System Programming

Com1564 Coursework

Student Name: Duong Bui Dinh

Student Roll Number: GT00010

Instructor: Hans Anderson

Contents

1. Introduction

Hangman Game is game software which allows many players to interact over the local area network or internet. The game software has client-server architecture.

How the game works:

- Each two players play with each others, they have to find a hidden word to finish the game.
- One player who starts the game first guesses a character, if the hidden word contains the character then this player can continue finding word and the character will be displayed. Opposite, if the word doesn't contain the character, the second player can play game.
- Who find the hidden word with all characters displayed first, he/she is the winner.

2. Analysis and Design

2.1 System architecture

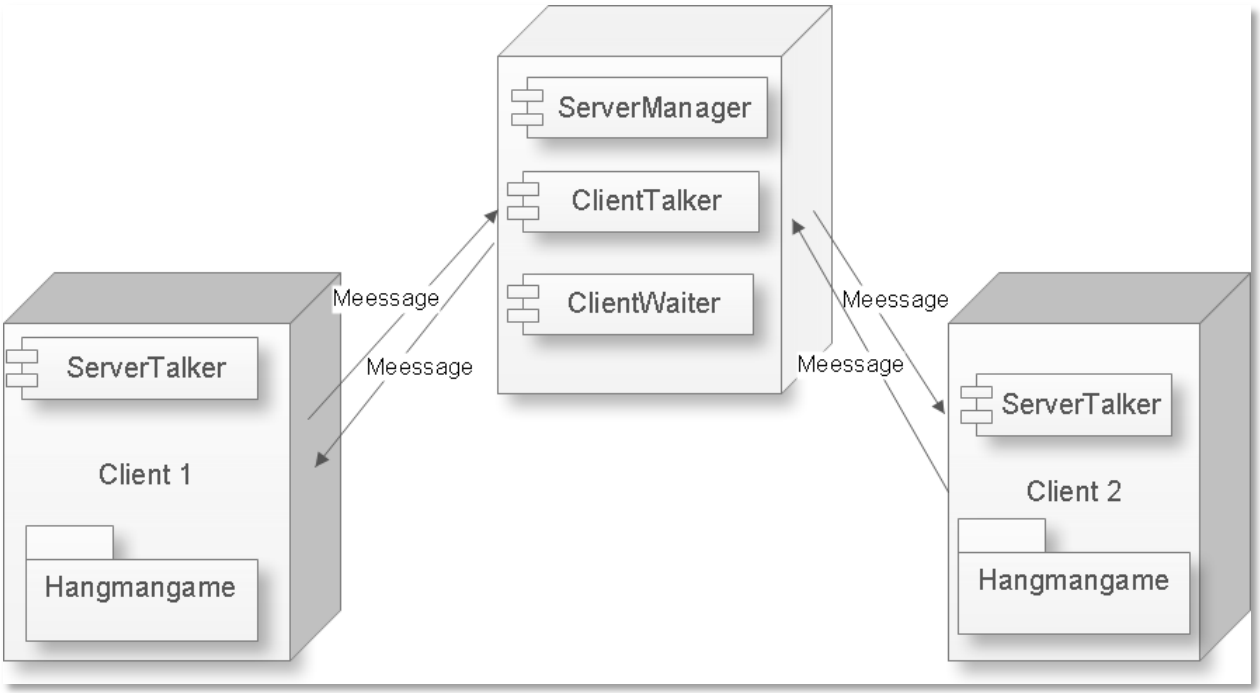

Figure 01: System architecture

- **ClientWaiter** always wait for client message and inform to **ServerManager.**
- When each client comes, the **ServerManager** create new **ClientTalker** thread to send and receive message with the client.
- **ServerTalker** and **ClientTalker** talk with each others to exchange message.
- **Hangmangame** package controls game in client. It includes : **GameRoom** and **ClientManager** class

