Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-002-S2024/it114-milestone-2-chatroom-2024/grade/ob75

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

Submissions:

Submission Selection

1 Submission [active] 3/31/2024 6:19:10 PM

Instructions

COLLAPSE

Implement the Milestone 2 features from the project's proposal document:
https://docs.google.com/document/d/1ONmvEvel97GTFPGfVwwQC96xSsobbSbk56145XizQG4/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: 12 Points: 10.00

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, equivalent of RollPayload, and any others
#2	1	Screenshots should include ucid and date comment
#3	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large

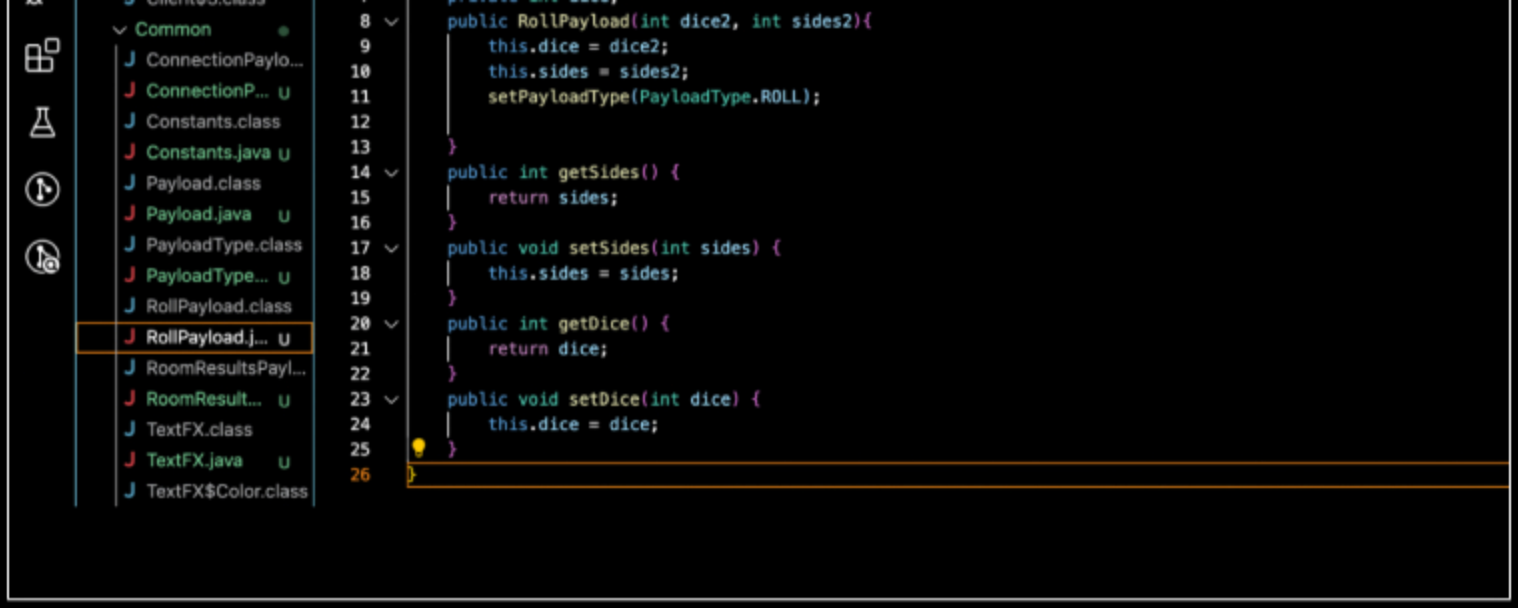
PayloadType class that now contains ROLL and FLIP as payload types.

Checklist Items (3)

#1 Payload, equivalent of RollPayload, and any others

#2 Screenshots should include ucid and date comment

#3 Each screenshot should be clearly captioned



This is new Payload class for Roll to handle the information sent from the client to here and sends this out to server side. Contains getters and setters and the RollPayload the extends Payload class. Also created int variable for dice and sides and sets payload type. UCID included and whole class is evidence.

Checklist Items (3)

#1 Payload, equivalent of RollPayload, and any others

#2 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

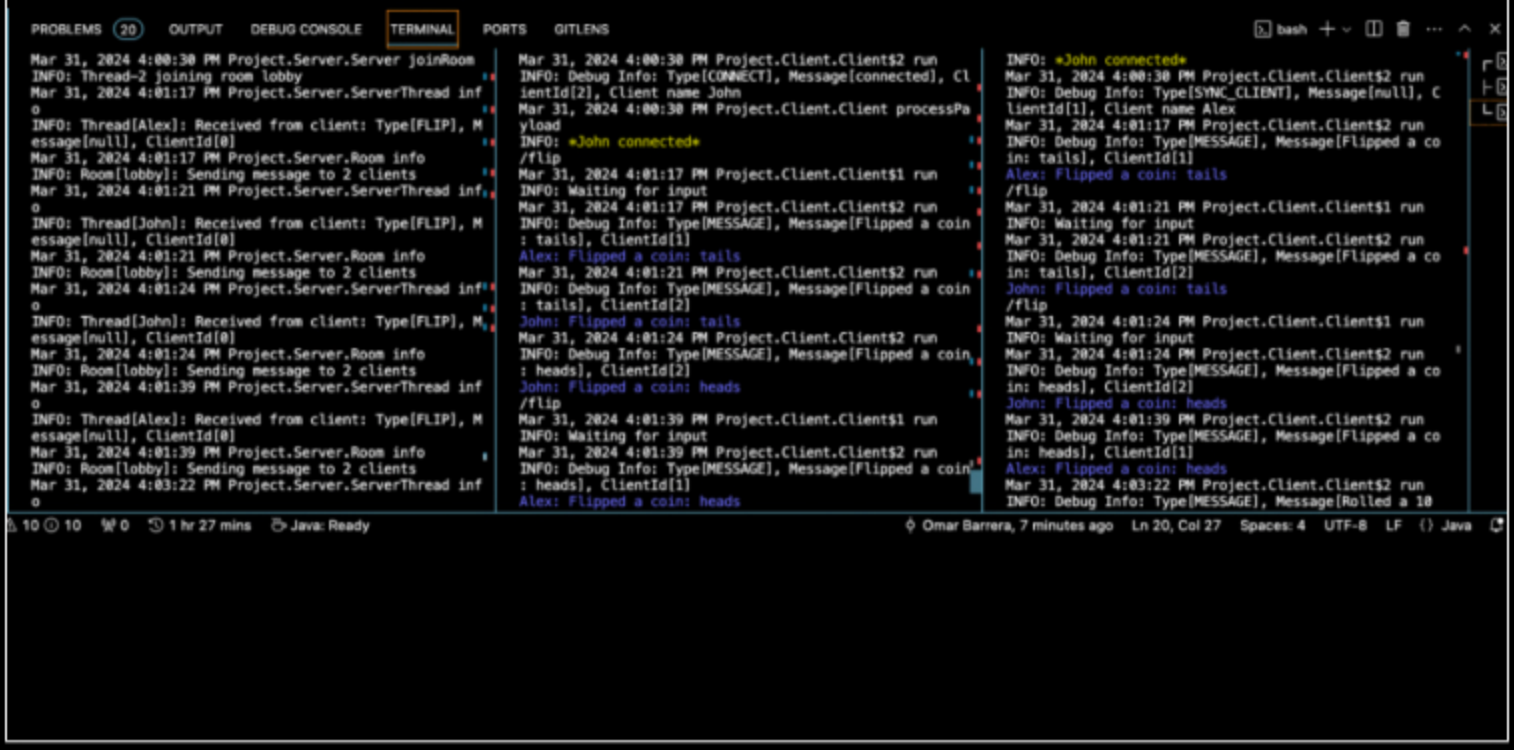
*The checkboxes are for your own tracking

