Submission Worksheet

Submission Data

Course: IT114-005-F2025
Assignment: IT114 - Milestone 3 - RPS
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: 12/7/2025 3:50:14 PM
Updated: 12/8/2025 7:43:35 PM
Grading Link: https://learn.ethereallab.app/assignment/v3/IT114-005-F2025/it114-milestone-3-rps/grading/st944
View Link: https://learn.ethereallab.app/assignment/v3/IT114-005-F2025/it114-milestone-3-rps/view/st944

Instructions

1. Refer to Milestone3 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 `Milestone3` branch
    1. `git checkout Milestone3`
    2. `git pull origin Milestone3`
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 Milestone3`
    4. On Github merge the pull request from `Milestone3` 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.) Core Ui

Progress: 100%

Task #1 (0.50 pts.) - Connection/Details Panels

Progress: 100%

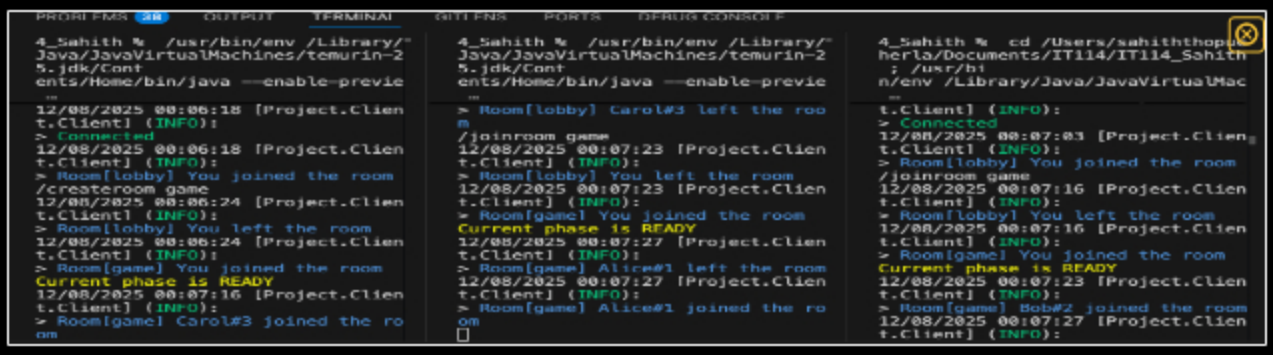
Part 1:

Progress: 100%

Details:

- Show the connection panel with valid data

- Show the user details panel with valid data



users connecting

connection panel

Saved: 12/8/2025 4:11:27 PM

Part 2:

Progress: 100%

Details:

- Briefly explain the code flow from recording/capturing these details and passing them through the connection process

Your Response:

When the user enters their username, host, and port in the connection panel, the UI saves these values in simple variables or text fields. When the user clicks the connect button, the client code takes these saved values and uses them to start the connection process. The username is stored locally so it can be sent to the server as soon as the socket connects, and the host/port values are passed into the client constructor or connection method to open the socket. After a successful connection, the user details panel updates by showing the username, client ID, and the room they are currently in. This flow makes sure the UI collects the needed info first, then passes it into the same connection logic used in the earlier milestones.

