Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-002-S2024/it114-milestone-2-rps-2024/grade/pd438

IT114-002-S2024 - [IT114] Milestone 2 RPS 2024

Submissions:

Submission Selection

1 Submission [active] 4/19/2024 2:47:01 PM

Instructions

COLLAPSE

Implement the Milestone 2 features from the project's proposal document:
https://docs.google.com/document/d/11SRMo7JkLAMM-PuuiGwl_Z-QXP3pyQ7xN3lRxwmcwCc/view
Make sure you add your ucid/date as code comments where code changes are done
All code changes should reach the Milestone2 branch
Create a pull request from Milestone2 to main and keep it open until you get the output PDF from this assignment.
Gather the evidence of feature completion based on the below tasks.
Once finished, get the output PDF and copy/move it to your repository folder on your local machine.
Run the necessary git add, commit, and push steps to move it to GitHub
Complete the pull request that was opened earlier
Upload the same output PDF to Canvas

Branch name: Milestone2

Tasks: 14 Points: 10.00

Server will have the functionality to switch users between "Rooms" and "GameRooms" (1 pt.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Task 1

Details:

(i.e., all start in "Lobby", but can then create or join existing rooms to be with a particular group)

Checklist

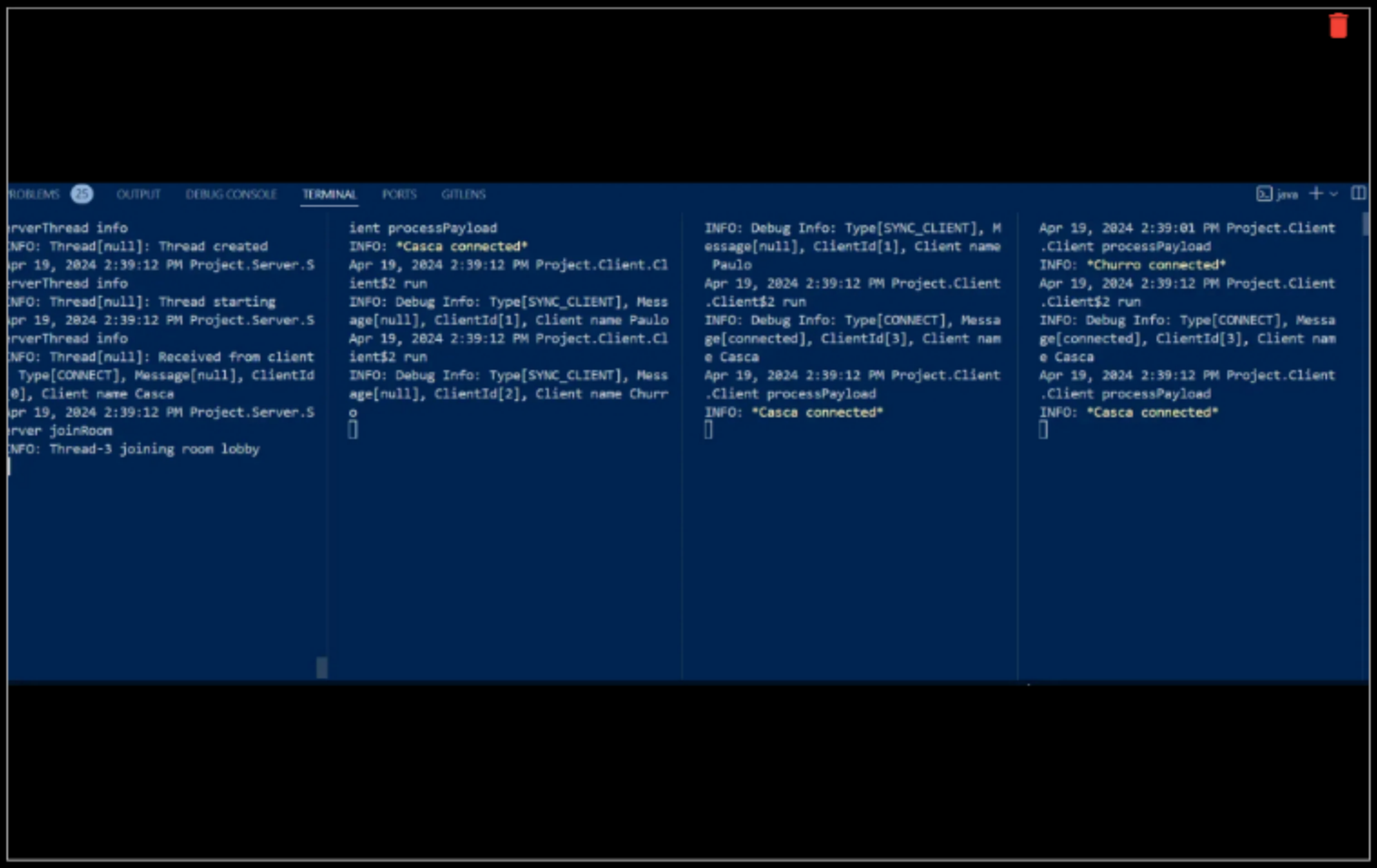
*The checkboxes are for your own tracking

#	Points	Details
#1	1	Show the lobby being created
#2	1	Show new room sessions being created
#3	1	Code screenshots should include ucid and date comment
#4	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large



this displays when user does /connect localhost:3000

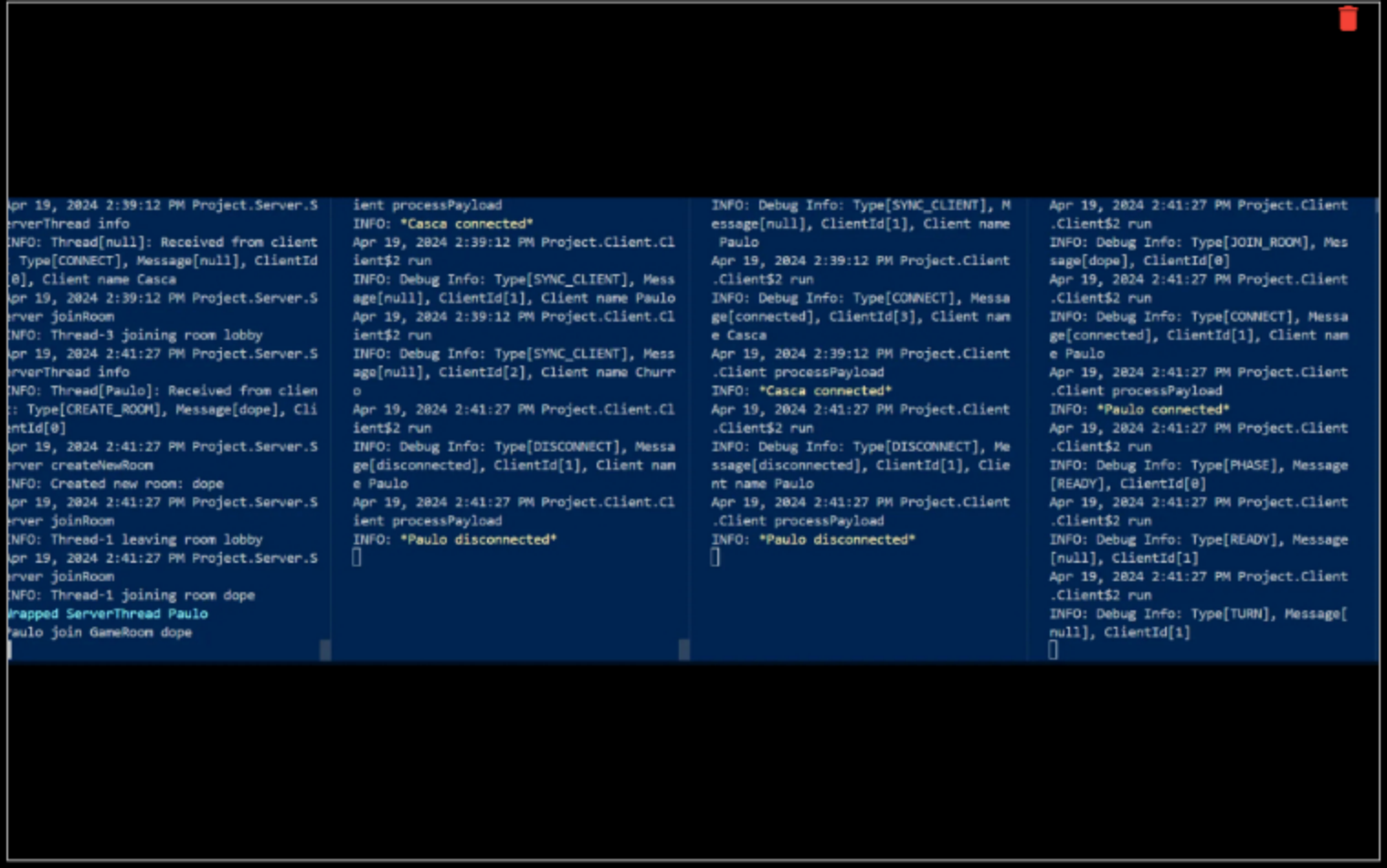
Checklist Items (4)

