****

**University Of Dhaka**

**Department of Computer Science and Engineering**

**CSE 3212 – Computer Networking Lab**

**Project Name: Implementation of a Multiplayer Call Bridge Card Game**

**Submitted By: Submitted to:**

**Shaikhum Monira(10) Md. Tawfiqul Islam**

**Salimullah Saqib(18) Mahmood Jasim**

**Jubayer Al Mahmud(26)**

**Tahsin Anwar(42)**

**Suddipta Deb Nath(52)**

**Abid Rahman(60)**

Table of Content

1. Introduction 1

2. Team Members 1

3. Organogram 2

4. Project Description 3

4.1 Game Design 3

4.2 User Interface (UI) Design 4

5. Implementation 4

6. Used Libraries 6

7. Screnshots 7

**1. Introduction:**

In this project, we will be implementing a multiplayer call bridge game where 4 players are able to play.

Call Bridge is a popular card game where 4 players play individually. 52 cards are distributed equally among the players for playing. At the beginning of each round, one calls the amount of points he thinks he would be able to achieve. If the calling person gets the exact point or one point more than the call, then the point called is added to his score. Otherwise, the point called is subtracted from his score.

As this is a network based game, there are 4 clients and 1 server. The clients will connect to the server before playing. The server will act as a dealer/score-keeper and the clients will act as players.

**2. Team Members:**

* Shaikhum Monira, Roll-10
* Salimullah Saqib, Roll-18
* Jubayer Al Mahmud, Roll-26
* M.Tahsin Anwar, Roll-42
* Suddipta Deb Nath, Roll-52Abid Rahman, Roll-60

**3. Organogram:**

***Team Leader***

Abid Rahman

***Front End Developer***

­

***Back End Developer***

***Documentation***

***Documentation***

Suddipta Deb Nath

**Game Logic**

**Game server and Player Interaction**

Shaikhum Monira

M Tahsin Anwar

Suddipta Deb Nath

Salimullah Saqib

Jubayer Al Mahmud

Shaikhum Monira

Jubayer Al Mahmud

Abid Rahman

M Tahsin Anwar

Figure 1: Organogram

**4. Project Description:**

**4.1. Game Design:**

* As the game is played among four players, in our project there are four clients and one server.
* At first, every client has to connect with the server. And the game will never start until all clients are connected.
* No client-to-client interaction is done here which means when a client plays a card, it does not inform the other clients directly about its card. It only contacts the server, and then the server sends this message to the other clients.
* Server is the main controller and it will take the decision about the result, shuffling and distributing cards among the clients i.e. players. Server will decide which player will move first and no player will be able to play a card until the server authorizes the player to do so.

Here is a use-case diagram showing various use-cases of our project.

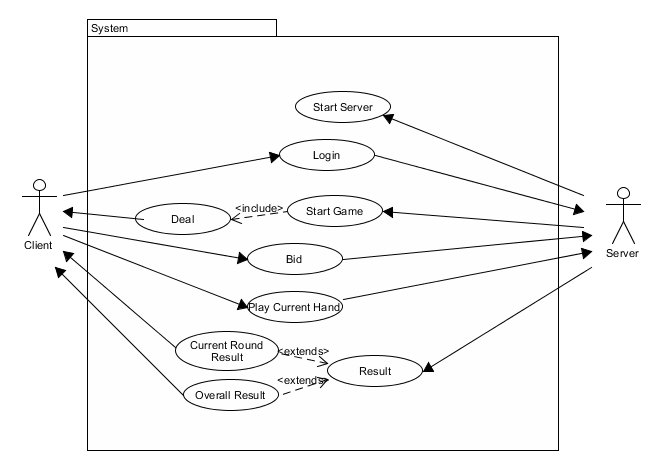


Figure 2: Use case diagram of

**4.2. User Interface (UI) Design:**

UI is implemented for both the server side and the client side. We used java GUI to design the UI.

**(i) Server side UI:**

* The server side UI is a very simple one. It has a “Start Server” button to start the server.
* After all the clients are connected, it can start the game with the "Start Game" button.

