



**Department of Computer Science**  
**CS3004 Network Computing Coursework**

Academic Year 2019 - 2020

**The Yahtzee Application**

Anish Limbu

1618834

## 1. Introduction

For this assignment, I have to build a console-based multiplayer Yahtzee game in Java. The game will have a server and multiple clients (three or more). A server is required to manage the scores of the players, to connect players, to receive information from the players, etc. The clients will connect to the server on a specific port, and the game will start when three or more players are connected to the server. A game consists of thirteen rounds. Each round, the players will interact with the server in a turn basis. A round starts off by the players rolling five dice to make certain combinations. After they roll, they have three chances to reroll to make various scoring combinations. Once the player picks the category to score in, it cannot be used again. After thirteen rounds, the winner is announced to all the players and the game ends.

## 2. Requirements

- Input and output operations are done through the IDE's console. For example, information sent by the server is printed to the console, and the number of dice to reroll are read from the console.
- Each client must contact the server in order to play the game. To do this, each client will use a unique port number and IP address from the rest of the players. This will allow the server to communicate back with the client in the game (e.g. sending scores).
- The server must be multi-threaded to handle multiple players at once. For example, sending the scores to all the players at once should be asynchronous.
- An array data structure needs to be implemented to store the players. If a client contacts the server and the array length is less than 3, then the game must wait until 3 or more clients are present in the array. This array can also be used to store the players (e.g. `player1`, `player2`, ..., `playern` where `n` is the number of players) to send scores, keep object references, etc.
- The server does most of the processing here. It must use synchronisation to prevent race-condition errors and shared objects being modified incorrectly. This can be done by coordinating the actions of players using synchronisation. Also, each player must have their turn in a round-robin fashion (one after another). After the last player's turn, it should be the first player's turn.
- Clients should be able to see the score board of all the players from the game (including theirs). To do this, the server must broadcast this information to all the players in the game. When the game ends, the server should inform each player who the winner is.
- The server must know who scored what. This will be used by the server to keep information of the players in the game (e.g. identifying who the winner is).

## 3. Design

A player can either be waiting for their turn or having their turn in each round. Players who are waiting for their turn will just listen for incoming messages from the server. When the server sends messages to the players who are waiting for their turn, they will receive the messages and display it to their console. This will allow them to receive information such as who scored what. When a player is having their turn, they will interact with the server (e.g. informing the server what category they want to score in); therefore, the player updates the server, and the server updates the rest of the players who are waiting. This allows only one player to have their turn at a time.

YAHTZEECLIENT		YAHTZEESERVER
		[RUN YahtzeeServer]
		PRINT "Yahtzee server initialised at port 4321"
		SET totalNumberOfPlayers to 0
		SET playersList to null
		SET startGame to FALSE
		SET startGameChoice to "n"
		SET userInput to null
		SET totalAcks to 0
		SET playerTurnIndex to 0
		SET turnManager to null
[RUN YahtzeeClient]		<b>WHILE NOT TERMINATED</b>
SET chosenCategories to null		
SET scoreboard to null		
SET totalScore to 0		
SET userInput to null		
SET rerollChances to 3		
[CONNECT to YahtzeeServer]		
PRINT "Client initialised: " + IP of host + " at " + port number of host		
<b>WHILE NOT TERMINATED</b>	#S1	<b>WHILE NOT startGame THEN</b>
		<b>IF totalNumberOfPlayers &gt;= 3 AND NOT startGame THEN</b>
		PRINT "Three players (or more) has joined the game"
		PRINT "Would you like to start the game (y/n)? "
		READ userInput and set it to startGameChoice
		<b>IF startGameChoice = "y" THEN</b>
		startGame = TRUE
		<b>GOTO #S2</b>
		<b>END IF</b>
		<b>ELSE if NOT startGame THEN</b>
		[ACCEPT YahtzeeClient connection]
		CREATE new player
		SET player's I/O to YahtzeeClient I/O streams
		SET totalNumberOfPlayers = totalNumberOfPlayers + 1
		SET player's name to "Player " + totalNumberOfPlayers
		ADD player to playersList
		PRINT "Accepted a connection from " + IP of YahtzeeClient + " at port " + port number of YahtzeeClient
		SEND "You are Player " + totalNumberOfPlayers to YahtzeeClient
RECEIVE "You are Player " + totalNumberOfPlayers to YahtzeeClient" FROM YahtzeeServer and print to console		
		PRINT "Total players in game: " + totalNumberOfPlayers
		PRINT playersList
		<b>END IF</b>
		<b>END WHILE</b>
	#S2	PRINT "The game will now begin"
		PRINT "The game contains " + totalNumberOfPlayers + " players."
		SEND "Game will start now!!!" to each player in playersList
RECEIVE "Game will start now!!!" and print to console		
		SET turnManager to new TurnManager(number of players in the game) <i>[Pass the number of players inside the TurnManager constructor]</i>
		<b>FOR player in playersList</b>
		SET threadPlayer to new thread(player) <i>[A new thread for each player]</i>
		SET player's turnManager to turnManager <i>[Each player will share this object]</i>
		START threadPlayer
		JOIN threadPlayer
		<b>END FOR</b>
		SET roundNumber to 1
	#S3	<b>WHILE roundNumber &lt;= 13</b>
		SEND "ROUND_NUMBER" to each player in playersList
RECEIVE "ROUND_NUMBER" from YahtzeeServer		
		SEND roundNumber to each player in playersList
RECEIVE roundNumber from YahtzeeServer		
PRINT "Round " + roundNumber + " of 13"		
		Roll dice of each player in playersList

		SEND "PLAYER_ROLL_DICES" to each player in playersList
	RECEIVE "PLAYER_ROLL_DICES" from YahtzeeServer	
		SEND dice rolled by each player, to all the players in playersList
	RECEIVE dice rolled from YahtzeeServer	
	PRINT dice rolled by each player in this round	
		SEND scoreboard of each player to players in playersList
	RECEIVE scoreboard from YahtzeeServer	
	PRINT scoreboard of each player in this round	
		<b>WHILE TRUE</b>
		SYNCHRONISED playersList
		SET playerTurnIndex to playerTurn from turnManager <i>[The server accesses a property from TurnManager class. This playerTurn indicates whose turn is next. This number will be used to index into playersList]</i>
		SET playerTurnNow to playersList[playerTurnIndex] <i>[playerTurnNow is the player who is having their turn]</i>
		SEND each player "OTHER_PLAYER_TURN" excluding playerTurnNow
	<b>WHILE NOT TERMINATED</b>	SEND each player playerTurnNow's name excluding playerTurnNow
	RECEIVE messageFromServer YahtzeeServer	SEND "PLAYER_TURN" TO playerTurnNow
	<b>IF messageFromServer = "OTHER_PLAYER_TURN" THEN</b>	
	<i>[The clients here are in a waiting state. It is not their turn; therefore, they will wait until the server sends them the "PLAYER_TURN" message. During this state, the clients will receive messages from YahtzeeServer and display it in their console, such as who scored what, if a player rerolled dice, etc. Therefore, the clients who are waiting for their turn will just print the messages sent by the YahtzeeServer.]</i>	
	<b>WHILE NOT "PLAYER_TURN" received THEN</b>	
	RECEIVE message sent by YahtzeeServer	
	PRINT message to console	
	<b>END WHILE</b>	
	<b>ELSE IF messageFromServer = "PLAYER_TURN" THEN</b>	
	REQUEST LOCK/TURN by playerTurnNow	
	SEND "PLAYER_REQUEST_TURN" to YahtzeeServer	
		RECEIVE "PLAYER_REQUEST_TURN" from playerTurnNow
		SEND "PLAYER_GRANTED_TURN" to playerTurnNow
	RECEIVE "PLAYER_GRANTED_TURN" from YahtzeeServer	
		SEND "It is your turn" to playerTurnNow
	RECEIVE "It is your turn" from YahtzeeServer and print to console	
		SEND "It is " + playerTurnNow's name + " turn" to each player excluding playerTurnNow <i>[Inform players who are waiting whose turn it is now]</i>
		SEND "PLAYER_REROLL_DICES" to playerTurnNow
	RECEIVE "PLAYER_REROLL_DICES" from YahtzeeServer	
	SET totalRerollsRemaining = 3	
#C1	SET rerollChoice to null	
	PRINT "Total re-rolls remaining: " + totalRerollsRemaining	
	PRINT "Would you like to re-roll dice (y/n)?"	
	READ userInput and set it to rerollChoice	
	<b>IF rerollChoice NOT "y" or "n" THEN</b>	
	<b>#GOTO C1</b>	
	<b>END IF</b>	
	<b>IF rerollChoice = "n" THEN</b> <i>[PLAYER DECIDED NOT TO RE-ROLL DICE]</i>	
#C2	SEND "PLAYER_GET_WHAT_CAN_BE_SCORED" to YahtzeeServer	
		RECEIVE "PLAYER_GET_WHAT_CAN_BE_SCORED" from playerTurnNow
		SEND scoring categories, the playerTurnNow can pick with their dice, to playerTurnNow <i>[Send the player who is having their turn what they can score in]</i>
		SEND "OTHER_PLAYER_GET_WHAT_CAN_BE_SCORED" to each player in playersList excluding playerTurnNow
		SEND playerTurnNow's name + " can score in the following: " + scoring categories to each player in playersList excluding playerTurnNow <i>[Inform players who are waiting what categories the player can score in]</i>

	RECEIVE scoring categories from YahtzeeServer		
	PRINT scoring categories		
	PRINT scoring categories they have not picked		
	READ userInput and set it to categoryChosen		
	SEND "PLAYER_CHOOSE_SCORING_CATEGORY" to YahtzeeServer		
	SEND categoryChosen to YahtzeeServer		
			RECEIVE "PLAYER_CHOOSE_SCORING_CATEGORY" from playerTurnNow
			RECEIVE categoryChosen from playerTurnNow
			UPDATE playerTurnNow's scoreboard using categoryChosen
			SEND "OTHER_PLAYER_CHOSE_SCORING_CATEGORY" to each player excluding playerTurnNow
			SEND playerTurnNow's name + " chose to score in " + categoryChosen + " giving them " + pointsScored + " points" to each player in playersList excluding playerTurnNow <i>[Inform players who are waiting what category the player chose to score in]</i>
	<b>ELSE IF rerollChoice = "y" AND totalRerollsRemaining != 0 THEN</b> <i>[Player decided to reroll dice]</i>		
	SET diceIndexes to null		
	SEND "PLAYER_REROLLED_DICES" to YahtzeeServer		
			RECEIVE "PLAYER_REROLLED_DICES" from playerTurnNow
	SEND "PLAYER_REQUEST_REROLL_CHANCES" to YahtzeeServer		
			RECEIVE "PLAYER_REQUEST_REROLL_CHANCES" from playerTurnNow
			DECREMENT playerTurnNow's rerollChances <i>[The server keeps a copy too to ensure the player cannot reroll more than 3]</i>
			SEND playerTurnNow's name + " rerolled and has " + playerTurnNow's reroll chances to each player in playersList, excluding playerTurnNow <i>[Inform players who are waiting that the player has rerolled dice and how many rerolls they have left]</i>
	SET rerollChances = rerollChances – 1		
	PRINT "Enter the number of dice to reroll: "		
	READ userInput and set it to numberOfDiceToReroll		
	SEND "PLAYER_NUMBER_OF_DICES_REROLLED" to YahtzeeServer		
			RECEIVE "PLAYER_NUMBER_OF_DICES_REROLLED" from playerTurnNow
	SEND numberOfDiceToReroll to YahtzeeServer		
			RECEIVE numberOfDiceToReroll from playerTurnNow
			SEND playerTurnNow's name + " chose " + numberOfDiceToReroll + " dice to reroll" to each player excluding playerTurnNow <i>[Inform players who are waiting how many dice the player wants to reroll]</i>
	<b>FOR 1 to numberOfDiceToReroll</b>		
	PRINT "Select a dice: "		
	READ userInput and add to diceIndexes		
	<b>END FOR</b>		
	SEND diceIndexes to YahtzeeServer		
			RECEIVE diceIndexes from playerTurnNow
			REROLL playerTurnNow's dice using numberOfDiceToReroll and diceIndexes
			SEND playerTurnNow's name + " chose " + diceIndexes + " dice index(es) to reroll" to each player excluding playerTurnNow <i>[Inform players who are waiting what dice indexes the player wants to reroll]</i>
			SET newRerolledDiceOfPlayer to playerTurnNow's rerolled dice
			SEND "PLAYER_NEW_REROLLED_DICE" to playerTurnNow
			SEND newRerolledDiceOfPlayer to playerTurnNow <i>[Send the player their rerolled dice]</i>
			SEND "OTHER_PLAYER_REROLL_DICES" to each player in playersList excluding playerTurnNow
			SEND "After rerolling, " + playerTurnNow's name + " new dice are " + newRerolledDiceOfPlayer to each player excluding playerTurnNow <i>[Inform players who are waiting the new rerolled dice of player]</i>
	RECEIVE "PLAYER_NEW_REROLLED_DICE" from YahtzeeServer		
	RECEIVE newRerolledDiceOfPlayer from YahtzeeServer		
	PRINT "After rerolling, your dice are "+ newRerolledDiceOfPlayer		
	<b>#GOTO C1</b>		
	<b>END IF</b>		
	<b>IF rerollChoice = "n" THEN</b> <i>[After the player is done with their turn, player sends a message to the YahtzeeServer]</i>		
	SEND "PLAYER_FINISH_REROLL_DICES" to YahtzeeServer		

	SEND "PLAYER_FINISH_TURN" to YahtzeeServer		
	END IF		
			RECEIVE "PLAYER_FINISH_REROLL_DICES" from playerTurnNow
			RECEIVE "PLAYER_FINISH_TURN" from playerTurnNow
	INCREMENT playerTurn from turnManager <i>[The player will increment this shared variable so the next player can have their turn. This value becomes 0 when every player has had their turn in a round.]</i>		
	RELEASE LOCK/TURN by playerTurnNow		
	END IF <i>[Player has finished their turn]</i>		
			IF "PLAYER_FINISH_TURN" was received THEN
			SET totalAcks = totalAcks + 1
			END IF
			IF totalAcks = total number of players in playersList THEN
			<i>[Each player has a sent an ACK to the server after having their turn]</i>
			SEND the score of players to each player in playersList
			SET totalAcks to 0
			SET roundNumber = roundNumber + 1
			END IF
			END WHILE <i>[After the player has had their turn, they will increment the shared variable. The server will now break the loop and allow the next player to have their turn.]</i>
			END WHILE <i>[After 13 rounds]</i>
			SEND "GAME_END" to each player in playersList <i>[The server will announce the winner before the game ends]</i>
	RECEIVE "GAME_END" from YahtzeeServer		
			COMPUTE the winner from playersList and set it to playerWinner <i>[The player(s) with the highest score will be added to playerWinner. Also, check for draws here.]</i>
			IF size of playerWinner >= 2 THEN
			SET playerWinner as the name of players in playerWinner <i>[Draw happened]</i>
			ELSE
			SET playerWinner as the name of player with the highest score <i>[A single winner]</i>
			END IF
			SEND "WINNER_ANNOUNCEMENT" to each player in playersList
	RECEIVE "WINNER_ANNOUNCEMENT" from YahtzeeServer		
			SEND playerWinner to players in playersList
			TERMINATE YahtzeeServer
	RECEIVE playerWinner from YahtzeeServer		
	PRINT playerWinner		
	TERMINATE YahtzeeClient		

## 4. Implementation

In this part, I will talk about how I implemented my solution using snippets of code from my program.

```
17 public class YahtzeeServer extends Thread {
18     private ServerSocket socket;
19     private int playerNumber = 0;
20     private Scanner inputKeyboard;
21
22     // The players
23     private final List<Player> players;
24     private String startGameOrNotOption = "";
25     private boolean startGame;
26     private Broadcast broadcast;
27     private ServerOperation serverOperation;
28     private TurnManager turnManager;
29
30     private YahtzeeServer(int portNumber) throws IOException {
31         // Handle only up to 50 connection requests
32         this.socket = new ServerSocket(portNumber, 50);
33         System.out.println("Yahtzee server initialised at port " + portNumber);
34         this.inputKeyboard = new Scanner(System.in);
35         this.players = Collections.synchronizedList(new ArrayList<>());
36         start();
37     }
38 }
```