#1 Show the lobby being created

#2 Show new room sessions being created

#3 Code screenshots should include ucid and date comment

#4 Each screenshot should be clearly captioned



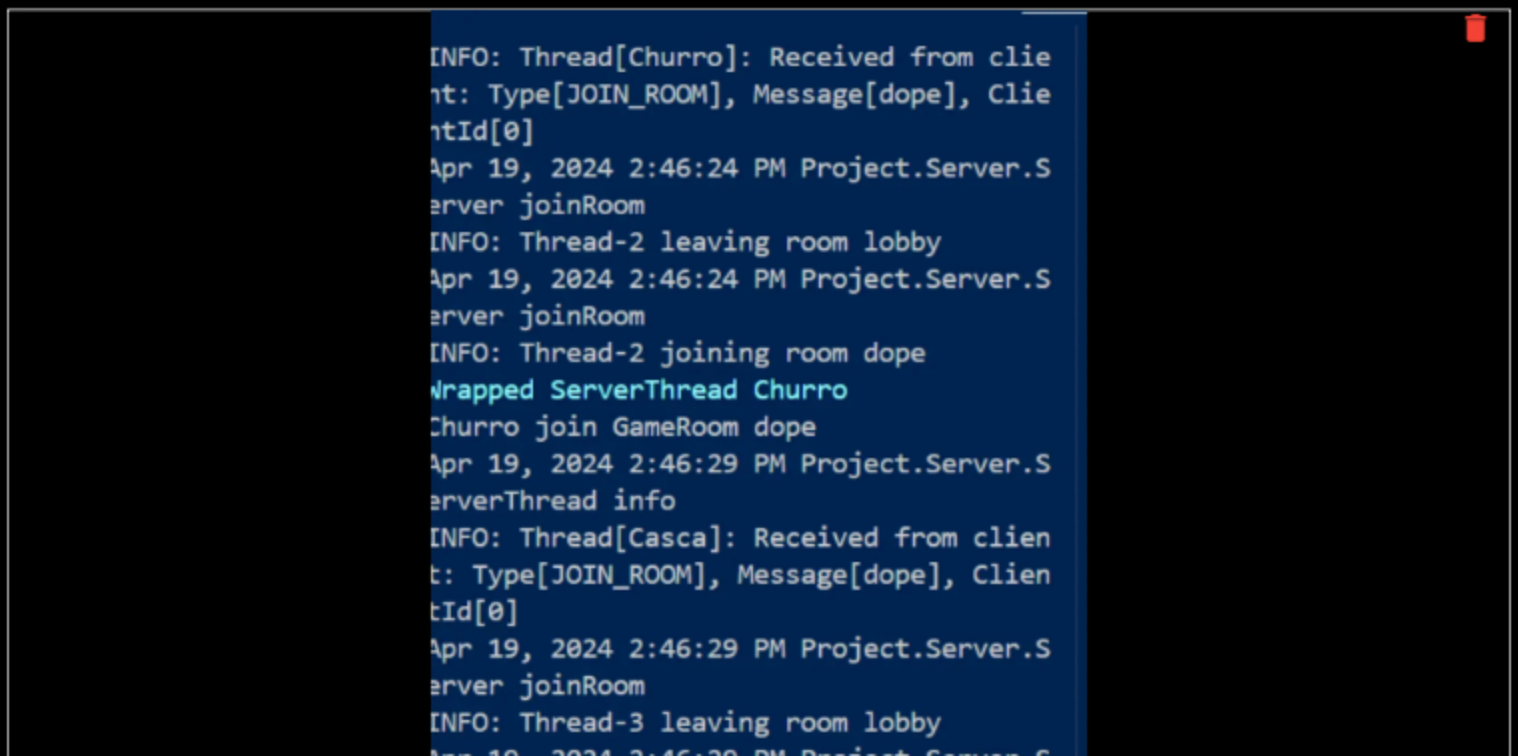
this image will display when user creates lobby

Checklist Items (3)

#2 Show new room sessions being created

#3 Code screenshots should include ucid and date comment

#4 Each screenshot should be clearly captioned



```
Apr 19, 2024 2:46:29 PM Project.Server.S
erver joinRoom
INFO: Thread-3 joining room dope
Wrapped ServerThread Casca
Casca join GameRoom dope
```

message to show what happens when user does /joinroom

Checklist Items (0)

Demonstrate Usage of Payloads (2 pts.)

^COLLAPSE^

^COLLAPSE^

Task #1 - Points: 1

Text: Screenshots of your Payload class and subclasses and PayloadType

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Payload, ReadyPayload, Phases, ChoicePayload, and any others or equivalents
#2	1	Code screenshots should include ucid and date comment
#3	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large

```
package Project.Common;

public enum PayloadType {
    CONNECT, DISCONNECT, MESSAGE, CREATE_ROOM, JOIN_ROOM, LIST_ROOMS, CLIENT_ID, SYNC_CLIENT, READY, PHASE, TURN,
    RESET_TURNS, RESET_READY, CURRENT_TURN,CHOICE,ELIMINATION
                                    //pd438 Created Payload Type Choice, and Elimination      You, 2 days
}
```

payload type.

