Submission Worksheet

Submission Data

Course: IT114-003-F2025
Assignment: IT114 Milestone 2 - RPS
Student: Rayyan K. (rk975)
Status: Submitted | **Worksheet Progress:** 100%
Potential Grade: 10.00/10.00 (100.00%)
Received Grade: 0.00/10.00 (0.00%)
Started: 11/26/2025 3:19:04 PM
Updated: 11/26/2025 5:19:44 PM
Grading Link: https://learn.ethereallab.app/assignment/v3/IT114-003-F2025/it114-milestone-2-rps/grading/rk975
View Link: https://learn.ethereallab.app/assignment/v3/IT114-003-F2025/it114-milestone-2-rps/view/rk975

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)

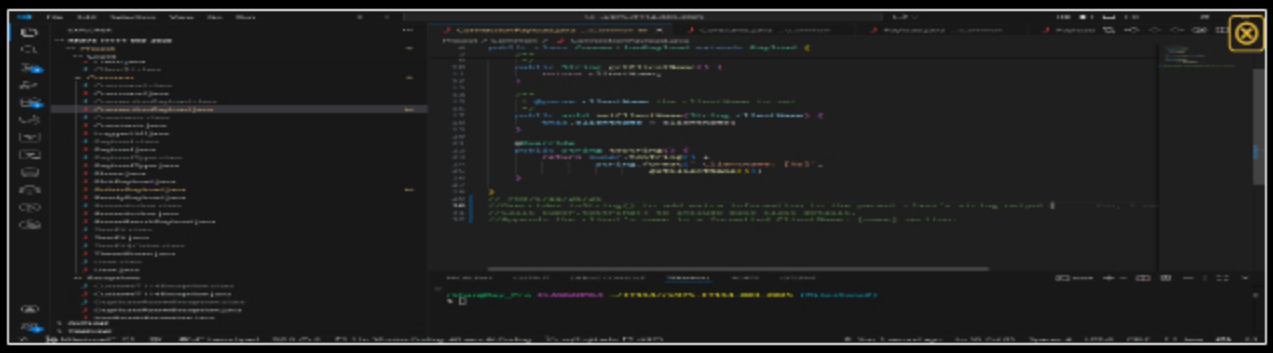
Part 1:

Progress: 100%

Details:

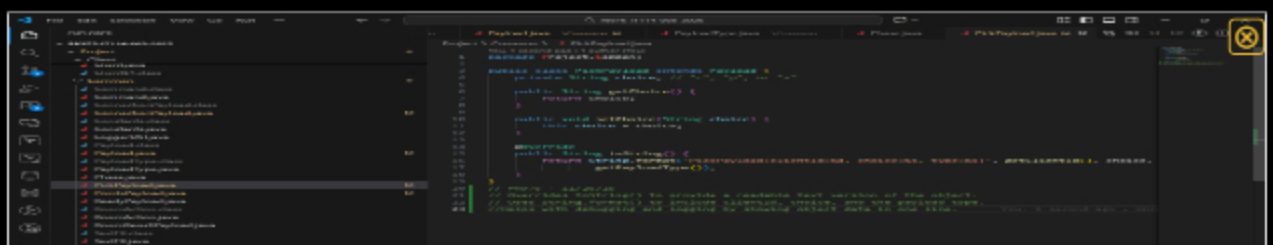
- Show the code related to your payloads (Payload, PointsPayload, and any new ones added)
- Each payload should have an overriden toString() method showing its internal data

