PROJECT 5 DESIGN DOCUMENT

Team 13

MAY 1, 2019

Saccha Agriel, Derek Altman, Heemani Brahmbhatt, Sakina Master

Section I, Program Functionality:

This document delineates the design for the program for the game Mystery Number. The rules/specifications of the game are explained in detail in the program's README file of the source code. The project framework is a network interaction between a single server and multiple (4+) clients. The majority of this interaction is visible in the NetworkConnection.java class of the respective programs. The main function of the server is to ensure the smooth event-flow of the clients that are connected to the game and communicate messages to client programs at appropriate intervals in the game.

The main responsibility of the Client program in this scenario is to provide a hypothetical user/player with adequate user interface elements to interact with other clients using the server as an intermediary. The client program should allow for all of the tools necessary to play the game, including display and interactive buttons, as well as other features that make it a visually appealing experience to play the game. To this effect, there are background scenes and music that add to the overall experience.

Section II, High-Level Design:

Some of the main structures that compose the overall design at a high-level glance:

- Server GUI: The JavaFX components required to build the server GUI programmatically are encompassed under this section. The visual effects surrounding the user interface was an important development in this regard. This also includes the necessary **TextAreas, TextFields, and Buttons** to provide functionality in terms of mediating the behavior of each client instance and communicating updates in that status across the clients connected.

- Client GUI: This structure to the program encompasses the JavaFX components required to design the client GUI programmatically. It aims to create all of the **Buttons and Text components** to provide the client with game instructions as well as a cohesive routine to how the game will function. The layout of the game on the scene suggests a step-by-step chronological flow to how the client should interact with the server and other clients. The GUI also enables and disables certain features (such as the submit button) as necessary.
- Network Connection: This segment of code is perhaps the most vital in maintaining a channel of communication between the server and clients. It is the adhesive that binds the flow of the game and orchestrates the coordination along the way in a timely manner (GUIs update in real time as messages are sent via server). The Network connection is the backbone of how the server can share information with the clients and it is the framework for reacting to events that take place on the client GUI. It operates the event-driven nature of the gam with the use of multi-threading.

Section III, Low-Level Design:

- Server GUI: The majority of the server GUI is dedicated to interacting with the clients.
    - The createContent() function sets up all of the programmatic elements such as those listed in the UML diagram. The setting of the Scenes, BorderPanes, and Buttons/TextFields is a part of this function.
    - The ServerOn button handles the logic change for server GUI. The ActionEvent for this button is what begins the listening for connections on the port and other basic networking function setup.

- Within the ServerOn ActionEvent, another major part is the Platform.runLater() segment. This controls all of the events that must be queued in order to update the client GUIs as a response to an incoming message from any of the clients.
- Overriding Start/Stop methods: we have included the overridden implementations of the start and stop methods that Java Application offers. The start method must be implemented as required by the JavaFX documentation. The benefit to overriding the stop method is for a graceful way to close the client connections instead of throwing Exceptions. It's an effective way to catch the exceptions being thrown and handle them in a seamless manner.

- Client GUI: The client GUI is a minimalist design in the sense that not much of the networking responsibility falls to the clients' shoulders. This is a design choice on the team's part. While not the only solution to providing the program proper networking functionality, it is the design path we chose to go down.
  - Start method: The start method instantiates all of the minor fields that add to the appearance of the program and are vital to communicating with the server. primaryStage is the entity that encompasses the data of this type. Some of the Buttons controlling the game functionality are expressed in further detail below.
  - Connect Button: It is the ActionEvent for this button that triggers the network connection code to be executed. This button propogates a try-catch sequence that continues to operate until that specific connection is closed from the client side.
  - Submit Button: This is the button that clients will be using the most throughout gameplay. This button sends the choice of client to the server and the server updates the "Numbers Guessed" TextArea for all of the clients as a result.

