Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-003-F2024/it114-milestone-2-trivia-2024-m24/grade/ma2633

Course: IT114-003-F2024
Assigment: [IT114] Milestone 2 Trivia 2024 (M24)
Student: Mohamad A. (ma2633)

Submissions:

Submission Selection

1 Submission [submitted] 11/13/2024 10:26:45 PM

Instructions

COLLAPSE

1. Implement the Milestone 2 features from the project's proposal document: https://docs.google.com/document/d/1h2aEWUoZ-etpz1CRI-StaWbZTjkd9BDMq0b6TXK4utI/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: 3
Points: 1

COLLAPSE

Task

100%

Group: Payloads
Task #1: Base Payload Class
Weight: ~33%
Points: ~0.33

COLLAPSE

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

Columns: 1

Sub-Task

100%

Group: Payloads
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

payload code

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain the purpose of each property and serialization

Response:

The Payload class has three main properties: payloadType to indicate what kind of data is being sent (like a message or command), clientId to identify which client is sending the data, and message to carry the actual content. Serialization allows this object to be turned into a format that can be sent over a network, so different clients and servers can easily share and understand the information.

Sub-Task

100%

Group: Payloads
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

Caption(s) (required) ✓

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

End of Task 1

Task

100%

Group: Payloads
Task #2: QAPayload Class
Weight: ~33%
Points: ~0.33

COLLAPSE

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

Columns: 1

Sub-Task

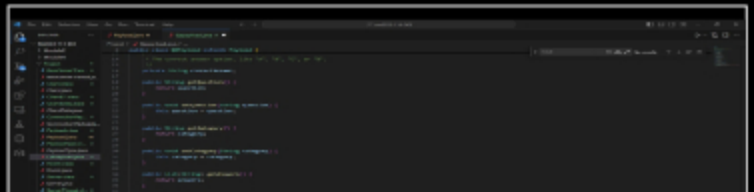
100%

Group: Payloads
Task #2: QAPayload Class
Sub Task #1: Show screenshot of the code for this file

Task Screenshots

Gallery Style: 2 Columns

4 2 1



the rest of the code

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain the purpose of each property

Response:

he QAPayload class has four main properties: question stores the text of the question that is presented to players, while category specifies the category or type of the question, such as Science or History. The answers property is a list of possible answer choices that the players can pick from. Finally, correctAnswer indicates the correct option, such as "A", "B", "C", or "D", which helps in evaluating players' responses.

Sub-Task

100%

Group: Payloads

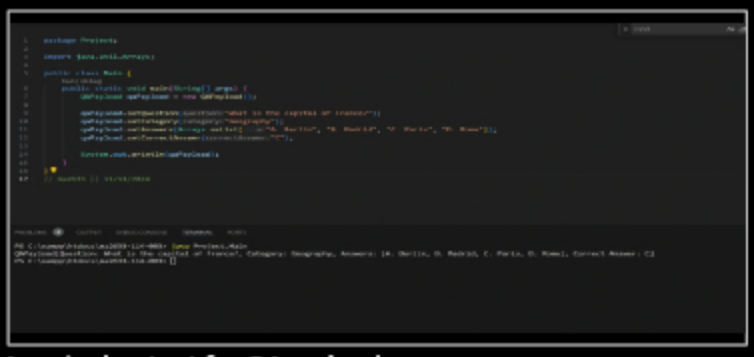
Task #2: QAPayload Class

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 QApayload

Caption(s) (required) ✓

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

End of Task 2

Task

100%

Group: Payloads

Task #3: PointsPayload Class

Weight: ~33%

Points: ~0.33

COLLAPSE

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

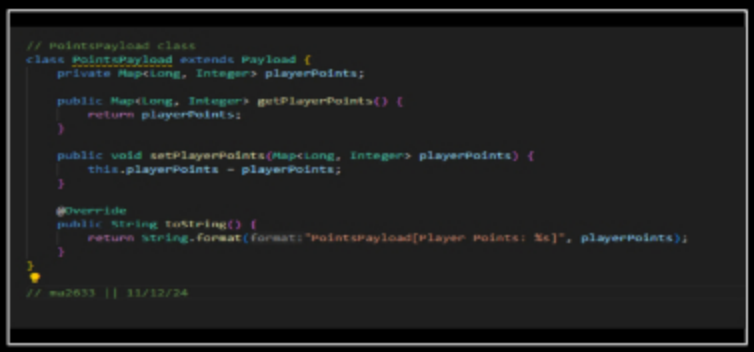
Columns: 1

Sub-Task
100%
Group: Payloads
Task #3: PointsPayload Class
Sub Task #1: Show screenshot of the code for this file