Checklist Items (3)

#1 Payload, ReadyPayload, Phases, ChoicePayload, and any others or equivalents

#2 Code screenshots should include ucid and date comment

#3 Each screenshot should be clearly captioned

```java
...
package Project.Common;

...
public class EliminationPayload extends Payload {
    //pd438 4/29/2024
    private boolean isEliminated;

    public boolean isEliminated() {
        return isEliminated;
    }

    public void setElimination(boolean isEliminated){
        this.isEliminated = isEliminated;
    }

    public EliminationPayload() {
        setPayloadType(PayloadType.ELIMINATION);
    }
} <- #3-18 public class EliminationPayload extends Payload

```

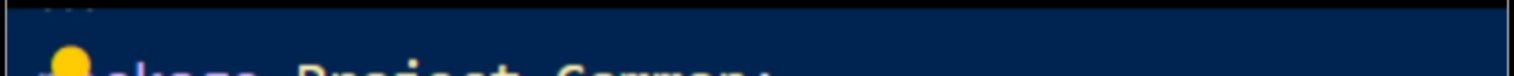
Elimination Payload

Checklist Items (3)

#1 Payload, ReadyPayload, Phases, ChoicePayload, and any others or equivalents

#2 Code screenshots should include ucid and date comment

#3 Each screenshot should be clearly captioned



```java
package Project.Common;
//pd438 4/19/2024
public enum Phase {
    READY, IN_PROGRESS, TURN, BATTLE
}
```

Phases

Checklist Items (3)

#1 Payload, ReadyPayload, Phases, ChoicePayload, and any others or equivalents

#2 Code screenshots should include ucid and date comment

#3 Each screenshot should be clearly captioned

```java
package Project.Common;

You, 2 days ago | 1 author (You)
public class ReadyPayload extends Payload {

    private boolean isReady;

    public boolean isReady() {
        return isReady;
    }

    public void setReady(boolean isReady) {
        this.isReady = isReady;
    }

    public ReadyPayload() {
        setPayloadType(PayloadType.READY);
    }
}
```

Ready Payload

Checklist Items (3)

#1 Payload, ReadyPayload, Phases, ChoicePayload, and any others or equivalents

#2 Code screenshots should include ucid and date comment

#3 Each screenshot should be clearly captioned

COLLAPSE

Task #2 - Points: 1

Text: Screenshots of the payloads being debugged/output to the terminal

Checklist

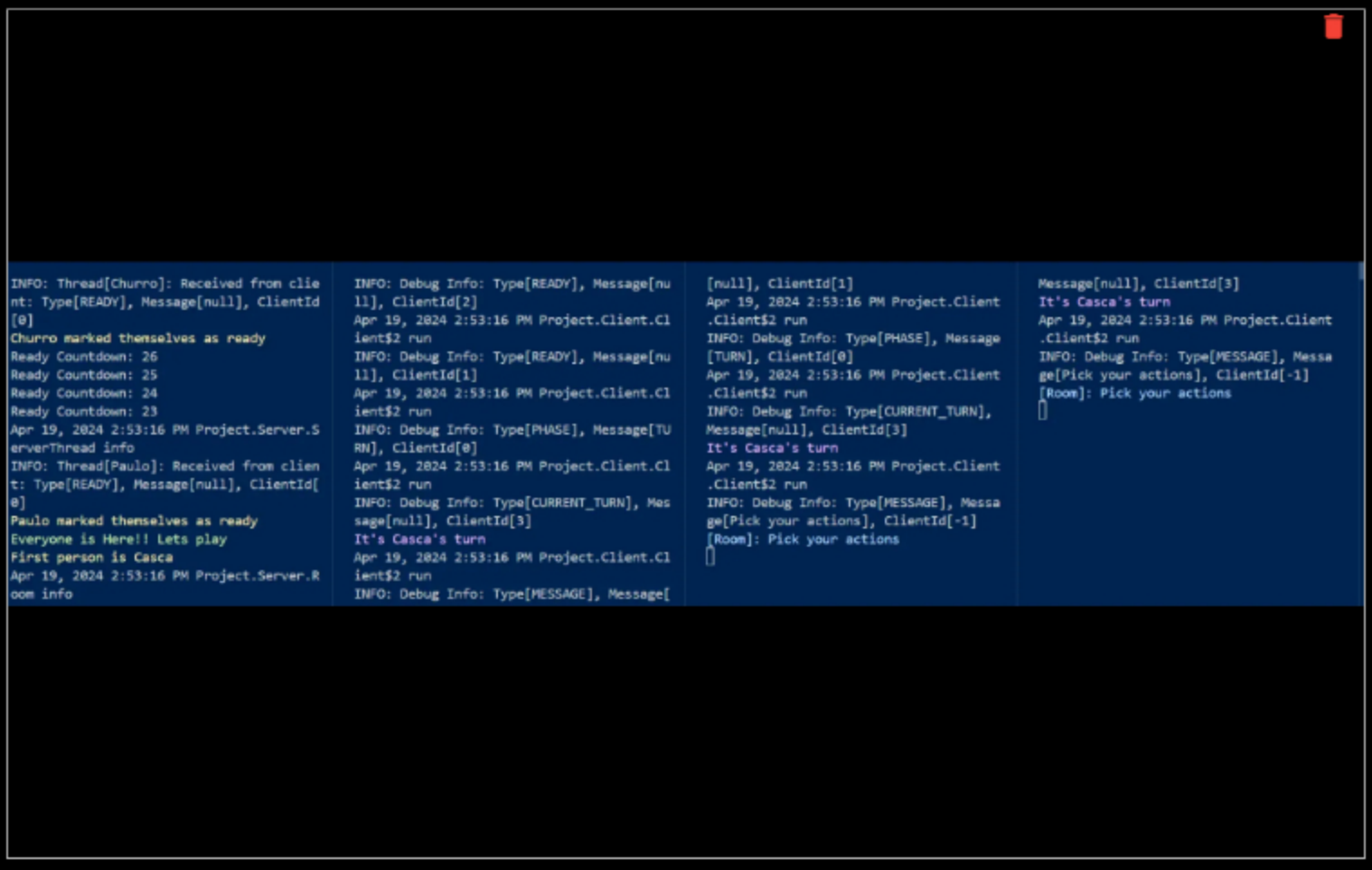
*The checkboxes are for your own tracking

#	Points	Details
#1	1	Demonstrate players sending choices and getting a confirmation
#2	1	Demonstrate players receiving "battle" results
#3	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large



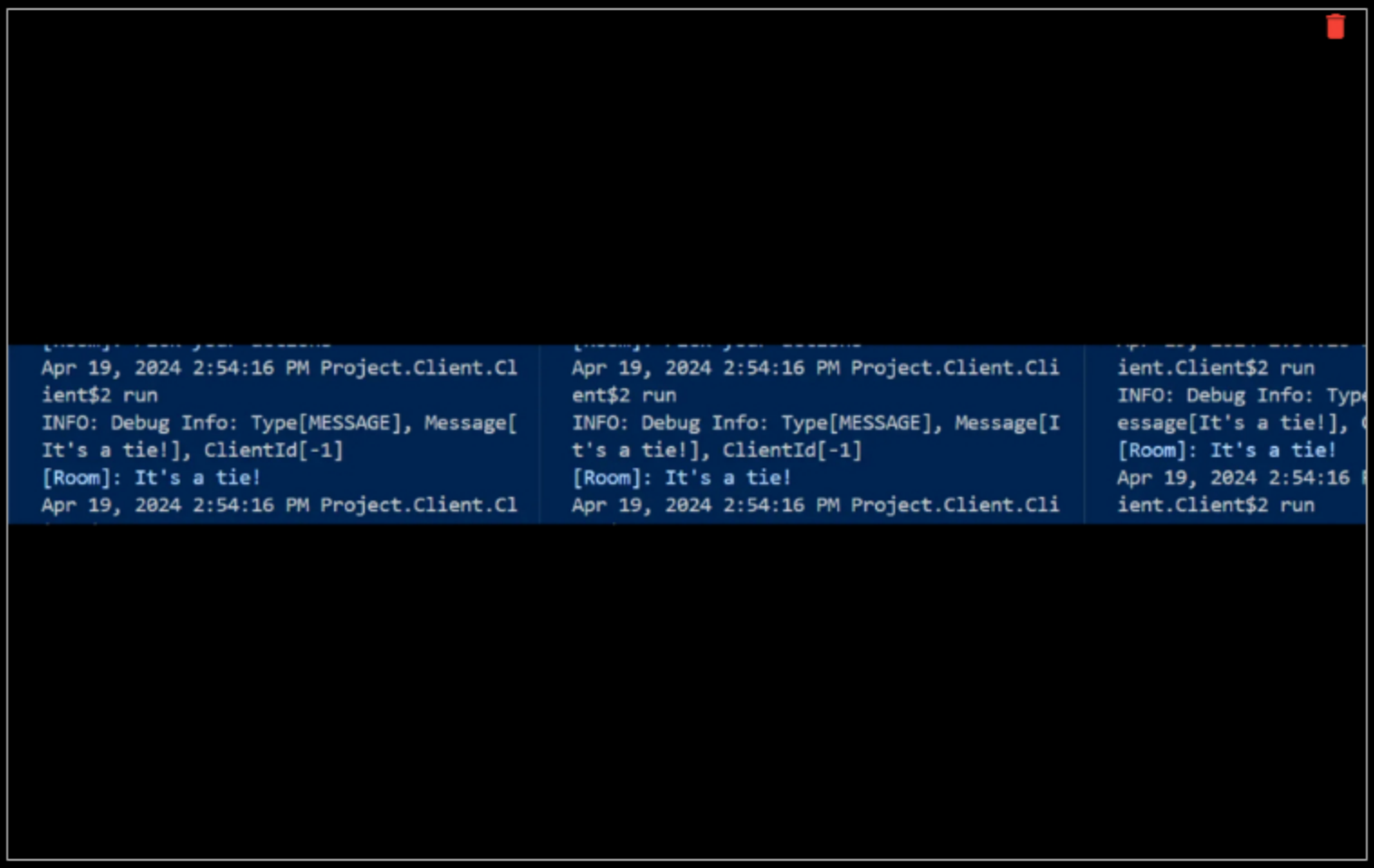
This is what shows when the game gets started.

