用系统。
- **大数据和分布式系统** Java 是许多大数据框架（如 Hadoop、Spark）的主要语言。
- **嵌入式系统** Java 也常用于开发小型嵌入式设备的软件。

第二节 五子棋简介

五子棋（Gomoku）是一种源于中国的传统棋类游戏，因其规则简单、对抗性强而备受欢迎。该游戏通常使用 19×19 的棋盘，但实际对局中也常用 15×15 或更小的棋盘。五子棋的棋具为黑白两色的棋子，无需区分棋子的形状或大小。

基本规则：

1. 棋子和棋盘：
   (a) 双方分别执黑白棋子。
   (b) 棋盘上设有交叉点，棋子下在交叉点处。
2. 游戏目标：率先将五颗同色棋子连续排列（横、竖、斜方向均可）的一方获胜。
3. 落子顺序：由执黑棋的一方先行，双方轮流落子。
4. 禁手规则（可选）：在某些比赛中，为平衡双方实力，黑棋禁止某些特定的落子方式（如长连、三三、四四等），白棋则不受此限制。

特点：

1. 策略性强：五子棋考验玩家的全局思维和逻辑推理能力，既需要考虑进攻，又要兼顾防守。
2. 简单易学：规则直观，几分钟即可上手，但要精通需要深厚的经验和智慧。
3. 适合多种场景：既可用于休闲娱乐，也被广泛应用于竞技比赛中。

五子棋在世界范围内有广泛的爱好者，并发展出了专业的赛事和棋力评级

体系，如国际五子棋联合会（Renju International Federation, RIF）组织的比赛及各类线上五子棋平台。

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）架构是一种软件设计模式，常用于用户界面的开发，它将程序逻辑划分为三个相互关联的部分。这样做的目的是将应用程序的内部表示与用户界面的展示和输入方式分开。MVC 最初用于桌面图形用户界面，后期广泛应用于 Web 应用程序的设计。

MVC 架构经历了发展和变化。最初，模型（Model）是程序的核心组件，负责管理应用程序的数据和业务逻辑，独立于用户界面。视图（View）则负责展示信息，而控制器（Controller）负责接收用户输入并将其转换为模型或视图的指令。随着 Web 应用的流行，业务逻辑逐渐从模型部分转移到控制器部分，模型变为仅负责数据存储的 Mapper。现代 MVC 架构，尤其在 Web 应用中，常采用这种设计。

在本程序中，MVC 架构被应用于两个主要部分：游戏大厅界面和对战界面。这两部分都采用 MVC 设计模式。游戏大厅负责管理多个对战界面，并保存这些对战界面的 Controller、Viewer 和 Mapper 对象的引用。一个游戏大厅可以同时管理多个对战界面，这些对战界面可以在同一设备或通过网络部署到多台设备上同时运行。

第二节　游戏大厅的架构和业务流程

游戏大厅的设计采用 MVC 架构，分为 Controller 层、Mapper 层和 Viewer 层。Controller 层负责业务流程的实现，并保存对 Mapper 层和 Viewer 层的引用。Controller 层使用单例模式，确保 Mapper 层和 Viewer 层能准确有效地获取 Controller 层。

