TECHNISCHE
HOCHSCHULE
LÜBECK

Bachelor thesis

Design, Implementation and Test of a Graphical Tool for Course Planning

Submitted by: Mr. Chang Chen

Department: Electrical Engineering and Computer Science

Degree program: Information Technology

First examiner: Prof. Dr. Blaurock

Date handing out: 1st March 2021
Date handing in: 1st June 2021

A. Hanemann

(Professor Dr. Andreas Hanemann)
Head of Examination Board

Task description:

Course planning is a vital step of navigation, especially for sailboats. In robotic sailing other representations for planned courses can be used than those suitable for humans. One approach is the use of vector-fields that describe the preferred course at any location in the area the robot will navigate in. This representation is very easy to use, especially when computational power on-board is restricted.
Using the superposition of parameterized elementary vector fields like radial repulsive, homogeneous, gradients and the like, combined with the option of restricting a field to a specific area, even complex scenarios can be modelled.
The aim of this thesis is the design, implementation and test of a graphical tool that allows editing the superposition of multiple vector-fields. An already existing, underlying Java-toolbox allows the use of certain elementary fields and selected operations on existing fields. The resulting vector-field shall be visualised while editing, allowing WYSIWYG technologies to be applied for course planning.
The resulting vector field shall be represented using JSON syntax in a given format that is suitable for saving in a file and lossless reconstruction for later use or editing.
The WYSIWYG-editor for the vector fields shall allow convenient editing including copy/cut and paste operations on sub-fields. The tool shall be based on the existing toolbox and add features to the vector-field-based representation such as labels.

Aim / expected results
In this work the following major components shall be implemented, tested and documented suitable for future use:

- GUI-based tool using WYSIWYG technology that allows the creation and editing of vector fields based on the elementary fields and operations specified by the given toolbox.
- Editing shall include (among others) the assignment of user-defined labels to sub-fields as well as paste operations on sub-vector fields, based on copy/cut of parts of the currently edited field as well as on vector fields loaded from file.
- Design of a file format extending the file format of the underlying toolbox, e.g. by embedding it, allowing to store the additional information like labels assigned to vector-fields.
- File operations on files containing the lossless representation of vector-fields in a given JSON-based format: Load (for editing), save (a changed or newly created vector-field).

The design, implementation and documentation of the software created in this work must be suitable for long-term use in the Autosail project. All graphical components shall be implemented based on the JavaFX framework.

Required Skills

- Basic software engineering skills, e.g. knowledge of UML, use-case analysis, modular design
- Programming skills of at least advanced intermediate level using Java (and Java FX).
- Vector calculus in 2D Euclidian space
- Basic technical German as some project documentation is written in German.
- Willingness to act on your own initiative.

Prof. Dr. Blaurock

Fachbereich
Elektrotechnik und Informatik

TECHNISCHE HOCHSCHULE LÜBECK

Eidesstattliche Versicherung

Chen, Chang
Name, Vorname

33247
Matrikelnummer

Ich versichere hiermit an Eides Statt, dass ich die vorliegende

□ Hausarbeit ☒ Bachelorarbeit □ Masterarbeit

mit dem Titel

Design, implement and Test of a Graphical Tool for Course Planning

eigenständig und ohne unerlaubte fremde Hilfe angefertigt habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel verwendet. Entlehnungen aus anderen Arbeiten habe ich kenntlich gemacht.

Für den Fall, dass die Arbeit zusätzlich elektronisch und/ oder digital eingereicht wird, erkläre ich, dass die schriftliche und die elektronische und/ oder digitale Form identisch sind.

Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Ich bin damit einverstanden, dass die vorliegende Hausarbeit/ Bachelorarbeit/ Masterarbeit für Veröffentlichungen, Ausstellungen und Wettbewerbe des Fachbereiches verwendet und Dritten zur Einsichtnahme vorgelegt werden kann.

☒ ja □ nein

Hamburg, 28.05.2021
Ort, Datum

Chen Chang
Unterschrift

Abstract of Thesis

Department: **Electrical Engineering and Computer Science**

University course: **Information Technology**

Subject: **Design, Implementation and Test of a Graphical Tool for Course Planning Based on Superposition of Multiple Vector-Fields**

Abstract:

In the Autosail project at the THL we build a fleet of computer-controlled model sailing agents (sailboats and sand yachts). One approach to course planning is the use of vector-fields describing the preferred course at any location in the sailing area. The vector field used for course planning is modelled by superposition of parameterized elementary fields. Based on an existing Java toolbox for 2D vector field operations, a graphical tool is designed, implemented, and tested that allows editing such course representations in a convenient way, using the WISYWIG approach.

Author: **Chang Chen**

Attending Professor: **Prof. Dr. Ole Blaurock**

WS / SS: **SS 2021**

Table of Contents

1 Introduction

Course planning is a vital step in navigation, especially for sailboats. In robotic navigation, routes can be planned using other representations than those appropriate for humans. One approach is to use vector fields that describe the preferred course at any location in the area to be navigated by the robot. This representation is easy to use, especially when the computational power onboard is limited. Using the superposition of parameterized elementary vector fields, such as radial repulsive, homogeneous, gradients, and the like, together with the option to restrict the vector field to a specific area, even complex scenarios can be modeled.

This thesis aims to design, implement, and test a graphical tool that allows editing the superposition of multiple vector fields based on an underlying Java-toolbox that allows the use of certain elementary fields and selected operations on existing fields.

The resulting vector fields are visualized when edited, allowing WYSIWYG techniques to be applied to course planning. The resulting vector fields are also represented using JSON syntax in a format suitable for saving in a file and for lossless reconstruction for later use or editing.

The WYSIWYG editor for vector fields allows for easy editing, including copy/cut and paste operations implemented by drag and drop on sub-fields. The editor is based on the existing toolbox with some additional features for vector field-based representations, such as labels,

2 Background

In this chapter, the existing Java toolbox for 2D vector field operations used for course planning in the Autosail project at the THL [2] on which this thesis is based is introduced. Then the mathematical basics for the handling of 2D-vector fields are described.

2.1 The existing toolbox

The existing toolbox implemented in Java for 2D vector field operations by Manuela Kastner [1] and optimized by Christian Weber [2] is constantly referenced in this document as the toolbox. Since this work is to design an editor for two-dimensional vector fields that uses the toolbox, a basic understanding of the existing toolbox is necessary.

2.1.1 Fundamental representation

A two-dimensional vector field can be imagined as a plane, where each point from the plane is assigned a vector. The plane's points are called location vectors, and the vectors located at the location are called direction vectors.

The function **F** for the computation of the direction vectors under specification of a location vector is formally defined as follows (see equation 2.2) [3]:

$$\mathbf{F}\colon D \subseteq \Re^2 \to \Re^2 : (x,y) \mapsto F(x,y) = \begin{bmatrix} f(x,y) \\ g(x,y) \end{bmatrix} \quad (2,2)$$

2.1.2 Elementary vector fields

In Kastner's work Section 2.2 [1], she introduces four elementary vector fields used in the toolbox.

The *homogeneous vector field* (Figure 2.1 a) has the most straightforward function, which assigns the same direction vector to each point independent of its position. The other vectors change the direction and magnitude depending on the point under consideration.

For the *boundary vector field* (Figure 2.1 b), all direction vectors point in the same direction, but the distance to the Y-axis influences the vectors' magnitude. There are different proportionality functions according to which the magnitude can be calculated depending on the Y-axis distance.

The *radial vector field* (Figure 2.1 c) is another primary field. In it, a central point is decisive for the direction and the magnitude of the direction vectors. The magnitude depends on the distance, as in the case of the boundary field.

The last primary field is the *tangential vector field* (Figure 2.1 d). In this field, again, the central point plays a role. Analogous to the radial field, a central point is also decisive for the magnitude of the direction vectors. The vectors' direction is defined so that they are always in the right angle to the straight line, which runs from the point under consideration through the central point.

