Week 3 Git Bundle Submission

Raphael Talento and Jacob Tuttle

Table of Contents

1. Chapter Project Report

1.1 Introduction

This project overview details the development of an Interactive Floor Plan Designer aimed at architecture students, interior designers, and hobbyists interested in space planning and design. The project involves enhancing a basic paint application framework to create a Java Swing application enabling users to design, visualize, and edit floor plans for rooms, houses, or offices.

Key Features:

1. Design Elements Palette: A toolbox/side bar that populates based on which option is selected. For example, furniture icons for easy placement on the drawing canvas will appear when furniture is selected.
2. Drawing Canvas: A grid-based workspace allowing users to draw walls and place other design elements with simple click-and-drag actions.
3. Element Manipulation: Tools for selecting, moving, rotating, and resizing placed elements to adjust their orientation and dimensions.
4. Save/Load Functionality: Users can save floor plan designs to a file and load them for future editing or sharing.

In essence, this Interactive Floor Plan Designer provides a user-friendly platform for creating and customizing floor plans efficiently and intuitively.

1.2 Implementation Details/Methodology

Java Swing Framework:
The project utilizes the Java Swing framework as the foundation for the Interactive Floor Plan Designer. Leveraging Swing's extensive GUI components and event-driven architecture, the application provides a responsive and intuitive user interface.

Class Hierarchy:
A well-defined class hierarchy is established to ensure the organization, scalability, and maintainability of the codebase. The class structure is designed to encapsulate distinct functionalities and promote code reusability. Key classes include:

- App: Serves as the main window of the application, integrating the menu bar, toolbox, and drawing canvas.
- MenuBar: Serves as a way for users to select from a series of options. Will be contained in the App and houses several options for the user. These options are menus in itself, so the classes are called "__Menu.java"
- FileMenu: Upon clicking, will show users a drop down menu that has a save and load option. It will be contained within the MenuBar class.
- DrawMenu: Upon clicking, will show users a drop down menu that has a wall, window, or door option. Users now will be able to click and drag across the drawing panel and draw segments. It will be contained within the MenuBar class and these segments are of DrawingPanelSegment class.
- FurnitureMenu: Upon clicking, will populate the ToolBox class with furniture buttons. When clicking a furniture button, users will be able to click on the drawing panel and add that furniture sprite. Will be contained within MenuBar.
- RoomMenu: Upon clicking, users will be able to click and drag across the drawing panel and create squares that are rooms. It will be contained within the MenuBar class and these rooms are of DrawingPanelRoom class.
- EraseMenu: Upon clicking, users will be able to click and drag across the drawing panel to erase elements they previously put, like segments, rooms, and furniture. It will be contained within the MenuBar class.
- DrawingPanel: The main way users will click and drag segments for walls, doors, and windows. Will also contain furniture that the user can put
- DrawingPanelSegment: A class that stores a starting point and ending point for a line. Necessary to store, as the panel must remember and repaint segments.
- DrawingPanelRoom: A class that stores a starting point and ending point for a rectangle, which is in essence a box. Necessary to store, as the panel must remember and repaint rooms,
- ToolBox: Manages the toolbox/sidebar containing design elements for user selection and placement. Will only open upon certain options from the MenuBar.

- SaveLoadHandler: Manages the save load functionality of the application. All corresponding logic is encompassed in this class to better organize the structure of our program

Design Patterns:
The implementation incorporates various design patterns to address common software design challenges and promote maintainability and extensibility:

- **Factory Method Pattern:** Utilized for creating instances of design elements (e.g., walls, doors, windows) based on user input or predefined configurations. This pattern enhances flexibility by allowing subclasses to determine the type of objects to be instantiated.
- **Observer Pattern:** Implemented to manage communication between components within the application. For instance, the drawing canvas observes changes in the design elements palette to update the available options for placement.
- **Composite Pattern:** Employed to represent complex structures, such as a floor plan composed of interconnected elements (e.g., rooms, walls, furniture). This pattern facilitates the treatment of individual elements and composites uniformly, simplifying operations across the hierarchy.
- **Singleton Pattern:** Applied to ensure that certain classes, such as the FileManager responsible for managing file operations, have only one instance throughout the application's lifecycle. This pattern guarantees global access to shared resources while preventing unnecessary instantiation.

By incorporating these implementation details, including the utilization of design patterns, the Interactive Floor Plan Designer achieves a robust, maintainable, and extensible codebase, providing users with a seamless and efficient design experience.

The design patterns are still a work in progress and the descriptions above may probably be changed. We are targeting functionality, then refactoring to fit proper design patterns.

1.3 Testing and Evaluation

Due to the lack of experience and time, we tested user actions through the UI. We acknowledge that testing through the UI is suboptimal and instead something like QuickTestPro or Silk would be better.

Testing Draw

Draw was tested by drawing in all possible directions that is supported by our application. This is diagonal, horizontal, and vertical drawing. In addition to this, we tried all possible ways to draw these lines, such as drawing horizontal lines from left to right vs right to left, diagonal going top right vs bottom left, etc. We did this for every draw option which are walls, doors, and windows. All were successful. We also tried drawing into the menu bar or the toolbox (so they would clip out of the drawing panel) to see if lines would draw, if we could erase these lines, or overlap these application elements. They didn't overlap and appeared the same/erased the same as segments fully appearing on the drawing panel, meaning they were implemented successfully.

Testing Room

Room was tested by drawing a box from top left to bottom right, top right to bottom right, bottom left to top right, and bottom right to top left. Testing in this way actually revealed some bugs and led to some fixes in our code, as there was improper logic when trying to draw boxes bottom right to top left and top right to bottom left. Similarly to draw, we tried drawing rooms where a portion of the room was missing from the drawing panel (going into the menu bar or tool box). These rooms were able to be drawn with our code and similarly were able to be erased.

Testing Furniture

To be done. Furniture is still undergoing development.

Testing Erase

Erase was tested by erasing all elements already talked about, like the different drawing segments, rooms, and furniture. These appeared to work as soon as the user dragged their mouse on an element. An interesting thing that was caught where rooms would erase if users left clicked inside a room, rather than the room boundaries itself. The team discussed that rooms should erase only when the literal walls are touched, as users should be able to erase elements inside the room without getting rid of the room. **[still need to erase furniture]**

Testing Save and Load

We tested saving and loading a different set of drawing panel elements. In other words, we tried saving solely drawing segments, rooms, furniture, combinations of them, etc. and would see if they would be able to be loaded properly. These were able to be done, and not only that we tried drawing and erasing onto them and saving and loading them again to see what would happen.

