Submission Worksheet

Submission Data

Course: IT114-005-F2025
Assignment: IT114 Milestone 1
Student: Sahith T. (st944)
Status: Submitted | **Worksheet Progress:** 100%
Potential Grade: 10.00/10.00 (100.00%)
Received Grade: 0.00/10.00 (0.00%)
Started: 11/3/2025 4:59:03 PM
Updated: 11/3/2025 7:27:04 PM
Grading Link: https://learn.ethereallab.app/assignment/v3/IT114-005-F2025/it114-milestone-1/grading/st944
View Link: https://learn.ethereallab.app/assignment/v3/IT114-005-F2025/it114-milestone-1/view/st944

Instructions

- Overview Link: https://youtu.be/9dZPFwi76ak

1. Refer to Milestone1 of any of these docs:
    2. Rock Paper Scissors
    3. Basic Battleship
    4. Hangman / Word guess
    5. Trivia
    6. Go Fish
    7. Pictionary / Drawing
2. Ensure you read all instructions and objectives before starting.
3. Ensure you've gone through each lesson related to this Milestone
4. Switch to the Milestone1 branch
    1. `git checkout Milestone1` (ensure proper starting branch)
    2. `git pull origin Milestone1` (ensure history is up to date)
5. Copy `Part5` and rename the copy as `Project` (this new folder should be in the root of your repo)
6. Organize the files into their respective packages `Client`, `Common`, `Server`, `Exceptions`
    1. Hint: If it's open, you can refer to the Milestone 2 Prep lesson
7. Fill out the below worksheet
    1. Ensure there's a comment with your UCID, date, and brief summary of the snippet in each screenshot
    2. Since this Milestone was majorly done via lessons, the required comments should be placed in areas of analysis of the requirements in this worksheet. There shouldn't need to be any actual code changes beyond the restructure.
8. Once finished, click "Submit and Export"
9. Locally add the generated PDF to a folder of your choosing inside your repository folder and move it to Github
    1. `git add .`
    2. `git commit -m "adding PDF"`
    3. `git push origin Milestone1`
    4. On Github merge the pull request from `Milestone1` to `main`
10. Upload the same PDF to Canvas
11. Sync Local

```
1. git checkout main
2. git pull origin main
```

Section #1: (1 pt.) Feature: Server Can Be Started Via Command Line And Listen To Connections

Progress: 100%

Task #1 (1 pt.) - Evidence

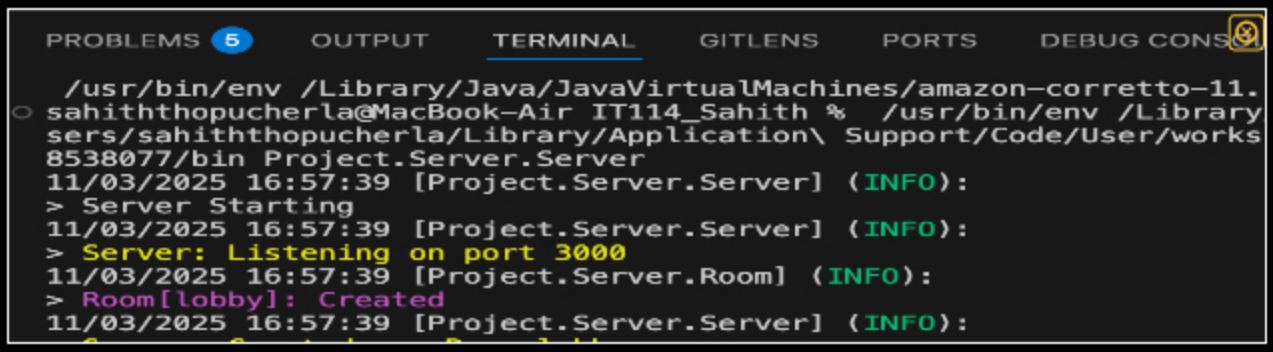
Progress: 100%

Part 1:

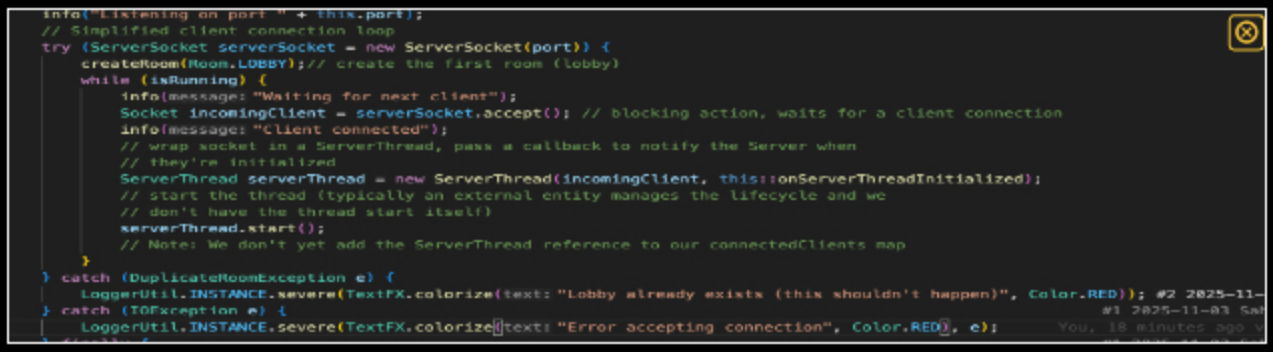
Progress: 100%

Details:

- Show the terminal output of the server started and listening
- Show the relevant snippet of the code that waits for incoming connections



output



code

Saved: 11/3/2025 5:11:31 PM

Part 2:

Progress: 100%

Details:

- Briefly explain how the server-side waits for and accepts/handles connections

Your Response:

The server waits for players to connect by using a ServerSocket that listens on a port. When a client tries to join, the server uses the accept command to open a connection. I think it is interesting how the server keeps waiting for more clients instead of stopping after one. Each new player gets their own thread so the server can handle many people at the same time without getting mixed up.

Saved: 11/3/2025 5:11:31 PM

Section #2: (1 pt.) Feature: Server Should Be Able To Allow More Than One Client To Be Connected At Once

Progress: 100%

Task #1 (1 pt.) - Evidence

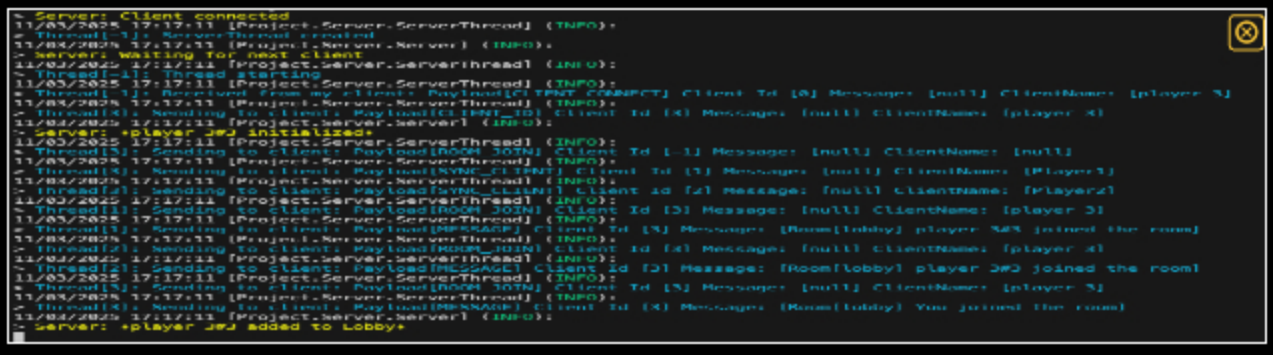
Progress: 100%

Part 1:

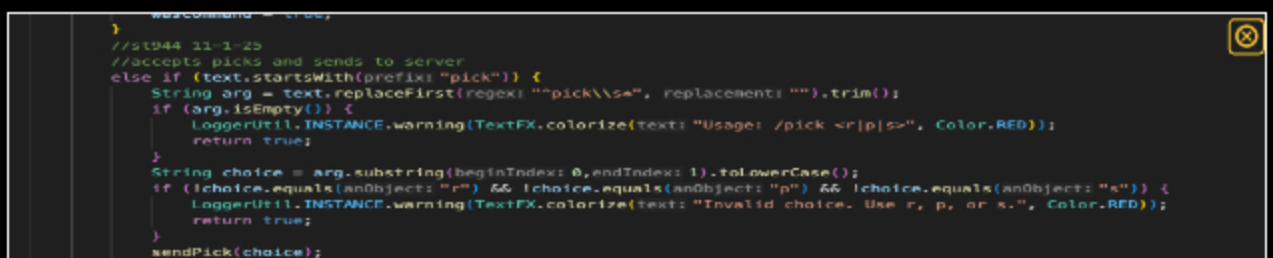
Progress: 100%

Details:

- Show the terminal output of the server receiving multiple connections
- Show at least 3 Clients connected (best to use the split terminal feature)
- Show the relevant snippets of code that handle logic for multiple connections



player connected



```
        LoggerUtil.INSTANCE.info(TextFX.colorize("You picked: " + choice, Color.GREEN));
        wasCommand = true;
    }
    You, 2 hours ago via  PR #9 • initial project files add  Zh Sahith initial project files add
return wasCommand;
```

code

Saved: 11/3/2025 5:20:33 PM

Part 2:

Progress: 100%

Details:

- Briefly explain how the server-side handles multiple connected clients

Your Response:

The server handles multiple clients by giving each person their own thread when they connect. This means when one client joins, the server starts a new ServerThread that talks only to that client. If another person joins, the server starts another thread for them. I think this is smart because it lets everyone connect and send messages at the same time without waiting for others. It feels like a real multiplayer game where everyone can play together smoothly.

Saved: 11/3/2025 5:20:33 PM

Section #3: (2 pts.) Feature: Server Will Implement The Concept Of Rooms (With The Default Being "l obby")

Progress: 100%

Task #1 (2 pts.) - Evidence

Progress: 100%

Part 1:

Progress: 100%

Details:

- Show the terminal output of rooms being created, joined, and removed (server-side)
- Show the relevant snippets of code that handle room management (create, join, leave, remove) (server-side)

```
> Connected
11/03/2025 17:15:27 [Project.Client.Client] (INFO):
> Room[lobby] You joined the room
11/03/2025 17:16:36 [Project.Client.Client] (INFO):
> Room[lobby] Player2#2 joined the room
11/03/2025 17:17:11 [Project.Client.Client] (INFO):
> Room[lobby] player 3#3 joined the room
/listrooms
11/03/2025 18:01:30 [Project.Client.Client] (INFO):
> Room Results:
11/03/2025 18:01:30 [Project.Client.Client] (INFO):
- lobby
/createroom game1
11/03/2025 18:01:57 [Project.Client.Client] (INFO):
```

```
> Room[lobby] You left the room
11/03/2025 18:01:57 [Project.Client.Client] (INFO):
> Room[game1] You joined the room
Current phase is READY
/listrooms
11/03/2025 18:02:11 [Project.Client.Client] (INFO):
> Room Results:
11/03/2025 18:02:11 [Project.Client.Client] (INFO):
> game1
lobby
```

output

```
private void start(int port) {
    this.port = port;
    // server listening
    info("Listening on port " + this.port);
    // Simplified client connection loop
    try (ServerSocket serverSocket = new ServerSocket(port)) {
        createRoom(Room.LOBBY);// create the first room (lobby)
        while (isRunning) {
            info(message: "Waiting for next client");
            Socket incomingClient = serverSocket.accept(); // blocking action, waits for a client connection
            info(message: "Client connected");
            // wrap socket in a ServerThread, pass a callback to notify the Server when
            // they're initialized
            ServerThread serverThread = new ServerThread(incomingClient, this::onServerThreadInitialized);
            // start the thread (typically an external entity manages the lifecycle and we
            // don't have the thread start itself)
            serverThread.start();
            // Note: We don't yet add the serverthread reference to our connectedClients map
            You, 4 hours ago via PR 29 • initial project files add  0b Nabith initial project files add
    } catch (DuplicateRoomException e) {
        LoggerUtil.INSTANCE.severe(TextFX.colorize(text: "Lobby already exists (this shouldn't happen)", Color.RED)); #2 2025-11-03
    } catch (IOException e) {                                                                                          #1 2025-11-03 Sahit
        LoggerUtil.INSTANCE.severe(TextFX.colorize(text: "Error accepting connection", color.RED), e);                 #2 2025-11-03
    } finally {                                                                                                        #1 2025-11-03 Nabit
        info(message: "Closing server socket");
    }
```

code

Saved: 11/3/2025 6:04:28 PM

Part 2:

Progress: 100%

Details:

- Briefly explain how the server-side handles room creation, joining/leaving, and removal

Your Response:

The server keeps things organized by using rooms. When it starts, it makes a default room called lobby. Every new player that joins goes into the lobby first. If someone creates a new room, the server makes it and gives it a name. Players can join or leave rooms, and the server moves them to the right place. When a room has no players left, the server removes it. I think this is a smart way to keep players separated so the game does not get confusing.

