Design of Embedded System

Final Report

Team	#2 [Black Pink]
Member	Kwon Dokyeong(20154077)
	Kim Nawon(20154437)
	Kim Gayoung(20150035)
	Choi Bowon(20155212)

Index

1. About Team

- Team Name

Black Pink

- Team Member (Role)

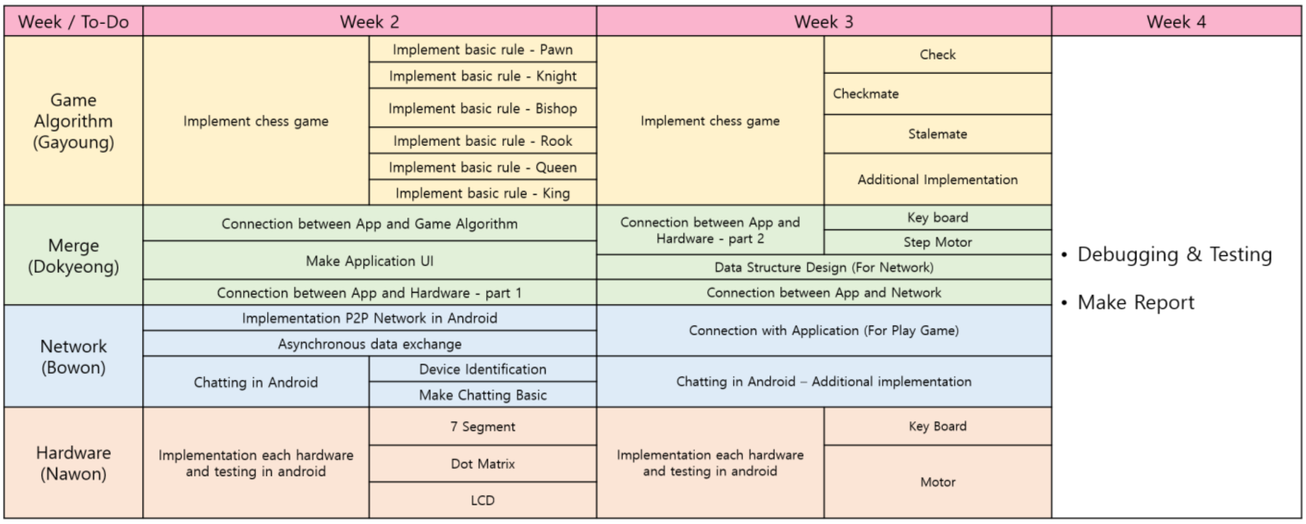
DoKyeong Kwon (Project Manager – Merge)

GaYoung Kim (Algorithm Manager – Make Chess Algorithm)

Nawon Kim (Hardware Manager – Make Hardware functions)

Bowon Choi (Network Manager – Make Socket Network)

2. Plan



Week / To-Do	Week 2		Week 3		Week 4
Game Algorithm (Gayoung)	Implement chess game	Implement basic rule - Pawn	Implement chess game	Check	• Debugging & Testing • Make Report
		Implement basic rule - Knight		Checkmate	
		Implement basic rule - Bishop		Stalemate	
		Implement basic rule - Rook		Additional Implementation	
		Implement basic rule - Queen			
		Implement basic rule - King			
Merge (Dokyeong)	Connection between App and Game Algorithm		Connection between App and Hardware - part 2	Key board	
				Step Motor	
	Make Application UI		Data Structure Design (For Network)		
	Connection between App and Hardware - part 1		Connection between App and Network		
Network (Bowon)	Implementation P2P Network in Android		Connection with Application (For Play Game)		
	Asynchronous data exchange				
	Chatting in Android	Device Identification	Chatting in Android – Additional implementation		
		Make Chatting Basic			
Hardware (Nawon)	Implementation each hardware and testing in android	7 Segment	Implementation each hardware and testing in android	Key Board	
		Dot Matrix		Motor	
		LCD			

Figure 1 Total Schedule

Figure 1 shows a summary of the four-week schedule. The team has completed the project according to the above schedule and if it is not possible to implement within the time or if the time is shortened, it is finished to be implemented before the deadline with some changes according to the schedule of the team. (Since we had a change in the overall schedule after Week 1, we posted it except for one week.)

3. Implementation

3.1 Overview

1) Requirements Table

Requirements	Contents		Implementation
Basic	Multiplayer Game over internet		O
	LCD		O
	7-Segment		O
	Dot Matrix		O
	Keyboard		O
	2vs2 Chess	Basic Rule	O
		Check / Check Mate	O
		Stale Mate	O
Additional	Available paths for selected chess piece		O
	Motor		O

Table 1 Requirement Table

Table 1 summarizes the requirements. We have implemented all of the basic requirements, implemented additional guidelines in the game process, and added a hardware motor. We used motor to notice Check Mate or Stale Mate status to players.

2) Class Diagram

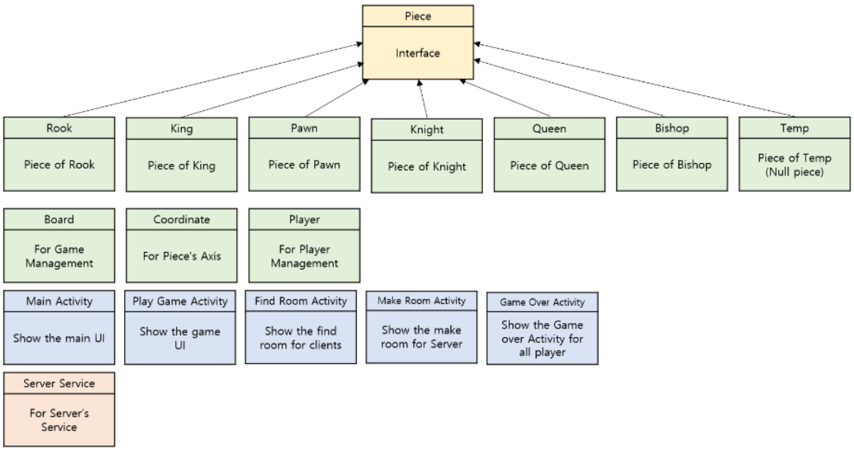


Figure 2 Class Diagram

Figure 2 is a brief class diagram for the project. First, we have created an Interface called 'Piece' to display horse information. We then instantiated and managed a total of 7 pieces using the interface. Next, we used a class called 'Board' to manage the rules of the whole game board. At the same time, we used a class called 'Coordinate' to conveniently manage the 'Piece' coordinate system. We also created a class called 'Player' to manage the basic information of 4 players.

The blue part of Figure 2 shows the activities representing the screen. 'Main Activity' shows the first screen of the game, and 'Play Game' shows the screen when playing the game. 'Find Room' is the screen for Clients to enter and 'Make Room' shows the screen for Server to create the room. Finally, 'Game Over Activity' is a screen that shows when the game is over.

Our team implemented 'Service' to prevent the thread created by the server from terminating. Since we implemented the thread so that it can keep running in the background, we created a server service like the orange table in Figure 2 and managed one server.

The methods and variables of each class are described in Section 3.2 and later.

3.2 Game Algorithm

1) Basic Rule & Moving

Pawn, Knight, Bishop, Rook, Queen, King's moves are followed by the basic rule of chess. Each piece has its own class and its own move method.

- Check

➢ A king is in check when it is under attack by at least one enemy piece.

➢ A piece is unable to move because it would place its own king in check may still deliver check to the opposing player.

The following ways to get out of check are :

(1) Move the king to a square where it is not in check.

(2) Capture the checking piece (possibly with the king).

(3) Block the check by placing a piece between the king and the opponent's threatening piece.

- Check Mate

If a player's king is placed in check and there is no legal move that player can make to escape check, then the king is said to be checkmated, the game ends, and that player loses.

- Stale Mate

When a player's king is not in check and no other moves to make. It is a type of draw.

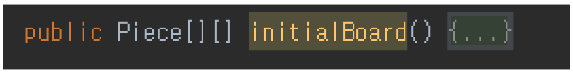
- Game Over

There are three cases of Game Over. Team A is Win, Team B is Win, Draw

2) Board Implementation

Board is a class that manages the game as a whole. Please note that we wrote only a brief description of the method because the code is too long.

- Initial Board Method

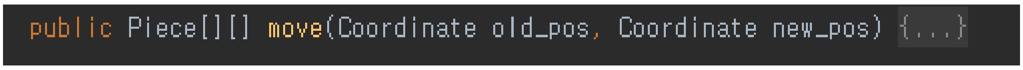


```
public Piece[][] initialBoard() {...}
```

Figure 3 Board Class Code

This method initializes the board. The board is set on.

- Move Method

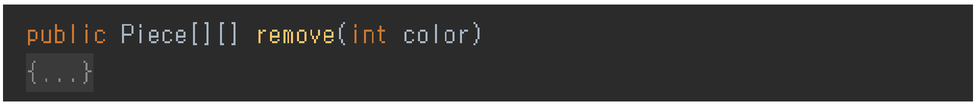


```
public Piece[][] move(Coordinate old_pos, Coordinate new_pos) {...}
```

Figure 4 Board Class Code

This method changes the board by moving a piece. It needs the old position and new position of a piece. It changes the piece on old position with the piece on new position.

- Remove Method

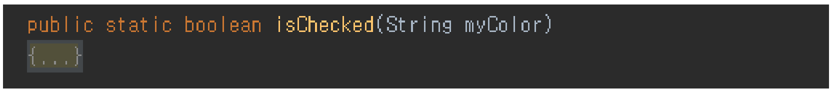


```
public Piece[][] remove(int color)
{...}
```

Figure 5 Board Class Code

This method removes all the pieces when theirs king died.

- isChecked Method



```
public static boolean isChecked(String myColor)
{...}
```

Figure 6 Board Class Code

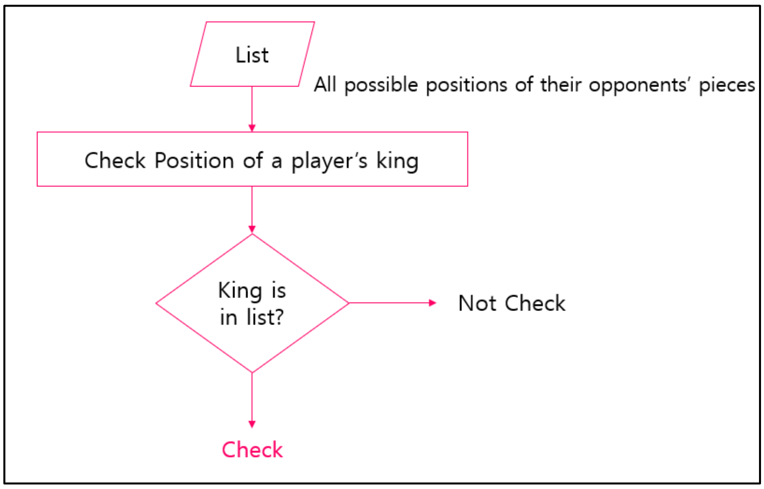
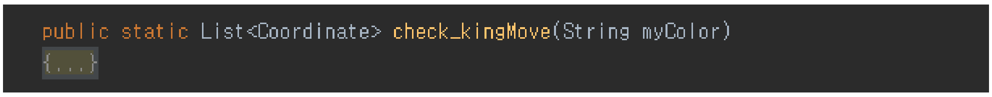

List
All possible positions of their opponents' pieces
Check Position of a player's king
King is
in list?
Not Check
Check

Figure 7 Check Algorithm

Implemented with the above algorithm.

- check_kingMove Method

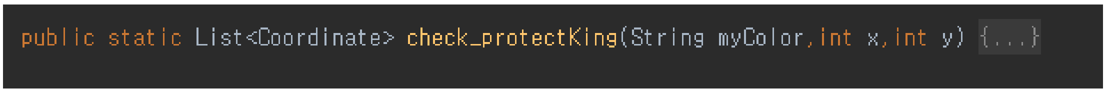


```
public static List<Coordinate> check_kingMove(String myColor)
{...}
```

Figure 8 Board Class Code

This method returns the available coordinate where the king can move avoiding the dangerous positions.

- check_protectKing Method

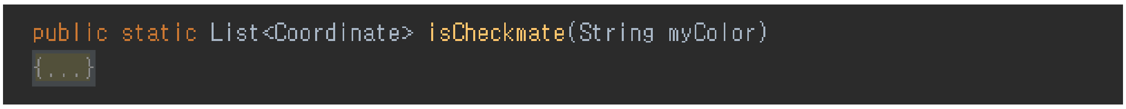


```
public static List<Coordinate> check_protectKing(String myColor,int x,int y) {...}
```

Figure 9 Board Class Code

This method returns the available coordinate where the piece on parameter x, y can kill the piece which is attacking the king.

- isCheckmate Method



```
public static List<Coordinate> isCheckmate(String myColor)
{...}
```

Figure 10 Board Class Code

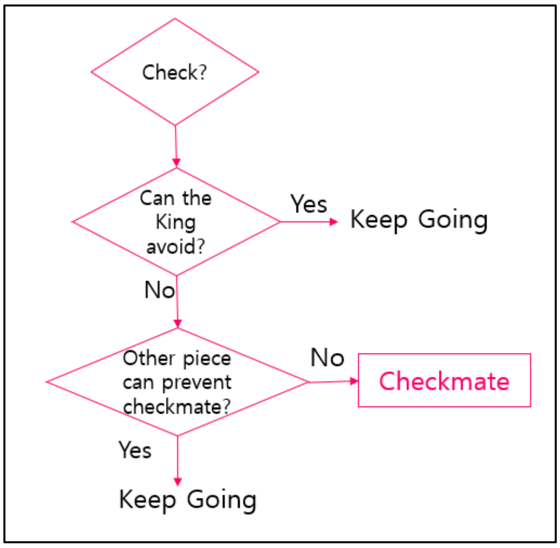
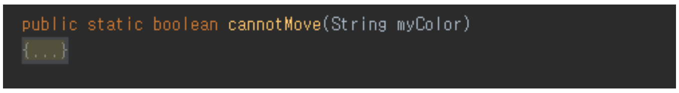


Figure 11 Check mate Algorithm

Implemented with the above algorithm.

