TRƯỜNG ĐẠI HỌC ĐẠI NAM

KHOA CÔNG NGHỆ THÔNG TIN

BÀI TẬP LỚN

HỌC PHẦN: LẬP TRÌNH MẠNG

ĐỀ TÀI: Lập Trình Game Ma Sói Java

Giảng viên: Nguyễn Quang Hưng

Lớp : CNTT 15-04

TT	Mã SV	Họ và Tên	Ngày sinh
1	1571020257	Nguyễn Ngọc Đan Trường	05/12/2003
2	1571020100	Nguyễn Đức Hiệp	02/08/2003
3	1571020260	Dương Văn Tuân	24/04/2003
4	1571020202	Nguyễn Vũ Phúc	24/01/2003

Hà Nội, tháng 4 năm 2024

TRƯỜNG ĐẠI HỌC ĐẠI NAM

KHOA CÔNG NGHỆ THÔNG TIN

BÀI TẬP LỚN

HỌC PHẦN: LẬP TRÌNH MẠNG

ĐỀ TÀI: Lập Trình Game Ma Sói Java

TT	Mã SV	Họ và Tên	Điểm	
			Bằng Số	Bằng Chữ
1	1571020257	Nguyễn Ngọc Đan Trường		
2	1571020100	Nguyễn Đức Hiệp		
3	1571020260	Dương Văn Tuân		
4	1571020202	Nguyễn Vũ Phúc		

CÁN BỘ CHẤM THI

Nguyễn Quang Hưng

Hà Nội, tháng 4 năm 2024

LỜI NÓI ĐẦU

Đầu tiên, chúng em xin gửi lời cảm ơn chân thành đến Trường Đại học Đại Nam đã đưa môn Lập Trình Mạng vào chương trình giảng dạy. Đặc biệt, chúng em xin gửi lời cảm ơn sâu sắc đến giảng viên bộ môn – thầy Nguyễn Quang Hưng đã dạy dỗ, truyền đạt những kiến thức quý báu cho chúng em trong suốt thời gian học tập vừa qua. Trong thời gian tham gia lớp học, nhóm chúng em đã có thêm cho mình nhiều kiến thức bổ ích, tinh thần học tập hiệu quả, nghiêm túc. Đây chắc chắn sẽ là những kiến thức quý báu, là hành trang để chúng em có thể vững bước sau này.

Bộ môn Lập Trình Mạng là môn học thú vị, vô cùng bổ ích và có tính thực tế cao. Đảm bảo cung cấp đủ kiến thức, gắn liền với nhu cầu thực tiễn của sinh viên. Tuy nhiên, do vốn kiến thức còn nhiều hạn chế và khả năng tiếp thu thực tế còn nhiều bỡ ngỡ. Mặc dù nhóm chúng em đã cố gắng hết sức nhưng bài tiểu luận khó có thể tránh khỏi những thiếu sót và nhiều chỗ còn chưa chính xác, kính mong thầy xem xét và góp ý để bài tiểu luận của chúng em được hoàn thiện hơn.

Nhóm chúng em xin chân thành cảm ơn!

Mục Lục

MỤC LỤC HÌNH ẢNH

Chương 1. KIẾN THỨC CƠ BẢN

1.Khái niệm Java

Java được biết đến là ngôn ngữ lập trình bậc cao, hướng đối tượng và giúp bảo mật mạnh mẽ, và còn được định nghĩa là một platform. Nó được sử dụng trong phát triển phần mềm, trang web, game hay ứng dụng trên các thiết bị di động. Java được phát triển bởi Sun Microsystems, do James Gosling khởi xướng và ra mắt năm 1995. Java hoạt động trên rất nhiều nền tảng như Windows, Mac và các phiên bản khác nhau của UNIX.

1.1.Đặc điểm

Được biết Java rất phổ biến và đã thống trị lĩnh vực này từ đầu những năm 2000, đến nay Java đã được sử dụng trong đa dạng các lĩnh vực khác nhau như:

- Phát triển ứng dụng cho thiết bị di động
- Phát triển ứng dụng cho máy tính
- Phát triển game
- Phát triển các ứng dụng sử dụng công nghệ blockchain
- Phát triển các ứng dụng dựa trên trí thông minh nhân tạo
- Phát triển công nghệ Internet vạn vật (IoT)
- Phát triển cơ sở dữ liệu

1.2.Ưu điểm và nhược điểm của Java

Ưu điểm của Java:

- Độ tin cậy cao
- Tính đa nền tảng
- Quản lý bộ nhớ tự động
- Công cụ phát triển phong phú
- Hỗ trợ đa luồng

Nhược điểm của Java:

- Tốc độ chậm hơn so với các ngôn ngữ lập trình gần sát với phần cứng, chẳng

hạn như C hoặc C++.

- Java có thể chạy trên nhiều nền tảng khác nhau, nhưng ứng dụng này có thể cần đến một trình biên dịch hoặc máy ảo Java riêng biệt để có thể chạy trên các thiết bị di động.
- Sử dụng bộ nhớ lớn hơn so với một số ngôn ngữ lập trình khác.
- Cú pháp phức tạp hơn so với một số ngôn ngữ lập trình khác

2.Khái niệm Lập trình mạng

Lập trình mạng (Network Programming) là hành động sử dụng mã máy tính để viết các chương trình hoặc quy trình có thể giao tiếp với các chương trình hoặc quy trình khác trên mạng. Các lập trình viên sử dụng các ngôn ngữ lập trình, thư viện mã và giao thức khác nhau để thực hiện công việc. Chính xác hơn, khả năng lập trình mạng là quá trình sử dụng mã, các khái niệm dựa trên vòng đời phát triển phần mềm và các công cụ khác để làm cho mạng thực hiện các hành động. Hiểu đơn giản, đây là một trong những nhiệm vụ cơ bản được sử dụng để phát triển các ứng dụng, hệ thống doanh nghiệp. Nó có thể bao gồm hệ các chương trình như phần mềm kế toán, nhân sự, … hoặc đến những ứng dụng giải trí như trò chơi, điều khiển, …

Trên thị trường hiện nay, chúng đang được sử dụng thông qua 4 loại cơ bản như:

- Local Area Network - LAN.
- Wide Area Network - WAN.
- Metropolitan Area Network - MAN.
- Personal Area Network - PAN.

Thông qua những điều trên, ta có thể thấy được lập trình mạng được xem là một công việc với mục đích thúc đẩy sự phát triển của các ứng dụng doanh nghiệp đang hoạt động trên Internet.

Lập trình mạng sẽ bao hàm những yếu tố sau:

- Kiến thức mạng truyền thông, bao gồm mạng máy tính, PSTN…
- Mô hình lập trình mạng, được thể hiện thông qua:
    - Mạng - LAN.
    - Mạng diện rộng - WAN.
    - Mạng đô thị - MAN.

    - Mạng cá nhân - PAN.
    - Virtual Private Network - VPN.
    - Storage-Area Network - SAN.
- Ngôn ngữ lập trình mạng Java .NET, C/C++, Delphi, Javascript…

2.1. Mô hình lập trình mạng

Với lập trình mạng, sẽ có 2 mô hình mà bạn cần lưu ý là mô hình OSI và mô hình 7 lớp. Mô hình Kết nối Hệ thống Mở (OSI) đóng vai trò là hướng dẫn cho các kỹ sư mạng, nhà phát triển và những người khác liên quan đến lập trình mạng. Mô hình này giúp họ hiểu cách các sản phẩm và chương trình phần mềm có thể giao tiếp và tương tác với nhau.

