Word Guessing Game Project Report

Anya Ellis, Darius Grassi, Paola Perez-Rivera, Tim Kazarinoff

General description of project logic/algorithms/design choices

Our project followed very similar concepts as Project 3, save for a few additions. The logic of our server was to receive information from clients, and send back responses through our serialized information class GuessInfo. Most of our logic for the server was contained inside of the Controller2 and Server classes. This is because the server receives the newest version of GuessInfo from the clients, and thus when a field is updated, we can implement conditionals for how the server should behave when a certain condition happens, and send it back to the clients. In terms of borrowed concepts from Project 3, we followed similar guidelines of implementing unique threads for each new client connected to the server. Thus we were able to isolate each client using keyword 'synchronized' to apply a lock for a thread when it was making its turn. Server-side didn't require many algorithms, since most of it was just updating information from clients. Useful algorithms that were present were utilized to calculate whether client guesses were correct for words, or if the character was present in the word. One design choice that was made code-wise was to implement a controller for each scene. Partially this was rooted in simplicity and bug fixes, but this also allowed for the implementation of our `runLater` in Controller2, making it much easier for us to continually update our JavaFX graphical tools.

The client logic was similar to the server logic, but differed in two main instances. The expected game information was to be primarily changed and updated, in respect to the server information. Two, the client used one Controller class for all FXML documents. We did this because many of the client information as well as reactions were shared between all screen designs. There were a couple interesting algorithms within the client development such as how to differentiate if a category had been picked beyond it max or even just the simple use of differentiating the letter button being clicked.

UI+UX development

For the server's user interface, the goal was to keep it as simple as possible. The plan was to have the focus be a ListView that would be used to output information coming from different clients. In favor of having a more unique design while staying less cluttered, a scoreboard for each client was added as well. Doing this allowed a user to keep track of all clients while seeing updates from each. For the UX of the server, there wasn't much to implement since there would be no point having many user controls. In a practical sense, the server instance to us would only be used by one person running the server and monitoring different clients. The keyword in this is 'monitoring', and so the goal of the design for the server was focused on that. Thus, the console and scoreboard served the purpose of effective monitoring tools, while staying in favour of the design principle of minimalism and reducing clutter.

For the client's user interface, the goal of minimalism was applied. Staying consistent with a common theme, the decision of using three scenes was made. Three scenes keeps things simple because it allows one scene for login, one for the user to choose categories, and one for the word to be guessed. This allows the developers to break the project up into sections, rather than stuffing everything into one or two scenes and having a lot more code in a single file. Controls were created that users would expect, such as being able to exit the game, select categories and choosing characters.

Teamwork

Work was divided very simply. Paola and Anya were team Client, and Tim and Darius were team Server. The team broke up into two like this, but still reconvened to make sure development wasn't going into two different directions by using Github and sharing screenshots of current interface design to stay on track.

As stated above, collaboration was handled through Github. Group chats were also created on various platforms in order to stimulate and assist group interactions. Communication was effective in that there were many options for contributing team members to communicate. However, it is difficult to have people working on different projects, and so occasionally there were instances where the team would have to work together to figure out a connection issue or an issue one team couldn't figure out, since sometimes lapses occur in how projects are sending or receiving data.

Contributions - Anya

I worked on the client logic side. I began by creating function prototypes for the Client class; then I used the buttons and design by Paola to create a more interactive client-server communication. This entailed having the buttons affect the game information and sending that to the server, as well as retrieving server information.

Contributions - Darius

I was on team Server. I focused on building the base logic and the UI/UX for the project. I first started out by building and designing the project interface in SceneBuilder based on our group's sketches for the designs. From this I decided to actually facilitate a way for the server to receive and send data by borrowing most of the logic I had created in Project 3. I implemented the runLater for the server and designed both the Controllers for the scenes. I also created the Server class, where the actual server would be created on the user-specified port. I created a Github repository and added the group as collaborators so we would have a common basis for the project. By the time I had finished my part of the project, we had a server with two scenes, which updated UI elements when a client connected to the server, including the clients connected counter, scoreboard and ListView console.

Contributions - Paola

I worked on the GUI for the client. I first started by creating the wireframe and planning out a simple design which left enough room for us to go back in and add any other details. The idea to ask the user for a username was meant to make the game a little more personalized. From there, I used the design to code the FXML and CSS files to create the three scenes used in GUI. I created a Controller class to hold the event handlers of all the widgets and to be used later for when the client-server connection logic was written. I also created the Serializable class which contained data members to keep track of the information that needed to be passed from the client to the server and vice versa.

Contributions - Tim

I worked on server side game algorithms and logic, and helped create the examples the game would use for categories and testing. I also updated various game mechanics to improve the flow of information from server to client and back and refined the programs methods regarding information transfer and game state update. I also helped plan out and define what metrics we would need to maintain the game state over the course of multiple clients connections and consecutive turns within the game.