

Submission Worksheet

CLICK TO GRADE

<https://learn.ethereallab.app/assignment/IT114-003-F2024/it114-module-2-rps-2024-m24/grade/cae6>

Course: IT114-003-F2024

Assignment: [IT114] Module 2 RPS 2024 (M24)

Student: Chizorom E. (cae6)

Submissions:

Submission Selection

1 Submission [submitted] 12/2/2024 3:32:46 AM

Instructions

[^ COLLAPSE ^](#)

1. Implement the Milestone 2 features from the project's proposal document:
https://docs.google.com/document/d/11SRMo7JkLAMM-PuuiGwl_Z-QXP3pyQ7xN3IRxwmcwCc/view
2. Make sure you add your ucid/date as code comments where code changes are done
3. All code changes should reach the Milestone2 branch
4. Create a pull request from Milestone2 to main and keep it open until you get the output PDF from this assignment.
5. Gather the evidence of feature completion based on the below tasks.
6. Once finished, get the output PDF and copy/move it to your repository folder on your local machine.
7. Run the necessary git add, commit, and push steps to move it to GitHub
8. Complete the pull request that was opened earlier
9. Upload the same output PDF to Canvas

Branch name: Milestone2

Group

Group: Payloads

Tasks: 2

Points: 1

100%

[^ COLLAPSE ^](#)

Task

Group: Payloads

100%

Task #1: Base Payload Class

Weight: ~50%

Points: ~0.50

▲ COLLAPSE ▾

1 Details:

All code screenshots must have ucid/date visible.



Columns: 1

Sub-Task

Group: Payloads

100%

Task #1: Base Payload Class

Sub Task #1: Show screenshot of the code for this file

Task Screenshots

Gallery Style: 2 Columns

4

2

1

```
package payloads;
import java.util.Date;
public class Payload implements Serializable {
    private String payloadType;
    private long clientId;
    private String message;
    private String clientName;
    public void setPayloadType() {
        return payloadType;
    }
    public void setPayloadType(String payloadType) {
        this.payloadType = payloadType;
    }
    public long getClientId() {
        return clientId;
    }
    public void setClientId(long clientId) {
        this.clientId = clientId;
    }
    public String getMessage() {
        return message;
    }
    public void setMessage(String message) {
        this.message = message;
    }
    @Override
    public String toString() {
        return String.format("Payload[%s] Client: %d] %s", payloadType, clientId, message);
    }
}
```

Base payload code

```
public void setClientId(long clientId) {
    this.clientId = clientId;
}
public String getClientName() {
    return clientName;
}
public void setClientName(String clientName) {
    this.clientName = clientName;
}
public String getMessage() {
    return message;
}
public void setMessage(String message) {
    this.message = message;
}
@Override
public String toString() {
    return String.format("Payload[%s] Client: %d] %s", payloadType, clientId, message);
}

```

Base Payload code

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

The base payload sends and receives information about the client through the payload. There is a payload type component that indicates the type of payload, the clientID is the identifier of the client, and the client Name stores the name, the message stores the message and command.

Sub-Task

Group: Payloads

100%

Task #1: Base Payload Class

Sub Task #2: Show screenshot examples of the terminal output for this object

Task Screenshots

Gallery Style: 2 Columns

4 2 1

terminal output for base payload

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

End of Task 1

Task

Group: Payloads

Task #2: PointsPayload (or equivalent)

Weight: ~50%

Points: ~0.50

^ COLLAPSE ^

Details:

All code screenshots must have ucid/date visible.

1

Columns: 1

Sub-Task

Group: Payloads

Task #2: PointsPayload (or equivalent)

Sub Task #1: Show screenshot of the code for this file

Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
package Project.Common;

public class PointsPayload extends Payload{
    private int points;

    public PointsPayload(){
        setPayloadType(PayloadType.POINTS);
    }

    public int getPoints() {
        return points;
    }

    public void setPoints(int points) {
```

```
this.points = points;
}
} < #3-18 public class PointsPayload extends Payload
```

Points payload code

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

≡, Task Response Prompt

Explain in concise steps how this logically works

Response:

The pointsPayload stores all the information sets the payload type to points and stores the information about the points.

Sub-Task

Group: Payloads

100%

Task #2: PointsPayload (or equivalent)

Sub Task #2: Show screenshot examples of the terminal output for this object

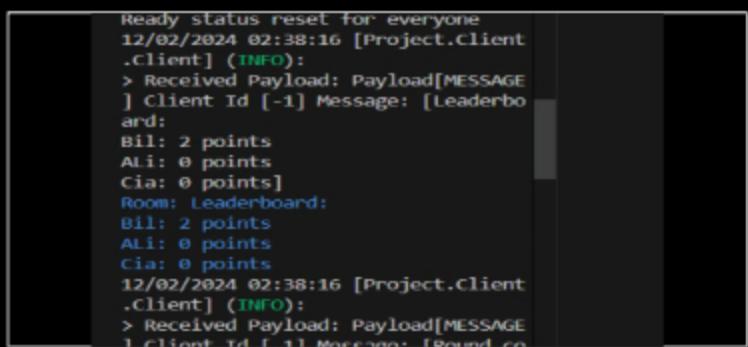
☒ Task Screenshots

Gallery Style: 2 Columns

4

2

1



