**《Java语言程序设计》开放式大作业报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **大作业题目** | **网络联机游戏-狼人杀** | | | **类型** | **游戏** | |
| **班 号** | **120L0221** | | **学 号** | **120L022115**  **120L022109**  **120L022116**  **120L022130** | | |
| **所在院系** | **基础学部** | **学 期** | **2021年夏季学期** | **任课教师** | **于海宁** | |
| **类型** | **综合设计型** | | | | | |
| **实验目的：** | | | | | | |
| * 掌握程序设计的基本算法和简单数据结构基础，能够综合运用基本控制语句、算法和数据结构，以及自顶向下、逐步求精的模块化设计方法和面向对象的设计方法，能够设计具有小规模的系统级Java语言程序，提高系统编程能力； * 针对计算相关的复杂工程问题，能够使用恰当的算法和数据结构，完成计算、统计、排序、检索、匹配等相关的软件系统的构造、测试与实现； * 能够基于面向对象的思想进行程序的设计与实现； * 掌握常用的程序调试和测试方法。 | | | | | | |
| **实验要求：** | | | | | | |
| * 采用自顶向下、逐步求精的模块化设计思想设计一个小型信息库管理系统，或者闯关式游戏程序。 * 要求解释说明采用了什么数据结构和算法，为什么选择这种数据结构或算法，系统实现过程中遇到了哪些问题，这些问题是如何解决的，还有什么问题尚未解决，今后打算从哪几个方面进行改进，本设计的亮点和难点在哪里，实验结果如何，有哪些收获和学习体会； * 编写程序完成以下大作业内容并完成大作业报告。 | | | | | | |
| **实验内容：**  设计并实现一个网络联机的狼人杀小游戏（有图形化界面），游戏由以下流程组成：   1. 各玩家进行连接 2. 进入等候房间，等待房主（作为服务器的主机）开始游戏。 3. 进行游戏   3.0抽取胜负  3.1天黑请闭眼：狼人选择，预言家选择  3.2狼人完成选择后，判断是否有人死亡（且女巫解药未使用），若有，向女巫询问是否使用解药  3.3若女巫毒药未使用，向女巫询问是否使用毒药  3.4判断猎人是否在当局死亡，若死亡，向猎人询问是否发动技能。  3.5天亮请睁眼：宣布最后死亡情况  3.6活着的玩家轮流发言  3.7所有活着的玩家经进行投票  3.8统计投票结果  3.9若有人投票出局，判断是否为猎人，若为猎人，询问是否使用技能  3.10判断是否得出胜负  3.11若未得出，进行下一轮游戏（步骤3.1），若得出胜负，结束游戏  4、结束游戏，宣布结果，广播玩家存活情况及身份 | | | | | | |
| **实验环境：** | | | | | | |
| 操作系统：Win10  Java版本：开发版本为JDK16，经测试JAVA8不能直接编译代码，但可运行jar包  集成开发环境：Eclipse及IntelliJ IDEA  外部库：无 | | | | | | |
| **输入输出设计：** | | | | | | |
| 程序输入的数据：  用户名：String类型，便于玩家区分自己和其他玩家（若不输入，可自动生成且保证不重复）  IP：String，用于和房主（作为服务器的主机）进行连接  若输入错误的IP，将会告知用户，服务器未开启  发言字符串：String类型，用于发言，与其他玩家交流  程序输出的数据：  程序输出的数据都在图形化界面内部展示  均为String型  用户体验：借助javax.swing下的类实现了图形化界面，用户只需要进行点击和少量的输入即可 | | | | | | |
| **系统设计与实现:** | | | | | | |
| 1. **系统功能模块划分**   对系统进行自顶向下的模块分解，画出系统各个功能模块之间的结构图如下：     1. **类设计**   本系统总计设计了36个类，每个类的数据成员和成员函数设计如下所示。  (1)KillWolf类的数据成员和成员函数设计：   |  |  |  |  | | --- | --- | --- | --- | | 序号 | 数据成员 | 数据类型 | 意义说明 | | 1 | server | sServer | 服务端对象 | | 2 | client | sClient | 客户端对象 | | 3 | freeSpeak | boolean | 是否自由发言 | | 4 | isServer | boolean | 是否为服务器 | | 5 | Serverlp | String | 服务器IP | | 6 | Mip | String | 用户自己的IP | | 7 | UserName | String | 用户自己的名称 | | 8 | Users | DataUser[] | 人员数组 | | 9 | UsersLen | int | 数组占用数 | | 10 | UsersRealLen | int | 人数 | | 11 | loginwindow | LoginWindow | 登录房间页面 | | 12 | WaitRoom | WaitRoom | 等待房间页面 | | 13 | GameMain | GameMain | 游戏房间页面 | | 14 | SelectWindow | SelectWindow | 选择页面 |  |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | main | 主函数，获取IP启动登录页面 | String | void | | 2 | goWaitRoom | 进入等待房间 | 无 | void | | 3 | goGameMain | 进入游戏房间 | 无 | void |   (2) serverData类的数据成员和成员函数设计：   |  |  |  |  | | --- | --- | --- | --- | | 序号 | 数据成员 | 数据类型 | 意义说明 | | 1 | StartSpeakName | String | 第一位发言的用户名 | | 2 | nowSpeakUser | String | 正在发言的用户名 | | 3 | deathUser | String | 狼人刀杀用户 | | 4 | deathUser2 | String | 女巫毒药用户 | | 5 | deathUser3 | String | 猎人枪杀用户 | | 6 | deathUser4 | String | 投票出局用户 | | 7 | GameMainThread | GameMainThread | 当前天的主线程 | | 8 | Day | int | 游戏天数 | | 9 | night | boolean | 是否夜晚 | | 10 | playMode | boolean | 游戏模式 | | 11 | freeSpeak | boolean | 是否自由发言 | | 12 | ServerUsers | DataUser[] | 服务端用户 | | 13 | UsersLen | int | 服务端用户数组占用 | | 14 | UsersRealLen | int | 服务端用户数量 | | 15 | selection | SelectionEvent[] | 选择活动数组 | | 16 | selectionLen | int | 选择活动数组占用 |  |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | getUsersWithID | 通过身份获取用户对象 | String | DataUser[] | | 2 | ArrayDatatoName | 将用户对象数组转换为用户名数组 | DataUser[] | String[] | | 3 | getLivingUsers | 获取未出局用户对象 | boolean | Object[] | | 4 | isAllWolfDied | 判断所有狼死亡 |  | boolean | | 5 | isAllPeoDied | 判断所有平民死亡 |  | boolean | | 6 | isAllGodDied | 判断所有神死亡 |  | boolean | | 7 | whoWin | 判断是否胜利 |  | String | | 8 | getUser | 通过用户名获取用户 | String | DataUser |   (3) DataUser类的数据成员和成员函数设计：   |  |  |  |  | | --- | --- | --- | --- | | 序号 | 数据成员 | 数据类型 | 意义说明 | | 1 | sk | Socket | 用户端口对象 | | 2 | port | int | 用户端口号 | | 3 | name | String | 用户名 | | 4 | live | boolean | 是否存活 | | 5 | identify | String | 身份 | | 7 | havePoison | boolean | 有毒药 | | 8 | haveAntidote | boolean | 有解药 |   (4) SelectCallBack接口的数据成员和成员函数设计：   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | run | 处理选择回调 | string | void |   (5) SelectionEvent类的数据成员和成员函数设计：   |  |  |  |  | | --- | --- | --- | --- | | 序号 | 数据成员 | 数据类型 | 意义说明 | | 1 | Selection | SelectionEvent[] | 静态 所有选择活动 | | 2 | SelectionLen | int | 静态 所有选择活动数量 | | 4 | open | boolean | 是否正在开启 | | 5 | users | String[] | 被投票用户 | | 6 | count | int[] | 计数 | | 7 | openusers | String[] | 参与用户 | | 8 | target | String[] | 目标用户 | | 9 | timeLimit | int | 时间限制 | | 10 | cb | SelectionEventCallBack | 回调函数 | | 11 | server | sServer | 服务器 |  |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | SelectionEvent | 构造选择活动 | Sting,SelectionEventCallBack |  | | 2 | update | 更新选择 | String | void | | 3 | isFinish | 判断是否完成 |  | boolean | | 4 | getResult | 获取结果 |  | int | | 5 | closeSelectio | 结束 | sServer,boolean | boolean | | 6 | openSeletion | 开启 | sServer,string,boolean,Datauser | void | | 7 | UserQuit | 用户退出 | String |  |   (6) ThreadWait类的数据成员和成员函数设计：   |  |  |  |  | | --- | --- | --- | --- | | 序号 | 数据成员 | 数据类型 | 意义说明 | | 1 | s | sServer | 服务器 |  |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | ThreadWait | 构造线程等待 | sServer |  | | 2 | run | 执行线程等待 |  | void |   (7)SelectionEventCallBack接口的数据成员和成员函数设计：   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | run | 回调函数 | String,int | void |   (8) ClientDeal类的数据成员和成员函数设计：   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | run | 回调函数 | String,socket | void |   (9) SleepThread类的数据成员和成员函数设计：   |  |  |  |  | | --- | --- | --- | --- | | 序号 | 数据成员 | 数据类型 | 意义说明 | | 1 | sss | int | 等待时长 |  |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | SleepThread | 构造线程等待 | string |  | | 2 | run | 执行线程等待 |  | void |   (10) GameMainThread类的数据成员和成员函数设计：   |  |  |  |  | | --- | --- | --- | --- | | 序号 | 数据成员 | 数据类型 | 意义说明 | | 1 | YuYancompleted | boolean | 预言是否完成 | | 2 | Wizardcompleted | boolean | 女巫是否完成 |  |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | run | 执行 |  | void | | 2 | gameNight | 入夜 |  | void | | 3 | gameWizard | 女巫选择 | string | void | | 4 | gameDay | 白天 |  | void |   (11) TimeTha类的数据成员和成员函数设计：   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | dayWork | 公布结果 |  | void | | 2 | gamevote | 投票 |  | void | | 3 | gameHunt | 猎人 | string | void | | 4 | gameNext | 下一步 |  | void | | 5 | gameEnd | 结束 |  | void | | 6 | isWin | 是否胜利 |  | boolean |   (12)ServerDeal类的数据成员和成员函数设计：   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | run | 处理回调 | String,socket | void | | 2 | nextSpeak | 下一个发言 | String,int | boolean |   (13)SpeakThread类的数据成员和成员函数设计：   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 数据成员 | 数据类型 | 意义说明 | 取值范围 | | 1 | user | String | 正在发言的用户 |  | | 2 | Day | int | 天数 |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | SpeakThread | 构造线程等待 | String,int |  | | 2 | run | 执行 |  | void |   (14)WolfSelect类的数据成员和成员函数设计：   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | run | 执行狼人选择 | String,int | void |   (15) VoteSelection类的数据成员和成员函数设计：   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | run | 执行投票选择 | String,int | void |   (16) GameMain类的数据成员和成员函数设计：   |  |  |  |  | | --- | --- | --- | --- | | 序号 | 数据成员 | 数据类型 | 意义说明 | | 1 | freeSpeak | boolean | 是否自由发言 | | 2 | UserListData | string | 存活JList数据 | | 3 | UserListTipData | string | 提示JList数据 | | 4 | UserOutListData | string | 淘汰JList数据 | | 5 | ClockRun | boolean | 是否正在运行Clock |  |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | reset | 启动 |  | void | | 3 | forbidSubmit | 禁用发言 |  | void | | 4 | ableSubmit | 启用发言 |  | void | |  |  |  |  |  |   (17)TimeTh类的数据成员和成员函数设计：   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | startSpeakClock | 启用发言时钟 |  | void | | 2 | submit | 提交发言 |  | void | | 3 | addToScreen | 添加到公屏 | String | void | | 4 | setDark | 设置窗口风格 |  | void | | 5 | setWhite | 设置窗口风格 |  | void | | 28 | getUserListData |  | String |  | | 29 | setUserListData |  | String | void | | 30 | getUserOutListData |  | String |  | | 31 | setUserOutListData |  | String | void |   (18) ClientDeal类的数据成员和成员函数设计：   |  |  |  |  | | --- | --- | --- | --- | | 序号 | 数据成员 | 数据类型 | 意义说明 | | 1 | ServerButton | JButton | 作为服务器按钮 | | 2 | LoginButton | JButton | 登录按钮 | | 3 | userName | JTextField | 用户名称输入框 | | 4 | Serverlp | JTextField | 服务器地址输入框 | | 5 | window | JFrame | 窗口 | | 6 | MIP | String | 自己的ip |  |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | Reset | 重置 | string | void |   (19) ButtonAction类的数据成员和成员函数设计：   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | actionPerformed | 按钮点击 | ActionEvent event | void |   (20) LinkAction类的数据成员和成员函数设计：   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | actionPerformed | 按钮点击 | ActionEvent event | void | | 2 | createClient | 创建客户端 | string | void | | 3 | setName | 设置名称 |  | boolean |   （21）SelectWindow类的数据成员和成员函数设计：   |  |  |  |  | | --- | --- | --- | --- | | 序号 | 数据成员 | 数据类型 | 意义说明 | | 1 | isRunCB | boolean | 是否已回调 | | 2 | ReceivelD | String | 特征值 | | 3 | ReceiveFromServer | boolean | 是否联网投票 | | 4 | DisableParent | boolean | 禁用父窗口 | | 5 | ParentWindow | JFrame | 父窗口 | | 6 | window | JFrame | 窗口 | | 7 | CenterLabel | JLabel | 中心标签 | | 8 | List | JList String | 列表 | | 9 | name | String | 名称 | | 10 | ConfirmButton | JButton | 确认按钮 | | 11 | CloseButton | JButton | 关闭按钮 | | 12 | originUsers | String | 原始用户 | | 13 | usersTip | String | 用户提示 | | 14 | targetUsers | String | 目标用户 | | 15 | selectCount | int | 选择计数 | | 16 | LastSelect | String | 最后选择 | | 17 | isSendFinal | boolean | 是否已发送 | | 18 | DarkMode | boolean | 窗口风格 | | 19 | cb | SelectCallBack | 回调函数 |  |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | setColorMode | 设置风格 | boolean | void | | 2 | start | 开始 | String,boolean,int | void | | 3 | SelectWindow | 构造 | String,boolean,JFrame) |  |   （22）TimeTh类的数据成员和成员函数设计：   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | setTitle | 设置标题 | string | void | | 2 | SELECTInfo |  | string | void |   （23）ButtonAction类的数据成员和成员函数设计：   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | ButtonAction | 按钮点击回调 | string |  | | 2 | actionPerformed |  | (ActionEvent) | void | |  | getSelectionAndClose |  | String | void |   （24）WaitRoom类的数据成员和成员函数设计：   |  |  |  |  | | --- | --- | --- | --- | | 序号 | 数据成员 | 数据类型 | 意义说明 | | 1 | LoginButton | JButton | 登录按钮 | | 2 | peoCount | JTextField | 平民人数 | | 3 | wolfCount | JTextField | 狼人数 | | 4 | wizardOption | JCheckBox | 启用女巫 | | 5 | hunterOption | JCheckBox | 启用猎人 | | 6 | yuyanjiaOption | JCheckBox | 启用预言家 | | 7 | winmodeOption | JCheckBox | 启用屠城模式 | | 8 | speakOption | JCheckBox | 启用自由发言 | | 9 | UserList | JList | 用户列表 | | 10 | window | JFrame | 窗口 | | 11 | Userlenlabel | JLabel |  | | 12 | Charlenlabel | JLabel |  | | 13 | CharLen | int |  | | 14 | peoLen | int |  | | 15 | wolfLen | int |  | | 16 | wizardLen | int |  | | 17 | hunterLen | int |  | | 18 | yuyanjiaLen | int |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | reset |  | String,boolean | void | | 2 | setUserCount() |  | String | void |   （25）CheckValueChanged 类的数据成员和成员函数设计：   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | CheckValueChanged |  | String |  | | 2 | itemStateChanged() |  | ItemEvent | void |   （26）TextListener类的数据成员和成员函数设计：   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | TextListener() |  | String |  | | 2 | insertUpdate(DocumentEvent) |  | DocumentEven | void | | 3 | removeUpdate(DocumentEvent) |  | DocumentEven | void | | 4 | changedUpdate(DocumentEvent) |  | DocumentEvent | void |   （27）CountKeyListener类的数据成员和成员函数设计：   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | keyTyped |  | KeyEvent | void | | 2 | keyPressed(KeyEvent) |  | KeyEvent | void | | 3 | keyReleased() |  | KeyEvent | void |   （30）CallBack接口的数据成员和成员函数设计：   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | run |  | String, Socket | void |   （31）ClientThread类的数据成员和成员函数设计：   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 数据成员 | 数据类型 | 意义说明 | 取值范围 | | 1 | PORT | int |  |  | | 2 | Log | boolean |  |  | | 3 | sk | Socket |  |  | | 4 | cb | CallBack |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | wlog |  | string | void | | 2 | close0 |  |  | boolean | | 3 | send |  | string | boolean | | 4 | start |  | String,CallBack) | boolean |   （32）sIP类的数据成员和成员函数设计：   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 数据成员 | 数据类型 | 意义说明 | 取值范围 | | 1 | Log | boolean |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | getIP |  | string |  |   （33）GetRealLocalIP类的数据成员和成员函数设计：   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | getRealIP() |  | string |  |   （34）sServer类的数据成员和成员函数设计：   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 数据成员 | 数据类型 | 意义说明 | 取值范围 | | 1 | PORT | int |  |  | | 2 | Log | boolean |  |  | | 3 | ss | ServerSocket |  |  | | 4 | users | Socket |  |  | | 5 | usersLen | int |  |  | | 6 | cb | CallBack |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | wlog(String) |  | string | void | | 2 | close |  |  | boolean | | 3 | sendto(String |  | String,socket | boolean | | 4 | send(String) |  | string | boolean | | 5 | start(CallBack) |  | CallBack | boolean | |  |  |  |  |  |   （35）ServerThreadAC类的数据成员和成员函数设计：   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 数据成员 | 数据类型 | 意义说明 | 取值范围 | | 1 | id | int |  |  | | 2 | Log | boolean |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | run |  |  | void |   （36）ServerThread类的数据成员和成员函数设计：   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 数据成员 | 数据类型 | 意义说明 | 取值范围 | | 1 | sk | Socket |  |  | | 2 | id | int |  |  | | 3 | Log | boolean |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | | 序号 | 成员函数 | 函数功能 | 函数参数 | 函数返回值 | | 1 | ServerThread |  | Socket |  | | 2 | run |  |  | void |   类及类之间的关系如图所示：     1. **数据结构**   整型数组、对象数组、容器类   1. **算法**   枚举/递推/迭代/分类统计、排序/查找、递归算法   1. **程序流程图**   系统总体流程图如下： | | | | | | |
| **实验过程中遇到的问题及解决方法与思路：** | | | | | | |