**Figure 4.1** YahtzeeServer instance variables and constructor.

The code above shows important properties of the YahtzeeServer. The server uses ServerSocket object in order to bind to a port on the machine. This port number will be passed by the user in the main method. Also, I have passed the number fifty inside the constructor of ServerSocket, so that the server can only handle fifty requests at once. Another important thing here is the players list. Each player object will implement the Runnable interface for performance and synchronisation. Therefore, I have to use a synchronised data structure object in order to store the players in the game. After researching online, I found out ArrayList alone is unsynchronised and not thread safe. This meant I could have race-condition problems. To solve this, I have used a synchronised list for extra protection. Even though only one server is accessing the list; the methods in Player class requires synchronisation. When clients join the game, new player objects are created and added to players list.

```
14 public class YahtzeeClient extends Thread {
15     private ObjectInputStream input;
16     private ObjectOutputStream output;
17     private Scanner inputKeyboard;
18     private int rerollChances;
19     private boolean gameHasEnded;
20     private ArrayList<Integer> categoriesToScore;
21
22     private YahtzeeClient() throws IOException {
23         InetAddress localhost = InetAddress.getLocalHost();
24         Socket socketConnection = new Socket(localhost, 4321);
25         System.out.println("Client initialised: " + InetAddress.getLocalHost() + " at " + socketConnection.getLocalPort());
26         output = new ObjectOutputStream(socketConnection.getOutputStream());
27         input = new ObjectInputStream(socketConnection.getInputStream());
28         inputKeyboard = new Scanner(System.in);
29         categoriesToScore = new ArrayList<>();
30         start();
31     }
32 }
```

**Figure 4.2.** YahtzeeClient instance variables and constructor.

YahtzeeClient is a separate class which is used to interact with the server and user. The user uses this class to contact the YahtzeeServer, send input to the server, see output sent by the server, etc. The figure above shows the YahtzeeClient constructor and its instance variables. A socket object is created in order to contact the YahtzeeServer on port 4321. This class also stores the score categories in an array list. This is to prevent the user from choosing the already picked scoring categories. Since each client will communicate with the server, it will require input and output streams. Therefore, each client will use ObjectInputStream and ObjectOutputStream for I/O network operations. These classes can be used to send and receive Java objects (e.g. String, Stream, arrays, etc.). For example, I will use

ObjectOutputStream to write String objects (e.g. messages) to YahtzeeServer socket, and ObjectInputStream to read messages sent by YahtzeeServer. These messages will be printed to the console for the user to see.

```

76 private void acceptPlayerConnection() throws IOException {
77     Player player = new Player(socket.accept());
78
79     Socket connectionSocket = player.getSocket();
80     if(!startGame)
81         System.out.println("Accepted a connection from " + connectionSocket.getInetAddress() + ":" + connectionSocket.getPort())
82     else
83         System.out.println("Game has started, but a client has been denied to join the game from" + connectionSocket.getInetAddress());
84
85     ObjectOutputStream outputStream = new ObjectOutputStream(connectionSocket.getOutputStream());
86     ObjectInputStream inputStream = new ObjectInputStream(connectionSocket.getInputStream());
87
88     // If there are enough players and the game has started
89     // decline the connection requests
90     if(playerNumber >= 3 && startGame) {
91         String message = "Cannot join game, as game is currently being played. Please try again later.";
92         outputStream.writeObject(message);
93     } else {
94         players.add(player);
95         // Write the player number to the client
96         players.get(playerNumber).setInputStream(inputStream);
97         players.get(playerNumber).setOutputStream(outputStream);
98         players.get(playerNumber).setPlayerNumber(playerNumber + 1);
99         players.get(playerNumber).setPlayerName("Player " + (playerNumber + 1));
100         String message = "You are Player " + (++playerNumber);
101         outputStream.writeObject(message);
102         System.out.println("Total players in game: " + playerNumber);
103     }
104     System.out.println(players);
105 }
106

```

**Figure 4.3.** Accepting player connection.

In figure 4.3, I show how the server accepts a single connection request from YahtzeeClient. The server runs this in a separate while loop to handle YahtzeeClient connections. When the game has not started, then the player is initialised and added to an array list. When the player is initialised, then the server sends important information to the player. These include player name and player number. In line 77, the socket is accepted and passed to the constructor of the player object. Therefore, the server can use the player's socket in order to send messages to the player. In line 101, I show an example of how I write to the player's socket.

```

10 public class Player implements Runnable {
11     private PlayerScoreBoard scoreBoard;
12     private Socket socket;
13     private ObjectOutputStream output;
14     private ObjectInputStream input;
15     private int playerNumber;
16     private String playerName;
17     private int[] dicesRolled;
18     private Dices dices;
19     private ScoreCheck scoreCheck;
20     private TurnManager turnManager;
21
22     public Player(Socket socket) {
23         this.socket = socket;
24         this.scoreBoard = new PlayerScoreBoard();
25         this.dices = new Dices();
26         this.scoreCheck = new ScoreCheck(dices);
27     }
28

```

**Figure 4.4.** The Player class.

The player class is used to instantiate player objects. It implements the Runnable interface, so the YahtzeeServer can spawn a thread for each player and execute it. Each player object has its own properties such as scoreboard, dice, player name, socket, etc. Therefore, each client in the game is represented by a player object. This makes it easy for the server to set and get the properties of the players in the game (e.g. score).



```
117 private void initialiseSharedObject() {  
118     turnManager = new TurnManager(players.size());  
119     // Start the thread of each player  
120     startPlayers();  
121 }  
122  
123 private void startPlayers() {  
124     synchronized (players) {  
125         for(Player player : players) {  
126             player.setTurnManager(turnManager);  
127             Thread t = new Thread(player);  
128             t.start();  
129             try {  
130                 t.join();  
131             } catch (InterruptedException e) {  
132                 e.printStackTrace();  
133             }  
134         }  
135     }  
136 }  
137
```

**Figure 4.5.** Starting players.

When the game starts, the server creates a TurnManager object that will be shared by all the players in the game. In line 118, in figure 4.5, I pass the total number of players in the game inside the TurnManager constructor. When one player requests the lock and has their turn, they will increment the value and release the lock. This allows only one player to have their turn while others wait for that player to finish. Each player will share this object with each other. In figure 4.5, I have created a startPlayers method. I use this method to start the thread of each player and make sure the players share the object correctly.

```
13 public class Broadcast {  
14     private List<Player> players;  
15  
16     Broadcast() {}  
17  
18     Broadcast(List<Player> players) {  
19         this.players = Collections.synchronizedList(players);  
20     }  
21  
22     private synchronized void broadcastMessage(String message) {  
23         for(final Player player: players) {  
24             player.sendMessage(message);  
25         }  
26     }  
27  
28     synchronized void broadcastMessage(Player currentPlayer, String message) {  
29         final String PLAYER_NAME = currentPlayer.getPlayerName();  
30         for(final Player player: players) {  
31             if(!player.getPlayerName().equals(PLAYER_NAME))  
32                 player.sendMessage(message);  
33         }  
34     }  
35  
36     private synchronized void broadcastDicesRolledMessage(String message) {  
37         for(final Player player: players) {  
38             final String PLAYER_ROLLED_DICES = message.replaceAll(player.getPlayerName(), "You");  
39             player.sendMessage(PLAYER_ROLLED_DICES);  
40         }  
41     }  
42 }
```

**Figure 4.6.** The Broadcast class.

The game requires the server to do most of the processing. These include sending each player the round number, what was scored each round, score of each player, etc; therefore, I created this broadcast class for this type of functionality. The YahtzeeServer uses this class to broadcast messages to each player in the game.

```

3 public class TurnManager {
4     // These values are shared amongst the players
5     private boolean isAccessing;
6
7     private int playerIndex;
8     private final int MAX_NUMBER_OF_PLAYERS;
9
10    public TurnManager(int playersInGame) {
11        this.isAccessing = false;
12        this.MAX_NUMBER_OF_PLAYERS = playersInGame;
13    }
14    private synchronized void increasePlayerIndex() {
15        playerIndex = playerIndex == MAX_NUMBER_OF_PLAYERS - 1 ? 0 : ++playerIndex;
16    }
17
18    public synchronized int getPlayerTurn() {
19        return playerIndex;
20    }
21
22    public synchronized void requestTurn() throws InterruptedException {
23        while(isAccessing) {
24            wait();
25        }
26        isAccessing = true;
27    }
28
29    public synchronized void releaseTurn() {
30        isAccessing = false;
31        notifyAll();
32        increasePlayerIndex();
33    }
34 }

```

**Figure 4.7.** The TurnManager class for synchronisation.

The class above is used to synchronise the turns. Therefore, when a player is having their turn, other players will have to wait until it is their turn. The player objects are responsible for updating the variables here. After a player has their turn, they will increment the playerIndex variable and release their turn. Therefore, the server knows whose turn it is next.

```

137    private synchronized void startGame() {
138        int totalAcks = 0;
139        initialiseSharedObject();
140        for(int roundNumber = 1; roundNumber <= 13; ) {
141            // Send the round number to all the players
142            broadcast.broadcastPlayersRoundNumber(roundNumber);
143            broadcast.broadcastDicesRolledByPlayers(serverOperation.getPlayersDicesRolled());
144            serverOperation.sendPlayersScoreBoard();
145
146            while(true) {
147                synchronized(players) {
148                    // Get the index of whose turn it is now
149                    int playerTurnIndex = turnManager.getPlayerTurn();
150
151                    // The player who will communicate with the server
152                    Player player = players.get(playerTurnIndex);
153
154                    // Inform each player whose turn it is right now
155                    broadcast.broadcastMessage(player, ServerOperationConstants.OTHER_PLAYER_TURN);
156                    broadcast.broadcastMessage(player, player.getPlayerName());
157
158                    System.out.println("-----Begin Transaction-----");
159                    System.out.println("Synchronised player index number: " + turnManager.getPlayerTurn());
160                    // Let the player have their turn
161                    // Lock to prevent from multiple threads accessing and modifying the player index
162                    try {
163                        player.getTurnManager().requestTurn();
164                        serverOperation.grantPlayerTurn(player);
165                        player.getTurnManager().releaseTurn();
166                    } catch (InterruptedException e) {
167                        e.printStackTrace();
168                    }
169
170                    // The user sends the PLAYER_FINISH_TURN message to the server
171                    String messageFromClient = player.readMessage();
172                    System.out.println(messageFromClient + " from " + player.getPlayerName());
173                    if(messageFromClient.equals(ServerOperationConstants.PLAYER_FINISH_TURN)) {
174                        totalAcks++;
175                        System.out.println("-----End Transaction-----\n");
176                    }
177
178                    if(totalAcks == players.size()) {
179                        System.out.println("Got all acks for round " + roundNumber);
180                        totalAcks = 0;
181                        broadcast.broadcastPlayerScoreTotal();
182                        roundNumber++;
183                        break;
184                    }
185                }
186            }
187        }
188        // Inform each player who the winner is
189        broadcast.declareWinner();
190    }

```

**Figure 4.8.** Yahtzee algorithm.

The figure above shows how the game works. Beginning of each round, the server will broadcast the round number, dice rolled by each player, and scoreboard of each player to all the players in the game. In line 149, the server gets the index of whose turn it is next. Since all the players in the game are stored in an array list, then a number is required in order to index into the array list. Therefore, the first player is represented by the index 0, second player is represented by 1, and so on. The game uses this method so that each player can have their turn in a round-robin fashion (one after another). Before a player has their turn, the server will inform each player whose turn it is next. Therefore, other players who are waiting for their turn will wait patiently until it is their turn. Before a player has their turn, they will request the lock in TurnManager. After the player is done, they will increment the index and release the lock, and the server will interact with the next player who is waiting for their turn.

The game uses a variable called totalAcks which is similar to the TCP protocol. It stores the total number of acknowledgements sent by the players. After a player has their turn, they will send a "PLAYER\_FINISH\_TURN" message to the server. When the server receives this message, it will increment the totalAcks, and know that player's turn is over. When the server receives all the acknowledgments from the players in that round, then the server knows each player has had their turn, and will reset the number of acks, broadcast the total score of each player to all the players, and increment the round number. After 13 rounds, the server will broadcast the winner to each player.

```

33 @Override
34 public void run() {
35     try {
36         printWelcomeMessage();
37         while(!gameHasEnded) {
38             // Receive a message from server
39             respondToServerOperations();
40         }
41     } catch (IOException | ClassNotFoundException e) {
42         e.printStackTrace();
43     }
44 }
45
46 private void printWelcomeMessage() {
47     String messageFromServer = null;
48     try {
49         messageFromServer = (String) input.readObject();
50     } catch (IOException | ClassNotFoundException e) {
51         e.printStackTrace();
52     }
53     System.out.println(messageFromServer);
54 }
55
56 private void respondToServerOperations() throws IOException, ClassNotFoundException {
57     while(!gameHasEnded) {
58         String operationFromServer = (String) input.readObject();
59         switch(operationFromServer) {
60             case ServerOperationConstants.ROUND_NUMBER:
61                 String roundNumber = (String) input.readObject();
62                 System.out.println("-----[ Round " + roundNumber + " of 13" + " ]-----");
63                 break;
64             case ServerOperationConstants.OTHER_PLAYER_TURN:
65                 String playerTurnName = (String) input.readObject();
66                 System.out.println("It is " + playerTurnName + "'s turn");
67                 break;
68             case ServerOperationConstants.PLAYER_TURN:
69                 output.writeObject(ServerOperationConstants.PLAYER_REQUEST_TURN);
70                 String replyFromServer = (String) input.readObject();
71                 if(replyFromServer.equals(ServerOperationConstants.PLAYER_GRANTED_TURN))
72                     System.out.println("It is your turn");
73                 break;
74             case ServerOperationConstants.PLAYER_FINISH_TURN:
75                 informServerTurnFinished();
76                 break;
77             case ServerOperationConstants.PLAYER_ROLL_DICES:
78                 String diceRolled = (String) input.readObject();
79                 System.out.println("You rolled: " + diceRolled);
80                 break;
81             case ServerOperationConstants.PLAYER_REROLL_DICES:
82                 // Read the number of reroll chances the player has currently
83                 rerollChances = 3;
84                 rerollDices();
85                 break;

```

**Figure 4.9.** YahtzeeClient algorithm.

The user uses YahtzeeClient in order to play the game. The user will use this class to see information sent by the server, send input to the server, and so on. This is accomplished by using the respondToServerOperations method in figure 4.9. For example, when the server sends 'ROUND\_NUMBER' to the players, the line 60 will be executed. This is done by comparing the operation sent by the server.

```

60 synchronized void declareWinner() {
61     // Inform them the game has finished
62     broadcastMessage(ServerOperationConstants.GAME_END);
63
64     // Players sorted by their score (highest to smallest)
65     List<Player> playerSortedByScores = players.stream()
66         .sorted(Comparator.comparing(Player::getTotalScore).reversed())
67         .collect(Collectors.toCollection(ArrayList::new));
68
69     // The highest score achieved by a player
70     final int HIGHEST_SCORE = playerSortedByScores.get(0).getTotalScore();
71     final Predicate<Player> SAME_SCORE_AS_TOP_PLAYER = player -> player.getTotalScore() == HIGHEST_SCORE;
72
73     // Obtain the winner
74     // There can be a draw in the game
75     List<Player> winner = playerSortedByScores.stream()
76         .filter(SAME_SCORE_AS_TOP_PLAYER)
77         .collect(Collectors.toCollection(ArrayList::new));
78
79     System.out.println("Max score: " + HIGHEST_SCORE);
80
81     broadcastMessage(ServerOperationConstants.WINNER_ANNOUNCEMENT);
82
83     if(winner.size() >= 2) {
84         String namesOfPlayer = winner.stream().map(Player::getPlayerName).collect(Collectors.joining(", "));
85         System.out.println("There is a draw between " + announcePlayersWithDrawScore(namesOfPlayer));
86         broadcastMessage("There is a draw between " + announcePlayersWithDrawScore(namesOfPlayer));
87     } else {
88         System.out.println(winner.get(0).getPlayerName() + " is the winner!");
89         broadcastMessage(winner.get(0).getPlayerName() + " is the winner!");
90     }
91     System.exit(0);

```

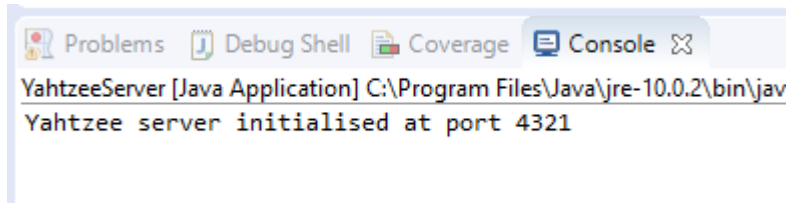
**Figure 4.10.** Winner algorithm.

