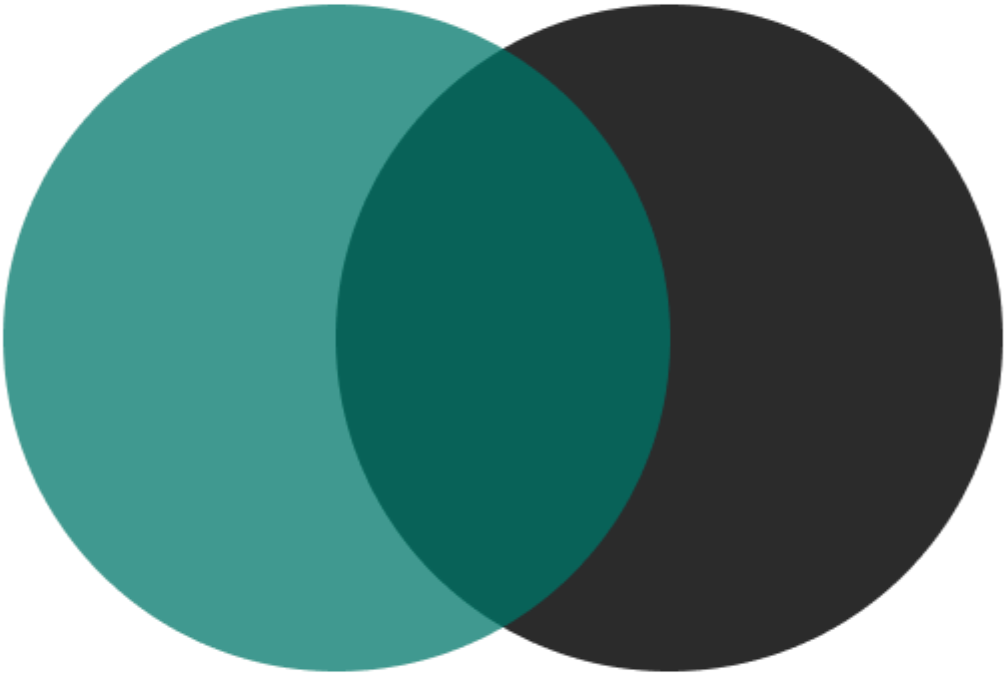


Venn Diagram
Design Document

Reuben Ninan, Eric Kwok, Edward Nwogwug

1. Introduction

1.1 Purpose

The purpose of this document is to describe the architecture of our project *VennFX* and the backend architecture behind all of its functionality. What our project does is give the user a simple, efficient and clear way to create or brainstorm information within the medium of a Venn Diagram. This tool contains many popular features found on many “idea” formulation softwares such as styling features, undo, redo, etc. Users also have the ability to import and export their data when they are finished to save the state of their work.

1.2 Scope

This design document is a deconstructed analysis and insight into the various processes and architectural decisions that the team chose to implement so that two main functions can be accomplished while using this software. The user should be able to create and modify a venn diagram and save/export their diagram into a portable file format for later analysis or to reimport.

2. Design Overview

2.1 Problem Description

Looking for ways to organize your data can sometimes be challenging in terms of how you would go about it. We feel like if users are able to use an interactive tool which makes their experience in categorizing their data easier, it would only enhance their experience and in turn allow them to shift their focus on the topic itself and not have to worry about how they would organize everything. In other words the user will just be putting information in while the application intuitively does the work the user would normally have to do themselves.

2.2 Usage Of Technologies

- Coded in **Java** (**JDK 1.8.0_241**)
- Automatically built with **Gradle** with external dependencies such as:
    - Apache POI
    - TestFX
- Continuously Deployed/Integrated With **CircleCI**
- GUI is Built using the **JavaFX 11** Framework and designed in **Gluon's SceneBuilder**
- Styled with **CSS**

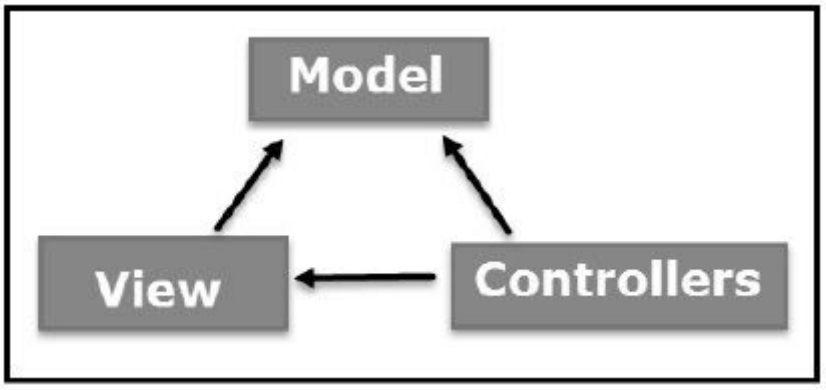
2.3 System Architecture

2.3.1 Architectural Framework

To achieve complete encapsulation and method privacy we decided to utilize the Model-View-Controller architectural framework.

2.3.2 Model-View-Controller (MVC)

This architectural pattern accomplishes the separation of the application into three components. This is a popular industry standard framework when creating applications with visual components such as an embedded GUI or web application.