| 问题1：用户选择过程中主进程阻塞。  原因：主线程Sleep。  解决方法：启用新线程进行Sleep，再执行回调。  问题2：用户淘汰的列表更新。  原因：用户断线、多端数据异步通信、底层数据库和图形页面通信。  解决方法：启用静态变量记录窗口，在用户断线时对所有用户的所有开放窗口进行广播。  问题3：用户中途退出导致发言顺序出错  原因：记录第一位发言的玩家的ID进行轮流发言，直到再次轮到该玩家则结束发言。  解决方法：用户退出则重新开始发言。  问题4：游戏主体窗口公屏显示问题。  原因：对swing中组件使用问题。  解决方法：增加滚动条：增加JPanel, JScrollPane并进行设置-设置屏幕显示：  screen.setEditable(false);//设置只读          screen.setLineWrap(true);        //激活自动换行功能          screen.setWrapStyleWord(true);            // 激活断行不断字功能 | | | | | | |
| **测试用例和系统测试结果：** | | | | | | |
| 测试用例1：    测试用例2：    测试用例3： | | | | | | |
| 程序源代码 | | | | | | |
| 1. //文件名KillWolf.java 2. package KillWolf; 3. import java.net.Socket; 4. import java.util.Scanner; 5. import javax.swing.JFrame; 6. import KillWolf.Data.DataUser; 7. import KillWolf.Data.SelectionEvent; 8. import KillWolf.Data.SelectionEventCallBack; 9. import KillWolf.KillWolf.serverData; 10. import KillWolf.SocketDeal.GameMainThread; 11. import KillWolf.Window.GameMain; 12. import KillWolf.Window.LoginWindow; 13. import KillWolf.Window.SelectWindow; 14. import KillWolf.Window.WaitRoom; 15. import SocketServe.\*; 16. public class KillWolf { 17. public class serverData { 18. public static String StartSpeakName = null; 19. //public static int startSpeakUserArrayId; 20. public static String nowSpeakUser; 21. //public static int nowSpeakUserArrayId; 22. public static String deathUser = null; //狼人杀死/女巫解救 23. public static String deathUser2 = null; //女巫毒死 为\*\*NONE\*\*表示无人死亡 24. public static String deathUser3 = null; //猎人杀死 25. public static String deathUser4 = null; //投票出局 26. public static GameMainThread GameMainThread = null; 27. public static int Day = 0; 28. public static boolean night = false; 29. public static boolean playMode = false; //true表示屠城玩法，false表示屠边玩法 30. public static boolean freeSpeak = false; 31. public static DataUser ServerUsers[] = new DataUser[50]; 32. public static int UsersLen = 0; 33. public static int UsersRealLen = 0; 34. public static SelectionEvent Selection[] = new SelectionEvent[200]; 35. public static int SelectionLen = 0; 36. public static DataUser[] getUsersWithID(String identify) { 37. int[] a = new int[ServerUsers.length]; 38. int n = 0; 39. for(int i=0;i< UsersLen;i++) { 40. if(ServerUsers[i] != null) 41. if(ServerUsers[i].identify.equals(identify)) { 42. a[n] = i; 43. n++; 44. } 45. } 46. DataUser[] b = new DataUser[n]; 47. for(int i=0;i<n;i++) b[i] = ServerUsers[a[i]]; 48. return b; 49. } 50. public static String[] ArrayDatatoName(DataUser[] d) { 51. int[] a = new int[d.length]; 52. int n = 0; 53. for(int i=0;i<d.length;i++) { 54. if(d[i] != null) 55. if(d[i].live) { 56. a[n] = i; 57. n++; 58. } 59. } 60. String[] b = new String[n]; 61. for(int i=0;i<n;i++) { 62. b[i] = d[a[i]].name; 63. } 64. return b; 65. } 66. public static Object[] getLivingUsers(boolean onlyName) { 67. int[] a = new int[ServerUsers.length]; 68. int n = 0; 69. for(int i=0;i<UsersLen;i++) { 70. if(ServerUsers[i] != null) 71. if(ServerUsers[i].live) { 72. a[n] = i; 73. n++; 74. } 75. } 76. String[] b = new String[n]; 77. DataUser[] c = new DataUser[n]; 78. for(int i=0;i<n;i++) { 79. if(onlyName) b[i] = ServerUsers[a[i]].name; 80. else c[i] = ServerUsers[a[i]]; 81. System.out.println("LIVING : "+ServerUsers[a[i]].name); 82. } 83. if(onlyName) return b; 84. else return c; 85. } 86. public static boolean isAllWolfDied() { 87. DataUser[] b = getUsersWithID("狼人"); 88. for(int i=0;i<b.length;i++) if(b[i].live) return false; 89. return true; 90. } 91. public static boolean isAllPeoDied() { 92. DataUser[] b = getUsersWithID("平民"); 93. for(int i=0;i<b.length;i++) if(b[i].live) return false; 94. return true; 95. } 96. public static boolean isAllGodDied() { 97. DataUser[] b = getUsersWithID("猎人"); 98. for(int i=0;i<b.length;i++) if(b[i].live) return false; 99. b = getUsersWithID("预言家"); 100. for(int i=0;i<b.length;i++) if(b[i].live) return false; 101. b = getUsersWithID("女巫"); 102. for(int i=0;i<b.length;i++) if(b[i].live) return false; 103. return true; 104. } 105. public static String whoWin() { 106. for (int i=0;i<serverData.UsersLen;i++){ 107. if(serverData.ServerUsers[i]!=null){ 108. System.out.println(serverData.ServerUsers[i].name +" " 109. +serverData.ServerUsers[i].identify+" "+serverData.ServerUsers[i].live); 110. } 111. } 112. if(playMode) { 113. if(isAllWolfDied()) return "平民"; 114. if(isAllPeoDied() && isAllGodDied()) return "狼人"; 115. return "无"; 116. }else { 117. if(isAllWolfDied()) return "平民"; 118. if(isAllPeoDied() || isAllGodDied()) return "狼人"; 119. return "无"; 120. } 121. } 122. public static DataUser getUser(String name) { 123. for(int i=0;i<UsersLen;i++) { 124. if(ServerUsers[i] != null) 125. if(ServerUsers[i].name.equals(name)) { 126. return ServerUsers[i]; 127. } 128. } 129. return null; 130. } 132. } 134. public static sServer server; 135. public static sClient client; 136. public static boolean freeSpeak = false; 137. public static boolean isServer = false; 138. public static String ServerIp; 139. public static String Mip; 140. public static String UserName; 141. public static DataUser Users[] = new DataUser[50]; 142. public static int UsersLen = 0; 143. public static int UsersRealLen = 0; 144. public static LoginWindow loginwindow; 145. public static WaitRoom WaitRoom; 146. public static GameMain GameMain; 147. public static SelectWindow SelectWindow; 148. public static void main(String[] args) { 149. //Scanner sn = new Scanner(System.in); 150. Mip = sIP.getIP(); 152. loginwindow = new LoginWindow(); 153. loginwindow.reset(Mip); 154. System.out.println("Server on IP: "+Mip); 155. }  158. public static void goWaitRoom() { 160. loginwindow.window.setVisible(false); 161. WaitRoom = new WaitRoom(); 162. WaitRoom.reset(Mip,UserName,isServer); 163. } 164. public static void goGameMain() { 165. WaitRoom.window.setVisible(false); 166. GameMain = new GameMain(); 167. GameMain.reset(); 168. } 169. } 170. // CallBack.java 171. package SocketServe; 172. /\* 173. \* author: wbx 174. \*/ 175. import java.net.Socket; 176. public interface CallBack { 177. public void run(String text,Socket sk); 178. } 179. // sClient.java 180. package SocketServe; 181. /\* 182. \* author: wbx 183. \*/ 184. import java.net.\*; 185. import java.io.\*; 186. import java.util.Scanner; 187. public class sClient { 188. public int PORT = 15648; 189. public boolean Log = true; 190. public Socket sk = null; 191. public CallBack cb = null; 192. void wlog(String log) { 193. if(Log) System.out.println(log); 194. } 195. public boolean close() { 196. try { 197. sk.close(); 198. }catch(Exception e) { 199. e.printStackTrace(); 200. return false; 201. } 202. return true; 203. } 204. public boolean send(String Text) 205. { 206. try{ 207. PrintStream ps = new PrintStream(sk.getOutputStream()); 208. ps.println(Text); 209. ps.flush(); 210. //if(Text.equals("END")) { 211. // sk.close(); 212. //}else { 213. this.wlog(sk.getLocalPort()+" Client Send: "+Text); 214. //} 215. } 216. catch(Exception e){ 217. e.printStackTrace(); 218. return false; 219. } 220. return true; 221. } 222. public boolean start(String ServerIp,CallBack cb) //= "127.0.0.1") 223. { 224. this.cb = cb; 225. try{ 226. sk = new Socket(ServerIp,PORT); 227. } 228. catch(Exception e){ 229. e.printStackTrace(); 230. return false; 231. } 232. this.wlog("Connecting Server on "+ServerIp+","+PORT); 233. try { 234. ClientThread st = new ClientThread(sk); 235. st.start(); 236. }catch(Exception e) { 237. e.printStackTrace(); 238. return false; 239. } 240. this.wlog("Server is success connected on "+ServerIp+","+PORT); 241. return true; 242. } 243. //�ͻ����߳��� 244. class ClientThread extends Thread 245. { 246. Socket sk; 247. public ClientThread(Socket sk){ 248. this.sk= sk; 249. } 250. public void run() { 251. try { 252. Scanner sn = new Scanner(sk.getInputStream()); 253. while(true) { 254. if(sn.hasNextLine()) { 255. String line = sn.nextLine();//br.readLine(); 256. //System.out.println(sk.getLocalPort()+" Client Receive: "+line); 257. cb.run(line,sk); 258. }else { 259. try { 260. sk.sendUrgentData(0); 261. } catch (IOException e) { 262. System.out.println("Disconnect from Server"); 263. cb.run("Disconnect from Server",sk); 264. break; 265. } 266. } 267. } 268. } catch (IOException e1) { 269. e1.printStackTrace(); 270. } 271. } 272. } 273. } 274. //sIP.java 275. package SocketServe; 276. /\* 277. \* author: wbx 278. \*/ 279. import java.net.\*; 280. import java.util.Enumeration; 281. public class sIP { 282. public static boolean Log = true; 283. public static String getIP() {//获取本机IP，返回空为失败 284. String MIP = ""; 285. try{ 286. //InetAddress addr = InetAddress.getLocalHost(); 287. //MIP = addr.getHostAddress(); 288. //if(Log) System.out.println("本机IP地址："+MIP); 289. //String hostname = addr.getHostName(); 290. //if(Log) System.out.println("本机名称："+hostname); 291. //getAllIpAddress(); 292. MIP = GetRealLocalIP.getRealIP(); 293. // System.out.println(); 294. }catch(Exception e){ 295. e.printStackTrace(); 296. return ""; 297. } 298. return MIP; 299. } 300. /\*\* 301. \* 获取本地真正的IP地址，即获得有线或者无线WiFi地址。 302. \* 过滤虚拟机、蓝牙等地址 303. \* @author yins 304. \* @date 2018年8月12日 下午9:53:58 305. \*/ 306. public class GetRealLocalIP { 307. /\*\* 308. \* 获取本地真正的IP地址，即获得有线或者无线WiFi地址。 309. \* 过滤虚拟机、蓝牙等地址 310. \* @author yins 311. \* @date 2018年8月12日下午9:56:35 312. \* @return 313. \*/ 314. public static String getRealIP() { 315. try { 316. Enumeration<NetworkInterface> allNetInterfaces = NetworkInterface.getNetworkInterfaces(); 317. while (allNetInterfaces.hasMoreElements()) { 318. NetworkInterface netInterface = (NetworkInterface) allNetInterfaces.nextElement(); 319. // 去除回环接口，子接口，未运行和接口 320. if (netInterface.isLoopback() || netInterface.isVirtual()|| !netInterface.isUp())continue; 321. if (!netInterface.getDisplayName().contains("Intel")&& 322. !netInterface.getDisplayName().contains("Realtek")) continue; 323. Enumeration<InetAddress> addresses = netInterface.getInetAddresses(); 324. System.out.println(netInterface.getDisplayName()); 326. while (addresses.hasMoreElements()) { 327. InetAddress ip = addresses.nextElement(); 328. if (ip != null) { 329. // ipv4 330. if (ip instanceof Inet4Address) { 331. System.out.println("ipv4 = " + ip.getHostAddress()); 332. return ip.getHostAddress(); 333. } 334. } 335. } 336. break; 337. } 338. } catch (SocketException e) { 339. System.err.println("Error when getting host ip address" 340. + e.getMessage()); 341. } 342. return null; 343. } 344. } 345. /\* 347. \* This method is used to get all ip addresses from the network interfaces. 348. \* network interfaces: eth0, wlan0, l0, vmnet1, vmnet8 350. public static void getAllIpAddress() { 351. try { 352. //get all network interface 353. Enumeration<NetworkInterface> allNetworkInterfaces = NetworkInterface.getNetworkInterfaces(); 354. NetworkInterface networkInterface = null; 356. //check if there are more than one network interface 357. while (allNetworkInterfaces.hasMoreElements()) { 358. //get next network interface 359. networkInterface = allNetworkInterfaces.nextElement(); 360. //output interface's name 361. //System.out.println("network interface: " + networkInterface.getDisplayName()); 363. //get all ip address that bound to this network interface 364. Enumeration<InetAddress> allInetAddress = networkInterface.getInetAddresses(); 366. InetAddress ipAddress = null; 368. //check if there are more than one ip addresses 369. //band to one network interface 370. while (allInetAddress.hasMoreElements()) { 371. //get next ip address 372. ipAddress = allInetAddress.nextElement(); 373. if (ipAddress != null && ipAddress instanceof Inet4Address) { 375. System.out.println("ip address: " + ipAddress.getHostAddress()); 376. } 377. } 378. } 380. } catch (SocketException e) { 381. e.printStackTrace(); 382. } 383. }//end method getAllIpAddress\*/ 384. } 385. //sSever.java 386. package SocketServe; 387. /\* 388. \* author: wbx 389. \*/ 390. import java.net.\*; 391. import java.io.\*; 392. import java.util.Scanner; 393. public class sServer { 394. public int PORT = 15648; 395. public boolean Log = true; 396. public ServerSocket ss = null; 397. public Socket[] users = new Socket[50]; 398. public int usersLen = 0; 399. public CallBack cb = null; 400. void wlog(String log) { 401. if(Log) System.out.println(log); 402. } 403. public boolean close() { 404. try { 405. ss.close(); 406. }catch(Exception e) { 407. e.printStackTrace(); 408. return false; 409. } 410. return true; 411. } 412. public boolean sendto(String Text,Socket sk) { 413. if(sk == null) return false; 414. PrintStream ps; 415. try { 416. ps = new PrintStream(sk.getOutputStream()); 417. ps.println(Text); 418. ps.flush(); 419. this.wlog("Send to "+sk.getPort()+" : "+Text); 420. } catch (IOException e) { 421. // TODO Auto-generated catch block 422. e.printStackTrace(); 423. return false; 424. } 426. return true; 427. } 428. public boolean send(String Text) { 429. try{ 430. for(int i=0;i<usersLen;i++) { 431. Socket sk = users[i]; 432. if(sk == null) continue; 433. PrintStream ps = new PrintStream(sk.getOutputStream()); 434. ps.println(Text); 435. ps.flush(); 436. this.wlog("Send "+(i+1)+"/"+usersLen+" to "+sk.getPort()+" : "+Text+"####"); 437. Thread.sleep(10); 438. } 439. } 440. catch(Exception e){ 441. e.printStackTrace(); 442. return false; 443. } 444. return true; 445. } 446. public boolean start(CallBack cb) //= "127.0.0.1") 447. { 448. this.cb = cb; 449. try { 450. ss = new ServerSocket(PORT); 451. ServerThreadAC st = new ServerThreadAC(); 452. st.start(); 453. } catch (IOException e) { 454. // TODO Auto-generated catch block 455. e.printStackTrace(); 456. return false; 457. } 458. return true; 459. } 460. //�������¿ͻ��߳��� 461. public class ServerThreadAC extends Thread 462. { 463. //String ServerIp; 464. int id = 0; 465. boolean Log = true; 466. //public ServerThreadAC(String ServerIp){ 467. // this.ServerIp = ServerIp; 468. //} 469. public void run() { 470. try { 471. if(Log) System.out.println("Server is open on "+PORT); 472. //��һ��whileѭ������ͬʱ��Ӧ����ͻ��˵����� 473. while(true){ 474. Socket sk = ss.accept();//������������Ӧ�˿ڵ����� 475. ServerThread st = new ServerThread(sk); 476. st.start(); 477. } 478. }catch(Exception e) { 479. e.printStackTrace(); 480. } 481. } 482. } 483. //�������ͻ��߳��� 484. public class ServerThread extends Thread 485. { 486. Socket sk; 487. int id = 0; 488. boolean Log = true; 489. public ServerThread(Socket sk){ 490. this.sk= sk; 491. } 492. public void run() { 493. try { 494. users[usersLen] = sk; 495. this.id = usersLen; 496. usersLen++; 497. if(Log) System.out.println("New User in "+sk.getPort()); 498. cb.run("New User: "+sk.getPort(),sk); 499. Scanner sn = new Scanner(sk.getInputStream()); 500. while(true) { 501. if(sn.hasNextLine()) { 502. String line = sn.nextLine();//br.readLine(); 503. //System.out.println("Server Receive from "+sk.getPort()+": "+line); 504. cb.run(line,sk); 505. }else { 506. try { 507. sk.sendUrgentData(0); 508. } catch (IOException e) { 509. System.out.println("User "+sk.getPort()+" has disconnected."); 510. cb.run("User Disconnect "+sk.getPort(),sk); 511. users[this.id] = null; 512. break; 513. } 514. } 515. } 516. } catch (IOException e1) { 517. e1.printStackTrace(); 518. } 519. } 520. } 521. } 522. //DataUser.java 523. package KillWolf.Data; 524. import java.net.Socket; 525. public class DataUser { 526. public Socket sk; 527. public int port; 528. public String name; 529. public boolean live; 530. public String identify; 531. public boolean hasConfirmID; //被预言家预言了，仅在预言家的客户端可以使用 532. public boolean havePoison = false; 533. public boolean haveAntidote = false; 534. } 535. //SelectCallBack.java 536. package KillWolf.Data; 537. public interface SelectCallBack { 538. public void run(String Selection); 539. } 540. // SelectionEvent.java 541. package KillWolf.Data; 542. import KillWolf.KillWolf; 543. import SocketServe.sServer; 544. public class SelectionEvent { 545. public static SelectionEvent Selection[] = new SelectionEvent[200]; 546. public static int SelectionLen = 0; 547. public static void UserQuit(String name) { 548. for(SelectionEvent i : Selection) { 549. if(i!