Submission Worksheet

Submission Data

Course: IT114-005-F2025
Assignment: IT114 Milestone 2 - 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: 11/24/2025 4:19:11 PM
Updated: 11/24/2025 11:08:28 PM
Grading Link: https://learn.ethereallab.app/assignment/v3/IT114-005-F2025/it114-milestone-2-rps/grading/st944
View Link: https://learn.ethereallab.app/assignment/v3/IT114-005-F2025/it114-milestone-2-rps/view/st944

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%

≔ Task #1 (1 pt.) - Show Payload classes and subclasses

Progress: 100%

Details:

- Reqs from the document
    - Provided Payload for applicable items that only need client id, message, and type
    - PointsPayload for syncing points of players

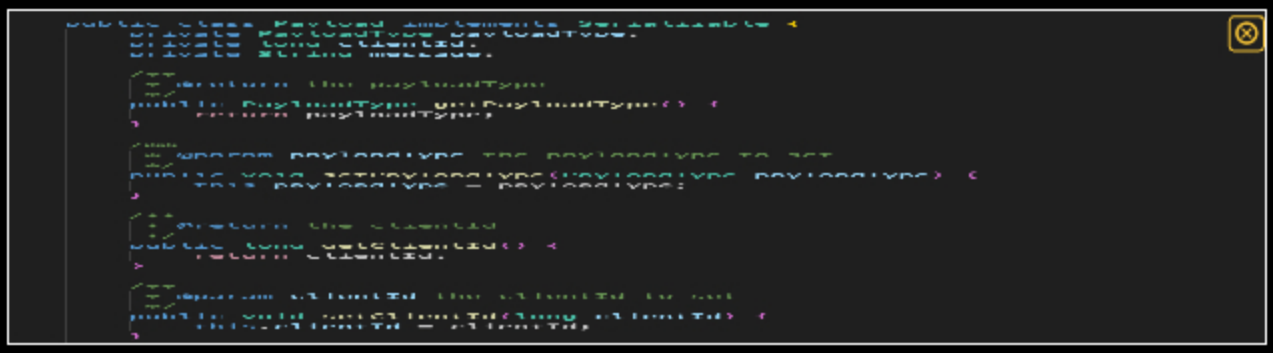
- Each payload will be presented by debug output (i.e, properly override the toString() method like the lesson examples)

Part 1:

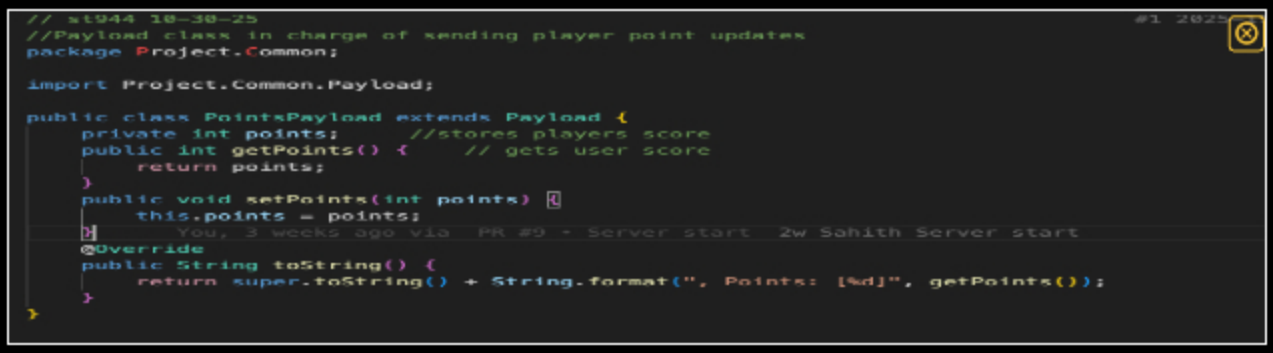
Progress: 100%

Details:

- Show the code related to your payloads (Payload, PointsPayload, and any new ones added)
- Each payload should have an overriden toString() method showing its internal data



code



```
// st944 10-30-25                                                    #1 2025
//Payload class in charge of sending player point updates
package Project.Common;

import Project.Common.Payload;

public class PointsPayload extends Payload {
    private int points;      //stores players score
    public int getPoints() {     // gets user score
        return points;
    }
    public void setPoints(int points) {
        this.points = points;
    }       You, 3 weeks ago via  PR #9 - Server start  2w Sahith Server start
    @Override
    public String toString() {
        return super.toString() + String.format(", Points: [%d]", getPoints());
    }
}
```

code

Saved: 11/24/2025 4:21:28 PM

Part 2:

Progress: 100%

Details:

- Briefly explain the purpose of each payload shown in the screenshots and their properties

Your Response:

The payloads in my project are used to send information between the client and the server. The main Payload class holds the client id, the type of message, and the message itself, and its toString method helps show clear debug output. PayloadType lists all the message types the system uses so the server knows how to handle each one. PointsPayload is used in the game to send a players id and updated points to everyone. Other payloads like ConnectionPayload and

ReadyPayload help with joining the server and marking players as ready. All of them make communication in the game easier to manage and understand.

Saved: 11/24/2025 4:21:28 PM

Section #2: (4 pts.) Lifecycle Events

Progress: 100%

Task #1 (0.80 pts.) - GameRoom Client Add/Remove

Progress: 100%

Part 1:

Progress: 100%

Details:

- Show the `onClientAdded()` code
- Show the `onClientRemoved()` code

```
/** {@inheritDoc} */
//st944 10-30-25
//when user joins syncs phase, ready, turn status.
@Override
protected void onClientAdded(ServerThread sp) {
    // sync GameRoom state to new client
    syncCurrentPhase(sp);
    syncReadyStatus(sp);
    syncTurnStatus(sp);
}
```

code

```
//st944 10-30-25
//cleans up timers and turns
@Override
protected void onClientRemoved(ServerThread sp) {
    // added after Summer 2024 Demo
    // Stops the timers so room can clean up
    LoggerUtil.INSTANCE.info("Player Removed, remaining: " + clientsInRoom.size());
    long removedClient = sp.getClientId();
    turnOrder.removeIf(player -> player.getClientId() == sp.getClientId());
    if (clientsInRoom.isEmpty()) {
        resetReadyTimer();
        resetTurnTimer();
        resetRoundTimer();
        onSessionEnd();
    } else if (removedClient == currentTurnClientId) {
        onTurnStart();
    }
}
```

code

Saved: 11/24/2025 4:25:38 PM

Part 2:

Progress: 100%

Details:

- Briefly note the actions that happen in `onClientAdded()` (app data should at least be synchronized to the joining user)
- Briefly note the actions that happen in `onClientRemoved()` (at least should handle logic for an empty session)

Your Response:

onClientAdded() When a new player joins the room, the game sends them all the current game information so they are caught up. This includes the current phase of the game, who is ready, and whose turn it is. It makes sure the new player is fully synced with everyone else.

onClientRemoved() When a player leaves the room, the game removes them from the turn order and updates the remaining players. If the room becomes empty, the game resets all timers and ends the session so nothing keeps running in the background. If the person who left was the current turn, the next turn starts right away.

