Data Structures

Murder Mystery - 88 Course Points

Welcome dear debugging detectives, to a night steeped in mystery. The Mansion awaits, ready for its guests, filled with eerie rooms and items. As the sun sets, the guests enter, and the storm rages on. But unknowingly, not all will leave...

In this assignment, you will investigate a virtual murder mystery by debugging its code. The code simulates a generic board game, in which characters move through a mansion and interact with items while being pursued by a rogue murderer. Completing this assignment will help you develop your skills related to using the debugger and understanding class structures.

Check our Programming Assignment FAQ for questions about VSCode, Java, the terminal and more.

Start your assignment early! You need time to understand the assignment code and to become comfortable with the VSCode debugger.

The assignment has two components:

1. Submit your auto-generated "Answers.out" file to Autolab (85 points).
    - You do not need to write any code to complete this assignment.
    - To generate this Answers.out, simply run the MurderMystery.java driver and answer the questions asked after the game runs.
2. Reflection (3 points) submitted through a form. Reflection Link
    - Submit the reflection **AFTER** you have completed the coding component.
    - Be sure to sign in with your RU credentials! (netid@scarletmail.rutgers.edu)
    - You cannot resubmit reflections but you can edit your responses before the deadline by clicking the Google Form link, signing in with your netid, and selecting “Edit your response”

Overview

It's a dark and stormy night. The wind howls through the trees as lightning flashes across the sky. Through the fog and out of the shadows, it appears...the Mansion.

You are tasked with investigating a murder mystery, by debugging certain aspects of the simulation found within the Mansion class.

- The Mansion class holds a grid of unique rooms, a cast of 6 characters (including one murderer), and a slew of items.
- The rooms, players, items, player decisions, questions, and more are all randomly determined by the netID entered in the MurderMystery driver. This means that each mystery is unique, and must be solved differently.

When you run MurderMystery, you will be asked a series of questions about the nights events. Use the VSCode debugger to find the answers to these questions, enter them via the terminal, and then submit your Answers.out file to AutoLab.

You must use your netID when running. Your NetID is what you use to log into Rutgers systems like Canvas -- don't confuse this with your RUID (the nine-digit number). If you submit a netID that is not yours, you will receive a 0 for the assignment. Make sure you enter your netID correctly.

*Do not modify code related to game behavior for the same reason. If you use print statements or write comments, it will shift line numbers -- use the UNedited Mansion.java file for these questions. See the **Debugging Guide** below.*

Implementation

Overview of files provided

- **Mansion.java** is the "game board" where the Murder Mystery takes place.
    - Important attributes are:
        - Room[] roomMap - 2D array of rooms that make up the mansion
        - Person[] players - array of the 6 characters in the game
        - int time - the current time, a multiple of 5. Equals minutes past 6:00pm.
            - i.e. time = 175 is 175 minutes past 6, or 8:55pm.
    - The method nextTurn() runs a single turn of the game (equaling 5 in-game minutes)
        - Information about this method's logic is contained in the code's comments
        - In a turn, players will each have a chance to move and pick up/drop items.
            - The murderer will move last each turn, and attempt to murder the other players.
        - The method will return false if the game has ended, and true otherwise.

You do **not** need to debug any of the following helper classes to solve your questions.

- **MurderMystery.java** is the Driver used to run the game and print the story:
    - When run, the driver will ask for a netID. Enter **yours**. If it is not your netID, you will receive a zero.
        - Then it will ask to print a story intro. After, it will print a short ending.
        - It then asks 10 Questions, which you can answer via the terminal. This will generate an Answers.out file you submit to Autolab.
    - *See the "How to Run" Section below for information on running the game*
- **Person.java** represents a character in the game.
    - Each character has attributes relating to their name, if the player is alive, current position/dice roll, and more.
    - The Person class has methods that implement movement decisions for players, which is different for innocent and murder.
- **Item.java** represents an item found within the mansion.
    - Every possible room has an associated item. After the mansion is generated, a set number of items will be randomly chosen and placed in their corresponding rooms.
    - Each item has attributes relating to its name, whether it's a murder weapon or not marked, what room it belongs to, and more.
        - If an item is used in a murder, it will be marked true, and players will no longer pick up that item.
- **Room.java** represents a room within the mansion.
    - All players start in the same room, "The Foyer"
    - Each room (*except The Foyer*) has a corresponding item that may or may not spawn inside it.
- **Multiple input files** are included, which are used to print out intro/ending text, and generate random player/room/item names. **Do not modify these, as Autolab expects them to be untouched.**
- **Answers.out** will be generated in the project directory upon running the **MurderMystery** driver and answering the questions. **"Answers.out" is the only file you submit to Autolab**.
    - **DO NOT MODIFY "Answers.out" BEFORE SUBMISSION**
    - **Only use YOUR netID when running the Driver**