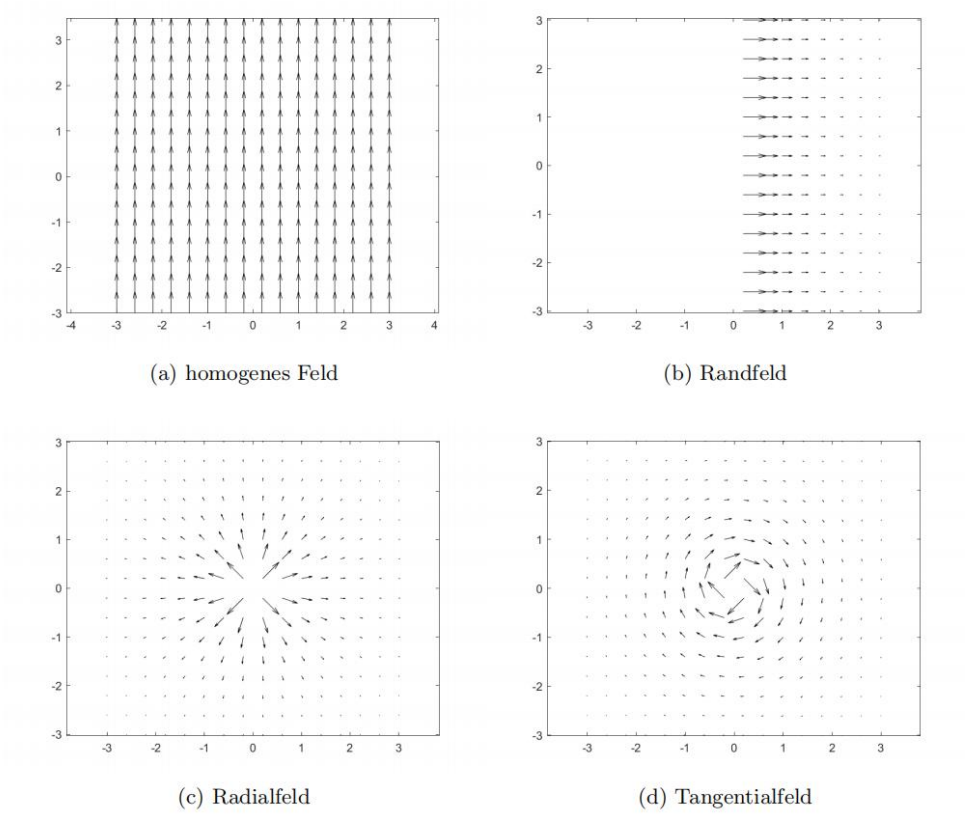


Figure 2.1 Four elementary vector fields of the toolbox illustrated with specific values in Weber's work Section 2.1 as explanatory figures [2]

2.1.3 Operations

Operations are applied to all these two-dimensional vector fields in the same way. [2] The toolbox's implemented operations allow to move, scale, shear, and rotate two-dimensional vector fields. It is also possible to mask two-dimensional vector fields- to scale certain areas of two-dimensional vector fields- and overlay two two-dimensional vector fields.

In Weber's work [2], the toolbox supports the following formulas [4] from equation 2.3 to equation 2.9 for implementing operations on two two-dimensional vector fields.

To implement move operation, location vector $\binom{x}{y}$ can be manipulated with a displacement vector $\overrightarrow{v}$ (see equation 2.3)

$$\begin{pmatrix} x + \overrightarrow{v_x} \\ y + \overrightarrow{v_y} \end{pmatrix} \mapsto \begin{pmatrix} x \\ y \end{pmatrix} \quad (2.3)$$

The scaling makes it possible to manipulate the direction vectors' magnitude, which means that the location vector is not scaled. The direction vector $\begin{pmatrix} x \\ y \end{pmatrix}$ determined by the given location vector is multiplied by α. (see equation 2.4)

$$\begin{pmatrix} x \\ y \end{pmatrix} \mapsto \alpha \cdot \begin{pmatrix} x \\ y \end{pmatrix}, \alpha \in R \ (2.4)$$

In shear, one axis remains fixed while shearing along the other axis. The following matrices achieve shear: along the X-axis with the matrix Sch_x (see equation 2.5 left) or along the Y-axis with the matrix Sch_y (see equation 2.5 right). Here α is the value multiplied by x of the input vector and added to the y value of the output. β, on the other hand, is multiplied by y of the input and added to the output of the x value.

$$Sch_x = \begin{pmatrix} 1 & 0 \\ \alpha & 1 \end{pmatrix} \quad Sch_y = \begin{pmatrix} 1 & \beta \\ 0 & 1 \end{pmatrix} \ (2.5)$$

Another standard operation that can be applied to vector fields is rotation. To rotate a direction vector $\begin{pmatrix} x \\ y \end{pmatrix}$ in the two-dimensional plane around the origin, the matrix from equation 2.6 is used. [5]

$$\begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} \cos(\theta) & -\sin\theta \\ \sin\theta & \cos(\theta) \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \ (2.6)$$

Also, a selected range of a two-dimensional vector field can be masked, and all values outside this range can be set to zero. Let $\vec{n}$ be the zero vector for which holds: $|\vec{n}| = 0$. A surface to be masked can be defined as a defined subset A of the definition domain D (see equation 2.7):

$$\vec{r}(\vec{p}) = \begin{cases} \vec{p} \in A: \vec{F}(\vec{p}) \\ others: \vec{n} \end{cases} with\ A \epsilon D\ and\ D \in \Re^2 \ (2.7)$$

The toolbox restricts the possible shapes of masking (of the set A) to rectangles and circles defined with center and side length or radius. The edges of a rectangle are always parallel to the X and Y axes.

In addition, the toolbox offers a specialized scaling case with the scaling of specific only direction vectors that lie in a defined range can be scaled.

The range A is defined as a subset of the definition range D. Now, a scaling factor f, which is multiplied with all direction vectors within A, is used for the scaling of direction vectors. Direction vectors that lie outside of the form remain unchanged. (see equation 2.8)

$$\vec{r}(\vec{p})=\begin{cases}\vec{p} \in A: f \cdot \vec{F}(\vec{p}) \\ others: \vec{F}(\vec{p})\end{cases} \; with\ A \epsilon D\ and\ D \in \Re^2 \quad (2.8)$$

At last, the superposition of different two-dimensional vector fields corresponds to the addition of the direction vectors of two two-dimensional vector fields $\overrightarrow{F_1}$ and $\overrightarrow{F_2}$ which have an identical location vector. Let the resulting field be F_{res} .

$$\overrightarrow{F_{res}}=\overrightarrow{F_1}+\overrightarrow{F_2}. \quad (2.9)$$

3 Requirements analysis

User and system requirements derived from the coursing assignment are served as guidelines for further work and must be fully implemented. The prototypes are designed following the criteria. The respective sections of the implementation, test, and evaluation also closely focus on user and system requirements.

3.1 Use case diagram

A use case diagram is a graphical depiction of a user's possible interactions with a system. The author uses this approach for user requirements mining and analysis. Figure 3.1 is the abbreviate use case diagram of the editor, which gives an immediate impression of the editor's functionality.

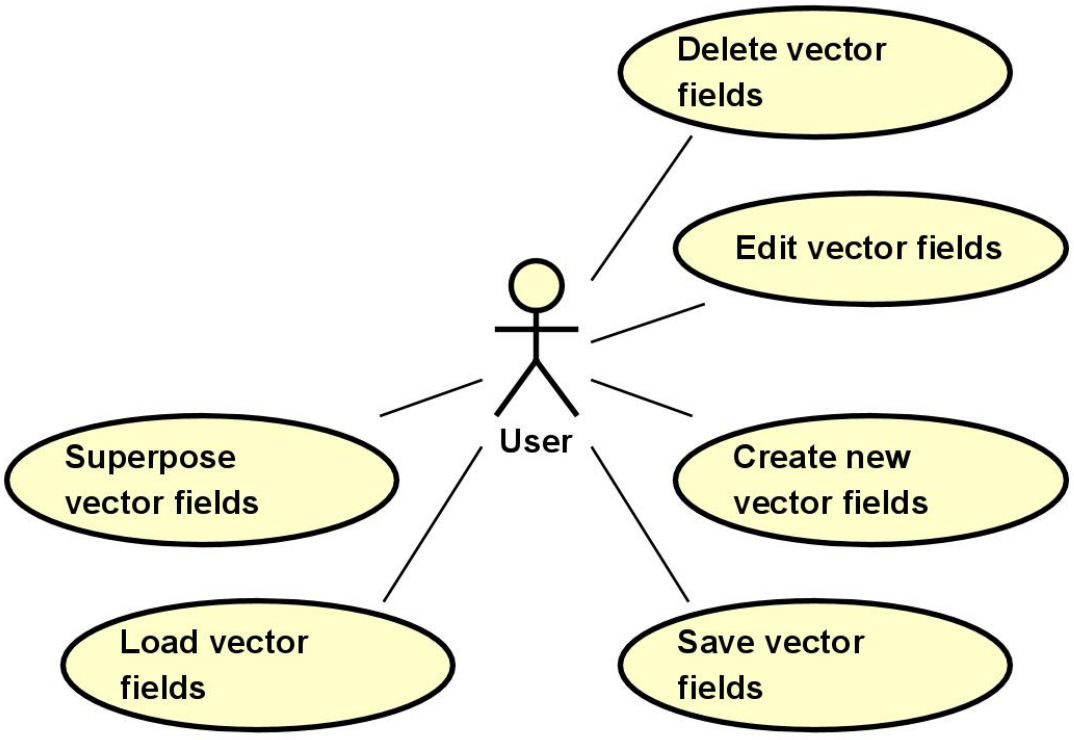


Figure 3.1 The abbreviate user case diagram for the editor

By extending the abbreviate use case diagram, the author generates 15 use cases based on Alistair Cockburn's template [10]. These use cases help to generate system requirements. Also, in the Test section, they are guidelines for test cases in black-box testing.