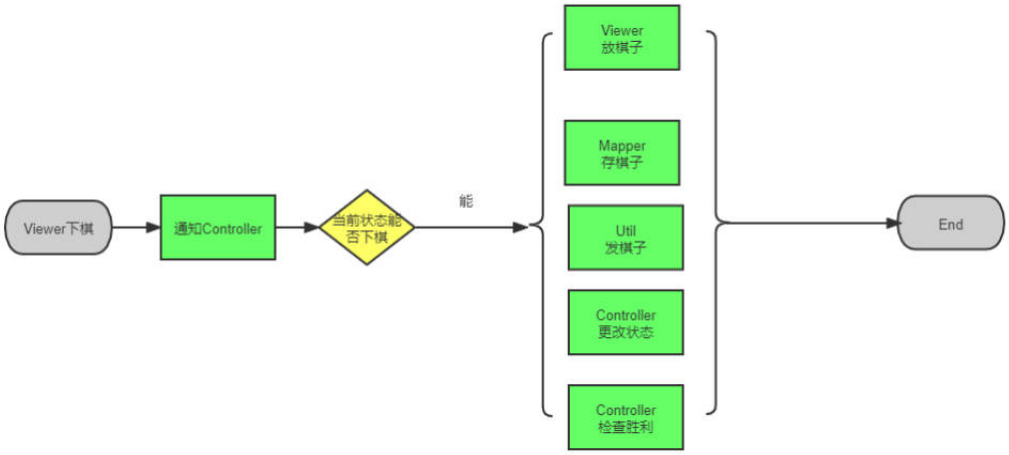

图 2.1 MVC 架构协同运作举例

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。

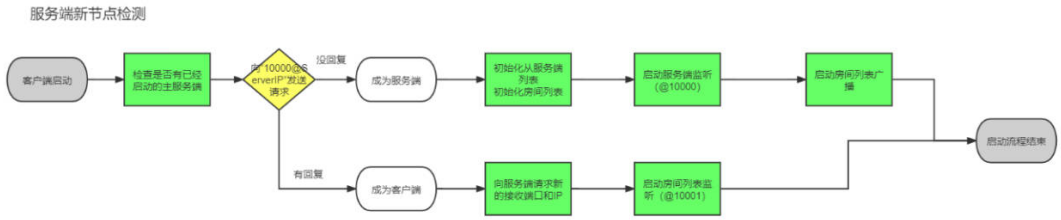

图 2.2 游戏大厅启动流程

在对战窗口启动时，若满足条件（如对战双方到齐或选择本地游戏），会依次创建新的 Controller、Mapper 和 Viewer 对象，并分配房间 Key 值及对战信息。通过主服务端的端口重定向机制，管理多个从服务端与主服务端之间的通信，确保房间和玩家的有效管理。

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

对战界面包括三种模式：网络对战模式、本地模式和复盘模式，均采用 MVC 架构设计。每种模式的对战界面都有独立的 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

保存对战状态、对战信息、玩家信息、棋盘状态、获胜者等信息，以及终止游戏的判断函数和相关的 Getter 和 Setter 方法。

在网络对战模式中，Controller 首先分配从游戏大厅获得的 Key 值给 Mapper 层和 Viewer 层，并初始化网络工具类。接着，设置对局信息并传递给 Viewer 层进行显示。进入“抢先后手”环节后，Viewer 层提示玩家选择棋子颜色。之后，Controller 开启 Post 线程进行消息发送和接收。

Controller 层根据不同的对战状态作出响应。它设有四种状态：空状态、下棋状态、等待状态和结束状态。例如，在等待状态下，Controller 只响应对方的消息，不响应本方的棋盘点击事件；在下棋状态下，Controller 响应本方棋盘点击，保存落子位置，通知 Post 发送网络信息并判断输赢。

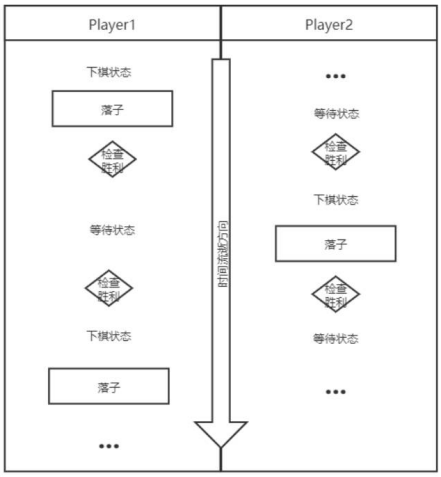

图 2.3 对战过程示意图

悔棋功能通过接收悔棋消息实现，Viewer 层将对应棋子设为不可见，Mapper 层保存悔棋信息。投降功能通过点击投降按钮触发，Controller 层修改输赢状态并通知 Viewer 层显示提示。保存存档功能允许用户将对战信息保存至文件，复盘时通过文件恢复玩家信息和棋局。

在本地对战和复盘模式中，操作进行了简化。例如，本地对战模式不启动网络发送，不显示聊天框和对方信息；复盘模式显示对战双方信息，不显示聊天框，仅提供上一步和下一步按钮。

第三章 GUI 设计

第一节 Java Swing 简介

Java Swing 是 Java 中用于创建图形用户界面（GUI，Graphical User Interface）的一个工具包，它是 Java 标准库的一部分。Swing 提供了一系列的类和接口，用于构建具有交互性和丰富用户体验的应用程序。以下是 Java Swing 的基本特点和介绍：

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。

1. **基于 Java 的跨平台性** Swing 是纯 Java 实现的 GUI 工具包，因此具有高度的跨平台性，可以在支持 Java 的任何操作系统上运行。
2. **轻量级组件**与早期的 AWT（Abstract Window Toolkit）相比，Swing 组件是轻量级的，即它们不依赖于本地平台的窗口系统。这使得 Swing 更加灵活和一致。
3. **丰富的组件库** Swing 提供了比 AWT 更加丰富的 GUI 组件，例如：
    - 按钮（JButton）
    - 标签（JLabel）
    - 文本框（JTextField）
    - 表格（JTable）
    - 树形控件（JTree）
    - 菜单（JMenu）
4. **可定制性强** Swing 支持外观和感觉（Look and Feel）的自定义，可以通过更改样式来使应用程序看起来符合特定平台的风格，或者使用第三方主题。
5. **事件驱动** Swing 使用事件监听器模式（Event Listener Pattern），所有的用户交互（例如鼠标点击、键盘输入）都通过事件处理来响应。

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。中央部分展示各个可用房间，由多个房间面板组成，用户可以创建新房间或加入已有房间。

