A. Game Name, Student names

Game: Rock Paper Scissors
Name: Jasper Reddin

B. Video Demo Link

https://www.youtube.com/watch?v=Z0NQMHf6TJk

C. Introduction

The game implemented was Rock Paper Scissors. Both players choose between Rock, Paper, or Scissors. Rock loses to Paper, which loses Scissors, which loses to Rock. If both players select the same move, it results in a tie. This particular implementation also uses rounds; players will go three times, and after the third round, the winner is decided.

This implementation is multiplayer. One process, the server, accepts client connections, and controls the game. Another process, the client process, connects to the server, listening for game updates and sending moves when the player makes a move.

D. Architecture Design

This implementation uses the TCP/IP server/client design pattern. One server process acts as the manager, and two client processes join to play the game. All three processes run independently from each other, and use sockets and streams to communicate with each other.

Communication

The server and client socket communication was built from scratch using Java's Socket APIs. Upon starting, the server will start a thread that listens for new connections until the server is stopped or something goes wrong. When a client is connected, an ObjectOutputStream and an ObjectInputStream is associated with the connected socket using the ClientThread class, and a new thread is started to begin listening for incoming data until the client is disconnected for any reason. In addition to the main thread, the server will always have n + 1 threads running, with n being the number of clients connected.

The client, upon starting, will connect with given host and port. Once connected, an ObjectInputStream and an ObjectOutputStream is set up, and a new thread is launched to listen for incoming data until the client is stopped or something goes wrong.

How do we send different types of data? How would we know the difference between receiving a chat message and a game move? We use Java's object streams in order to easily exchange data in the form of serializable classes. When an ObjectInputStream listens for data, it automatically builds the object instance and all we have to do in the handler is get the class of the received object.

API Events and Hooks

A beautiful software design involves compartmentalization of the programs. Therefore, both client and server socket communications were compartmentalized. In order to do this I created an interface for handling server and client events.

The ServerHandler interface has methods that are called when the server is started and stopped and when a client connects, disconnects, and sends a message. Through these methods, implementations of this interface have access to public methods of the ClientThread class that represents the connected client. Through this class, we can send the client messages or break the connection if needed.

Likewise, the ClientHandler interface has methods for communication events. There aren't as many events for the client, just one for client connected, client disconnected, and receiving messages.

The main class for our server process, ServerMain, implements the ServerHandler interface. Similarly, the main class for our client processes, ClientMain, implements ClientHandler.

User Interface

Both server and client programs utilize a Model-View-Controller design pattern. The main class of both programs act as the controller. The RockPaperScissors class acts as the model, and each program has their own view, both being a subclass of the JFrame class

The user interface for client and server are nearly identical; they both have a player list view, a chat log view, and a game view. There are a few differences. The server view has controls for starting and stopping the server but not for playing the game. The client has controls for playing the game and sending a chat message. Additionally, while the server shows all moves in a round, the client will not display the opponents move for that round if they went first.

Both view classes, ClientView and ServerView, build the view using Java Swing, add action listeners that communicate back to the controller, and have methods that the controller calls to update the various panels, such as chat log, player list, or game panel.

ServerMain

ServerMain is the controller class for the server program and contains the main method. It manages an instance of the Server class and keeps track of logged in players (and links each player with the associated ClientThread object) and the current game, updating its view when needed. In addition, it handles incoming data because it implements the ServerHandler. If incoming data is a move, it updates the current game accordingly, sending back the updated game to the clients and updating the view.

