

## Project 4 Documentation - Team 24

Piotr Ramza - [pramza2@uic.edu](mailto:pramza2@uic.edu)

Daniel Dean Asuncion - [dasunc2@uic.edu](mailto:dasunc2@uic.edu)

Samuel Bankole - [sbanko2@uic.edu](mailto:sbanko2@uic.edu)

Justin Trieu - [jtrieu5@uic.edu](mailto:jtrieu5@uic.edu)

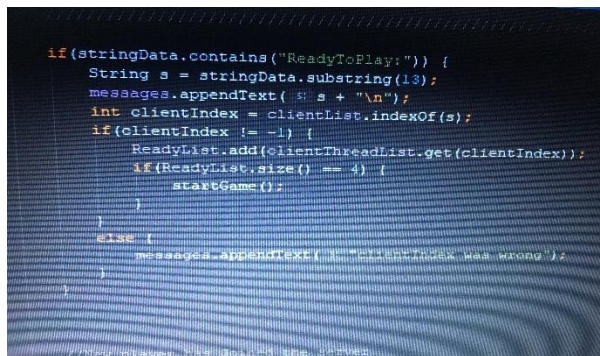
## Project Name and purpose

The group created a game of blackjack for play between four clients connecting to a server. The initial stages of the project were focused on starting the game with at least four people since that was a requirement beforehand. The game was set up using JavaFX.

As mentioned before the game would only start if there are four clients connected. There would be a dealer who would act as the server and would deal, shuffle and distribute winning pots. Each player would also have their own view of the game and all the options available to them to make influences on the game.

## GUI for people on the server.

To implement this functionality, we needed to handle exactly four players connecting to the server. In the class `ServerThread`, we stored each client connecting to the server into an `ArrayList` to make it easier to keep track of clients connecting to the server. Once the `ArrayList` size is equal to the required number of people to play the game, the game starts.

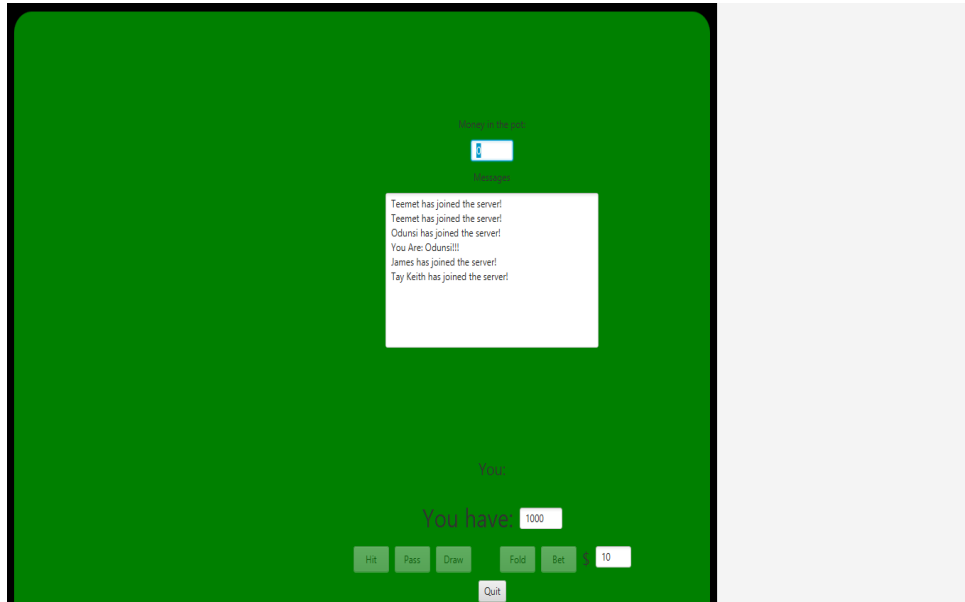


```
//when player hits connect the server
if(stringData.contains("ReadyToPlay:")) {
    String s = stringData.substring(13);
    messages.appendText( s + "\n");
    int clientIndex = clientList.indexOf(s);
    if(clientIndex != -1) {
        ReadyList.add(clientThreadList.get(clientIndex));
        if(ReadyList.size() == 4) {
            startGame();
        }
    }
    else {
        messages.appendText( s + "ClientID was wrong");
    }
}
```

In order to actually show the list of people connected to the server, we utilized an `ObservableList` and used `ListView` to show the list of players on the right side of the playScreen for the clients under `ClientFX.java`. The same implementation for `ServerFX.java` was applied so you can see who is connected to the server through the server GUI as well.

