在线五子棋游戏实验报告

Experiment Report: Online Gomoku Game

廖望 @github.com:aokimi0

2nd-year UG student of College of Computer Science, Nankai University

2024.12.20

南開大學
Nankai University

Contents

Java 简介

Java 是一种广泛使用的面向对象的编程语言，由 Sun Microsystems 在 1995 年首次发布，后来被 Oracle 公司收购。Java 的设计理念是“一次编写，处处运行”（Write Once, Run Anywhere, WORA），即编写的程序可以在不同的平台上运行，前提是该平台有合适的 Java 虚拟机（JVM）。Java 具有跨平台性、面向对象、网络编程、垃圾回收等特点。它的应用广泛，尤其在企业级应用、Web 开发、大数据处理和移动设备（如 Android）开发中有着重要地位。

Java 的主要特点包括：

- **平台独立性**： Java 程序运行依赖于 JVM，任何支持 JVM 的操作系统都可以运行 Java 程序。
- **面向对象**： Java 强调类和对象的使用，支持封装、继承和多态等面向对象的基本特性。
- **多线程支持**： Java 提供内建的多线程机制，适用于需要并发的程序设计。
- **垃圾回收**： Java 自动管理内存，程序员无需手动释放内存，从而避免内存泄漏问题。
- **丰富的类库**： Java 提供了大量标准类库，包括数据结构、输入输出、网络通信等，使开发变得更加简单。

Java 被广泛应用于各种类型的开发，包括桌面应用、Web 应用、企业应用、移动应用和大数据处理。

五子棋简介

五子棋是一种源自中国的传统棋类游戏，通常由两人对弈。游戏的目标是在棋盘上通过连续五个相同颜色的棋子（横、竖、斜都可以）连成一线来获胜。五子棋规则简单，操作灵活，但也具有极高的策略性，因而深受世界各地玩家的喜爱。

五子棋的主要规则如下：

- 游戏使用一个正方形棋盘，最常见的是 15x15 或 19x19 的棋盘。
- 两名玩家轮流在棋盘上放置棋子，通常使用黑白两种颜色的棋子。
- 第一个在水平、垂直或斜线方向上连成五个相同颜色棋子的玩家获胜。
- 游戏没有固定的回合数，直到有玩家胜出或棋盘满为止。

五子棋的策略性体现在开局、布局、进攻和防守等多个方面。尽管游戏规则简单，但要掌握高级策略，尤其是预判对手的意图和安排棋局，则需要较高的技巧。五子棋除了作为娱乐消遣外，也常常成为计算机科学研究中的一个研究对象，尤其是在人工智能和算法研究方面。

Contents

MVC 架构与 Maven 构建

MVC（模型-视图-控制器）架构是一种常用于用户界面开发的软件设计模式，将程序逻辑分为三个部分：模型（Model）、视图（View）和控制器（Controller），旨在分离应用程序的内部数据处理与用户界面的展示和输入。最初用于桌面应用程序开发，后来广泛应用于 Web 应用中。

随着 Web 应用的发展，传统 MVC 架构的模型负责数据和业务逻辑，而视图负责展示，控制器接收用户输入并指引模型或视图的操作。然而，在现代 Web 应用中，业务逻辑逐步移至控制器，而模型仅负责数据存储，通常被称为 Mapper。

在本程序中，MVC 架构应用于**游戏大厅界面**和**对战界面**两个主要部分。游戏大厅负责管理多个对战界面，并保存这些对战界面的控制器（Controller）、视图（Viewer）和模型（Mapper）的引用。一个游戏大厅可以同时管理多个对战界面，这些界面可以在同一设备或跨多个设备运行。