ClientMain

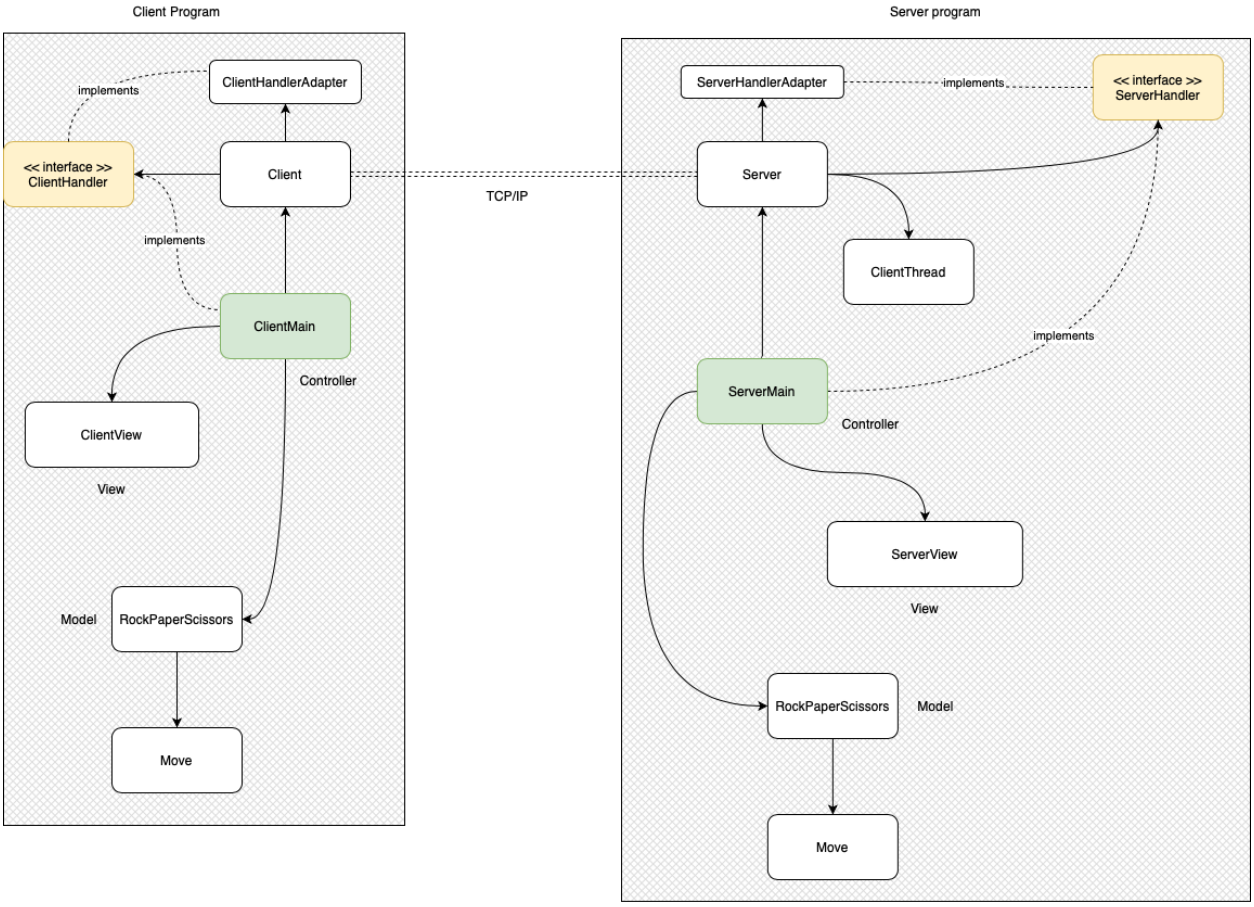
ClientMain is the controller class for the client program and contains the main method. It keeps track of a Client instance in addition to logged in players, chat messages, the current game, and other messages it receives from the server. Depending on the message it receives, this class will update the view accordingly. The view also sends this class click events, which prepares the data to send to the server. ClientMain implements the ClientHandler interface, handling communication events.

Model: RockPaperScissors and Move

RockPaperScissors is the model class, keeping track of players in the game, their moves, and the current round number. It calculates the winner if there is one, player scores, and whether or not the game is over. Additionally, it processes a move when the ServerMain sends it a MoveMsg.

Move is an enumeration and also contains the method for comparing itself to another instance. This is used when calculating the score.