Saving and loading a second time through this way was successful. Lastly, we tried loading a file that was saved on a different computer. This also passed. **[still need to save and load furniture]**

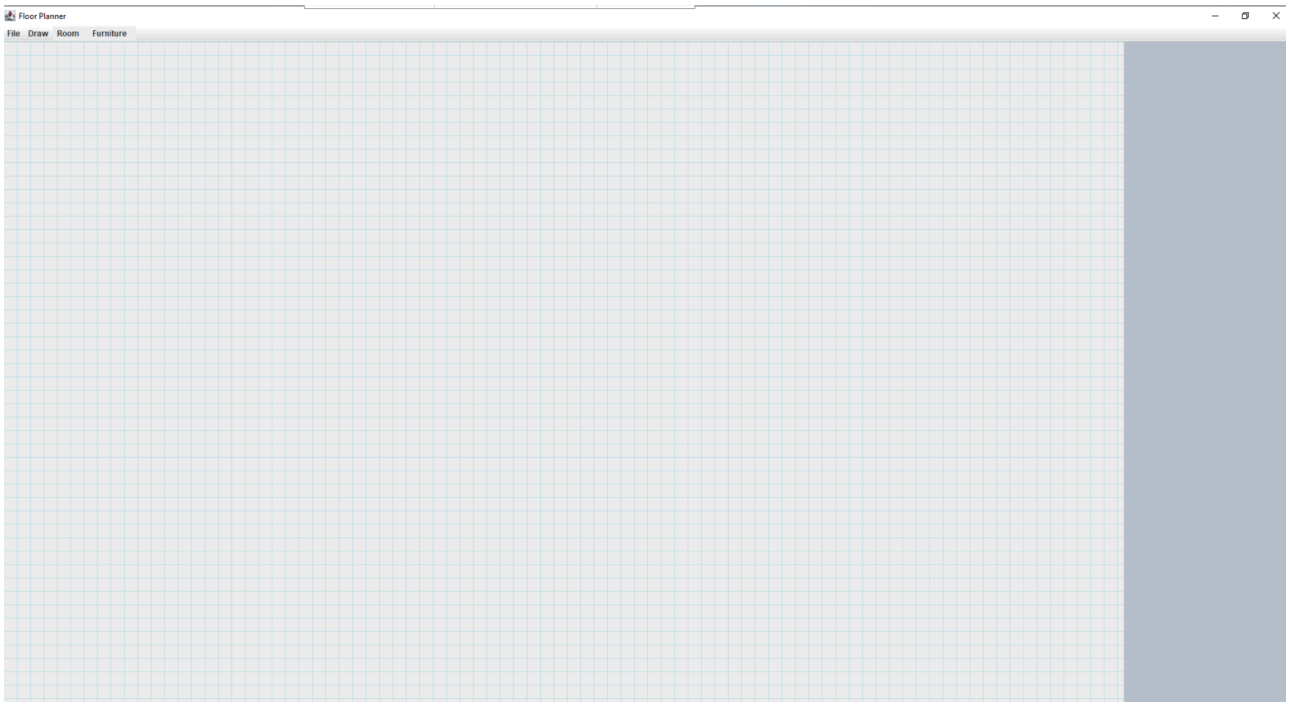
1.4 Documentation of Project Lifecycle

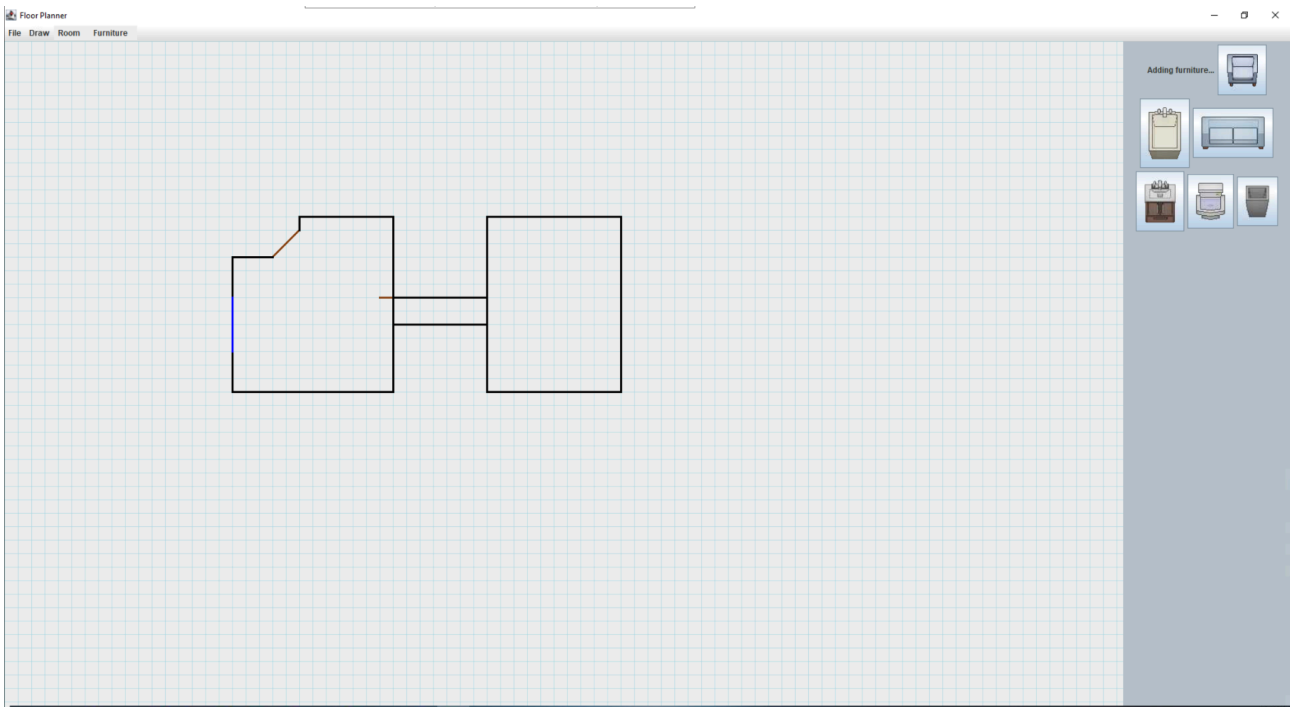
2/12 - Uploaded base files on Github. Refactored file names and created pom.xml
2/13 - Did tests with sprites (to see if these could be found through a path) and created the initial grid system with drawing walls, doors, and windows.
2/17 - Name refactoring.
2/20 - Added all buttons to menu bars and made classes for all options for the menu bar and drawing panel.
2/22 - Further optimization with the grid system. Draw functionality is essentially complete but could use refactoring
2/24 - Added toolbox labels as well as furniture buttons.
2/25 - Can now right click the drawing panel to clear everything. Attempts to draw furniture on the drawing panel
2/28 - Can now draw rooms which are boxes.
3/3 - Lifted mouse listener logic and various boolean logic out of DrawingPanel class (heavy refactoring of DrawingPanel class) and fixed some toolbox label header issues, making the label align at the center at all times and the furniture buttons to be start on a different axis. Implemented some save and load functionality through serializable, but still contains bugs.
3/5 - Started to implement furniture. Nothing on the screen yet, but beginner logic added to FurnitureMenu
3/9 - Fixed save and load and encapsulated all relevant logic under SaveLoadHandler. In addition, added an erase option on the toolbar and you can now erase rooms and segments.
3/11 - Refactoring. Worked on report and testing.
3/11 - Furniture sprites now able to be loaded onto the application. However, still need to set up data structure so sprite persists upon calling repaint.
3/12 - Furniture sprites now persist upon calling repaint. Can now right click to rotate them.

