Testing Document

Union App

Group 13

Shivam Patel - 216385247

Colin D’Souza - 216283582

Rushabh Chauhan - 215926108

Introduction

The Venn diagram application we created, Union App, is one that strives to provide an application which can be used by a variety of demographics and for many use cases. We have created a variety of testing procedures to ensure that the application will provide maximum stability throughout many use cases.

Testing Procedure

Each test is performed multiple times and on multiple machines and/or operating systems to ensure that such variables are always accounted for.

Test Cases

1. **The application launches and fills the screen**
   The first test simply launches the application and tests to make sure that the window fills the entire screen and does not crash.
   The test ensures that the core of the application launches and compiles, which in turn runs the bulk of the code as well.

2. **The application is able to display and add entries**
   An example entry is added to the list of newly added entries.
   The test simply calls the internal addEntry() function, which adds an entry to the internal registry and displays it on the sidebar.

3. **The application is able to display and add multiple entries**
   Multiple example entries are added, to ensure that the entries are properly spaced and laid out on the screen. This also ensures that the entries are not cut off when there are more than can fit on the screen, and rather shows a scrollbar for the user to use.
   The test calls the same addEntry() function from before, but 100 times.

4. **The hovering of sections changes the color**
   When hovering over sections in the venn diagram, the colors of the venn diagram are changed to be slightly darker to allow users to easily see where they are in the diagram. Once the mouse moves away from the section, the color reverts to its original state.
   The test case controls the mouse and moves the mouse to the various sections of the venn diagram, and stops over each section.

5. **The add entry dialog is able to be displayed**
   A window appears which allows the user to enter text which is added to the left sidebar.
   The test case calls the static method add() in the class VennEntryModalHandler which presents a dialog to the user. The developer then types in "Test" and updates the display options of the entry as they wish. They then push the enter/return button on their keyboard to add the entry, and then verifies that the entry is visible on the left sidebar.

6. **Manual Test**
   Due to the amount of interactions with the system that rely on dragging and dropping, there is a manual section of the test which must be performed by the developer running the tests. 3 example entries are added to the diagram, and the following test cases are performed in the order presented:

   Notes for the instructions below:
   - "Left section" refers to the blue section on the screen on initial launch indicated with "Left" as a title.
   - "Right section" refers to the pink section on the screen on initial launch indicated with "Right" as a title.
   - "Middle section" refers to the gray section on the screen on initial launch indicated with "Intersection" as a title, below.
   - The sections with the bullet point refer to the internal methods called to perform the functions. The other points specified with letters are instructions for the developer.

   1. The developer is able to drag each of the three provided entries to each section of the venn diagram (left section, middle section, right section).
      - The internal drag and drop handlers store the data of the entry being dragged, check that the drop is allowed where the user is attempting to place the entry, and if it is allowed, it moves the entry to the desired section. The internal static function bindDragHandlers() in the VennEntryHandler class and initDropHandlers() in the VennSection class are called to handle the methods outlined.
   2. The developer is able to undo and redo actions
      a. The developer moves an entry slightly and then undos and redos the movement of the entry through the use of the "Undo" and "Redo" buttons.
      - All movements and updates are recorded in VennChangeHandler which then replays or rewinds the changes to perform the undo() and redo() actions respectively.

3. The developer is able to double click on the entry placed in the center of the diagram and update its name on the to “Centre”.
    a. The developer ensures that the change was successfully displayed on the venn diagram.
    - When double clicking, the internal static function edit() on VennEntryModalHandler is called, which presents a window with the current data filled in, and allows the user to edit it.
4. The developer is able to double click on the title of the center section and edit it from “Intersection” to “Centre”.
    a. The developer ensures that the change was successfully displayed on the screen.
    - When double clicking, the internal static function edit() on VennEntryModalHandler is called, which presents a window with the current title filled in, and allows the user to edit it.
5. The developer is able to drag the entry placed in the left section to the trash bin to remove the entry from the screen.
    a. The developer ensures that the entry is removed from the screen.
    - This method calls the internal initHandlers() function on the VennDeleteEntry class. This method detects a drop on the trash can, removes the entry from the global list, removes it from the section it was previously in, and removes it from the screen.
6. The developer is able to click on the “Options” button, and change the colors of the venn diagram sections.
    a. The developer clicks on the color picker of each section, and changes it to an arbitrary color presented in the list
    b. The developer ensures that the color change is displayed in the venn diagram after being updated.
    - The button runs the show() method of the VennOptions class, which creates a window similar to the add entry one. This window has color pickers to allow the user to specify their desired color for the section. When the color has been set, the color property of the VennSection class is updated, which in turn changes the color of the section on the venn diagram.
7. The developer is able to click on the “Options” button, and change the language of the application using the dropdown.
    a. The developer clicks on the dropdown and ensures that all visible strings are updated to the language desired.
8. The developer is able to click the “Save Screenshot” button, choose a location on their computer, and save the file as “Venn.png”.
    a. The developer ensures that the file has been successfully saved to the location specified.

   - The system calls the saveScreenshot() function of the VennLeftColumn class, which takes a snapshot of the main group containing all of the venn diagram. A FileChooser is then presented, which allows the user to select where they want the file to be saved to. The file is then saved to the specified by the developer once finalized.

