Submission Worksheet

Submission Data

Course: IT114-450-M2025
Assignment: IT114 Milestone 2 - RPS
Student: Mukaddis I. (mi348)
Status: Submitted | **Worksheet Progress:** 100%
Potential Grade: 10.00/10.00 (100.00%)
Received Grade: 0.00/10.00 (0.00%)
Started: 7/21/2025 3:52:04 PM
Updated: 7/21/2025 5:36:15 PM
Grading Link: https://learn.ethereallab.app/assignment/v3/IT114-450-M2025/it114-milestone-2-rps/grading/mi348
View Link: https://learn.ethereallab.app/assignment/v3/IT114-450-M2025/it114-milestone-2-rps/view/mi348

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

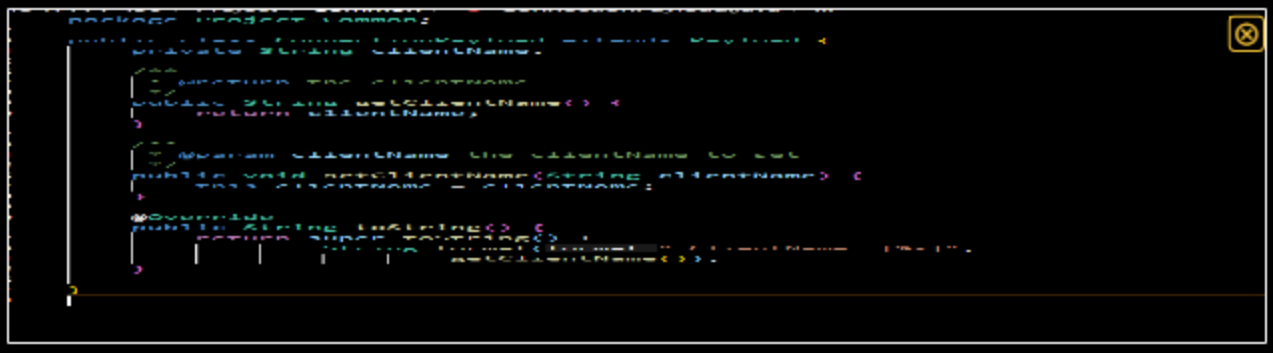
- PointsPayload for syncing points of players
- Each payload will be presented by debug output (i.e, properly override the toString() method like the lesson examples)

Part 1:

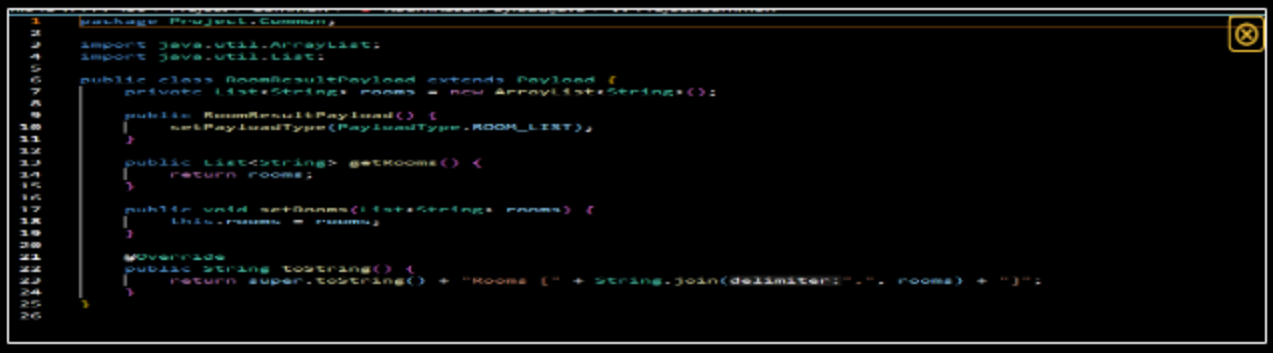
Progress: 100%

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

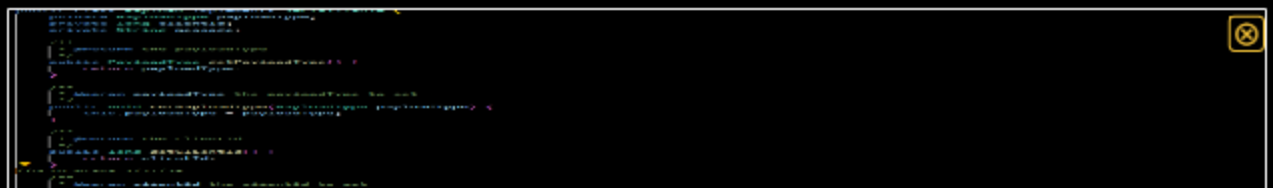


Payload.java

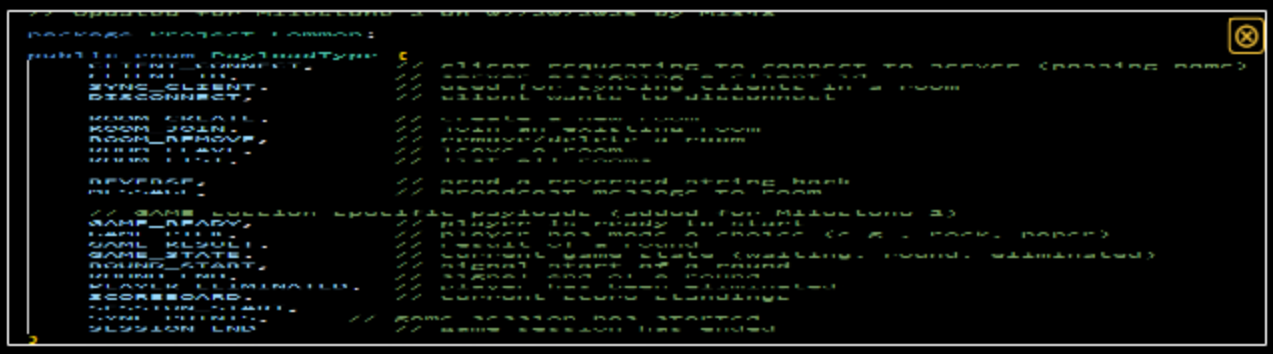


PointsPayload.java

RoomResultPayload.java



ConnectionPayload.java



PayloadPayloadType.java

Saved: 7/21/2025 4:07:41 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

The Payload class is the base message object that includes the sender's client ID, message text, and a PayloadType to indicate what kind of message it is. The PointsPayload extends it to carry game-specific data like the player's score and elimination status during Rock-Paper-Scissors sessions. RoomResultPayload is used to send back a list of room names when a client requests available rooms. ConnectionPayload is used for join/leave notifications and carries both the client ID and name to help synchronize the client list.

Saved: 7/21/2025 4:07:41 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

```
    return clientsInRoom.values();
}
//UCID MI348 7/21/25
protected synchronized void addClient(ServerThread client) {
    if (!isRunning || clientsInRoom.containsKey(client.getClientId())) return;
    clientsInRoom.put(client.getClientId(), client);
    client.setCurrentRoom(this);
    client.sendResetUserList();
    syncExistingClients(client);
    joinStatusRelay(client, didJoin:true);
}

protected synchronized void removeClient(ServerThread client) {
    if (!isRunning || !clientsInRoom.containsKey(client.getClientId())) return;
    ServerThread removedClient = clientsInRoom.remove(client.getClientId());
    if (removedClient != null) {
        joinStatusRelay(removedClient, didJoin:false);
        autoCleanup();
    }
```

It is addclient and removeclient instead

Saved: 7/21/2025 4:20:08 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:

When a client joins, the server assigns them to the current room and resets their user list. It then synchronizes the existing room users to the new client and informs all other clients about the new user. This ensures everyone sees up-to-date user data. When a client leaves, they are removed from the room, a leave notification is sent to others, and if the room is empty, it is closed to clean up the session.