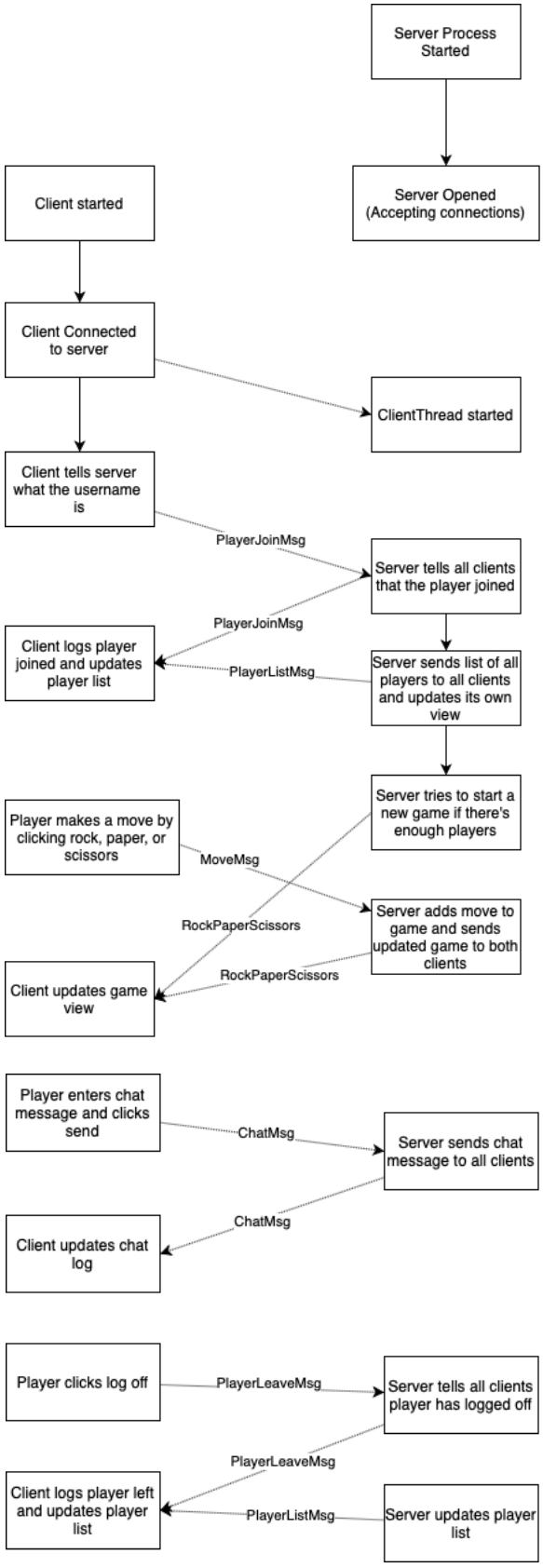
Class Diagram



This simplified class diagram shows the relationship between the different classes. Additionally it shows how TCP/IP is implemented as well as the MVC design pattern.

Program Flow

The following diagrams shows program flow of both client and server, as well as how messages are transferred and handled between the processes. The class name in between a client/server communication shows what type of object is sent in that particular case.