The game needs to announce the winner or draw before it terminates itself. To do this, the server broadcasts a 'GAME\_END' message to each player. When each player receives this message, they all expect another message next. Therefore, the server identifies if there is a draw or a winner using the Stream object, sends a 'WINNER\_ANNOUNCEMENT' message, and broadcasts it to all the players in the game.

## 5. Testing

In this part, I will test my Yahtzee application. I will begin with a test plan that outlines what I am going to be testing, the expected result, and the actual outcome. For each test, I will provide screenshot(s) to show if the test passed successfully or not.

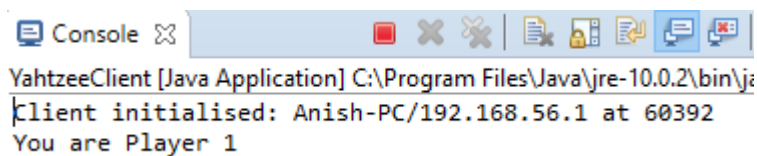
Test	Testing Purpose	Expected Outcome	Actual Outcome	Proof
1	Check whether or not the YahtzeeServer binds to the given port number.	The YahtzeeServer should use the port number given by the user and listen for incoming connections on that port.	The YahtzeeServer successfully binds to the port number 4321 and listens for incoming connections.	5.1.1
2	Test the YahtzeeServer to see if it accepts connection requests from players.	The server should accept connection requests from players who wish to join the game.	The YahtzeeServer successfully accepts connection requests from clients.	5.2.1 5.2.2 5.2.3
3	Check whether or not the server gives the player a name once it joins the game.	Once a player joins the game, the server should send the player its name by its socket. The name must be "Player n" where n is the position of the player in the game.	The Yahtzee successfully sends the player its name by its socket. Also, the server increments the player position by 1 each time a player connects.	5.3.1 5.3.2
4	Test the YahtzeeServer to see if the game starts when three or more players join the game.	Game should start when there are three or more players in the game and the server receives "y" to start. If the server receives "n" then wait for one more player, then ask the question again.	The game only starts when there are three or more players present in the game and the server receives "y". When the input is "n", the server waits for one more player and asks if the game should start or not.	5.4.1 5.4.2 5.4.3
5	Test if each player receives the messages sent by the YahtzeeServer or not when the game starts.	After the game starts, the server sends some messages to each player. The players should receive and print these messages to their console.	The players correctly receive the messages sent by the server. These messages include a welcome banner and the scoreboard of the players. Each player also prints the messages.	5.5.1
6	Check if the server sends the round number and dice rolled by each player to all the players every round.	At the start of each round, the server should broadcast the round number and the dice rolled by each player. Each player must receive this and display it.	This works successfully. Each round, the server sends the round number and the dice rolled by each player to all the players. Each player also receives the messages sent by the server and displays it.	5.6.1
7	Test whether or not all the players can have their turn at the same time. I will test synchronisation between the players.	Other players should be blocked when a player is having their turn. Other players must wait for their turn.	Only one player can interact with the server at a given time. Other players are blocked and they must wait until it is their turn.	5.7.1 5.7.2 5.7.3
8	Check if the user can reroll dice each round up to three times. I will test the communication between the server and the player here.	Each player should be able to reroll dice up to three times each round. If the player enters 'y', then allow them to reroll dice. Otherwise, cancel the reroll dice action. When a player is rerolling dice, other players must be blocked.	The player can reroll up to three times each round. If the user enters 'y', then they are allowed to reroll after the server gives the player permission. If 'n' is entered, then the rerolling action is cancelled. Other players are blocked when a player is rerolling, but they are notified.	5.8.1
9	Test if the player can pick a category to score in. I am testing the communication between the server and the player here.	The server should send what the user can score in that round. Once the player receives this, the player should input an integer to inform the server what category they want to score in. Once picked, the server must broadcast this to waiting players and update the player's scoreboard.	The player successfully receives what they can score in that round. They can also pick the category they want to score in. The server also updates the player's scoreboard and broadcasts important messages to all players (what they scored in, how much points for, etc.).	5.9.1
10	Test if the server can announce the winner or not before the game ends.	The player with the highest score should be declared the winner when the game ends. The server should broadcast this information to all the players in the game. If there is a draw in the game, then the server should broadcast this to all the players.	The game successfully identifies if there is a draw or not. If there is not a draw, the winner is announced to each player. If there is a draw, then the server broadcasts this successfully.	5.10.1



```
Problems Debug Shell Coverage Console
YahtzeeServer [Java Application] C:\Program Files\Java\jre-10.0.2\bin\jav
Yahtzee server initialised at port 4321
```

**Figure 5.1.1.** YahtzeeServer binding to port 4321 and listening.

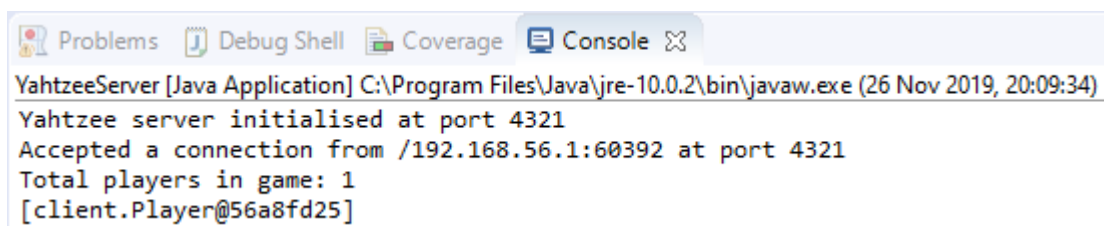
In figure 5.1.1, the YahtzeeServer binds to port 4321 and listens for incoming client connections on that port.



```
Console
YahtzeeClient [Java Application] C:\Program Files\Java\jre-10.0.2\bin\j
Client initialised: Anish-PC/192.168.56.1 at 60392
You are Player 1
```

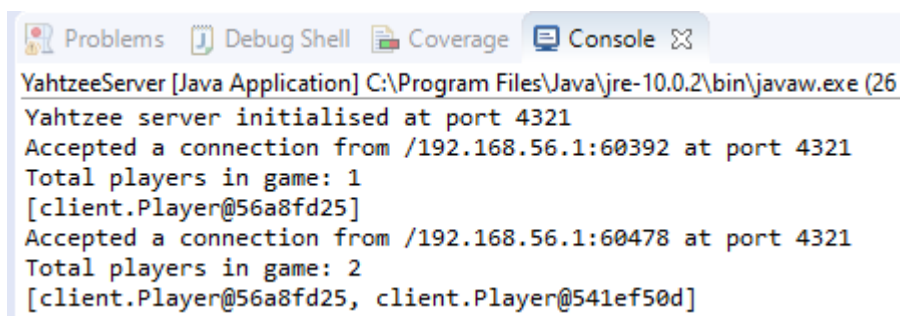
**Figure 5.2.1.** YahtzeeClient contacts YahtzeeServer on port 4321.

In figure 5.2.1, a YahtzeeClient contacts the YahtzeeServer on port 4321. The YahtzeeServer accepted the connection and sent the name of the player (Player 1). The new client uses a unique port number to contact the server. This port number is used by the server to send the player name to the client.



```
Problems Debug Shell Coverage Console
YahtzeeServer [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (26 Nov 2019, 20:09:34)
Yahtzee server initialised at port 4321
Accepted a connection from /192.168.56.1:60392 at port 4321
Total players in game: 1
[client.Player@56a8fd25]
```

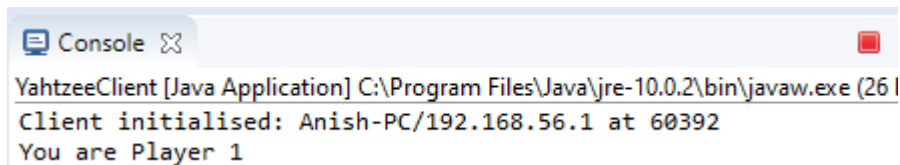
**Figure 5.2.2.** YahtzeeServer accepting a player connection from port 60392.



```
Problems Debug Shell Coverage Console
YahtzeeServer [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (26
Yahtzee server initialised at port 4321
Accepted a connection from /192.168.56.1:60392 at port 4321
Total players in game: 1
[client.Player@56a8fd25]
Accepted a connection from /192.168.56.1:60478 at port 4321
Total players in game: 2
[client.Player@56a8fd25, client.Player@541ef50d]
```

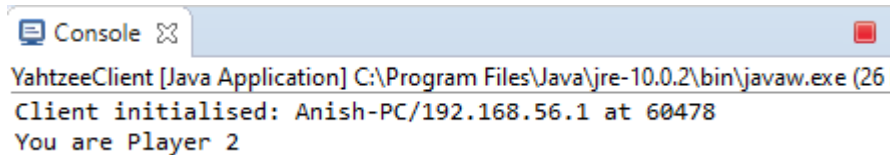
**Figure 5.2.3.** YahtzeeServer accepting another player connection from port 60478.

In figure 5.2.3, the server accepts two new players from ports 60392 and 60478. Therefore, now the game has two players in total. The players are stored in an array list; therefore, the players will have their turn in a round-robin (one after another) basis. The game will not start until there are 3 or more players present.



```
Console
YahtzeeClient [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (26 Nov 2019, 20:09:34)
Client initialised: Anish-PC/192.168.56.1 at 60392
You are Player 1
```

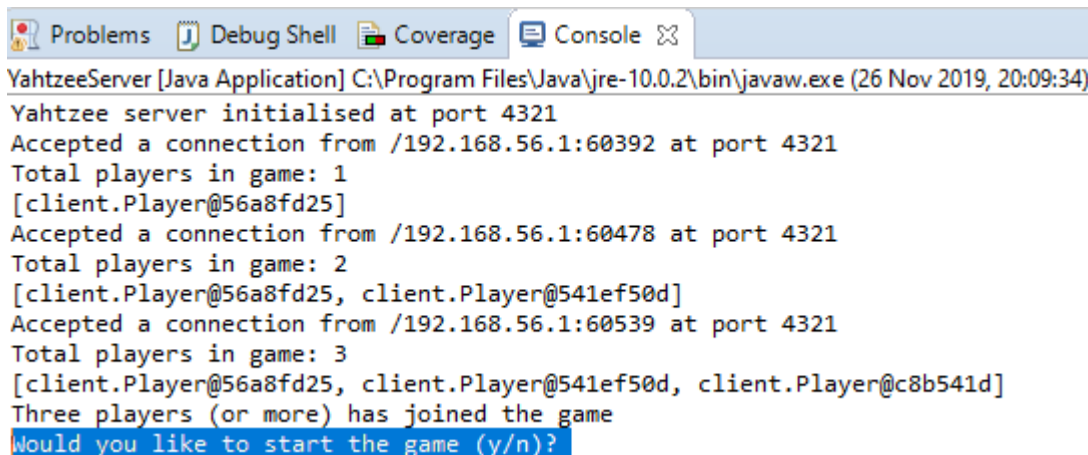
Figure 5.3.1. YahtzeeClient, player 1, receives its name (Player 1).



```
Console
YahtzeeClient [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (26 Nov 2019, 20:09:34)
Client initialised: Anish-PC/192.168.56.1 at 60478
You are Player 2
```

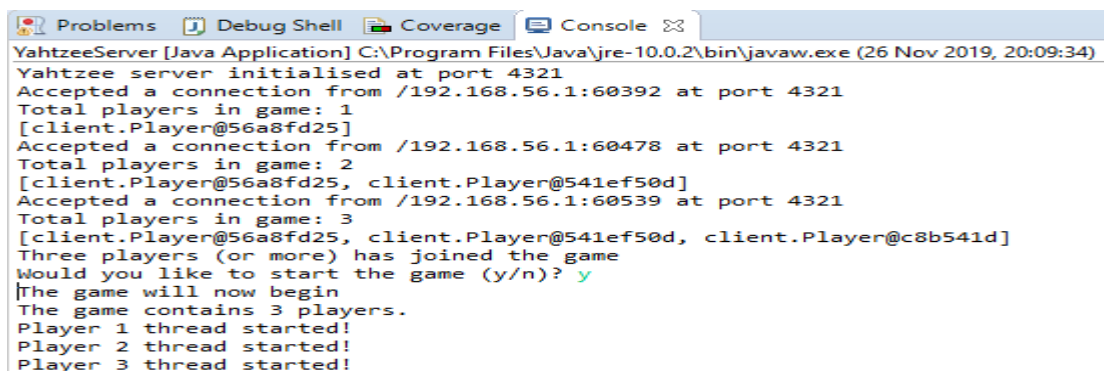
Figure 5.3.2. YahtzeeClient, player 2, receives its name (Player 2).

In figures 5.3.1 and 5.3.2, the two clients correctly received their player name from the server.



```
Problems Debug Shell Coverage Console
YahtzeeServer [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (26 Nov 2019, 20:09:34)
Yahtzee server initialised at port 4321
Accepted a connection from /192.168.56.1:60392 at port 4321
Total players in game: 1
[client.Player@56a8fd25]
Accepted a connection from /192.168.56.1:60478 at port 4321
Total players in game: 2
[client.Player@56a8fd25, client.Player@541ef50d]
Accepted a connection from /192.168.56.1:60539 at port 4321
Total players in game: 3
[client.Player@56a8fd25, client.Player@541ef50d, client.Player@c8b541d]
Three players (or more) has joined the game
Would you like to start the game (y/n)?
```

Figure 5.4.1. YahtzeeServer asking if the game should start or not.



```
Problems Debug Shell Coverage Console
YahtzeeServer [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (26 Nov 2019, 20:09:34)
Yahtzee server initialised at port 4321
Accepted a connection from /192.168.56.1:60392 at port 4321
Total players in game: 1
[client.Player@56a8fd25]
Accepted a connection from /192.168.56.1:60478 at port 4321
Total players in game: 2
[client.Player@56a8fd25, client.Player@541ef50d]
Accepted a connection from /192.168.56.1:60539 at port 4321
Total players in game: 3
[client.Player@56a8fd25, client.Player@541ef50d, client.Player@c8b541d]
Three players (or more) has joined the game
Would you like to start the game (y/n)? y
The game will now begin
The game contains 3 players.
Player 1 thread started!
Player 2 thread started!
Player 3 thread started!
```

Figure 5.4.2. YahtzeeServer receiving “y” causing the game to start.



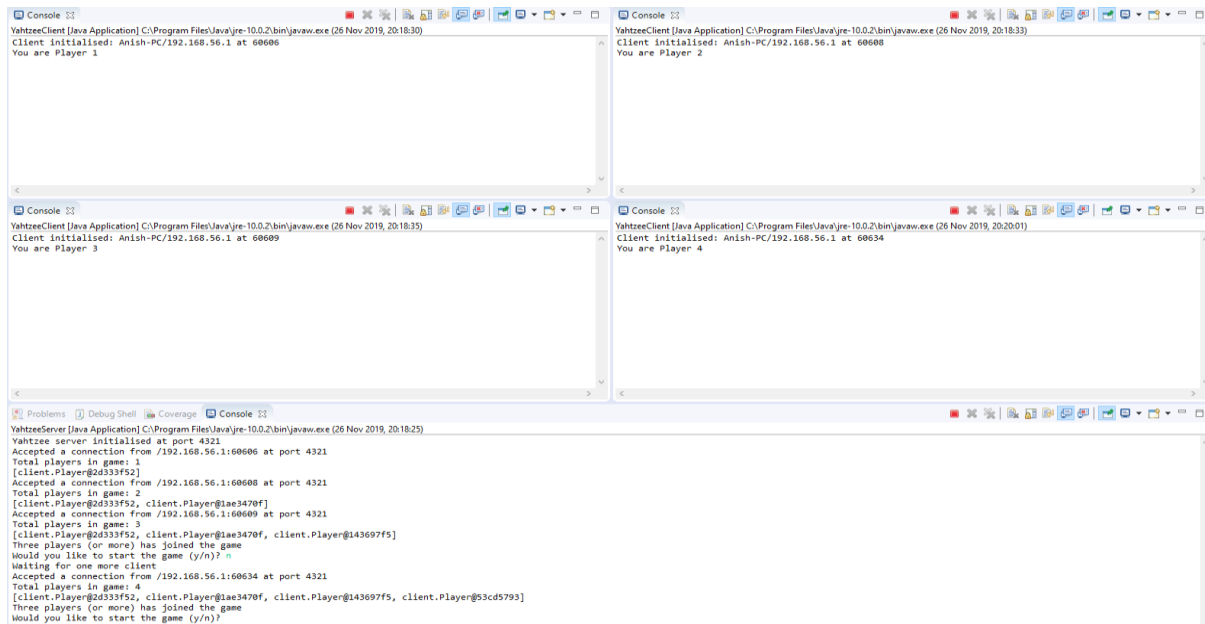


Figure 5.4.3. YahtzeeServer waiting for input after waiting for the fourth player.