```
Ready status reset for everyone
12/02/2024 02:38:16 [Project.client
.Client] (INFO):
> Received Payload: Payload[MESSAGE
] Client Id [-1] Message: [Leaderbo
ard:
Bil: 2 points
Ali: 0 points
Cia: 0 points]
Room: Leaderboard:
Bil: 2 points
Ali: 0 points
cia: 0 points
12/02/2024 02:38:16 [Project.client
.Client] (INFO):
> Received Payload: Payload[MESSAGE
] Client Id [-1] Message: [Leaderbo
```

Terminal output for points

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

End of Task 2

End of Group: Payloads

Task Status: 2/2

Group

Group: Session Start

Tasks: 1

Points: .5

▲ COLLAPSE ▲

Task

Task

Group: Session Start

Task #1: Trigger Round Start

Weight: ~100%

Points: ~0.50

[▲ COLLAPSE ▾](#)

1 Details:

All code screenshots must have ucid/date visible.



Sub-Task

Group: Session Start

Task #1: Trigger Round Start

Sub Task #1: Show the related code

100%

Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
01 // Lifecycle methods
02 protected void onSessionStart() {
03     logger.info("onSessionStart() - " + messages("onSessionStart() - " + id));
04     changePhase("Phase-INIT", 0);
05     resetReadyStatus();
06     resetChoices();
07 }
08
09 // This is generated code
10
11 @Override
12 protected void onRoundStart() {
13     logger.info("onRoundStart() - " + messages("onRoundStart() - " + id));
14
15     // Reset player status for the new round
16     resetReadyStatus();
17     resetChoices();
18
19     // Start the round timer
20     resetRoundTimer(); // Clearing previous timer is canceled
21     startRoundTimer();
22
23     // Change the phase to indicate the game is in progress
24     changePhase("Phase-READY");
25
26     if (!roundIsPrepared()) {
27         sendPhase("Phase-INIT", "Waiting for all players to ready up..."); // message for all players to ready up...
28         return; // not proceed until all players are ready
29     }
30
31     // Transition to the next phase
32     changePhase("Phase-CHOOSING");
33 }
```

Trigger on round start code

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

The onRoundStart is triggered when the session starts. The onRoundStart begins with resetting the ready status for all the players for the new round. Then all the player's choices are reset, then the phase is changed to ready again for the players. Then resets the round timer and starts the round timer. Then if all the players are ready the phase gets turned to TURN, this is the "choosing" phase where they can make their choices and then a message is sent to all the clients.

End of Task 1

End of Group: Session Start

Task Status: 1/1

Group

Group: Round Start

Tasks: 3

Points: 1.25

▲ COLLAPSE ▲

Task

Group: Round Start

Task #1: Reset Choices

Weight: ~33%

Points: ~0.42

▲ COLLAPSE ▲

i Details:

All code screenshots must have ucid/date visible.

