CHAPTER 5
IMPLEMENTATION AND TESTING

Implementation

Server

```
1. ServerSocket listener = new ServerSocket(8901);
2. listener.setSoTimeout(300000);
3. public Player(Socket socket, char mark) {
4. this.socket = socket;
5. this.mark = mark;
6. try {
7.        in        =        new        BufferedReader(        new
InputStreamReader(socket.getInputStream()));
8. out= new PrintWriter(socket.getOutputStream(), true);
9. out.println("GAMEMESSAGE waiting for opponent to connect");
10. }
11. catch (IOException e) {
12. System.out.println("Player died: " + e);
13.}
14.}
```

In line 1 serverocket initialisation with listener name and give PORT serversocket with number 8901. line 2 gives command to serverocket to disconnect to anyone who does not interact with server for 300000 ms. In line 3 create a constructor for the class player and provide a value socket and mark, a socket to handle the clients to connect and mark to give the player sign (X, O, A) to the client. The 4th and 5th rows of socket and mark initialization. The 7th line initializes in to read the message that the client sends to the server via socket, Line 8 of the outbound initialization to send the message to the client through the socket. The 9th row gives a message to the client in the form of "GAMEMESSAGE waiting for opponent to connect" protocol message. The 12th row if the try stage fails then the server will be listed information "player died".

Client

```
1. public GuessHeroClient(String serverAddress) throws Exception {
2. socket = new Socket(serverAddress, PORT);
3. in = new BufferedReader(new InputStreamReader(
4. socket.getInputStream()));
5. out = new PrintWriter(socket.getOutputStream(), true);
```

Line 1 creates a constructor of the GuessHeroClient class with the value serverAddress ("localhost"). Line 2 initializes the socket with value serverAddress ("localhost") and PORT (8901) socket initialization is intended to connect to servers that have the same port. Line 3 initialization in with BufferedReader data type, the goal is to be able to read messages from the server via socket. The 5th line initializes out with the printwriter data type, as the sender messages to the server via socket.

Protocol

In-Game Protocol

Server

```
1. public void run() {
2. try {
3. while (true) {
4. out.println("SUBMITNAME");
5. name = in.readLine();
6. if (name == null) {
7. return;
8.  }
9. synchronized (names) {
10. if (!names.contains(name)) {
11. names.add(name);
12. break;
13. }
14. }
```

```
15. }
16. out.println("NAMEACCEPTED");
17. writers.add(out);
```

Client

```
18. private void run() throws IOException{
19. while (true) {
20. String line = in.readLine();
21. BufferedImage img;
22. if (line.startsWith("SUBMITNAME")) {
23. out.println(getName());
24. }
25. if (line.startsWith("NAMEACCEPTED")) {
26. textField.setEditable(true);
27. }
```

In server at line 1 create method for thread service. The 2nd line is the block of the program that the block is expected to occur exception. Line 3 is a looping function with certain conditions whose contents are in rows 4-12. Line 4 sends a protocol message to the client "SUBMITNAME". Line 5 receiving messages from clients is initialized into name. The 6th row if the name has an empty result, the name will be listed as null. Line 7 returns the name. Line 9 synchronizes the names, names representing the data storage of the form name that has the hashset data type. Line 10 if in the names there is no value equal to the name, then in line 11 names will add name. Line 12 if condition has been reached then while will be dismissed. Line 16 sends a protocol message to the client "NAMEACCEPTED". Line 17 receives a message from this client and broadcasts it, writers is a storage that has a hashset data type that has a value from printwriter, this set is stored so it will be easy to broadcast messages to all clients.

In a client on line 18 creates a method given throwsIO exception, throws are used to read an error or not method. The 19th row performs a continuous loop as the client will always interact with the server. Line 20 creates a string data type

with the line name that later when the server sends a message, it will enter the line. The 21st row creates a bufferedimage data type with the name img. The 22nd line is a condition where if the client receives the "SUBMITNAME" protocol message from the server then the client must fill in the name in the space provided. The line 23 client sends the name to the server. Row 24 is a condition when the client gets the message "NAMEACCEPTED", then on the 25th line the client can now access in the chat section.

Pre-Game Protocol

Server

```
1. out.println("GAMEMESSAGE all players connected , wait for your
turn");
2. if (mark == 'X') {
3. out.println("GAMEMESSAGE your move"); }
```

Client

```
4. if (line.startsWith("GAMEMESSAGE"))
5. lb.setText(line.substring(12)); }
```

In the server at line 1, the server sends the protocol message "GAMEMESSAGE all players connected, wait for your turn". The 2nd line is a condition when the mark is x, then on line 3 sends a message to the client that has been marked x by the server "your move" means it can start the first movement in the game.

In the client on line 4 is a condition when the client receives a protocol message that originally started with the phrase "GAMEMESSAGE", on line 5 lb will set the text sent from the "GAMEMESSAGE" protocol message like "all players connected, wait for your turn" , Lb is a Jlabel.

Box Protocol

Server

```
1. if (command.startsWith("BOX1")) {
2. int location = 0;
3. if (legalMove(location, this)) {
4. counter=0;
5. out.println("VALID_MOVE_BOX1");
6. }
7. else {
8. out.println("GAMEMESSAGE Can't Move. Its Not Your Turn Yet / The Box already choosed");
9. }
10. }
11. public synchronized boolean legalMove(int location,Player player) {
12. if (player == currentPlayer && board[location] == null) {
13. board[location] = currentPlayer;
14. currentPlayer = currentPlayer.opponent;
15. currentPlayer.otherPlayerMovedTwo(location);
16. currentPlayer.previous.otherPlayerMoved(location);
17. return true;
18. }
19. return false; }
```

Client

```
20. bt1.addActionListener(new ActionListener(){
21. public void actionPerformed(ActionEvent ea){
22. out.println("BOX1");
23. }
24. });
25. if (line.startsWith("VALID_MOVE_BOX1")) {
26. img = ImageIO.read(new File(ImageListing.get(index).getGambar1()));
27. Image newimg = img.getScaledInstance(bt1.getWidth(),bt1.getHeight(),Image.SCALE_SMOOTH);
```

```
28. bt1.setIcon(new ImageIcon(newimg));
29. lb.setText("Valid move, please wait");
30. }
31. if (line.startsWith("OPPONENT_MOVED")) {
32. int loc = Integer.parseInt(line.substring(15));
33. if(loc == n1){
34.              img              =              ImageIO.read(new
File(ImageListing.get(index).getGambar1()));
35.                  Image                  newimg                  =
img.getScaledInstance(bt1.getWidth(),bt1.getHeight(),Image.SCALE_S
MOOTH);
36. bt1.setIcon(new ImageIcon(newimg));
37. lb.setText("Opponent moved, please wait for your turn");
38. }
39. if (line.startsWith("OPPONENT_MOVEZ")) {
40. int locc = Integer.parseInt(line.substring(15));
41. if(locc == n1){
42.              img              =              ImageIO.read(new
File(ImageListing.get(index).getGambar1()));
43.                  Image                  newimg                  =
img.getScaledInstance(bt1.getWidth(),bt1.getHeight(),Image.SCALE_S
MOOTH);
44. bt1.setIcon(new ImageIcon(newimg));
45. lb.setText("Opponent moved, your turn"); }
```

