Jeopardy Project Report

What Is Our Project Idea?

For this semester's project, we had to create a project that required the use of Socket programming. Socket programming is “a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while the other socket reaches out to the other to form a connection. The server forms the listener socket while the client reaches out to the server” (GeeksforGeeks). Our group project idea was to create a Jeopardy game. In this project, multiple players can connect to the host and play the game together. Jeopardy originated in the United States of America and is played by millions of people in the form of both board games and online multiplayer games. This game involves a quiz competition where contestants or players are presented with numerous questions specialized for different types of audiences in the form of answers or phrases. They must respond in the form of questions.

Why Did We Choose TCP Over UDP?

In socket programming, there are two types of connections that can be used. UDP and TCP connection. TCP (Transfer Control Protocol) connection allows extremely well-founded reliable communications between different applications. TCP is commonly used over the internet protocol, which is why TCP is commonly referred to as TCP/IP (TCP over the IP). On the other hand, we have UDP (User Datagram Protocol) which does the same jobs as TCP but with no guaranteed reliability. UDP is a protocol that “transmits independent packets over the network with no guarantee of arrival and no guarantee of the order of delivery” (Baeldung). Because there is less overhead in UDP it is much faster than TCP. Many applications used UDP because they

don’t need a hundred percent reliability, for instance, streaming services don’t require TCP, as losing a few frames won’t affect the service drastically. In doing this project, we decided to go with the TCP model, because we didn’t want to lose any of the text that flows between clients and the host. In this project, we needed to compare the string or answers of the clients to see if they have the correct answer. If we lose bits of the strings, it will be difficult to check the player's answers.

For this project, we decided to code in Java because my group knew how to program in Java the best. We were thinking of using python because of all the tools and information online on socket programming but decided against it as no one was too comfortable with python and we didn't want to deal with learning a new language.

How To Play the Game?

For this project, we couldn’t fully implement the features of Jeopardy, but we implemented the core functions of the game. To play the game the host must first start the server-side of the program. The clients or players are then able to join by running the client-side of the program. Players are able to join from different devices as long as they're on the same Wi-Fi network and have the same socket. The clients also need to know the IP address of the host. The host will notify the players that the game is starting by broadcasting it through the instant message feature. Upon notifying the players that the game has started, the host will ask one player to go. The player will choose the field of knowledge they wanted to be tested on and choose the level of difficulty of the question. They will let the host know of their choice and use the instant message feature, to let the host know as well as the other players. Then the host will

notify the player of the question and the player will respond. The host will then check the answer and keep a record of the scores of each player.

What did Mahfuz do? (50 %)

In this project, I dealt with the frontend aspect of the project. For instance, I worked on creating the Jeopardy board, and the scoreboard. I found this to be difficult because creating front-end attributes in Java is difficult. I am usually comfortable making the front end with HTML, CSS, and JavaScript but in Java, we had to use Java libraries to do the front end, also known as GUI. I used a SWING to create components of the front end. Swing is an API and has a "set of extensible GUI Components to ease the developer's life to create JAVA based Front End/GUI Applications".

I first made a JFrame Object from the Swing library, which is like the window of the game, which then was modified and given attributes such as title, bounds, and resizable or not, etc. Then I made a Jpanel Object, a container in Java that will hold all other components of the game. Jpanel will help organize components.

I first created the title of the board; I just used a JTextArea by calling the constructor with empty parameters. Then I set the text of the JTextArea to "Welcome to Jeopardy 2022!", which I then styled using different set methods of the class. Some methods that were used are setForeground, setFont, setEditable, setBackground, setBound, setText. After styling the title of the game, I then added it to the main panel. I also created a scroll bar for the message area making a JScrollPane object and adding it to the main panel and setting its location in the panel.

For the scoreboard title, I made a new object of JTextArea and set attributes similar to the original title. For the message input area and the name input area, I also used JTextArea setting

similar attributes as the title. For the input specifically, I created the JTextField object, so users can enter messages and their names. Then the inputs, as well as the JTextArea holding “Name” and message, were added to the main panel. For the message area, I also added a scroll panel, so that when there are more messages users can scroll up and down. These are most of the components are put inside the Jpanel, the Jpanel keeps everything together in an organized fashion similar to that of DIVs in HTML.

For the scoreboard, I called the JTable object and passed it in a 2d-array to store the name and score of each player. I also passed in a second parameter for the names of the 2d array column, “Name” and “Points”.

There were a few problems I countered while creating the GUI of the jeopardy game. Getting the scroll pane to work properly took some time. When a lot of messages were being sent the scroll bar wasn’t showing up, after some time debugging, I realized I was using the object the wrong way. Another problem I had was positioning all the different components in the main panel. It took a lot of running the program, checking the window, and adjusting the positions. Furthermore, I wasn’t sure how to implement a scoreboard, so I had to take some time and do research to learn how to create it. This also applied a scroll bar for the message box.

For the jeopardy questions, we just scraped the internet for random questions with answers. To implement this feature, we used different components. To show the score of each box we just used a for loop iterating over the rows and columns, doing arithmetic which resulted in the box’s score. For the boxes, we also stylized them using different features such as background color, foreground colors, size, and other things. When users click on the box with the score the question is displayed. After clicking on the box, the box shows the questions, and

the box is blacked out to show that someone already used it. To store the questions and answers we used a switch statement and had various cases for the multitude of questions and answers.