Task Screenshots

Gallery Style: 2 Columns

4 2 1



pointpayload

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain the purpose of each property

Response:

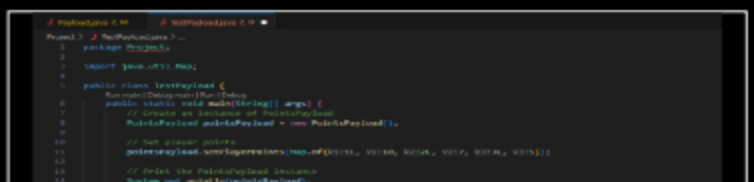
layerPoints (Map<Long, Integer>): This property stores the points of each player in the game. The Map uses the player's ID (of type Long) as the key and the player's points (of type Integer) as the value. This helps keep track of each player's score throughout the game.

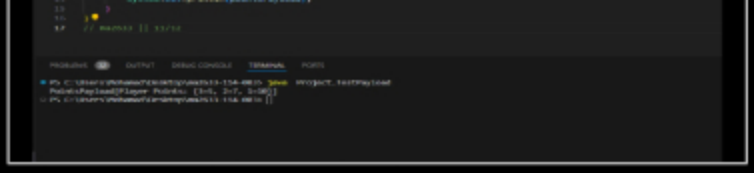
Sub-Task
100%
Group: Payloads
Task #3: PointsPayload Class
Sub Task #2: Show screenshot examples of the terminal output for this object

Task Screenshots

Gallery Style: 2 Columns

4 2 1





terminal output of piontpayload

Caption(s) (required) ✓

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

End of Task 3

End of Group: Payloads
Task Status: 3/3

Group

100%

Group: Questions
Tasks: 1
Points: 1.25

COLLAPSE

Task

100%

Group: Questions
Task #1: Data Structure
Weight: ~100%
Points: ~1.25

COLLAPSE

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

Columns: 1

Sub-Task

100%

Group: Questions
Task #1: Data Structure
Sub Task #1: Show the code related to Questions/Answers (text, category, answers, correct)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Show the code related to Questions/Answers

Caption(s) (required) ✓

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

Task Response Prompt

Explain the class and each property

Response:

The Question class is a simple data container for a trivia game. It has four properties: questionText, category, answerOptions, and correctAnswer. The questionText is the actual question being asked, while the category helps group questions (like "Math" or "History"). The answerOptions is a list of possible answers players can pick from, and correctAnswer is the right answer from those options. There's a constructor to set all these properties, and getters to access them easily. There's also a toString() method to neatly print out the question details, which is helpful for seeing what's stored in a question.

Sub-Task

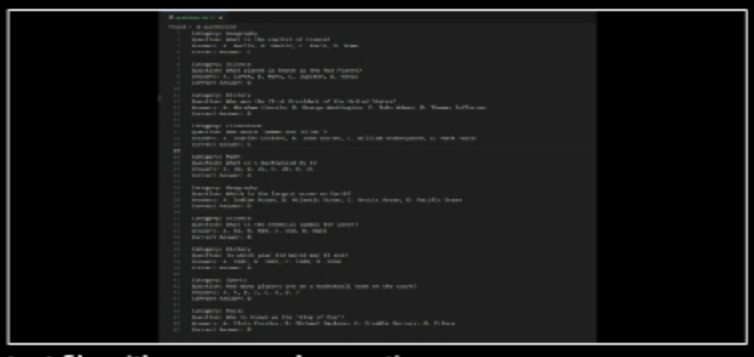
100%

Group: Questions
Task #1: Data Structure
Sub Task #2: Show the text file with your sample questions and answers