Checklist Items (3)

#1 Demonstrate players sending choices and getting a confirmation

#2 Demonstrate players receiving "battle" results

#3 Each screenshot should be clearly captioned

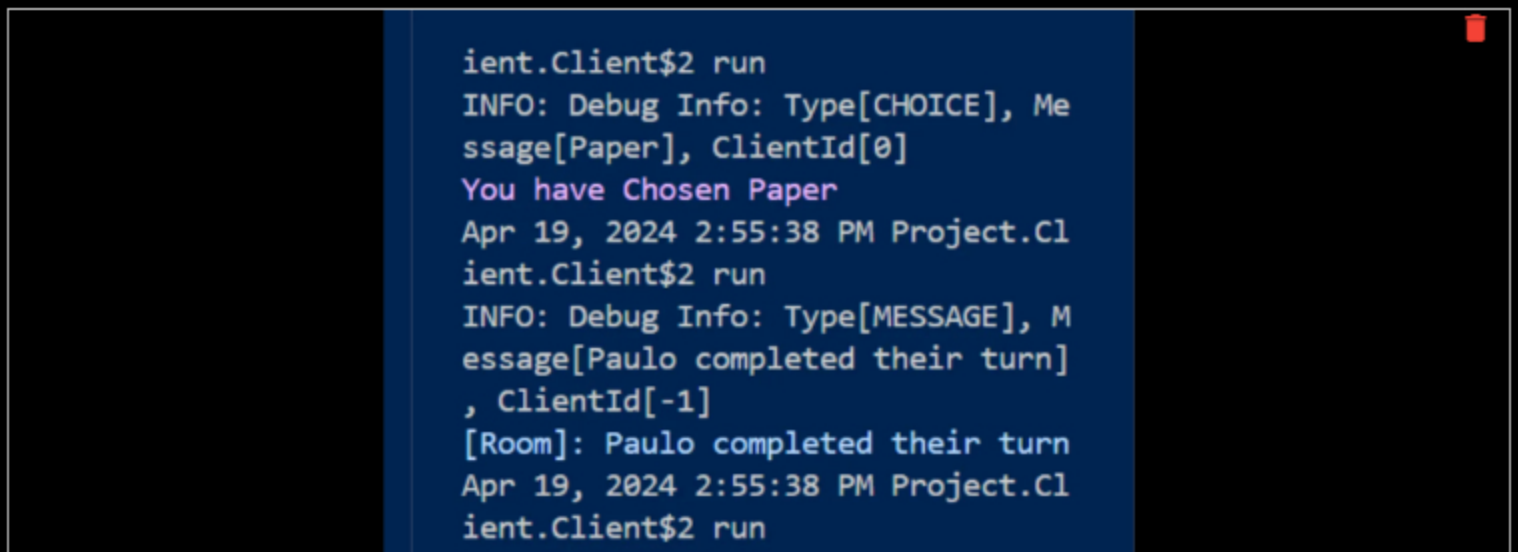


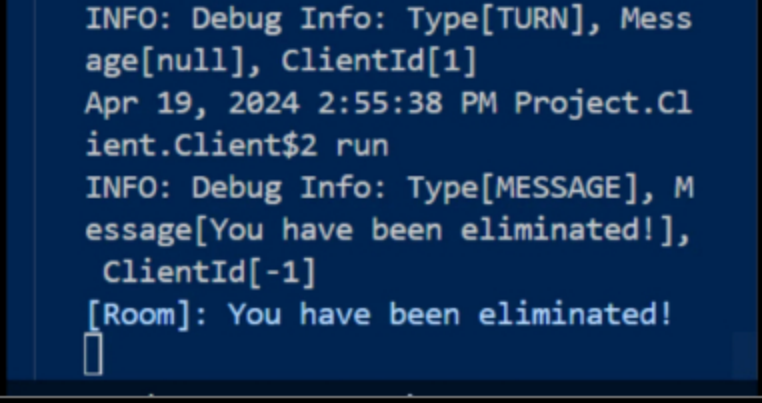
this shows when the user does not place thier choice.

Checklist Items (2)

#2 Demonstrate players receiving "battle" results

#3 Each screenshot should be clearly captioned





This displays when user confirms their choice. and if they are eliminated.

Checklist Items (0)

COLLAPSE

Task #3 - Points: 1

Text: Explain the purpose of each payload and how each of the new payload subclass and payload types are used

Response:

The purpose of each payload is to allow the user to be able to confirm their choice since they are handled through the client. then it gets passed to the serverthread and then gets placed into the gameroom in which it gets processed and gets sent back to the user.

COLLAPSE

Turn Handling (3 pts.)

COLLAPSE

Task #1 - Points: 1

Text: Server-side screenshots of the following

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Code related to choosing turn order or separating turn phases if turns aren't used
#2	1	Code related to picking and syncing turns or recording that a turn was handled if turns aren't used
#3	1	Code related to the turn/round timer and how an expired turn/round is handled
#4	1	Code screenshots should include ucid and date comment
#5	1	Each screenshot should be clearly captioned

Task Screenshots:

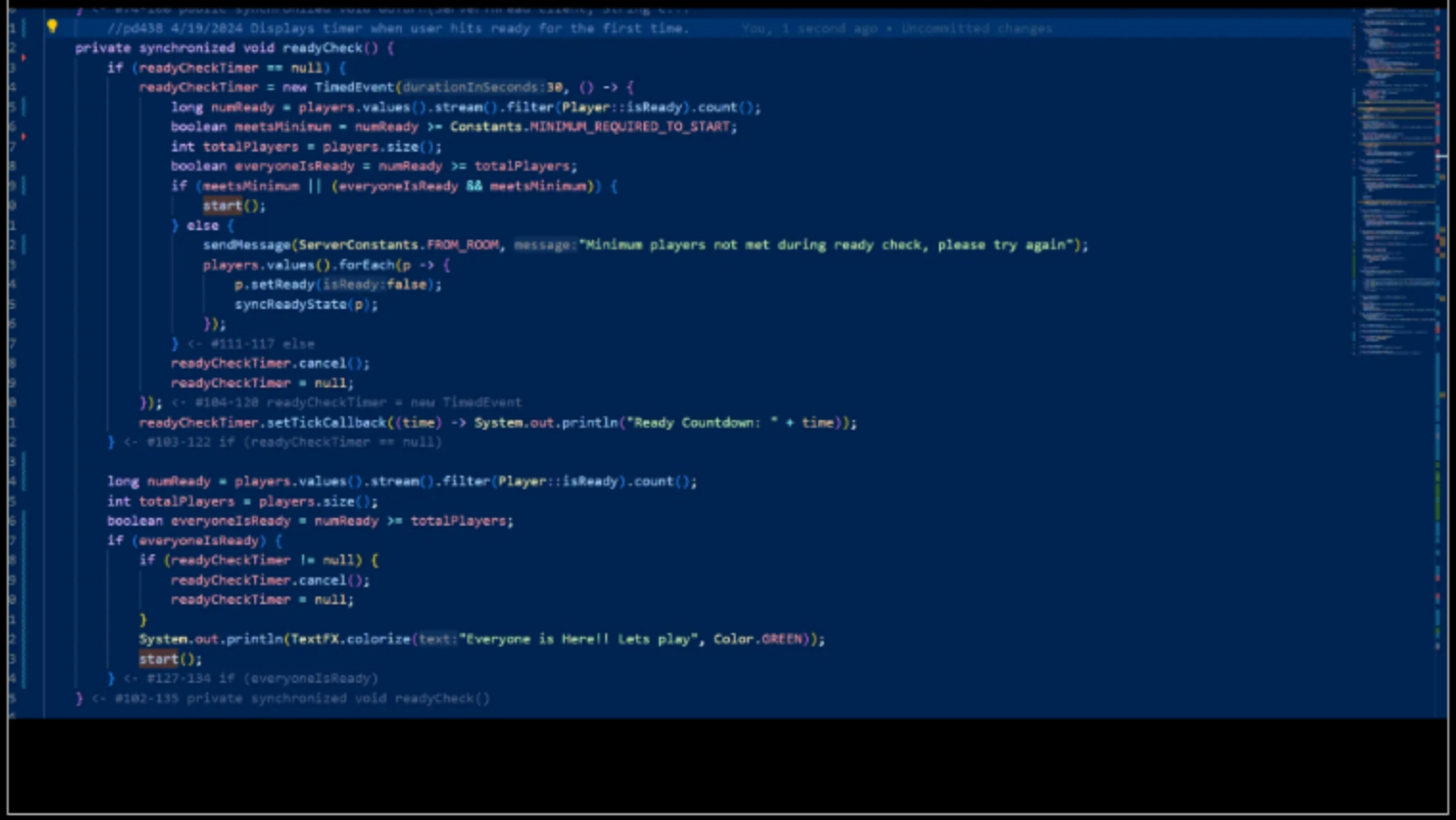
Gallery Style: Large View

Small Medium Large

```
    // 4/19/2024 pd438 displays randomized turn      You, 1 second ago • Uncommitted changes
private void setupTurns() {
    turnOrder = new ArrayList<>(players.keySet());
    Collections.shuffle(turnOrder);
    currentPlayer = players.get(turnOrder.get(index:0));
    System.out.println(TextFX.colorize("First person is " + currentPlayer.getClientName(), Color.YELLOW));
    sendCurrentPlayerTurn();
} <- #147-153 private void setupTurns()
```

this code displays on turn order and who will go first. this shows that at random who will go first.

Checklist Items (0)



```
    //pd438 4/19/2024 Displays timer when user hits ready for the first time.      You, 1 second ago • Uncommitted changes
private synchronized void readyCheck() {
    if (readyCheckTimer == null) {
        readyCheckTimer = new TimedEvent(durationInSeconds:30, () -> {
            long numReady = players.values().stream().filter(Player::isReady).count();
            boolean meetsMinimum = numReady >= Constants.MINIMUM_REQUIRED_TO_START;
            int totalPlayers = players.size();
            boolean everyoneIsReady = numReady >= totalPlayers;
            if (meetsMinimum || (everyoneIsReady && meetsMinimum)) {
                start();
            } else {
                sendMessage(ServerConstants.FROM_ROOM, message:"Minimum players not met during ready check, please try again");
                players.values().forEach(p -> {
                    p.setReady(isReady:false);
                    syncReadyState(p);
                });
            } <- #111-117 else
            readyCheckTimer.cancel();
            readyCheckTimer = null;
        }); <- #104-120 readyCheckTimer = new TimedEvent
        readyCheckTimer.setTickCallback((time) -> System.out.println("Ready Countdown: " + time));
    } <- #103-122 if (readyCheckTimer == null)

    long numReady = players.values().stream().filter(Player::isReady).count();
    int totalPlayers = players.size();
    boolean everyoneIsReady = numReady >= totalPlayers;
    if (everyoneIsReady) {
        if (readyCheckTimer != null) {
            readyCheckTimer.cancel();
            readyCheckTimer = null;
        }
        System.out.println(TextFX.colorize(text:"Everyone is Here!! Lets play", Color.GREEN));
        start();
    } <- #127-134 if (everyoneIsReady)
} <- #102-135 private synchronized void readyCheck()
```

This is the readyCheck counts down in 30 seconds and what will happen.

Checklist Items (3)

#1 Code related to choosing turn order or separating turn phases if turns aren't used

#2 Code related to picking and syncing turns or recording that a turn was handled if turns aren't used

#3 Code related to the turn/round timer and how an expired turn/round is handled

COLLAPSE

Task #2 - Points: 1

Text: Explain the server-side logic from the previous task

Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	How is turn order handled (details about the turn-by-turn process or how phases are used if not using turns)
#2	1	What are the steps for picking and syncing turns (turn-by-turn process or usage of phases)
#3	1	Explain the usage of the timer and what its expiry does

Response:

Check if the clients are valid with names, and when players are ready, user must input their choice and will compare both answers to decide on who won. If the time is up and no one is ready, players will receive the message that it is ended. Starting of the turns are random. One player will be decided to start. if player did not select a choice, then the game cannot continue.

COLLAPSE

Task #3 - Points: 1

Text: Client-side screenshots of the following

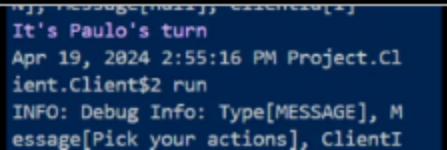
Checklist *The checkboxes are for your own tracking

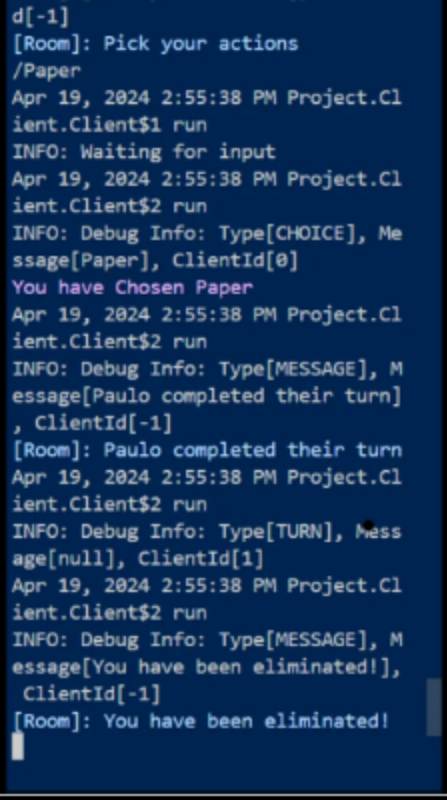
#	Points	Details
#1	1	Being notified of whose turn it is (or if not using turns showing that it's the correct phase to make choices)
#2	1	Responses to taking a turn (success message of recording a choice, etc)
#3	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large





Here is a screenshot to display who turn it is. and in this instance displays messages confirmation of what the client selected. and whether or not they are eliminated.