=null) { 550. if(i.open) { 551. for(int j=0;j<i.openusers.length;j++) { 552. if(i.openusers[j].equals(name)) { 553. if(i.target[j] == null) { 554. i.target[j] = "\*\*NONE\*\*"; 555. i.isFinish(); 556. break; 557. } 558. } 559. } 560. } 561. } 562. } 563. } 564. public boolean open; 566. public String[] users; //要选的目标人 567. public int[] count; 569. public String[] openusers; //需要选人的人 570. public String[] target; 572. public String ReceiveID; 573. public int timeLimit; 574. public SelectionEventCallBack cb; 575. public sServer server; 576. public SelectionEvent(String ReceiveID,String[] users,int timeLimit,SelectionEventCallBack cb) { 577. this.users = new String[users.length]; 578. this.count = new int[users.length]; 579. this.cb = cb; 580. this.ReceiveID = ReceiveID; 581. this.timeLimit= timeLimit; 582. for(int i=0;i<users.length;i++) 583. if(users[i] != null) this.users[i] = users[i]; 584. else this.users[i] = "\*\*NONE\*\*"; 585. } 586. public void update(String user,String toUser) { 587. for(int i=0;i<users.length;i++) { 588. if(users[i].equals(user)) { 589. for( int j=0;j<openusers.length;j++) { 590. if(openusers[j].equals(user)) { 591. target[j] = toUser; 592. } 593. } 594. } 595. if(users[i].equals(toUser)) count[i]++; 596. } 597. System.out.println("Finish ? " + isFinish()); 598. } 599. public boolean isFinish() { 600. System.out.println("###### IS FINISH ? " + target.length); 601. for(String i : target) { 602. if(i == null) { 603. //System.out.println("#######################"); 604. return false; 605. } 606. } 607. closeSelection(this.server,false); 608. return true; 609. } 610. public int[] getResult() { 611. return count; 612. } 613. public boolean closeSelection(sServer server,boolean run) { 614. //if(run) if(this.cb != null) this.cb.run(ReceiveID, count, users, target); 615. System.out.println("CLOSESELECTION"); 616. if(this.open) { 617. this.open = false; 618. for(int i=0;i<openusers.length;i++){ 619. if(openusers[i]!=null){ 620. DataUser user = KillWolf.serverData.getUser(openusers[i]); 621. if(user!=null) 622. server.sendto("CLOSESELECTION"+ReceiveID,user.sk); 623. } 624. } 625. //没选的默认弃权 626. if(this.cb != null) this.cb.run(ReceiveID, count, users, target, openusers); 627. } 628. return true; 629. } 630. public void openSeletion(sServer server,DataUser du[],int RealLen, 631. String WindowTitle,String ButtonTitle,String Tip,boolean isDark) { 632. this.open = true; 633. this.server = server; 634. Selection[SelectionLen] = this; 635. SelectionLen++;   639. String STR = ""; 640. for(String i : users) { 641. STR = STR +"\t"+ i; 642. } 643. openusers = new String[RealLen]; 644. this.target = new String[RealLen]; 645. int j = 0; 646. for(DataUser i : du) { 647. if(i != null) { 648. openusers[j] = i.name; 649. server.sendto("OPENSELETION"+timeLimit+"\t\t"+ReceiveID+"\t"+ 650. STR+"\t\t"+WindowTitle+"\t\t"+Tip+"\t\t"+ButtonTitle+"\t\t"+ (isDark?"DARK":"LIGHT"),i.sk); 651. //RECEIVEID \t\t user1 \t user2 652. j++; 653. } 654. } 655. ThreadWait a = new ThreadWait(server); 656. a.start(); 657. } 658. public class ThreadWait extends Thread 659. { 660. sServer s; 661. public ThreadWait(sServer server) { 662. s = server; 663. } 664. public void run() { 665. try { 666. Thread.sleep(timeLimit\*1000); 667. closeSelection(s,false); 668. }catch(Exception e) { 669. e.printStackTrace(); 670. } 671. } 672. } 673. } 674. // SelectionEventCallBack.java 675. package KillWolf.Data; 676. public interface SelectionEventCallBack { 677. public void run(String ReceiveID,int[] count,String users[],String target[],String[] openUsers); 678. } 679. // ClientDeal.java 680. package KillWolf.SocketDeal; 681. import java.awt.Color; 682. import java.io.IOException; 683. import java.net.Socket; 684. import javax.swing.JFrame; 685. import javax.swing.JOptionPane; 686. import KillWolf.Data.SelectCallBack; 687. import KillWolf.KillWolf; 688. import KillWolf.Data.DataUser; 689. import KillWolf.Window.GameMain; 690. import KillWolf.Window.SelectWindow; 691. import SocketServe.CallBack; 692. public class ClientDeal implements CallBack { 693. //public SelectWindow SelectWindow; 694. public void run(String text,Socket sk){ 695. System.out.println(sk.getLocalPort()+" Client Receive: "+text+"####"); 696. if(text.startsWith("ENTERSUCCESS")) 697. { 698. KillWolf.UserName = text.substring(12); 699. KillWolf.loginwindow.LoginButton.setEnabled(false); 700. KillWolf.loginwindow.userName.setEnabled(false); 701. KillWolf.goWaitRoom(); 702. }else if(text.startsWith("ENTERFAIL")){ 703. JOptionPane.showMessageDialog(null, text.substring(9)); 704. /\*try { 705. sk.close(); 706. } catch (IOException e) { 707. // TODO Auto-generated catch block 708. e.printStackTrace(); 709. }\*/ 710. }else if(text.startsWith("NEWWAITUSERLIST")){ 711. String users[] = text.substring(15).split("\t"); 712. //KillWolf.WaitRoom.setList 713. for(int i=0;i<users.length;i++) { 714. KillWolf.Users[i]= new DataUser(); 715. KillWolf.Users[i].name = users[i]; 716. } 717. KillWolf.UsersRealLen = KillWolf.UsersLen = users.length; 718. //KillWolf.WaitRoom.UserList.setListData(users); 719. KillWolf.WaitRoom.setUserCount(users); 721. System.out.println(sk.getLocalPort()+" USERS: " + KillWolf.UserName+"\t" 722. +KillWolf.UsersRealLen); 723. }else if(text.startsWith("Disconnect from Server")){ 724. if(KillWolf.GameMain != null) { 725. KillWolf.GameMain.getFrame().setAlwaysOnTop(true); 726. JOptionPane.showMessageDialog(KillWolf.GameMain.getFrame(), 727. "服务器连接丢失"); 728. }else { 729. JOptionPane.showMessageDialog(null, "服务器连接丢失"); 730. } 731. //JOptionPane.setDefaultLocale(null); 732. System.exit(0); 733. }else if(text.startsWith("USERQUIT")){ 734. String user = text.substring(8),STR = ""; 735. for(int i=0;i<KillWolf.Users.length;i++) { 736. if(KillWolf.Users[i] != null) 737. if(KillWolf.Users[i].name.equals(user)) { 738. KillWolf.Users[i] = null; 739. }else { 740. STR = STR + "\t" +KillWolf.Users[i].name; 741. } 742. } 743. String users[] = STR.substring(1).split("\t"); 744. //KillWolf.WaitRoom.UserList.setListData(); 745. KillWolf.UsersRealLen--; 746. //KillWolf.WaitRoom.settitle("玩家数量："+ KillWolf.UsersRealLen)//+" 总数：" 747. +KillWolf.UsersLen); 748. KillWolf.WaitRoom.setUserCount(users); 749. if(KillWolf.GameMain == null) return; 750. int usdi=0,temp=1; 751. for(int i=0;i<KillWolf.GameMain.UserOutListData.length;i++){ 752. if(KillWolf.GameMain.UserOutListData[i].startsWith(user+" ")){ 753. temp=0; 754. KillWolf.GameMain.UserOutListData[i] = 755. KillWolf.GameMain.UserOutListData[i] + "(断线)"; 756. break; 757. }//else usod[i]=KillWolf.GameMain.UserOutListData[i]; 758. } 759. String usod[]=new String[KillWolf.GameMain.UserOutListData.length+temp]; 760. for(int i=0;i<KillWolf.GameMain.UserOutListData.length;i++){ 761. usod[i] = KillWolf.GameMain.UserOutListData[i]; 762. } 763. if(temp==1){ 764. String usda[]=new String[KillWolf.GameMain.UserListData.length-1], 765. usd[]=new String[KillWolf.GameMain.UserListData.length-1], 766. usdt[]=new String[KillWolf.GameMain.UserListData.length-1]; 767. usod[usod.length-1] = user+" (断线)"; 768. for(int i=0;i<KillWolf.GameMain.UserListData.length;i++) { 769. if(KillWolf.GameMain.UserListData[i].equals(user)) 770. { 771. usod[usod.length-1] = usod[usod.length-1] + KillWolf.GameMain.UserListTipData[i]; 772. }else{ 773. usd[usdi]=KillWolf.GameMain.UserListData[i]; 774. usdt[usdi]=KillWolf.GameMain.UserListTipData[i]; 775. usda[usdi]=usd[usdi]+usdt[usdi]; 776. usdi++; 777. } 778. } 779. KillWolf.GameMain.UserListData = usd; 780. KillWolf.GameMain.UserListTipData = usdt; 781. KillWolf.GameMain.getUserList().setListData(usda); 782. } 783. KillWolf.GameMain.UserOutListData = usod; 784. KillWolf.GameMain.getOutUserList().setListData(usod); 785. KillWolf.GameMain.addToScreen("【系统】用户 "+user+" 退出房间。"); 786. //KillWolf.UsersLen = KillWolf.Users.length; 787. }else if(text.startsWith("GAMESTART")) { 788. String[] STR = text.substring(9).split("#u#"); 789. DataUser[] du = new DataUser[STR.length]; 790. if(STR[0].equals("TRUE")){ 791. KillWolf.freeSpeak = true; 792. }else{ 793. KillWolf.freeSpeak = false; 794. } 795. for(int i=0;i<STR.length-1;i++) { 796. du[i] = new DataUser(); 797. String[] p = STR[i+1].split("\t"); 798. du[i].name = p[0]; 799. du[i].identify = p[1]; 800. } 801. KillWolf.Users = du; 802. KillWolf.goGameMain(); 803. }else if(text.startsWith("SELECT")) { 804. if(KillWolf.SelectWindow != null) KillWolf.SelectWindow.SELECTInfo(text.substring(6)); 805. }else if(text.startsWith("OPENSELETION")) { 806. String a[] = (text.substring(12).split("\t\t")) ; 807. JFrame fra = null; 808. if(KillWolf.GameMain!=null) fra = KillWolf.GameMain.getFrame(); 809. KillWolf.SelectWindow = new SelectWindow(a[3],true,a[1],true,fra); 810. if(a[6].equals("DARK"))KillWolf.SelectWindow.setColorMode(true); 811. String[] b = a[2].split("\t");//KillWolf.GameMain.UserListData; 812. String myId = KillWolf.GameMain.userSelf.identify; 813. String[] tip = new String[b.length]; 814. if(myId.equals("狼人")) {//|| myId.equals("预言家") 815. for (int j = 0; j < b.length; j++) { 816. for (int i = 0; i < KillWolf.UsersLen; i++) { 817. if (KillWolf.Users[i] != null) { 818. if (KillWolf.Users[i].name.equals(b[j])) { 819. if(myId.equals("狼人")){ 820. if(KillWolf.Users[i].identify.equals("狼人")){ 821. tip[j] = "("+KillWolf.Users[i].identify+")"; 822. } 823. }/\*else if(myId.equals("预言家" )&& KillWolf.Users[i].hasConfirmID){ 824. { 825. if(KillWolf.Users[i].identify.equals("狼人")){ 826. “)"; 827. } 828. } 829. }\*/else{ 830. tip[j] = ""; 831. } 832. } 833. } 834. } 835. } 836. } 837. KillWolf.SelectWindow.start(a[4], a[5], b, true,null, Integer.parseInt(a[0]),tip); 838. }else if(text.startsWith("CLOSESELECTION")) { 839. KillWolf.SelectWindow.getSelectionAndClose(text.substring(14)); 840. }else if(text.startsWith("MESSAGE")) { 841. if(text.substring(7).startsWith("【系统】天黑请闭眼")){ 842. KillWolf.GameMain.setDark(); 843. KillWolf.GameMain.getFrame().setTitle("狼人杀 "+text.substring(19)); 844. //天黑请闭眼 - Night 845. } 846. if(text.substring(7).startsWith("【系统】天亮请睁眼")){ 847. KillWolf.GameMain.getFrame().setTitle("狼人杀 "+text.substring(19)); 848. KillWolf.GameMain.setWhite(); 849. } 850. KillWolf.GameMain.addToScreen(text.substring(7)); 851. }else if(text.startsWith("YUYANJIASTART")){ 852. if(!KillWolf.GameMain.userSelf.identify.equals("预言家"))return; 853. KillWolf.SelectWindow = new SelectWindow("预言",false,"预言家",true,KillWolf.GameMain.getFrame()); 854. KillWolf.SelectWindow.setColorMode(true); 855. //String[] users=new String[] 856. SelectCallBack cb = Selection -> { 857. System.out.println("SelectFinal:" + Selection); 858. //System.out.println("预言："+st); 859. if(Selection.equals("\*\*NONE\*\*")) { 860. KillWolf.GameMain.addToScreen("【预言】你没有进行预言。"); 861. KillWolf.SelectWindow.getSelectionAndClose("预言家"); 862. KillWolf.client.send("YUYANEND"); 863. return; 864. } 865. String str = ""; 866. for(int i=0;i<KillWolf.UsersLen;i++){ 867. if(KillWolf.Users[i] == null) continue; 868. if(KillWolf.Users[i].name.equals(Selection)){ 869. for(int j=0;j<KillWolf.GameMain.UserListData.length;j++){ 870. System.out.println("###################List:" + KillWolf.GameMain.UserListData[j]+"#"); 871. if(KillWolf.GameMain.UserListData[j].equals(Selection)){ 872. str = (KillWolf.Users[i].identify.equals("狼人")?"