- cannotMove Method



```
public static boolean cannotMove(String myColor)
{...}
```

Figure 12 Board Class Code

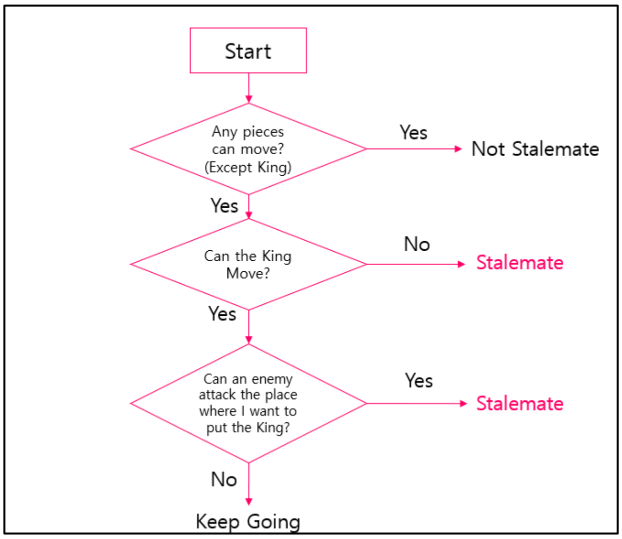

Start
Any pieces can move? (Except King)
Yes
Not Stalemate
Yes
Can the King Move?
No
Stalemate
Yes
Can an enemy attack the place where I want to put the King?
Yes
Stalemate
No
Keep Going

Figure 13 Stale mate Algorithm

This method checks whether the color is stalemate. It checks whether the king can move on any available position. It also checks whether other pieces except the king can move. If the king cannot move safe position, which means the position its opponents don't attack and other pieces cannot move, this method returns true.

3.3 Server

1) Network Data Structure

Contents	Message
Start Game	STARTGAME&NickName
Piece's Moving	MOVING&NowTurn&NextTurn&OriginAxis&MovingAxis
User's Chatting	CHATTING&NickName&ChattingContents
Turn Out	MOVING&TurnOut&NowTurn&NextTurn&OriginAxis&OriginAxis
Game Over	GAMEOVER&GameResultMsg

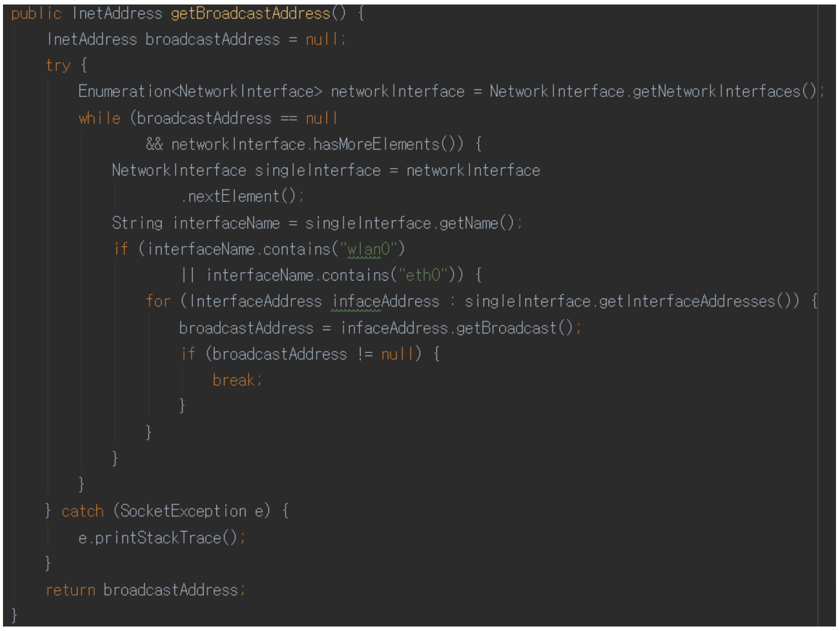
Table 2 Network Data Structure

The above table is about summarizing the message required for the network. We communicated using the above message and send the necessary information.

2) Server Service Implementation

Server Service Class is a class that creates the thread of the server and performs the necessary tasks for the Socket Network.

- getBroadcastAddresss Method

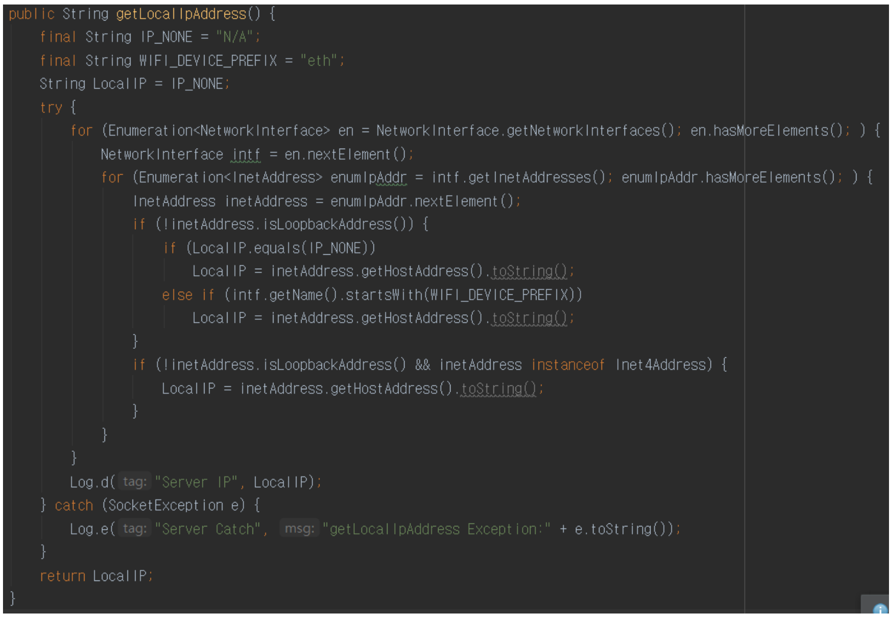


```
public InetAddress getBroadcastAddress() {
    InetAddress broadcastAddress = null;
    try {
        Enumeration<NetworkInterface> networkInterface = NetworkInterface.getNetworkInterfaces();
        while (broadcastAddress == null
                && networkInterface.hasMoreElements()) {
            NetworkInterface singleInterface = networkInterface
                    .nextElement();
            String interfaceName = singleInterface.getName();
            if (interfaceName.contains("wlan0")
                    || interfaceName.contains("eth0")) {
                for (InterfaceAddress infaceAddress : singleInterface.getInterfaceAddresses()) {
                    broadcastAddress = infaceAddress.getBroadcast();
                    if (broadcastAddress != null) {
                        break;
                    }
                }
            }
        }
    } catch (SocketException e) {
        e.printStackTrace();
    }
    return broadcastAddress;
}
```

Figure 14 Server Service Code

This method is to generate broadcastAddress variable to broad cast server IP address.

- getLocalAddress Method

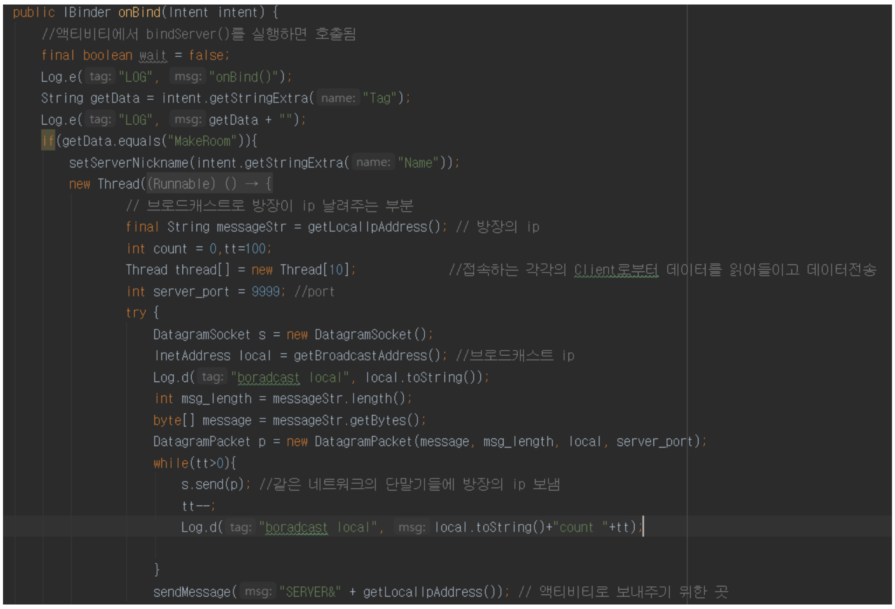


```
public String getLocalIpAddress() {
    final String IP_NONE = "N/A";
    final String WIFI_DEVICE_PREFIX = "eth";
    String LocalIP = IP_NONE;
    try {
        for (Enumeration<NetworkInterface> en = NetworkInterface.getNetworkInterfaces(); en.hasMoreElements(); ) {
            NetworkInterface intf = en.nextElement();
            for (Enumeration<InetAddress> enumIpAddr = intf.getInetAddresses(); enumIpAddr.hasMoreElements(); ) {
                InetAddress inetAddress = enumIpAddr.nextElement();
                if (!inetAddress.isLoopbackAddress()) {
                    if (LocalIP.equals(IP_NONE))
                        LocalIP = inetAddress.getHostAddress().toString();
                    else if (intf.getName().startsWith(WIFI_DEVICE_PREFIX))
                        LocalIP = inetAddress.getHostAddress().toString();
                }
                if (!inetAddress.isLoopbackAddress() && inetAddress instanceof Inet4Address) {
                    LocalIP = inetAddress.getHostAddress().toString();
                }
            }
        }
        Log.d( tag: "Server IP", LocalIP);
    } catch (SocketException e) {
        Log.e( tag: "Server Catch",  msg: "getLocalIpAddress Exception:" + e.toString());
    }
    return LocalIP;
}
```

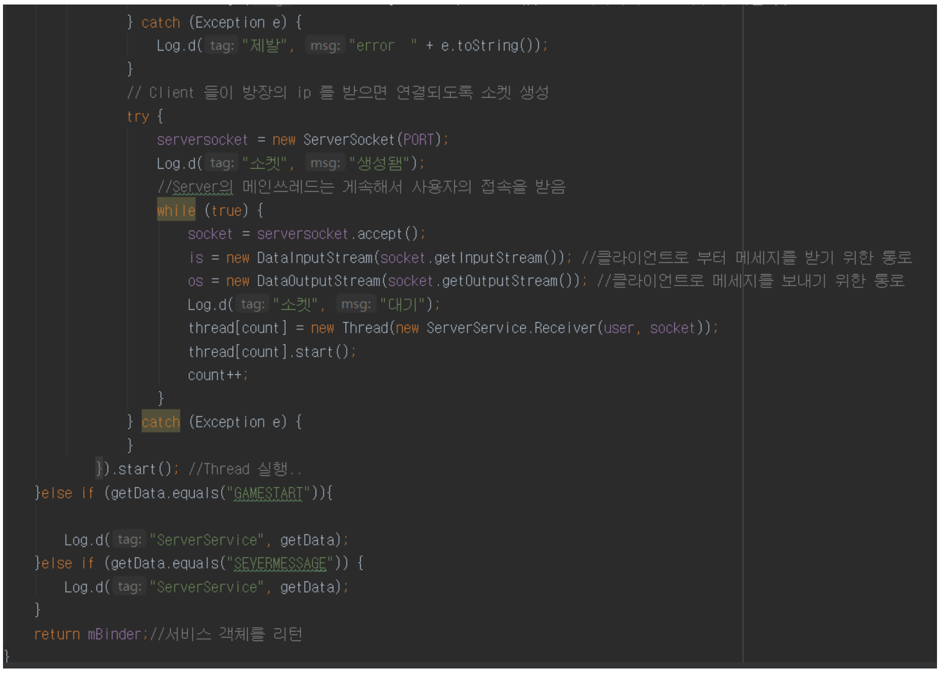
Figure 15 Server Service Code

It returns device's local IP address that used to broad cast.

- onBind Method



```
public IBinder onBind(Intent intent) {
    //액티비티에서 bindServer()를 실행하면 호출됨
    final boolean wait = false;
    Log.e( tag: "LOG",  msg: "onBind()");
    String getData = intent.getStringExtra( name: "Tag");
    Log.e( tag: "LOG",  msg: getData + "");
    if(getData.equals("MakeRoom")){
        setServerNickname(intent.getStringExtra( name: "Name"));
        new Thread((Runnable) () → {
                // 브로드캐스트로 방장이 ip 날려주는 부분
                final String messageStr = getLocalIpAddress(); // 방장의 ip
                int count = 0,tt=100;
                Thread thread[] = new Thread[10];             //접속하는 각각의 Client로부터 데이터를 읽어들이고 데이터전송
                int server_port = 9999; //port
                try {
                    DatagramSocket s = new DatagramSocket();
                    InetAddress local = getBroadcastAddress(); //브로드캐스트 ip
                    Log.d( tag: "boradcast local", local.toString());
                    int msg_length = messageStr.length();
                    byte[] message = messageStr.getBytes();
                    DatagramPacket p = new DatagramPacket(message, msg_length, local, server_port);
                    while(tt>0){
                        s.send(p); //같은 네트워크의 단말기들에 방장의 ip 보냄
                        tt--;
                        Log.d( tag: "boradcast local",  msg: local.toString()+"count "+tt);

                    }
                    sendMessage( msg: "SERVER&" + getLocalIpAddress()); // 액티비티로 보내주기 위한 곳
```

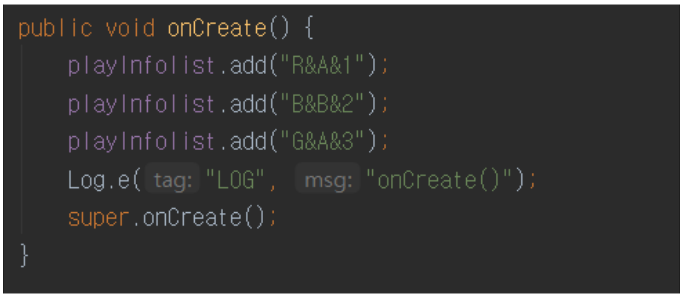


```
                } catch (Exception e) {
                    Log.d( tag: "제발",  msg: "error  " + e.toString());
                }
                // Client 들이 방장의 ip 를 받으면 연결되도록 소켓 생성
                try {
                    servversocket = new ServerSocket(PORT);
                    Log.d( tag: "소켓",  msg: "생성됨");
                    //Server의 메인쓰레드는 계속해서 사용자의 접속을 받음
                    while (true) {
                        socket = serversocket.accept();
                        is = new DataInputStream(socket.getInputStream()); //클라이언트로 부터 메세지를 받기 위한 통로
                        os = new DataOutputStream(socket.getOutputStream()); //클라이언트로 메세지를 보내기 위한 통로
                        Log.d( tag: "소켓",  msg: "대기");
                        thread[count] = new Thread(new ServerService.Receiver(user, socket));
                        thread[count].start();
                        count++;
                    }
                } catch (Exception e) {
                }
            }).start(); //Thread 실행..
    }else if (getData.equals("GAMESTART")){

        Log.d( tag: "ServerService", getData);
    }else if (getData.equals("SEVERMESSAGE")) {
        Log.d( tag: "ServerService", getData);
    }
    return mBinder;//서비스 객체를 리턴
}
```

Figure 16 Server Service Code

onBind method is a function of Android Service. This acts as data communication between Service object and Activity.