**(ii) Client side UI:**

* At first the client’s home screen comes up, here there are two option. The client can press the “Join Game” button to go to the login screen or it can see the name of the developers by pressing the “Credits” button.
* After pressing the “Join Game” button in the home screen, a login window appears for the clients to log in. Here the clients put their login credentials and log in to game. After logging in, the game window will open.
* Positions of four clients are shown on the four side of the window. Client’s own 13 cards are only visible to him in 13 small boxes above his name.
* What other players have played in a hand will be shown in three separate boxes in the game window.
* At the top, there is a message box showing different information like whose turn it is now.
* At the top-right corner, the number of hands each player have won alongside their calls are shown.
* At the top-left corner, the overall score of each player is shown.

Multiple screenshots of both the UIs are added near the end of the document.

**5. Implementation:**

**Server:** The server is the main controller of the game. This is implemented in *ServerWindow.java* file. Players need to first connect to the server to play the game. This is done by making the server listen for incoming connections from the players. We used multithreading in this part so that the server can listen and accept connections from multiple clients at the same time. When the server accepts a connection from a client, it saves its DataInputStream and DataOutputStream buffers in an array so that whenever the server needs to communicate to particular client, it can use those buffers to do so. After all the players are connected, the server’s job is to start the game as well.

**Client:** The client is implemented in *LoginWindow.java* file. At first clients i.e. players connect to the server using the *connectToServer()* method. They do so by requesting a TCP connection and when the server accepts that connection, the clients send their login credentials to the server and if they match with the data stored in the server, the clients are given access to the main game UI made for the clients.

**Client – Server Communication:** The server and the client mainly pass String messages to each other via their dedicated TCP connections to interact with one another. We handled it in this manner –

1. There are mainly two methods that are used for this - *sendToCliet()* method which is used by the server to send any string data to the client and *sendToServer()* method which is used by the client to send any String data to the server.
2. Whenever any of these 2 methods are called, they actually send two String variables named *command* and *value* to their desired recipient and whenever the recipient receives these strings, it takes actions according to data in those variables.
3. The *command* string specifies which sort of action to take and the *value* string specifies the necessary information needed to actually complete that action. For example, if “Player 1” was trying to log in to the game, at first it would send to the server (using the *sendToServer()* method) a *command* string that said “auth” which means the client i.e. player wants to log in and in the *value* string it would contain his login credentials. Then after checking with the stored login information in the server for every player, the server would reply to the client using the (using the *sendToCliet()* method) whether the information given by the player match with the record stored in the server. This time the server would send “auth” as the *command* string to notify the client that the next message which is the *value* string contains the reply from the server for his login request and the value string might contain “Accepted” if the request was successful and might contain “Ignored” if the request was unsuccessful.
4. All the other server-client interactions are handled in the same way. Just with different *command* strings and different *value* strings.

**Player UI:** The player UI is invoked by a client whenever it successfully logs in to the server. The player UI communicates with the server via the client. Whenever cards are shuffled and given to players, that information is received by the client first and passed on to the player UI and then the player UI calls necessary methods to display the cards onto the UI. And similarly when any move is made, this information is passed to the server from the player UI via the client and the server takes necessary actions to pass this information to other clients connected to the game.

**Game Logic:** The main logic of the game is implemented in the *GameLogic*.*java* file. Shuffling the cards, determining the winner of a hand after all the players have played their cards, keeping track of the score are all done here. To shuffle the cards or to determine the winner, the server invokes the *shuffleCards() or getWinner()* methods and sends this information to the client’s UI so that the result is visible to all the players currently playing.

**6. Used Libraries:**

The Java packages that were used to complete this project is given below:

**javax.swing\*:** We used this package for making the UI. We used components like *JFrame, JButton, JLabel, JOptionPane, ImageIcon etc* from this package.

**java.util\*:** This package contains the collections framework, legacy collection classes, event model, date and time facilities, and miscellaneous utility classes. We used *Random, TimeUnit, Arrays, HashMap, Map etc* from this package for various purposes.

**java.awt\*:** This package contains classes for creating user interfaces and for painting graphics and images and also event listeners to listen for an event. We used *ActionEvent, ActionListener* from this package to respond to button presses.