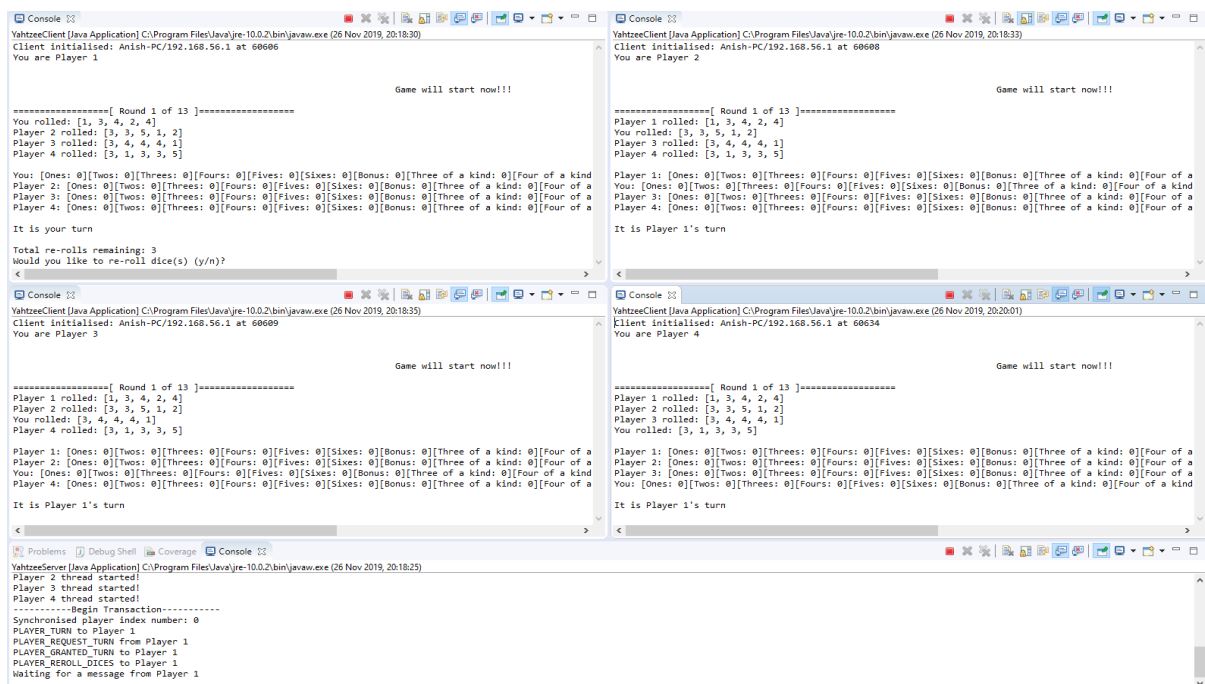


Figure 5.5.1. Players printing the messages sent by YahtzeeServer.

In figure 5.5.1, each player receives the welcome banner and scoreboard of all the players in the game and prints it to its console. The game starts off at round one, and it is player one's turn, and other players are waiting for player one to finish their turn.



```

Console
YahzeeClient [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (26 Nov 2019, 20:18:30)
Client initialised: Anish-PC\192.168.56.1 at 60606
You are Player 1

Game will start now!!!

===== Round 1 of 13 =====
You rolled: [1, 3, 4, 2, 4]
Player 2 rolled: [3, 3, 5, 1, 2]
Player 3 rolled: [3, 4, 4, 4, 1]
Player 4 rolled: [3, 1, 3, 3, 5]

You: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]
Player 2: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]
Player 3: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]
Player 4: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]

It is your turn

Total re-rolls remaining: 3
Would you like to re-roll dice(s) (y/n)?
<

Console
YahzeeClient [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (26 Nov 2019, 20:18:33)
Client initialised: Anish-PC\192.168.56.1 at 60608
You are Player 2

Game will start now!!!

===== Round 1 of 13 =====
Player 1 rolled: [1, 3, 4, 2, 4]
You rolled: [3, 3, 5, 1, 2]
Player 3 rolled: [3, 4, 4, 4, 1]
Player 4 rolled: [3, 1, 3, 3, 5]

Player 1: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]
You: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]
Player 3: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]
Player 4: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]

It is Player 1's turn

Console
YahzeeClient [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (26 Nov 2019, 20:18:35)
Client initialised: Anish-PC\192.168.56.1 at 60609
You are Player 3

Game will start now!!!

===== Round 1 of 13 =====
Player 1 rolled: [1, 3, 4, 2, 4]
Player 2 rolled: [3, 3, 5, 1, 2]
You rolled: [3, 4, 4, 4, 1]
Player 4 rolled: [3, 1, 3, 3, 5]

Player 1: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]
Player 2: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]
You: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]
Player 4: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]

It is Player 1's turn

Console
YahzeeClient [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (26 Nov 2019, 20:20:01)
Client initialised: Anish-PC\192.168.56.1 at 60634
You are Player 4

Game will start now!!!

===== Round 1 of 13 =====
Player 1 rolled: [1, 3, 4, 2, 4]
Player 2 rolled: [3, 3, 5, 1, 2]
Player 3 rolled: [3, 4, 4, 4, 1]
You rolled: [3, 1, 3, 3, 5]

Player 1: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]
Player 2: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]
Player 3: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]
You: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]

It is Player 1's turn

YahzeeServer [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (26 Nov 2019, 20:18:25)
Player 2 thread started!
Player 3 thread started!
Player 4 thread started!
-----Begin Transaction-----
Synchronised player index number: 0
PLAYER_TURN to Player 1
PLAYER_REQUEST_TURN from Player 1
PLAYER_GRANTED_TURN to Player 1
PLAYER_REROLL_DICES to Player 1
Waiting for a message from Player 1

```

**Figure 5.6.1.** Console output consisting of round number and scoreboard of each player.

In figure 5.6.1, I am testing the socket operations: read and write. The server writes messages to the socket of every player in the game. Then each player will read the messages from its socket and print it to its console. This seems to be working because each player receives the messages from the server.

```

Console
YahzeeClient [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (26 Nov 2019, 20:18:30)
Client initialised: Anish-PC\192.168.56.1 at 60606
You are Player 1

Game will start now!!!

===== Round 1 of 13 =====
You rolled: [1, 3, 4, 2, 4]
Player 2 rolled: [3, 3, 5, 1, 2]
Player 3 rolled: [3, 4, 4, 4, 1]
Player 4 rolled: [3, 1, 3, 3, 5]

You: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]
Player 2: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]
Player 3: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]
Player 4: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]

It is your turn

Total re-rolls remaining: 3
Would you like to re-roll dice(s) (y/n)?
<

Console
YahzeeClient [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (26 Nov 2019, 20:18:33)
Client initialised: Anish-PC\192.168.56.1 at 60608
You are Player 2

Game will start now!!!

===== Round 1 of 13 =====
Player 1 rolled: [1, 3, 4, 2, 4]
You rolled: [3, 3, 5, 1, 2]
Player 3 rolled: [3, 4, 4, 4, 1]
Player 4 rolled: [3, 1, 3, 3, 5]

Player 1: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]
You: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]
Player 3: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]
Player 4: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]

It is Player 1's turn

Console
YahzeeClient [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (26 Nov 2019, 20:18:35)
Client initialised: Anish-PC\192.168.56.1 at 60609
You are Player 3

Game will start now!!!

===== Round 1 of 13 =====
Player 1 rolled: [1, 3, 4, 2, 4]
Player 2 rolled: [3, 3, 5, 1, 2]
You rolled: [3, 4, 4, 4, 1]
Player 4 rolled: [3, 1, 3, 3, 5]

Player 1: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]
Player 2: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]
You: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]
Player 4: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]

It is Player 1's turn

Console
YahzeeClient [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (26 Nov 2019, 20:20:01)
Client initialised: Anish-PC\192.168.56.1 at 60634
You are Player 4

Game will start now!!!

===== Round 1 of 13 =====
Player 1 rolled: [1, 3, 4, 2, 4]
Player 2 rolled: [3, 3, 5, 1, 2]
Player 3 rolled: [3, 4, 4, 4, 1]
You rolled: [3, 1, 3, 3, 5]

Player 1: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]
Player 2: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]
Player 3: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]
You: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0]

It is Player 1's turn

YahzeeServer [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (26 Nov 2019, 20:18:25)
Player 2 thread started!
Player 3 thread started!
Player 4 thread started!
-----Begin Transaction-----
Synchronised player index number: 0
PLAYER_TURN to Player 1
PLAYER_REQUEST_TURN from Player 1
PLAYER_GRANTED_TURN to Player 1
PLAYER_REROLL_DICES to Player 1
Waiting for a message from Player 1

```

**Figure 5.7.1.** Player one's turn and input is available.

In figure 5.7.1, only player one is able to have their turn. This means player two, three, and four will have to wait until it is their turn. Also, at the bottom console, the server is waiting for a message from player one; therefore, the turn is synchronised in a round-robin fashion.

```

Console [1]
YahzeeClient [Java Application] C:\Program Files\Java\jre-10.0.2\bin\java.exe (26 Nov 2019, 20:18:30)
Would you like to re-roll dice(s) (y/n)? n
With your dice, you can score in the following: [Ones: 1][Twos: 2][Threes: 3][Fours: 0][Fives: 0][Sixes: 0][Three of a kind: 0][Four of a kind: 0][Full house: 0]
Enter 1 to select Ones
Enter 2 to select Twos
Enter 3 to select Threes
Enter 4 to select Fours
Enter 5 to select Fives
Enter 6 to select Sixes
Enter 7 to select Three of a kind
Enter 8 to select Four of a kind
Enter 9 to select Full House
Enter 10 to select Small Straight
Enter 11 to select Large Straight
Enter 12 to select Chance
Enter 13 to select Yahtzee
Enter option to score in: 4
Your turn is finished for this round, please wait until all the players have their turn

It is Player 2's turn

Console [2]
YahzeeClient [Java Application] C:\Program Files\Java\jre-10.0.2\bin\java.exe (26 Nov 2019, 20:18:33)
===== Round 1 of 13 =====
Player 1 rolled: [1, 3, 4, 2, 4]
You rolled: [3, 3, 5, 1, 2]
Player 3 rolled: [3, 4, 4, 4, 1]
Player 4 rolled: [3, 1, 3, 3, 5]

Player 1: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0][Full house: 0]
Player 2: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0][Full house: 0]
Player 3: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0][Full house: 0]
Player 4: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0][Full house: 0]

It is Player 1's turn
Player 1 can score in the following: [Ones: 1][Twos: 2][Threes: 3][Fours: 0][Fives: 0][Sixes: 0][Three of a kind: 0][Four of a kind: 0][Full house: 0]
Player 1 chose to score in Fours giving them 8 points
Player 1 finished re-rolling

It is your turn
Total re-rolls remaining: 3
Would you like to re-roll dice(s) (y/n)? n

Console [3]
YahzeeClient [Java Application] C:\Program Files\Java\jre-10.0.2\bin\java.exe (26 Nov 2019, 20:18:35)
===== Round 1 of 13 =====
Player 1 rolled: [1, 3, 4, 2, 4]
Player 2 rolled: [3, 3, 5, 1, 2]
You rolled: [3, 4, 4, 4, 1]
Player 3 rolled: [3, 1, 3, 3, 5]

Player 1: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0][Full house: 0]
Player 2: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0][Full house: 0]
Player 3: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0][Full house: 0]
Player 4: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0][Full house: 0]

It is Player 1's turn
Player 1 can score in the following: [Ones: 1][Twos: 2][Threes: 3][Fours: 0][Fives: 0][Sixes: 0][Three of a kind: 0][Four of a kind: 0][Full house: 0]
Player 1 chose to score in Fours giving them 8 points
Player 1 finished re-rolling

It is Player 2's turn

Console [4]
YahzeeClient [Java Application] C:\Program Files\Java\jre-10.0.2\bin\java.exe (26 Nov 2019, 20:18:35)
-----Begin Transaction-----
Synchronised player index number: 1
PLAYER_TURN to Player 2
PLAYER_REQUEST_TURN from Player 2
PLAYER_GRANTED_TURN to Player 2
PLAYER_REROLL_DICES to Player 2
Waiting for a message from Player 2
  
```

Figure 5.7.2. Player two unblocked, and other players are waiting for their turn.

```

Console [1]
YahzeeClient [Java Application] C:\Program Files\Java\jre-10.0.2\bin\java.exe (26 Nov 2019, 20:18:30)
Enter 8 to select Four of a kind
Enter 9 to select Full House
Enter 10 to select Small Straight
Enter 11 to select Large Straight
Enter 12 to select Chance
Enter 13 to select Yahtzee
Enter option to score in: 4
Your turn is finished for this round, please wait until all the players have their turn

It is Player 2's turn
Player 2 rolled and has 2 chances to re-roll
Player 2 chose 1 dice(s) to re-roll
Player 2 chose [4] dice index(es) to re-roll
After re-rolling, Player 2's new dice are [3, 3, 5, 1, 2]
Player 2 can score in the following: [Ones: 1][Twos: 2][Threes: 6][Fours: 0][Fives: 5][Sixes: 0][Three of a kind: 0][Four of a kind: 0][Full house: 0]
Player 2 chose to score in Threes giving them 6 points
Player 2 finished re-rolling

It is Player 3's turn

Console [2]
YahzeeClient [Java Application] C:\Program Files\Java\jre-10.0.2\bin\java.exe (26 Nov 2019, 20:18:33)
Would you like to re-roll dice(s) (y/n)? n
With your dice, you can score in the following: [Ones: 1][Twos: 2][Threes: 6][Fours: 0][Fives: 5][Sixes: 0][Three of a kind: 0][Four of a kind: 0][Full house: 0]
Enter 1 to select Ones
Enter 2 to select Twos
Enter 3 to select Threes
Enter 4 to select Fours
Enter 5 to select Fives
Enter 6 to select Sixes
Enter 7 to select Three of a kind
Enter 8 to select Four of a kind
Enter 9 to select Full House
Enter 10 to select Small Straight
Enter 11 to select Large Straight
Enter 12 to select Chance
Enter 13 to select Yahtzee
Enter option to score in: 1
Your turn is finished for this round, please wait until all the players have their turn

It is Player 3's turn

Console [3]
YahzeeClient [Java Application] C:\Program Files\Java\jre-10.0.2\bin\java.exe (26 Nov 2019, 20:18:35)
===== Round 1 of 13 =====
Player 1 rolled: [1, 3, 4, 2, 4]
Player 2 rolled: [3, 3, 5, 1, 2]
You rolled: [3, 4, 4, 4, 1]
Player 3 rolled: [3, 1, 3, 3, 5]

Player 1: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0][Full house: 0]
Player 2: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0][Full house: 0]
Player 3: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0][Full house: 0]
Player 4: [Ones: 0][Twos: 0][Threes: 0][Fours: 0][Fives: 0][Sixes: 0][Bonus: 0][Three of a kind: 0][Four of a kind: 0][Full house: 0]

It is Player 1's turn
Player 1 can score in the following: [Ones: 1][Twos: 2][Threes: 3][Fours: 0][Fives: 0][Sixes: 0][Three of a kind: 0][Four of a kind: 0][Full house: 0]
Player 1 chose to score in Fours giving them 8 points
Player 1 finished re-rolling

It is Player 2's turn
Player 2 rolled and has 2 chances to re-roll
Player 2 chose 1 dice(s) to re-roll
Player 2 chose [4] dice index(es) to re-roll
After re-rolling, Player 2's new dice are [3, 3, 5, 1, 2]
Player 2 can score in the following: [Ones: 1][Twos: 2][Threes: 6][Fours: 0][Fives: 5][Sixes: 0][Three of a kind: 0][Four of a kind: 0][Full house: 0]
Player 2 chose to score in Threes giving them 6 points
Player 2 finished re-rolling

It is your turn
Total re-rolls remaining: 3
Would you like to re-roll dice(s) (y/n)? n

Console [4]
YahzeeClient [Java Application] C:\Program Files\Java\jre-10.0.2\bin\java.exe (26 Nov 2019, 20:18:35)
-----Begin Transaction-----
Synchronised player index number: 2
PLAYER_TURN to Player 3
PLAYER_REQUEST_TURN from Player 3
PLAYER_GRANTED_TURN to Player 3
PLAYER_REROLL_DICES to Player 3
Waiting for a message from Player 3
  
```

Figure 5.7.3. Player three unblocked, and other players are waiting for their turn.