Saved: 7/21/2025 4:20:08 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()`

```
        this.room = room;
    }
//Mikaa DATE 7/21/25
    // Starts the game session
    private void startSession() {
        inProgress = true;
        round = 0;
        LoggerUtil.INSTANCE.info(TextFX.colorize(text:"GameSession: Starting game", Color.GREEN));
        for (ServerThread client : room.getClients()) {
            PlayerState p = new PlayerState(client);
            players.put(client.getClientId(), p);
        }
        startRound();
    }

    // Begins a new round
    private void startRound() {
        round++;
        LoggerUtil.INSTANCE.info(TextFX.colorize("GameSession: Starting Round " + round, Color.YELLOW));
        players.values().forEach(p -> {
            if (!p.isEliminated()) {
                p.setPick(pick:null);
            }
        });

        broadcast("Starting Round " + round + ". Make your pick: /pick <r|p|s>");
```

I did startsession instead of OnsessionStart

Saved: 7/21/2025 4:28:25 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:

The startSession() method sets the game as in progress and resets the round number to zero. It initializes each connected client as a PlayerState and stores them in the players map to track their game status. After setup, it triggers the start of the first round by calling startRound(), which prompts players to make their picks. The next lifecycle trigger occurs when all active players have submitted a valid pick or the round timer expires, leading to endRound().

Saved: 7/21/2025 4:28:25 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()`

```java
// Begins a new round
private void startRound() {
    round++;
    LoggerUtil.INSTANCE.info(TextFX.colorize("GameSession: Starting Round " + round, Color.YELLOW));
    players.values().forEach(p -> {
        if (!p.isEliminated()) {
            p.setPick(pick:null);
        }
    });

    broadcast("Starting Round " + round + ". Make your pick: /pick <r|p|s>");
    startRoundTimer();
}
```

StartRound

Saved: 7/21/2025 4:34:41 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

The startRound() method initializes a new round by incrementing the round counter and resetting each active (non-eliminated) player's pick to null. It logs the start of the round and prompts all players to make a move by broadcasting a message. This ensures everyone starts the round with a clean state. Finally, it sets a 30-second timer to automatically end the round if not all players submit their picks.

Saved: 7/21/2025 4:34:41 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()`

```
private void endRound() {
    stopRoundTimer();
    LoggerUtil.INSTANCE.info(TextFX.colorize("GameSession: Ending Round " + round, Color.RED));

    List<PlayerState> active = getActivePlayers();
    List<PlayerState> missingPick = new ArrayList<>();
    for (PlayerState p : active) {
        if (p.getPick() == null) {
            p.setEliminated(eliminated:true);
            missingPick.add(p);
        }
    }

    for (PlayerState p : missingPick) {
        broadcast(p.getName() + " was eliminated (no pick)");
    }

    resolveBattles();
    evaluateGameStatus();
}
```

RoundEnd

Saved: 7/21/2025 4:38:29 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

The endRound() method first stops the round timer and logs that the round is ending. It checks

which active players failed to submit a pick and eliminates them from the game. Then, it resolves all battles between players to determine round winners and updates points. Finally, it checks if the game should continue with another round or end by evaluating remaining active players.

Saved: 7/21/2025 4:38:29 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()`

sessionEnd

Saved: 7/21/2025 4:41:41 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

ChatGPT said: The endSession() method handles the cleanup and finalization of the game. It marks the session as no longer in progress and stops the round timer if it's active. It then sorts all players by their scores in descending order and broadcasts the final scoreboard to all clients. Finally, it clears the player list to reset the session state, preparing for any future sessions.

Saved: 7/21/2025 4:41:41 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:

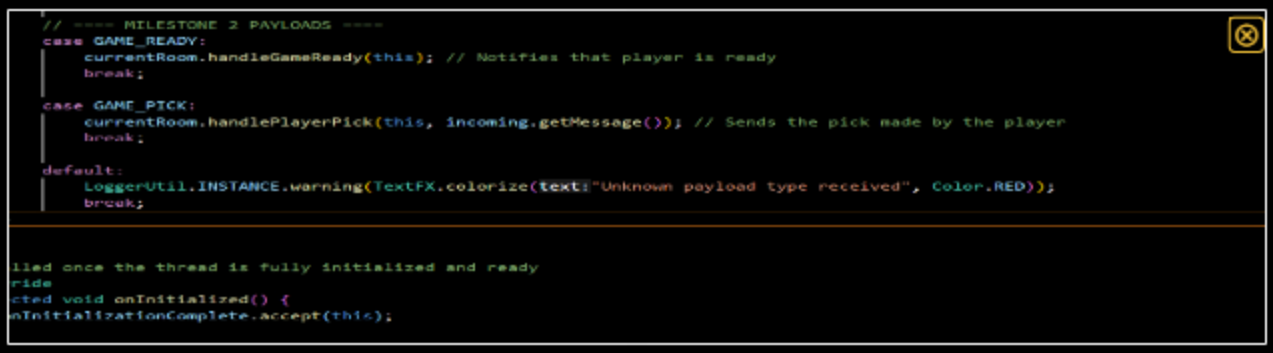
Progress: 100%

Details:

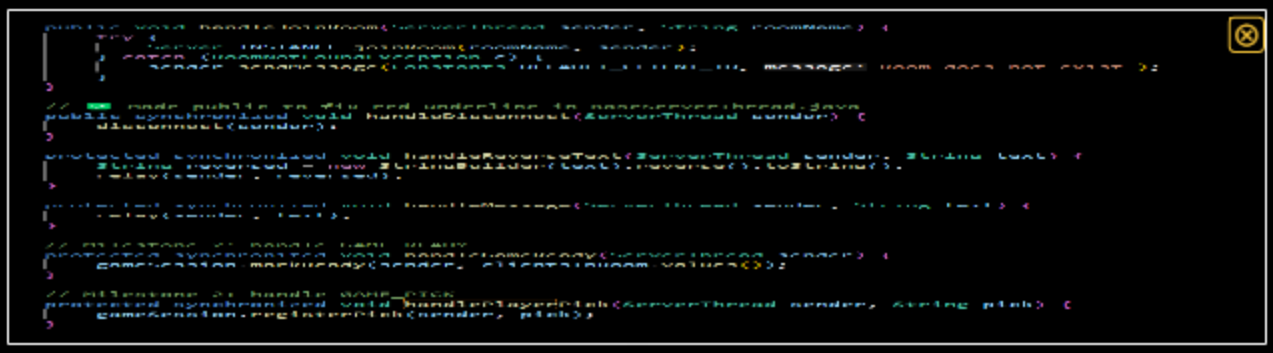
- Show the code snippets of the following, and clearly caption each screenshot
- Show the Client processing of this command (process client command)
- Show the ServerThread processing of this command (process method)
- Show the GameRoom handling of this command (handle method)
- Show the sending/syncing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

```
        sendRoomAction(text, RoomAction.JOIN);
        wasCommand = true;
    } else if (text.startsWith(Command.LEAVE_ROOM.command)) {
        sendRoomAction(text, RoomAction.LEAVE);
        wasCommand = true;
    } else if (text.startsWith(Command.LIST_ROOMS.command)) {
        text = text.replace(Command.LIST_ROOMS.command, replacement:"").trim();
        sendRoomAction(text, RoomAction.LIST);
        wasCommand = true;
    } else if (text.equalsIgnoreCase(Command.READY.command)) {
        sendGameReady();
        wasCommand = true;
    } else if (text.startsWith(Command.PICK.command)) {
        text = text.replace(Command.PICK.command, replacement:"").trim();
        if (text.isEmpty()) {
            LoggerUtil.INSTANCE.warning(TextFX.colorize(text:"Usage: /pick <option>", Color.RED));
            return true;
        }
```