**java.net\*:** This package provides the classes for implementing network applications. It involves Socket, Server Socket classes and many more. We used *Socket, ServerSocket* from this package to create sockets for server and client.

**java.io\*:** This package contains the basics about input and output (I/O) streams. It also contains buffered streams. We used *DataInputStream, DataOutputStream* to receive/send data from one socket to another.

**7. Screenshots:**

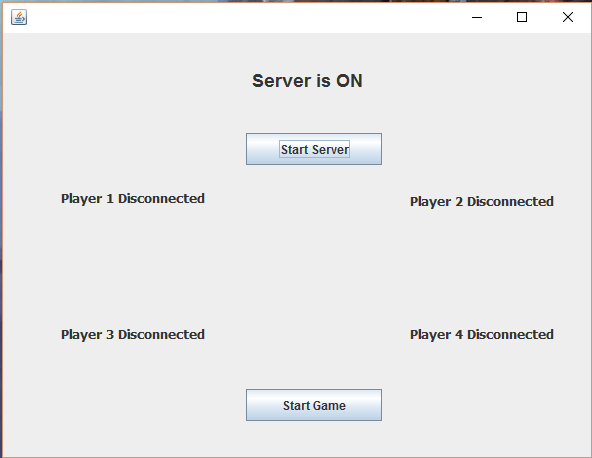
****

Figure 3: Server

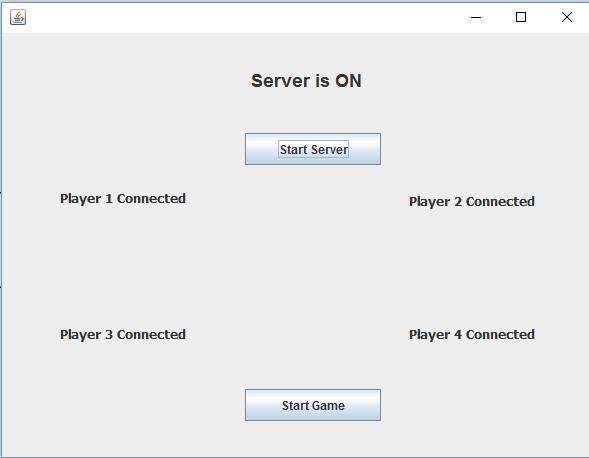
****

Figure 4: Server (after all the players are logged in)

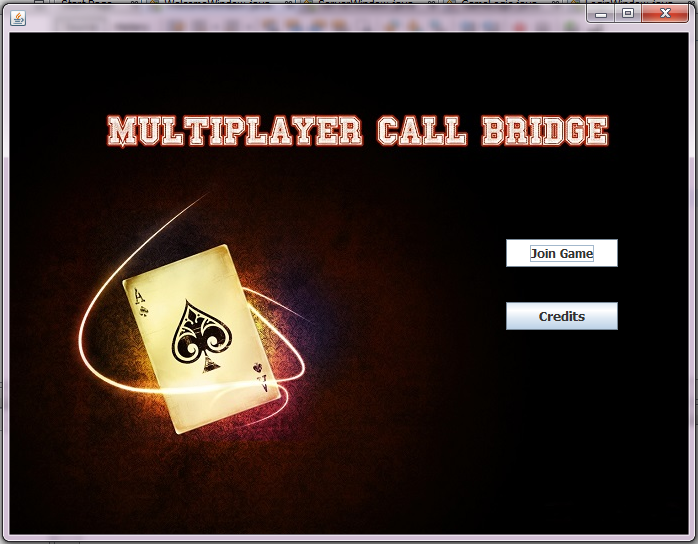
****

Figure 5: Welcome Screen

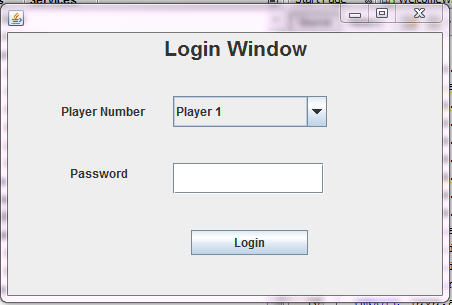
****

Figure 6: Login Screen(after joining game)