This diagram gives us a visual representation of the relationship that each logical component has with each other.

Model
This component contains and stores all data related logic that the user can interact with in the background or from an user interface. The data can be transferred between the view and controller components.

View
This component controls all UI logic such as the look and feel or animations.

Controllers
This component acts as a middleware between the *Model* and *View* components to handle and process all data according to predefined logic and manipulate the frontend user interface accordingly. All interactions and inputs are handled by the controller classes.

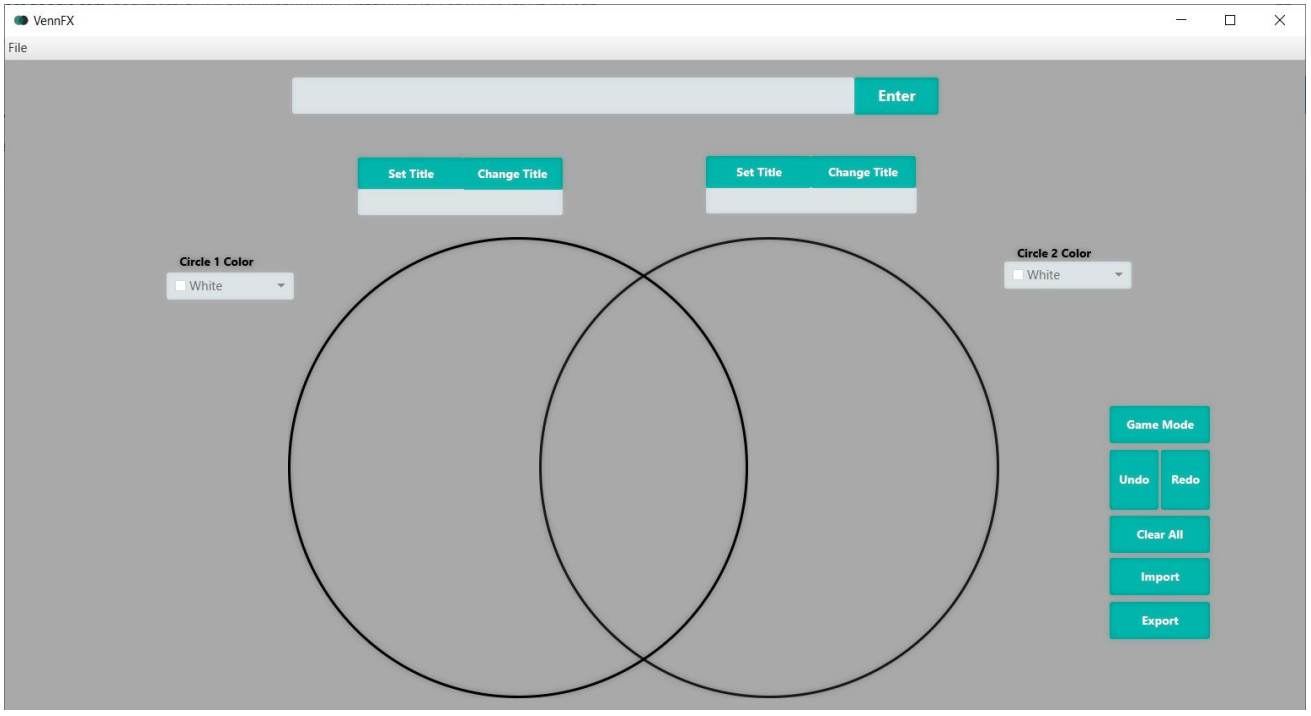
2.4 Constraints

By choosing to use Java SE 1.8 we have ultimately made this application unable to support backwards compatibility with previous versions of Java.

The decision to make this software a desktop application also prohibits us to easily migrate this application to a modern web browser.

Some retina display macbooks have trouble rendering JavaFX scenes, so this proves to be a slight disadvantage in our goal of cross-platform use.