## Gameplay

Each player has a set amount that he starts the game with and once that amount finishes, the client is out of the game and is unable to play. Besides all that though, users can choose to bet or fold or quit. The user is also alerted on the moves the other users make, like when a client has just connected.

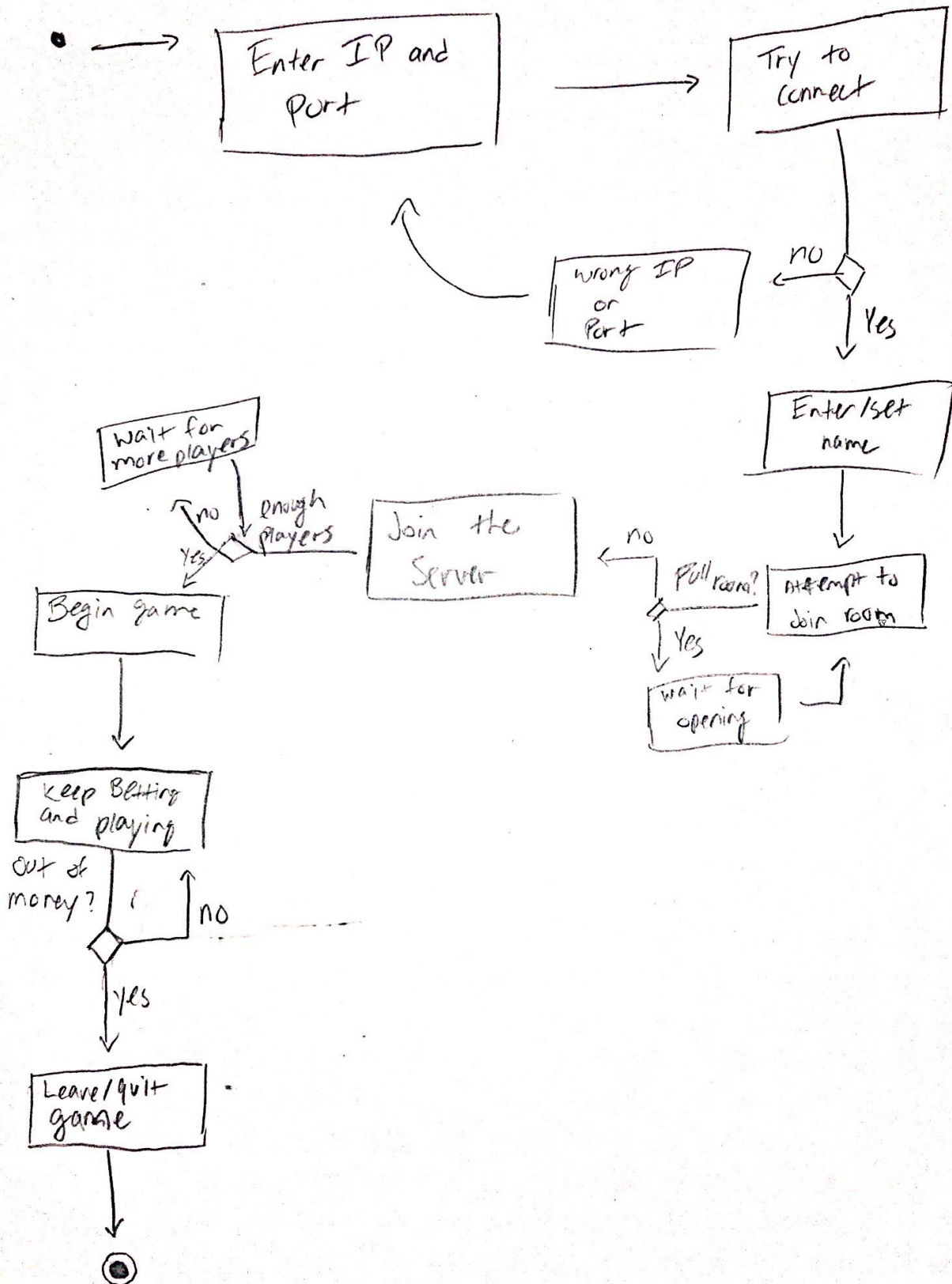


The whole code uses about five different class files containing different parts of the project we needed to come together like the card class, the deck which is basically for setting up the deck of cards, the player class to handle what each player does, and then finally, the client and server class.

