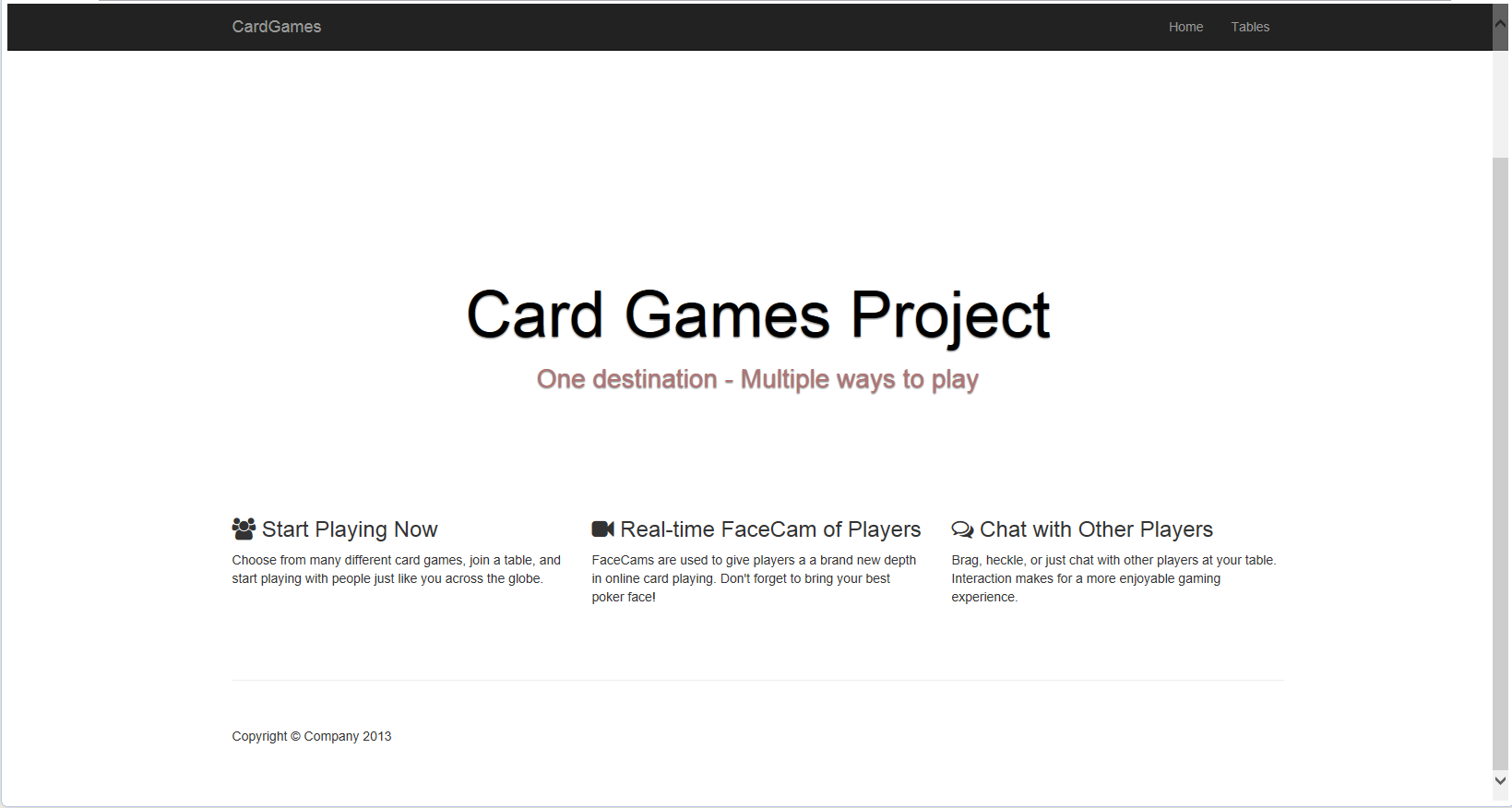
Online Card Game System

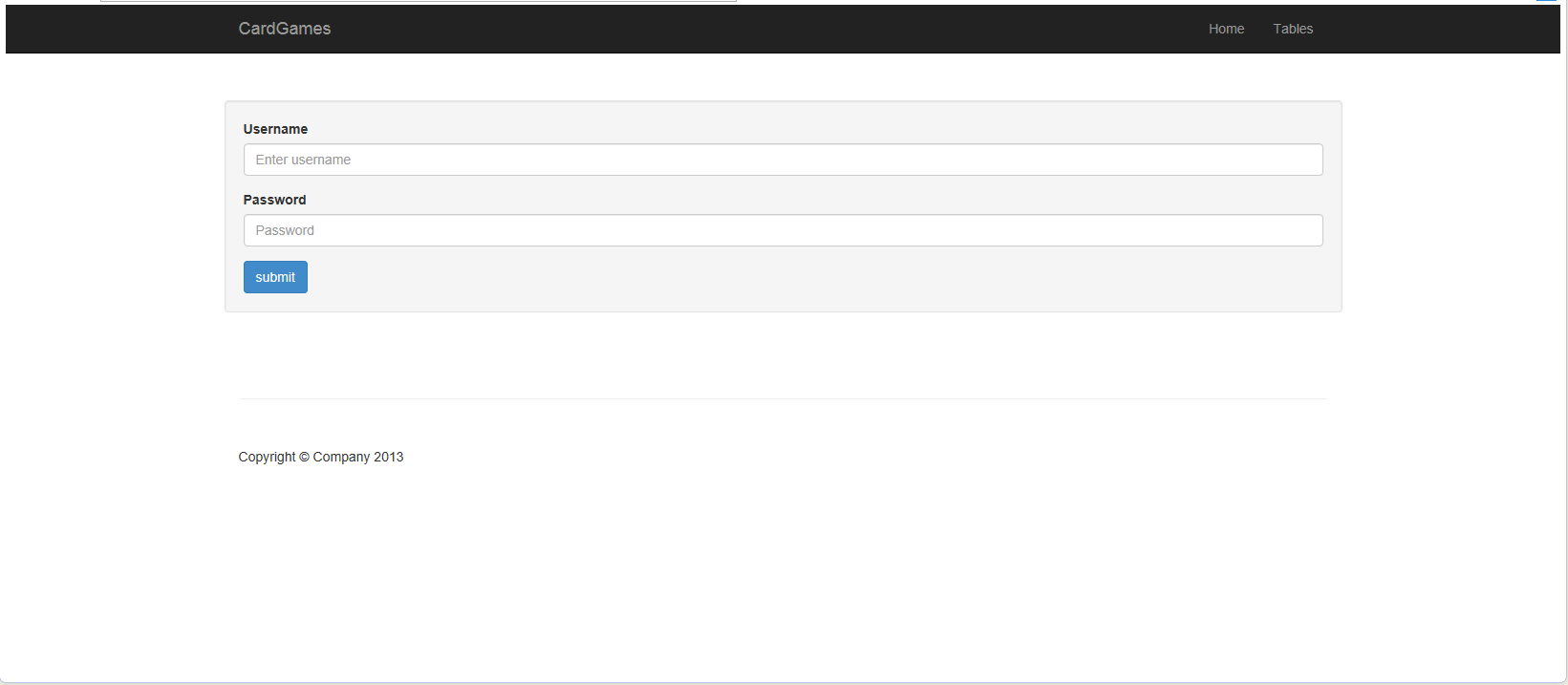
Our project is about demonstrating our ability to perform network programming by producing an online card game system. This system is set up to run on a server where clients from different computers will be able to connect to it by navigating to the URL in their preferred browser. Once the user is connected to the server, they will be displayed a landing page the described our system and what it all includes. From this landing page they will either have the option to login or they can create an account to access to games. After completing one of those processes they will finally be able to start playing the available card games. The user will see buttons corresponding to the types of games that are offered through this system. When the user clicks on one of these buttons, a list will appear showing the tables that are open for that user to play the game on. Clicking the “Join” button on one of those tables will take you to that game, place the user at a seat, and include him in the next available game. The rest of this report will outline different aspects of this system, giving development documents used to develop the system, sample code segments with explanations of what they do, as well as screen shots from the working product.