- Quit Button: This button terminates the client-side connection and in the case that this puts the clientCount below 4, the game will remain paused until there is at least one other client that joins.
- Extra Features:
  - Music: There is also the added functionality of enjoying the nice ambiance of some background music while the client plays the game. Of course, if the client's taste in music does not align with that of the developers, a MUTE button has also been included in the design.
  - Instructions Manual: The client can click on the INSTRUCTIONS button at any time to change to a scene that lays out the instructions of the game.

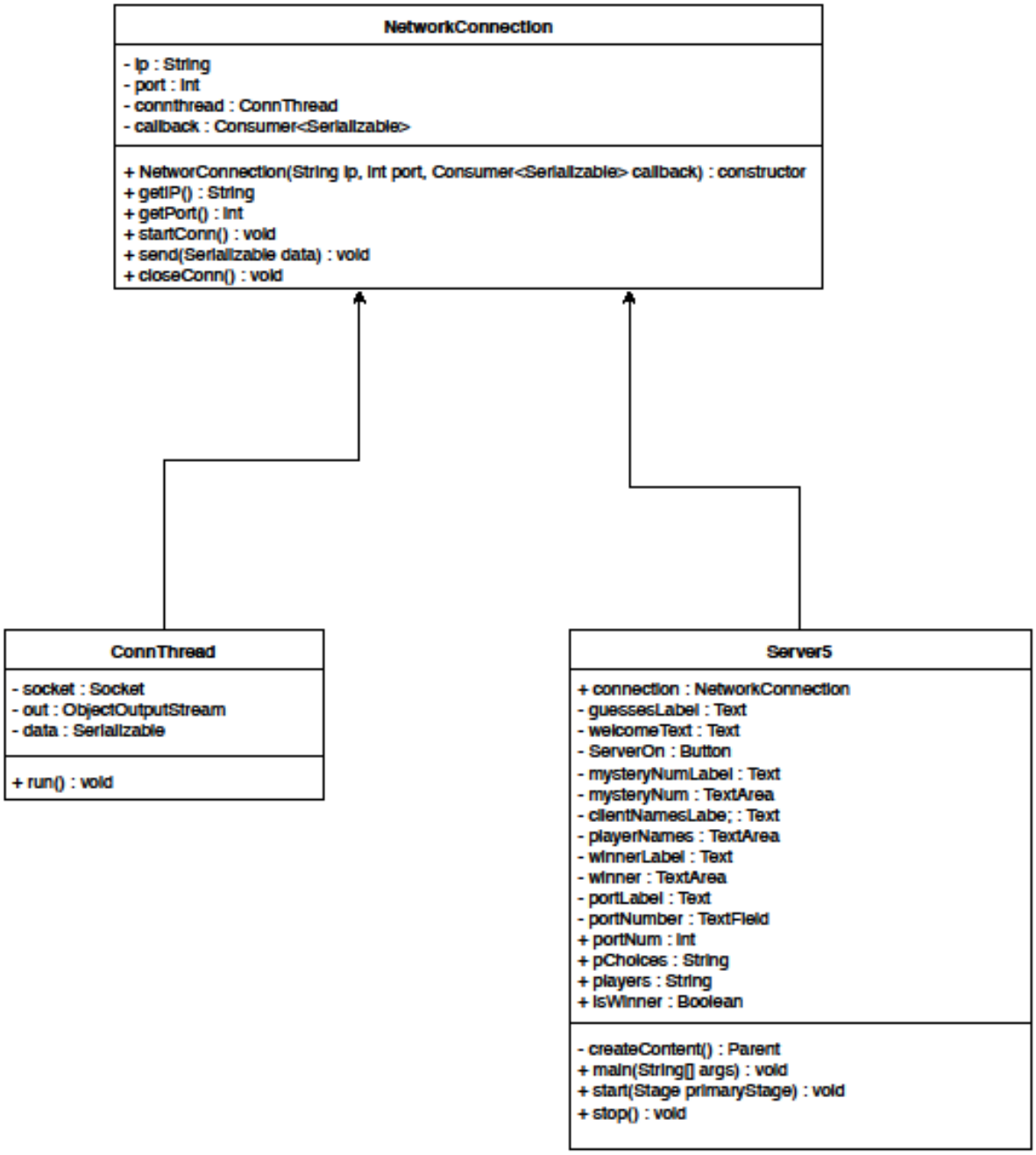
- Network Connection:
  - Server Connection: This class includes the multi-threading capabilities required to operate a cohesive game experience. The ConnThread continues to listen and spawn new ClientThreads as more clients join the game.
    - One of the most noteworthy structures implemented here are the various ArrayLists maintained at class scope in the beginning. These are the facets to sending messages to clients and updating their GUIs in real-time. This was the design decision of the team specific to this kind of a function.

