Distributed Systems EE4023

Final Project

Liam Walsh 11122048

Cathal O'Halloran 11123834

Intro

The project we decided on is the TicTacToe game. It will be a multiplayer game of TicTacToe that allows people to play from either an Android app or a java client. Players will be able to use the same account from either client and will be able to play against other players using either client also.

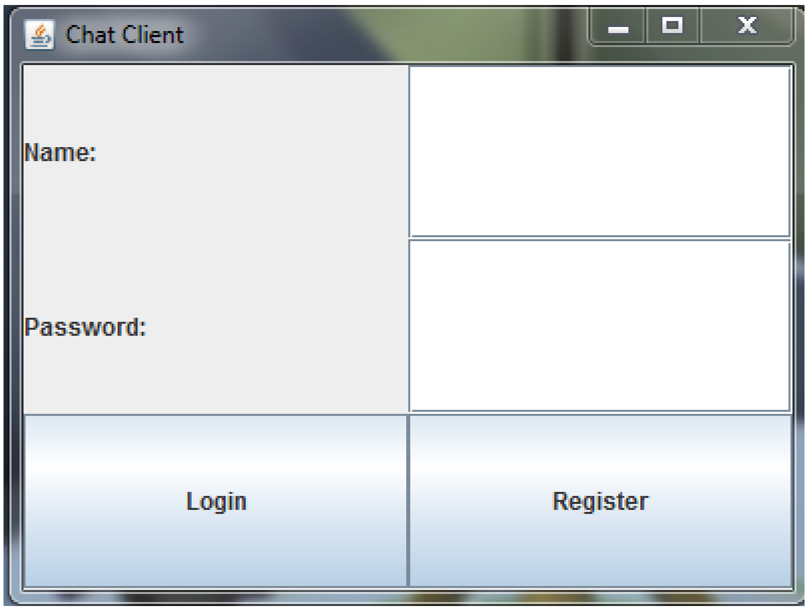
Background

Both myself and Cathal had experience with python and will be using it a lot in our co-op placement. We felt it would be much easier to write the server side scripts in python as opposed to php with our prior knowledge. Similarly, both of us have experience with JSON and felt it suited the system more as it communicates easily with Java and Android. Liam had used it in a similar fashion to this project for work over the summer. As for using Android, we both have completed a module in Android app development and Cathal has used android with JSON and MySQL before so we decided to go with the Android/Java Option of the project. We have completed modules on relational databases also so our decision to use them was easy.

Design

Java Client:

For the java client the main focus was on the GUI design. There would need to be a panel to handle each part of the client such as login, register, lobby and a game. These would have all the UI elements needed for that function. Something would be needed to manage each of these panels and the changing between them. For this a CardLayout would be useful. It would store all the panel instances with a key, in our case its name, and will display the panel when its name is invoked.

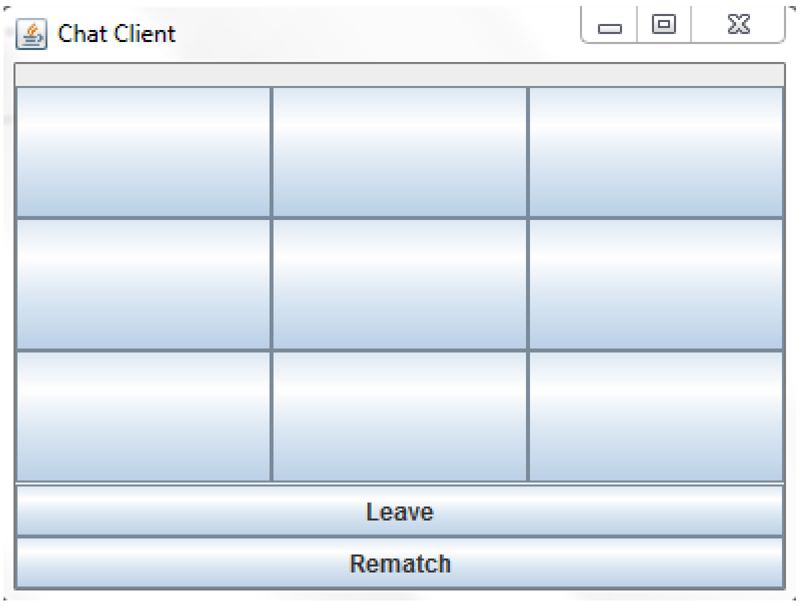


The panels used in the java client will be

- Login Panel: Users can either login or register a new account from this screen
- Lobby Panel: Main screen when logged in. From here users can create a game, join an open game, view the leaderboards or log out.
- Game Panel: Each game of tic tac toe is played from here. It contains 9 buttons for each position in the board. It also has buttons for leaving a match and requesting a rematch.
- Leaderboard: This panel will display the leaderboards of the top 10 players in the database