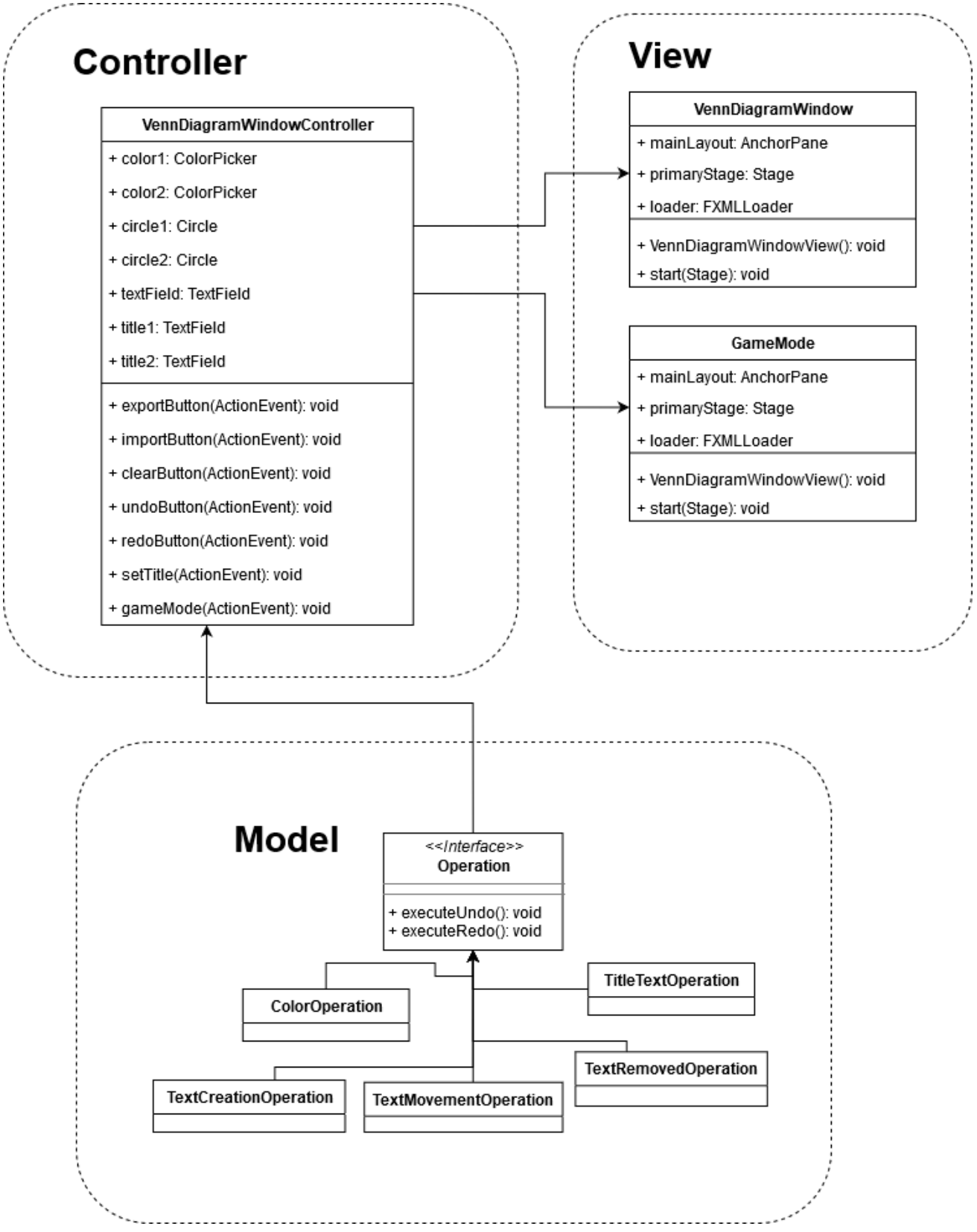
2.5 UI Design Framework



The color palette and design language chosen is based on Google's *Material Design System*. We decided to do a flat light theme to contrast with the dark font colors the user will see from their created textboxes. The teal coloured buttons reflect functionality as the accent color of the application. The two primary application colors are true white and a soft black-grey. To read more about the color design system we followed see https://material.io/design/color/the-color-system.html#color-usage-palettes