- onCreate Method

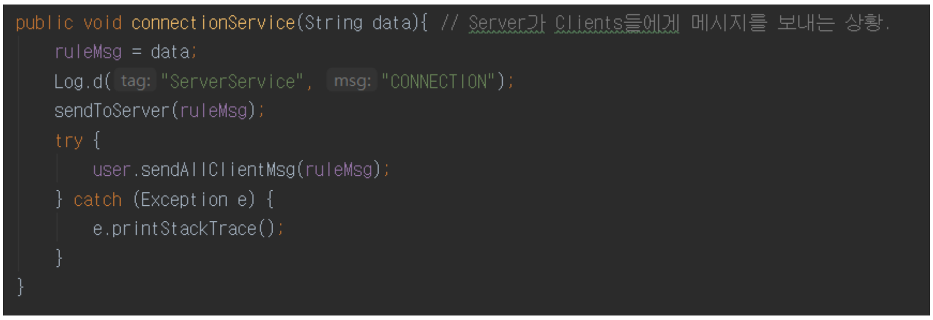


```
public void onCreate() {
    playInfolist.add("R&A&1");
    playInfolist.add("B&B&2");
    playInfolist.add("G&A&3");
    Log.e( tag: "LOG",  msg: "onCreate()");
    super.onCreate();
}
```

Figure 17 Server Service Code

onCreate method implemented for adding players as the first function to run the service.

- connectionService Method

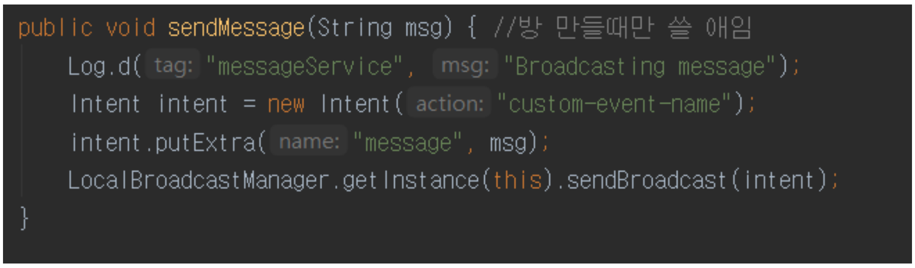


```java
public void connectionService(String data){ // Server가 Clients들에게 메시지를 보내는 상황.
    ruleMsg = data;
    Log.d( tag: "ServerService",  msg: "CONNECTION");
    sendToServer(ruleMsg);
    try {
        user.sendAllClientMsg(ruleMsg);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Figure 18 Server Service Code

connectionService function sends information about the game situation from the server to all clients and to the server itself, including messages sent by the game player.

- sendMessage Method

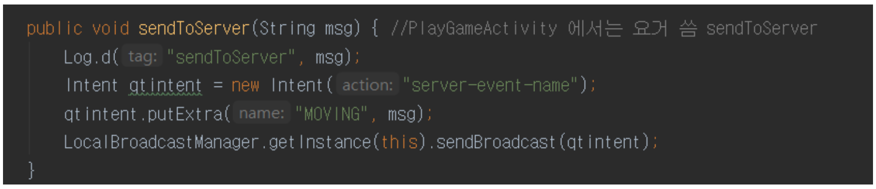


```java
public void sendMessage(String msg) { //방 만들때만 쓸 애임
    Log.d( tag: "messageService",  msg: "Broadcasting message");
    Intent intent = new Intent( action: "custom-event-name");
    intent.putExtra( name: "message", msg);
    LocalBroadcastManager.getInstance(this).sendBroadcast(intent);
}
```

Figure 19 Server Service Code

If the server player clicks the game start button, all clients and server's screen is changed to play screen, sending the nickname of each player to server.

- sendToServer Method

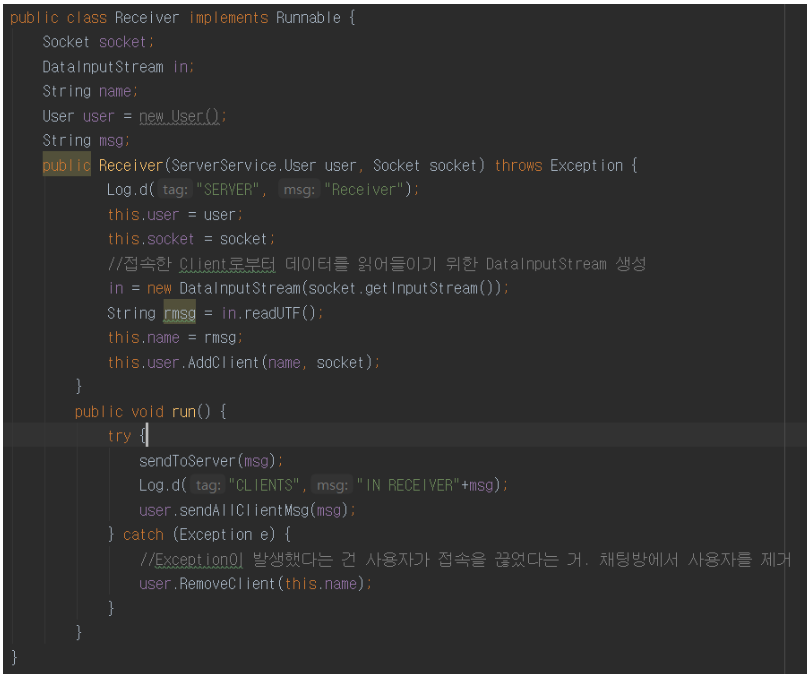


```java
public void sendToServer(String msg) { //PlayGameActivity 에서는 요거 씀 sendToServer
    Log.d( tag: "sendToServer", msg);
    Intent qtintent = new Intent( action: "server-event-name");
    qtintent.putExtra( name: "MOVING", msg);
    LocalBroadcastManager.getInstance(this).sendBroadcast(qtintent);
}
```

Figure 20 Server Service Code

sendToServer method sends messages from the clients to server using LocalBroadcastManager.

- Receiver Class (Inner Class)

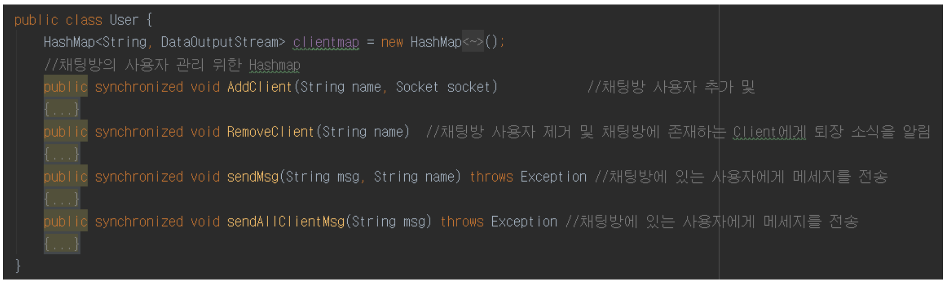


```java
public class Receiver implements Runnable {
    Socket socket;
    DataInputStream in;
    String name;
    User user = new User();
    String msg;
    public Receiver(ServerService.User user, Socket socket) throws Exception {
            Log.d( tag: "SERVER",  msg: "Receiver");
            this.user = user;
            this.socket = socket;
            //접속한 Client로부터 데이터를 읽어들이기 위한 DataInputStream 생성
            in = new DataInputStream(socket.getInputStream());
            String rmsg = in.readUTF();
            this.name = rmsg;
            this.user.AddClient(name, socket);
        }
        public void run() {
            try {
                sendToServer(msg);
                Log.d( tag: "CLIENTS",  msg: "IN RECEIVER"+msg);
                user.sendAllClientMsg(msg);
            } catch (Exception e) {
                //Exception이 발생했다는 건 사용자가 접속을 끊었다는 거. 채팅방에서 사용자를 제거
                user.RemoveClient(this.name);
            }
        }
}
```

Figure 21 Receiver Class

A Receiver class is received when a server receives data from clients using DataInputStream.

3) Server Service - User Class

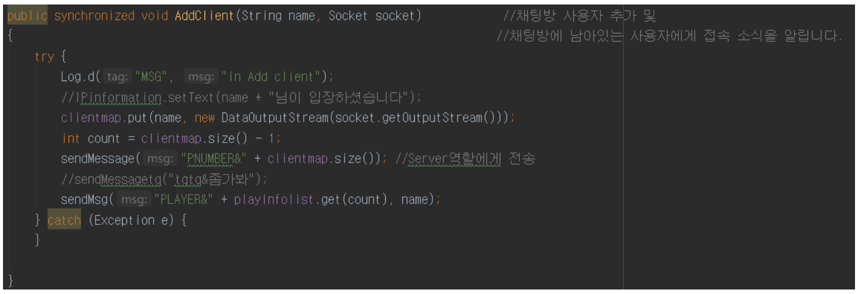


```java
public class User {
    HashMap<String, DataOutputStream> clientmap = new HashMap<~>();
    //채팅방의 사용자 관리 위한 Hashmap
    public synchronized void AddClient(String name, Socket socket)          //채팅방 사용자 추가 및
    {...}
    public synchronized void RemoveClient(String name)  //채팅방 사용자 제거 및 채팅방에 존재하는 Client에게 퇴장 소식을 알림
    {...}
    public synchronized void sendMsg(String msg, String name) throws Exception //채팅방에 있는 사용자에게 메세지를 전송
    {...}
    public synchronized void sendAllClientMsg(String msg) throws Exception //채팅방에 있는 사용자에게 메세지를 전송
    {...}
}
```

Figure 22 User Class – Overview

User class stores players entered as clients with user objects.

- AddClient Method

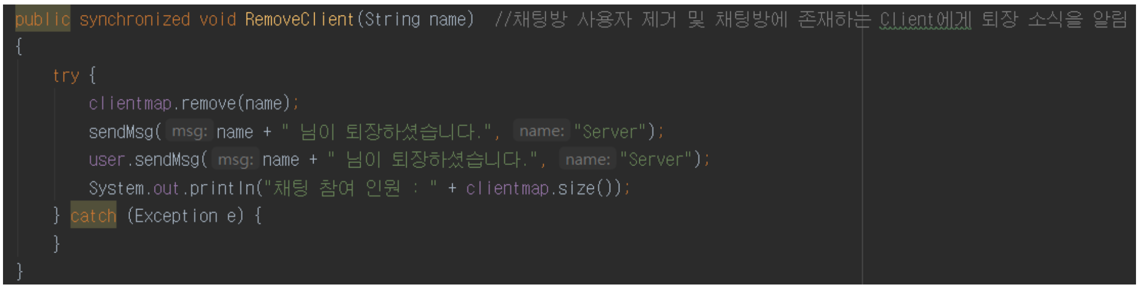


```
public synchronized void AddClient(String name, Socket socket)          //채팅방 사용자 추가 및
{                                                                      //채팅방에 남아있는 사용자에게 접속 소식을 알립니다.
    try {
        Log.d( tag: "MSG",  msg: "in Add client");
        //IPinformation.setText(name + "님이 입장하셨습니다");
        clientmap.put(name, new DataOutputStream(socket.getOutputStream()));
        int count = clientmap.size() - 1;
        sendMessage( msg: "PNUMBER&" + clientmap.size()); //Server역할에게 전송
        //sendMessagetg("tgtg&좀가봐");
        sendMsg( msg: "PLAYER&" + playInfolist.get(count), name);
    } catch (Exception e) {
    }

}
```

Figure 23 User Class Code

Addclient method is an internal method of the user class. It is a function that adds a new player entered to game.

- Remove Client Method

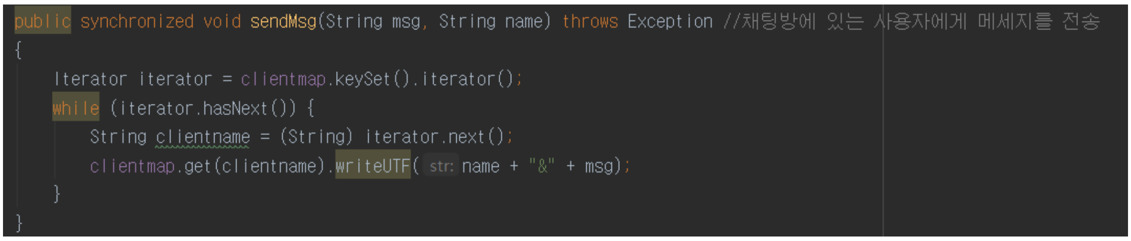


```
public synchronized void RemoveClient(String name)  //채팅방 사용자 제거 및 채팅방에 존재하는 Client에게 퇴장 소식을 알림
{
    try {
        clientmap.remove(name);
        sendMsg( msg: name + " 님이 퇴장하셨습니다.",  name: "Server");
        user.sendMsg( msg: name + " 님이 퇴장하셨습니다.",  name: "Server");
        System.out.println("채팅 참여 인원 : " + clientmap.size());
    } catch (Exception e) {
    }
}
```

Figure 24 User Class Code

RemoveClient method is also internal method of the user class. It deletes only the out player when each player finishes the game.

- sendMsg Method

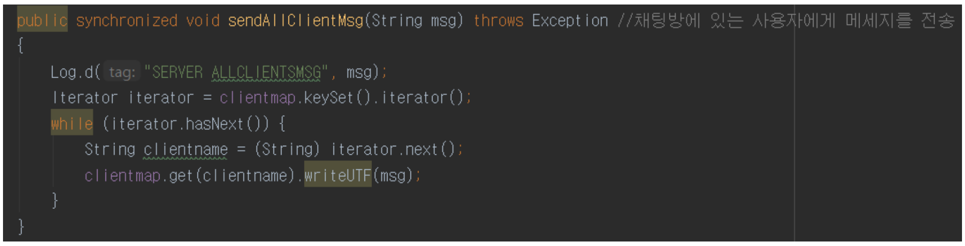