Task Screenshots

Gallery Style: 2 Columns

4 2 1



text file with your sample questions

Caption(s) (required) ✓

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

End of Task 1

End of Group: Questions
Task Status: 1/1

Group

Group: Session Start
Tasks: 2

100%

Points: 1.25

COLLAPSE

Task

100%

Group: Session Start
Task #1: Question List
Weight: ~50%
Points: ~0.63

COLLAPSE

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

Sub-Task

100%

Group: Session Start
Task #1: Question List
Sub Task #1: Show the GameRoom loading the question list from a file

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Method to load questions from a file

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

In this method, the code reads questions from a file called "questions.txt". It reads each line, extracts the category, question, answer options, and the correct answer, and then adds the questions to questionList. This helps load all the game questions from an external file, making it easy to update or add new questions.

End of Task 1

Task

100%

Group: Session Start
Task #2: First round
Weight: ~50%
Points: ~0.63

COLLAPSE

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

Sub-Task

100%

Group: Session Start
Task #2: First round
Sub Task #1: Show the code that triggers the first round

Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
    // ma2633 || 11/12

// Method to start the first round
public void startFirstRound() {
    if (!questionList.isEmpty()) {
        currentRound++;
        System.out.println("Starting round " + currentRound); // Debug message
        startRound();
    } else {
        System.out.println("No questions available to start the game.");
    }
}
```

code that triggers the first round

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

End of Task 2

End of Group: Session Start
Task Status: 2/2

Group

100%

Group: Round Start
Tasks: 3
Points: 1.25

COLLAPSE

Task

100%

Group: Round Start
Task #1: Picking a question
Weight: ~33%
Points: ~0.42

COLLAPSE

Sub-Task

100%

Group: Round Start
Task #1: Picking a question
Sub Task #1: Show the code related to getting a random question from the list (it should get removed so it's not used again)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

getting a random question from the list

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

currentQuestion = questionList.remove((int) (Math.random() * questionList.size()));: This line gets a random index from the questionList. Math.random() * questionList.size() generates a random index. questionList.remove(...) removes the selected question from the list, so it's not used again.

End of Task 1

Task

100%

Group: Round Start
Task #2: Syncing Question
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.

Columns: 1

Sub-Task
100%
Group: Round Start
Task #2: Syncing Question
Sub Task #1: Show the code the syncs the question and answer choices to all clients (don't send the correct answer)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

code the syncs the question and answer choices to all clients

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

The broadcastQuestionToClients(Question question) method: QAPayload payload = new QAPayload(...): Creates a payload with the question text and answer choices. The payload does not include the correct answer, ensuring clients only receive the question and possible answer choices. for (ClientData client : clients): Iterates over all connected clients. client.getServerThread().sendPayload(payload): Sends the payload (question and answer choices) to each client.

Sub-Task
100%
Group: Round Start
Task #2: Syncing Question
Sub Task #2: Show the terminal output of all clients receiving it

Task Screenshots

Gallery Style: 2 Columns

4 2 1

clients receiving questions

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.

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

