Submission Worksheet

Submission Data

Course: IT114-450-M2025
Assignment: IT114 - Milestone 3 - RPS
Student: Daniel C. (dvc2)
Status: In Progress | **Worksheet Progress:** 100%
Potential Grade: 10.00/10.00 (100.00%)
Received Grade: 0.00/10.00 (0.00%)
Started: 8/7/2025 11:47:46 PM
Updated: 8/8/2025 4:42:30 AM
Grading Link: https://learn.ethereallab.app/assignment/v3/IT114-450-M2025/it114-milestone-3-rps/grading/dvc2
View Link: https://learn.ethereallab.app/assignment/v3/IT114-450-M2025/it114-milestone-3-rps/view/dvc2

Instructions

1. Refer to Milestone3 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 `Milestone3` branch
    1. `git checkout Milestone3`
    2. `git pull origin Milestone3`
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 Milestone3`
    4. On Github merge the pull request from `Milestone3` 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.) Core Ui

Progress: 100%

Task #1 (0.50 pts.) - Connection/Details Panels

Progress: 100%

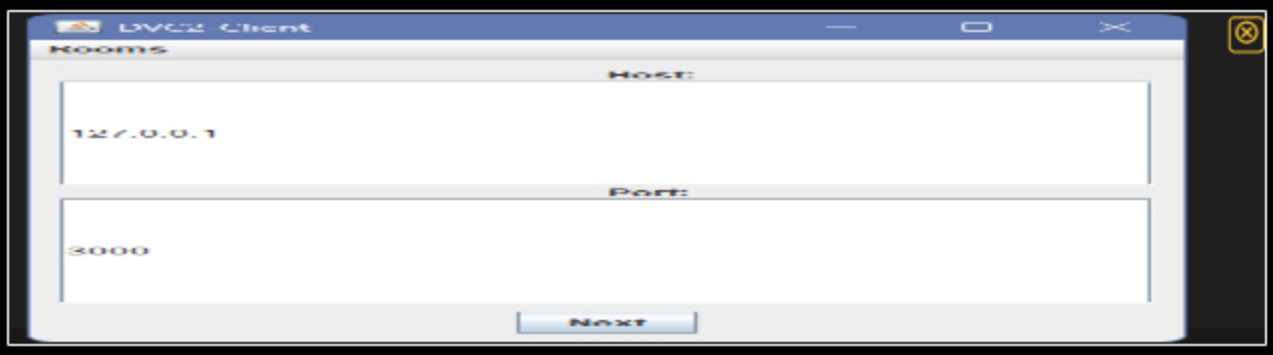
Part 1:

Progress: 100%

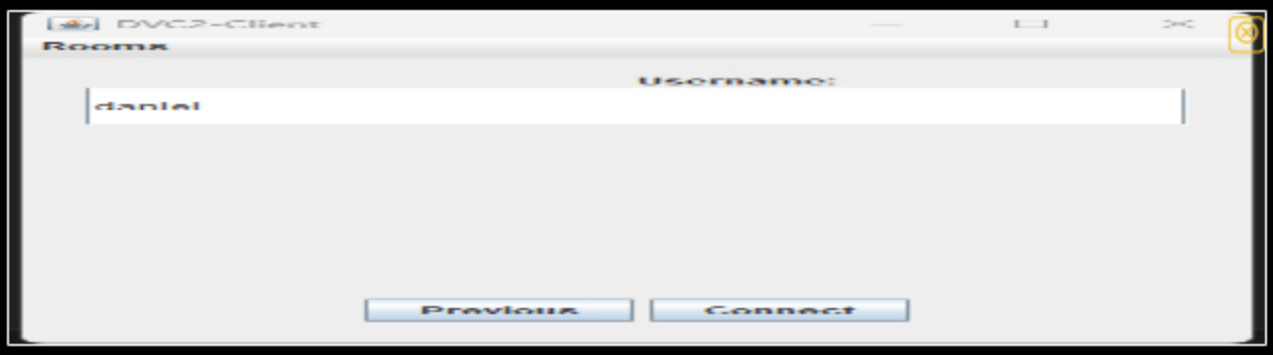
Details:

- Show the connection panel with valid data

- Show the user details panel with valid data



connection panel



user details panel

Saved: 8/8/2025 1:11:06 AM

Part 2:

Progress: 100%

Details:

- Briefly explain the code flow from recording/capturing these details and passing them through the connection process

Your Response:

The application first gathers the server's host and port from a connection panel, followed by the user's name from a details panel. Once all details are provided, it initiates a connection to the server and sends the username to complete the handshake.

Saved: 8/8/2025 1:11:06 AM

Task #2 (0.50 pts.) - Ready Panel

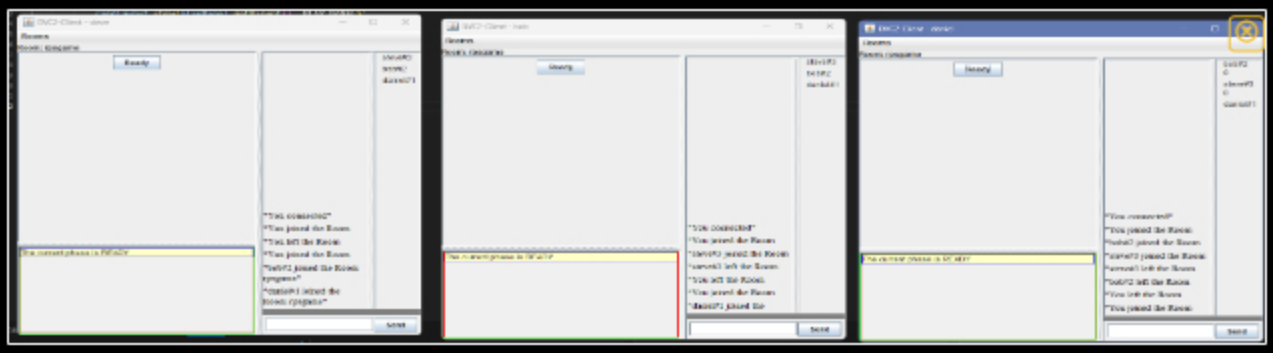
Progress: 100%

Part 1:

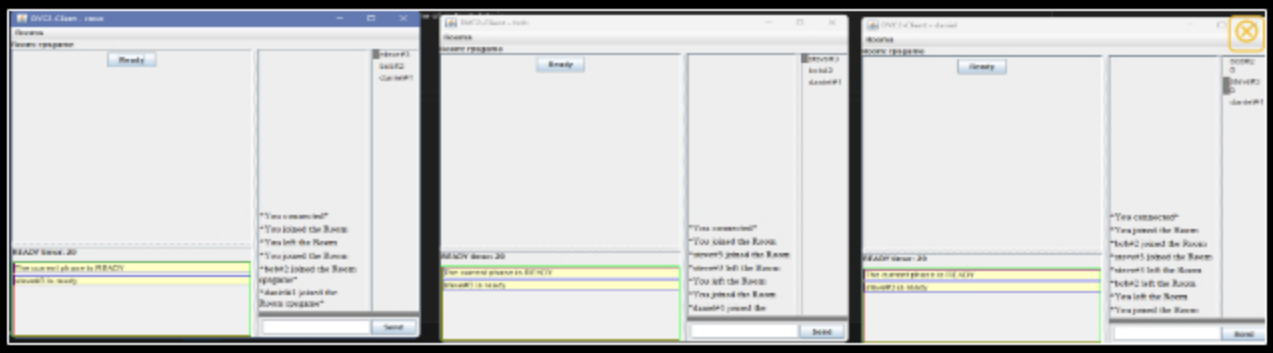
Progress: 100%

Details:

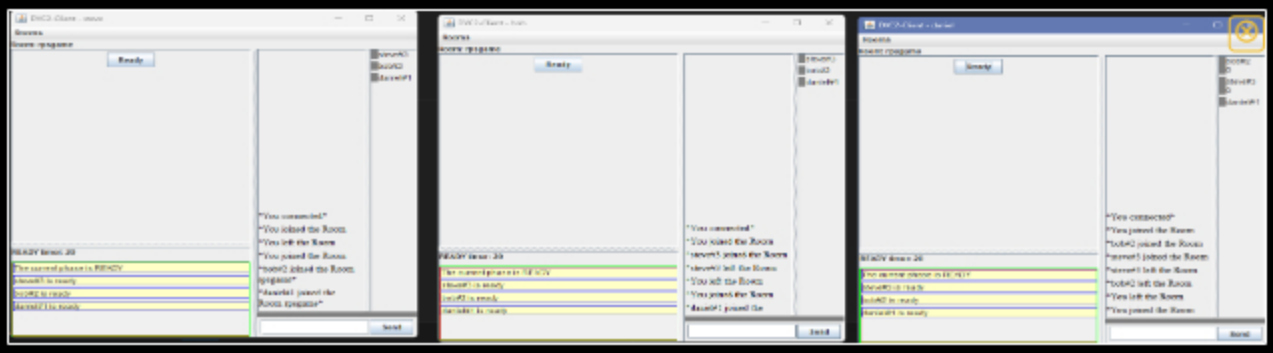
- Show the button used to mark ready
- Show a few variations of indicators of clients being ready (3+ clients)



ready screen for all three users



ready screen when only 1 user is ready



ready screen after all 3 users have marked themselves as ready

Saved: 8/8/2025 2:13:29 AM

Part 2:

Progress: 100%

Details:

- Briefly explain the code flow for marking READY from the UI
- Briefly explain the code flow from receiving READY data and updating the UI

Your Response:

When a user clicks the "Ready" button, the client sends a READY payload to the server, which then broadcasts this status change to all other clients in the room. Upon receiving this READY data, each client's UI updates to visually indicate which players are ready (Box next to the client's which have marked themselves as ready).

