

南 开 大 学

Java 课 程 设 计

中文题目: 基于 TCP 通信的网络五子棋

外文题目: Network gobang based on TCP communication

学 号: 2012677

姓 名: 刘宇轩

年 级: 2020 级

学 院: 计算机学院

系 别: 计算机科学与技术

专 业: 计算机科学与技术

完成日期: 2021 年 12 月

指导教师: 刘嘉欣

摘 要

JAVA 课程大作业的要求实现一个能够进行网络通信的五子棋小游戏，其中包含基本的下棋操作，能够进行悔棋请求，结束后能够进行复盘操作，实现双方文字聊天等功能。本项目选用 CS 的网络通信设计框架，采用主服务器和子服务器结合通信，为同时在线的多个客户端玩家提供通信服务。客户端玩家通过和主服务端进行请求进入游戏大厅，根据客户端玩家的选择，将请求处理权限转移给对应的房间子服务端，进而使同一房间内的客户端玩家通过子服务端，采用 RPC 框架进行过程调用。在客户端游戏界面则采用传统的 MVC 设计框架。在实现课程要求的基础上，还增加了与数据库连接的登陆界面，游戏大厅，旁观模式以及背景音乐及音效等附加功能。

关键词：Java，网络通信，多线程，五子棋，MVC

Abstract

The requirement of assignment in JAVA course is to realize a small game of five chesses that can be used for telecommunication, which includes basic chess operation, regret chess request, and review operation after the end, so as to realize the function of text chat between the two sides. This project selects the CS telecommunication design framework, using the main server and sub-server communication, to provide communication services for multiple client players online at the same time. The client player enters the game hall by requesting with the main server. According to the client player's choice, the request processing permissions are transferred to the corresponding room sub-server, so that the client player in the same room passes the sub-server and uses the RPC framework for process calls. The traditional MVC design framework is used in the client game interface. On the basis of realizing the curriculum requirements, it also adds additional functions such as login interface, game hall, bystander mode, background music and sound effect connected with database.

Key Words: Java, telecommunication, multithread, five chess, MVC

目录

摘要	I
Abstract	II
目录	III
第一章 前言	1
第二章 前期调研	2
第一节 游戏市场	2
第二节 开发框架	2
2.2.1 基于 CS 架构的 TCP 网络通信	2
2.2.2 基于 RPC 架构的请求处理	3
2.2.3 基于 MVC 架构的客户端游戏界面	5
第三章 项目架构	6
第一节 整体框架	6
第二节 整体流程	9
第三节 服务端	11
3.3.1 主服务端	13
3.3.2 子服务端	13
第四节 客户端	15
3.4.1 控制端	15
3.4.2 游戏界面	17
第五节 网络通信	19
第六节 数据库存储	20
第四章 项目效果	21
第五章 总结和展望	25
参考文献	27
致 谢	XXVIII

第一章 前言

五子棋作为中华民族优秀传统文化，有着近千年的悠久历史，但随着当今智能时代的到来，各类网络游戏已经占据了人们日常娱乐的大部分时间，五子棋这项悠久的传统棋类对弈策略游戏，已经逐渐淡出人们的日常生活。而当前已有的一些网络五子棋对战游戏，存在着大量的广告推广，充值服务，商业化程度较重，失去了原本下棋简单的快乐。为了解决人们平时进行简单网络对局的需求，因此开发了此款基于 JAVA 的网络五子棋小游戏。

该项目既能够传承和发扬五子棋这项优秀的传统文化，又能够综合运用本学期学习的 JAVA 内容，将逻辑控制，图形化界面开发，网络通信，多线程并发与同步，数据库存储等技术综合运用，很好地锻炼自身的项目设计能力和开发能力。

本文将在第二章前期调研中介绍，选择采用 JAVA 语言，基于 CS 架构、RPC 架构和 MVC 架构进行项目开发的优势。在第三章具体介绍项目各端如何结合 CS 架构、RPC 架构和 MVC 架构实现逻辑功能和网络通信。在第四章中将展示项目的具体运行效果。第五章中则对项目进行简要的总结概括，并对项目未完成待扩展的功能进行展望。

项目相关代码发布在 Github, <https://github.com/NKULYX/WebGobang>

项目效果视频发布在 bilibili, <https://www.bilibili.com/video/BV14m4y1X7YA>

第二章 前期调研

第一节 游戏市场

游戏，作为人们日常消遣的重要途径，在人们的生活中已经占据了十分重要的地位。在 2019 年中国游戏市场实销售收入达到 2308.0 亿元，同比增长 7.7%。从细分市场观察，移动端游戏占整体销售收入近七成，处于主导地位；客户端于网页游戏占比分别降到 26.6% 和 4.3%。2020 年中国游戏市场的占比中，游戏产业接近 3000 亿的市场规模里，国内游戏市场的实际营销总额为 2786.87 亿元，比上一年增加了 478.1 亿元，同比增长 20.71%。与此同时，游戏用户数量也保持着稳定的增长，规模已经超过 6.65 亿人，同比增长 3.7%。随着 00 后甚至 10 后用户市场的逐渐扩大，这一占比趋势也正在继续扩大，游戏市场的格局也发生着翻天覆地的变化。^[1]

第二节 开发框架

2.2.1 基于 CS 架构的 TCP 网络通信

TCP 是一种面向连接的、可靠的、基于字节流的传输层通信协议，旨在适应支持多网络应用的分层协议层次结构。连接到不同但互连的计算机通信网络的主计算机中的成对进程之间依靠 TCP 提供可靠的通信服务。是基于流的方式，面向连接的可靠通信方式。

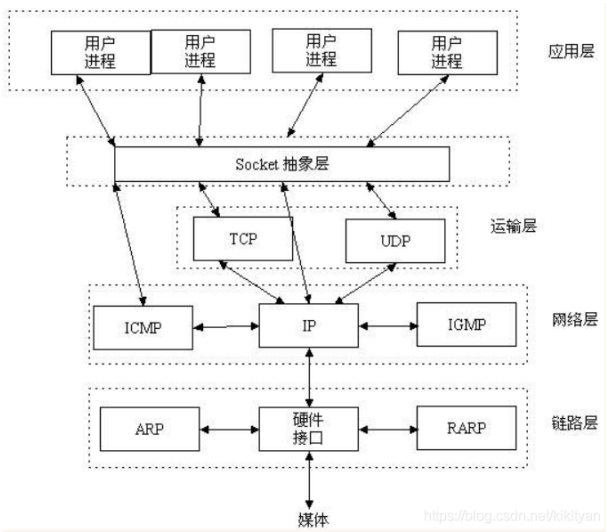


图 2.1 TCP 架构

CS 即：client/server，是服务器客户端结构。是一种“一对多”的模式，一台服务器，通过 Socket 接受处理多个客户端发来的请求，完成了业务逻辑之后，再将结果信息返回给客户端。其中，服务器不会主动向客户端发起请求，都是被动接受客户端的请求并进行处理。而客户端则通过用户的操作，向服务端发起对应的请求，再将服务端的结果返回与用户进行交互。