pointspayload

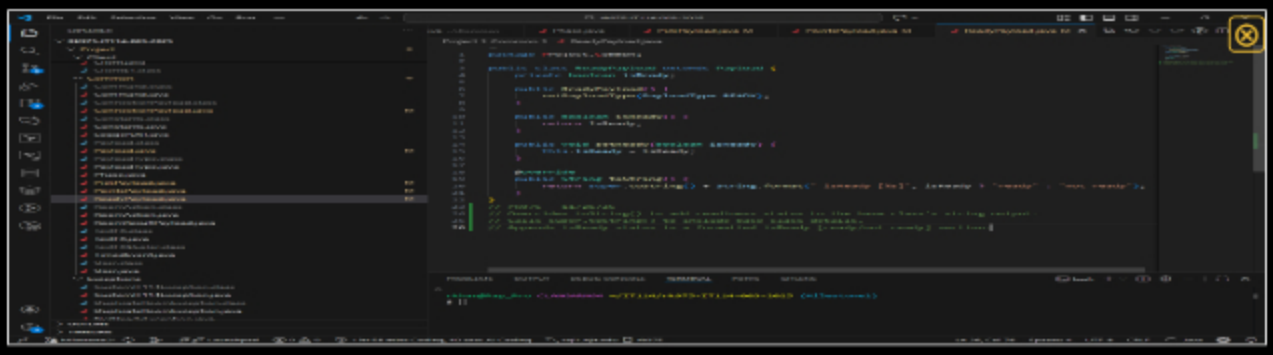


connectionpayload

payload



pickpayload



readypayload

roomresultpayload

Saved: 11/26/2025 3:29:55 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

PointsPayload serves the purpose of transmitting point/score information between the client and server during gameplay. It extends the base Payload class and contains a single property points (int) that stores the score value. The class includes getter and setter methods (getPoints() and setPoints()) to access and modify the points value. The toString() method is overridden to provide a readable string representation that displays the clientId, points value, and payload type in a formatted string like PointsPayload{clientId=1, points=100, type=POINTS}, which aids in debugging and logging by showing all the object's internal data in a single line.

Saved: 11/26/2025 3:29:55 PM

Section #3: (4 pts) Lifecycle Events

Section #2: (4 pts.) Lifecycle Events

Progress: 100%

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

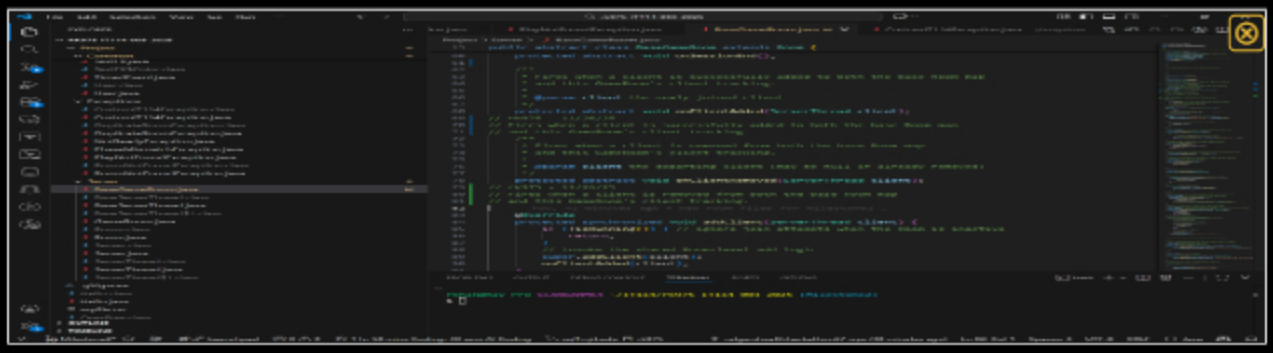
Progress: 100%

Part 1:

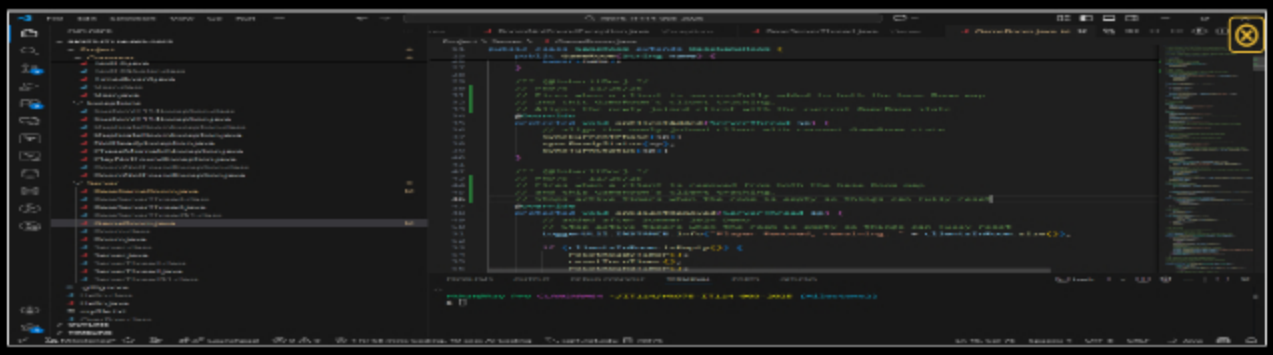
Progress: 100%

Details:

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



client add/remove



client add/remove

Saved: 11/26/2025 3:35:21 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:

In order to ensure that the new client is completely in sync with the ongoing session, the GameRoom instantly synchronizes all significant game state to the newly joined player when onClientAdded() executes. This includes the current game phase, everyone's ready status, and each player's turn status. This guarantees the player joins with a precise view of the room and avoids desync. The GameRoom logs the removal and determines whether the room is now empty when onClientRemoved() is executed. All current timers (ready, round, and turn timers) are stopped if there are no players left, and onSessionEnd() is triggered to completely restart the session. This prevents the GameRoom from executing pointless game logic after everyone has departed.