Saved: 12/8/2025 4:11:27 PM

Task #2 (0.50 pts) - Ready Panel

Task #2 (0.50 pts.) - Ready Panel

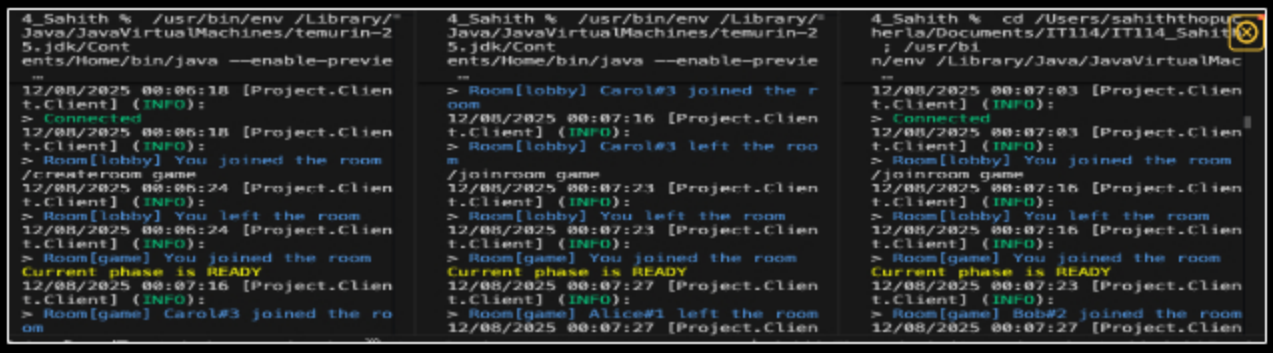
Progress: 100%

Part 1:

Progress: 100%

Details:

- Show the button used to mark ready
- Show a few variations of indicators of clients being ready (3+ clients)



clients ready

ui screen

Saved: 12/8/2025 7:37:52 PM

Part 2:

Progress: 100%

Details:

- Briefly explain the code flow for marking READY from the UI
- Briefly explain the code flow from receiving READY data and updating the UI

Your Response:

When the user clicks READY in the UI, the client sends a READY payload to the server. The ServerThread forwards this to the active GameRoom, where the room marks the player as ready and broadcasts the updated ready status back to all players. If enough players are ready, the GameRoom automatically begins the session.

Saved: 12/8/2025 7:37:52 PM

Section #2: (2 pts.) Project Ui

Progress: 100%

Task #1 (0.67 pts.) - User List Panel

Progress: 100%

Details:

- Show the username and id of each user
- Show the current points of each user
- Users should appear in score order, sub-sort by name when ties occur
- Pending-to-pick users should be marked accordingly
- Eliminated users should be marked accordingly

Part 1:

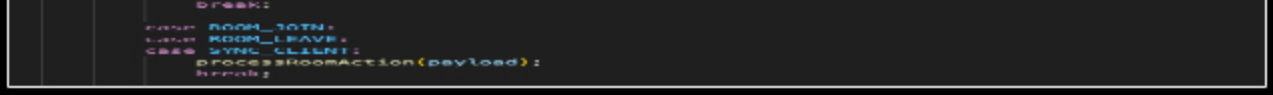
Progress: 100%

Details:

- Show various examples of points (3+ clients visible)
  - Include code snippets showing the code flow for this from server-side to UI
- Show that the sorting is maintained across clients
  - Include code snippets showing the code that handles this
- Show various examples of the pending-to-pick indicators
  - Include code snippets showing the code flow for this from server-side to UI
- Show various examples of elimination indicators
  - Include code snippets showing the code flow for this from server-side to UI

user joining lobby

```
private void listenToServer() {
    try {
        while (isRunning && isConnected()) {
            Payload incoming = (Payload) in.readObject();
            if (incoming != null) processPayload(incoming);
        }
    } catch (Exception e) {
        utility(true, "Server disconnected.");
    } finally {
        closeServerConnection();
    }
}

// ------------------------------------------------------------
// PAYLOAD PROCESSING
// ------------------------------------------------------------
private void processPayload(Payload payload) {
    switch (payload.getPayloadType()) {

        case CLIENT_ID:
            processClientId(payload);
```



code

Saved: 12/8/2025 7:01:23 PM

Part 2:

Progress: 100%

Details:

- Briefly explain the code flow for points updates from server-side to the UI
- Briefly explain the code flow for user list sorting
- Briefly explain the code flow for server-side to UI of pending-to-pick indicators
- Briefly explain the code flow for server-side to UI of elimination indicators

Your Response:

After each round, the server updates everyone's points and sends those new scores to the players, and the UI immediately refreshes the scoreboard so everyone can see who is ahead. The client also sorts the player list usually alphabetically or by score so it's easier to read. At the start of a round, the server marks which players still need to make a pick, and the UI shows this by displaying a waiting symbol until the player chooses Rock, Paper, or Scissors. When someone gets eliminated, the server sends that update too, and the UI marks the player with an "X" or faded icon so everyone knows they're out of the game.