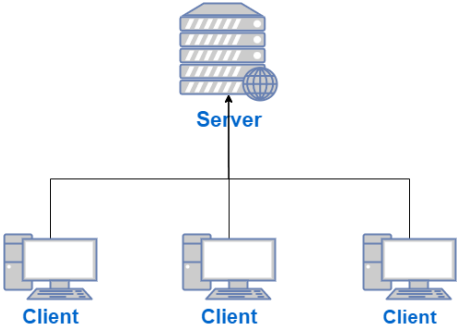


图 2.2 CS 架构

2.2.2 基于 RPC 架构的请求处理

RPC（Remote Procedure Call）远程过程调用协议，一种通过网络从远程计算机程序上请求服务，而不需要了解底层网络技术的协议。

Client: RPC 协议的调用方。就像上文所描述的那样，最理想的情况是 Client 在完全不知道有 RPC 框架存在的情况下发起对远程服务的调用。但实际情况

来说 Client 或多或少的都需要指定 RPC 框架的一些细节。Server: RPC 协议的处理方。在 RPC 规范中, 这个 Server 并不是提供 RPC 服务器 IP、端口监听的模块。而是远程服务方法的具体实现 (在 JAVA 中就是 RPC 服务接口的具体实现)。其中的代码是最普通的和业务相关的代码, 能够实现调用方的方法调用, 并返回所需要的结果给调用方。

具体流程如下:

1. 客户端 client 发起服务调用请求。
2. client stub 可以理解成一个代理, 会将调用方法、参数按照一定格式进行封装, 通过服务提供的地址, 发起网络请求。
3. 消息通过网络传输到服务端。
4. server stub 接受来自 socket 的消息。
5. server stub 将消息进行解包、告诉服务端调用的哪个服务, 参数是什么。
6. 结果返回给 server stub。
7. sever stub 把结果进行打包交给 socket。
8. socket 通过网络传输消息。
9. client stub 从 socket 拿到消息。
10. client stub 解包消息将结果返回给 client。

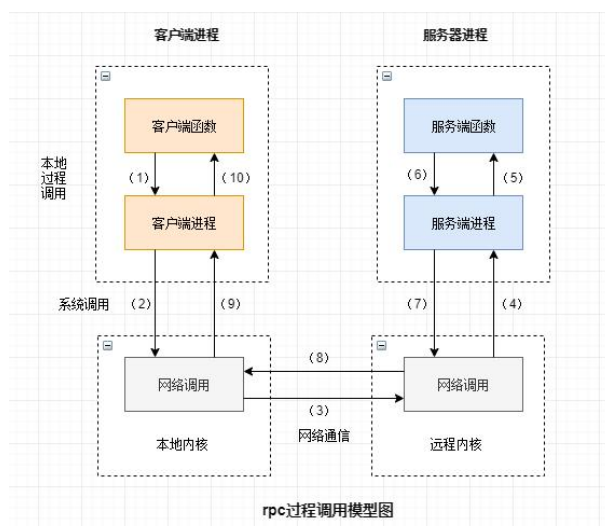


图 2.3 RPC 架构

2.2.3 基于 MVC 架构的客户端游戏界面

MVC 的全名是 Model View Controller，是模型 (model) — 视图 (view) — 控制器 (controller) 的缩写，是一种软件设计典范。它是用一种业务逻辑、数据与界面显示分离的方法来组织代码，将众多的业务逻辑聚集到一个部件里面，在需要改进和个性化定制界面及用户交互的同时，不需要重新编写业务逻辑，达到减少编码的时间。

V 即 View 视图是指用户看到并与之交互的界面，例如软件的客户端界面。MVC 的好处之一在于它能为应用程序处理很多不同的视图。在视图中其实没有真正的处理发生，它只是作为一种输出数据并允许用户操纵的方式。

M 即 model 模型是指模型表示业务规则。在 MVC 的三个部件中，模型拥有最多的处理任务。被模型返回的数据是中立的，模型与数据格式无关，这样一个模型能为多个视图提供数据，由于应用于模型的代码只需写一次就可以被多个视图重用，所以减少了代码的重复性。

C 即 controller 控制器。是指控制器接受用户的输入并调用模型和视图去完成用户的需求，控制器本身不输出任何东西和做任何处理。它只是接收请求并决定调用哪个模型构件去处理请求，然后再确定用哪个视图来显示返回的数据。

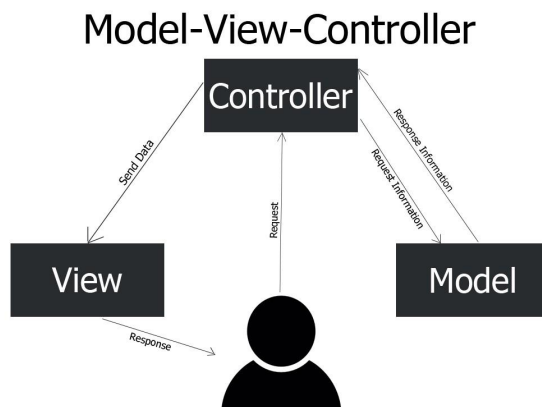


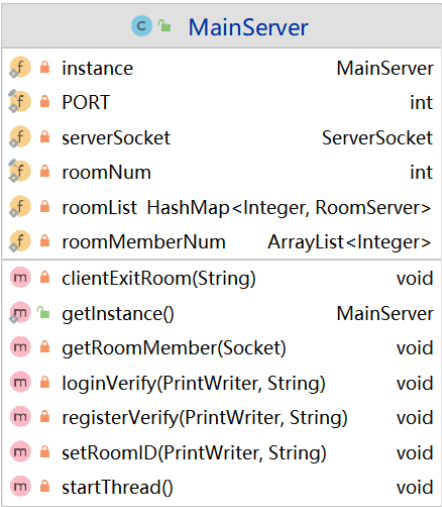
图 2.4 MVC 架构

第三章 项目架构

第一节 整体框架

游戏主要分成 5 大部分，主服务端，房间子服务端，客户端，棋局模型和游戏界面，除此之外还有一个命令控制类。

主服务端中需要和存储用户信息的数据库建立连接，负责接受用户刚建立连接时的注册和登录请求，以及进入游戏大厅后的数据拉取和房间选择，在接收到客户端的房间选择后，就会将该房间服务端对应的端口号返回给客户端，之后该客户的请求处理权便转移给了对应房间的子服务端。



MainServer		
f	instance	MainServer
f	PORT	int
f	serverSocket	ServerSocket
f	roomNum	int
f	roomList	HashMap<Integer, RoomServer>
f	roomMemberNum	ArrayList<Integer>
m	clientExitRoom(String)	void
m	getInstance()	MainServer
m	getRoomMember(Socket)	void
m	loginVerify(PrintWriter, String)	void
m	registerVerify(PrintWriter, String)	void
m	setRoomID(PrintWriter, String)	void
m	startThread()	void

图 3.1 MainServer

房间子服务端中存储着该房间内棋局的具体信息，该信息以棋局 Model 类的对象进行存储，包括棋盘中的棋子信息，用户的聊天信息，棋局的胜负状态，悔棋请求，认输请求。房间子服务端需要接受客户端的请求，解析客户端通过 Socket 发送过来的请求命令，并在服务端调用 model 的相应函数，对棋局状态信息进行相应的更新。包括处理下棋操作，聊天更新，悔棋请求，认输请求，胜负判断等操作。