In the server on line 1 is a condition when the server receives a protocol message from the client with the prefix "BOX1" then on line 2 the server initialize the int data type with the name of the location and given the value of value 0. Line 3 is a condition with a method that has a value of location (0), and this (currently playing client) will be managed on line 11. Line 4 is to change the counter value to 0, counter is the value of the timer will be explained in Procol Timer. Line 5 sends a protocol message to the client "VALID_MOVE_BOX1". Line 6 marker if condition is not same, then send protocol message to client "GAMEMESSAGE can not move its not yet turn / the box already choosed". Line 11 creates a synchronization method with a boolean type named legalmove with a value

location of type int data and a player of type obj player data which later as a check box if the box has been selected and avoids cheating on players who have not had time to open the box. Line 12 is a condition of checking whether the client requesting the opening box is the same as the currentplayer that is currently serviced and whether the location on the board is still null (no value yet). Board is the name of the data player type in the form of arrays, used to determine the position of the current location selected by the client. Line 13 if conditions are met then the location of the board will be marked ownership on the client who has successfully qualified to select the box. Line 14 turn client will now move to the next client. The 15th row goes to the otherplayermovedtwo method which sends the protocol message "OPPONENT_MOVEZ" and the location of the board to the client with the next turn. The 16th line goes to the previous.otherplayermoved method which also sends protocol messages to other clients "OPPONENT_MOVED" and other client board locations. Line 17 if condition is fulfilled it will be given return value that is true, if condition not fulfilled line 19 will give return value false.

In the client on line 20, bt1 has an action where in line 21 when clicked, it sends a message to the server on line 22. on line 25 is a condition when the client gets a message from the server "VALID_MOVE_BOX1", then on line 26 and 27 clients will do the process of reading the image contained in the resource folder. Line 28 sets the selected box to an image. Line 29 sets the lb to "valid move, please wait". Row 31 is a condition when cloient gets protocol message from server "OPPONENT_MOVED", then line 32 client will parse location sent by server to int, then entry to line 33 client will go into a condition check if the location sent equal the location provided in the client in this case n1 has a value of 0, it will be the process of reading the image on lines 34 and 35 and set the image in accordance with the location in Figure 36. then set the lb with the words "Opponent moved, please wait for your turn ". Line 39 is a condition when the

client receives a protocol message from the server in the form of "OPPONENT_MOVEZ", then on line 40 the client will parse the location sent by the server into the int data type, then go to line 41 client will go into a check condition if location which is sent the same as the location provided on the client in this case n1 has a value of 0, it will be the process of reading the image on lines 42 and 43 and set the image in accordance with the location in Figure 44. then set the lb with the words "Opponent moved, your turn "which means now it's his turn to do box selection.

Timer Protocol

Server

```
1. timer.scheduleAtFixedRate(new TimerTask() {
2. @Override
3. public void run() {
4. counter++;
5. if(counter >= second){
6. System.out.println("Times Up");
7. TimeUp();
8. counter=0;
9. }
10. else{
11. System.out.println("Time left:" + counter); } }
12. }, 3000, 3000);
13. public void TimeUp(){
14. currentPlayer.TimeUps();
15. currentPlayer.opponentTimeUpEnemyOne();
16. currentPlayer.previous.TimeUpEnemyTwo();
17. currentPlayer=currentPlayer.opponent;}
```

Client

```
18. if(line.startsWith("OPPONENT_MOVEE")){
```

```
19. lb.setText("Opponent moved, your turn"); }
20. if(line.startsWith("OPPONENT_MOVEF")){
21. lb.setText("Opponent moved , please wait for your turn"); }
22. if(line.startsWith("OPPONENT_MOVEL")){
23. JOptionPane.showMessageDialog(frame,"Times Up , Please Wait
for your turn");
24. InformationArea.append("Server :Times Up , Please Wait for
your turn"+"\n");
```

In server at line 1 do a schedule call from timertask. Line 3 initializes the run method. Row 4 counter will always increase. Line 5 if the counter has reached more than the second (30) then the server will appear TIMES UP information on line 6, then line 7 calls the method TimeUp (). Line 8 counter becomes 0 and will increase again until it has the same value as second. Line 10 If the counter is not the same as the second, Line 11 on the server will update on the counter. Row 13 is a method of timesup whose contents are in rows 14-17. line 15 sends a message to the player's turn now "TIMESUP", line 15 invites the timeupenemyone method whose contents send the "OPPONENT_MOVEE" protocol message to the next client, while line 16 invites the timeupenemytwo method whose message sends the "OPPONENT_MOVEF" protocol message to the other client, line 17 changes turn the client now to the next client turn.

In Client on line 18 if client get message "OPPONENT_MOVEE" from server then line 19 client will set lb to "opponent move, your turn". Line 20 if the client gets the "OPPONENT_MOVEF" protocol message from the server then line 21 client sets the lb to "opponent move, please wait for your turn". Line 22 if the client gets the "OPPONENT_MOVEL" protocol message from the server then line 23 client will create a "Times Up, please wait for your turn" message box. Line 24 of InformationArea will be shown a message sent by the server.

Answer Protocol

Server

```
1. String command = in.readLine();
2. if (command.startsWith(answerList[index])){
3. skor++;
4. AnswerTrue();
5. index++;
6. for(PrintWriter writer :writers){
7. writer.println("WINNER ");
8.                 }
9. for (PrintWriter writer : writers) {
10. writer.println("SCORE " + name + ": " + skor);
11. }
12. AnswerMove();
13. }
14. }
15. }
```

Client

```
30. if(line.startsWith("WINNER")){
32. index++;
33. bt1.setIcon(null);
34. bt2.setIcon(null);
35. bt3.setIcon(null);
36. bt4.setIcon(null);
37. bt5.setIcon(null);
38. bt6.setIcon(null);
39. bt7.setIcon(null);
40. bt8.setIcon(null);
41. bt9.setIcon(null);
42. JOptionPane.showMessageDialog(frame,"This round is over , we have the winner. Wait for your turn");
43. InformationArea.append("Server :This round is over , we have the winner. Wait for your turn"+"\n");
```

```
44. }
45. if(line.startsWith("OPPONENT_MOVEK")){
46. JOptionPane.showMessageDialog(frame,"Your Move");
47. InformationArea.append("Server :Your Move"+"\n");
48. }
```

In the server on line 1 creates a string data type with the given command name the value of messages sent by the client to the server. Line 2 is the condition if the message sent by the client is the same as the answer image that has been provided in the answerlist such as index to 0, then in line 3 goes to a condition whether the mark of the client is X, if true then the client with mark x will get additional score on line 4 and go to the AnswerTrue method. Lines 6-13 have the same principle as lines 2-4. If all conditions are met the index will increase. Line 15 performs broadcasts on all clients by using writer type data printwriter that is filled with data from hashset writers. Line 16 sends the protocol message "SCORE: name: + score" on the client. Line 17 is a method of AnswerTrue whose method content is in lines 18-29. line 18 sends the "WINNER" protocol message to the client. Line 19 invites the loserone method that contains the protocol message "OPPONENT_LOSEE" to the next player. Line 20 invites the losertwo method whose contents send the protocol message "OPPONENT_LOSER" to another player. Lines 21-29 set the board location to null.