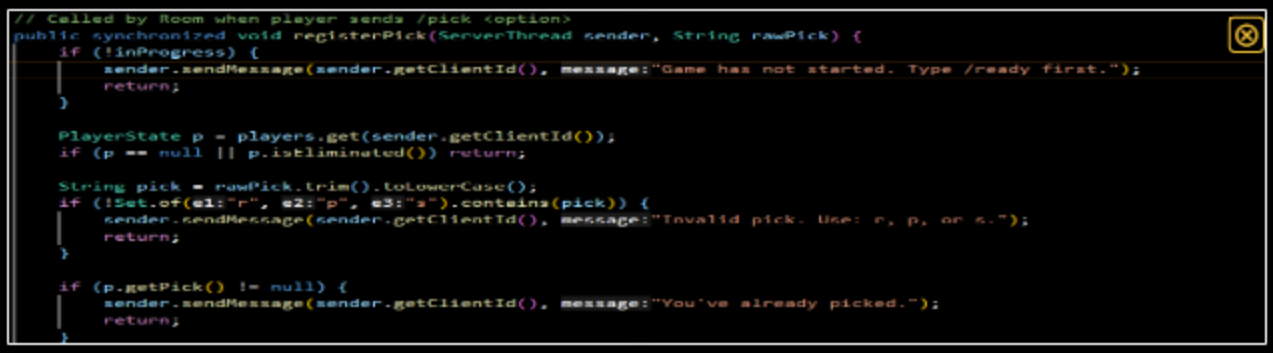
/pick option



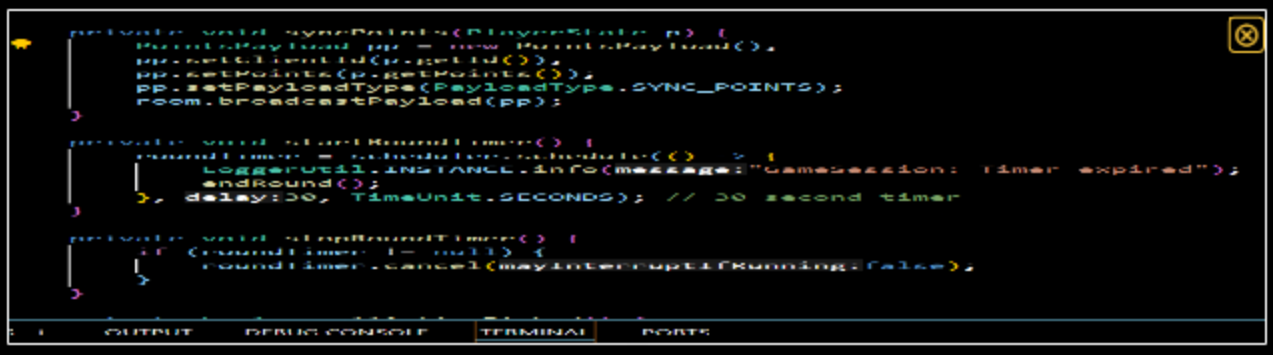
Game_PICK/Ready



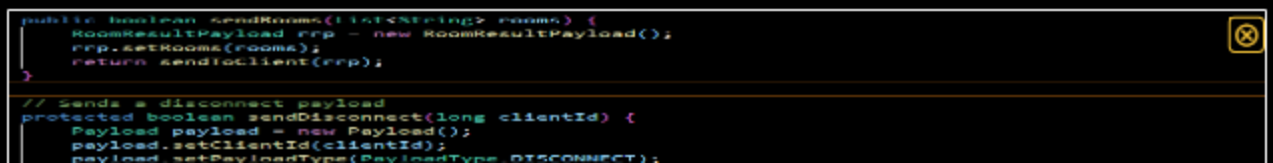
handling



Registering Pick



SyncPoints



```
        return sendToClient(payload);
    }

    // Resets the user list on the client side
    protected boolean sendResetUserList() {
        return sendClientInfo(Constants.DEFAULT_CLIENT_ID, clientName:null, RoomAction.JOIN);
    }

    // Sends client info (join/leave) to another client
    protected boolean sendClientInfo(long clientId, String clientName, RoomAction action) {
        return sendClientInfo(clientId, clientName, action, isSync:false);
    }
```

sending payload to client

Saved: 7/21/2025 5:26:36 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 client types /pick <r|p|s>, the client constructs a Payload of type GAME_PICK containing the chosen option and sends it to the server. On the server, ServerThread receives this payload and calls handlePlayerPick() in the associated Room. The Room forwards this to GameSession, which registers the pick and, if all active players have picked, triggers the end of the round. The round results are calculated, and any points earned are sent to clients using a PointsPayload via room.broadcastPayload(). Each client receives this payload and displays updated scores, completing the round flow.