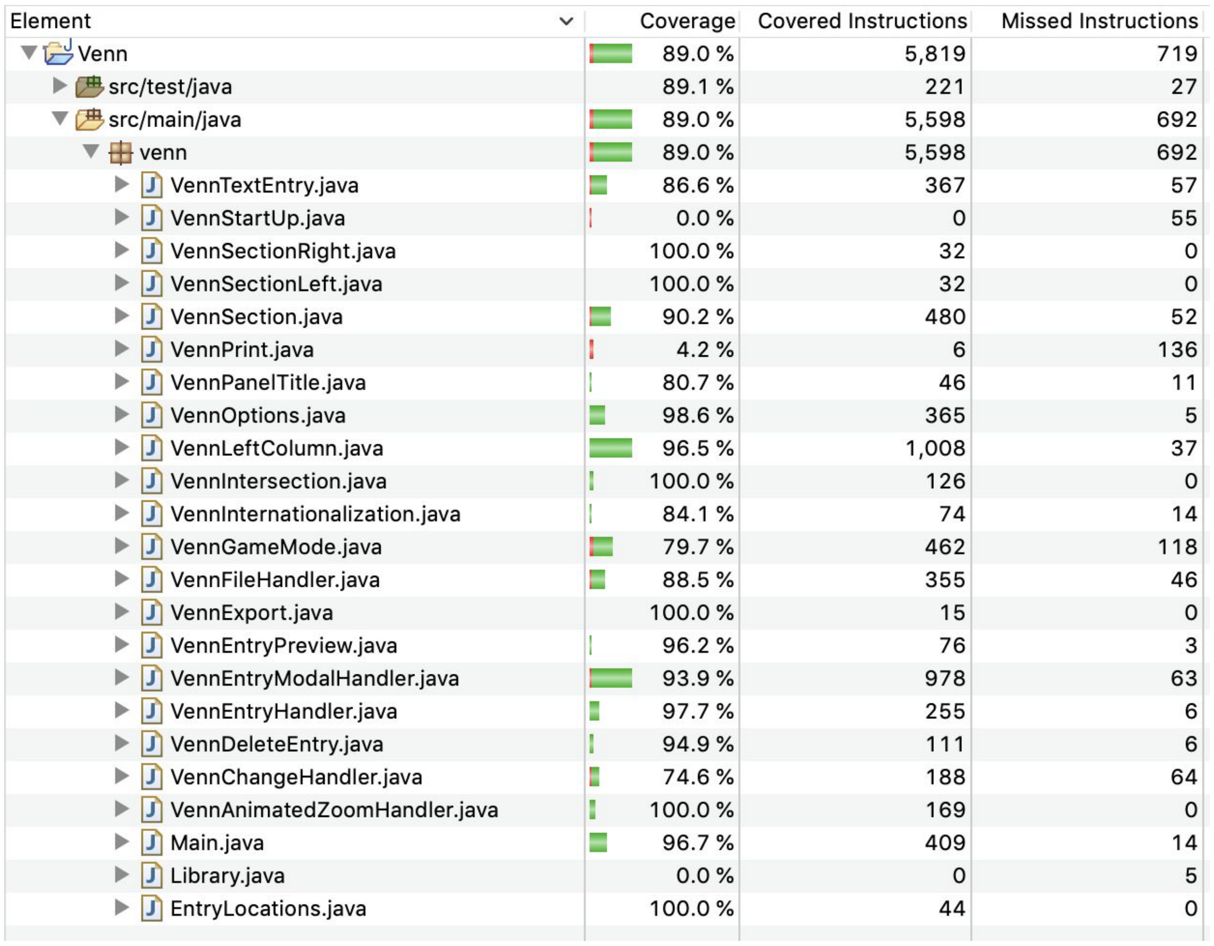
9. The developer is able to test "Game Mode"
   a. The developer selects the "Start" button on the left column, selects the premade game layout (included in the files), and begins the game.
   b. The developer places the entries into their correct circles ("Left" into the Left section, "Right" into the Right section, and "Centre" into the Middle section).
   c. The developer clicks on the "Verify" button and verifies that all entries are in the correct locations.
   d. The developer clicks the "Stop" button which stops the game.
   - The system calls the validate() method of the VennGameMode class, which validates the entries on the diagram with the solutions provided in the setup phase. There are also the methods initialize() and reset() which handle the starting and stopping respectively.

Once the above steps are followed and the program has not crashed during this test, the final test case is validated. The test case itself simply spawns the application and gives the developer time to perform the aforementioned actions.

Summary

The above test cases test all of the core functionality of the application, and mimic behaviour performed by users using the application. They ensure that the current systems behave as expected under most use cases.

Coverage



Element	Coverage	Covered Instructions	Missed Instructions
Venn	89.0 %	5,819	719
src/test/java	89.1 %	221	27
src/main/java	89.0 %	5,598	692
venn	89.0 %	5,598	692
VennTextEntry.java	86.6 %	367	57
VennStartUp.java	0.0 %	0	55
VennSectionRight.java	100.0 %	32	0
VennSectionLeft.java	100.0 %	32	0
VennSection.java	90.2 %	480	52
VennPrint.java	4.2 %	6	136
VennPanelTitle.java	80.7 %	46	11
VennOptions.java	98.6 %	365	5
VennLeftColumn.java	96.5 %	1,008	37
VennIntersection.java	100.0 %	126	0
VennInternationalization.java	84.1 %	74	14
VennGameMode.java	79.7 %	462	118
VennFileHandler.java	88.5 %	355	46
VennExport.java	100.0 %	15	0
VennEntryPreview.java	96.2 %	76	3
VennEntryModalHandler.java	93.9 %	978	63
VennEntryHandler.java	97.7 %	255	6
VennDeleteEntry.java	94.9 %	111	6
VennChangeHandler.java	74.6 %	188	64
VennAnimatedZoomHandler.java	100.0 %	169	0
Main.java	96.7 %	409	14
Library.java	0.0 %	0	5
EntryLocations.java	100.0 %	44	0

After performing the test cases above while a test coverage tool (EclEmma) is monitoring the code run we achieve a **89.0%** coverage percentage. The amount of missing coverage is the result of various edge cases that we did not explore in the testing procedure.