**Sub-Task**

Group: Round Start

Task #1: Reset Choices

Sub Task #1: Show the code that resets the remaining Player's choices to null

Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
8
9     private void resetPlayerChoices() {
10        playersInRoom.values().forEach(player -> {
11            if (!player.isEliminated()) {
12                player.setChoice(choice:null);
13                player.setTakeTurn(tookTurn:false);
14            }
15        });
16    } <- #310-315 playersInRoom.values().forEach
17    <- #309-316 private void resetPlayerChoices()
18
19
```

code that changes remaining player choices to null

Caption(s) (required) ✓Caption Hint: *Describe/highlight what's being shown***Task Response Prompt***Explain in concise steps how this logically works*

Response:

This reset player choice sets the player's choices to null and whether or not they took a turn false, which is for players who have not been eliminated.

End of Task 1

Task



Group: Round Start

Task #2: Choosing Phase (or similar)

Weight: ~33%

Points: ~0.42

[▲ COLLAPSE ▾](#)

① Details:

All code screenshots must have ucid/date visible.



Columns: 1

Sub-Task



Group: Round Start

Task #2: Choosing Phase (or similar)

Sub Task #1: Show the code that changes to Phase to the choosing phase

🖼 Task Screenshots

Gallery Style: 2 Columns

4 2 1

The screenshot displays two side-by-side code snippets. The left snippet shows a Java enum definition for 'Phase' with three values: READY, IN_PROGRESS, and TURN. The right snippet shows a sequence of messages and logic related to transitioning to the TURN phase, where players make choices between Rock (R), Paper (P), Scissors (S), or skip (skip). The code includes comments explaining the process: 'process if no net present until all players are ready', 'Transition to the TURN phase, where players make choice', and 'send message [ServerConstants.FISH_HOOK, message("Choices are Rock (R), Paper (P), Scissors (S), or skip (skip)")]'. It also handles player notifications and updates the turn count.

choosing phase code

choosing phase code

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

≡ Task Response Prompt

Explain in concise steps how this logically works

Response:

This changes the phase to TURN where the players make their choice Rock, paper scissors, or skip where they can skip their chance. Once the phase changes to Turn players can make their choice by entering /R, /P, /S, or /skip.

Sub-Task

Group: Round Start

Task #2: Choosing Phase (or similar)



Sub Task #2: Show terminal output of this change

Task Screenshots

Gallery Style: 2 Columns

Choosing phase

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

End of Task 2

Task

Group: Round Start

Task #3: Round Timer

Weight: ~33%

Points: ~0.42

COLLAPSE

● Details:

**All code screenshots must have ucid/date visible.
Any terminal output should have at least 3 clients involved.**

11

Columns: 1

Sub-Task

Group: Round Start

Task #3: Round Timer

Sub Task #1: Show the code that triggers the round timer (including the expiry callback)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Round timer Trigger

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

In onRoundStart, resetRoundTimer and startRoundTimer are called. Reset round timer checks for timers that aren't null and then cancels them and sets them to null. The startRoundTimer creates a new timed event for 50 seconds and once the timer expires the round ends. It also shows how much time is remaining every 10 seconds and when there are 3 seconds left.

Sub-Task

Group: Round Start



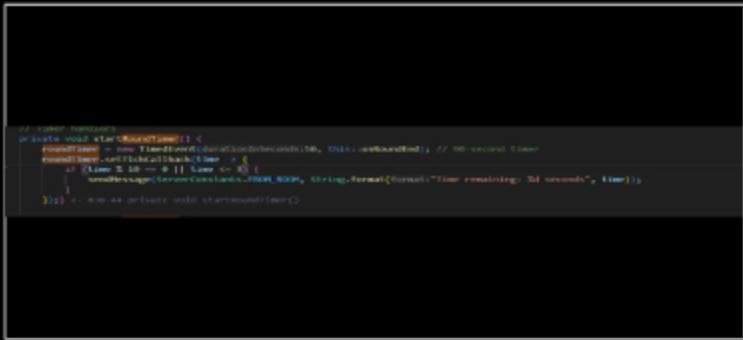