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。

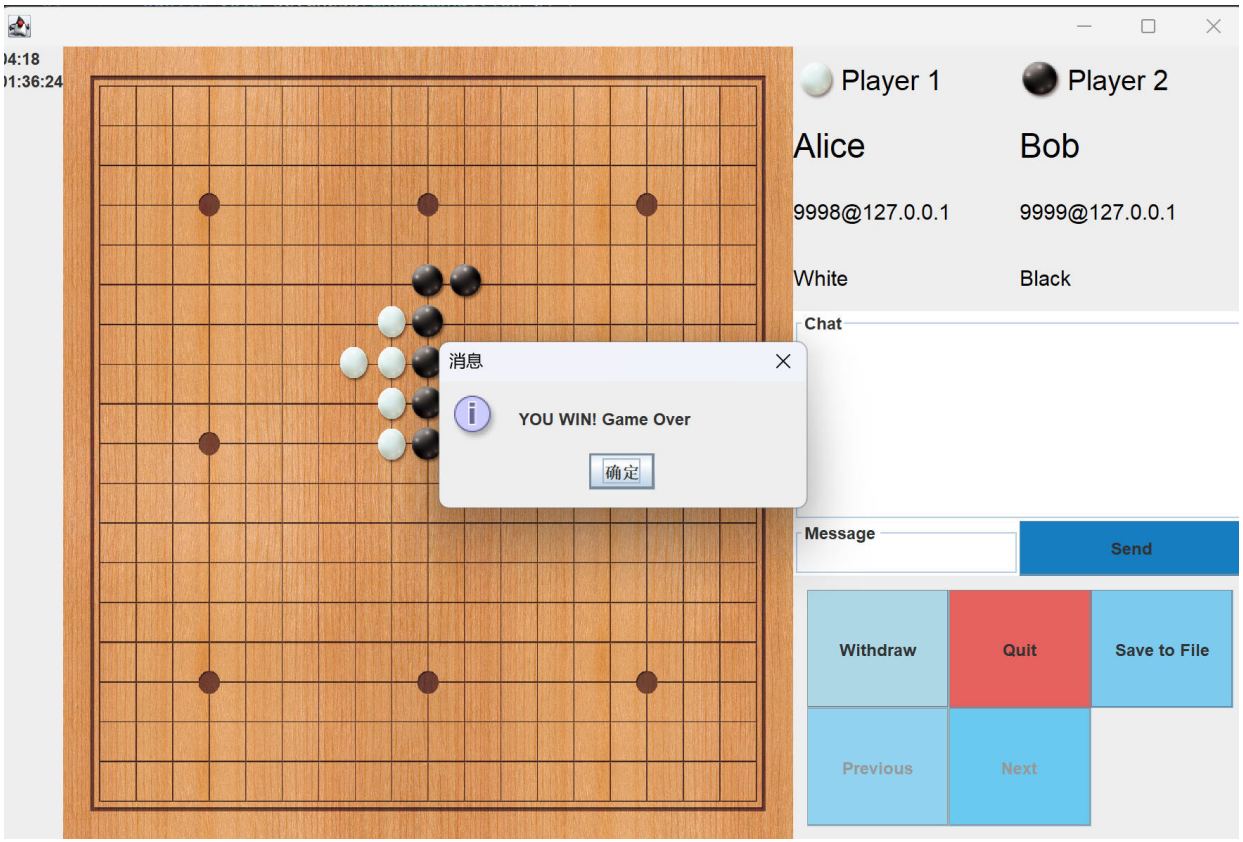
本程序使用 **Maven** 构建项目，并通过在**.xml**（Maven 的配置文件）文件中添加 **javazoom.jl.player** 包依赖，实现了背景音乐播放功能。

GUI 设计

Border Layout 布局。

游戏大厅界面

GUI 设计



网络对战面板设计

本地对战面板和复盘面板相较网络对战面板进行了简化。

通信分离机制

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。

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。为此设计了两种通信分离机制：

1. 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。
2. 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。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。

