**成都信息工程学院计算机学院**

**课**

**程**

**实**

**验**

**报**

**告**

|  |  |
| --- | --- |
| **实验课程：** | **网络程序编程技术** |
| **实验项目：** | **房间系统** |
| **指导教师：** | **张睿** |
| **学生姓名:** | **江雨涛** |
| **学生学号：** | **2022021128** |
| **班 级：** | **数媒221** |
| **实验地点：** | **B215** |
| **实验时间：** | **2024 年 11 月 27 日 16 点~ 17:40点** |
| **实验成绩：** |  |

## 一【上机实验内容】

1. 实验二的登录成功后跳转到房间场景；

2. 接收服务器发来的房间列表；

3. 根据房间列表显示各个房间；

4. 以某种方式选中一个房间进入；

5. 接收到服务器发来的进入游戏协议后，跳转到游戏场景。

## 二【上机实验步骤】（重点）

### 操作步骤

（与代码无关的操作步骤）

1. 创建房间场景，客户端登录后尽心跳转
2. 客户端通过接收roomList字符串来接收房间并显示
3. 客户端通过向服务端发送字符串来创建房间
4. 客户端通过接收服务端发送的消息来进行游戏开始
5. 编写服务端相关代码

### 关键代码

服务端代码：

package com.cy.socket;  
  
import com.cy.mapper.UserMapper;  
import com.cy.pojo.User;  
import com.cy.util.MD5Util;  
import com.cy.util.MybatisUtil;  
import org.apache.ibatis.session.SqlSession;  
  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.io.PrintWriter;  
import java.net.ServerSocket;  
import java.net.Socket;  
import java.util.HashSet;  
import java.util.Scanner;  
import java.util.Set;  
import java.util.concurrent.Executors;  
import java.util.concurrent.ScheduledExecutorService;  
import java.util.concurrent.TimeUnit;  
  
public class ChatServer {  
 private static final int *PORT* = 10002;  
 private static final Set<ClientHandler> *clientHandlers* = new HashSet<>();  
 private static SqlSession *sqlSession* = MybatisUtil.*sqlSessionFactory*();  
 private static UserMapper *userMapper* = *sqlSession*.getMapper(UserMapper.class);  
  
 public static void main(String[] args) {  
 *startServer*();  
 *listenForMessages*();  
 }  
  
 public static void startServer() {  
 System.*out*.println("服务器启动，监听端口：" + *PORT*);  
 new Thread(() -> {  
 try (ServerSocket serverSocket = new ServerSocket(*PORT*)) {  
 while (true) {  
 Socket clientSocket = serverSocket.accept();  
 ClientHandler handler = new ClientHandler(clientSocket);  
 synchronized (*clientHandlers*) {  
 *clientHandlers*.add(handler);  
 System.*out*.println("客户端已添加，当前数量为" + *clientHandlers*.size());  
 }  
 handler.start();  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }).start();  
 }  
  
 public static void listenForMessages() {  
 new Thread(() -> {  
 try (Scanner scanner = new Scanner(System.*in*)) {  
 while (true) {  
 String message = scanner.nextLine();  
 *sendMessageToAllClients*(message);  
 }  
 }  
 }).start();  
 }  
  
 public static void sendMessageToAllClients(String message) {  
 synchronized (*clientHandlers*) {  
 System.*out*.println("当前连接的客户端数量: " + *clientHandlers*.size());  
 for (ClientHandler handler : *clientHandlers*) {  
 handler.sendMessage(message);  
 // 打印客户端信息  
 System.*out*.println("消息发送到客户端: " + handler.getId());  
 }  
 }  
 }  
  
  
 private static class ClientHandler extends Thread {  
 private Socket socket;  
 private PrintWriter out;  
 private BufferedReader in;  
 private ScheduledExecutorService scheduler;  
  
 private StringBuilder messageToSend = new StringBuilder("roomList 0"); // 定义要发送的消息内容  
  
 public ClientHandler(Socket socket) {  
 this.socket = socket;  
 System.*out*.println("新客户端连接: " + socket.getInetAddress().getHostAddress() + ":" + socket.getPort());  
 try {  
 in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
 out = new PrintWriter(socket.getOutputStream(), true);  
  
 // 初始化定时器  
 scheduler = Executors.*newScheduledThreadPool*(1);  
 for (ClientHandler handler : *clientHandlers*) {  
 if (!handler.messageToSend.equals("roomList 0")) {  
 this.messageToSend = handler.messageToSend;  
 }  
 }  
 scheduler.scheduleAtFixedRate(() -> sendMessage(messageToSend.toString()), 0, 2, TimeUnit.*SECONDS*);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void run() {  
 try {  
 String message;  
  
