Network HW1 - Bingo Game

- Joohyung Kim
- 2014-11618

Problem statement

In this project, we should implement a Bingo Game, which is simple game application that enables exchange of messages among clients. We are implementing both server and client sides

Specifications

- Each client has a unique ID
- There's 1 bingo game room on the server
- And for simplicity, assume there are only five clients
    - 1 Main culprit, 1 Copartner, 3 Pseudo players
    - Pseudo players are implemented on the server
- When a client logged in by typing client's ID, the server transmits a game room status
- When server bootstrap, creates a game room
- Each client enters a game room created by the server
- When the number of clients in the game room becomes five, the server starts the game
- Bingo matrix for each client is 5x5, and the number range is 1~99
- The server assign turn for each player with random order, but fixed after turn is first assigned to clients
- Each client takes turns choosing a number
    - The server selects a random number on a pseudo player's turn

- When one row of bingo is completed, the client wins the game and the serveer notifies the winner to all players
- There's conspirator group with two clients one is Main culprit, one Copartner
    - The culprit of the conspirator group chooses a number on his/her turn and sends the number he/she want to the other member, copartner
    - If the copartner of the conspirator group has a number received from the culprit member on his/her turn, he/she selects the corresponding number
    - Otherwise, the copartner of the conspirator group selects a number he/she wants

Environment

- Language: Java, SDK 11.0.1

Getting Started

Bootstrap Server

```
$ <unzip bingo game file>
$ cd <project directory>
$ ./gradlew run --args='server'
```

> NOTE: Server use port 8888

Bootstrap Client

```
$ <unzip bingo game file>
$ cd <project directory>
$ ./gradlew run --args='client'
```

Implementation

Server

BingoServer.java

```java
public class BingoServer {
    private GameManager gameManager;

    private Map<String, ObjectOutputStream> outputStreamPool;

    public BingoServer(GameManager gameManager) {
        this.gameManager = gameManager;
        this.outputStreamPool = new HashMap<>();
    }

    public void run() {
        try {
            ServerSocket server = new ServerSocket( port: 8888);

            while (true) {
                System.out.println("Waiting for the client request");

                Socket socket = server.accept();

                ObjectOutputStream oos = new ObjectOutputStream(socket.getOutputStream());
                ObjectInputStream ois = new ObjectInputStream(socket.getInputStream());

                System.out.println("Accepted connection request");

                ServerHandler handler = new ServerHandler(
                        socket,
                        oos,
                        ois,
                        outputStreamPool,
                        gameManager
                );
                handler.start();

                System.out.println("Running client handler...");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

This is the entrypoint of server. After creating `ServerSocket`, it enters infinite loop and waiting client connection on `server.accept()`.

When client connects to server, it creates `ServerHandler` and `ServerHandler` runs on its own thread. At this point `ServerHandler` shares `outputStreamPool` and `gameManager` with other threads.

ServerHandler.java

```java
public class ServerHandler extends Thread {
    private String id;

    private boolean init = false;

    private Socket socket;
    private ObjectOutputStream oos;
    private ObjectInputStream ois;

    private Map<String, ObjectOutputStream> outputStreamPool;

    private GameManager gameManager;

    public ServerHandler(Socket socket,
                         ObjectOutputStream oos,
                         ObjectInputStream ois,
                         Map<String, ObjectOutputStream> outputStreamPool,
                         GameManager gameManager) {...}

    public void run() {
        try {
            while (true) {
                if (gameManager.isGameStarted()) {
                    String playerId = gameManager.getCurrentTurnPlayerId();

                    System.out.println(String.format("Current turn user id is [%s]", playerId));

                    if (gameManager.isPseudoPlayer(playerId)) {
                        boolean gameIsDone = pseudoPlayerAction(playerId);
                        if (gameIsDone) {
                            break;
                        } else {
                            continue;
                        }
                    }
                }

                Message msg = (Message) ois.readObject();
                System.out.println("Received message from client: " + msg);

                oos.writeObject(handle(msg));

                if (gameManager.readyToStart() && !gameManager.isGameStarted()) {
                    setupGame();
                }
            }

        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        } finally {
            synchronized (outputStreamPool) {
                outputStreamPool.remove(id);
            }
```

This is the entrypoint of ServerHandler. At `run()` method all important business logic of ServerHandler is contained.

First in line 43, handler checks whether game is started, if not continue to listen client's request, otherwise check whether current turn is Pseudo player.

If current turn is on Pseudo player, do `pseudoPlayerAction()`.

```
161     private boolean pseudoPlayerAction(String playerId) {
162         boolean gameIsOver = false;
163
164         int selected = gameManager.doPseudoPlayerAction(playerId);
165
166         String winnerCandidate = gameManager.isSomeoneBingo();
167         if (winnerCandidate != null) {
168             broadcastWinnerMessage(winnerCandidate);
169             gameIsOver = true;
170         }
171
172         broadcastUpdatedMatrixMessage(selected);
173
174         gameManager.increaseCurrentTurn();
175
176         return gameIsOver;
177     }
```

In `pseudoPlayerAction()` pseudo choose random number and broadcast its picked number to other clients. At the point of Pseudo player choosing number, if bingo is made, then broadcast winner to other clients and returns whether game is over of not.

```
190     private void broadcastUpdatedMatrixMessage(int selected) {
191         gameManager.getAllPlayers()
192                 .forEach(player -> {
193                     if (gameManager.isPseudoPlayer(player.getId())) {
194                         return;
195                     }
196                     try {
197                         send(player.getId(), Message.ofMatrixUpdated(player.getId(), selected));
198                     } catch (IOException e) {
199                         System.out.println("Exception occurred while broadcastUpdatedMatrixMessag
200                     }
201                 });
202     }
203
204     private void broadcastWinnerMessage(String winner) {
205         gameManager.getAllPlayers()
206                 .forEach(player -> {
207                     if (gameManager.isPseudoPlayer(player.getId())) {
208                         return;
209                     }
210                     try {
211                         send(player.getId(), Message.ofWinner(player.getId(), winner));
212                     } catch (IOException e) {
213                         System.out.println("Exception occurred while broadcastWinnerMessage() in
214                     }
215                 });
216     }
217
218 }
219
```

And this is detail of `broadcastUpdateMatrixMessage()`, and `broadcastWinnerMessage()` functions. Both function works in a similar way. Iterate over all players and if that player is not Pseudo, then send the message.

```
101 @    private Message handle(Message message) throws IOException {
102          switch (message.getMethod()) {
103              case Method.JOIN:
104                  return handleJoin(message);
105              case Method.CHOOSE:
106                  return handleChoose(message);
107
108          }
109          return null;
110      }
111
112 @    private Message handleJoin(Message message) {
113          synchronized (outputStreamPool) {
114              outputStreamPool.put(message.getId(), oos);
115          }
116
117          this.id = message.getId();
118          this.init = true;
119
120          // Bingo player join the room, and assigned role
121          synchronized (gameManager) {
122              BingoPlayer bingoPlayer = gameManager.join(this.id);
123              System.out.println(String.format("Bingo player ID [%s] join the room", this.id));
124
125              return Message.ofJoinResult(
126                      this.id,
127                      bingoPlayer.getType().name(),
128                      bingoPlayer.getMatrix()
129              );
130          }
131      }
132
133 @    private Message handleChoose(Message message) throws IOException {
134          gameManager.chooseNumber(message.getId(), message.getNumber());
135
136          String winnerCandidate = gameManager.isSomeoneBingo();
137          if (winnerCandidate != null) {
138              broadcastWinnerMessage(winnerCandidate);
139              // game done!
140              System.out.println("Bingo game is finished");
141          }
142
143          broadcastUpdatedMatrixMessage(message.getNumber());
144
145          if (gameManager.isCulpritPlayer(message.getId())) {
146              gameManager.askInSecret(message.getId(), message.getSecret());
147              // send secret message to copartner
148              String copartnerId = gameManager.getCopartner().getId();
149              send(copartnerId, Message.ofSecret(copartnerId, message.getSecret()));
150          }
151          gameManager.increaseCurrentTurn();
152
153          return Message.ofChooseResult(
154                  this.id,
155                  message.getNumber()
156          );
157      }
```

This is how server handle message from client. In `handle()`, based no the message method, the way to deal with the message is changed. Basically client send `JOIN`, `CHOOSE` message. `JOIN` to join the bingo game, `CHOOSE` to choose bingo number.

In `handleJoin()` function, add OutputStreamPool with key of client id and respond with `Message.ofJoinResult` .

In `handleChoose()` fufnction, after choose number check whether someone made bingo. After that if client is Culprit accept secret number to send back to copartner.

Client

BingoApplication.java