In the client on line 30 is a condition when the client gets a protocol message from the server "WINNER" then on line 31 the client will do the text set on lb with the words "You are the winner, Your turn". The line 32 index will add, the index here is for the pointer of the imagelist showing the image to be selected at the value of the current imagelist. Lines 33-41 set each button to null means that the original images will be omitted. Line 42 creates a message box with the message "This round is over, we have the winner. Line 43 on InformationArea will be shown a message from the server, "This round is over, we have the winner .wait for your turn". In Line 45 if the client gets the message

"OPPONENT_MOVEK" then on line 46 the client will create a message box with the message "Your Move", and on line 47 on informationArea will be shown a message from server "Your Move".

Chat Protocol

Server

```
1. for (PrintWriter writer : writers) {
2. writer.println("MESSAGE " + name + ": " + command);
3. }
```

Client

```
4. textField.addActionListener(new ActionListener() {
5. public void actionPerformed(ActionEvent ez) {
6. out.println(textField.getText());
7. textField.setText("");}});
8. if (line.startsWith("MESSAGE"))   {
9. messageArea.append(line.substring(8) + "\n"); }
```

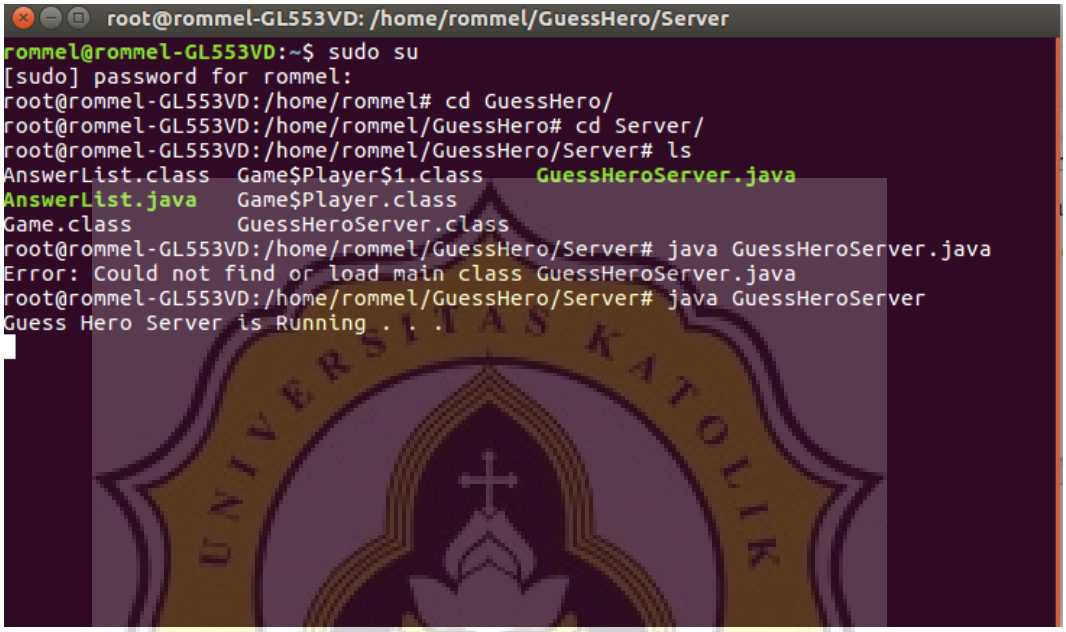
In the server on line 1 creates a printwriter with the name writer filled with data from the writers of the destination server to make it easier for the server to directly broadcast to all servers. Line 2 sends a protocol message to the client in the form "MESSAGE + name +: + command"

In client on line 4 make textField has an action which is where in line 5 when in click enter, then in line 6 will send message written by client on textfield to server at. Row 7 sets of text becomes empty. Line 8 is a condition if the client gets a protocol message with the prefix "MESSAGE" from the server, then line 9 client will add the message sent by the server on messageArea.