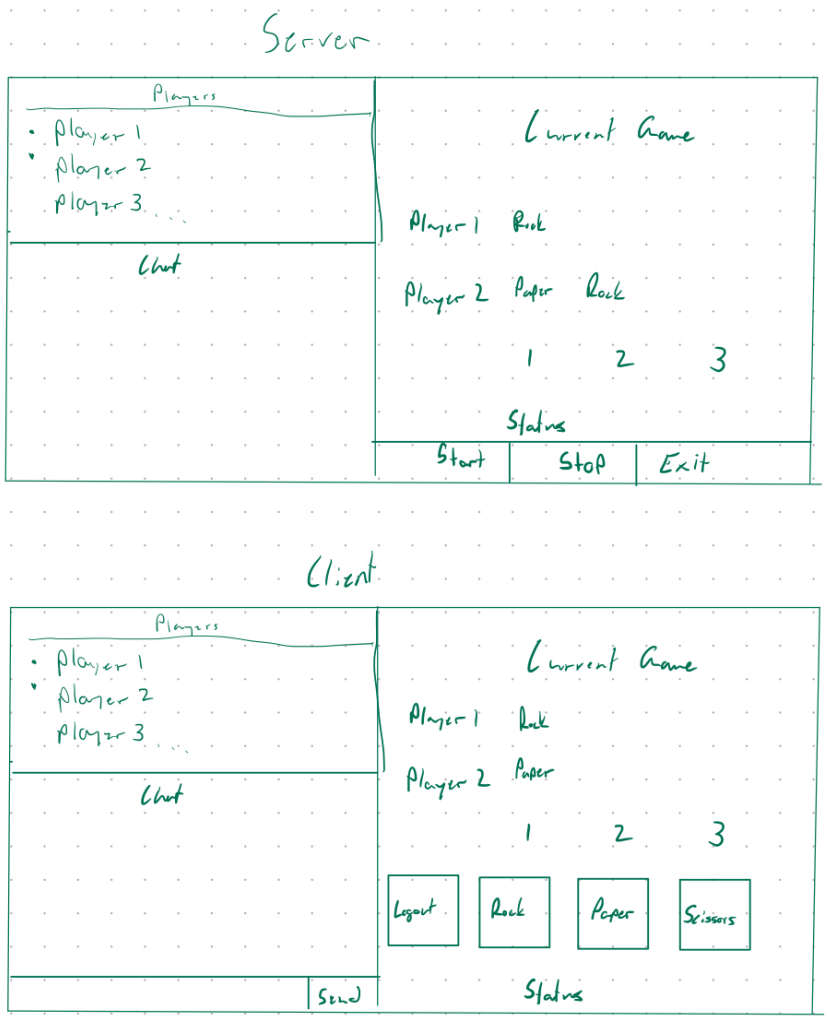

Server Process Started
Server Opened (Accepting connections)
Client started
Client Connected to server
ClientThread started
Client tells server what the username is
PlayerJoinMsg
Server tells all clients that the player joined
PlayerJoinMsg
Client logs player joined and updates player list
PlayerListMsg
Server sends list of all players to all clients and updates its own view
Server tries to start a new game if there's enough players
Player makes a move by clicking rock, paper, or scissors
MoveMsg
Server adds move to game and sends updated game to both clients
RockPaperScissors
RockPaperScissors
Client updates game view
Player enters chat message and clicks send
ChatMsg
Server sends chat message to all clients
ChatMsg
Client updates chat log
Player clicks log off
PlayerLeaveMsg
Server tells all clients player has logged off
PlayerLeaveMsg
Client logs player left and updates player list
PlayerListMsg
Server updates player list

E. Functional Requirements

1. Player must be able to login with a chosen username (nickname)
2. Player must be able to view players on the server
3. Player must be able to view current state of the game (each round, who wins, scores, etc)
4. Player must be able to start a game by making initial move.
5. Player must be able to make a move, either Rock, Paper, or Scissors
6. Player must be able to send a message in the chat
7. Player must be able to view chat messages
8. Player must be able to logout when desired

F. GUI Design

Prototypes of the GUI was drawn by hand before starting the development.



G. Implementation Environments

The entire project was built using Java. The GUI was built with Java Swing. Communication was implemented with Java's sockets and object streams. Both the client and server programs are in the same Java project, and compiled with the same code base. However, since there are two main classes, starting the server requires running the ServerMain main method, and starting the client requires running the ClientMain main class.