In figure 5.7.2, player one has had their turn, and it is player two's turn. This means only player two is unblocked and can interact with the server. The rest of the players are waiting for their turn. In figure 5.7.3, player three is unblocked and the rest of the players are waiting for their turn. The server updates each player what was scored, whose turn it is right now, whether the player rerolled dice or not, what category they scored in, etc, when they are waiting.

The image shows two IDE console windows side-by-side. The left window, titled 'VahzeeClient [Java Application]', displays the game state for Round 2 of 13. It shows the scores for four players: Player 1 (8), Player 2 (6), Player 3 (8), and Player 4 (9). It then prompts Player 1 to re-roll dice, showing the current dice (3, 2, 1, 5) and the number of re-rolls remaining (3). The right window, titled 'VahzeeServer [Java Application]', shows the server's response to Player 1's request to re-roll. It displays the current dice (3, 2, 1, 5) and the number of re-rolls remaining (3). Below the console windows, a 'Problems' window shows a list of messages sent between the client and server, including 'PLAYER\_GRANTED\_TURN to Player 1', 'PLAYER\_REQUEST\_REROLL\_DICES to Player 1', and 'PLAYER\_REQUEST\_REROLL\_CHANCES to Player 1'.

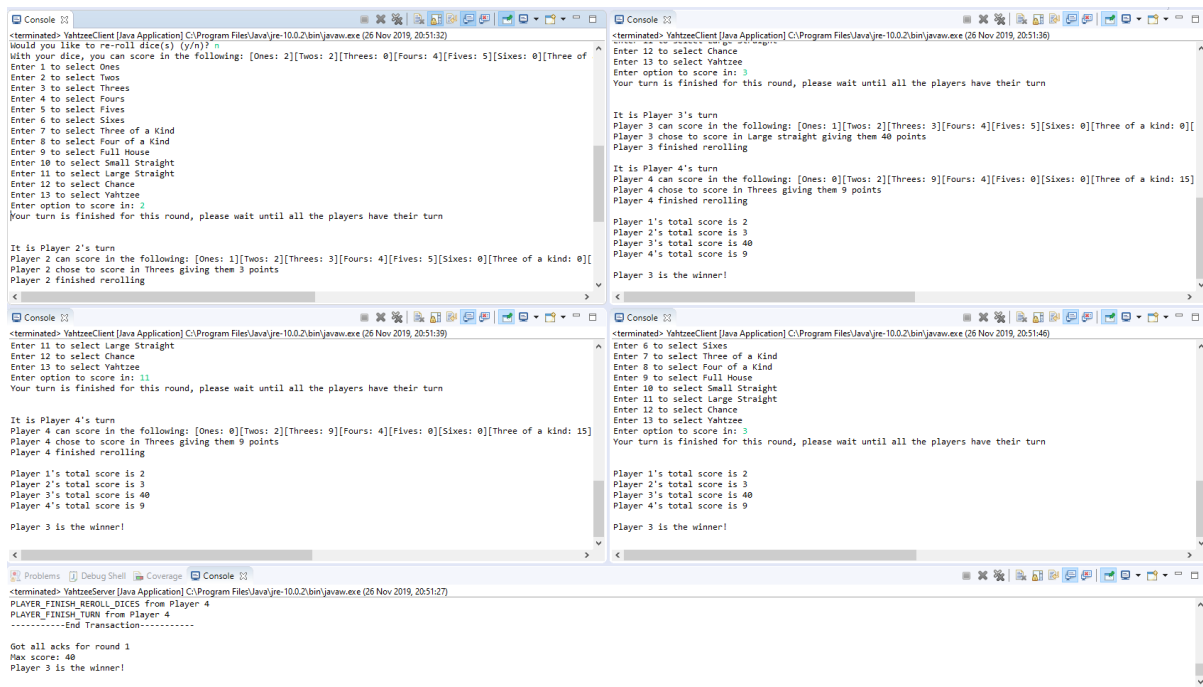
Figure 5.8.1. Player one rerolling dice.

In figure 5.8.1, player one rerolled dice. After rerolling dice, the server informed each player who are waiting, how many rerolls player one has remaining, how many dice the player decided to reroll, the dice indexes to reroll, and the new rerolled dice. Player one can still reroll, or they can choose a category to score in.

The image shows two IDE console windows side-by-side. The left window, titled 'VahzeeClient [Java Application]', displays the game state for Round 2 of 13. It shows the scores for four players: Player 1 (8), Player 2 (6), Player 3 (8), and Player 4 (9). It then prompts Player 1 to re-roll dice, showing the current dice (3, 2, 1, 5) and the number of re-rolls remaining (3). The right window, titled 'VahzeeServer [Java Application]', shows the server's response to Player 1's request to re-roll. It displays the current dice (3, 2, 1, 5) and the number of re-rolls remaining (3). Below the console windows, a 'Problems' window shows a list of messages sent between the client and server, including 'PLAYER\_GRANTED\_TURN to Player 1', 'PLAYER\_REQUEST\_REROLL\_DICES to Player 1', and 'PLAYER\_REQUEST\_REROLL\_CHANCES to Player 1'.

Figure 5.9.1. Player one choosing a category to score in.

In figure 5.9.1, player one was sent what category they could score in by the server. Also, a list of categories they have not selected is displayed. The player has entered 5 to score in Fives category for 5 points. After player one picked to score in Fives for 5 points, each player is informed what category player one chose and the amount of points they scored that round.



The figure consists of four screenshots of a Java console application, showing the progression of a Yahtzee game. The application is titled 'YahtzeeClient [Java Application]' and is running on 'C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (26 Nov 2019, 20:51:32)'. The game involves four players (1, 2, 3, 4) who take turns rolling dice and selecting a category to score. The console output shows the current state of the game, including the dice rolls, the selected category, and the resulting score. The game ends when all players have finished their turn, and the winner is announced. The final scores are: Player 1: 2, Player 2: 3, Player 3: 40, Player 4: 9. Player 3 is the winner.

```
<terminated> YahtzeeClient [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (26 Nov 2019, 20:51:32)
Would you like to re-roll dice(s) (y/n)? n
With your dice, you can score in the following: [Ones: 2][Twos: 2][Threes: 0][Fours: 4][Fives: 5][Sixes: 0][Three of a Kind: 0]
Enter 1 to select Ones
Enter 2 to select Twos
Enter 3 to select Threes
Enter 4 to select Fours
Enter 5 to select Fives
Enter 6 to select Sixes
Enter 7 to select Three of a Kind
Enter 8 to select Four of a Kind
Enter 9 to select Full House
Enter 10 to select Small Straight
Enter 11 to select Large Straight
Enter 12 to select Chance
Enter 13 to select Yahtzee
Enter option to score in: 3
Your turn is finished for this round, please wait until all the players have their turn

It is Player 2's turn
Player 2 can score in the following: [Ones: 1][Twos: 2][Threes: 3][Fours: 4][Fives: 5][Sixes: 0][Three of a Kind: 0]
Player 2 chose to score in Threes giving them 3 points
Player 2 finished reolling

It is Player 4's turn
Player 4 can score in the following: [Ones: 0][Twos: 2][Threes: 9][Fours: 4][Fives: 0][Sixes: 0][Three of a kind: 15]
Player 4 chose to score in Threes giving them 9 points
Player 4 finished reolling

Player 1's total score is 2
Player 2's total score is 3
Player 3's total score is 40
Player 4's total score is 9

Player 3 is the winner!
```

```
<terminated> YahtzeeClient [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (26 Nov 2019, 20:51:36)
Enter 12 to select Chance
Enter 13 to select Yahtzee
Enter option to score in: 3
Your turn is finished for this round, please wait until all the players have their turn

It is Player 3's turn
Player 3 can score in the following: [Ones: 1][Twos: 2][Threes: 3][Fours: 4][Fives: 5][Sixes: 0][Three of a kind: 0]
Player 3 chose to score in Large straight giving them 40 points
Player 3 finished reolling

It is Player 4's turn
Player 4 can score in the following: [Ones: 0][Twos: 2][Threes: 9][Fours: 4][Fives: 0][Sixes: 0][Three of a kind: 15]
Player 4 chose to score in Threes giving them 9 points
Player 4 finished reolling

Player 1's total score is 2
Player 2's total score is 3
Player 3's total score is 40
Player 4's total score is 9

Player 3 is the winner!
```

```
<terminated> YahtzeeClient [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (26 Nov 2019, 20:51:39)
Enter 11 to select Large Straight
Enter 12 to select Chance
Enter 13 to select Yahtzee
Enter option to score in: 11
Your turn is finished for this round, please wait until all the players have their turn

It is Player 4's turn
Player 4 can score in the following: [Ones: 0][Twos: 2][Threes: 9][Fours: 4][Fives: 0][Sixes: 0][Three of a kind: 15]
Player 4 chose to score in Threes giving them 9 points
Player 4 finished reolling

Player 1's total score is 2
Player 2's total score is 3
Player 3's total score is 40
Player 4's total score is 9

Player 3 is the winner!
```

```
<terminated> YahtzeeServer [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (26 Nov 2019, 20:51:27)
PLAYER_FINISH_REROLL_DICES from Player 4
PLAYER_FINISH_TURN from Player 4
-----End Transaction-----

Got all acks for round 1
Max score: 40
Player 3 is the winner!
```

**Figure 5.10.1.** Winner announcement.

In figure 5.10.1, I am testing the winner functionality. To do this, I have changed the number of rounds to 1. After one round, player 3 has the highest score and is the winner. The server broadcasts this information to all the players and ends the game.

## 6. Conclusions

Building this multiplayer Yahtzee game was complex. I underestimated the difficulty of this assignment and needed help a few times (from my lecturer) to understand synchronisation and sockets. After completing this assignment, I have learned how to build client-server applications in Java. I have also tackled concurrency which is a topic understood by senior developers!

## Appendix A: Code

### Dices

This class represents the dice in the game. This is used by the players when they want to roll dice, reroll dice, etc.

```
1. package client;
2.
3. import java.security.SecureRandom;
4. import java.util.Arrays;
5.
6. public class Dices {
7.     private SecureRandom engine;
8.     private int[] dicesRolled;
9.     private final int ROLL_UPPER_BOUND = 6;
10.
11.     Dices() {
12.         engine = new SecureRandom();
13.         dicesRolled = new int[5];
14.     }
15.
16.     private int rollDice() {
17.         return engine.nextInt(ROLL_UPPER_BOUND) + 1;
18.     }
19.
20.     public void rerollDices(int[] indexes) {
21.         for(int index : indexes) {
22.             dicesRolled[index - 1] = rollDice();
23.         }
24.     }
25.
26.     int[] rollDices() {
27.         dicesRolled = engine.ints(5, 1, ROLL_UPPER_BOUND).toArray();
28.         return dicesRolled;
29.     }
30.
31.     public int[] getRolledDices() {
32.         return dicesRolled;
33.     }
34.
35.     @Override
36.     public String toString() {
37.         return Arrays.toString(dicesRolled);
38.     }
39. }
```

## Player

Each player in the game is an instance of this class. Each player has an I/O streams, socket, dice, etc. The server can invoke the methods in this class to send message to the player, get the score of the player, etc.

```
1.  package client;
2.
3.  import score.ScoreCheck;
4.  import java.io.IOException;
5.  import java.io.ObjectInputStream;
6.  import java.io.ObjectOutputStream;
7.  import java.net.Socket;
8.  import java.util.Arrays;
9.
10. public class Player implements Runnable {
11.     private PlayerScoreBoard scoreBoard;
12.     private Socket socket;
13.     private ObjectOutputStream output;
14.     private ObjectInputStream input;
15.     private int playerNumber;
16.     private String playerName;
17.     private int[] dicesRolled;
18.     private Dices dices;
19.     private ScoreCheck scoreCheck;
20.     private TurnManager turnManager;
21.
22.     public Player(Socket socket) {
23.         this.socket = socket;
24.         this.scoreBoard = new PlayerScoreBoard();
25.         this.dices = new Dices();
26.         this.scoreCheck = new ScoreCheck(dices);
27.     }
28.
29.     @Override
30.     public void run() {
31.         System.out.println(playerName + " thread started!");
32.         boolean running = true;
33.         Thread t = new Thread(() -> {
34.             while(running) {
35.                 // Run the thread forever, and ever, and ever...,until the game ends
36.             }
37.         });
38.     }
39.
40.     public void setTurnManager(TurnManager turnManager) {
41.         this.turnManager = turnManager;
42.     }
43.
44.     public TurnManager getTurnManager() {
45.         return turnManager;
46.     }
47.
48.     public String getRolledDices() {
49.         dicesRolled = dices.rollDices();
50.         return Arrays.toString(dicesRolled);
51.     }
52.
53.     public void sendMessage(String message) {
54.         try {
55.             output.writeObject(message);
56.             output.flush();
57.         } catch (IOException e) {
58.             e.printStackTrace();
59.         }
60.     }
```

```
61.
62.     public String readMessage() {
63.         try {
64.             return (String) input.readObject();
65.         } catch(IOException | ClassNotFoundException e) {
66.             e.printStackTrace();
67.         }
68.         return null;
69.     }
70.
71.     public int[] readArrayMessage() {
72.         try {
73.             return (int[]) input.readObject();
74.         } catch(IOException | ClassNotFoundException e) {
75.             e.printStackTrace();
76.         }
77.         return null;
78.     }
79.
80.     public String getScoreBoard() {
81.         return scoreBoard.toString();
82.     }
83.
84.     public Socket getSocket() {
85.         return socket;
86.     }
87.
88.     public void setInputStream(ObjectInputStream input) {
89.         this.input = input;
90.     }
91.
92.     public void setOutputStream(ObjectOutputStream output) {
93.         this.output = output;
94.     }
95.
96.     public ObjectInputStream getInputStream() {
97.         return input;
98.     }
99.
100.    public ObjectOutputStream getOutputStream() {
101.        return output;
102.    }
103.
104.    public void setScore(ScoreType scoreType, int points) {
105.        scoreBoard.setScore(scoreType, points);
106.    }
107.
108.
109.    public void setPlayerName(String name) {
110.        this.playerName = name;
111.    }
112.
113.    public String getPlayerName() {
114.        return playerName;
115.    }
116.
117.
118.    public void setPlayerNumber(int num) {
119.        this.playerNumber = num;
120.    }
121.
122.    public int getPlayerNumber() {
123.        return playerNumber;
124.    }
125.
126.
```



```
127.     public int getTotalScore() {
128.         synchronized (this) {
129.             int bonusTotal = scoreBoard.getUpperSectionBonus();
130.             if (bonusTotal != 0)
131.                 scoreBoard.setScore(ScoreType.BONUS, bonusTotal);
132.             return scoreBoard.total();
133.         }
134.     }
135.
136.     public String getWhatCanBeScored() {
137.         return scoreCheck.getScorablePoints();
138.     }
139.
140.     public int getCategoryScoredPoint(String index) {
141.         return scoreCheck.getScorePoint(Integer.valueOf(index));
142.     }
143.
144.     public String getPlayerScoreBoard() {
145.         return scoreBoard.toString();
146.     }
147.
148.     public Dices getDices() {
149.         return dices;
150.     }
151.
152. }
```