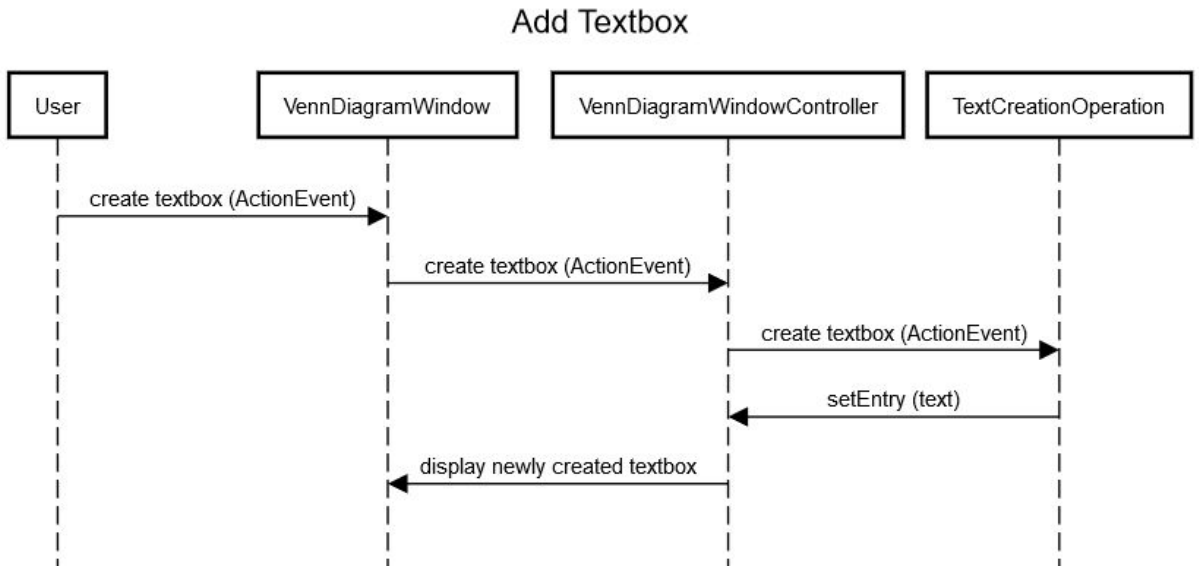
3. Class Diagrams

3.1 MVC UML Diagram



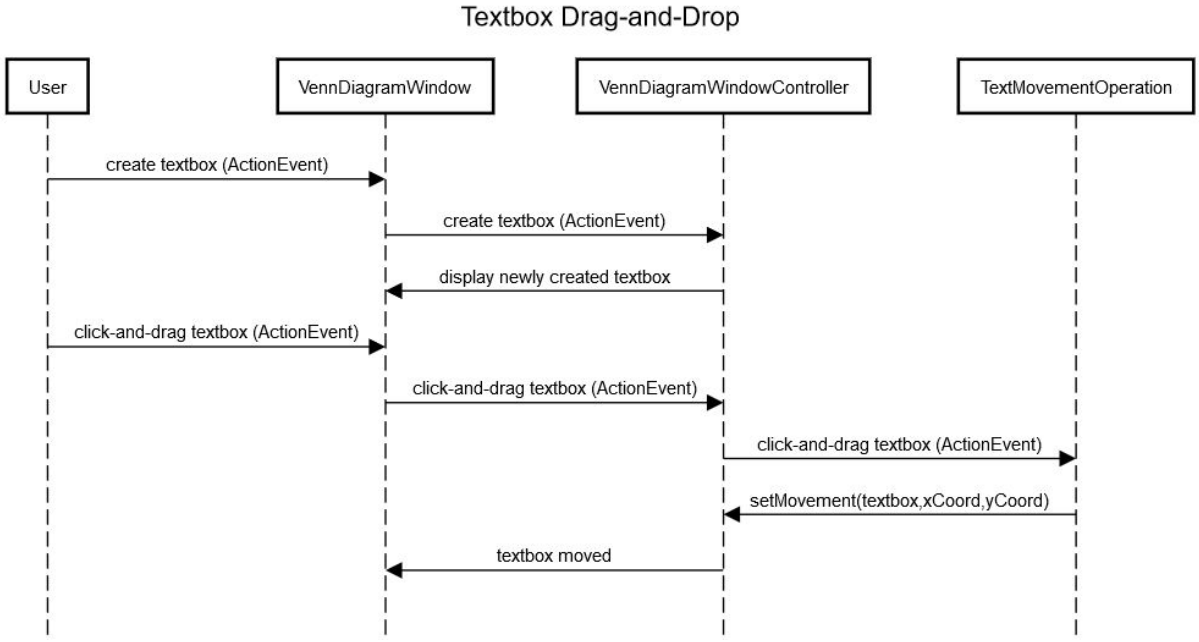
4. Dynamic Model Sequence Diagrams

4.1 Add Entry



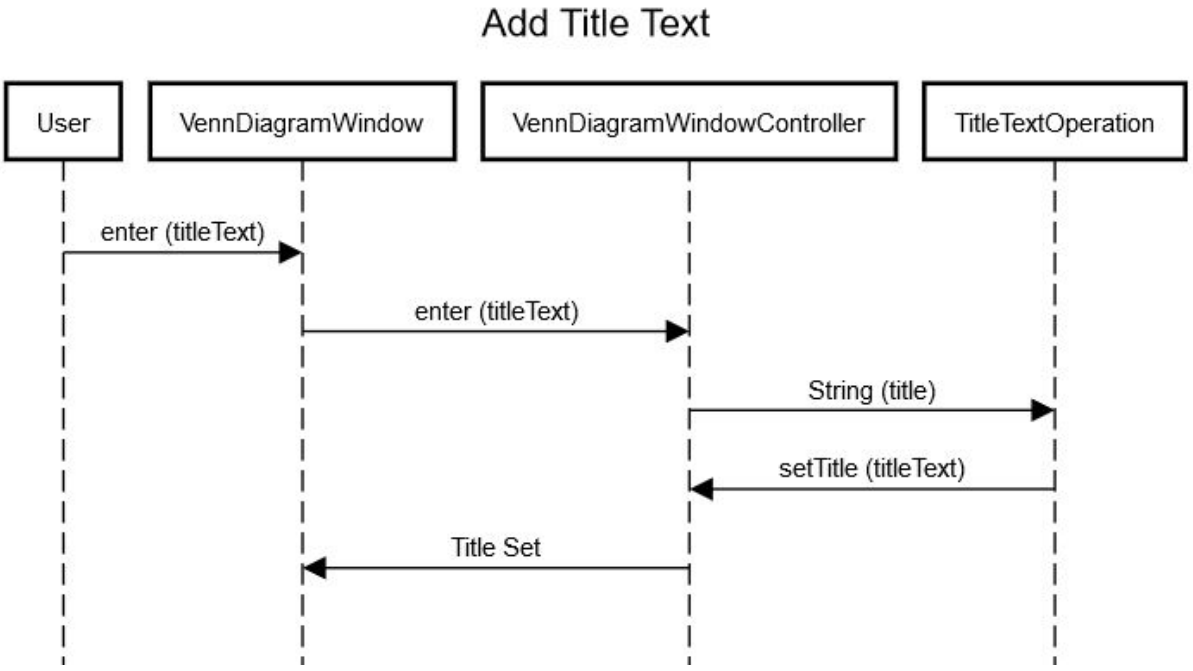
Sequence Diagram Representing the Add Entry Function