```
public synchronized void sendMsg(String msg, String name) throws Exception //채팅방에 있는 사용자에게 메세지를 전송
{
    Iterator iterator = clientmap.keySet().iterator();
    while (iterator.hasNext()) {
        String clientname = (String) iterator.next();
        clientmap.get(clientname).writeUTF( str: name + "&" + msg);
    }
}
```

Figure 25 User Class Code

sendMsg method sends messages to all clients entered to game.

- sendAllClients Method



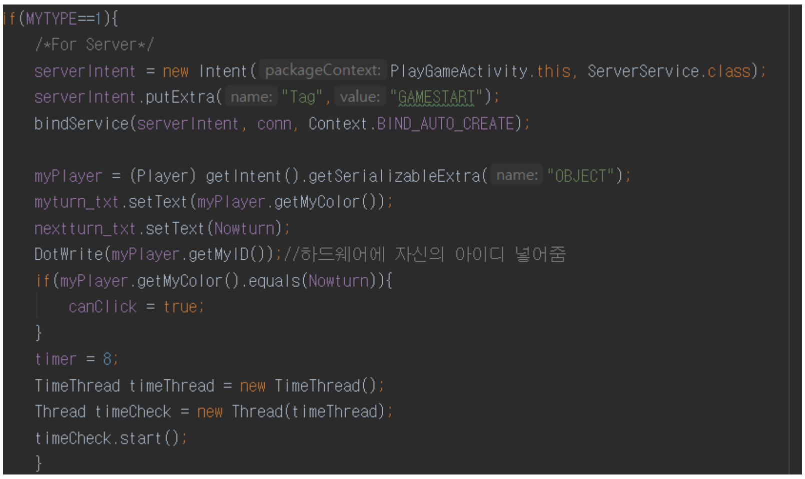
```java
public synchronized void sendAllClientMsg(String msg) throws Exception //채팅방에 있는 사용자에게 메세지를 전송
{
    Log.d( tag: "SERVER ALLCLIENTSMSG", msg);
    Iterator iterator = clientmap.keySet().iterator();
    while (iterator.hasNext()) {
        String clientname = (String) iterator.next();
        clientmap.get(clientname).writeUTF(msg);
    }
}
```

Figure 26 User Class Code

4) Play Game Activity's Server Part

We have handled the communication situation in the Play Game as follows : In particular, this activity required both the server and the clients, so the first things to do was separate the types (client / server) and make logic to get the game going.

- Type Check(Server)

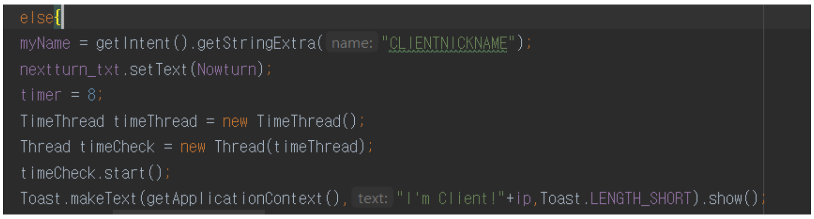


```java
if(MYTYPE==1){
    /*For Server*/
    serverIntent = new Intent( packageContext: PlayGameActivity.this, ServerService.class);
    serverIntent.putExtra( name: "Tag", value: "GAMESTART");
    bindService(serverIntent, conn, Context.BIND_AUTO_CREATE);

    myPlayer = (Player) getIntent().getSerializableExtra( name: "OBJECT");
    myturn_txt.setText(myPlayer.getMyColor());
    nextturn_txt.setText(Nowturn);
    DotWrite(myPlayer.getMyID());//하드웨어에 자신의 아이디 넣어줌
    if(myPlayer.getMyColor().equals(Nowturn)){
        canClick = true;
    }
    timer = 8;
    TimeThread timeThread = new TimeThread();
    Thread timeCheck = new Thread(timeThread);
    timeCheck.start();
    }
```

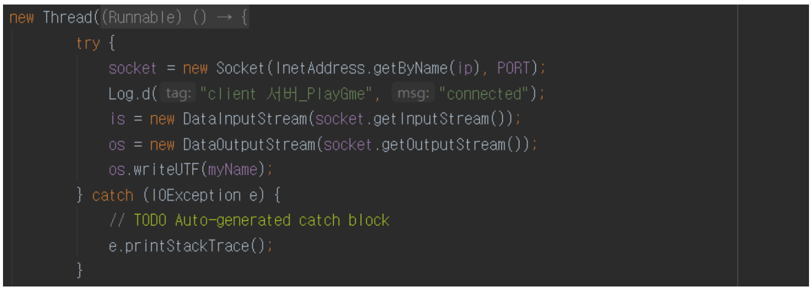
Figure 27 Play Game Class Code

If the screen is the server's game screen, we can tell it's the server by setting MYTYPE == 1.

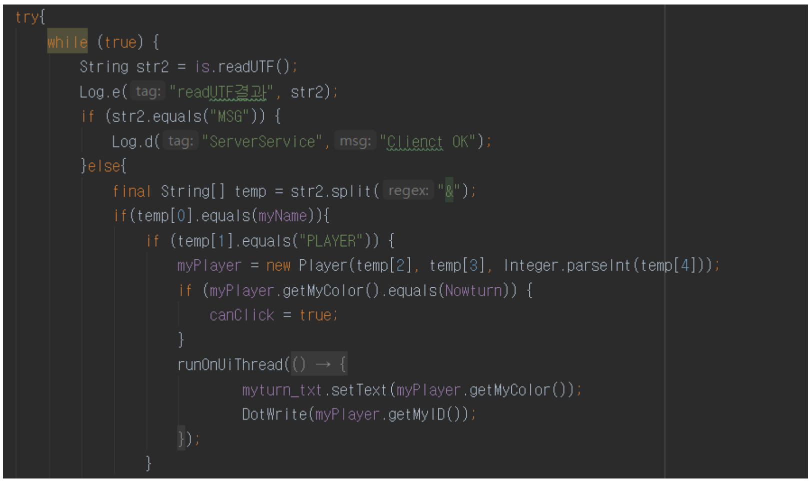
- Type Check(Client)



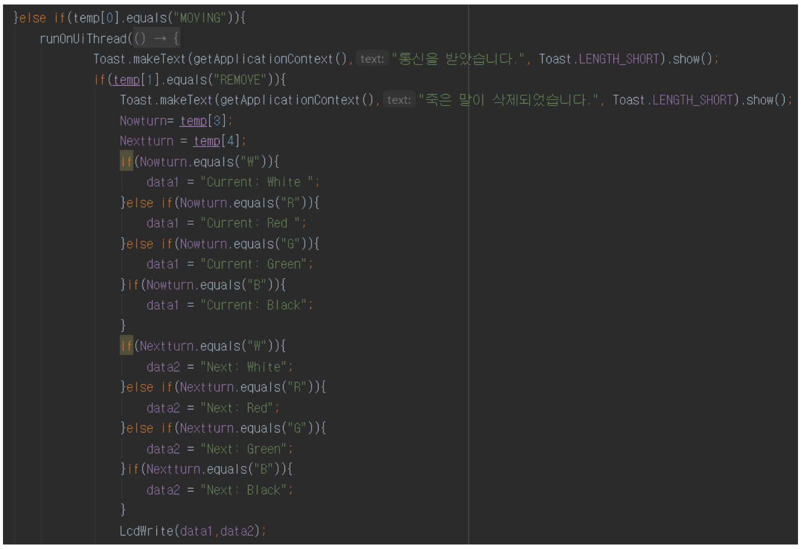
```
else{
myName = getIntent().getStringExtra( name: "CLIENTNICKNAME");
nextturn_txt.setText(Nowturn);
timer = 8;
TimeThread timeThread = new TimeThread();
Thread timeCheck = new Thread(timeThread);
timeCheck.start();
Toast.makeText(getApplicationContext(), text: "I'm Client!"+ip,Toast.LENGTH_SHORT).show();
```



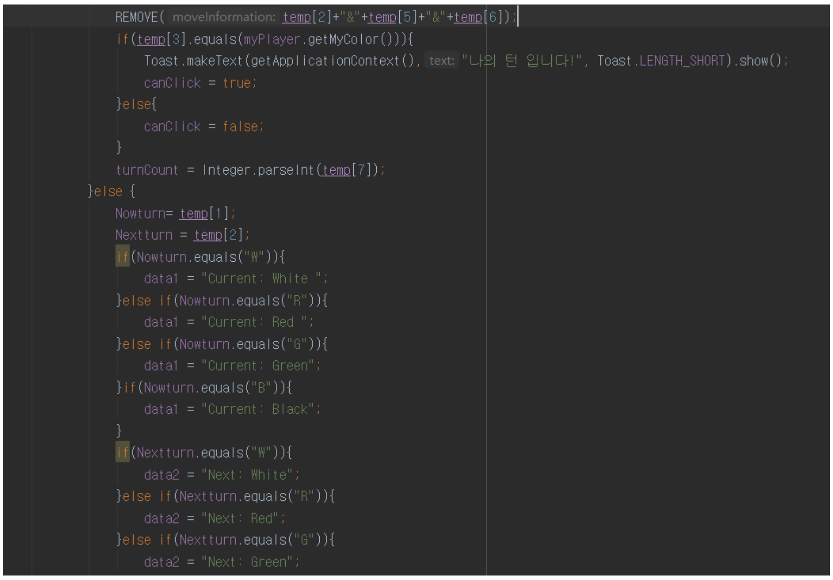
```
new Thread((Runnable) () → {
        try {
            socket = new Socket(InetAddress.getByName(ip), PORT);
            Log.d( tag: "client 서버_PlayGme", msg: "connected");
            is = new DataInputStream(socket.getInputStream());
            os = new DataOutputStream(socket.getOutputStream());
            os.writeUTF(myName);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
```



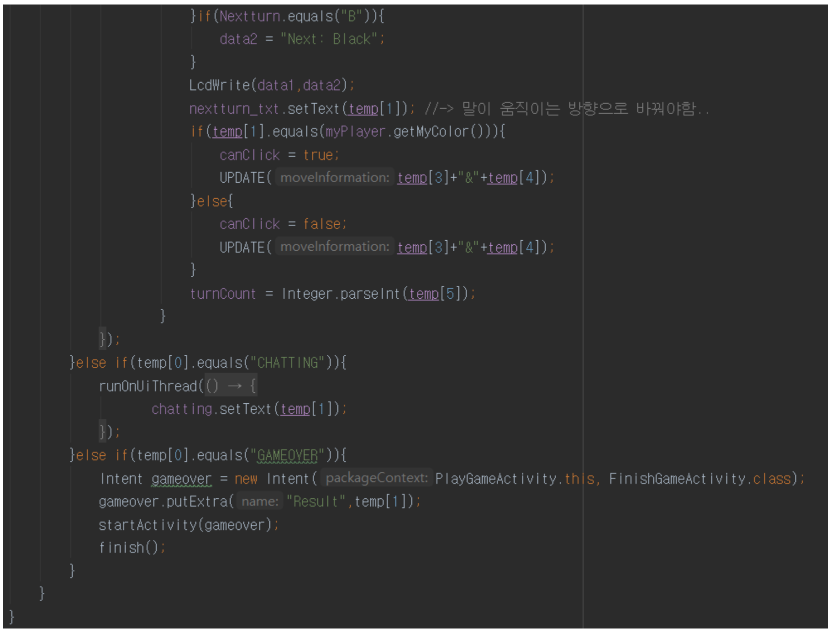
```
try{
    while (true) {
        String str2 = is.readUTF();
        Log.e( tag: "readUTF결과", str2);
        if (str2.equals("MSG")) {
            Log.d( tag: "ServerService", msg: "Cliienct OK");
        }else{
            final String[] temp = str2.split( regex: "&");
            if(temp[0].equals(myName)){
                if (temp[1].equals("PLAYER")) {
                    myPlayer = new Player(temp[2], temp[3], Integer.parseInt(temp[4]));
                    if (myPlayer.getMyColor().equals(Nowturn)) {
                        canClick = true;
                    }
                    runOnUiThread(() → {
                            myturn_txt.setText(myPlayer.getMyColor());
                            DotWrite(myPlayer.getMyID());
                    });
                }
```



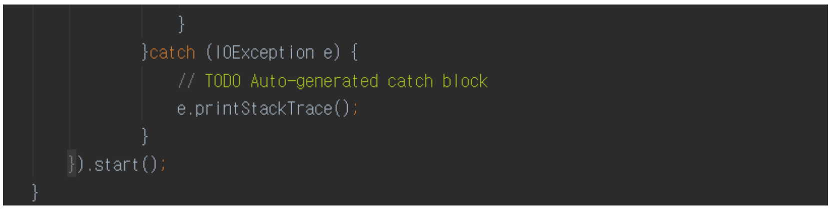
```
}else if(temp[0].equals("MOVING")){
    runOnUiThread(() → {
            Toast.makeText(getApplicationContext(), text: "통신을 받았습니다.", Toast.LENGTH_SHORT).show();
            if(temp[1].equals("REMOVE")){
                Toast.makeText(getApplicationContext(), text: "죽은 말이 삭제되었습니다.", Toast.LENGTH_SHORT).show();
                Nowturn= temp[3];
                Nextturn = temp[4];
                if(Nowturn.equals("W")){
                    data1 = "Current: White ";
                }else if(Nowturn.equals("R")){
                    data1 = "Current: Red ";
                }else if(Nowturn.equals("G")){
                    data1 = "Current: Green";
                }if(Nowturn.equals("B")){
                    data1 = "Current: Black";
                }
                if(Nextturn.equals("W")){
                    data2 = "Next: White";
                }else if(Nextturn.equals("R")){
                    data2 = "Next: Red";
                }else if(Nextturn.equals("G")){
                    data2 = "Next: Green";
                }if(Nextturn.equals("B")){
                    data2 = "Next: Black";
                }
                LcdWrite(data1,data2);
```



```
            REMOVE( moveInformation: temp[2]+"&"+temp[5]+"&"+temp[6]);
            if(temp[3].equals(myPlayer.getMyColor())){
                Toast.makeText(getApplicationContext(), text: "나의 턴 입니다!", Toast.LENGTH_SHORT).show();
                canClick = true;
            }else{
                canClick = false;
            }
            turnCount = Integer.parseInt(temp[7]);
        }else {
            Nowturn= temp[1];
            Nextturn = temp[2];
            if(Nowturn.equals("W")){
                data1 = "Current: White ";
            }else if(Nowturn.equals("R")){
                data1 = "Current: Red ";
            }else if(Nowturn.equals("G")){
                data1 = "Current: Green";
            }if(Nowturn.equals("B")){
                data1 = "Current: Black";
            }
            if(Nextturn.equals("W")){
                data2 = "Next: White";
            }else if(Nextturn.equals("R")){
                data2 = "Next: Red";
            }else if(Nextturn.equals("G")){
                data2 = "Next: Green";
```