坏人":"好人"); 873. KillWolf.GameMain.UserListTipData[j] = "(" + str + ")"; 874. str = "预言："+KillWolf.Users[i].name+"是"+str; 875. KillWolf.SelectWindow.CenterLabel.setText(str); 876. KillWolf.SelectWindow.CenterLabel.setForeground(Color.yellow); 877. break; 878. } 879. } 880. } 881. } 882. String [] usn = KillWolf.GameMain.UserListData; 883. String [] usnt = KillWolf.GameMain.UserListTipData; 884. String [] usna = new String[usn.length]; 885. for (int i=0;i<usn.length;i++) { 886. usna[i]=usn[i]+" "+usnt[i]; 887. } 888. KillWolf.GameMain.getUserList().setListData(usna); 889. KillWolf.SelectWindow.getSelectionAndClose("预言家"); 890. KillWolf.GameMain.addToScreen("【预言】"+str.substring(3)); 891. KillWolf.client.send("YUYANEND"); 892. //JOptionPane.showMessageDialog(KillWolf.GameMain.getFrame(),str); 894. }; 895. KillWolf.SelectWindow.start("请选择你要预言的人", "预言", KillWolf.GameMain.UserListData,true,cb,20,KillWolf.GameMain.UserListTipData); 896. SleepThread sth = new SleepThread("预言家"); 897. sth.start(); 899. //System.out.println("预言："+st); 900. //if(st.equals("\*\*NONE\*\*")) { 901. // KillWolf.GameMain.addToScreen("【预言】你没有进行预言。"); 902. //} 903. } 904. else if(text.startsWith("HUNTERSTART")){ 905. if(!KillWolf.GameMain.userSelf.identify.equals("猎人"))return; 906. int res=JOptionPane.showConfirmDialog(KillWolf.GameMain.getFrame(), "你已死亡，是否使用技能？", "消息", JOptionPane.YES\_NO\_OPTION); 907. if(res==JOptionPane.YES\_OPTION){ 908. if(text.substring(11).startsWith("LAST"))KillWolf.client.send("HUNTSTARTLASTYES"); 909. else KillWolf.client.send("HUNTSTARTYES"); 910. System.out.println("选择是后执行的代码"); //点击“是”后执行这个代码块 911. }else{ 912. if(text.substring(11).startsWith("LAST"))KillWolf.client.send("HUNTSTARTLASTYES"); 913. else KillWolf.client.send("HUNTSTARTNO"); 914. System.out.println("选择否后执行的代码"); //点击“否”后执行这个代码块 915. return; 916. } 917. }else if(text.startsWith("HUNTSELECT")){ 918. if(!KillWolf.GameMain.userSelf.identify.equals("猎人"))return; 919. KillWolf.SelectWindow = new SelectWindow("开枪",false,"猎人技能",true,KillWolf.GameMain.getFrame()); 920. KillWolf.SelectWindow.setColorMode(true); 921. SelectCallBack cb = Selection -> { 922. System.out.println("SelectFinal:" + Selection); 923. //System.out.println("预言："+st); 924. if(Selection.equals("\*\*NONE\*\*")) { 925. KillWolf.GameMain.addToScreen("【猎人】你没有开枪。"); 926. KillWolf.SelectWindow.getSelectionAndClose("猎人技能"); 927. if(text.substring(10).startsWith("LAST"))KillWolf.client.send("HUNTSELELASTNO"); 928. else KillWolf.client.send("HUNTSELENO"); 929. return; 930. } 931. KillWolf.GameMain.addToScreen("【猎人】你对"+Selection+"进行射击。"); 932. KillWolf.SelectWindow.getSelectionAndClose("猎人技能"); 933. if(text.substring(10).startsWith("LAST"))KillWolf.client.send("HUNTSELELASTYES"+KillWolf.UserName+"\t"+Selection); 934. else KillWolf.client.send("HUNTSELEYES"+KillWolf.UserName+"\t"+Selection); 935. }; 936. KillWolf.SelectWindow.start("你要对谁开枪？", "开枪", KillWolf.GameMain.UserListData,true,cb,20,null); 937. SleepThread sth = new SleepThread("猎人技能"); 938. sth.start(); 939. }else if(text.equals("FORBIDSPEAK")){ 940. KillWolf.GameMain.forbidSubmit(); 941. }else if(text.equals("ENABLESPEAK")){ 942. KillWolf.GameMain.ableSubmit(); 943. }else if(text.equals("ENABLESPEAKEND")){ 944. KillWolf.freeSpeak = true; 945. KillWolf.GameMain.ableSubmit(); 946. String usda[]=new String[KillWolf.GameMain.UserListData.length]; 947. for(int i=0;i<KillWolf.GameMain.UserListData.length;i++) { 948. for(int j=0;j<KillWolf.UsersLen;j++){ 949. if(KillWolf.Users[j]==null)continue; 950. if(KillWolf.Users[j].name.equals(KillWolf.GameMain.UserListData[i])){ 951. usda[i]=KillWolf.GameMain.UserListData[i]+" ("+KillWolf.Users[j].identify+")"; 952. } 953. } 954. } 955. KillWolf.GameMain.getUserList().setListData(usda); 956. }else if(text.equals("ENABLESPEAKAC")){ 957. KillWolf.GameMain.ableSubmit(); 958. KillWolf.GameMain.startSpeakClock(); 959. }else if(text.startsWith("WIZARDSTART")) { 960. if(!KillWolf.GameMain.userSelf.identify.equals("女巫"))return; 961. String name = text.substring(11); 962. if(KillWolf.GameMain.userSelf.haveAntidote) { 963. if(name.equals("\*\*NONE\*\*")) { 964. KillWolf.client.send("WIZARDANTINO"); 965. return; 966. } 967. int res=JOptionPane.showConfirmDialog(KillWolf.GameMain.getFrame(), "昨夜"+name+"死亡，是否使用解药？", "消息", JOptionPane.YES\_NO\_OPTION); 968. if(res==JOptionPane.YES\_OPTION){ 969. KillWolf.GameMain.userSelf.haveAntidote = false; 970. KillWolf.client.send("WIZARDANTIYES"); 971. System.out.println("选择是后执行的代码"); //点击“是”后执行这个代码块 972. }else{ 973. KillWolf.client.send("WIZARDANTINO"); 974. System.out.println("选择否后执行的代码"); //点击“否”后执行这个代码块 975. return; 976. } 977. }else { 978. KillWolf.client.send("WIZARDANTINONE"); 979. } 980. }else if(text.startsWith("WIZARDPOS")) { 981. if(!KillWolf.GameMain.userSelf.identify.equals("女巫"))return; 982. KillWolf.SelectWindow = new SelectWindow("毒药",false,"女巫毒药",true,KillWolf.GameMain.getFrame()); 983. KillWolf.SelectWindow.setColorMode(true); 984. //String[] users=new String[] 985. SelectCallBack cb = Selection -> { 986. System.out.println("SelectFinal:" + Selection); 987. //System.out.println("预言："+st); 988. if(Selection.equals("\*\*NONE\*\*")) { 989. KillWolf.GameMain.addToScreen("【女巫】你没有使用毒药。"); 990. KillWolf.SelectWindow.getSelectionAndClose("女巫毒药"); 991. KillWolf.client.send("WIZARDPOSNO"); 992. return; 993. } 994. KillWolf.GameMain.addToScreen("【女巫】你对"+Selection+"使用了毒药。"); 995. KillWolf.SelectWindow.getSelectionAndClose("女巫毒药"); 996. KillWolf.client.send("WIZARDPOSYES"+Selection); 997. //JOptionPane.showMessageDialog(KillWolf.GameMain.getFrame(),str); 998. }; 999. KillWolf.SelectWindow.start("你要对谁使用毒药？", "使用", KillWolf.GameMain.UserListData,true,cb,20,KillWolf.GameMain.UserListTipData); 1000. SleepThread sth = new SleepThread("女巫毒药"); 1001. sth.start(); 1002. }else if(text.startsWith("USERDIED")){ 1003. String user = text.substring(8),STR = ""; 1004. /\*for(int i=0;i<KillWolf.Users.length;i++) { 1005. if(KillWolf.Users[i] != null) 1006. if(KillWolf.Users[i].name.equals(user)) { 1007. KillWolf.Users[i].live = false; 1008. }else { 1009. //STR = STR + "\t" +KillWolf.Users[i].name; 1010. } 1011. } 1012. String users[] = STR.substring(1).split("\t"); 1013. \*///KillWolf.WaitRoom.UserList.setListData(); 1014. //KillWolf.UsersRealLen--; 1015. //KillWolf.WaitRoom.settitle("玩家数量："+ KillWolf.UsersRealLen)//+" 总数："+KillWolf.UsersLen); 1016. //KillWolf.WaitRoom.setUserCount(users); 1017. //if(KillWolf.GameMain == null) return; 1018. int pp=0; 1019. for(;pp<KillWolf.UsersLen;pp++){ 1020. if(KillWolf.Users[pp]==null)continue; 1021. if(KillWolf.Users[pp].name.equals(user)){ 1022. pp=-1; 1023. break; 1024. } 1025. } 1026. if(pp!=-1) return; 1027. String usda[]=new String[KillWolf.GameMain.UserListData.length-1], 1028. usd[]=new String[KillWolf.GameMain.UserListData.length-1], 1029. usdt[]=new String[KillWolf.GameMain.UserListData.length-1], 1030. usod[]=new String[KillWolf.GameMain.UserOutListData.length+1]; 1031. int usdi=0; 1032. for(int i=0;i<KillWolf.GameMain.UserOutListData.length;i++){ 1033. usod[i]=KillWolf.GameMain.UserOutListData[i]; 1034. } 1035. usod[usod.length-1] = user+" (淘汰)"; 1036. if(user.equals(KillWolf.UserName)){ 1037. KillWolf.GameMain.getIdentity().setText(KillWolf.UserName+"("+KillWolf.GameMain.userSelf.identify+")，你已被淘汰"); 1038. } 1039. for(int i=0;i<KillWolf.GameMain.UserListData.length;i++) { 1040. if(KillWolf.GameMain.UserListData[i].equals(user)) 1041. { 1042. usod[usod.length-1] = usod[usod.length-1] + KillWolf.GameMain.UserListTipData[i]; 1043. }else{ 1044. usd[usdi]=KillWolf.GameMain.UserListData[i]; 1045. usdt[usdi]=KillWolf.GameMain.UserListTipData[i]; 1046. usda[usdi]=usd[usdi]+usdt[usdi]; 1047. usdi++; 1048. } 1049. } 1050. KillWolf.GameMain.UserListData = usd; 1051. KillWolf.GameMain.UserListTipData = usdt; 1052. KillWolf.GameMain.UserOutListData = usod; 1053. KillWolf.GameMain.getUserList().setListData(usda); 1054. KillWolf.GameMain.getOutUserList().setListData(usod); 1055. // System.out.println(text); 1056. // String[] result = text.substring(8).split("\t"); 1057. // String[] users = new String[KillWolf.GameMain.UserListData.length- result.length]; 1058. // String[] outUsers = new String[KillWolf.GameMain.UserOutListData.length+result.length]; 1059. // for(int j=0;j< result.length;j++){ 1060. // if(result[j].equals(KillWolf.GameMain.userSelf.name))KillWolf.GameMain.userSelf.live=false; 1061. // for(int k=0;k< KillWolf.GameMain.UserListData.length;k++){ 1062. // if(KillWolf.GameMain.UserListData[k].startsWith(result[j])){ 1063. // result[j]=KillWolf.GameMain.UserListData[k]; 1064. // KillWolf.GameMain.UserListData[k]=null; 1065. // } 1066. // } 1067. // } 1068. // int i=0; 1069. // for (;i<KillWolf.GameMain.UserOutListData.length;i++){ 1070. // outUsers[i] = KillWolf.GameMain.UserOutListData[i]; 1071. // } 1072. // int j=i; 1073. // for(;i<outUsers.length;i++){ 1074. // outUsers[i]=result[i-j]+"(淘汰)"; 1075. // } 1076. // KillWolf.GameMain.UserOutListData=outUsers; 1077. // KillWolf.GameMain.getOutUserList().setListData(outUsers); 1078. // int i2 = 0; 1079. // for(int i1 = 0;i1< KillWolf.GameMain.UserListData.length;i1++){ 1080. // if(KillWolf.GameMain.UserListData[i1]!=null&& !KillWolf.GameMain.UserListData[i1].equals("\*\*NONE\*\*")){ 1081. // users[i2++]=KillWolf.GameMain.UserListData[i1]; 1082. // } 1083. // } 1084. // KillWolf.GameMain.UserListData = users; 1085. // KillWolf.GameMain.getUserList().setListData(users); 1086. } 1087. // +timeLimit+"\t\t"+ReceiveID+"\t"+STR,du[j].sk); //RECEIVEID \t\t user1 \t user2 1089. } 1090. } 1091. class SleepThread extends Thread{ 1092. String sss; 1093. public SleepThread(String a) { 1094. this.sss = a; 1095. } 1096. public void run() { 1097. try { 1098. Thread.sleep(20\*1000); 1099. } catch (InterruptedException e) { 1100. e.printStackTrace(); 1101. } 1102. KillWolf.SelectWindow.getSelectionAndClose(sss); 1103. } 1104. } 1105. // GameMainThread.java 1106. package KillWolf.SocketDeal; 1107. import KillWolf.KillWolf; 1108. import KillWolf.Data.DataUser; 1109. import KillWolf.Data.SelectionEvent; 1110. import KillWolf.Window.GameMain; 1111. public class GameMainThread extends Thread{ 1112. public boolean YuYanCompleted = false; 1113. public boolean WizardCompleted = false; 1114. public void run() { 1115. KillWolf.serverData.deathUser = "\*\*NONE\*\*"; 1116. KillWolf.serverData.deathUser2 = "\*\*NONE\*\*"; 1117. KillWolf.serverData.deathUser3 = "\*\*NONE\*\*"; 1118. KillWolf.serverData.deathUser4 = "\*\*NONE\*\*"; 1119. gameNight(); 1120. } 1121. public void gameNight(){ 1122. if(KillWolf.serverData.GameMainThread.isWin()) return; 1123. KillWolf.serverData.Day++; 1124. KillWolf.serverData.night = true; 1125. KillWolf.server.send("MESSAGE【系统】天黑请闭眼 - Night"+KillWolf.serverData.Day); 1126. KillWolf.server.send("FORBIDSPEAK"); 1127. //KillWolf.server.send("ENABLESPEAK"); 1128. String[] allUserName = (String[])KillWolf.serverData.getLivingUsers(true); 1129. //给狼人弹窗 1130. DataUser[] wolfUser0 = KillWolf.serverData.getUsersWithID("狼人"); 1131. DataUser[] wolfUser = new DataUser[wolfUser0.length]; 1132. int wi = 0; 1133. for(int k=0;k<wolfUser0.length;k++) { 1134. if(wolfUser0[k].live) { 1135. wolfUser[wi] = wolfUser0[k]; 1136. wi++; 1137. } 1138. } 1139. //String[] wolfUserName = KillWolf.serverData.ArrayDatatoName(wolfUser); 1140. try { 1141. Thread.sleep(2\*1000); 1142. } catch (InterruptedException e) { 1143. e.printStackTrace(); 1144. } 1145. KillWolf.server.send("MESSAGE【系统】狼人请睁眼，请选择你要暗杀的目标"); 1146. KillWolf.serverData.Selection[KillWolf.serverData.SelectionLen] = new SelectionEvent(Integer.toString(KillWolf.serverData.SelectionLen),allUserName,30,new WolfSelect()); 1147. KillWolf.serverData.Selection[KillWolf.serverData.SelectionLen].openSeletion(KillWolf.server, wolfUser, wi,"狼人出没","暗杀他","请选择你要暗杀的人",true); 1148. KillWolf.serverData.SelectionLen++; 1149. //预言家弹窗 1150. KillWolf.server.send("MESSAGE【系统】预言家请睁眼，请选择你要预言的目标"); 1151. DataUser d[] = KillWolf.serverData.getUsersWithID("预言家"); 1152. if (d.length == 1 && d[0].live) { 1153. YuYanCompleted = false; 1154. KillWolf.server.sendto("YUYANJIASTART",d[0].sk); 1155. }else YuYanCompleted = true; 1157. //女巫弹窗在狼人结束才开始，此处先置为false 1158. WizardCompleted = false; 1159. //等待预言、女巫完成再进入白天，进入白天自动进行判断是否能进入，在女巫结束还会调用一次。 1160. try { 1161. Thread.sleep(20\*1000); 1162. YuYanCompleted = true;//等20秒，预言一定已经完成了 1163. gameDay(); 1164. } catch (InterruptedException e) { 1165. // TODO Auto-generated catch block 1166. e.printStackTrace(); 1167. } 1168. } 1170. public void gameWizard(String diedUser) { 1171. DataUser d[] = KillWolf.serverData.getUsersWithID("女巫"); 1172. //System.out.println(null.length); 1173. if (d.length == 1 && d[0].live) { 1174. WizardCompleted = false; 1175. KillWolf.server.sendto("WIZARDSTART"+diedUser,d[0].sk); 1176. }else { 1177. if(!diedUser.equals("\*\*NONE\*\*")) { 1178. KillWolf.serverData.getUser(diedUser).live = false; 1179. } 1180. WizardCompleted = true; 1181. KillWolf.serverData.GameMainThread.gameDay(); 1182. } 1183. } 1184. boolean isRunGameDay = false; 1185. public void gameDay() { 1186. if(isRunGameDay) return; 1187. if(!YuYanCompleted || !WizardCompleted) return; 1188. //如果都完成了则继续 1189. String death = KillWolf.serverData.deathUser; 1190. String death2 = KillWolf.serverData.deathUser2; 1191. isRunGameDay = true; 1192. KillWolf.server.send("MESSAGE【系统】天亮请睁眼 - Day"+KillWolf.serverData.Day); 1193. try { 1194. Thread.sleep(1000); 1195. } catch (InterruptedException e) { 1196. e.printStackTrace(); 1197. } 1198. String death3 = KillWolf.serverData.deathUser3; 1199. if(death2.equals("\*\*NONE\*\*") && death.equals("\*\*NONE\*\*")) { 1200. KillWolf.server.send("MESSAGE【系统】昨夜无人死亡"); 1201. }else { 1202. if(!death.equals("\*\*NONE\*\*")){ 1203. DataUser u = KillWolf.serverData.getUser(death); 1204. if(u!=null) u.live =false; 1205. KillWolf.server.send("USERDIED"+death); 1206. } 1207. if(!death2.equals("\*\*NONE\*\*")){ 1208. DataUser u = KillWolf.serverData.getUser(death2); 1209. if(u!=null) u.live =false; 1210. KillWolf.server.send("USERDIED"+death2); 1211. } 1212. if(!death3.equals("\*\*NONE\*\*")){ 1213. DataUser u = KillWolf.serverData.getUser(death3); 1214. if(u!