Checklist Items (3)

#1 Being notified of whose turn it is (or if not using turns showing that it's the correct phase to make choices)

#2 Responses to taking a turn (success message of recording a choice, etc)

#3 Each screenshot should be clearly captioned

Handling Game actions (3 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshots of the following server-side code

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	If using turns, show the code that handles the correct person, otherwise show how a choice is recorded (out/skipped players can't take an action)
#2	1	The code that handles a player skipping (they shouldn't be included in this round and will not be removed from next round)
#3	1	The code to check if all applicable players made a choice or skipped
#4	1	The code that processes the player "battles" in round-robin style (i.e., p1 -> p2 -> p3 -> p0)

#5	1	The code that checks if there's a winner (1 remaining), a tie (none remaining), or another round (2+ remaining)
#6	1	The code that resets the state for a new round (2+ remaining)
#7	1	The code that resets the state for a new session (win/tie)
#8	1	Code screenshots should include ucid and date comment
#9	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large

```
@Override
protected synchronized void addClient(ServerThread client) {
    super.addClient(client);
    if (!players.containsKey(client.getClientId())) {
        ServerPlayer sp = new ServerPlayer(client);
        players.put(client.getClientId(), sp);
        System.out.println(TextFX.colorize(client.getClientName() + " join GameRoom " + getName(), Color.WHITE));

        sp.sendPhase(currentPhase);

        players.values().forEach(p -> {
            sp.sendReadyState(p.getClientId(), p.isReady());
            sp.sendPlayerTurnStatus(p.getClientId(), p.didTakeTurn());
        });
        if (currentPlayer != null) {
            sp.sendCurrentPlayerTurn(currentPlayer.getClientId());
        }
    } <- #32-46 if (!players.containsKey(client.getClientId()))
} <- #30-47 protected synchronized void addClient(ServerThread client)

@Override
protected synchronized void removeClient(ServerThread client) {
    super.removeClient(client);
    if (players.containsKey(client.getClientId())) {
        players.remove(client.getClientId());
        System.out.println(TextFX.colorize(client.getClientName() + " left GameRoom " + getName(), Color.WHITE));
    }
} <- #50-56 protected synchronized void removeClient(ServerThread client)
```

gives the message to the client about the tie.

Checklist Items (0)

```
public synchronized void setReady(ServerThread client) {
    if (currentPhase != Phase.READY) {
        client.sendMessage(Constants.DEFAULT_CLIENT_ID, message:"Can't initiate ready check at this time");
        return;
    }
    long playerId = client.getClientId();
    if (players.containsKey(playerId)) {
        players.get(playerId).setReady(isReady:true);
        syncReadyState(players.get(playerId));
        System.out.println(TextFX.colorize(players.get(playerId).getClientName() + " marked themselves as ready.", Color.YELLOW));
```

```
        readyCheck();
    } else {
        System.err.println(TextFX.colorize("Player doesn't exist: " + client.getClientName(), Color.RED));
    }
} <- #58-72 public synchronized void setReady(ServerThread client)
```

setReady to get ready

Checklist Items (0)

```
public synchronized void doTurn(ServerThread client, String choice) {
    if (currentPhase != Phase.TURN) {
        client.sendMessage(Constants.DEFAULT_CLIENT_ID, message:"You can't do turns just yet");
        return;
    }

    long clientId = client.getClientId();
    if (players.containsKey(clientId)) {
        ServerPlayer sp = players.get(clientId);
        if (!sp.isReady()) {
            client.sendMessage(Constants.DEFAULT_CLIENT_ID, message:"Sorry, you weren't ready in time and can't participate");
            return;
        }
        if (!sp.didTakeTurn()) {
            sp.setChoice(choice);
            sp.sendChoice(choice);
            sp.setTakenTurn(takenTurn:true);
            sendMessage(ServerConstants.FROM_ROOM, String.format(format:"%s completed their turn", sp.getClientName()));
            syncUserTookTurn(sp);
            if (currentPlayer != null && currentPlayer.didTakeTurn()) {
                handleEndOfTurn();
            }
        } else {
            client.sendMessage(Constants.DEFAULT_CLIENT_ID, message:"You already completed your turn, please wait");
        }
    } <- #81-99 if (players.containsKey(clientId))
} <- #74-100 public synchronized void doTurn(ServerThread client, String c...
```

doTurn code to display

Checklist Items (0)

```
} <- #74-100 public synchronized void doTurn(ServerThread client, String c...
    //pd438 4/19/2024 Displays timer when user hits ready for the first time.
private synchronized void readyCheck() {
    if (readyCheckTimer == null) {
        readyCheckTimer = new TimedEvent(durationInSeconds:30, () -> {
            long numReady = players.values().stream().filter(Player::isReady).count();
            boolean meetsMinimum = numReady >= Constants.MINIMUM_REQUIRED_TO_START;
            int totalPlayers = players.size();
            boolean everyoneIsReady = numReady >= totalPlayers;
            if (meetsMinimum || (everyoneIsReady && meetsMinimum)) {
                start();
            } else {
                sendMessage(ServerConstants.FROM_ROOM, message:"Minimum players not met during ready check, please try again");
                players.values().forEach(p -> {
                    p.setReady(isReady:false);
                    syncReadyState(p);
                });
            } <- #111-117 else
            readyCheckTimer.cancel();
```

```
            readyCheckTimer = null;
        }); <- #104-120 readyCheckTimer = new TimedEvent
        readyCheckTimer.setTickCallback((time) -> System.out.println("Ready Countdown: " + time));
    } <- #103-122 if (readyCheckTimer == null)

    long numReady = players.values().stream().filter(Player::isReady).count();
    int totalPlayers = players.size();
    boolean everyoneIsReady = numReady >= totalPlayers;
    if (everyoneIsReady) {
        if (readyCheckTimer != null) {
            readyCheckTimer.cancel();
            readyCheckTimer = null;
        }
        System.out.println(TextFX.colorize(text:"Everyone is Here!! Lets play", Color.GREEN));
        start();
    } <- #127-134 if (everyoneIsReady)
} <- #102-135 private synchronized void readyCheck()
```

ReadyCheck displayed

Checklist Items (0)

```
    private void start() {
        if (currentPhase != Phase.READY) {
            System.err.println(x:"Invalid phase called during start()");
            return;
        }
        changePhase(Phase.TURN);
        setupTurns();
        startTurnTimer();
    } <- #137-145 private void start()
        // 4/19/2024 pd438 displays randomized turn
    private void setupTurns() {
        turnOrder = new ArrayList<>(players.keySet());
        Collections.shuffle(turnOrder);
        currentPlayer = players.get(turnOrder.get(index:0));
        System.out.println(TextFX.colorize("First person is " + currentPlayer.getClientName(), Color.YELLOW));
        sendCurrentPlayerTurn();
    } <- #147-153 private void setupTurns()

    private void nextTurn() {
        int index = turnOrder.indexOf(currentPlayer.getClientId());
        index = (index + 1) % turnOrder.size();
        currentPlayer = players.get(turnOrder.get(index));
        System.out.println(TextFX.colorize("Next person is " + currentPlayer.getClientName(), Color.YELLOW));
        sendCurrentPlayerTurn();
    } <- #155-161 private void nextTurn()
        //pd438 4/19/2024 This displays the timer for the person to let them know how much time they have left.
    private void startTurnTimer() {
        if (turnTimer != null) {
            turnTimer.cancel();
            turnTimer = null;
        }       You, 2 days ago • Milestone2 Improvements
        if (turnTimer == null) {
            turnTimer = new TimedEvent(durationInSeconds:30, this::handleEndOfTurn);
            turnTimer.setTickCallback(this::checkEarlyEndTurn);
            sendMessage(ServerConstants.FROM_ROOM, message:"Pick your actions");
        } <- #168-172 if (turnTimer == null)
    } <- #163-173 private void startTurnTimer()
```

to display start,setup turns, nextturn and start the turn timer.