1.5 Documentation of GUI Development

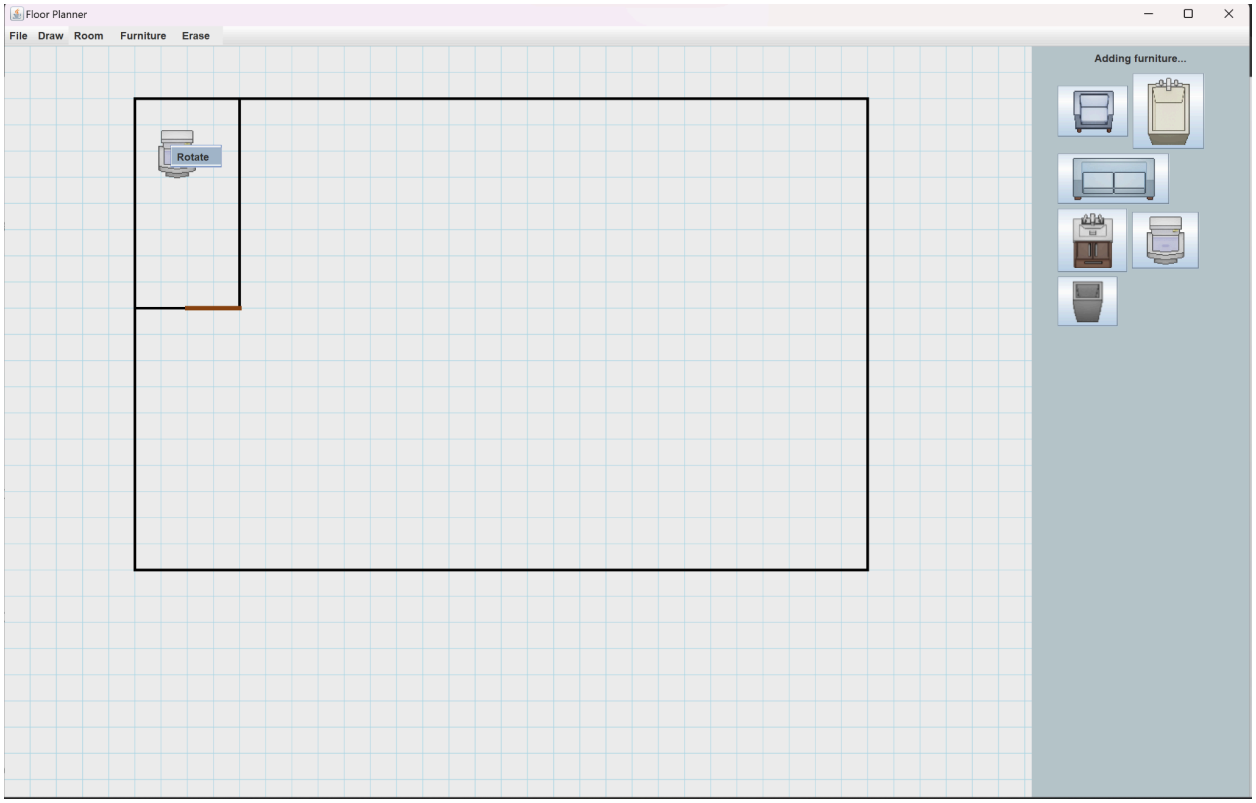
The GUI is different from our initial sketch (refer to Chapter Software Design) but only slightly. Instead of options being bolded, we now have a header/text in the toolbox that users can read.

2/29 GUI Screenshots





3/12 GUI Screenshot



Notice how the toolbox header is now better aligned when compared to the 2/29 GUI screenshots. In addition, the door segments are now thicker than regular wall segments and we can rotate furniture.

1.6 Results and Discussion

Some limitations with testing was that on different devices/laptops, the preview drawing segment/room would sometimes not update properly and there would be two lines. The idea of the preview line is to show the line users will end up drawing when letting go of their left click, but it would be unclear on certain devices. We predict it's probably due to the refresh rate of certain laptops or perhaps the touch screen capabilities of some laptop hybrids. The solution to this could be to optimize our code to better fit these slower devices, but there we may end up trading off some speed/reactivity when our application is run on stronger devices. This is something to be looked into, but we really just want to focus on functionality as a number one priority.

In addition, there was discussion of allowing users to pick flooring for certain rooms in order to see which color combinations they liked with furniture. The problem with this is that if they use our application for aesthetic purposes, the amount of furniture we need to support can dramatically increase. In other words we need to consider chairs, tables, counters, etc. of different colors and sizes, along with supporting a wide variety of flooring. Although this is definitely possible, we felt like this route would be off track for this project, as we would be delving a bit more into the aesthetic, GUI realm instead of our functionality, OOP, and design-oriented goals. It would also take a lot of resources developing sprites which is not really in the scope of the course and project. Our application should simply be a floor planner, so users can get a feel for a framework for how to design their building with different furniture and room sizes.

[Add some things we'd like to add or perhaps talk away here]

1.7 Conclusion

This project was definitely a good introduction and experience into software architecture. Being new to Java Swing, the team was initially intimidated and anticipated we would undergo a deep learning curve. However, it was admittedly not that bad and ChatGPT did resolve a lot of the annoying logic, minor errors, and a lot of questions we had regarding the framework. The floor plan designer, although not the most technical domain, required us to delve more into the GUI of Java Swing, like repainting the drawing panel and how to store/keep what was being drawn on the drawing panel so that it could be saved/loaded for later. Because the floor plan designer is more GUI oriented, we admittedly redesigned some parts multiple times or focused on parts just so it can look more visually appealing (like the toolbox and the size of the menu bar buttons which took some time to fix)

Some improvements we'd like to make is a stronger use of design patterns, specifically a stronger relationship to the composite or the decorator pattern. Right now we kind of have a composite pattern going on (objects being housed inside other objects and being added onto during program execution), but we feel there could be a better job in separating out the components more. The implementation feels too interconnected between classes and isn't the most scalable. This is because we right now have the drawn elements as an array existing inside the drawing panel as our data structure, which couples the implementation to each of its parts too much. We'd like to achieve looser coupling between our classes and the implementation a bit more so that features can be more readily added and implemented into our existing code. In addition, there are certain GUI things we'd like to do like the size of the buttons running on different devices or trying out different fonts and testing them with users. We'd also like to implement some flooring and more types of furniture if we expand our project to not just be a floor design planner, but also an interior design planner.