OSI bao gồm bảy lớp hiển thị cách dữ liệu di chuyển qua và trong các mạng. Các lớp là: vật lý, liên kết dữ liệu, mạng, truyền tải, phiên, bản trình bày và ứng dụng.

2.2. Các kiến thức đi kèm

Ngoài những điều quan trọng trên cần phải nắm về lập trình mạng, bạn cần phải trang bị thêm cho mình những kiến thức liên quan như:

- Ansible: Công cụ mã nguồn mở cho IaC.
- Docker: Nền tảng chứa mã nguồn mở.
- XML: Ngôn ngữ đánh dấu có thể mở rộng.
- Mạng Linux: cùng một số kỹ năng liên quan đến Linux.
- API REST: Hỗ trợ chuyển trạng thái đại diện.
- JSON: Ảnh hưởng đến định dạng tệp tiêu chuẩn mở và định dạng trao đổi dữ liệu.
- Git và GitHub: Phần mềm điều khiển phiên bản nguồn mở / giao diện dựa trên website.
- NETCONF - giao thức, A - ngôn ngữ mô hình hóa dữ liệu, NFV - ảo hóa các chức năng mạng.

2.3. Socket

Socket là một điểm cuối(end-point) của liên kết truyền thông hai chiều(two-way communication) giữa hai chương trình chạy trên mạng. Các lớp Socket được sử dụng để biểu diễn kết nối giữa client và server. Gói java.net cung cấp hai lớp - Socket và ServerSocket - thể hiện kết nối giữa client và server.

Trong Java, lớp Socket được sử dụng để tạo và quản lý kết nối mạng giữa các ứng dụng chạy trên các thiết bị khác nhau. Socket là cơ sở của giao thức TCP/IP (Transmission Control Protocol/Internet Protocol), cho phép truyền và nhận dữ liệu qua mạng và một số điểm chính như : Tạo kết nối , Gửi và nhận dữ liệu , Xử lý kết nối đồng thời .

2.4. BufferedReader

BufferedReader là một lớp Java để đọc văn bản từ luồng đầu vào (như tệp) bằng cách đệm và đọc liền mạch các ký tự, mảng hoặc dòng. Nói chung, mỗi yêu cầu đọc được tạo bởi Reader sẽ khiến yêu cầu đọc tương ứng được tạo từ ký tự hoặc luồng byte bên dưới.

2.5. PrintWriter

PrintWriter là một lớp trong Java được sử dụng để ghi các dữ liệu định dạng text vào một luồng (stream) đầu ra. Nó là một phần của gói java.io và thường được sử dụng để ghi dữ liệu vào các tập tin, sockets, hoặc các luồng dữ liệu khác và có một số điểm chính như :

- Khởi tạo PrintWriter
- Ghi dữ liệu
- Đóng luồng

3. Khái niệm về tcp/udp

TCP (Transmission Control Protocol) và UDP (User Datagram Protocol) là hai giao thức chính được sử dụng trong lập trình mạng để truyền dữ liệu giữa các thiết bị trên mạng.

3.1. TCP (Transmission Control Protocol):

TCP là một giao thức kết nối hướng luồng. Nó thiết lập một kết nối hai chiều giữa các thiết bị trên mạng trước khi truyền dữ liệu, đảm bảo dữ liệu được gửi đến đúng địa chỉ và đến đúng thứ tự. TCP sử dụng các cơ chế như kiểm tra tổng kiểm tra, tái kết nối tự động và điều khiển luồng để đảm bảo dữ liệu được gửi một cách an toàn và tin cậy. Ví dụ về ứng dụng sử dụng TCP bao gồm trình duyệt web (HTTP), truyền tệp (FTP), và gửi email (SMTP).

3.2. UDP (User Datagram Protocol):

UDP là một giao thức không kết nối và không tin cậy. Nó không yêu cầu việc thiết lập kết nối trước khi gửi dữ liệu và không có cơ chế để đảm bảo dữ liệu được gửi đến đúng địa chỉ hoặc đúng thứ tự. UDP được sử dụng cho các ứng dụng cần truyền dữ liệu nhanh chóng và không quan trọng như trò chơi trực tuyến, streaming video, và DNS (Domain Name System). Mặc dù UDP không đảm bảo tin cậy như TCP, nhưng nó có thể hoạt động tốt hơn

trong một số tình huống nhất định như trong các ứng dụng yêu cầu hiệu suất cao và thời gian đáp ứng thấp hơn.

3.3. Tổng quan về TCP và UDP

Nhìn chung , TCP được sử dụng khi cần độ tin cậy cao và đảm bảo dữ liệu được gửi an toàn và đúng đắn, trong khi UDP được sử dụng khi cần tốc độ truyền dữ liệu nhanh chóng và không quan trọng đến việc dữ liệu bị mất hoặc không đến đích.

4.Khái niệm InputStream và OutputStream

Trong Java, InputStream và OutputStream là hai lớp trừu tượng được sử dụng để đọc và ghi dữ liệu từ và đến các nguồn khác nhau, bao gồm tệp tin, mạng, và nhiều nguồn dữ liệu khác.

4.1.InputStream

InputStream là một lớp trừu tượng trong Java, đại diện cho một luồng đầu vào byte. Nó cung cấp một số phương thức để đọc dữ liệu từ nguồn đầu vào, bao gồm phương thức read() để đọc một byte, read(byte[] b) để đọc một mảng byte, và các phương thức khác như skip(long n) để bỏ qua n byte, available() để xác định số byte có sẵn để đọc, v.v. Các lớp con cụ thể của InputStream bao gồm FileInputStream, ByteArrayInputStream, và các lớp khác tùy thuộc vào nguồn dữ liệu cụ thể mà bạn muốn đọc.

4.2.OutputStream

OutputStream là một lớp trừu tượng trong Java, đại diện cho một luồng đầu ra byte. Nó cung cấp các phương thức để ghi dữ liệu byte đến một đích đầu ra, bao gồm phương thức write(int b) để ghi một byte, write(byte[] b) để ghi một mảng byte, và các phương thức khác như flush() để đẩy dữ liệu đã ghi vào đích đầu ra, close() để đóng luồng đầu ra, v.v. Các lớp con cụ thể của OutputStream bao gồm FileOutputStream, ByteArrayOutputStream, và các lớp khác tùy thuộc vào đích đầu ra cụ thể mà bạn muốn ghi.