2.2 Supported functions

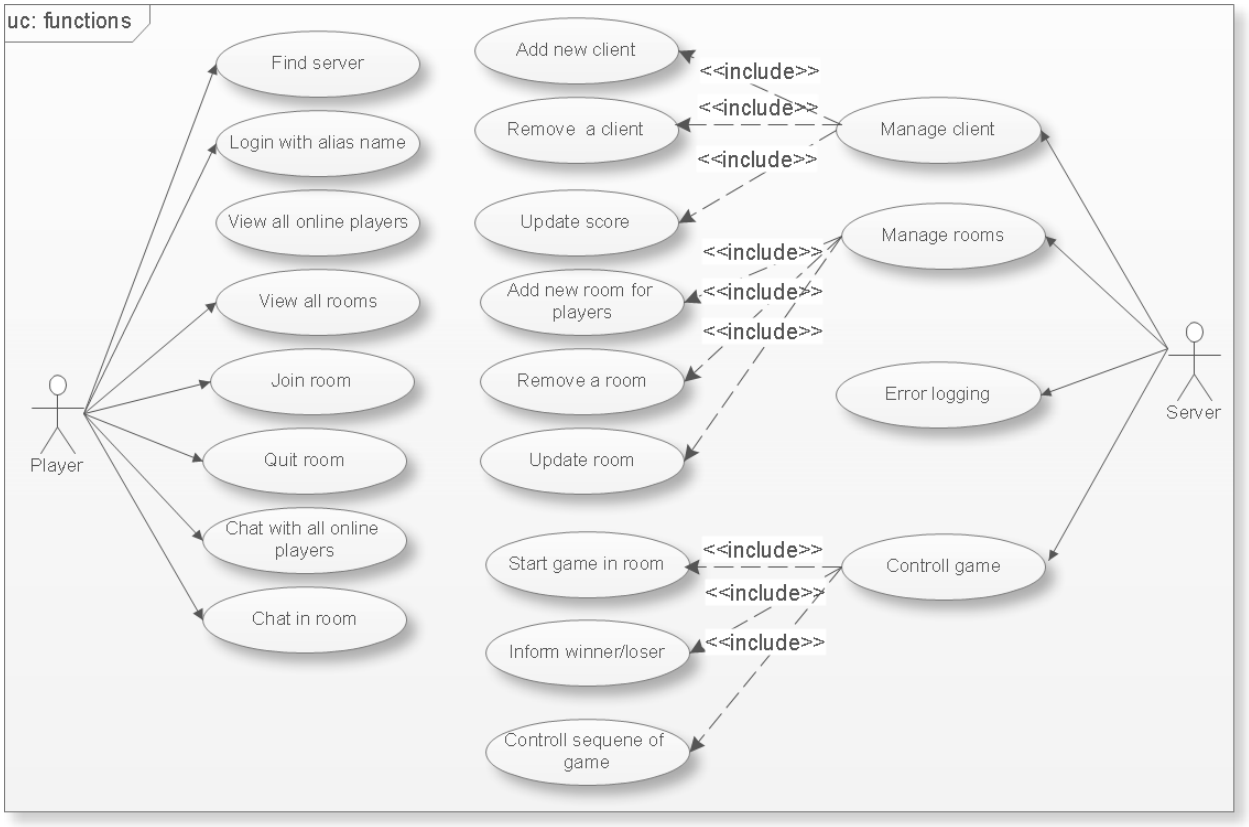

Figure 02: Use case diagram

- **Player functions**
  - **Find server in LAN (Using UDP)**

When user runs the client program, the client application can automatically find the server in LAN.

- Client broadcast a UDP data package to all computer in LAN (Port is fix), the data include: IP and Port of client.
- After that, Client waits for UDP data send back from server.
- Server always listens for UDP data.
- If server receive UDP data from client, send the server IP to client

  - **Login to server with alias Name:**
  - User has to enter a name to connect server. The client application will send the name to server. the server manages all names received from all clients
  - **View all online players**

After logging in to server, the server sends an online player list to client. The client application will display the list to user.