I would need a Game class and a User class to hold instances of the logged in user and the current game being played. These classes will store the data needed for these objects with some added functionality. The Game class would need to be able to convert a string returned from the server into a two dimensional array to represent the game board. It will also have to do the reverse and convert the 2D array into a string.

When a game is being played a timer will be running and every second will make a request on the server to see if the game has changed. If it has then the new game state will be returned to the client.



Android Client

On the Android side we needed to create multiple separate activities, as follows:

•Main Activity: Activity that launches on start up. This activity has two simple text fields to hold the username and password (which uses a password text field to hide characters) , and two buttons, one to register and one to log in.

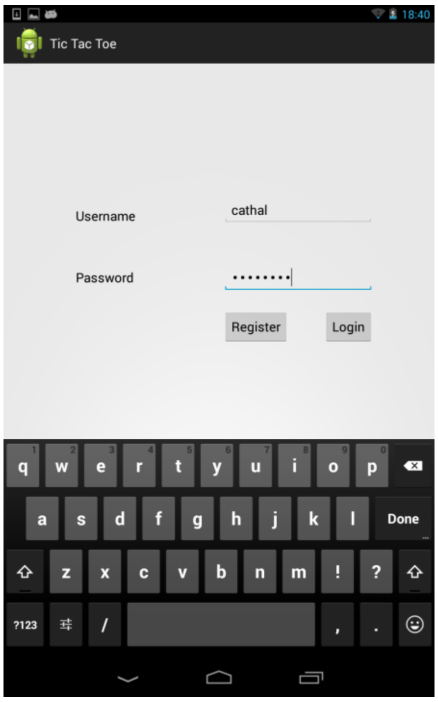
•Menu List: This activity holds a list of the three options, upon clicking one, the user will be sent to another activity. The create game option should insert a new game into the database, the join lobby will bring the user to a list of open lobbies, and the leaderboard will bring the user to the leaderboard.

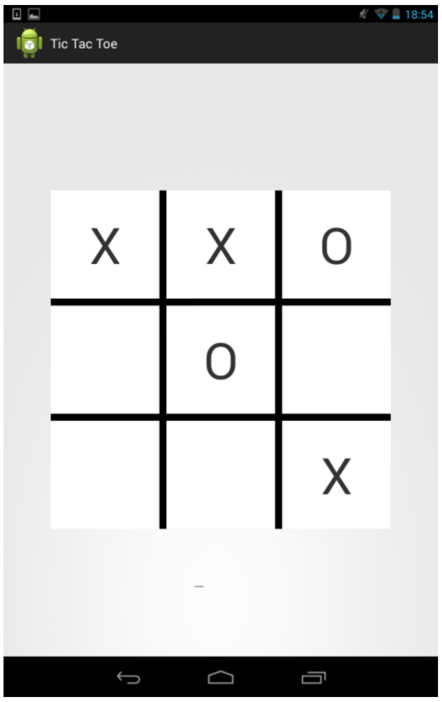
•Game Activity: This activity will be where the user plays the game. It should show a 3x3 grid which users can interact with and also, on winning users should get an option to leave or rematch. This should call all the scripts to update the database, and it should check to see if there has been a change in the game, if so the grid should be updated.

•Lobby List Activity: This activity will show a list of all the open lobbies. Upon clicking any item in the list the user should be added to the game in the database and then be brought to the game activity.

•Leaderboard Activity: This activity should simply show the leaderboard of the users with the best Games Played : Games Won ratio in a grid format.