Testing

Functional Testing

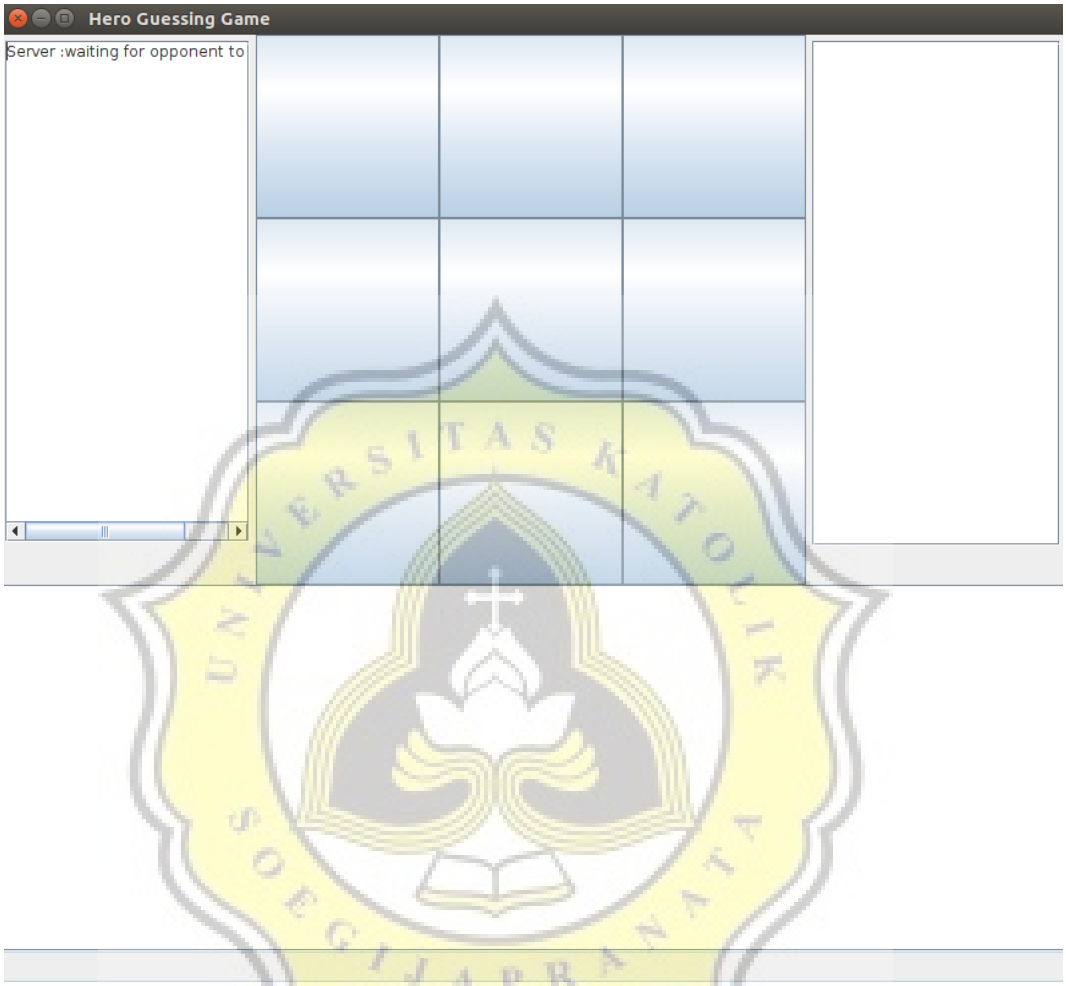
Server



The server must be run first so that the client can connect with the server. If the server has not been running will be sure the client has an error because the port that has not been accessed.

In this case the server successfully implemented into GuessTheHero game.

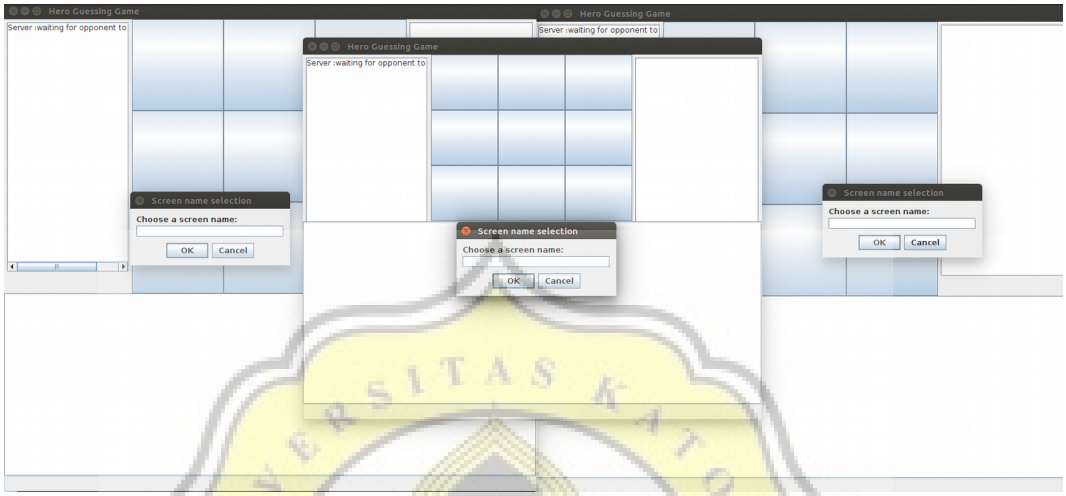
Client



When the client successfully accesses the server, the client will load the user interface as in the picture above. The client will get a message "waiting for opponent to connect" from the server until there are 3 clients connected to the server.

In this case the client successfully implemented into GuessTheHero game.

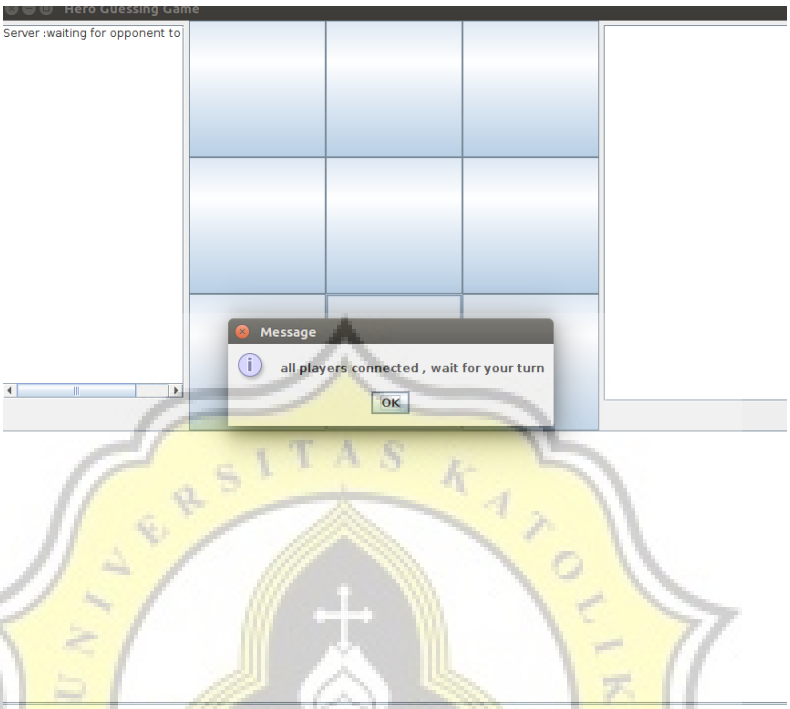
In-Game Protocol



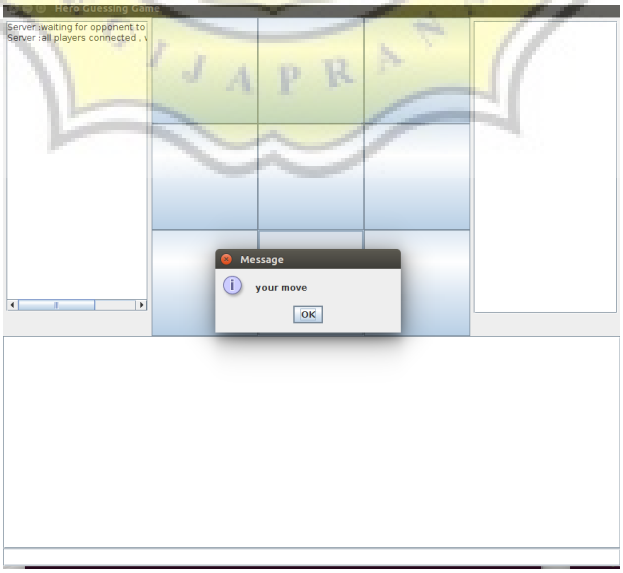
If there are already 3 connected clients, the server will send a protocol message to the client "SUBMITNAME". The name filled by the client will be sent to the server and the server will confirm the name. In this case if there are 3 clients that have been connected to the server, the server managed to send commands to the client to fill in his name.

In this Game protocol successfully implemented into GuessTheHero game.

