Yael Romero
109210768
CSE 310 Rock-Paper-Scissors Documentation

Overview:

The entirety of the project specifications for Rock-Paper-Scissors has been implemented. This includes user login and establishing a TCP connection with the server, game start, game play, timeout, player statistics, and logout. The program successfully handles timeouts, input errors (if the syntax is incorrect for a certain command), and ties. All commands are implemented as well.

User Documentation:

First, the server code should be started. It will be waiting for a player to begin a game. Next, the client code should be started. The user will be welcomed and shown basic rules of the game.

The user will then be prompted to either view the help menu or login.
- If the user types in "**help**", the user will be shown all of the supported commands in the game and the proper syntax for each one, along with a brief description of what it does. A user can only view the help menu before they log in.
- If the user chooses to log in, the user must type "**login**" followed by his/her user id. An example of this is "login yael". If the user types only "login" or other incorrect input, the client will tell the user to check his/her syntax and will allow him/her to try again.

Once the user is successfully logged in, the server will send a message to the client saying that the user has been logged in. An example of this is "yael has been logged in". This will be announced at the client. After this, the server automatically starts the game. It announces it by displaying "The game has now officially begun!" The server will also send a message to the client describing that the server has chosen its object to throw and that the user has 5 seconds to make his/her move. The client will display this announcement. The client will prompt the user to either make a move or logout.
- If the user takes longer than 5 seconds to respond, a timeout will occur, the client will send a message to the server, and the server will announce that the game has been timed out. This will be displayed at the client. The game is over at this point and the server starts another one immediately. The counter for timed out games increases, as well as the counter for total number of games.
- If the user responds within 5 seconds with the correct syntax (which in this implementation means inputting "**r**", "**s**", or "**p**"), the server will announce the outcome of the game along with the objects the user and the server threw. This will be displayed at the client.
- If the user inputs an invalid move, the client will send a message to the server saying it is invalid, and the server will again choose an object to throw. The server then sends a "That is not

a valid move!" message along with a declaration that the server has chosen its object, which is displayed by the client and the client will prompt the user for a new input.
- If the outcome is a tie, the server will announce a tie and the objects the user and the server threw. This will be displayed at the client. At this point, the server randomly chooses another object to throw and will announce when it has chosen. It will tell the user to throw an object within 5 seconds. The client will display this announcement. Again, the client prompts the user to either make a move or logout.
- If the outcome is not a tie, the server will announce whether the user won or lost, and the objects the user and the server threw. This will be displayed at the client. The counter for number of player wins increases appropriately (not at all if you lost), and the counter for total number of games will increase.

At any point in the game after logging in, the client will give the user a choice to either throw an object or logout. The syntax for logging out is simply "**logout**". This sends a logout request to the server. The server then sends the logout confirmation and statistics of the player session to the client and the client displays them. These statistics include: the total number of games played, the total number of games won, and the total number of timed-out games. The client connection is then closed. The server waits for another player, and displays "Waiting for a player…".

System Documentation

Client code:
Global variables:

- An array of valid commands called commands
- A string for the welcome message called welcome
- A string to display for the help menu called helpline
- A string detailing the rules called rules
- A string for the message to send a rock to the server, called rockMessage
- A string for the message to send paper to the server, called paperMessage
- A string for the message to send scissors to the server, called scissorsMessage
- A string for the message to send a logout request to the server, called logoutMessage
- A string for the message to send a timeout message to the server, called timeoutMessage
- A string for the message to send an invalid message to the server, called invalid
- A string to display for syntax errors, called checkSyntax.
- An integer to set the timeout value, called TIMEOUT.

The class TimeoutException is a custom exception class made to be raised when a timeout occurs. The timeout method specific what happens when a timeout occurs; it raises the TimeoutException.

There are two major components to the main method in the client code. The first is the portion that validates the user login and the second is the game play portion.

For the user login validation, there is a boolean value called validLogin that is used to test whether the login is valid or not. Then, there is a while loop that runs while the login is invalid. The loop asks the user if he/she would like to see the help menu or login and if he/she wishes to login, it checks his/her input for the correct syntax. Error messages are shown to the user for incorrect syntax. The client keeps asking for input from the user until it gets a valid login. Once the user enters correct syntax for login, the client sends a LOGIN message to the server. The login message is constructed as follows: "**LOGIN user_id_here HTTP/1.1"**. The client then receives the login confirmation from the server and prints it.