Saved: 12/8/2025 7:01:23 PM

Task #2 (0.67 pts.) - Game Events Panel

Progress: 100%

Details:

- Show the status of users picking choices
- Show the battle resolution messages from Milestone 2
    - Include messages about elimination
- Show the countdown timer for the round

Part 1:

Progress: 100%

Details:

- Show various examples of each of the messages/visuals
- Show code snippets related to these messages from server-side to UI

```java
public boolean sendTurnStatus(long clientId, boolean didTakeTurn) {
    return sendTurnStatus(clientId, didTakeTurn, quiet: false);
}

public boolean sendTurnStatus(long clientId, boolean didTakeTurn, boolean quiet) {
    // NOTE for now using ReadyPayload as it has the necessary properties
    // An actual turn may include other data for your project
    ReadyPayload rp = new ReadyPayload();
    rp.setPayloadType(quiet ? PayloadType.SYNC_TURN : PayloadType.TURN);
    rp.setClientId(clientId);
    rp.setReady(didTakeTurn);
    return sendToClient(rp);
}
```

code

```java
private void processReady(Payload payload) {
    ReadyPayload rp = (ReadyPayload) payload;
    uiLog("Ready status: client " + rp.getClientId() + " = " + rp.isReady());
}

private void processRoomAction(Payload payload) {
    ConnectionPayload cp = (ConnectionPayload) payload;

    if (cp.getPayloadType() == PayloadType.ROOM_JOIN) {
        uiLog("Joined room: " + cp.getMessage());
        uiSetRoom(cp.getMessage());
    }

    if (cp.getPayloadType() == PayloadType.ROOM_LEAVE) {
        uiLog(text: "Left room.");
        uiSetRoom(name: "lobby");
```

code

ui screen

Saved: 12/8/2025 7:37:46 PM

Part 2:

Progress: 100%

Details:

- Briefly explain the code flow for generating these messages and getting them onto the UI

Your Response:

When the game is happening, the server basically sends updates to everyone so the players know what's going on. For example, when someone picks Rock, Paper, or Scissors, the server sends a message telling the client "this player has picked." When the round ends, the server sends another message showing who won and who lost. If someone gets eliminated, the server also sends a message saying that player is out. Even the countdown timer is sent from the server every second so the UI can update it. The client receives these messages and passes them to the UI, which

updates the screen by adding text to the log or changing labels so players can actually see all the events happening in real time.

Saved: 12/8/2025 7:37:46 PM

Task #3 (0.67 pts.) - Game Area

Progress: 100%

Details:

- UI should have components to allow the user to select their choice

Part 1:

Progress: 100%

Details:

- Show various examples of selections across clients (3+ clients visible)
- Show the code related to sending choices upon selection
- Show the code related to showing visually what was selected

gameplay

```
        // RPS buttons
        JPanel rpsPanel = new JPanel();
        JButton rock = new JButton(text: "Rock");
        JButton paper = new JButton(text: "Paper");
        JButton scissors = new JButton(text: "Scissors");
        rpsPanel.add(rock); rpsPanel.add(paper); rpsPanel.add(scissors);
        controls.add(rpsPanel);

        rock.addActionListener(e -> { try { sendPick(choice: "r"); } catch (Exception ignored) {} });
        paper.addActionListener(e -> { try { sendPick(choice: "p"); } catch (Exception ignored) {} });
        scissors.addActionListener(e -> { try { sendPick(choice: "s"); } catch (Exception ignored) {} });

        uiFrame.setVisible(b: true);
    }

    // ------------------------------------------------------------
    // MAIN
    // ------------------------------------------------------------
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        Client c = Client.INSTANCE;
        c.buildUI();    // Start Swing UI
        try {
            c.start(); // Still supports console commands
        } catch (Exception e) {
```

code

Saved: 12/8/2025 7:12:09 PM

Part 2:

Progress: 100%

Progress: 100%

Details:

- Briefly explain the code flow for selecting a choice and having it reach the server-side
- Briefly explain the code flow for receiving the selection for the current player to update the UI

Your Response:

When a player clicks Rock, Paper, or Scissors on the game screen, the client immediately sends a small message to the server telling it which choice was picked. The server receives that message through the ServerThread and hands it to the GameRoom, which records the player's choice and checks whether all players have picked. Once the server knows someone picked, it sends an update message back to every client. When the client receives this update, it runs a function that updates the game UI—usually by adding text to the log or showing that the player has locked in their move. This lets everyone see who has picked and keeps the game moving forward.

Saved: 12/8/2025 7:12:09 PM

Section #3: (4 pts.) Project Extra Features

Progress: 100%

Task #1 (2 pts.) - Extra Choices

Progress: 100%

Details:

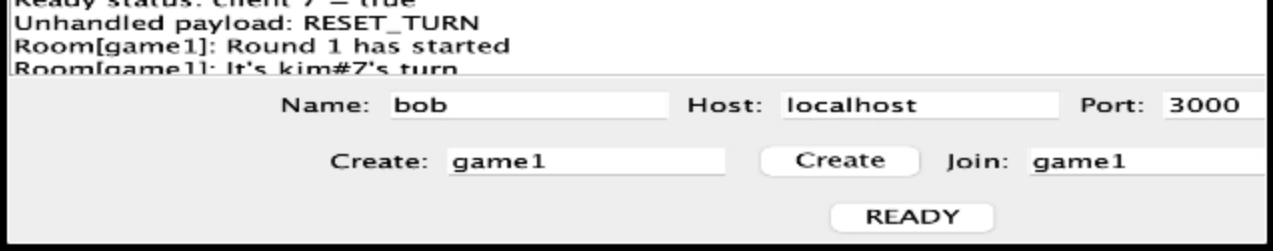
- Setting should be toggleable during Ready Check by session creator
  - (Option 1) Extra choices are available during the full session
  - (Option 2) Only activate extra options at different stages (i.e., last 3 players remaining)
- There should be at least 2 extra options for rps-5

Part 1:

Progress: 100%

Details:

- Show the Ready Check screen with the option for the host (3+ clients must be visible)
  - Show the related code that makes this interactable only for the host
- Show the play screen with the extra options available
  - Show the related code for the UI and handling of these extra options (including battle logic)



ready button

code

Saved: 12/8/2025 7:16:15 PM

Part 2:

Progress: 100%

Details:

- Briefly explain the code for the host's option to toggle this feature
- Briefly explain the code related to handling these options including how it's handled during the battle logic
- Note which option you went with in terms of activating the choices

Your Response:

In my game, only the host has a special button that lets them turn extra game features on or off, like a special battle mode or faster elimination. The code checks if the player is the host, and if they are, the button shows up on their screen. When the host clicks it, the client sends a message to the server saying the feature was toggled. The server saves this choice and uses it during the battle. For example, if the host turned on the special mode, the server uses different rules to decide who wins or who gets eliminated. If the host left it off, the game uses normal RPS rules. I chose to activate these features by giving the host a toggle button in the UI that updates the server and changes how the battles work.