Pre-Game Protocol



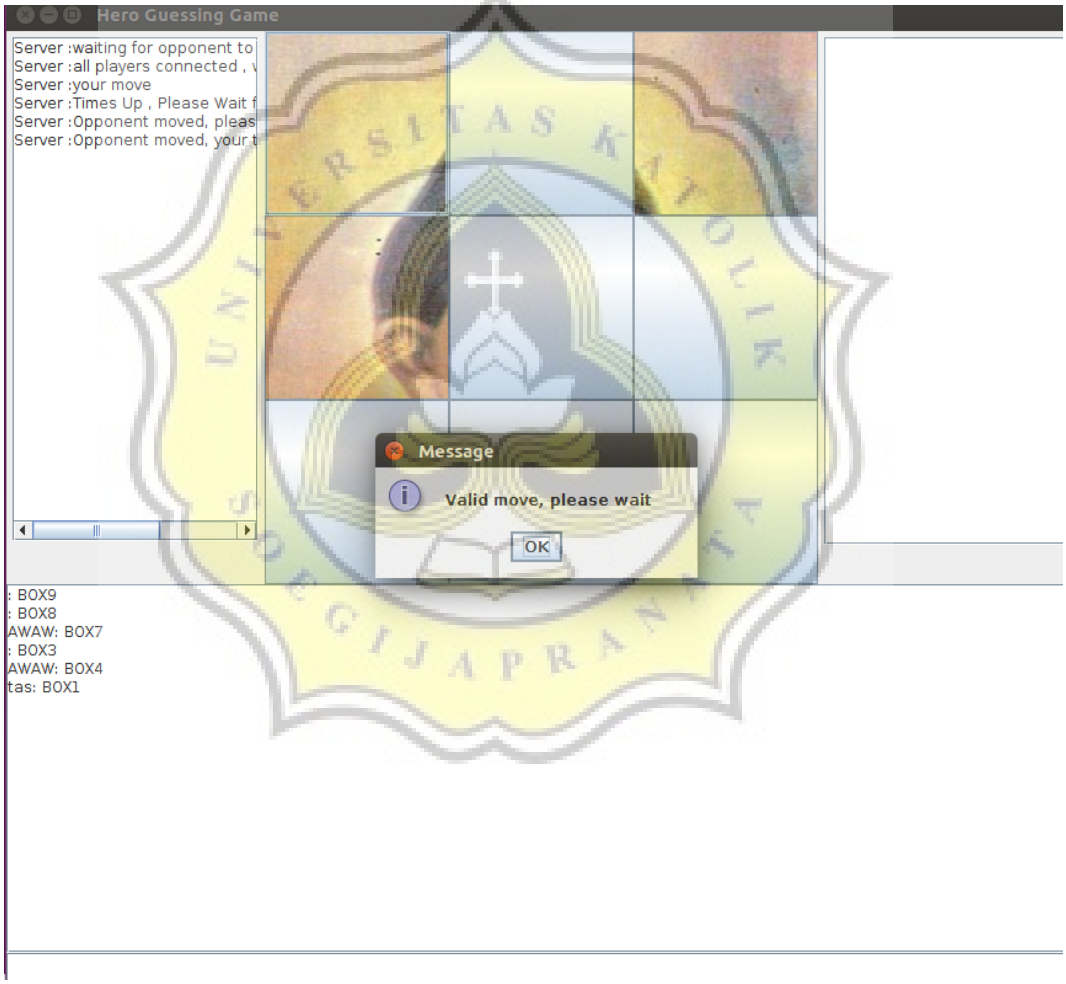
When it has finished entering the name, the server will tell the client to wait their turn to play.



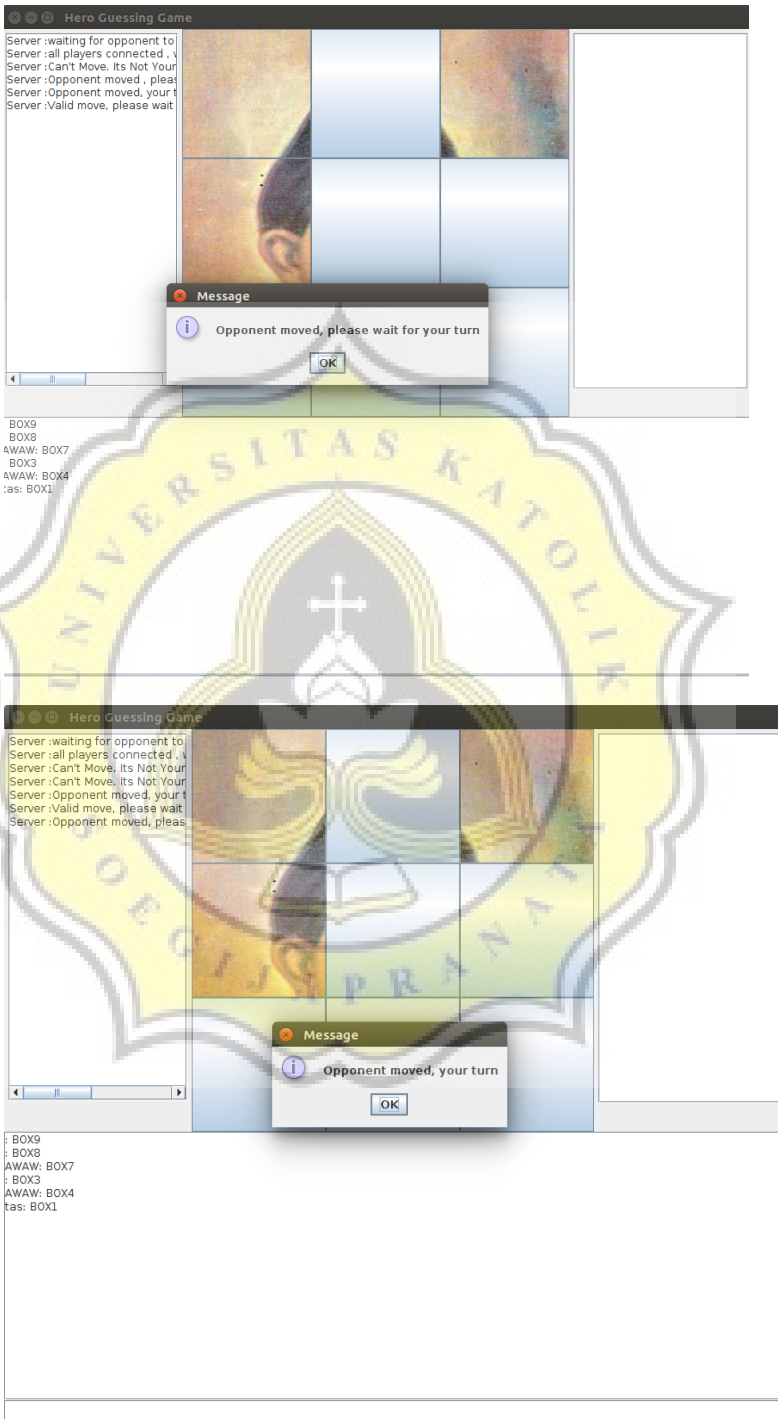
Clients who get the message "your move" from the server can to start the game first.