For the game play portion, there is a boolean value called end that indicates whether the session is over. There is a while loop that runs while the session is still active. The client prompts the user to either throw an object or logout. The client checks for correct input at this stage. When the user inputs an invalid move, the client sends an invalid move message constructed as "**INVALID HTTP/1.1"** to the server. The client receives and displays the server's response. If the user times out, the client sends a message constructed as "**TIMEOUT HTTP/1.1"** to the server. The client receives and displays the server's response. Otherwise, a message that specifies the player's move is sent to the server.

If the player chooses to throw rock, the message is constructed as: "**ROCK HTTP/1.1"**. If the player chooses to throw paper, the message is constructed as: "**PAPER HTTP/1.1"**. If the player chooses to throw scissors, the message is constructed as: "**SCISSORS HTTP/1.1"**. If the player chooses to logout, the message is constructed as "**LOGOUT HTTP/1.1"**. Once the logout request is send, player statistics are received and displayed on the client and the socket is then closed.

Server code:
Global variables:

- lower and upper, used for random number generation
- An array of valid moves in the game called moves
- An array of moves valid in the session called choices
- Counters W, G, and O used to keep track of game statistics
- A string to announce the start of a new game, called newGame
- Strings used to announce when a server has selected an object, called serverDone and serverDoneRepeat
- A string for the message to send a timeout to the client, called timeOut
- A string for the message to send invalid input to the client, called invalid

There are two functions in the server code. The first is called getObjectName, which takes a string as an argument, and the main.

getObjectName(move) is used to map the move passed to the function to the name of an object in the game. For example, if you call getObjectName("R"), "Rock" will be returned.

The main function is where the game logic occurs. The server socket is created and bound to a port. First, there is code for receiving the user id from the client and then sending a log in confirmation to the client. The login confirmation message is constructed as "**user_id has been logged in. \n | HTTP/1.1 200 OK"**. Then, the game begins. There is a boolean value called repeat to keep track of whether or not the server needs to re-throw, and there is another boolean value called restart to keep track of when the logout request is received. A while loop runs as long as restart is set to true. Inside the loop, the game starts, and a boolean variable called end is set to false. End keeps track of the end of a game.

From here, there is another while loop that runs while the game has not ended. In this loop, the game occurs. The server makes its move by randomly choosing between rock, paper, and scissors and sends a message to the client describing that the player has 5 seconds to respond. If the player responds within 5 seconds, the code compares the user input to the server's move and determines the outcome of the game accordingly. There are cases for what happens in case of a tie, in case of a win, and in case of a loss, and in case of an invalid input. There is another case for what happens when the user takes longer than 5 seconds to respond. If the user input was a logout request, the server sends a logout confirmation and statistics to the client and the connection is closed.

Messages defined:

- In case of a tie: "**It is a tie. You and the server both threw " + getObjectName(playerMove) + ". Try again! | HTTP/1.1 200 OK**"
- In case of a timeout: "**The game has been timed out.\n | HTTP/1.1 200 OK**"
- In case of an invalid input: "**That is invalid input!\n | HTTP/1.1 200 OK**"
- In case of a win: "**You win! You threw " + getObjectName(playerMove) + "and the server threw " + getObjectName(serverMove) + ".\n" + "| HTTP/1.1 200 OK \n**"
- In case of a loss: **"You lost! You threw " + getObjectName(playerMove) + "and the server threw " + getObjectName(serverMove) + ".\n" + "| HTTP/1.1 200 OK\n"**
- In a case of a login request: **"The player " + userid + " has been logged out.\n" + "Here are the statistics from the session: \n" + "Total number of games played: " + str(G) + "\n" + "Total number of wins: " + str(W) + "\n" + "Total number of timed-out games: " + str(O) + "\n" + "| HTTP/1.1 200 OK"**

There is also an except clause that sends a response message to the client for any errors that may occur. The error message is constructed as so: **"\nHTTP/1.1 400 Bad Request\n\n"**. The connection socket is closed here.

How to run the programs:

First, take the client code, called Client.py. Use psftp to upload the file to any allv. For now we are going to use allv25.all.cs.stonybrook.edu. Once Client.py is on allv25.all.cs.stonybrook.edu, we also need the server code py file we need to run our client code to be on allv25.all.cs.stonybrook.edu. Use psftp once again to upload the server code to allv25.all.cs.stonybrook.edu. Let us say the server code py file is named Server.py. Make sure both files are in the same directory.