```
                    }if(Nextturn.equals("B")){
                        data2 = "Next: Black";
                    }
                    LcdWrite(data1,data2);
                    nextturn_txt.setText(temp[1]); //-> 말이 움직이는 방향으로 바꿔야함..
                    if(temp[1].equals(myPlayer.getMyColor())){
                        canClick = true;
                        UPDATE( moveInformation: temp[3]+"&"+temp[4]);
                    }else{
                        canClick = false;
                        UPDATE( moveInformation: temp[3]+"&"+temp[4]);
                    }
                    turnCount = Integer.parseInt(temp[5]);
                }
            });
        }else if(temp[0].equals("CHATTING")){
            runOnUiThread(() → {
                    chatting.setText(temp[1]);
            });
        }else if(temp[0].equals("GAMEOVER")){
            Intent gameover = new Intent( packageContext: PlayGameActivity.this, FinishGameActivity.class);
            gameover.putExtra( name: "Result",temp[1]);
            startActivity(gameover);
            finish();
        }
    }
}
```

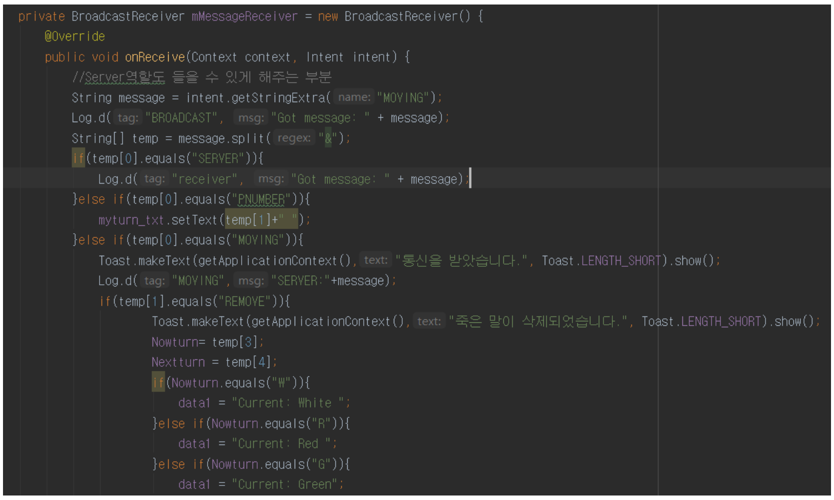


```
                }
            }catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
    }).start();
}
```

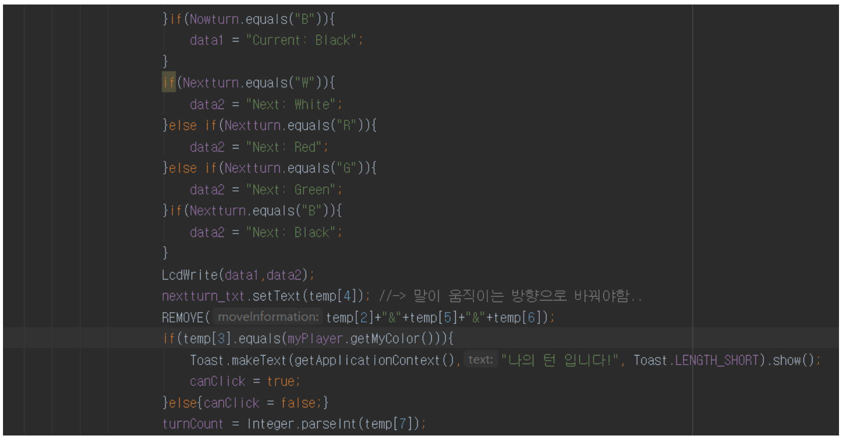
Figure 28 Play Game Class Code

On the other hand, if it's a client screen, we've got MYTYPE = 0, and we've implemented socket communicatinon with the server. And cli>ents we kept getting message form the server.

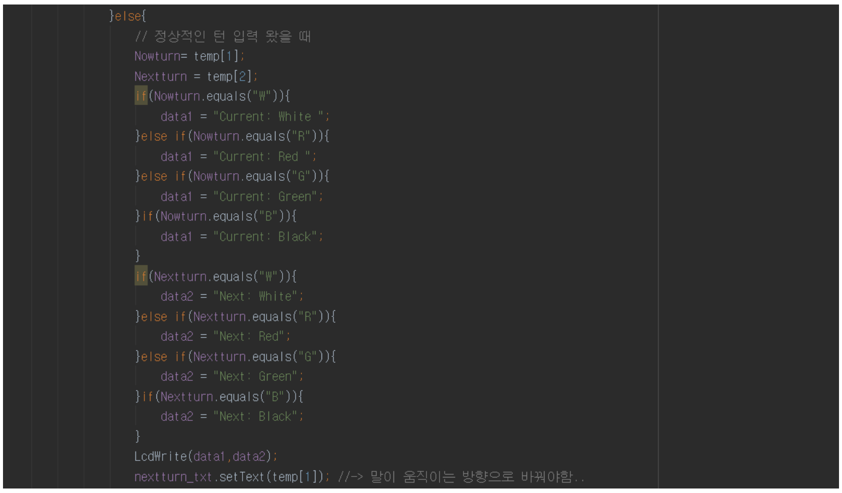
- Broadcast Receiver Method



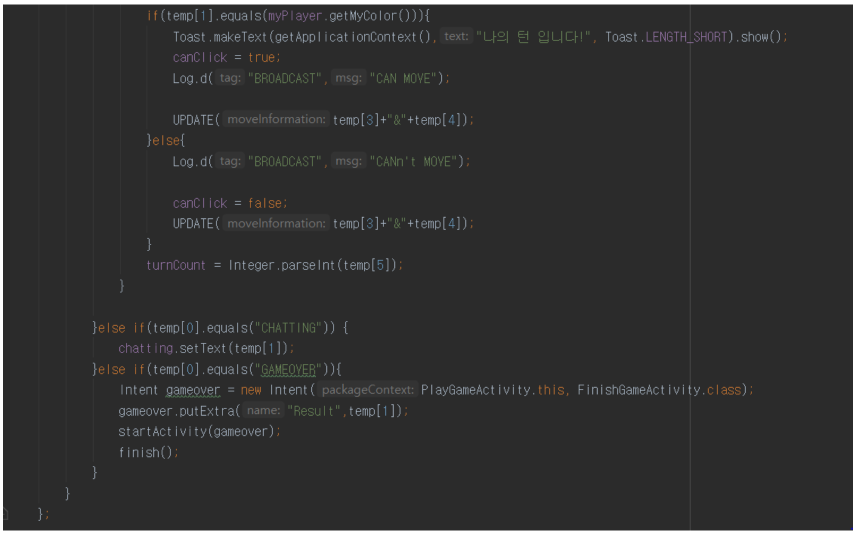
```
private BroadcastReceiver mMessageReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        //Server역할도 들을 수 있게 해주는 부분
        String message = intent.getStringExtra( name: "MOVING");
        Log.d( tag: "BROADCAST",  msg: "Got message: " + message);
        String[] temp = message.split( regex: "&");
        if(temp[0].equals("SERVER")){
            Log.d( tag: "receiver",  msg: "Got message: " + message);
        }else if(temp[0].equals("PNUMBER")){
            myturn_txt.setText(temp[1]+" ");
        }else if(temp[0].equals("MOVING")){
            Toast.makeText(getApplicationContext(), text: "통신을 받았습니다.", Toast.LENGTH_SHORT).show();
            Log.d( tag: "MOVING", msg: "SERVER:"+message);
            if(temp[1].equals("REMOVE")){
                    Toast.makeText(getApplicationContext(), text: "죽은 말이 삭제되었습니다.", Toast.LENGTH_SHORT).show();
                    Nowturn= temp[3];
                    Nextturn = temp[4];
                    if(Nowturn.equals("W")){
                        data1 = "Current: White ";
                    }else if(Nowturn.equals("R")){
                        data1 = "Current: Red ";
                    }else if(Nowturn.equals("G")){
                        data1 = "Current: Green";
```



```
                    }if(Nowturn.equals("B")){
                        data1 = "Current: Black";
                    }
                    if(Nextturn.equals("W")){
                        data2 = "Next: White";
                    }else if(Nextturn.equals("R")){
                        data2 = "Next: Red";
                    }else if(Nextturn.equals("G")){
                        data2 = "Next: Green";
                    }if(Nextturn.equals("B")){
                        data2 = "Next: Black";
                    }
                    LcdWrite(data1,data2);
                    nextturn_txt.setText(temp[4]); //-> 말이 움직이는 방향으로 바꿔야함..
                    REMOVE( moveInformation: temp[2]+"&"+temp[5]+"&"+temp[6]);
                    if(temp[3].equals(myPlayer.getMyColor())){
                        Toast.makeText(getApplicationContext(), text: "나의 턴 입니다!", Toast.LENGTH_SHORT).show();
                        canClick = true;
                    }else{canClick = false;}
                    turnCount = Integer.parseInt(temp[7]);
```



```
            }else{
                // 정상적인 턴 입력 왔을 때
                Nowturn= temp[1];
                Nextturn = temp[2];
                if(Nowturn.equals("W")){
                    data1 = "Current: White ";
                }else if(Nowturn.equals("R")){
                    data1 = "Current: Red ";
                }else if(Nowturn.equals("G")){
                    data1 = "Current: Green";
                }if(Nowturn.equals("B")){
                    data1 = "Current: Black";
                }
                if(Nextturn.equals("W")){
                    data2 = "Next: White";
                }else if(Nextturn.equals("R")){
                    data2 = "Next: Red";
                }else if(Nextturn.equals("G")){
                    data2 = "Next: Green";
                }if(Nextturn.equals("B")){
                    data2 = "Next: Black";
                }
                LcdWrite(data1,data2);
                nextturn_txt.setText(temp[1]); //-> 말이 움직이는 방향으로 바꿔야함..
```

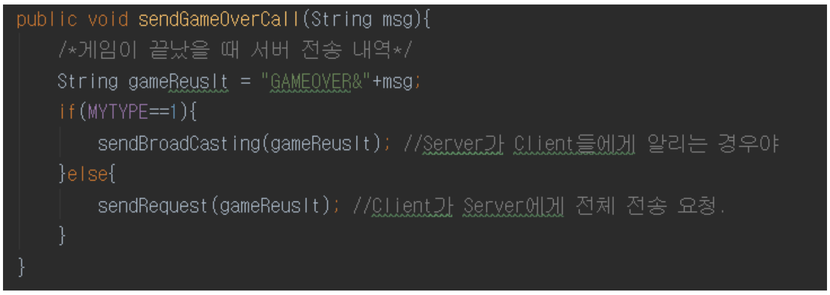


```
                if(temp[1].equals(myPlayer.getMyColor())){
                    Toast.makeText(getApplicationContext(), text: "나의 턴 입니다!", Toast.LENGTH_SHORT).show();
                    canClick = true;
                    Log.d( tag: "BROADCAST", msg: "CAN MOVE");

                    UPDATE( moveInformation: temp[3]+"&"+temp[4]);
                }else{
                    Log.d( tag: "BROADCAST", msg: "CANn't MOVE");

                    canClick = false;
                    UPDATE( moveInformation: temp[3]+"&"+temp[4]);
                }
                turnCount = Integer.parseInt(temp[5]);
            }

        }else if(temp[0].equals("CHATTING")) {
            chatting.setText(temp[1]);
        }else if(temp[0].equals("GAMEOVER")){
            Intent gameover = new Intent( packageContext: PlayGameActivity.this, FinishGameActivity.class);
            gameover.putExtra( name: "Result",temp[1]);
            startActivity(gameover);
            finish();
        }
    }
};
```

Figure 29 Play Game Class Code

Similar to the format in which a message was received from a server, the client side is also needed to receive the message such as server side. We implemented it using BroadCastReceiver, and we separated the messages client received into tags to see what they were.

- sendGameOverCall Method

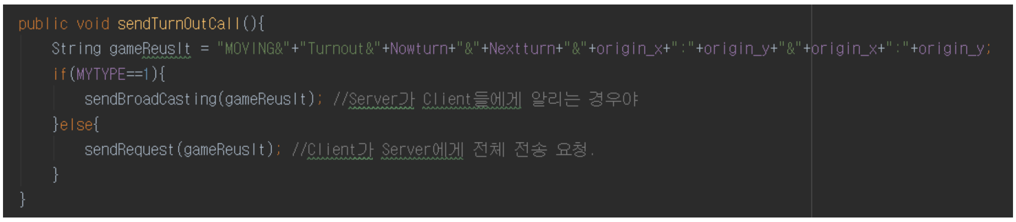


```
public void sendGameOverCall(String msg){
    /*게임이 끝났을 때 서버 전송 내역*/
    String gameReuslt = "GAMEOVER&"+msg;
    if(MYTYPE==1){
        sendBroadCasting(gameReuslt); //Server가 Client들에게 알리는 경우야
    }else{
        sendRequest(gameReuslt); //Client가 Server에게 전체 전송 요청.
    }
}
```

Figure 30 Play Game Class Code

sendGameOverCall method sent “game over” information. We have three game over state. “Team A is Win” or “Team B is Win” or “Draw“.

- sendTurnOutCall Method

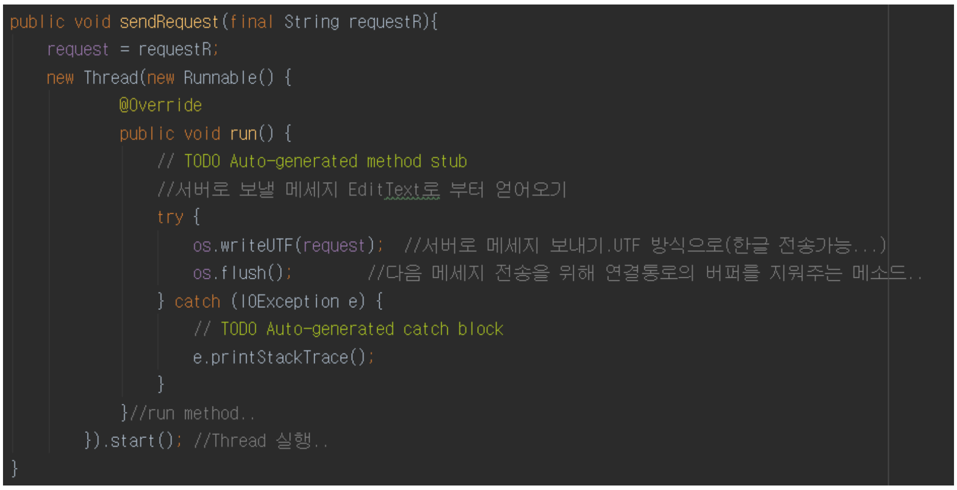


```
public void sendTurnOutCall(){
    String gameReuslt = "MOVING&"+"Turnout&"+Nowturn+"&"+Nextturn+"&"+origin_x+":"+origin_y+"&"+origin_x+":"+origin_y;
    if(MYTYPE==1){
        sendBroadCasting(gameReuslt); //Server가 Client들에게 알리는 경우야
    }else{
        sendRequest(gameReuslt); //Client가 Server에게 전체 전송 요청.
    }
}
```

Figure 31 Play Game Class Code

sendTurnOutCall method sends ”turn out” information.

- sendRequest Method