```java
    @Override
    public void start(Stage stage) {
        GridPane gridPane = new GridPane();
        gridPane.setPadding(new Insets( top: 10,  right: 10,  bottom: 10,  left: 10));
        gridPane.setVgap(5);
        gridPane.setHgap(5);

        // Define the name text field
        final TextField idTextField = new TextField();
        idTextField.setPromptText("Enter your id");
        idTextField.setPrefColumnCount(10);
        idTextField.getText();

        GridPane.setConstraints(idTextField,  columnIndex: 0,  rowIndex: 0);
        gridPane.getChildren().add(idTextField);

        // Define the number text field
        final TextField numberTextField = new TextField();
        numberTextField.setPromptText("Enter your number");

        GridPane.setConstraints(numberTextField,  columnIndex: 0 ,  rowIndex: 1);
        gridPane.getChildren().add(numberTextField);

        // Define the secret number text field
        final TextField secretTextField = new TextField();
        secretTextField.setPromptText("Enter your secret to send to copartner");

        GridPane.setConstraints(secretTextField,  columnIndex: 1,  rowIndex: 1);
        gridPane.getChildren().add(secretTextField);

        // Define the login button
        Button loginBtn = new Button( text: "Join Room");
        GridPane.setConstraints(loginBtn,  columnIndex: 1,  rowIndex: 0);
        gridPane.getChildren().add(loginBtn);

        // Define the submit button
        Button submitBtn = new Button( text: "Submit");
        GridPane.setConstraints(submitBtn,  columnIndex: 2,  rowIndex: 1);
        gridPane.getChildren().add(submitBtn);

        // Define list view
        ListView<String> messageList = new ListView<>();
        ObservableList<String> items = FXCollections.observableArrayList();
        messageList.setItems(items);
        messageList.setPrefWidth(200);
        messageList.setPrefHeight(100);

        GridPane.setConstraints(messageList,  columnIndex: 0,  rowIndex: 2);
        gridPane.getChildren().add(messageList);

        // Adding labels
        final Label statusLabel = addLabel(gridPane,  labelText: "[System Status]",  col: 0,  row: 3,
        final Label accountLabel = addLabel(gridPane,  labelText: "[Your ID]",  col: 0,  row: 5,  span
        final Label roleLabel = addLabel(gridPane,  labelText: "[Your Role]",  col: 0,  row: 7,  span:
        final Label matrixLabel = addLabel(gridPane,  labelText: "[Your matrix]",  col: 0,  row: 9,
        final Label gameStartLabel = addLabel(gridPane,  labelText: "[Game Started?]",  col: 0,  row:
        final Label turnLabel = addLabel(gridPane,  labelText: "[Your Turn is]",  col: 0,  row: 13,
```

This is UI part of bingo application made by JavaFx framework.

```
110         playerInfoObserver.setRoleLabel(roleLabel);
111         playerInfoObserver.setGameStartLabel(gameStartLabel);
112         playerInfoObserver.setMatrixLabel(matrixLabel);
113         playerInfoObserver.setTurnLabel(turnLabel);
114         playerInfoObserver.setSecretLabel(secretLabel);
115         playerInfoObserver.setWinnerLabel(winnerLabel);
116
117         // Setting an action for the login button
118         loginBtn.setOnAction(event -> {
119             String id = idTextField.getText();
120             if (id != null && !id.isEmpty()) {
121                 try {
122                     bingoClient.send(
123                             Message.ofJoinRequest(id, Method.JOIN) );
124                 } catch (IOException e) {
125                     e.printStackTrace();
126                 }
127                 statusLabel.setText(String.format("Login id is %s", id));
128                 accountLabel.setText(id);
129                 items.add(id);
130                 idTextField.clear();
131             }
132         });
133
134         // Setting an action for the submit button
135         submitBtn.setOnAction(event -> {
136             if (accountLabel.getText().isEmpty()) {
137                 statusLabel.setText("You first need to login to bingo room");
138                 return;
139             }
140
141             if (numberTextField.getText().isEmpty()) {
142                 statusLabel.setText("You need to fill in your number");
143                 return;
144             }
145             Integer number = Integer.parseInt(numberTextField.getText());
146
147             Integer secret = null;
148             if (!secretTextField.getText().isEmpty()) {
149                 secret = Integer.parseInt(secretTextField.getText());
150             }
151
152             try {
153                 bingoClient.send(
154                         Message.ofChooseRequest(accountLabel.getText(), number, secret) );
155             } catch (IOException e) {
156                 e.printStackTrace();
157             }
158             statusLabel.setText(String.format("Submit number is %d", number));
159             items.add(number.toString());
160             numberTextField.clear();
161             secretTextField.clear();
162         });
163
```

From line110-115, there's `PlayerInfoObserver` which register JavaFx object. `PlayerInfoObserver` is for sending callback whenever `PlayerInfo` changes.

`PlayerInfo` is saving current client status such as role, bingo matrix, winner etc.

PlayerInfoObserver.java

```java
@Setter
public class PlayerInfoObserver implements PropertyChangeListener {
    private Label roleLabel;
    private Label gameStartLabel;
    private Label matrixLabel;
    private Label turnLabel;
    private Label secretLabel;
    private Label winnerLabel;

    @Override
    public void propertyChange(PropertyChangeEvent evt) {
        switch (evt.getPropertyName()) {
            case "role":
                Platform.runLater(() -> roleLabel.setText(evt.getNewValue().toString()));
                break;
            case "bingoMatrix":
                Platform.runLater(() -> matrixLabel.setText(evt.getNewValue().toString()));
                break;
            case "gameStarted":
                Platform.runLater(() -> gameStartLabel.setText(evt.getNewValue().toString()));
                break;
            case "turn":
                Platform.runLater(() -> turnLabel.setText(((Integer) evt.getNewValue()).toString(
                break;
            case "secret":
                Platform.runLater(() -> secretLabel.setText(((Integer) evt.getNewValue()).toStrir
                break;
            case "winner":
                Platform.runLater(() -> winnerLabel.setText((String) evt.getNewValue()));
            default:
                return;
        }
    }
}

```

First, `PlayerInfoObserver` implements `PropertyChangeListner` whenever Observable object' field change `propertyChange(PropertyChangeEvent)` function called. And based on the event property name, client update UI.

PlayerInfo.java

```java
@Data
public class PlayerInfo {

    private PropertyChangeSupport propertyChangeSupport;

    private String id;

    private String role;

    private Integer number;

    private Integer secret;

    private BingoMatrix bingoMatrix;

    private int selected;

    private boolean gameStarted = false;

    private Integer turn;

    private String winner;

    public PlayerInfo() { propertyChangeSupport = new PropertyChangeSupport( sourceBean: this); }

    public void addPropertyChangeListener(PropertyChangeListener pcl) {
        propertyChangeSupport.addPropertyChangeListener(pcl);
    }

    public void removePropertyChangeListener(PropertyChangeListener pcl) {
        propertyChangeSupport.removePropertyChangeListener(pcl);
    }

    public PlayerInfo id(String id) {
        this.id = id;
        return this;
    }

    public PlayerInfo role(String role) {
        propertyChangeSupport.firePropertyChange( propertyName: "role", this.role, role);
        this.role = role;
        return this;
    }
```

This is `PlayerInfo` it has `PropertyChangeSupport` which helps fire event whenever its field updated.

For example, in line 50

```
propertyChangeSupport.firePropertyChange("role", this.role,
role);
```

This fires `PlayerInfoObserver`'s `propertyChange` method.

BingoClient.java

```java
public class BingoClient {
    private Socket socket;

    private ObjectOutputStream oos;

    private ObjectInputStream ois;

    private DataHandler handler;

    public BingoClient(PlayerInfo playerInfo) throws IOException {
        socket = new Socket( host: "localhost", port: 8888);
        ois = new ObjectInputStream(socket.getInputStream());
        oos = new ObjectOutputStream(socket.getOutputStream());

        this.handler = new DataHandler(socket, ois, playerInfo);
        this.handler.start();

        System.out.println("DataHandler running...");
    }

    public void send(Message message) throws IOException {
        System.out.println("Sending request to Socket Server");

        oos.writeObject(message);
    }

    public void close() {
        try {
            ois.close();
            oos.close();
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

This is socket client entry point. In `BingoClient` constructor `DataHandler` thread start. With `BingoClient`'s `send()` function we can send the message to server.

