Submission Worksheet

Submission Data

Course: IT114-450-M2025
Assignment: IT114 Milestone 2 - RPS
Student: Daniel C. (dvc2)
Status: Submitted | **Worksheet Progress:** 100%
Potential Grade: 10.00/10.00 (100.00%)
Received Grade: 0.00/10.00 (0.00%)
Started: 8/5/2025 12:33:22 AM
Updated: 8/5/2025 8:27:29 PM
Grading Link: https://learn.ethereallab.app/assignment/v3/IT114-450-M2025/it114-milestone-2-rps/grading/dvc2
View Link: https://learn.ethereallab.app/assignment/v3/IT114-450-M2025/it114-milestone-2-rps/view/dvc2

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 code snippet

Project > Common > PointsPayload.java

```java
package M5.Part5;
// PointsPayload file to fulfill part 1 requirement - dvc2 8/4/2025
public class PointsPayload extends Payload {
    private int points;

    public int getPoints() {
        return points;
    }

    public void setPoints(int points) {
        this.points = points;
    }

    @Override
    public String toString() {
        return super.toString() +
                String.format(" Points: [%s]", getPoints());
    }
}
```

PointsPayload.java code snippet

Project > Common > PayloadType.java

```java
package M5.Part5;

public enum PayloadType {
    CLIENT_CONNECT, // client requesting to connect to server (passing of initialization data
                    // [name])
    CLIENT_ID, // server sending client id
    SYNC_CLIENT, // silent syncing of clients in room
    DISCONNECT, // distinct disconnect action
    ROOM_CREATE,
    ROOM_JOIN,
    ROOM_LEAVE,
    REVERSE,
    MESSAGE // sender and message
    POINTS  // POINTS added - dvc2 8/4/2025
}
```

PayloadType.java code snippet

Saved: 8/5/2025 12:39:54 AM

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 a general container for data transmitted between the client and server; it has a type, a client ID, and an optional message. The ConnectionPayload and PointsPayload subclasses enhance this property by including a client name and point values.

Saved: 8/5/2025 12:39:54 AM

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

```
    protected synchronized void addClient(ServerThread client) {
        if (!isRunning) { // block action if Room isn't running
            return;
        }
        if (clientsInRoom.containsKey(client.getClientId())) {
            info("Attempting to add a client that already exists in the room");
            return;
        }
        clientsInRoom.put(client.getClientId(), client);
        client.setCurrentRoom(this);
        client.sendResetUserList();
        syncExistingClients(client);
        // notify clients of someone joining
        joinStatusRelay(client, true);

    }
```

onClientAdded() code snippet

```
    protected synchronized void removeClient(ServerThread client) {
        if (!isRunning) { // block action if Room isn't running
            return;
        }
        if (!clientsInRoom.containsKey(client.getClientId())) {
            info("Attempting to remove a client that doesn't exist in the room");
            return;
        }
        ServerThread removedClient = clientsInRoom.get(client.getClientId());
        if (removedClient != null) {
            // notify clients of someone joining
            joinStatusRelay(removedClient, false);
            clientsInRoom.remove(client.getClientId());
            autoCleanup();
        }
    }
```

onClientRemoved() code snippet

Saved: 8/5/2025 12:48:52 AM

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 room, the application data is synced by providing a reset user list to the new client and synchronizing current client information. The method then tells the remaining clients in the room that a new user has joined. When a client leaves the room, the other clients are notified of their exit. This method also does an autoCleanup, which closes the room if it becomes empty but is not in the main lobby.

Saved: 8/5/2025 12:48:52 AM

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()`

```
Project > Common > GameRoom.java
package MS.Part5;

public class GameRoom extends Room {

    public GameRoom(String name) {
        super(name);
    }

    // GameRoom.java new file and code added - dwc2 8/4/2025
    protected void onSessionStart() {
        // Reset player data for each client
        clientsInRoom.values().forEach(client -> {

            client.resetGameState();
        });

        // Set the status and trigger the start of the first round
        info("A new game session has started. The timer begins now.");

    }
}
```

OnSessionStart() code snippet

Saved: 8/5/2025 12:54:15 AM

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 onSessionStart() method restores the initial session state by looping through all connected clients and clearing their game data. It then initiates the following lifecycle event, the "during the round" phase, by announcing the start of a new game and starting the round timer.