In the extended version, the use case "Create new vector fields" can include two sub-cases depicted in Figure 3.2, namely, "Create an elementary vector field" and "Create a customized vector field."

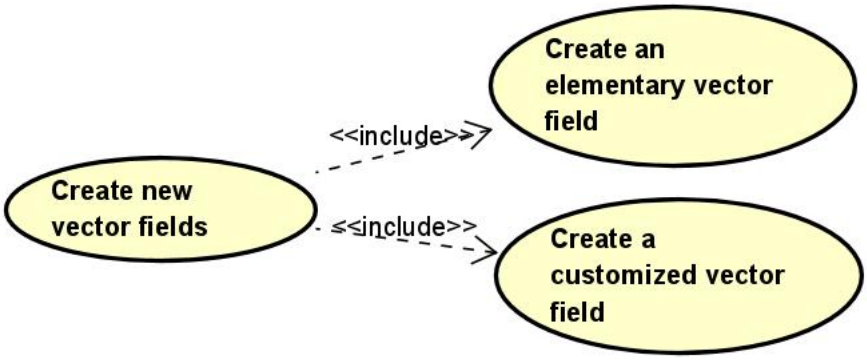


Figure 3.2 Extension of "Create new vector fields" use case

As Figure 3.3 depicts, the use case "Edit vector fields" can include six other sub-cases, such as "Perform operations on a superposed vector field," "Perform operations on a customized vector field," "Perform operations on an elementary vector field,"" Change properties of an elementary vector field,"" Change the name of a customized vector field,"" Change sub-vector fields' properties of a customized vector field."

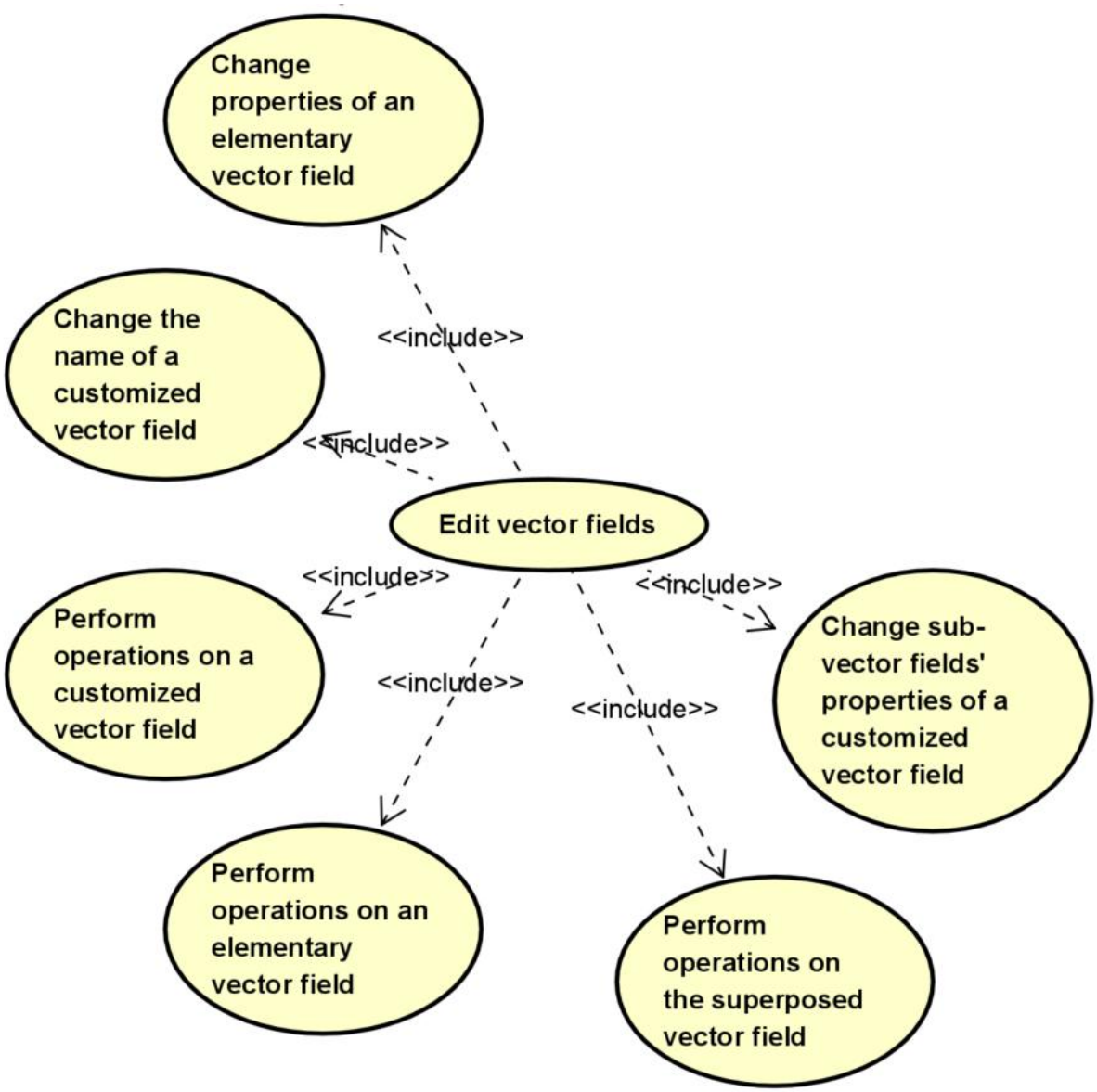


Figure 3.3 Extension of "Edit vector fields" use case

For the use case "Delete vector fields," as Figure 3.4 depicts, it includes four other sub-cases, such as "Delete a customized vector field," "Recover the customized vector field from Rubbish Bin," "Delete an elementary vector field,"" Recover the elementary vector field from Rubbish Bin."

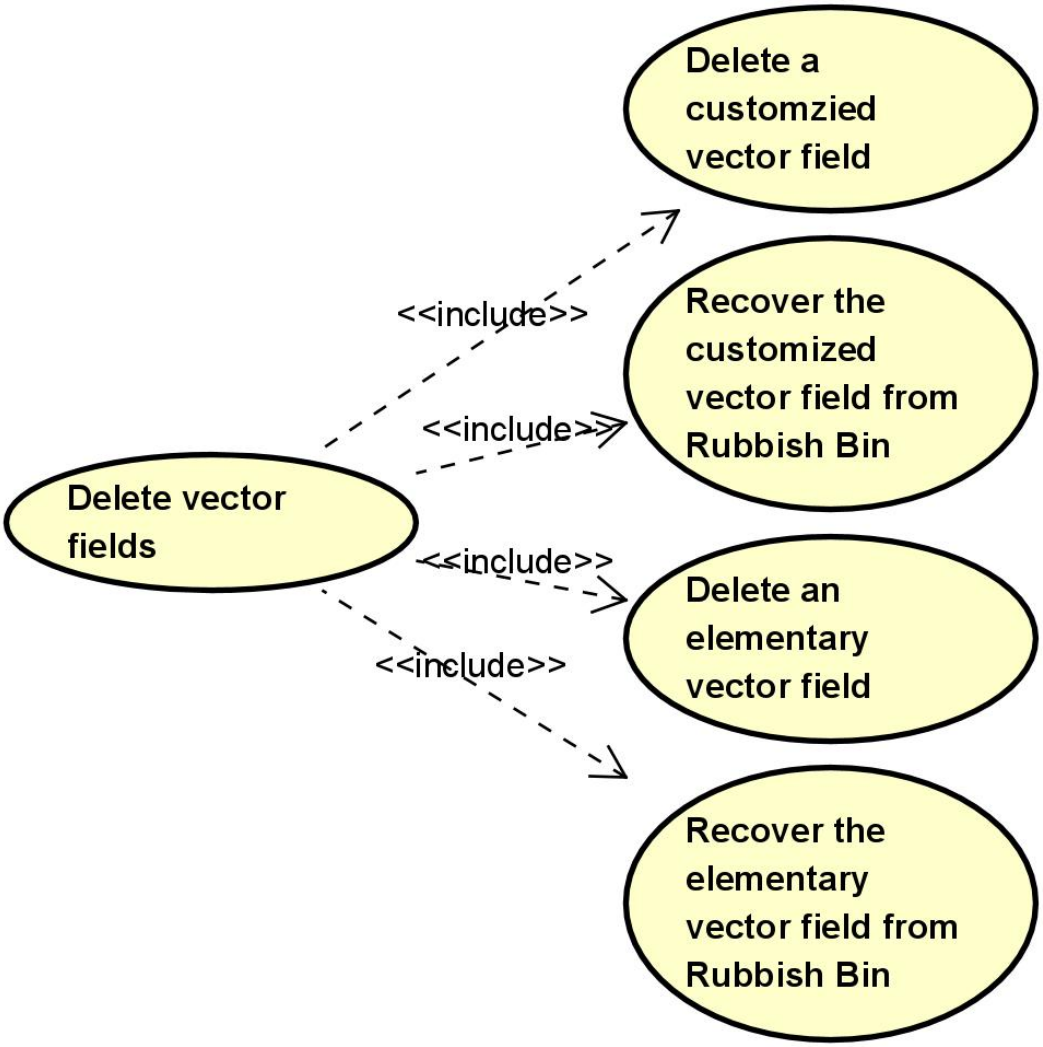


Figure 3.4 Extension of "Delete vector fields" use case

3.2 User requirement analysis

Designing, implementing, and testing a graphical tool for the toolbox is the purpose of the thesis. The user requirements of this work are:

- UA1 User can create new vector fields.
- UA2 User can edit vector fields, including changing field properties, performing operations, and changing names.
- UA3 User can save vector fields.
- UA4 User can load vector fields.
- UA5 User can zoom in/out vector fields.
- UA6 User can delete vector fields.
- UA7 User can superpose vector fields.