Checklist Items (0)

```
    private void checkEarlyEndTurn(int timeRemaining) {
        if (currentPlayer != null && currentPlayer.didTakeTurn()) {
            handleEndOfTurn();
        }
    } <- #175-179 private void checkEarlyEndTurn(int timeRemaining)
/pd438 4/19/2024
    private void handleEndOfTurn() {
        if (turnTimer != null) {
            turnTimer.cancel();
            turnTimer = null;
        }
        System.out.println(TextFX.colorize(text:"Handling end of turn", Color.YELLOW));

        List<ServerPlayer> playerList = new ArrayList<>(players.values());
        ServerPlayer eliminatedPlayer = resolveGame(playerList);
```

```
    if (eliminatedPlayer != null) {
        removeClient(eliminatedPlayer.getServerThread());
        eliminatedPlayer.getServerThread().sendMessage(Constants.DEFAULT_CLIENT_ID, message:"You have been eliminated!");
        sendMessage(ServerConstants.FROM_ROOM, TextFX.colorize(eliminatedPlayer.getClientName() + " is eliminated!", Color.RED));
        if (players.size() == 1) {
            end();
            return;
        }
    } <- #191-199 if (eliminatedPlayer != null)

    resetTurns();
    nextTurn();

    if (currentPhase != Phase.READY && players.size() > 2) {
        startTurnTimer();
    } else if (currentPhase != Phase.READY && players.size() == 2) {
        handleEndOfGame(); // Immediately resolve the game when there are only two players left
    }
```

Handling end of turn.

Checklist Items (0)

```
//pd438 4/19/2024 This displays what happens when end of game occurs.
private void handleEndOfGame() {
    System.out.println(TextFX.colorize(text:"Handling end of game", Color.YELLOW));

    // Resolve the game based on the choices of the two players
    List<ServerPlayer> playerList = new ArrayList<>(players.values());
    ServerPlayer eliminatedPlayer = resolveGame(playerList);

    if (eliminatedPlayer != null) {
        removeClient(eliminatedPlayer.getServerThread());
        eliminatedPlayer.getServerThread().sendMessage(Constants.DEFAULT_CLIENT_ID, message:"You have been eliminated!");
        sendMessage(ServerConstants.FROM_ROOM, TextFX.colorize(eliminatedPlayer.getClientName() + " is eliminated!", Color.RED));
        end(); // End the game
    } <- #218-223 if (eliminatedPlayer != null)
} <- #211-224 private void handleEndOfGame()
```

To handle the end of the game

Checklist Items (0)

```
private ServerPlayer resolveGame(List<ServerPlayer> players) {
    String[] choices = players.stream().map(ServerPlayer::getChoice).toArray(String[]::new);
    String[] uniqueChoices = Arrays.stream(choices).distinct().toArray(String[]::new);

    if (uniqueChoices.length == 1) {
        sendMessage(ServerConstants.FROM_ROOM, message:"It's a tie!");
        return null;
    }

    if (uniqueChoices.length == 3 || uniqueChoices.length == 1) {
        return null; // No elimination if everyone chose differently or if everyone chose the same
    }
```

```
        String choice1 = uniqueChoices[0];
        String choice2 = uniqueChoices[1];

        String winnerChoice = getWinnerChoice(choice1, choice2);

        ServerPlayer eliminatedPlayer = null;
        for (ServerPlayer player : players) {
            if (!player.getChoice().equalsIgnoreCase(winnerChoice)) {
                eliminatedPlayer = player;
                break;
            }
        } <- #245-250 for (ServerPlayer player : players)

        return eliminatedPlayer;
    } <- #226-253 private ServerPlayer resolveGame(List<ServerPlayer> players)
        //pd438 4/19/2024
```

ResolveGame

Checklist Items (0)

```
    //pd438 4/19/2024
private String getWinnerChoice(String choice1, String choice2) {
    if (choice1 == null || choice2 == null) {
        // Handle null choices here, return a default winner or null
        return null;
    }

    if (choice1.equalsIgnoreCase(anotherString:"rock") && choice2.equalsIgnoreCase(anotherString:"scissors")) {
        return "rock";
    } else if (choice1.equalsIgnoreCase(anotherString:"scissors") && choice2.equalsIgnoreCase(anotherString:"paper")) {
        return "scissors";
    } else if (choice1.equalsIgnoreCase(anotherString:"paper") && choice2.equalsIgnoreCase(anotherString:"rock")) {
        return "paper";
    } else {
        // If not explicitly defined, second choice wins
        return choice2;
    }
} <- #255-271 private String getWinnerChoice(String choice1, String choice2)

private void resetTurns() {
    players.values().forEach(p -> p.setTakenTurn(takenTurn:false));
    sendResetLocalTurns();
}

private void end() {
    System.out.println(TextFX.colorize(text:"Doing game over", Color.YELLOW));
    turnOrder.clear();
    players.clear();
    changePhase(Phase.READY);
    sendMessage(ServerConstants.FROM_ROOM, message:"You Win!!! Game over! Start a new game by setting up players and issuing ready checks.");
} <- #278-284 private void end()
```

ResetTurns and winnerchoice

Checklist Items (0)

```
    private void sendCurrentPlayerTurn() {
        Iterator<ServerPlayer> iter = players.values().iterator();
        while (iter.hasNext()) {
            ServerPlayer sp = iter.next();
            sp.sendCurrentPlayerTurn(currentPlayer == null ? Constants.DEFAULT_CLIENT_ID : currentPlayer.getClientId());
        }
    } <- #286-292 private void sendCurrentPlayerTurn()

    private void sendResetLocalTurns() {
        players.values().forEach(ServerPlayer::sendResetLocalTurns);
    }
```

```
    private void syncUserTookTurn(ServerPlayer isp) {
        players.values().forEach(sp -> sp.sendPlayerTurnStatus(isp.getClientId(), isp.didTakeTurn()));
    }

    private void changePhase(Phase incomingChange) {
        if (currentPhase != incomingChange) {
            currentPhase = incomingChange;
            syncCurrentPhase();
        }
    } <- #302-307 private void changePhase(Phase incomingChange)

    private void syncCurrentPhase() {
        players.values().forEach(t -> t.sendPhase(currentPhase));
    }

    private void syncReadyState(ServerPlayer csp) {
        players.values().forEach(sp -> sp.sendReadyState(csp.getClientId(), csp.isReady()));
    }
}- #16-316 public class GameRoom extends Room
```

Current player turn and Current Phase.

Checklist Items (9)

#1 If using turns, show the code that handles the correct person, otherwise show how a choice is recorded (out/skipped players can't take an action)

#2 The code that handles a player skipping (they shouldn't be included in this round and will not be removed from next round)

#3 The code to check if all applicable players made a choice or skipped

#4 The code that processes the player "battles" in round-robin style (i.e., p1 -> p2 -> p3 -> p0)

#5 The code that checks if there's a winner (1 remaining), a tie (none remaining), or another round (2+ remaining)

#6 The code that resets the state for a new round (2+ remaining)

#7 The code that resets the state for a new session (win/tie)

#8 Code screenshots should include ucid and date comment

#9 Each screenshot should be clearly captioned

COLLAPSE

Task #2 - Points: 1

Text: Explain the logic/implementation of the following

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	How are the player choices handled (valid choice, skip, invalid choice)
#2	1	How does the code work for determining the end of a round before the "battle" checking

#3	1	How does the "battle" logic work, do players lose as attacker, defender, or in either scenario
#4	1	How are skips handled with respect to "battle" logic
#5	1	How is the new round logic interpreted/handled
#6	1	How is the end-of-session logic interpreted/handled

Response:

User has the ability to choose either /Paper,/Rock,/Scissor

User can lose in either scenario whether they are on defense or offense. The code gathers the users entry and compares them to decide who has won and who has lost. Once one player gets eliminated, the other two continue until they find the one who is the winner. the round logic is shown based on who gets out. then the user that lost gets disconnected from the room. but in the case of a tie. the server will spit out a message to all clients and the games will continue.