4.3.Ví dụ về InputStream và OutputStream

```java
import java.io.*;

public class Main {
    public static void main(String[] args) {
        try {
            // Ghi dữ liệu vào tệp tin
            OutputStream outputStream = new FileOutputStream("output.txt");
            outputStream.write("Hello, world!".getBytes());
            outputStream.close();

            // Đọc dữ liệu từ tệp tin
            InputStream inputStream = new FileInputStream("output.txt");
            int data;
            while ((data = inputStream.read()) != -1) {
                System.out.print((char) data);
            }
            inputStream.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Hình 1Ví dụ về Input Stream và OutputStream

Trong ví dụ này, chúng ta sử dụng FileOutputStream để ghi dữ liệu và FileInputStream để đọc dữ liệu từ tệp tin "output.txt".

5.Khái niệm ObjectInputStream và ObjectOutputStream

ObjectInputStream và ObjectOutputStream là hai lớp được sử dụng để đọc và ghi các đối tượng Java dưới dạng byte từ và đến các luồng đầu vào và đầu ra.

5.1.ObjectInputStream:

ObjectInputStream là một lớp con của InputStream và được sử dụng để đọc các đối tượng đã được ghi dưới dạng byte từ một luồng đầu vào. Nó cung cấp các phương thức để đọc các đối tượng từ luồng đầu vào, bao gồm phương thức readObject() để đọc một đối tượng và phương thức readInt(), readDouble(), v.v. để đọc các kiểu dữ liệu nguyên thủy. Để sử dụng

ObjectInputStream, bạn cần tạo một đối tượng ObjectInputStream từ một InputStream, sau đó gọi phương thức readObject() để đọc các đối tượng từ luồng đầu vào.

5.2.ObjectOutputStream