Saved: 11/24/2025 4:25:38 PM

Task #2 (0.80 pts.) - GameRoom Session Start

Progress: 100%

Details:

- Reqs from document
  - First round is triggered
- Reset/set initial state

Part 1:

Progress: 100%

Details:

- Show the snippet of `onSessionStart()`

```
//st944 10-30-25
// sets phase to IN_PROGRESS and starts the first round
@Override
protected void onSessionStart() {      You, 3 weeks ago via  PR #9 • Server s
    LoggerUtil.INSTANCE.info("onSessionStart() start");
    changePhase(Phase.IN_PROGRESS);
    currentTurnClientId = Constants.DEFAULT_CLIENT_ID;
    setTurnOrder();
    round = 0;
    LoggerUtil.INSTANCE.info("onSessionStart() end");
    onRoundStart();
}
```

code

Saved: 11/24/2025 4:27:55 PM

Part 2:

Progress: 100%

Details:

- Briefly explain the logic that occurs here (i.e., setting up initial session state for your project) and next lifecycle trigger

Your Response:

When the session starts, the game resets everything so the match begins in a clean state. It sets the phase to in progress, clears the current turn, builds a fresh turn order, and resets the round number back to zero. Once the starting values are set, the next lifecycle step is triggered, which is beginning the first round. This makes the game move smoothly from setup into active play.

Saved: 11/24/2025 4:27:55 PM

Task #3 (0.80 pts.) - GameRoom Round Start

Progress: 100%

Details:

- Reqs from Document
  - Initialize remaining Players' choices to null (not set)
  - Set Phase to "choosing"
  - GameRoom round timer begins

Part 1:

Progress: 100%

Details:

- Show the snippet of `onRoundStart()`

```
//st944 10-30-25
// starts a new round and resets turn status
@Override
protected void onRoundStart() {
    LoggerUtil.INSTANCE.info("onRoundStart() start");
    resetRoundTimer();
    resetTurnStatus();
    //st944 10-30-25
    //starts new round and clears choices/announces round number
    picks.clear();
    round++;
    relay(null, String.format("Round %d has started", round));
    // startRoundTimer(); Round timers aren't needed for turns
    // if you do decide to use it, ensure it's reasonable and based on the number of
    // players
    LoggerUtil.INSTANCE.info("onRoundStart() end");
    onTurnStart();
}
```

code

Saved: 11/24/2025 4:35:18 PM

Saved: 11/24/2025 4:35:18 PM

Part 2:

Progress: 100%

Details:

- Briefly explain the logic that occurs here (i.e., setting up the round for your project)

Your Response:

When a new round starts, the game switches the phase to choosing so players know it is time to make their pick. The round number goes up by one, and each player is updated so their choices are reset. A round timer is then started, which controls how long players have to choose. Once the timer finishes, or everyone has picked, the game moves to the next step of the round.

Saved: 11/24/2025 4:35:18 PM

Task #4 (0.80 pts.) - GameRoom Round End

Progress: 100%

Details:

- Reqs from Document
  - **Condition 1:** Round ends when round timer expires
  - **Condition 2:** Round ends when all active Players have made a choice
  - All Players who are not eliminated and haven't made a choice will be marked as eliminated
  - Process Battles:
    - [ ] Round-robin battles of eligible Players (i.e., Player 1 vs Player 2 vs Player 3 vs Player 1)
      - [ ] Determine if a Player loses if they lose the "attack" or if they lose the "defend" (since each Player has two battles each round)
        - [ ] Give a point to the winning Player
        - [ ] Points will be stored on the Player/User object
        - [ ] Sync the points value of the Player to all Clients
      - [ ] Relay a message stating the Players that competed, their choices, and the result of the battle
      - [ ] Losers get marked as eliminated (Eliminated Players stay as spectators but are skipped for choices and for win checks)
    - [ ] Count the number of non-eliminated Players
      - [ ] If one, this is your winner (onSessionEnd())
      - [ ] If zero, it was a tie (onSessionEnd())
      - [ ] If more than one, do another round (onRoundStart())

Part 1:

Progress: 100%

Details:

- Show the snippet of `onRoundEnd()`

```java
//st944 10-30-25
// ends the round and decides next step (session end or another round)
@Override
protected void onRoundEnd() {
    LoggerUtil.INSTANCE.info("onRoundEnd() start");
    resetRoundTimer(); // reset timer if round ended without the time expiring

    LoggerUtil.INSTANCE.info("onRoundEnd() end");
    if (round >= 3) {
        onSessionEnd();
    } else {
        onRoundStart();
    }
}
```

code

Saved: 11/24/2025 4:36:54 PM

Part 2:

Progress: 100%

Details:

- Briefly explain the logic that occurs here (i.e., cleanup, end checks, and next lifecycle events)

Your Response:

When the round ends, the game checks which players did not make a choice and marks them as eliminated. Then it looks at the players who are still active and runs the battles between them to decide who won each matchup. After the battles, the game counts how many players are still not eliminated. If only one player is left, that person wins the whole game. If no one is left, the game ends in a tie. If more than one player is still active, the game starts another round. This makes sure each round is cleaned up correctly and the next part of the game happens in the right order.

Saved: 11/24/2025 4:36:54 PM

Task #5 (0.80 pts.) - GameRoom Session End

Progress: 100%

Details:

- Reqs from Document
  - **Condition 1:** Session ends when one Player remains (they win)
  - **Condition 2:** Session ends when no Players remain (this is a tie)
  - Send the final scoreboard to all clients sorted by highest points to lowest (include a game over message)
  - Reset the player data for each client server-side and client-side (do not disconnect them or move them to the lobby)

- A new ready check will be required to start a new session

Part 1:

Progress: 100%

Details:

- Show the snippet of `onSessionEnd()`

```
// st944 10-30-25
// cleans up after session and returns to READY phase
@Override
protected void onSessionEnd() {
    LoggerUtil.INSTANCE.info("onSessionEnd() start");
    turnOrder.clear();
    currentTurnClientId = Constants.DEFAULT_CLIENT_ID;
    resetReadyStatus();
    resetTurnStatus();
    changePhase(Phase.READY);
    LoggerUtil.INSTANCE.info("onSessionEnd() end");
}
// end lifecycle methods
```

code

Saved: 11/24/2025 7:08:53 PM

Part 2:

Progress: 100%

Details:

- Briefly explain the logic that occurs here (i.e., cleanup/reset, next lifecycle events)

Your Response:

When the session ends, the game sends out the final scoreboard to all players and shows who won or if it was a tie. After that, each player is reset so they can start a new game without leaving the room. All timers and round data are cleared so nothing continues running in the background. The phase then switches back to the ready state, which means the next step is for players to mark ready again before a new session can begin.

Saved: 11/24/2025 7:08:53 PM

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)

```
//accepts picks and sends to server
else if (text.startsWith("pick")) {
    String arg = text.replaceFirst("^pick\\s*", "").trim();
    if (arg.isEmpty()) {
        LoggerUtil.INSTANCE.warning(TextFX.colorize("Usage: /pick <r|p|s>", Color.RED));
        return true;
    }
    String choice = arg.substring(0,1).toLowerCase();
    if (!choice.equals("r") && !choice.equals("p") && !choice.equals("s")) {
        LoggerUtil.INSTANCE.warning(TextFX.colorize("Invalid choice. Use r, p, or s.", Color.RED));
        return true;
    }
    sendPick(choice);
    LoggerUtil.INSTANCE.info(TextFX.colorize("You picked: " + choice, Color.GREEN));
    wasCommand = true;
}

return wasCommand;
```

code

```
    //st944 10-30-25
    //routes PICK payload to active Gameroom
    case PICK:
        try {
            ((GameRoom) currentRoom).handlePick(this, incoming.getMessage());
        }
        catch (Exception e) {
            sendMessage(Constants.DEFAULT_CLIENT_ID, "You must be in a Gameroom to pick.");
        }
        break;
    default:
        LoggerUtil.INSTANCE.warning(TextFX.colorize("Unknown payload type received", Color.RED));
        break;
}
```