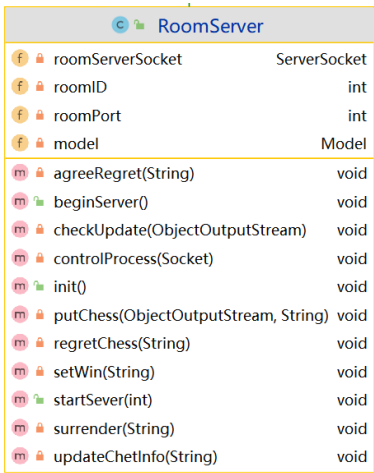


图 3.2 RoomServer

客户端采用 MVC 的设计框架，ClientPlayer 充当 Control，GameFrame 充当 View，棋局 model 充当 Model。ClientPlayer 主要负责接受用户在 GameFrame 界面上的相应操作，并提供对应的方法，在方法内通过 Socket 将客户端对模型的修改操作以及相应的参数，通过 Socket 将一条 acquire 字符串发送到房间服务端，在房间服务端对模型进行更新。ClientPlayer 每 500ms 会向房间服务端请求一次棋局信息的更新，房间服务端会通过 Socket 经过序列化返回整个棋局 model，客户端接收到 model 后，会将 model 中的棋子栈和聊天信息提取出来，调用 GameFrame 的更新函数，将游戏界面上的内容进行一次刷新。

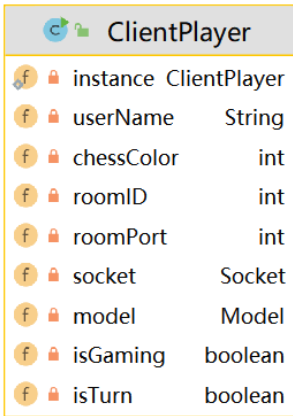


图 3.3 ClientPlayer

棋局模型实现了 Serializable 接口，其中主要包含棋局的基本要素和一些用以进行情况判断的 flag。chessBoard 用来记录棋盘信息，chessStack 是用来存放

棋子的棋子栈，`chetInfo` 是用来存放房间内的聊天信息，`winner` 是获胜方标记，`regretChessColor` 是悔棋方标记，`surrenderChessColor` 是认输方标记，`reshowIndex` 用来辅助进行复盘操作。

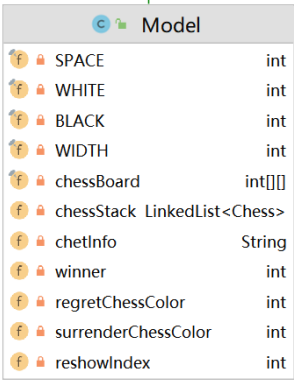


图 3.4 Model

游戏界面划分为两大部分，`ChessPanel` 和 `ChetPanel`。`ChessPanel` 负责展示当前的棋局信息，并监听客户端的下棋操作，同时能够追踪展示鼠标进行位置标记。`ChetPanel` 能够获取客户端输入的聊天信息并通过 `ClientPlayer` 将消息发送给房间服务端，能够展示客户端的聊天信息。并通过设置按钮，监听客户端的悔棋和认输请求，在事件相应之后，通过 `ClientPlayer` 将相应的请求发送给房间服务端进行处理。

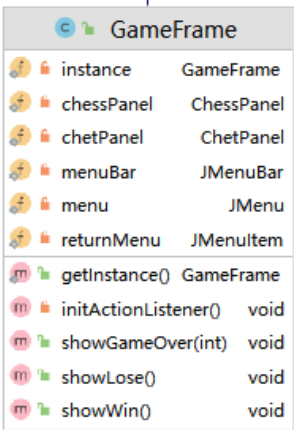


图 3.5 GameFrame

命令控制类中保存着客户端同服务端进行请求时，请求字符串的命令前缀，这些前缀在 `CommandOption` 类中被定义为字符串常量，用以标记客户端希望服

务端进行何种操作。

CommandOption		
GET_ROOM_MEMBER_NUM	String	
REGISTER_VERIFY	String	
LOGIN_VERIFY	String	
CHECK_UPDATE	String	
PUT_CHESS	String	
REGRET_CHESS	String	
AGREE_REGRET	String	
SEND_CHET_MESSAGE	String	
WIN	String	
SURRENDER	String	
EXIT	String	

图 3.6 CommandOption

第二节 整体流程

首先运行游戏的主服务端，包括同存储用户信息的数据库建立连接，建立 ServerSocket 监听用户端的注册和登录请求。

客户端运行游戏程序后，会首先和主服务端通过 Socket 建立连接，然后会在客户端显示出登陆界面。用户可以根据需求进行注册和登录。在进行用户提交注册信息后，客户端将会通过 Socket 将用户名和经过 hashcode 的用户密码发送到主服务端，主服务端会和数据库请求数据库内现存的用户信息，并逐条进行匹配比对，如果发现相同的用户名则会通过 Socket 向客户端返回一条错误信息，客户端接收到错误信息后将会弹出提示框；如果注册验证通过，主服务端会在数据库中增添一条新的用户信息，并通过 Socket 向客户端返回一条成功信息，客户端接收到成功信息后，将会弹出消息框提醒用户登陆进入游戏大厅。

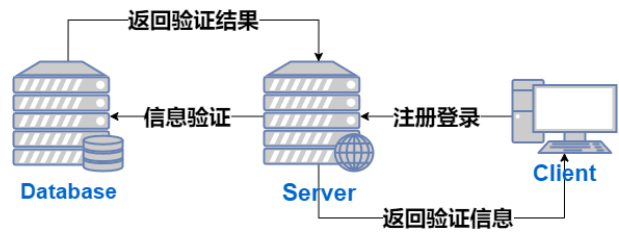


图 3.7 登录验证

在客户端成功进入游戏大厅后，客户端将启动一条更新请求线程，每 100ms 向主服务端发起一次更新请求，从主服务端拉取当前游戏大厅内的房间信息，

例如各个房间内现在已经拥有多少玩家。客户端获取到游戏大厅的房间信息后，会将这些信息发送给大厅界面，大厅界面根据当前房间内的人数，更新界面上的人数信息以及加入房间按钮上的提示信息（超过两人后只能进行观战）。在游戏大厅内，各个房间的进入按钮会监听用户的房间选择，在接受的用户进入房间的点击事件后，客户端将会通过 **Socket** 将玩家选择的房间号发送给主服务端。主服务端在接收到客户端选择的房间号之后，会根据用户所选择的房间号，启动相应房间内的子服务端进程，然后将子服务端的端口号通过 **Socket** 返回给客户端，客户端在接收到端口号后，将会断开和主服务器之间的网络连接，在之后进行网络通信都是和房间对应的子服务端进行通信请求的。同时，客户端将会更新自己的游戏状态，并初始化游戏界面。默认先进入房间的玩家为白子后手，需要等待黑子先手玩家进入房间后才能开始游戏。