ObjectOutputStream là một lớp con của OutputStream và được sử dụng để ghi các đối tượng Java dưới dạng byte đến một luồng đầu ra. Nó cung cấp các phương thức để ghi các đối tượng vào luồng đầu ra, bao gồm phương thức writeObject() để ghi một đối tượng và phương thức writeInt(), writeDouble(), v.v. để ghi các kiểu dữ liệu nguyên thủy. Để sử dụng ObjectOutputStream, bạn cần tạo một đối tượng ObjectOutputStream từ một OutputStream, sau đó gọi phương thức writeObject() để ghi các đối tượng vào luồng đầu ra.

5.3.Serializable Interface

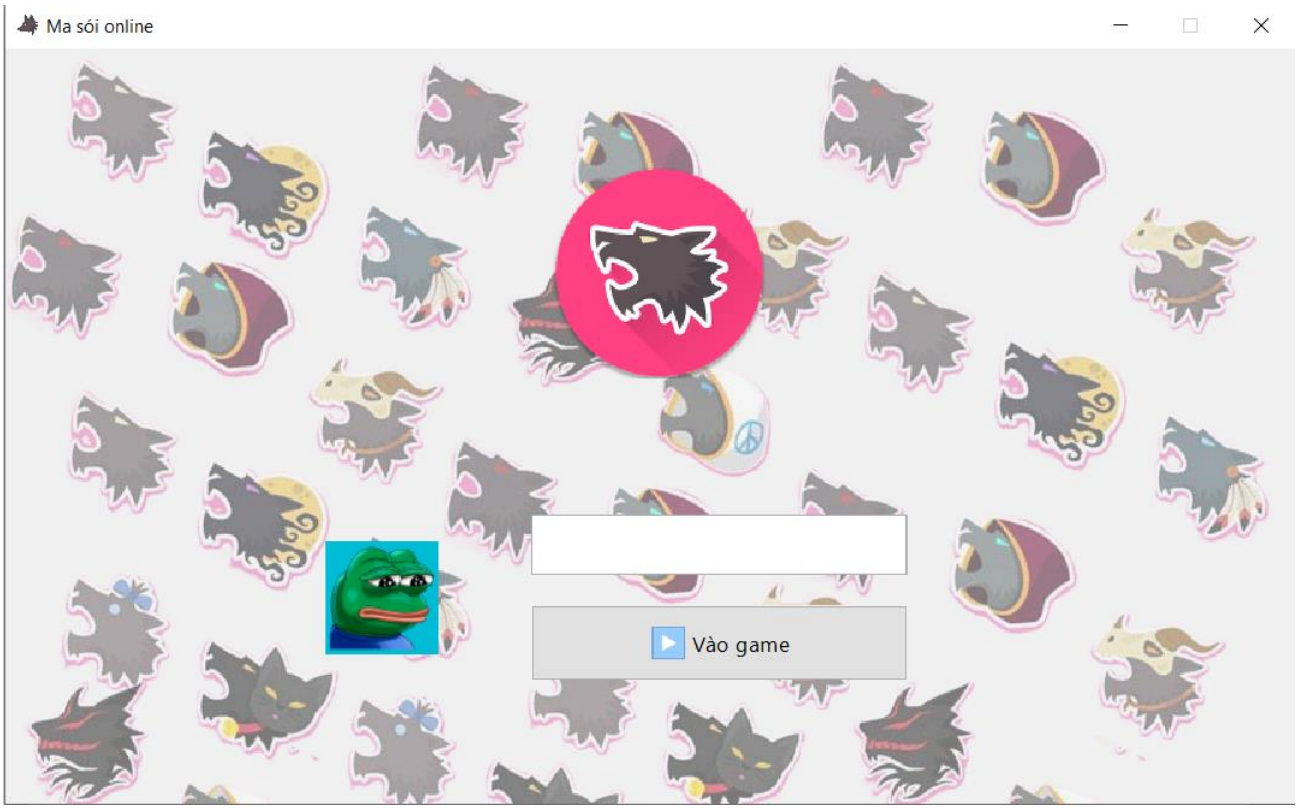
Để sử dụng ObjectInputStream và ObjectOutputStream, các lớp cần triển khai giao diện Serializable. Điều này cho phép các đối tượng của lớp đó được chuyển đổi thành dạng byte để có thể được ghi và đọc từ luồng. Serializable là một giao diện đánh dấu không có phương thức, chỉ đánh dấu lớp là có thể được ghi và đọc.

5.4.Tổng quan

Trên thực tế, khi bạn cần gửi hoặc nhận các đối tượng Java qua mạng hoặc lưu trữ chúng vào tệp tin, bạn có thể sử dụng ObjectOutputStream và ObjectInputStream để thực hiện việc này một cách dễ dàng.

Chương 2. XÂY DỰNG GIAO DIỆN

1.Giao diện đăng ký

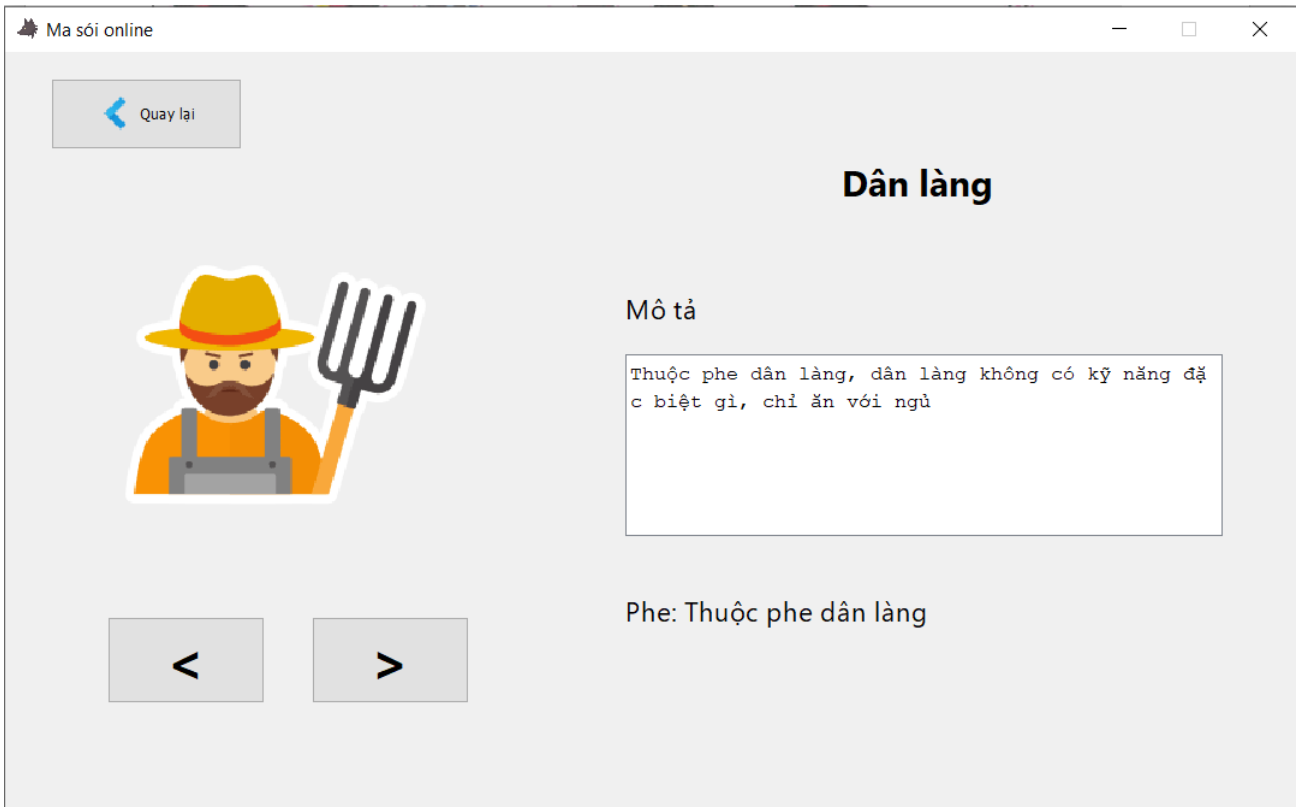


Hình 2 Giao diện đăng ký tên nhân vật

2.Giao diện chính

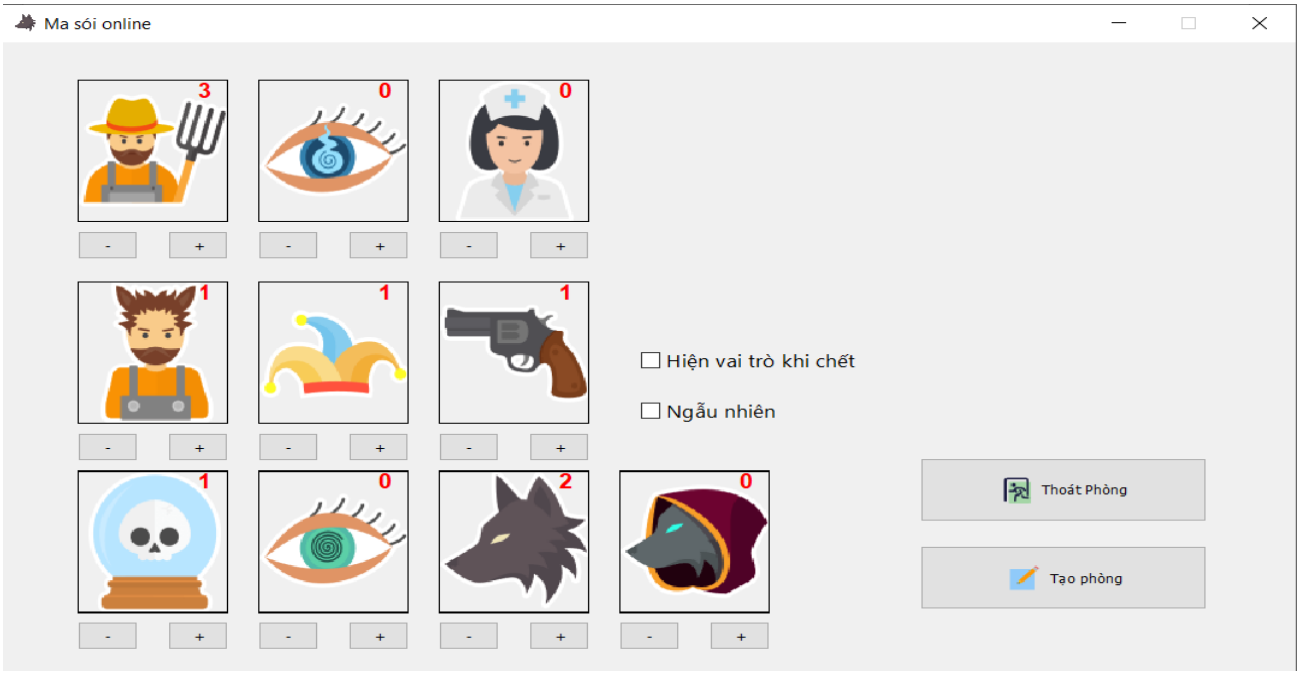
Hình 3 Giao diện chính

3.Giao diện mô tả



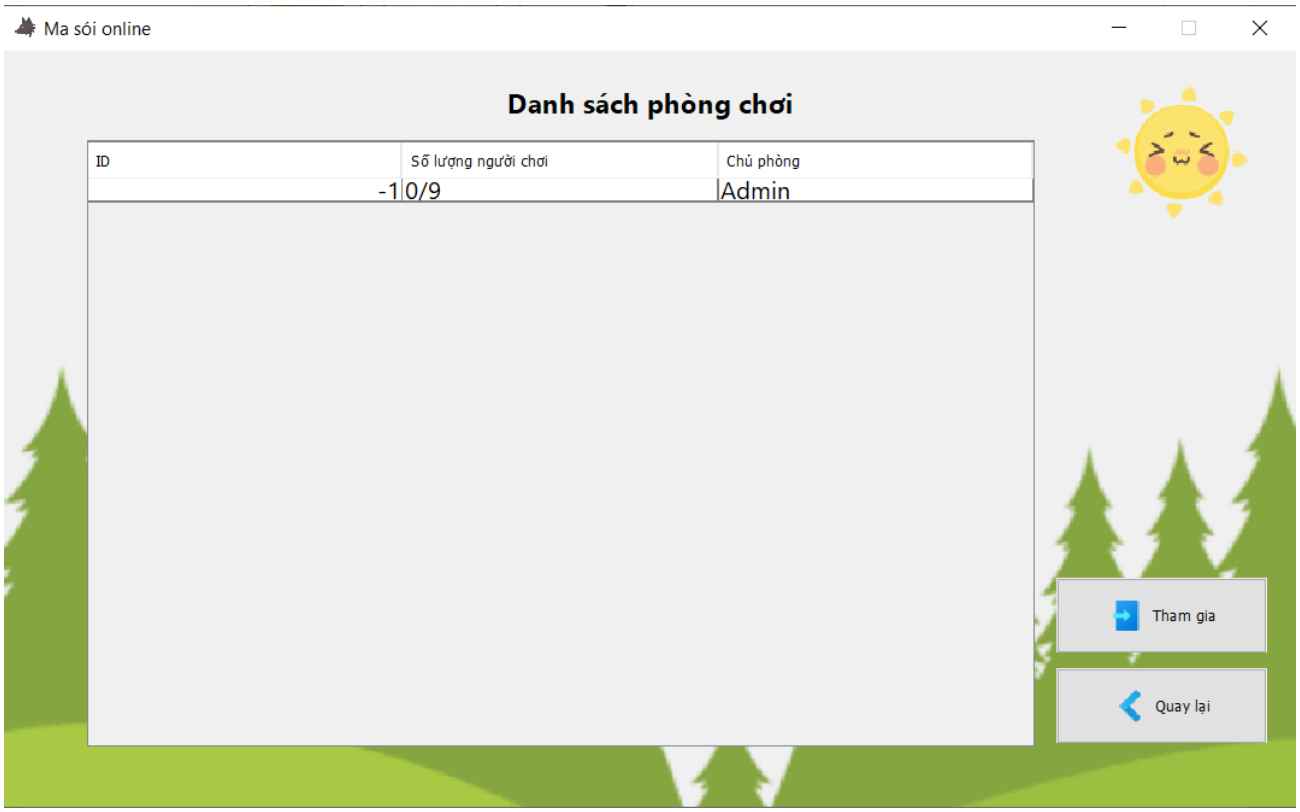
Hình 4 Giao diện mô tả vai trò

4.Giao diện tạo phòng



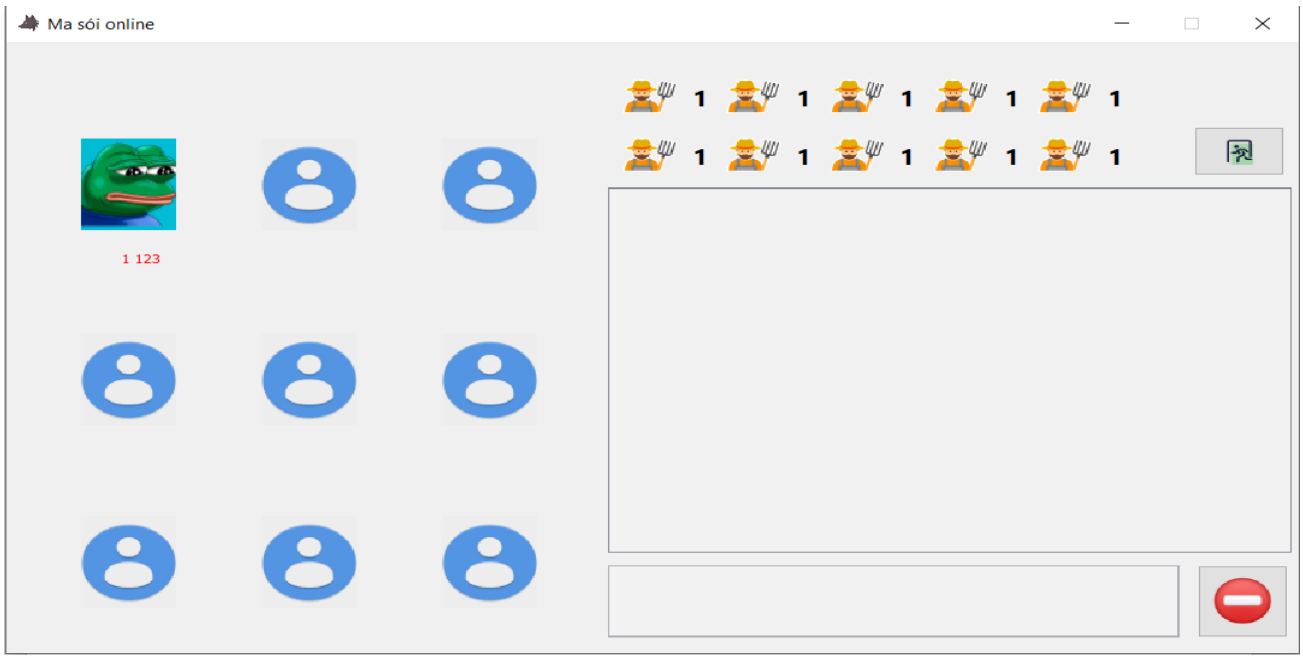
Hình 5 Giao diện tạo phòng trò chơi

5.Giao diện tìm phòng



Hình 6 Giao diện tìm phòng chơi

6.Giao diện chơi game



Hình 7 Giao diện chơi game

Chương 3. XÂY DỰNG HỆ THỐNG

1. Code xử lý phòng