```
public void sendRequest(final String requestR){
    request = requestR;
    new Thread(new Runnable() {
        @Override
        public void run() {
            // TODO Auto-generated method stub
            //서버로 보낼 메세지 EditText로 부터 얻어오기
            try {
                os.writeUTF(request);  //서버로 메세지 보내기.UTF 방식으로(한글 전송가능...)
                os.flush();         //다음 메세지 전송을 위해 연결통로의 버퍼를 지워주는 메소드..
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }//run method..
    }).start(); //Thread 실행..
}
```

Figure 32 Play Game Class Code

sendRequest method from the client code sends messages to server.

3.4 Hardware

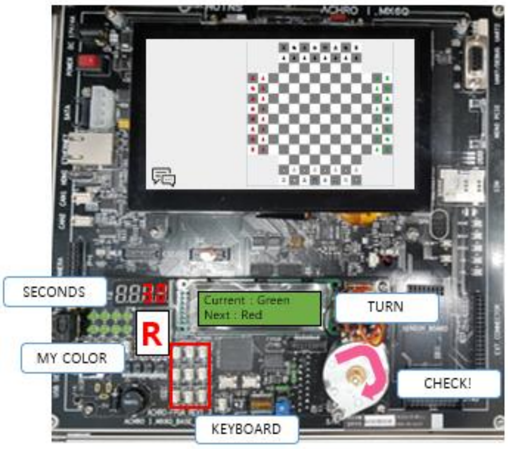
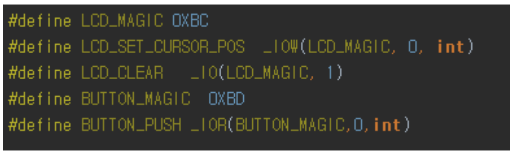


Figure 33 Hardware Overview

We implemented 5 functions of hardware that you can see on Figure 28. Details as follows,

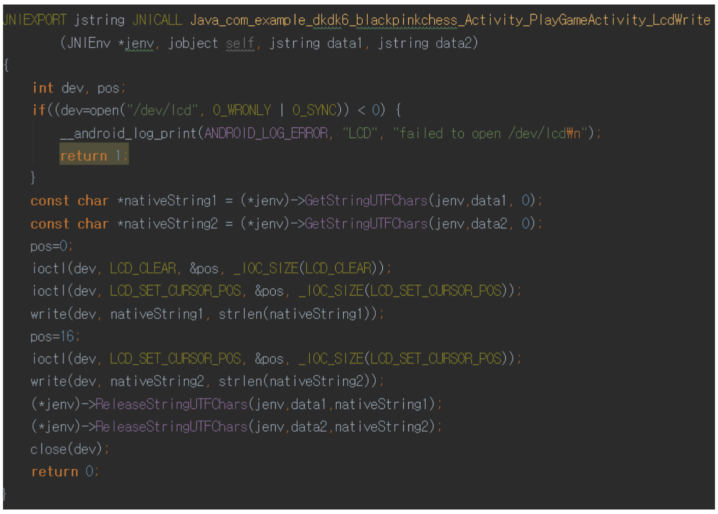


```
#define LCD_MAGIC OXBC
#define LCD_SET_CURSOR_POS  _IOW(LCD_MAGIC, 0, int)
#define LCD_CLEAR   _IO(LCD_MAGIC, 1)
#define BUTTON_MAGIC  OXBD
#define BUTTON_PUSH _IOR(BUTTON_MAGIC,0,int)
```

Figure 34 Magic numbers

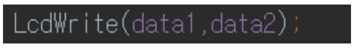
We used ioctl when implementing the LCD and buttons for chat. The magic number in Figure 29 was used in the corresponding ioctl implementation.

- LCD



```
JNIEXPORT jstring JNICALL Java_com_example_dkdk6_blackpinkchess_Activity_PlayGameActivity_LcdWrite
        (JNIEnv *jenv, jobject self, jstring data1, jstring data2)
{
    int dev, pos;
    if((dev=open("/dev/lcd", O_WRONLY | O_SYNC)) < 0) {
        __android_log_print(ANDROID_LOG_ERROR, "LCD", "failed to open /dev/lcd₩n");
        return 1;
    }
    const char *nativeString1 = (*jenv)->GetStringUTFChars(jenv,data1, 0);
    const char *nativeString2 = (*jenv)->GetStringUTFChars(jenv,data2, 0);
    pos=0;
    ioctl(dev, LCD_CLEAR, &pos, _IOC_SIZE(LCD_CLEAR));
    ioctl(dev, LCD_SET_CURSOR_POS, &pos, _IOC_SIZE(LCD_SET_CURSOR_POS));
    write(dev, nativeString1, strlen(nativeString1));
    pos=16;
    ioctl(dev, LCD_SET_CURSOR_POS, &pos, _IOC_SIZE(LCD_SET_CURSOR_POS));
    write(dev, nativeString2, strlen(nativeString2));
    (*jenv)->ReleaseStringUTFChars(jenv,data1,nativeString1);
    (*jenv)->ReleaseStringUTFChars(jenv,data2,nativeString2);
    close(dev);
    return 0;
}
```

Figure 35 LCD Write – Android JNI

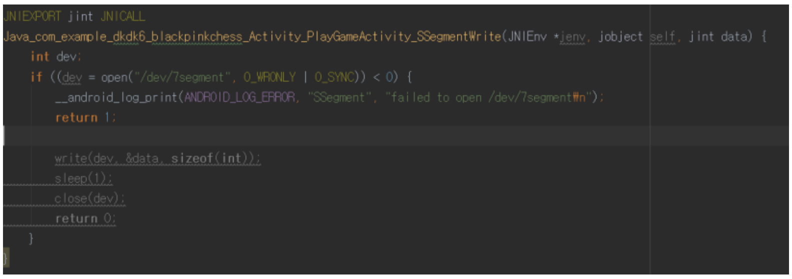


```
LcdWrite(data1,data2);
```

Figure 36 Lcd Write – Android

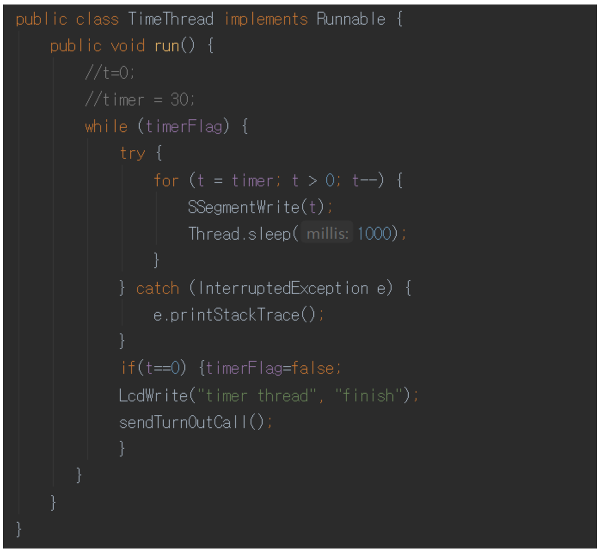
The LCD is used to display information about the turn of the current game. The first line shows what words are being used in the current Turn. For example, "Current: White" appears on the LCD. The second line shows which words should move in the next turn. For example, "Next: Red" appears on the LCD.

- 7-Segment



```
JNIEXPORT jint JNICALL
Java_com_example_dkdk6_blackpinkchess_Activity_PlayGameActivity_SSegmentWrite(JNIEnv *jenv, jobject self, jint data) {
    int dev;
    if ((dev = open("/dev/7segment", O_WRONLY | O_SYNC)) < 0) {
        __android_log_print(ANDROID_LOG_ERROR, "SSegment", "failed to open /dev/7segment₩n");
        return 1;

        write(dev, &data, sizeof(int));
        sleep(1);
        close(dev);
        return 0;
    }
}
```

Figure 37 7Segement Write – Android JNI

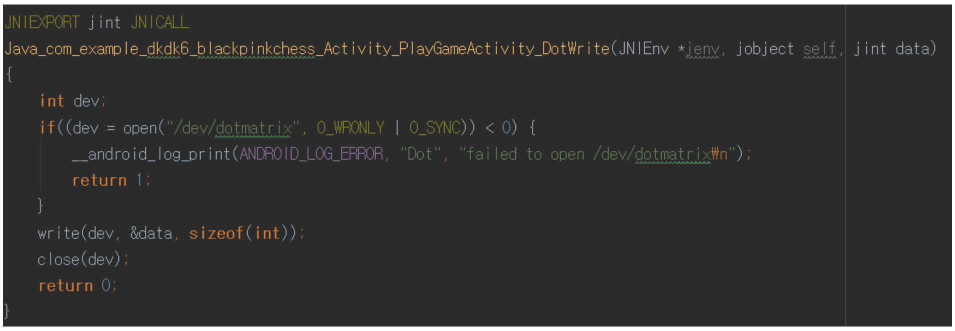


```
public class TimeThread implements Runnable {
    public void run() {
        //t=0;
        //timer = 30;
        while (timerFlag) {
            try {
                for (t = timer; t > 0; t--) {
                    SSegmentWrite(t);
                    Thread.sleep( millis: 1000);
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            if(t==0) {timerFlag=false;
            LcdWrite("timer thread", "finish");
            sendTurnOutCall();
            }
        }
    }
}
```

Figure 38 Time Out Thread - Android

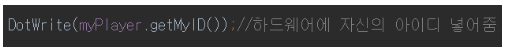
We have set a timer for each player turn. The game allows 30 seconds per turn, and after 30 seconds the turn is over.

- Dot Matrix



```
JNIEXPORT jint JNICALL
Java_com_example_dkdk6_blackpinkchess_Activity_PlayGameActivity_DotWrite(JNIEnv *jenv, jobject self, jint data)
{
    int dev;
    if((dev = open("/dev/dotmatrix", O_WRONLY | O_SYNC)) < 0) {
        __android_log_print(ANDROID_LOG_ERROR, "Dot", "failed to open /dev/dotmatrix₩n");
        return 1;
    }
    write(dev, &data, sizeof(int));
    close(dev);
    return 0;
}
```

Figure 39 Dot Write – Android JNI

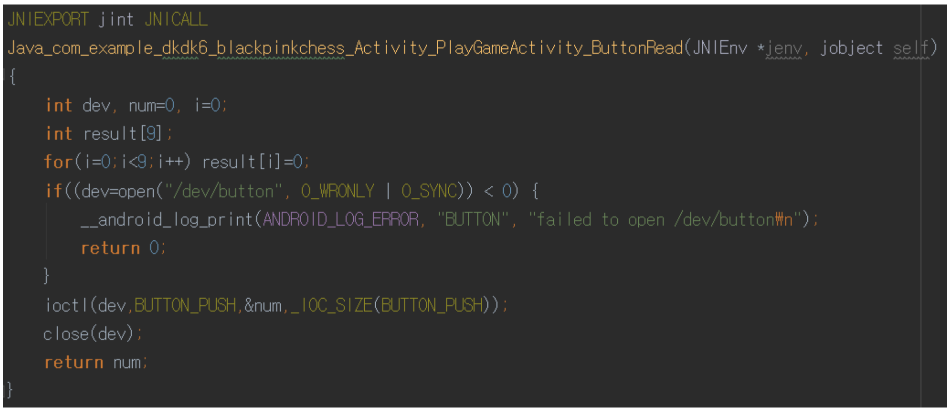


```
DotWrite(myPlayer.getMyID());//하드웨어에 자신의 아이디 넣어줌
```

Figure 40 Dot Write - Android

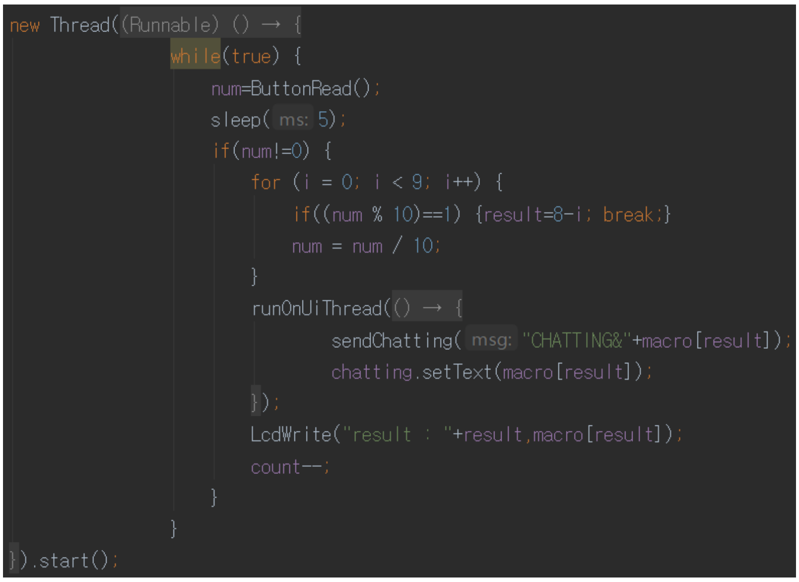
The player can see his or her horse information (color) as he/her enters the game through the Dot matrix. There are four states of display, "W, R, G, B".

- Keyboard(Push Button)



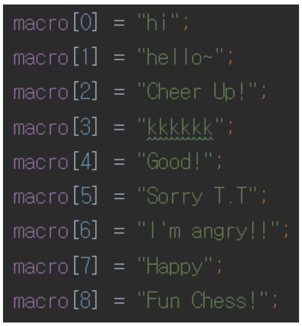
```
JNIEXPORT jint JNICALL
Java_com_example_dkdk6_blackpinkchess_Activity_PlayGameActivity_ButtonRead(JNIEnv *jenv, jobject self)
{
    int dev, num=0, i=0;
    int result[9];
    for(i=0;i<9;i++) result[i]=0;
    if((dev=open("/dev/button", O_WRONLY | O_SYNC)) < 0) {
        __android_log_print(ANDROID_LOG_ERROR, "BUTTON", "failed to open /dev/button₩n");
        return 0;
    }
    ioctl(dev,BUTTON_PUSH,&num,_IOC_SIZE(BUTTON_PUSH));
    close(dev);
    return num;
}
```

Figure 41 Button Read – Android JNI