The client class handles the GUI that each client interacts with, containing buttons initially to connect to the server and then have the client wait until the right number of users connect before starting the game. The in-game GUI also gives the user a lot of options as to what to do to affect the game.

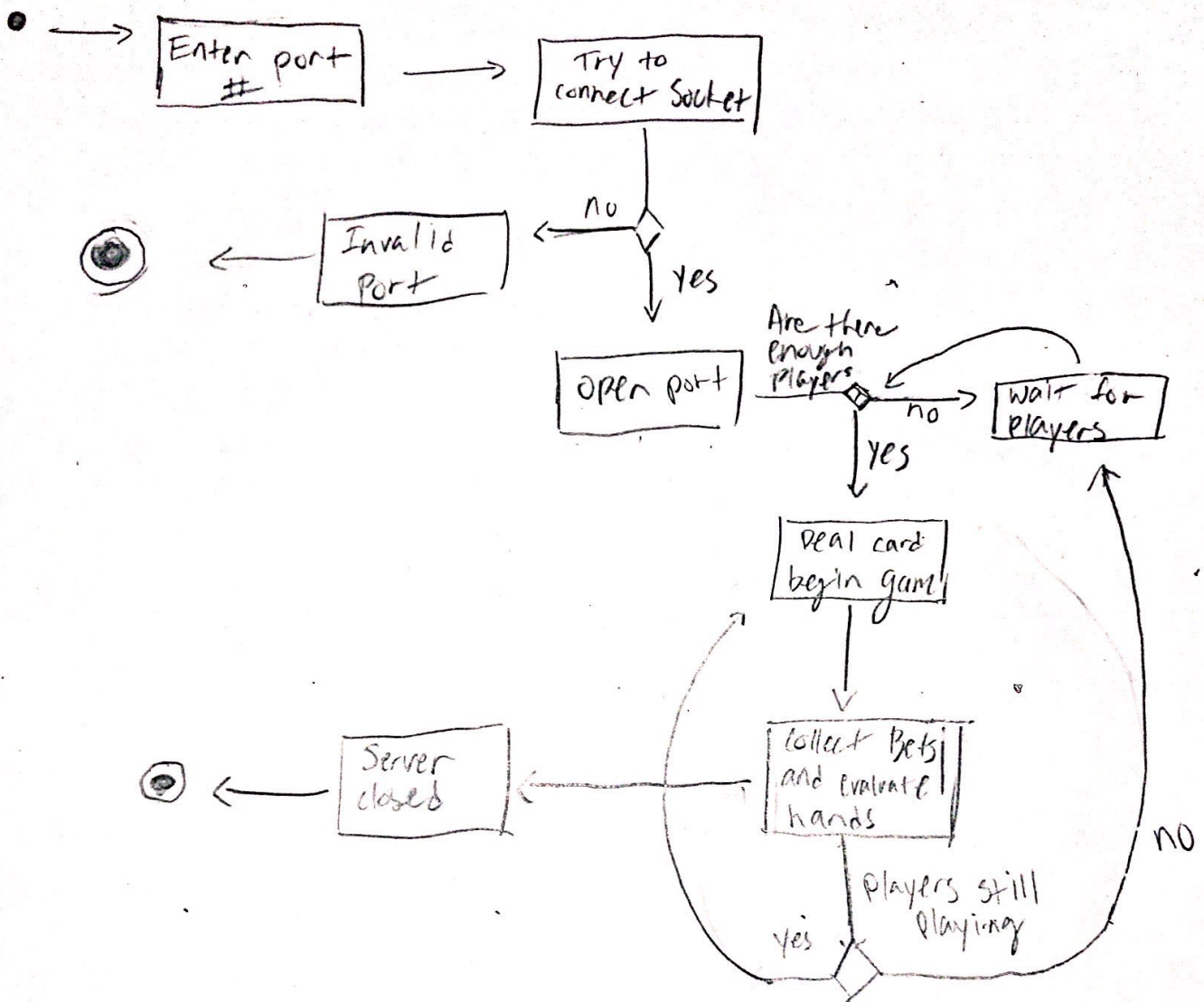
The UML activity diagram and UML class or component diagram are described in the next two [ages respectively:

# Client UML Activity Diagram





# Server UML Activity Diagram



Card

Card();  
getNumber();  
getScore();



Deck

Deck();  
shuffle();  
drawCard();  
reset();



Player

Player(string n);  
getHandValue();  
getReady();  
getCurrentBet();



Server

Server  
ServerThread  
to Client Thread  
Thread



Client

Client  
getIP();  
getPort();  
ConnThread