How To Run your Murder Mystery

Running the game:

Run MurderMystery.java via the VSCode run button, either with or without the debugger.

- When run, the driver will ask for a netID via the terminal. Enter **your netID**. If it is not your netID, you will receive a zero.
  - Then it will ask to print a story intro. After, it will instantly simulate the game and print a short ending.
  - It then asks 10 Questions, which you can answer via the terminal. This will generate an Answers.out file you submit to Autolab.
  - Enter all names exactly as spelled in the Person/Room/Item objects. Enter any times as X:YYpm.
- *See the "How to Run" Section below for information on running the game*

Debugging Guide

The first step to debugging is understanding the program that you want to debug. Especially make sure you understand how the Mansion class stores people/items in its code, as well as in its rooms.

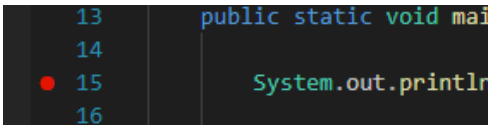
View the Debugging Guide for additional info on using the VSCode debugger.

Or watch watch this video. for an additional debugger walkthrough. Note: This showcases a terminal "number of turns to run?" feature not included in the game.

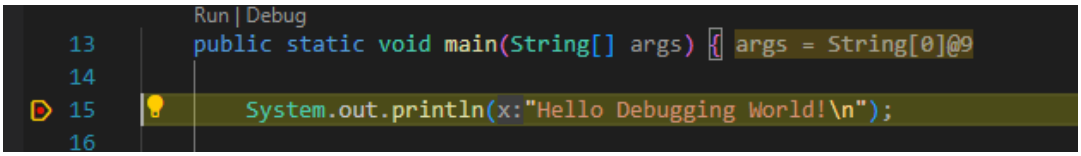
Refer to our Programming Assignments FAQ for instructions on how to install VSCode and the related Debugger extensions.

Generally, to debug your program you will set breakpoints in the nextTurn() method in Mansion.java, and use the "Debug Java" feature in VSCode to run in debug mode. VSCode's Debug Mode has the following main features.

- Setting Breakpoints:
  - To set a **breakpoint** on a line, click to the left of the line number. A red circle should appear where you click.



- - Now, when you run the program in Debug mode, the program will stop before executing that line. It will highlight the line to indicate that it has NOT been ran yet.



  - You can also set **Conditional Breakpoints**, which only pause IF the given condition is true at that time.
    - These are VERY useful, and can be used to answer most of you questions! (but are not needed for all questions)

While paused on a line, the debug toolbar will appear with the following options:

- Continue  – Continue the program from where it is paused until it hits a breakpoint or the program ends. Also functions as a pause button.
  - Useful to jump between breakpoints, or evaluate the same breakpoint(s) as the program executes them multiple times.

- Step Over  – Execute only the current line. Useful to investigate areas directly around your breakpoints

    - Useful to investigate a methods code, by setting a breakpoint before/in areas of interest, and “stepping over” the lines to see how variables change.
    - When encountering a method, this will execute it all instantly instead of line by line, aka “stepping over” them. This is not the case when you are already inside the method.

The **Variables** window located to the left under the **Run and Debug** menu will show you the values of any variables that are in scope on the paused line. You can use this to stop program running at a certain point, then view the state of the game via the variable values.

- You can right click any specific variable, and select “Add to Watch”. This will add it to the “Watch” window, which functions the same as the Variable window. This allows you to pay attention only to specific variable or object values (such as time, a certain item, etc.)

Experiment with using this window, as it is crucial to good debugging. Understanding how variables and objects are nested is important to finding your solutions!