```
new Thread((Runnable) () → {
                while(true) {
                    num=ButtonRead();
                    sleep( ms: 5);
                    if(num!=0) {
                        for (i = 0; i < 9; i++) {
                            if((num % 10)==1) {result=8-i; break;}
                            num = num / 10;
                        }
                        runOnUiThread(() → {
                                sendChatting( msg: "CHATTING&"+macro[result]);
                                chatting.setText(macro[result]);
                        });
                        LcdWrite("result : "+result,macro[result]);
                        count--;
                    }
                }
}).start();
```

Figure 42 Chatting Thread - Android

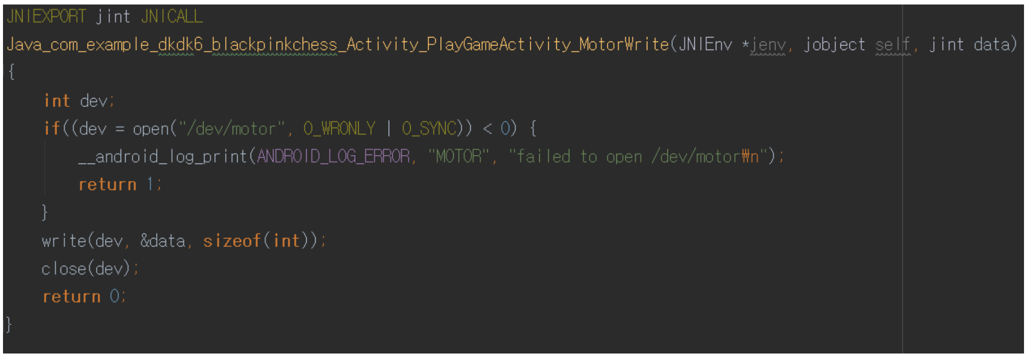


```
macro[0] = "hi";
macro[1] = "hello~";
macro[2] = "Cheer Up!";
macro[3] = "kkkkkk";
macro[4] = "Good!";
macro[5] = "Sorry T.T";
macro[6] = "I'm angry!!";
macro[7] = "Happy";
macro[8] = "Fun Chess!";
```

Figure 43 Macro Content - Android

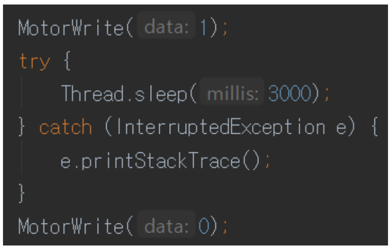
We have implemented the chat macro function using Push Button. We have implemented 9 macros in advance so that chatting can be sent by pressing each button. The contents of each can be checked through Figure 38.

- Motor



```
JNIEXPORT jint JNICALL
Java_com_example_dkdk6_blackpinkchess_Activity_PlayGameActivity_MotorWrite(JNIEnv *jenv, jobject self, jint data)
{
    int dev;
    if((dev = open("/dev/motor", O_WRONLY | O_SYNC)) < 0) {
        __android_log_print(ANDROID_LOG_ERROR, "MOTOR", "failed to open /dev/motor₩n");
        return 1;
    }
    write(dev, &data, sizeof(int));
    close(dev);
    return 0;
}
```

Figure 44 Motor Write – Android JNI