```
// Method to start the round timer
private void startRoundTimer() {
    roundTimer.schedule(new TimerTask() {
        @Override
        public void run() {
            System.out.println(x:"Round timer expired.");
            endRound();
        }
    }, delay:30000); // 30 seconds per round
}
    // ma2633 || 11/12
```

Method to start the round timer

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

Timer Initialization: The startRoundTimer() method uses the Timer class to schedule a timer task.

Timer Task: A new TimerTask is created and scheduled to run after 30,000 milliseconds (or 30 seconds). This means that the round will automatically expire after 30 seconds.

Expiry Callback: When the timer expires, the run() method inside the TimerTask is called, and it prints "Round timer expired." to the terminal, then it calls the endRound() method to handle the logic for ending the round.

Sub-Task

100%

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

Method to start the round timer with periodic updates to clients

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

The startRoundTimer() method schedules a repeating task that decreases the remaining time by a fixed interval (updateInterval). The broadcastTimeUpdate() method is called to inform all clients of the time remaining in the round.

Sub-Task

100%

Group: Round Start
Task #3: Round Timer
Sub Task #3: Show an example from the terminal

Task Screenshots

Gallery Style: 2 Columns

Gallery Style: 2 Columns

4 2 1

Cients are informed of the time

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

100%

Group: During Round
Tasks: 2
Points: 1.5

COLLAPSE

Task

100%

Group: During Round
Task #1: Answer Command
Weight: ~50%
Points: ~0.75

COLLAPSE

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

Columns: 1

Sub-Task

100%

Group: During Round
Task #1: Answer Command
Sub Task #1: Show the code that processes the /answer text

Task Screenshots

Gallery Style: 2 Columns

4 2 1

answer processes

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

handleAnswerPayload(Payload payload): This is a private method that processes an answer payload received from a client.

if (currentRoom != null && currentRoom instanceof GameRoom): This condition checks if the client is currently in a room and if the room is an instance of GameRoom. This ensures that the answer is processed only when the player is in a game room.

Sub-Task

100%

Group: During Round

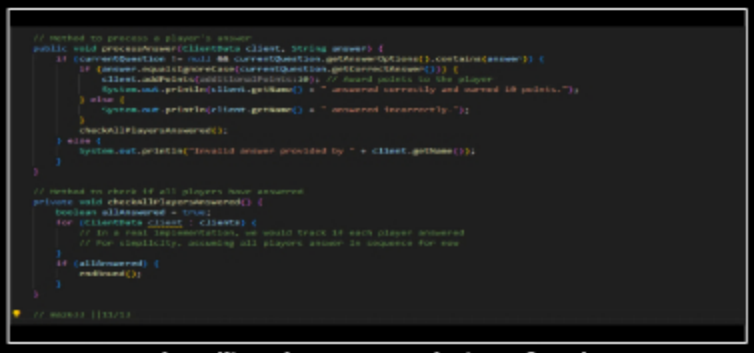
Task #1: Answer Command

Sub Task #2: Show the code of the GameRoom handling the answer choice of a Player (and how correct answers are handled) Show that the answer is blocked from changing.

Task Screenshots

Gallery Style: 2 Columns

4 2 1



GameRoom handling the answer choice of a Player

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Explain in concise steps how this logically works

Response:

processAnswer(ClientData client, String answer): This method processes the answer from the client. if (currentQuestion != null && currentQuestion.getAnswerOptions().contains(answer)): Checks if the current question is available and if the provided answer is valid.

Sub-Task

100%

Group: During Round
Task #1: Answer Command
Sub Task #3: Show the code from the GameRoom that tells all Players a Player locked in an answer

Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
// Method to notify all players when a player locks in an answer
public void notifyPlayersAnswerLocked(ClientData player) {
    String message = player.getName() + " has locked in an answer.";
    Payload payload = new Payload(clientId:"Server", message, PayloadType.NOTIFICATION);
    for (ClientData client : clients) {
        client.getServerThread().sendPayload(payload);
    }
}
    //ma2048 || 11/15
```

code from the GameRoom that tells all Players a Player locked in an answer

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

Player Submits Answer: When a player provides an answer, the processAnswer() method is called.

Answer Evaluation: The processAnswer() method checks if the provided answer is valid for the current question.

Award Points: If the answer is correct, the player is awarded points, and a message is printed for feedback.

Broadcast Answer Lock-In: Regardless of correctness, a message is broadcasted to all players to inform them that the player has locked in their answer.

Notify Players: The new broadcastPlayerLockedInAnswer() method is invoked, which sends a notification payload to all players, indicating that a player has locked in an answer.