In this case Pre-Game Protocol successfully implemented into GuessTheHero game.

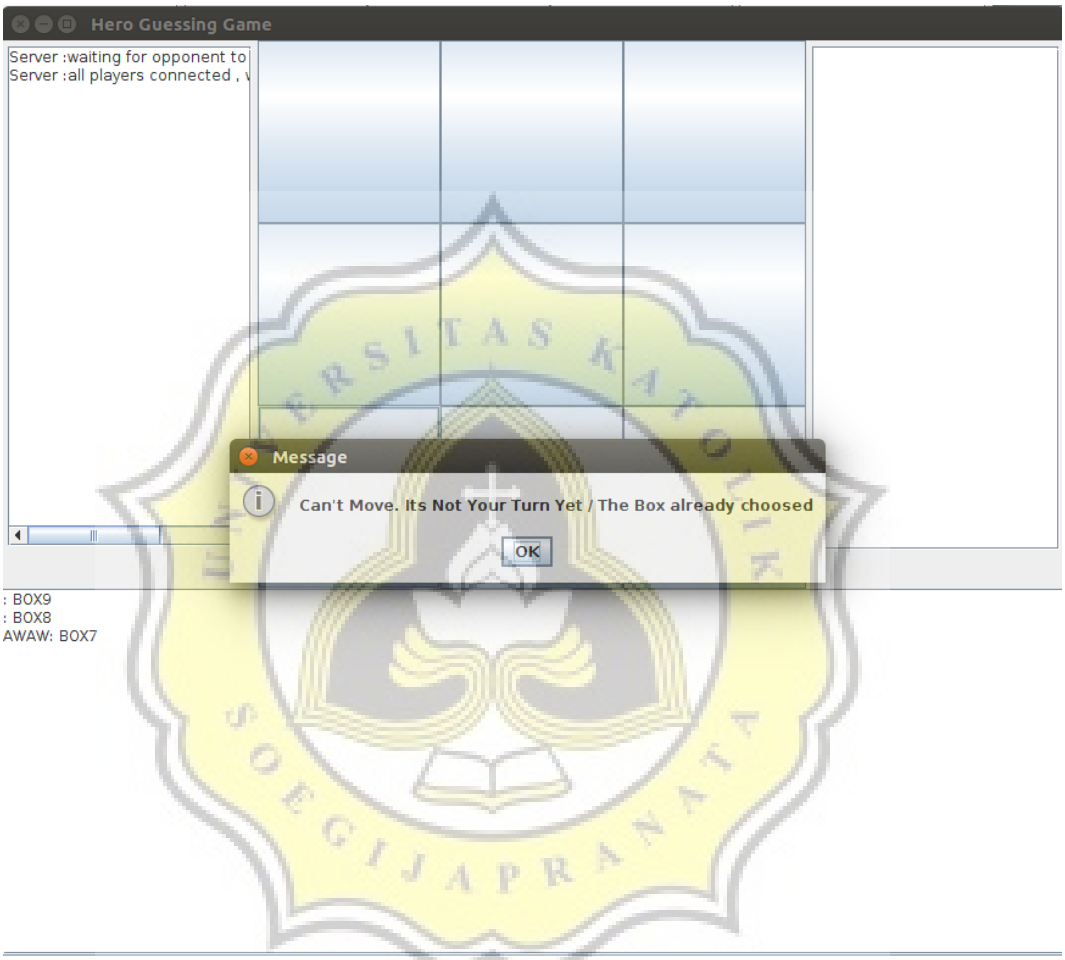
Box Protocol



Suppose players choose box 1, and open the box 1 on the top left corner. When the player selects the box, the player sends a message to the server with the message "BOX1", if the server allows the server will reply the bag with the message "valid move, please wait".


Hero Guessing Game
Server :waiting for opponent to
Server :all players connected , \
Server :Can't Move. Its Not Your
Server :Opponent moved , pleas
Server :Opponent moved, your t
Server :Valid move, please wait
Message
Opponent moved, please wait for your turn
OK
BOX9
BOX8
AWAW: BOX7
BOX3
AWAW: BOX4
as: BOX1
Hero Guessing Game
Server :waiting for opponent to
Server :all players connected , \
Server :Can't Move. Its Not Your
Server :Can't Move. Its Not Your
Server :Opponent moved, your t
Server :Valid move, please wait
Server :Opponent moved, pleas
Message
Opponent moved, your turn
OK
BOX9
BOX8
AWAW: BOX7
BOX3
AWAW: BOX4
tas: BOX1

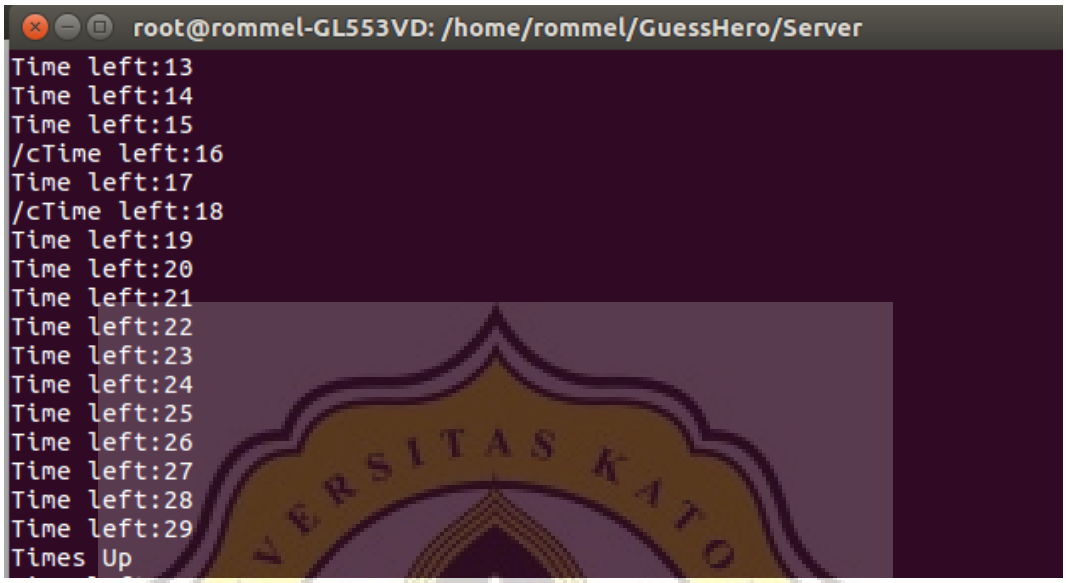
If the server has allowed players to open box 1, then the server will send a message to other players that turn the player who opened the box will be replaced