主从服务端通信

主从服务端关系示意图

端口分配

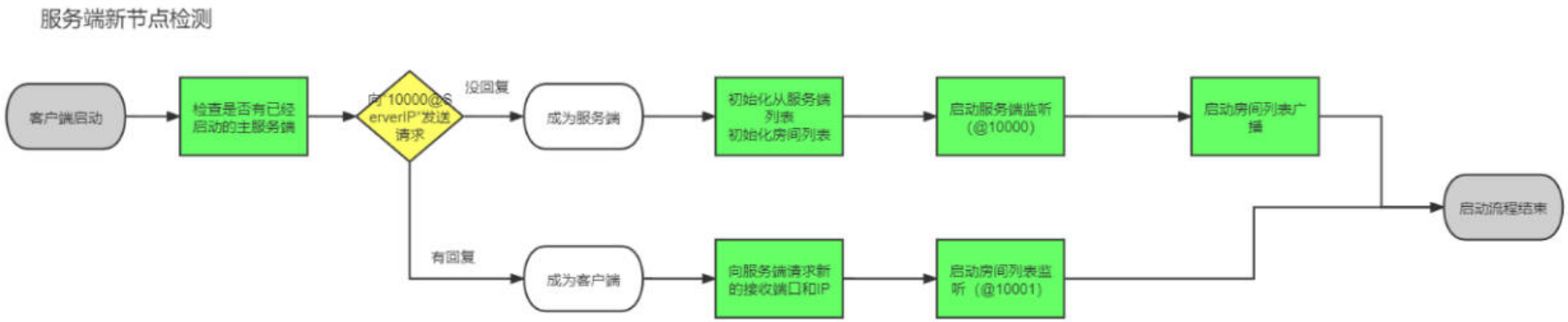
端口分配示意图

Contents

游戏大厅的架构和业务流程

游戏大厅采用 MVC 架构，包括 Controller 层、Mapper 层和 Viewer 层。Controller 层负责业务逻辑，管理与 Mapper 和 Viewer 层的引用，并使用单例模式确保有效性。游戏大厅有主服务端和从服务端两种启动模式，启动过程中，检查主服务端是否已运行，若有则启动从服务端模式并更新房间列表，若无则启动主服务端模式并进行广播和房间同步。

对战窗口启动时，会创建新的 Controller、Mapper 和 Viewer 对象，分配房间 Key 值。主服务端通过端口重定向机制管理与从服务端的通信，确保房间和玩家的有效管理。Mapper 层维护多个列表，管理房间、客户端和对战信息，Viewer 层负责界面显示和更新。



游戏大厅启动流程

房间面板及其状态变化

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。

房间面板及其状态变化

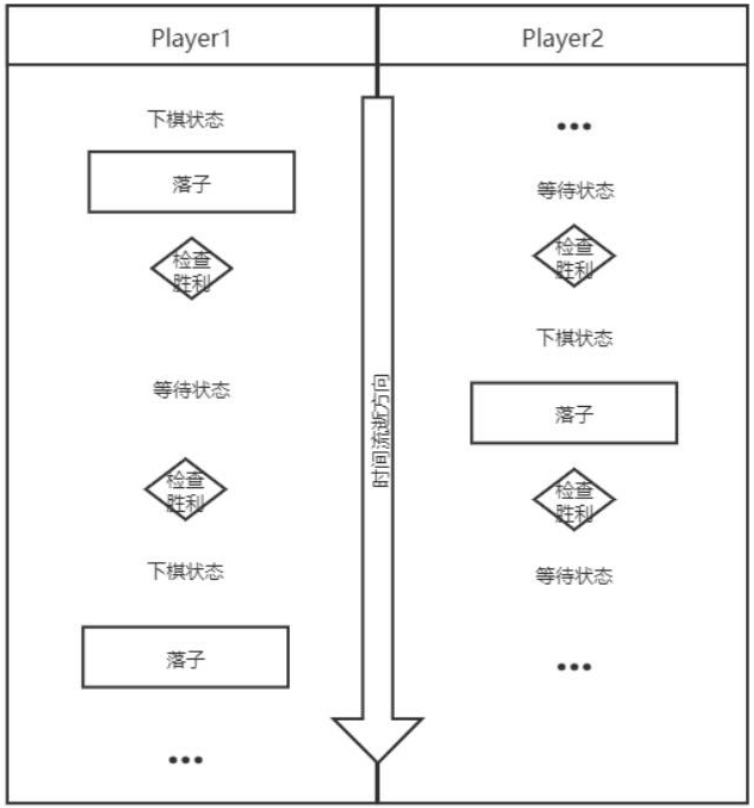
对战页面的架构和业务流程

对战界面包括网络对战模式、本地模式和复盘模式，均采用 MVC 架构设计。每种模式都有独立的 Controller 层、Mapper 层和 Viewer 层，Controller 层负责存储对战状态、玩家信息、棋盘状态、获胜者等，并提供相应的 Getter 和 Setter 方法。