3.3 System requirement analysis

Specific system requirements can be derived from the user requirements and the use case diagram. They are listed below and explained in more detail.

- SA1 Implementation of vector fields creation
  -SA1.1 The editor requires lists for all usable elementary vector fields and certain operations (Translate, Rotate, Scale by factor, and Shear).
  -SA1.2 Each type of vector field needs a unique detail pane.
  -SA1.3 The editor requires a tree structure component for saving created vector field.
  -SA1.4 The editor requires a button to create vector fields.

- SA2 Implementation of vector fields edition
  -SA2.1 Each vector field requires a tree structure component for holding operations performed on it.
  -SA2.2 The vector field storage component mentioned in SA1.3 supports vector fields' selection, cut, and paste.
  -SA2.3 The editor supports a rename function for customized vector fields.
  -SA2.4 The editor supports operations performing on the result vector field.

- SA3 Visualization of the resulting vector fields
  -SA 3.1 The display pane supports vector field display after each Save operation.
  -SA 3.2 The display pane supports zooming and changing scopes of the result vector fields in x and y axes.
  -SA 3.3 The display pane supports automatic adjustment of the result vector fields displays according to the window size.

- SA4 Implementation of vector fields save and load
  -SA 4.1 The editor can save a vector field in JSON format.
  -SA 4.2 The editor can load vector fields from compatible JSON files.

- SA5 Implementation of vector fields deletion and restoration
  -SA 5.1 The editor supports the deletion of vector fields created.
  -SA 5.2 The editor can quickly recover the vector field deleted.

- SA6 Implementation of vector fields superposition
  -SA 6.1 The editor supports the superposition of vector fields.

4 Design

In this chapter, the graphic user interface design is discussed. The author firstly describes the agile development process of the project. Secondly, the author points out the specialized naming terms relating to the editor. After that, the author discusses the class diagrams of the editor. Then, the author explains how the vector fields are displayed without handy libraries in JavaFX.

4.1 Prototyping and iterative design

Based on Jakob Nielsen's eleven-step model [8] to increase the probability of designing a GUI with high usability, before starting to implement actual code, it is crucial to prototype a GUI [8]. Prototypes help to understand requirements, concertize, and supplement. The prototypes can include Lo-Fi-Prototypes such as paper prototypes or Hi-Fi-Prototypes such as realistic GUI design. [8] The author chooses the realistic GUI design to create interactive editor prototypes. Also, the author uses the principle of iterative design in, but not limited to, the early project stages when prototypes are presented to the user and changes based on the user evaluation are integrated into the next iteration step of the prototype.

4.1.1 Prototype for the first iteration of the graphic user interface

In the first iteration of the editor project, as Figure 4.1 depicts, the graphic user interface has a menu bar that consists of three catalogs- "File," "Edit," "Help." There are the "Library," the "Operations," and the "Root Node" for the superposition of vector fields in the left sidebar, all in the tree structure. In the right sidebar, there is the detail panel. If the user clicks the "File," a menu consists of "New, ""Open," "Save as, ""Output as a picture, "and "Quit" will be presented.

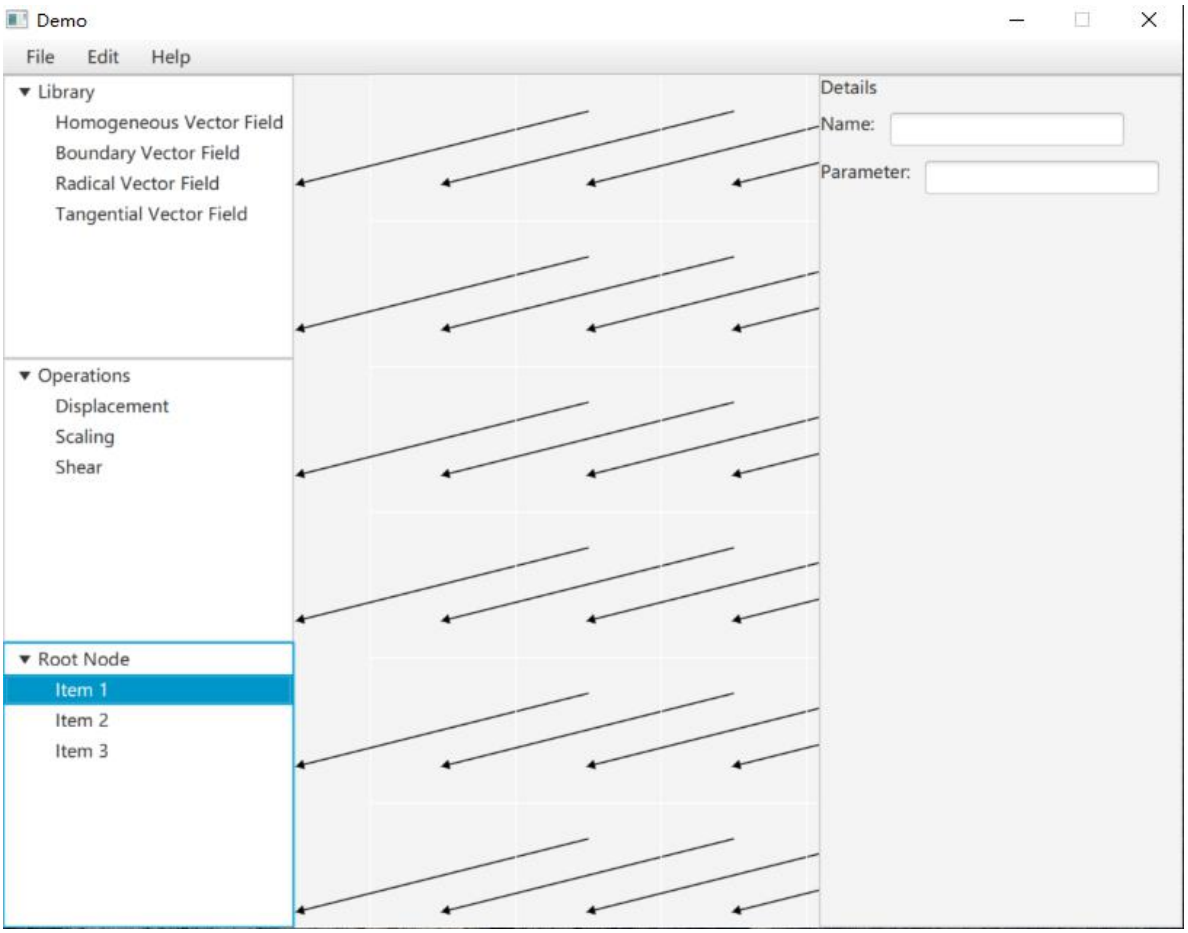


Figure 4.1 The prototype for the first iteration of the graphic user interface

In detail, in the “Library” tree view component, there are four elementary two-dimensional vector fields-“Homogeneous vector field, “” Boundary vector field, ““Radial vector field,” and “Tangential vector field. “People can drag them out of the “Library” tree view component to the “Root Node” tree view component to perform copy and paste operations. Alternatively, they can click the “Add” and “Delete” buttons in the “Edit” menu to do the same operations. Also, in the “Edit” menu, users can do copy, cut, and paste operations. Furthermore, there are several operations in the “Operations” tree view that can be performed, such as “Displacement,” “Scaling,” and “Shear”. By this time, how users can perform operations is undecided. In the middle tile, the vector field is designed to be presented there.

After the first iteration is presented to and evaluated by the user, changes based on the user evaluation are integrated into the next iteration step of the prototype. This means that the author needs to find out the solutions to the following questions. For example, how operations are shown when they are part of the fields? Is it possible to select a field (or operation) in the tree-view and change specific parameters? How are the properties/parameters of selectable tree nodes displayed? How can users zoom in/out or move the visible part of the field (the canvas)?