4.2 Drag Entry



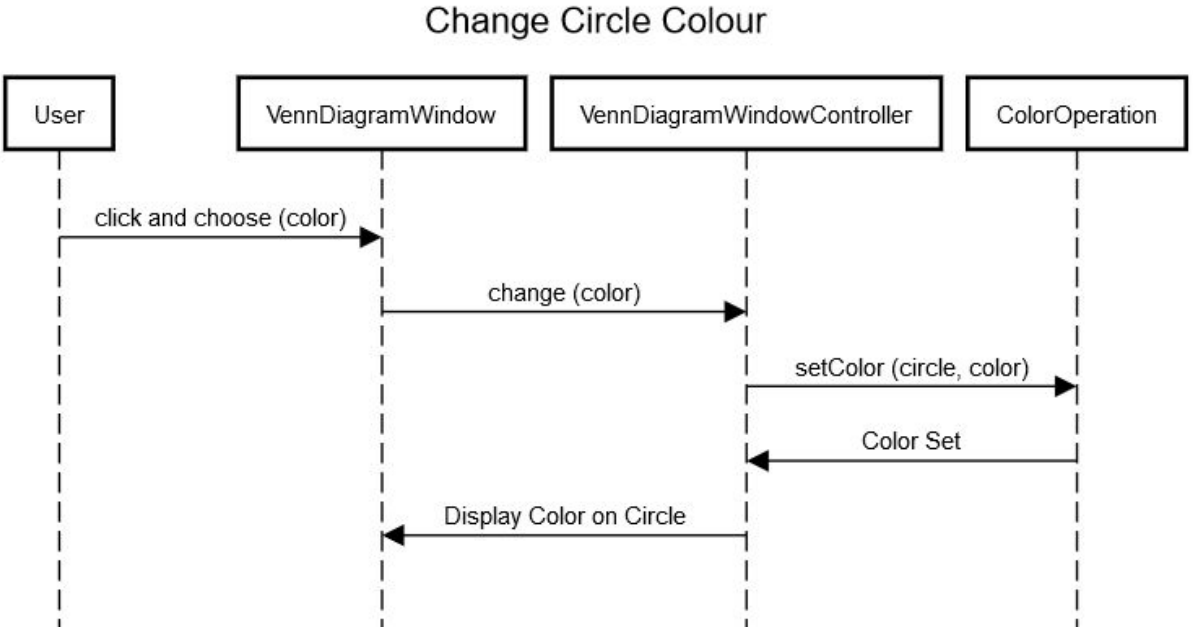
Sequence Diagram Representing the Drag Entry Function

4.3 Add Title



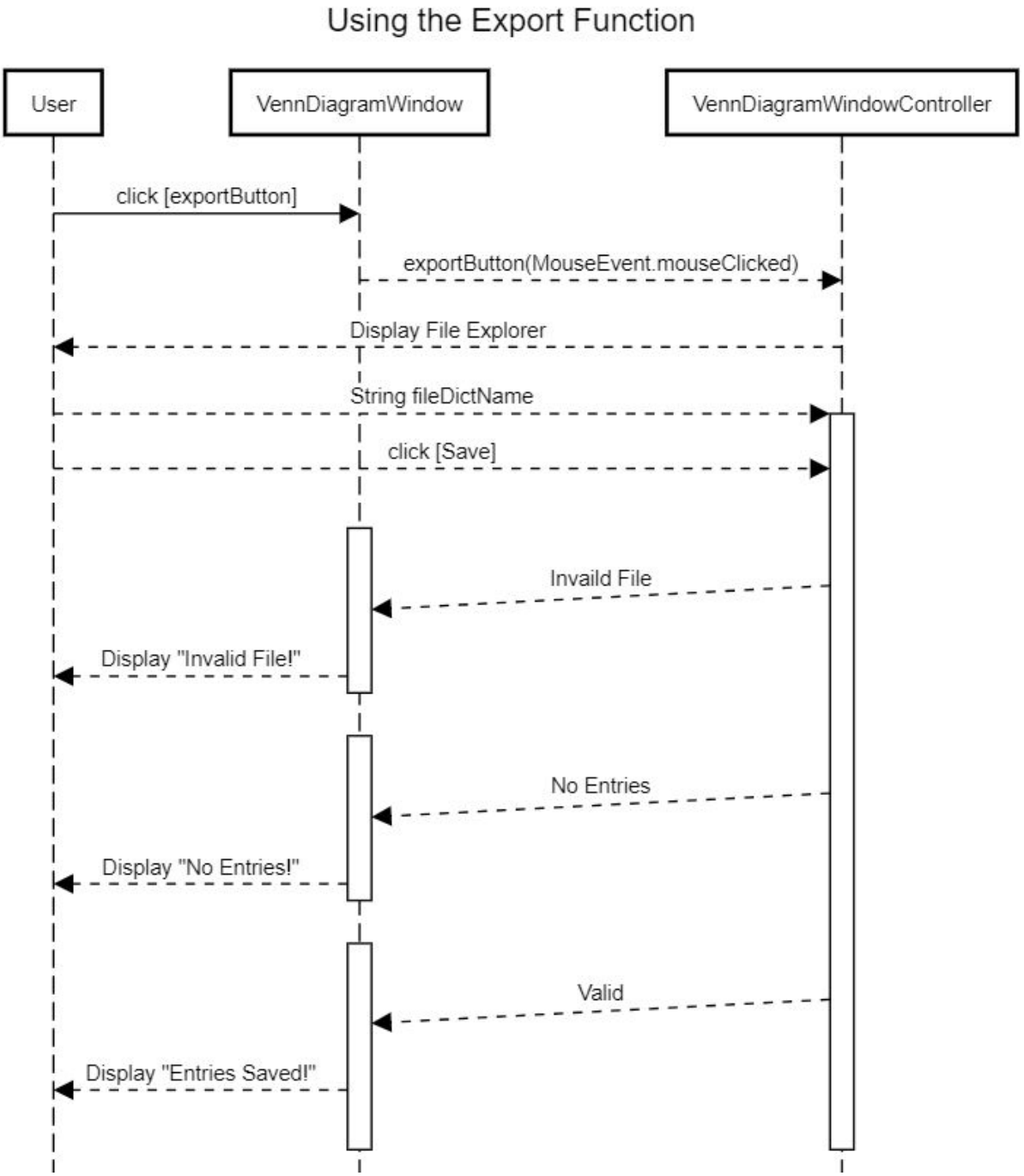
Sequence Diagram Representing the Add Title Function