DataHandler.java

```java
public class DataHandler extends Thread {
    private Socket socket;

    private ObjectInputStream ois;

    private PlayerInfo playerInfo;

    public DataHandler(Socket socket, ObjectInputStream ois, PlayerInfo playerInfo) {
        this.socket = socket;
        this.ois = ois;
        this.playerInfo = playerInfo;
    }

    public void run() {
        try {
            while (true) {
                Message message = (Message) ois.readObject();

                System.out.println("Received message from server:" + message);

                handle(message);

                System.out.println(String.format("Handled message ! [%s]", message.getMethod()));
            }
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        } finally {
            if (ois != null) {
                try {
                    ois.close();
                } catch (IOException e) {
                    System.out.println("ObjectInputStream closed");
                }
            }

            if (socket != null) {
                try {
                    socket.close();
                } catch (IOException e) {
                    System.out.println("Socket closed");
                }
            }
        }
    }
```

Client run its own thread called `DataHandler` to listen message from server. In `run()` method whenever object passed from `ObjectInputStream` , handle that message using `handle(message)` method.

```
55  @      private void handle(Message message) {
56             switch (message.getMethod()) {
57                 case Method.JOIN_RESP:
58                     synchronized (playerInfo) {
59                         playerInfo = playerInfo
60                                 .id(message.getId())
61                                 .role(message.getRole())
62                                 .bingoMatrix(message.getBingoMatrix());
63                     }
64                     break;
65                 case Method.GAME_START:
66                     synchronized (playerInfo) {
67                         playerInfo = playerInfo
68                                 .gameStarted(true)
69                                 .turn(message.getTurn());
70                     }
71                     break;
72                 case Method.MATRIX_UPDATED:
73                     synchronized (playerInfo) {
74                         System.out.println("Selected at this turn: " + message.getSelected());
75                         playerInfo = playerInfo
76                                 .selected(message.getSelected());
77
78                     }
79                     break;
80                 case Method.CHOOSE_RESP:
81                     System.out.println("Successfully choose bingo number:" + message.getNumber());
82                     break;
83                 case Method.SECRET:
84                     synchronized (playerInfo) {
85                         playerInfo = playerInfo
86                                 .secret(message.getSecret());
87                     }
88                     break;
89                 case Method.WINNER:
90                     synchronized (playerInfo) {
91                         playerInfo = playerInfo
92                                 .winner(message.getWinner());
93                     }
94                     break;
95                 default:
96                     throw new IllegalArgumentException(
97                             String.format("unexpected method, got [%s]", message.getMethod()) );
98             }
99         }
100    }
```

In `handle()` method, based on message method, it change how it behave. But the mechanism is similar. Whenever it receive message, it updates `PlayerInfo` status.

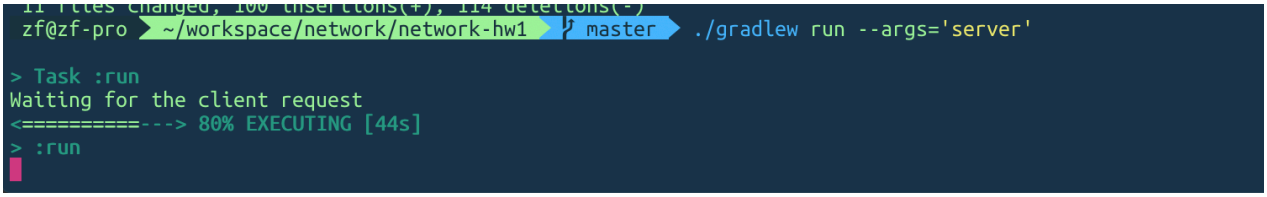
And with the help of `PlayerInfoObserver` it can automatically update UI based on updated `PlayerInfo` status.

Results

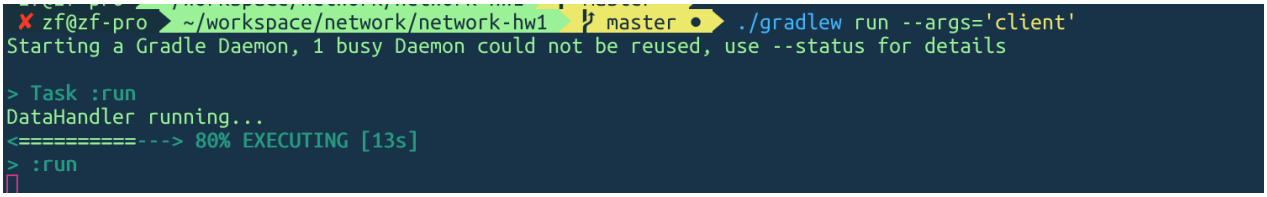
Start server and client

Based on the instruction of 'Getting Started', we can bootstrap server and client.

