CS342 Project 4: Team 19

Luis Castaneda : Lcasta6

Het Banker : Hbanke2

Ria Gupta : Rgupta40

Parth Patel : Parthgp2

Client-Server Logic

There is nothing really new in regards to the Client and the Server actually connecting other than one feature we have where the Client will actually shut down if it cannot connect to a Server since the Client will be all but useless if it is not connected to the Server. The more interesting thing is about how the Clients and the Server actually communicate. Let's begin with how the Client side behaves when sending a guess or category to the Server. Firstly, the user will choose one category from the three categories available. The client will send this category to the Server thread assigned to it which will then choose a word for the client to guess then send back the number of letters to the Client and the Client will output information on the state of the game - guesses left, letters found, how many letters are left, etc. Then, the user will input a guess to send to the server. We have limited the input field to accept only one character since a user should only be guessing one letter at any given moment and it also helps our algorithm for finding missing letters easier to take care of since we wouldn't have to deal with getting one letter out of the user input. The way we actually find characters that the user has successfully guessed is by looping through the word that the user has to guess and checking if any characters match the user's guess, if so, then those letters are outputted on the Client display.

Moving on to the Server side, there are a couple ArrayLists that are used to keep all the Clients in order as well as the information of the various states of each individual game going on in each Client. When a Client request is received, the Server will firstly update that particular Clients game state in the ArrayList of game states. Then, the Server will check if the Client is trying to start a new round, if so, then it will randomly choose a word of the chosen category from a word bank that is kept in the Server. If there is a word currently being guessed, then the Server will just send back the information on the word according to the guess the user inputted.

Development of UIUX

We kept the User Interface quite simple to navigate, and designed it while keeping cognitive load in mind. To make it more interactive with the user we added music for different phases. As in when the user is connecting to the server it has different music, when the user is at the game page the music changes. For every input the user enters we try to give some response like this is a correct guess, when they can't guess the word there is a message along with music so that the user stays interacted. The display is quite clear, it shows the number of won and lost rounds, the number of letters they have already guessed for the current word and how many more chances they have left. The categories are pretty simple and can be selected by buttons, we also added in some pictures to make it a little more fun along with the music.

User experience design was pretty straight forward in total, the player has only 5 different things to interact with in the main game. Those include the 3 buttons used to choose between the categories, and the TextField/Button combo used to send letters to the server. We wanted to keep the UI as intuitive as possible, for that reason we disabled and enabled things as the game progressed; for example, when the player chose a category, the other category buttons and images where disabled and the TextField/Button field where enabled, this would indicate to your player that now he has no other resolve but to start trying to guess the word. If the user guessed the word correctly then the category that he chose correctly is now faded out for the remained of the game, if not, then the category is re-enabled.

As was stated above, we wanted to make the game as intuitive as possible, for that reason we added a variety of sounds and effects to help the player keep track of the game's progress. That includes when a new word was chosen, when the player had an incorrect guess, a correct guess, and when the player has won or lost a game. We also kept music into the game to make it a little more fun.

Work Division

The work wasn't strictly or explicitly divided among the group members. It was a very organic division of work where someone would tell the group what they were planning on working on so others didn't work on the same things at the same time. We had a group chat that was used as the main source of communication between the group members, and we used GitHub as our version control to make sure we all had up to date files. With this loose division of labor, while it was very flexible to creativity, it was also a source of problem in areas that required a strict set of guidelines. Namely, the communication between the Client and Server requires a very strict protocol as the Serializable class used for communication must be exactly the same. We got around this problem by making a Google Doc and writing down how the communication will take place between the Client and Server. By having those details in one place, individual group members could reference back to that document or drop a question in the group chat if there was some confusion. Other than that one hurdle, the rest of the development went along fine as there were no other sources of conflict between each individual's work.

Parth : Client Setup, Player Info

I worked on setting up the Client side of communication as well as worked with Player Info, our Serializable class used for communication between the Client and the Server. I set up a rudimentary GUI that was very bare bones and only had the functionality required of the project. This GUI would be styled later on in development by Luis.