[Apply back to class content more here]

[We may combine Results and Discussion and Conclusion sections together. Both are evalustic in nature about what we've done for the project and does have its intersections, but we'll keep it separate for now]

2. Chapter User Manual

2.1 Making your First Project

Step 1: Launching the Application Start by launching the Interactive Floor Plan Designer application on your device.
Upon opening the application, you will be greeted with a clean and user-friendly interface, ready for you to unleash your creativity.

Step 2: Exploring the Application
Take a moment to familiarize yourself with the application. There are three main components: a drawing canvas, a menu bar (located at the top), and the toolbox (located on the left). Elements will only appear in the toolbox on the left based on the option (like furniture). A drop down menu may also appear depending on which option is selected (like file).

Step 3: Creating Your Floor Plan Begin by selecting the Room option or Draw -> Wall
Click on the drawing canvas to place the starting point of your wall, then continue clicking to draw the desired shape of your room. Utilize the grid-based workspace to ensure precision and alignment in your design. Alternatively, you can manually draw walls through the Draw option.

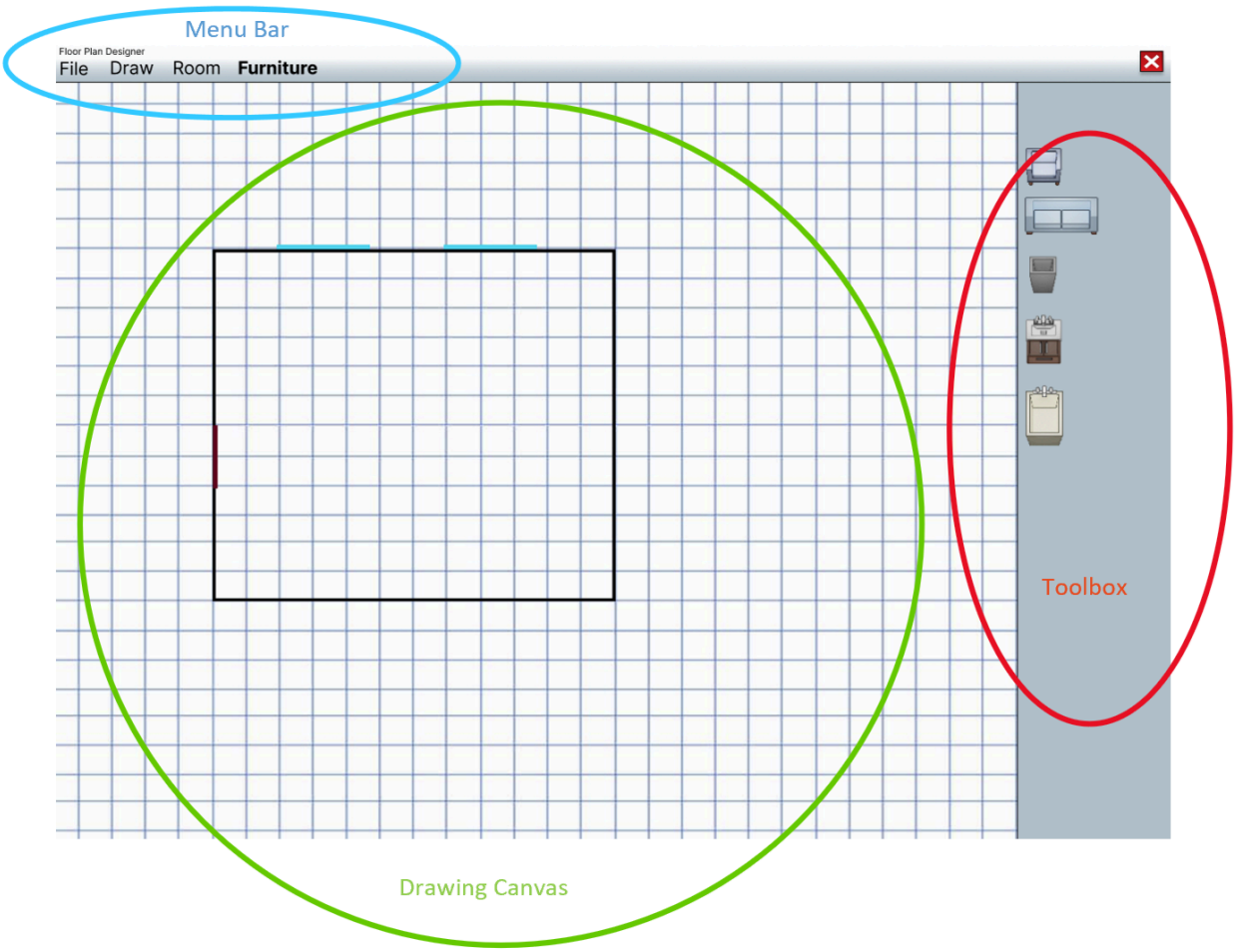
Step 4: Adding Doors, Windows, and Furniture Once you've outlined the walls of your room
You can add doors and windows through the draw option. These can be manually added, similar to how you can draw walls. To add furniture, press the furniture option on the top menu bar. The toolbox on the right will then be populated with various furniture.

Step 5: Manipulating Elements
Need to make adjustments to your design? No problem! Utilize the element manipulation tools provided by the Interactive Floor Plan Designer. Select, move, rotate, or resize elements as needed to achieve the desired layout and aesthetics. **[This feature is still a work in progress]**

Step 6: Saving Your Design
Once you're satisfied with your floor plan, it's time to save your work. Navigate to the File menu and select the "Save" option. Choose a location on your device and enter a filename to save your floor plan for future editing or sharing.

Step 7: Loading Existing Designs
Want to revisit a previous design or collaborate with others? Simply select the "Load" option from the File menu, navigate to the location where your floor plan is saved, and open it within the Interactive Floor Plan Designer.

2.2 Visual Guide: Understanding Components



The **menu bar** is in the blue circle. This consists of several **options.**

The **drawing canvas** is in the green circle. This is where all room drawing and furniture placing should take place.

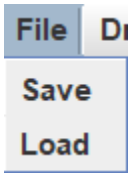
The **toolbox** is in the red circle. This may be empty depending on which **option** is selected.

2.3 Going Over Menu Bar Options

Menubar

This is the menu bar.

File Menu Option

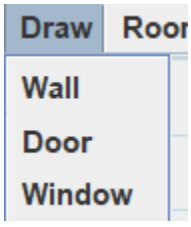


The file menu option has two options: Save and Load

Save - After working on your project for some time, you'll probably want to save your work for later. Press the save option, find a place for your file, give your file a name, and open it later using the load option. The saved file will have a .draw extension.

Load - To open a project previously saved, select this option. Find your project on your computer through the file explorer and you'll see your previously saved work appear on the drawing panel.