=null) u.live =false; 1215. KillWolf.server.send("USERDIED"+death3); 1216. } 1217. //KillWolf.server.send("USERDIED"+(death.equals("\*\*NONE\*\*")?"":death)+(death2.equals("\*\*NONE\*\*")?"":("\t"+death2))+(death3.equals("\*\*NONE\*\*")?"":("\t"+death3))); 1218. if(death2.equals("\*\*NONE\*\*"))KillWolf.server.send("MESSAGE【系统】昨夜"+(death.equals("\*\*NONE\*\*")?"":death)+(death3.equals("\*\*NONE\*\*")?"":("、"+death3))+"死亡"); 1219. else KillWolf.server.send("MESSAGE【系统】昨夜"+(death.equals("\*\*NONE\*\*")?"":death) +(death2.equals("\*\*NONE\*\*")?"":("、"+death2))+(death3.equals("\*\*NONE\*\*")?"":("、"+death3))+"死亡"); 1220. } 1221. if(KillWolf.serverData.GameMainThread.isWin())return; 1222. //判断猎人是否存活，若死亡，向猎人弹窗，使用技能 1223. DataUser user = KillWolf.serverData.getUser(death); 1224. DataUser user2 = KillWolf.serverData.getUser(death2); 1225. if(user == null && user2 == null) dayWork(); 1226. else if(user != null && user.identify.equals("猎人")){KillWolf.server.sendto("HUNTERSTART",user.sk);} 1227. else if(user2 != null && user2.identify.equals("猎人")){KillWolf.server.sendto("HUNTERSTART",user2.sk);} 1228. else dayWork(); 1229. //向所有玩家弹窗，投票 1230. //向公屏输出，投票结果，若平票，则再次投票 1231. //再次判断猎人是否存活，若死亡，向猎人弹窗 1232. // } 1233. //KillWolf.server.send("MESSAGE【系统】"+whoWin+"胜，将在5秒后退出该窗口"); 1234. //重置游戏 1235. } 1236. class TimeTha extends Thread{ 1237. public void run() { 1238. try { 1239. Thread.sleep(1000\*120); 1240. gameVote(); 1241. }catch(Exception e) { 1242. e.printStackTrace(); 1243. } 1244. } 1245. } 1246. public void dayWork (){ 1247. String death = KillWolf.serverData.deathUser; 1248. String death2 = KillWolf.serverData.deathUser2; 1249. //String death3 = KillWolf.serverData.deathUser3; 1250. String firstdeath = ""; 1251. //}else if(death2.equals("\*\*NONE\*\*") && !death.equals("\*\*NONE\*\*")){ 1252. // //KillWolf.server.send("MESSAGE【系统】昨夜"+death+"死亡"); 1253. // firstdeath = death; 1254. // }else if(death2.equals("\*\*NONE\*\*") && death.equals("\*\*NONE\*\*")) { 1255. // //KillWolf.server.send("MESSAGE【系统】昨夜无人死亡"); 1256. // } 1257. // else if(!death2.equals("\*\*NONE\*\*") && !death.equals("\*\*NONE\*\*")){ 1258. // firstdeath = death; 1259. // //KillWolf.server.send("MESSAGE【系统】昨夜"+death+"、"+death2+"死亡"); 1260. // } 1261. if(!death.equals("\*\*NONE\*\*") ){ 1262. firstdeath = death; 1263. }else if(!death2.equals("\*\*NONE\*\*") ){ 1264. firstdeath = death2; 1265. } 1266. if(KillWolf.serverData.GameMainThread.isWin())return; 1267. //判断是否获胜 1268. //玩家轮流发言 1269. if(KillWolf.serverData.freeSpeak){ 1270. KillWolf.server.send("MESSAGE【系统】现在有120秒的交流时间"); 1271. KillWolf.server.send("ENABLESPEAKAC"); 1272. TimeTha spt = new TimeTha(); 1273. spt.start(); 1274. return; 1275. } 1276. KillWolf.server.send("MESSAGE【系统】进入轮流发言阶段，每人有30秒发言时间"); 1277. try { 1278. Thread.sleep(2000); 1279. } catch (InterruptedException e) { 1280. e.printStackTrace(); 1281. } 1282. DataUser user = null;//(DataUser[]) KillWolf.serverData.getUser(firstdeath); 1283. if(firstdeath.equals("")) for(int j=0;j<KillWolf.serverData.UsersLen;j++){ 1284. if(KillWolf.serverData.ServerUsers[j] != null) 1285. if(KillWolf.serverData.ServerUsers[j].live) firstdeath = KillWolf.serverData.ServerUsers[j].name; 1286. } 1287. for(int i = 0;i<KillWolf.serverData.UsersLen;i++){ 1288. if(KillWolf.serverData.ServerUsers[i]==null) continue; 1289. if(firstdeath.equals(KillWolf.serverData.ServerUsers[i].name)){ 1290. int j=i+1; 1291. for(;j<KillWolf.serverData.UsersLen;j++){ 1292. if(KillWolf.serverData.ServerUsers[j] != null) 1293. if(KillWolf.serverData.ServerUsers[j].live) user = KillWolf.serverData.ServerUsers[j]; 1294. } 1295. if(user == null) for(j=0;j<i;j++){ 1296. if(KillWolf.serverData.ServerUsers[j] != null) 1297. if(KillWolf.serverData.ServerUsers[j].live) user = KillWolf.serverData.ServerUsers[j]; 1298. } 1299. if(user == null) { 1300. KillWolf.serverData.GameMainThread.gameNext(); 1301. System.out.println("ERROR !!!!!!!!!!!!!"); 1302. return; 1303. } 1304. KillWolf.serverData.StartSpeakName = user.name; 1305. KillWolf.serverData.nowSpeakUser = user.name; 1306. KillWolf.server.send("MESSAGE【系统】轮到"+user.name+"发言"); 1307. KillWolf.server.sendto("ENABLESPEAKAC",user.sk); 1308. SpeakThread spt = new SpeakThread(user.name,KillWolf.serverData.Day); 1309. spt.start(); 1310. break; 1311. } 1312. //if(i==0) 1313. } 1314. } 1315. public void gameVote() { 1316. KillWolf.server.send("MESSAGE【系统】发言结束，现在开始进行投票"); 1317. //投票弹窗 1318. String[] allUserName = (String[])KillWolf.serverData.getLivingUsers(true); 1319. DataUser[] allUser = (DataUser[]) KillWolf.serverData.getLivingUsers(false); 1320. //String[] wolfUserName = KillWolf.serverData.ArrayDatatoName(wolfUser); 1321. KillWolf.serverData.Selection[KillWolf.serverData.SelectionLen] = new SelectionEvent(Integer.toString(KillWolf.serverData.SelectionLen),allUserName,20,new VoteSelection()); 1322. KillWolf.serverData.Selection[KillWolf.serverData.SelectionLen].openSeletion(KillWolf.server, allUser, allUser.length,"开始投票","投他","请选择",false); 1323. KillWolf.serverData.SelectionLen++; 1324. } 1325. public void gameHunt(String death) { 1326. DataUser user = KillWolf.serverData.getUser(death); 1327. if (user != null && user.identify.equals("猎人")) { 1328. KillWolf.server.sendto("HUNTERSTARTLAST", user.sk); 1329. } 1330. } 1331. public void gameNext(){ 1332. if(!isWin()){ 1333. (KillWolf.serverData.GameMainThread = new GameMainThread()).start(); 1334. } 1335. } 1336. public boolean isWin(){ 1337. String whoWin = KillWolf.serverData.whoWin(); 1338. if(!(whoWin.equals("无"))){ 1339. KillWolf.server.send("MESSAGE【系统】游戏结束，"+whoWin+"胜"); 1340. gameEnd(); 1341. return true; 1342. //KillWolf.server.send("MESSAGE【系统】将在5秒后关闭窗口"); 1343. } 1344. return false; 1345. } 1346. public void gameEnd(){ 1347. KillWolf.server.send("MESSAGE【系统】游戏统计"); 1348. DataUser[] d = KillWolf.serverData.ServerUsers; 1349. for(int i=0;i<KillWolf.serverData.UsersLen;i++){ 1350. if(d[i]!=null){ 1351. KillWolf.server.send("MESSAGE【系统】"+d[i].name+" "+d[i].identify+" "+(d[i].live?"存活":"淘汰")); 1352. } 1353. } 1354. KillWolf.serverData.freeSpeak = true; 1355. KillWolf.server.send("ENABLESPEAKEND"); 1356. } 1357. } 1358. // ServerDeal.java 1359. package KillWolf.SocketDeal; 1360. import java.net.Socket; 1361. import java.security.PublicKey; 1362. import KillWolf.Data.SelectionEventCallBack; 1363. import KillWolf.KillWolf; 1364. import KillWolf.Data.DataUser; 1365. import KillWolf.Data.SelectionEvent; 1366. import KillWolf.KillWolf.serverData; 1367. import SocketServe.CallBack; 1368. import com.sun.source.tree.ReturnTree; 1369. public class ServerDeal implements CallBack { 1370. public void run(String text,Socket sk){ 1371. System.out.println(sk.getPort()+" Server Receive: "+text+"####"); 1372. if(text.startsWith("ENTER")){ 1373. if(KillWolf.GameMain!=null) { 1374. KillWolf.server.sendto("ENTERFAIL游戏已经开始，请稍后再试",sk); 1375. return; 1376. } 1377. String name = text.substring(5); 1378. for(int i=0;i<KillWolf.serverData.UsersLen;i++) { 1379. if(KillWolf.serverData.ServerUsers[i]!=null) 1380. if(name.equals(KillWolf.serverData.ServerUsers[i].name)) { 1381. //System.out.println("NAME: "+name+" "+i); 1382. KillWolf.server.sendto("ENTERFAIL当前房间存在重名，请更换用户名后再尝试连接。",sk); 1383. return; 1384. } 1385. } 1386. KillWolf.server.sendto("ENTERSUCCESS["+(KillWolf.serverData.UsersLen+1)+"]"+name,sk); 1387. KillWolf.serverData.ServerUsers[KillWolf.serverData.UsersLen] = new DataUser(); 1388. KillWolf.serverData.ServerUsers[KillWolf.serverData.UsersLen].name="["+(KillWolf.serverData.UsersLen+1)+"]"+name; 1389. KillWolf.serverData.ServerUsers[KillWolf.serverData.UsersLen].sk=sk; 1390. KillWolf.serverData.ServerUsers[KillWolf.serverData.UsersLen].port=sk.getPort(); 1391. KillWolf.serverData.UsersLen++; 1392. KillWolf.serverData.UsersRealLen++; 1393. String STR = ""; 1394. for(int i=0;i<KillWolf.serverData.UsersLen;i++) { 1395. if(KillWolf.serverData.ServerUsers[i]!=null) 1396. STR = STR +'\t'+ KillWolf.serverData.ServerUsers[i].name; 1397. } 1398. KillWolf.server.send("NEWWAITUSERLIST"+STR.substring(1)); 1399. return; 1400. } 1401. else if(text.equals("GETUSERS")) { 1402. String STR = ""; 1403. for(DataUser i : KillWolf.serverData.ServerUsers) { 1404. if(i!=null) 1405. STR = STR +'\t'+ i.name; 1406. } 1407. KillWolf.server.send(STR.substring(1)); 1408. return; 1409. }else if(text.startsWith("User Disconnect ")) { 1410. int port = Integer.parseInt(text.substring(16)); 1411. for(int i=0;i<KillWolf.serverData.UsersLen;i++) { 1412. if(KillWolf.serverData.ServerUsers[i]!=null) 1413. if(port == KillWolf.serverData.ServerUsers[i].port) { 1414. System.out.println("NAME QUIT: "+KillWolf.serverData.ServerUsers[i].name); 1415. KillWolf.server.send("USERQUIT"+KillWolf.serverData.ServerUsers[i].name);//KillWolf.server.sendto("ENTERFAIL",sk); 1416. SelectionEvent.UserQuit(KillWolf.serverData.ServerUsers[i].name); 1417. KillWolf.serverData.ServerUsers[i] = null; 1418. KillWolf.serverData.UsersRealLen--; 1419. //添加选择时用户退出 1420. return; 1421. } 1422. }  1425. }else if(text.startsWith("SELECT")) { 1426. KillWolf.server.send(text); 1427. }else if(text.startsWith("FINALSELECT")) { 1428. String s[] = text.substring(11).split("\t"); 1429. SelectionEvent se = KillWolf.serverData.Selection[Integer.parseInt(s[0])]; 1430. se.update(s[1],s[2]); 1431. }else if(text.startsWith("GAMEREADY")){ 1432. serverData.getUser(text.substring(9)).live = true; 1433. for(int i=0;i<serverData.UsersLen;i++){ 1434. if(serverData.ServerUsers[i]!=null) 1435. if(!serverData.ServerUsers[i].live) return; 1436. } 1437. KillWolf.server.send("MESSAGE【系统】玩家全部进入，游戏将于3秒后开始"); 1438. try { 1439. Thread.sleep(3000); 1440. } catch (InterruptedException e) { 1441. e.printStackTrace(); 1442. } 1443. KillWolf.serverData.Day = 0; 1444. KillWolf.serverData.night = false; 1445. (KillWolf.serverData.GameMainThread = new GameMainThread()).start(); 1446. }else if(text.startsWith("USERSPEAK")){ 1447. String[] submitMsg = text.substring(9).split("\t"); 1448. KillWolf.server.send("MESSAGE【玩家】"+submitMsg[0]+" ："+submitMsg[1]); 1449. if(!KillWolf.serverData.freeSpeak) nextSpeak(submitMsg[0],serverData.Day); 1450. }else if(text.startsWith("YUYANEND")){ 1451. KillWolf.serverData.GameMainThread.YuYanCompleted = true; 1452. KillWolf.serverData.GameMainThread.gameDay(); 1453. }else if(text.startsWith("WIZARDANTI")) { 1454. String result = text.substring(10); 1455. if(result.equals("YES")) { 1456. DataUser u = KillWolf.serverData.getUser(KillWolf.serverData.deathUser); 1457. if(u!=null) u.live = true; 1458. KillWolf.server.sendto("MESSAGE【女巫】你为"+KillWolf.serverData.deathUser+"使用了解药。", sk); 1459. KillWolf.serverData.deathUser = "\*\*NONE\*\*"; 1460. KillWolf.serverData.GameMainThread.WizardCompleted = true; 1461. KillWolf.serverData.GameMainThread.gameDay(); 1462. }else if(result.equals("NO")){ 1463. if(!KillWolf.serverData.deathUser.equals("\*\*NONE\*\*")) { 1464. DataUser u = KillWolf.serverData.getUser(KillWolf.serverData.deathUser); 1465. if(u!=null) u.live = false; 1466. KillWolf.server.sendto("MESSAGE【女巫】昨夜"+KillWolf.serverData.deathUser+"死亡，你没有使用解药。", sk); 1467. } 1468. else KillWolf.server.sendto("MESSAGE【女巫】昨夜无人死亡。", sk); 1469. KillWolf.server.sendto("WIZARDPOS", sk); 1470. }else if(result.equals("NONE")){ 1471. if(!KillWolf.serverData.deathUser.equals("\*\*NONE\*\*")){ 1472. DataUser u = KillWolf.serverData.getUser(KillWolf.serverData.deathUser); 1473. if(u!=null) u.live = false; 1474. } 1475. KillWolf.server.sendto("WIZARDPOS", sk); 1476. } 1477. }else if(text.startsWith("WIZARDPOS")) { 1478. String result = text.substring(9); 1479. KillWolf.serverData.deathUser2 = "\*\*NONE\*\*"; 1480. if(result.startsWith("YES")) { 1481. DataUser u = KillWolf.serverData.getUser(result.substring(3)); 1482. if(u!=null) u.live = false; 1483. KillWolf.serverData.deathUser2 = result.substring(3); 1484. }else if(result.equals("NO")){ 1486. }else if(result.equals("NONE")){ 1487. } 1488. KillWolf.serverData.GameMainThread.WizardCompleted = true; 1489. KillWolf.serverData.GameMainThread.gameDay(); 1490. } 1491. else if(text.startsWith("HUNTSTART")){ 1492. String result = text.substring(9); 1493. if(result.equals("YES")){ 1494. KillWolf.server.sendto("HUNTSELECT", sk); 1495. }else if(result.equals("NO")){ 1496. KillWolf.server.sendto("MESSAGE【猎人】你已死亡，没有使用技能。",sk); 1497. KillWolf.serverData.GameMainThread.dayWork(); 1498. }else if(result.startsWith("LAST")){ 1499. if(result.substring(4).equals("YES")){ 1500. KillWolf.server.sendto("HUNTSELECTLAST", sk); 1501. }else if(result.substring(4).equals("NO")) { 1502. KillWolf.server.sendto("MESSAGE【猎人】你已死亡，没有使用技能。", sk); 1503. serverData.GameMainThread.gameNext(); 1504. } //KillWolf.serverData.GameMainThread.dayWork(); 1505. } 1506. }else if(text.startsWith("HUNTSELE")){ 1507. //System.out.println("猎人技能成功#########"); 1508. String result = text.substring(8); 1509. if(result.startsWith("YES")) { 1510. String[] u = result.substring(3).split("\t"); 1511. DataUser us = KillWolf.serverData.getUser(u[1]); 1512. if(us!=null) us.live = false; 1513. KillWolf.serverData.deathUser3 = u[1]; 1514. System.out.println("猎人技能成功#########"); 1515. KillWolf.server.send("MESSAGE【猎人】"+u[0]+"死前开枪杀死了"+ serverData.deathUser3); 1516. KillWolf.server.send("USERDIED"+serverData.deathUser3); 1517. KillWolf.serverData.GameMainThread.dayWork(); 1518. } 1519. else if(result.equals("NO")){ 1520. KillWolf.serverData.GameMainThread.dayWork(); 1521. }else if (result.startsWith("LAST")){ 1522. result = result.substring(4); 1523. if(result.startsWith("YES")) { 1524. String[] u = result.substring(3).split("\t"); 1525. DataUser us = KillWolf.serverData.getUser(u[1]); 1526. if(us!=null) us.live = false; 1527. //KillWolf.serverData.getUser(u[1]).live = false; 1528. KillWolf.serverData.deathUser3 = u[1]; 1529. System.out.println("猎人技能成功#########"); 1530. KillWolf.server.send("MESSAGE【猎人】"+u[0]+"死前开枪杀死了"+ serverData.deathUser3); 1531. KillWolf.server.send("USERDIED"+serverData.deathUser3); 1532. //KillWolf.serverData.GameMainThread.dayWork(); 1533. try { 1534. Thread.sleep(2000); 1535. } catch (InterruptedException e) { 1536. e.printStackTrace(); 1537. } 1538. } 1539. else if(result.equals("NO")) { 1540. } 1541. serverData.GameMainThread.gameNext(); 1542. } 1543. } 1544. //KillWolf.server.send("["+sk.getPort()+"]"+text); 1545. } 1546. public static boolean nextSpeak(String str,int Day){ 1547. System.out.println("NOW SPEAK:"+KillWolf.serverData.nowSpeakUser+" CALLBACK:"+str); 1548. if(!KillWolf.serverData.nowSpeakUser.equals(str) || Day != serverData.Day) return false; 1549. else { 1550. DataUser[] liveUsers = (DataUser[]) KillWolf.serverData.getLivingUsers(false); 1551. String lastSpeak = str; 1552. if(serverData.StartSpeakName.equals("\*\*VOTE\*\*")) return true; 1553. for(int i = 0;i<liveUsers.length;i++){ 1554. if(lastSpeak.equals(liveUsers[i].name)){ 1555. KillWolf.server.sendto("FORBIDSPEAK",liveUsers[i].sk); 1557. if(i==liveUsers.length-1) i=-1; 1558. i=i+1; 1560. System.out.println("CHANGE SPEAK "+KillWolf.serverData.nowSpeakUser+" TO "+liveUsers[i].name); 1561. if(liveUsers[i].name.equals(serverData.StartSpeakName)){ 1562. //一轮发言结束 1563. serverData.StartSpeakName = "\*\*VOTE\*\*"; 1564. KillWolf.serverData.GameMainThread.gameVote(); 1565. return true; 1566. } 1567. KillWolf.serverData.nowSpeakUser = liveUsers[i].name; 1568. KillWolf.server.send("MESSAGE【系统】轮到"+liveUsers[i].name+"发言"); 1569. KillWolf.server.sendto("ENABLESPEAKAC",liveUsers[i].sk); 1570. SpeakThread spt = new SpeakThread(liveUsers[i].name, serverData.Day); 1571. spt.start(); 1572. return true; 1573. } 1574. //if(i==0) 1575. } 1576. //找遍了所有人，但还没找到上一个发言人，可能已经退出，从第一个人开始重新发言。 