Saved: 8/8/2025 2:13:29 AM

Section #2: (2 pts.) Project Ui

Progress: 100%

Task #1 (0.67 pts.) - User List Panel

Progress: 100%

Details:

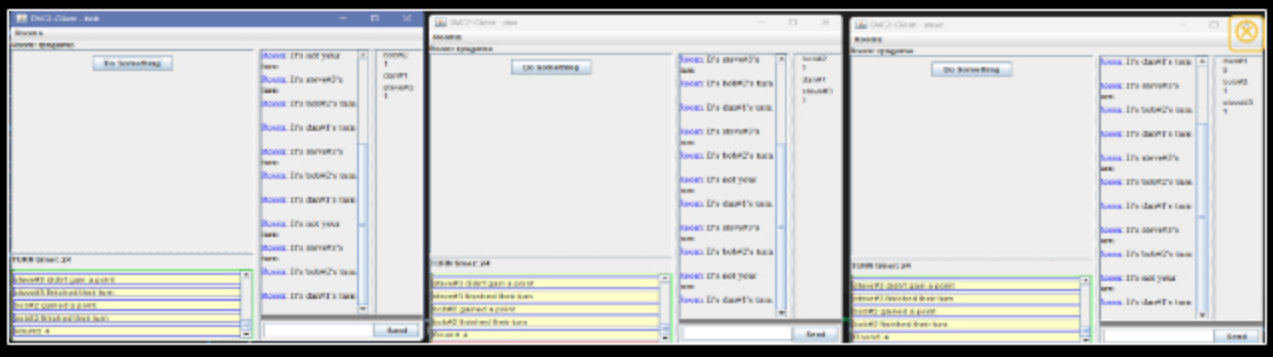
- Show the username and id of each user
- Show the current points of each user
- Users should appear in score order, sub-sort by name when ties occur
- Pending-to-pick users should be marked accordingly
- Eliminated users should be marked accordingly

Part 1:

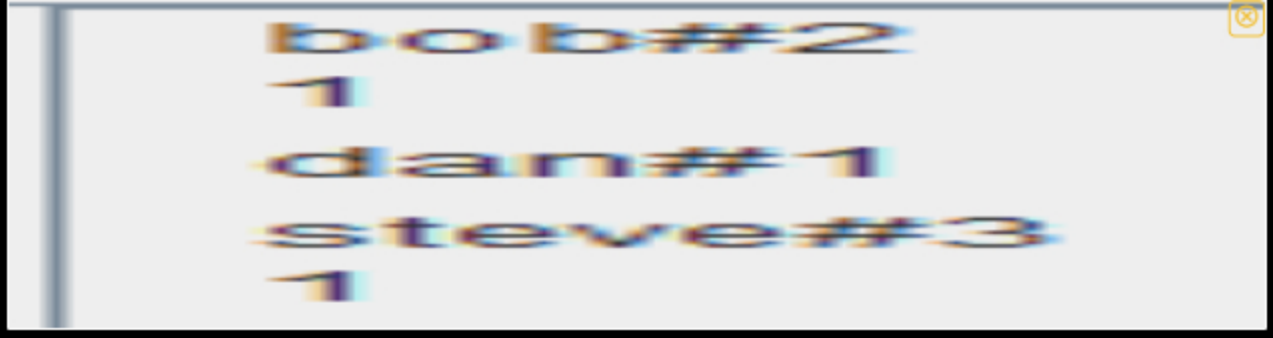
Progress: 100%

Details:

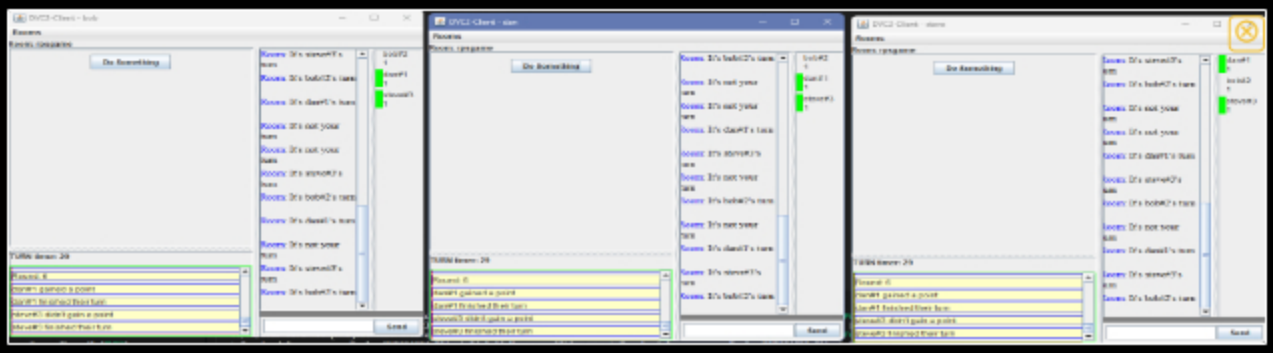
- Show various examples of points (3+ clients visible)
  - Include code snippets showing the code flow for this from server-side to UI
- Show that the sorting is maintained across clients
  - Include code snippets showing the code that handles this
- Show various examples of the pending-to-pick indicators
  - Include code snippets showing the code flow for this from server-side to UI
- Show various examples of elimination indicators
  - Include code snippets showing the code flow for this from server-side to UI



panel with 3 clients and points showcased



user details panel with 3 clients



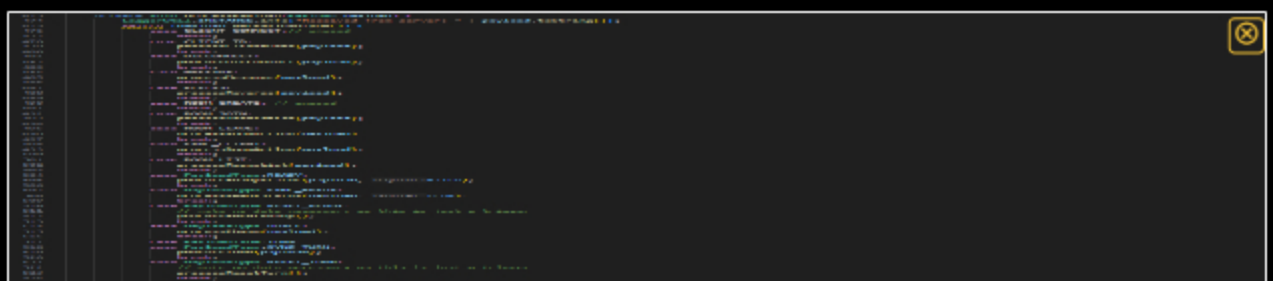
pending to pick indicators with clients

```
@Override
protected void onRoundEnd() {
    LoggerUtil.INSTANCE.info(message:"onRoundEnd() start");
    cancelRoundTimer();
    processRound();

    long activePlayers = clientsInRoom.values().stream().filter(p -> !p.isEliminated() && !p.isSpectator() && !p.isAway()).count();
    if (activePlayers <= 1) {
        onSessionEnd();
    } else {
        onRoundStart();
    }
    LoggerUtil.INSTANCE.info(message:"onRoundEnd() end");
}

private void processRound() {
    List<ServerThread> activePlayers = clientsInRoom.values().stream()
            .filter(p -> p.isReady() && !p.isAway() && !p.isSpectator() && !p.isEliminated())
            .collect(Collectors.toList());

    // Eliminate players who didn't make a choice dvc2 8/4/2025
    activePlayers.stream()
        .filter(p -> !playerChoices.containsKey(p.getClientId()))
        .forEach(p -> {
            p.setEliminated(eliminated:true);
            sendEliminationStatus(p.getClientId(), isEliminated:true);
            relay(sender:null, p.getDisplayName() + " was eliminated for not making a choice.");
        });
```

processRound method sends player points to the various clients, changing after the battle results have been obtained.

```
46 {
47     public boolean sendPlayerPoints(long clientId, int points) {
48         PointsPayload rp = new PointsPayload();
49         rp.setPoints(points);
50         rp.setClientId(clientId);
51         return sendToClient(rp);
52     }
53 // dvc2 8/4/2025
```