在网络对战模式中，Controller 层首先分配 Key 值给 Mapper 和 Viewer 层，并初始化网络工具类。之后，设置对局信息并传递给 Viewer 层，进入“抢先后手”环节后提示玩家选择棋子颜色，并启动线程进行消息发送和接收。Controller 根据不同状态（如空状态、下棋状态、等待状态和结束状态）作出响应，处理棋盘点击和网络信息传输。悔棋和投降功能分别通过接收消息和按钮触发，保存信息并更新界面。保存存档功能允许将对战信息保存至文件，复盘时恢复玩家和棋局信息。

在本地对战和复盘模式中，操作简化，不启动网络通信，复盘模式仅提供上一步和下一步按钮，不显示聊天框。

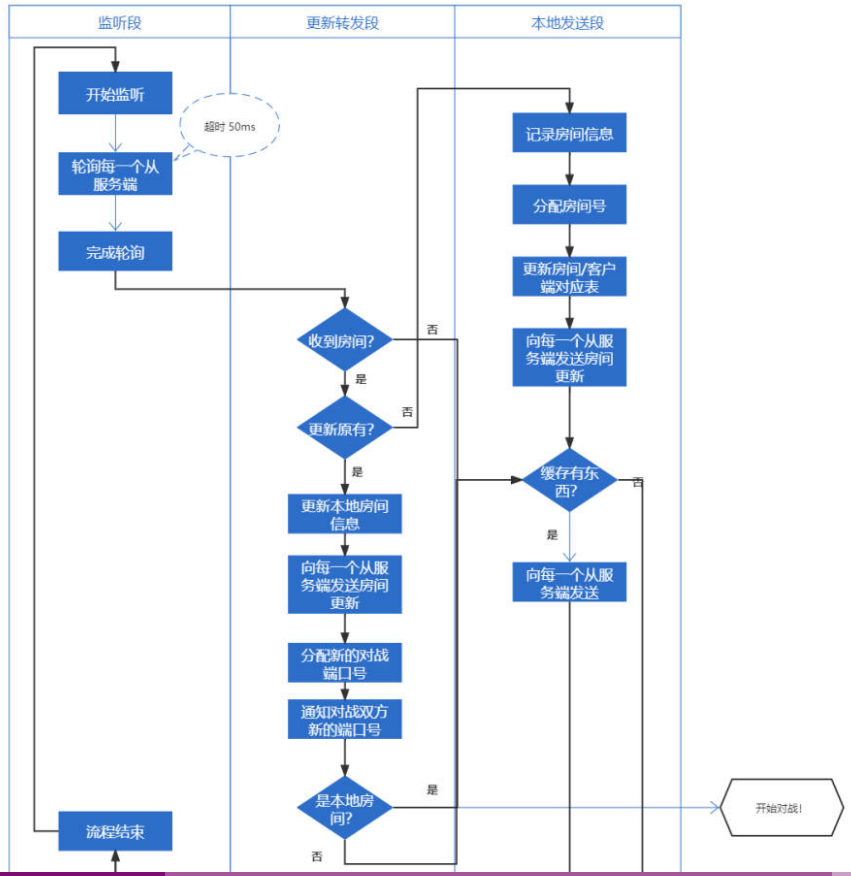
对战页面的架构和业务流程



对战过程示意图

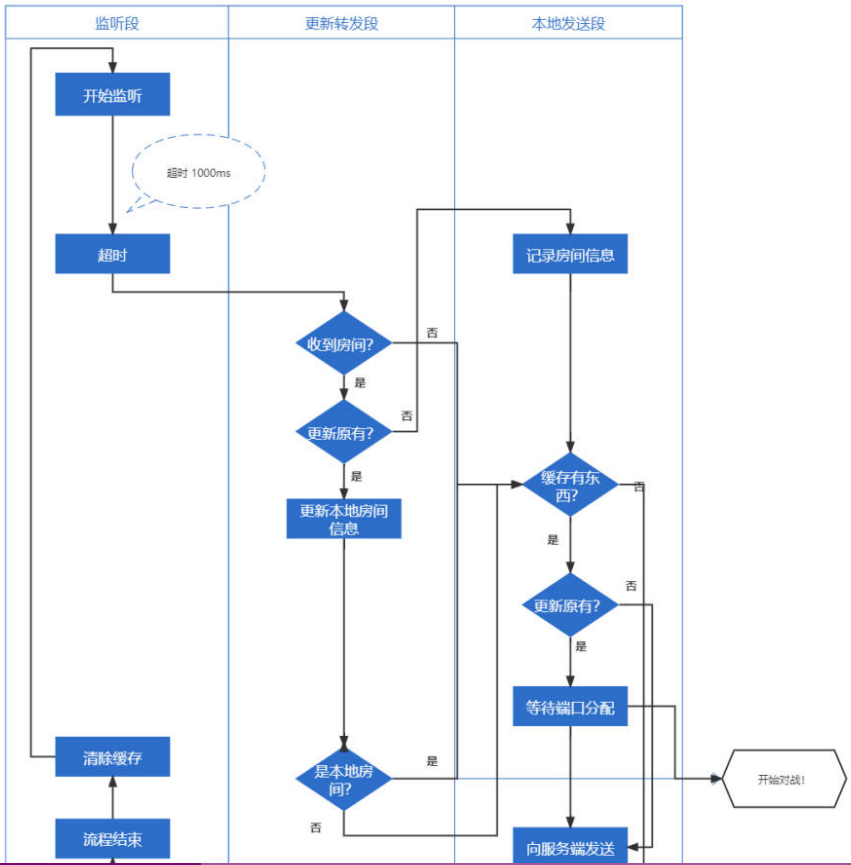
房间列表的更新与同步

在主服务端，监听线程轮询从服务端的端口，检查是否有新的更新信息。若有，判断是加入已有房间还是创建新房间。加入房间时，分配新的对战端口并通知双方开始点对点对战；若是新房间，更新房间信息并分配房间号。主服务端将更新的房间信息发送给从服务端，若有缓存数据，则发送缓存内容。流程结束后，监听线程进入下一轮轮询。