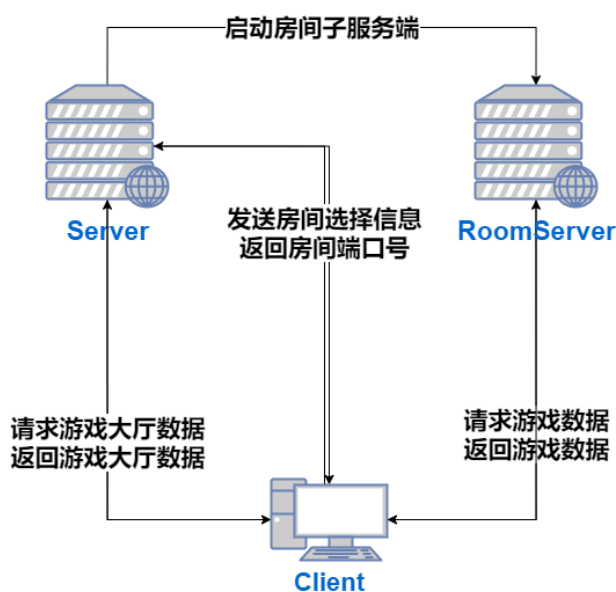


图 3.8 请求权限转移

客户端玩家进入到游戏界面后，整个游戏界面将首先进行初始化，客户端每 500ms 向房间服务端请求一次棋局信息 **model** 的更新，根据从房间服务端请求来的棋局信息，调用 **GameFrame** 中的 **update** 方法，更新当前游戏界面中的游戏信息。

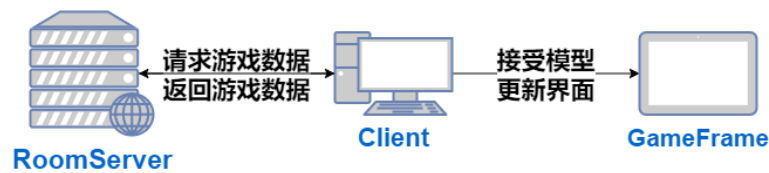


图 3.9 游戏界面更新

GameFrame 游戏界面负责监听玩家操作并和对玩家的相应操作进行及时的反馈交互。通过监听玩家的操作，调用 ClientPlayer 中对应的方法，在该方法中，构建请求字符串 acquire，将对应的操作的标记作为起始前缀，将操作所需要用到的参数依次跟在其后。构建好请求字符串之后，客户端与房间服务器建立连接，通过 Socket 发送请求，然后等待房间服务器返回结果。房间服务器接收到客户端的请求后，启动一条请求处理线程，解析客户端发送过来的请求字符串 acquire，并根据其所调用的方法和参数，在 model 中调用相关的方法，更新 model 中的相关信息，如果需要然后返回相应的结果。

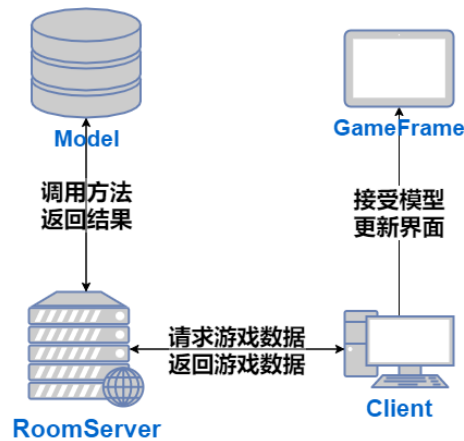


图 3.10 远程过程调用

第三节 服务端

为了能够满足多个玩家同时在线的需求，考虑到多个玩家可能会各服务端同时进行通信，为了能够节约网络通信所占据的资源，考虑采用”需要时连接“的方式进行 Socket 网络通信，即客户端与服务端之间并不是保持长链接，而是在客户端需要向服务端发起请求，发送更新，拉取数据的时候才会和服务端建立连接并发送请求，服务端接收到请求后，便会启动一条子线程去处理该条请

求，随后客户端便断开与服务端的连接，释放资源。

由于在项目中设计了游戏大厅的界面，不同客户端的玩家会在进入游戏后首先进入到游戏厅中，然后根据玩家的选择，进入到相应的房间进行游戏。考虑到可能会有很多客户端同时在线，同时对主服务端发送请求，拉取更新数据信息。如果后期玩家进入游戏后仍是和主服务端进行通信，则会导致主服务端的压力过大，处理的请求数量过多，也会导致在编写代码的时候设计复杂。因此考虑采用“主服务端和子服务端结合”的模式，即玩家在注册登录和在游戏大厅中时，是和主服务端进行通信，而在玩家选择好相应的房间，准备游戏后，主服务端会根据玩家选择的房间信息，启动对应房间的子服务端，并将该房间对应的端口号返回个客户端。在房间子服务端内，保存有当前房间内的棋局模型，有能够实现对棋局模型进行更新读取的方法。客户端在接收到房间的端口号后，在房间内的游戏操作所需要进行的网络通信，都是和房间子服务端进行的，在同一房间内的全部客户端都是对该房间内的同一个棋局模型进行操作的，保证了数据在各个客户端之间的一致性。

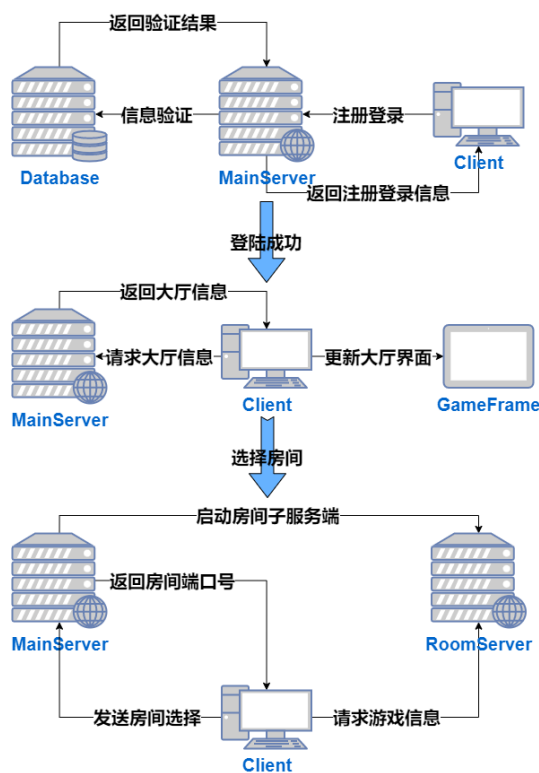


图 3.11 服务端整体流程

3.3.1 主服务端

主服务端连接了一个存储客户端玩家信息的数据库，维护了一个存储房间人数信息数组和一个存储房间子服务端的数组。主要负责玩家的注册登录验证，大厅信息维护，客户端房间分配等工作。

注册登录验证。客户端将会根据玩家的选择，向主服务端发送一条请求 `acquire`，请求的开头是一个命令标记。如果以“REGISTER”开头，则是进行注册验证。主服务端将会在存储用户信息的数据库中查询该玩家注册使用的用户名是否存在，如果存在的话会返回一条错误信息，如果不存在的话会返回成功信息。如果以“LOGIN”开头，则是进行登录验证。主服务端会在存储用户信息的数据库中查询该玩家的用户名和密码，此处玩家的密码经过了 `hashcode` 编码，并没有采用明文的方式存储在数据库中，增强了信息的私密性和安全性。如果匹配到的玩家用户名和密码均一致，则登录认证通过，返回成功信息，否则返回错误信息。