This method creates PointsPayload and sends it to the client



```
    @Override
    public void onPointsUpdate(long clientId, int points) {
        if (userItemMap.containsKey(clientId)) {
            SwingUtilities.invokeLater(() -> {
                try {
                    userItemMap.get(clientId).setPoints(points);
                    sortUserList();
                } catch (Exception e) {
                    LoggerUtil.INSTANCE.severe(message:"Error setting user item", e);
                }
            });
        } else if (clientId == Constants.DEFAULT_CLIENT_ID) {
            SwingUtilities.invokeLater(() -> {
                try {
                    userItemMap.values().forEach(u -> u.setPoints( 1));
                    sortUserList();
                } catch (Exception e) {
                    LoggerUtil.INSTANCE.severe(message:"Error resetting user items", e);
                }
            });
        }
    }
```

Receives the PointsPayload and updates the user data

```
    public void setTurn(boolean didTakeTurn){
        setTurn(didTakeTurn, Color.GREEN);
    }

    public void setTurn(boolean didTaketurn, Color trueColor) {
        turnIndicator.setBackground(didTaketurn ? trueColor : new Color(r:0, g:0, b:0, a:0));
        if (!didTaketurn && !isEliminated && !isSpectator && !isAway) {
            turnIndicator.setBackground(Color.YELLOW); // Pending pick
        }
        repaint();
    }
```

sets the users points within the leaderboard

```
    public boolean sendEliminationStatus(long clientId, boolean eliminated) {
        Payload payload = new Payload();
        payload.setPayloadType(PayloadType.ELIMINATION_STATUS);
        payload.setClientId(clientId);
        payload.setMessage(String.valueOf(eliminated));
        return sendToClient(payload);
    }
```

setTurn indicated is responsible for the clients' visual indicators

sendEliminationStatus shows the clients who has been eliminated

```
    public void setEliminated(boolean eliminated) {
        isEliminated = eliminated;
        updateAppearance();
    }

    private void updateAppearance() {
        if (isSpectator) {
            textContainer.setText(getClientName().split(regex:" ")[0] + " (Spectator)");
            textContainer.setForeground(Color.GRAY);
        } else if (isEliminated) {
            textContainer.setText(getClientName().split(regex:" ")[0] + " (Eliminated)");
            textContainer.setForeground(Color.RED);
        } else if (isAway) {
            textContainer.setForeground(Color.LIGHT_GRAY);
        } else {
            textContainer.setForeground(Color.BLACK);
        }
        repaint();
    }
}
```

this sets the clients as eliminated and updates the visual of the elim for the users.

Saved: 8/8/2025 2:31:44 AM

Part 2:

Progress: 100%

Details:

- Briefly explain the code flow for points updates from server-side to the UI
- Briefly explain the code flow for user list sorting
- Briefly explain the code flow for server-side to UI of pending-to-pick indicators
- Briefly explain the code flow for server-side to UI of elimination indicators

Your Response:

The server awards points and sends an update to all clients. The UI receives this update, changes the player's score, and then automatically re-sorts the user list with the highest scores at the top. At the start of each round, the server resets every player's turn status to "not taken." The UI then shows a yellow indicator next to the name of any active player who has not yet made their move for the round. When a player is eliminated, the server broadcasts this status change to all clients. The UI then updates that player's entry in the user list, changing their name to red and adding "(Eliminated)"to show they are out of the game.

Saved: 8/8/2025 2:31:44 AM

Task #2 (0.67 pts.) - Game Events Panel

Progress: 100%

Details:

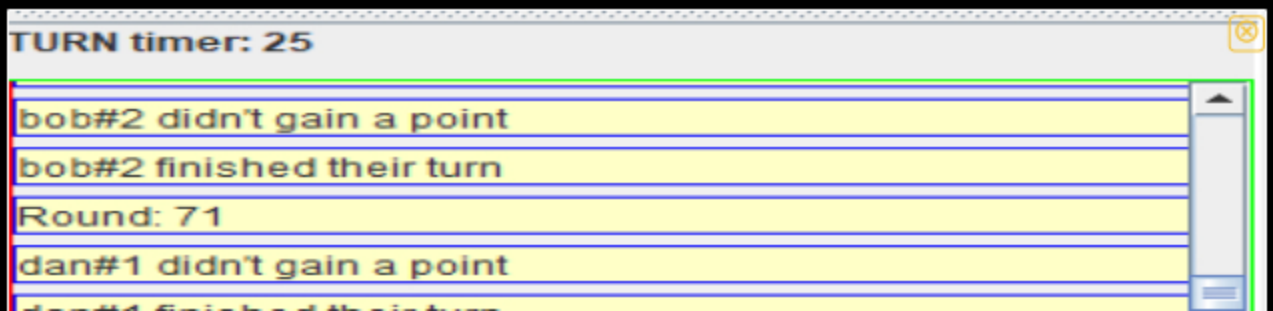
- Show the status of users picking choices
- Show the battle resolution messages from Milestone 2
  - Include messages about elimination
- Show the countdown timer for the round

Part 1:

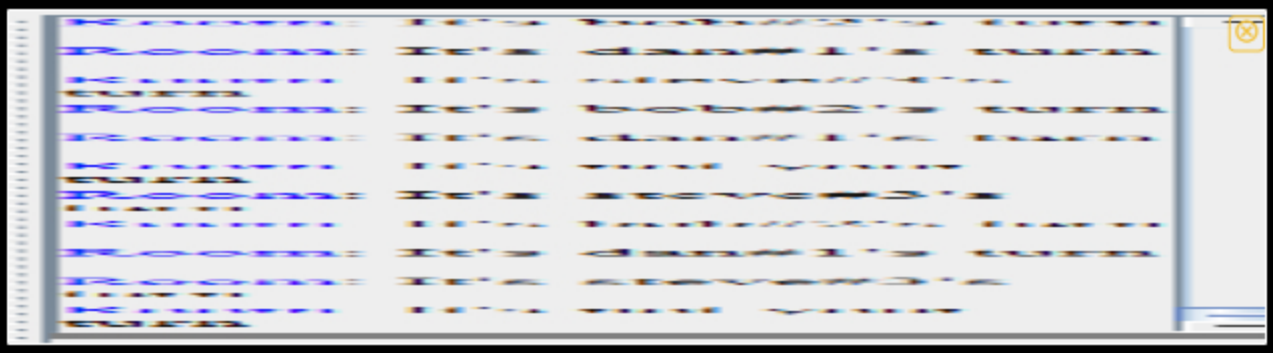
Progress: 100%

Details:

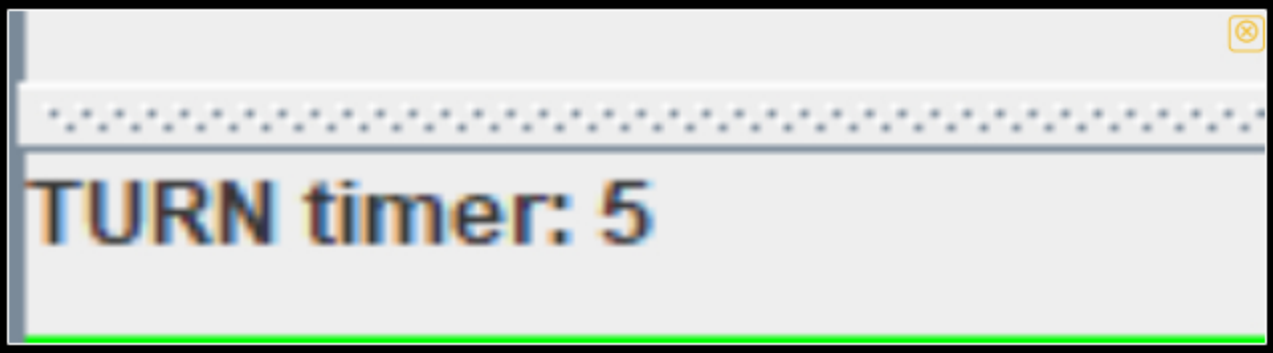
- Show various examples of each of the messages/visuals
- Show code snippets related to these messages from server-side to UI