图 3.1 游戏大厅界面

每个房间面板包括**房间名**、**Player1** 和 **Player2** 的昵称输入框以及**功能键**。房间面板有三种状态：空房间状态、待加入状态和对局开始后的状态。空房间状态下，用户可以输入昵称并点击“**ADD**”按钮创建房间并成为房主；待加入状态下，房主所在的客户端房间功能按钮变成灰色不可点击的“**WAITING**”，房间名改为主服务端分配的名称，其他客户端的另一玩家可以输入自己的昵称并点击”**JOIN!**”按钮开启新的对局；对局开始后，功能键变成灰色不可点击的“**On BATTLE**”，显示房间名和对战者昵称。

图 3.2 房间面板及其状态变化

上方有“**Local Game**”按钮，用于开始本地游戏；左侧有“**Music**”按钮，用于选择背景音乐；下方有“**Local Replay**”按钮，用于选择存档文件进行复盘。

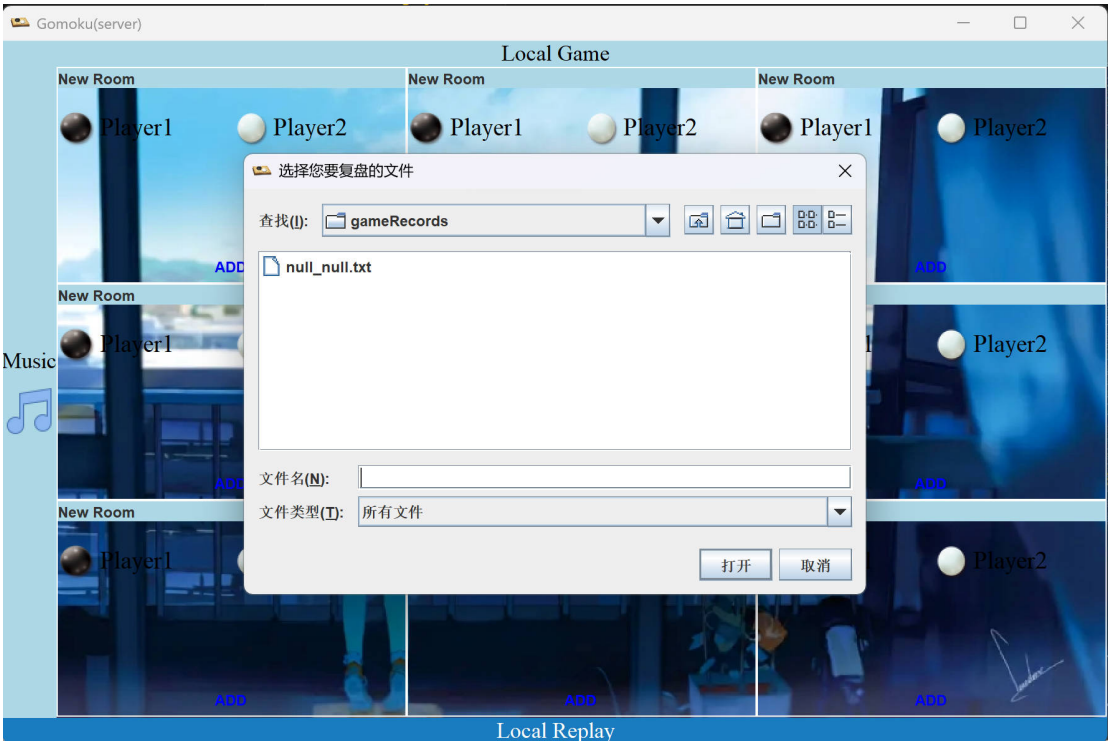

图 3.3 复盘游戏存档文件选择界面

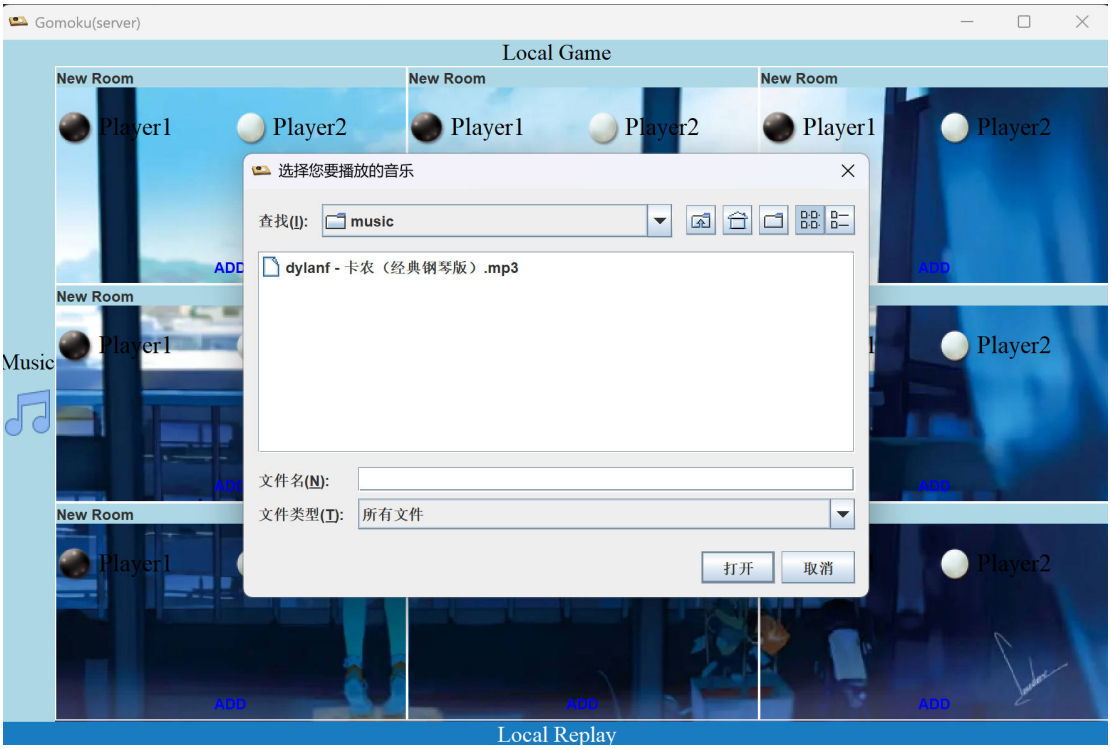

图 3.4 背景音乐文件选择界面

本地对战面板以及本地复盘面板将在下一节介绍。

第三节　对战面板的设计

对战界面由棋盘、玩家信息、功能键组、对话面板和倒计时牌等部分组成。棋盘背景为一张图片，覆盖 19×19 个透明 JPanel。玩家点击后，Viewer 层判断玩家持方并设置相应位置的背景为黑色或白色棋子，以显示落子。右侧功能键包括悔棋（Withdraw）、退出（Quit）、保存游戏存档（Save to File）和上一步（Previous）及下一步（Next）（在网络对战中不可用）。玩家信息显示两个玩家的昵称、对战端口号、IP 地址及持方。聊天面板包括显示聊天记录的 JTextArea、输入框 MessageBox 和发送按钮 Send。倒计时牌显示当前时间并以五分钟为限倒计时。