 while ((message = in.readLine()) != null) {  
 System.*out*.println("收到消息: " + message);  
 String s1 = message.split(" ")[0];  
 //注册操作  
 if ("Regist".equals(s1.replace("\uFEFF", ""))) {  
 String password = MD5Util.*getMD5*(message.split(" ")[2].replace("\uFEFF", ""));  
 String username = message.split(" ")[1].replace("\uFEFF", "");  
 String ip = socket.getInetAddress().getHostAddress() + ":" + socket.getPort();  
 User user = *userMapper*.select(username);  
 //用户名存在  
 if (user != null) {  
 sendMessage("RegistFail 1 ");  
 System.*out*.println("用户名存在");  
 } else {  
 User user1 = new User();  
 user1.setUsername(username);  
 user1.setPassword(password);  
 user1.setOnline(0);  
 user1.setScore(0);  
 user1.setUserIp(ip);  
 *userMapper*.regist(user1);  
 *sqlSession*.commit();  
 sendMessage("RegistSuccess ");  
 System.*out*.println("注册成功");  
 }  
 } else if (message.split(" ")[0].replace("\uFEFF", "").equals("Login")) {//登录操作  
 String password = MD5Util.*getMD5*(message.split(" ")[2].replace("\uFEFF", ""));  
 String username = message.split(" ")[1].replace("\uFEFF", "");  
 User user = *userMapper*.select(username);  
 if (user == null) {  
 sendMessage("LoginFail 1");  
 System.*out*.println("用户不存在");  
 } else if (!user.getPassword().equals(password)) {  
 sendMessage("LoginFail 2");  
 System.*out*.println("密码错误");  
 } else if (user.getOnline() == 1) {  
 sendMessage("LoginFail 3");  
 System.*out*.println("用户已登录");  
 } else {  
 String ip = socket.getInetAddress().getHostAddress() + ":" + socket.getPort();  
 *userMapper*.online(username, ip);  
 *sqlSession*.commit();  
 sendMessage("LoginSuccess");  
 System.*out*.println("登录成功");  
 }  
 } else if (message.split(" ")[0].replace("\uFEFF", "").equals("createRoom")) {  
 String[] s = messageToSend.toString().split(" ");  
 StringBuilder ms = new StringBuilder("roomList " + (Integer.*parseInt*(s[1]) + 1) + " ");  
 String username = *userMapper*.findByIp(socket.getInetAddress().getHostAddress() + ":" + socket.getPort()).getUsername();  
 for (int i = 2; i < s.length; i++) {  
 ms.append(s[i]).append(" ");  
 }  
 ms.append(username).append(" ");  
 messageToSend = ms;  
 broadcast(ms.toString());  
 updateRooms(ms.toString());  
 } else if (message.split(" ")[0].replace("\uFEFF", "").equals("enterRoom")) {  
 String username = message.split(" ")[1];  
 User select = *userMapper*.select(username);  
 for (ClientHandler handler : *clientHandlers*) {  
 if ((handler.socket.getInetAddress().getHostAddress() + ":" + handler.socket.getPort()).equals(select.getUserIp())) {  
 handler.sendMessage("begin 1");  
 }  
 }  
 sendMessage("begin 0");  
 }  
 //这会把消息重新发送给所有客户端  
// broadcast(message);  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 } finally {  
 try {  
 // 关闭定时器  
 if (scheduler != null) {  
 scheduler.shutdown();  
 }  
 synchronized (*clientHandlers*) {  
 *clientHandlers*.remove(this);  
 String ip = socket.getInetAddress().getHostAddress() + ":" + socket.getPort();  
 User user = *userMapper*.findByIp(ip);  
 if (user != null) {  
 *userMapper*.updateByIp(ip);  
 *sqlSession*.commit();  
 }  
 System.*out*.println("客户端数量为" + *clientHandlers*.size());  
 }  
 socket.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 }  
 }  
  
 public void sendMessage(String message) {  
 out.println(message);  
 }  
  
 private void broadcast(String message) {  
 synchronized (*clientHandlers*) {  
 for (ClientHandler handler : *clientHandlers*) {  
 handler.sendMessage(message);  
 }  
 }  
 }  
  
 private void updateRooms(String message) {  
 synchronized (*clientHandlers*) {  
 for (ClientHandler handler : *clientHandlers*) {  
 if (!handler.messageToSend.equals(this.messageToSend)) {  
 handler.messageToSend = new StringBuilder(message);  
 }  
 }  
 }  
 }  
 }  
}

客户端代码：

using System;

using System.Collections;

using System.Collections.Generic;

using System.IO;

using System.Net.Sockets;

using System.Text;

using TMPro;

using UnityEngine;

using UnityEngine.SceneManagement;

using UnityEngine.UI;

public class Room : MonoBehaviour

{

GameObject[] roomTexts;

GameObject[] roomBtns;

public Socket clientSocket;

NetworkStream networkStream;

StreamWriter writer;

StreamReader reader;

string recvStr;

bool isConnected = false;

public string myUsername;

//测试字段

string recvStr1 = "roomList 3 127.0.0.1:47100 127.0.0.1:12000 ddddddd ";

// Start is called before the first frame update

void Start()

{

getSocket();

Debug.Log(myUsername);

}

void refreshRooms(string roomListStr)

{

if (roomTexts != null)

for (int i = 0; i < roomTexts.Length; i++) Destroy(roomTexts[i]);

if (roomBtns != null)

for (int i = 0; i < roomBtns.Length; i++) Destroy(roomBtns[i]);

string[] args = roomListStr.Split(' ');

int roomCount = int.Parse(args[1]);

roomTexts = new GameObject[roomCount];

roomBtns = new GameObject[roomCount];

for (int i = 0; i < roomCount; i++)

{

roomTexts[i] = GameObject.Instantiate(Resources.Load("TMP\_Text", typeof(GameObject))) as GameObject;

roomTexts[i].GetComponent<TMP\_Text>().text = args[i + 2];

roomTexts[i].transform.SetParent(GameObject.Find("Canvas/Image/GameObject").transform,false);

roomBtns[i] = GameObject.Instantiate(Resources.Load("Button", typeof(GameObject))) as GameObject;

roomBtns[i].transform.SetParent(GameObject.Find("Canvas/Image/GameObject").transform, false);

if (args[i + 2] == myUsername)

{

roomBtns[i].gameObject.transform.Find("Text").GetComponent<TMP\_Text>().text = "myself";

roomBtns[i].GetComponent<Button>().enabled = false;

}

string username = args[i + 2];

roomBtns[i].GetComponent<Button>().onClick.AddListener(delegate (){

this.onClick(username);

