Bingo Game Design Document  
Jakub Krzeptowski-Mucha

The purpose of our project was to create an online version of the classic game of Bingo. Our target audience is anyone who has access to a computer and would like to play others in a game of Bingo. The game itself is played between clients connected to a server. There must be a minimum of four clients connected before a game can start. The way the game works is that the clients and server communicate by sending strings of instructions back and forth. These instructions contain the necessary data and conditions for the game to be played. For example, when a client first connects to the server, the server will send the playerID and the bingo instruction which sets the clients playerID and their bingo card. This instruction effect can be seen on the client’s GUI as the playerID and bingo card will be updated.

The most important component to this project is the instruction interpreter. Since both the server and client have an individual FX thread to run their GUI’s and a thread for socket communications, they have two instruction interpreters to decode any instructions received. One interpreter is decoding the string instructions being sent back and forth via socket communications. These instructions may set conditions or data members. If the instruction needs to update the GUI, the other instruction interpreter will have the necessary procedures to parse the data and output it onto the GUI.

Another important component to the project is the GUI. The GUI is the direct method of communication between the client and the server. The client’s GUI offers the ability to send a number of instructions to retrieve certain information or to signify a potential Bingo. Each client can send these instructions from the click of a button. For example, if a client wanted to see the history of drawn numbers, they would click the history button which sends an instruction to the server. The server interpreter then decodes the history instruction and sends back the relevant data via another instruction. The client then receives this instruction, interprets it, and updates the GUI accordingly.

In order to send instructions to one another, the client and server must keep track of each other ObjectOutputStreams. It is through these streams that the instructions actually travel back and forth. If a client wants to request a certain piece of data, they simply press the respective button which sends the instruction by writing to the ObjectOutputStream from which the server can receive it. Once the server sends the relevant data back through an instruction, the instruction interpreter will decode the data. If the client’s GUI needs to be updated, the runnable (which is a runLater) will be called through the clients socket communication thread. By calling the runnable, this gives us the ability to modify the GUI by using the next instruction interpreter through the FX thread.

The instruction interpreter proves useful as it is easy to implement and add onto. There is no limit to the amount of instructions the server or client may receive and so the interpreter must be able to expand accordingly. A downfall of the instruction interpreter that exception handling had not been implemented in the case of valid instructions being formatted incorrectly. Upon receiving an invalidly formatted instruction, the client will receive an exception message through the terminal. However, though the exception handler had not been implemented the case of invalidly formatted instructions, the game can still continue to be played. In the case of undefined instructions, the instruction interpreter will simply disregard the instruction and continue.