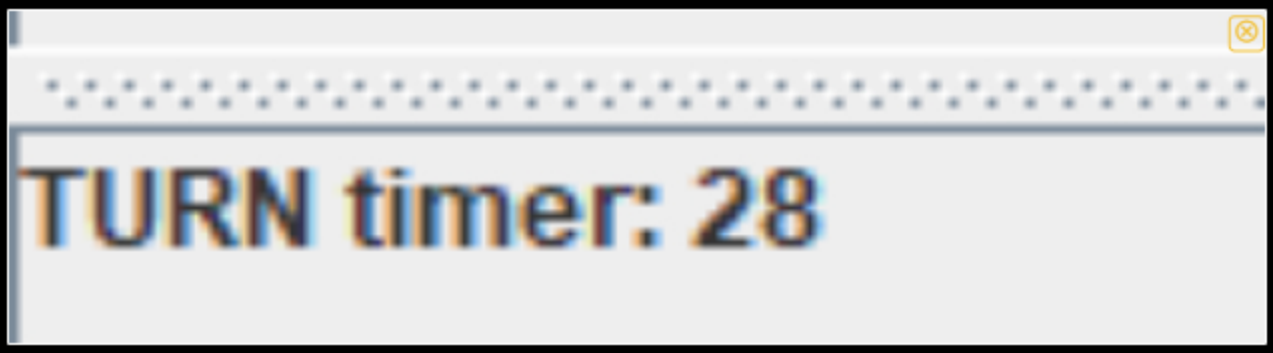
battle resolution messages in game chat



Users picking choices getting recorded

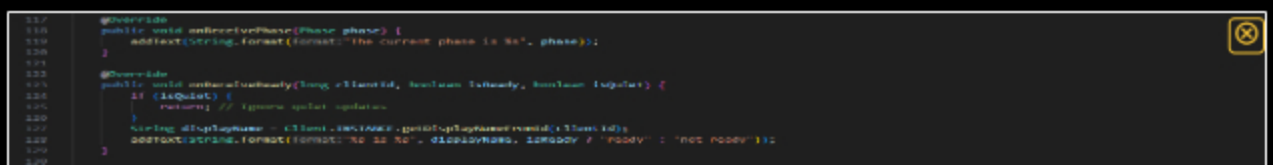


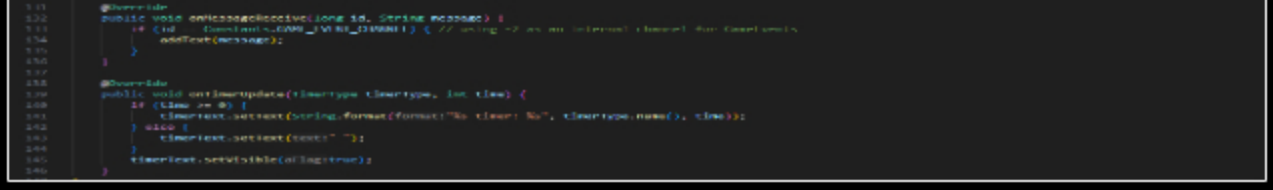
turn time ticking down to 0 then resetting



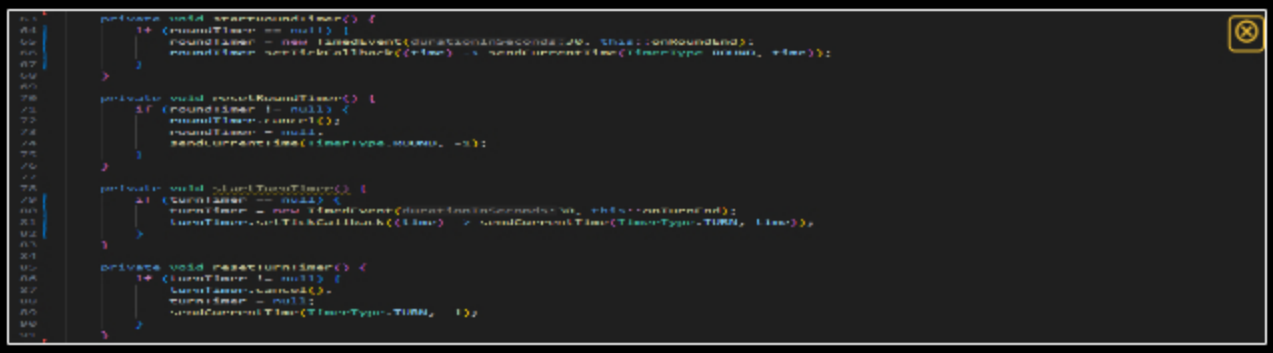
turn timer initially starts at 30

handles clients when they are away, spectating, and their choices.





GameEventsPanel.java handles displaying this code on screen.



Code that resets and starts the round and turn timer.

Saved: 8/8/2025 3:58:58 AM

Part 2:

Progress: 100%

Details:

- Briefly explain the code flow for generating these messages and getting them onto the UI

Your Response:

All game-related messages are generated on the server within the GameRoom.java file and then broadcast to all connected clients. The clients receive these messages and display them in the GameEventsPanel.java, which is a dedicated area in the UI for game-specific information.

Saved: 8/8/2025 3:58:58 AM

Task #3 (0.67 pts.) - Game Area

Progress: 100%

Details:

- UI should have components to allow the user to select their choice

Part 1:

Progress: 100%

Details:

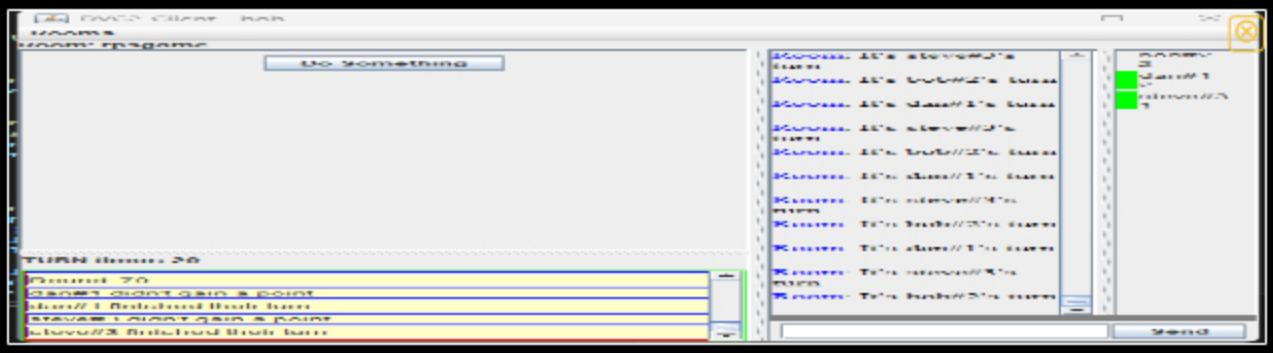
- Show various examples of selections across clients (3+ clients visible)
- Show the code related to sending choices upon selection
- Show the code related to showing visually what was selected

```
public GamePanel(TurnControl turnCtrl) {
    super(new BorderLayout());

    // Create the buttons for Rock, Paper, Scissors
    JButton rockButton = new JButton("Rock");
    rockButton.addActionListener(event -> sendChoice("rock"));

    JButton paperButton = new JButton("Paper");
    paperButton.addActionListener(event -> sendChoice("paper"));

    JButton scissorsButton = new JButton("Scissors");
    scissorsButton.addActionListener(event -> sendChoice("scissors"));

    JButton lizardButton = new JButton("Lizard");
    lizardButton.addActionListener(event -> sendChoice("lizard"));

    JButton spockButton = new JButton("Spock");
    spockButton.addActionListener(event -> sendChoice("spock"));

    buttonPanel.add(rockButton);
    buttonPanel.add(paperButton);
    buttonPanel.add(scissorsButton);
    buttonPanel.add(lizardButton);
    buttonPanel.add(spockButton);
```

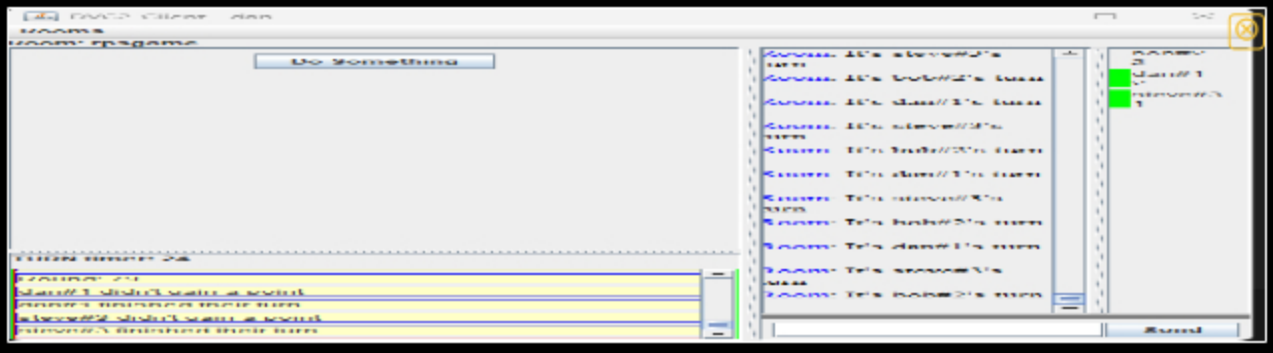
snippet of GamePanel.java handling choices upon selection

```
public String getClientName() {
    return textContainer.getText();
}

public int getPoints() {
    return points;
}

public void setTurn(boolean didTakeTurn){
    setTurn(didTakeTurn, Color.GREEN);
}

public void setTurn(boolean didTakeTurn, Color trueColor) {
    turnIndicator.setBackground(didTakeTurn ? trueColor : new Color(r:0, g:0, b:0, a:0));
    if (!didTakeTurn && !isEliminated && !isSpectator && !isAway) {
        turnIndicator.setBackground(Color.YELLOW); // Pending pick
    }
    repaint();
}
```