});

}

}

public void CreateRoomBtnClicked()

{

string sendStr = "createRoom ";

writer.WriteLine(sendStr);

writer.Flush();

GameObject.Find("Canvas/CreateRoomBtn").GetComponent<Button>().enabled = false;

}

public void onClick(string username)

{

string sendStr = "enterRoom " + username + " ";

writer.WriteLine(sendStr);

writer.Flush();

Debug.Log(sendStr);

}

private IEnumerator ReceiveData() // 场景二的ReceiveData

{

while (isConnected)

{

if (networkStream.DataAvailable)

{

try

{

recvStr = reader.ReadLine();

if (!string.IsNullOrEmpty(recvStr))

{

Debug.Log("收到服务器消息: " + recvStr);

string[] args = recvStr.Split(' ');

// 检查 args 数组的长度

if (args.Length > 0)

{

if (args[0] == "roomList")

{

refreshRooms(recvStr);

} else if (args[0] == "begin")

{

SceneManager.LoadScene(2);

}

}

else

{

Debug.LogWarning("接收到的消息为空或格式不正确: " + recvStr);

}

}

}

catch (Exception e)

{

Debug.LogError("接收消息失败: " + e.Message);

isConnected = false;

}

}

yield return null;

}

}

void getSocket()

{

Netword\_u1 u1 = FindObjectOfType<Netword\_u1>(); // 获取场景一的实例

myUsername = u1.myUsername;

Debug.Log(u1);

if (u1 != null)

{

clientSocket = u1.clientSocket;

networkStream = new NetworkStream(clientSocket);

writer = new StreamWriter(networkStream, Encoding.UTF8);

reader = new StreamReader(networkStream, Encoding.UTF8);

isConnected = true;

u1.StopReceivingData(); // 停止场景一的ReceiveData

StartCoroutine(ReceiveData()); // 启动场景二的ReceiveData

}

else

{

Debug.LogError("未找到场景一的 Netword\_u1 实例");

}

}

// Update is called once per frame

void Update()