code

```
//st944 10-30-25
//handles rps choices
protected void handlePick(ServerThread currentUser, String choice) {      You, 3 weeks ago via  PR #9 • Server Sta
    try {
        checkPlayerInRoom(currentUser);
        checkCurrentPhase(currentUser, Phase.IN_PROGRESS);
        checkIsReady(currentUser);
        picks.put(currentUser.getClientId(), choice.toLowerCase());
```

```
            currentUser.setTookTurn(true);
            sendTurnStatus(currentUser, true);
            checkAllTookTurn();
        }
        catch(Exception e) {
            currentUser.sendMessage(Constants.DEFAULT_CLIENT_ID, "You can't pick right now.");
        }
    }
```

code

```
    // start send*() methods
    public boolean sendResetTurnStatus() {
        ReadyPayload rp = new ReadyPayload();
        rp.setPayloadType(PayloadType.RESET_TURN);
        return sendToClient(rp);
    }

    public boolean sendTurnStatus(long clientId, boolean didTakeTurn) {
        return sendTurnStatus(clientId, didTakeTurn, false);
    }

    public boolean sendTurnStatus(long clientId, boolean didTakeTurn, boolean quiet) {
        // Null 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);
    }

    public boolean sendCurrentPhase(Phase phase) {
        Payload p = new Payload();
        p.setPayloadType(PayloadType.PHASE);
        p.setMessage(phase.name());
        return sendToClient(p);
    }

    public boolean sendResetReady() {
        ReadyPayload rp = new ReadyPayload();
        rp.setPayloadType(PayloadType.RESET_READY);
```

code

```
//st944 10-30-25
// starts a new round and resets turn status
@Override
protected void onRoundStart() {
    LoggerUtil.INSTANCE.info("onRoundStart() start");
    resetRoundTimer();
    resetTurnStatus();
    //st944 10-30-25
    //starts new round and clears choices/announces round number
    picks.clear();
    round++;
    relay(null, String.format("Round %d has started", round));      You, 3 weeks ago via  PR #9 -
    // startRoundTimer(); Round timers aren't needed for turns
    // if you do decide to use it, ensure it's reasonable and based on the number of
    // players
    LoggerUtil.INSTANCE.info("onRoundStart() end");
    onTurnStart();
}
```

code

```
        break;
    case MOVEAVL:
        processMoveAvl(payload);
        break;
    case REVERSE:
        processReverse(payload);
        break;
    case ROOM_CLAIL:  // unused
        break;                                  You, 3 weeks ago via  PR #9 - Initial project files
    case ROOM_JOIN:
        processRoomAction(payload);
        break;
    case ROOM_LEAVE:
        processRoomAction(payload);
        break;
    case SYNC_CLIENT:
        processRoomAction(payload);
        break;
    case ROOM_LIST:
        processRoomList(payload);
        break;
    case READY:
        processReadyStatus(payload, true);
        break;
    case SYNC_READY:
        processReadyStatus(payload, true);
        break;
    case RESET_READY:
```

code

Saved: 11/24/2025 7:45:05 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:

When a player enters the pick command on the client, the client reads the choice and creates a payload that contains the selected option. The client then sends this payload to the server. When the server receives it, the ServerThread checks the type of the payload and passes the choice to the GameRoom. The GameRoom checks if the option is valid and records it on the players data. It then sends a message to all players telling them that the user picked their choice. The ServerThread sends this message back to every connected client. Finally, each client receives the message and prints it to the screen so everyone can see the update.