#	Points	Details
#1	1	Demonstrate flip
#2	1	Demonstrate roll (both versions)
#3	1	Demonstrate formatted message along with any others
#4	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large



Terminal showing flip command working properly both ways for heads and tails and shows message being sent to all clients in room. Opened with 2 client window on the right to show message properly being sent to all clients in room.

Checklist Items (2)

#1 Demonstrate flip

#4 Each screenshot should be clearly captioned

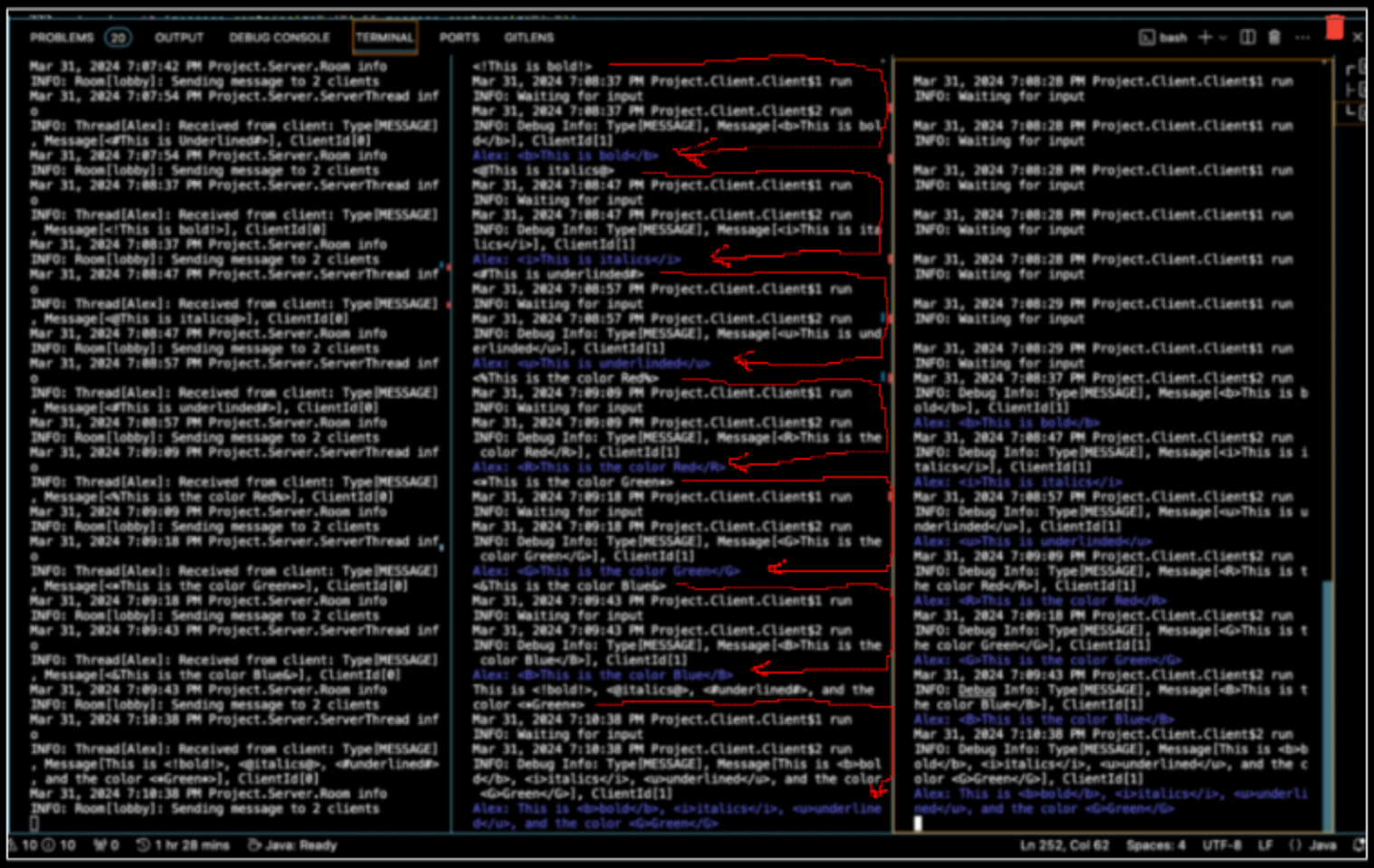
Terminal in middle and right show clients typing in command for both ways of roll. /roll# and /roll #d#. Message is

properly sent out to all clients and shows who rolled and what they rolled and the result of the roll for the client.