Luis: Server, Player Info, GUI styling, Game Logic

I focused on developing and designing the GUI of the program and setting up the Server side of the communication, I designed the PlayerInfo class, but needed some help with finalizing it's design which was provided for by Parth. For the GUI logic I used what was provided for by Het and Ria and simply converted it over to work with our GUI.

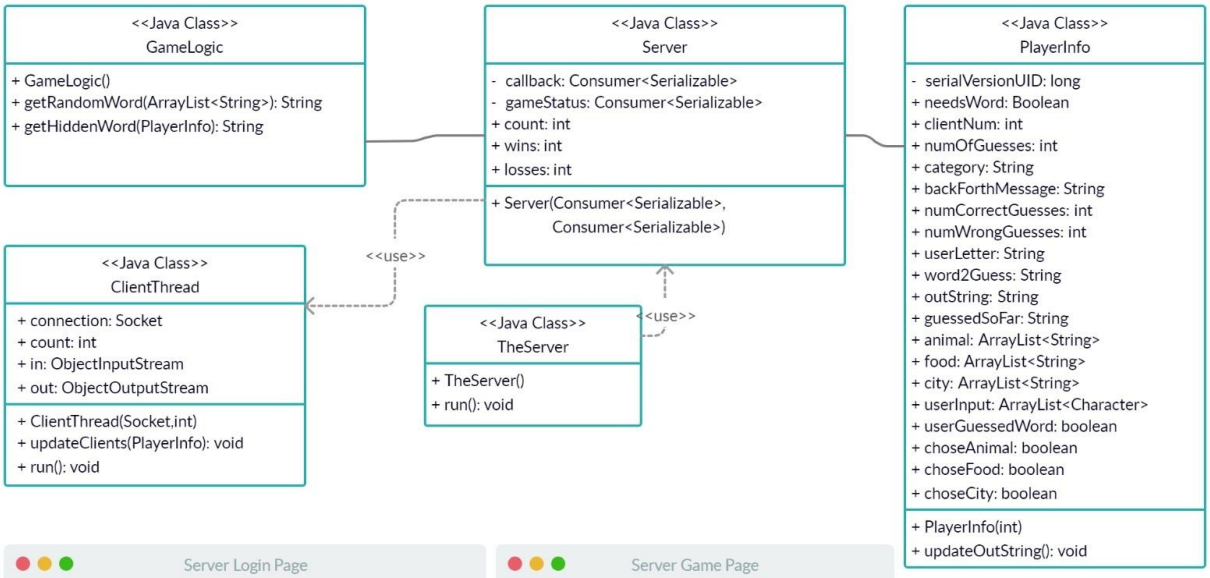
Het: Server, Game Logic, UML diagram

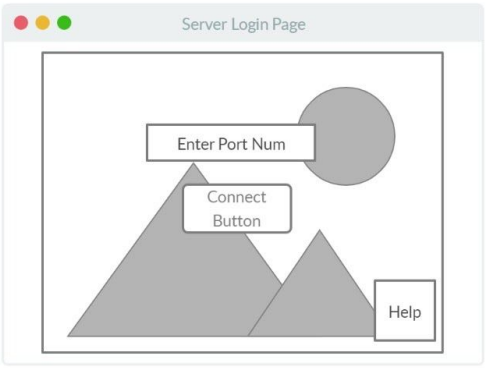
I fixed the bugs in the server and the clients, they weren't receiving and sending the stuff properly. I also worked on the game logic of the program. I also helped Ria create UML diagrams after our program was done. For me, implementing the logic directly into the GUI did not work so I first created the program in the command line.