Saved: 7/21/2025 5:26:36 PM

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

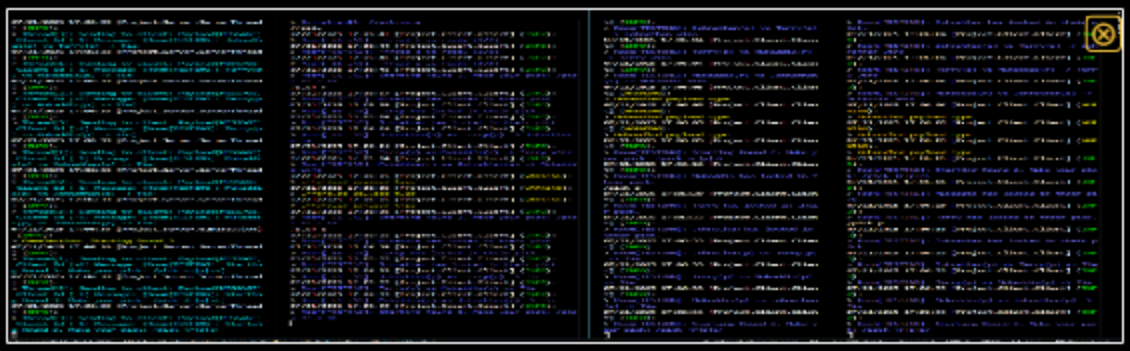
Progress: 100%

Details:

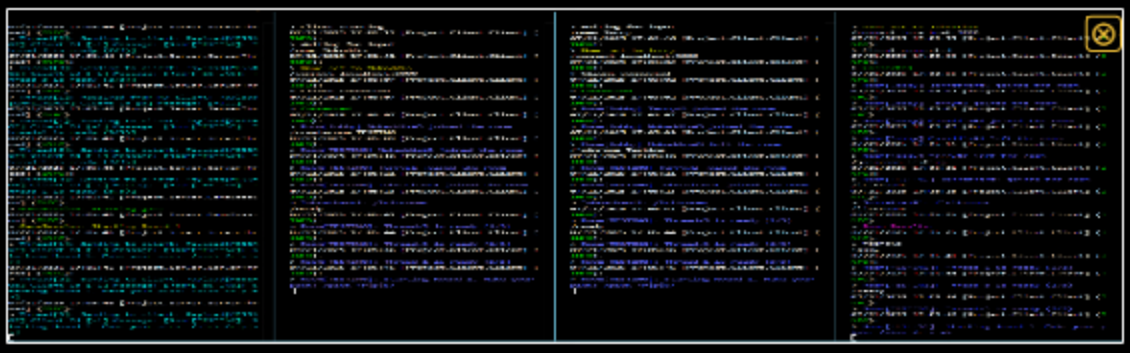
- Show examples from the terminal of a full session demonstrating each command and progress output
- This includes battle outcomes, scores and scoreboards, etc
- Ensure at least 3 Clients and the Server are shown
- Clearly caption screenshots

```
7/21/2025 17:09:46 [Project.Client.Client] (INFO):
 Room[TESTING]: == FINAL SCORES ==
7/21/2025 17:09:46 [Project.Client.Client] (INFO):
 Room[TESTING]: Mukaddis - 7 point(s)
7/21/2025 17:09:46 [Project.Client.Client] (INFO):
 Room[TESTING]: Johnathan - 4 point(s)
7/21/2025 17:09:46 [Project.Client.Client] (INFO):
 Room[TESTING]: Jerry - 4 point(s)
```