Sub-Task

100%

Group: During Round
Task #1: Answer Command
Sub Task #4: Show at least 3 terminal examples of wrong and correct answers (should be shown on all clients)

Task Screenshots

Task Screenshots

Gallery Style: 2 Columns

4 2 1

3 terminal examples of wrong and correct answers

Caption(s) (required) ✓

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

End of Task 1

Task

100%

Group: During Round
Task #2: Join in progress
Weight: ~50%
Points: ~0.75

COLLAPSE

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

Sub-Task

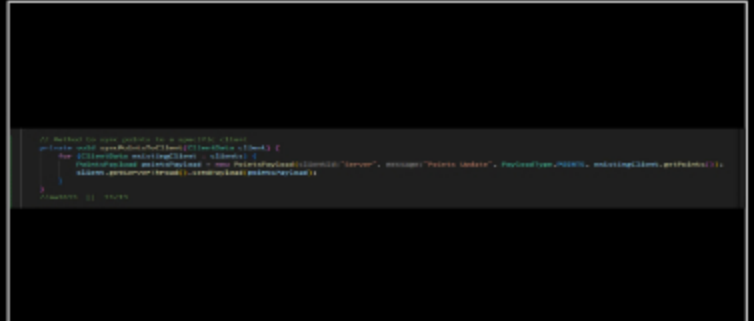
100%

Group: During Round
Task #2: Join in progress
Sub Task #1: Show the code related to syncing game state to a joining client if the game is in progress

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Method to sync points to a specific client

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

he method iterates through each client in the list and sends the points update to the newly joined client. It creates a PointsPayload object with relevant data, such as the client ID, message, payload type, and the points value, and sends it using the sendPayload() method.

End of Task 2

End of Group: During Round
Task Status: 2/2

Group

100%

Group: Round End
Tasks: 3
Points: 1.5

COLLAPSE

Task

100%

Group: Round End
Task #1: End Round Conditions
Weight: ~33%
Points: ~0.50

COLLAPSE

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

Columns: 1

Sub-Task

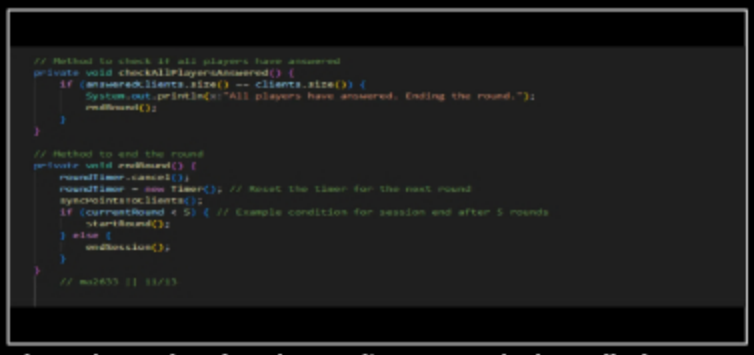
100%

Group: Round End
Task #1: End Round Conditions
Sub Task #1: Show the code related to ending a round when all Players answer correctly

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Show the code related to ending a round when all Players answer correctly

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

processAnswer(): Adds clients to answeredClients after they answer and checks if all players have answered using checkAllPlayersAnswered(). checkAllPlayersAnswered(): Ends the round by calling endRound() if all players have answered. endRound(): Cancels the timer, synchronizes points, and either starts a new round or ends the session after a certain number of rounds.

Sub-Task

100%

Group: Round End

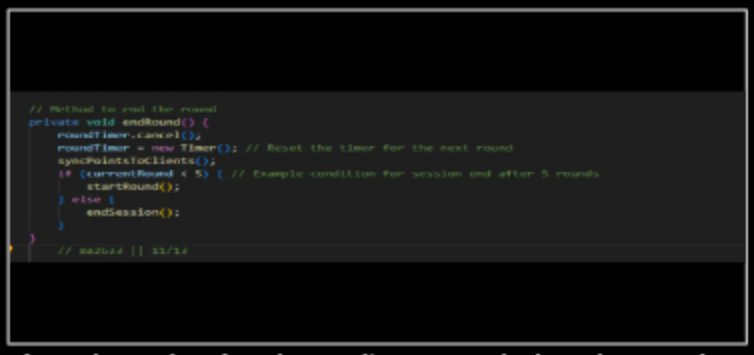
Task #1: End Round Conditions

Sub Task #2: Show the code related to ending a round when the round timer expires

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Show the code related to ending a round when the round timer expires

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

startRoundTimer(): Sets a timer to track the duration of the round. If the timer runs out, it calls endRound() to stop the round. endRound(): Cancels the timer, resets it, synchronizes points to clients, and either starts a new round or ends the session based on the current round count.