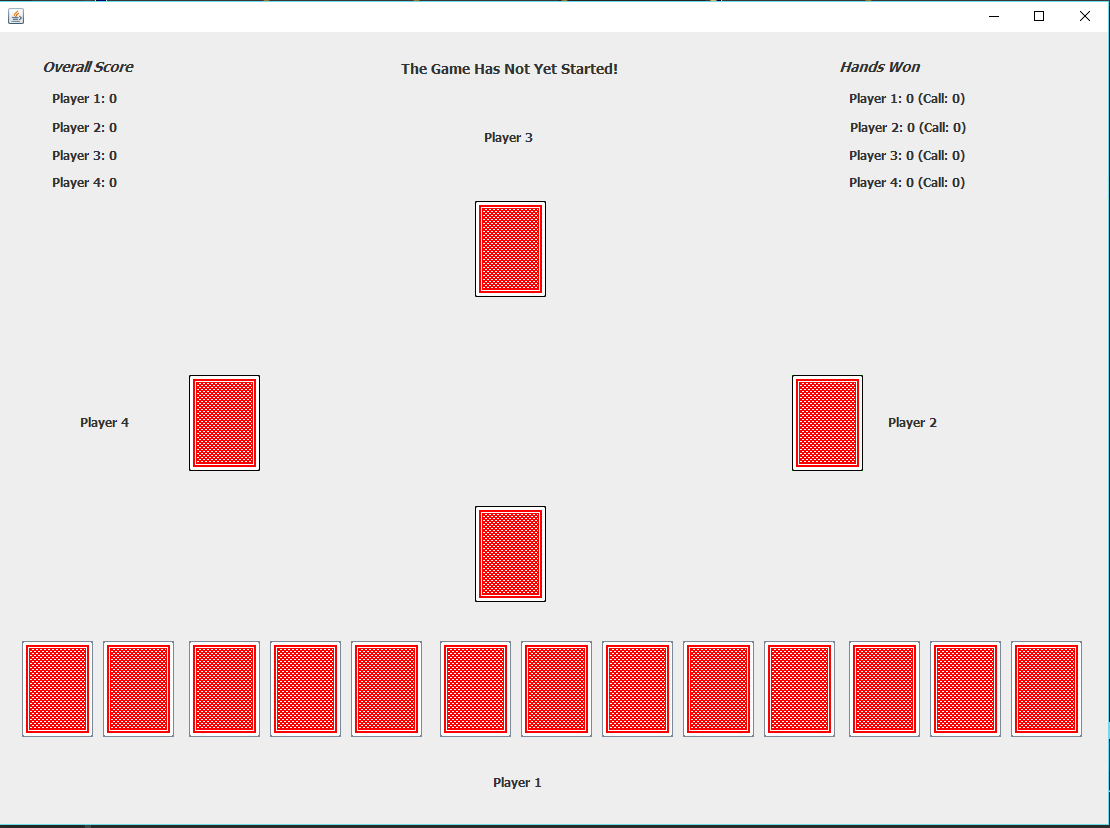
****

Figure 7:. Player UI (after logging in successfully)