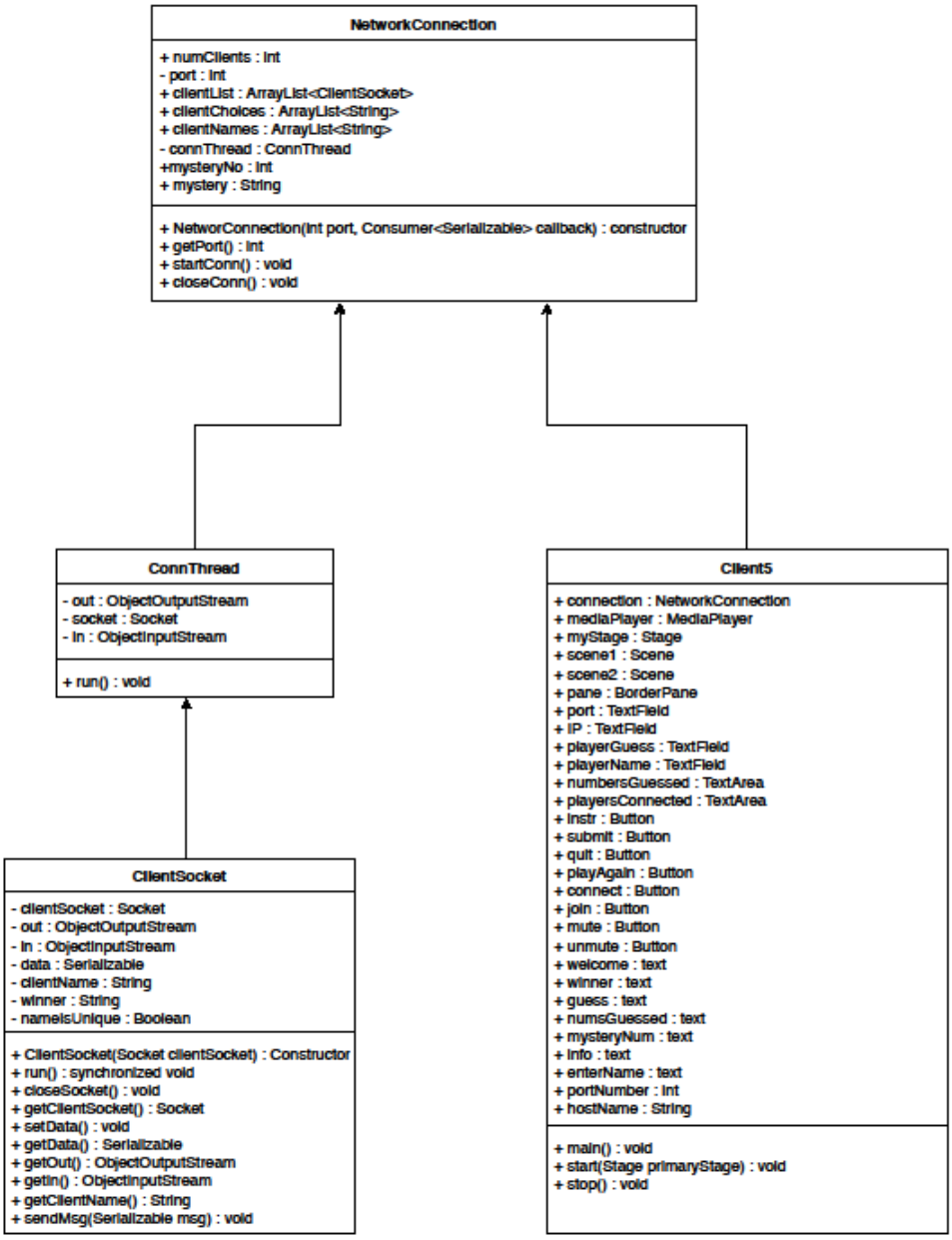
    Apart from the major trading of information that takes place here, this is also where the mystery number central to the game is chosen and set by the server before a game starts.