End of Task 1

Task

100%

Group: Round End
Task #2: Points
Weight: ~33%
Points: ~0.50

COLLAPSE

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

Columns: 1

Sub-Task

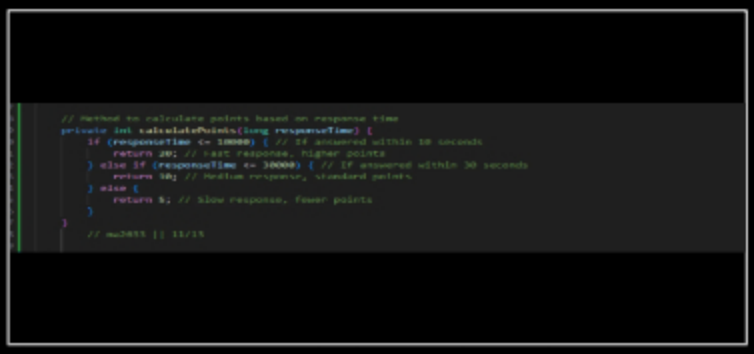
100%

Group: Round End
Task #2: Points
Sub Task #1: Show how points are calculated (fastest = higher score)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Show how points are calculated (fastest = higher score)

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

rack Response Time: When a player answers, the response time is calculated from when the question was broadcasted. Calculate Points: The calculatePoints() method assigns: 20 points if answered within 10 seconds. 10

points if answered within 30 seconds. 5 points if answered after 30 seconds.

Sub-Task

100%

Group: Round End
Task #2: Points
Sub Task #2: Show the code related to syncing points to the Clients

Task Screenshots

Gallery Style: 2 Columns

4 2 1

code related to syncing points to the Clients

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

This method loops through all clients in the game room and sends their points to their respective server threads.

Sub-Task

100%

Group: Round End
Task #2: Points
Sub Task #3: Show the code related to the Client updating their local player map

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Update the local player points map

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

The client receives a PointsPayload from the server, which contains the player's ID (getClientId()) and their updated points (getPoints()). The client updates its local map (playerPoints) using the player ID as the key and the points as the value:

Sub-Task

100%

Group: Round End

Task #2: Points

Sub Task #4: Show the code related to the in-progress scoreboards

Task Screenshots

Gallery Style: 2 Columns

4 2 1

code related to the in-progress scoreboards

code related to the in-progress scoreboards

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

Method to Sync Points to All Clients:Sends updated points to all clients, ensuring everyone has the latest score information. This method is usually called at the end of each round to broadcast scores. Method to Sync Points to a Specific Client:Sends the points of all clients to a specific client, usually when a new client joins an in-progress game. This ensures that the newly joined client has the most up-to-date scoreboard. Updating Local Player Map with Score Information:When the client receives a PointsPayload from the server, it updates its local player points map (playerPoints). This keeps each client informed of the current scores during the game, allowing for an up-to-date in-progress scoreboard.

Sub-Task

100%

Group: Round End

Task #2: Points

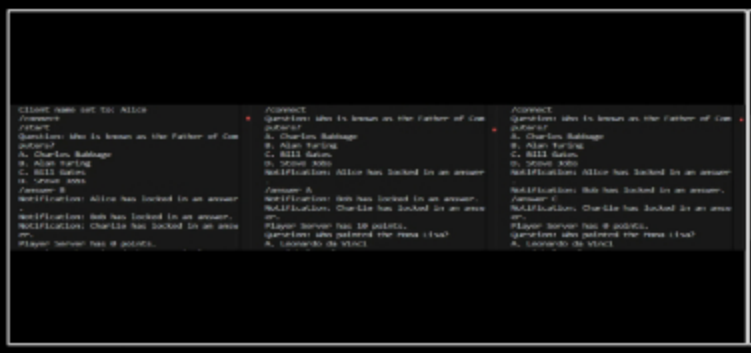
Sub Task #5: Show at least 3 examples of terminal output of the scoreboards