Then, login to an allv, say allv25.all.cs.stonybrook.edu at port 130. Once we are logged in, to test the client code, go to the directory where the files are, and type “python Server.py” to start the server in the terminal. Assuming my port number is 5768, open another terminal window (you can open another terminal window in allv25.all.cs.stonybrook.edu) and run the client program by typing “python Client.py allv25.all.cs.stonybrook.edu 5768”. If you run the server on another allv, this command must be changed to reflect the allv machine where the server is running. At this point, the client will ask you for moves or commands. The server is a log of events and will accept a login, print out the user name, start a game, make a move, receive a move, announce a game result, start a new game, and process logout requests.

Note: The server code must be run before the client code.

Testing Documentation:

First Test Case:

Client:

```
allv25:~> python Client.py allv26.all.cs.stonybrook.edu 5768
Welcome to Rock-Paper-Scissors!
To see a list of valid commands for this game, please type "help" before you log
 in.

Throwing a Rock against Scissors results in a win.
Throwing Paper against Rock results in a win.
Throwing Scissors against Paper results in a win.
A tie occurs when you throw the same object as your opponent (Rock and Rock, Pap
er and Paper, and Scissors and Scissors).

View the help menu or login: help
List of supported commands in this game:
help: Prints a list of supported commands with descriptions of their functions a
nd syntax of usage
login: Takes your player name (userid or nickname that uniquely identifies you)
as an argument and sends it to the server
r: Takes no arguments. Used by the player to throw a Rock to the server (the opp
onent)
p: Takes no arguments. Used by the player to throw Paper to the server (the oppo
nent)
s: Takes no arguments. Used by the player to throw Scissors to the server (the o
pponent)
logout: Takes no arguments. Sends a logout request to the server, displys game s
tatistics, and terminates the connection
```

Here, we see a welcome message displayed to the user once they are connected along with the rules of the game. Then there is a prompt to view the help menu or login. Here the user asks to see the help menu. This is the expected outcome for this test case.

Second Test Case:

Client:

```
View the help menu or login: login yromero
yromero has been logged in.

The game has now officially begun!
The server has thrown its object. You have 5 seconds to respond.
Throw an object or logout: p
You lost! You threw Paper and the server threw Scissors.

The game has now officially begun!
The server has thrown its object. You have 5 seconds to respond.
Throw an object or logout: q
That is invalid input!

The server has thrown another object. You have 5 seconds to respond.
Throw an object or logout: r
It is a tie. You and the server both threw Rock. Try again!
The server has thrown another object. You have 5 seconds to respond.
Throw an object or logout: p
You win! You threw Paper and the server threw Rock.

The game has now officially begun!
The server has thrown its object. You have 5 seconds to respond.
Throw an object or logout: logout

The player yromero has been logged out.
Here are the statistics from the session:
Total number of games played: 2
Total number of wins: 1
Total number of timed-out games: 0

allv25:~>
```

Here, the welcome message and the rules are printed (but not pictured here) and the user logs in correctly. The user plays some games (with a tie), inputs some invalid moves, and then logs out. The logout confirmation and the statistics from the session are printed. This is the expected outcome of this test case.

Third Test Case:

Client:

```
allv25:~> python Client.py allv26.all.cs.stonybrook.edu 5768
Welcome to Rock-Paper-Scissors!
To see a list of valid commands for this game, please type "help" before you log
 in.

Throwing a Rock against Scissors results in a win.
Throwing Paper against Rock results in a win.
Throwing Scissors against Paper results in a win.
A tie occurs when you throw the same object as your opponent (Rock and Rock, Pap
er and Paper, and Scissors and Scissors).

View the help menu or login: login
Please check your syntax for login.
View the help menu or login: login yromero
yromero has been logged in.
```

This is displayed at the client when the user does not use correct syntax for logging in. This is the expected outcome for this test case.

Fourth Test Case:

Client:

```
The game has now officially begun!
The server has thrown its object. You have 5 seconds to respond.
Throw an object or logout:
The game has been timed out.
```

This is displayed at the client when there is a timeout. This is the expected outcome for this test case.