Debugging with Print Statements:

- You can add print statements throughout the Mansion code to output the values of variables.
- You can combine these with the existing if-statements, and your own if-statements, to only print given certain conditions.
- These are useful but can lead to NullPointerExceptions if you are trying to access data that does not exist. You can account for this in your if-condition.
- These can also be more limited than breakpoints, since viewing the variables directly as they change will always give you more information, and the debugging toolbar will always give you more control.

You will need to combine your debugging/detective skills with your knowledge of how the game runs to answer the questions given in the MurderMystery driver.

View the Debugging Guide for additional info and pictures on using the VSCode debugger.

Or watch this video for an additional debugger walkthrough. Note: This showcases a terminal “number of turns to run?” feature not included in the game.

Getting Started

After you read the above description,

1) Run the MurderMystery driver in story mode until the game ends, and **write down your questions somewhere!**

- If an item was not touched, enter "Untouched" for location or "6:00pm" for the time. If a player was never murdered, enter "Alive"
- Enter all names exactly as spelled in the Person/Room/Item objects. Enter any times as X:YYpm.
- Pay attention to which attributes are decided upon mansion generation! Try to answer these first.

2) Locate important lines of code which may relate to your questions.

- Most of these lines will be within the nextTurn() method in the Mansion.java class. This code related to picking up/dropping items, player movement, and other game actions.

3) Add breakpoints, conditional breakpoints, and print statements. You can answer all your questions in one run if you place breakpoints correctly.

- Use conditional breakpoints to narrow down **when** you are checking.
- Combine this with **where** you place the breakpoint to instantly find question answers.

    - i.e. Set a normal breakpoint at the start of the method to check room count, item count, murderer name, and more. Set a breakpoint on the two "return false" statements in nextTurn() to stop at the end of the game.
    - i.e. Set a breakpoint at the .kill() call, with a condition to check if it's killing a certain player.
    - i.e. Set a breakpoint everywhere player.pickUp(item) happens, with a condition for item.getItemName() to equal a certain item name.
    - i.e. Set a breakpoint at the end of the method, to find out where players are/moved.

4) Rerun the MurderMystery driver in story mode and answer the questions, then submit the generated Answers.out file to Autolab.

- You **NEED** to answer the questions for **your** netID. If you submit an incorrect netID you will receive a 0.
- Pay attention to how you spell the names of characters, items, and rooms. It matters for grading. This includes periods and spaces.

Implementation Notes

- **DO NOT MODIFY THE "Answers.out" FILE BEFORE YOU SUBMIT IT**
- You ***MUST*** answer the questions for *your* netID. If you submit a netID that is not yours, you will receive a 0.
- YOU MAY add code to Mansion.java, as long as you do not modify in-game behavior. Again, if you write comments or print statements, it will shift specific line numbers. Use an unedited Mansion.java file for line number questions.
    - Modifying game behavior may cause you to find different answers than Autolab.
    - You may mainly add if-statements to check logic for print statements. Do not use **break**, **continue**, **return**, or other similar keywords.
- DO NOT add/rename the project or package statements.
- DO NOT change the **MurderMystery** class or other helper classes.

VSCode Extensions

You can install VSCode extension packs for Java. Take a look at this tutorial. We suggest:

- Extension Pack for Java
- Project Manager for Java
- Debugger for Java

Importing VSCode Project

1. Download MurderMystery.zip from Autolab Attachments.
2. Unzip the file by double clicking.
3. Open VSCode
    - Import the folder to a workspace through **File** > **Open Folder**

Before submission

Make sure you have not modified Answers.out. If you submit a netID that is not yours in this file, you will receive a zero.

Collaboration policy. Read our course collaboration policy here.

Submitting the assignment. Submit *Answers.out* separately via the web submission system called Autolab. To do this, click the *Assignments* link from the course website; click the *Submit* link for that assignment.

Getting help

If anything is unclear, don't hesitate to drop by office hours or post a question on Piazza.

- Find instructors office hours *here*
- Find tutors office hours on Canvas -> Tutoring
- Find head TAs office hours *here*
- In addition to office hours we have the Coding and Social Lounge (CSL) , a community space staffed with lab assistants which are undergraduate students further along the CS major to answer questions.

By Colin Sullivan and Steven Chen