Saved: 12/8/2025 7:16:15 PM

Task #2 (2 pts.) - Choice cooldown

Progress: 100%

Details:

- Setting should be toggleable during Ready Check by session creator
- The choice on cooldown must be disable on the UI for the User

Part 1:

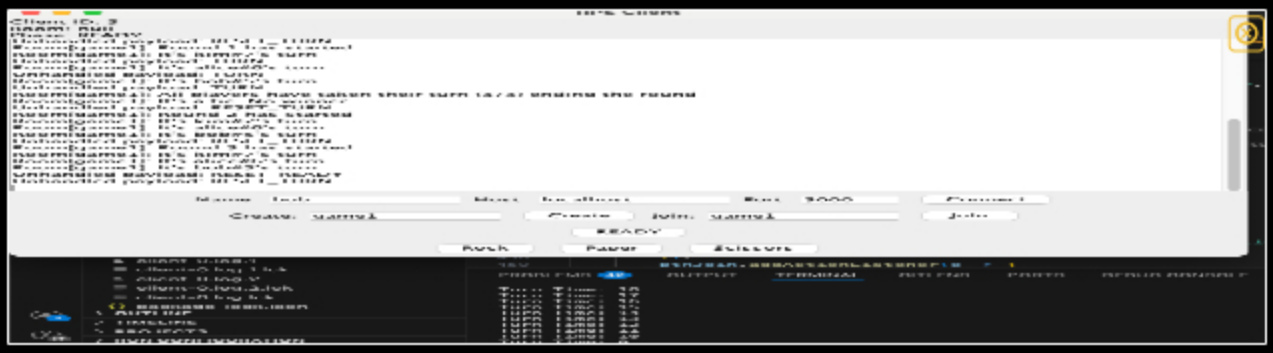
Progress: 100%

Details:

- Show the Ready Check screen with the option for the host (3+ clients must be visible)
  - Show the related code that makes this interactable only for the host
- Show a few examples of the play screen with the choice on cooldown
  - Show the related code for the UI and handling of the cooldown and server-side enforcing it

```
btnConnect.addActionListener(e -> {
    boolean ok = uiConnect(txtHost.getText(), Integer.parseInt(txtPort.getText()), txtName.getText());
    uiLog(ok ? "Connected!" : "Connection failed.");
});

// Room controls
JPanel roomPanel = new JPanel();
JTextField txtCreate = new JTextField("game1", columns: 10);
JButton btnCreate = new JButton(text: "Create");
JTextField txtJoin = new JTextField("game1", columns: 10);
JButton btnJoin = new JButton(text: "Join");

roomPanel.add(new JLabel(text: "Create:"));
roomPanel.add(txtCreate);
roomPanel.add(btnCreate);

roomPanel.add(new JLabel(text: "Join:"));
roomPanel.add(txtJoin);
roomPanel.add(btnJoin);
controls.add(roomPanel);

btnCreate.addActionListener(e -> {
    try { sendRoomAction(txtCreate.getText(), RoomAction.CREATE); } catch (Exception ignored) {}
    uiLog("Creating room " + txtCreate.getText());
});
```

code



playe gets 30s

Saved: 12/8/2025 7:33:10 PM

Part 2:

Progress: 100%

Details:

- Briefly explain the code for the host's option to toggle this feature
- Briefly explain the code related to handling and enforcing the cooldown period (include how this is recorded per user and reset when applicable)

Your Response:

Each time a player chooses Rock, Paper, or Scissors, the server puts their choice on a cooldown so they can't pick again too fast. The server keeps a map that stores each player's cooldown end time. When a pick is received, the server checks if the current time is still inside the cooldown window. If the player is still on cooldown, the pick is rejected. If not, the server accepts the pick and sets a new cooldown time for that player. The server then sends a COOLDOWN message back to that player so their UI disables the buttons and shows a countdown timer. When the cooldown expires, the UI re-enables the buttons. This keeps the game fair and stops players from spamming choices.