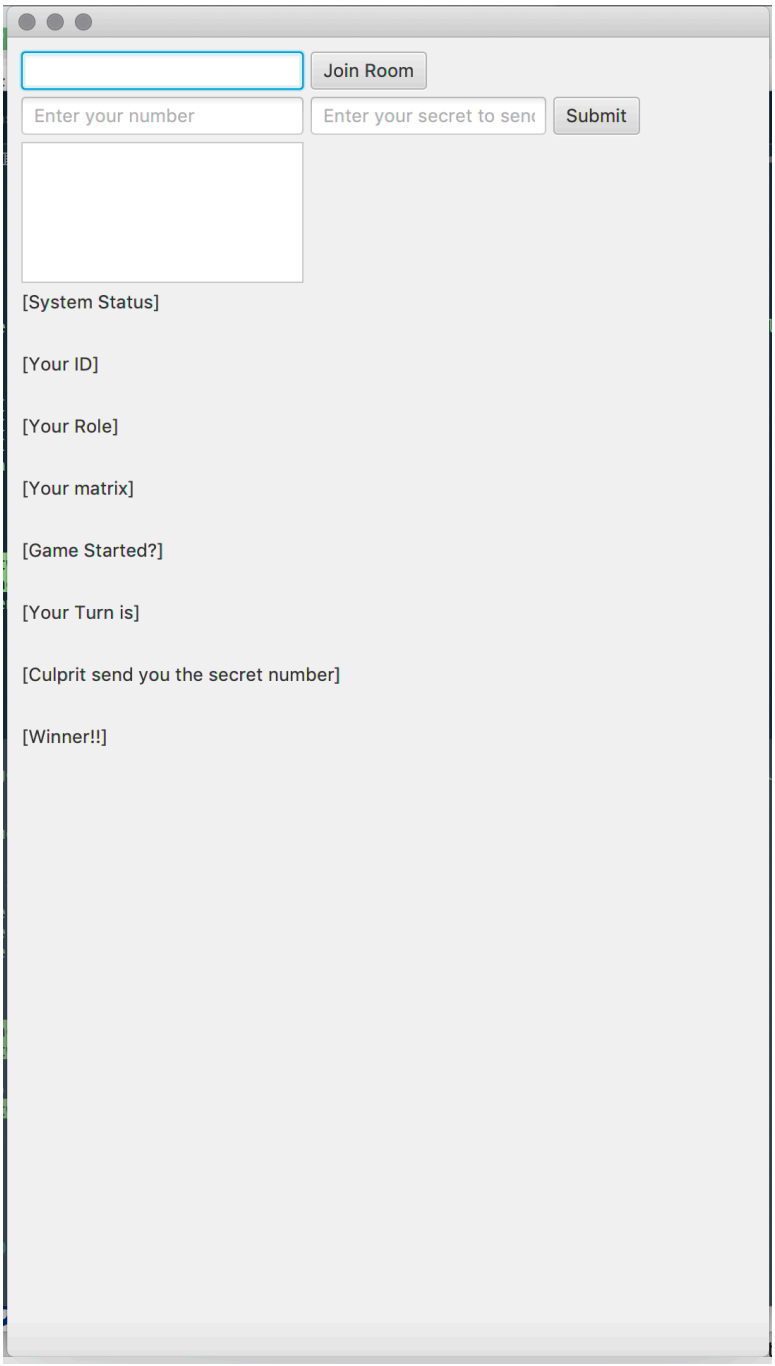
Bootstrap server



Bootstrap client

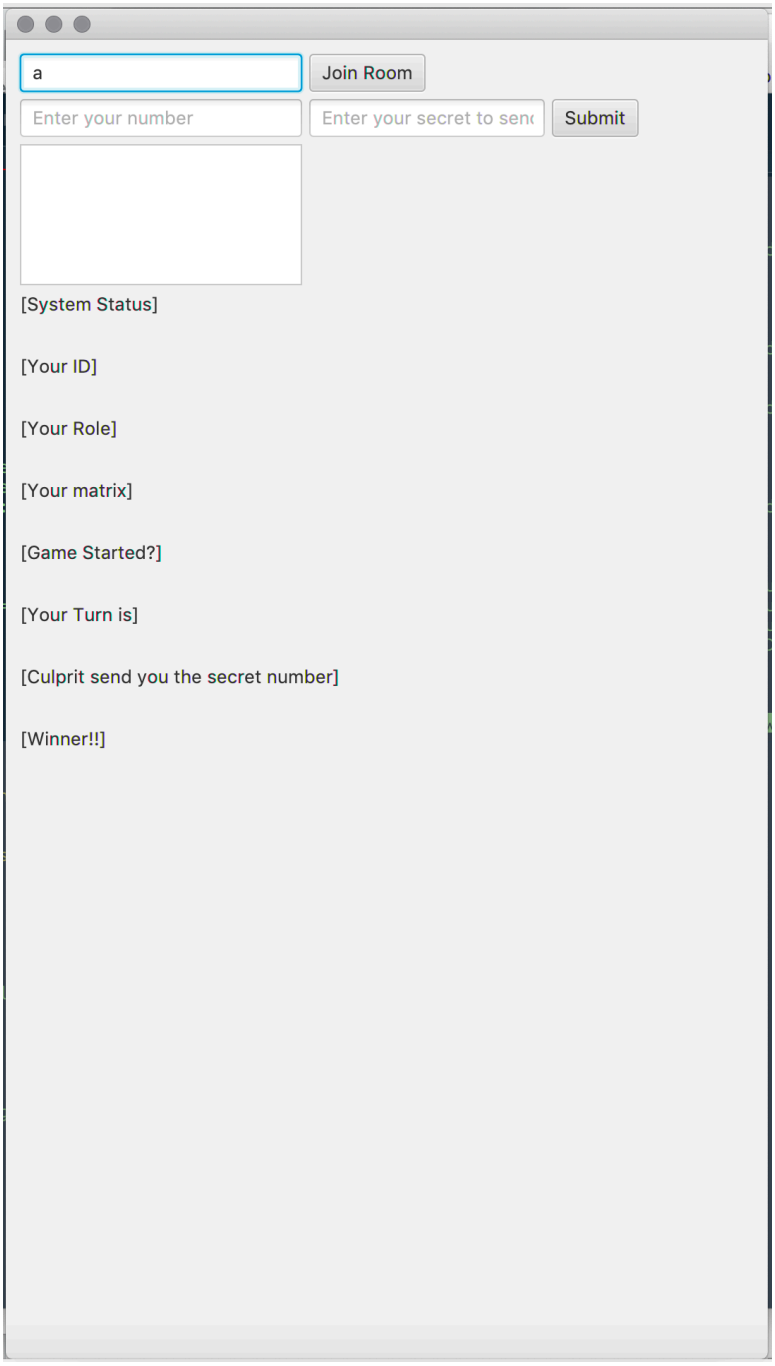


Client UI

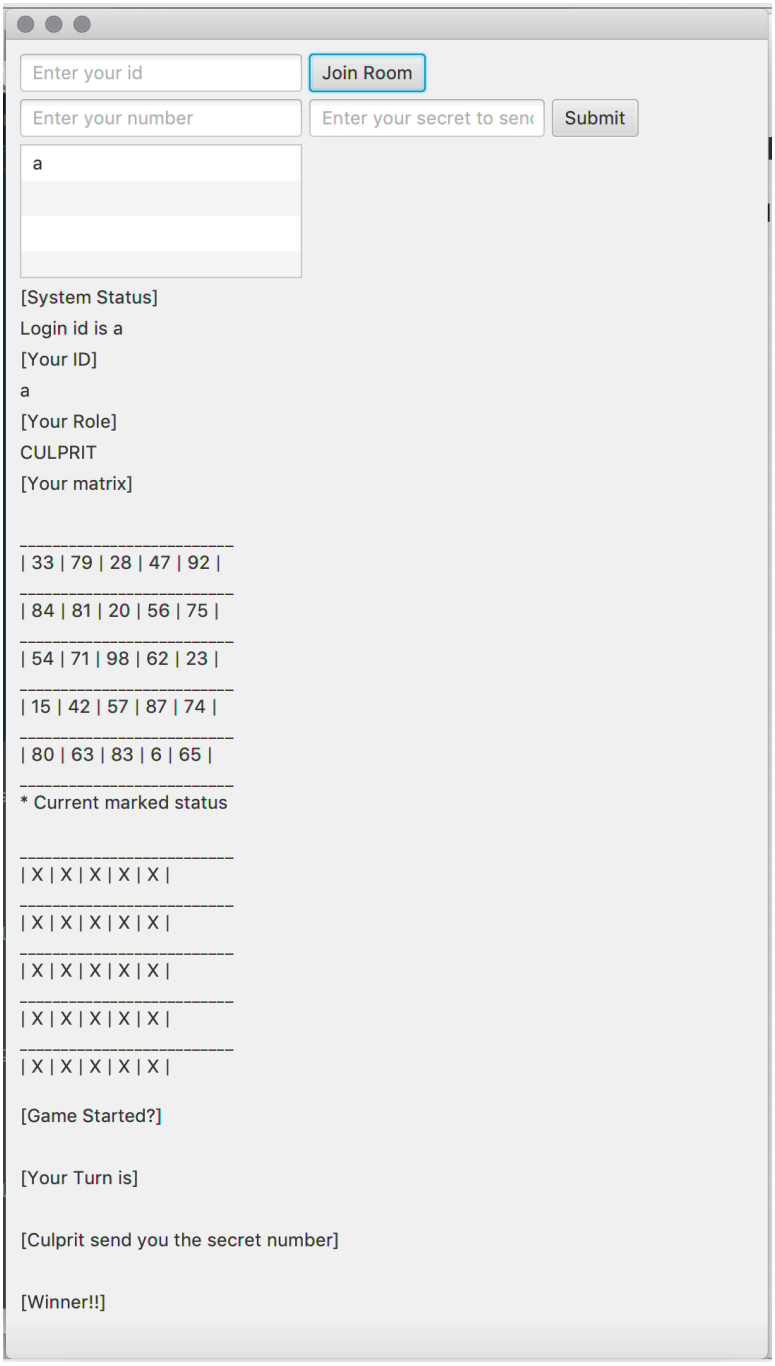


Client join the room

Type your client id on the field. Then click 'Join Room' button. And client 'a' join the game



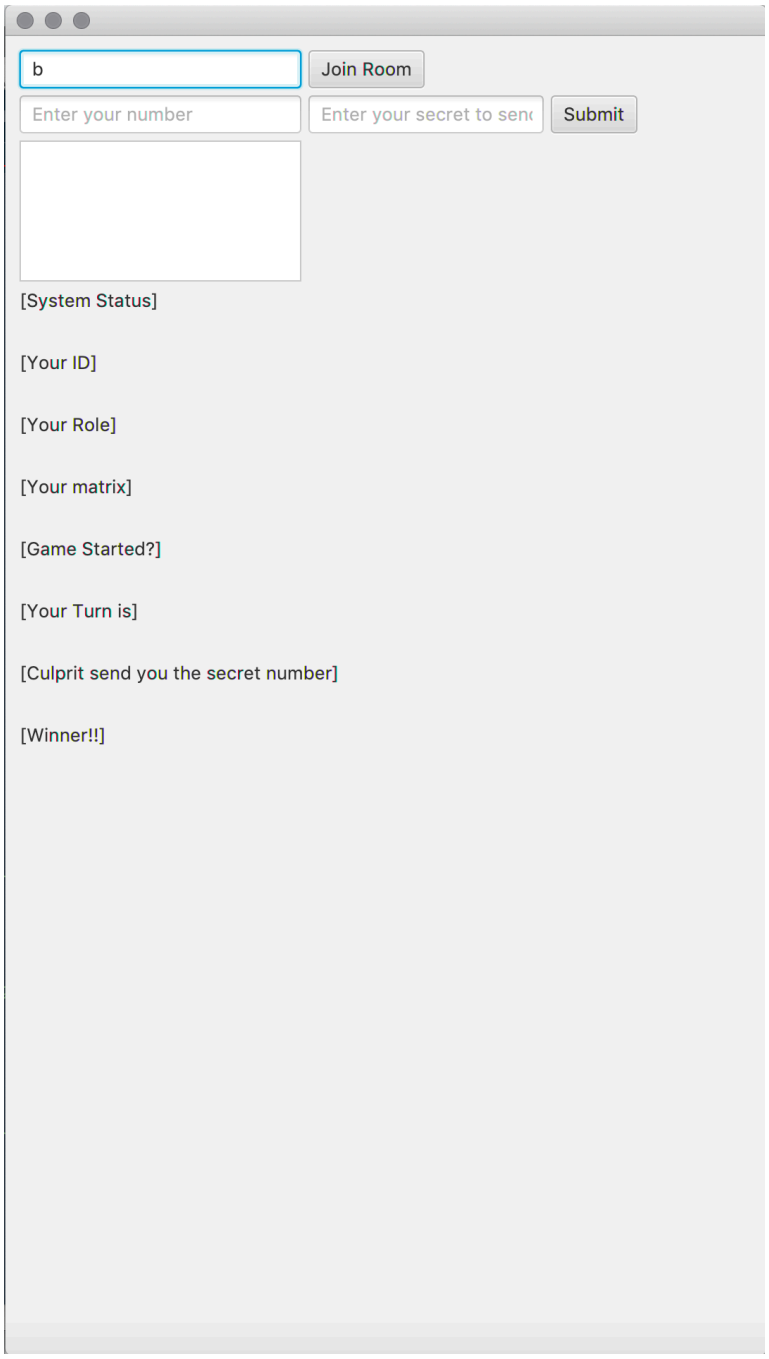
Then you can see your own bingo matrix with marker and other important player status such as role, whether game has started and so on.