COLLAPSE

Task #3 - Points: 1

Text: Screenshots of the client-side code for the following

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	The state of the individual players (out, skipped, or active)
#2	1	Code screenshots should include ucid and date comment
#3	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large

```
            //pd438 4/19/2024  To Display client side code of the current turn and elimination
        case CURRENT_TURN:
            /*
             * if (clientsInRoom.containsKey(p.getClientId())) {
             * clientsInRoom.get(p.getClientId()).setMyTurn(true);
             * }
             */
            clientsInRoom.values().stream().forEach(c -> {
                boolean isMyTurn = c.getClientId() == p.getClientId();
                c.setMyTurn(isMyTurn);
                if (isMyTurn) {
                    System.out.println(
                            TextFX.colorize(String.format(format:"It's %s's turn", c.getClientName()), Color.PURPLE));
                }
            }); <- #524-531 clientsInRoom.values().stream().forEach
            break;
        case ELIMINATION:
        try {
            EliminationPayload ep = (EliminationPayload) p;
            boolean isEliminated;
            isEliminated = ep.isEliminated();

            if(isEliminated == true){System.out.println(TextFX.colorize(text:"You are Out!", Color.RED));}
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }
        // case END_SESSION: //clearing all local player data
        default:
```

this displays when user gets eliminated from the game.

Checklist Items (3)

#1 The state of the individual players (out, skipped, or active)

#2 Code screenshots should include ucid and date comment

#3 Each screenshot should be clearly captioned

^COLLAPSE^

Task #4 - Points: 1

Text: Screenshots of client-side output

Checklist

*The checkboxes are for your own tracking

#	Points	Details
☑ #1	1	Player being eliminated
☐ #2	1	Player being skipped or choosing to skip
☑ #3	1	Results of the battles for a round
☑ #4	1	Feedback on winner, tie, or new round
☑ #5	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large

```
INFO: *Casca disconnected*
Apr 19, 2024 2:56:08 PM Project.Client.Cli
ent$2 run
INFO: Debug Info: Type[MESSAGE], Message[C
asca is eliminated!], ClientId[-1]
[Room]: Casca is eliminated!
Apr 19, 2024 2:56:08 PM Project.Client.Cli
ent$2 run
INFO: Debug Info: Type[MESSAGE], Message[Y
ou Win!!! Game over! Start a new game by s
etting up players and issuing ready checks
```

```
.], ClientId[-1]
[Room]: You Win!!! Game over! Start a new
game by setting up players and issuing rea
dy checks.
```

This displays when the game is over and the message is displayed.

Checklist Items (4)

#1 Player being eliminated

#3 Results of the battles for a round

#4 Feedback on winner, tie, or new round

#5 Each screenshot should be clearly captioned

Misc (1 pt.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Add the pull request link for the branch

Details:
Note: the link should end with /pull/#

URL #1
https://github.com/PD438/PD438_IT114_002/pull/13

COLLAPSE

Task #2 - Points: 1

Text: Talk about any issues or learnings during this assignment

Response:

I had alot of the issues coming from the endgame logic. Another thing i had an issue with was to be able for the user to display messages. I needed alot of outside help in order to complete this.

COLLAPSE

Task #3 - Points: 1

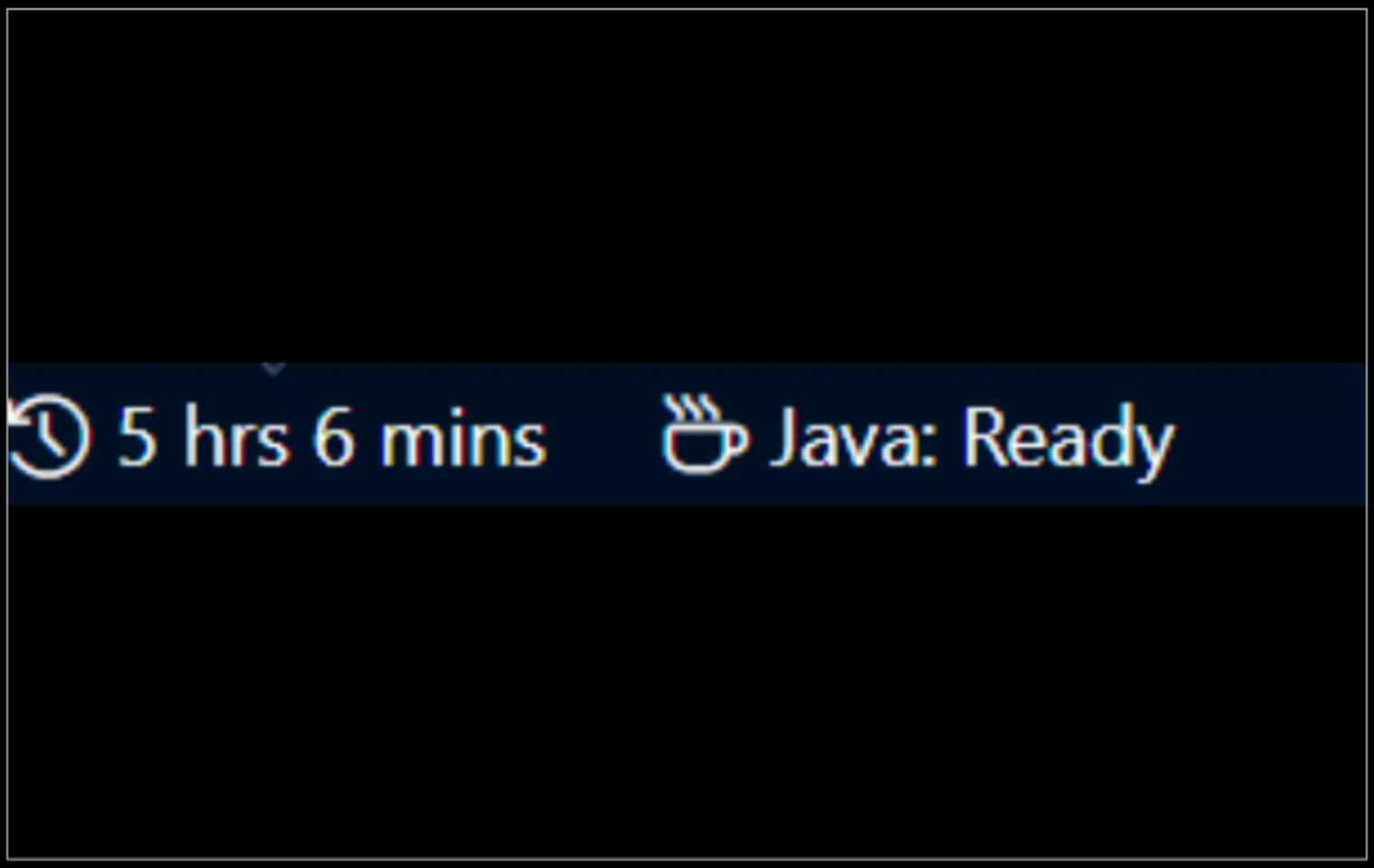
Text: WakaTime Screenshot

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: Large View

Small Medium Large



WakaTime Screenshot From Visual Studio Code

End of Assignment