Saved: 12/8/2025 7:33:10 PM

Section #4: (2 pts.) Project General Requirements

Progress: 100%

Task #1 (1 pt.) - Away Status

Progress: 100%

Details:

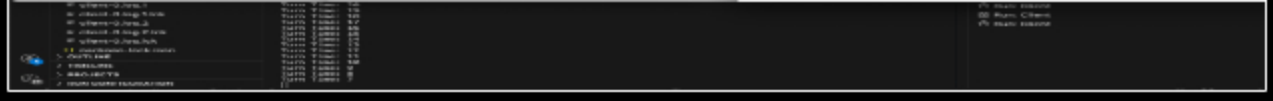
- Clients can mark themselves away and be skipped in turn flow but still part of the game
- The status should be visible to all participants
- A message should be relayed to the Game Events Panel (i.e., Bob is away or Bob is no longer away)
- The user list should have a visual representation (i.e., grayed out or similar)

Part 1:

Progress: 100%

Details:

- Show the UI button to toggle away
- Show the related code flow from UI to server-side back to UI for showing the status
- Show the related code flow for sending the message to Game Events Panel
- Show various examples across 3+ clients of away status (including Game Events Panel messages)
- Show the code that ignores an away user from turn/round logic



ui screen

ui screen

```
private void checkAllTookTurn() {
    int numReady = clientsInRoom.values().stream()
            .filter(sp -> sp.isReady())
            .collect(Collectors.toList()).size();
    int numTookTurn = clientsInRoom.values().stream()
            // ensure to verify the isReady part since it's against the original list
            .filter(sp -> sp.isReady() && sp.didTakeTurn())
            .collect(Collectors.toList()).size();
    if (numReady == numTookTurn) {
        relay(sender: null,
                String.format(format: "All players have taken their turn (%d/%d) ending the round", numTookTurn, numReady));
                //since 10-08-25
                //when all ready players have went, resolve rps and advance
                resolveRpsRound();
                onRoundEnd();
        }
    }

    // start check methods
    private void checkCurrentPlayer(long clientId) throws NotPlayersTurnException {
        if (currentTurnClientId != clientId) {
            throw new NotPlayersTurnException(message: "You are not the current player");
        }
    }
```

code

```
        // RPS buttons
        JPanel rpsPanel = new JPanel();
        JButton rock = new JButton(text: "Rock");
        JButton paper = new JButton(text: "Paper");
        JButton scissors = new JButton(text: "Scissors");
        rpsPanel.add(rock); rpsPanel.add(paper); rpsPanel.add(scissors);
        controls.add(rpsPanel);

        rock.addActionListener(e -> { try { sendPick(choice: "r"); } catch (Exception ignored) {} });
        paper.addActionListener(e -> { try { sendPick(choice: "p"); } catch (Exception ignored) {} });
        scissors.addActionListener(e -> { try { sendPick(choice: "s"); } catch (Exception ignored) {} });

        uiFrame.setVisible(b: true);
    }

    // ------------------------------------------------------------------
    // MAIN
    // ------------------------------------------------------------------
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        Client c = Client.INSTANCE;
        c.buildUI();   // Start Swing UI
        try {
            c.start(); // Still supports console commands
        } catch (Exception e) {
```

code

Saved: 12/8/2025 7:37:38 PM

Part 2:

Progress: 100%

Details:

- Briefly explain the code flow for the away action from UI to server-side and back to UI
- Briefly explain how the server-side ignores the user from turn/round logic

Your Response:

When a player clicks the "Away" button in the UI, the client sends an AWAY message to the server

When a player clicks the "Away" button in the UI, the client sends an AWAY message to the server. The server marks that player as "away" and then tells all other clients so the UI can show the player as inactive or greyed out. While the player is away, the server completely ignores them during the game's logic—meaning they are skipped when checking who is ready, who picked, or who is still in the round. This is done by a simple check like "if the user is away, skip them," so the player doesn't slow down the game or block the next round from starting. When they return, the server updates the UI again and adds them back into the game flow.