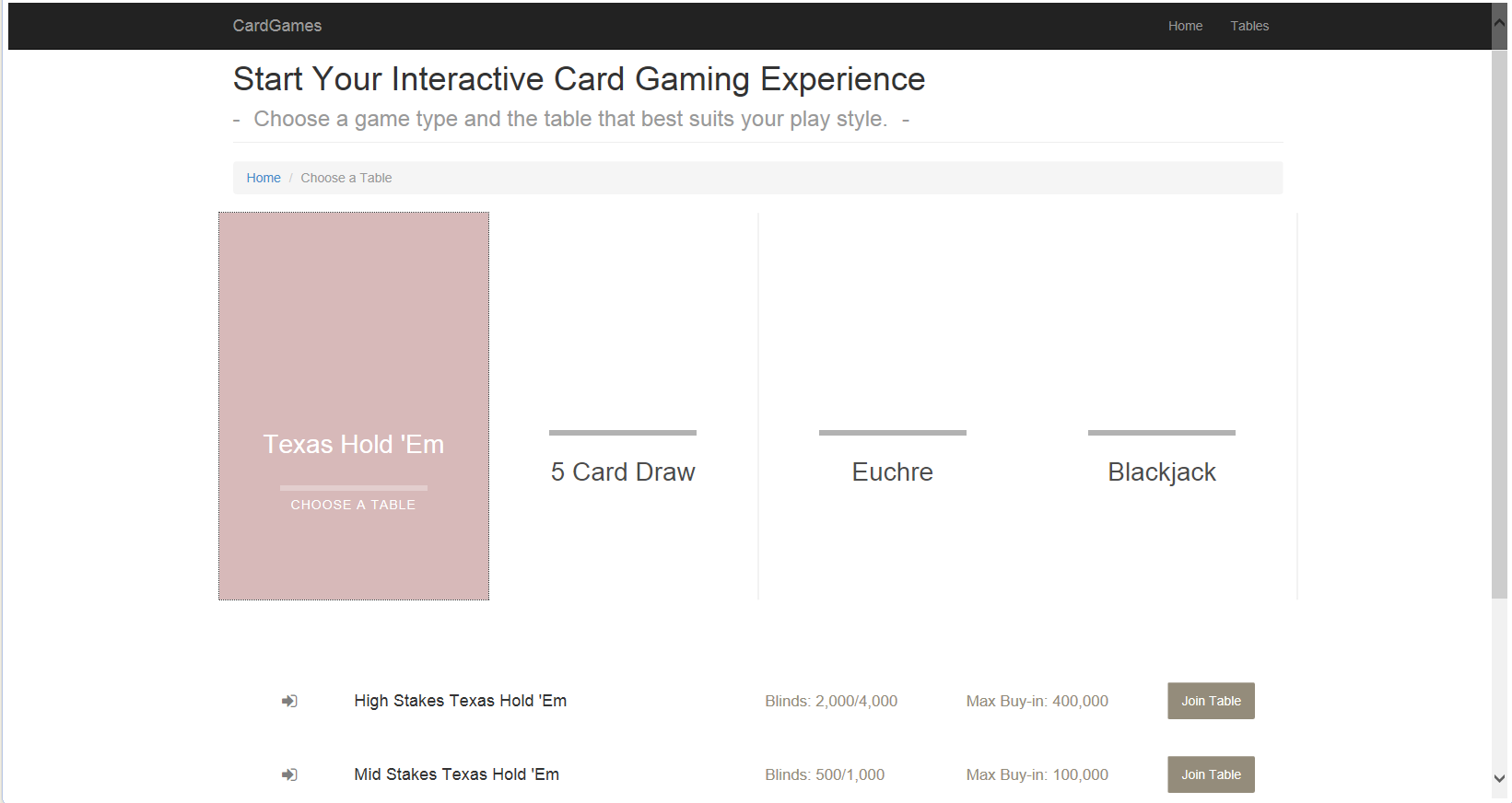
* **Main Components**

There are several main components that bring this game together to make a great experience for the user. Those components are as follows: Login, Game Selection, Video/Voice/Text Chat, and Game Experience.