## PlayerScoreBoard

Each player in the game will have a scoreboard. When a player chooses a category to score in, the server will update the player's scoreboard.

```
1.     package client;
2.
3.     import java.io.Serializable;
4.     import java.util.LinkedHashMap;
5.     import java.util.Map;
6.
7.     public class PlayerScoreBoard implements Serializable
8.     {
9.         private LinkedHashMap<ScoreType, Integer> playerScoreBoard;
10.        private String result;
11.
12.        PlayerScoreBoard() {
13.            playerScoreBoard = new LinkedHashMap<>();
14.            setScoreTypesInPlayerScoreBoard();
15.        }
16.
17.        private void setScoreTypesInPlayerScoreBoard() {
18.            final ScoreType[] SCORE_TYPES = ScoreType.values();
19.            for(int i = 0; i < SCORE_TYPES.length; i++) {
20.                playerScoreBoard.put(SCORE_TYPES[i], 0);
21.            }
22.        }
23.
24.        void setScore(ScoreType type, int value) {
25.            playerScoreBoard.put(type, value);
26.        }
27.
28.        // Function to test the (keys, values)
29.        private void formatBoardString() {
30.            StringBuilder formattedPlayerScoreBoard = new StringBuilder();
31.            for (Map.Entry<ScoreType, Integer> item : playerScoreBoard.entrySet()) {
```



```
32.     String scoreType = item.getKey().getName();
33.     String scorePoints = item.getValue().toString();
34.     final String SCORE_ROW = "[" + scoreType + ": " + scorePoints + "]";
35.     formattedPlayerScoreBoard.append(SCORE_ROW);
36. }
37. formattedPlayerScoreBoard.append("\n");
38. result = formattedPlayerScoreBoard.toString();
39. }
40.
41. public int total() {
42.     int playerTotalScore = 0;
43.     for(Map.Entry<ScoreType, Integer> item : playerScoreBoard.entrySet()) {
44.         int pointScored = item.getValue();
45.         playerTotalScore += pointScored;
46.     }
47.     return playerTotalScore;
48. }
49.
50. public synchronized int getUpperSectionBonus() {
51.     int playerTotalUpperScore = 0;
52.     int counter = 0;
53.     for(Map.Entry<ScoreType, Integer> item : playerScoreBoard.entrySet()) {
54.         if(counter == 6)
55.             break;
56.         int pointScored = item.getValue();
57.         playerTotalUpperScore += pointScored;
58.         counter++;
59.     }
60.     // Upper score bonus check
61.     return playerTotalUpperScore >= 63 ? 35 : 0;
62. }
63.
64.
65. @Override
66. public String toString() {
67.     formatBoardString();
68.     return result;
69. }
70. }
```

## ScoreType

This class defines the scoring categories in the game.

```
1. package client;
2.
3. import java.util.ArrayList;
4.
5. public enum ScoreType {
6.     ONES("Ones"),
7.     TWOS("Twos"),
8.     THREES("Threes"),
9.     FOURS("Fours"),
10.    FIVES("Fives"),
11.    SIXES("Sixes"),
12.    BONUS("Bonus"),
13.
14.    THREE_OF_A_KIND("Three of a kind"),
15.    FOUR_OF_A_KIND("Four of a kind"),
16.    FULL_HOUSE("Full house"),
17.    SMALL_STRAIGHT("Small straight"),
18.    LARGE_STRAIGHT("Large straight"),
```

```
19.  CHANCE("Chance"),
20.  YAHTZEE("Yahtzee");
21.
22.
23.  public static ArrayList<ScoreType> SCORE_TYPES;
24.
25.  final static int STARTING_UPPER_CATEGORY_INDEX= ONES.ordinal();
26.  final static int ENDING_UPPER_CATEGORY_INDEX = SIXES.ordinal();
27.
28.  final static int STARTING_LOWER_CATEGORY_INDEX= THREE_OF_A_KIND.ordinal();
29.  final static int ENDING_LOWER_CATEGORY_INDEX = YAHTZEE.ordinal();
30.
31.  static {
32.      initialiseScores();
33.  }
34.
35.  private static void initialiseScores() {
36.      SCORE_TYPES = new ArrayList<>();
37.      addUpperCategories();
38.      addLowerCategories();
39.  }
40.
41.  public static ScoreType getScoreType(String index) {
42.      return SCORE_TYPES.get(Integer.valueOf(index) - 1);
43.  }
44.
45.  private static void addUpperCategories() {
46.      final ScoreType[] SCORES = ScoreType.values();
47.      for(int i = STARTING_UPPER_CATEGORY_INDEX; i <= ENDING_UPPER_CATEGORY_INDEX; i++) {
48.          SCORE_TYPES.add(SCORES[i]);
49.      }
50.  }
51.
52.  private static void addLowerCategories() {
53.      final ScoreType[] SCORES = ScoreType.values();
54.      for(int i = STARTING_LOWER_CATEGORY_INDEX; i <= ENDING_LOWER_CATEGORY_INDEX; i++) {
55.          SCORE_TYPES.add(SCORES[i]);
56.      }
57.  }
58.
59.  private String name;
60.
61.  ScoreType(String name) {
62.      this.name = name;
63.  }
64.
65.  public String getName() {
66.      return name;
67.  }
68. }
```

## TurnManager

This class is used to synchronise the turns in the game. This allows only one player to obtain the lock at a time; therefore, only one player can have their turn at a time.

```
1.  package client;
2.
3.  public class TurnManager {
4.      // These values are shared amongst the players
5.      private boolean isAccessing;
6.
7.      private int playerIndex;
8.      private final int MAX_NUMBER_OF_PLAYERS;
9.
10.     public TurnManager(int playersInGame) {
11.         this.isAccessing = false;
12.         this.MAX_NUMBER_OF_PLAYERS = playersInGame;
13.     }
14.     private synchronized void increasePlayerIndex() {
15.         playerIndex = playerIndex == MAX_NUMBER_OF_PLAYERS - 1 ? 0 : ++playerIndex;
16.     }
17.
18.     public synchronized int getPlayerTurn() {
19.         return playerIndex;
20.     }
21.
22.     public synchronized void requestTurn() throws InterruptedException {
23.         while(isAccessing) {
24.             wait();
25.         }
26.         isAccessing = true;
27.     }
28.
29.     public synchronized void releaseTurn() {
30.         isAccessing = false;
31.         notifyAll();
32.         increasePlayerIndex();
33.     }
34. }
```

## YahtzeeClient

Players contact the server using this class. This class will be used the players to play the game.

```
1.  package client;
2.
3.  import constants.ServerOperationConstants;
4.
5.  import java.io.IOException;
6.  import java.io.ObjectInputStream;
7.  import java.io.ObjectOutputStream;
8.  import java.net.InetAddress;
9.  import java.net.Socket;
10. import java.util.ArrayList;
11. import java.util.InputMismatchException;
12. import java.util.Scanner;
13.
14. public class YahtzeeClient extends Thread {
15.     private ObjectInputStream input;
16.     private ObjectOutputStream output;
17.     private Scanner inputKeyboard;
18.     private int rerollChances;
19.     private boolean gameHasEnded;
20.     private ArrayList<Integer> categoriesToScore;
21.
22.     private YahtzeeClient() throws IOException {
23.         InetAddress localhost = InetAddress.getLocalHost();
24.         Socket socketConnection = new Socket(localhost, 4321);
25.         System.out.println("Client initialised: " + InetAddress.getLocalHost() + " at " + socketConnection.g
           etLocalPort());
26.         output = new ObjectOutputStream(socketConnection.getOutputStream());
27.         input = new ObjectInputStream(socketConnection.getInputStream());
28.         inputKeyboard = new Scanner(System.in);
29.         categoriesToScore = new ArrayList<>();
30.         start();
31.     }
32.
33.     @Override
34.     public void run() {
35.         try {
36.             printWelcomeMessage();
37.             while(!gameHasEnded) {
38.                 // Receive a message from server
39.                 respondToServerOperations();
40.             }
41.         } catch(IOException | ClassNotFoundException e) {
42.             e.printStackTrace();
43.         }
44.     }
45.
46.     private void printWelcomeMessage() {
47.         String messageFromServer = null;
48.         try {
49.             messageFromServer = (String) input.readObject();
50.         } catch (IOException | ClassNotFoundException e) {
51.             e.printStackTrace();
52.         }
53.         System.out.println(messageFromServer);
54.     }
55.
56.     private void respondToServerOperations() throws IOException, ClassNotFoundException {
57.         while(!gameHasEnded) {
58.             String operationFromServer = (String) input.readObject();
59.             switch(operationFromServer) {
60.                 case ServerOperationConstants.ROUND_NUMBER:
```

```

61.         String roundNumber = (String) input.readObject();
62.         System.out.println("=====[ Round " + roundNumber + " of 13" + " ]====="
        =====");
63.         break;
64.         case ServerOperationConstants.OTHER_PLAYER_TURN:
65.             String playerTurnName = (String) input.readObject();
66.             System.out.println("It is " + playerTurnName + "'s turn");
67.             break;
68.         case ServerOperationConstants.PLAYER_TURN:
69.             output.writeObject(ServerOperationConstants.PLAYER_REQUEST_TURN);
70.             String replyFromServer = (String) input.readObject();
71.             if(replyFromServer.equals(ServerOperationConstants.PLAYER_GRANTED_TURN))
72.                 System.out.println("It is your turn");
73.             break;
74.         case ServerOperationConstants.PLAYER_FINISH_TURN:
75.             informServerTurnFinished();
76.             break;
77.         case ServerOperationConstants.PLAYER_ROLL_DICES:
78.             String diceRolled = (String) input.readObject();
79.             System.out.println("You rolled: " + diceRolled);
80.             break;
81.         case ServerOperationConstants.PLAYER_REROLL_DICES:
82.             // Read the number of reroll chances the player has currently
83.             rerollChances = 3;
84.             rerollDices();
85.             break;
86.         case ServerOperationConstants.OTHER_PLAYER_REROLL_DICES:
87.         case ServerOperationConstants.OTHER_PLAYER_CHOSE_SCORING_CATEGORY:
88.         case ServerOperationConstants.OTHER_PLAYER_GET_WHAT_CAN_BE_SCORED:
89.             String messageFromServer = (String) input.readObject();
90.             System.out.println(messageFromServer);
91.             break;
92.         case ServerOperationConstants.GAME_END:
93.             String messageFromServer2 = (String) input.readObject();
94.             if(messageFromServer2.equals(ServerOperationConstants.WINNER_ANNOUNCEMENT)) {
95.                 String winnerName = (String) input.readObject();
96.                 System.out.println(winnerName);
97.             }
98.             gameHasEnded = true;
99.             break;
100.        default:
101.            System.out.println(operationFromServer);
102.            break;
103.    }
104. }
105. }
106.
107. private boolean chooseRerollOption() {
108.     String optionChosen;
109.     boolean INVALID_CHOICE;
110.     do {
111.         System.out.println("\nTotal re-rolls remaining: " + rerollChances);
112.         System.out.print("Would you like to re-roll dice(s) (y/n)? ");
113.         optionChosen = inputKeyboard.nextLine();
114.         INVALID_CHOICE = !((optionChosen.equals("y") || optionChosen.equals("n")));
115.         if(INVALID_CHOICE)
116.             System.out.println("Please enter y or n.");
117.     } while(INVALID_CHOICE);
118.     return optionChosen.equals("y");
119. }
120.
121. private void rerollDices() {
122.     do {
123.         final boolean OPTION_CHOSEN_IS_Y = chooseRerollOption();
124.         if(OPTION_CHOSEN_IS_Y && rerollChances >= 1) {
125.             // Send a message to the server to inform the player rerolled dices

```

```
126.         try {
127.             output.writeObject(ServerOperationConstants.PLAYER_REROLLED_DICES);
128.         } catch (IOException e) {
129.             e.printStackTrace();
130.         }
131.
132.         // Decrease the number of rerolling chances
133.         rerollChances--;
134.
135.         // The number of dices to reroll
136.         final int NUMBER_OF_DICE_TO_REROLL = askNumberOfDicesToReroll();
137.
138.         try {
139.             output.writeObject(ServerOperationConstants.PLAYER_NUMBER_OF_DICES_REROLLED);
140.             output.writeObject(String.valueOf(NUMBER_OF_DICE_TO_REROLL));
141.         } catch (IOException e) {
142.             e.printStackTrace();
143.         }
144.
145.         int[] dicesIndexToReroll = askRerollInputChoices(NUMBER_OF_DICE_TO_REROLL);
146.         // Send the index of the dices to change
147.         try {
148.             // Array must be sent in int, not string
149.             output.writeObject(dicesIndexToReroll);
150.         } catch (IOException e) {
151.             e.printStackTrace();
152.         }
153.         System.out.println("Total number of rerolls remaining: " + rerollChances);
154.
155.         // Print the new rerolled dices
156.         try {
157.             String message = (String) input.readObject();
158.             if(message.equals(ServerOperationConstants.PLAYER_NEW_REROLLED_DICE)) {
159.                 System.out.println("After rerolling, your dices are " + input.readObject());
160.             }
161.         } catch (IOException | ClassNotFoundException e) {
162.             e.printStackTrace();
163.         }
164.
165.         //System.out.println(Arrays.toString(dicesRolled));
166.         System.out.println();
167.     } else {
168.         break;
169.     }
170. } while(rerollChances >= 1);
171. // Once done, reset the re-roll chances to 3 for the next round
172. informServerTurnFinished();
173. System.out.print("Your turn is finished for this round, please wait until all the players have their
turn\n\n");
174. //inputKeyboard.nextLine();
175. }
176.
177. // This method handles the bad input exception
178. private int readInteger() {
179.     int enteredInteger = 0;
180.     try {
181.         enteredInteger = inputKeyboard.nextInt();
182.         inputKeyboard.nextLine(); // Remove the '\n' character
183.     } catch (InputMismatchException e) {
184.         inputKeyboard.nextLine(); // Remove the bad input
185.         System.out.println();
186.     }
187.     return enteredInteger;
188. }
189.
190. private int[] askRerollInputChoices(int numberOfDicesToReroll) {
```

```
191.     int[] dicesNumberToReroll = new int[numberOfDicesToReroll];
192.     for(int i = 1; i <= numberOfDicesToReroll; ) {
193.         System.out.print("Select a dice: ");
194.         final int DICE_INDEX = readInteger();
195.         final boolean INVALID_CHOICE = (DICE_INDEX <= 0 || DICE_INDEX >= 6);
196.         if(INVALID_CHOICE) {
197.             System.out.println("Please enter an integer in [1,5] range");
198.         } else {
199.             dicesNumberToReroll[i - 1] = DICE_INDEX;
200.             i++;
201.         }
202.     }
203.     return dicesNumberToReroll;
204. }
205.
206. private int askNumberOfDicesToReroll() {
207.     int numberOfDiceToReroll;
208.     boolean INVALID_CHOICE;
209.     do {
210.         System.out.print("Enter the number of dice(s) to reroll: ");
211.         numberOfDiceToReroll = readInteger();
212.         INVALID_CHOICE = (numberOfDiceToReroll <= 0 || numberOfDiceToReroll >= 6);
213.         if(INVALID_CHOICE)
214.             System.out.println("Please enter an integer in [1,5] range");
215.     } while(INVALID_CHOICE);
216.     return numberOfDiceToReroll;
217. }
218.
219. private void askServerWhatCanBeScored() {
220.     try {
221.         output.writeObject(ServerOperationConstants.PLAYER_GET_WHAT_CAN_BE_SCORED);
222.         System.out.print("With your dice, you can score in the following: ");
223.         String scoreBoard = (String) input.readObject();
224.         System.out.println(scoreBoard);
225.     } catch (IOException | ClassNotFoundException e) {
226.         e.printStackTrace();
227.     }
228. }
229.
230. private void chooseScoringCategory(int categoryChosen) {
231.     try {
232.         final String CATEGORY_INDEX = String.valueOf(categoryChosen);
233.         output.writeObject(ServerOperationConstants.PLAYER_CHOOSE_SCORING_CATEGORY);
234.         output.writeObject(CATEGORY_INDEX);
235.     } catch (IOException e) {
236.         e.printStackTrace();
237.     }
238. }
239.
240. private void printCategoryToScoreInMessage(int number, String message) {
241.     if(!categoriesToScore.contains(number))
242.         System.out.println(message);
243. }
244.
245. private void printUpperSectionScoringCategories() {
246.     printCategoryToScoreInMessage(1, "Enter 1 to select Ones");
247.     printCategoryToScoreInMessage(2, "Enter 2 to select Twos");
248.     printCategoryToScoreInMessage(3, "Enter 3 to select Threes");
249.     printCategoryToScoreInMessage(4, "Enter 4 to select Fours");
250.     printCategoryToScoreInMessage(5, "Enter 5 to select Fives");
251.     printCategoryToScoreInMessage(6, "Enter 6 to select Sixes");
252. }
253.
254. private void printLowerSectionScoringCategories() {
255.     printCategoryToScoreInMessage(7, "Enter 7 to select Three of a Kind");
256.     printCategoryToScoreInMessage(8, "Enter 8 to select Four of a Kind");
```

```
257.     printCategoryToScoreInMessage(9, "Enter 9 to select Full House");
258.     printCategoryToScoreInMessage(10, "Enter 10 to select Small Straight");
259.     printCategoryToScoreInMessage(11, "Enter 11 to select Large Straight");
260.     printCategoryToScoreInMessage(12, "Enter 12 to select Chance");
261.     printCategoryToScoreInMessage(13, "Enter 13 to select Yahtzee");
262. }
263.
264. private void printScoringCategories() {
265.     printUpperSectionScoringCategories();
266.     printLowerSectionScoringCategories();
267. }
268.
269. private void chooseWhatToScoreIn() {
270.     boolean INVALID_CHOICE;
271.     printScoringCategories();
272.     do {
273.         System.out.print("Enter option to score in: ");
274.         int numberToScoreIn = readInteger();
275.         boolean numberToScoreInHasBeenScoredAlready = categoriesToScore.contains(numberToScoreIn);
276.         INVALID_CHOICE = (numberToScoreIn <= 0 || numberToScoreIn >= 14) || numberToScoreInHasBeenScored
Already;
277.         if(numberToScoreInHasBeenScoredAlready) {
278.             System.out.println("This scoring category has already been scored in");
279.         } else if(INVALID_CHOICE) {
280.             System.out.println("Please enter an integer in [1,13] range");
281.         } else {
282.             categoriesToScore.add(numberToScoreIn);
283.             chooseScoringCategory(numberToScoreIn);
284.         }
285.     } while(INVALID_CHOICE);
286.
287. }
288.
289. private void informServerTurnFinished() {
290.     try {
291.         // What can be scored
292.         askServerWhatCanBeScored();
293.         chooseWhatToScoreIn();
294.         output.writeObject(ServerOperationConstants.PLAYER_FINISH_REROLL_DICES);
295.         output.writeObject(ServerOperationConstants.PLAYER_FINISH_TURN);
296.     } catch (IOException e) {
297.         e.printStackTrace();
298.     }
299. }
300.
301. public static void main(String[] args) throws IOException {
302.     new YahtzeeClient();
303. }
304. }
```