Saved: 11/26/2025 3:35:21 PM

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

Progress: 100%

Details:

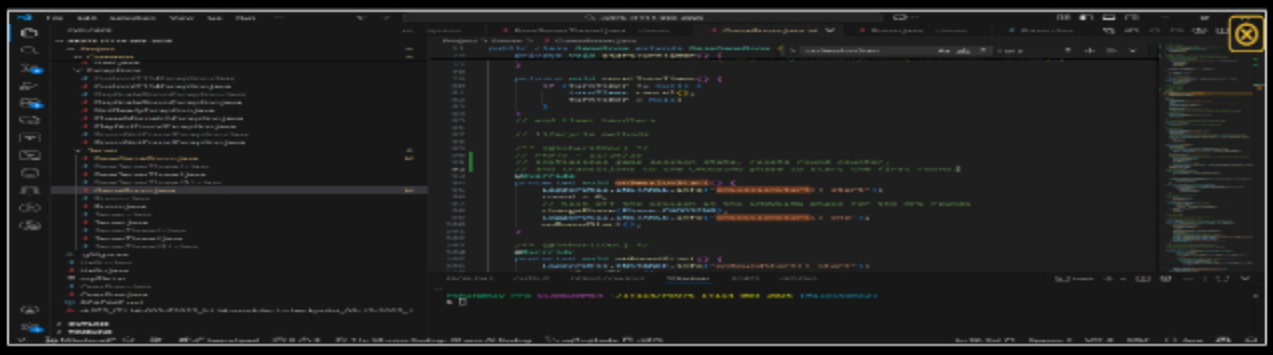
- Reqs from document
  - First round is triggered
- Reset/set initial state

Part 1:

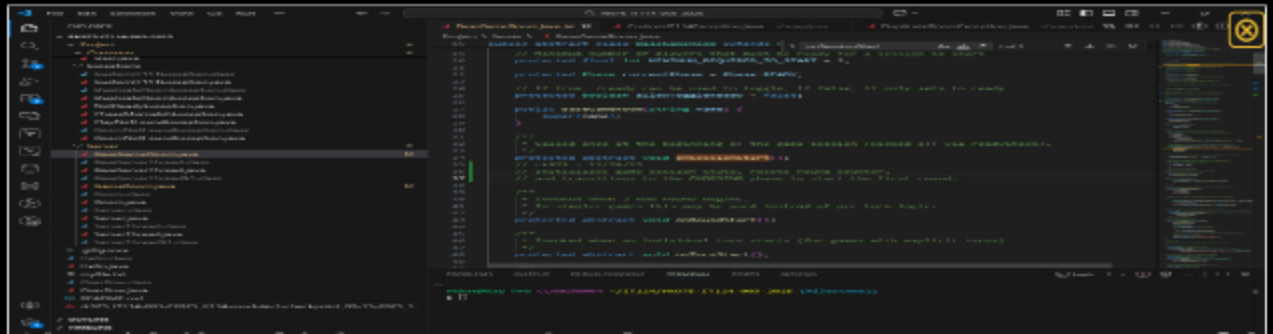
Progress: 100%

Details:

- Show the snippet of `onSessionStart()`



session start



session start

Saved: 11/26/2025 3:38:27 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 GameRoom initiates the session by resetting the round counter and moving the game into the CHOOSING phase, which initiates gameplay for all connected players, when onSessionStart() is executed. Additionally, it logs the session's beginning and instantly initiates onRoundStart(), advancing the project to the next stage of its lifetime. The room resets timers, clears all player options, increases the round number, and alerts players that a new round has begun inside onRoundStart(). After that, it starts the round timer and waits for players to enter their data. The session starts smoothly and moves straight into the active game phase thanks to this lifecycle transition.