Checklist Items (2)

#2 Demonstrate roll (both versions)

#4 Each screenshot should be clearly captioned



Terminal in middle showing text being typed with special characters and the output being the replaced tags for them. Created brush red brush lines to show a bit more clear of the input of the message wrapped with special characters and pointing to their new tags that replaced them. Client window on the right shows message being properly received into the room and reading new message.

Checklist Items (2)

#3 Demonstrate formatted message along with any others

#4 Each screenshot should be clearly captioned

COLLAPSE

Task #3 - Points: 1

Text: Explain the purpose of payloads and how your flip/roll payloads were made

Response:

The purpose of payloads is to receive data and turn them into objects to send out to the server to take that payload and know how to handle it and will then send it to the Room class where it is then handled to do the method that is

associated with it and send the information out to the room. In this case, flip and roll payloads were first made by adding them into the PayloadType class naming them FLIP and ROLL. A RollPayload class was created to take in the data sent from the client side to use those values and also set the PayloadType to ROLL in this class. There is also a sendRoll method in the client side to pass in the value of dice and sides and create the payload with the values the client or user typed. With FLIP, there was no separate FLIP payload class made and setPayloadType(FLIP) was made in the client side inside sendFlip() method to create the payload of FLIP. Both payloads are sent to server side and then sent to the Room side to handle the payloads and send back the results of either the flip or roll to the room with the clients.

Demonstrate Roll Command (2 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshot of the following items

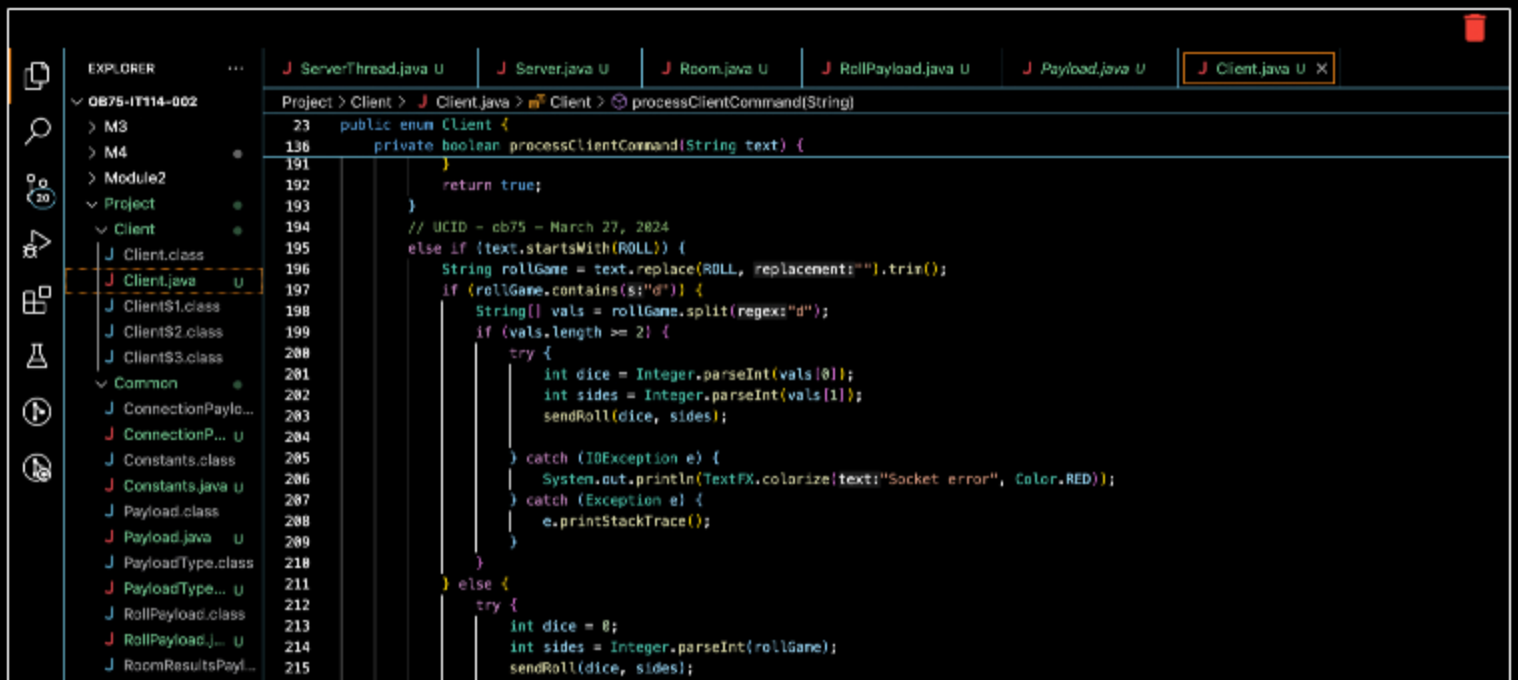
Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Client code that captures the command and converts it to a RollPayload (or equivalent) for both scenarios /roll # and /roll #d#
#2	1	ServerThread code receiving the payload and passing it to the Room
#3	1	Room handling the roll action correctly for both scenarios (/roll # and /roll #d#) including the message going back out to all clients
#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



```
RoomResult...
TextFX.class
TextFX.java
TextFX$Color.class
Server
Room.class
Room.java
Room$1.class
Server.class

216
217        } catch (IOException e) {
218            System.out.println(TextFX.colorize(text:"Socket error", Color.RED));
219        } catch (Exception e) {
220            e.printStackTrace();
221        }
222    }
223
224    return true;
225 }
226
```

Completed processingClientCommand for roll and handles it in the else if statement under UCID comment on line 195 - up until line 225.