Saved: 11/3/2025 6:04:28 PM

Section #4: (1 pt.) Feature: Client Can Be Started Via The Command Line

Progress: 100%

Task #1 (1 pt.) - Evidence

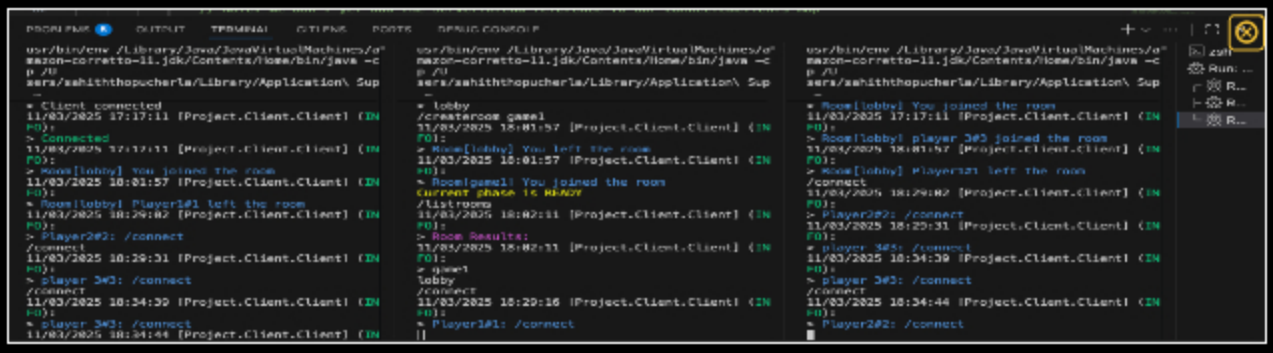
Progress: 100%

Part 1:

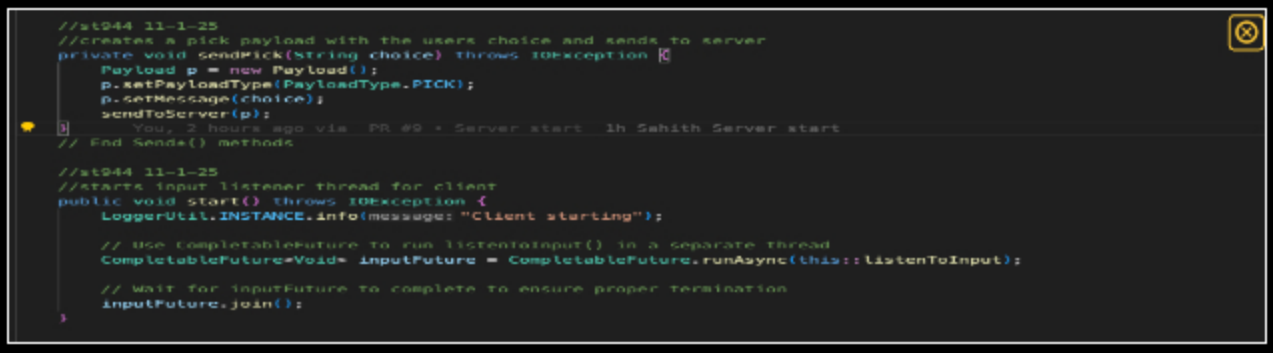
Progress: 100%

Details:

- Show the terminal output of the /name and /connect commands for each of 3 clients (best to use the split terminal feature)
- Output should show evidence of a successful connection
- Show the relevant snippets of code that handle the processes for /name, /connect, and the confirmation of being fully setup/connected



output



```
//st944 11-1-25
//creates a pick payload with the users choice and sends to server
private void sendPick(String choice) throws IOException {
    Payload p = new Payload();
    p.setPayloadType(PayloadType.PICK);
    p.setMessage(choice);
    sendToServer(p);
}      You, 2 hours ago via  PR #9 • Server start  1h Sahith Server start
// End Sends() methods

//st944 11-1-25
//starts input listener thread for client
public void start() throws IOException {
    LoggerUtil.INSTANCE.info(message: "Client starting");

    // Use CompletableFuture to run listenToInput() in a separate thread
    CompletableFuture<Void> inputFuture = CompletableFuture.runAsync(this::listenToInput);

    // Wait for inputFuture to complete to ensure proper termination
    inputFuture.join();
}
```

code

Saved: 11/3/2025 6:37:11 PM

Part 2:

Progress: 100%

Details:

- Briefly explain how the /name and /connect commands work and the code flow that leads to a successful connection for the client

Your Response:

When a player types name it saves their name so the server knows who they are. When they type connect localhost3000 the client uses that name to join the server. The code checks if it looks like a real address then it calls a connect method that opens the link to the server. It sets up input and output so the client and server can talk to each other. After that the client sends the players name to the server and shows Connected when it works. I think this is cool because it makes it feel like you are joining a real game server.