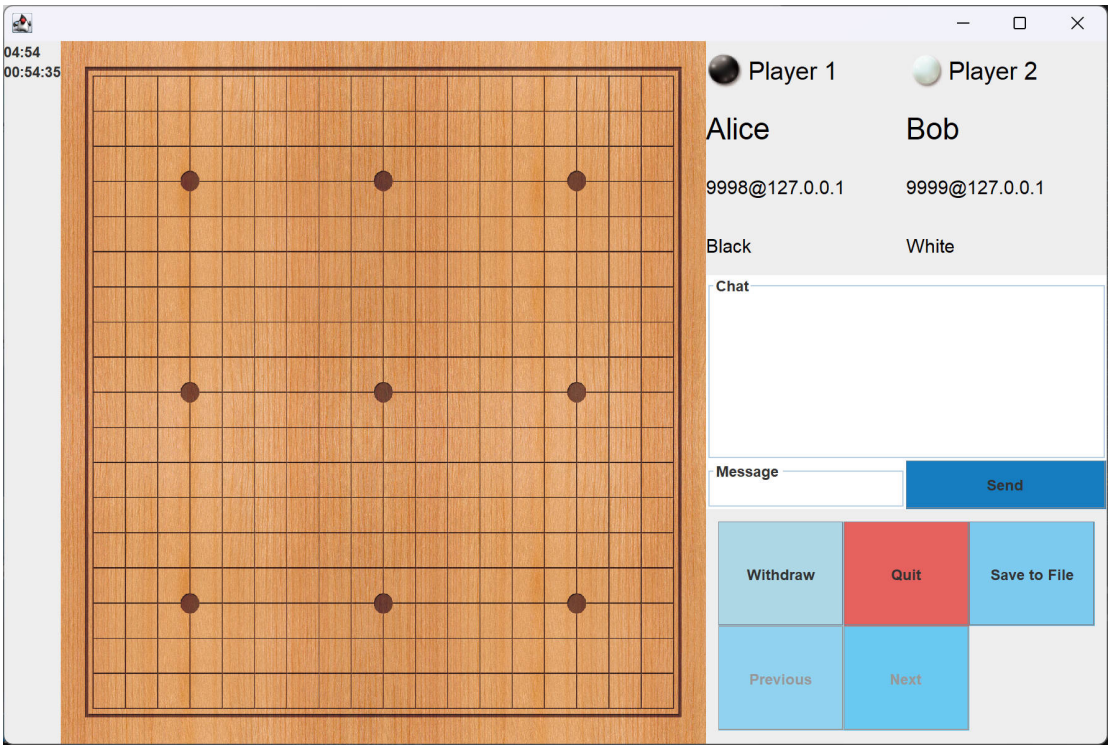

图 3.5 网络对战面板设计

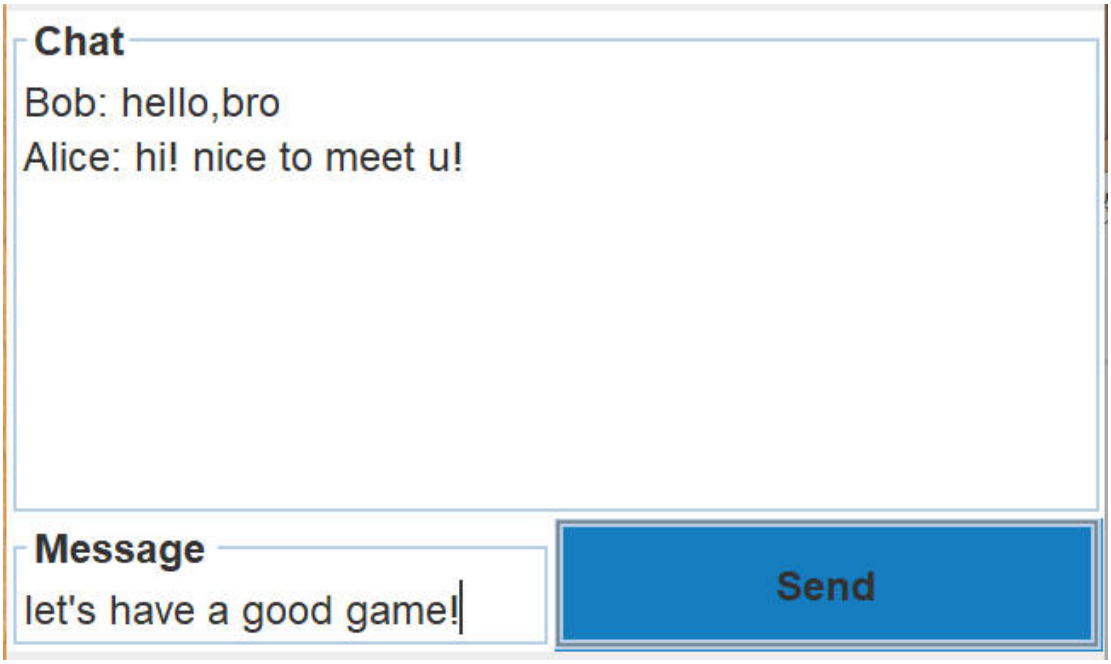

图 3.6 即时聊天面板

本地对战面板相较网络对战面板进行了简化，只保留棋盘、功能键和倒计时牌。

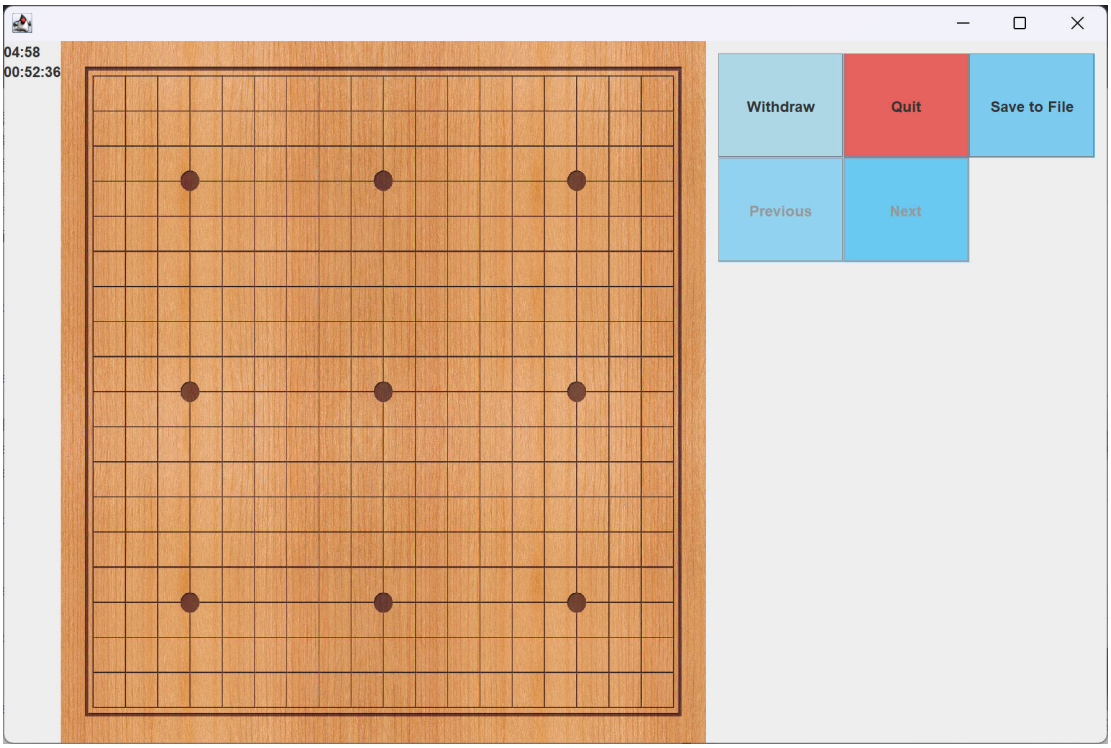

图 3.7 本地对局界面

复盘对战面板也在网络对战面板基础上简化，仅保留棋盘、功能键、玩家信息和倒计时牌。复盘者可以选择存档文件，并使用 Previous 和 Next 按键查看每一步。

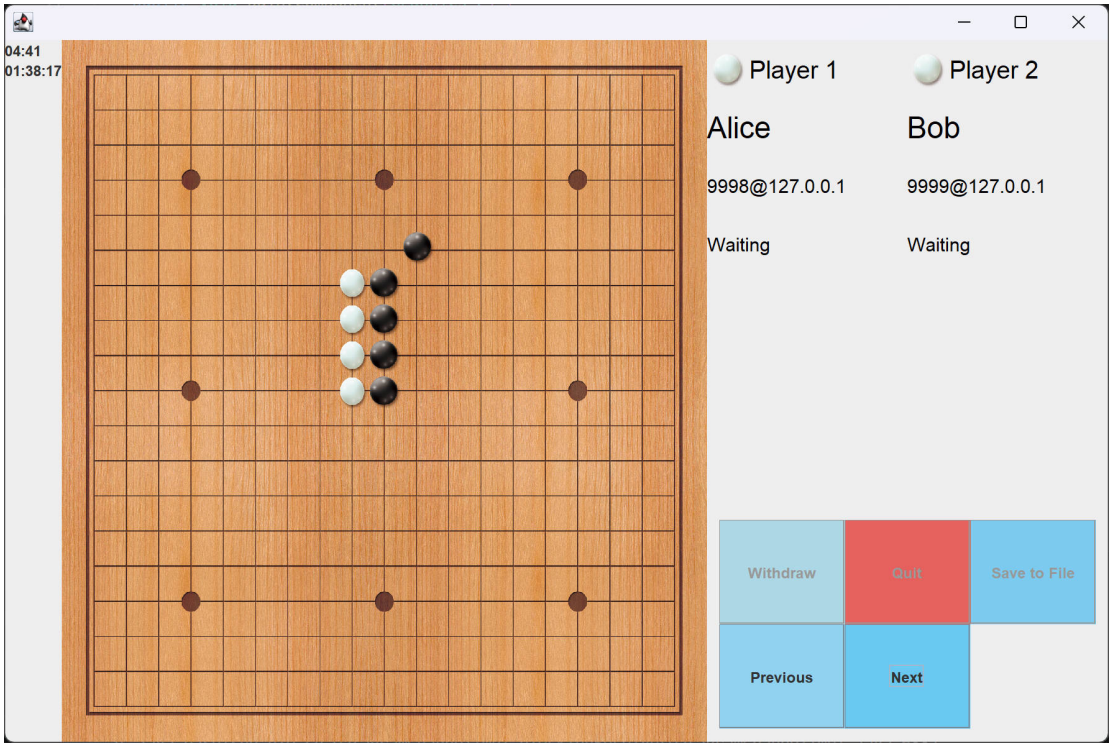

图 3.8 复盘面板设计

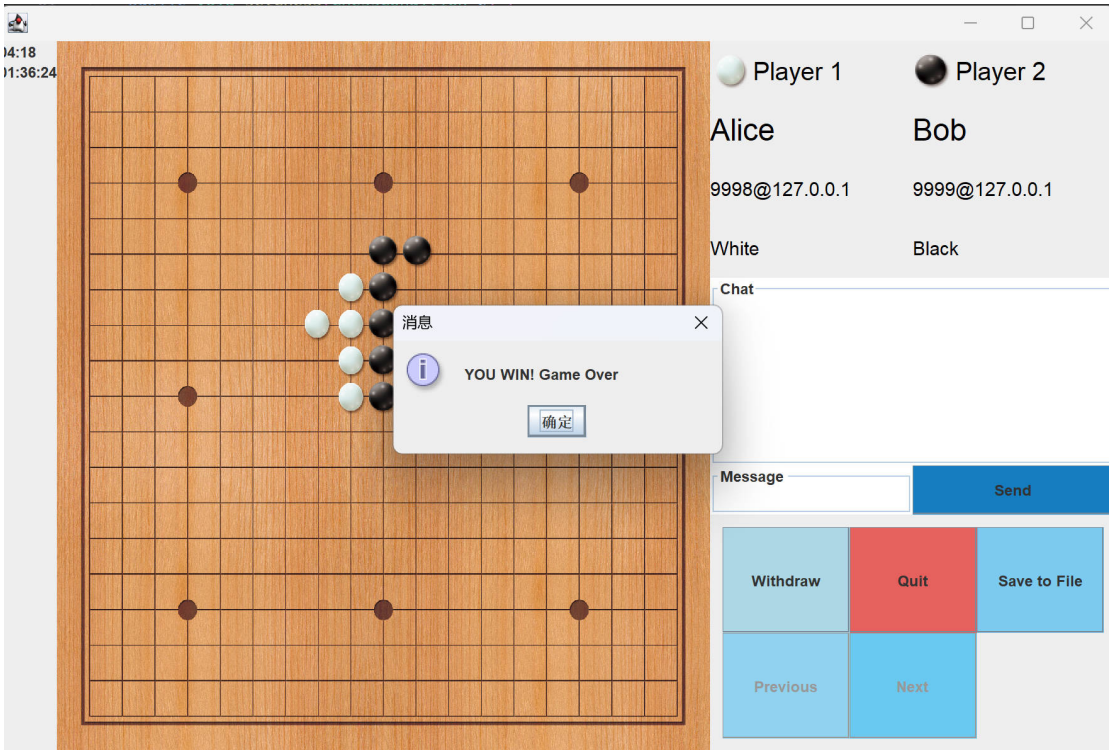

图 3.9 终局输赢判断