```
case Constaint.MESSAGE_JOIN_SERVER:{
    Player p = (Player)message.getData();
    Player old = PlayerService.gI().getPlayerByUsername(p.namePlayer);
    if(old != null){
        response.setData(null);
    }
    else{
        p.playerEffect = new PlayerEffect();
        PlayerService.gI().addPlayer(p);
        this.player = p;
        this.player.writer = writer;
        response.setData(p);
    }
    writeMessage(response);
    break;
}
case Constaint.MESSAGE_GET_CONFIG_ROLE:{
    response.setData(RoleService.gI().roles);
    writeMessage(response);
    break;
}
case Constaint.MESSAGE_CREATE_ROOM:{
    Room room = (Room)message.getData();
    room.idRoom = Room.id++;
    room.players.add(player);
    room.owner = player;
    response.setData(room);
    player.room = room;
    RoomService.gI().rooms.add(room);
    writeMessage(response);
    break;
}
case Constaint.MESSAGE_LOAD_ROOM:{
    response.setData(RoomService.gI().rooms);
    writeMessage(response);
    break;
```

Hình 8 Ảnh minh họa code

```
110         }
111         case Constaint.MESSAGE_JOIN_ROOM:{
112             int id = (int)message.getData();
113             Room room = RoomService.gI().getRoomById(id);
114             GameController gC = GameController.getGameByRoom(room);
115             if(room != null && gC == null){
116                 room.players.add(player);
117                 player.room = room;
118                 response.setData(room);
119             }
120             else{
121                 response.setData(null);
122             }
123             writeMessage(response);
124             RoomService.gI().reloadPlayer(room);
125             break;
126         }
127         case Constaint.ADD_BOT:{
128             Room room = RoomService.gI().getRoomById(player.room.idRoom);
129             if(room != null && room.owner.namePlayer.equals(player.namePlayer)){
130                 int numBot = 9 - room.players.size();
131                 for(int i = 0; i < numBot; i++){
132                     Player p = PlayerService.gI().getBot(i);
133                     room.players.add(p);
134                     p.room = room;
135                 }
136                 RoomService.gI().reloadPlayer(room);
137             }
138             break;
139         }
140         case Constaint.MESSAGE_LEAVE_ROOM:{
141             Room room = RoomService.gI().getRoomById(player.room.idRoom);
142             if(room != null){
143                 RoomService.gI().leavedPlayer(player, room);
144                 RoomService.gI().reloadPlayer(room);
145             }
146             break;
```

Hình 9 Ảnh minh họa code

Đoạn mã trên là một phần của một ứng dụng trò chơi mạng sử dụng giao thức TCP/IP để truyền dữ liệu giữa máy chủ và các máy khách. Dưới đây là phân tích từng phần của đoạn mã:

Case Constaint.MESSAGE_JOIN_SERVER: Khi một máy khách muốn tham gia vào máy chủ, máy khách gửi thông điệp có kiểu là Constaint.MESSAGE_JOIN_SERVER và chứa thông tin về người chơi (Player) muốn tham gia. Máy chủ nhận thông điệp này và kiểm tra xem người chơi đã tồn tại hay chưa bằng cách kiểm tra trong danh sách các người chơi đã tham gia trước đó. Nếu người chơi đã tồn tại, máy chủ gửi lại thông điệp với dữ liệu là null, ngược lại, máy chủ tạo một Player mới, thêm vào danh sách các người chơi và gửi lại thông điệp với dữ liệu là thông tin của người chơi mới này.

Case Constaint.MESSAGE_GET_CONFIG_ROLE: Khi máy khách muốn nhận cấu hình về các vai trò trong trò chơi, máy khách gửi thông điệp với kiểu là Constaint.MESSAGE_GET_CONFIG_ROLE. Máy chủ nhận thông điệp này và gửi lại thông điệp với dữ liệu là cấu hình về các vai trò.

Case Constaint.MESSAGE_CREATE_ROOM: Khi một người chơi muốn tạo phòng mới, máy khách gửi thông điệp với kiểu là Constaint.MESSAGE_CREATE_ROOM và chứa thông tin về phòng (Room) muốn tạo. Máy chủ nhận thông điệp này, tạo một phòng mới, thêm người chơi tạo phòng vào danh sách các người chơi trong phòng và gửi lại thông điệp với dữ liệu là thông tin của phòng mới tạo.

Case Constaint.MESSAGE_LOAD_ROOM: Khi máy khách muốn tải danh sách các phòng có sẵn, máy khách gửi thông điệp với kiểu là Constaint.MESSAGE_LOAD_ROOM. Máy chủ nhận thông điệp này và gửi lại thông điệp với dữ liệu là danh sách các phòng.

Case Constaint.MESSAGE_JOIN_ROOM: Khi một người chơi muốn tham gia vào một phòng, máy khách gửi thông điệp với kiểu là Constaint.MESSAGE_JOIN_ROOM và chứa ID của phòng mà người chơi muốn tham gia. Máy chủ nhận thông điệp này, kiểm tra xem phòng tồn tại và chưa bắt đầu trận đấu nào, nếu đúng, thì thêm người chơi vào phòng và gửi lại thông điệp với dữ liệu là thông tin của phòng đã tham gia.

Case Constaint.ADD_BOT: Khi chủ phòng muốn thêm bot vào phòng, máy khách gửi thông điệp với kiểu là Constaint.ADD_BOT. Máy chủ nhận thông điệp này, kiểm tra xem người gửi có phải là chủ phòng không, nếu đúng, thì thêm số lượng bot cần thiết vào phòng và cập nhật danh sách người chơi trong phòng.

Case Constaint.MESSAGE_LEAVE_ROOM: Khi một người chơi muốn rời khỏi phòng, máy khách gửi thông điệp với kiểu là Constaint.MESSAGE_LEAVE_ROOM. Máy

chủ nhận thông điệp này, loại bỏ người chơi ra khỏi phòng và cập nhật danh sách người chơi trong phòng.

2. Code xử lý game