Task #3: Round Timer

Sub Task #2: Show how the clients are informed of the time (i.e., synced or stopwatch-like)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



clients being informed of the time

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

startRoundTimer creates a new timed event for 50 seconds and once the timer expires the round ends. It also shows how much time is remaining every 10 seconds and when there are 3 seconds left. It shows the time remaining, the seconds and then it displays the time.

Sub-Task

Group: Round Start



Task #3: Round Timer



Sub Task #3: Show an example from the terminal

Task Screenshots

Gallery Style: 2 Columns

4 2 1

terminal example

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

End of Task 3

End of Group: Round Start

Task Status: 3/3

Group

Group: During Round

Tasks: 1

Points: 2.5

◀ COLLAPSE ▶

Task

Group: During Round

Task #1: Choice Command

Weight: ~100%

Points: ~2.50

^ COLLAPSE ^

1 Details:
All code screenshots must have ucid/date visible.

Columns: 1

Sub-Task

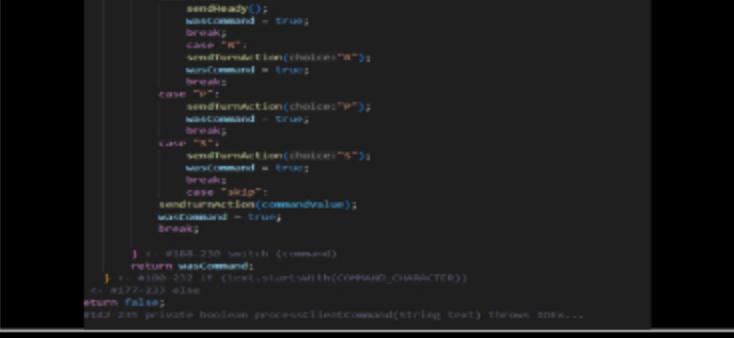
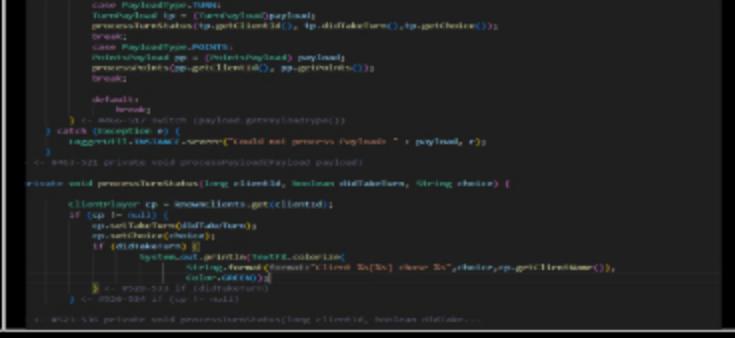
Group: During Round

100%

Task #1: Choice Command
Sub Task #1: Show the code that processes the /choice text

Task Screenshots

Gallery Style: 2 Columns

4	2	1
		
choice being processed	choice being processed	choice being processed

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

First the client side checks to see how the client typed in their command and only accepts the types listed which are R,P,S and skip following the command value /. So if the client enters any of those values the "wascommand" will be set to true otherwise the command will be read as null. The process Turn will process the choice the client made and store it.

Sub-Task

100%

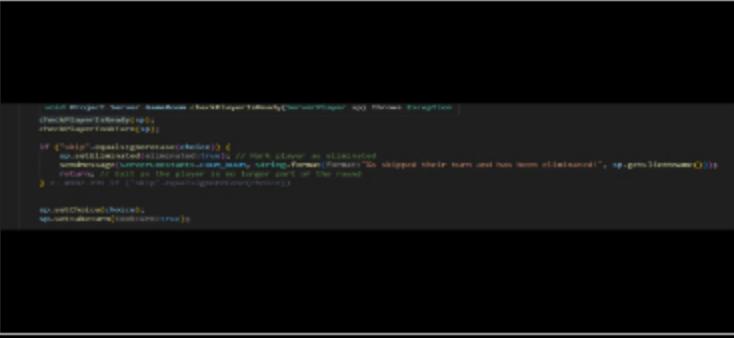
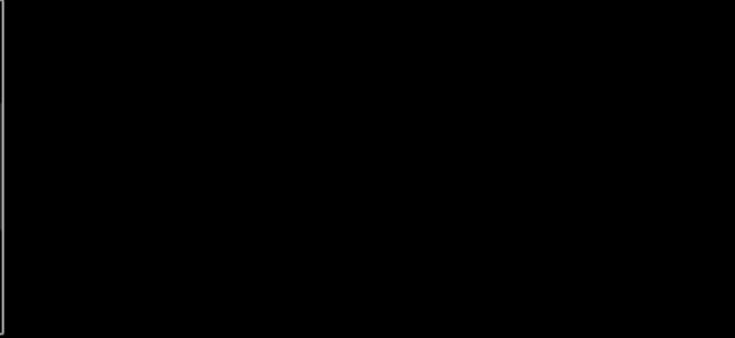
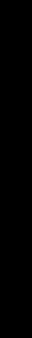
Group: During Round

Task #1: Choice Command

Sub Task #2: Show the GameRoom code that handles the choice (stores it on the Player and sends a message) (show any checks to ensure only valid Players are handled)

Task Screenshots

Gallery Style: 2 Columns

4	2	1
		
gameroom code that handles the choice	gameroom code that handles the choice	gameroom code that handles the choice

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

This checks to see if the player skipped their choice and if they did skip their choice the player is eliminated for that round. Then it stores the player's choice by setting it and then it changes setTakeTurn to true to signal that players made their choice and took their turn and if all players took their turn, then the round ends and the result gets handled the end of the round gets evaluated.

Sub-Task

Group: During Round

100%

Task #1: Choice Command

Sub Task #3: Show 3 examples of terminal output showing Players picking their choice (opponents shouldn't see what was chosen)

Task Screenshots

Gallery Style: 2 Columns

terminal output from players picking their choice

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

End of Task 1

End of Group: During Round

Task Status: 1/1

Group

Group: Round End

Tasks: 4

Points: 2.5

COLLAPSE

Task

Group: Round End

Task #1: Round End Conditions

Weight: ~25%

▲ COLLAPSE ▲

i Details:

All code screenshots must have ucid/date visible.
 Any terminal output should have at least 3 clients involved.



Columns: 1

Sub-Task

Group: Round End

Task #1: Round End Conditions

Sub Task #1: Show the code related to the round ending from round timer expiring

☒ Task Screenshots

Gallery Style: 2 Columns

4 2 1