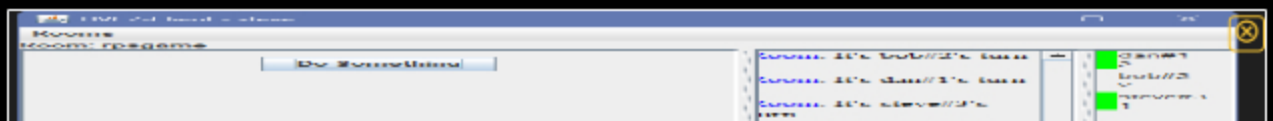
UserListItem.java is responsible for showing visually what was selected

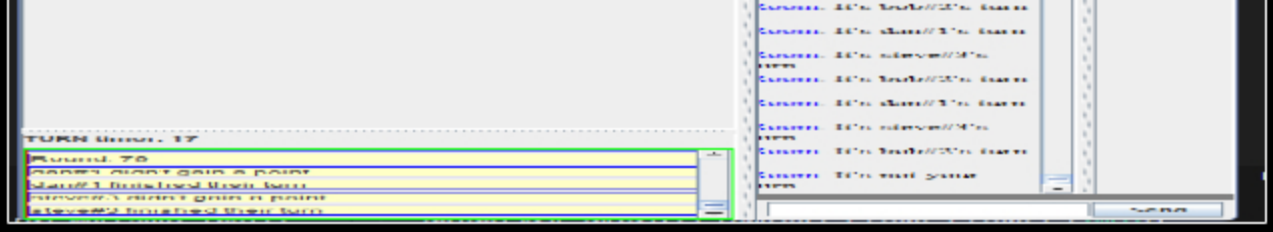


client 1



client 2





client 3

Saved: 8/8/2025 4:04:48 AM

Part 2:

Progress: 100%

Details:

- Briefly explain the code flow for selecting a choice and having it reach the server-side
- Briefly explain the code flow for receiving the selection for the current player to update the UI

Your Response:

When a player clicks one of the choice buttons in the GamePanel, it triggers the sendChoice method. This method creates a PICK_CHOICE payload containing the selected move and sends it to the server. The server's GameRoom then receives this payload in its handlePick method, records the choice, and informs all other players that a choice has been made. After the server processes a player's choice, it broadcasts a TURN payload to all clients. The Client class receives this payload and updates the local User object to mark that the player has taken their turn. This triggers the onTookTurn event in the UserListPanel, which then calls the setTurn method in the corresponding UserListItem to change the color of the indicator next to the player's name, visually confirming their action.

Saved: 8/8/2025 4:04:48 AM

Section #3: (4 pts.) Project Extra Features

Progress: 100%

Task #1 (2 pts.) - Extra Choices

Progress: 100%

Details:

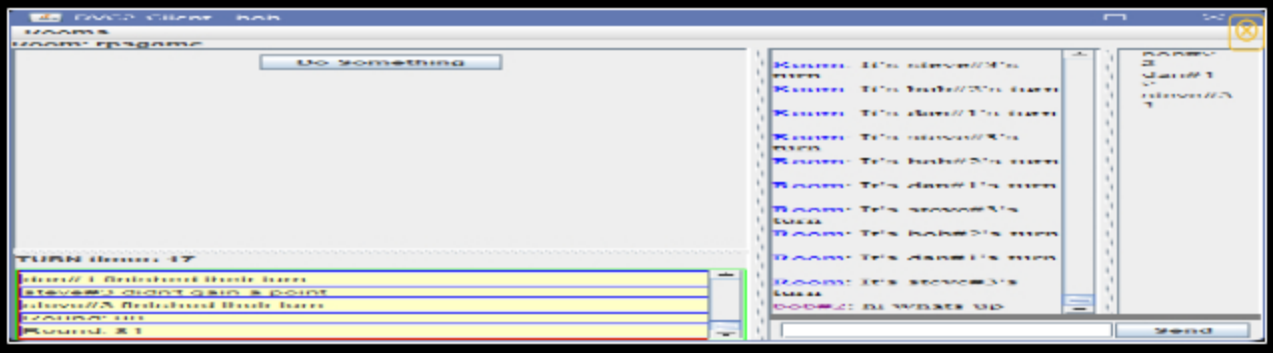
- Setting should be toggleable during Ready Check by session creator
  - (Option 1) Extra choices are available during the full session
  - (Option 2) Only activate extra options at different stages (i.e., last 3 players remaining)
- There should be at least 2 extra options for rps-5

Part 1:

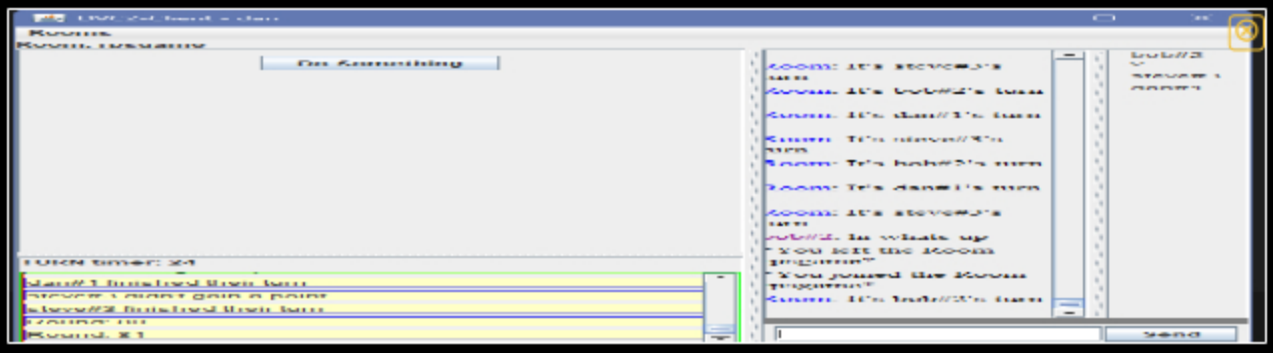
Progress: 100%

Details:

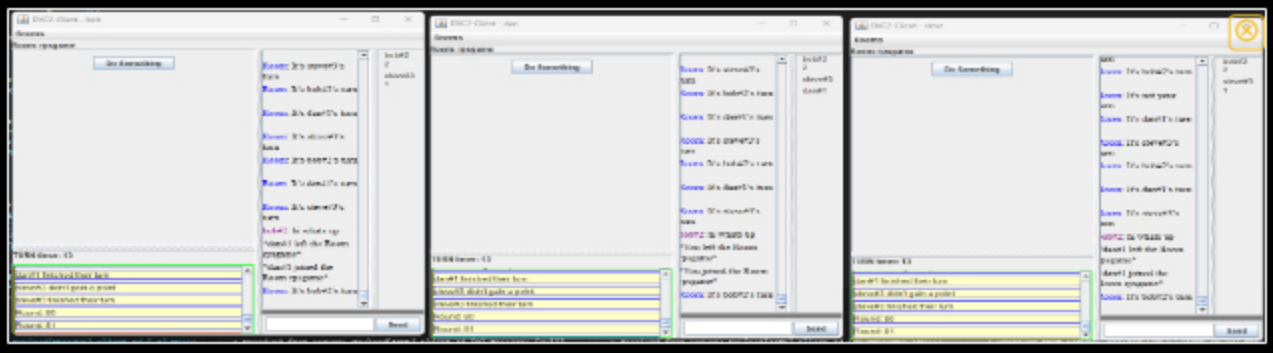
- Show the Ready Check screen with the option for the host (3+ clients must be visible)
  - Show the related code that makes this interactable only for the host
- Show the play screen with the extra options available
  - Show the related code for the UI and handling of these extra options (including battle logic)



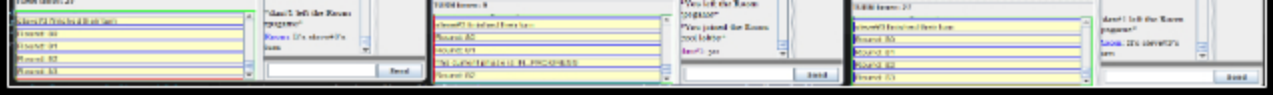
client can interact in chat with opponents while game continues.



Client can leave and rejoin the game, to reset their score.