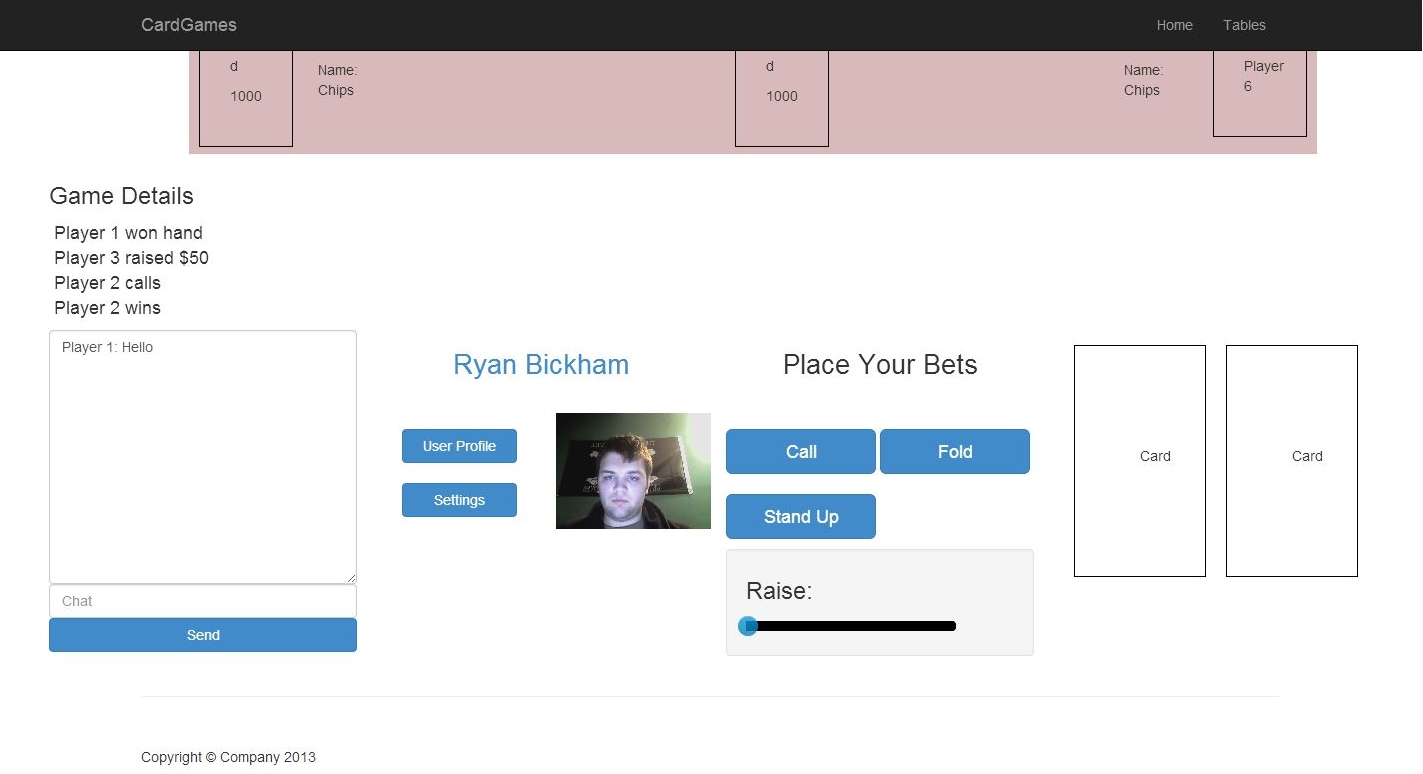
The Login feature allows the user to save their preferences and track their game statistics as they play the different games available. It also allows them to keep track of the chips they earn or lose while playing some of the betting games. This motivates the users to strive to reach different chip goals and pick up where they left off with their chip count.



The Game Selection provides a variety of games for the user to choose from. They range from simple 4 person games that are just for casual play and not betting, to games with 9 people that allow betting and more interaction with the other users. It is easy to navigate and with a couple clicks, you can see all the available tables for the different games.



Some games allow the users to communicate with the other users at the tables over a variety of media formats. First is a simple text chat. This allows users to type messages that are then displayed to all other users. The other two formats are combined into a Video/Voice chat system. This will use the users’ webcam and/or microphone and display/transmit that information so the users at the table can see and hear the other players.



The games provided in this system are set up for the users to have fun, interact with the other users, and test your skills against the other players. With all of these added components, we make it feel like you are sitting in the same room playing against the other users, instead of just playing against a computer.