```
149                 //Note
150                 case Constaint.MESSAGE_START_GAME:{
151                     if(this.player.room.owner.namePlayer.equals(this.player.namePlayer)){
152                         this.player.room.startedGame = true;
153                         Game game = GameController.initNewGame(this.player.room);
154                         response.setData(game);
155                         MessageService.gI().sendMessageInRoom(this.player.room, response);
156                     }
157                     break;
158                 }
159                 case Constaint.MESSAGE_CHAT:{
160                     MessageService.gI().chat(this.player, (String)message.getData());
161                     break;
162                 }
```

Hình 10 Ảnh minh họa code

Case Constaint.MESSAGE_START_GAME: Trong trường hợp này, khi máy chủ nhận được một tin nhắn với mã thông điệp là Constaint.MESSAGE_START_GAME, nó kiểm tra xem người chơi hiện tại có phải là chủ sở hữu của phòng không. Nếu đúng, nó sẽ đánh dấu trò chơi trong phòng đã bắt đầu bằng cách thiết lập cờ startedGame của phòng thành true. Sau đó, máy chủ tạo một trò chơi mới thông qua GameController.initNewGame(this.player.room) và gửi lại thông điệp này với dữ liệu của trò chơi tới tất cả các người chơi trong phòng sử dụng MessageService.gI().sendMessageInRoom(this.player.room, response).

Case Constaint.MESSAGE_CHAT: Trong trường hợp này, khi máy chủ nhận được một tin nhắn với mã thông điệp là Constaint.MESSAGE_CHAT, nó sẽ gửi nội dung của tin nhắn này đến MessageService để xử lý. Tin nhắn này thường là tin nhắn chat của người chơi và sẽ được gửi đến tất cả các người chơi khác trong cùng phòng.

```
478                 case Constaint.MESSAGE_CHAT_IN_GAME:{
479                     String cont = player.namePlayer + ": " + message.getData();
480                     Message msg = new Message(Constaint.MESSAGE_CHAT, cont);
481                     //Người chết nói chuyện với nhau
482                     if(gameState == Constaint.STAGE_SLEEPING && (player.isDie || player.playerEffect.isThayDong)){
483                         if(player.playerEffect.isThayDong && !player.isDie){
484                             msg.setData((String)"Thầy đồng: " + message.getData());
485                         }
486                         else{
487                             msg.setData("[HELL] " + cont);
488                         }
489                         MessageService.gI().sendMessageForTeam(game.players, msg,Constaint.TEAM_HELL, true);
490                     }
491                     //Sói chat với nhau vào ban đêm
492                     else if(gameState == Constaint.STAGE_SLEEPING && player.playerEffect.isWolfTeam()){
493                         msg.setData("[WOLF] " + cont);
494                         MessageService.gI().sendMessageForTeam(game.players, msg,Constaint.TEAM_WOLF, false);
495                     }
496                     //Chat ban ngày bình thường
497                     else{
498                         MessageService.gI().sendMessageInRoom(game, msg);
499                     }
```

Hình 11 Ảnh minh họa code

```
503             case Constaint.MESSAGE_PLAYER_VOTES:{
504                 String targetVote = (String)message.getData();
505                 if(gameState == Constaint.STAGE_VOTING && !player.isDie){
506                     Player pTarget = getPlayerByUsername(targetVote);
507                     //Nếu trước đó có vote cho ai khác thì gỡ đi
508                     if(player.playerVote.target != null){
509                         player.playerVote.target.playerVote.voteCount-=1;
510                     }
511                     player.playerVote.target = pTarget;
512                     pTarget.playerVote.voteCount ++;
513                     updateStateVoting(game.players);
514                 }
515                 break;
516             }
517             //Done
518             case Constaint.MESSAGE_PLAYER_CANCEL_VOTES:{
519                 String targetVote = (String)message.getData();
520                 if(gameState == Constaint.STAGE_VOTING){
521                     Player pTarget = getPlayerByUsername(targetVote);
522                     if(pTarget != null){
523                         player.playerVote.target = null;
524                         pTarget.playerVote.voteCount -=1 ;
525                     }
526                     updateStateVoting(game.players);
527                 }
528                 break;
529             }
```

Hình 12 Ảnh minh họa code

```
531             case Constaint.MESSAGE_XATHU_SHOOT:{
532                 Player target = getPlayerByUsername((String)message.getData());
533                 //Kiểm tra đạn và vai trò
534                 if(player.playerEffect.isXaThu && player.playerEffect.bullet > 0 && !target.isDie){
535                     //Thực hiện hành động của player
536                     player.playerEffect.bullet--;
537                     player.playerEffect.isShowRole = true;
538                     //Hủy vote của đối tượng bị bắn và setDie
539                     if(target.playerVote.target != null){
540                         target.playerVote.target.playerVote.voteCount -= 1;
541                         target.playerVote.target = null;
542                     }
543                     target.playerVote.voteCount = 0;
544                     //Hủy vote của mọi người vào đối tượng bị bắn
545                     for(var p : game.players){
546                         if(p.playerVote.target == target){
547                             p.playerVote.target = null;
548                         }
549                     }
550                     //Set đã shoot trong ngày
551                     game.isShootSameDay = true;
552                     //Update state cho các player khác
553                     String alert = String.format("[Server]: Xạ thủ %s đã bắn chết %s",
554                             player.namePlayer, target.namePlayer);
555                     MessageService.gI().sendMessageInRoom(game,
556                             new Message(Constaint.MESSAGE_CHAT, alert));
557                     showRolePlayer(player);
558                     setDiePlayer(target);
559                     updateStateVoting(game.players);
560                 }
561                 //Hết đạn
562                 if(player.playerEffect.isXaThu && player.playerEffect.bullet == 0){
563                     MessageService.gI().sendMessagePrivate(player,
564                             new Message(Constaint.MESSAGE_XATHU_OUT_OF_BULLET, null));
565                 }
566                 break;
```

Hình 13 Ảnh minh họa code