4.1.2 Prototype for the second iteration of the graphic user interface

In order to realize the ideas from user and system requirements and the evaluations from the user, the second iteration of the graphic user interface is released. As Figure 4.2 depicts, operations of each vector field can now be added into an “Operation List” tree view component in the right sidebar in the case that operations perform on a vector field. Users can now drag and drop operations from the “Operations” tree view in the left sidebar to the “Operation List” in the right sidebar to copy and paste them. Also, the vector field’s parameters can be adjusted in the text field components in the right sidebar above. By clicking certain operations in the “Operation List” tree view component, users can adjust the parameters of each operation in the text field components below. However, the zooming issue remains in this iteration. Also, the author skims the “File” and deletes the redundant menu items not mentioned by requirements to “Save” and “Load.”

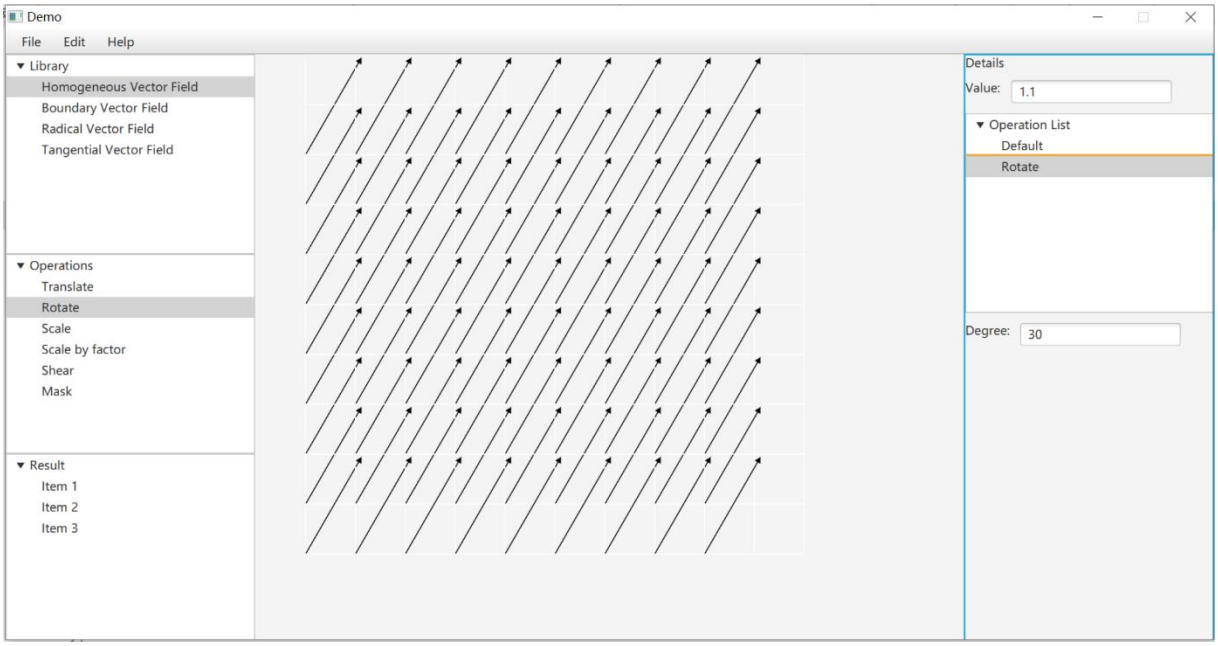


Figure 4.2 The prototype for the second iteration of the graphic user interface

4.1.3 The final version of the graphic user interface

The agile development process comes with the cooperation between the inspector and the author. After 3-4 more iterations, more functions are settled down.

As Figure 4.3 depicts, a more controllable display pane is enabled, such as resizable window and adjustable display parameters with units in the top HBox component. All nodes in the “Result” and “Operation List” tree view components are editable, friendly to construct by dragging and dropping from the “Library” and “Operations” tree view components. They also support parameter changing and superposing. The file saving, loading, and name changing functions are also added in this version of the project, which allows users to name, save, and load their personally created vector fields.

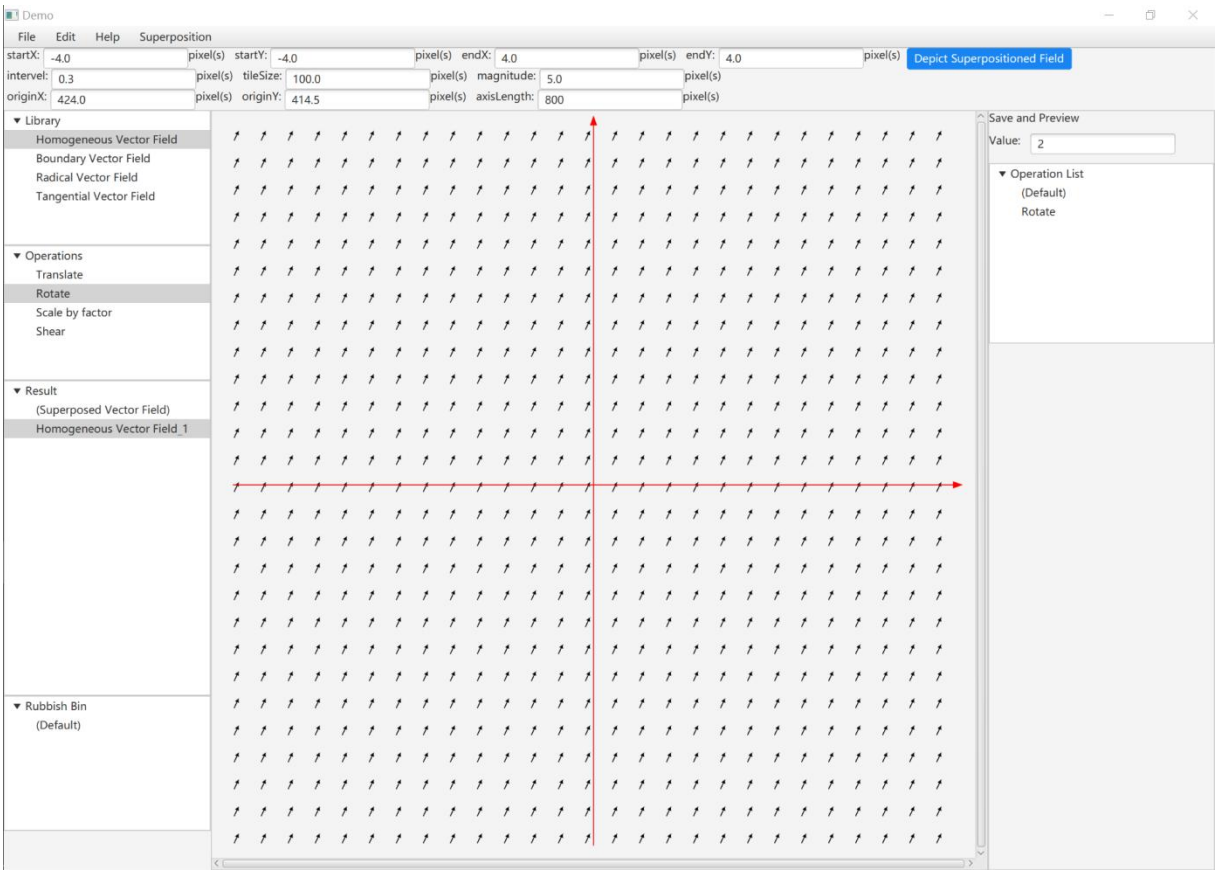


Figure 4.3 The final version of the graphic user interface

4.2 Specialized naming terms

In the author's opinion, pointing out some crucial specialized naming terms of the editor is beneficial for the reader to understand the layout and structure of the graphic user interface because specialized naming terms are important in the transformation from prototypes to classes.

In design, the root of the editor consists of two parts: MyMenu.java's `menuBar` and BorderPane `console`. As Figure 4.4 depicts, the `console` consists of four parts: in the left, a VBox `myLeftBar`; in the middle, a ScrollPane `myPanel`; in the right, a ScrollPane `mySidePanel`; in the top, an HBox `hbox5` for field depiction control.

Figure 4.4 The specialized naming terms for different parts of the editor's graphic user interface

As Figure 4.5 depicts, the `myLeftBar` consists of four TreeView components: `resultTreeView`, `libTreeView`, `optTreeView`, and `binTreeView`. The `myPanel`

consists of a pane: `displayPane`. The `mySidePanel` consists of a VBox component in JavaFX called `detailPaneVBox`. In `detailPaneVBox`, there is a Label component called `detailLabel`, an HBox component called `hb1`, a TreeView component `optInTreeView`, and a VBox component called `operationParametersVBox`. The `mySidePanel` is customized according to different vector fields and operations.