Draw Menu Option



The draw menu option has three options: Wall, Door, and Window

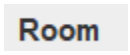
Wall - Left click and drag on the drawing panel to draw some walls. These are straight black lines.

Door - Left clickand drag on the drawing panel to draw some doors. These are straight brown lines that are thicker than walls.

Window - Left clickand drag on the drawing panel to draw some doors. These are straight bluelines that are thicker than walls.

The draw menu only allows for lines that are perfectly vertical, horizontal, or diagonal. In addition, they must be connected to intersection points of the grid.
****We probably will allow users to draw lines of any degree if the start and endpoints are connected to grid intersection points. The issue with this is that doors and windows will never perfectly align with a wall for example, as the only way these can align is if they match the start and endpoints of the entire wall segment. This doesn't make sense as a door should only be a subsection of a wall.****

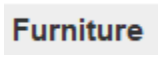
Room Menu Option



The room menu option has no options. Instead, you click to toggle on room drawing mode.

In this mode, left click and drag across the drawing panel. You'll be able to create rectangles which are rooms. You cannot draw straight lines in this mode as these are not rooms.

Furniture Menu Option



The furniture menu option, upon clicking, will populate the toolbox with furniture buttons, as seen on the right. Click on a furniture button to start populating the drawing panel with furniture by left clicking onto the drawing panel.
[**implementation details are still being worked on. Will be updated]**

Erase Menu Option

The erase menu option has no options. Instead, you click to toggle on erase drawing mode. In this mode, left click and drag across the drawing panel. You'll be able to erase segments, rooms, and furniture that has been placed on the canvas.

3. Chapter Software Design

3.1 User stories

1. As a user, I want to be able to move and resize any piece of furniture so that I can make adjustments in my floor plan.
2. As a user, I want to conveniently put rooms and not have to draw straight lines for every wall so that I can save some manual labor.
3. As a user I want to be able to remove walls from an already existing room.
4. As a user, I want to be able to save my work to open it to work on for later.
5. As a user, I want this application to feel like my other desktop applications, being able to live on my desktop and have its own icon, so that it's convenient to find.
6. As a user, I want to be able to use this application regardless of operating system so I can open it on my Windows computer or Apple laptop.
7. As a hobbyist, I want to be able to use this software without having to undergo a deep learning curve so that I can use it easily.
8. As an interior designer, I want the product to have enough features to support my work in industry.
9. As a hobbyist and interior designer, I would like to be able to see measurements on the canvas to have a better idea for how big everything is.
10. As a hobbyist and interior designer, I would like there to be a good range of different types of furniture so that I can experiment with different room layouts for different settings.
11. As a hobbyist and industry designer, I want the product to give me an estimated cost based on square feet and furniture present so I can assess the amount of resources to put in.