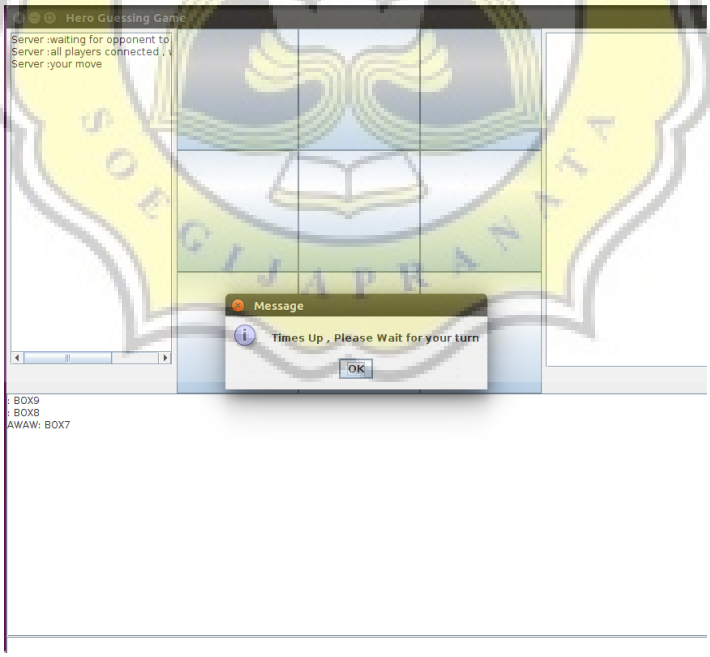
Players who commit acts of fraud such as selecting a box before a turn or selecting a box that has been opened will get a message from the server as in the picture.

In this case the box protocol successfully implemented into GuessTheHero game.

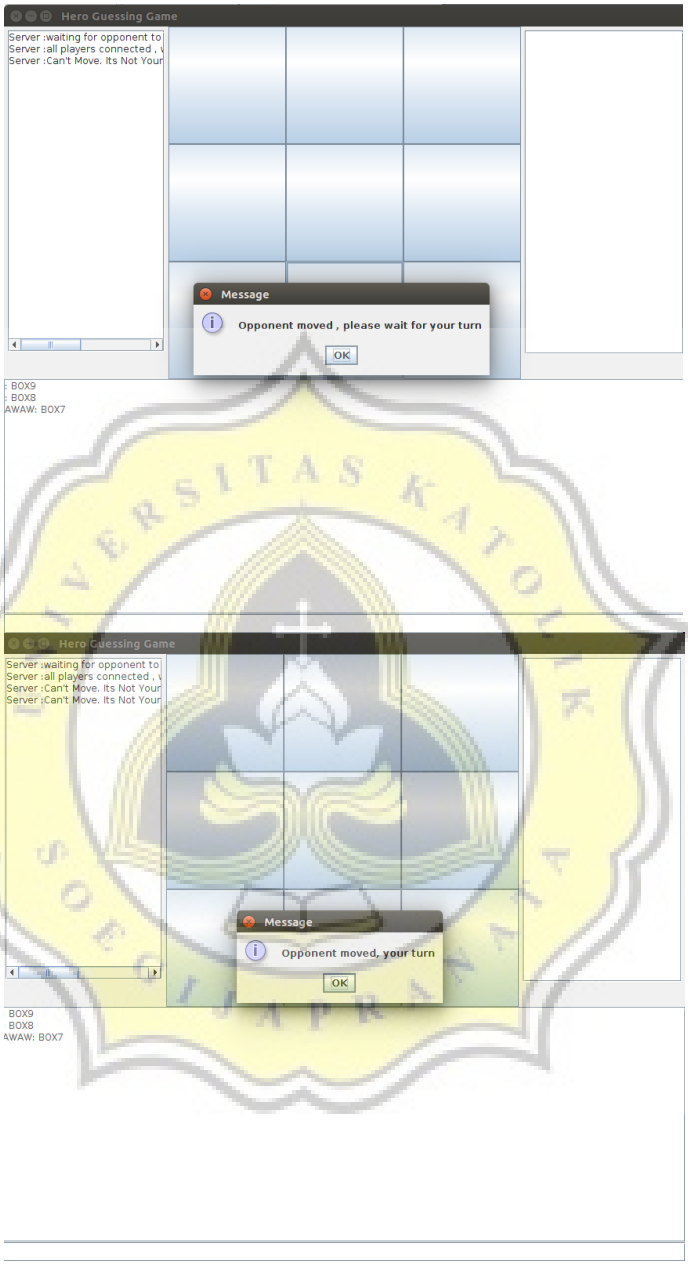
Timer Protocol



The timer will run when the game starts, if the timer is at number 30 then the server will send a command to the client in the form of "TIMESUP"



When a client gets a chat from the server in the form of "TIMESUP" then the client who in turn does not move during the given time will get the message from the server.



The player's turn will be transferred to the next player by the server. The server will send a message to the other players that it is now the player's turn / other players have done the movement, please wait

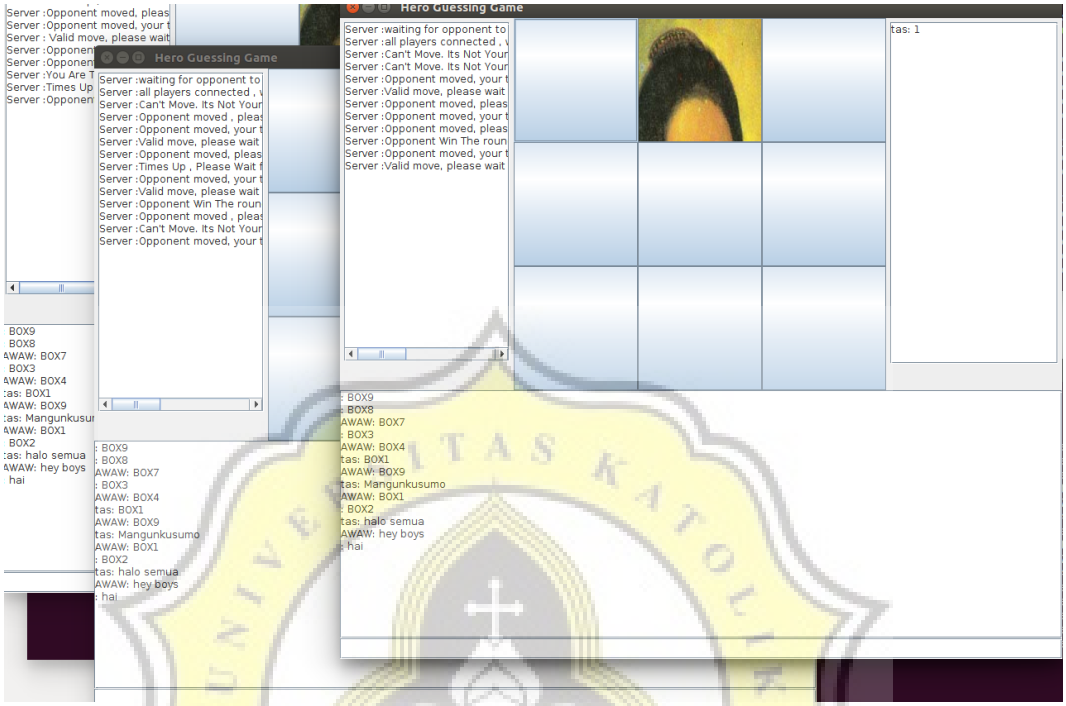
In this case timer protocol successfully implemented into GuessTheHero.