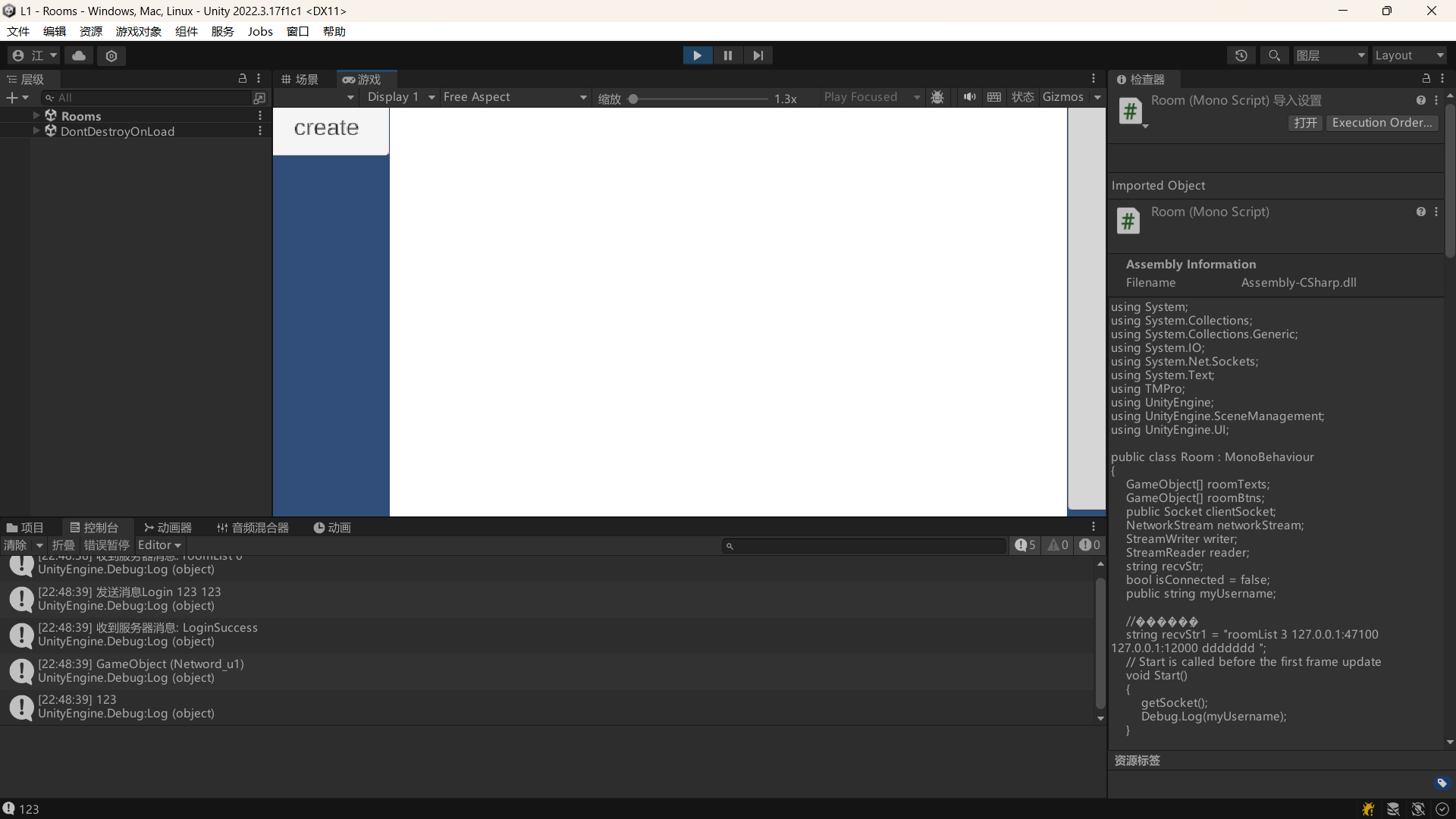
{

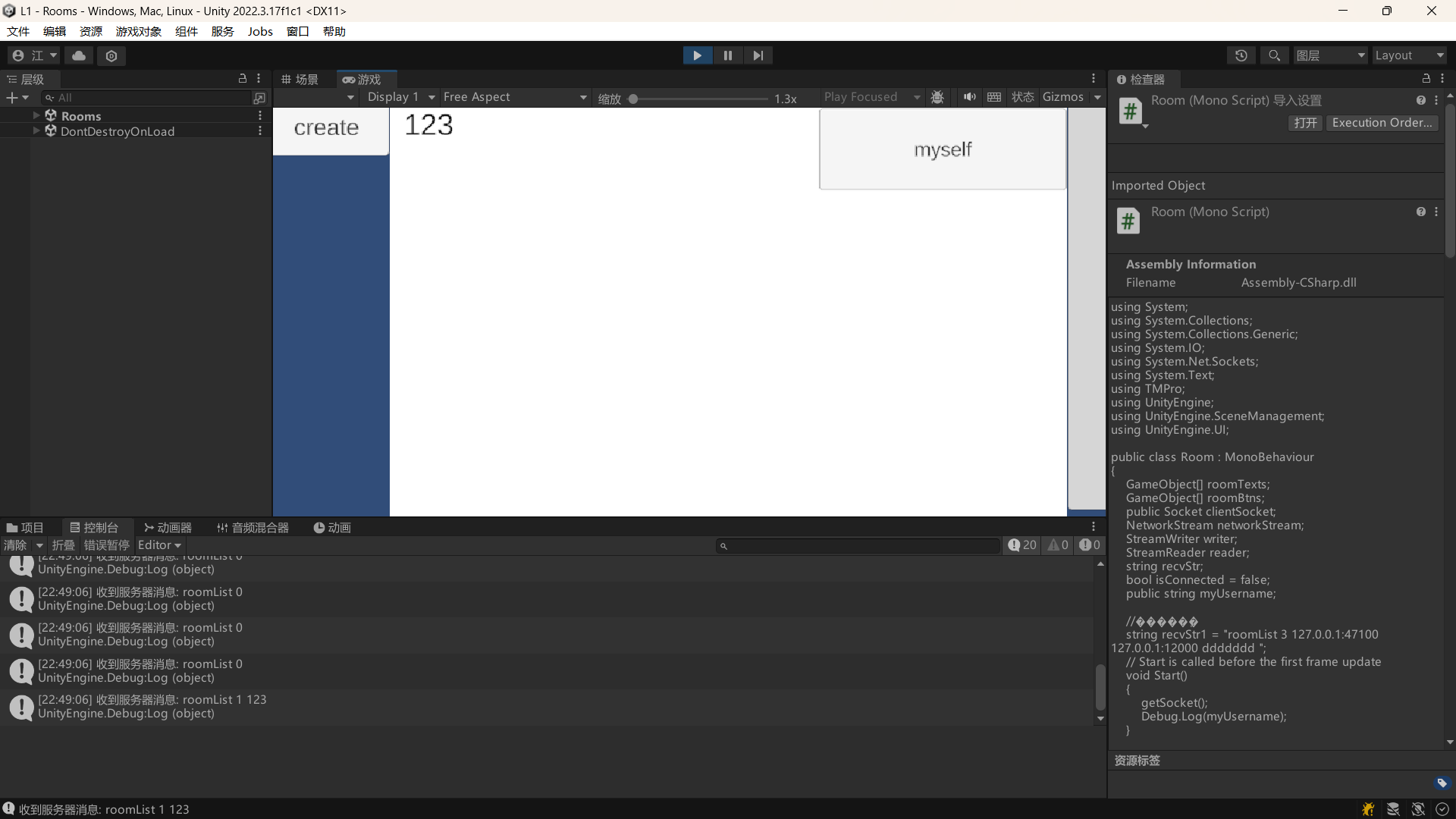
}