•Json Helper: A Helper class used to read json from the scripts

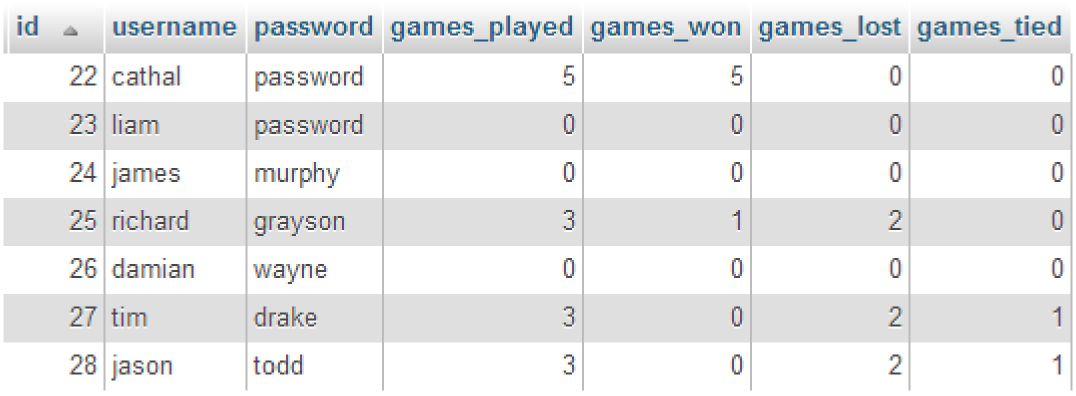




Server

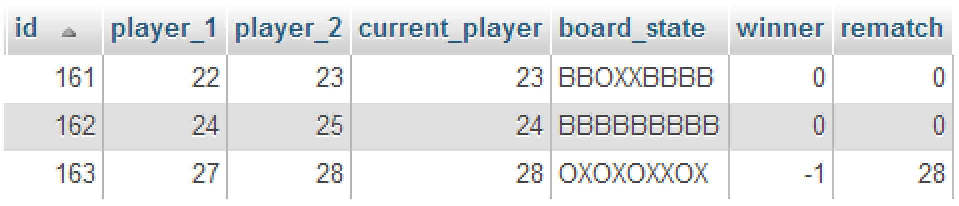
We had to design the schema for the database. We are using a relational database and decided to use mySQL to interact with it. There would be two tables, one Users table and one Games table. The Users table will hold the user details and their match record (wins, lose and draw). The Games table will hold all games currently active. It will hold the ID of the two players, the current player and the current board state.

Users Table



id	username	password	games_played	games_won	games_lost	games_tied
22	cathal	password	5	5	0	0
23	liam	password	0	0	0	0
24	james	murphy	0	0	0	0
25	richard	grayson	3	1	2	0
26	damian	wayne	0	0	0	0
27	tim	drake	3	0	2	1
28	jason	todd	3	0	2	1

Games Table



id	player_1	player_2	current_player	board_state	winner	rematch
161	22	23	23	BBOXXBBBB	0	0
162	24	25	24	BBBBBBBBB	0	0
163	27	28	28	OXOXOXXOX	-1	28

Design Issues

One issue we had when designing the game was how to fit rematch requests and players leaving into the system. We worked through a number of ideas until finally a good and working solution was found.

The two columns winner and rematch were added to the games table. These are 0 when a game is being played. A winner is detected in updateGame and the winner's ID is placed in the winner column. When checkGameState is called it will notify the users that the game has been won or ended in a draw. Each user can now click rematch on their client. If both click the game resets. If one player leaves then the other will be in an open lobby. If a user leaves during a game then they are awarded a loss and the other player receives a win. The game will be reset and the remaining player becomes player 1.