Saved: 8/5/2025 12:54:15 AM

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

Progress: 100%

Details:

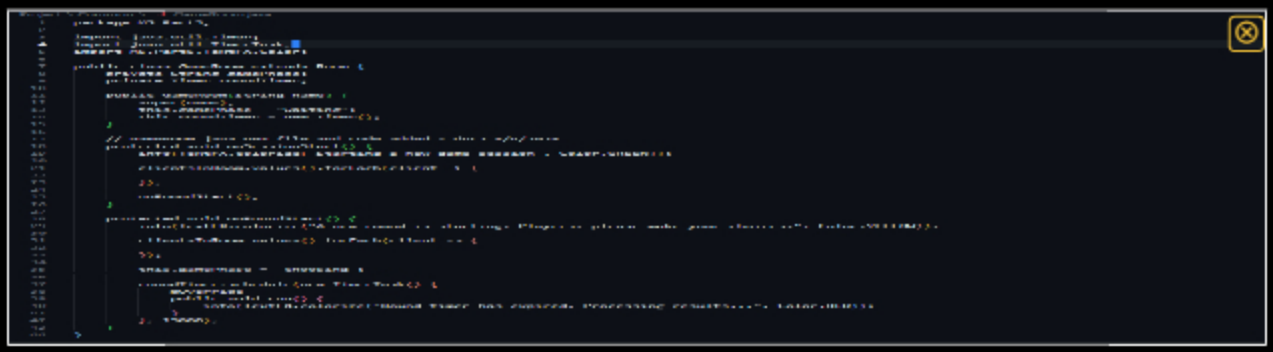
- Reqs from Document
  - Initialize remaining Players' choices to null (not set)
  - Set Phase to "choosing"
  - GameRoom round timer begins

Part 1:

Progress: 100%

Details:

- Show the snippet of `onRoundStart()`



onRoundStart() code snippet located lines 28-44 in my GameRoom.java file

Saved: 8/5/2025 1:03:28 AM

Part 2:

Progress: 100%

Details:

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

Your Response:

The onRoundStart() method initiates a new round by resetting player selections, changing the game phase to "choosing", and starting a round timer. When the timer expires, the round ends and the results are processed.

Saved: 8/5/2025 1:03:28 AM

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()`

```
    protected void onRoundEnd() {
        info("Round has ended. Processing results.");

        clientsInRoom.values().forEach(client -> {

        });

        long nonEliminatedCount = clientsInRoom.values().stream()
            .filter(client -> !client.isEliminated())
            .count();

        if (nonEliminatedCount == 1) {
            info("We have a winner!");
            onSessionEnd();
        } else if (nonEliminatedCount == 0) {
            info("The session is a tie.");
            onSessionEnd();
        } else {
            onRoundStart();
        }
    }

    protected void onSessionEnd() {
        info("Game session has ended. Resetting for new game.");
    }
}
```

onRoundEnd() code snippet

Saved: 8/5/2025 1:08:27 AM

Part 2:

Progress: 100%

Details:

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

Your Response:

The onRoundEnd() method starts with cleaning and classifying players who did not make a decision as eliminated. It then runs an end check, calculating the number of players who have not been eliminated, to determine the session's state. This check initiates the next lifecycle event, either onSessionEnd() if a single winner or tie is discovered, or onRoundStart() if the game has to continue for another round.

Saved: 8/5/2025 1:08:27 AM

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

- 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()`

```
    protected void onSessionEnd() {
        info("Game session has ended.");

        clientsInRoom.values().forEach(client -> {

        });

        info("The game is over. A new ready check will be required to start a new session.");
    }
}
```

onSessionEnd() code snippet

Saved: 8/5/2025 1:17:31 AM

Part 2:

Progress: 100%

Details:

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

Your Response:

The onSessionEnd() method cleans up and resets all clients' player data. It also delivers the final scoreboard and a game-ending message to all clients. The next lifecycle event is not triggered automatically; instead, the method tells clients that a fresh readiness check is necessary before beginning a new session.