* **Development Documents**

The following documents are called “Use Case Diagrams”. They give a brief description of the interactions between the users and the system during certain components of the program.

|  |  |
| --- | --- |
| UC: Join a Table | |
| Actor: User | System: Card Game System |
|  | 0. System displays the home screen. (Figure 1) |
| 1. TUCBW the user clicking Start Playing Now | 2. The system displays the page with a list of available games. (Figure 2) |
| 3. The user chooses a game they would like to play. | 4. The system displays a list of tables based on the chosen game. (Figure 3) |
| 5. The user clicks Join Table on the table they would like to play at. | 6. The system displays the game and the table to the user. (Figure 4) |
| 7. The user begins to play the game. |  |
|  |  |
| UC: Message Other Users | |
| Actor: User | System: Card Game System |
|  | 0. System displays a game/table. |
| 1. The user clicks on the chat text input field, types a message, and clicks send. (Figure 5) | 2. The system displays the message entered to everyone at the table. (Figure 4) |
| 3. The user reads the messages sent by other users. |  |

This next document is a rough example of the class structure for the program. It does not give much example on what methods or variables will be used, but it does tell what type of relations might be visible between the different aspects of the program.



* **Code Samples**

The following are sections of code taken directly from our project. They will outline some of the bigger portions of the program. After each segment of code, a description of what is happening or what is being done in that code will follow.

**private** **final** **int** MAX\_PLAYERS = 9;

**private** Deck deck;

**private** PokerBettingHelper bettingHelper;

**private** HoldemWinChecker winChecker;

**private** List<BettingPlayer> players;

**private** List<BettingPlayer> activePlayers;

**private** List<Integer> board;

**private** **int** chipLimit;

**private** **int** bigBlind;

/\*\*

\* Constructor for HoldemDealer

\* **@param** maxChips int holding the chip limit for the table

\* **@param** bb int holding the value of the big blind for the table

\*/

**public** HoldemDealer(**int** maxChips, **int** bb) {

deck = **new** Deck();

players = **new** ArrayList<>(MAX\_PLAYERS);

bettingHelper = **new** PokerBettingHelper(activePlayers, bigBlind);

winChecker = **new** HoldemWinChecker();

chipLimit = maxChips;

bigBlind = bb;

Random r = **new** Random();

Collections.*rotate*(players, r.nextInt(MAX\_PLAYERS));

}

This code shows the class variables and constructor for the HoldemDealer class. This class is used to start and keep track of a Texas Hold’em poker game. The constructor is initializing the variables needed while the game is on progress as well as calling some function to get the game started.

/\*\*

\* Function to handle starting a hand of Texas Hold'em

\*/

**public** **void** startHand() {

**if**(players.size() < 2) {

**throw** **new** IllegalArgumentException("At least 2 players are needed.");

}

Collections.*rotate*(players, 1);

activePlayers = **new** ArrayList<>(players);

board = **new** ArrayList<>();

deck.collectCards();

bettingHelper = **new** PokerBettingHelper(activePlayers, bigBlind);

**for**(BettingPlayer player : players) {

player.resetHand();

}

deck.shuffle();

System.*out*.printf("The dealer is Player %d\n",

players.get(players.size() - 1).getSeatNumber());

dealHands();

bettingHelper.startNewRound(**true**);

}

The startHand() function is used to initiate the start of a new poker hand. It will verify there is enough players to start they hand, collect and re-shuffle the cards from the hand before, reset the players hands, and deal the players their new hands.

/\*\*

\* Function to determine the Wining hand

\* **@param** activePlayers List of BettingPlayer that holds the players still in the round

\* **@param** inBoard List of Integer to hold cards on board

\*/

**public** **void** findWinningHand(List<BettingPlayer> activePlayers, List<Integer> inBoard) {

players = activePlayers;

board = inBoard;

hands = **new** ArrayList<>();

winningPlayers = **new** ArrayList<>();

**for**(BettingPlayer player : players) {

hands.addAll(player.getHand());

}

**int** i, j, k;

**boolean** handFound = **false**;

List<Integer> rankWinners = **new** ArrayList<>();

List<Integer> fiveCardHands;

dontCheck = **new** BitSet(ranks.size());

eliminateHands();

**for**(i = 0; i < 5; i++) {

thisHand.set(i, board.get(i));

}

**for**(i = 8; handFound == **false**; i--) {

**if**(!dontCheck.get(i)) {

**for**(j = 0; j < hands.size(); j += 2) {

thisHand.set(5, hands.get(j));

thisHand.set(6, hands.get(j + 1));

handValues = getValueList(thisHand);