4.4 Change Color



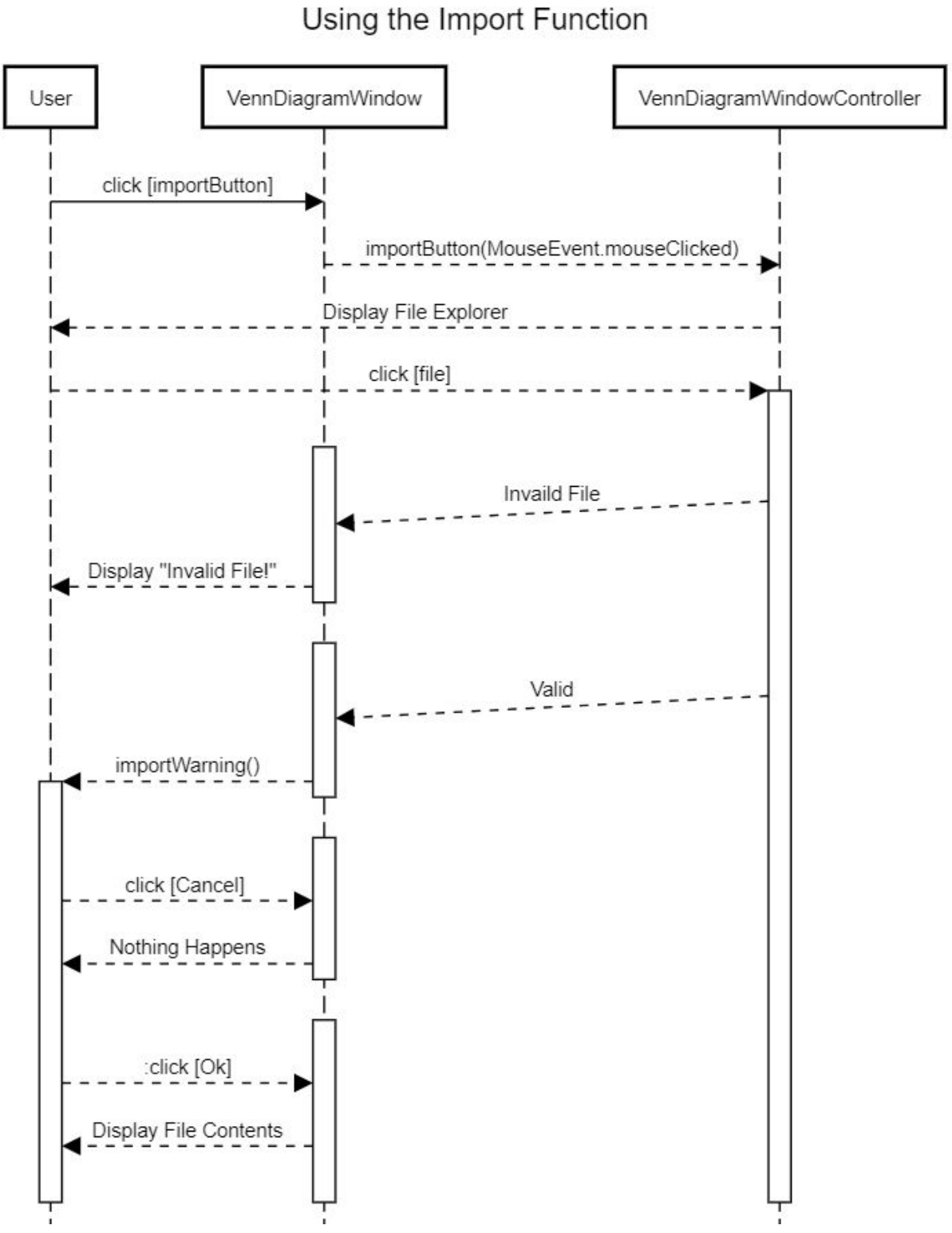
Sequence Diagram Representing the Change Color Function