  - **Join a room to play game**

After logging in to server, the server sends a room list to client. The client application will display the room list to user. Each room has maximum two players; the user can join any room which has one player inside. If a room has two players, then the room can start playing game.

- **Chat with all online players**

After logging in to server, the client can chat with all online players.

- **Chat with competitor player**

After logging in to server, the client can chat with all online players.

- **Server functions:**
  - **Error Logging**
  - The server has to track all errors when running.
  - **Manage all client**
  - The server has to manage all client, one client includes following information:
    - Player information (Name, ID)
    - Socket information
  - Add Client: When client connected, add client
  - Remove: When client disconnected, remove client
  - **Manage rooms**

The server can calculate the number of rooms for all clients.

Add Room: When client connect, if lack of room, add new room.

Remove room: When client disconnect, remove all unnecessary rooms.

Update room: update number of players in room.

- **Control sequence of game**

Any room has two players, the room can start playing. The player who joins the room first can start first.

The Plays play with the rule of game until the hidden word is found.

Each time players guess a character of hidden word:

- Client processes the word, if the word contains the character then set the state of character is "1". By default all characters of hidden word is "0".
- Send the processed word to server.
- Server can check the received word again, compares it with the word in server. If there is any change, then the client who sends the word can continue playing game and server update the word, the server also send the updated word to client . Or not, the second player can start the game.

2.3 Protocol Data Unit

The message passes between client and server using TCP/IP. They must have following format:

- Header: includes message type, from Player, to Player
- Body: message contents

Field	Data type	Size (bit)	Using
Message_type	Int	32	Determine kind of message to process
From_PlayerID	Int	32	Determine sender
To_PlayerID	Int	32	Determine receiver
Body	Object	Non-fix	Content of message.

3. Implementation

3.1 Key Program Listing

- Send broadcast message

```
public void sendBroadcastUDPData(int port)
{
    try {
        //create data gram socket
        DatagramSocket socket = new DatagramSocket();
        //get local IP address
        String myIP=InetAddress.getLocalHost().getHostAddress();
        //send local IP address and port to server
        byte[] dataToSend=(myIP+":1234").getBytes();
        byte[] addressByte={(byte)255,(byte)255,(byte)255,(byte)255};
        InetAddress address=InetAddress.getByAddress(addressByte);
        DatagramPacket dataPacket=new DatagramPacket(dataToSend,dataToSend.length, address, port);
        socket.send(dataPacket);
    }
    catch(Exception ex)
    {
      System.out.print("err: "+ex.getMessage());
    }

}
```

- Wait for server send back the IP

```
@Override
public void run() {
    //listen to UDP data from server
    while (true) {
        try {
            //clear destination IP address
            destinationIP="";
            loginForm.getjTextField_server().setText("");
            //create data socket
            DatagramSocket socket = (new DatagramSocket(port));
            //create data lenght to store received data
            byte[] buf = new byte[InetAddress.getLocalHost().getAddress().length];
            DatagramPacket paack = new DatagramPacket(buf, buf.length);
            //get data
            socket.receive(paack);
            byte[] serverIP=new String(buf, 0, paack.getLength()).getBytes();
            //show server IP
            if(serverIP!=null&&serverIP.length>0){
            loginForm.getjTextField_server().setText(getIpAddressFromByte(serverIP));
            socket.close();
            this.stop();
            return;

            }
            socket.close();

        } catch (IOException ex) {
            port++;
         System.out.print(ex.getMessage());
        }

    }
```

- Server wait for client connect

```
public void waitClient() {

    try {
         ser_socket = new ServerSocket(setting.getMainSocketPort());
        System.out.print("running...");
        while (true) {
            System.out.print("wating...");
            //wait for client socket
            //if the number of clients dose not get max
            if (manager.getClientList().size()<setting.getMaxClient() ) {
                Socket client_socket = ser_socket.accept();
                System.out.print("accepted");
                //create new theard to serve client
                manager.createThread(client_socket);
            }
        }
    } catch (Exception ex) {
        //log
      HWFile.AppendStringToFile(this.manager.getSetting().getLogfile(),ex.getMessage());
    }
}
```