- Client Connection: The client NetworkConnection is not as involved in terms of data structures and flow as its server counterpart. Most of the constructs are similar to that of the server.
    - One noteworthy addition here is the extra conditions set that disables a client from choosing a name that has already been taken. It provides for the unique name functionality.

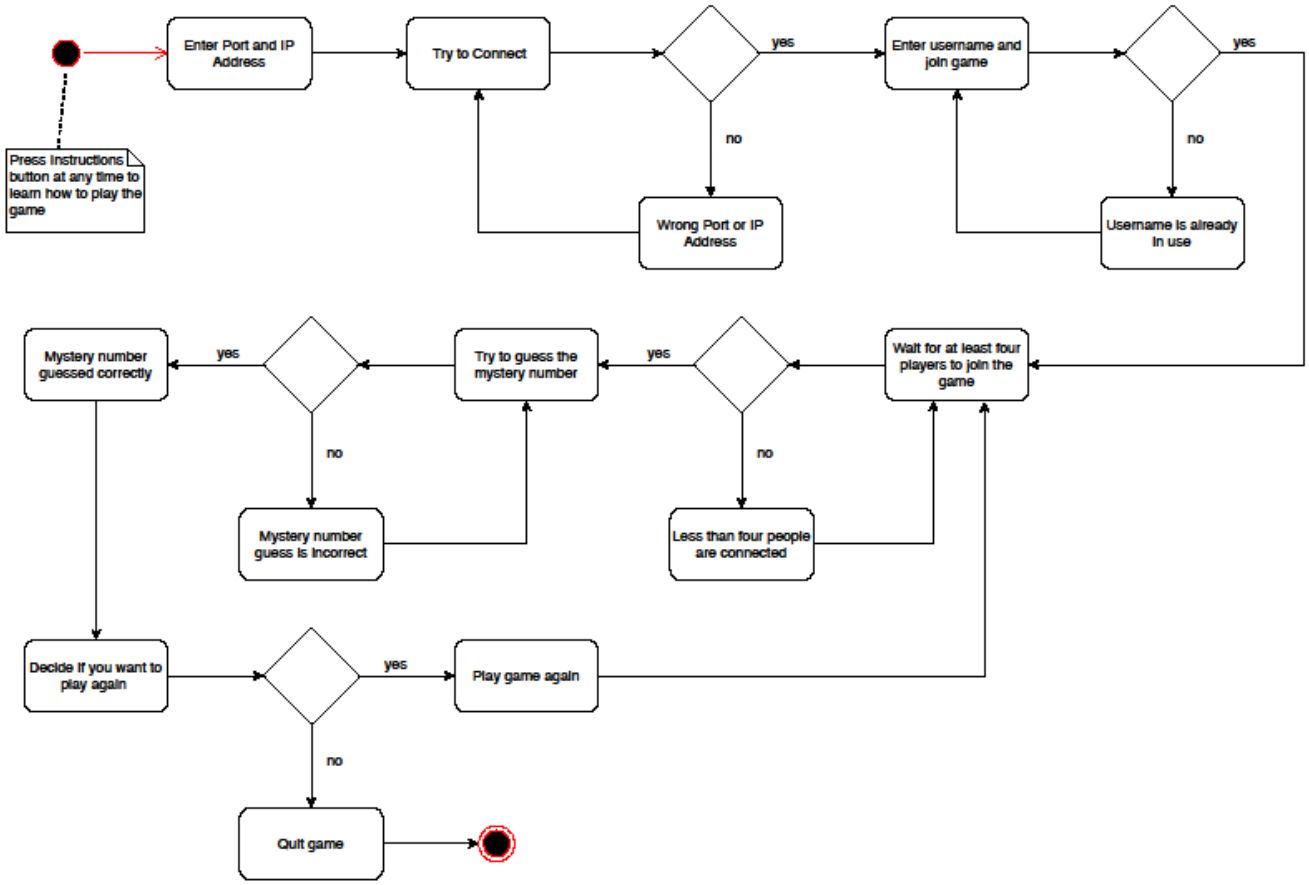
  This is also where a lot of the GUI updating for the client part is reflected.