1577. KillWolf.server.send("MESSAGE【系统】有用户退出，发言阶段重新开始。"); 1578. for(int j=0;j<KillWolf.serverData.UsersLen;j++){ 1579. if(KillWolf.serverData.ServerUsers[j] != null) 1580. if(KillWolf.serverData.ServerUsers[j].live) { 1581. serverData.StartSpeakName = KillWolf.serverData.ServerUsers[j].name; 1582. KillWolf.serverData.nowSpeakUser = KillWolf.serverData.ServerUsers[j].name; 1583. KillWolf.server.send("MESSAGE【系统】轮到"+KillWolf.serverData.ServerUsers[j].name+"发言"); 1584. KillWolf.server.sendto("ENABLESPEAKAC",KillWolf.serverData.ServerUsers[j].sk); 1585. SpeakThread spt = new SpeakThread(KillWolf.serverData.ServerUsers[j].name, serverData.Day); 1586. spt.start(); 1587. break; 1588. } 1589. } 1590. return true; 1591. } 1592. }  1595. } 1596. class SpeakThread extends Thread{ 1597. String user; 1598. int Day; 1599. public SpeakThread (String user,int Day){ 1600. this.user = user; 1601. this.Day = Day; 1602. } 1603. @Override 1604. public void run() { 1605. try { 1606. Thread.sleep(30\*1000); 1607. }catch (InterruptedException e){ 1608. e.printStackTrace(); 1609. } 1610. ServerDeal.nextSpeak(user,Day); 1611. } 1612. } 1613. class WolfSelect implements SelectionEventCallBack { 1614. public void run(String ReceiveID,int[] count,String users[],String target[],String[] openusers) { 1615. String name = "\*\*NONE\*\*",sameName = "\*\*NONE\*\*"; 1616. int max = 0,samemax = 0; 1617. System.out.println("####狼人投票--------"); 1618. String STR = "Night"+KillWolf.serverData.Day; 1619. for(int i=0;i<users.length;i++){ 1620. if(max < count[i]){ 1621. max = count[i]; 1622. name = users[i]; 1623. }else if(max == count[i]) { 1624. sameName = users[i]; 1625. samemax = max; 1626. } 1627. System.out.println("#"+users[i]+" "+count[i]); 1628. if(count[i] == 0) continue; 1629. STR = STR +"\n"+ "("+count[i]+")"+users[i]; 1630. boolean first = true; 1631. for(int j=0;j<target.length;j++) { 1632. if(target[j] != null) 1633. if(target[j].equals(users[i])) { 1634. if(first) { 1635. first = false; 1636. STR = STR + " <- "+openusers[j]; 1637. }else STR = STR +"、"+openusers[j]; 1638. } 1639. } 1640. } 1641. System.out.println(name + "--------"+ max); 1642. // KillWolf.serverData.getUser(name).live = false; 1643. String tip =""; 1644. if(name == "\*\*NONE\*\*") { 1645. name = "存在平票，无人出局"; 1646. KillWolf.serverData.deathUser="\*\*NONE\*\*"; 1647. }else if(samemax == max){ 1648. name = "存在平票，无人出局"; 1649. KillWolf.serverData.deathUser="\*\*NONE\*\*"; 1650. }else { 1651. tip = "被狼人暗杀"; 1652. KillWolf.serverData.deathUser=name; 1653. //KillWolf.serverData.getUser(KillWolf.serverData.deathUser).live = false; 1654. } 1655. DataUser[] wu = KillWolf.serverData.getUsersWithID("狼人"); 1656. String[] s = STR.split("\n"); 1657. for(int i=0;i<wu.length;i++) { 1658. for(int j=0;j<s.length;j++) { 1659. KillWolf.server.sendto("MESSAGE【狼人投票】"+s[j],wu[i].sk); 1660. } 1661. KillWolf.server.sendto("MESSAGE【最终结果】"+name+"("+max+")"+tip,wu[i].sk); 1662. } 1663. KillWolf.serverData.GameMainThread.gameWizard(KillWolf.serverData.deathUser); 1664. } 1665. } 1666. class VoteSelection implements SelectionEventCallBack { 1667. public void run(String ReceiveID,int[] count,String users[],String target[],String[] openusers) { 1668. String name = "\*\*NONE\*\*",sameName = "\*\*NONE\*\*"; 1669. int max = 0,samemax = 0; 1670. System.out.println("####投票--------"); 1671. String STR = "投票结束"; 1672. for(int i=0;i<users.length;i++){ 1673. if(max < count[i]){ 1674. max = count[i]; 1675. name = users[i]; 1676. }else if(max == count[i]) { 1677. sameName = users[i]; 1678. samemax = max; 1679. } 1680. System.out.println("#"+users[i]+" "+count[i]); 1681. if(count[i] == 0) continue; 1682. STR = STR +"\n"+ "("+count[i]+")"+users[i]; 1683. boolean first = true; 1684. for(int j=0;j<target.length;j++) { 1685. if(target[j] != null) 1686. if(target[j].equals(users[i])) { 1687. if(first) { 1688. first = false; 1689. STR = STR + " <- "+openusers[j]; 1690. }else STR = STR +"、"+openusers[j]; 1691. } 1692. } 1693. } 1694. System.out.println(name + "--------"+ max); 1695. // KillWolf.serverData.getUser(name).live = false; 1696. String tip =""; 1697. if(name == "\*\*NONE\*\*") { 1698. name = "存在平票，无人出局"; 1699. KillWolf.serverData.deathUser4="\*\*NONE\*\*"; 1700. }else if(samemax == max){ 1701. name = "存在平票，无人出局"; 1702. KillWolf.serverData.deathUser4="\*\*NONE\*\*"; 1703. }else { 1704. tip = "投票出局"; 1705. KillWolf.serverData.deathUser4=name; 1706. DataUser u =KillWolf.serverData.getUser(KillWolf.serverData.deathUser4); 1707. if(u!=null) u.live = false; 1708. } 1709. // System.out.println("--------------------"+STR); 1710. // DataUser[] wu = KillWolf.serverData.getUsersWithID("狼人"); 1711. //DataUser[] wu = new DataUser[users.length]; 1712. // if(KillWolf.serverData.getLivingUsers(false) instanceof DataUser[]) 1713. // wu = (DataUser[]) KillWolf.serverData.getLivingUsers(false); 1714. String id = ""; 1715. //参与本场投票的人 1716. for (int i =0;i<users.length;i++){ 1717. //wu[i]=KillWolf.serverData.getUser(users[i]); 1718. if(users[i].equals(name)) 1719. { 1720. DataUser u = KillWolf.serverData.getUser(users[i]); 1721. if(u!=null) id = u.identify; 1722. } 1723. } 1724. String[] s = STR.split("\n"); 1725. //for(int i=0;i<wu.length;i++) { 1726. for(int j=0;j<s.length;j++) { 1727. KillWolf.server.send("MESSAGE【投票】"+s[j]); 1728. //KillWolf.server.sendto(); 1729. } 1730. //if("".equals(id)) 1731. //else KillWolf.server.sendto("MESSAGE【最终结果】"+name+"("+max+")"+tip+"，他是"+id,wu[i].sk); 1732. //} 1733. try { 1734. Thread.sleep(2\*1000); 1735. } catch (InterruptedException e) { 1736. e.printStackTrace(); 1737. } 1738. KillWolf.server.send("MESSAGE【最终结果】"+name+"("+max+")"+tip); 1739. if(!"存在平票，无人出局".equals(name))KillWolf.server.send("USERDIED"+name); 1740. if(serverData.GameMainThread.isWin()) return; 1741. try { 1742. Thread.sleep(2000); 1743. } catch (InterruptedException e) { 1744. e.printStackTrace(); 1745. } 1746. if(id.equals("猎人"))KillWolf.serverData.GameMainThread.gameHunt(name); 1747. else{ 1748. serverData.GameMainThread.gameNext(); 1749. } 1750. } 1751. } 1752. // GameMain.java 1753. package KillWolf.Window; 1754. import KillWolf.Data.DataUser; 1755. import KillWolf.KillWolf; 1756. import javax.swing.\*; 1757. import java.awt.\*; 1758. import java.awt.event.ActionEvent; 1759. import java.awt.event.ActionListener; 1760. public class GameMain { 1761. // public static void main(String[] args) { 1762. // GameMain game = new GameMain(); 1763. // game.reset(); 1764. // } 1765. public boolean freeSpeak = false; 1766. private JFrame frame = new JFrame("开始游戏"); 1767. private JPanel panel = new JPanel(); 1768. //玩家列表 1769. private JLabel userListLabel = new JLabel("在场玩家"); 1770. private JList<String> userList = null; 1771. //出局玩家列表 1772. private JLabel outUserListLabel = new JLabel("淘汰玩家"); 1773. private JList<String> OutUserList = null; 1774. // 1775. private JLabel identity = null; 1776. //公屏 1777. private JLabel screenStr = new JLabel("公屏"); 1778. private JTextArea screen = new JTextArea(); 1779. private JScrollPane jsp = null; 1780. // //投票按钮 1781. // private JButton vote = new JButton("投票"); 1782. //发言框 1783. private JLabel submitBoxStr = new JLabel("发言框"); 1784. private JTextField submitStr = new JTextField(); 1785. //发言按钮 1786. private JButton submit = new JButton("发言"); 1787. public String[] UserListData ; 1788. public String[] UserListTipData ; 1789. public String[] UserOutListData ; 1790. public DataUser userSelf=null; 1791. public void reset(){ 1792. //本地玩家信息初始化 1793. for(DataUser user : KillWolf.Users){ 1794. if(user != null) { 1795. if (user.identify.equals("女巫")) { 1796. user.haveAntidote = true; 1797. user.havePoison = true; 1798. } 1799. user.live = true; 1800. } 1801. } 1802. //设置窗体参数 1803. frame.setSize(800,600); 1804. frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); 1805. frame.setLayout(null); 1806. frame.setResizable(false); 1807. frame.setVisible(false); 1808. frame.setLocationRelativeTo(null); 1809. //在场玩家列表设置 1810. userListLabel.setBounds(20,5,100,20); 1811. frame.add(userListLabel); 1812. String[] items1=new String[10]; 1813. userList=new JList(items1); //创建JList 1814. userList.setBounds(10, 30, 200, 240); 1815. userList.setFixedCellHeight(20); 1816. frame.add(userList); 1817. //淘汰玩家列表设置 1818. outUserListLabel.setBounds(20,275,100,20); 1819. frame.add(outUserListLabel); 1820. String[] items2=new String[10]; 1821. OutUserList=new JList(items2); //创建JList 1822. OutUserList.setBounds(10, 300, 200, 240); 1823. OutUserList.setFixedCellHeight(20); 1824. frame.add(OutUserList); 1825. //顶端字体设置 1826. DataUser[] users= KillWolf.Users; 1827. String[] usn = new String[KillWolf.UsersRealLen]; 1828. String[] usna = new String[KillWolf.UsersRealLen]; 1829. String[] usnt = new String[KillWolf.UsersRealLen]; 1830. int n = 0; 1831. for (int i=0;i<KillWolf.UsersLen;i++) { 1832. DataUser user = users[i]; 1833. if(user == null) continue; 1834. if(user.name.equals(KillWolf.UserName)){ 1835. userSelf=user; 1836. } 1837. usn[n]= user.name; 1838. //System.out.println("name:" + user.name); 1839. usna[n]=usn[n]; 1840. usnt[n] = ""; 1841. n++; 1842. } 1843. n=0; 1844. if(userSelf.identify.equals("狼人")){ 1845. for (int i=0;i<KillWolf.UsersLen;i++) { 1846. DataUser user = users[i]; 1847. if(user == null) continue; 1848. if(user.identify.equals("狼人")) { 1849. usnt[n] = "(狼人)"; 1850. } 1851. usna[n]=usn[n]+" "+usnt[n]; 1852. n++; 1853. } 1854. } 1855. userList.setListData(usna); 1856. UserListData = usn; 1857. UserListTipData = usnt; 1858. UserOutListData = new String[0]; 1859. identity = new JLabel(userSelf.name +"，你的身份是：" +userSelf.identify); ; 1860. identity.setBounds(270,0,450,100); 1861. identity.setFont(new Font(Font.DIALOG, Font.BOLD, 20)); 1862. frame.add(identity); 1863. //公屏设置 1864. panel.setBounds(270,100,450,380); 1865. panel.setLayout(null); 1866. jsp = new JScrollPane(screen); 1867. jsp.setBounds(0,0,450,380); 1868. screen.setBounds(0,0,450,380); 1869. screen.setFont(new Font(Font.DIALOG,Font.BOLD,16)); 1870. screen.setEditable(false); 1871. screen.setLineWrap(true); //激活自动换行功能 1872. screen.setWrapStyleWord(true); // 激活断行不断字功能 1873. screenStr.setBounds(270,60,100,50); 1874. jsp.setViewportView(screen); 1875. // screen.setBackground(Color.black); 1876. // jsp.setBackground(Color.BLUE); 1877. // panel.setBackground(Color.green); 1878. //jsp.setOpaque(false); 1879. jsp.setVerticalScrollBarPolicy(JScrollPane.VERTICAL\_SCROLLBAR\_AS\_NEEDED); 1880. panel.add(jsp); 1881. frame.add(panel); 1882. frame.add(screenStr); 1883. submitBoxStr.setBounds(270,470,100,50); 1884. submitStr.setBounds(270,510,300,30); 1885. submitStr.setFont(new Font(Font.DIALOG,Font.BOLD,15)); 1886. //submitStr.setText("请发言"); 1887. submitStr.setColumns(3); 1888. frame.add(submitBoxStr); 1889. frame.add(submitStr); 1890. submit.setBounds(600,510,120,30); 1891. submit.addActionListener(new ActionListener() { 1892. public void actionPerformed(ActionEvent e) { 1893. submit(); 1894. } 1895. }); 1896. frame.add(submit); 1897. this.freeSpeak = KillWolf.freeSpeak; 1898. ableSubmit(); 1899. frame.setVisible(true); 1900. KillWolf.client.send("GAMEREADY"+KillWolf.UserName); 1901. addToScreen("【系统】等待玩家..."); 1902. } 1903. public void forbidSubmit(){ 1904. if(freeSpeak)return; 1905. ClockRun = false; 1906. submit.setText("发言"); 1907. this.submit.setEnabled(false); 1908. } 1909. public void ableSubmit(){ 1910. //if(freeSpeak)return; 1911. this.submit.setEnabled(true); 1912. } 1913. public boolean ClockRun = false; 1914. class TimeTh extends Thread{ 1915. int time; 1916. public TimeTh(int time) { 1917. this.time = time-1; 1918. } 1919. public void run() { 1920. try { 1921. submit.setText("发言("+time+"s)"); 1922. Thread.sleep(1000); 1923. while(time>1 && ClockRun) { 1924. time--; 1925. submit.setText("发言("+time+"s)"); 1926. Thread.sleep(1000); 1927. } 1928. submit.setText("发言"); 1929. forbidSubmit(); 1930. }catch(Exception e) { 1931. e.printStackTrace(); 1932. } 1933. } 1934. } 1935. public void startSpeakClock(){ 1936. if(freeSpeak){ 1937. TimeTh tt = new TimeTh(120); 1938. ClockRun = true; 1939. tt.start(); 1940. return; 1941. } 1942. TimeTh tt = new TimeTh(30); 1943. ClockRun = true; 1944. tt.start(); 1945. } 1946. public void submit(){ 1947. System.out.println(submitStr.getText()); 1948. if(submitStr != null && (submitStr.getText() != null && !"".equals(submitStr.getText()))){ 1949. KillWolf.client.send("USERSPEAK"+userSelf.name+"\t"+submitStr.getText()); 1950. submitStr.setText(""); 1951. if(!KillWolf.freeSpeak){ 1952. this.submit.setEnabled(false); 1953. ClockRun = false; 1954. submit.setText("发言"); 1955. } 1956. } 1957. } 1958. public void addToScreen(String str){ 1959. //添加问本到公屏 1960. screen.append(str+"\r\n"); 1961. //显示最新内容 1962. screen.setCaretPosition(screen.getText().length()); 1963. } 1964. public void setDark(){ 1965. Color c = new Color(-12828863); 1966. if(userList != null) { 1967. userList.setBackground(c); 1968. userList.setForeground(Color.white); 1969. } 1970. if(OutUserList != null) { 1971. OutUserList.setBackground(c); 1972. OutUserList.setForeground(Color.white); 1973. } 1974. if(jsp != null){ 1975. jsp.setBackground(c); 1976. jsp.setForeground(Color.white); 1977. } 1978. if(screen != null){ 1979. screen.setBackground(c); 1980. screen.setForeground(Color.white); 1981. } 1982. if(submitStr != null){ 1983. submit.setBackground(c); 1984. submit.setForeground(Color.white); 1985. } 1986. if(userListLabel != null)userListLabel.setForeground(Color.white); 1987. if(outUserListLabel != null)outUserListLabel.setForeground(Color.white); 1988. if(screenStr != null)screenStr.setForeground(Color.white); 1989. if(submitBoxStr != null)submitBoxStr.setForeground(Color.white); 