Saved: 8/5/2025 1:17:31 AM

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)

```java
        } else if (text.startsWith(Command.PICK.command)) {
            text = text.replace(Command.PICK.command, "").trim();
            if (text == null || text.length() == 0 || !("r".equalsIgnoreCase(text) || "p".equalsIgnoreCase(text) || "s".equalsIgnoreCase(text))) {
                System.out.println(TextFX.colorize("Invalid choice. Please pick 'r', 'p', or 's'", Color.RED));
                return true;
            }
            sendChoice(text);
            wasCommand = true;
        }
    }
```

process client command code snippet

```java
            case ROOM_LEAVE:
                currentRoom.handleJoinRoom(this, Room.LOBBY);
                break;
            case PICK_CHOICE:
                ((GameRoom) currentRoom).handlePick(this, incoming.getMessage());
                break;
            default:
                System.out.println(TextFX.colorize("Unknown payload type received", Color.RED));
                break;
        }
    }
```

process method code snippet

```java
    protected synchronized void handlePick(ServerThread sender, String choice) {
        if (!("r".equalsIgnoreCase(choice) || "p".equalsIgnoreCase(choice) || "s".equalsIgnoreCase(choice))) {
            sender.sendMessage("Invalid choice. Please pick 'r', 'p', or 's'");
            return;
        }

        clientsInRoom.values().forEach(client -> {
```

```
        clients.values().forEach(client -> {

        });

        relay(null, String.format("%s has made their choice.", sender.getDisplayName()));
    }
}
```

handle method code snippet

```
    protected synchronized void relay(ServerThread sender, String message) {
        if (!isRunning) { // block action if Room isn't running
            return;
        }

        // Note: any desired changes to the message must be done before this line
        String senderString = sender == null ? String.format("Room[%s]", getName())
                : sender.getDisplayName();
        // Note: formattedMessage must be final (or effectively final) since outside
        // scope can't be changed inside a callback function (see removeIf() below)
        final String formattedMessage = String.format("%s: %s", senderString, message);

        // loop over clients and send out the message; remove client if message failed
        // to be sent
        // Note: this uses a lambda expression for each item in the values() collection.
        // It's one way we can safely remove items during iteration
        info(String.format("sending message to %s recipients: %s", clientsInRoom.size(), formattedMessage));

        clientsInRoom.values().removeIf(serverThread -> {
            boolean failedToSend = serverThread.sendMessage(formattedMessage);
            if (failedToSend) {
                System.out.println(
                        String.format("Removing disconnected %s from list", serverThread.getDisplayName()));
                disconnect(serverThread);
            }
            return failedToSend;
        });
    }
```

send/sync method code snippet

```
    protected boolean sendMessage(String message) {
        Payload payload = new Payload();
        payload.setPayloadType(PayloadType.MESSAGE);
        payload.setMessage(message);
        return sendToClient(payload);
    }
```

send method code snippet

```
            case MESSAGE:
                processMessage(payload);
                break;
```

process method code snippet

```
    private void processMessage(Payload payload) {
        System.out.println(TextFX.colorize(payload.getMessage(), Color.BLUE));
    }
```

process method code snippet 2

Saved: 8/5/2025 2:51:25 AM

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:

1. Client Command: The user sends the /pick command to the client, which is processed by processClientCommand.
2. Payload Creation: The client's sendChoice method generates a Payload containing the PICK_CHOICE type and the user's selection, which is then delivered to the server.
3. Server Processing: The payload is received by the server's ServerThread, and the processPayload method passes the command to the GameRoom's handlePick method.
4. GameRoom Logic: The handlePick function checks the selection, saves it, and then utilizes the relay method to deliver a confirmation message to all players.
5. Message Relaying: The Room class's relay method iterates across each ServerThread in the room and invokes the sendMessage method.
6. Client Reception: Each ServerThread transmits a fresh MESSAGE payload, which is received by the client's processPayload and shown on the console as processMessage.