Saved: 11/26/2025 3:38:27 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

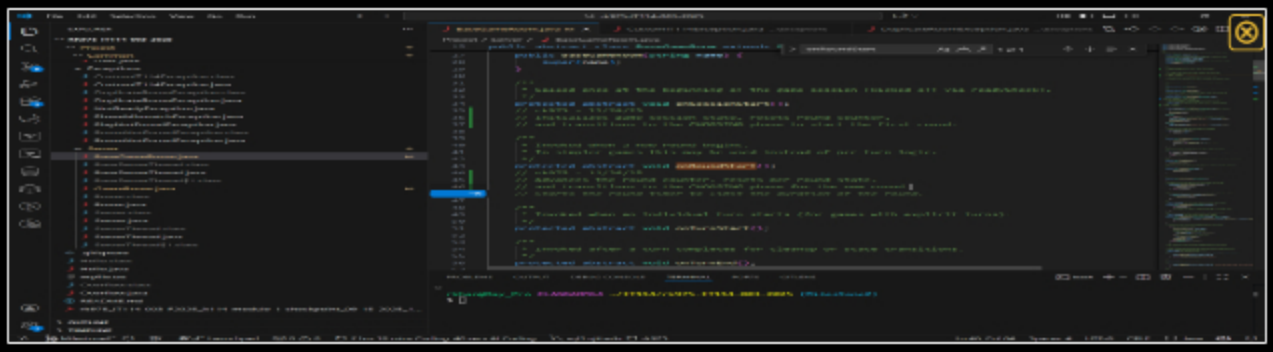
Part 1:

Progress: 100%

Details:

- Show the snippet of `onRoundStart()`

roundstart



roundstart

Saved: 11/26/2025 3:47:22 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 GameRoom is getting ready for another round of the Rock, Paper, Scissors game in this section. In order to prevent leftovers from prior rounds from spilling into the current one, onRoundStart() first clears the per-turn state and resets any existing round timer. The round counter is then increased, the phase is changed to CHOOSING, and each non-eliminated player's stored choice is cleared so they can make a new selection. A message such as "Round X started" is broadcast by the server. To inform all clients that a new round has begun and how to play, use /pick <r|p|s>. Lastly, it initiates a new round timer that sets a time limit for players to make their choices before the round is immediately closed and processed.

Saved: 11/26/2025 3:47:22 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())

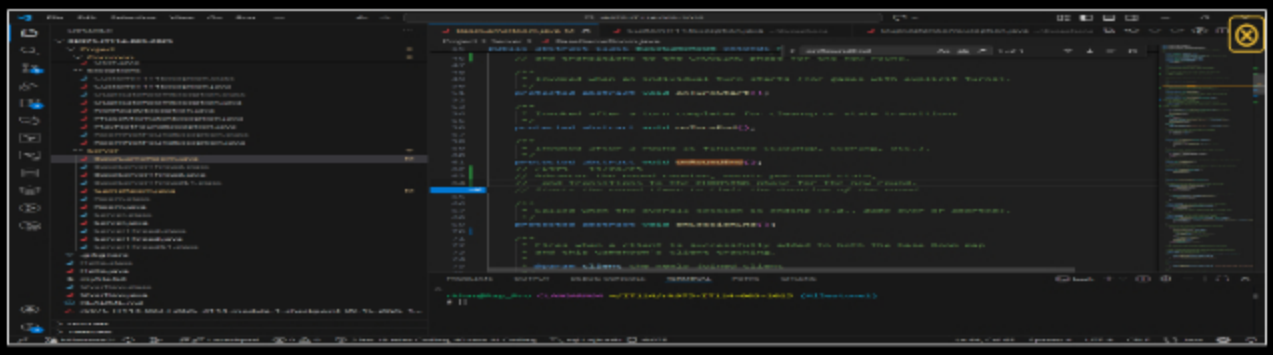
Part 1:

Progress: 100%

Details:

- Show the snippet of `onRoundEnd()`

roundend1



roundend2

Saved: 11/26/2025 3:50:35 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 a round is over, onRoundEnd() takes care of all cleanup and transition logic. It alerts every participant when the round is over and that the results are being processed after stopping and clearing the active round timer to stop it from shooting again. The method invokes processRoundResults(), which assesses options, allocates points, and removes players if necessary. The gaming session finishes and goes to onSessionEnd() if there are only one or zero active (non-eliminated) players left. If not, the game proceeds smoothly by starting a new round by initiating the subsequent lifecycle event, onRoundStart().

Saved: 11/26/2025 3:50:35 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

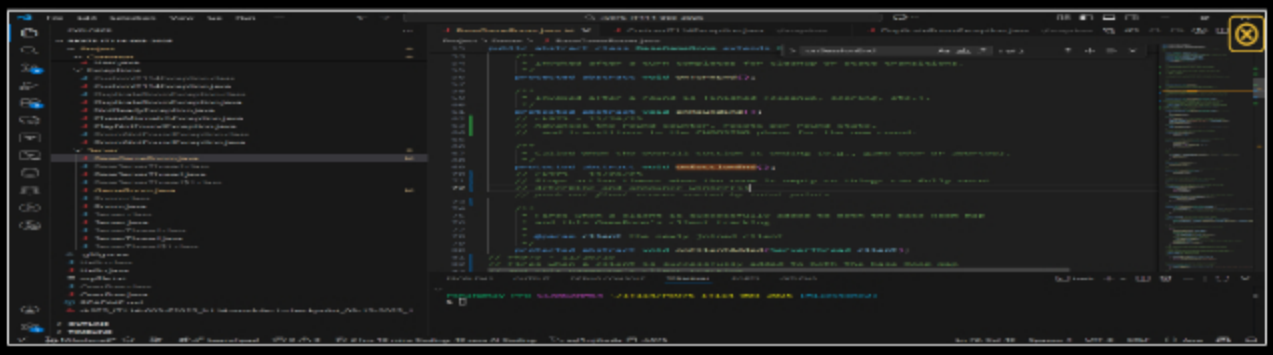
Part 1:

Progress: 100%

Details:

- Show the snippet of `onSessionEnd()`

sessionend1



sessionend2

Saved: 11/26/2025 3:53:26 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

The game state is completely cleaned up by onSessionEnd(), which also gets the room ready for a new session. To ensure that no state is carried over into the following game, it first resets all readiness and turn-related tracking. After that, it broadcasts the final results, chooses the winner (or tie), and provides each client with a complete final scoreboard. The room is then cleared for the following match by resetting each player's in-session data, including points, elimination status, choices, ready flags, and turn flags. Lastly, it returns the GameRoom to the READY phase, which is the stage of the lifecycle where participants must be ready before the next game can start.

Saved: 11/26/2025 3:53:26 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:

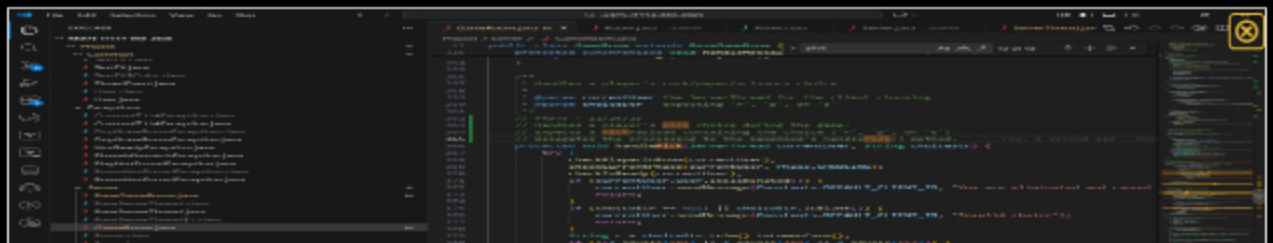
Progress: 100%

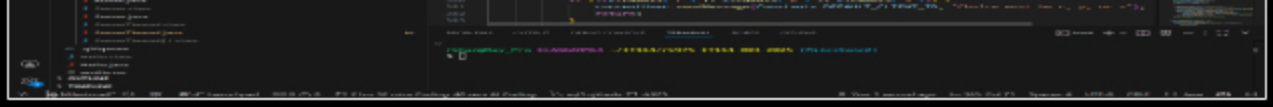
Details:

- 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)