Implementation

Java Client :

To implement the panel switching for the Cardlayout I created a method called changeCard which is invoked whenever a panel transition is needed. The method takes in the name of the panel and uses this to display the correct one.

```
public void changeCard(String cardName)
{
        if(cardName.equals("LOBBY"))
        {
                lobbyPanel.updateGameList();
        }
        CardLayout cl = (CardLayout)(this.getLayout());
        cl.show(this, cardName);
}

public void actionPerformed(ActionEvent e) {
        CardPanel.this.changeCard("LEADERBOARD");
}
```

To convert the game string into an array I wrote two methods updateBoard and setBoardFromString. setBoardFromString will take in the 9 character long string and iterate through each character. Each one is sent to updateBoard to update the array.

```
public void setBoardFromString(String boardString)
{
        int c = 0;
        for(int i = 0; i < 3; i++)
        {
                for(int j = 0; j < 3; j++)
                {
                        updateBoard(i, j, boardString.charAt(c));
                        c++;
                }
        }
}

public void updateBoard(int x, int y, char player)
{
        board[x][y] = player;
}
```

The timer was implemented using Timer and Timertask. An inner class is created that extends timer task. This is what will be executed when the timer ticks. The timer is set to run continuously and call every 1000ms. The inner class will request the game state from the server by passing up the game id and the clients current board state.

```
private class CheckGameTask extends TimerTask
{
        @Override
        public void run() {
                try {
                        //get the server response for checkGameState
                        JSONObject json =
URLRequest.readJsonFromUrl(URLRequest.checkGameState(game.toString(), game.getID()));

        }
}
```

```
/**start the timer*/
public void startCheck()
{
       timer = new Timer();
       timer.schedule(new CheckGameTask(),0, 1000);
}

/**top the timer*/
public void endCheck()
{
       timer.cancel();
}
```

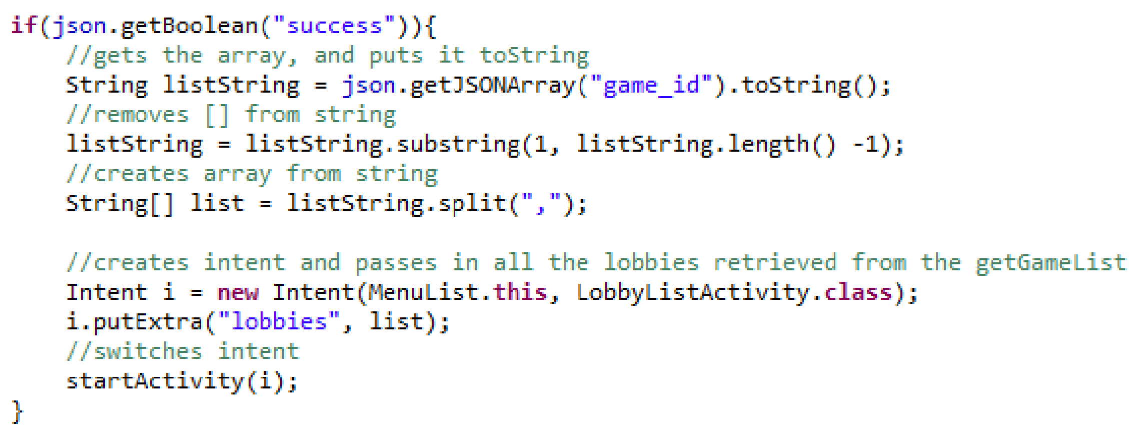
Server responses were retrieved using Input Stream from the url. The url gets created using the parameters from the client. It then opens the stream using that url. The response is read back and converted to a json object. This can be manipulated using json.get("success").

```
/**create the url for createGame*/
public static String createGame(User user)
{
  String urlString = "http://" + ip + "/webservice/createGame.py?id=fail";
  if(user.getID() != 0){
  urlString = "http://" + ip + "/webservice/createGame.py?id="+ user.getID();
 }
 return urlString;
}

/**Return the string from the reader */
private static String readAll(Reader rd) throws IOException {
    StringBuilder sb = new StringBuilder();
    int cp;
    //add each character to the stringbuilder
    while ((cp = rd.read()) != -1) {
    sb.append((char) cp);
    }
    return sb.toString();
}

/**Get and return the json response from the url */
 public static JSONObject readJsonFromUrl(String url) throws IOException, JSONException {
    InputStream is = new URL(url).openStream();
    try {
       //read in the stream from the url
    BufferedReader rd = new BufferedReader(new InputStreamReader(is, Charset.forName("UTF-
8")));
      //get the string from the stream
      String jsonText = readAll(rd);
      //convert the string into a json object
      JSONObject json = new JSONObject(jsonText);
      return json;
    } finally {
      is.close();
    }
}
```

```
if(json.getInt("change") == 1)
{
        //update the game and the gui
        game.setBoardFromString(json.getString("board_state"));
        game.setPlayer(json.getInt("current_player"));
        updateBoard();
        setLabel();
        firstTime = true;
        gameOver = false;
}
```