Checklist Items (3)

#1 Client code that captures the command and converts it to a RollPayload (or equivalent) for both scenarios /roll # and /roll #d#

#4 Code screenshots should include ucid and date comment

#5 Each screenshot should be clearly captioned

```
EXPLORER
OB75-IT114-002
M3
M4
Module2
Project
Client
Client.class
Client.java

ServerThread.java U | Server.java U | Room.java U | RollPayload.java U | Payload.java U | Client.java U
Project > Client > Client.java > Client > processClientCommand(String)

23  public enum Client {
40      private static final String DISCONNECT = "/disconnect";
41
42      // UCID - ob75 - March 27, 2024
43      private static final String ROLL = "/roll";
44
45      private static final String FLIP = "/flip";
46
```

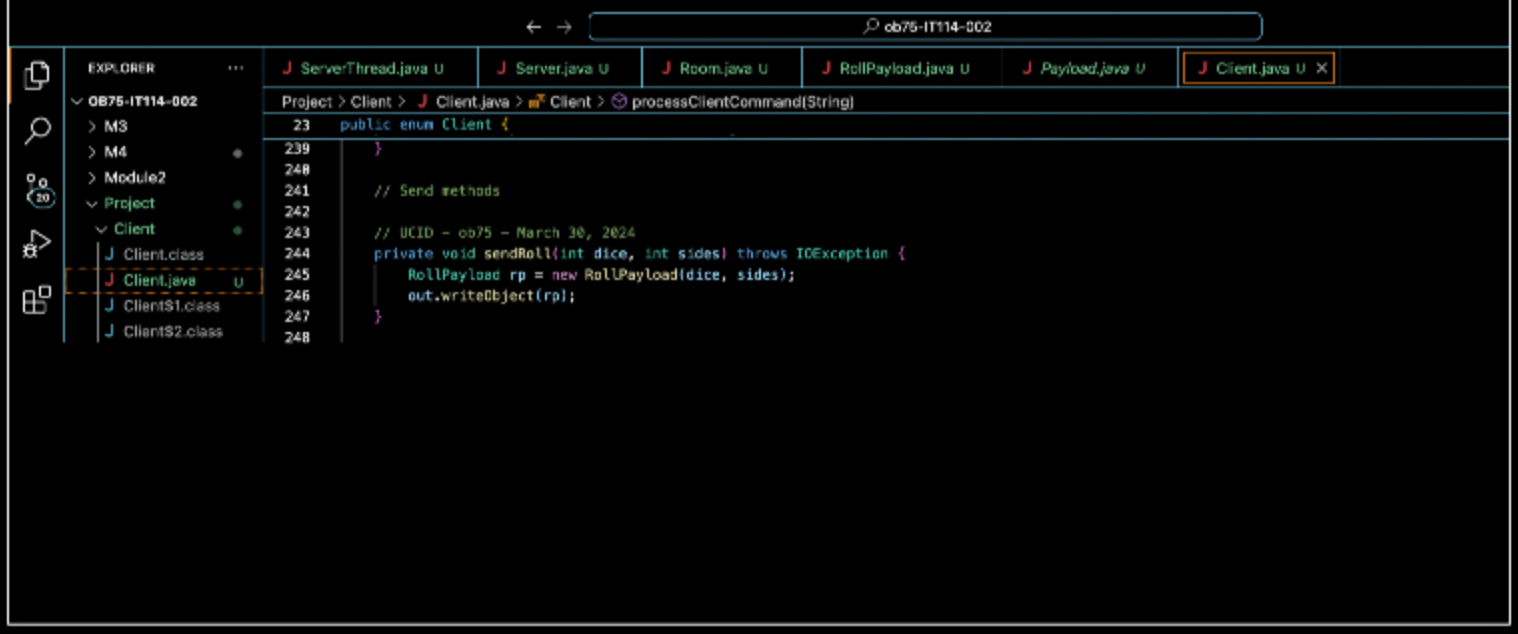
Setting the command "/roll" to ROLL when the user types to roll and the value they set. Under UCID comment, Line 43.

Checklist Items (3)

#1 Client code that captures the command and converts it to a RollPayload (or equivalent) for both scenarios /roll # and /roll #d#

#4 Code screenshots should include ucid and date comment

#5 Each screenshot should be clearly captioned



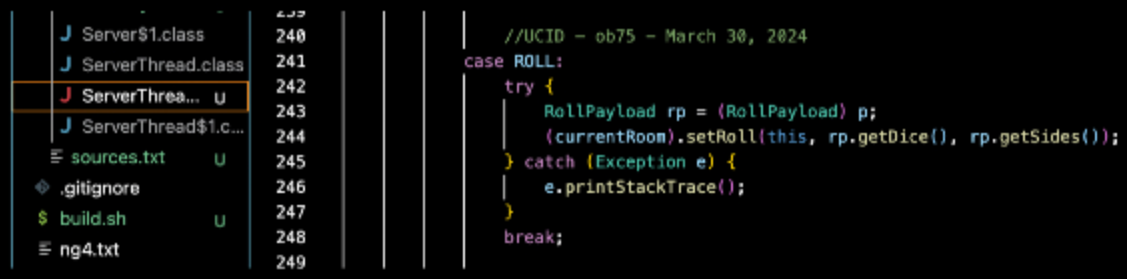
The sendRoll method in the Client side that takes the value of dice and sides and creates the payload to send it out. Under UCID, Lines 244 - 247

Checklist Items (3)

#1 Client code that captures the command and converts it to a RollPayload (or equivalent) for both scenarios /roll # and /roll #d#