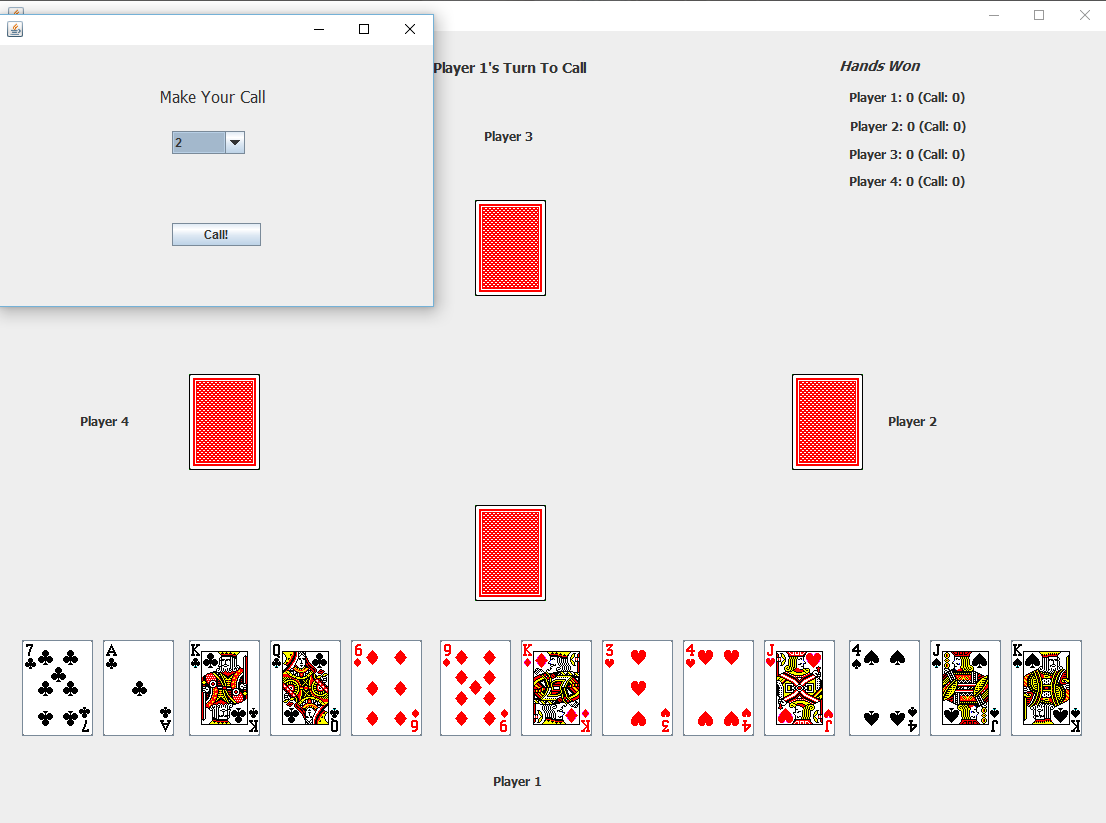


Figure 8: Player UI (after server has distributed the cards).