## ServerOperationConstants

This class represents the messages exchanged by the server and players. Each constant in this interface defines an operation.

```
1. package constants;
2.
3. public interface ServerOperationConstants {
4.     String GAME_END = "GAME_END";
5.     String PLAYER_REQUEST_TURN = "PLAYER_REQUEST_TURN";
6.     String PLAYER_GRANTED_TURN = "PLAYER_GRANTED_TURN";
7.     String PLAYER_FINISH_TURN = "PLAYER_FINISH_TURN";
8.     String ROUND_NUMBER = "ROUND_NUMBER";
9.     String PLAYER_TURN = "PLAYER_TURN";
10.    String OTHER_PLAYER_TURN = "OTHER_PLAYER_TURN";
11.    String PLAYER_ROLL_DICES = "PLAYER_ROLL_DICES";
12.    String OTHER_PLAYER_ROLL_DICES = "OTHER_PLAYER_ROLL_DICES";
13.    String PLAYER_REROLL_DICES = "PLAYER_REROLL_DICES";
14.    String PLAYER_REQUEST_REROLL_CHANCES = "PLAYER_REQUEST_REROLL_CHANCES";
15.    String PLAYER_REROLLED_DICES = "PLAYER_REQUEST_REROLL_CHANCES";
16.    String PLAYER_FINISH_REROLL_DICES = "PLAYER_FINISH_REROLL_DICES";
17.    String OTHER_PLAYER_REROLL_DICES = "OTHER_PLAYER_REROLL_DICES";
18.    String PLAYER_GET_WHAT_CAN_BE_SCORED = "PLAYER_GET_WHAT_CAN_BE_SCORED";
19.    String OTHER_PLAYER_GET_WHAT_CAN_BE_SCORED = "OTHER_PLAYER_GET_WHAT_CAN_BE_SCORED";
20.    String PLAYER_GET_SCOREBOARD = "PLAYER_GET_SCOREBOARD";
21.    String PLAYER_CHOOSE_SCORING_CATEGORY = "PLAYER_CHOOSE_SCORING_CATEGORY";
22.    String OTHER_PLAYER_CHOSE_SCORING_CATEGORY = "OTHER_PLAYER_CHOSE_SCORING_CATEGORY";
23.    String PLAYER_SET_SCORE_IN_CATEGORY = "PLAYER_SET_SCORE_IN_CATEGORY";
24.    String OTHER_PLAYER_SET_SCORE_IN_CATEGORY = "OTHER_PLAYER_SET_SCORE_IN_CATEGORY";
25.    String WINNER_ANNOUNCEMENT = "WINNER_ANNOUNCEMENT";
26.    String PLAYER_NUMBER_OF_DICES_REROLLED = "PLAYER_NUMBER_OF_DICES_REROLLED";
27.    String PLAYER_NEW_REROLLED_DICE = "PLAYER_NEW_REROLLED_DICE";
28. }
```

## ScoreCheck

Players in the game use this class to check what category they can score in with their dice. This class checks the upper and lower sections of the scoring categories.

```
1. package score;
2.
3. import client.Dices;
4.
5. import java.util.ArrayList;
6. import java.util.Collections;
7. import java.util.stream.Collectors;
8. import java.util.stream.IntStream;
9.
10. public class ScoreCheck {
11.     private Dices diceRolled;
12.     private StringBuilder scoreablePointsString;
13.     private ArrayList<Integer> pointsScored;
14.
15.     public ScoreCheck(Dices dices) {
16.         this.diceRolled = dices;
17.     }
18.
19.     public String getScorablePoints() {
20.         scoreablePointsString = new StringBuilder();
21.         pointsScored = new ArrayList<>();
22.         checkUpperSections();
23.         checkLowerSections();
24.         return scoreablePointsString.toString();
25.     }
}
```

```

26.
27.     private static int checkSingleUpperSectionCombination(int[] dices, int score) {
28.         return IntStream.of(dices).filter(x -> x == score).sum();
29.     }
30.
31.     // Upper Sections
32.     private synchronized void checkUpperSections() {
33.         int[] dices = diceRolled.getRolledDices();
34.         final int ONES_SUM = checkSingleUpperSectionCombination(dices, 1);
35.         final int TWOS_SUM = checkSingleUpperSectionCombination(dices, 2);
36.         final int THREES_SUM = checkSingleUpperSectionCombination(dices, 3);
37.         final int FOURS_SUM = checkSingleUpperSectionCombination(dices, 4);
38.         final int FIVES_SUM = checkSingleUpperSectionCombination(dices, 5);
39.         final int SIXES_SUM = checkSingleUpperSectionCombination(dices, 6);
40.
41.         storePoints(ONES_SUM, TWOS_SUM, THREES_SUM, FOURS_SUM, FIVES_SUM, SIXES_SUM);
42.
43.         scoreablePointsString.append("[").append("Ones: ").append(ONES_SUM).append("]");
44.         scoreablePointsString.append("[").append("Twos: ").append(TWOS_SUM).append("]");
45.         scoreablePointsString.append("[").append("Threes: ").append(THREES_SUM).append("]");
46.         scoreablePointsString.append("[").append("Fours: ").append(FOURS_SUM).append("]");
47.         scoreablePointsString.append("[").append("Fives: ").append(FIVES_SUM).append("]");
48.         scoreablePointsString.append("[").append("Sixes: ").append(SIXES_SUM).append("]");
49.         //System.out.println("Bonus: " + BONUS_SUM);
50.     }
51.
52.
53.     private synchronized void checkLowerSections() {
54.         int[] dices = diceRolled.getRolledDices();
55.         final int TOK = getThreeOfAKindScore(dices);
56.         final int FOK = getFourOfAKindScore(dices);
57.         final int FULL_HOUSE = getFullHouseScore(dices);
58.         int SMALL_STRAIGHT = getSmallStraightScore(dices);
59.         int LARGE_STRAIGHT = getLargeStraightScore(dices);
60.         int CHANCE = getChanceScore(dices);
61.         int YAHTZEE = getYahtzeeScore(dices);
62.
63.         storePoints(TOK, FOK, FULL_HOUSE, SMALL_STRAIGHT, LARGE_STRAIGHT, CHANCE, YAHTZEE);
64.
65.         scoreablePointsString.append("[").append("Three of a kind: ").append(TOK).append("]");
66.         scoreablePointsString.append("[").append("Four of a kind: ").append(FOK).append("]");
67.         scoreablePointsString.append("[").append("Full house: ").append(FULL_HOUSE).append("]");
68.         scoreablePointsString.append("[").append("Small straight: ").append(SMALL_STRAIGHT).append("]");
69.         scoreablePointsString.append("[").append("Large straight: ").append(LARGE_STRAIGHT).append("]");
70.         scoreablePointsString.append("[").append("Chance: ").append(CHANCE).append("]");
71.         scoreablePointsString.append("[").append("Yahtzee: ").append(YAHTZEE).append("]");
72.     }
73.
74.     private void storePoints(int ... scores) {
75.         for(final int score : scores) {
76.             pointsScored.add(score);
77.         }
78.     }
79.
80.     public int getScorePoint(int index) {
81.         return pointsScored.get(index - 1);
82.     }
83.
84.     // Very important the array is sorted before it is returned to the user
85.     private static ArrayList<Integer> getDicesArrayList(int[] dicesRolled) {
86.         return IntStream.of(dicesRolled)
87.             .sorted()
88.             .boxed()
89.             .collect(Collectors.toCollection(ArrayList::new));
90.     }
91.

```

```

92. // Working
93. private static int getThreeOfAKindScore(int[] dices) {
94.     ArrayList<Integer> dicesRolledArrayList = getDicesArrayList(dices);
95.     for(Integer numberRolled : dicesRolledArrayList) {
96.         final boolean HAS_THREE_OF_A_KIND = Collections.frequency(dicesRolledArrayList, numberRolled) >=
3;
97.         if(HAS_THREE_OF_A_KIND)
98.             return IntStream.of(dices).sum();
99.     }
100.    return 0;
101. }
102.
103. // Working
104. private static int getFourOfAKindScore(int[] dices) {
105.     ArrayList<Integer> dicesRolledArrayList = getDicesArrayList(dices);
106.     for(Integer numberRolled : dicesRolledArrayList) {
107.         final boolean HAS_FOUR_OF_A_KIND = Collections.frequency(dicesRolledArrayList, numberRolled) >=
4;
108.         if(HAS_FOUR_OF_A_KIND)
109.             return IntStream.of(dices).sum();
110.     }
111.    return 0;
112. }
113.
114. // Working
115. private static int getFullHouseScore(int[] dices) {
116.     ArrayList<Integer> dicesRolledArrayList = getDicesArrayList(dices);
117.     final Integer NUMBER_WITH_THREE_OF_A_KIND = dicesRolledArrayList.stream().filter(x -
> Collections.frequency(dicesRolledArrayList, x) >= 3).findFirst().orElse(0);
118.     if(NUMBER_WITH_THREE_OF_A_KIND != 0) {
119.         dicesRolledArrayList.removeIf(x -> x.equals(NUMBER_WITH_THREE_OF_A_KIND));
120.         if(dicesRolledArrayList.size() == 2 && (dicesRolledArrayList.get(0).equals(dicesRolledArrayList.
get(1)))) {
121.             return 25;
122.         }
123.     }
124.    return 0;
125. }
126.
127. private static ArrayList<Integer> getArrayListInRange(int start, int end) {
128.    return IntStream.rangeClosed(start, end).boxed().collect(Collectors.toCollection(ArrayList::new));
129. }
130.
131. private static int getSmallStraightScore(int[] dices) {
132.     ArrayList<Integer> dicesRolledArrayList = getDicesArrayList(dices);
133.     ArrayList<Integer> combinationOne = getArrayListInRange(1, 4); // [1,2,3,4]
134.     ArrayList<Integer> combinationTwo = getArrayListInRange(2, 5); // [2,3,4,5]
135.     ArrayList<Integer> combinationThree = getArrayListInRange(3, 6); // [3,4,5,6]
136.     final boolean CONTAINS_A_COMBINATION = dicesRolledArrayList.containsAll(combinationOne) || dicesRoll
edArrayList.containsAll(combinationTwo) || dicesRolledArrayList.containsAll(combinationThree);
137.    return CONTAINS_A_COMBINATION ? 30 : 0;
138. }
139.
140. private static int getLargeStraightScore(int[] dices) {
141.     ArrayList<Integer> dicesRolledArrayList = getDicesArrayList(dices);
142.     ArrayList<Integer> combinationOne = getArrayListInRange(1, 5); // [1,2,3,4,5]
143.     ArrayList<Integer> combinationTwo = getArrayListInRange(2, 6); // [2,3,4,5,6]
144.     final boolean CONTAINS_A_COMBINATION = dicesRolledArrayList.containsAll(combinationOne) || dicesRoll
edArrayList.containsAll(combinationTwo);
145.    return CONTAINS_A_COMBINATION ? 40 : 0;
146. }
147.
148. private static int getChanceScore(int[] dices) {
149.    return IntStream.of(dices).sum();
150. }
151.

```

```
152.     private static int getYahtzeeScore(int[] dices) {
153.         ArrayList<Integer> dicesRolledArrayList = getDicesArrayList(dices);
154.         for(Integer numberRolled : dicesRolledArrayList) {
155.             final boolean HAS_YAHTZEE = Collections.frequency(dicesRolledArrayList, numberRolled) == 5;
156.             if(HAS_YAHTZEE)
157.                 return 50;
158.         }
159.         return 0;
160.     }
161. }
```

## Broadcast

The server uses this class to send messages to the players in the game. This class is also used to announce the winner or draw before the game ends.

```
1.  package server;
2.
3.  import client.Player;
4.  import constants.ServerOperationConstants;
5.
6.  import java.util.ArrayList;
7.  import java.util.Collections;
8.  import java.util.Comparator;
9.  import java.util.List;
10. import java.util.function.Predicate;
11. import java.util.stream.Collectors;
12.
13. public class Broadcast {
14.     private List<Player> players;
15.
16.     Broadcast() {}
17.
18.     Broadcast(List<Player> players) {
19.         this.players = Collections.synchronizedList(players);
20.     }
21.
22.     private synchronized void broadcastMessage(String message) {
23.         for(final Player player: players) {
24.             player.sendMessage(message);
25.         }
26.     }
27.
28.     synchronized void broadcastMessage(Player currentPlayer, String message) {
29.         final String PLAYER_NAME = currentPlayer.getPlayerName();
30.         for(final Player player: players) {
31.             if(!player.getPlayerName().equals(PLAYER_NAME))
32.                 player.sendMessage(message);
33.         }
34.     }
35.
36.     private synchronized void broadcastDicesRolledMessage(String message) {
37.         for(final Player player: players) {
38.             final String PLAYER_ROLLED_DICES = message.replaceAll(player.getPlayerName(), "You");
39.             player.sendMessage(PLAYER_ROLLED_DICES);
40.         }
41.     }
42.
43.     synchronized void broadcastPlayersRoundNumber(int roundNumber) {
44.         broadcastMessage(ServerOperationConstants.ROUND_NUMBER);
45.         final String ROUND_NUMBER = String.valueOf(roundNumber);
46.         broadcastMessage(ROUND_NUMBER);
47.     }
```

```
48.
49.     synchronized void broadcastDicesRolledByPlayers(String dicesRolled) {
50.         broadcastDicesRolledMessage(dicesRolled);
51.     }
52.
53.     void broadcastPlayerScoreTotal() {
54.         StringBuilder scoresString = new StringBuilder();
55.         for(Player player : players)
56.             scoresString.append(player.getPlayerName()).append("'s total score is ").append(player.getTotalScore()).append("\n");
57.         broadcastMessage(scoresString.toString());
58.     }
59.
60.     synchronized void declareWinner() {
61.         // Inform them the game has finished
62.         broadcastMessage(ServerOperationConstants.GAME_END);
63.
64.         // Players sorted by their score (highest to smallest)
65.         List<Player> playerSortedByScores = players.stream()
66.             .sorted(Comparator.comparing(Player::getTotalScore).reversed())
67.             .collect(Collectors.toCollection(ArrayList::new));
68.         // The highest score achieved by a player
69.         final int HIGHEST_SCORE = playerSortedByScores.get(0).getTotalScore();
70.         final Predicate<Player> SAME_SCORE_AS_TOP_PLAYER = player -
71.             > player.getTotalScore() == HIGHEST_SCORE;
72.         // Obtain the winner
73.         // There can be a draw in the game
74.         List<Player> winner = playerSortedByScores.stream()
75.             .filter(SAME_SCORE_AS_TOP_PLAYER)
76.             .collect(Collectors.toCollection(ArrayList::new));
77.
78.         System.out.println("Max score: " + HIGHEST_SCORE);
79.
80.         broadcastMessage(ServerOperationConstants.WINNER_ANNOUNCEMENT);
81.
82.         if(winner.size() >= 2) {
83.             String namesOfPlayer = winner.stream().map(Player::getPlayerName).collect(Collectors.joining(",
84.             ));
85.             System.out.println("There is a draw between " + announcePlayersWithDrawScore(namesOfPlayer));
86.             broadcastMessage("There is a draw between " + announcePlayersWithDrawScore(namesOfPlayer));
87.         } else {
88.             System.out.println(winner.get(0).getPlayerName() + " is the winner!");
89.             broadcastMessage(winner.get(0).getPlayerName() + " is the winner!");
90.         }
91.         System.exit(0);
92.     }
93.     private synchronized String announcePlayersWithDrawScore(String names) {
94.         String replace = ", ";
95.         String replacement = ", and ";
96.         int start = names.lastIndexOf(replace);
97.         return names.substring(0, start) +
98.             replacement +
99.             names.substring(start + replace.length()) + ".";
100.     }
101.
102. }
```