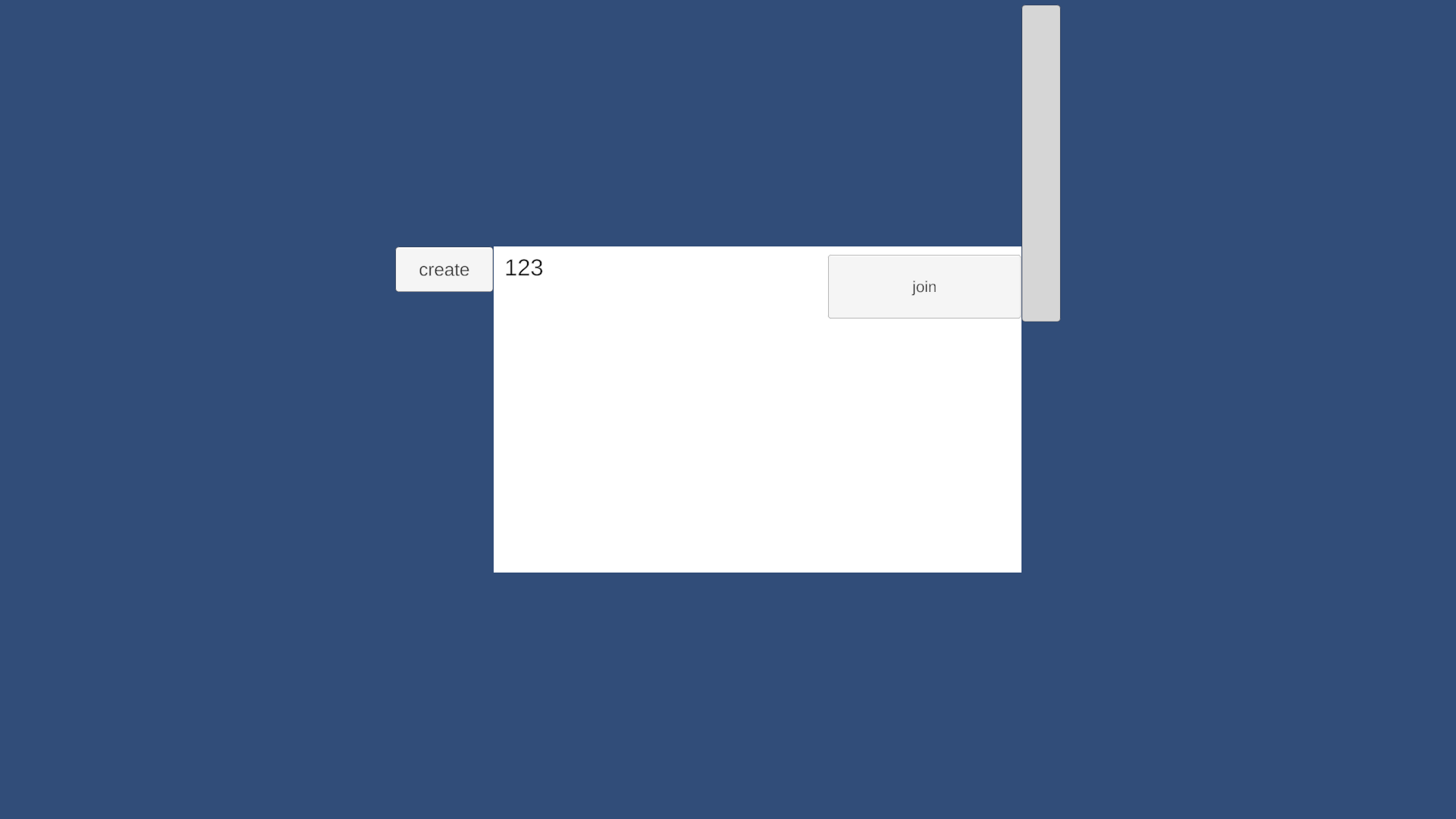
}

### 运行结果截图

房间界面：

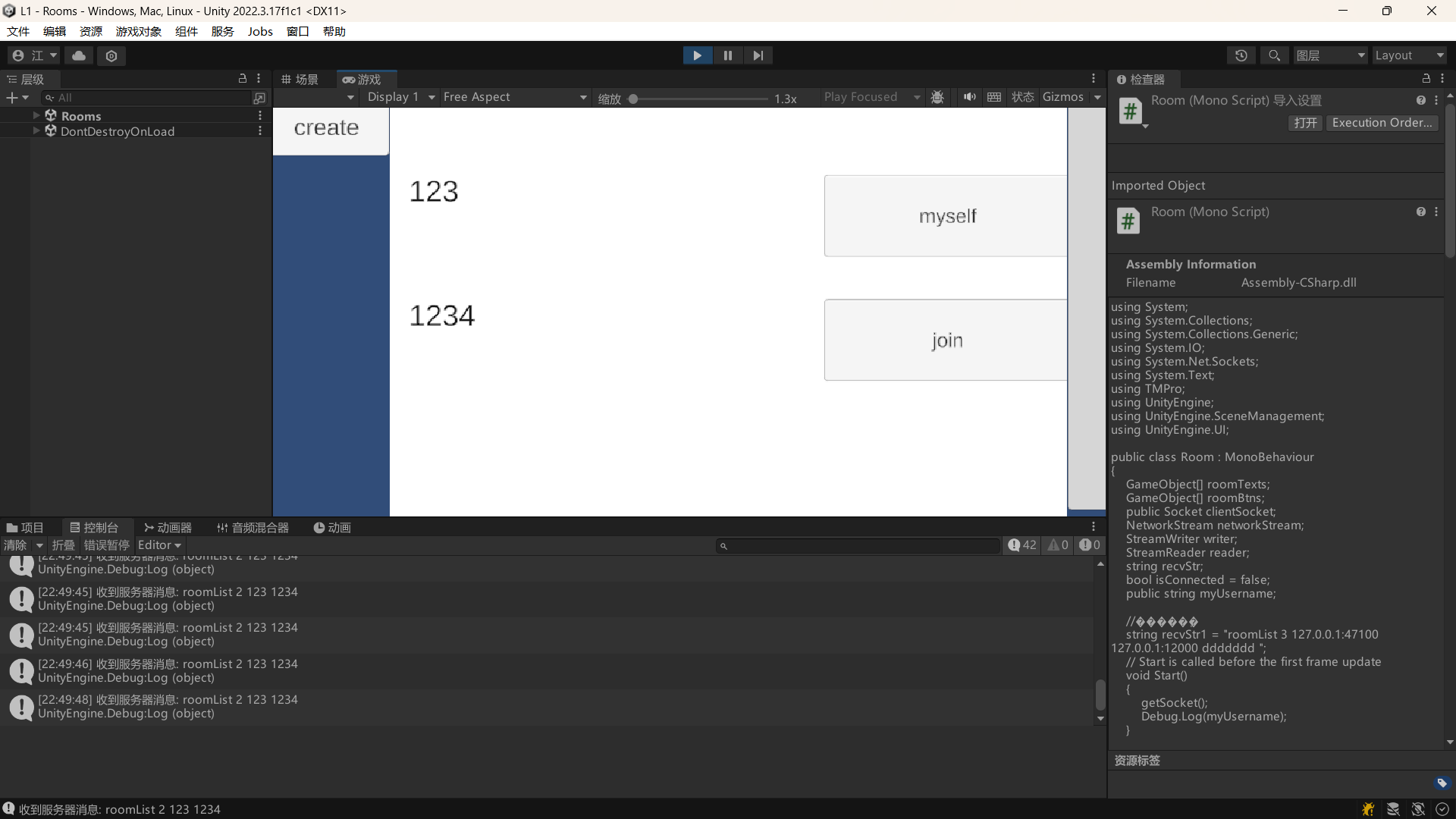