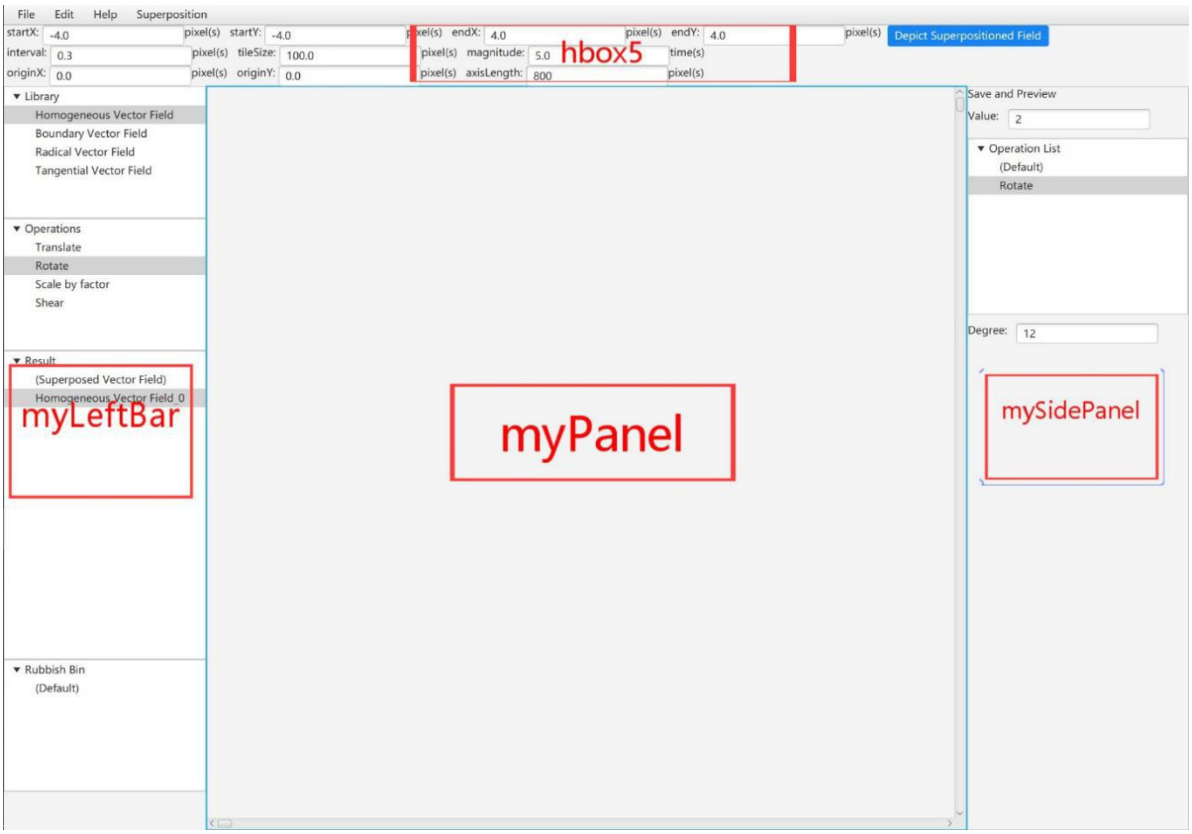


Figure 4.5 The specialized naming terms for different parts of the editor's GUI

4.3 Class Diagram

The author uses the model–view–controller pattern to structure the project as Figure 4.7 shows. This pattern divides the related program logic into three interconnected elements. [12] As Figure 4.6 depicts, the model part directly manages the data, logic, and rules of the application. It receives user input from the controller. The view part renders a presentation of the model in a specific format. Meanwhile, the controller part responds to the user input and interacts with the data model objects. It receives the input, optionally validates it, and then passes the input to the model.

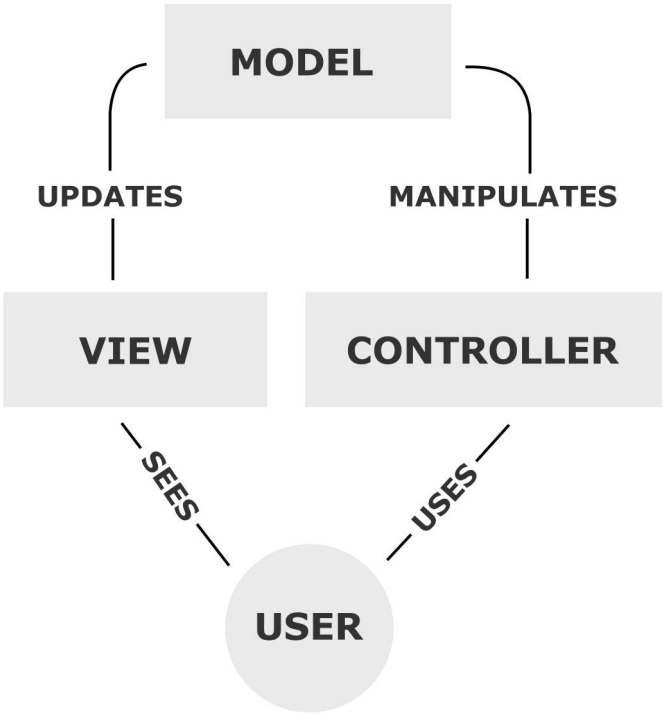


Figure 4.6 The diagram of interactions within the MVC pattern [12]

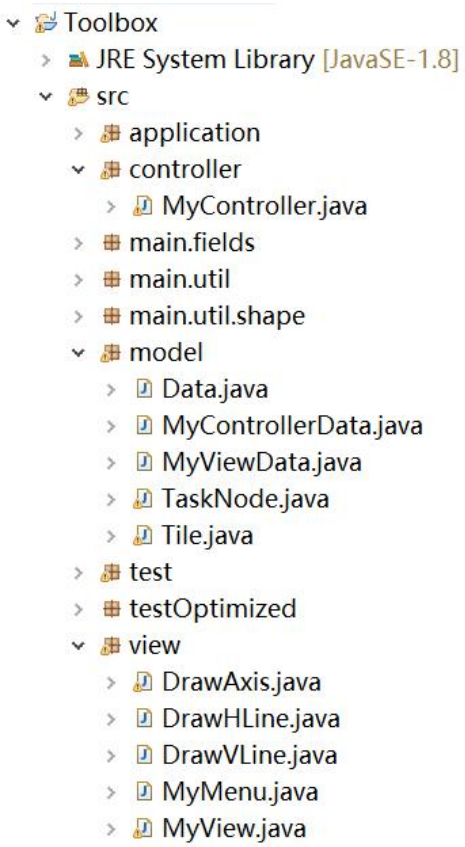


Figure 4.7 The java classes in the packages of controller, model, and view

In this project, for the view part, as Figure 4.8 depicts, there are various draw methods to build the graphic interface. Also, there is a `display(VectorField)` method to display vector fields, `giveInfo(Data, TreeItem<String>)` method to show detailed information in the result tree view, `depict()` method to present the vector field according to the parameters customizable for users.

Figure 4.8 The view part in the class diagram of the editor

For the controller part, as Figure 4.9 depicts, there is an extensive `actionPerformed` method to handle processes followed by the user's interaction with the editor. The `doOperationOnCustomizedVectorField()` method performs operations on customized vector fields and saves them. The `calculateParentVectorField(String)` method calculates the customized vector fields by superposing their sub-vector fields. The `superposition()` method superposes the vector fields in the "Result" tree view. The `drawOptInPanel(Node)` method lets the view part to construct the `operationParametersVBox` in `mySidePanel`. The `generateFileChooser()` method generates a file chooser used in saving and loading the vector fields. The `play` methods build different elementary vector fields' `mySidePanel` and draw different elementary vector fields. The `setCellFac(TreeView<String>)` method, as well as the `drop`, `dragOver`, and `dragDetected` methods, handle the drag and drop operations.