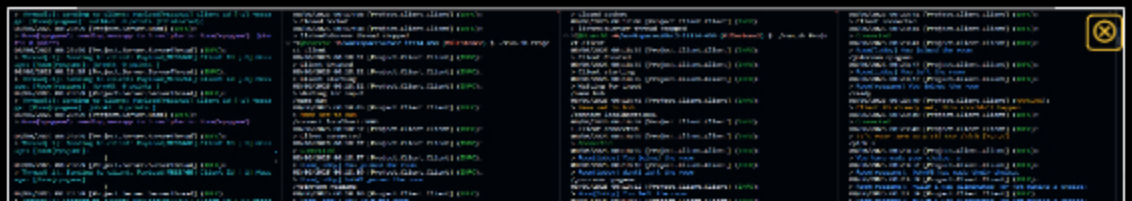
Saved: 8/5/2025 2:51:25 AM

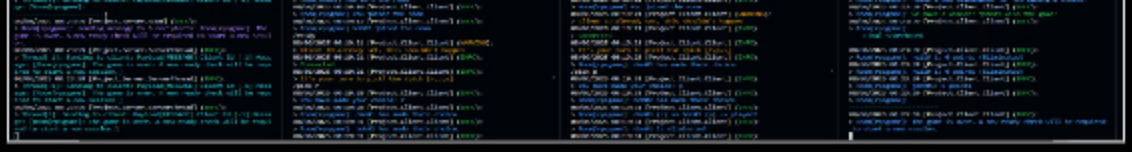
Task #2 (2 pts.) - Game Cycle Demo

Progress: 100%

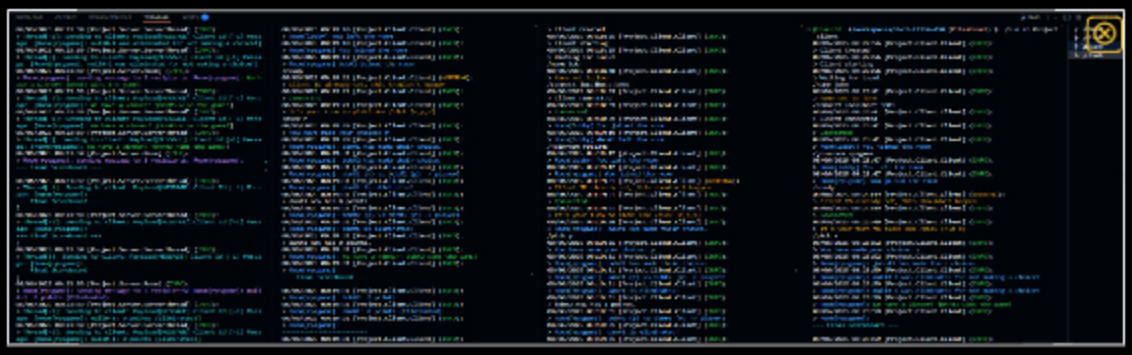
Details:

- 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





output snippet showcasing the RPS functions working properly with 3 different clients as well as the scoreboard and tracked scor



output snippet 2

Saved: 8/5/2025 8:27:29 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

Screenshot

Saved: 8/5/2025 4:13:13 AM

Part 2:

Progress: 100%

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

URL #1
https://github.com/dcarch2/dvc2-it114-450/pull/7

URL
https://github.com/dcarch2/dvc2-

Saved: 8/5/2025 4:13:13 AM

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

Screenshot, same duration issue

Saved: 8/5/2025 4:13:36 AM

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 that managing the full lifecycle of my game constantly was important to making the functionality of of commands work. I also learned how to handle player commands like "/pick" and the code needed for my server to handle the necessary outputs.

Saved: 8/5/2025 4:15:28 AM

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 taking the code snippets for the first few tasks as most of the code was already there and I just had to find it.

Saved: 8/5/2025 4:16:03 AM

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 getting the /pick command to work properly with all 3 users.

Saved: 8/5/2025 4:16:45 AM