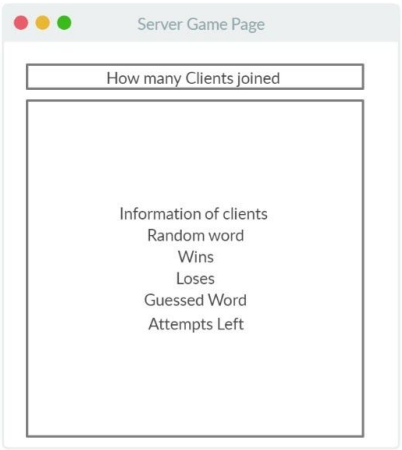
Ria: Server, Game Logic, UML diagram

I worked on the game logic along with Het, we created a command line logic and then merged it with the GUI of the game. At first it was getting messier with fixing the GUI and the logic together, so we decided to make the logic work at the command line first. I also worked on the UML diagram along with the wireframe.

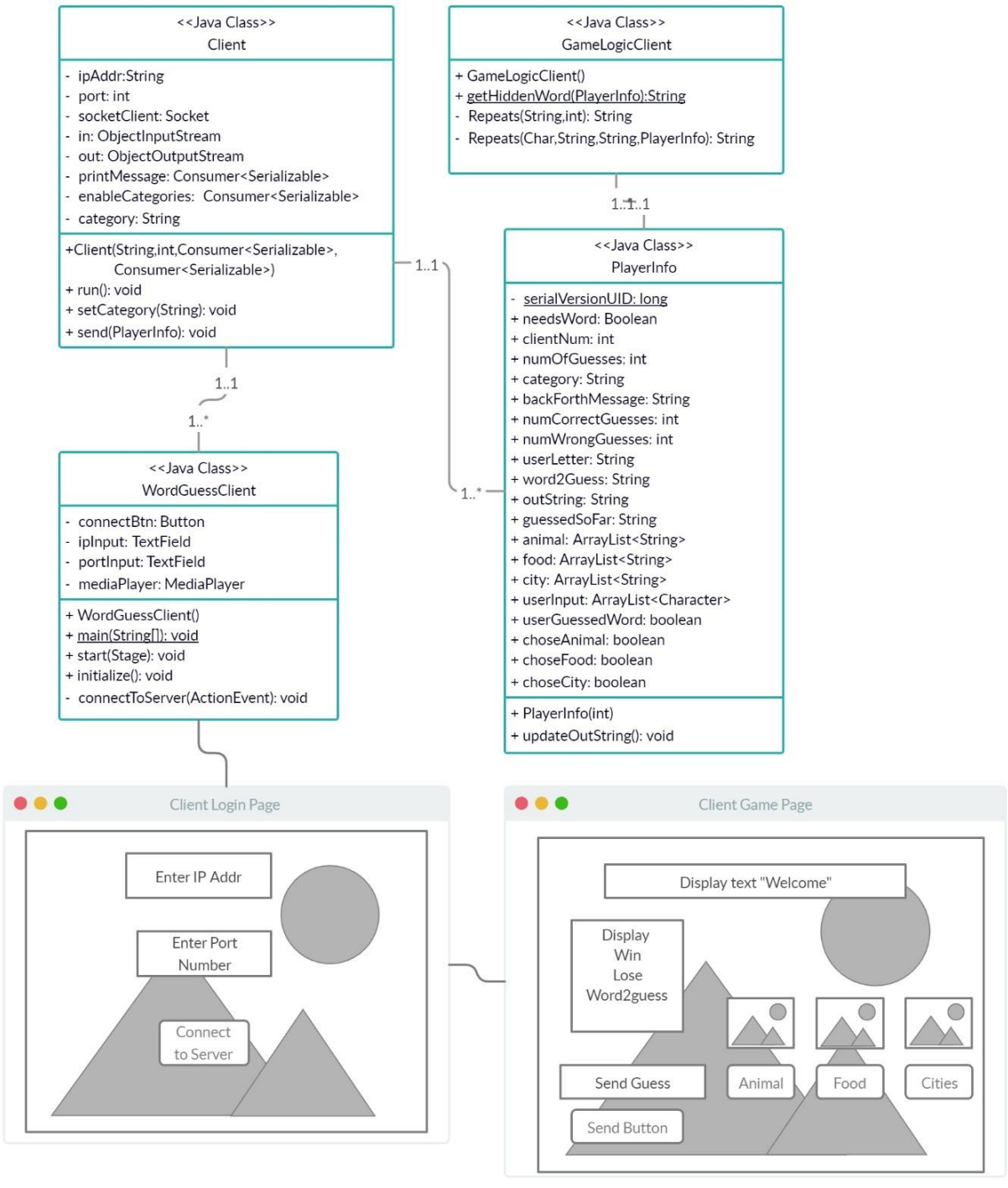
Server UML


<<Java Class>>
GameLogic
+ GameLogic()
+ getRandomWord(ArrayList<String>): String
+ getHiddenWord(PlayerInfo): String
<<Java Class>>
Server
- callback: Consumer<Serializable>
- gameStatus: Consumer<Serializable>
+ count: int
+ wins: int
+ losses: int
+ Server(Consumer<Serializable>, Consumer<Serializable>)
<<Java Class>>
PlayerInfo
- serialVersionUID: long
+ needsWord: Boolean
+ clientNum: int
+ numOfGuesses: int
+ category: String
+ backForthMessage: String
+ numCorrectGuesses: int
+ numWrongGuesses: int
+ userLetter: String
+ word2Guess: String
+ outString: String
+ guessedSoFar: String
+ animal: ArrayList<String>
+ food: ArrayList<String>
+ city: ArrayList<String>
+ userInput: ArrayList<Character>
+ userGuessedWord: boolean
+ choseAnimal: boolean
+ choseFood: boolean
+ choseCity: boolean
+ PlayerInfo(int)
+ updateOutString(): void
<<Java Class>>
ClientThread
+ connection: Socket
+ count: int
+ in: ObjectInputStream
+ out: ObjectOutputStream
+ ClientThread(Socket,int)
+ updateClients(PlayerInfo): void
+ run(): void
<<use>>
<<Java Class>>
TheServer
+ TheServer()
+ run(): void
<<use>


Server Login Page
Enter Port Num
Connect Button
Help


Server Game Page