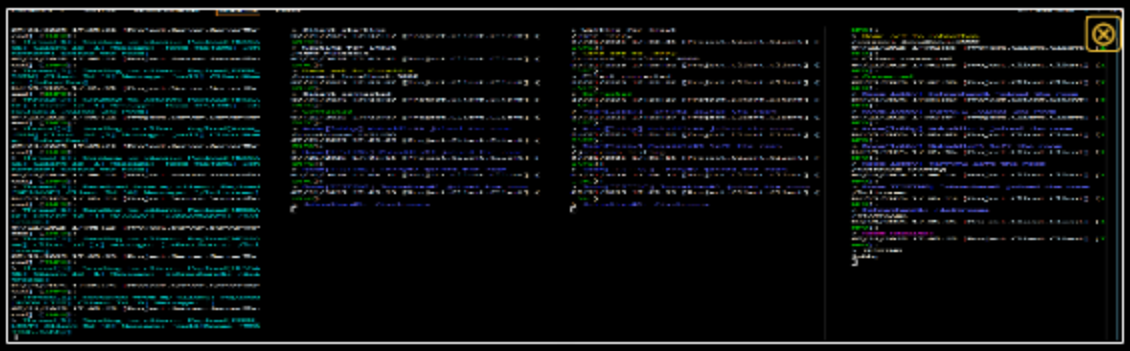
declaring winner



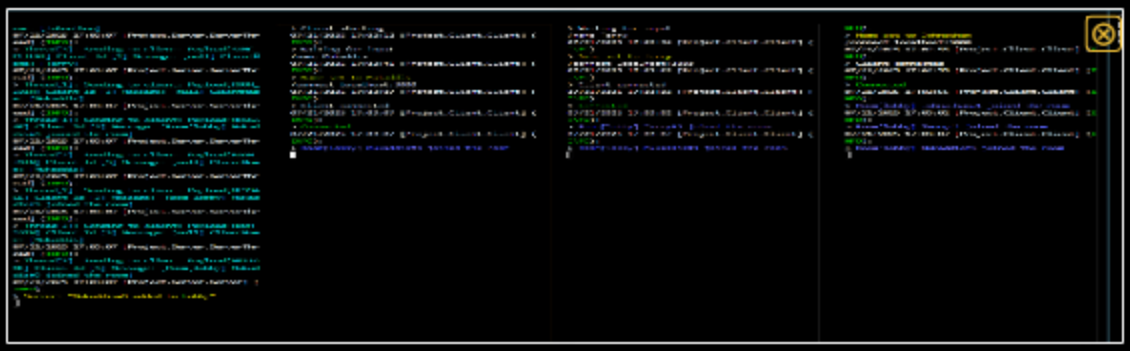
picking r p or s



Readying up



Creating/Joining Room



Connecting to Server

Saved: 7/21/2025 5:11:59 PM

Section #4: (1 pt.) Misc

Progress: 100%

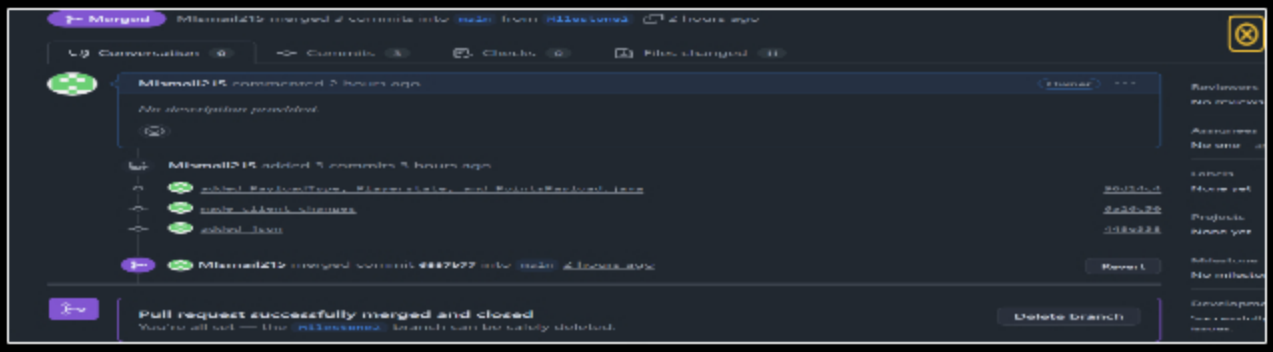
Task #1 (0.33 pts.) - Github Details

Progress: 100%

Part 1:

Progress: 100%

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



commits

Saved: 7/21/2025 5:28:17 PM

Part 2:

Progress: 100%

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

URL #1
https://github.com/MIsmail215/mi348-IT114-450

URL
https://github.com/MIsmail215/m

Saved: 7/21/2025 5:28:17 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

Bottom Half

Top Half

Saved: 7/21/2025 5:29:33 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 to make a game session work for Rock-Paper-Scissors using a server and multiple clients. I learned how to start a game when all players are ready and how to track their picks. I also learned how to show scores and eliminate players. The project taught me how to sync game data between server and clients.

Saved: 7/21/2025 5:36:01 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 setting up the /ready and /pick commands. It was simple to track if a player was ready or what move they picked. Sending messages to all players was also not too hard. The structure was already set up from Milestone 1.

Saved: 7/21/2025 5:36:10 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 game to handle rounds correctly. It was tricky to make sure the game started at the right time and ended when only one player was left. Making sure players who didn't pick were eliminated also took work. Debugging the session reset and point syncing was the most difficult.

Saved: 7/21/2025 5:36:15 PM