Saved: 12/8/2025 7:37:38 PM

Task #2 (1 pt.) - Spectators

Progress: 100%

Details:

- Spectators are users who didn't mark themselves ready
  - Optionally you can include a toggle on the Ready Check page
- The can see all chat but are ignored from turn/round actions and can't send messages
- Spectators will have a visual representation in the user list to distinguish them from other players
- A message should be relayed to the Game Events Panel that a spectator joined (i.e., during an in-progress session)

Part 1:

Progress: 100%

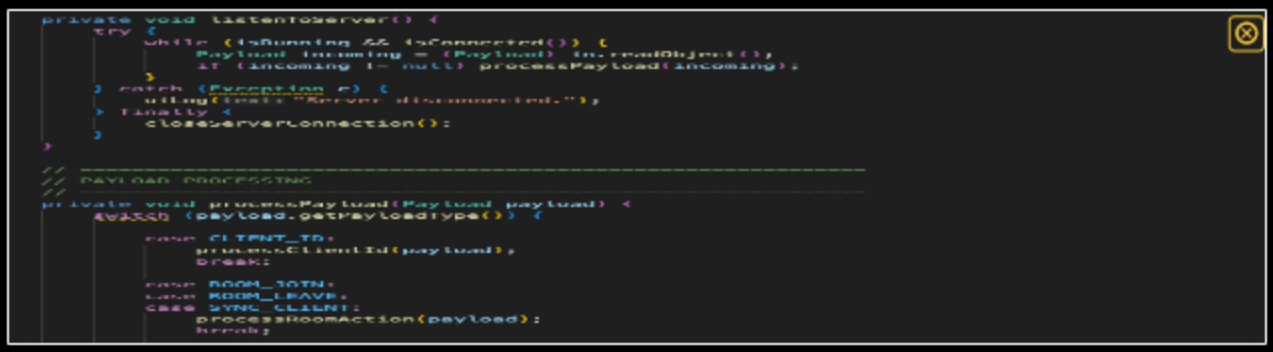
Details:

- Show the UI indicator of a spectator (visual and message)
- Show the related code flow from UI to server-side back to UI for showing the status
- Show the related code flow for sending the message to Game Events Panel
- Show various examples across 3+ clients of spectator status (including Game Events Panel messages)
- Show the code that ignores a spectator from turn/round logic
- Show the code that prevents spectators from sending messages (server-side)
- Show the spectator's view of the session
- Show the code related to the spectator seeing the session data (including things participants won't see)

ui screen

ui screen

ui screen



code

Saved: 12/8/2025 7:41:12 PM

Part 2:

Progress: 100%

Details:

- Briefly explain the code flow for the spectator logic from server-side and to UI
- Briefly explain how the server-side ignores the user from turn/round logic
- Briefly explain the logic that prevents spectators from sending a message
- Briefly explain the logic that shares extra details to the spectator (information normal participants won't see)

Your Response:

When a player becomes a spectator, the server marks them with a special “spectator” flag and immediately sends this update to all clients so the UI can show them as a viewer instead of a participant. Because spectators aren’t playing, the server ignores them during turns and rounds by checking “if spectator, skip,” which keeps the game running smoothly. The server also blocks spectators from sending game actions—so if they try to pick rock, ready up, or send a battle message, the server stops it and returns a warning. However, spectators can receive extra information that normal players can’t see, such as everyone’s choices or battle results in real time, because their client is allowed to display these details without affecting the game. This way, spectators can watch everything without interfering with the round.