4.5 Export



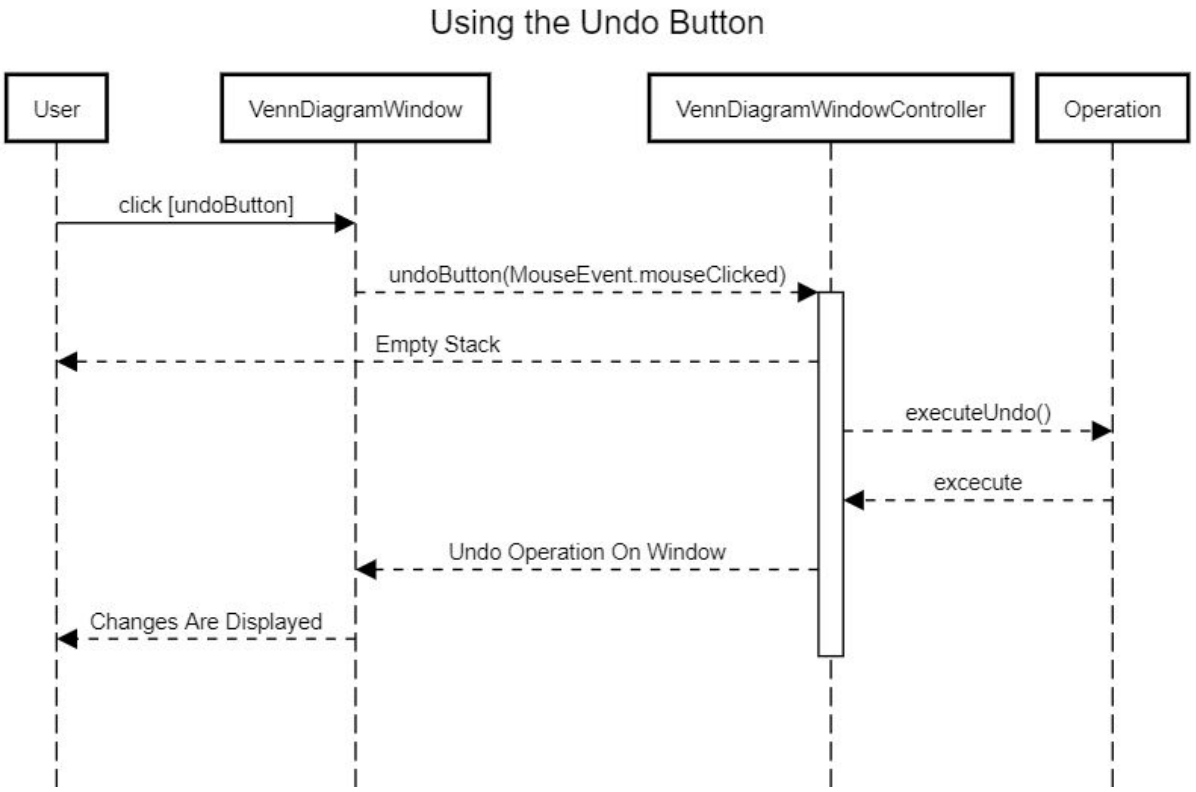
Sequence Diagram Representing the Export Function

4.6 Import



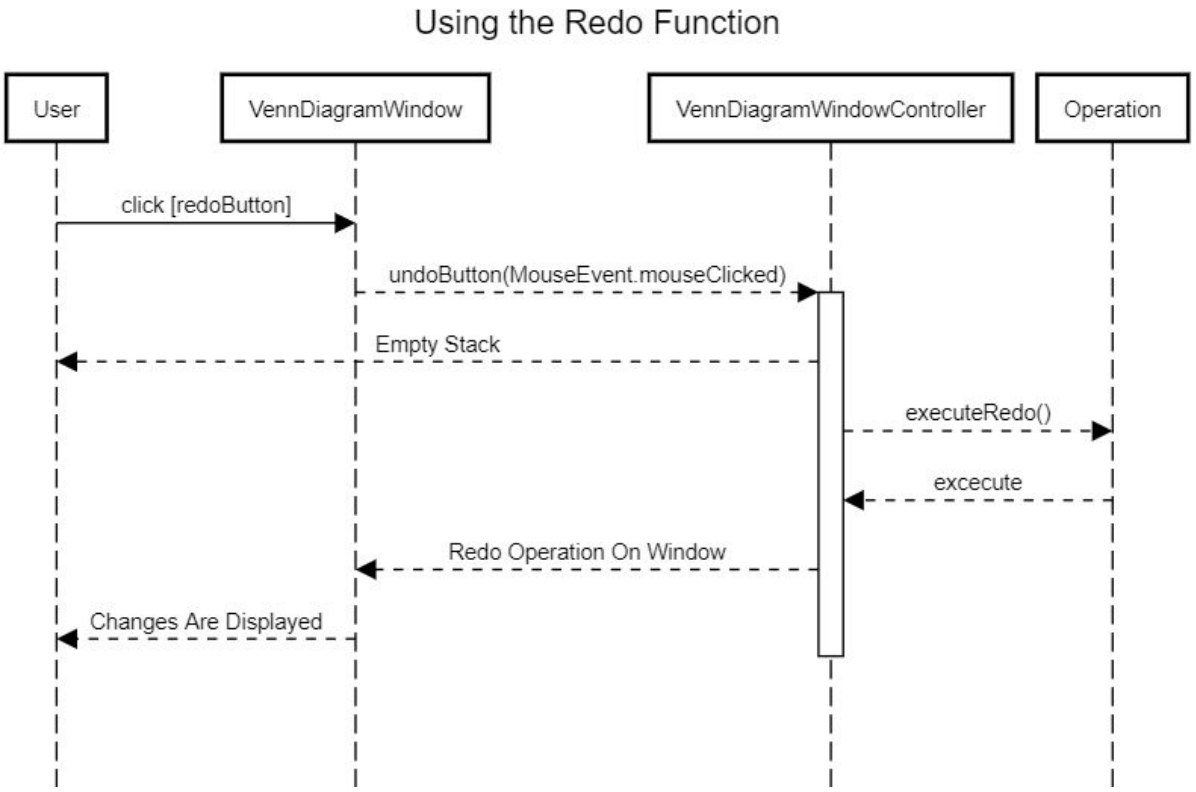
Sequence Diagram Representing the Import Function