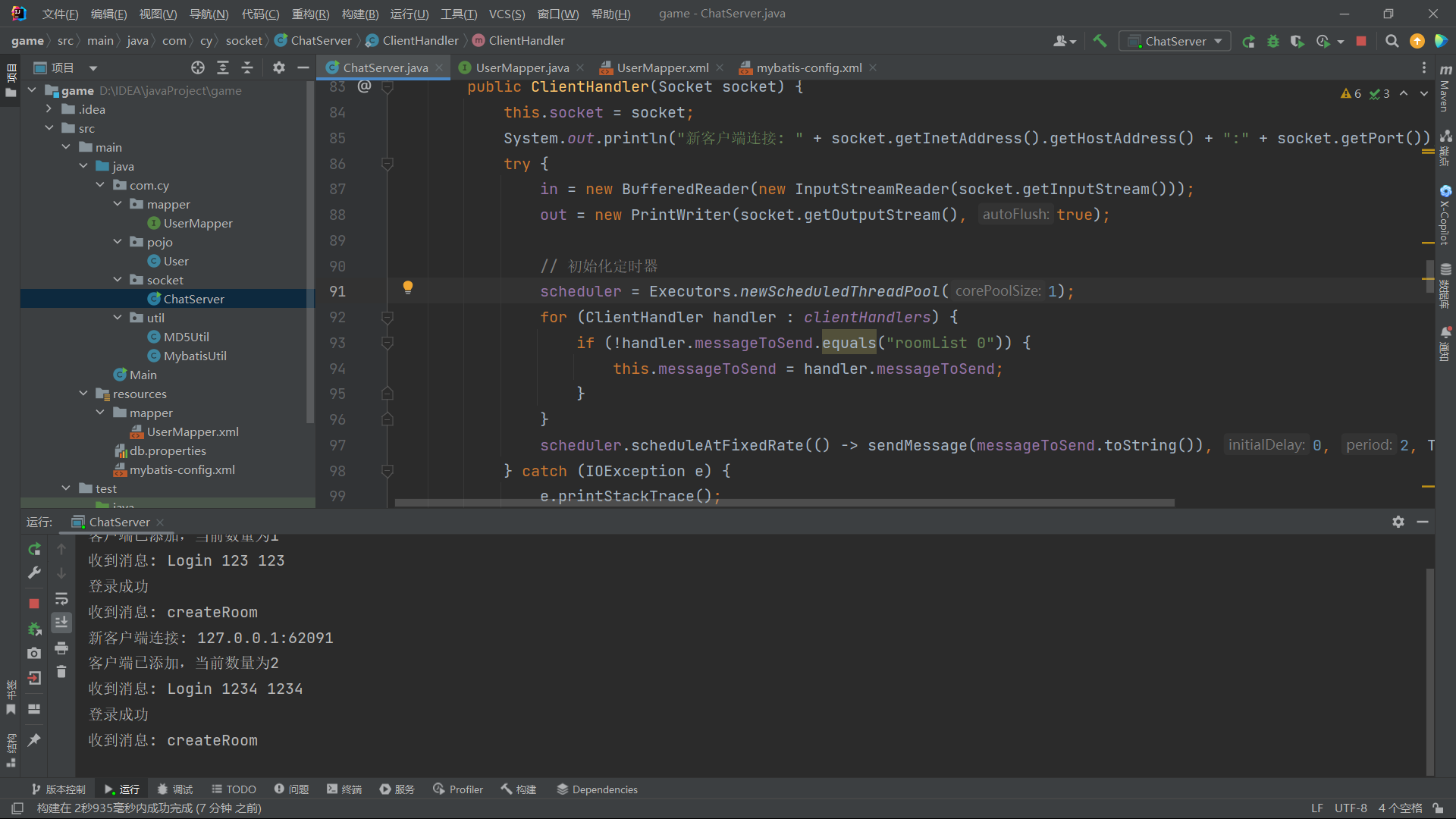




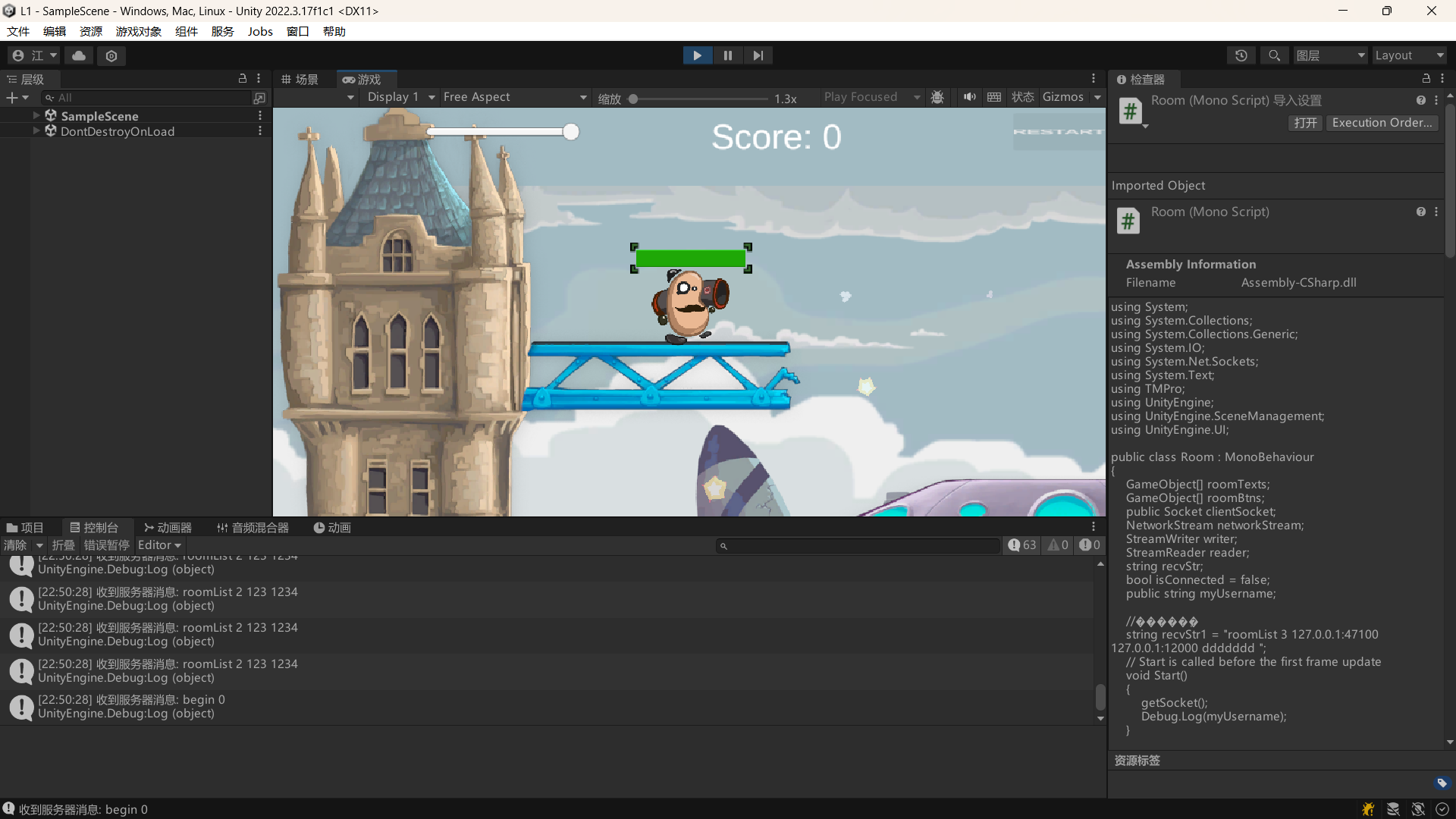
创建房间：

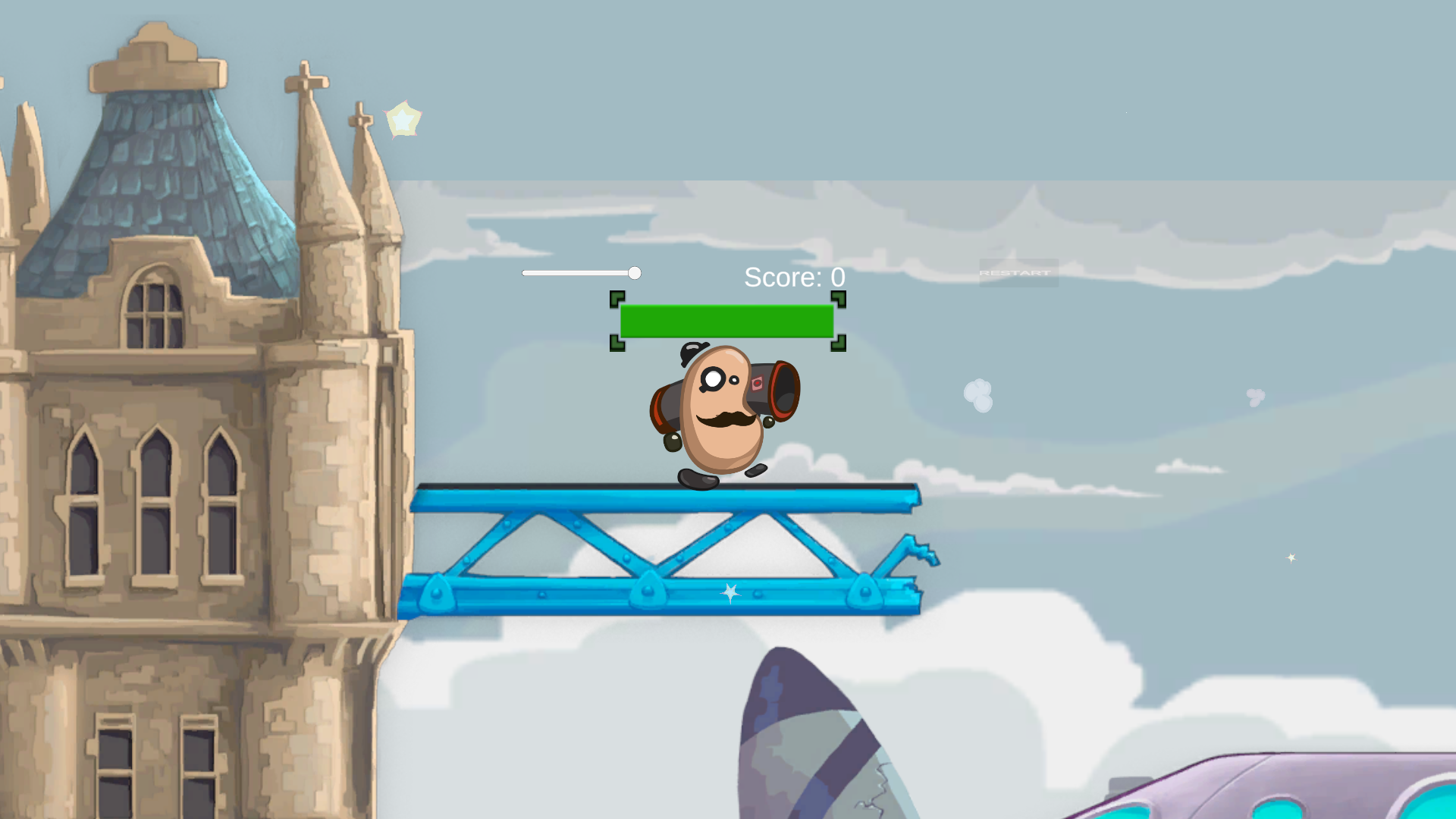