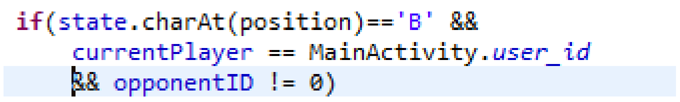
Android Client:

One of the main issues while implementing the Android client was trying to run the server scripts. Android doesn't allow for any opening of URLs on it's main thread, so I wasn't able to run them directly in my activity. To get around this, I researched and found that Android's AsyncTask class seemed perfect for the task at hand. “AsyncTask is designed to be a helper class around **Thread** and **Handler** and does not constitute a generic threading framework. AsyncTasks should ideally be used for short operations (a few seconds at the most.)”[1]. This was perfect for running python scripts that are all running mySQL statements which all took under a second to execute. After implementing how to run the SQL scripts, the next task was to read the json that is returned. I researched, and implemented a JsonHelper class which reads the Json from a URL passed in. This worked perfectly, until I had to manipulate an array from Json. In order to tackle this, I read the Json array, use to string then use the String class' .split() method to get the String Array I wanted to read back. The following code snippet shows this implemented



```
if(json.getBoolean("success")){
    //gets the array, and puts it toString
    String listString = json.getJSONArray("game_id").toString();
    //removes [] from string
    listString = listString.substring(1, listString.length() -1);
    //creates array from string
    String[] list = listString.split(",");

    //creates intent and passes in all the lobbies retrieved from the getGameList
    Intent i = new Intent(MenuList.this, LobbyListActivity.class);
    i.putExtra("lobbies", list);
    //switches intent
    startActivity(i);
}
```

The next big challenge faced in the android application was the game. I had already used GridView and Array Adapters in a number of previous Android applications, so that was fine. What I needed to be able to do is keep track of the board state and the state shown on the grid. I decided to hold a string and an array(boardstate), the string (state) would hold the state (e.g. BBBXOXBBB) and the array would hold the state to be shown in the grid (e.g. {“ ”, “ ”, “ ”, “X”, “O”, “X” “ ”, “ ”, “ ”}). To achieve this, after reading in the update, and making sure that the state of the board has been changed, I read through the 9 characters in the string (state) and if there is a B I put a space into the array, otherwise I just put in the X or O. Then I update the grid. The next problem was to make sure the user can only make “legal” moves. To do this, I keep tack of the current player, which is returned from the getGameState.py script, make sure there is an opponent in the game and finally that the square pressed is empty. The following if statement checks these three conditions :



```
if(state.charAt(position)=='B' &&
    currentPlayer == MainActivity.user_id
    && opponentID != 0)
```

Server

Implementing the server scripts was pretty straight forward. Python is a very easy language to work with. The scripts contained mySQL calls that would interact with the database. The parameters passed in using the url could be got using cgi.FieldStorage.

```
fs = cgi.FieldStorage()

cur.execute("""INSERT INTO games(player_1, current_player)

        VALUES (%s, %s)""", (fs["id"].value, fs["id"].value))
```

Testing

Testing was done iteratively. We worked on each component of the system at a time. We made sure that both the android and the java client could complete the task fully and correctly before moving on to the next stage. In order to test on Android, we used Cathal's Nexus 7 tablet as the test platform.

A server was set up using XAMPP. Our server side scripts were placed here and the database was created. The server was hosted on one laptop and we both connected to it while testing. For each script we could call it using a browser to test that it works and gives the correct response. Once this was determined we would implement the client side and test it.

We never ran into major issues with the project. Good decisions in the design stage helped a lot with our implementation. Some minor bugs had to be fixed with each stage. We had some issues when we set the refresh timer to be very short. The clients were not receiving updates properly and it caused a few issues. We decided on a refresh rate of 1 second.