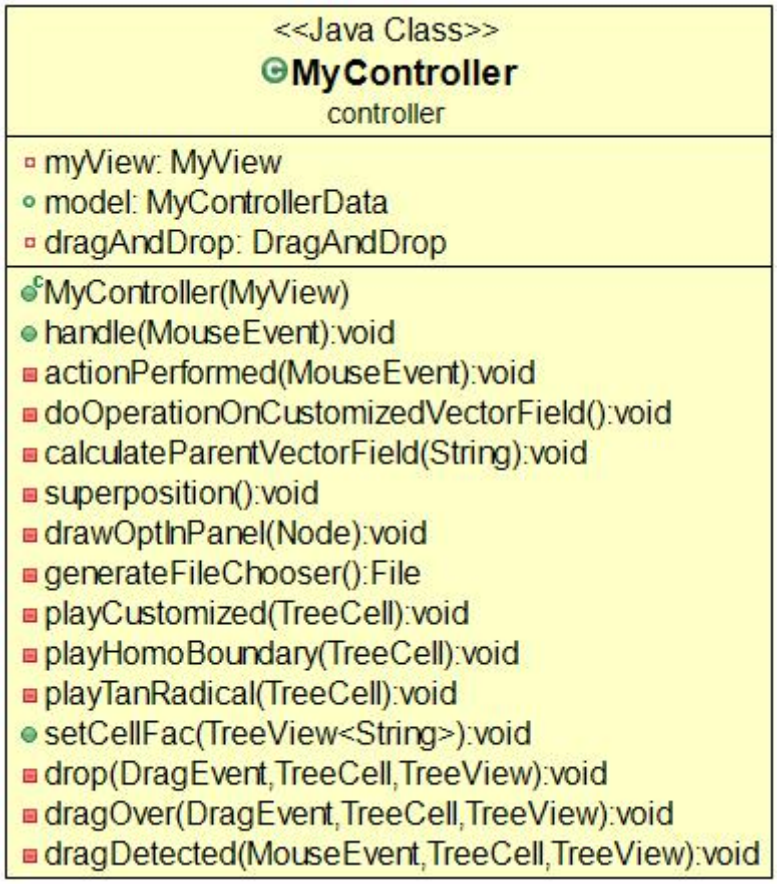


Figure 4.9 The controller part in the class diagram of the editor

For the model part, as Figure 4.10 depicts, the application's dynamic data structure is there, independent of the user interface. The Tile class is for the presentation of vector fields. The MyViewData class is for the depiction of vector fields. The Data class is for vector field storage. The MyControllerData class is for the necessary information needed in the controller. The TaskNode class is for the drag and drop function in the program.

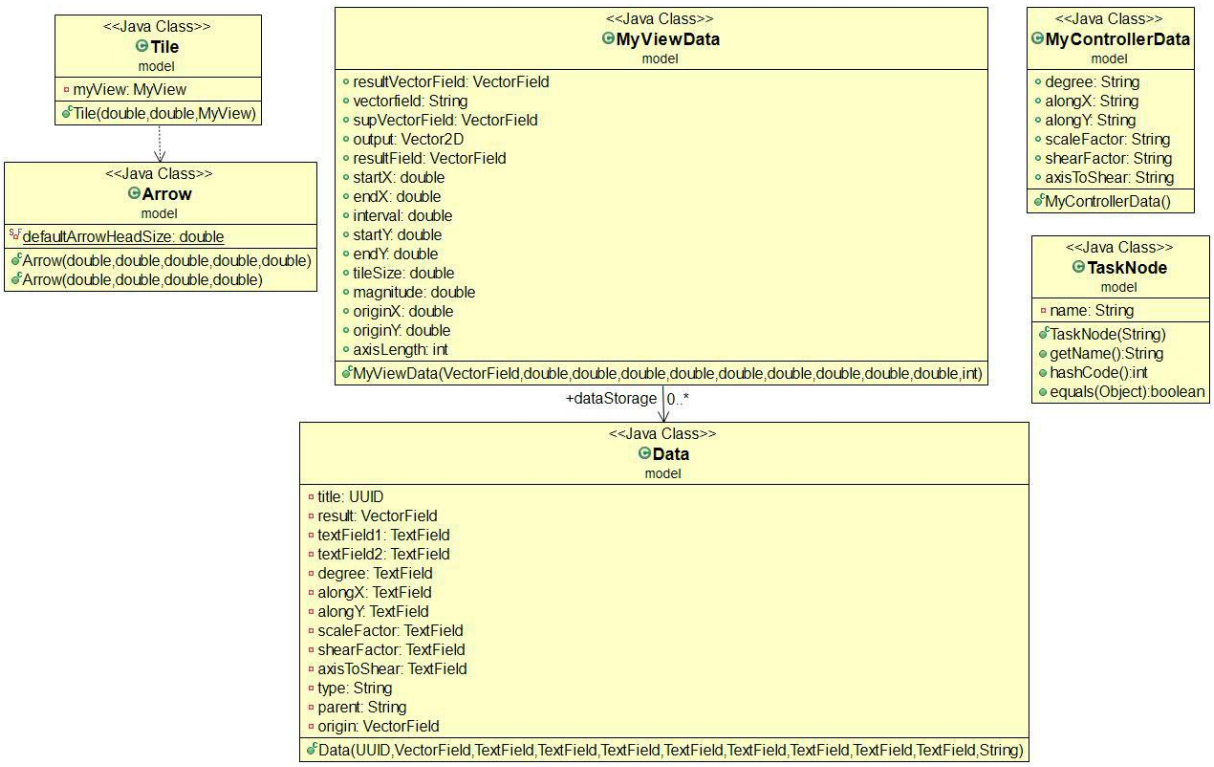


Figure 4.10 The model part in the class diagram of the editor's graphic user interface

4.4 Vector fields visualization

After weeks of research and experience from the toolbox developers, the author found that JavaFX has no usable library for visualizing the vector fields needed in the toolbox. So, the author firstly designs a kit for vector field visualization. As is explained in the Background chapter, a vector field consists of direction vectors and location vectors. Hence, the author designs a Tile class to display each direction vector and translate each tile according to the location vectors. That is to say that the author presents the vector field in tiles.

For the Tile class, the author refactors codes from kn0412 in GitHub [11] to draw an arrow representing the direction vector. As Figure 4.11 depicts, Point B is the end coordinate of the direction vector with coordinate (endX, endY). Suppose there is a straight line crossing point B which is parallel to the x-axis, let $\angle\alpha$ be the angle θ. If $\triangle$BDE is an equilateral triangle, then $\angle\beta$ is $\theta - \frac{\pi}{2}$ - $\frac{\pi}{6}$ and $\angle\gamma$ is $\theta - \frac{\pi}{2} + \frac{\pi}{6}$. Moreover, let the side length of the triangle be `arrowHeadSize`.

In the Cartesian coordinate system, the X-coordinate of D equals $\left(-\frac{1}{2}cos(\theta) + \frac{\sqrt{3}}{2}sin(\theta)\right) * arrowHeadSize + endX$, and the Y-coordinate of D equals $\left(-\frac{1}{2}sin(\theta) - \frac{\sqrt{3}}{2}cos(\theta)\right) * arrowHeadSize + endY$.

The same method can be used similarly to get the coordinate of E. Then, by connecting the points and filling the triangle, the user can draw a perfect arrow. After that, set the arrow's `startX` and `startY` to 0 and the alignment to `Pos.BOTTOM_LEFT`. Finally, translate each tile representing the direction vector field into the coordinates of their corresponding location vector field. The vector field can now be presented in a pane.

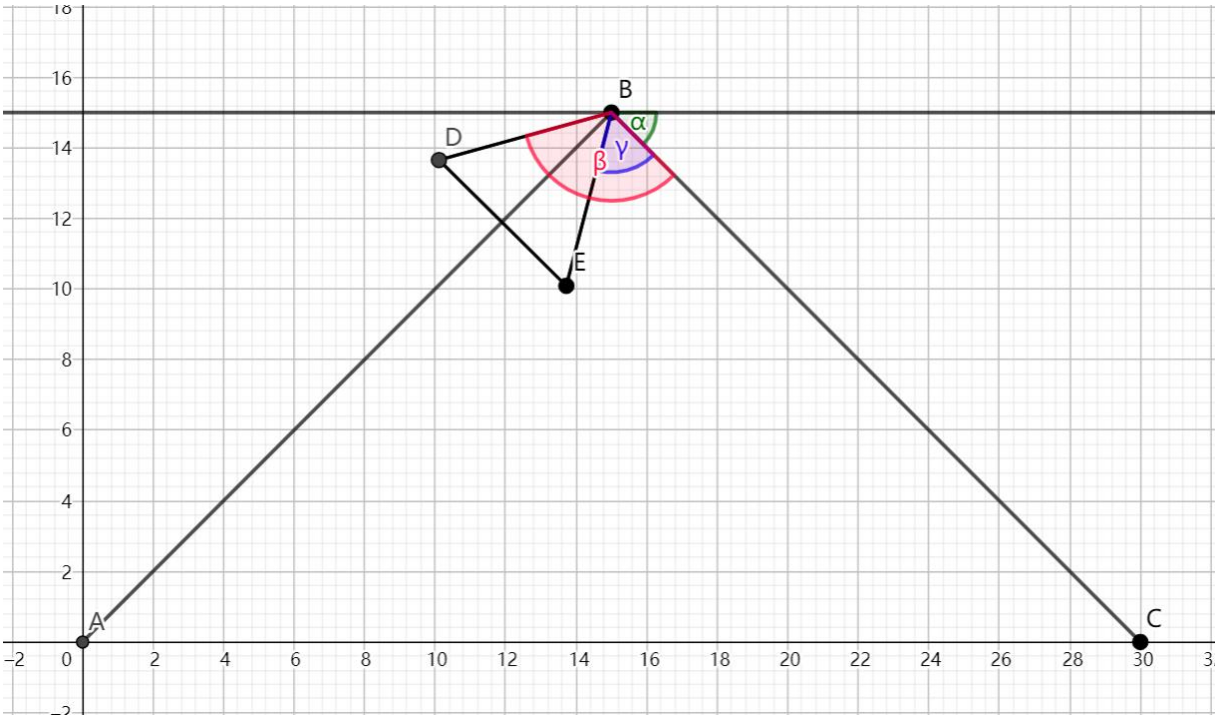


Figure 4.11 The example used to illustrate how the arrow is drawn in the pane

5 Implementation

This chapter documents the implementation of vector fields visualization, the drag and drop function, vector fields' superposition, save and load of the vector fields, and vector field creation and edition.