#4 Code screenshots should include ucid and date comment

#5 Each screenshot should be clearly captioned



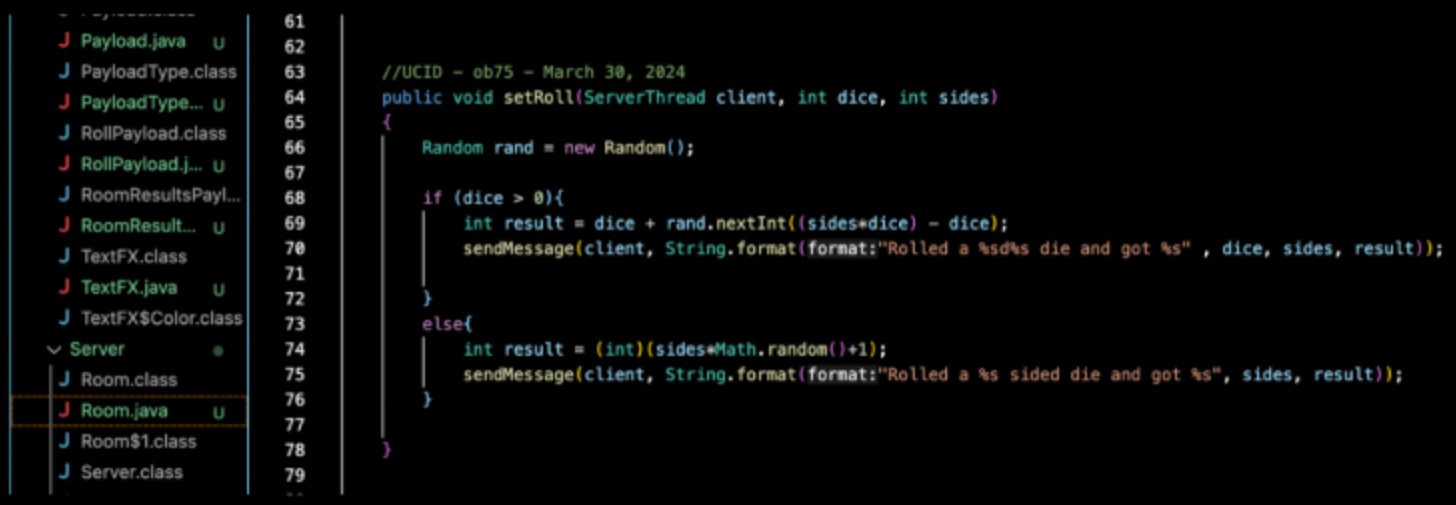
ServerThread side handling the RollPayload if the case is ROLL and passes it onto the room to handle the method in creating the Roll with the values of dice and sides. Under UCID comment, Lines 241-248.

Checklist Items (3)

#2 ServerThread code receiving the payload and passing it to the Room

#4 Code screenshots should include ucid and date comment

#5 Each screenshot should be clearly captioned



Created the method for Roll and sending a message containing the client, the type of roll and the results of the roll back out to all clients. Under UCID comment, Lines 64-78.

Checklist Items (3)

#3 Room handling the roll action correctly for both scenarios (/roll # and /roll #d#) including the message going back out to all clients

#4 Code screenshots should include ucid and date comment

#5 Each screenshot should be clearly captioned

COLLAPSE

Task #2 - Points: 1

Text: Explain the logic in how the two different roll formats are handled and how the message flows from the client, to the Room, and shared with all other users

Response:

The logic in this code is an else if statement going into the processClientCommand in the client side to check if the the command start with the roll command and then perform the action of splitting d if the roll contains "d" where the first number before d is dice and the number after is sides and sending those values via sendRoll(dice, sides). If the command does not contain #d#, it will go to the else block where dice is set to 0 since there is just sides being typed in this situation with a normal roll for example /roll10 where 10 is the max range for sides. This else statement also has sendRoll(dice, sides) inside to send the value typed by the user with dice turning to 0 so its not used and sides being the value by the client. The sendRoll method in the client side will pass the values of dice and sides into the payload and create it to send it to the server thread side. Here, the server thread will take the payload and will get the current room and the values and dice and sides to send it to the Room class calling the method setRoll. In the room, the method for rolling is done and takes in the client name and int dice and int sides to determine whether to roll for a number of dice and sides if dice is greater than 0. It is then randomized on the roll and the result is then given and set to the variable int result. A message is then sent back out to the clients in the room stating the client and the message saying they "rolled a #d# die" and then the result. If the dice is 0 however, this is where normal dice roll would happen and would randomize the result given just the sides range and then that message is sent back to clients in the room the same way but instead says they "rolled a #sided die" and the result.

Demonstrate Flip Command (1 pt.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshot of the following items

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Client code that captures the command and converts it to a payload
#2	1	ServerThread receiving the payload and passing it to the Room
#3	1	Room handling the flip action correctly
#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

```
23  public enum Client {
40      private static final String DISCONNECT = "/disconnect";
41
42      // UCID - ob75 - March 27, 2024
43      private static final String ROLL = "/roll";
44
45      private static final String FLIP = "/flip";
46
```

Setting the command "/flip" to FLIP when the user types to flip a coin. Line 45.

Checklist Items (3)

#1 Client code that captures the command and converts it to a payload

#4 Code screenshots should include ucid and date comment

#5 Each screenshot should be clearly captioned

```
23  public enum Client {
136     private boolean processClientCommand(String text) {
225         }
226
227             // UCID - ob75 - March 30, 2024
228         else if (text.startsWith(FLIP)) {
229             try {
230                 sendFlip();
231
232             } catch (Exception e) {
233                 e.printStackTrace();
234             }
235             return true;
236         }
237         return false;
238
239     }
```

Completed processClientCommand for flip in the else if statement under UCID. Lines 228 - 236.

Checklist Items (3)

#1 Client code that captures the command and converts it to a payload

#4 Code screenshots should include ucid and date comment

#5 Each screenshot should be clearly captioned

Completed sendFlip method in client side to create payload type and send over the payload. Under UCID, lines 250 - 254.

Checklist Items (3)

#1 Client code that captures the command and converts it to a payload

#4 Code screenshots should include ucid and date comment

#5 Each screenshot should be clearly captioned

```
Server.java
Server$1.class
ServerThread.class
ServerThrea...
ServerThread$1.c...
sources.txt
.gitignore
build.sh
```

```
249
250         //UCID - ob75 - March 31, 2024
251     case FLIP:
252         try{
253             (currentRoom).setFlip(this);
254         }catch (Exception e) {
255             e.printStackTrace();
256         }
257         break;
258
```

Completed case FLIP for ServerThread side receiving the payload and sending to Room. Under UCID, lines 251 - 257.

Checklist Items (3)

#2 ServerThread receiving the payload and passing it to the Room

#4 Code screenshots should include ucid and date comment

#5 Each screenshot should be clearly captioned

```
TextFX.class
TextFX.java
TextFX$Color.class
Server
Room.class
Room.java
Room$1.class
Server.class
Server.java
Server$1.class
ServerThread.class
```

```
80
81  //UCID - ob75 - March 31, 2024
82  public void setFlip(ServerThread client)
83      {
84          if (Math.random() < 0.5) {
85              sendMessage(client, message:"Flipped a coin: tails");
86
87          } else {
88              sendMessage(client, message:"Flipped a coin: heads");
89          }
90      }
91
92
93
```

Completed setFlip method for Room side to handle the flip and to send a message back out to the room containing the client, the message the flipped a coin and result. Under UCID, lines 82 - 90.