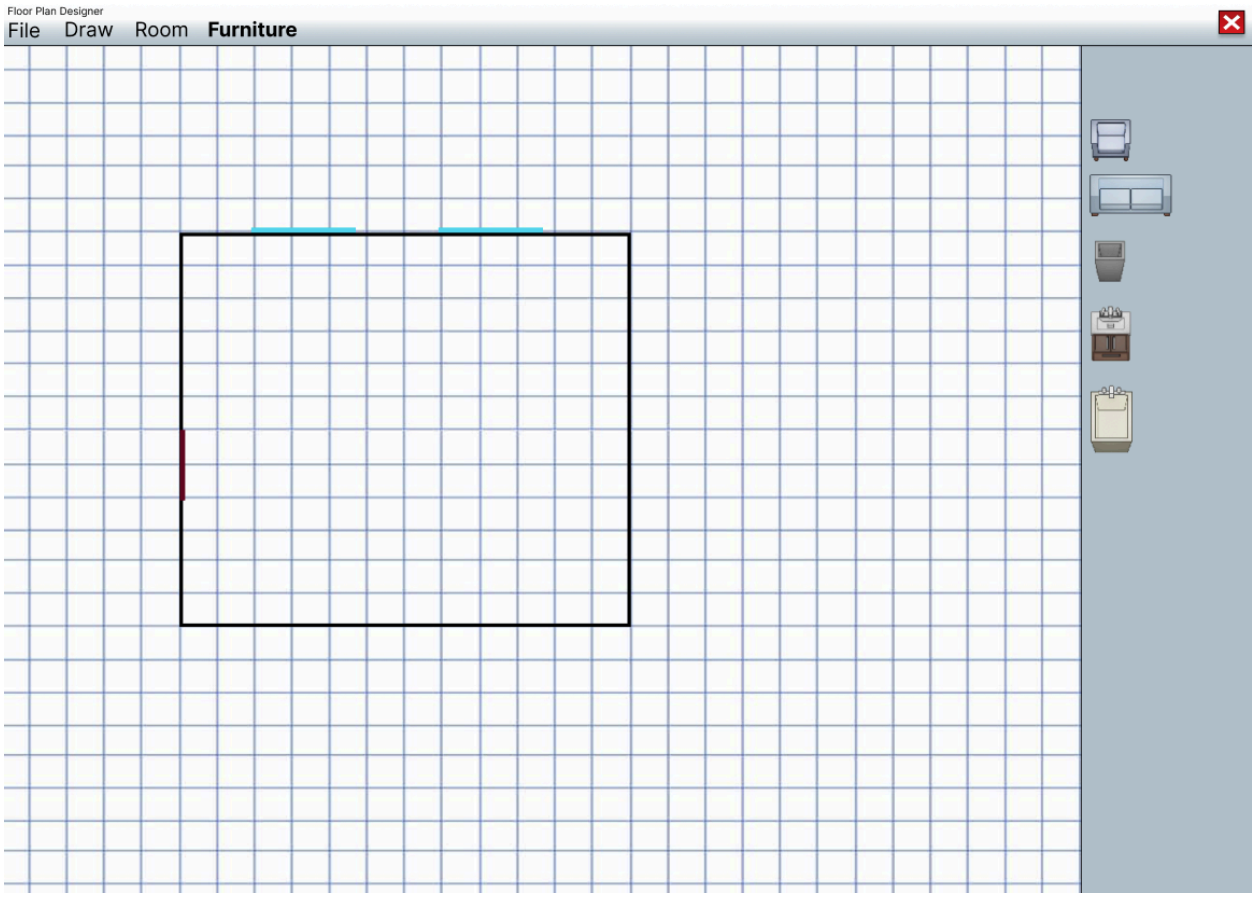
3.2 Architecture Overview

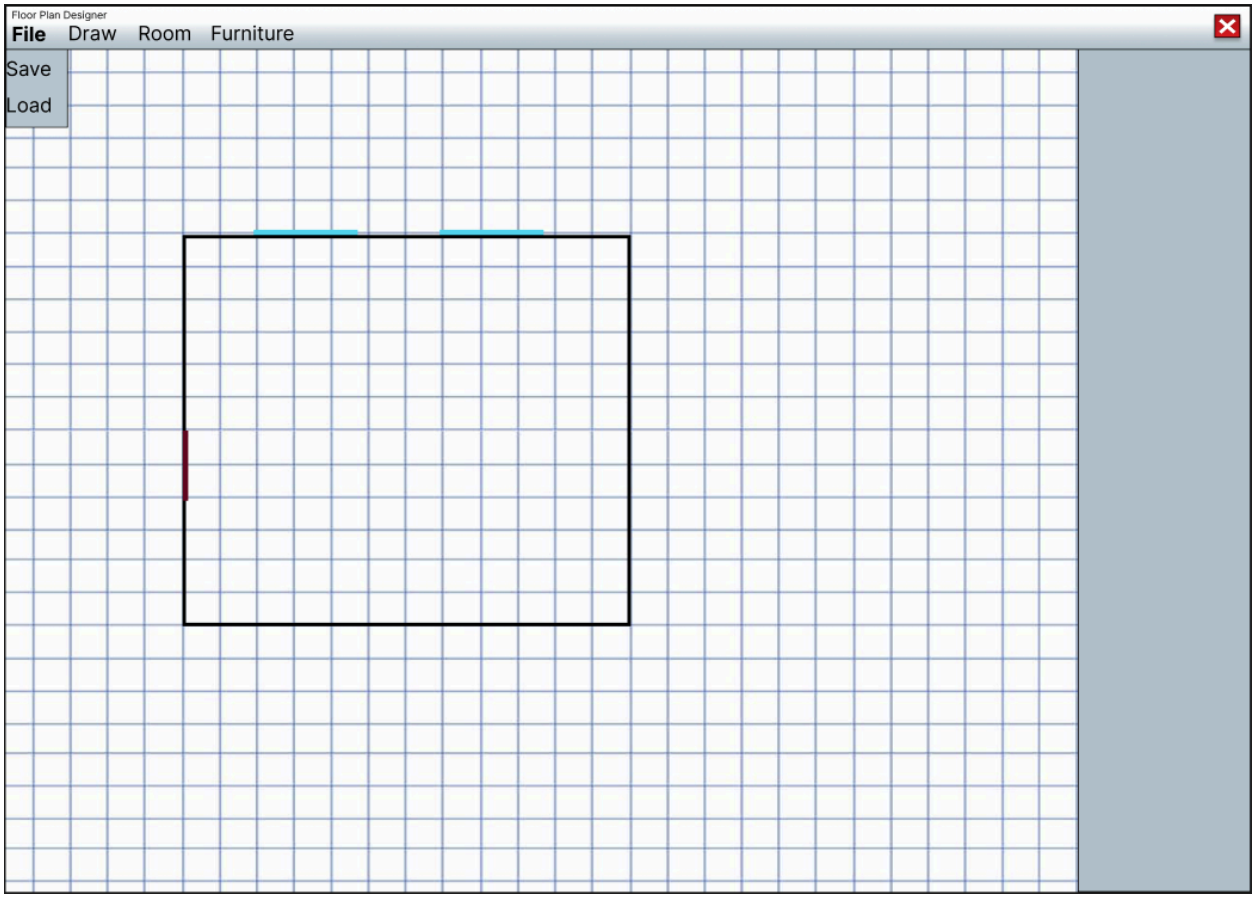
The main screen of the application is the class App, which stores a JFrame that will have various components added to it. The first component is a JMenuBar named MenuBar that has four options on it and is designed in a way that supports future features to be added. It has two JMenus named File and Draw, it also has two JMenuItems Room and Furniture. The various JMenuBar components are linked with the menu bar and then that is added to the JFrame on construction of the menu bar. The menu bar is created following the singleton pattern to maintain one instance of it across the application.

The idea of the architecture is an OOP based approach using Java/JavaSwing. The main class will be the App class. The next layer will be things housed within the App, which are mainly the DrawingPanelClass, the MenuBar class, and the ToolBox class. From then on, we get deeper within each of these 3 components housed within the App, adding classes as necessary. This enables a way to add more features and functionality with ease due to our class based structure. Design patterns are meant to be implemented where necessary to keep design robust, clean, and optimal.

Although our project seems to entirely house things within other things, there are applications of classes that are more meant to be containers of logic. For example, the SaveLoadHandler class is meant to be a container of logic and house the functions related to saving contents to a file, loading contents of a file, and anything related to that. In addition, we have a resources folder which houses the sprites for furniture for our application, where developers could easily add more sprites to this folder if they wish to find more furniture.