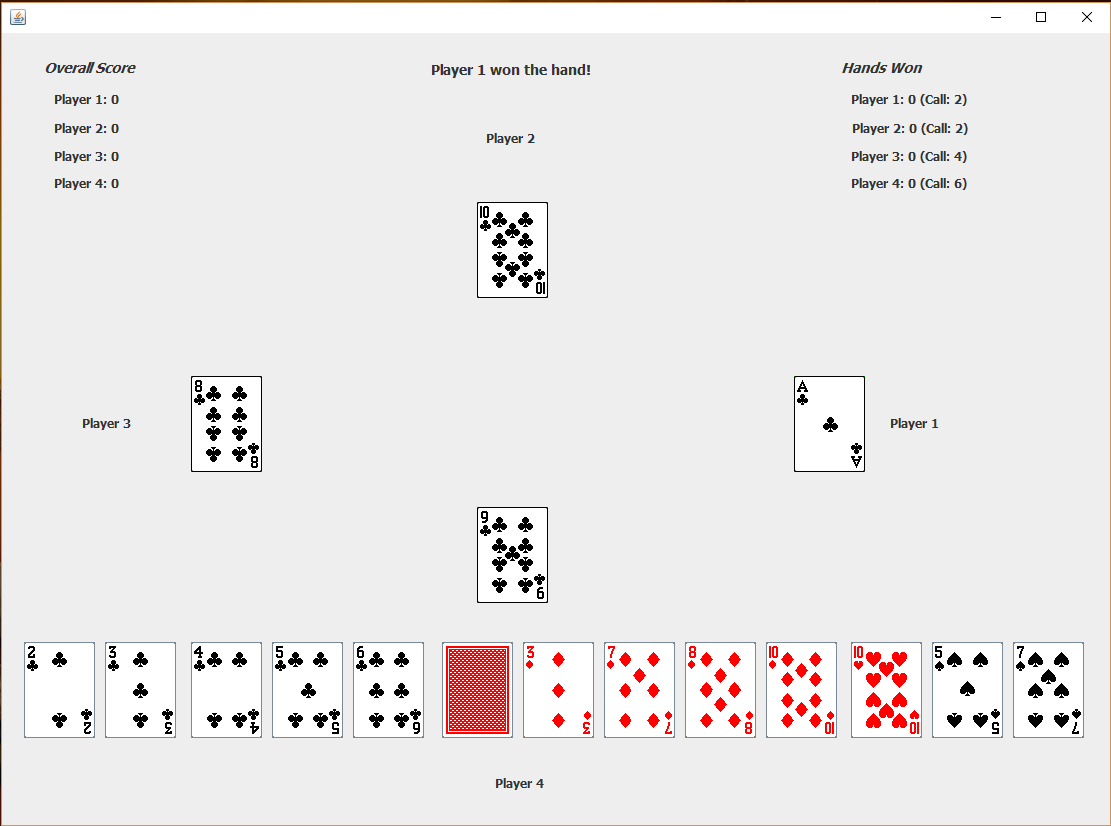
****

Figure 9: Player UI (Winning a hand)

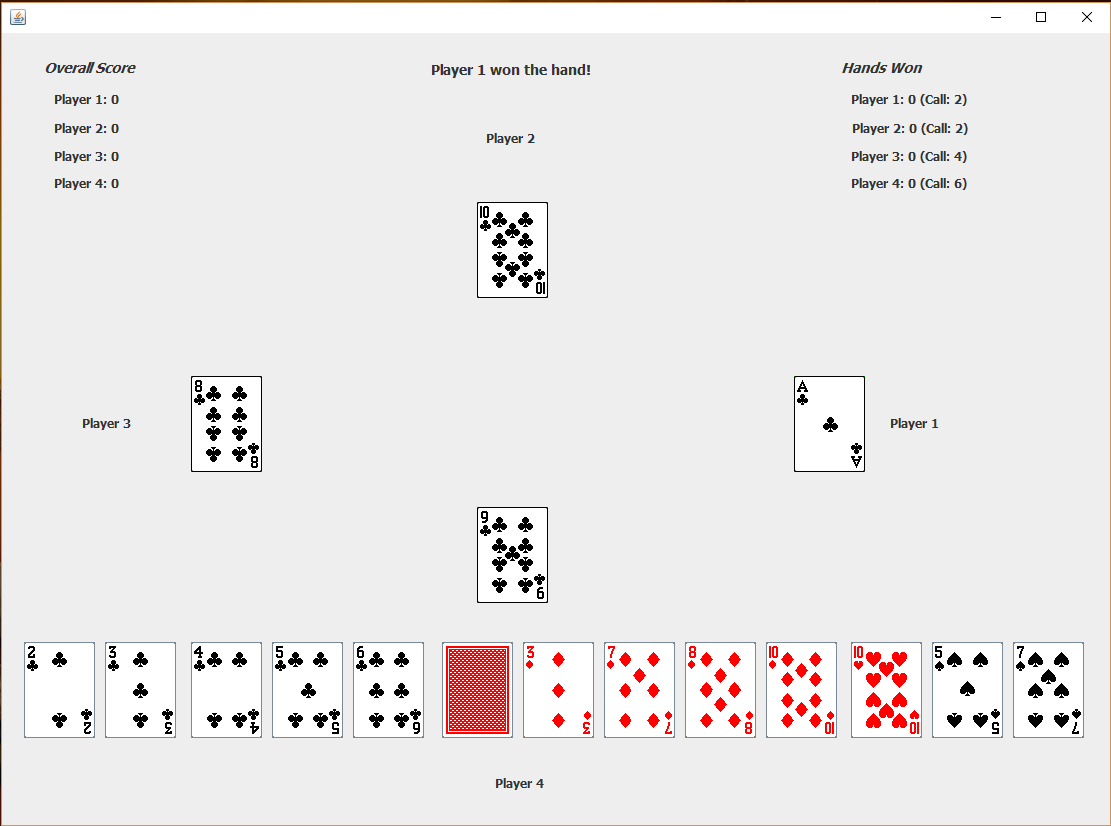
****

Figure 10: Player UI (After a player has won a hand)