1990. if(identity != null)identity.setForeground(Color.white); 1991. if(frame != null) frame.getContentPane().setBackground(Color.black); 1992. // if(CenterLabel != null) CenterLabel.setForeground(Color.white); 1993. } 1994. public void setWhite(){ 1995. if(userList != null) { 1996. userList.setBackground(Color.white); 1997. userList.setForeground(Color.black); 1998. } 1999. if(OutUserList != null) { 2000. OutUserList.setBackground(Color.white); 2001. OutUserList.setForeground(Color.black); 2002. } 2003. if(screen != null){ 2004. screen.setBackground(Color.white); 2005. screen.setForeground(Color.black); 2006. } 2007. // if(jsp != null){ 2008. // jsp.setBackground(Color.white); 2009. // jsp.setForeground(Color.black); 2010. // } 2011. if(submitStr != null){ 2012. submit.setBackground(Color.white); 2013. submit.setForeground(Color.black); 2014. } 2015. if(userListLabel != null)userListLabel.setForeground(Color.black); 2016. if(outUserListLabel != null)outUserListLabel.setForeground(Color.black); 2017. if(screenStr != null)screenStr.setForeground(Color.black); 2018. if(submitBoxStr != null)submitBoxStr.setForeground(Color.black); 2019. if(identity != null)identity.setForeground(Color.black); 2020. if(frame != null) frame.getContentPane().setBackground(new Color(-1118482)); 2021. // if(CenterLabel != null) CenterLabel.setForeground(Color.white); 2022. } 2023. public JFrame getFrame() { 2024. return frame; 2025. } 2026. public void setFrame(JFrame frame) { 2027. this.frame = frame; 2028. } 2029. public JLabel getUserListLabel() { 2030. return userListLabel; 2031. } 2032. public void setUserListLabel(JLabel userListLabel) { 2033. this.userListLabel = userListLabel; 2034. } 2035. public JList<String> getUserList() { 2036. return userList; 2037. } 2038. public void setUserList(JList<String> userList) { 2039. this.userList = userList; 2040. } 2041. public JLabel getOutUserListLabel() { 2042. return outUserListLabel; 2043. } 2044. public void setOutUserListLabel(JLabel outUserListLabel) { 2045. this.outUserListLabel = outUserListLabel; 2046. } 2047. public JList<String> getOutUserList() { 2048. return OutUserList; 2049. } 2050. public void setOutUserList(JList<String> outUserList) { 2051. OutUserList = outUserList; 2052. } 2053. public JLabel getIdentity() { 2054. return identity; 2055. } 2056. public void setIdentity(JLabel identity) { 2057. this.identity = identity; 2058. } 2059. public JLabel getScreenStr() { 2060. return screenStr; 2061. } 2062. public void setScreenStr(JLabel screenStr) { 2063. this.screenStr = screenStr; 2064. } 2065. public JTextArea getScreen() { 2066. return screen; 2067. } 2068. public void setScreen(JTextArea screen) { 2069. this.screen = screen; 2070. } 2071. public JLabel getSubmitBoxStr() { 2072. return submitBoxStr; 2073. } 2074. public void setSubmitBoxStr(JLabel submitBoxStr) { 2075. this.submitBoxStr = submitBoxStr; 2076. } 2077. public JTextField getSubmitStr() { 2078. return submitStr; 2079. } 2080. public void setSubmitStr(JTextField submitStr) { 2081. this.submitStr = submitStr; 2082. } 2083. public JButton getSubmit() { 2084. return submit; 2085. } 2086. public void setSubmit(JButton submit) { 2087. this.submit = submit; 2088. } 2089. public String[] getUserListData() { 2090. return UserListData; 2091. } 2092. public void setUserListData(String[] userListData) { 2093. UserListData = userListData; 2094. } 2095. public String[] getUserOutListData() { 2096. return UserOutListData; 2097. } 2098. public void setUserOutListData(String[] userOutListData) { 2099. UserOutListData = userOutListData; 2100. } 2101. } 2102. // LoginWindow.java 2103. package KillWolf.Window; 2104. import java.awt.event.ActionEvent; 2105. import java.awt.event.ActionListener; 2106. import java.util.Random; 2107. import javax.swing.JButton; 2108. import javax.swing.JFrame; 2109. import javax.swing.JLabel; 2110. import javax.swing.JOptionPane; 2111. import javax.swing.JTextField; 2112. import KillWolf.KillWolf; 2113. import KillWolf.SocketDeal.ClientDeal; 2114. import KillWolf.SocketDeal.ServerDeal; 2115. import SocketServe.sClient; 2116. import SocketServe.sServer; 2117. public class LoginWindow { 2118. public JButton ServerButton = null; 2119. public JButton LoginButton = null; 2120. public JTextField userName = null; 2121. public JTextField ServerIp = null; 2122. public JFrame window = null; 2123. public String MIP = null; 2124. public void reset(String ip) { 2125. window = new JFrame("狼人杀 V.0.1"); 2126. window.setVisible(false); 2127. window.setSize(600, 200);//设置大小 2128. window.setLocationRelativeTo(null);//设置居中 2129. window.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);//设置可关闭 2130. window.setLayout(null);//设置绝对布局（窗口里面的内容不会随着窗口的改变而改变） 2131. window.setResizable(false);//设置窗口不可拉伸改变大小 2132. //设置用户名标签 2133. JLabel username\_label =new JLabel("用户名"); 2134. username\_label.setBounds(100,30,100,20); 2135. window.add(username\_label); 2136. //设置文本框 2137. userName = new JTextField(); 2138. userName.setBounds(160, 30, 300, 20); 2139. Random r = new Random(); 2140. //r.nextInt(); 2141. r.setSeed(System.currentTimeMillis()); 2142. userName.setText("user"+r.nextInt(10000)); 2143. window.add(userName); 2144. //设置标签 2145. JLabel ServerIp\_label =new JLabel("服务器地址"); 2146. ServerIp\_label.setBounds(80,80,100,20); 2147. window.add(ServerIp\_label); 2148. //设置文本框 2149. ServerIp = new JTextField(); 2150. ServerIp.setBounds(160, 80, 300, 20); 2151. ServerIp.setText(ip); 2152. this.MIP = ip; 2153. window.add(ServerIp); 2154. //JPasswordField pwd=new JPasswordField();//隐藏密码 2155. //pwd.setBounds(150, 200, 300, 50); 2156. //window.add(pwd); 2157. //设置按钮 2158. LoginButton = new JButton("连接服务器"); 2159. //Login.setEnabled(false); 2160. LoginButton.setBounds(180, 120, 100, 20); 2161. window.add(LoginButton); 2162. ServerButton = new JButton("作为服务器"); 2163. //Login.setEnabled(false); 2164. ServerButton.setBounds(320, 120, 100, 20); 2165. window.add(ServerButton); 2166. //设置作为主机动作 2167. ButtonAction ServerAction = new ButtonAction(); 2168. ServerButton.addActionListener(ServerAction); 2169. //设置连接服务器动作 2170. LinkAction LinkAction = new LinkAction(); 2171. LoginButton.addActionListener(LinkAction); 2172. window.setVisible(true);//设置面板可见 2173. } 2174. class ButtonAction implements ActionListener{ 2175. public void actionPerformed(ActionEvent event){ 2176. if(!setName()) return; 2177. ServerIp.setText(MIP); 2178. KillWolf.server = new sServer(); 2179. ServerDeal cb = new ServerDeal(); 2180. if(!KillWolf.server.start(cb)){ 2181. JOptionPane.showMessageDialog(null, "端口被占用，服务器启动失败！"); 2182. return; 2183. } 2184. if(!createClient(MIP)){ 2185. JOptionPane.showMessageDialog(null, "客户端启动失败！"); 2186. return; 2187. } 2188. //window.setTitle(ServerIp.getText()); 2189. KillWolf.ServerIp = MIP; 2190. KillWolf.isServer = true; 2192. ServerIp.setEnabled(false); 2193. ServerButton.setEnabled(false); 2194. KillWolf.client.send("ENTER"+KillWolf.UserName); 2196. //window.setTitle(ServerIp.getText()); 2197. } 2198. } 2199. class LinkAction implements ActionListener{ 2200. public void actionPerformed(ActionEvent event){ 2201. if(!setName()) return; 2202. if(KillWolf.ServerIp == null) { 2203. if(!createClient(ServerIp.getText())){ 2204. JOptionPane.showMessageDialog(null, "客户端启动失败！可能是服务器未连接成功。"); 2205. return; 2206. } 2207. KillWolf.ServerIp = ServerIp.getText(); 2208. KillWolf.isServer = false; 2209. } 2211. ServerIp.setEnabled(false); 2212. ServerButton.setEnabled(false); 2213. KillWolf.client.send("ENTER"+KillWolf.UserName); 2214. } 2215. } 2216. public boolean createClient(String IP) { 2217. KillWolf.client = new sClient(); 2218. ClientDeal cb = new ClientDeal(); 2219. //System.out.println("Connecting Server on "+IP+"..."); 2220. return KillWolf.client.start(IP,cb); 2221. } 2222. public boolean setName() { 2223. if( userName.getText().equals("")) { 2224. JOptionPane.showMessageDialog(null, "请输入用户名"); 2225. return false; 2226. } 2227. KillWolf.UserName = userName.getText(); 2228. return true; 2230. } 2231. } 2232. // SelectWindow.java 2233. package KillWolf.Window; 2234. import java.awt.Color; 2235. import java.awt.event.ActionEvent; 2236. import java.awt.event.ActionListener; 2237. import javax.swing.JButton; 2238. import javax.swing.JFrame; 2239. import javax.swing.JLabel; 2240. import javax.swing.JList; 2241. import javax.swing.JRootPane; 2242. import javax.swing.ListSelectionModel; 2243. import javax.swing.event.ListSelectionEvent; 2244. import javax.swing.event.ListSelectionListener; 2245. import KillWolf.KillWolf; 2246. import KillWolf.Data.SelectCallBack; 2247. /\* 2248. 构造类：String name,boolean ReceiveFromServer,String ReceiveID,boolean DisableParent,JFrame ParentWindow; 2249. 方法：设置标题（可用于显示剩余时间） setTitle(title) 2250. 启动 start(String tip,String ButtonTitle,String[] users, boolean canClose,SelectCallBack cb) 2251. 关闭 getSelectionAndClose() 2252. 接口：SelectCallBack 实现方法： run(String Selection) 当选择完毕后会回调，如果Selection为 \*\*NONE\*\* 表示没有选择/弃权 2254. for example: 2255. SelectWindow = new SelectWindow("选择",true,"123",true,loginwindow.window); 2256. String[] b = {"a","b","c",UserName}; 2257. SelectWindow.start("请选择", "投票", b, true,cb); 2259. cb另需构造 2260. \*/ 2261. public class SelectWindow { 2262. public boolean isRunCB = false; 2263. public String ReceiveID; 2264. public boolean ReceiveFromServer; 2265. public boolean DisableParent; 2266. public JFrame ParentWindow; 2267. public JFrame window; 2268. public JLabel CenterLabel; 2269. public JList<String> List; 2270. public String name; 2271. public JButton ConfirmButton; 2272. public JButton CloseButton; 2273. public String[] originUsers; 2274. public String[] usersTip; 2275. public String[] targetUsers; 2276. public int[] selectCount; 2277. public String LastSelect = "\*\*NONE\*\*"; 2278. public boolean isSendFinal = false; 2279. public SelectWindow(String name,boolean ReceiveFromServer,String ReceiveID,boolean DisableParent,JFrame ParentWindow) { 2280. this.ReceiveFromServer = ReceiveFromServer; 2281. this.ReceiveID = ReceiveID; 2282. this.DisableParent = DisableParent; 2283. this.ParentWindow = ParentWindow; 2284. this.name = name; 2285. } 2286. public boolean DarkMode = false; 2287. public SelectCallBack cb; 2288. public void setColorMode(boolean dark) { 2289. this.DarkMode = dark; 2290. if(dark) { 2291. if(List != null) { 2292. List.setBackground(Color.black); 2293. List.setForeground(Color.white); 2294. } 2295. if(window != null) window.getContentPane().setBackground(Color.black); 2296. if(CenterLabel != null) CenterLabel.setForeground(Color.white); 2297. //if() 2298. } 2299. } 2300. public void start(String tip,String ButtonTitle,String[] users, boolean canClose,SelectCallBack cb, int TimeLimit, String[] usertip) { 2301. this.cb = cb; 2302. if(usertip==null) { 2303. usertip = new String[users.length]; 2304. } 2305. this.usersTip = usertip; 2306. window = new JFrame(name); 2307. window.setVisible(false); 2308. window.setSize(300, 400);//设置大小 2309. window.setLocationRelativeTo(null);//设置居中 2310. window.setDefaultCloseOperation(JFrame.DO\_NOTHING\_ON\_CLOSE);//设置可关闭 2311. window.setLayout(null);//设置绝对布局（窗口里面的内容不会随着窗口的改变而改变） 2312. window.setResizable(false);//设置窗口不可拉伸改变大小 2313. window.setAlwaysOnTop(true); 2314. window.setUndecorated(true); //不显示标题栏,最大化,最小化,退出按钮 2315. window.getRootPane().setWindowDecorationStyle(JRootPane.WARNING\_DIALOG);//使frame只剩下标题栏 2316. CenterLabel =new JLabel(tip); 2317. CenterLabel.setBounds(10,10,260,20); 2318. window.add(CenterLabel); 2319. users = users.clone(); 2320. originUsers = new String[users.length]; 2321. targetUsers = new String[users.length]; 2322. for(int i=0;i<users.length;i++) { 2323. originUsers[i] = users[i]; 2324. targetUsers[i] = ""; 2325. if(usersTip[i]==null) usersTip[i]=""; 2326. if(ReceiveFromServer) users[i] = "(0)"+users[i]+" "+usersTip[i]; 2327. else users[i] = users[i]+" "+usersTip[i]; 2328. } 2329. selectCount = new int[users.length]; 2330. List = new JList(users); //创建JList 2331. List.setBounds(10, 40, 260, 260); 2332. List.setFixedCellHeight(20); 2333. List.addListSelectionListener(new ListSelectionHandler()); 2334. List.setSelectionMode(ListSelectionModel.SINGLE\_SELECTION); 2335. window.add(List); 2337. ConfirmButton = new JButton(ButtonTitle); 2338. ConfirmButton.setEnabled(false); 2339. ConfirmButton.setBounds(30, 310, 100, 40); 2340. ConfirmButton.addActionListener(new ButtonAction("confirm")); 2341. window.add(ConfirmButton);  2344. CloseButton = new JButton("弃权"); 2345. if(canClose) { 2346. CloseButton.setEnabled(true); 2347. }else { 2348. CloseButton.setEnabled(false); 2349. } 2350. CloseButton.setBounds(160, 310, 100, 40); 2351. CloseButton.addActionListener(new ButtonAction("cancel")); 2352. window.add(CloseButton);  2355. if(this.DisableParent && this.ParentWindow != null) this.ParentWindow.setEnabled(false); 2357. TimeTh at = new TimeTh(TimeLimit); 2358. at.start(); 2360. setColorMode(this.DarkMode); 2362. window.setVisible(true); 2363. } 2364. class TimeTh extends Thread{ 2365. int time; 2366. public TimeTh(int time) { 2367. this.time = time-1; 2368. } 2369. public void run() { 2370. try { 2371. window.setTitle(time+"s "+name); 2372. while(time>0) { 2373. Thread.sleep(1000); 2374. time--; 2375. window.setTitle(time+"s "+name); 2376. } 2377. window.setTitle(name); 2378. ConfirmButton.setEnabled(false); 2379. CloseButton.setEnabled(false); 2380. }catch(Exception e) { 2381. e.printStackTrace(); 2382. } 2383. } 2384. } 2385. public void setTitle(String Title) { 2386. window.setTitle(name + Title); 2387. } 2388. public void SELECTInfo(String ServerText) { 2389. if(!ReceiveFromServer) return; 2390. String[] c = ServerText.split("\t"); 2391. // c[0] ReceiveID ,c[1] 发出者 ,c[2] 目标者, c[3] 原始目标（被取消） 2392. String[] nu = new String[this.originUsers.length]; 2394. if(c[0].equals(this.ReceiveID)) { 2396. for(int i=0;i<this.originUsers.length;i++) { 2397. //nu[i] = this.originUsers[i]; 2399. if(this.originUsers[i].equals(c[1])) { 2400. if(c[2].equals("\*\*NONE\*\*")) targetUsers[i] = " 弃权"; 2401. else targetUsers[i] = " -> " + c[2]; 2402. } 2403. if(this.originUsers[i].equals(c[2])) { 