```

roundTimer = new TimerEvent(DurationInseconds(99, this), onRoundEnd); // 99-second timer
roundTimer.start();
if (time < 0) {
    sendMessage(ServerCommands.PROP_ROUND, String.format("%s", "Time remaining: 00 seconds"), 500);
}
  
```

round ending from timer expiring

Caption(s) (required) ✓Caption Hint: *Describe/highlight what's being shown***=, Task Response Prompt***Explain in concise steps how this logically works*

Response:

The start round timer sets a timer for 50 seconds for the client. Once the timer is up onRoundEnd is called and then the game is handled once the timer has expired.

Sub-Task

Group: Round End

Task #1: Round End Conditions

Sub Task #2: Show the code related to the round ending from all eligible Players marking their choices

☒ Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
if (eligiblePlayers() > 0) {  
    sendPlayerMessage(ServerConstants.FROM_ROOM, message("All players have submitted their choices!"),  
    resetRoundTimer();  
    doneWithTask();  
}  
else {  
    doneWithTask();  
}
```

round ending for eligible players

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

In handleTurn, once it's been checked that all players made their choice and took their turn, the round timer ends and the round ends as well once all eligible players have taken their turns.

End of Task 1

Task

Group: Round End

Task #2: Battles

Weight: ~25%

Points: ~0.63

COLLAPSE

Details:

All code screenshots must have ucid/date visible.
Any terminal output should have at least 3 clients involved.



Columns: 1

Sub-Task

Group: Round End

Task #2: Battles

Sub Task #1: Show the code related to doing all the battles in round-robin



Task Screenshots

Gallery Style: 2 Columns

4

2

1

```
private void handleRoundEnd() {  
    System.out.println("Handling end of round. Color: " + color);  
    if (eligiblePlayers() > 0) {  
        players.forEach(player -> player.sendMessage(color + " end"));  
        if (logger.isInfoEnabled()) {  
            logger.info("All eligible players have made their choices. Starting the next round!");  
        }  
    }  
    else {  
        logger.info("No eligible players found. No message sent.");  
    }  
}  
  
protected void onRoundEnd() {  
    loggerUtil.INSTANCE.info(message("onroundend() start"));  
    resetRoundTimer();  
}  
  
private void resetRoundTimer() {  
    System.out.println("Resetting round timer");  
    roundTimer = new Timer("Round Timer", () -> {  
        if (logger.isInfoEnabled()) {  
            logger.info("Round timer has expired. Starting the next round!");  
        }  
        handleRoundEnd();  
    }, 1000);  
    roundTimer.start();  
}
```

```

handleResult() {
    evaluateRoundEnd(); // reset timer if round ended without the time expiring
    onSessionEnd();
    LoggerUtil.INSTANCE.info(message="onRoundEnd() end");
} <- #152-161 protected void onRoundEnd()

```

Round end calling handle result

```

handleResult() {
    evaluateRoundEnd(); // reset timer if round ended without the time expiring
    onSessionEnd();
    LoggerUtil.INSTANCE.info(message="onRoundEnd() end");
} <- #152-161 protected void onRoundEnd()

```

battles in round robin

```

    } else if (currentPlayer.getChoices().length == 2) {
        String choice1 = currentPlayer.getChoices()[0];
        String choice2 = currentPlayer.getChoices()[1];
        player1.setChoices(choice1);
        player2.setChoices(choice2);
    }
}

for (int i = 0; i < numPlayers; i++) {
    currentPlayer = players.get(i);
    if (!currentPlayer.isEliminated() && !player1.isEliminated() && !player2.isEliminated()) {
        checkMatch(currentPlayer.getChoices(), choice1, choice2);
    }
}

sendRoundEnd();

```

round robin

```

    int result = checkMatch(choice1, choice2);

    if (result == 0) {
        sendRoundEnd();
        String message = String.format("No wins against No [Nx - Nx]");
    } else if (result == 1) {
        sendRoundEnd();
        String message = String.format("X wins against No [Nx - Nx]");
        player1.setWins(player1.getWins() + 1);
        player1.setEliminated(true);
    } else if (result == -1) {
        sendRoundEnd();
        String message = String.format("No wins against X [Nx - Nx]");
        player2.setWins(player2.getWins() + 1);
        player2.setEliminated(true);
    }
}

```

round robin

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

Once the round ends, the result is handled in the handle result. So first the handle result filters out players who made their choice, who weren't eliminated, and players set as ready players and then listed them. If no players in the room make a choice , then the round will restart. First the varibale numPlayers is declared and it storeds the number of players in the game. Then begins an indexed loop that loops through the list of players in playerstoProcess using their indices. Player q will be the current player and Player 2 is the next player in line, (i+1)% numPlayers is what makes the list go full circle. This makes sure that the last player faces off with the the first player. Then we get the player choices for one and two and then we check the match. The check match method creates a list where R's index is 0, P is 1, and S is 2. So A is equal to the index of choice 1 and b is equal to the index of choice 2. So if a==b it will return a tie, if (a-b +3) % 3 to account for the range of players and if it is all equal to 1 player one wins and it returns one, if it's equalt to 2 or anything else, player 2 wins and if it's 0 its a tie. Here if player one doesn't win it returns negative 1. Any player who wins gets a point and the other player or players get eliminated and their name and who they beat gets displayed.

Sub-Task

Group: Round End



Task #2: Battles

Sub Task #2: Show the code related to relaying the appropriate outcome messages

Task Screenshots

Gallery Style: 2 Columns

4 2 1

```

int result = checkMatch(choices, choice2);

if (result == 0) {
    sendRoundEnd();
    String message = String.format("No wins against No [Nx - Nx]");
} else if (result == 1) {
    sendRoundEnd();
    String message = String.format("X wins against No [Nx - Nx]");
    player1.setWins(player1.getWins() + 1);
    player1.setEliminated(true);
} else if (result == -1) {
    sendRoundEnd();
    String message = String.format("No wins against X [Nx - Nx]");
    player2.setWins(player2.getWins() + 1);
    player2.setEliminated(true);
}