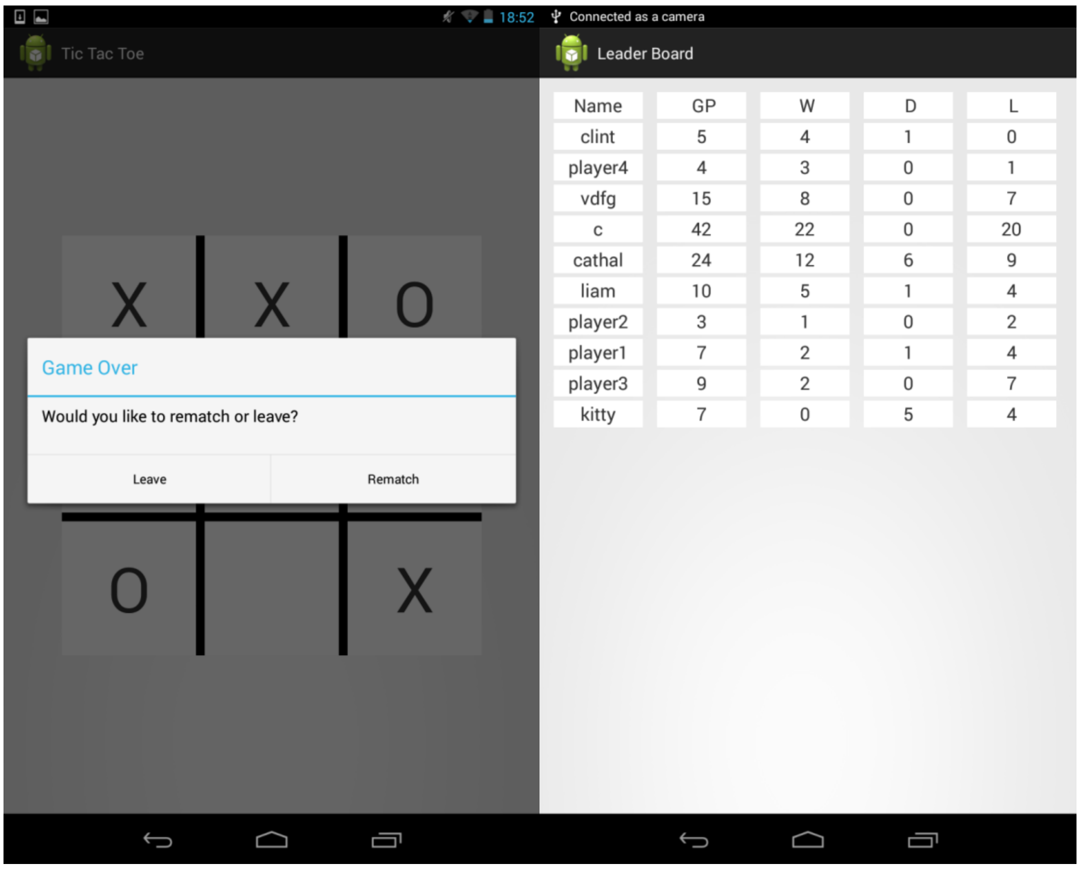
Result

When the project was finished we had a fully working multiplayer game of tic tac toe. There were two different clients that could play the game. Users can register and login to the game. They can create games or join an open game. Once a game is started they play against each other taking turns. User records are stored in the database and can be viewed in the leaderboard.

Conclusion

This project was an interesting one to work on. It was the first college project that included the use of multiple languages. Our choice to use python and java was good because we had experience in both. Choosing PHP would have made this project more difficult as it would have meant learning a new language. XAMPP was very useful for testing purposes. This project was good to prepare us for our work experience starting in January. We were very pleased with our project when we finished.