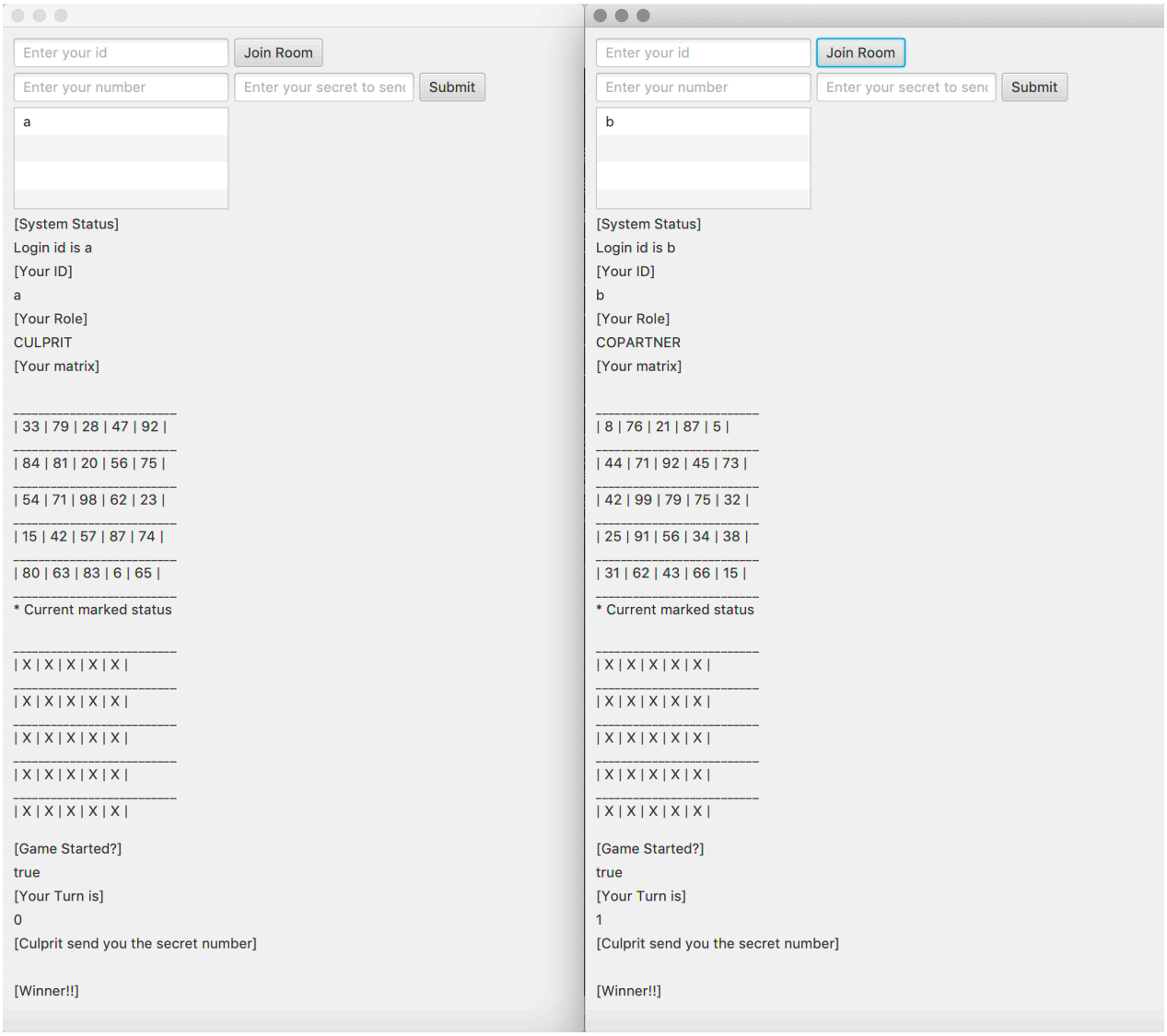

```
Waiting for the client request
Accepted connection request
Running client handler...
Waiting for the client request
Received message from client: Message(id=a, method=join, number=null, secret=null, bingoMatrix=null, role=
Bingo player ID [a] join the room
Room size is 4
<==========---> 80% EXECUTING [3m 21s]
> :run
```

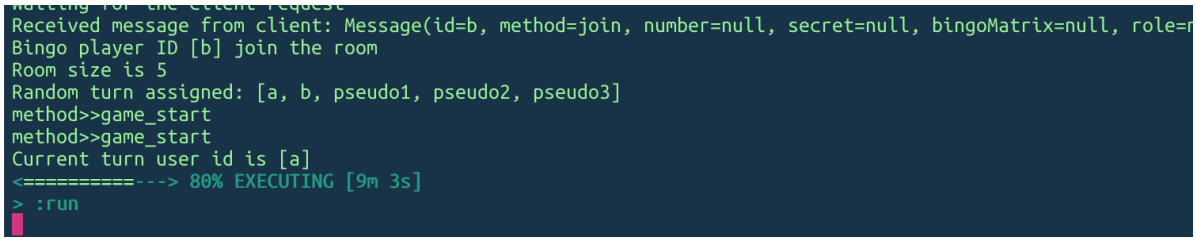
> This is server log, which show current players join the game is 4 so game is not started.