Saved: 12/8/2025 7:41:12 PM

Section #5: (1 pt.) Misc

Progress: 100%

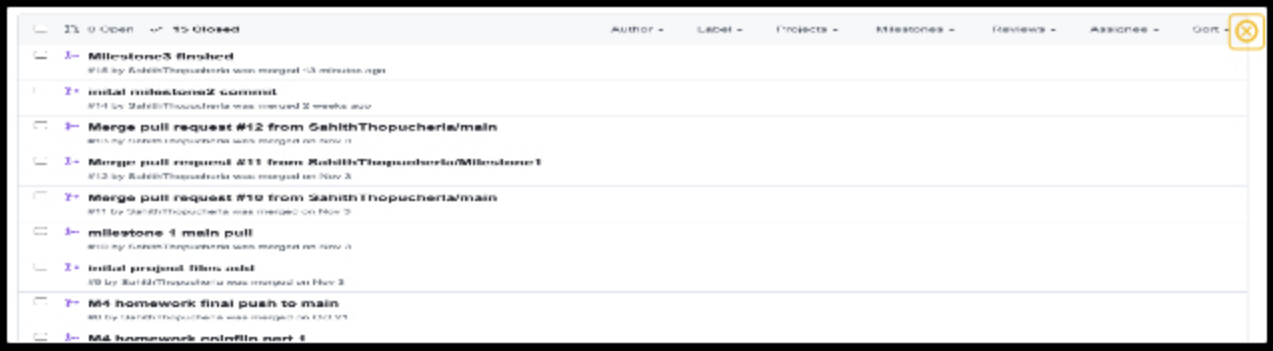
Task #1 (0.33 pts.) - Github Details

Progress: 100%

Part 1:

Progress: 100%

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



git pull

Saved: 12/8/2025 7:40:59 PM

Part 2:

Progress: 100%

Details:
Include the link to the Pull Request for Milestone3 to main (should end in `/pull/#`)

URL #1
https://github.com/SahithThopucherla/IT114_Sahith/pull?q=is%3Apr+is%3Aclosed

URL
https://github.com/SahithThopucl

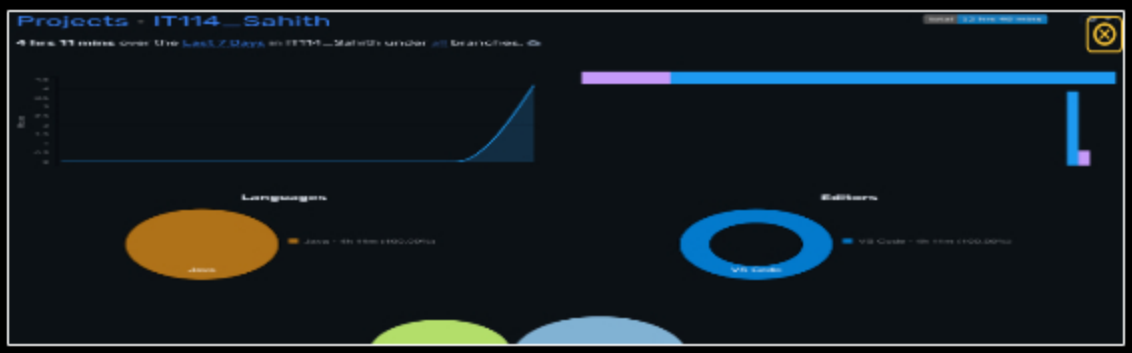
Saved: 12/8/2025 7:40:59 PM

Task #2 (0.33 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



waka

Saved: 12/8/2025 7:42:14 PM

Task #3 (0.33 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:

In this assignment, I learned how the client and server work together to run a multiplayer game. I also learned how to update the UI based on messages coming from the server, like when players are ready, picking choices, or joining rooms. Overall, I gained a better understanding of how networked programs communicate and how to organize code so everything stays in sync.

Saved: 12/8/2025 7:43:15 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 of the assignment was setting up the basic UI and connecting buttons to simple actions. It was straightforward to display text, update labels, and trigger methods when the user clicked something. Once I understood the layout, adding controls and updating the screen felt simple and manageable.

Saved: 12/8/2025 7:43:25 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 getting the UI and server logic to communicate correctly. It took time to understand which payloads were being sent, how the server processed them, and how the UI should update afterward. Keeping track of all the different states—like ready, picking, eliminated, or spectating—was challenging, especially when multiple clients were involved at the same time.

Saved: 12/8/2025 7:43:35 PM