大厅信息维护。主服务端维护了一个存储大厅内各个房间中玩家人数的数组，客户端玩家需要根据该数组的内容，绘制大厅界面中的房间人数 `Label`，以及设置进入房间按钮上的提示信息（“加入游戏” / “加入观战”）。为了保证数据的实时同步性，客户端将会每 `100ms` 向主服务端请求更新一次房间人数信息，发送的请求信息以“GET_ROOM_MEMBER_NUM”开头，在收到主服务端返回的房间信息后，将会刷新界面上的内容。

客户端房间分配。在客户端选择好相应的房间进入游戏后，将会向主服务端发送一条请求信息，消息中包含客户端选择的房间号。主服务端在接收到这条请求消息后，将会解析该消息字符串，从中提取出来客户端选择的房间编号，然后根据房间编号，启动对应房间的子服务端，并将该房间子服务端的端口号返回给客户端，客户端会根据该端口号更新内部属性，之后的便会和主服务端断开连接，主服务端对该客户端的服务到此结束。

3.3.2 子服务端

子服务端，即房间服务端，在客户端选择房间后，将为客户端玩家提供游戏服务。其中维护了一个 `SocketServer` 用来接受客户端连接请求，一个 `model` 用

来存储棋局模型。在子服务端中，实现了 `ControlProcess` 的处理方法，负责处理客户端发送的请求，对 `model` 进行相关的操作和更新。

客户端会通过 `Socket` 向房间服务端发送请求字符串，该字符串以“`CommandOption`”中的字符串常量开头，后跟相应操作所需要的参数。房间服务端在接收到请求后，会启动一条线程去处理该条请求，解析字符串获取相应的操作以及所需要的参数，然后再调用相应的模型 `model` 中的方法进行处理。

房间服务端在接收到客户端发送的“`CHECK_UPDATE`”的请求后，会将房间内的棋局信息 `model` 进行序列化后，通过 `Socket` 传输给客户端，客户端接收到该 `model` 后便可以进行界面更新。

房间服务端在接收到客户端发送的“`PUT_CHESS : chessColor : x : y`”的请求后，将会解析获取相关参数，调用 `model` 中的 `putChess` 操作，将棋子栈和棋盘信息进行更新。

房间服务端在接收到客户端发送的“`REGRET_CHESS : chessColor`”的请求后，将会根据后面紧跟的棋子颜色信息，调用 `model` 中的 `setRegretChessColor` 方法，设置申请悔棋方的棋子颜色。

房间服务端在接收到客户端发送的“`AGREE_REGRET : 0/1`”的请求后，会解析出用户名是否同意悔棋，如果同意悔棋则调用 `model` 中的 `regretChess` 方法，并将 `model` 中关于悔棋请求的 `flag` 清空。

房间服务端在接收到客户端发送的“`SEND_CHET_MESSAGE : userName : chetInfo`”的请求后，会解析出用户名和该用户发送的消息，然后调用 `model` 中的 `updateChetInfo` 方法，更新当前的聊天信息。

房间服务端在接收到客户端发送的“`WIN : chessColor`”的请求后，会解析出获胜方的棋子颜色，然后调用 `model` 中的 `setWinner` 方法，更新当前的胜负情况。

房间服务端在接收到客户端发送的“`SURRENDER : chessColor`”的请求后，会解析出认输方的棋子颜色，然后调用 `model` 中的 `setSurrenderChessColor` 方法，对认输状态的 `flag` 进行标记。

第四节 客户端

客户端在 MVC 设计模式种充当 V 和 C 的角色，即 ClientPlayer 作为控制，GameFrame 作为界面。

游戏界面主要负责绘制棋盘和棋子，展示房间内的聊天信息，并且提供悔棋和认输的按钮。游戏界面在棋盘范围内会监听鼠标的移动事件，绘制出当前光标的焦点；监听鼠标点击事件，通过控制端向房间服务端发送下棋请求。在聊天界面内，监听用户的文字输入和发送按钮，同时还会监听悔棋和认输两个功能按钮的点击事件，在接收到用户事件后，将会通过控制端向房间服务端发送相应的请求。

控制端提供各种连接游戏界面和房间服务端的方法，通过接受游戏界面事件监听的调用相应，在方法内与房间服务端建立 Socket 连接，将相应操作的 CommandOption 前缀和所需参数拼接成字符串，发送给房间服务端，用以更新房间内棋局信息的 model。控制端还需要负责定时向房间服务端请求数据更新，每 500ms 向房间服务端发送一次请求，拉取新的数据信息，并调用游戏界面的 update 方法进行界面刷新。

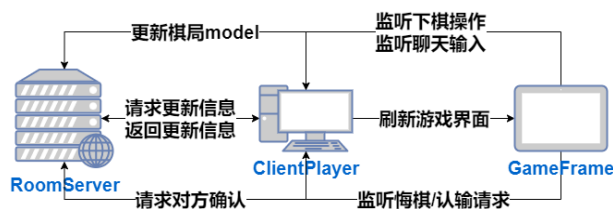


图 3.12 客户端整体流程

3.4.1 控制端

控制端内维护了玩家的用户名，房间对应的端口号，用于通信的套接字 socket，玩家当前的状态 isGaming 标识是否正在游戏，isTurn 标识是否轮到本方下棋，chessColor 标识玩家的棋子颜色（观战方一律为 Chess.SPACE）。此外，在控制端内，还有一个棋局信息 model 的本地副本，之所以会保存一个本地副本，是考虑到更新请求每 500ms 进行一次，在线人数较少时，相对更新比较及时，但如果在线人数过多，由于在服务端是多个线程同时对房间内的棋局信息 model 进行操作，所以会存在阻塞现象，由于排队等待会导致本方的信息更新不

够及时，用户点击下棋后不能够及时展示下棋结果。为此我们在本地暂存了一个 model 副本，只是为了能够方便本地及时更新下棋和聊天等必要信息，和做一些简单的判断，使得在客户端玩家有比较好的及时交互，但真正对于房间棋局 mode 的更新操作都是通过控制端向服务端发送相应的请求完成的。

控制端会每隔 500ms 主动和房间服务端建立一次连接，发送请求字符串"CHECK_UPDATE"，然后从 socket 中读取房间服务端发挥的 model，并进行反序列化获得 model 的本地副本。然后调用界面的更新函数，更新棋盘信息和聊天信息。此外，控制端还会检查 model 中的状态标志。如果 winner 不为 Chess.SPACE，则证明出现胜负，则根据 winner 中的棋子颜色，确定展示胜利信息还是失败信息，如果时观战者则提示结束信息；如果 regretChessColor 不为空，则判断是否时对方发起了悔棋请求，如果发起，则在本地弹出消息框和玩家进行交互，请求是否同意对方悔棋，根据玩家的选择，调用方法向房间服务端发送请求字符串"AGREE_REGRET : true/false"，更新 model 中的相关信息。如果 surrenderChessColor 不为空，则判断是否是对方发起了认输请求，如果是则在本地弹出消息框和玩家进行交互，请求是否同意对方认输，如果同意，则控制端直接向房间服务端发送获胜信息。