Another client with id 'b' join the game



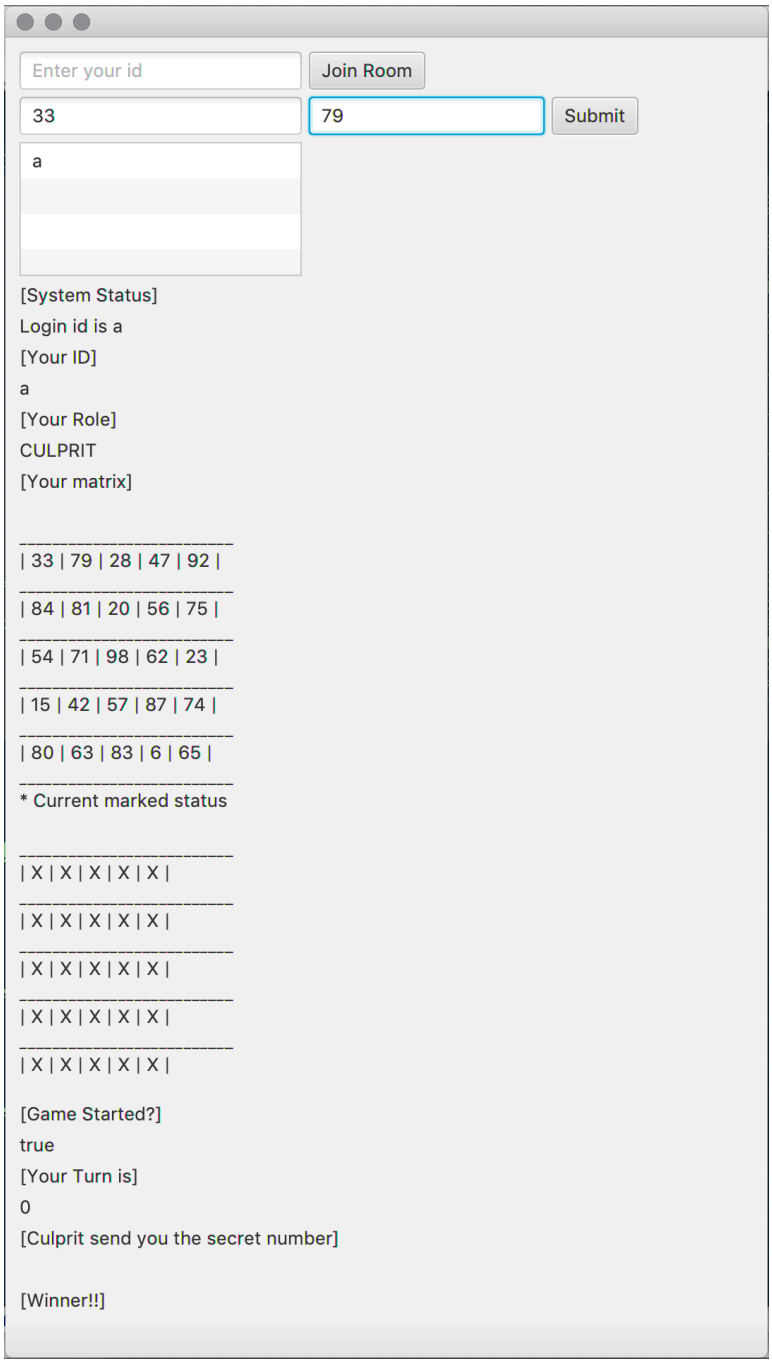
Then '[Game Started?]' label show true and show when is my turn on '[Your turn is]' label. Because the number of player is 5, turn number range is from 0 to 4.



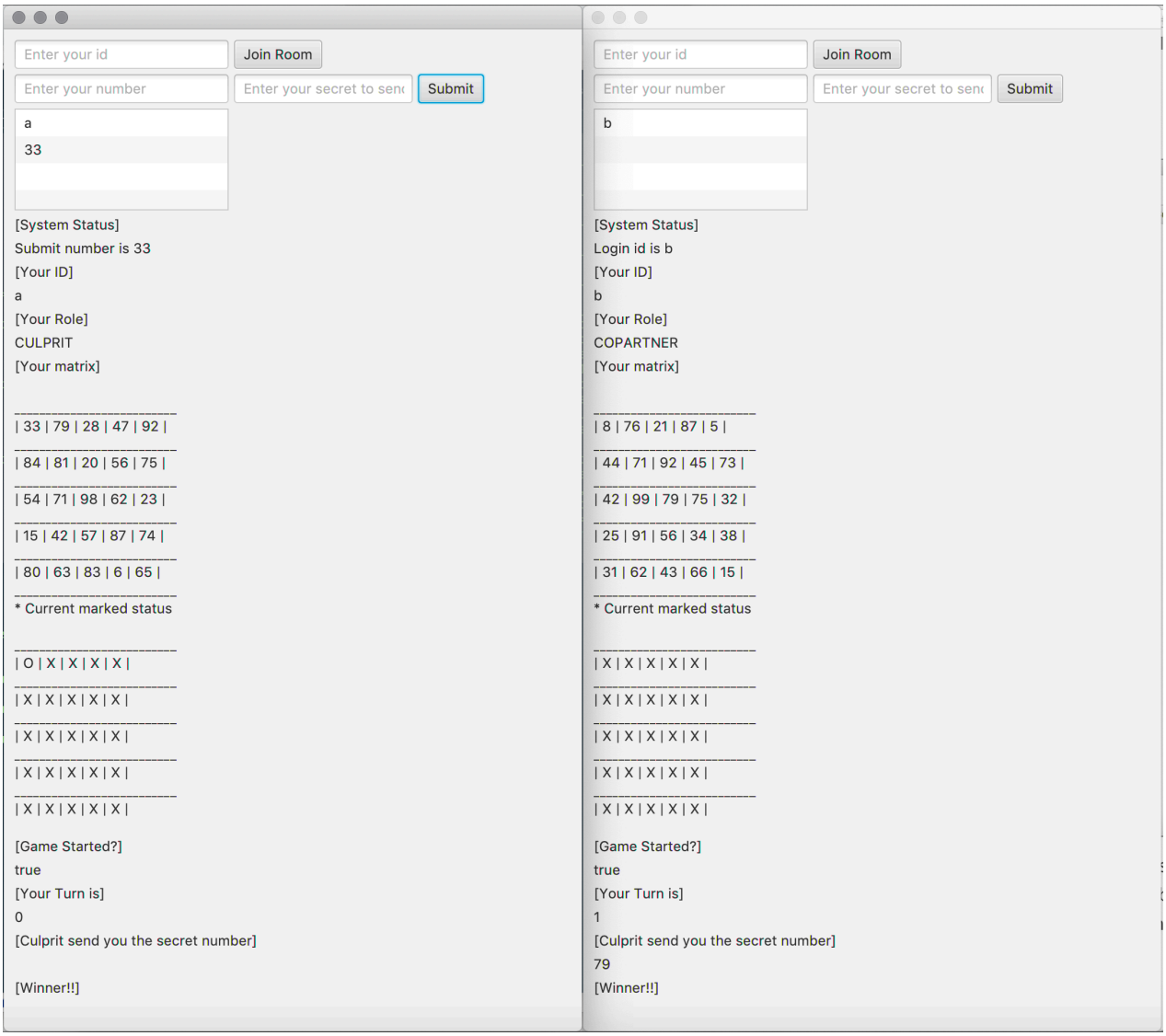


> Server logs shows that room size is 5 and turn is randomly assigned when game is start. In this demo the turn is a, b, pseudo1, pseudo2, pseudo3.

Choose number!

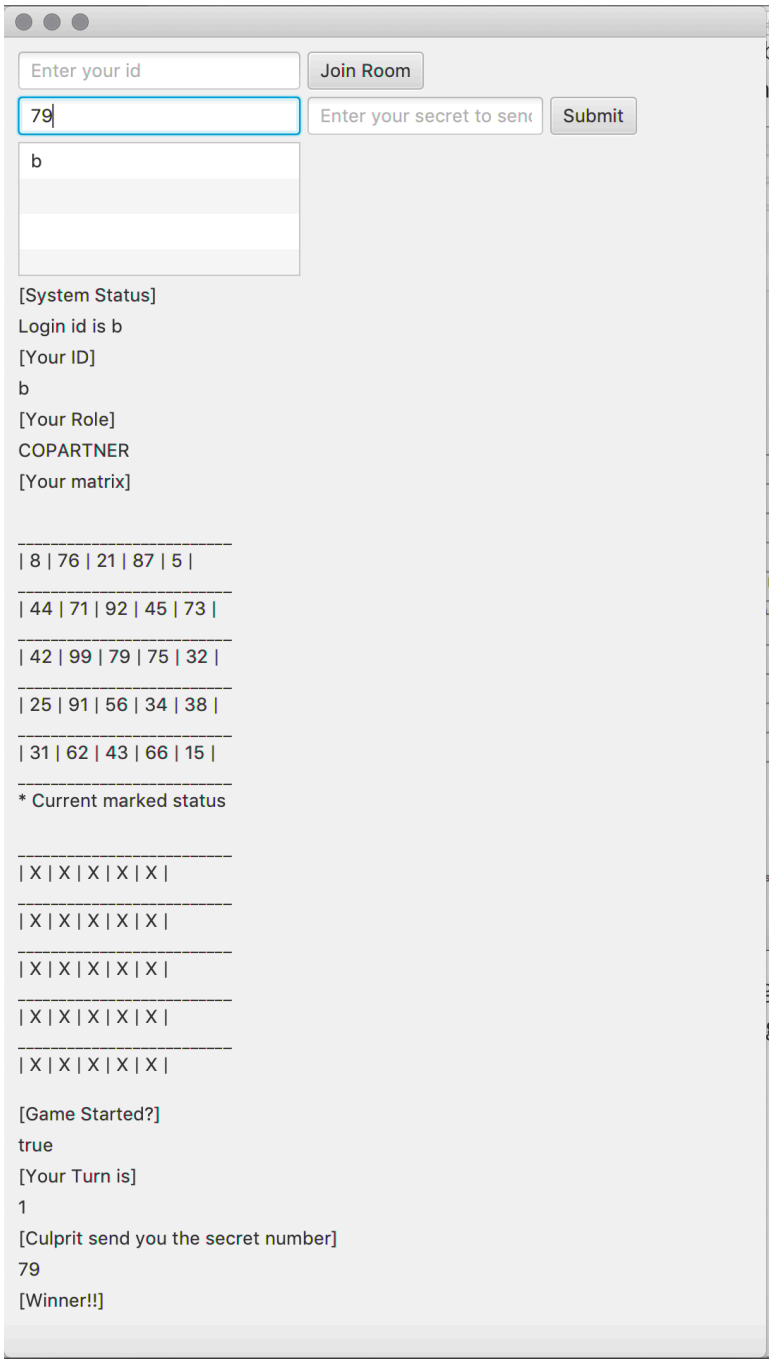


Now client a's turn. And 'a' is Culprit so 'a' must send secret ask number with its own choosing number. In the screenshot above number 33 is choosing number on the left panel and 79 is secret ask number on the right panel.