All 3 clients shown at once.



client can create new room from rps lobby.

```
public class ReadyPanel extends JPanel {

    public ReadyPanel() {
        JCheckBox extraOptions = new JCheckBox(text:"Extra RPS Options");
        JCheckBox cooldowns = new JCheckBox(text:"Cooldowns");

        JButton readyButton = new JButton(text:"Ready");
        readyButton.addActionListener(event -> {
            try {
                // Example of sending settings with the ready signal
                String settings = String.format(format:"/ready %b %b", extraOptions.isSelected(), cooldowns.isSelected());
                Client.INSTANCE.sendMessage(settings);
            } catch (IOException e1) {
                e1.printStackTrace();
            }
        });
```

Extra rps options extension button in my ReadyPanel.java

```
        JButton lizardButton = new JButton(text:"Lizard");
        lizardButton.addActionListener(event -> sendChoice(choice:"lizard"));

        JButton spockButton = new JButton(text:"Spock");
        spockButton.addActionListener(event -> sendChoice(choice:"spock"));

        buttonPanel.add(rockButton);
        buttonPanel.add(paperButton);
        buttonPanel.add(scissorsButton);
        buttonPanel.add(lizardButton);
        buttonPanel.add(spockButton);
```

Lizard and Spock buttons in Game panel to show up for the client as an extra add on to the rps game.

```
            if (choice1 == null || choice2 == null) continue;

            int result = getWinner(choice1, choice2);
            String battleLog = String.format(format:"%s (%s) vs %s (%s) -> ", p1.getDisplayName(), choice1, p2.getDisplayName(), choice2);

            if (result == 1) {
                p1.changePoints(points:1);
                p2.setEliminated(eliminated:true);
                battleLog += p1.getDisplayName() + " wins!";
            } else if (result == -1) {
                p2.changePoints(points:1);
                p1.setEliminated(eliminated:true);
                battleLog += p2.getDisplayName() + " wins!";
            } else {
                battleLog += "It's a tie!";
            }
```

the getWinner method contains the code for the extra buttons.

Saved: 8/8/2025 4:14:08 AM

Part 2:

Progress: 100%

Details:

- Briefly explain the code for the host's option to toggle this feature
- Briefly explain the code related to handling these options including how it's handled during the battle logic
- Note which option you went with in terms of activating the choices

Your Response:

The host toggles features like "Extra RPS Options" using checkboxes in the ReadyPanel, and these settings are sent to the GameRoom on the server when the host readies up. The battle logic in the getWinner method uses a conditional check to switch to an expanded set of rules that include the extra choices, like "Lizard" and "Spock," if the feature is enabled. I implemented Option 1, so when the host activates the extra choices, they are available for the entire game session.

Saved: 8/8/2025 4:14:08 AM

Task #2 (2 pts.) - Choice cooldown

Progress: 100%

Details:

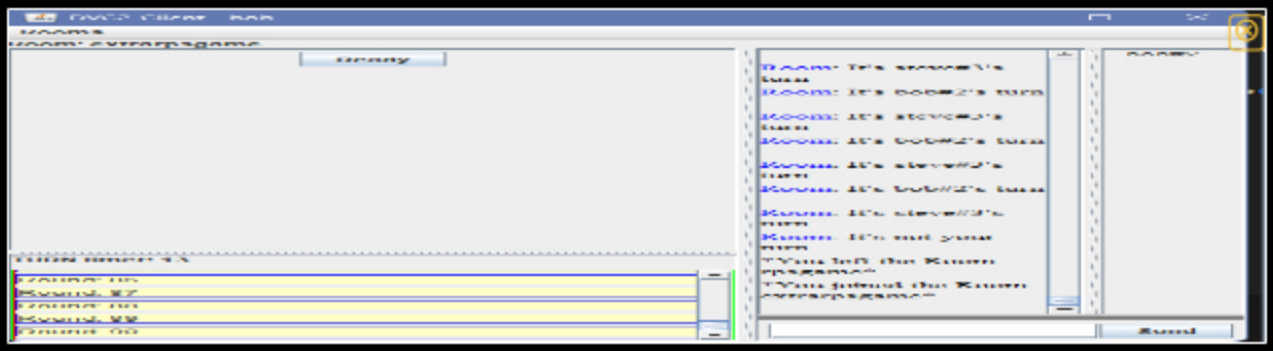
- Setting should be toggleable during Ready Check by session creator
- The choice on cooldown must be disable on the UI for the User

Part 1:

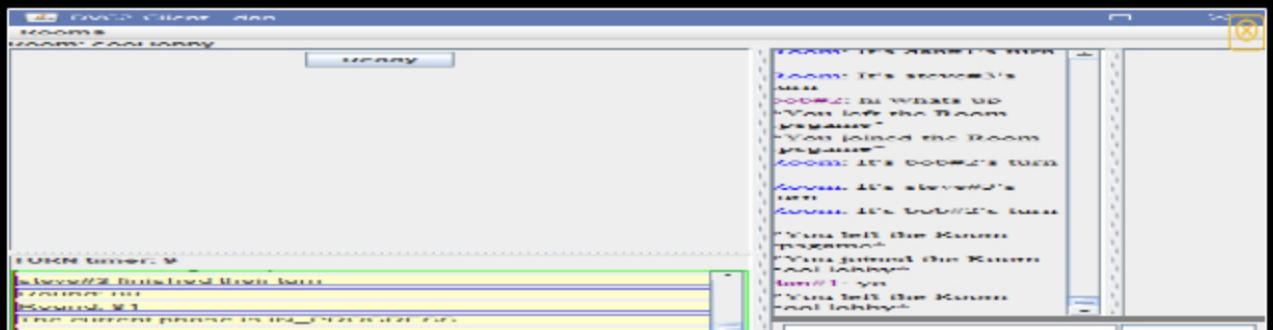
Progress: 100%

Details:

- Show the Ready Check screen with the option for the host (3+ clients must be visible)
  - Show the related code that makes this interactable only for the host
- Show a few examples of the play screen with the choice on cooldown
  - Show the related code for the UI and handling of the cooldown and server-side enforcing it



extrarpsgame lobby with another client connected.



player has to wait to ready up while the host selects options.

```
You, 4 hours ago | 1 author (You)
9   public class ReadyPanel extends JPanel {
10
11      public ReadyPanel() {
12          JCheckBox extraOptions = new JCheckBox(text:"Extra RPS Options");
13          JCheckBox cooldowns = new JCheckBox(text:"Cooldowns");
14
```

extraoptions and cooldowns are shown in the readypanel.java

```
343             var1.setTookTurn(true);
344             this.sendTurnStatus(var1, var1.didTakeTurn());
345             this.onTurnEnd();
346         } catch (NotPlayersTurnException var4) {
347             var1.sendMessage(-1L, "It's not your turn");
348             LoggerUtil.INSTANCE.severe("handleTurnAction exception", var4);
349         } catch (NotReadyException var5) {
350             LoggerUtil.INSTANCE.severe("handleTurnAction exception", var5);
351         } catch (PlayerNotFoundException var6) {
352             var1.sendMessage(-1L, "You must be in a GameRoom to do the ready check");
353             LoggerUtil.INSTANCE.severe("handleTurnAction exception", var6);
354         } catch (PhaseMismatchException var7) {
355             var1.sendMessage(-1L, "You can only take a turn during the IN_PROGRESS phase");
356             LoggerUtil.INSTANCE.severe("handleTurnAction exception", var7);
357         } catch (Exception var8) {
358             LoggerUtil.INSTANCE.severe("handleTurnAction exception", var8);
```

handlePick in the UserDetailsPanel.java contains the ui for the cooldown server-side enforcement.

Saved: 8/8/2025 4:20:18 AM

Part 2:

Progress: 100%

Details:

- Briefly explain the code for the host's option to toggle this feature
- Briefly explain the code related to handling and enforcing the cooldown period (include how this is recorded per user and reset when applicable)

Your Response:

The host's ability to toggle game features is handled through the UI and server-side logic working together. In the ReadyPanel.java file, there are JCheckBox components for "Extra RPS Options" and "Cooldowns." When the "Ready" button is clicked, the client sends a command to the server that includes whether these boxes are checked. On the server, the GameRoom.java file's handleReady method receives this command and, if the sender is the host, updates the game settings for the entire session. The cooldown is handled on both the client and server sides. On the server, the GameRoom.java file's handlePick method checks if a player's choice is the same as their last choice. If the cooldown feature is enabled, it rejects the choice and sends an error message. On the client, the GamePanel.java file would be responsible for disabling the button for the last-used choice, preventing the user from clicking it again until the next round. The last choice for each player is recorded in a map in the GameRoom and is reset at the start of each new round.