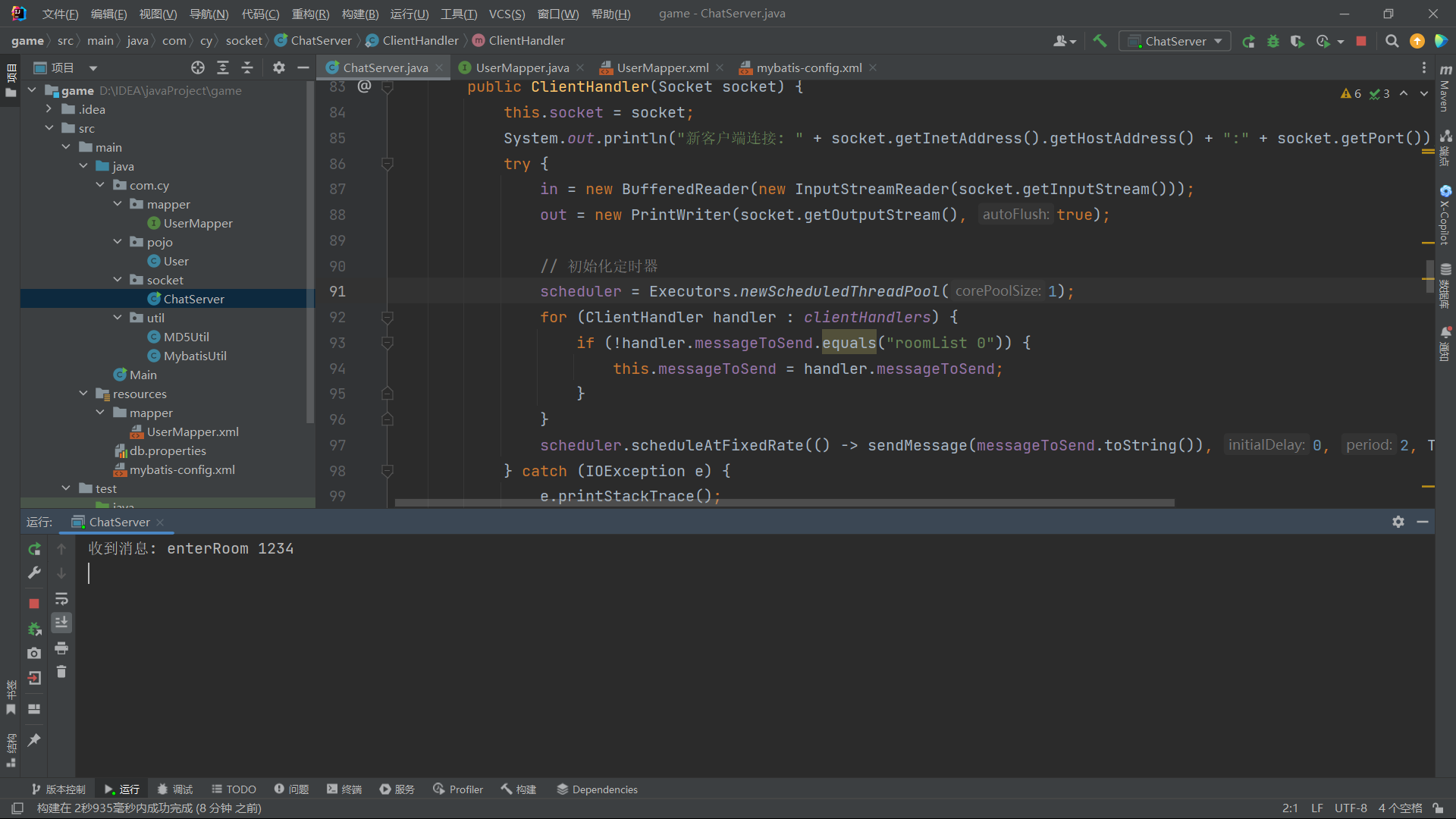




加入房间后游戏开始：







## 三【上机调试中出现的错误信息、错误原因及解决办法】

实现房间同步功能。

每创建一个房间都向所有客户端发送房间信息来进行功能实现。

## 四【上机实验中的其他它问题及心得】

通过这次实验我学会了进行服务器和客户端通信与客户端中间的同步操作。