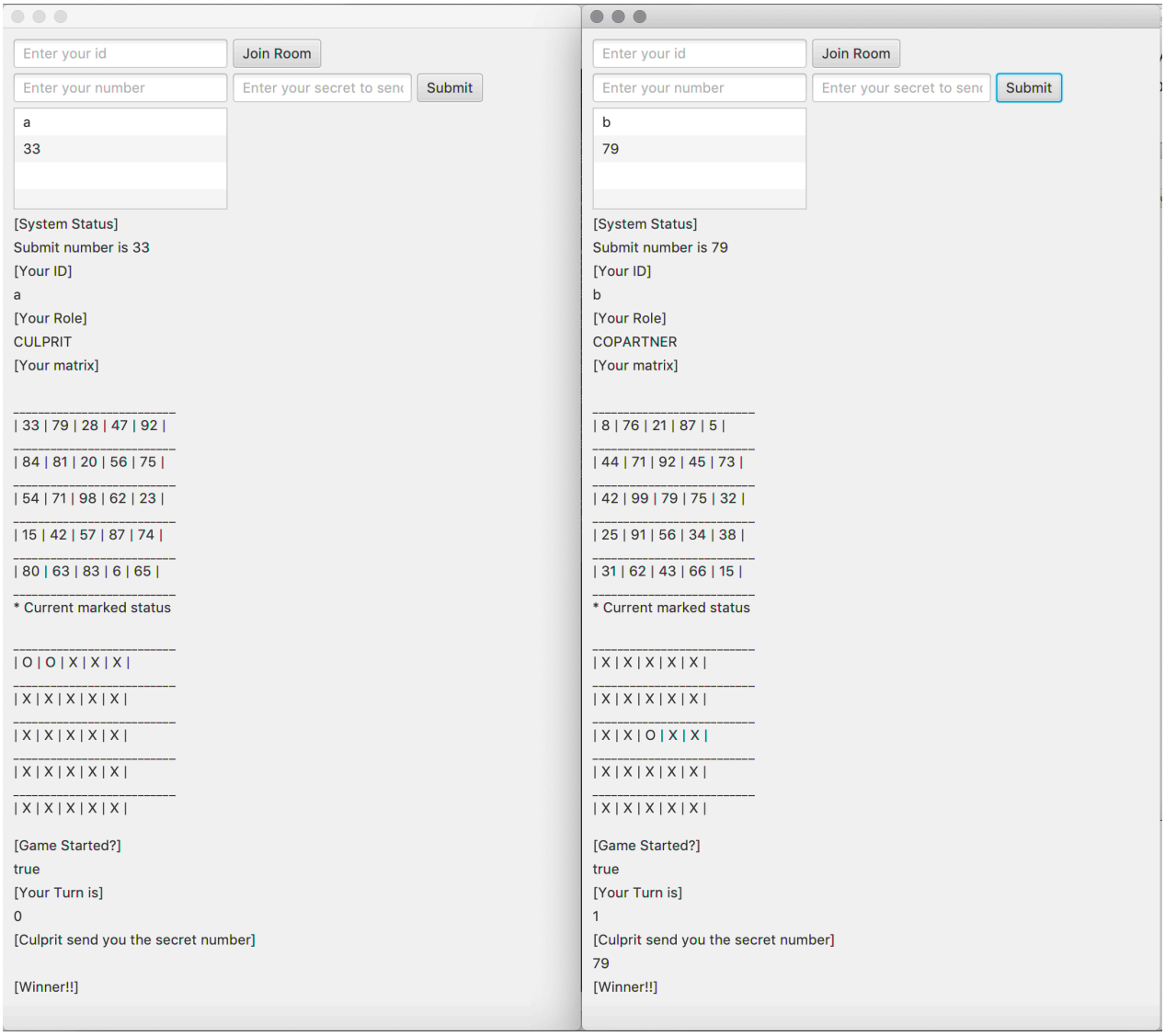


After 'a' submit numbers, it automatically update bingo markers. And you can '[Culprit send you the secret number]' panel on the right number 79. This number is culprit's secret ask number.

Now the client b's turn. Because b has 79 choose 79. And because 'b' is copartner we don't need to fill in secret number panel. (But culprit must fill in the secret number otherwise it throw NullPointerException.)



And it automatically updates both player's markers.