Saved: 8/8/2025 4:20:18 AM

Section #4: (2 pts.) Project General Requirements

Progress: 100%

Task #1 (1 pt.) - Away Status

Progress: 100%

Details:

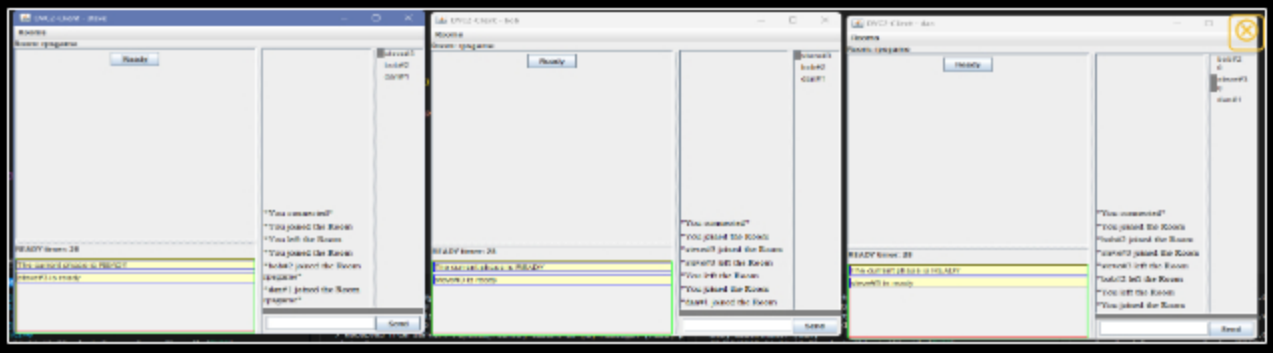
- Clients can mark themselves away and be skipped in turn flow but still part of the game
- The status should be visible to all participants
- A message should be relayed to the Game Events Panel (i.e., Bob is away or Bob is no longer away)
- The user list should have a visual representation (i.e., grayed out or similar)

Part 1:

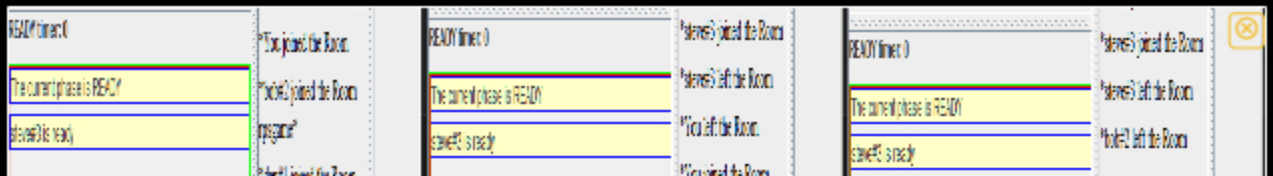
Progress: 100%

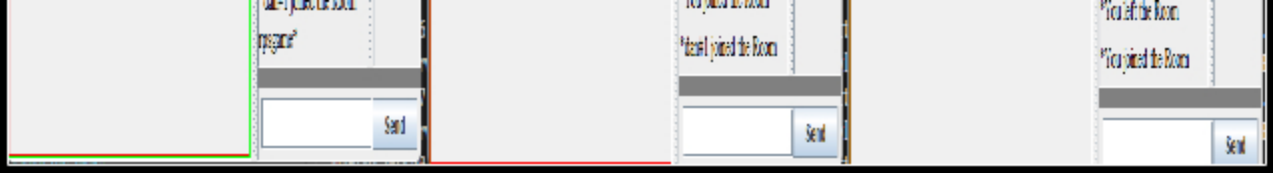
Details:

- Show the UI button to toggle away
- Show the related code flow from UI to server-side back to UI for showing the status
- Show the related code flow for sending the message to Game Events Panel
- Show various examples across 3+ clients of away status (including Game Events Panel messages)
- Show the code that ignores an away user from turn/round logic

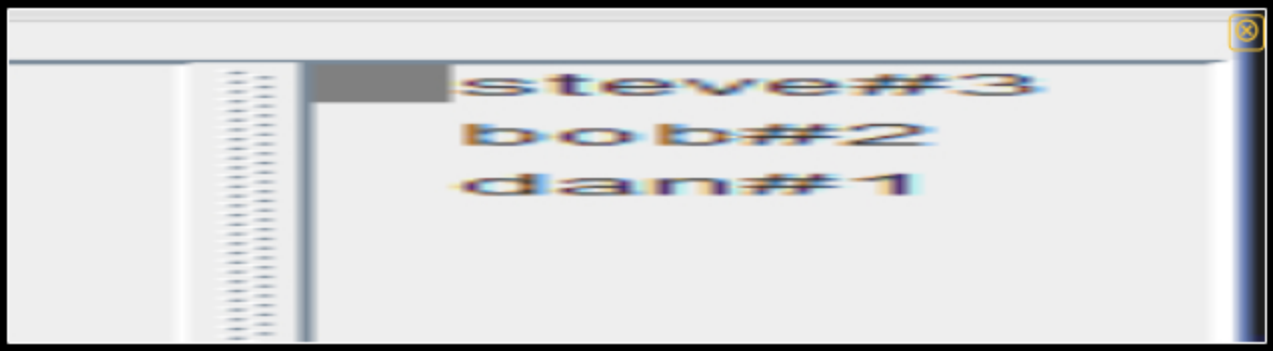


steve is marked ready while the other 2 clients are not marked and are away and the ready status is broadcasted.





steve is ready in the game events panel.



visual representation of steve being ready and the others being away.

```
25
26          JButton awayButton = new JButton(text:"Toggle Away");
27          awayButton.addActionListener(event -> {
28              try {
29                  Client.INSTANCE.sendMessage(message:"/away");
30              } catch (IOException e) {
31                  e.printStackTrace();
32              }
33          });
34
```

toggleAway button in ReadyPanel.java

```
292
293     public void handleAway(ServerThread client) {
294         client.setAway(!client.isAway());
295         String status = client.isAway() ? "is now away" : "is no longer away";
296         relay(sender:null, client.getDisplayName() + " " + status);
297     }
298
```

handleAway code in GameRoom.java

```
80
81      public void setAway(boolean away) {
82          isAway = away;
83          updateAppearance();
84      }
85
```

setAway code in UserListItem.java

```java
    @Override
    protected void onClientRemoved(ServerThread sp) {
        LoggerUtil.INSTANCE.info("Player Removed, remaining: " + clientsInRoom.size());
        if (sp == null) return;
        long removedClient = sp.getClientId();
        turnOrder.removeIf(player -> player.getClientId() == sp.getClientId());
        if (clientsInRoom.isEmpty()) {
            resetReadyTimer();
            resetTurnTimer();
            resetRoundTimer();
            onSessionEnd();
        } else if (removedClient == currentTurnClientId) {
            onTurnStart();
        }
    }
```

onTurnStart code process if player is away.

```java
    private void processRound() {
        List<ServerThread> activePlayers = clientsInRoom.values().stream()
                .filter(p -> p.isReady() && !p.isAway() && !p.isSpectator() && !p.isEliminated())
                .collect(Collectors.toList());

        // Eliminate players who didn't make a choice dvc2 8/4/2025
        activePlayers.stream()
            .filter(p -> !playerChoices.containsKey(p.getClientId()))
            .forEach(p -> {
                p.setEliminated(eliminated:true);
                sendEliminationStatus(p.getClientId(), isEliminated:true);
                relay(sender:null, p.getDisplayName() + " was eliminated for not making a choice.");
            });

        // Re-filter active players after elimination
        activePlayers = activePlayers.stream().filter(p -> !p.isEliminated()).collect(Collectors.toList());
```

processRound code also ignores an away user in the next round when it is started.

Saved: 8/8/2025 4:32:44 AM

Part 2:

Progress: 100%

Details:

- Briefly explain the code flow for the away action from UI to server-side and back to UI
- Briefly explain how the server-side ignores the user from turn/round logic

Your Response:

When a user clicks the "Toggle Away" button in the ReadyPanel, it sends an /away command to the client. The client then sends an AWAY_STATUS payload to the server. The server's GameRoom receives this, toggles the player's isAway status, and broadcasts the change to all clients, which then update their UI to visually indicate the player's new status. The server ignores away players in two key places within the GameRoom. First, in the onTurnStart method, it checks if the current player is away and, if so, immediately skips their turn. Second, in the processRound method, it filters out all away players from the list of active participants, ensuring they are not included in any battles for that round.