Checklist Items (3)

#3 Room handling the flip action correctly

#4 Code screenshots should include ucid and date comment

#5 Each screenshot should be clearly captioned

COLLAPSE

Task #2 - Points: 1

Text: Explain the logic in how the flip command is handled and processed and how the message flows from the client, to the Room, and shared with all other users

Response:

The flip command is first handled in the client side by having and else if statement in the processClientCommand for when the user types /flip. Inside the else if statement, it will check if the text starts with the command for FLIP (/flip) and will sendFlip() in the try block if it does start with that command. The sendFlip() creates the payload and sends it to the server thread side where in the server thread, it will call the method setFlip() from the room if the case is FLIP. When it reaches the Room side now, a method is created called setFlip() that takes in the client and will go through an if statement inside the method and do a 50/50 flip inside by using Math.random and if that value is less than 0.5, it will send a message back out to the clients in the room stating who did the command and that they "Flipped a coin: heads". Else if that value is more than 0.5, it will send a message back out the clients in the room again saying who did the command and this time the opposite side of the coin which is tails: "Flipped a coin: tails".

Demonstrate Formatted Messages (4 pts.)

COLLAPSE

Task #1 - Points: 1

Text: Screenshot of Room how the following formatting is processed from a message

Details:

Note: this processing is server-side

Slash commands are not valid solutions for this and will receive 0 credit

Checklist

*The checkboxes are for your own tracking

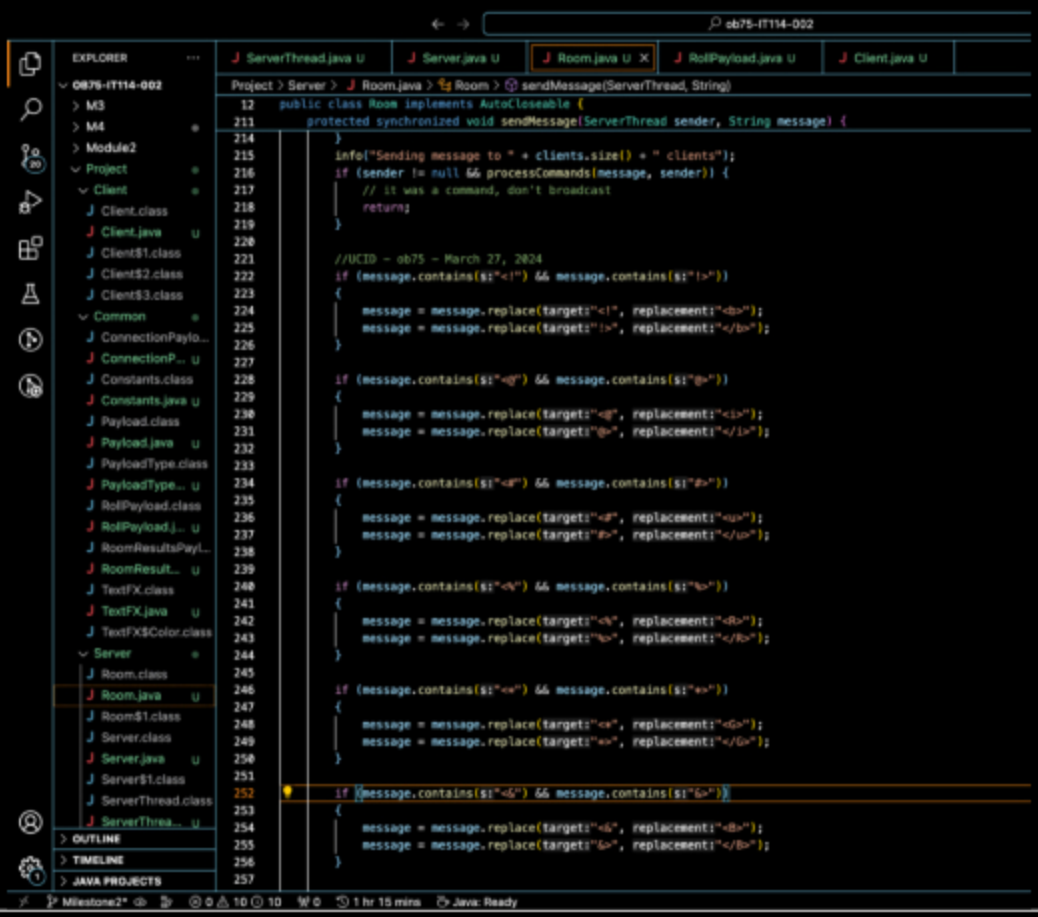
#	Points	Details
#1	1	Room code processing for bold
#2	1	Room code processing for italic

#3	1	Room code processing for underline
#4	1	Room code processing for color (at least R, G, B or support for hex codes)
#5	1	Show each one working individually and one showing a combination of all of the formats and 1 color from the terminal
#6	1	Must not rely on the user typing html characters, but the output can be html characters
#7	1	Code screenshots should include ucid and date comment
#8	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large



Screenshot contains all if statements for processing all types of formats with special characters; <! !> for bold. <@ @> for italics. <# #> for underline. Colors Red<% %>. Green<* *>. Blue<& &> . and replacing them with new tags. Under UCID, lines 222 - 256.

Checklist Items (7)