- Listen data and process in server

```
@Override
public void run()
{
    while(true)
    {
        try {//receive data
            Object data=input.readObject();
            if ( data != null) {
                if(data instanceof Message)
                {
                   Message ms=(Message)data;
                   if(ms.getHeader().getMsg_type()==MessageHeader.MessageType.Chat.ordinal())
                   informServerToSendMessage(ms);
                   if(ms.getHeader().getMsg_type()==MessageHeader.MessageType.Disconnect.ordinal())
                   {//stop and remove this thread
                    this.serverGUI.removeThread(this);
                   }
                   if(ms.getHeader().getMsg_type()==MessageHeader.MessageType.joinRoom.ordinal())
                   {//join player to room
                       this.serverGUI.joinPLayerToRoom(Integer.valueOf(ms.getBody().toString()),this.player);
                   }
                   if(ms.getHeader().getMsg_type()==MessageHeader.MessageType.quitRoom.ordinal())
                   {// remove player in room
                       int playerid=ms.getHeader().getFrom_playerID();
                       int roomID=Integer.valueOf(ms.getBody().toString());
                       serverGUI.quitPlayerFromRoom(roomID, playerid);
                   }
                    if(ms.getHeader().getMsg_type()==MessageHeader.MessageType.checkWord.ordinal())
                    {//check the word from client
                        serverGUI.checkWord((Word)ms.getBody(),this.player.getPlayer_id());
                    }
                    if(ms.getHeader().getMsg_type()==MessageHeader.MessageType.endTime.ordinal())
                    {//client end of time , next player
                        serverGUI.nextPlayer(this.player.getPlayer_id(),ms.getHeader().getTo_playerID());
                    }
                } }

        } catch (IOException ex) {
```

- Remove thread when client disconnected

```
/*
 * remove a thread which serves client
 */
public boolean removeThread(ClientTalker cT) {
    //remove client
    for (int i = 0; i < clientThreads.size(); i++) {
        ClientTalker c = (ClientTalker) clientThreads.get(i);
        if (c.getPlayer().getPlayer_id() == cT.getPlayer().getPlayer_id()) {
            //each thread alsoe contains Player Information, so
            //compare if this thread equal param thread ,by PlayerID
            int playerID = c.getPlayer().getPlayer_id();
            clientThreads.set(i, null);
            clientThreads.remove(i);
            int index = 0;
            //remove client information
            for (Client cl : clientList) {
                if (cl.getPlayer().getPlayer_id() == playerID) {
                    clientList.set(index, null);
                    clientList.remove(index);
                    break;
                }
                index++;
            }
            //dispplay all client
            displayClients();
            //infrom to all client o update  Online player, NULL mean All Client when comparing
            pushOnlinePlayersToAllClient(null);
            System.out.println("Removed client");
            cT.stop();
            return true;
        }
    }
    return false;
}
```

- Start a room

```
public void startRoom(Room room) {
    Word sendWord = randomWord();
    //set word to room
    room.setWord(sendWord);
    for (ClientTalker ct : clientThreads) {
        for (Player p : room.getPlayers()) {
            if (ct.getPlayer().getPlayer_id() == p.getPlayer_id()) {
                //call the client talker to talk with client
                ct.sendWordToClientsInRoom(sendWord);
                //start player who join the room first
                startPlayer(room.getPlayers().get(0).getPlayer_id());
            }
            else
            {
                //player 2 update word
                 ct.sendWordToClientsInRoom(sendWord);
                 //and stop player 2
                stopPlayer(room.getPlayers().get(1).getPlayer_id());
            }}}
}
```