gameroom1

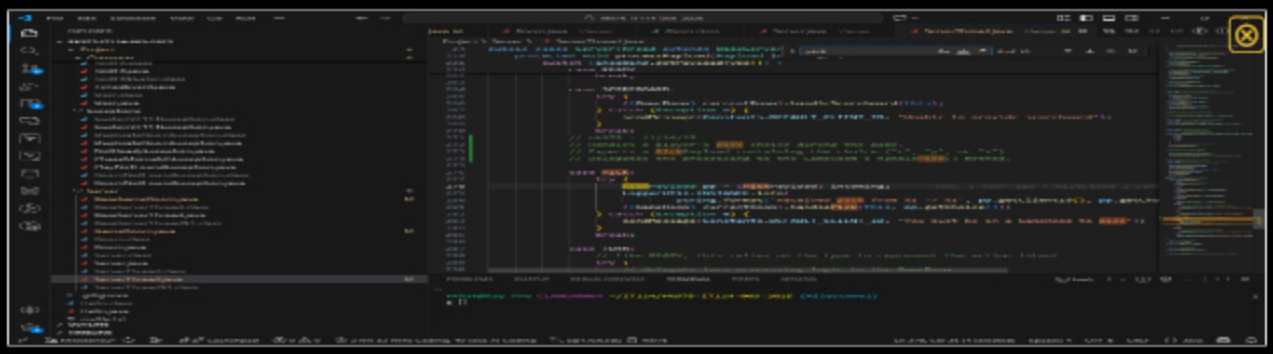
gameroom2



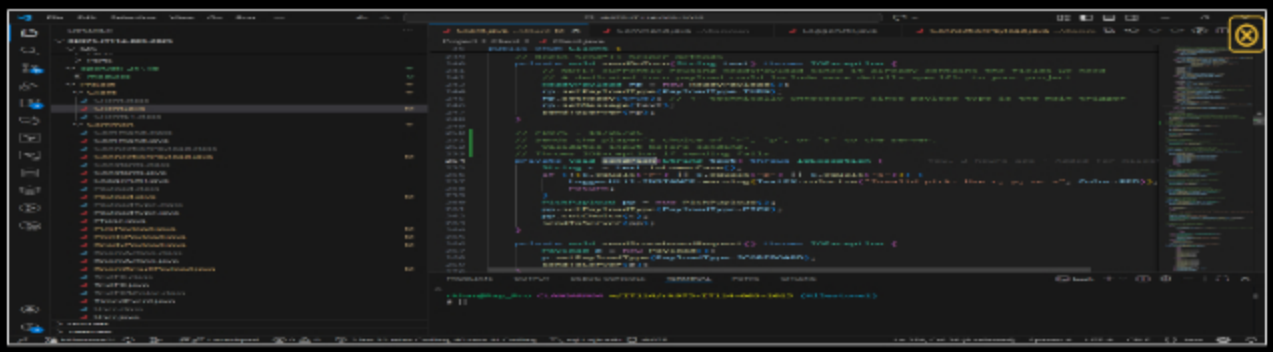


gameroom3

gameroom4



gameroom5



gameroom6

Saved: 11/26/2025 4:11:46 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 the /pick command is used by the client. In the customer.processClientCommand uses sendPick() to construct a PickPayload and sendToServer() to write it to the server.After reading the PickPayload, the server's ServerThread forwards it to GameRoom.handlePick, which verifies the user's or player's selection. Lastly, GameRoom.processRoundResults() transmits the scoreboard to all clients after checking the game and updating User.points.

Saved: 11/26/2025 4:11:46 PM

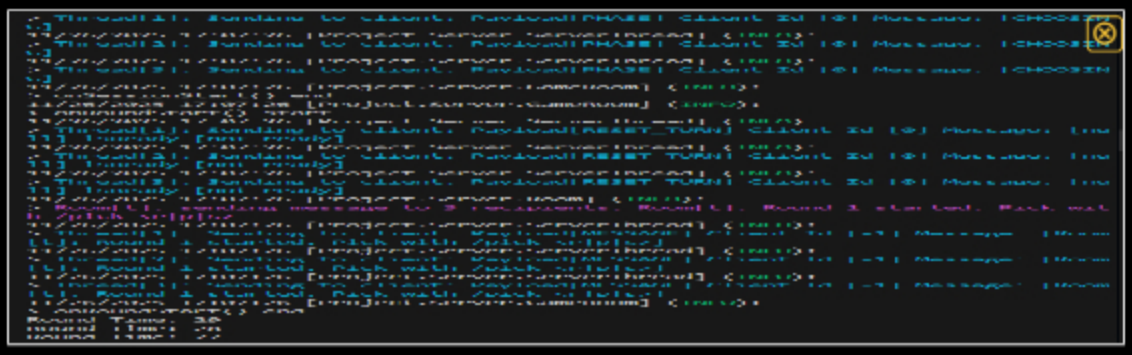
Task #2 (2 pts.) - Game Cycle Demo

Progress: 100%

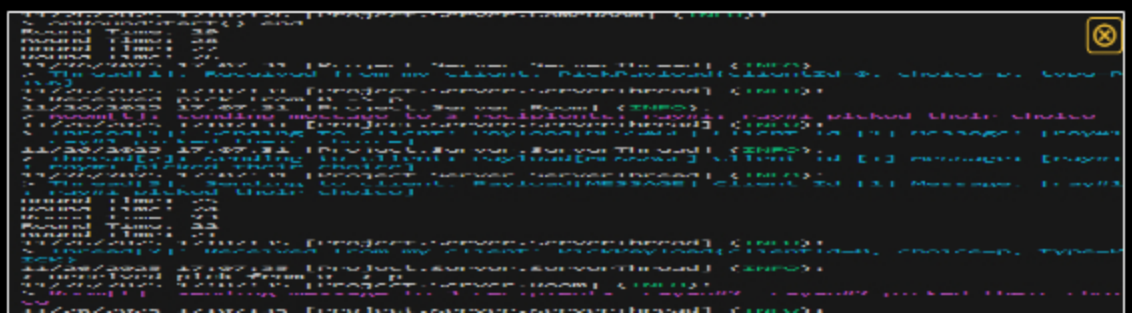
Details:

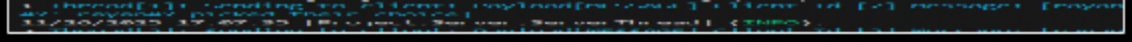
- Show examples from the terminal of a full session demonstrating each command and progress output
- This includes battle outcomes, scores and scoreboards, etc
- Ensure at least 3 Clients and the Server are shown
- Clearly caption screenshots

serverpick1

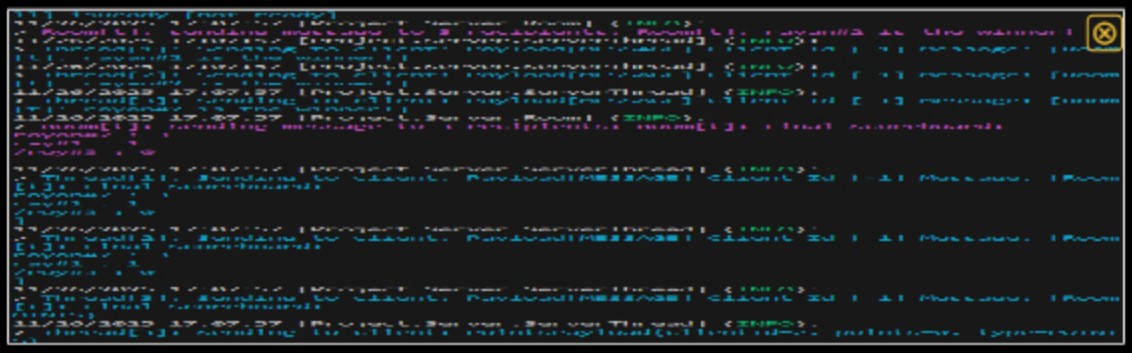


serverpick2

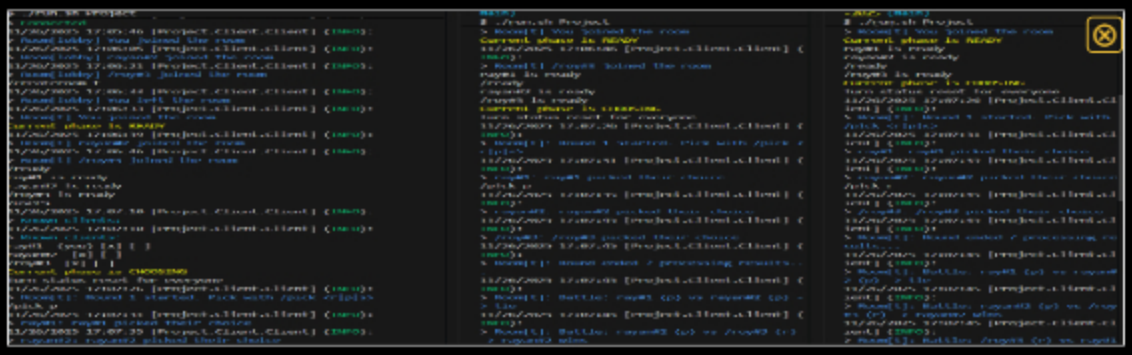




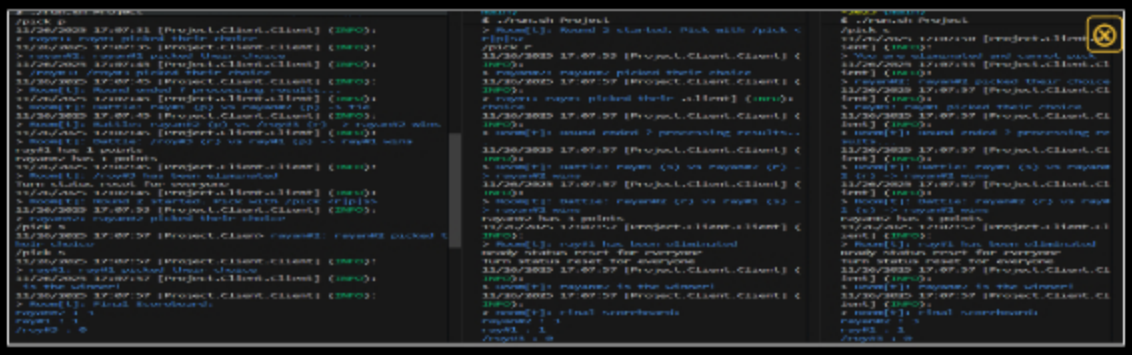
sererpick3



serverpick4



client



client2

Saved: 11/26/2025 5:13:20 PM

Section #4: (1 pt.) Misc

Progress: 100%

Task #1 (0.33 pts.) - Github Details

Progress: 100%

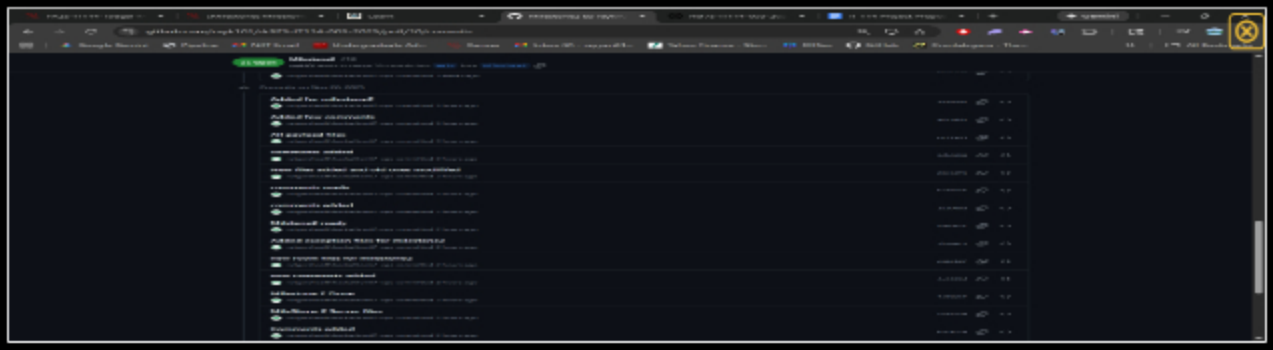
Part 1:

Progress: 100%

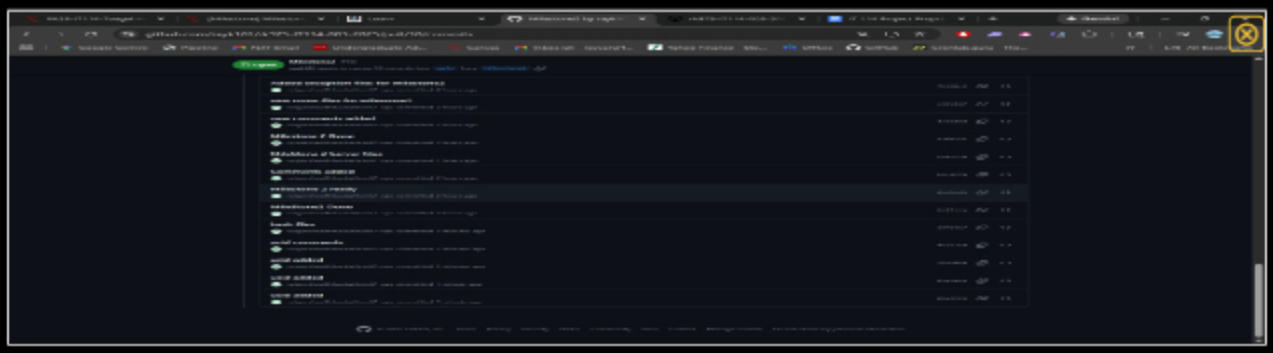
Details:

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

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



pull request1



pull request2

Saved: 11/26/2025 5:19:44 PM

Part 2:

Progress: 100%

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

URL #1
https://github.com/rayk101/rk975-IT114-003-2025/

URL
https://github.com/rayk101/rk975

Saved: 11/26/2025 5:19:44 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

wakatime

wakatime

Saved: 11/26/2025 5:14:54 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:

Working through this taught me how every lifecycle method in the GameRoom contributes to the smooth operation of a multiplayer session, from the beginning of rounds to their conclusion. I now see how crucial it is to sync states, such as phases, readiness, and turn status, so that each player entering or exiting the room always sees the proper game state. I also discovered how the server controls timers, takes care of cleanup, and switches between phases according on conditions and player activities. All things considered, this made it easier for me to understand how a real-time multiplayer system maintains consistency and avoids stale data or desync problems.

Saved: 11/26/2025 3:58:35 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:

For me, figuring out what each lifecycle method was doing and expressing the general reasoning in concise phrases was the easiest portion of the task. It was easy to explain each step once I grasped the flow—session start, round start, round end, and session end. It was easy to see what was going on behind the scenes by following the method names and comments. It was extremely manageable to write brief explanations using that framework.

Saved: 11/26/2025 3:59:05 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 of the assignment was keeping track of how all the different timers, states, and player actions connected throughout the game flow. There are many moving pieces—round timers, turn timers, readiness checks, elimination rules, and syncing state across clients—which made it challenging to fully digest at first. Understanding how each lifecycle method triggered the next took the most time. I had to carefully follow the logic to make sure I wasn't missing any hidden interactions between phases. Overall, the complexity of the game loop and state management was the most difficult part to understand.

Saved: 11/26/2025 3:59:29 PM