4.7 Undo



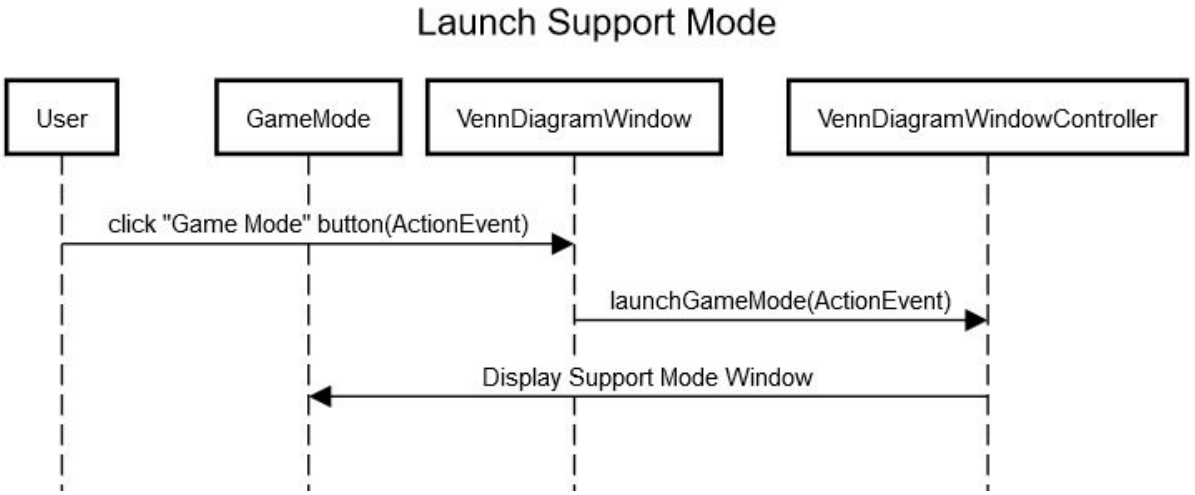
Sequence Diagram Representing the Undo Function

4.8 Redo



Sequence Diagram Representing the Add Redo Function

4.9 Launch Support Mode (Game Mode)



Sequence Diagram representing launching Support Mode

5. Maintenance Scenarios

5.1 Corrective Scenarios

These scenarios detail current bugs that our system encounters.

5.1.1 Responsiveness on varying screen sizes

Currently the sizes and positions of the buttons and other visual components do not reactively react to the size of the window. This is problematic as varying screen sizes will not receive the same experience. To fix this one must actively follow responsive design principles and leverage the padding and margin fields in the relationship between regular panes and anchor panes.

5.1.2 Larger screen sizes don't center visual nodes

On larger screen sizes, the elements and visual components are shifted to the top left of the screen due to their relative anchor points defined in the scene's FXML file. To fix this, one will have to use a listener that tracks current screen size to compute the position of the various nodes.

5.2 Perfective Scenarios

These scenarios detail features that have not been implemented in the current release but will overall contribute to the completion of the VennFX as a whole.

5.2.1 Text styling

Adding text styling such as changing font, color, font-tracking, and other styles is a valuable addition to making this software more immersive to the user. This can easily be implemented by adding a text styling controller class which will modify the data model of the text itself.

5.2.2 VennDiagram image export

Exporting your image to a pdf/jpg/png viewing format could be an essential need for the user to display their brainstorming. To add this feature one could just modify the controller class and leverage the existing Apache POI external library for easy exporting into a custom file format.

5.2.3 VennDiagram menu bar enhancements

Common menu bar functions would increase the experience of the user. Actions such as closing the program, zooming in/out and putting redo and undo buttons on the menu bar could greatly improve the experience. Modifying the main window controller class could easily implement the interface for many of these application functions.

5.2.4 Multiple comparison circles

Adding multiple circles means duplicating the hot zones, positioning and padding of the existing two circles. Modifying the interface and implementations in the models package is also required as they are currently configured for interaction with two circles.