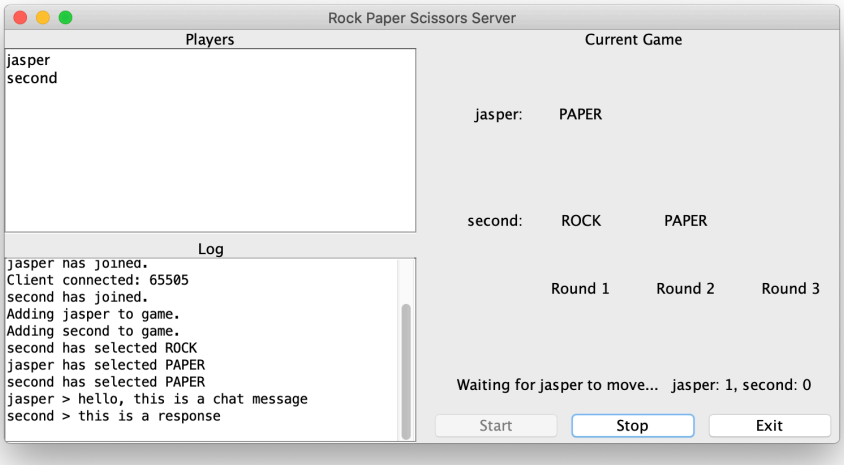
Two JAR files are prepared for execution. On most systems they can be double clicked in order to run. If not, or if Java is not installed in the traditional manner for that operating system, the programs can be launched in the command line using the Java command:

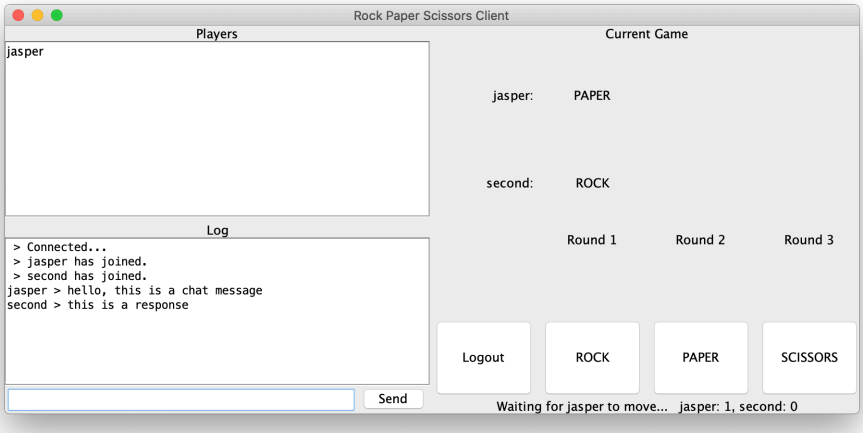
```
java -jar ServerMain.jar
java -jar ClientMain.jar <host:port> <username>
```

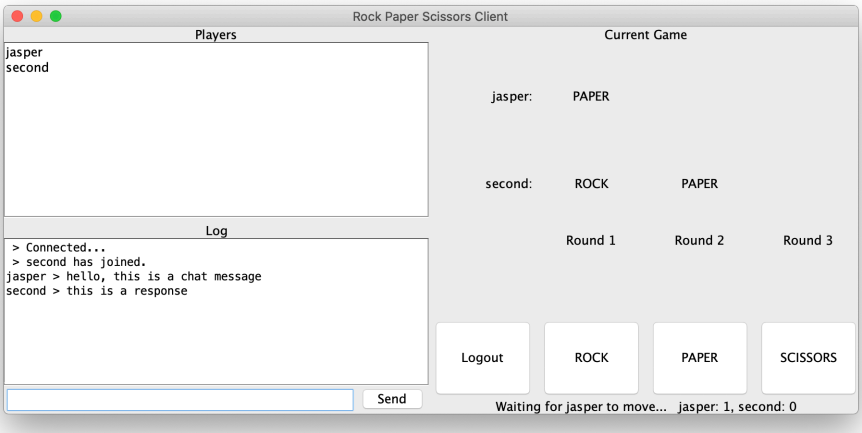
If command line arguments are not provided, the client will prompt for them before starting.

These programs will run on different systems if the network allows it and the associated firewalls allow it.

H. Screenshots






Rock Paper Scissors Client
Players
jasper
second
Log
> Connected...
> second has joined.
jasper > hello, this is a chat message
second > this is a response
Send
Current Game
jasper:
PAPER
second:
ROCK
PAPER
Round 1
Round 2
Round 3
Logout
ROCK
PAPER
SCISSORS
Waiting for jasper to move... jasper: 1, second: 0