#1 Room code processing for bold

#2 Room code processing for italic

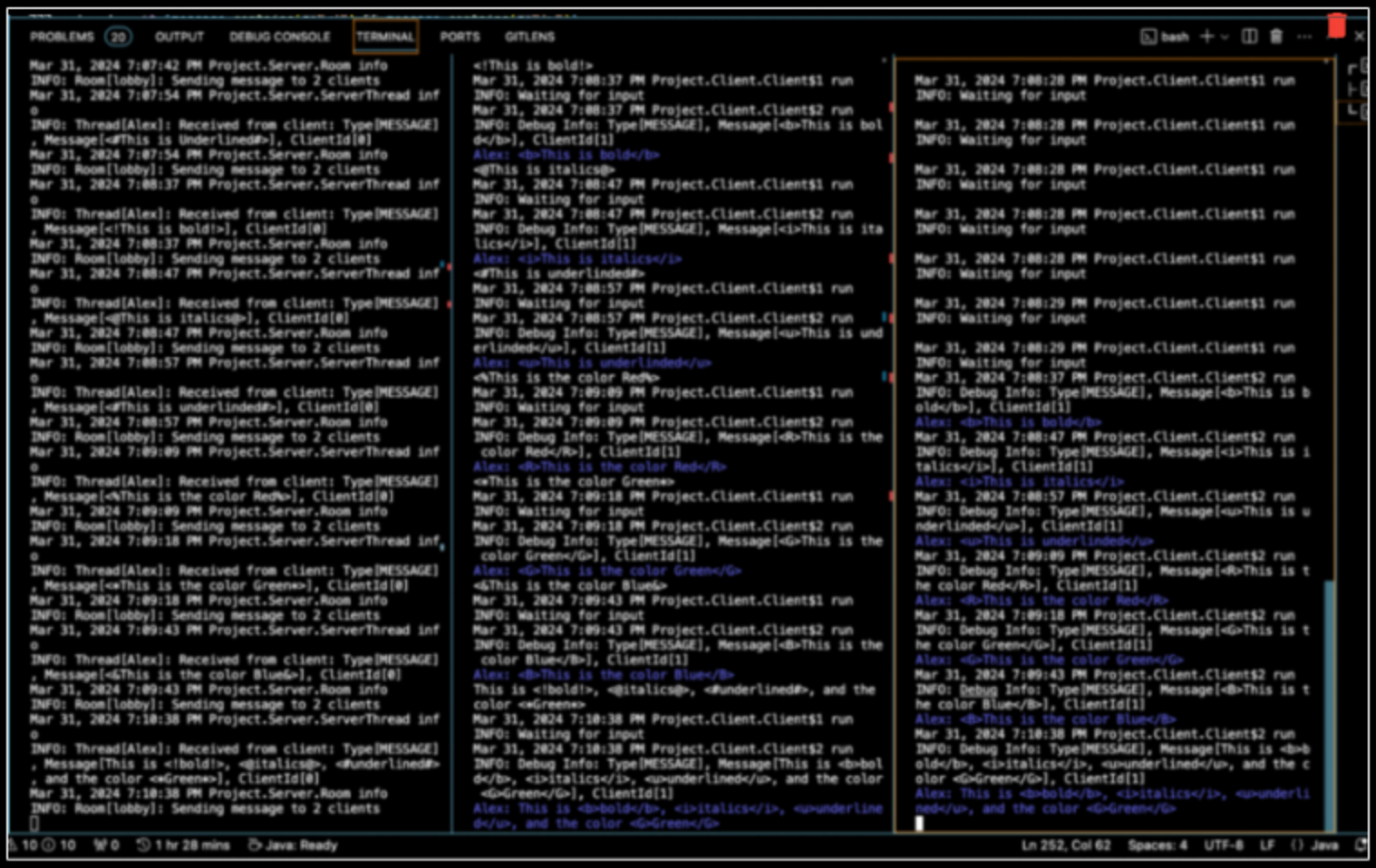
#3 Room code processing for underline

#4 Room code processing for color (at least R, G, B or support for hex codes)

#6 Must not rely on the user typing html characters, but the output can be html characters

#7 Code screenshots should include ucid and date comment

#8 Each screenshot should be clearly captioned



Terminal output showing client typing in command and the message being replaced with their respective tags and message being sent to all clients.

Checklist Items (3)

#5 Show each one working individually and one showing a combination of all of the formats and 1 color from the terminal

#6 Must not rely on the user typing html characters, but the output can be html characters

#8 Each screenshot should be clearly captioned

COLLAPSE

Task #2 - Points: 1

Text: Explain the following

Checklist

*The checkboxes are for your own tracking

#	Points	Details

#1	1	Which special characters translate to the desired effect
#2	1	How the logic works that converts the message to its final format

Response:

<! !> for bold translates to output
<@ @> for italics translates to output
<# #> for underline translates to output <u> </u>.
Color Red<% %> translates to output <R> </R>
Color Green<* *> translates to output <G> </G>
Color Blue<& &> translates to output <B> </B>

The way this logic in code works is that if the user types in their message for example "this is <!bold!>", it will check first to see if the message contains those special characters by doing if message.contains() && message.contains() in the room side since it will be taking the messages that are being sent in the room that have the special characters will be the special character, which is <! !> in this example). Since there are two <! being start of the message to check in in the other message.contains() in this example. It will then target the special characters in this message and will replace them by doing message.replace() twice, to get both the start and end of the special character, and replace them with the new tag that is associated with each character, in this case the replacement will be with the start and end wrapping the desired message with . The new output for the example would become "this is bold" since the old special characters were replaced with new tags.

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/OmarBarrera1/ob75-IT114-002/pull/9

COLLAPSE

Task #2 - Points: 1

Text: Talk about any issues or learnings during this assignment

Response:

There were issues on discovering how to process commands for flip and roll and how to change formats of messages according to what special characters use to wrap their text in like bold, or italics, etc. Most of these issues were resolved by learning about them first and creating methods on handling each command with payloads or dealing with formatting messages to wrap them with specific tags that are replacing their special characters that was typed first. Learning these first took the most time but in the end proved to create solution to the problems.

COLLAPSE

Task #3 - Points: 1

Text: WakaTime Screenshot

Details:

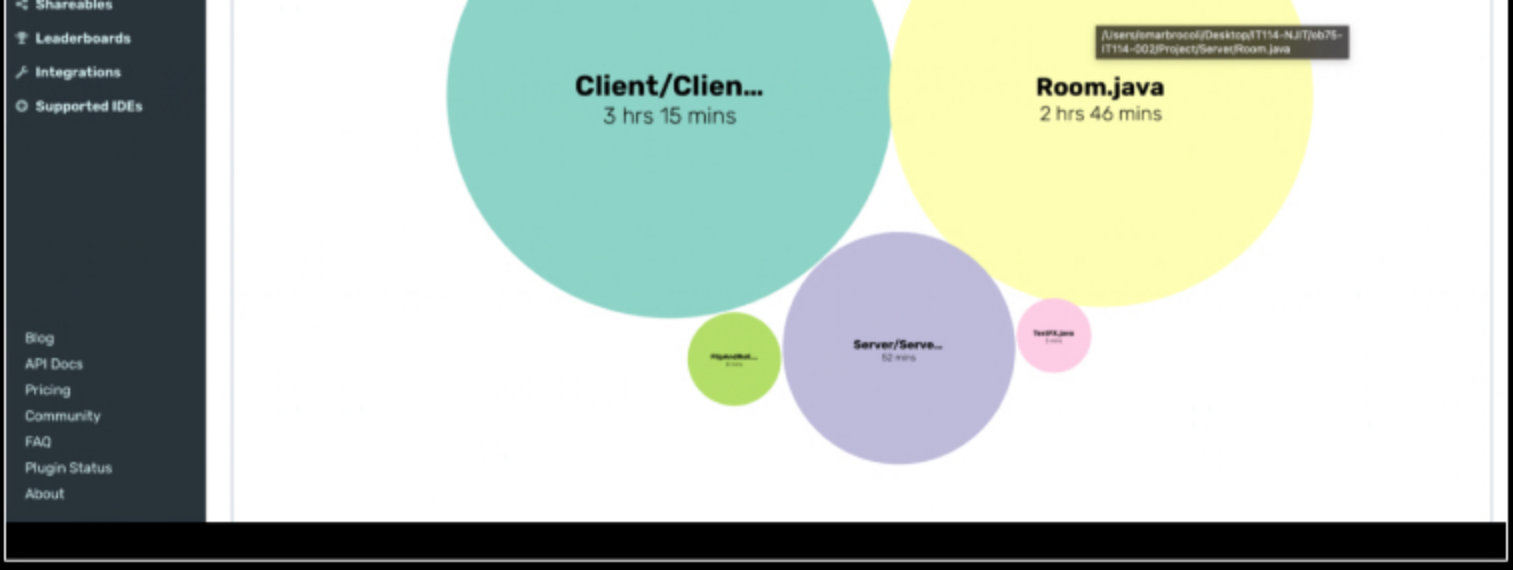
Grab a snippet showing the approximate time involved that clearly shows your repository. The duration isn't considered for grading, but there should be some time involved

Task Screenshots:

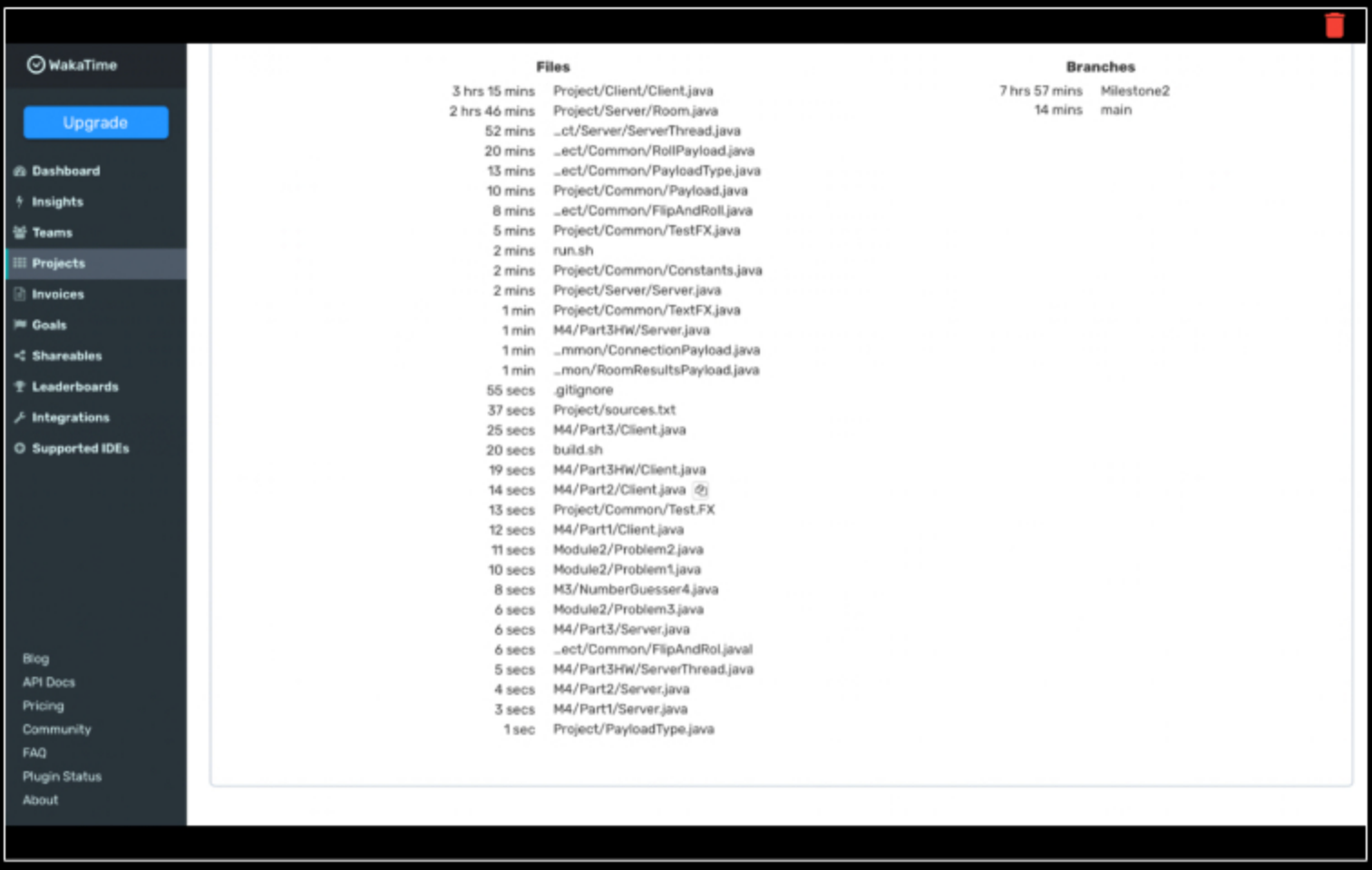
Gallery Style: Large View

Small Medium Large

Top page for Waka Time on project Milestone2.



Middle of page of Waka Time for Milestone2 showing classes worked on.



Bottom of Waka Time page showing all files and branches worked on for Milestone2.

End of Assignment