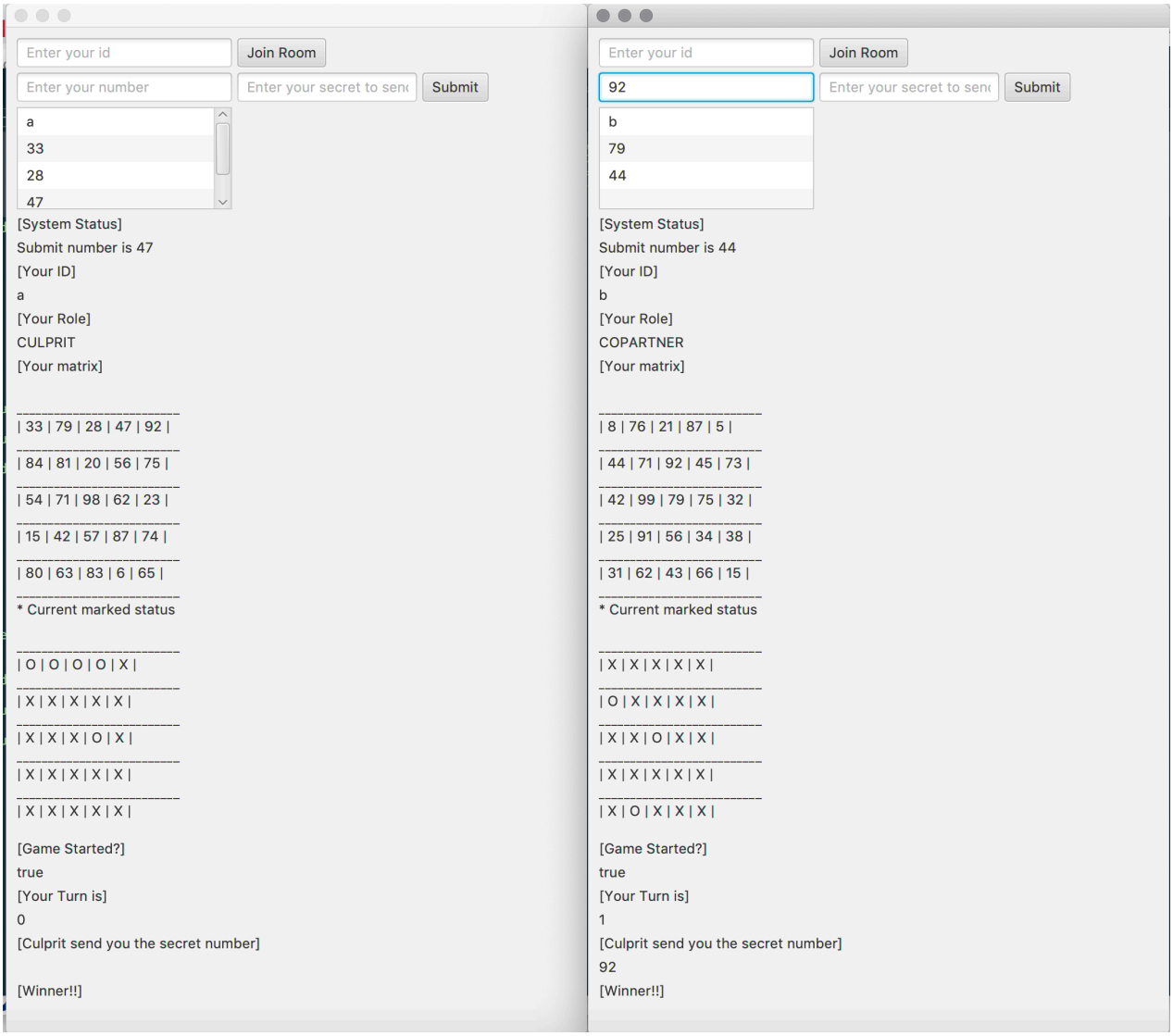

Enter your id
Join Room
Enter your number
Enter your secret to sen
Submit
a
33
[System Status]
Submit number is 33
[Your ID]
a
[Your Role]
CULPRIT
[Your matrix]
_____
| 33 | 79 | 28 | 47 | 92 |
_____
| 84 | 81 | 20 | 56 | 75 |
_____
| 54 | 71 | 98 | 62 | 23 |
_____
| 15 | 42 | 57 | 87 | 74 |
_____
| 80 | 63 | 83 | 6 | 65 |
_____
* Current marked status
_____
| O | O | X | X | X |
_____
| X | X | X | X | X |
_____
| X | X | X | X | X |
_____
| X | X | X | X | X |
_____
| X | X | X | X | X |
[Game Started?]
true
[Your Turn is]
0
[Culprit send you the secret number]
[Winner!!]
Enter your id
Join Room
Enter your number
Enter your secret to sen
Submit
b
79
[System Status]
Submit number is 79
[Your ID]
b
[Your Role]
COPARTNER
[Your matrix]
_____
| 8 | 76 | 21 | 87 | 5 |
_____
| 44 | 71 | 92 | 45 | 73 |
_____
| 42 | 99 | 79 | 75 | 32 |
_____
| 25 | 91 | 56 | 34 | 38 |
_____
| 31 | 62 | 43 | 66 | 15 |
_____
* Current marked status
_____
| X | X | X | X | X |
_____
| X | X | X | X | X |
_____
| X | X | O | X | X |
_____
| X | X | X | X | X |
_____
| X | X | X | X | X |
[Game Started?]
true
[Your Turn is]
1
[Culprit send you the secret number]
79
[Winner!!]

```
Current turn user id is [pseudo1]
Marked at [3][0]
Result of bingo matrix marking: true, id=pseudo1
Not marked this turn ... :-(
Result of bingo matrix marking: false, id=pseudo2
Marked at [3][4]
Result of bingo matrix marking: true, id=pseudo3
Not marked this turn ... :-(
Result of bingo matrix marking: false, id=a
Not marked this turn ... :-(
Result of bingo matrix marking: false, id=b
########################################
###############[pseudo1]#################
########################################
method>>matrix_updated
method>>matrix_updated
Current turn user id is [pseudo2]
Not marked this turn ... :-(
Result of bingo matrix marking: false, id=pseudo1
Marked at [2][1]
Result of bingo matrix marking: true, id=pseudo2
Not marked this turn ... :-(
Result of bingo matrix marking: false, id=pseudo3
Not marked this turn ... :-(
Result of bingo matrix marking: false, id=a
Not marked this turn ... :-(
Result of bingo matrix marking: false, id=b
########################################
###############[pseudo2]#################
########################################
method>>matrix_updated
method>>matrix_updated
Current turn user id is [pseudo3]
Not marked this turn ... :-(
Result of bingo matrix marking: false, id=pseudo1
Not marked this turn ... :-(
Result of bingo matrix marking: false, id=pseudo2
Marked at [0][3]
Result of bingo matrix marking: true, id=pseudo3
Not marked this turn ... :-(
Result of bingo matrix marking: false, id=a
Not marked this turn ... :-(
Result of bingo matrix marking: false, id=b
########################################
###############[pseudo3]#################
########################################
method>>matrix_updated
method>>matrix_updated
Current turn user id is [a]
<==========---> 80% EXECUTING [17m 16s]
```

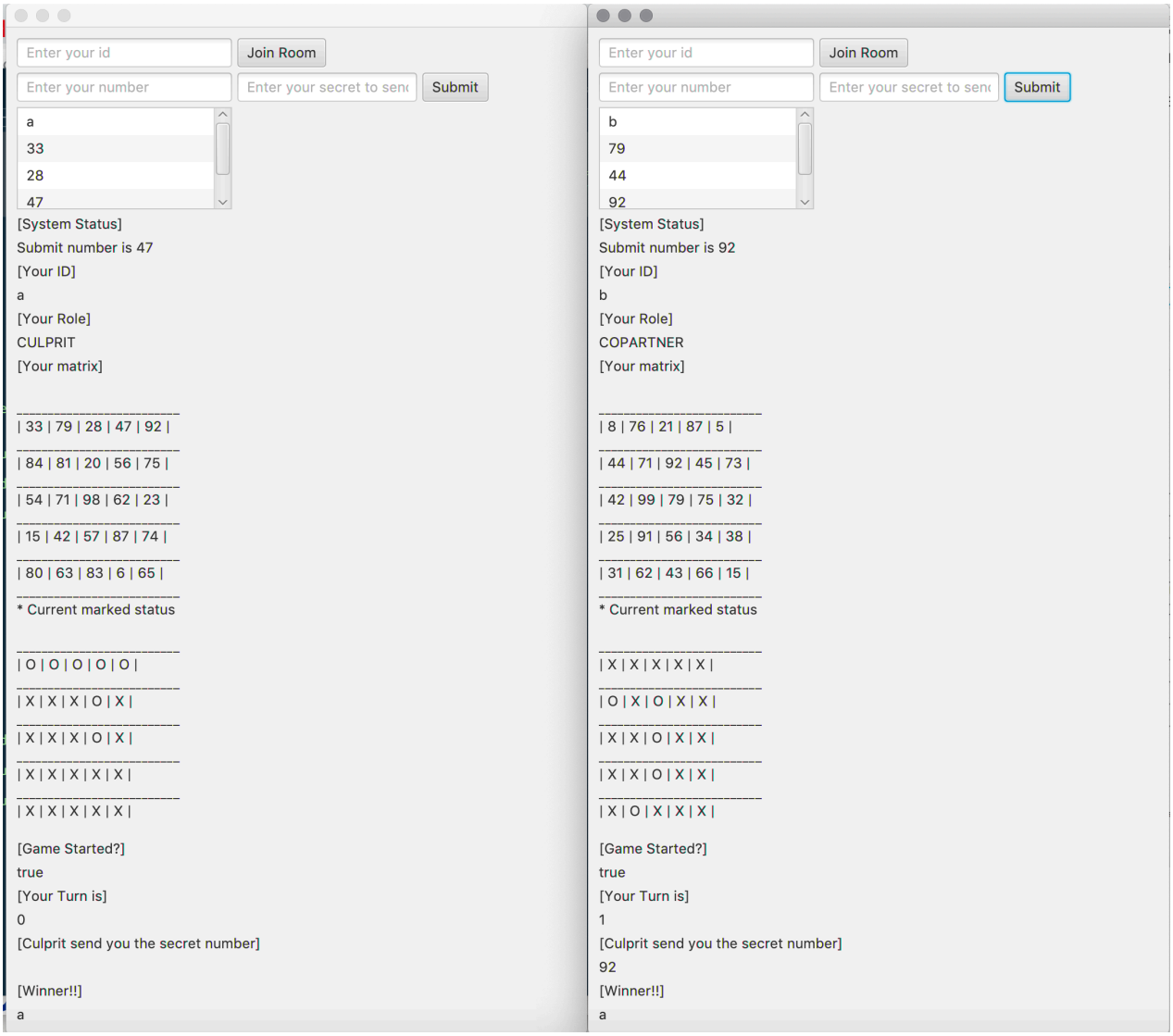
> And because now is the Pseudo player's turn. Whenever player choose number on the server, it logs whether other player's marker is updated or not.

Now the player 'a' turn again and doing things again and again until marker made bingo.

Player 'a' won



Now this is the last turn. 'a' pass secret number 92 to copartner and 'b' has number 92, so submit 92.



Finally players can see 'a' is winner on the '[Winner!]' panel.

Discussion

Exception Handling

There's lots of try-catch clause and in this homework almost all codes in the catch clause is just `e.printStackTrace();` I want to search how can we handle various exception in a nice way.

Gracefully disconnect socket

When ctrl+c to shutdown client, there's error message on the server. So I was curious about how can we gracefully shutdown the client.