Saved: 8/8/2025 4:32:44 AM

Task #2 (1 pt.) - Spectators

Progress: 100%

Details:

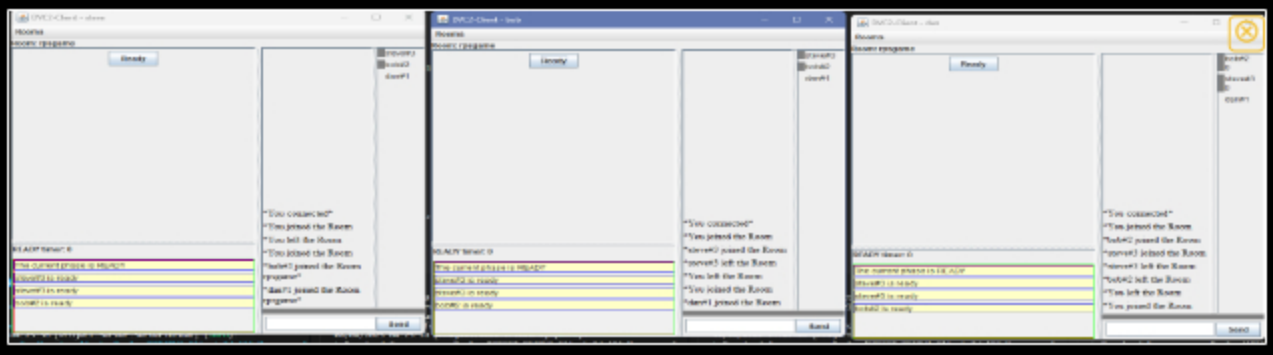
- Spectators are users who didn't mark themselves ready
    - Optionally you can include a toggle on the Ready Check page
- The can see all chat but are ignored from turn/round actions and can't send messages
- Spectators will have a visual representation in the user list to distinguish them from other players
- A message should be relayed to the Game Events Panel that a spectator joined (i.e., during an in-progress session)

Part 1:

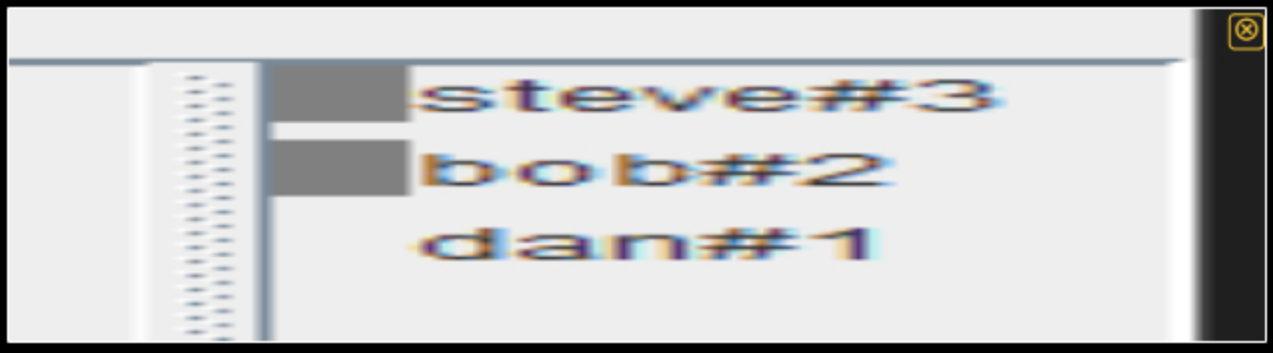
Progress: 100%

Details:

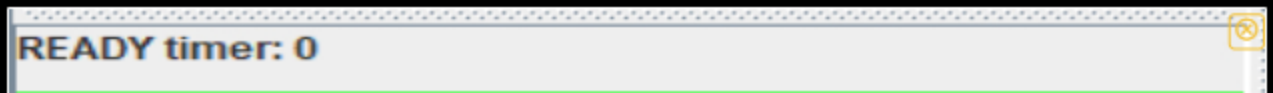
- Show the UI indicator of a spectator (visual and message)
- Show the related code flow from UI to server-side back to UI for showing the status
- Show the related code flow for sending the message to Game Events Panel
- Show various examples across 3+ clients of spectator status (including Game Events Panel messages)
- Show the code that ignores a spectator from turn/round logic
- Show the code that prevents spectators from sending messages (server-side)
- Show the spectator's view of the session
- Show the code related to the spectator seeing the session data (including things participants won't see)

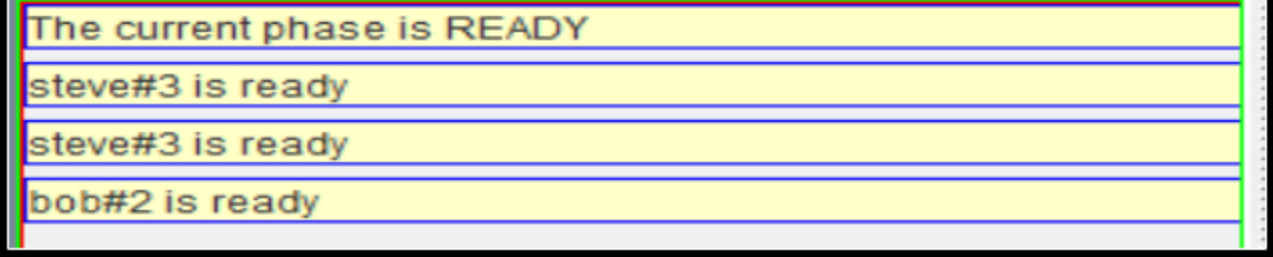


visible across all clients, dan is marked as spectator.



dan marked as spectator on user list.





shows all but dan as being ready.

```
34
35          JButton spectatorButton = new JButton(text:"Join as Spectator");
36          spectatorButton.addActionListener(event -> {
37              try {
38                  Client.INSTANCE.sendMessage(message:"/spectate");
39              } catch (IOException e) {
40                  e.printStackTrace();
41              }
42          });
43
```

code for spectate ui in ReadyPanel.java

```
298
299     public void handleSpectate(ServerThread client) {
300         client.setSpectator(spectator:true);
301         relay(sender:null, client.getDisplayName() + " has joined as a spectator.");
302     }
303
```

handleSpectate code in GameRoom.java

```
85
86      public void setSpectator(boolean spectator) {
87          isSpectator = spectator;
88          updateAppearance();
89      }
```

setSpectator code in UserListItem.java

```
126     @Override
127     protected void onRoundEnd() {
128         LoggerUtil.INSTANCE.info(message:"onRoundEnd() start");
129         resetRoundTimer();
130         processRound();
131
132         long activePlayers = clientsInRoom.values().stream().filter(p -> !p.isEliminated() && !p.isSpectator() && !p.isAway()).count();
133         if (activePlayers <= 1) {
134             onSessionEnd();
135         } else {
136             onRoundStart();
137         }
138         LoggerUtil.INSTANCE.info(message:"onRoundEnd() end");
139     }
```

processRound in GameRoom handles filtering out spectators when the round begins

processRound in GameRoom handles filtering out spectators when the round begins.

```
271
272     public boolean isSpectator() {
273         return this.user.isSpectator();
274     }
275
276     public void setSpectator(boolean spectator) {
277         this.user.setSpectator(spectator);
278     }
```

before processing, isSpectator and setSpectator code in ServerThread.java handles preventing spectators from sending messages.

Saved: 8/8/2025 4:42:30 AM

Part 2:

Progress: 100%

Details:

- Briefly explain the code flow for the spectator logic from server-side and to UI
- Briefly explain how the server-side ignores the user from turn/round logic
- Briefly explain the logic that prevents spectators from sending a message
- Briefly explain the logic that shares extra details to the spectator (information normal participants won't see)

Your Response:

When a user joins as a spectator, the server's GameRoom updates their status and notifies all clients, prompting the UI to visually mark them with a "(Spectator)" tag. The server then ignores spectators in all game logic by filtering them out of the active player list for turns and battles, and the ServerThread is configured to block any chat messages from them. To provide a richer viewing experience, the GameRoom sends special game-event messages containing extra details, like all players' choices, exclusively to clients flagged as spectators.

Saved: 8/8/2025 4:42:30 AM

Section #5: (1 pt.) Misc

Progress: 100%

Task #1 (0.33 pts.) - Github Details

Progress: 100%

Part 1:

Progress: 100%

Details:

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

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

screenshot 1

screenshot 2

Saved: 8/8/2025 2:13:17 AM

Part 2:

Progress: 100%

Details:
Include the link to the Pull Request for Milestone3 to main (should end in `/pull/#`)

URL #1
https://github.com/dcarch2/dvc2-
it114[illegible]

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

Saved: 8/8/2025 2:13:17 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

The time issue is finally fixed, my correct session time for Milestone 3 is shown.

Saved: 8/8/2025 2:09:04 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 how to build a full-stack Java application with a graphical user interface. I also learned how to manage client-server communication, handle game logic, and implement advanced features like player statuses and game settings.

Saved: 8/8/2025 2:07:51 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 getting my project files to compile and run

without any issues. The provided build.sh and run.sh scripts were very helpful and made it easy to test the application after making changes.

Saved: 8/8/2025 2:04:13 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 implementing the user interface and getting all the different panels to work together correctly. It took a lot of trial and error to get the layout and event handling right, especially with the game logic.

Saved: 8/8/2025 2:06:59 AM