## ServerOperation

The server uses this class to perform specific operations, such as welcoming the players and allowing them to have their turn.

```

1. package server;
2.
3. import client.Player;
4. import client.ScoreType;
5. import constants.ServerOperationConstants;
6.
7. import java.util.Arrays;
8. import java.util.List;
9.
10. class ServerOperation {
11.     private List<Player> players;
12.     private Broadcast broadcast;
13.
14.     ServerOperation(List<Player> players) {
15.         this.players = players;
16.         this.broadcast = new Broadcast(players);
17.     }
18.
19.     void welcomePlayers() {
20.         for(Player player : players) {
21.             player.sendMessage("\n\n\t\t\t\t\t\t\t\tGame will start now!!!\t\t\t\t\t\t\t");
22.         }
23.     }
24.
25.     private String getScoreBoardOfAllPlayers() {
26.         // Create a StringBuilder to store the score board of the players
27.         StringBuilder scoreBoardOfAllPlayers = new StringBuilder();
28.         for(Player player : players) {
29.             final String RESULT = player.getPlayerName() + ": " + player.getPlayerScoreBoard();
30.             scoreBoardOfAllPlayers.append(RESULT);
31.         }
32.         return scoreBoardOfAllPlayers.toString();
33.     }
34.
35.     String getPlayersDicesRolled() {
36.         StringBuilder result = new StringBuilder();
37.         for(Player player : players) {
38.             result.append(player.getPlayerName()).append(" rolled: ").append(player.getRolledDices()).append
39. ("\\n");
40.         }
41.         return result.toString();
42.     }
43.
44.     void sendPlayersScoreBoard() {
45.         // Send all the players of the scores
46.         for(Player player : players) {
47.             String scoreBoard = getScoreBoardOfAllPlayers();
48.             String formattedScoreBoard = scoreBoard.replaceAll(player.getPlayerName(), "You");
49.             player.sendMessage(formattedScoreBoard);
50.         }
51.     }
52.
53.     void grantPlayerTurn(Player player) {
54.         player.sendMessage(ServerOperationConstants.PLAYER_TURN);
55.         System.out.println(ServerOperationConstants.PLAYER_TURN + " to " + player.getPlayerName()); // Inform
56. m player that it is their turn
57.         final String REPLY_FROM_CLIENT = player.readMessage();
58.         System.out.println(REPLY_FROM_CLIENT + " from " + player.getPlayerName()); // This is the respon
59. se from server
60.         if (REPLY_FROM_CLIENT.equals(ServerOperationConstants.PLAYER_REQUEST_TURN)) {

```

```

58.         player.sendMessage(ServerOperationConstants.PLAYER_GRANTED_TURN);
59.         System.out.println(ServerOperationConstants.PLAYER_GRANTED_TURN + " to " + player.getPlayerName(
    ));
60.         letPlayerRerollDice(player);
61.     }
62. }
63.
64. private void letPlayerRerollDice(Player player) {
65.     boolean rerollingDices = true;
66.     int rerollChances = 3;
67.
68.     player.sendMessage(ServerOperationConstants.PLAYER_REROLL_DICES);
69.     System.out.println(ServerOperationConstants.PLAYER_REROLL_DICES + " to " + player.getPlayerName());
70.
71.     while(rerollingDices) {
72.         System.out.println("Waiting for a message from " + player.getPlayerName());
73.         String messageFromPlayer = player.readMessage();
74.         System.out.println(messageFromPlayer + " from " + player.getPlayerName());
75.         switch (messageFromPlayer) {
76.             case ServerOperationConstants.PLAYER_FINISH_REROLL_DICES:
77.                 broadcast.broadcastMessage(player, player.getPlayerName() + " finished rerolling\n");
78.                 rerollingDices = false;
79.                 break;
80.             case ServerOperationConstants.PLAYER_REROLLED_DICES:
81.                 rerollChances--;
82.                 broadcast.broadcastMessage(player, player.getPlayerName() + " rerolled and has " + rerol
    lChances + " chances to reroll");
83.                 break;
84.             case ServerOperationConstants.PLAYER_NUMBER_OF_DICES_REROLLED:
85.                 rerollDice(player);
86.                 break;
87.             case ServerOperationConstants.PLAYER_GET_WHAT_CAN_BE_SCORED:
88.                 displayWhatCanBeScored(player);
89.                 break;
90.             case ServerOperationConstants.PLAYER_FINISH_TURN:
91.                 System.out.println(messageFromPlayer);
92.                 break;
93.             case ServerOperationConstants.PLAYER_CHOOSE_SCORING_CATEGORY:
94.                 updateCategoryChosen(player);
95.                 break;
96.         }
97.     }
98. }
99.
100. private void rerollDice(Player player) {
101.     String numberOfDicesToReroll = player.readMessage(); // Get the number of dice to reroll
102.     int[] dicesIndexToReroll = player.readArrayMessage(); // Get the dice indexes to change
103.     player.getDices().rerollDices(dicesIndexToReroll); // Reroll dice by passing the indexes
104.     broadcast.broadcastMessage(player, player.getPlayerName() + " chose " + numberOfDicesToReroll + " di
    ce(s) to reroll");
105.     broadcast.broadcastMessage(player, player.getPlayerName() + " chose " + Arrays.toString(dicesIndexTo
    Reroll) + " dice index(es) to reroll");
106.
107.     // Obtain the new rerolled dices
108.     final String NEW_REROLLED_DICES = Arrays.toString(player.getDices().getRolledDices());
109.     // Send the player the newly rerolled dices
110.     player.sendMessage(ServerOperationConstants.PLAYER_NEW_REROLLED_DICE);
111.     player.sendMessage(NEW_REROLLED_DICES);
112.     // Send the rerolled dices to every players
113.     broadcast.broadcastMessage(player, ServerOperationConstants.OTHER_PLAYER_REROLL_DICES);
114.     broadcast.broadcastMessage(player, "After rerolling, " + player.getPlayerName() + "'s new dices are
    " + NEW_REROLLED_DICES);
115. }
116.
117. private void displayWhatCanBeScored(Player player) {

```

```
118.     final String WHAT_CAN_BE_SCORED = player.getWhatCanBeScored();
119.     player.sendMessage(WHAT_CAN_BE_SCORED);
120.     // Inform other players what the user can score in
121.     broadcast.broadcastMessage(player, ServerOperationConstants.OTHER_PLAYER_GET_WHAT_CAN_BE_SCORED);
122.     broadcast.broadcastMessage(player, player.getPlayerName() + " can score in the following: " + WHAT_C
    AN_BE_SCORED);
123. }
124.
125. private void updateCategoryChosen(Player player) {
126.     String categoryChosenByPlayer = player.readMessage();
127.     int pointScored = player.getCategoryScoredPoint(categoryChosenByPlayer);
128.     final String NAME_OF_CATEGORY_CHOSEN = ScoreType.getScoreType(categoryChosenByPlayer).getName();
129.     player.setScore(ScoreType.getScoreType(categoryChosenByPlayer), pointScored);
130.     updatePlayersCategoryChosen(player, NAME_OF_CATEGORY_CHOSEN, pointScored);
131. }
132.
133. private void updatePlayersCategoryChosen(Player player, String categoryChosen, int pointScored) {
134.     System.out.println("Scored point: " + pointScored);
135.     System.out.println(player.getPlayerName() + " picked to score in " + categoryChosen);
136.     System.out.println("Player's new scoreboard: " + player.getScoreBoard());
137.     broadcast.broadcastMessage(player, ServerOperationConstants.OTHER_PLAYER_CHOSE_SCORING_CATEGORY);
138.     broadcast.broadcastMessage(player, player.getPlayerName() + " chose to score in " + categoryChosen +
    " giving them " + pointScored + " points");
139. }
140. }
```

## YahtzeeServer

This class represents the server in the game.

```
1. package server;
2.
3. import client.Player;
4. import client.TurnManager;
5. import constants.ServerOperationConstants;
6.
7. import java.io.IOException;
8. import java.io.ObjectInputStream;
9. import java.io.ObjectOutputStream;
10. import java.net.ServerSocket;
11. import java.net.Socket;
12. import java.util.ArrayList;
13. import java.util.Collections;
14. import java.util.List;
15. import java.util.Scanner;
16.
17. public class YahtzeeServer extends Thread {
18.     private ServerSocket socket;
19.     private int playerNumber = 0;
20.     private Scanner inputKeyboard;
21.
22.     // The players
23.     private final List<Player> players;
24.     private String startGameOrNotOption = "";
25.     private boolean startGame;
26.     private Broadcast broadcast;
27.     private ServerOperation serverOperation;
28.     private TurnManager turnManager;
29.
30.     private YahtzeeServer(int portNumber) throws IOException {
31.         // Handle only up to 50 connection requests
32.         this.socket = new ServerSocket(portNumber, 50);
33.         System.out.println("Yahtzee server initialised at port " + portNumber);
```



```
34.     this.inputKeyboard = new Scanner(System.in);
35.     this.players = Collections.synchronizedList(new ArrayList<>());
36.     start();
37. }
38.
39. @Override
40. public void run() {
41.     runServer();
42. }
43.
44. private void runServer() {
45.     while(true) {
46.         try {
47.             handlePlayerConnectionRequests();
48.             serverOperation = new ServerOperation(players);
49.             broadcast = new Broadcast(players);
50.             serverOperation.welcomePlayers();
51.             startGame();
52.         } catch (IOException e) {
53.             e.printStackTrace();
54.             break;
55.         }
56.     }
57. }
58.
59. private void handlePlayerConnectionRequests() throws IOException {
60.     while(true) {
61.         if(playerNumber >= 3 && !startGame) {
62.             startGame = askStartGameOrNot();
63.             if(startGame) {
64.                 System.out.println("The game will now begin");
65.                 System.out.println("The game contains " + playerNumber + " players.");
66.                 break;
67.             } else {
68.                 System.out.println("Waiting for one more client");
69.                 startGameOrNotOption = "";
70.             }
71.         }
72.         acceptPlayerConnection();
73.     }
74. }
75.
76. private void acceptPlayerConnection() throws IOException {
77.     Player player = new Player(socket.accept());
78.
79.     Socket connectionSocket = player.getSocket();
80.     if(!startGame)
81.         System.out.println("Accepted a connection from " + connectionSocket.getInetAddress() + ":" + connectionSocket.getPort() + " at port " + connectionSocket.getLocalPort());
82.     else
83.         System.out.println("Game has started, but a client has been denied to join the game from" + connectionSocket.getInetAddress() + ":" + connectionSocket.getPort() + " at port " + connectionSocket.getLocalPort());
84.
85.     ObjectOutputStream outputStream = new ObjectOutputStream(connectionSocket.getOutputStream());
86.     ObjectInputStream inputStream = new ObjectInputStream(connectionSocket.getInputStream());
87.
88.     // If there are enough players and the game has started
89.     // decline the connection requests
90.     if(playerNumber >= 3 && startGame) {
91.         String message = "Cannot join game, as game is currently being played. Please try again later.";
92.
93.         outputStream.writeObject(message);
94.     } else {
95.         players.add(player);
96.         // Write the player number to the client
```

```
96.         players.get(playerNumber).setInputStream(inputStream);
97.         players.get(playerNumber).setOutputStream(outputStream);
98.         players.get(playerNumber).setPlayerNumber(playerNumber + 1);
99.         players.get(playerNumber).setPlayerName("Player " + (playerNumber + 1));
100.        String message = "You are Player " + (++playerNumber);
101.        outputStream.writeObject(message);
102.        System.out.println("Total players in game: " + playerNumber);
103.    }
104.    System.out.println(players);
105. }
106.
107. private boolean askStartGameOrNot() {
108.     System.out.println("Three players (or more) has joined the game");
109.     while(!(startGameOrNotOption.equalsIgnoreCase("n") || startGameOrNotOption.equals("y"))) {
110.         System.out.print("Would you like to start the game (y/n)? ");
111.         startGameOrNotOption = inputKeyboard.nextLine();
112.     }
113.     return startGameOrNotOption.equals("y");
114. }
115.
116. private void initialiseSharedObject() {
117.     turnManager = new TurnManager(players.size());
118.     // Start the thread of each player
119.     startPlayers();
120. }
121.
122. private void startPlayers() {
123.     synchronized (players) {
124.         for(Player player : players) {
125.             player.setTurnManager(turnManager);
126.             Thread t = new Thread(player);
127.             t.start();
128.             try {
129.                 t.join();
130.             } catch (InterruptedException e) {
131.                 e.printStackTrace();
132.             }
133.         }
134.     }
135. }
136.
137. private synchronized void startGame() {
138.     int totalAcks = 0;
139.     initialiseSharedObject();
140.     for(int roundNumber = 1; roundNumber <= 13; ) {
141.         // Send the round number to all the players
142.         broadcast.broadcastPlayersRoundNumber(roundNumber);
143.         broadcast.broadcastDicesRolledByPlayers(serverOperation.getPlayersDicesRolled());
144.         serverOperation.sendPlayersScoreBoard();
145.
146.         while(true) {
147.             synchronized(players) {
148.                 // Get the index of whose turn it is now
149.                 int playerTurnIndex = turnManager.getPlayerTurn();
150.
151.                 // The player who will communicate with the server
152.                 Player player = players.get(playerTurnIndex);
153.
154.                 // Inform each player whose turn it is right now
155.                 broadcast.broadcastMessage(player, ServerOperationConstants.OTHER_PLAYER_TURN);
156.                 broadcast.broadcastMessage(player, player.getPlayerName());
157.
158.                 System.out.println("-----Begin Transaction-----");
159.                 System.out.println("Synchronised player index number: " + turnManager.getPlayerTurn());
160.
161.                 // Let the player have their turn
```

```
161.         // Lock to prevent from multiple threads accessing and modifying the player index
162.         try {
163.             player.getTurnManager().requestTurn();
164.             serverOperation.grantPlayerTurn(player);
165.             player.getTurnManager().releaseTurn();
166.         } catch (InterruptedException e) {
167.             e.printStackTrace();
168.         }
169.
170.         // The user sends the PLAYER_FINISH_TURN message to the server
171.         String messageFromClient = player.readMessage();
172.         System.out.println(messageFromClient + " from " + player.getPlayerName());
173.         if (messageFromClient.equals(ServerOperationConstants.PLAYER_FINISH_TURN)) {
174.             totalAcks++;
175.             System.out.println("-----End Transaction-----\n");
176.         }
177.
178.         if (totalAcks == players.size()) {
179.             System.out.println("Got all acks for round " + roundNumber);
180.             totalAcks = 0;
181.             broadcast.broadcastPlayerScoreTotal();
182.             roundNumber++;
183.             break;
184.         }
185.     }
186. }
187. }
188. // Inform each player who the winner is
189. broadcast.declareWinner();
190. }
191.
192. public static void main(String[] args) throws IOException {
193.     new YahtzeeServer(4321);
194. }
195. }
```