Saved: 11/3/2025 6:37:11 PM

Section #5: (2 pts.) Feature: Client Can Create/j oin Rooms

Progress: 100%

Task #1 (2 pts.) - Evidence

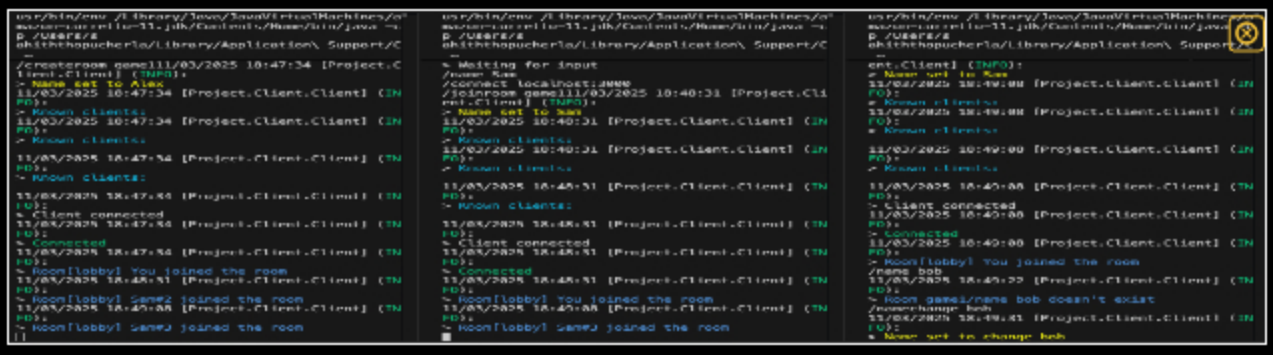
Progress: 100%

Part 1:

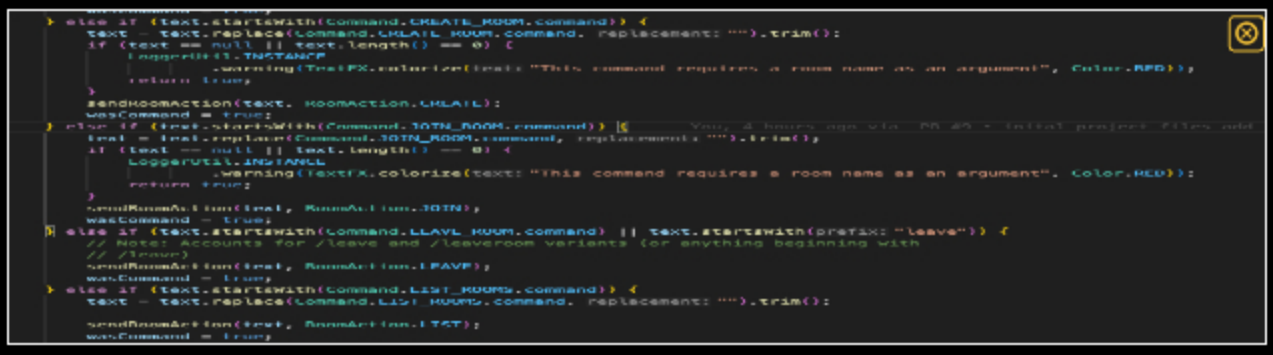
Progress: 100%

Details:

- Show the terminal output of the /createroom and /joinroom
- Output should show evidence of a successful creation/join in both scenarios
- Show the relevant snippets of code that handle the client-side processes for room creation and joining



output



code

Saved: 11/3/2025 6:51:59 PM

Part 2:

Progress: 100%

Details:

- Briefly explain how the /createroom and /join room commands work and the related code

flow for each

Your Response:

When a player types createroom and gives it a name, the client sends that name to the server using a message called a payload. The server reads it, creates a new room with that name, and moves the player into it. Then the client gets a message back saying the room was created. When a player types joinroom, the client sends another message to the server asking to join that room. The server checks if the room exists and then adds the player to it. The client gets a message showing they joined. I think this setup is smart because it makes it easy for players to make or join rooms without restarting the game.

