Lukas Anderson, Dhruv Yadav

Janos Sallai

Computer Networks

Final Project Design Documentation

**Introduction**

Our project was to design and implement a mobile application for the popular group game, “Assassin”. The rules of the game are as follows: As many or as few people can play as desired. Each player in the game is assigned a target at the beginning whom they must “kill” through some arbitrary method. When a player is killed, they are out of the game, and their target becomes their killer’s new target. The game proceeds in this way until only one player remains, who is the winner. In app form, the game could be played without first needing to assign and distribute each player’s target, as this could be done automatically.

To create the game in app form, we wrote a client protocol in Android (our chosen programming language not discussed in class), and a host protocol on an Amazon EC2 server. The basic flow of our application is that the client, in Android, is given the option to either start a new game or to continue an existing game (continue, as in, getting the name of their target or reporting their death). On making their selection, they are prompted to enter the corresponding necessary data. This is then compiled into a command String that is sent to the server. The server, in turn, processes these commands, adding or updating its data for the games as needed, then sends a response String back to the client.

**Server**

Our server side is a simple Java class, run on an Amazon EC2 host. It was written to hold data for the games in HashMap, to read commands from the client using BufferedReaders, and to write responses to the client using PrintWriters. Commands for the server are created using the following format: “action; game ID; player name(s)”. The first component of the command, the desired action, is either “0”, “1”, or “2”, which correspond to starting a new game, getting one’s target, or reporting one’s death, respectively. The second component of the command is the game ID, which is used as a key for either creating or accessing the game’s HashMap. The third component of the command is dependent on the action.

If the action is 0, then the third component will contain a list of names, separated by commas. The server randomizes the list and puts it into a HashMap, with the players as the keys and their targets as the values, and sends a confirmation response to the client. If the action is 1 or 2, then the third component is the accessing player’s name; if the action is 1, then server retrieves the player’s target from the HashMap, and if the action is 2, then the server removes the player from the HashMap. In the former case, the server’s response is the player’s target, and in the latter case, the response is a confirmation of the player’s removal from the game. Once the server has sent its response to the client, it closes the socket and returns to a waiting stage.

**Client**

Our client side is an Android application. First, the MainActivity prompts the client to choose between starting a game or continuing a game using Buttons. If they click the ‘Start Game’ button, then a StartGameActivity is launched; if they click the ‘Continue Game’ button, then a ContinueGameActivity is launched.

In StartGameActivity, the user is prompted to enter a game ID and a list of the players’ names, separated by commas. When the user clicks the ‘Submit’ button, the user’s entries are parsed into a command String, with the action component being automatically set to 0, and a StartGameTask is launched (an extension of AsyncTask). The StartGameTask then establishes a socket connection with the host, writes the command String with a PrintWriter, then awaits and reads the server’s response with a BufferedReader. Once the response String is received, it is returned as the result to the MainActivity, where it is set in a TextView for the user to read.

ContinueGameActivity has essentially the same functionality as StartGameActivity (including its extension of AsyncTask), with two key distinctions. First, the player name entry field in ContinueGameActivity is a single line EditText, while the players’ name list entry field in StartGameActivity is a multiple line EditText. Second, because there are two possible options when continuing a game (the options corresponding to the actions 1 and 2, as described above), a RadioGroup is used in the UI to choose between them. Thus, when the ‘Submit’ button is clicked and the command String is parsed, the action component is set according to which RadioButton in the RadioGroup is selected, unlike in StartGameActivity where the action component is always 0.