**Introduction**

Our project was to implement the Assassin game. The game works as follows. There are n players in the game, and each player is assigned another player to “kill”. Nobody has more than one target – each person has a player trying to kill them and a player that they are trying to kill. Once a kill is made, the person killed is out of the game. The person who killed is now assigned the killed person’s target. The game continues until there is only one person remaining. To implement this game, we chose to use Android as our new technology and set up our server instance on Amazon EC2. The basic flow of our application is such that the client (Android) decides to start a new game or continue an existing game. This response is then sent to the server. If the client decides to start a new game, the server takes in the group ID and the list of players in the game and assigns targets. If the client decides to continue an existing game, it is taken to a screen where they enter in the group ID and the player’s name and select whether they made a kill or have been killed. Again, this response is sent to the server and the server sends back the appropriate response. If the client made a kill, it assigns a new target. If the client got killed, the server kicks this client out of the game.

**Server**

Our server is a simple Java server. It is implanted using an Amazon EC2 instance. The first information the server receives is whether the client is starting a new game or continuing an existing one. In Server.java, we create our BufferedReader object, ready to read from the client. The client reads data in the following format – “command; game ID; playerID/(playerID1”. If the server receives command=0, this means the client wants to start a new game. The order of the players is randomized in an array, and we put this array into a HashMap. For the last entry in the Map, we set the target to be the first entry. The game has now been created and we send this response back to the client. However if we want to continue and existing game (command is not 0), we ask if the client made a kill or got killed. Before that we do a few consistency checks (checking if the player exists, checking if the game exists, etc). If the client made a kill (command=1), and the size of the Map is 1, then this client won the game. Otherwise, it reassigns the client to a new target. If the client got killed (command=2), we simply remove them from the Map and find the killer a new target. This is all saved in our HashMap. We then send the response and flush the PrintWriter. Finally, we close the socket.

**Client**

Our client side was implemented on an Android application. We have 3 activities – MainActivity, StartGameActivity, and ContinueGameActivity. MainActivity is the home screen, where we provide the option of either starting a game or continuing a game. We create two buttons and have onClickListeners for each of the buttons that when activated, creates Intents for the appropriate activity and starts the activity for result. Let’s take a further look into each of these two activities.

StartGameActivity

This activity keeps track of a few variables – mGameID, mPlayerNames, myCommand, and myResponse. This activity also has an AsyncTask which reads and writes to the server. In the AsyncTask we implement the doInBackground method. This method creates a Java socket, a PrintWriter and a BufferedReader object. We write the command (which we set in the onPreExecute method) to the server. We await the response from the server, and the response will be that our game was successfully created and assigns us our target.. We then close the socket and go back to the main screen.

ContinueGameActivity

This activity keeps track of mGameID and mPlayerName, but also some request codes – TARGET\_REQUEST and DEATH\_REQUEST. The AsyncTask works in exactly the same way as it did in the StartGameActivity, however the results it will be getting back from the server will obviously be different.

Upon completion of each of these activities, since we did create an activity for a result, we will go back to our main activity’s onActivityResult method, which simply sets the text to be the response from the server. The application flow is now complete.