Saved: 11/3/2025 6:51:59 PM

Section #6: (1 pt.) Feature: Client Can Send Messages

Progress: 100%

Task #1 (1 pt.) - Evidence

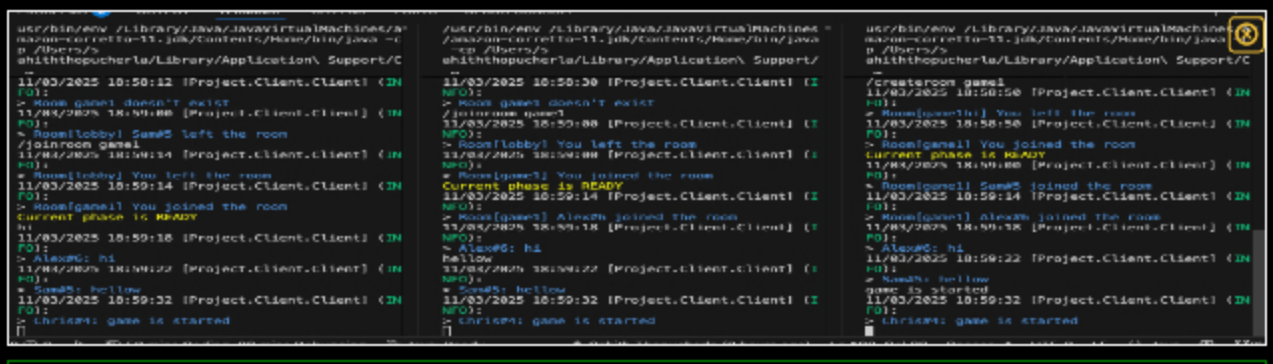
Progress: 100%

Part 1:

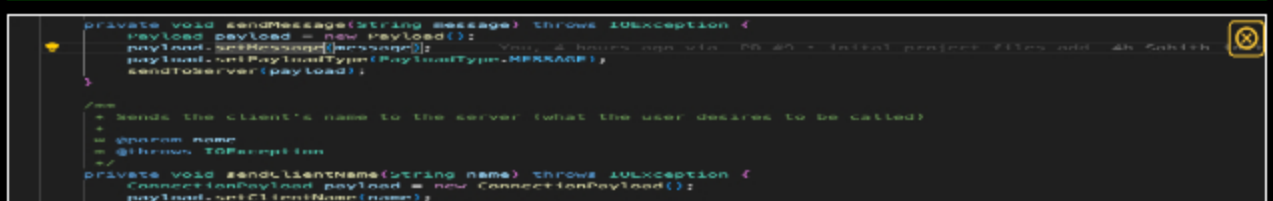
Progress: 100%

Details:

- Show the terminal output of a few messages from each of 3 clients
- Include examples of clients grouped into other rooms
- Show the relevant snippets of code that handle the message process from client to server-side and back



output



code

Saved: 11/3/2025 7:01:24 PM

Part 2:

Progress: 100%

Details:

- Briefly explain how the message code flow works

Your Response:

When a player types a message, the client first checks if it is a command. If it is not, the client wraps that text into a message object and sends it to the server. The server gets the message and looks at which room the player is in. It then sends the message to everyone else in that same room, including the sender. When the other clients receive it, they show the message in their terminals with the player's name in front. This makes it work like a normal chat room where everyone in the same group can see each other's messages.

Saved: 11/3/2025 7:01:24 PM

Section #7: (1 pt.) Feature: Disconnection

Progress: 100%

Task #1 (1 pt.) - Evidence

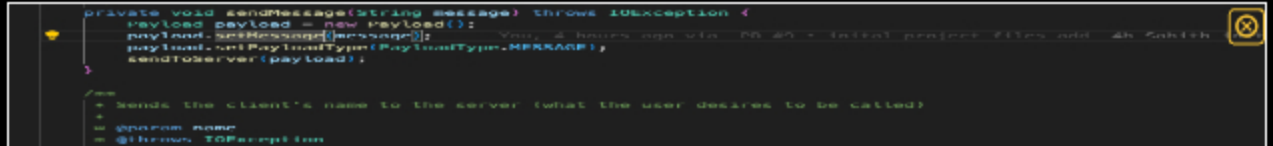
Progress: 100%

Part 1:

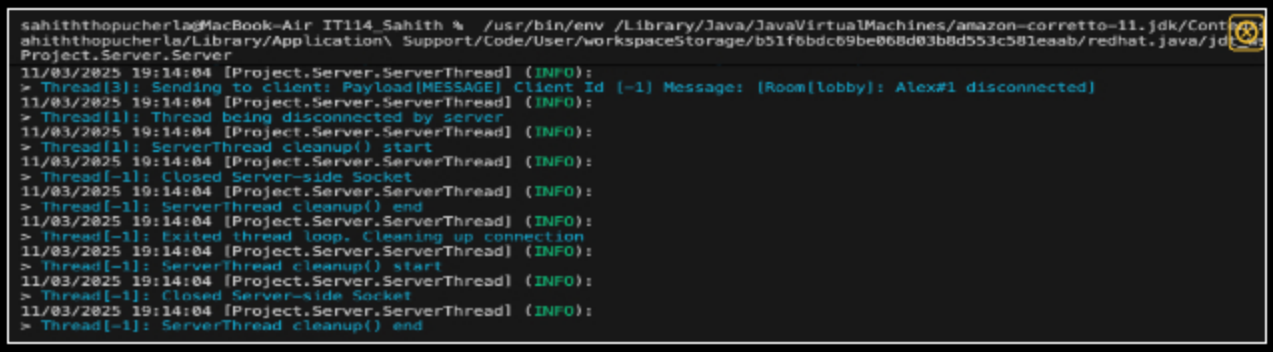
Progress: 100%

Details:

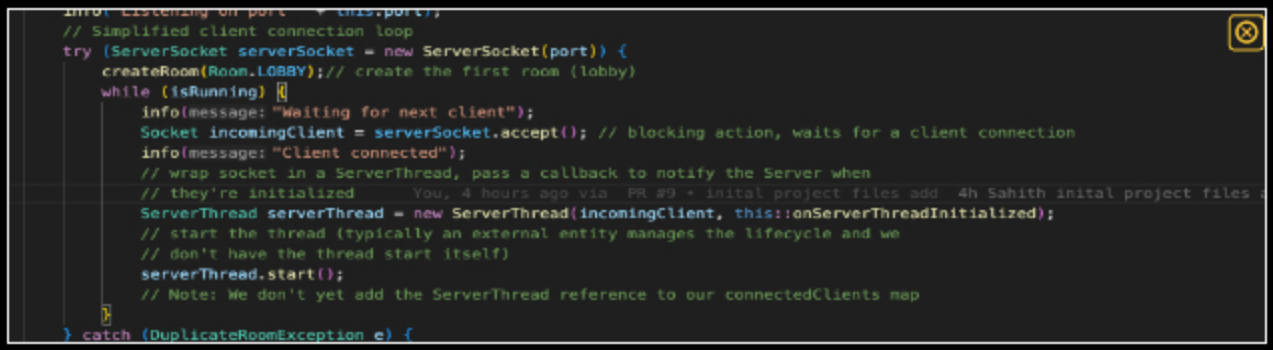
- Show examples of clients disconnecting (server should still be active)
- Show examples of server disconnecting (clients should be active but disconnected)
- Show examples of clients reconnecting when a server is brought back online
- Examples should include relevant messages of the actions occuring
- Show the relevant snippets of code that handle the client-side disconnection process
- Show the relevant snippets of code that handle the server-side termination process



output



output



code

Saved: 11/3/2025 7:16:12 PM

Part 2:

Progress: 100%

Details:

- Briefly explain how both client and server gracefully handle their disconnect/termination logic

Your Response:

When a player types disconnect, their client sends a signal to the server and closes its connection. The server then removes that player but keeps running for others. If the server goes offline, the clients notice that they can't talk to it anymore and show a message saying the connection dropped. When the server starts again, players can reconnect using the connect command. I think this is good because it keeps the game running smoothly even if someone leaves or the server restarts.