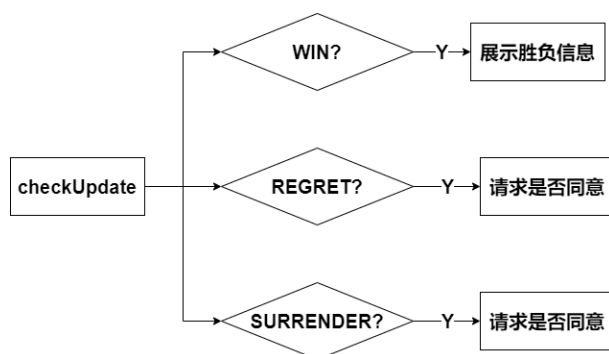


图 3.13 更新检查逻辑

控制端通过游戏界面监听玩家的下棋操作，从界面获取玩家下棋的 XY 坐标，结合自身的棋子颜色，在本地缓存的 model 副本中进行临时更新，首先判断下棋的位置是否合理，如果合理，将会把相应位置设置为本方颜色，及时刷新界面，使玩家得到反馈。然后与房间服务端建立连接，将棋子颜色，X，Y 坐标拼接成一条请求字符串"PUT_CHESS : chessColor : x : y"，发送给房间服务端

进行 model 的更新。

控制端通过聊天界面监听玩家的聊天信息输入，当玩家敲回车或点击发送按钮时，游戏界面会将玩家输入的聊天信息传递给控制端，控制端会首先利用本地缓存的 model 副本，更新当前界面的聊天信息并及时进行反馈。然后与房间服务端建立连接，将用户名和聊天信息拼接成一条请求字符串“SEND_CHET_MESSAGE : userName : chetInfo”，发送给房间服务端进行 model 的更新。

控制端在进行下棋操作后，会检查当前的落子是否导致出现胜负，会根据最后一个落子的位置，沿着八个方向进行计数，并将相对方向的数量加和，如果大于等于 5 则本方获胜。控制端会和房间服务端建立连接，发送请求字符串“WIN : chessColor”，房间服务端进行 model 的更新。

控制端通过聊天界面监听玩家的悔棋请求。当轮到玩家下棋时，玩家可以请求进行悔棋。此时界面会通过按钮的点击事件，调用控制端的悔棋方法。控制端在接收到调用后，会和房间服务端建立连接，将自己作为悔棋方的棋子颜色拼接成请求字符串“REGRET_CHESS : chessColor”，发送给房间服务端进行 model 更新。

控制端通过聊天界面监听玩家的认输请求。通过界面监听认输按钮的点击事件，一旦发生点击事件，将会调用控制端的认输方法，控制端在接收到调用后，会和房间服务端建立连接，将自己作为认输方的棋子颜色拼接成请求字符串“SURRENDER : chessColor”，发送给房间服务端进行 model 更新。

3.4.2 游戏界面

游戏界面设计分为 ChessPanel 和 ChetPanel 两部分。ChessPanel 负责监听用户在棋盘上的操作，并绘制棋盘信息和用户进行交互。ChetPanel 负责监听用户的聊天信息输入，和悔棋认输等功能请求，并通过控制端向房间服务端发起请求。

ChessPanel 内维护了一个棋子栈和聊天信息，还有一个光标类 CursorTriger。控制端每次和服务端请求更新数据信息后，都会调用 ChessPanel 的 update 方法，更新棋子栈和聊天信息，然后调用 repaint 函数，重绘界面。在重绘界面时，会

绘制棋盘背景，当前棋子栈内的全部棋子，以及鼠标光标的标记位置。更新聊天信息会将收到的聊天信息显示在聊天框内。

ChessPanel 中会监听客户端玩家的鼠标事件，包括移动和点击。监听鼠标移动事件，获取光标当前的位置，利用基准坐标 (BASEX,BASEY) 和 BLOCKWIDTH 进行标准化，转化到在棋盘中的位置，然后更新 CursorTriger 中光标的位置信息。监听鼠标点击事件，获取鼠标点击的位置，同样是利用基准坐标 (BASEX,BASEY) 和 BLOCKWIDTH 进行标准化，转化到棋子在棋盘中的位置，并向控制端发送相关的棋子信息。

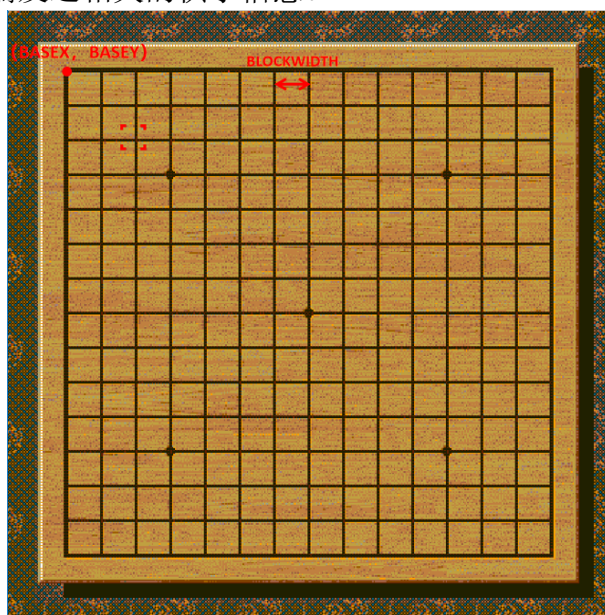


图 3.14 棋盘布局坐标转化

ChetPanel 中不仅仅实现了聊天功能，还将一些功能请求键（悔棋/认输）放置在了 ChetPanel 中。在 ChetPanel 中，主要监听客户端玩家的文字输入和按钮点击。当客户端玩家在聊天输入框内输入完成后，可以通过敲回车或点击发送按钮发送信息，监听到事件后，ChetPanel 会解析当前的聊天内容，并将相关信息发送给控制端进行处理；当玩家点击悔棋按钮时，界面会调用控制端有关悔棋的方法，通过控制端向房间服务端发送相关请求，更新棋局 model 中的悔棋状态标记，当对方通过更新请求读取到悔棋状态标记发生变化后，会进行是否同意悔棋的提示；当玩家点击了认输按钮时，界面会调用控制端有关认输的方法，通过控制端向房间服务端发送相关请求，更新棋局 model 中的认输状态标

记，当对方通过更新请求读取到认输状态标记发生变化后，会进行是否同意认输的提示。

第五节 网络通信

由于在项目设计之初，考虑设计游戏大厅，实现多人同时在线，允许两人对局，多人观战。因此，如果采用简单的端对端 (P2P) 模式的话，在多人同时在线时，消息的分发传输就会变得比较复杂，可能需要在每一个玩家客户端都维护一个房间内的玩家列表，在发送相关消息时向全列表广播消息，如此一来实现难度大，各客户端之间还可能出现较大的延迟，数据信息不同步等现象。