3.3 Initial Sketch of GUI

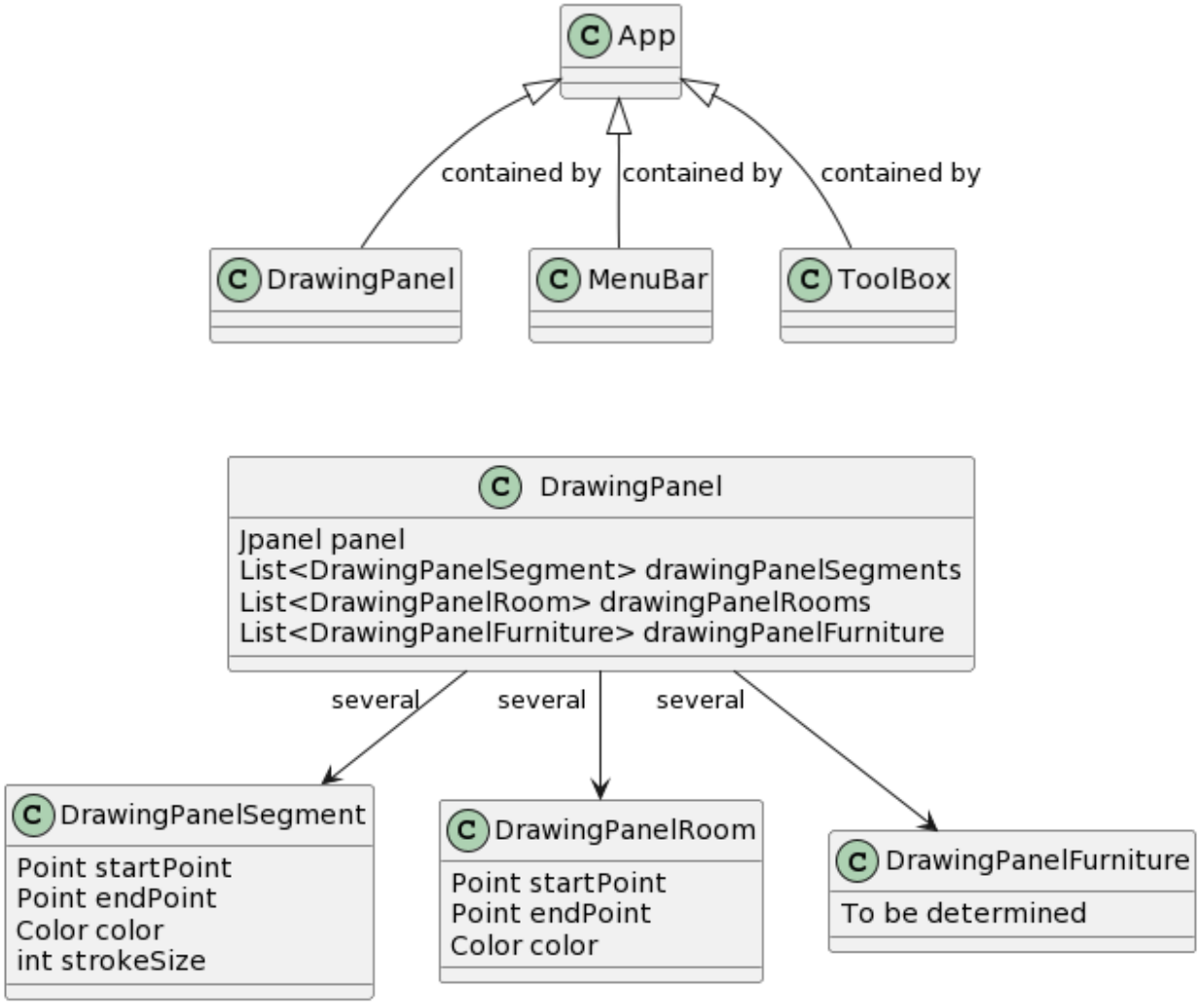


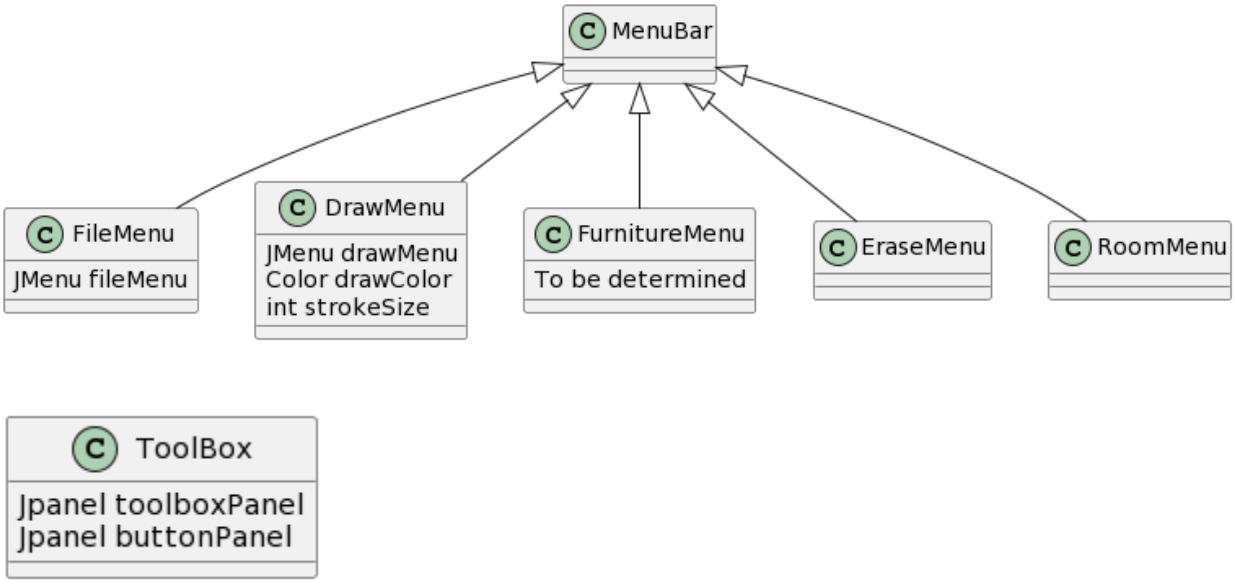


When a certain option is selected (indicated by the bold), the toolbox will be empty and instead a dropdown menu may appear (room will have neither and just allow users to click and drag on the canvas)

3.4 Class Structure: UML Diagrams

The nature of these UML diagrams are to capture important sub pieces of our class structure. It is not meant to show the whole representation of our structure all at once, as that would be quite the mess to read and not intuitive.





We updated our UML diagrams according to our week 3 Git Bundle Submission. However, anticipate this to undergo changes for our final submission.

Appendix A

ChatGPT Logs for Raphael Talento:

All logs are located in ‘project.zip’. These logs are located in the folder ‘chatgpt-raphaeltalento’

Draw Elements FloorPlan214..mhtml - The first chat log I did with ChatGPT relating to the project. The idea was to familiarize myself with Java Swing as well as try to get some basic functionality down. I communicated with Chat GPT to help me with the drawing canvas, specifically to allow me to draw different components indicated with different colors and to have a grid-based looking layout. This log ended up not being really implemented for our first week mostly because it didn't account for any classes whatsoever. It really was just a way for me to get my feet wet with Java Swing for the first time.

Draw room by dragging214.mhtml - This was done on the same day and was a different log because Chat GPT was getting stuck and giving me useless answers. I wanted to be able to make a room on the drawing canvas and do so by holding down my left mouse button and dragging. It introduced and taught me some things with .drawLine and a mouse listener. This once again was just to familiarize myself with Java Swing and is not implemented in the code whatsoever because we wanted to get our basic GUI figured out for the first week. It was also disregarded because Chat GPT didn’t give the proper class hierarchy we wanted.

Project Report Overview217.mhtml - This Chat GPT session aided me in doing project report for week 1. I really just gave the prompts that were supplied on Canvas and did some prompting so that it could generate some base writing I can go off of. Some of it was sufficient and was used, but there were areas that were edited to match our intentions and actual design choices with the project more. For example, the ‘Class Hierarchy’ section it provided was modified to match our purposes more.

Interactive Floor Plan Guide221 - This Chat GPT session aided me in the user manual part for the design document. The response it gave was actually quite good and a lot was used for Chapter User Manual. It was lightly edited to match our project more however.

Screen Resizing Solution221.mhtml - This Chat GPT session was to make it so the application would resize to the user’s monitor size, rather than being a static 400x300 pixels. This was after Jacob’s work, so the code that I supplied to GPT most likely comes from his responses somewhere. This session also gave the proper code to make grid lines appear on the drawing canvas. There were very few edits made with the code it provided.

Resize JmenuItemWidth221.mhtml - This Chat GPT session aided us in implementing a tool box on the right hand side of the screen and proper sizing of the options provided on the menu bar. Chat GPT suggested to user border layout and to setPreferredSize to null and set a maximum size to achieve consistent sizing.

Restrict Horizontal_Vertical Drawing 222 - Initially, I requested help in restricting users to drawing only vertical or horizontal lines, which ChatGPT promptly addressed with code modifications. Next, I needed support for enabling users to draw diagonal lines on top of this functionality, to which ChatGPT provided refined logic for the event handlers. Shifting focus, I sought assistance in implementing file saving and loading functionality for the application's file menu. ChatGPT guided me through the process, offering structured solutions using JFileChooser. However, this needs improvements and there are many bugs with save/load right now.