Collections.*sort*(handValues);

**if**(rankCheck(i)) {

handFound = **true**;

rankWinners.add(thisHand.get(5));

rankWinners.add(thisHand.get(6));

}

}

}

}

winningRank = i + 1;

**if**(rankWinners.size() == 2) { //If there's only one hand with the winning rank

winningPlayers.add(BettingPlayer.*getPlayerByCard*(players, rankWinners.get(0)));

} **else** {

fiveCardHands = getFiveCardHands(rankWinners, winningRank);

BitSet possibleWinner = **new** BitSet(fiveCardHands.size() / 5);

possibleWinner.set(0, fiveCardHands.size() / 5);

**int** possibleCount = fiveCardHands.size() / 5;

**for**(i = 4; i >= 0; i--) {

**if**(possibleCount == 1) {

**break**;

}

**for**(j = 0; j < fiveCardHands.size(); j += 5) {

**if**(possibleWinner.get(j / 5)) {

**for**(k = 0; k < fiveCardHands.size(); k += 5) {

**if** (k != j) {

**if**(possibleWinner.get(k / 5) && fiveCardHands.get(j + i) < fiveCardHands.get(k + i)) {

possibleWinner.set(j / 5, **false**);

possibleCount--;

**break**;

}

}

}

}

}

}

**for**(i = 0; i < possibleWinner.size(); i++) {

**if**(possibleWinner.get(i)) {

winningPlayers.add(BettingPlayer.*getPlayerByCard*(players, rankWinners.get(i \* 2)));

}

}

}

}

The function above is the main work horse behind determining which player had the winning poker hand. It gathers the house cards, and all the players hands that were still left in the game. It then goes through and starts to eliminate the hands that are not possible with the available cards. After that it goes down the list of possible hands until it finds a player or players that have that hand to determine who had the winning hand.

/\*\*

\* Class to gather any messages from the server

\* **@author** Andrew Haegar

\*

\*/

**class** ServerListener **extends** Thread {

/\*\*

\* Function to run the server listener

\*/

**public** **void** run() {

**while** (**true**) {

**try** {

String message = input.readLine();

System.*out*.println(message);

} **catch** (IOException e) {

}

}

}

}

This code is a part of the ChatClient. It is a thread listener that listens for any messages sent from the server that it is connected to. When a client sends a message it is sent to the server. The server then uses a thread to loop through the connected clients and send the inbound message to all clients so they can see the message.

/\*\*

\* Send message received by any client to all attached clients

\* **@param** message

\*/

**private** **synchronized** **void** sendMessage(String message) {

System.*out*.println(message);

**for** (**int** i = (client.size() - 1); i >= 0; i--) {

ClientT thrd = client.get(i);

**if** (!thrd.writeMessage(message)) {

client.remove(i);

}

}

}

This code is the part of the ChatServer that sends the message to all the connected clients. Upon receiving a message, the server calls this function. The server the loops through all the know clients, attempting to send each client the message that was just received. If the server fails to send the message, it assumes the client is no longer connected and removed it from the list.

* **Struggles in development**

During the development of this whole system we ran into a couple things that almost slowed production down to a complete stop. The biggest obstacle that we ran into was figuring out a way to connect our main game engine, which is written in Java code, to our Graphical User Interface, which was written in HTML/Javascript/CSS. We did a great deal of research on the web, trying to find a service or a plugin that would allow us to connect these two, but we were not finding anything that documented well or worked in all situations. Then we found a service that allowed us to create RESTful requests that would transfer the data back and forth between the two so the who system would function properly. The second obstacle that we encountered was the integration of the Video/Voice/Text chat system. Getting everything to connect seamlessly between the different users and reducing the lag in the video feeds that were being seen by the other players at the table was puzzling issue, but was figured out with some research online.

* **Future Development**

With the initial aspect of this game working there are many things that we are looking to develop in the near future. Of course, we are always looking to add more card games to the system so users can always find a game they like. After users start using this game, we will re-evaluate the layout of the GUI and determine if anything needs to be reworked to make it easier on the users as they play the game and navigate between the different pages. Lastly, we could add more ways for the user to track their progress in different games. Tracking the hands won and lost. Biggest chip amount won in different games. What was your best hand in a particular game and what was their biggest chip loss.