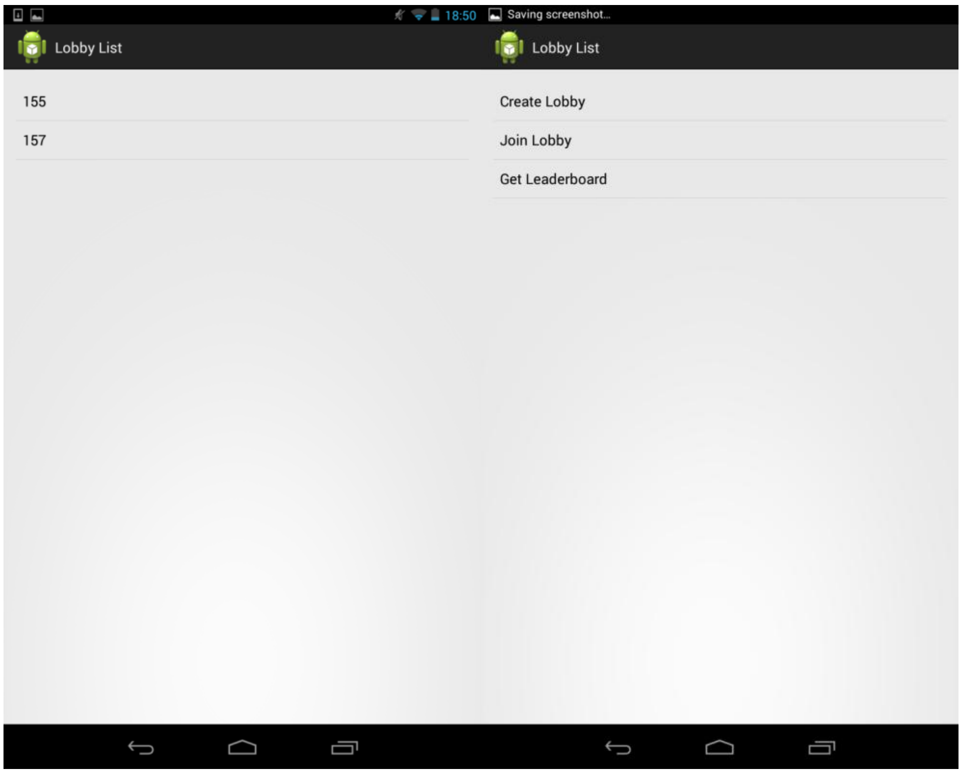
Screenshots

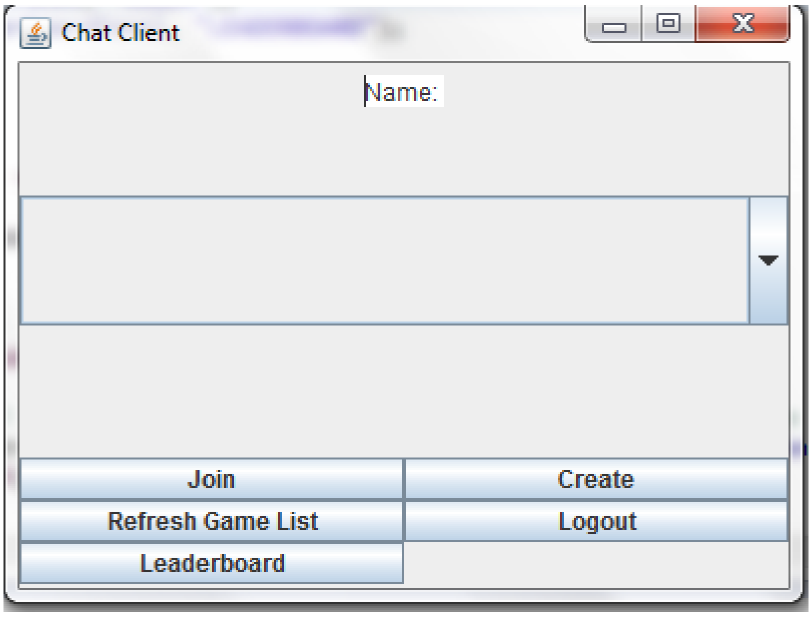


Game Over

Would you like to rematch or leave?

Name	GP	W	D	L
clint	5	4	1	0
player4	4	3	0	1
vdfg	15	8	0	7
c	42	22	0	20
cathal	24	12	6	9
liam	10	5	1	4
player2	3	1	0	2
player1	7	2	1	4
player3	9	2	0	7
kitty	7	0	5	4





References

1. Android Developers *AsyncTask* Available: http://developer.android.com/reference/android/os/AsyncTask.html Last Accessed : 29/11/2013

2. Java Tutorial, Using the timer and timertask class, http://enos.itcollege.ee/~jpoial/docs/tutorial/essential/threads/timer.html Last Accessed: 28/11/2013