- Stop a room

```
public void stopRoom(Room rom,int winerID) {
    int loser = 0;
    //find loser
    for(Player p:rom.getPlayers())
    {
        if(p.getPlayer_id()!=winerID)
            loser=p.getPlayer_id();
    }
    //find client talker to talk with 2 client- winner, loser
    for(ClientTalker c:clientThreads)
    {
            if(c.getPlayer().getPlayer_id()==winerID)
            {
                //plus score for winner
                c.getPlayer().setScore(c.getPlayer().getScore()+10);
                //send message to player
                Message ms_winner=new Message();
                MessageHeader header=new MessageHeader();
                header.setMsg_type(MessageType.win.ordinal());
                ms_winner.setHeader(header);
                c.sendMessageToClient(ms_winner);
            }
            if(c.getPlayer().getPlayer_id()==loser)
            {
                //send message to player
                 Message lose=new Message();
                MessageHeader header=new MessageHeader();
                header.setMsg_type(MessageType.lose.ordinal());
                lose.setHeader(header);
                c.sendMessageToClient(lose);
            }
        }
}
```

- Update word state for two player in room

```
public void checkWord(Word w, int playerID) {
    Room roomtoFind = null;
    for (Room r : roomList) {        //find room of player
        for (Player p : r.getPlayers()) {
            if (p.getPlayer_id() == playerID) {
                roomtoFind = r;
                break; } }  }
    //compare word,check if there are a number of characters are true,then the client can continue
    int[] currentWordState = roomtoFind.getWord().getWordState();
    int[] clientWordState = w.getWordState();
    int numbersOK = 0;
    for (int i = 0; i < currentWordState.length; i++) {
        if (currentWordState[i] != clientWordState[i]) {
            numbersOK++; } }
    if (numbersOK > 0) { //if  number of words which were opened # 0
        roomtoFind.setWord(w);//update word in room
        infromClientToUpdateWord(w, roomtoFind); //inform client to update word
        boolean   allOK=true; //check if whole word is ok
          for (int i = 0; i < w.getWordState().length; i++) {
                 if (w.getWordState()[i] !=characterSate.Open.ordinal())
               {
                    allOK = false;
                    break;  }}
        if(allOK) //and if whole word was opened,infrom  winner,stop game
        {
         stopRoom(roomtoFind, playerID);return;}
        else{ //else this client can continue
        startPlayer(playerID); }
    }//if nothing is true. start second player and stop playing of player
    else {
        for (Player p : roomtoFind.getPlayers()) {
            if (p.getPlayer_id() != playerID) {
                startPlayer(p.getPlayer_id());
                stopPlayer(playerID); //and stop this client
                return;} } }
}
```

3.2 Test functions

Case	Expect result	Result
Player login	All other players know the new player came	Ok
Player login	The player can see all online players	Ok
Player login	The player can see all room	Ok
Player logout	All other players know the new player went	Ok
Player sends public chat message	All other players receive the message	Ok
Player sends private message	The player inside room with sender can see message	Ok
Player joins/quit room	All other players who are not inside any room can see the updated number of players in room	Ok
Player joins/quit room	All other players who are inside a	Ok

	room can see the new player join to room	
Player chose a character when playing game	The other player is inside the room can see updated word	Ok
Player doesn't chose any character after 30 seconds	The other player is inside the room can play game	Ok
Player has found the whole hidden word	Stop game, inform winner, loser	Ok

4 Critical evaluation

This HangmanGame software allows many players interact over network at the same time. The game software uses both UDP (finding server) and TCP/IP to make communication between client and server. UDP is used only to find the server in LAN because it can broadcast message to all client in LAN without an exact IP. TCP/IP is more reliable than UDP. It can inform if there is any problem occurred when conveying data.

Protocol Data Unit is important when programming for client/server. PDU is the message format for client/server to understand each other. PDU in this game is a message with header and body. The body can have a small size or big size and all of transmissions are normally. Both client and server process data easily.

The HangmanGame software provides some basic functions; the GUI of software is simple, each of players can easily interact with other one.