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。

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　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。

1. 主从节点通信端口分配

- 服务端启动与检测：每个从服务端在启动时，会向主服务端的默认端口发送检测消息，确认主服务端是否存在。
- 端口分配与通信：如果主服务端存在，它会为新的从服务端分配一个新的端口，并通过默认端口通知从服务端。之后，从服务端根据该端口进行通信，避免占用默认端口。如果主服务端不存在，这个服务端就会成为主服务端并开始广播线程。

- 端口管理：主服务端会记录每个从服务端的端口号和地址，确保主从服务端之间的通信使用独立分离的端口，避免端口拥堵和信息过滤困难，保证了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。。

图 4.1 主从服务端关系示意图

2. 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**。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。

图 4.2 端口分配示意图

第三节　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。

1. 主服务端更新机制

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。

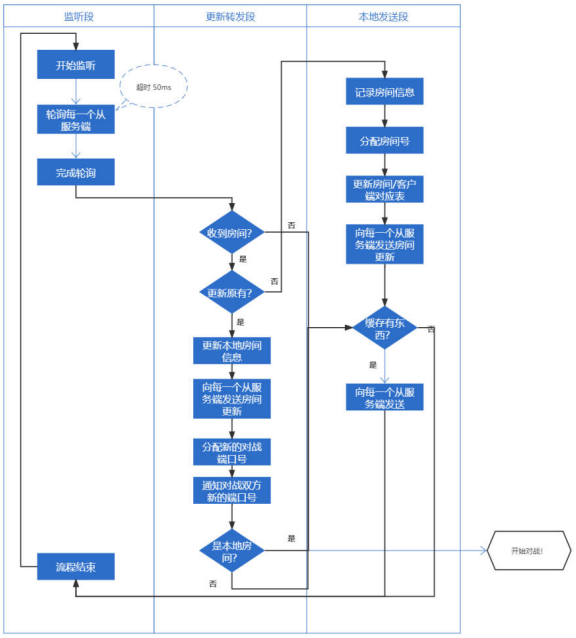

图 4.3 主服务端侧的房间信息更新转发流程

2. 从服务端更新机制

