Submission Worksheet

Submission Data

Course: IT114-450-M2025
Assignment: IT114 Milestone 2 - RPS
Student: Lamia M. (lm87)
Status: Submitted | **Worksheet Progress:** 100%
Potential Grade: 10.00/10.00 (100.00%)
Received Grade: 0.00/10.00 (0.00%)
Started: 8/10/2025 5:01:19 PM
Updated: 8/10/2025 11:27:54 PM
Grading Link: https://learn.ethereallab.app/assignment/v3/IT114-450-M2025/it114-milestone-2-rps/grading/lm87
View Link: https://learn.ethereallab.app/assignment/v3/IT114-450-M2025/it114-milestone-2-rps/view/lm87

Instructions

1. Refer to Milestone2 of Rock Paper Scissors
    1. Complete the features
2. Ensure all code snippets include your ucid, date, and a brief description of what the code does
3. Switch to the `Milestone2` branch
    1. `git checkout Milestone2`
    2. `git pull origin Milestone2`
4. Fill out the below worksheet as you test/demo with 3+ clients in the same session
5. Once finished, click "Submit and Export"
6. 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 Milestone2`
    4. On Github merge the pull request from `Milestone2` to `main`
7. Upload the same PDF to Canvas
8. Sync Local
    1. `git checkout main`
    2. `git pull origin main`

Section #1: (1 pt.) Payloads

Progress: 100%

--- Section Collapsed ---

Section #2: (4 pts.) Lifecycle Events

Progress: 100%

--- Section Collapsed ---

Section #3: (4 pts.) Gameroom User Action

And State

Progress: 100%

Task #1 (2 pts.) - Choice Logic

Progress: 100%

Details:

- Reqs from document
  - Command: `/pick <[r,p,s]>` (user picks one)
    - GameRoom will check if it's a valid option
    - GameRoom will record the choice for the respective Player
    - A message will be relayed saying that "X picked their choice"
    - If all Players have a choice the round ends

Part 1:

Progress: 100%

Details:

- Show the code snippets of the following, and clearly caption each screenshot
- Show the Client processing of this command (process client command)
- Show the ServerThread processing of this command (process method)
- Show the GameRoom handling of this command (handle method)
- Show the sending/syncing of the results of this command to users (send/sync method)
- Show the ServerThread receiving this data (send method)
- Show the Client receiving this data (process method)

```
                break;

                // UCID: jm87 | Date: 2025-08-10
    // Brief: Route PICK to the room so GameRoom can record the choice.
                case PICK:
                System.out.println("Thread[" + getId() + "] PICK <- " + getClientName()
                + " choice=" + incoming.getMessage());
                currentRoom.handlePick(this, incoming.getMessage()); // "r","p","s"
                break;

                case START:
                if (currentRoom instanceof GameRoom) {
                    System.out.println("Thread[" + getId() + "]: START requested by " + getClientName());
                    ((GameRoom) currentRoom).onSessionStart();  // make this method public if needed
                } else {
                    sendMessage(message:"This command only works in a GameRoom.");
                }
                break;

            default:
                System.out.println(TextFX.colorize(text:"Unknown payload type received", Color.RED));
```

This method runs on the client to parse user-entered commands such as /start, /pick, /createroom, and /joinroom, and builds the

```
// UCID: jm87 | Date: 2025-08-10
// Brief: Validate /pick, record it, announce, and check if the round should end.
@Override
protected synchronized void handle(ServerThread sender, String roomName) {
    if (phase != Phase.CHOOSING) {
        sender.sendMessage(message:"You cannot pick right now. Wait for the next round.");
        return;
    }
    long uid = sender.getId();
    if (eliminated.getOrDefault(uid, defaultValue: false)) {
        sender.sendMessage(message:"You're eliminated this session and are now a spectator.");
        return;
    }
    if (picks.get(uid) != null && picks.get(uid) != Choice.NONE) {
        sender.sendMessage(message:"You already picked for this round.");
        return;
    }
    Choice choice;
    switch (raw.trim().toLowerCase()) {
        case "r": choice = Choice.ROCK; break;
        case "p": choice = Choice.PAPER; break;
        case "s": choice = Choice.SCISSORS; break;
        default:
            sender.sendMessage(message: "Invalid pick. Use /pick r|p|s");
            return;
    }
    if (phase == null) {
        phase = Phase.WAIT;
    }
    picks.put(uid, choice); // now keys match
    broadcast(sender.getDisplayNameOrId() + " picked their choice.");
    if (allActivePicked()) {
        cancelRoundTimer();
        resolveRound();
    }
```

This method executes on the server's client thread, receiving payloads from a connected client and routing them to the correct h

```java
// UCID: lm87 | Date: 2025-08-10
private void syncPoints() {
    // Take a safe snapshot
    Map<Long, Integer> snapshot = new java.util.LinkedHashMap<>(points);

    // Send to all active ServerThreads in the room
    for (ServerThread st : new java.util.ArrayList<>(clientsInRoom.values())) {
        st.sendPoints(snapshot, reason:"[SCOREBOARD]");
    }
}
```

These methods prepare and send updated game state to all clients in the room, including scoreboards, eliminations, and system me

```java
    protected boolean sendToClient(Payload payload) {
        if (!isRunning) {
            return true;
        }
        try {
            info("Sending to client: " + payload);
            out.writeObject(payload);
            out.flush();
            return true;
        } catch (IOException e) {
            info(message:"Error sending message to client (most likely disconnected)");
            // comment this out to inspect the stack trace
            // e.printStackTrace();
            cleanup();
            return false;
        }
    }
```