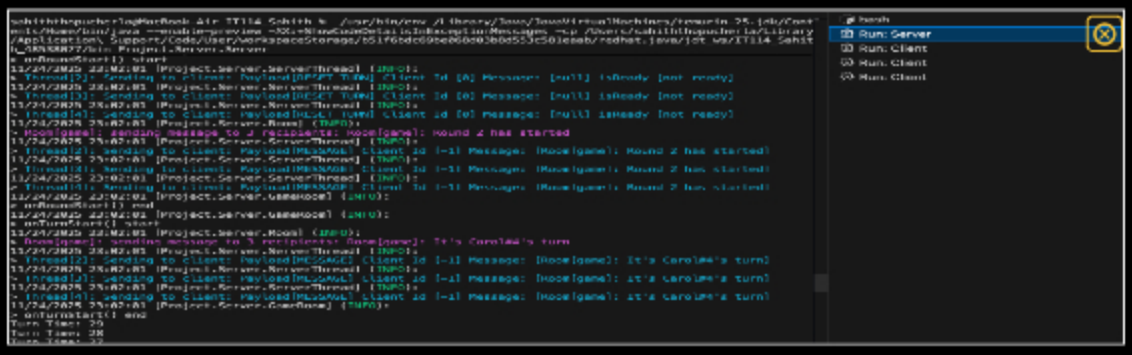
Saved: 11/24/2025 7:45:05 PM

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

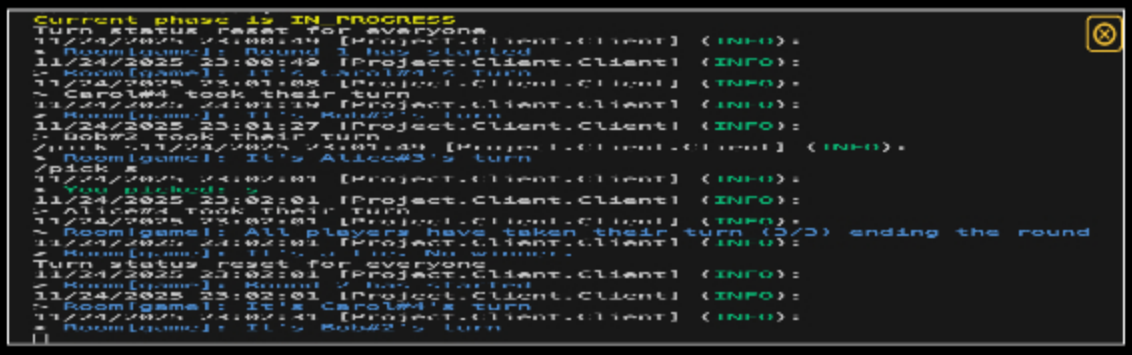
Progress: 100%

Details:

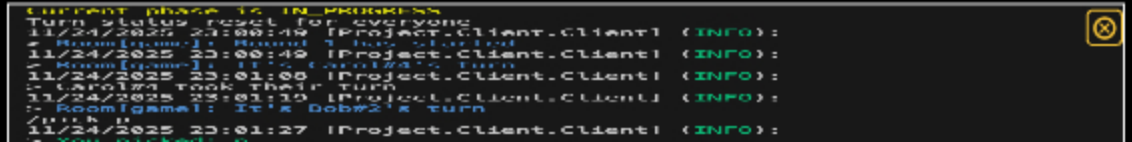
- Show examples from the terminal of a full session demonstrating each command and progress output
- This includes battle outcomes, scores and scoreboards, etc
- Ensure at least 3 Clients and the Server are shown
- Clearly caption screenshots