```

code relaying the appropriate outcome

Caption(s) (required) ✓

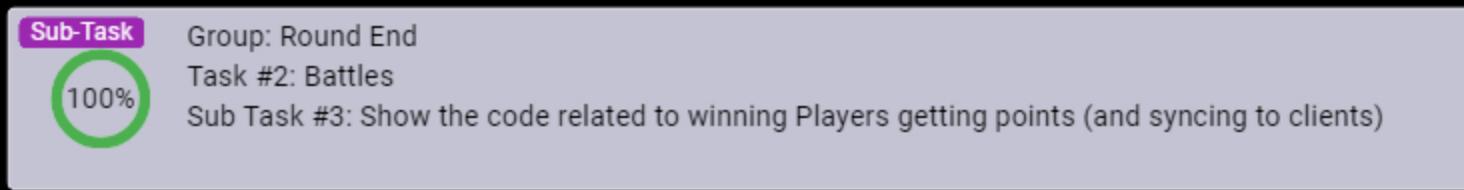
Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

Here if player one wins it will get the clients name, and the choice they made along with the choice of the other player. It will show that one player beat the other and the winning players choice will appear first.



Task Screenshots

Gallery Style: 2 Columns

4 2 1

winning players getting points

points being sent

```
public void setPoints(int p){  
    this.points = p;  
}  
public void changePoints(int p){  
    this.points += p;  
    this.points = Math.max(this.points, b:0); // minimum 0 points  
}  
public int getPoints(){  
    return this.points;  
}
```

points/ player payload

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

The winning player gets a point added if they beat their opponent. The leaderboard will take players in the room points and compare them and then in descending order show the players points. It will also display the clients data relating to and around points.

Sub-Task

Group: Round End

Task #2: Battles

Sub Task #4: Show terminal output of the battle outcomes

Task Screenshots

Gallery Style: 2 Columns

4 2 1

terminal output of battle outcomes

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

End of Task 2

Task

Group: Round End

Task #3: Elimination

Weight: ~25%

Points: ~0.63

COLLAPSE

Details:

All code screenshots must have ucid/date visible.
Any terminal output should have at least 3 clients involved.

Columns: 1

Sub-Task

Group: Round End

Task #3: Elimination

Sub Task #1: Show the code related to Players being eliminated during a battle (subset of the battle code above likely)

Task Screenshots

4 2 1

```

private boolean eliminated = false;

public boolean isEliminated() {
    return eliminated;
}

public void setEliminated(boolean eliminated) {
    this.eliminated = eliminated;
}

```

code related to player being eliminated

```

} else if (result == 1) {
    sendMessage(server, String.format("%s wins against %s (%s - %s)", player1.getClientName(), player2.getClientName(), choices1, choices2));
    player1.setPoints(player1.getPoints() + 1);
    player2.setEliminated(eliminated);
} else {
    sendMessage(server, String.format("%s wins against %s (%s - %s)", player2.getClientName(), player1.getClientName(), choices2, choices1));
    player2.setPoints(player2.getPoints() + 1);
    player1.setEliminated(eliminated);
}

```

elimination during a battle

Caption(s) (required) ✓Caption Hint: *Describe/highlight what's being shown***=> Task Response Prompt***Explain in concise steps how this logically works*

Response:

When player 1 wins and the result is equal to one, player 2's elimination status gets set to true and they are removed from the game. If player 2 wins and the result is equal to more than one than player one's elimination status gets set to true and then they are removed from the game.

Sub-Task

Group: Round End

Task #3: Elimination

Sub Task #2: Show the code that eliminates Players who didn't pick a choice in time

**☒ Task Screenshots**

4 2 1

```

for(Player player : players.values().stream()
        .filter(player -> !player.isEliminated() && (player.getChoice() == null || !player.choiceIsSet())))
    for(Player player -> player.setEliminated(true));
}

```

Code that eliminates players if they did not pick a choice in time

Caption(s) (required) ✓Caption Hint: *Describe/highlight what's being shown***=> Task Response Prompt***Explain in concise steps how this logically works*

Response:

This code filters out players who weren't eliminated and hadn't made their choice and didn't take their turn yet. Then it eliminates those players by setting eliminated to true to make sure they don't continue in the game.

Sub-Task

100%

Group: Round End

Task #3: Elimination

Sub Task #3: During a battle, what determines a Player gets eliminated, is it only if they're attacking, only if they're defending, or either?

Task Response Prompt

Explain

Response:

A player gets eliminated based on who wins and based on the index that is returned. If index one is returned it means player one has won the game, so by default, you would eliminate player 2. If an index above one has been returned then that signifies that player 2 has won the game and player one should be eliminated.

End of Task 3

Task

100%

Group: Round End

Task #4: Next Round or Session End

Weight: ~25%

Points: ~0.63

▲ COLLAPSE ▾

Details:

All code screenshots must have ucid/date visible.
Any terminal output should have at least 3 clients involved.