5 Appendixes

5.1 GUI

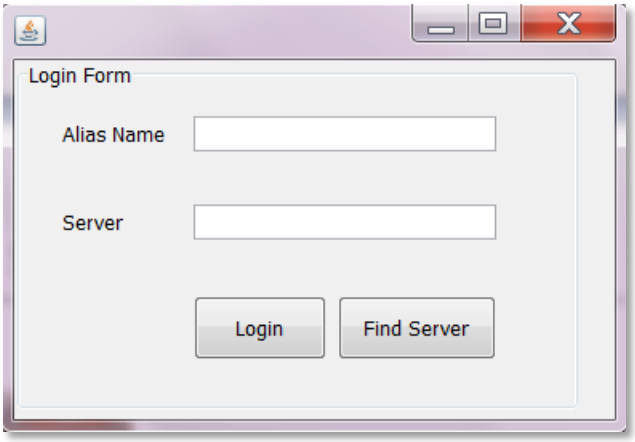

Figure 03: Login GUI

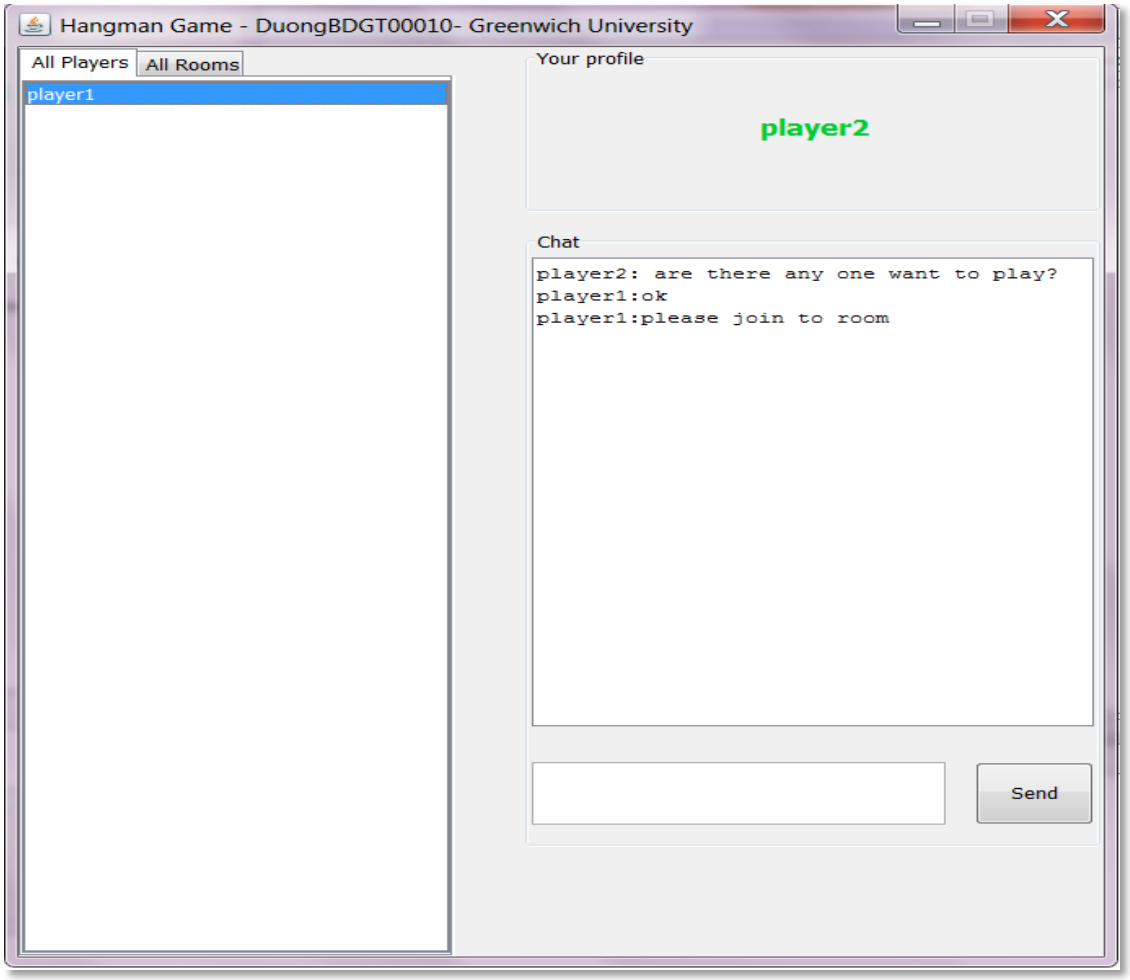

Figure 04: main GUI

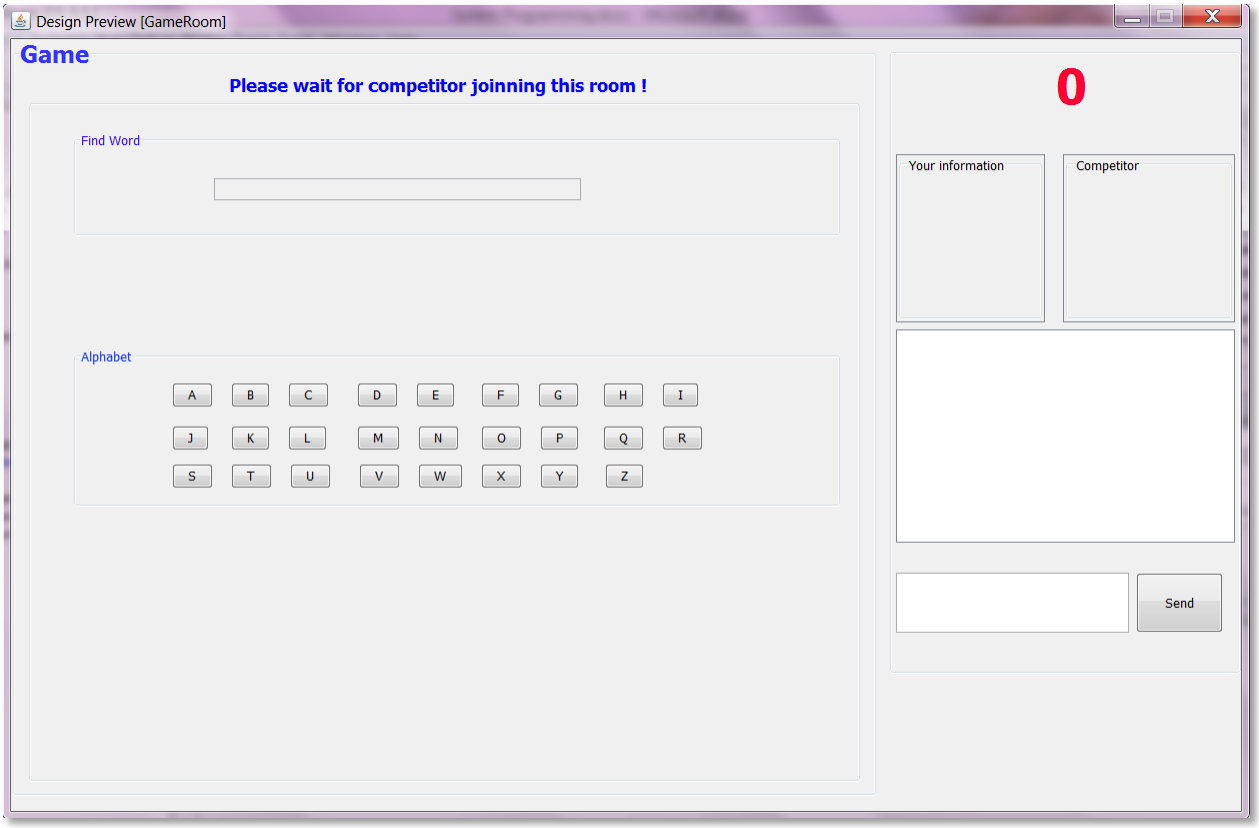

Figure 05: GUI when playing game