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。

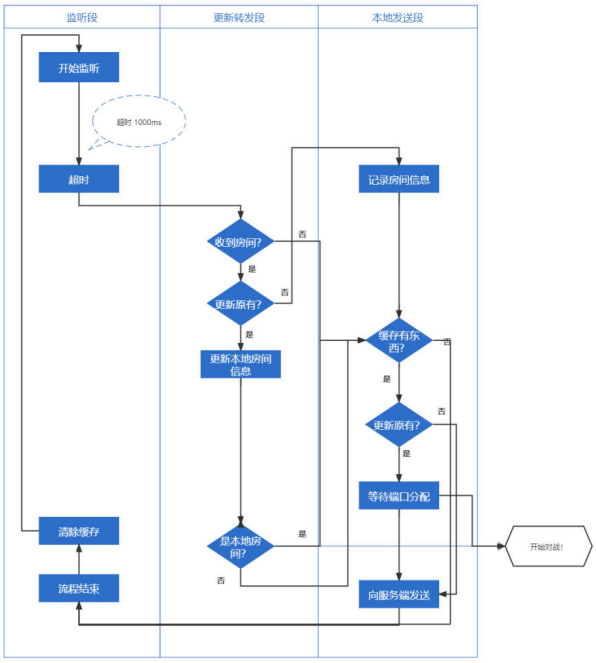

图 4.4 从服务端侧的房间信息更新转发流程

第四节 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。:

1. 数据接收

邮局线程在指定的对战通信端口进行监听，超时后判断是否收到对战相关消息。若收到消息，邮局通知 Controller，Controller 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处于等待对方落子的状态，Controller 会通知 Viewer 层进行落子。

如果是悔棋、投降等信息，Controller 会通知 Viewer 层进行相应处理。如果是文字信息，Controller 会通知 Viewer 层显示在文本框中。