```
569             case Constaint.MESSAGE_WOLF_VOTES:{
570                 String targetVote = (String)message.getData();
571                 if(player.game.gameState == Constaint.STAGE_SLEEPING){
572                     Player pTarget = getPlayerByUsername(targetVote);
573                     //Nếu trước đó có vote cho ai khác thì gỡ đi
574                     if(player.playerVote.target != null){
575                         player.playerVote.target.playerVote.voteCount -=1;
576                     }
577                     player.playerVote.target = pTarget;
578                     pTarget.playerVote.voteCount+=1;
579                     updateStateVotingWolfs(game.players);
580                 }
581                 break;
582             }
583             //Done
584             case Constaint.MESSAGE_WOLF_CANCEL_VOTES:{
585                 String targetVote = (String)message.getData();
586                 if(player.game.gameState == Constaint.STAGE_SLEEPING){
587                     Player pTarget = getPlayerByUsername(targetVote);
588                     if(pTarget != null){
589                         player.playerVote.target = null;
590                         pTarget.playerVote.voteCount -=1 ;
591                     }
592                     updateStateVotingWolfs(game.players);
593                 }
594                 break;
595             }
596             //Done
597             case Constaint.MESSAGE_TIENTRI_SEE:{
598                 Player pTarget = RoomService.gI().getPlayerByUsername(player.room, (String)message.getData());
599                 if(!pTarget.isDie){
600                     MessageService.gI().sendMessagePrivate(player,
601                             new Message(Constaint.MESSAGE_TIENTRI_SEE, pTarget.playerEffect.idRole));
602                 }
603                 break;
604             }
```

Hình 14 Ảnh minh họa code

```
606             case Constaint.MESSAGE_THAYBOI_SEE:{
607                 Player pTarget = RoomService.gI().getPlayerByUsername(player.room, (String)message.getData());
608                 if(!pTarget.isDie){
609                     PlayerEffect pE = pTarget.playerEffect;
610                     byte teamTarget = pE.isWolf() ? Constaint.TEAM_WOLF : pE.isUnknown() ? Constaint.TEAM_THIRD : Constain
611                     MessageService.gI().sendMessagePrivate(player,
612                             new Message(Constaint.MESSAGE_THAYBOI_SEE, teamTarget));
613                 }
614                 break;
615             }
616             //Done
617             case Constaint.MESSAGE_BACSI_BAOVE:{
618                 String targetVote = (String)message.getData();
619                 if(gameState == Constaint.STAGE_SLEEPING){
620                     for(var p: game.players){
621                         p.playerEffect.duocBaoVe = false;
622                     }
623                     Player pTarget = getPlayerByUsername(targetVote);
624                     pTarget.playerEffect.duocBaoVe = true;
625                     MessageService.gI().sendMessagePrivate(player,
626                             new Message(Constaint.MESSAGE_BACSI_BAOVE, pTarget.namePlayer));
627                 }
628                 break;
629             }
630             //Done
631             case Constaint.MESSAGE_BACSI_HUY_BAOVE:{
632                 if(gameState == Constaint.STAGE_SLEEPING){
633                     for(var p: game.players){
634                         p.playerEffect.duocBaoVe = false;
635                     }
636                     MessageService.gI().sendMessagePrivate(player,
637                             new Message(Constaint.MESSAGE_BACSI_BAOVE, null));
638                 }
639                 break;
640             }
```

Hình 15 Ảnh minh họa code