Saved: 11/3/2025 7:16:12 PM

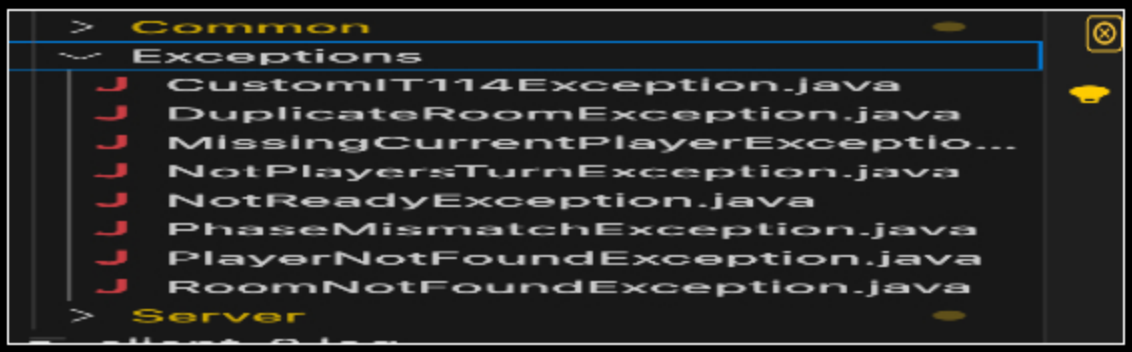
Section #8: (1 pt.) Misc

Progress: 100%

Task #1 (0.25 pts.) - Show the proper workspace structure with the new Client, Common, Server, and Exceptions packages

Progress: 100%

files



files

Saved: 11/3/2025 7:21:35 PM

Task #2 (0.25 pts.) - Github Details

Progress: 100%

Part 1:

Progress: 100%

Details:
From the Commits tab of the Pull Request screenshot the commit history

pull

Saved: 11/3/2025 7:23:04 PM

Part 2:

Progress: 100%

Details:
Include the link to the Pull Request (should end in `/pull/#`)

URL #1
https://github.com/SahithThopucherla/IT114_Sahith/pulls

URL
https://github.com/SahithThopucl

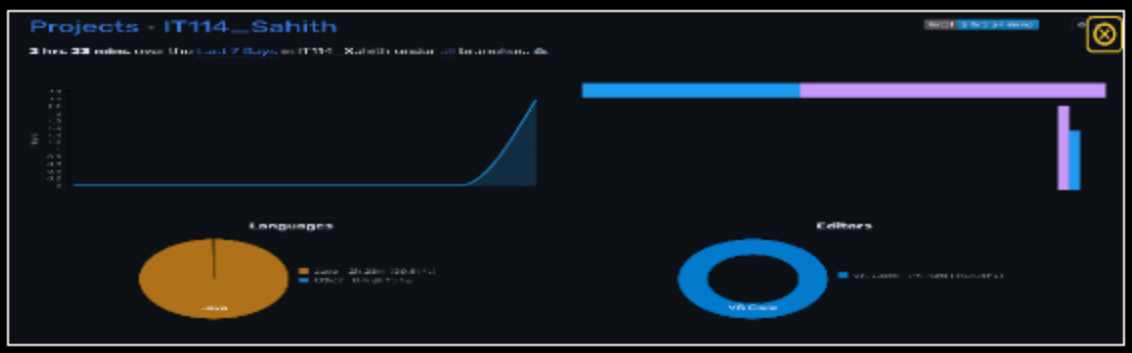
Saved: 11/3/2025 7:23:04 PM

Task #3 (0.25 pts.) - WakaTime - Activity

Progress: 100%

Details:

- Visit the WakaTime.com Dashboard
- Click `Projects` and find your repository
- Capture the overall time at the top that includes the repository name
- Capture the individual time at the bottom that includes the file time
- Note: The duration isn't relevant for the grade and the visual graphs aren't necessary



time

Saved: 11/3/2025 7:23:57 PM

Task #4 (0.25 pts.) - Reflection

Progress: 100%

Task #1 (0.33 pts.) - What did you learn?

Progress: 100%

Details:
Briefly answer the question (at least a few decent sentences)

Your Response:

I learned how servers and clients talk to each other in a networked program. I also learned how rooms work in multiplayer games, how players can join and leave them, and how messages are sent between clients. This project helped me understand the flow of data from one computer to another and how to handle errors when someone disconnects.

Saved: 11/3/2025 7:26:51 PM

Task #2 (0.33 pts.) - What was the easiest part of the assignment?

Progress: 100%

Details:
Briefly answer the question (at least a few decent sentences)

Your Response:

The easiest part was starting the server and connecting the clients. Once everything was set up, it was pretty simple to run the commands like name, connect, and joinroom. Seeing the messages appear between clients was fun and made it feel like a real chat app.

Saved: 11/3/2025 7:26:57 PM

Task #3 (0.33 pts.) - What was the hardest part of the assignment?

Progress: 100%

Details:
Briefly answer the question (at least a few decent sentences)

Your Response:

The hardest part was figuring out how the server handled multiple clients at the same time. I had to understand how threads worked and how each client was kept separate. Debugging connection errors was also tricky because even small mistakes like missing localhost could break the connection.

Saved: 11/3/2025 7:27:04 PM