Conditions:

Columns: 1

Sub-Task

100%

Group: Round End

Task #4: Next Round or Session End

Sub Task #1: Show the code that checks how many eligible Players are left

Task Screenshots

Gallery Style: 2 Columns

4

2

1

```
private void evaluateRoundEnd() {
    List<ServerPlayer> remainingPlayers = playersDatabase.values().stream()
        .filter(player >= 3).filter(player.getIsDead() == false)
        .collect(Collectors.toList());
    if (remainingPlayers.size() == 1) {
        ServerPlayer winner = remainingPlayers.get(0);
        sendMessage(ServerConstants.PROF_ROOK, winner.getUcid(), "winner", Color.YELLOW);
        message("1 player remains");
    } else if (remainingPlayers.size() > 1) {
        sendMessage(ServerConstants.PROF_ROOK, message("more than 1 player remains"));
        evaluatePlayerChoices();
        eliminate();
    } else {
        sendMessage(ServerConstants.PROF_ROOK, message("0% ucid"));
        evaluate();
    }
}
```

```
private void evaluateRoundEnd() {
    List<ServerPlayer> remainingPlayers = playersDatabase.values().stream()
        .filter(player >= 3).filter(player.getIsDead() == false)
        .collect(Collectors.toList());
    if (remainingPlayers.size() == 1) {
        ServerPlayer winner = remainingPlayers.get(0);
        sendMessage(ServerConstants.PROF_ROOK, winner.getUcid(), "winner", Color.YELLOW);
        message("1 player remains");
    } else if (remainingPlayers.size() > 1) {
        sendMessage(ServerConstants.PROF_ROOK, message("more than 1 player remains"));
        evaluatePlayerChoices();
        eliminate();
    } else {
        sendMessage(ServerConstants.PROF_ROOK, message("0% ucid"));
        evaluate();
    }
}
```

code that checks how many players are left

the method being called after handle result in round end

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

If there is one player left then a winner is declared and the session ends. If there is more than one remaining player the player's choices are reset and another round begins. If there is a tie then the session ends.

Sub-Task

Group: Round End

Task #4: Next Round or Session End

Sub Task #2: Show the relevant terminal output

Task Screenshots

Gallery Style: 2 Columns

4 2 1

next round/session end

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

End of Task 4

End of Group: Round End

Task Status: 4/4

Group

Group: Session End

Tasks: 3

Points: 1.25

[COLLAPSE](#)

Task

100%

Group: Session End

Task #1: Session End Conditions

Weight: ~33%

Points: ~0.42

[▲ COLLAPSE ▾](#)

i Details:

All code screenshots must have ucid/date visible.



Columns: 1

Sub-Task

100%

Group: Session End

Task #1: Session End Conditions

Sub Task #1: Show the code that ends a session with a winner (likely similar to a previous task)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
private void handleSessionEnd() {
    List<ServerPlayer> remainingPlayers = players.values().stream()
        .filter(player -> !player.isRemoved() && player.getChoice() != null)
        .collect(Collectors.toList());
    String message = ServerConstants.SCREEN_NAME + " " + remainingPlayers.size() + " left! " + color.toString();
    sendMessage(ServerConstants.SCREEN_NAME, message);
}

if (remainingPlayers.size() == 1) {
    ServerPlayer winner = remainingPlayers.get(0);
    String message = "Match over! " + winner.getName() + " won! " + color.toString();
    sendMessage(ServerConstants.SCREEN_NAME, message);
}
```

ends a session with a winner, session end called right after

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

If only one player remains in the game, then the session ends and the a message is broadcast to clients about the who won.

Sub-Task

Group: Session End

Task #1: Session End Conditions

Sub Task #2: Show the code that ends a session with a tie (likely similar to a previous task)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
    else {
        sendUsage(ServerConstants.FROM_ROOM, message("TIE GAME"));
        endSession();
    } <- return else
} <- ends the private void evaluateRound()
```

ends the session with a tie

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

If there are 0 players left then the session ends and a message stating that its a tie gets sent out to all clients.

End of Task 1

Task

Group: Session End
Task #2: Scoreboard
Weight: ~33%
Points: ~0.42

▲ COLLAPSE ▲

i Details:

All code screenshots must have ucid/date visible.
Any terminal output should have at least 3 clients involved.



Columns: 1

Sub-Task

Group: Session End
Task #2: Scoreboard
Sub Task #1: Show the code related to the final scoreboard



Task Screenshots

Gallery Style: 2 Columns

4 2 1

code related to the final scoreboard

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

First it sorts the players and puts them in list by getting all the players in the room. Then from the list it gets the points of player one and player two and compares them and list them, so that the person with the most points will go first. Then it takes the sorted players and outputs their information such as their names, the amount of points they have and whether or not they have been eliminated and the message is printed vertically.

Sub-Task

Group: Session End



Task #2: Scoreboard

Sub Task #2: Show the final scoreboard from the terminal

Task Screenshots

Gallery Style: 2 Columns

final scoreboard

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

End of Task 2

Task

Group: Session End



Task #3: Cleanup

Weight: ~33%

Points: ~0.42

1 Details:

All code screenshots must have ucid/date visible.
Any terminal output should have at least 3 clients involved.