5.2 Class Diagram

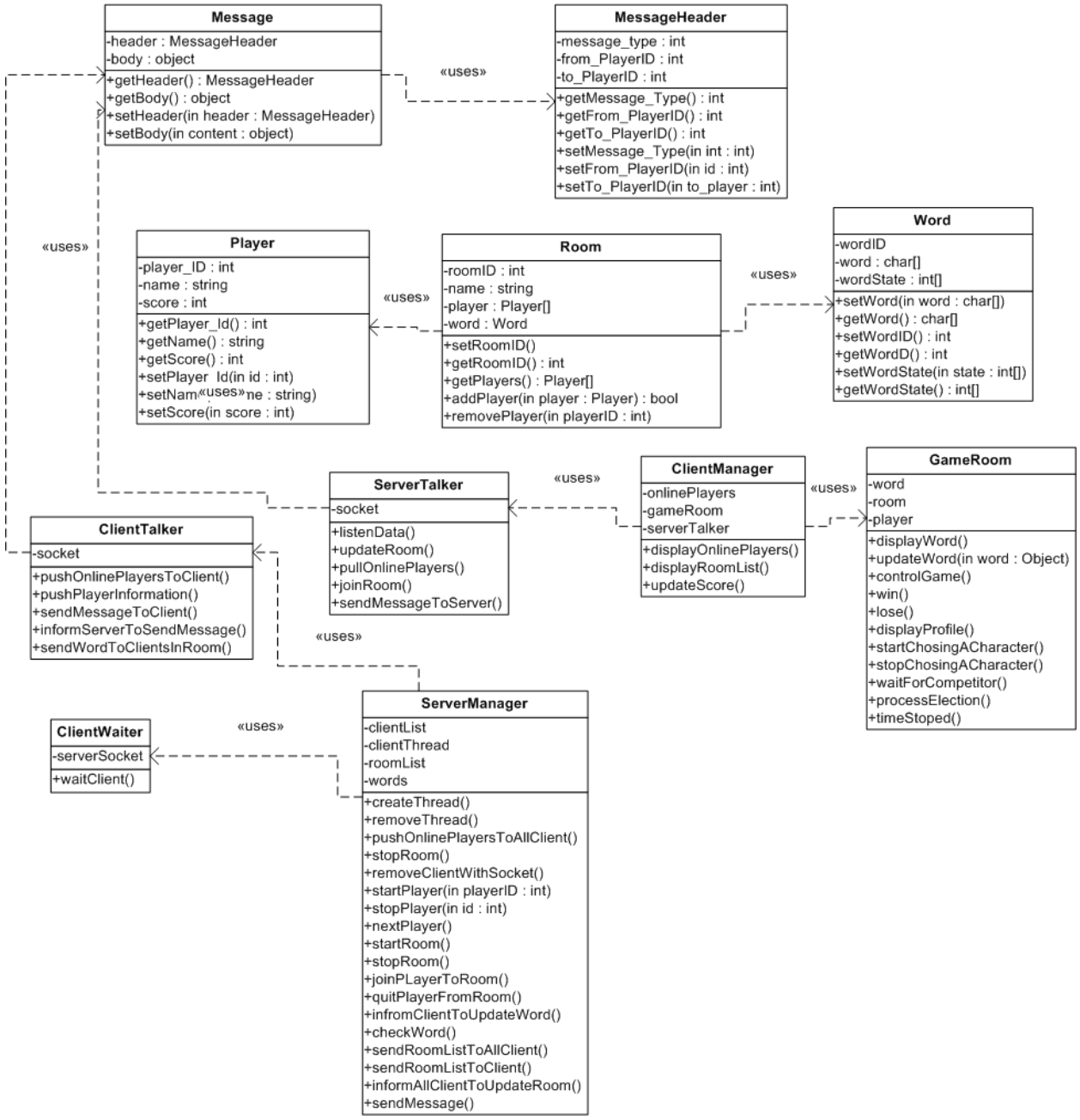

Figure 06: Scenario class diagram