Section IV, UML Models:




NetworkConnection
+ numClients : Int
- port : Int
+ clientList : ArrayList<ClientSocket>
+ clientChoices : ArrayList<String>
+ clientNames : ArrayList<String>
- connThread : ConnThread
+mysteryNo : Int
+ mystery : String
+ NetworConnection(Int port, Consumer<Serializable> callback) : constructor
+ getPort() : Int
+ startConn() : void
+ closeConn() : void
ConnThread
- out : ObjectOutputStream
- socket : Socket
- in : ObjectInputStream
+ run() : void
Client5
+ connection : NetworkConnection
+ mediaPlayer : MediaPlayer
+ myStage : Stage
+ scene1 : Scene
+ scene2 : Scene
+ pane : BorderPane
+ port : TextField
+ IP : TextField
+ playerGuess : TextField
+ playerName : TextField
+ numbersGuessed : TextArea
+ playersConnected : TextArea
+ instr : Button
+ submit : Button
+ quit : Button
+ playAgain : Button
+ connect : Button
+ join : Button
+ mute : Button
+ unmute : Button
+ welcome : text
+ winner : text
+ guess : text
+ numsGuessed : text
+ mysteryNum : text
+ info : text
+ enterName : text
+ portNumber : Int
+ hostName : String
+ main() : void
+ start(Stage primaryStage) : void
+ stop() : void
ClientSocket
- clientSocket : Socket
- out : ObjectOutputStream
- in : ObjectInputStream
- data : Serializable
- clientName : String
- winner : String
- nameIsUnique : Boolean
+ ClientSocket(Socket clientSocket) : Constructor
+ run() : synchronized void
+ closeSocket() : void
+ getClientSocket() : Socket
+ setData() : void
+ getData() : Serializable
+ getOut() : ObjectOutputStream
+ getIn() : ObjectInputStream
+ getClientName() : String
+ sendMsg(Serializable msg) : void

Section V, Interaction:



Section VI, Evaluation/Prospects for Further Development:

One of the assumptions that this program makes is that the user is comfortable with a desktop application instead of an online based game (such as something Node.js could provide).

One of the alternative design choices would have been utilizing the Node.js framework. In such a case, the costs and benefits of each must be evaluated. Node.js is singly threaded and would have reduced overhead than the current multithreaded JavaFX platform. However, one of the advantages of using JavaFX was its seamless support of incorporating CSS which made the styling of the client GUI much more convenient. We acknowledge that design choices come with trade-offs, and in our group's opinion, JavaFX was more fit for this purpose. Overall, we realize that the game's complexity can be improved, and user experience can be upgraded visually by adding some subtle features as necessary. All pieces of software are a work in progress, and this is no exception. We look forward to being able to develop more aspects as we get suggestions and feedback from users.