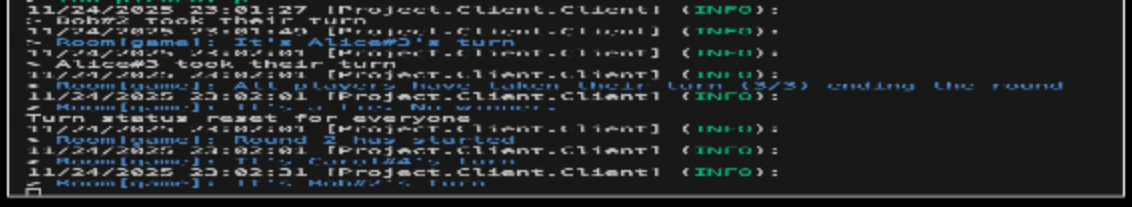


server

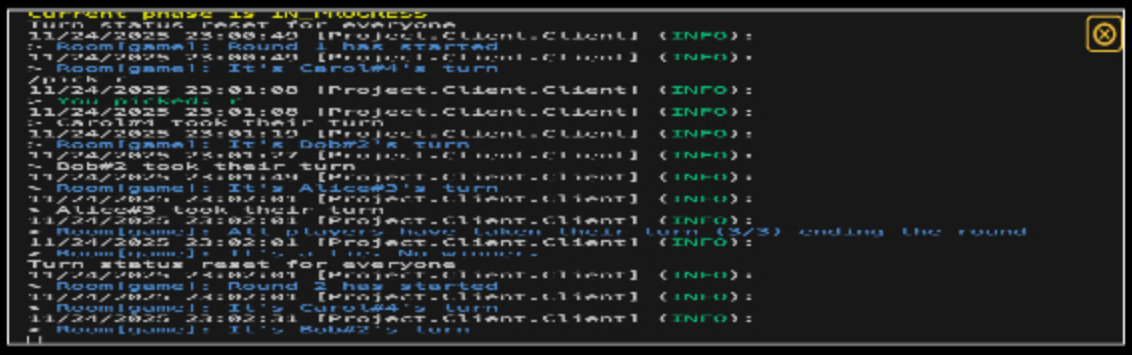


client 1





client 2



client 3

Saved: 11/24/2025 11:04:25 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

Saved: 11/24/2025 11:06:28 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

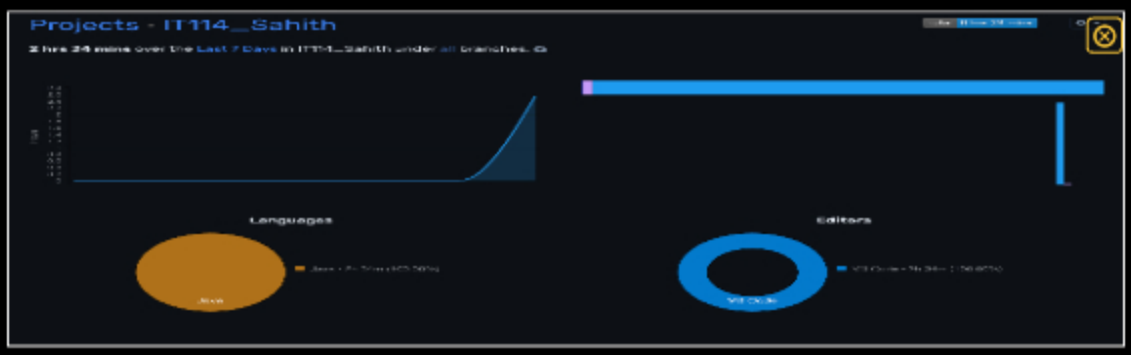
Saved: 11/24/2025 11:06:28 PM

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: 11/24/2025 11:07:20 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:

I learned how the client and server communicate to run a full multiplayer game. I saw how payloads send information between users and how the GameRoom handles things like joining, starting rounds, taking turns, and ending the session. I also learned how important it is to follow the command flow correctly and how to test the game using multiple clients at the same time. Overall, this milestone helped me understand how networking and game logic work together.

Saved: 11/24/2025 11:08:10 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 this assignment was running the server and clients once everything was set up. Using commands like /name, /connect, and /joinroom felt simple after I understood how they worked. It was also easy to see the game logs update in real time, which helped me confirm that the system was working correctly.

Saved: 11/24/2025 11:08:17 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 understanding how rooms were created and why the "game" room didn't exist at first. I had to figure out how commands were processed on both the client and server sides. Setting up three clients, testing turns, and getting the full cycle to work took time because one small mistake in the command flow could stop the whole session. Debugging these issues helped me learn more about how the system works.

Saved: 11/24/2025 11:08:28 PM