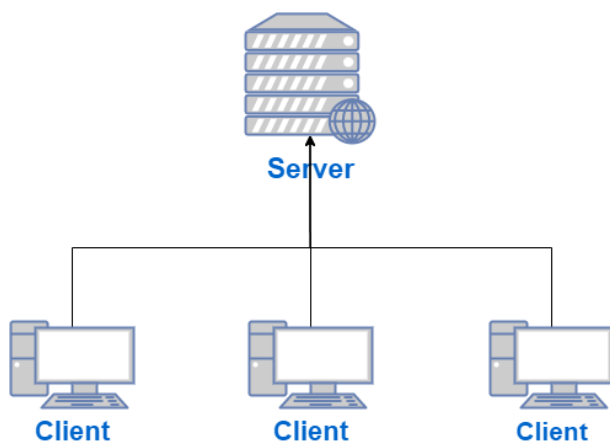


图 3.15 CS 架构

为此，我们采用了 CS 架构，即客户端-服务端模式。在玩家刚刚登陆游戏时，整个大厅由一个主服务端维护，所有进入大厅的玩家都会和主服务端去请求房间人数信息，并以此更新客户端的本地消息。在玩家选择好房间后，会向主服务端发送相关的请求，主服务端在接收到相关请求后，会将该房间的端口号返回给客户端，随后，客户端的请求处理权就转移给了相应的房间。客户端玩家进入房间后，所有在房间内的客户端玩家都会调用 `CheckUpdate` 方法，向房间服务端请求当前棋局 `model`，也就是说，房间内的所有玩家其实是共用的同一个 `model`，而其本地 `model` 只是这个共用 `model` 的缓存副本，为了能够在本地及时和玩家进行交互。所有的客户端共用 `model`，便可以保证了棋局信息在各玩家之间的一致同步。

基于 CS 架构，我们进一步摒弃了传统的本地处理方法，而是采用 RPC (Remote Procedure Call)，即远程过程调用。客户端在接收到界面的相关事件相应后，只会在本地缓存副本 model 中进行临时的更新实现及时交互，但对于 model 的真正操作，都是通过控制端向房间服务端发送请求字符串 acquire，该字符串结构为：以 CommandOption 中的方法标记开头，通过 “:” 分割方法标记和各个参数。而房间服务端在接收到客户端请求后，会启动一条线程，解析处理该客户端请求，根据需求在服务端调用 model 中的相关方法，更新其中的参数信息，需要时返回更新结果给客户端。

通过 RPC 架构，客户端不需要再在本地对 model 进行更新，然后将更新后的 model 发送给服务端，既减少了客户端在本地操作的复杂度，又降低了网络传输过程的成本，还比较简单地实现了 model 在各个服务端之间地同步性。

第六节 数据库存储

当前阶段，数据库主要提供客户端注册登录的信息验证功能。在客户端玩家进入登陆界面时，可以根据自身需要，注册账号或登陆游戏。当客户端输入用户名和密码，点击注册后，客户端会获取玩家的用户名和密码，并将密码进行 hashcode，避免网络通信过程中进行明文传输，然后向主服务端发送注册请求，请求字符串 acquire 为”REGISTER : userName : password”。主服务端在接收到客户端的注册请求后，会启动处理线程，调用数据库相关方法，读取数据库中现存的玩家信息，如果匹配到注册所用的用户名已经存在在数据库中，则会发挥一个错误信息给客户端，如果用户名不存在于数据库中，则该玩家注册成功，并将新的玩家信息写入到数据库中，并进行上传更新。当客户端用户名和密码，点击登录后，客户端同样会发送一个登录请求给主服务端，请求字符串为”LOGIN : userName : password”。主服务端在接收到客户端的登录请求后，会调用数据库的方法，去数据库中匹配玩家信息，只有当用户名和密码全部匹配一致才能够通过登录认证，否则会返回给客户端一条错误信息。

第四章 项目效果

客户端打开游戏后，首先会进入登陆界面，如果还没有账号，可以在界面内输入用户名和密码，然后点击注册，数据库会根据请求增加玩家信息。如果已经拥有账号，则可以输入用户名和密码后点击登录。



图 4.1 注册验证

	username	password
1	aaa	48657
2	abc	1234aaa
3	bbb	49650
4	hhh	lksj
5	java	1509442

图 4.2 数据库信息更新

玩家注册成功或登陆成功后，会进入到游戏大厅，大厅内初始有 12 各房间供玩家选择，客户端玩家可以根据自身需要，选择适当的房间加入游戏或加入观战，如果房间内的人数小于 2 两人则可以加入游戏，如果大于等于 2 人则只能加入观战。