Furniture Toolbar Setup224.mhtml - This conversation aided me in populating my toolbox with furniture buttons and the sprite directory in the project. we identified that clicking on furniture buttons didn't draw the images on the canvas as intended. To resolve this, we modified the DrawingPanel class to handle drawing functionality. By introducing a new method, addImage(File imageFile), and adjusting addFurnitureWithClicking(File imageFile), users can now select an image from the toolbox and place it on the canvas by clicking. These changes enable interactive placement of furniture images, with potential for further enhancements such as refining drawing features and improving error handling **but still has bugs.**

Draggable Room Preview 229.mhtml - In our interaction, I sought guidance to enable users to draw rooms by clicking and dragging their mouse across a drawing panel. I proposed organizing the drawing logic into a distinct class named DrawingPanelRoom. We discussed methods for managing starting and ending points within this class. ChatGPT assisted me in integrating this logic into the existing DrawingPanel class, particularly within the mousePressed and mouseDragged methods, ensuring users could preview the room as they dragged the mouse. Throughout our interaction, we addressed various challenges such as aligning the preview box with grid lines and ensuring the start point remained fixed at the initial click position.

Swing Application State Serialization33.mhtml - In my conversation with ChatGPT, I learned how to serialize and save data in a Java Swing application for the first time. Initially facing challenges during deserialization, I received guidance on implementing proper serialization techniques. This involved ensuring that custom classes like DrawingPanelSegment and DrawingPanelRoom implemented the Serializable interface. Despite encountering errors such as java.io.OptionalDataException, I learned about version mismatches, corrupted data, and the importance of handling non-serializable objects during serialization.

Concurrent Modification Issue Fix 39.mhtml - Throughout our conversation, I used ChatGPT's assistance to refine my code for loading data into a drawing panel. Initially, I encountered errors like ConcurrentModificationException. ChatGPT identified the issue with concurrent modification of collections and suggested using iterators and the remove() method. I also sought advice on optimizing the file loading method, and ChatGPT recommended modifying it to accept an existing instance of InformationToSaveHandler for efficiency. Additionally, I requested help in implementing error handling for exceptions during the file loading process, and ChatGPT suggested using a try-catch block for graceful error handling.

Drawing Panel Erase Functionality 39.mhtml - Throughout our collaboration, I utilized ChatGPT to refine the erasing functionality within my code, encompassing both rooms and segments. We concentrated on ensuring that erasure occurred precisely when the cursor point intersected with the actual lines of the rooms' rectangles or segments' lines, thereby improving the accuracy of the erasure process. With ChatGPT's guidance, I fine-tuned the logic to achieve this precise erasure, enhancing the overall functionality of the code. Through our collaborative efforts, I leveraged ChatGPT's assistance to develop a more robust erasing mechanism tailored to my specific requirements.

Appendix B

ChatGPT Logs for Jacob Tuttle:

Swing Toolbar file handling217.mhtml - Asked ChatGPT to create a toolbar that supported a file option that had save and load option.

Add MenuBar to HomeScreen217.mhtml - I gave the edited menu bar code it gave me previously and a JFrame home screen that I wrote and had it connect to the home screens JFrame.

RoomMenu Singleton JMenuItem220 - I tried to have ChatGPT create a button on the menu bar instead of it having only JMenus.

Create Drawing Surface220 - Tried to have ChatGPT create a drawing surface out of a JPanel that room schematics would be placed on.

Draw Lines220 - Asked ChatGPT to add the ability to draw lines on the drawing surface JPanel.

Java Swing Image Drawing228 -
Here's a summary of what we covered:
Basic Java Swing Application: We created a basic Java Swing application with a JPanel for drawing images.
Main Method Integration: We added a main method to the application to start the ImageDrawingApp on startup.
Placing Sprites: We allowed users to click on the panel to place sprites (images), which are represented by PNG files.
Right-Click Menu for Rotation: We implemented a right-click menu for rotating placed sprites.
Dragging Sprites: We enabled the ability to click and drag placed sprites to new locations on the panel.
Adding Sprite Selection Button: We added another JPanel with a button containing the sprite image on the right side of the GUI. Clicking this button sets the selected sprite for placement.
Handling Final Variable Issue: We resolved an error related to accessing non-final variables from within an anonymous inner class by using a final array to hold the sprite image reference.
Overall, we created a Java Swing application for drawing and manipulating sprites, including placing, rotating, and dragging them. We also addressed several implementation details and resolved issues encountered during development.

Drawing JFrame with JPanel229 -

Today, we've worked on a Java program for drawing rooms in a JFrame. Here's a summary of what we've covered:
Basic Drawing Functionality: We created a JFrame with a JPanel as the drawing surface. Users can left-click and drag to draw rooms on the panel.
Room Drawing Logic: We implemented the logic to draw rooms as rectangles. When the user left-clicks and holds, a temporary rectangle is displayed while dragging, and when the user releases the mouse, the final room is drawn.
Room Highlighting: We added functionality to highlight individual walls of already placed rooms when the user clicks on them. Only one wall of a room can be highlighted at a time.
Room Construction: We ensured that each room is constructed from four wall segments, and adjusted the logic accordingly.
Thicker Walls: We made the walls thicker by adjusting the stroke width when drawing them.
Highlighting Fix: We corrected the logic for highlighting individual walls of already placed rooms to ensure proper highlighting.
Throughout these steps, we iteratively modified the code to address various issues and implement new features.

Drawing Sprites Troubleshooting312-mhtml

Here's a summary of the key points we discussed regarding the issue of sprites not being drawn on the screen:
Drawing Method Invocation: Ensure that the paintComponent() method in your FurnitureMenu class, responsible for drawing the sprites, is correctly invoked after the selected sprite image is set. This method should be called whenever the panel needs to be redrawn.
Panel Visibility: Confirm that the panel containing the sprites (FurnitureMenu) is added to a visible container, such as a JFrame. If the panel is not added to any visible container, its contents won't be visible on the screen.
Repaint Invocation: After setting the selected sprite image, make sure to call the repaint() method on the panel containing the sprites (FurnitureMenu). This triggers a redraw of the panel and updates the displayed sprites.
Sprite Positioning: Verify that the positions of the sprites are set correctly within the bounds of the panel. If the sprites are drawn outside the visible area of the panel, they won't be visible on the screen.
Check for Exceptions: Ensure that no exceptions are being thrown during the loading of sprite images or when setting the selected sprite image. Exceptions might prevent the sprites from being drawn on the screen.
By addressing these points and ensuring that the selected sprite image is properly set and the panel is correctly redrawn after the image is set, you should be able to resolve the issue of sprites not being drawn on the screen.