```
642             case Constaint.MESSAGE_THAYDONG_HOISINH:{
643                 Player pRevival = getPlayerByUsername((String)message.getData());
644                 if(player.playerEffect.isThayDong && player.playerEffect.revivalTime > 0){
645                     player.playerEffect.revivalTime -= 1;
646                     game.lastPlayerRevival = pRevival;
647                 }
648                 break;
649             }
650
651             //Done
652             case Constaint.MESSAGE_SOITIENTRI_SEE:{
653                 Player pTarget = RoomService.gI().getPlayerByUsername(player.room, (String)message.getData());
654                 if(!pTarget.isDie){
655                     Message msg =  new Message(Constaint.MESSAGE_SOITIENTRI_SEE, pTarget);
656                     msg.setTmp(pTarget.playerEffect.idRole);
657                     MessageService.gI().sendMessagePrivate(player.game.teamWolf,
658                             msg);
659                 }
660                 break;
661             }
662         }
663     }
664 }
665
```

Hình 16 Ảnh minh họa code

Constaint.MESSAGE_CHAT_IN_GAME: Trong trường hợp này, khi server nhận được một tin nhắn chat trong trò chơi từ client, nó tạo một chuỗi tin nhắn mới với tên của người chơi kèm theo nội dung tin nhắn. Tin nhắn này được gửi tới MessageService để xử lý. Nếu trò chơi đang ở trạng thái ngủ (đêm) và người chơi là người chơi đã chết hoặc là thầy đồng, tin nhắn sẽ được gửi tới nhóm Hell (nếu là thầy đồng) hoặc nhóm Hell (nếu là người chơi đã chết). Nếu là sói và đang vào ban đêm, tin nhắn sẽ được gửi tới nhóm sói. Trong trường hợp khác, tin nhắn sẽ được gửi tới tất cả các người chơi trong phòng.

Constaint.MESSAGE_PLAYER_VOTES: Khi một người chơi gửi một phiếu bầu cho một người chơi khác, server sẽ kiểm tra xem trạng thái của trò chơi có phải là giai đoạn bầu cử hay không, và người chơi không phải là người chơi đã chết. Nếu có, server sẽ cập nhật phiếu bầu của người chơi và cập nhật trạng thái bầu cử.

Constaint.MESSAGE_PLAYER_CANCEL_VOTES: Trong trường hợp này, người chơi gửi một tin nhắn để hủy phiếu bầu cho một người chơi khác. Server sẽ kiểm tra xem trạng thái của trò chơi có phải là giai đoạn bầu cử không, và nếu có, nó sẽ hủy phiếu bầu của người chơi.

Constaint.MESSAGE_XATHU_SHOOT: Khi một xạ thủ gửi một tin nhắn để bắn một người chơi khác, server sẽ kiểm tra xem xạ thủ có đủ đạn để bắn không, và nếu đủ, nó sẽ thực hiện hành động bắn và cập nhật trạng thái của người chơi đã bị bắn.

Constaint.MESSAGE_WOLF_VOTES: Khi một sói gửi một phiếu bầu cho một người chơi khác, server sẽ kiểm tra xem trạng thái của trò chơi có phải là giai đoạn ngủ (đêm) không, và nếu có, nó sẽ cập nhật phiếu bầu của sói và cập nhật trạng thái bầu cử cho sói.

Constaint.MESSAGE_WOLF_CANCEL_VOTES: Trong trường hợp này, sói gửi một tin nhắn để hủy phiếu bầu cho một người chơi khác. Server sẽ kiểm tra xem trạng thái của trò chơi có phải là giai đoạn ngủ (đêm) không, và nếu có, nó sẽ hủy phiếu bầu của sói.

Constaint.MESSAGE_TIENTRI_SEE: Khi một tiên tri gửi một tin nhắn để xem vai trò của một người chơi khác, server sẽ gửi kết quả này cho người chơi tiên tri.

Constaint.MESSAGE_THAYBOI_SEE: Trong trường hợp này, thầy bói gửi một tin nhắn để xem nhóm của một người chơi khác, server sẽ gửi kết quả này cho người chơi thầy bói.

Constaint.MESSAGE_BACSI_BAOVE: Trong trường hợp này, khi máy chủ nhận được một tin nhắn với mã thông điệp là Constaint.MESSAGE_BACSI_BAOVE, nó kiểm tra xem trò chơi đang ở giai đoạn ngủ không. Nếu đúng, nó sẽ lặp qua tất cả các người chơi trong trò chơi và đặt thuộc tính duocBaoVe của mỗi người chơi thành false, tức là không ai được bảo vệ. Sau đó, nó tìm người chơi mục tiêu (được chỉ định bởi dữ liệu tin nhắn) và đặt thuộc tính duocBaoVe của họ thành true, cho biết họ đang được bảo vệ. Cuối cùng, nó gửi một tin nhắn riêng tư cho người chơi gửi tin nhắn ban đầu, thông báo rằng họ đang được bảo vệ bởi một bác sĩ.

Constaint.MESSAGE_BACSI_HUY_BAOVE: Trong trường hợp này, nếu trò chơi đang ở giai đoạn ngủ, máy chủ sẽ lặp qua tất cả các người chơi trong trò chơi và đặt thuộc tính duocBaoVe của họ thành false, tức là không ai được bảo vệ. Sau đó, nó gửi một tin nhắn riêng tư cho người chơi gửi tin nhắn ban đầu, thông báo rằng không có ai đang được bảo vệ bởi bác sĩ.

Constaint.MESSAGE_THAYDONG_HOISINH: Trong trường hợp này, nếu người chơi đang chơi vai trò thầy đồng và vẫn còn thời gian hồi sinh, máy chủ sẽ giảm đi 1 đơn vị từ thời gian hồi sinh của người chơi.

Constaint.MESSAGE_SOITIENTRI_SEE: Trong trường hợp này, nếu mục tiêu của tin nhắn chưa chết, máy chủ sẽ gửi một tin nhắn riêng tư cho người chơi gửi tin nhắn ban đầu, thông báo vai trò của mục tiêu .Tin nhắn này được gửi chỉ đến nhóm sói.

Chương 4. Ý TƯỞNG ỨNG DỤNG, PHÁT TRIỂN

1.Ý tưởng ứng dụng

Trong phát triển ứng dụng game bạn có thể sử dụng nhiều công nghệ và thư viện khác nhau để tạo ra một ứng dụng đa nền tảng và mạnh mẽ và có thể nói đến như sau :

+ Game Ma Sói Đa Nền Tảng: Phát triển một ứng dụng game ma sói có thể chạy trên nhiều nền tảng khác nhau, bao gồm desktop, web và di động. Sử dụng Java để xây dựng ứng dụng giúp đơn giản hóa việc phát triển đa nền tảng và cung cấp một trải nghiệm chơi game đồng nhất trên các thiết bị khác nhau.

+ Giao Diện Người Dùng Thân Thiện: Thiết kế giao diện người dùng đẹp mắt và thân thiện giúp người chơi dễ dàng tương tác và tham gia vào trò chơi. Giao diện nên được tối ưu hóa để phản ánh các tình huống trong trò chơi một cách rõ ràng và dễ hiểu.

+ Hệ Thống Tương Tác Trực Tuyến: Xây dựng một hệ thống tương tác trực tuyến cho phép người chơi kết nối và tham gia vào các trận đấu với người chơi khác trên toàn thế giới. Hệ thống này cũng cung cấp các tính năng như chat, tạo phòng chơi, mời bạn bè và quản lý người chơi.

+ Tính Năng Mở Rộng Vai Trò và Quy Tắc: Tạo ra các bản cập nhật định kỳ để mở rộng các vai trò và quy tắc trong trò chơi. Điều này giúp tạo ra sự đa dạng và thú vị trong trải nghiệm chơi game và giữ cho người chơi luôn cảm thấy hứng thú.

2.Ý tưởng phát triển

Trong phát triển game , chúng ta có thể sử dụng nhiều công nghệ và thư viện khác nhau để tạo ra một ứng dụng đa nền tảng và mạnh mẽ có thể nói đến như sau :

+ Sử Dụng Java Swing hoặc JavaFX: Sử dụng Java Swing hoặc JavaFX để xây dựng giao diện người dùng đồ họa cho trò chơi. Hai thư viện này cung cấp các thành phần giao diện phong phú và dễ sử dụng để tạo ra các cửa sổ, nút bấm, và các phần tử khác cho ứng dụng của bạn.

+ Xử Lý Logic Trò Chơi: Sử dụng Java để xử lý logic của trò chơi, bao gồm quản lý người chơi, phân phối vai trò, thực hiện các hành động của người chơi và tính toán kết quả của mỗi vòng chơi.

+ Kết Nối Mạng và Tương Tác Trực Tuyến: Sử dụng Java để xây dựng một hệ thống mạng để kết nối người chơi với nhau và cho phép họ tương tác trong trò chơi. Điều này bao

gồm việc xử lý kết nối mạng, truyền dữ liệu giữa các máy chủ và máy khách, và đồng bộ hóa trạng thái trò chơi giữa các máy chủ và máy khách.

+ Lưu Trữ Dữ Liệu: Sử dụng Java để lưu trữ dữ liệu như thông tin người chơi, lịch sử trò chơi và cấu hình trò chơi. Có thể sử dụng các công nghệ như cơ sở dữ liệu SQL hoặc NoSQL để lưu trữ dữ liệu này.

+ Kiểm Tra và Xử Lý Lỗi: Viết mã Java một cách chắc chắn và kiểm tra lỗi thường xuyên để đảm bảo tính ổn định và bảo mật của ứng dụng của bạn. Sử dụng các công cụ như log và xử lý ngoại lệ để ghi nhận và xử lý lỗi một cách hiệu quả.

+ Kiểm Tra và Tối Ưu Hóa Hiệu Năng: Kiểm tra và tối ưu hóa hiệu năng của ứng dụng của bạn để đảm bảo rằng nó hoạt động mượt mà và ổn định trên mọi nền tảng và điều kiện mạng. Sử dụng các công cụ như profiler để xác định và loại bỏ các điểm yếu trong mã của bạn.

Chương 5. KẾT LUẬN

Trò chơi Ma Sói là một trò chơi trí tuệ phổ biến và hấp dẫn, đã thu hút được một lượng lớn người chơi trên toàn thế giới. Bằng cách kết hợp yếu tố chiến lược, tài năng diễn xuất và sự phán đoán, trò chơi này tạo ra những trải nghiệm độc đáo và thú vị cho người chơi. Trong phiên bản Java, trò chơi Ma Sói cung cấp một nền tảng linh hoạt và dễ dàng tiếp cận, cho phép người chơi tham gia vào các cuộc phiêu lưu ma thuật ngay trên máy tính của mình.

Trò chơi Ma Sói giúp kết nối cộng đồng người chơi thông qua mạng trực tuyến, tạo ra một không gian tương tác đầy sôi động và sự hòa mình vào thế giới ma quái. Với sự linh hoạt trong cách chơi và cách tương tác, người chơi có thể tham gia vào các trận đấu đầy kịch tính, trải qua những màn chơi đầy thách thức và kịch tính.

Dù làm theo quy tắc cổ điển hoặc thêm vào những biến thể mới, trò chơi Ma Sói vẫn giữ được sức hút và sức sống của mình. Đây không chỉ là một trò chơi giải trí, mà còn là một cách tuyệt vời để kích thích tư duy chiến lược, phản xạ nhanh và khả năng phán đoán của người chơi. Chính vì vậy, trò chơi Ma Sói không chỉ là một trải nghiệm giải trí, mà còn là một cách để kết nối và giao lưu với cộng đồng game thủ trên khắp thế giới.

DANH MỤC TÀI LIỆU THAM KHẢO

https://www.w3schools.com/java/java_intro.asp (Data., 1990 - 2024)

https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html

https://aws.amazon.com/what-is/java/

https://t3h.com.vn/tin-tuc/lap-trinh-socket-trong-java

https://shareprogramming.net/outputstream-trong-java/

https://openplanning.net/13543/java-outputstream