房间列表的更新与同步

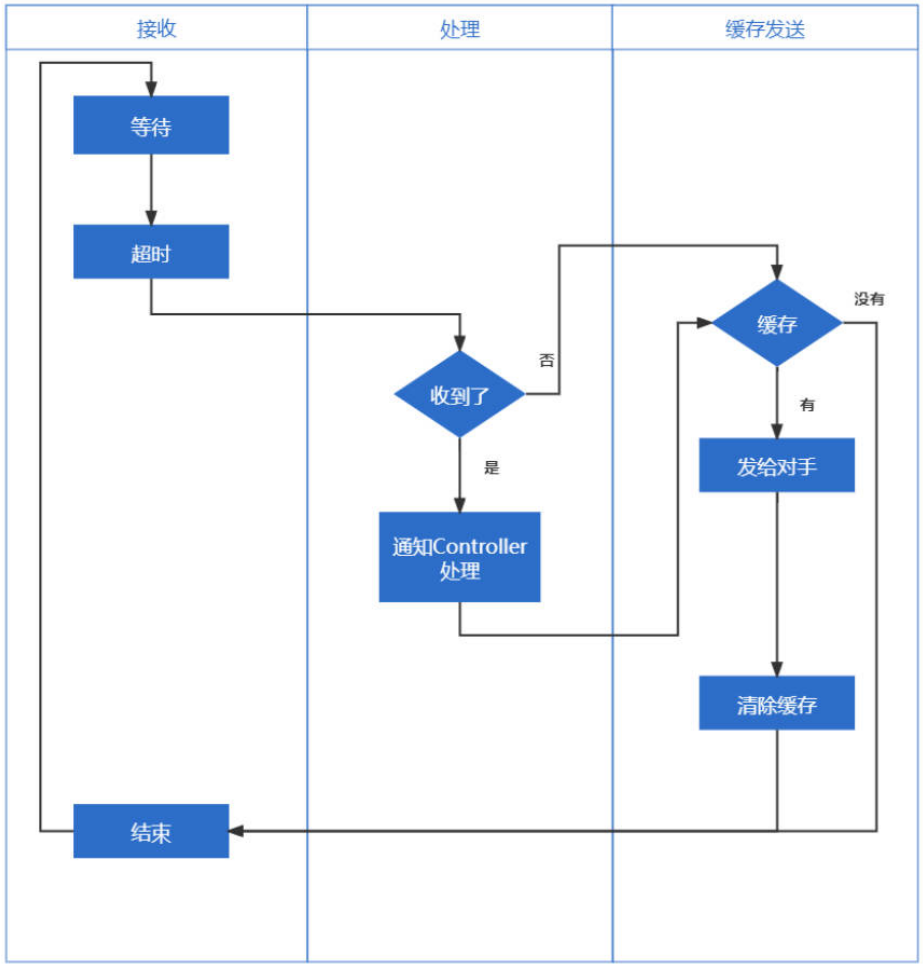
在从服务端节点，接收线程监听超时为 1000 毫秒。当收到房间信息时，分为两种情况：一是主节点发来创建新房间的信息，更新本地房间信息；二是本地服务端收到新对战者加入已有房间的信息，开始对战并更新本地房间记录。若本地缓存中有待发送的信息，则向主节点汇报。汇报后清除缓存，进入下一轮监听。



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收发，设计了一个“邮局”（Post）线程。该线程在指定的对战通信端口进行监听，若超时后收到消息，通知 Controller 进行处理。根据消息类型，Controller 会通知 Viewer 层进行相应操作，如落子、悔棋、投降或显示文字信息。邮局还会检查本方缓存中是否有待发送的数据，并发送给对手，清除缓存后开始下一轮监听。数据通过“棋子”对象进行打包，每个“棋子”对象包含类型信息，指示保存的内容（如落子位置、颜色、悔棋位置、投降信息或聊天消息）。这种机制提高了对战过程中的数据传输和处理效率。