How many Clients joined
Information of clients
Random word
Wins
Loses
Guessed Word
Attempts Left

Client UML


<<Java Class>>
Client
- ipAddr:String
- port: int
- socketClient: Socket
- in: ObjectInputStream
- out: ObjectOutputStream
- printMessage: Consumer<Serializable>
- enableCategories: Consumer<Serializable>
- category: String
+Client(String,int,Consumer<Serializable>, Consumer<Serializable>)
+ run(): void
+ setCategory(String): void
+ send(PlayerInfo): void
<<Java Class>>
GameLogicClient
+ GameLogicClient()
+ getHiddenWord(PlayerInfo):String
- Repeats(String,int): String
- Repeats(Char,String,String,PlayerInfo): String
<<Java Class>>
PlayerInfo
- serialVersionUID: long
+ needsWord: Boolean
+ clientNum: int
+ numOfGuesses: int
+ category: String
+ backForthMessage: String
+ numCorrectGuesses: int
+ numWrongGuesses: int
+ userLetter: String
+ word2Guess: String
+ outString: String
+ guessedSoFar: String
+ animal: ArrayList<String>
+ food: ArrayList<String>
+ city: ArrayList<String>
+ userInput: ArrayList<Character>
+ userGuessedWord: boolean
+ choseAnimal: boolean
+ choseFood: boolean
+ choseCity: boolean
+ PlayerInfo(int)
+ updateOutString(): void
<<Java Class>>
WordGuessClient
- connectBtn: Button
- ipInput: TextField
- portInput: TextField
- mediaPlayer: MediaPlayer
+ WordGuessClient()
+ main(String[]): void
+ start(Stage): void
+ initialize(): void
- connectToServer(ActionEvent): void
1..1
1..*
1..1
1..*
1..1
Client Login Page
Enter IP Addr
Enter Port Number
Connect to Server
Client Game Page
Display text "Welcome"
Display Win Lose Word2guess
Send Guess
Animal
Food
Cities
Send Button