Answer Protocol

When a player answers successfully, the server sends a message to all clients "This round is over. We have the winner. Please wait for your turn ". The player who answers will get the score and all the box-box images on the player will be reset to be original. Then the server loads the next answer and the client loads the next picture. Then the server will send a message to players who can start the movement on the game.

In this case Answer Protocol successfully implemented into GuessTheHero game.

Chat Protocol



The players can exchange messages by writing in the space provided below, the message will be sent to the server. The server will then give the message to all players so that all players can see the messages sent by the player who sent the message. The place to send the message is also a place to answer the puzzle of the image being played.

In this case the chat protocol successfully implemented into GuessTheHero game. When someone answers when it is not the timer turn on the server gets messed up. Behind it if the client who answered in accordance with the turn of the game can still run smoothly.

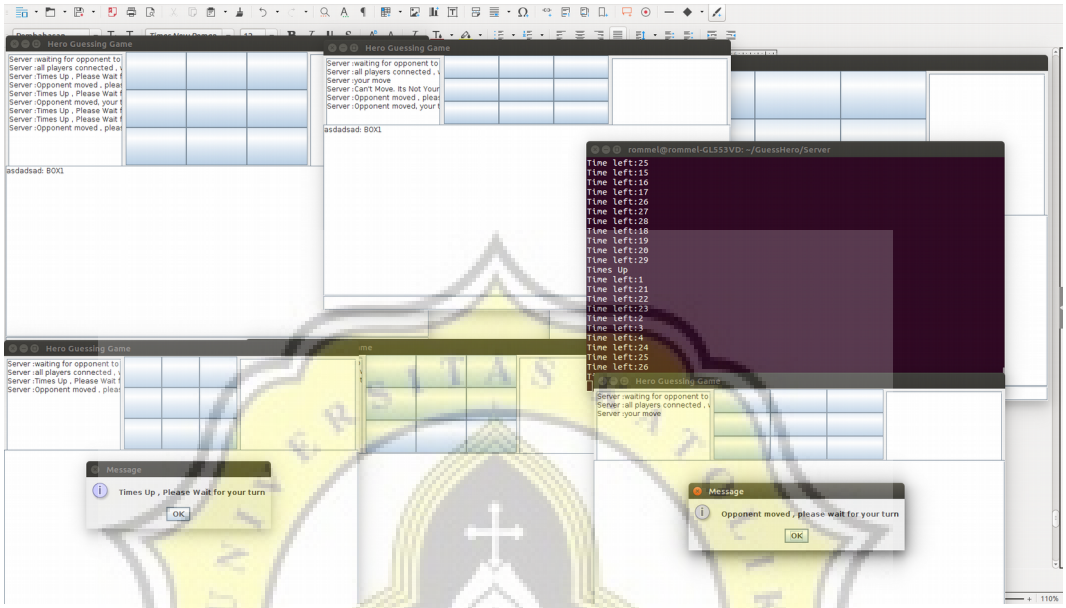
Tabel Resume

Testing	Status	Action
Server	Success	Server successfully accepts the connection from the client.
Client	Success	Client successfully connected to the server and enter the game interface.
In-Game Protocol	Success	Server successfully sent "SUBMITNAME" message to the client. Client successfully receives a message from the server and fills the name requested by the server.
Pre-Game Protocol	Success	Server managed to tell each client that the client had entered into the game Client successfully received from the server and start the game
Box Protocol	Success	Server successfully receives messages from clients who want to open the box and process that the box can be opened client or not Client managed to get permission from the server that the box can be opened otherwise the client also managed to receive warning from the client if the box has been opened / not turn so the client can not open the box
Timer Protocol	Success	Server successfully sends a message to each client that the player's turn is now exhausted and proceeded to the next player. Client manages to receive protocol messages from the server
Answer Protocol	Success	Server successfully sends a message to all players that the round has a winner and loads the next answer, and sends a message to the player who will make the movement Client successfully received messages from the server, and managed to create the next

		image
Chat Protocol	Success	Server successfully receives the message from the client and sends the message to the other client Client manages to receive messages from servers sent by other clients

Performance Testing

Limitation Player Test



In this test will try if there are clients who play more than 3 on 1 server the same. The result server will still run smoothly, and create a new game instance to the 3 clients connected.

Server Perfomance Test

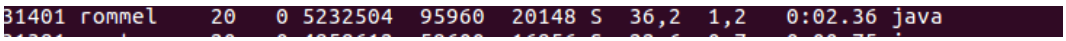
In the test this the server will be sent many data at once in the same time to see how much CPU and Memory usage required by the server and what the impact for the client who is connected. Test will be divided into 3 that is sending 100 data, 1000 data, and 10000 data at once on server. Test is done on ASUS ROG laptop with linux ubuntu as its operating system..

```
31520 rommel    20   0 5232504  68112  20176 S  17,3  0,8   0:01.75 java
```

When the server sent 100 data at once, the use of cpu on the server to 17.3% while for memory used 0.8%.

```
31194 rommel    20   0 5232504 101220  20100 S  21,3  1,3   0:02.52 java
```

When the server sent 1000 data at once, the use of cpu on the server to be 21.3% while for the memory used 1.3%.



When the server sent 10000 data at once, the usage of cpu on the server to be 36.2% while for the used memory 1.2%.

Impacts that occur, the client will experience hangs when sent 100, 1000, and 10000 data.

Total Data	CPU Usage	Memory Usage
100	17.3%	0.8%
1000	21.3%	1.3%
10000	36.2%	1.2%

Based on the analysis of the server performance test, the server will experience a hang if sent a lot of data at once on a package. While on the guessthehero game, the data sent is 1 data in 1 packet and the delivery of data is regular because it uses protocol to manage game performance and communication relationship between client and server. This allows the server to load multiple clients

Tested a total of 100 clients connected to the server and let the timer protocol system run. The result is the server is still fluent in exchanging messages to the 100 clients. Analyzing the client, the client uses 0.3 cpu usage and 0.1 memory usage to exchange messages. while to exchange client image protocols using 1.3 cpu usage and 1.0 memory usage.