点对点对战过程中的通信

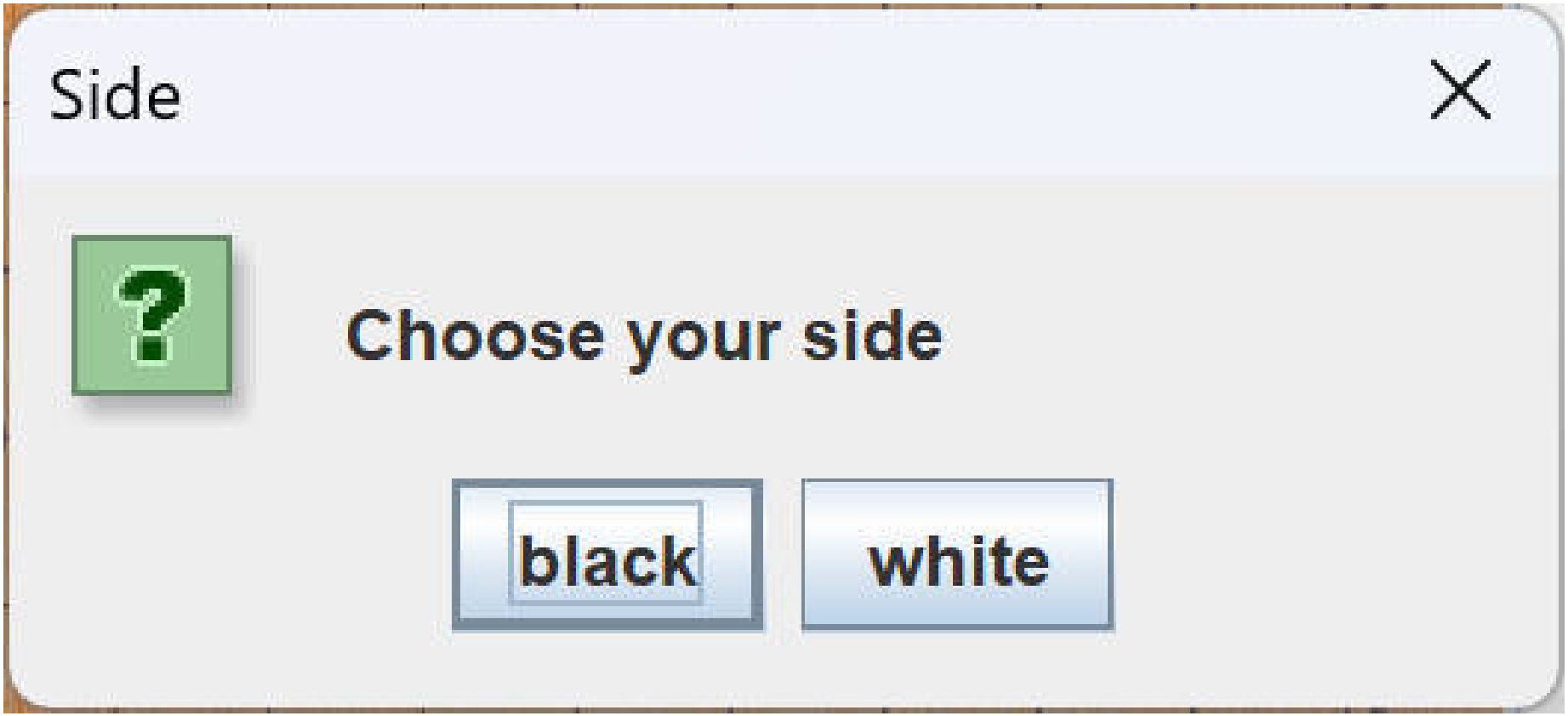


局线程工作流程

邮

其他功能

游戏还实现了抢夺先后手、文字聊天、悔棋、认输、保存对战记录、复盘、本地对战和其他装饰性功能，如计时器和背景音乐等。



选择持方提示信息

Contents

总结与展望

1. **引入 Kafka 架构与旁观模式**。由于当前对战环节的通信复杂且信息丢失概率较高，程序未实现观战模式。为优化信息传输过程，计划引入 Kafka 架构。Kafka 是一个高吞吐量、低延迟的实时数据流平台，采用发布/订阅模式，将不同类型的消息存储在不同的 topic 中，确保信息及时分发。每个房间将有独立的 topic，确保对战双方和观战方能接收到相关的战局更新，从而解决了信息传输复杂性和观战功能缺失的问题。
2. **主服务端和从服务端的分离**。原始程序中，游戏大厅同时承担主服务端和从服务端的角色，导致业务逻辑复杂且维护困难。优化计划是将服务端和客户端功能分开：服务端仅负责客户端之间的通信和数据同步，而客户端负责实际的对战。这一优化简化了代码，改善了房间同步和转发流程，提高了系统的可扩展性和维护性。
3. **通信线程的优化**。当前程序的通信线程采用轮询或超时后发送的模式，可能导致在发送过程中无法接收到对方的消息，从而存在信息丢失的隐患。为了解决这一问题，下一步计划使用多线程方式优化通信过程。监听线程将独立运行并保持监听状态，仅在需要发送数据时暂停监听并立即发送，之后重新启动监听线程。这样可以减少信息丢失的概率，尤其是在双方同时发送消息时，降低遗漏的风险。

Thank you!