2404. selectCount[i]++; 2405. System.out.println(this.originUsers[i]+" +1 => "+ selectCount[i]); 2406. } 2407. if(this.originUsers[i].equals(c[3])) { 2408. selectCount[i]--; 2409. System.out.println(this.originUsers[i]+" -1 => "+ selectCount[i]); 2410. } 2411. System.out.println(this.originUsers[i]+" "+c[2]+" "+ selectCount[i]); 2412. //System.out.println("LIST0: "+nu[i]); 2413. //System.out.println("LISTc: "+selectCount[i]); 2414. } 2415. for(int i=0;i<nu.length;i++) { 2416. nu[i] = "(" +selectCount[i] + ")" +originUsers[i]+" "+usersTip[i]+ targetUsers[i]; 2417. //System.out.println("LIST: "+nu[i]); 2418. } 2419. List.setListData(nu); 2420. } 2422. } 2423. class ListSelectionHandler implements ListSelectionListener { 2424. public void valueChanged(ListSelectionEvent e) { 2425. if(e.getValueIsAdjusting()) { 2426. String t; 2427. if(List.getSelectedIndex() == -1) { 2428. //t = "\*\*NONE\*\*"; 2429. return; 2430. } 2431. else t = originUsers[List.getSelectedIndex()]; 2432. System.out.printf("Select: %s\n",t); 2433. ConfirmButton.setEnabled(true); 2435. String a = LastSelect; 2436. if(List.getSelectedIndex() == -1) 2437. LastSelect = "\*\*NONE\*\*"; 2438. else LastSelect = originUsers[List.getSelectedIndex()]; 2440. if(ReceiveFromServer) KillWolf.client.send("SELECT"+ReceiveID+"\t"+KillWolf.UserName+"\t"+t+"\t"+a); 2441. } 2443. //output.append("LeadSelectionIndex is " + lsm.getLeadSelectionIndex() + "\n");  2446. } 2447. } 2448. class ButtonAction implements ActionListener{//开始游戏 2449. String ID; 2450. public ButtonAction(String ID) { 2451. this.ID = ID; 2452. } 2453. public void actionPerformed(ActionEvent event){ 2454. isSendFinal = true; 2455. List.setEnabled(false); 2456. CloseButton.setEnabled(false); 2457. ConfirmButton.setEnabled(false); 2458. isRunCB = true; 2459. if(ID.equals("confirm")) { 2460. //List.setEnabled(false); 2461. if(ReceiveFromServer) { 2462. KillWolf.client.send("FINALSELECT"+ReceiveID+"\t"+KillWolf.UserName+"\t"+LastSelect); 2463. } 2464. if(cb != null) cb.run(LastSelect); 2465. }else { 2466. //取消 2467. if(ReceiveFromServer) { 2468. KillWolf.client.send("SELECT"+ReceiveID+"\t"+KillWolf.UserName+"\t"+"\*\*NONE\*\*"+"\t"+LastSelect); 2469. KillWolf.client.send("FINALSELECT"+ReceiveID+"\t"+KillWolf.UserName+"\t"+"\*\*NONE\*\*"); 2470. } 2471. if(cb != null) cb.run("\*\*NONE\*\*"); 2472. }   2476. } 2477. } 2478. public String getSelectionAndClose(String ReceiveID) { 2479. if(!ReceiveID.equals(this.ReceiveID)) return ""; 2480. boolean a = isRunCB; 2481. isRunCB = true; 2482. window.setVisible(false); 2483. window.setEnabled(false); 2484. if(!a && cb != null) { 2485. cb.run(LastSelect); 2486. } 2487. if(!isSendFinal && ReceiveFromServer){ 2488. isSendFinal = true; 2489. KillWolf.client.send("FINALSELECT"+ReceiveID+"\t"+KillWolf.UserName+"\t"+LastSelect); 2490. } 2491. if(DisableParent && ParentWindow != null) ParentWindow.setEnabled(true); 2493. return LastSelect; 2494. } 2495. } 2496. // WaitRoom.java 2497. package KillWolf.Window; 2498. import javax.swing.\*; 2499. import javax.swing.border.Border; 2500. import javax.swing.event.\*; 2501. import javax.swing.text.BadLocationException; 2502. import javax.swing.text.Document; 2503. import KillWolf.KillWolf; 2504. import KillWolf.Data.DataUser; 2505. import KillWolf.SocketDeal.ClientDeal; 2506. import java.awt.\*; 2507. import java.awt.event.\*; 2508. import java.util.Random; 2509. import SocketServe.\*; 2510. public class WaitRoom { 2511. public JButton LoginButton = null; 2512. public JTextField peoCount = null; 2513. public JTextField wolfCount = null; 2514. public JCheckBox wizardOption; 2515. public JCheckBox hunterOption; 2516. public JCheckBox yuyanjiaOption; 2517. public JCheckBox winmodeOption; 2518. public JCheckBox speakOption; 2519. //public JCheckBox optionD; 2520. public JList UserList; 2521. public JFrame window = null; 2522. public JLabel Userlenlabel; 2523. public JLabel Charlenlabel; 2524. public String MIP = null; 2525. public int CharLen = 0; 2526. public int peoLen = 1; 2527. public int wolfLen = 1; 2528. public int wizardLen = 0; 2529. public int hunterLen = 0; 2530. public int yuyanjiaLen = 0; 2531. public void reset(String ip,String name,boolean isServer) { 2532. KillWolf.serverData.freeSpeak = false; 2533. KillWolf.serverData.playMode = false; 2535. window = new JFrame("狼人杀 "+name+" "+ip); 2536. window.setVisible(false); 2537. window.setSize(400, 420);//设置大小 2538. window.setLocationRelativeTo(null);//设置居中 2539. window.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);//设置可关闭 2540. window.setLayout(null);//设置绝对布局（窗口里面的内容不会随着窗口的改变而改变） 2541. window.setResizable(false);//设置窗口不可拉伸改变大小 2542. //设置用户名标签 2543. JLabel username\_label =new JLabel("用户名："+name); 2544. username\_label.setBounds(10,10,200,20); 2545. window.add(username\_label); 2546. //设置标签 2547. JLabel ServerIp\_label =new JLabel("服务器地址："+KillWolf.ServerIp); 2548. ServerIp\_label.setBounds(10,40,200,20); 2549. window.add(ServerIp\_label); 2550. //设置标签 2551. Userlenlabel =new JLabel("在线玩家数："+KillWolf.UsersRealLen); 2552. Userlenlabel.setBounds(10,150,200,20); 2553. window.add(Userlenlabel); 2554. JLabel label2 =new JLabel("等待开始..."); 2555. label2.setBounds(10,70,200,20); 2556. window.add(label2); 2557. this.MIP = ip; 2558. //设置按钮 2560. if(isServer) { 2561. LoginButton = new JButton("开始游戏"); 2562. //LoginButton.setEnabled(false); 2563. }else { 2564. LoginButton = new JButton("等待房主开始"); 2565. LoginButton.setEnabled(false); 2566. } 2567. LoginButton.setBounds(10, 120, 100, 20); 2568. window.add(LoginButton); 2570. LinkAction LinkAction = new LinkAction(); 2571. LoginButton.addActionListener(LinkAction);  2574. String[] items=new String[10]; 2575. UserList=new JList(items); //创建JList 2576. UserList.setBounds(200, 10, 150, 300); 2577. UserList.setFixedCellHeight(20); 2578. window.add(UserList); 2580. if(KillWolf.isServer) { 2581. //设置文本框 2582. Charlenlabel =new JLabel("已选角色数："+CharLen); 2583. Charlenlabel.setBounds(10,170,200,20); 2584. window.add(Charlenlabel); 2586. JLabel Peolenlabel =new JLabel("平民"); 2587. Peolenlabel.setBounds(10,200,40,20); 2588. window.add(Peolenlabel); 2589. peoCount = new JTextField(); 2590. peoCount.setBounds(50,200,40,20); 2591. peoCount.setText(Integer.toString(peoLen)); 2592. window.add(peoCount); 2594. JLabel Wolflenlabel =new JLabel("狼人"); 2595. Wolflenlabel.setBounds(10,220,40,20); 2596. window.add(Wolflenlabel); 2597. wolfCount = new JTextField(); 2598. wolfCount.setBounds(50,220,40,20); 2599. wolfCount.setText(Integer.toString(wolfLen)); 2600. window.add(wolfCount);  2603. Document dt = wolfCount.getDocument(); 2604. dt.addDocumentListener(new TextListener("wolf")); 2605. wolfCount.addKeyListener(new CountKeyListener());  2608. Document dt2 = peoCount.getDocument(); 2609. dt2.addDocumentListener(new TextListener("people")); 2610. peoCount.addKeyListener(new CountKeyListener()); 2612. //TextOnValueChanged WAL = new TextOnValueChanged(); 2613. //wolfCount.addActionListener(WAL); 2614. //peoCount.addActionListener(WAL); 2616. wizardOption = new JCheckBox("女巫"); 2617. wizardOption.setBounds(10,240,80,20); 2618. hunterOption = new JCheckBox("猎人"); 2619. hunterOption.setBounds(10,260,80,20); 2620. yuyanjiaOption = new JCheckBox("预言家"); 2621. yuyanjiaOption.setBounds(10,280,100,20); 2622. //optionD = new JCheckBox("D.");  2625. wizardOption.addItemListener(new CheckValueChanged("wizard")); 2626. hunterOption.addItemListener(new CheckValueChanged("hunter")); 2627. yuyanjiaOption.addItemListener(new CheckValueChanged("yuyanjia")); 2629. window.add(wizardOption); 2630. window.add(hunterOption); 2631. window.add(yuyanjiaOption);  2634. winmodeOption = new JCheckBox("屠城玩法（狼人需要全部淘汰民和神）"); 2635. winmodeOption.setBounds(10,320,300,20); 2636. speakOption = new JCheckBox("允许随时发言"); 2637. speakOption.setBounds(10,340,100,20); 2639. winmodeOption.addItemListener(new CheckValueChanged("获胜玩法")); 2640. speakOption.addItemListener(new CheckValueChanged("发言")); 2642. window.add(winmodeOption); 2643. window.add(speakOption); 2645. updateCharCount(false); 2646. } 2647. window.setVisible(true);//设置面板可见  2650. //run(new ListTest(),250,375); 2652. } 2653. public void setUserCount(String users[]) { 2654. Userlenlabel.setText("在线玩家数："+KillWolf.UsersRealLen); 2655. UserList.setListData(users); 2656. UserList.setFixedCellHeight(20); 2658. updateCharCount(false); 2659. //window.setTitle(Title); 2660. } 2661. class LinkAction implements ActionListener{//开始游戏 2662. public void actionPerformed(ActionEvent event){ 2663. if(updateCharCount(true)) { 2664. LoginButton.setEnabled(false); 2665. Random r = new Random(); 2666. //r.nextInt(); 2667. r.setSeed(System.currentTimeMillis()); 2668. int n; 2669. if(wizardOption.isSelected()) { 2670. do{n = r.nextInt(KillWolf.serverData.UsersLen);} 2671. while(KillWolf.serverData.ServerUsers[n] == null || KillWolf.serverData.ServerUsers[n].identify != null); 2672. KillWolf.serverData.ServerUsers[n].identify = "女巫"; 2673. } 2674. if(hunterOption.isSelected()) { 2675. do{n = r.nextInt(KillWolf.serverData.UsersLen);} 2676. while(KillWolf.serverData.ServerUsers[n] == null || KillWolf.serverData.ServerUsers[n].identify != null); 2677. KillWolf.serverData.ServerUsers[n].identify = "猎人"; 2678. } 2679. if(yuyanjiaOption.isSelected()) { 2680. do{n = r.nextInt(KillWolf.serverData.UsersLen);} 2681. while(KillWolf.serverData.ServerUsers[n] == null || KillWolf.serverData.ServerUsers[n].identify != null); 2682. KillWolf.serverData.ServerUsers[n].identify = "预言家"; 2683. } 2684. for(int i=0;i<wolfLen;i++){ 2685. do{n = r.nextInt(KillWolf.serverData.UsersLen);} 2686. while(KillWolf.serverData.ServerUsers[n] == null || KillWolf.serverData.ServerUsers[n].identify != null); 2687. KillWolf.serverData.ServerUsers[n].identify = "狼人"; 2688. } 2690. for(int i=0;i<KillWolf.serverData.UsersLen;i++){ 2691. if(KillWolf.serverData.ServerUsers[i] != null && KillWolf.serverData.ServerUsers[i].identify == null) { 2692. KillWolf.serverData.ServerUsers[i].identify = "平民"; 2693. } 2694. }  2697. String STR = ""; 2698. for(int i=0;i<KillWolf.serverData.UsersLen;i++) { 2699. DataUser p = KillWolf.serverData.ServerUsers[i]; 2700. if(p!=null)STR = STR + "#u#" + p.name+"\t"+p.identify; 2701. } 2702. System.out.print(STR); 2703. KillWolf.server.send("GAMESTART"+(KillWolf.serverData.freeSpeak?"TRUE":"FALSE")+STR); 2704. } 2705. } 2706. } 2708. class CheckValueChanged implements ItemListener{ 2709. String ID; 2710. public CheckValueChanged(String ID) { 2711. this.ID = ID; 2712. } 2713. public void itemStateChanged(ItemEvent e) { 2714. JCheckBox jcb = (JCheckBox) e.getItem();// 将得到的事件强制转化为JCheckBox类 2715. if (jcb.isSelected()) {// 推断是否被选择 2716. if(this.ID.equals("wizard")) wizardLen = 1; 2717. else if(this.ID.equals("hunter")) hunterLen = 1; 2718. else if(this.ID.equals("yuyanjia")) yuyanjiaLen = 1; 2719. else if(this.ID.equals("发言")) KillWolf.serverData.freeSpeak = true; 2720. else if(this.ID.equals("获胜玩法")) KillWolf.serverData.playMode = true; 2721. } else { 2722. if(this.ID.equals("wizard")) wizardLen = 0; 2723. else if(this.ID.equals("hunter")) hunterLen = 0; 2724. else if(this.ID.equals("yuyanjia")) yuyanjiaLen = 0; 2725. else if(this.ID.equals("发言")) KillWolf.serverData.freeSpeak = false; 2726. else if(this.ID.equals("获胜玩法")) KillWolf.serverData.playMode = false; 2727. } 2728. updateCharCount(false); 2729. } 2730. } 2731. class TextListener implements DocumentListener{ 2732. String ID; 2733. public TextListener(String ID) { 2734. this.ID = ID; 2735. } 2736. public void insertUpdate(DocumentEvent e) { 2737. //System.out.println("insertUpdate" + e.toString()); 2738. changedUpdate(e); 2739. } 2740. public void removeUpdate(DocumentEvent e) { 2741. //System.out.println("removeUpdate"+e.toString()); 2742. changedUpdate(e); 2743. } 2745. public void changedUpdate(DocumentEvent e) { 2747. try { 2748. int num = Integer.parseInt(e.getDocument().getText(0, e.getDocument().getLength())); 2749. if(ID.equals("people")) { 2750. peoLen = Math.abs(num); 2751. }else { 2752. wolfLen = Math.abs(num); 2753. } 2754. updateCharCount(false); 2756. }catch (BadLocationException e1) { 2757. // TODO Auto-generated catch block 2758. e1.printStackTrace(); 2759. }catch(Exception e21) { 2760. updateCharCount(false); 2761. }  2764. } 2765. } 2766. public class CountKeyListener implements KeyListener { 2767. @Override 2768. public void keyTyped(KeyEvent e) { 2769. // TODO Auto-generated method stub 2770. int keyChar=e.getKeyChar(); 2771. if (keyChar>=KeyEvent.VK\_0 && keyChar<=KeyEvent.VK\_9) { 2772. } else { 2773. e.consume(); 2774. } 2775. } 2776. @Override 2777. public void keyPressed(KeyEvent e) { 2778. // TODO Auto-generated method stub 2779. } 2780. @Override 2781. public void keyReleased(KeyEvent e) { 2782. // TODO Auto-generated method stub 2783. } 2784. } 2785. boolean updateCharCount(boolean isFinal) { 2786. CharLen = wolfLen+peoLen+wizardLen+hunterLen+yuyanjiaLen; 2787. if(Charlenlabel!=null) Charlenlabel.setText("已选角色数："+CharLen); 2788. if(!KillWolf.isServer) return true; 2789. if(wolfLen<1 && isFinal) { 2790. JOptionPane.showMessageDialog(null, "狼人角色数量至少为1"); 2791. return false; 2792. } 2794. if(CharLen == KillWolf.serverData.UsersRealLen) { 2796. LoginButton.setEnabled(true); 2797. LoginButton.setText("开始游戏"); 2798. return true; 2799. }else { 2800. LoginButton.setEnabled(false); 2801. LoginButton.setText("角色不匹配"); 2802. return false; 2803. } 2804. } 2805. //定义新类，实现Exit按钮的时间监听 2806. /\*class btnAction implements ActionListener 2807. { //接收事件 2808. public void actionPerformed(ActionEvent event) 2809. { 2810. Object object = event.getSource(); 2811. if (object == JButton1) 2812. JButton1\_actionPerformed(event); 2813. } 2814. } \*/ 2815. } | | | | | | |
| **分析总结、收获和体会:** | | | | | | |
| 优点：  图形化页面、对所有用户的退出存在监测并实时应用到游戏中、联网游戏。  创新之处：  网络联机  不足之处：  图形页面没有图片、图标等  需要改进的地方：  图形页面没有图片、图标等 | | | | | | |
| **自查自纠：** | | | | | 是 | 否 |
| 程序是否有尚未解决的问题或bug？ | | | | |  | 否 |
| 程序代码是否符合代码规范(对齐与缩进，有必要的注释)？ | | | | | 是 |  |
| 是否按模块化要求进行了程序设计，系统功能是否完善？ | | | | | **是** |  |
| 是否独立完成，未参考其他人的设计或代码？ | | | | | 是 |  |
|  | | | | | | |
| **报告完成日期：2021-07-30** | | | | | | |