2. 数据发送

邮局检查本方缓存中是否有待发送的数据。如果有，发送给对手并清除缓存。邮局开始下一轮的监听等待。

3. 数据打包

游戏中发送的各种信息都打包成“棋子”对象。每个棋子对象包含类型信息，用于判断其保存的内容，如下棋位置、颜色、悔棋位置、投降信息或聊天消息等。这种机制通过“邮局”线程和“棋子”对象的设计，实现了对战过程中数据的高效传输和处理。

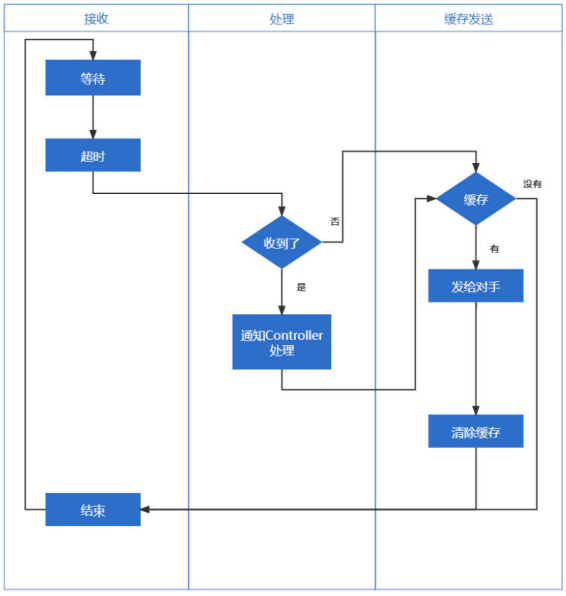

图 4.5 邮局线程工作流程

第五节　抢夺先后手环节

为了增加对战的趣味性，设计了一个争夺先后手环节。在对战面板打开时，系统首先弹出提示信息，提醒玩家选择先手或后手。玩家做出选择后，系统会自动记录玩家的持方和选择时间，并将这些信息封装成一个对象发送给对方。对方接收到该对象后，通过比较双方的选择时间来确定谁先抢到持方，从而在双方选择相同持方的情况下决定最终的持方归属。

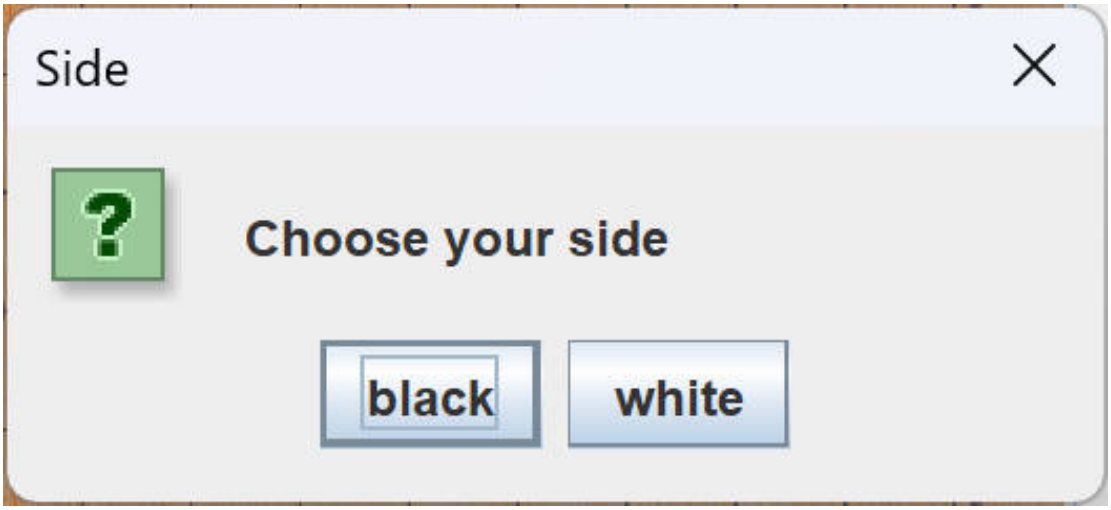

图 4.6 选择持方提示信息

第五章　Maven 项目构建和背景音乐功能插件

本程序使用 Maven 构建项目。Maven 是一个流行的 Java 项目管理和构建工具，具有自动化构建、依赖管理、标准化项目结构、跨平台支持、丰富的插件生态系统等优势。本项目通过在**.xml**（Maven 的配置文件）文件中添加 **javazoom.jl.player** 包依赖，实现了背景音乐播放功能。

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。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。

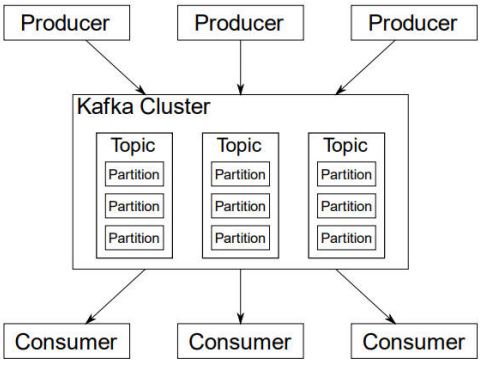

图 6.1 Kafka 架构示意图

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。