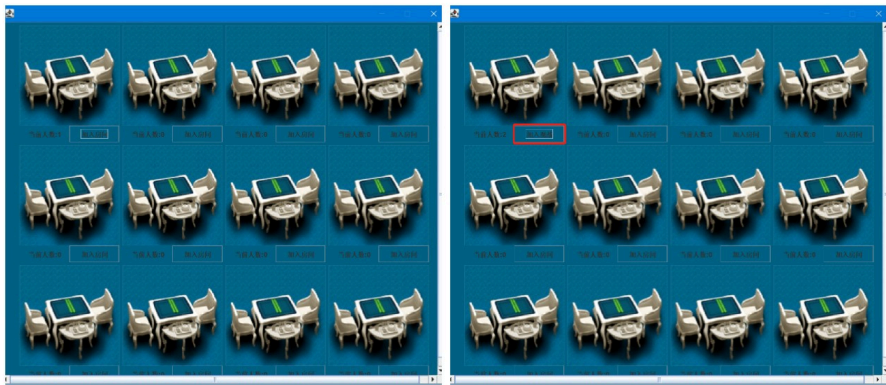


图 4.3 游戏大厅界面

玩家选择好房间后，点击加入按钮，进入房间，显示游戏界面，在棋盘界面中，界面会追踪光标的位置并进行标记。界面会监听玩家的点击事件，进行下棋操作。

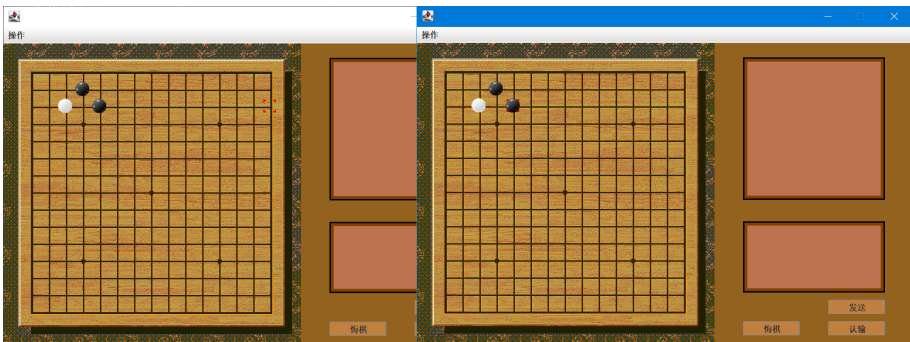


图 4.4 玩家下棋操作

玩家可以通过点击右侧的悔棋按钮向对方发起悔棋请求，如果得到对方的同意，则悔棋成功，否则继续游戏。

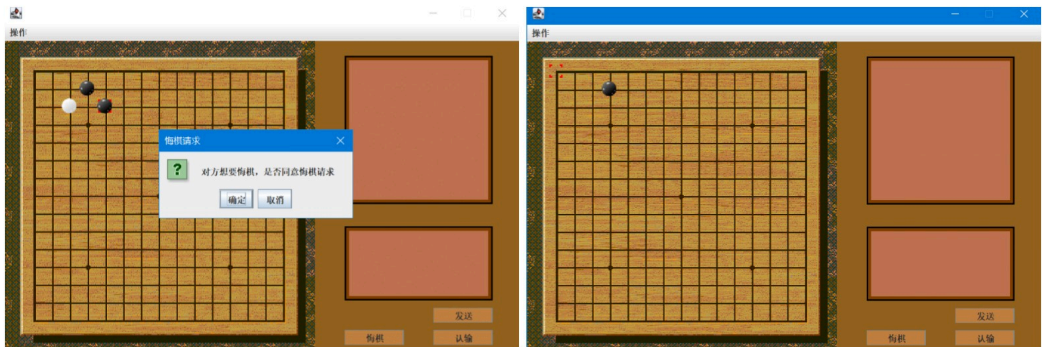


图 4.5 玩家悔棋请求

玩家可以通过点击右侧的认输按钮向对方发起认输请求，若得到对方的同意，判定对方胜利，否则继续游戏。



图 4.6 玩家认输请求

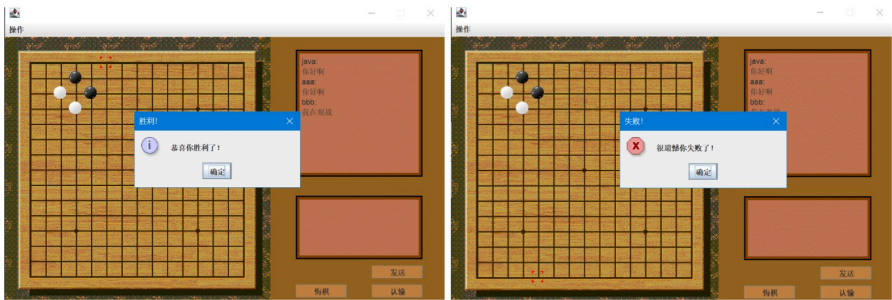


图 4.7 展示胜负信息

玩家可以在右侧的聊天输入框内输入聊天信息，点击发送按钮即可向对方发送聊天信息。



图 4.8 聊天通信功能

加入房间的观战方能够观看到棋局的全部过程，并且也能够通过聊天和房

间内的其他玩家进行交互。



图 4.9 观战方视角

游戏胜负已定后，之前的悔棋按钮会更改提示信息，变成复盘按钮，在第一次点击后，便会开始复盘操作，提示信息变为下一步，每点击一次会复盘一步。



图 4.10 复盘操作

游戏结束后，如果玩家想要返回游戏大厅，可以通过上方的菜单栏返回大厅。



图 4.11 返回游戏大厅

第五章 总结和展望

经过一个学期的 Java 学习，对 Java 这门语言有了初步的了解，感受到了这门面向对象的编程语言在编写项目时的方便高效。这次 Java 课程设计，完成了基于 TCP 通信的网络五子棋项目，实现了五子棋的基本功能，包括下棋，判定胜负，悔棋，认输以及复盘等操作，除此之外，还根据课程所学，增加了一些新的功能。根据数据库有关的知识，搭建了与 MySQL 数据库连接的主服务端，用于进行玩家注册和登录验证；根据自己了解的 RPC 框架，借鉴其思想，搭建了主服务端子服务端结合的服务体系，共同处理多个客户端同时在线的请求；此外，还对游戏内容进行了一定的扩展，设计了游戏大厅，能够实现更多的玩家同时在线游戏；增加了观战模式，玩家除了能够体验到游戏的乐趣，还能够通过观战，学习高手下棋的经验；为了丰富游戏的体验，还设置了游戏的背景音乐以及下棋时的落子音效，使游戏的感官体验更加丰富。

这次项目的实践过程中，也遇到了很多的困难。如，多个客户端玩家同时在线时会发生较严重的延迟现象，为此我将棋局 model 在本地缓存一个副本，当客户端点击下棋或发送聊天信息时，会首先更改本地的 model 副本，然后去刷新游戏界面，让用户获得实时的反馈，随后再将相关的参数信息发送给房间服务端，去更新线上共用的 model。又如，在解决非 P2P 模式下的玩家交互，如何实现悔棋和认输等两个玩家之间的通讯问题。我才用了在 model 中增加标志信息，客户端在发起请求时，会向房间服务端发送相关的参数信息，房间服务端以此更改 model 中的标志信息，在对方请求更新拿到 model 后，会检查这些标志信息是否经过标记，如果经过标记，证明对方玩家有相应的请求，于是本客户端要和玩家进行交互，是否同意对方请求，并将相关的标记信息进行更新。请求方根据标记信息的变化情况来判断对方是否同意了自己的请求，然后将标记信息复原。

通过这次课程作业，我更好的体会到了项目开发和程序编写之间的区别。以往我们只是编写一些小的方法，并不需要考虑各个系统之间的协调一致，而在编写项目的时候，不仅仅要注重代码的逻辑和质量，更要在最开始设计好项

目的框架，确定好各个系统之间的关系，这在后期的开发过程中会给我们提供一个比较明确的思路，能够减少开发过程中遇到的问题和 bug。

当前的游戏只实现了一些基本功能，界面的设计相对比较简陋，随着后期对于图形化开发的进一步了解，希望能够掌握更多组件的属性，能够设计出更美观，交互性更好的游戏界面。当前的游戏大厅只能够支持初始化的 12 个房间，但已经在开发的过程中留出了相应的接口，在后期可以增加创建房间的功能，增加更多的房间。在游戏中，目前只支持文字聊天信息的发送，后期可以增加语音聊天功能。

参考文献

- [1] 龚道军. 中国游戏产业发展问题浅析. 高科技与产业化, 2021, 27(07): 62 ~ 67.

致 谢

首先我要感谢党和国家的培养，能够给我提供良好的学习环境和丰富的学习资源。在国家重视和大力发展计算机领域的趋势下，我能够在南开大学获得更多的专业知识，能够尽早体验一个小项目的开发过程，这都不仅是对我的视野扩展，更是对我能力的提升。

我要感谢我的父母，在我完成项目过程中给予我的支持和帮助，无论我遇到什么困难，他们总是能够更成为我坚强的后盾。我遇到困难，项目推进不下去的时候，我的父母总会给我鼓励和安慰，让我能够静下心来思考，寻找解决问题的途径和方法。

我还要感谢刘嘉欣老师一个学期以来的悉心指导和无私付出。刘老师总是能够结合自身的经历，言传身教。老师的课程风趣幽默，深入浅出，不仅仅向我们讲述了 **Java** 语言的语法特性，更从底层为我们剖析了原理，老师传授的不仅仅是知识，更是一种编程的思想。这在本次课程设计的项目构件中，起到了至关重要的作用。每次和老师探讨问题，和老师寻求帮助，老师都会耐心负责地为我们解决问题，给我们提供各种地学习资源。可以说刘老师把握带到了学习计算机，学习编程的新天地。

我还要感谢所有在课程设计中给予我支持和帮助的各位同学。和邹先予的交流给我项目架构的启发和灵感，为我前期设计框架提供和很好的思路，也是项目能够顺利推进的重要原因。在后期的开发过程中，我还会经常和他探讨一些遇到的问题，和自己发现的一些不错的解决方法，邹哥也会非常无私的给予我各种帮助。

再次感谢在课程设计中为我提供帮助的各位老师同学。