The low-level send method that writes a payload over the socket to the client. All broadcast and sync calls ultimately use this

```java
private void processPayload(Payload payload) {
    switch (payload.getPayloadType()) {
        case HELLO_SERVER // unused
            break;
        case HELLO_CLI
            processFromHelloClient(payload);
            break;
        case DISCONNECT:
            processDisconnect(payload);
            break;
        case MESSAGE:
            processMessage(payload);
            break;
        case REVERSE:
            processReverse(payload);
            break;
        case MARK_DEAD // unused
            break;
        case MARK_DEAD:
            processMarkDead(payload);
            break;
        case MARK_LAVA:
            processMarkLava(payload);
            break;
        case SYNC_POINTS:
            processUnknownAction(payload);
            break;
        // UCID: lm87 | Date: 2025-08-10
        case HUNTER_SYNC:
            processHunterSync(payload);
            break;
        default:
            System.out.println(FxAPI.colorize(text:"Unhandled payload type", color:YELLOW));
            break;
    }
}
```

The client's main payload dispatcher, which interprets incoming server messages, updates the UI/console, and handles special pay

Saved: 8/10/2025 11:15:21 PM

Part 2:

Progress: 100%

Details:

- Briefly explain/list in order the whole flow of this command being handled from the client-side to the server-side and back

Your Response:

Client command entry – The player types /pick r in the client console.

Client command processing – Client.processClientCommand() matches the command, creates a PICK payload with "r" as the choice, and sends it to the server with sendPayload().

Server receives payload – ServerThread.processPayload() reads the incoming PICK payload from the socket and calls GameRoom.handlePick() for the player's current room.

GameRoom handles choice – GameRoom.handlePick() validates and stores the choice, then checks if all active players have picked. If so, it calls onRoundEnd() to resolve the round.

GameRoom resolves and sends results – onRoundEnd() determines winners/losers, calls broadcast() to send messages, and calls syncPoints() to send updated scores to all clients in the room.

Server sends to clients – broadcast() and syncPoints() loop over each player's ServerThread, calling sendPayload(), which writes the data to each client's socket.

Client receives updates – Client.processPayload() on each client reads the incoming messages and score updates, and displays them in the console.

Saved: 8/10/2025 11:15:21 PM

Task #2 (2 pts.) - Game Cycle Demo

Progress: 100%

Details:

- Show examples from the terminal of a full session demonstrating each command and progress output
- This includes battle outcomes, scores and scoreboards, etc
- Ensure at least 3 Clients and the Server are shown
- Clearly caption screenshots

terminal outpot showing all the session scores, scoreboards etc with 3 clients.

Saved: 8/10/2025 11:18:12 PM

Section #4: (1 pt.) Misc

Progress: 100%

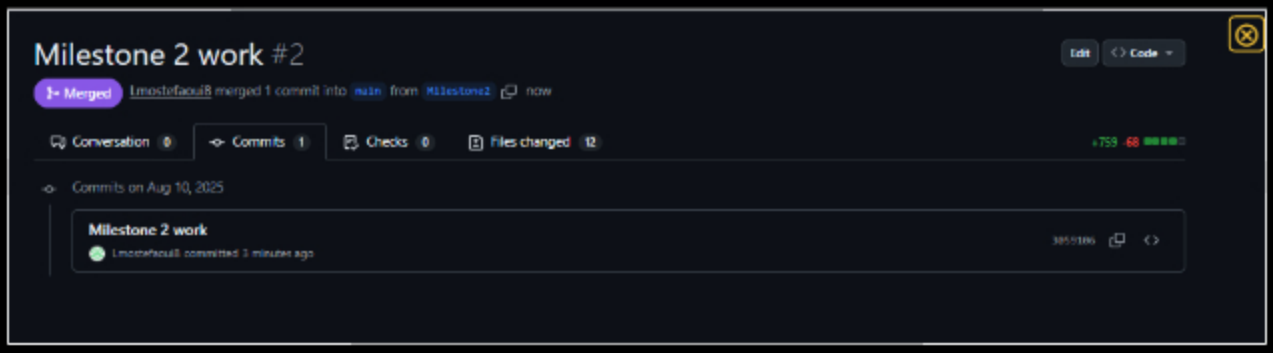
Task #1 (0.33 pts.) - Github Details

Progress: 100%

Part 1:

Progress: 100%

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



commits history from pull request

Saved: 8/10/2025 11:25:23 PM

Part 2:

Progress: 100%

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

URL #1
https://github.com/Lmostefaoui8/lm87-it114/pull/2

URL
https://github.com/Lmostefaoui8/

Saved: 8/10/2025 11:25:23 PM

Task #2 (0.33 pts.) - WakaTime - Activity

Progress: 100%

--- Task Collapsed ---

Task #3 (0.33 pts.) - Reflection

Progress: 100%

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

Progress: 100%

--- Task Collapsed ---

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 running multiple client instances and issuing the game commands. Once the server and clients were compiled and running, creating a room, joining it from different clients, and sending commands like /start or /pick was straightforward and gave immediate feedback that the system was working.

Saved: 8/10/2025 11:27:39 PM

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

Progress: 100%

--- Task Collapsed ---