What did Redwanul do? (50 %)

My goal in this project was to implement the socket connection using TCP. While my partner also helped with the socket connects, I did most of the part. When doing the socket connection, I didn't know where to start. After some research on the internet, I came up with some ideas as to how to create a socket connection in Java. When implementing the socket connections, I first started with the host server and then moved on to the client-server. The socket connection between the host and the client-server is fairly similar, so once I got the host server socket, I just copied that and pasted it into the client-server and modified the code a little. After setting up the socket, I then proceeded to test the sockets by sending a message into the console and seeing if both sides receive the message. After some trial and error, I finally got them to work. After this process, I then moved on to the big step, which was creating multiple socket connections using the GUI interface. After Mahfuz, set up the frontend version of the program, I then started to modify my code to make the socket work in the GUI interface, which will be the main window for the Jeopardy game.

In the GUI interface, there are 3 main boxes, and they are getName (user will enter their name), getMessage (user will enter their message), and textaArea (display the name with the text). First, to get the messaging aspect of the socket to work in the host server, I added an action listener which will get the name and the messages of the host. In the getMessage action listener, I first added an if loop which checks if the input is valid, if they are then move on, if not then move to the else if loop which will ask the user for the name input and message input. After that,

I did a for loop to connect the client with the host and append the host messages to the client-server. In the for loop, first, there is a hash map that stores all the clients that are connected to the server. After the for loop is settled, there is a try and catch statement which writes the data output stream to the client-server. When this was finished, I just appended the getName and getMessage to the textArea so whatever the host writes, it will appear on the host side of the GUI interface. On the client-side of the interface, for the client to receive the host message, I made 2 string variables which were hName and hMessage and connected them with the data input stream. After the socket connection was set, I then proceeded, to append both strings to the client GUI interface. After some trial and error with setting up the connection, I was finally able to send messages from the host to the client-server.

After the host server messaging system was set up, I then started with the client-server. And I have to say this was very tough, not what I expected. In the client-server, I just copied the host server action listener and just pasted it into the client-server. After that was set up, I then proceeded with setting up the thread in the host server, so the message sent by the client will appear on the host server. The thread was very difficult but with some research and trials and error, I finally got it to work. On the thread run method, there is a while loop that accepts all the messages sent by the client and put them to strings which does a data input stream. After that, I just append the strings to the textArea. With that done both the host and the client were able to send messages to each other.

After the basic host and client-server messaging system were set up, I finally proceeded with the multi-client server. To do the multi-client-server I had no idea where to even begin. So, I did some research on the internet and found out that I can implement a synchronized method that will handle the host and multiple clients being able to receive each other messages. To get

started with the synchronization, I first put that inside the thread run method on the host server. Inside the synchronization, I first did a for loop which gets all the clients that are connected to the server after that I did a try and catch statement which writes the multi-client-chats with the host and the other clients. At first, it wasn't perfect, but with some trials and error, I was finally able to get the multi-client aspect to work.

There were many issues that I faced while implementing the socket connection. One problem was that when the host send messages to multiple clients, the message would repeat in the host textArea for all numbers of clients that are connected to the host server. To fix the error, I just put the textArea append outside of the for loop and that handled the error. A second issue was that figuring out how to connect to the client server with sockets. On the internet, I did some research and found out that I had to do data input stream and output stream. Another issue that I faced was figuring out how to handle the multi-client-server. After wasting sometimes, the solution that hit me was just make a hash map and store all the clients in the server there.

Things We Could Have Improved On

To begin with our project is not perfect. Nothing is ever perfect, but it can always be improved. There are many things we can do to improve our project in the future. One thing we can improve on is to make the box picking synchronized. For example, when the client picks a box, that same box will be picked in the host server. My group tried to implement this feature, but it was somewhat difficult to do. Maybe in the future once we have more knowledge about socket programing, we can make a change to this. Another thing we can improve on make the host server score board synchronized. Just like picking the box, my group also tried to implement this feature but once again it proved to be very difficult.

What did we learn?

This project was by far one of the best projects that me and my group did. There are many things that me and my group learned while doing this project. One thing is that our programing skills in Java have improved a lot. A second thing that we learned by doing this project is that our knowledge about socket programing have improved a lot. Before doing this project, we had very little knowledge about implementing sockets, but after doing this project our knowledge has improved significantly. Another thing we learned was that our values in teamwork. Teamwork makes the dream work. At first this project was a lot, but after dividing the parts to our group mates, the task have been reduced significantly.

References (used to understand Socket Programing)

1. **https://gyawaliamit.medium.com/multi-client-chat-server-using-sockets-and-threads-in-java-2d0b64cad4a7**
2. **https://www.geeksforgeeks.org/multi-threaded-chat-application-set-1/**
3. **https://www.youtube.com/watch?v=gchR3DpY-8Q&t=232s&ab_channel=WittCode**
4. **https://www.youtube.com/watch?v=gLfuZrrfKes**