Columns: 1

Sub-Task

Group: Session End

Task #3: Cleanup

100%

Sub Task #1: Show the code related to resetting the Player data client-side and server-side (do not disconnect them or move them to the lobby)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
38 private void resetPlayer() {
39     playerSimmon.values().foreach(player::reset);
40 }
41
42
43
44
45
```

resetting player data 1

```
protected void onSessionEnd() {
    LoggerUtil.INSTANCE.info(message:"onSessionEnd() start");
    resetReadyStatus();
}
```

resetting player data 2

```
private void processReadyStatus(Player client) {
    knownClients.values().foreach(player::resetReadyStatus);
    System.out.println("Ready status " + client.getClientID());
}
private void processReadyStatus(long clientId, boolean isReady, boolean isSelf) {
    if (knownClients.containsKey(clientId)) {
        ClientPlayer cp = knownClients.get(clientId);
        if (isReady) {
            cp.setReady(true);
            cp.setClientName(cp.getClientName());
            cp.setClientID(clientId);
            cp.setClientType(cp.getClientType());
            cp.setEliminated(false);
            cp.setPoints(0);
            String format = String.format("%s[%s] is %s", cp.getClientName(), cp.getClientID(),
                isReady ? "Ready" : "Not ready");
            System.out.println(format);
        } else {
            cp.setReady(false);
            cp.setClientName(null);
            cp.setClientID(null);
            cp.setClientType(null);
            cp.setEliminated(true);
            cp.setPoints(0);
        }
    }
}
```

client side

```
public void reset() {
    this.clientID = Player.DEFAULT_CLIENT_ID;
    this.takeTurn = false;
    this.points = 0;
    this.choice = null;
    this.clientName = null;
    this.eliminated = false;
}
<- #92-100 public void reset()
}
<- #6-101 public class Player
```

Server Side

Caption(s) (required) ✓

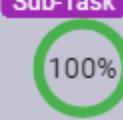
Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

The reset player sets choices to null, brings back eliminated players, and resets the their choice, and resets their points as well, and whether or not they took their turn, sets the clients name to null , point to zero and gives them the default client_ID. It calls on the reset from the player. To reset whether or not the player is ready, I used the resetReadyStatus method.



Group: Session End
Task #3: Cleanup
Sub Task #2: Show any terminal evidence

Task Screenshots

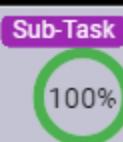
Gallery Style: 2 Columns

4 2 1

cleanup

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*



Group: Session End

Task #3: Cleanup

Sub Task #3: Show the code related to shifting back to the ready Phase

Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
    @Override  
    protected void onSessionEnd() {  
        LoggerUtil.INSTANCE.info(message:"onSessionEnd() start");  
        resetReadyStatus();  
        changePhase(Phase.READY);  
        LoggerUtil.INSTANCE.info(message:"onSessionEnd() end");  
    } <-- #164-169 protected void onSessionEnd()
```

code shifting back to ready phase

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

In the `OnSessionEnd` change phase is set to the ready phase. It also resets the player's ready status with the `resetReadyStatus`. Where the `resetReadyStatus` checkers for the amount players of the room and then for each player sets whether or not they are ready, and then it resets the `resetreadytrigger`.

End of Task 3

End of Group: Session End

Task Status: 3/3

Group



Group: Misc

Tasks: 3

Points: 1

[▲ COLLAPSE ▲](#)

Task



Group: Misc

Task #1: Add the pull request link for the branch

Weight: ~33%

Points: ~0.33

[▲ COLLAPSE ▲](#)

 Details:

Note: the link should end with /pull/#

🔗 Task URLs

URL #1

<https://github.com/Cae6/cae6-IT114-003/pull/9/>

URL

<https://github.com/Cae6/cae6-IT114-003/pull/9/>

End of Task 1

Task



Group: Misc

Task #2: Talk about any issues or learnings during this assignment

Weight: ~33%

Points: ~0.33

[▲ COLLAPSE ▲](#)

📝 Task Response Prompt

Response:

I had a few issues doing this milestone, especially when it came to making sure the flow between the client, server thread, and game room was correct and made sense. I kept forgetting small things and it made me think that I had a larger issue. So I ended up redoing parts multiple times because of this. But as I went along and learned more as I did it, I started to look for the smaller mistakes first. I had an issue where my choices weren't being processed and it was due

I started to look for the smaller mistakes first. I had an issue where my choices weren't being processed and it was due to a small issue as well that went unnoticed. I also think I should have been more vocal about my issues because I spent a lot of time on small things for a very long time. I also intend to work on managing my time better with these assignments and making my logic a better as I go along.

End of Task 2

Task



Group: Misc

Task #3: WakaTime Screenshot

Weight: ~33%

Points: ~0.33

[▲ COLLAPSE ▲](#)

Details:

Grab a snippet showing the approximate time involved that clearly shows your repository. The duration isn't considered for grading, but there should be some time involved



Task Screenshots

Gallery Style: 2 Columns

4 2 1



Wakatime screenshot

End of Task 3

End of Group: Misc

Task Status: 3/3

End of Assignment