```
MotorWrite( data: 1);
try {
    Thread.sleep( millis: 3000);
} catch (InterruptedException e) {
    e.printStackTrace();
}
MotorWrite( data: 0);
```

Figure 45 Motor Write - Android

As an additional implementation, we have run a motor to inform Check Mate and Stale Mate. If any player is in check mate or stale mate situation, the motor on his board will run for a certain period.

4. Result

4.1 Total Application

- Main

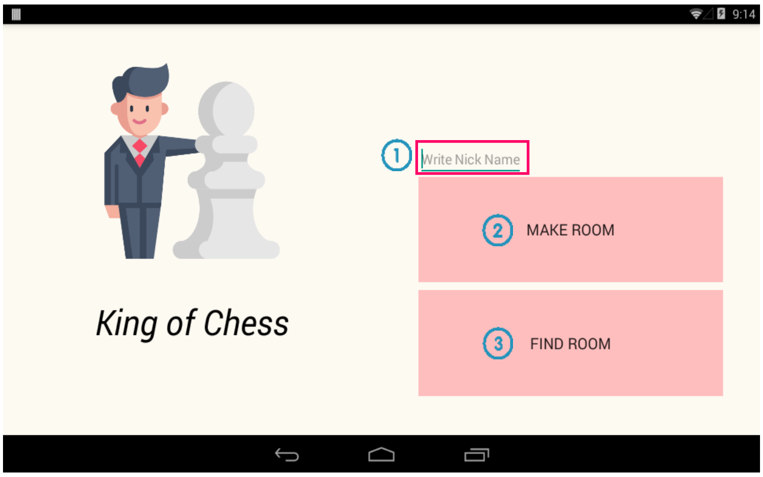
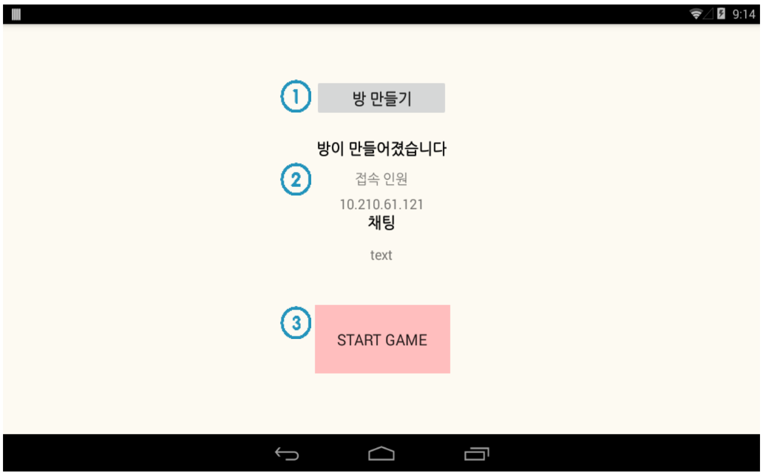


Figure 46 Main

This is the main UI part that the user's first accesses. In part (1), user can enter own's nickname. In part (2), a person acting as a server creates a room and plays the role of broad casting server's IP so the other clients can connect to server's network. And In part (3), Clients can access for the game using this button. Other clients, other than the server, connect through this button and wait for the IP address that the server is receiving.

- Make Room



방이 만들어졌습니다

접속 인원

10.210.61.121

채팅

text

Figure 47 Make Room

The picture above is the screen that appears when you click the "Make Room button". When you click the button (1), you can check the result of your IP and Player's Person Information. And The game starts with a total of four players. If all the players are connected, the "START GAME" button will be activated and the server player will be able to enter the game.

- Find Room

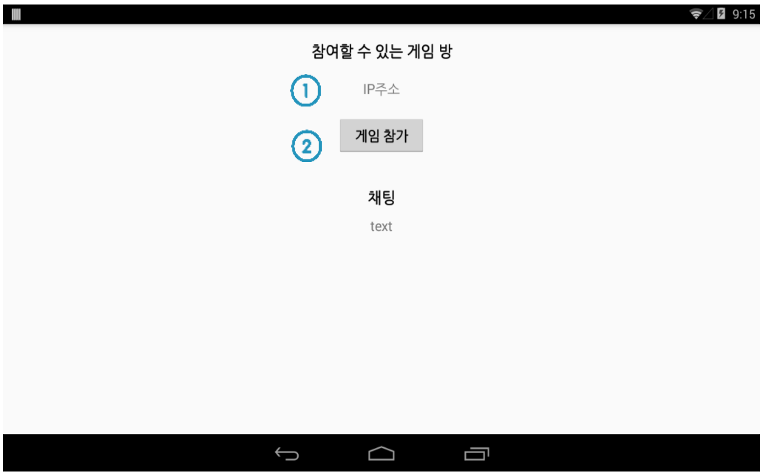


Figure 48 Find Room

When you click "Find Room" Button, you can see the IP address in (1). And When you receive an IP address, you can enter the game through the "게임 참가" button.

- Play Game

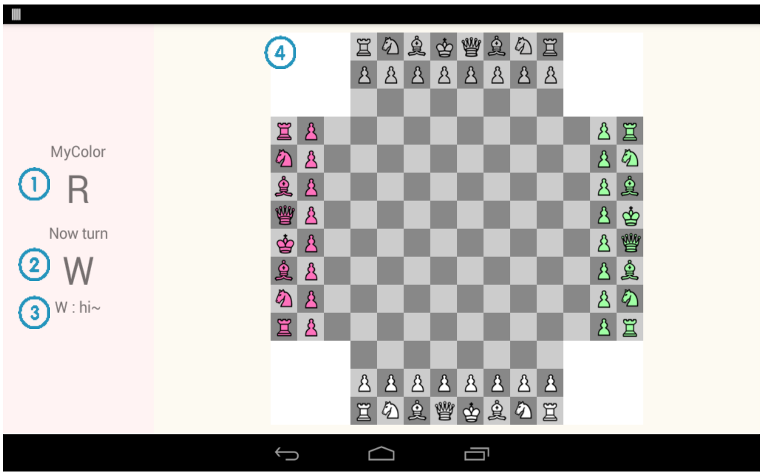
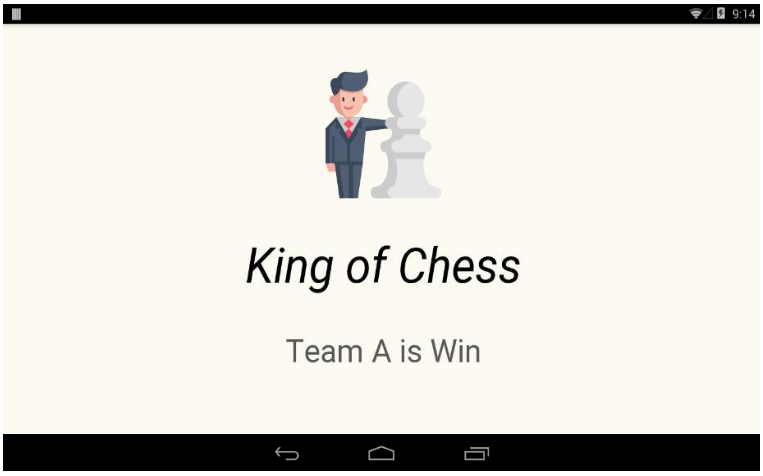


Figure 49 Play Game

The picture of above is Game main screen. The text of (1) shows the current status. And the text of (2) shows the next status. And the text of (3) shows the chatting contents. Finally, the game is played on the game board of (4), and it is possible to move only on your own turn.

- Game Over



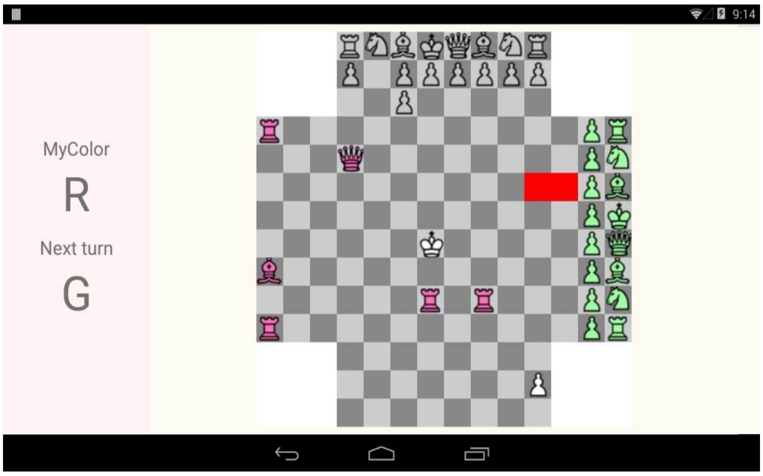
King of Chess

Team A is Win

Figure 50 Game Over

When the game is finished, the above screen will appear and the game result will be displayed.

- Pawn Moving



MyColor

R

Next turn

G

Figure 51 Pawn's Moving

The above picture is the basic movement of Pawn

- Bishop Moving

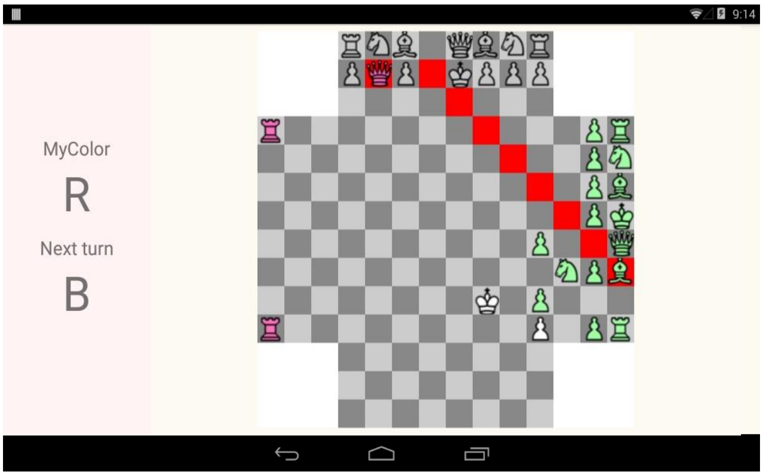


Figure 52 Bishop's Moving

The above picture is the basic movement of Bishop

- Knight Moving

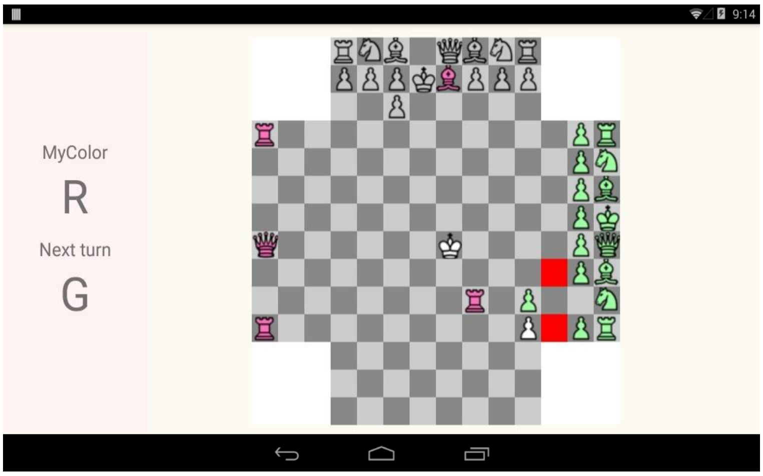


Figure 53 Knight's Moving

The above picture is the basic movement of Knight

- Rook Moving

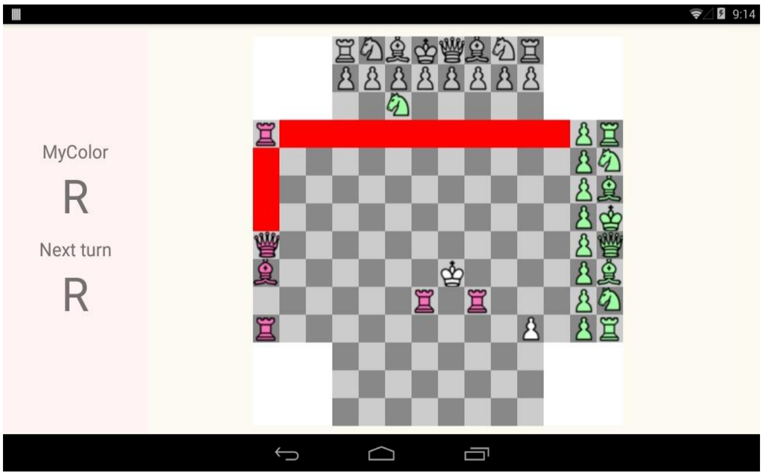


Figure 54 Rook's Moving

The above picture is the basic movement of Rook

- King Moving

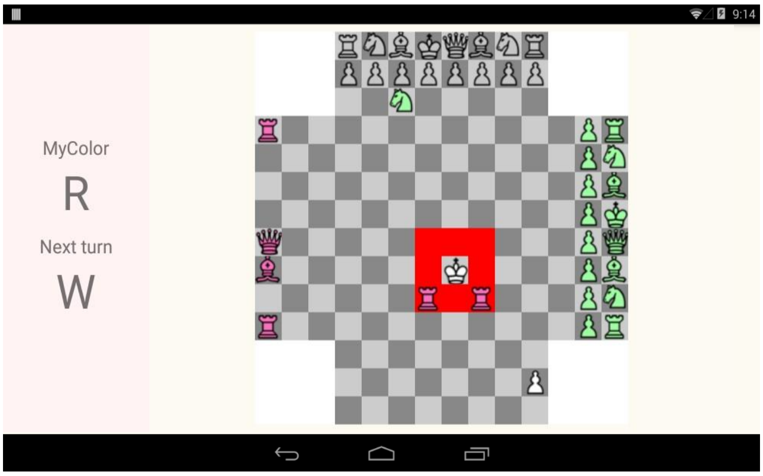


Figure 55 King's Moving

The above picture is the basic movement of King

- Queen Moving

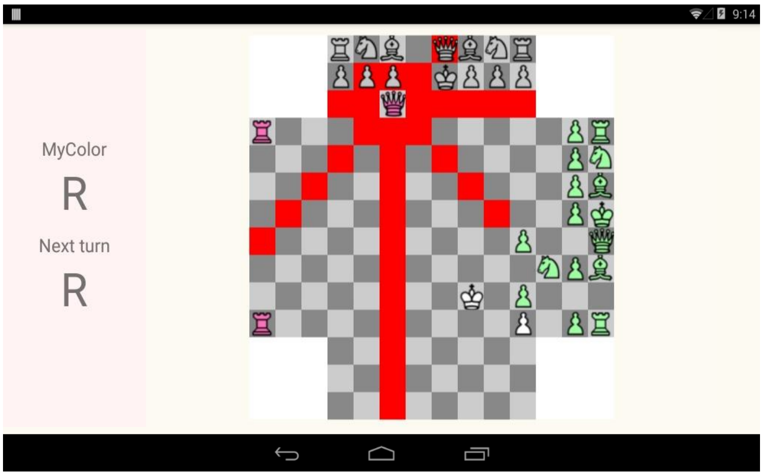
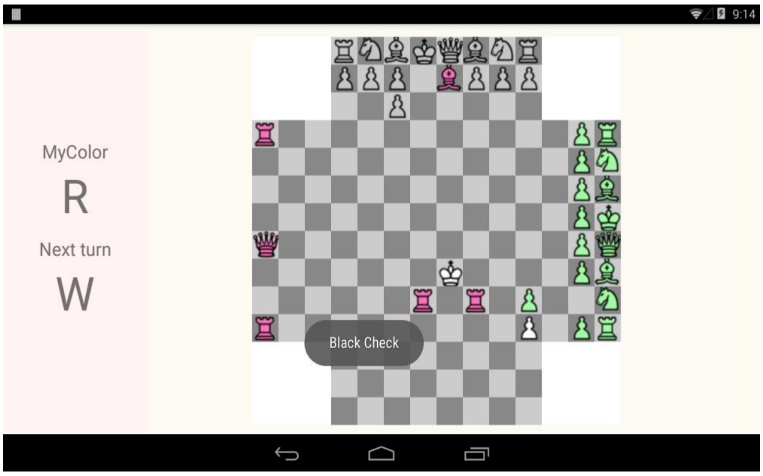


Figure 56 Queen's Moving

The above picture is the basic movement of Queen

- Check



Black Check

Figure 57 Check

Red Bishop is attacking the Black King. You can see the "Black Check" Toast message.

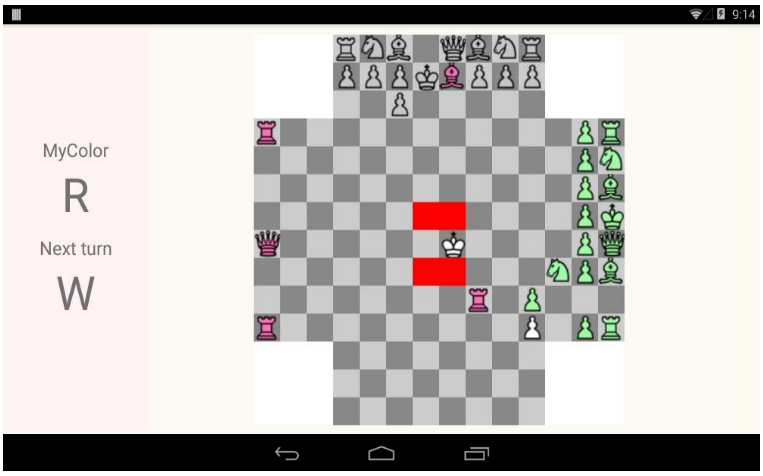


Figure 58 King in Check

White's King's movement was restricted due to the attack of Red Rook and Red Queen.

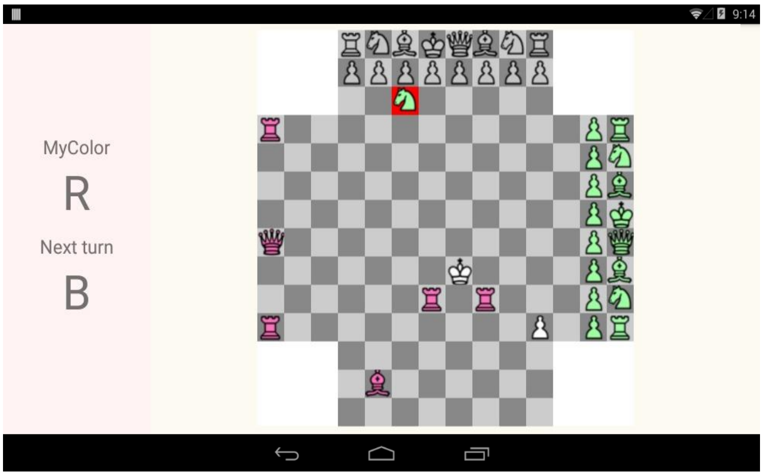
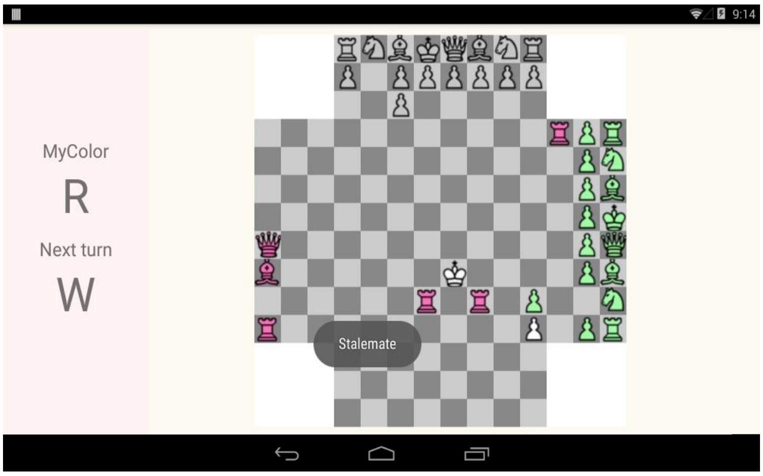


Figure 59 Black Check, Pawn's Moving

Black is in Check so Black Pawn is forced to kill Green Knight.

- Statlemate



MyColor

R

Next turn

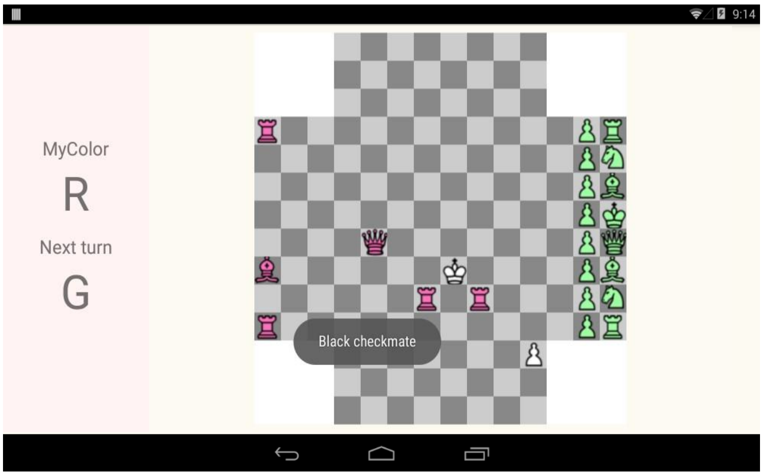
W

Stalemate

Figure 60 Stalemate

All white status can not move. Where King can go, he can attack next, so the King can not move.

- Checkmate



MyColor

R

Next turn

G

Black checkmate

Figure 61 Checkmate

In the above picture, Black is Checkmate and the all pieces are removed from the board.

4.2 Hardware

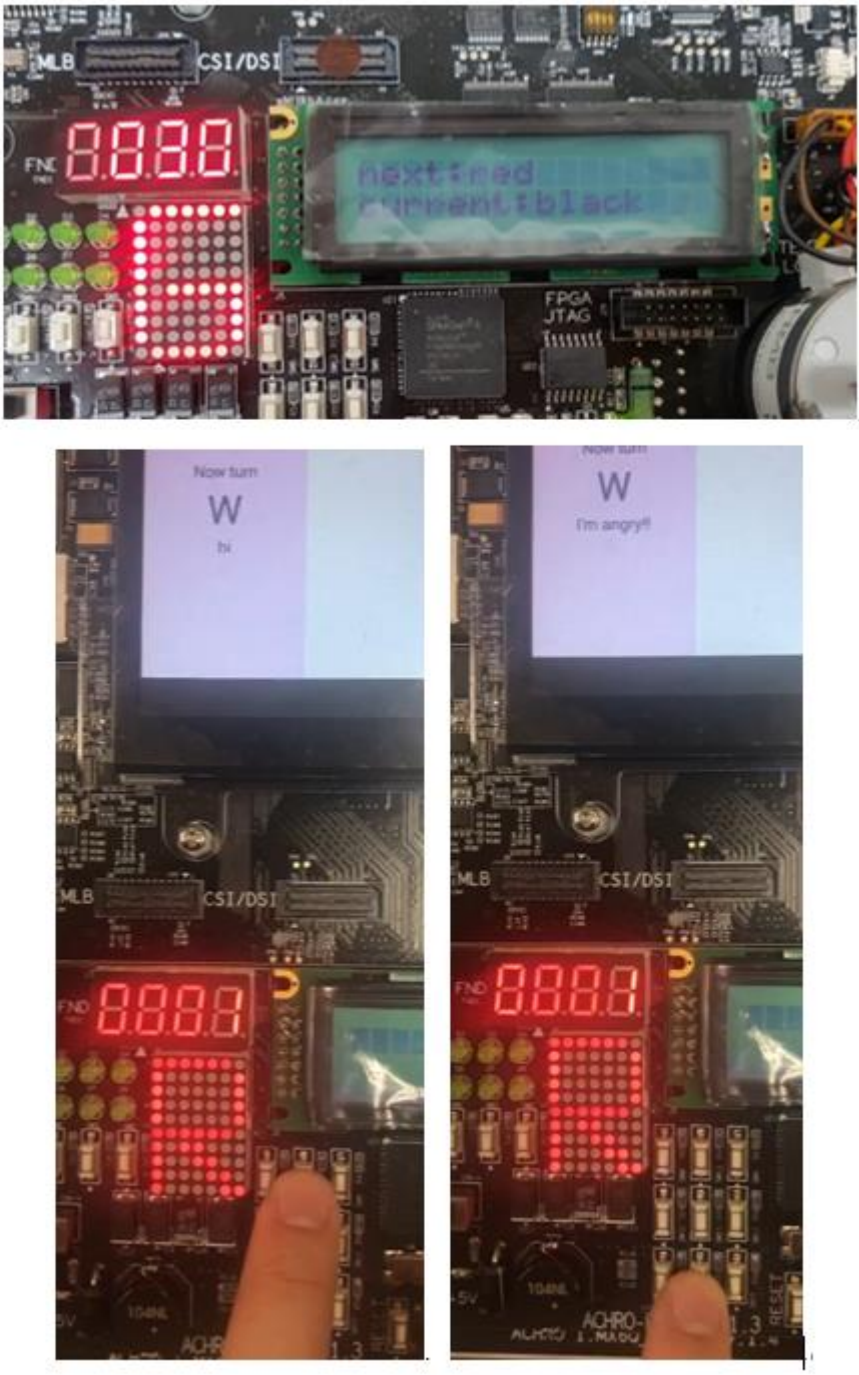


Figure 62 Hardware Result

Through the dot matrix, you can see that the player's color is green, and you can see through the 7segment that 30 seconds have elapsed before the end of the turn. The LCD shows the current and next turns. The following two pictures show the chat (hi, I'm angry !!, etc.) function on the screen by pressing the button.

4.3 Team Review

Name	Contents
Kwon Dokyoeng	I was worried that I could do well when I took a class "embedded system". It seemed that I could grow a lot by learning one by one. I think it was a good opportunity to experience many other things besides the built-in system, especially through the project. I also felt that it was really difficult to develop the module of hardware directly, and it was the first time I used Linux so much, and I was able to solve the difficult parts while developing with my team members.
Kim Gayoung	I have been able to fill a wider range of knowledge as well as embedded through projects that comprehensively integrate Android, communications, and hardware control. I was able to see the built-in system more deeply by turning the hardware control learned in the class in conjunction with Android. During the 4 week project, I showed my team members that they did not know each other and they also helped me. This has helped us build teamwork and help each other.
Kim Nawon	It was a great opportunity to practice embedded systems on Linux-based devices. Since we have been mainly doing software development so far, I think that implementing the project with the built-in program on the hardware will become the basis of new things to come. The difficult part was solved by sharing ideas with our team members and other students studying together, and we were able to grow together.
Choi Bowon	During the project, I was able to think about the embedded system technology that I learned in class. I learned that a driver module can apply various programs, and I think it would be more useful if I can make more efficient codes. And in this project, my role was network manager so I learned the socket network.