Task Screenshots

Task Screenshots

Gallery Style: 2 Columns

4 2 1



1examples

2examples

3examples

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 first picture only one got it right and the score update to 10 the second picture in the next question the also one client got it right score update to 10 the 3rd picture in the next question the also one client got it right score update to 10

each one of them only got one answer right

End of Task 2

Task

100%

Group: Round End

Task #3: Next Round or Session End

Weight: ~33%

Points: ~0.50

COLLAPSE

Details:

All code screenshots must have ucid/date visible.

Sub-Task

100%

Group: Round End
Task #3: Next Round or Session End
Sub Task #1: Show the code related to triggering the next round or session end (end game condition)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Method to end the round

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

endRound() is called when a round ends. It either starts the next round or ends the session based on the game conditions.

End of Task 3

End of Group: Round End
Task Status: 3/3

Group

100%

Group: Session End
Tasks: 3
Points: 1.25

COLLAPSE

Task

100%

Group: Session End
Task #1: Session End Condition
Weight: ~33%

Points: ~0.42

COLLAPSE

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

Sub-Task

100%

Group: Session End
Task #1: Session End Condition
Sub Task #1: Show the code related to the session ending when X rounds have passed

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Method to end the session

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

When the condition for ending the session is met, the endSession() method is called to send the final scores to all clients and reset the game.

End of Task 1

Task

100%

Group: Session End
Task #2: Scoreboards
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.

Columns: 1

Sub-Task

100%

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Send final scoreboard to all clients

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

This uses the PointsPayload to provide each player with their final points, labeled as "Final Score".

Sub-Task

100%

Group: Session End
Task #2: Scoreboards
Sub Task #2: Show the final scoreboard from the terminal

Task Screenshots

Gallery Style: 2 Columns

4 2 1

final scoreboard from the terminal

Caption(s) (required) ✓

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

End of Task 2

Task

100%

Group: Session End
Task #3: Cleanup
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.

Columns: 1

Sub-Task

100%

Group: Session End
Task #3: Cleanup
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

Method to reset the game

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

currentRound is reset to 0. The questionList is cleared, indicating that a new set of questions should be loaded for

the next game. The answeredClients list is cleared to remove tracking of answers from previous rounds.

Sub-Task

100%

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Round is reset to 0

terminal evidence

Caption(s) (required) ✓

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

Sub-Task

100%

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

Method to shift all players 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:

This method is used to shift all players back to a ready state, preparing them for a new session after the current game session ends. Reset Player Points: The method iterates through each client in the clients list and resets their points to zero using client.setPoints(0);. Send Notification: A Payload object is created with the message "Game is ready for a new session" and a type of PayloadType.NOTIFICATION. This payload is then sent to each client to let them know they are ready for a new session. Log Message: Finally, a log message ("Players are shifted back to the ready phase.") is printed to indicate the operation has been completed.

End of Task 3

End of Group: Session End
Task Status: 3/3

Group

100%

Group: Misc
Tasks: 3
Points: 1

COLLAPSE

Task

100%

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/Mohamad4322/ma2633-114-003/

URL
https://github.com/Mohamad4322/ma2633-114-

End of Task 1

Task

100%

Group: Misc
Task #2: Talk about any issues or learnings during this assignment
Weight: ~33%
Points: ~0.33

COLLAPSE

Task Response Prompt

Response:

what I found challenging was the processData() function, which involved multiple nested loops and conditional checks. At first, it was hard to follow the logic because each condition affected different variables in subtle ways. I had to carefully go through each line to understand how data flowed through the function. Working through it helped me appreciate the importance of structuring methods clearly, and I learned that breaking down complex logic into smaller helper functions could make code much easier to read and debug.

End of Task 2

Task

100%

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

WakaTime

End of Task 3

End of Group: Misc
Task Status: 3/3

End of Assignment