5.1 Vector fields visualization

As the author has introduced in the Design chapter, the vector fields in the display pane are made up of different tiles, which represent the direction vector fields. In this section, the author explains how to use tiles to present vector fields. As Figure 5.1 shows, the vector fields are visualized in a Cartesian coordinate system. The location vectors are from the coordination (`startX`, `startY`), which has the smallest x and y values to the coordination (`endX`, `endY`) with the largest x and y values. In the toolbox, the location vectors in the Y-axis direction are first generated and then in the X-axis. Each direction vectors start at the location vector's coordinate and ends at the coordinate decided by the `getFunctionValueOptimized` method in the toolbox. After that, each tile is translated according to the origin point and the tile size. This makes the tiles expand evenly in the display pane.

```
542   for (double x = model.startX; x <= model.endX; x += model.interval) {
543       for (double y = model.startY; y <= model.endY; y += model.interval) {
544           model.output = field.getFunctionValueOptimized(x, y);
545           //decide the magnitude of the arrow
546           Tile tile = new Tile(model.output.getX()*model.magnitude,model.output.getY()*model.magnitude, this);
547           //change the starting point in the panel
548           tile.setTranslateX(model.originX+x*model.tileSize);
549           tile.setTranslateY(model.originY+y*model.tileSize);
550           info.displayPane.getChildren().add(tile);
551       }
552   }
```

Figure 5.1 The code on how the vector fields are drawn

There are many parameters deciding how the vector field is displayed in the display pane. Parameters like `startX`, `startY`, `endX`, `endY` decide its X or Y axis scope. The `interval` decides the interval between each coordinate in the X or Y axis range. The `tileSize` decides each direction vector's space. The `magnitude` decides the length of the arrow. The `originX` and `originY` decide the origin's location, and `axisLength` is the length of axes. The default values of these parameters are set in the class MyView, and each depiction will be triggered by clicking on the "Result" tree view's tree items, "Save and Preview," "Superposition," and "Depict Superposed Field." If the user clicks on "Depict Superposed Field," the parameters mentioned above can be changed, and the superposed vector field will be shown. Otherwise, based on the auto adjustment function, the `originX` and `originY` will be automatically adjusted to the pane's center after each depiction.

Axes are drawn as four parts x-axis, y-axis, x-axis tip, and y-axis tip. X and Y axes are extended from `javafx.scene.shape.Line`. Tips are polygons that consist of 6 points. In the author's code, as Figure 5.2 depicts, they are triangles with base-side 8 pixels and height 10 pixels. The color is set to red to make axes stand out from the vector fields. Because the coordination system in the pane starts from the top-left, the line `myView.displayPane.setScaleY(-1);` is added to do a y-axis flip.

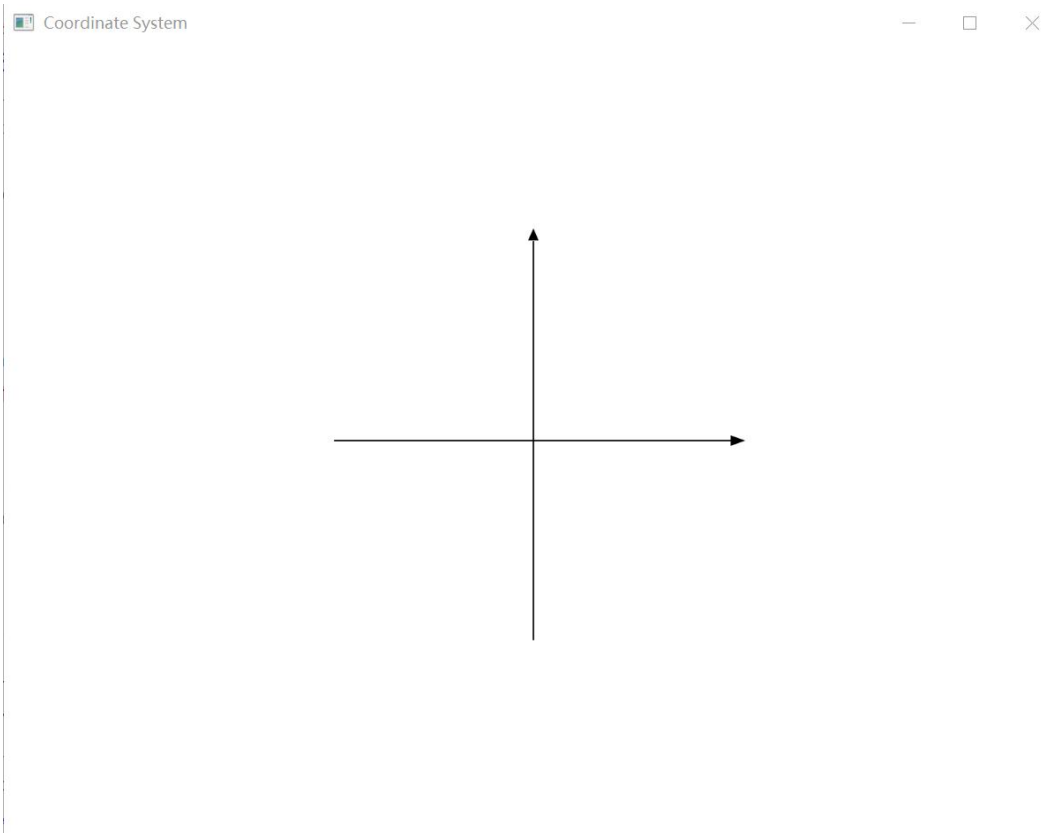


Figure 5.2 The Cartesian coordinate system test and the axes

5.2 Drag and Drop

JavaFX's TreeView is a powerful component, but the code needed to implement some of the finer details is not always obvious. For example, the ability to rearrange tree nodes by dragging and dropping is a feature that users typically expect to implement in a tree component. Also, to improve usability, drag-and-drop images and drag-and-drop position hints should be used. In this section, the author explains how to satisfy all these things.

To enable drag and drop operations on tree views, the author sets a cell factory on each tree view. A cell factory is a renderer of the cell from the cell item. As Figure 5.3 shows, this makes the tree item in each tree view firstly become a tree cell. Then, the user adds a listener on the tree item's property—value. This makes manually generated cells have values, while automatically generated cells have no value. After that, the user sets node properties on each

step in the drag and drop operations. Finally, the user transforms the cell back into the tree item.

```
627    public void setCellFac(TreeView<String> treeView) {
628        treeView.setCellFactory(view -> {
629            TreeCell<String> cell = new TreeCell<>();
630            cell.treeItemProperty().addListener((value, oldValue, newValue) -> {
631                if (newValue != null) {
632                    cell.setText(newValue.getValue());
633                } else {
634                    cell.setText("");
635                }
636            });
637            cell.setOnDragDetected((MouseEvent event) -> dragDetected(event, cell, treeView));
638            cell.setOnDragOver((DragEvent event) -> dragOver(event, cell, treeView));
639            cell.setOnDragDropped((DragEvent event) -> drop(event, cell, treeView));
640            cell.getTreeItem();
641            return cell;
642        });
643    }
```

Figure 5.3 The codes to set cell factory on tree view in JavaFX

5.2.1 DragDetected

Inside dragDetected(), whether a node is actually draggable must be decided. If it is, the underlying value is added to the clipboard content. In the implementation, when the mouse detects the dragging on the tree cell, firstly, the tree cell changes into a tree item. Secondly, it differentiates whether the tree item is the root or not. Thirdly, the method initiates the drag and drop gesture and passes the `TransferMode.MOVE` to the `startDragAndDrop()` method. Finally, the method sets dragged items' value to the drag board.

5.2.2 DragOver

The dragOver() method is triggered when the user is dragging a node over the cell. In this method, whether the node being dragged could be dropped in this location must be decided, and if so, set a style on this cell that yields a visual hint as to where the dragged node will be placed if dropped. In the implementation, as Figure 5.5 shows, when the mouse is over the destination tree view, this method checks whether the drag board has content. Then the method checks whether to drop on the dragged item itself or its tree view. Thirdly, the method checks whether to drop on the "Library" or "Operations" tree view. Finally, the method

checks whether the event transfer mode is `TransferMode.MOVE`.

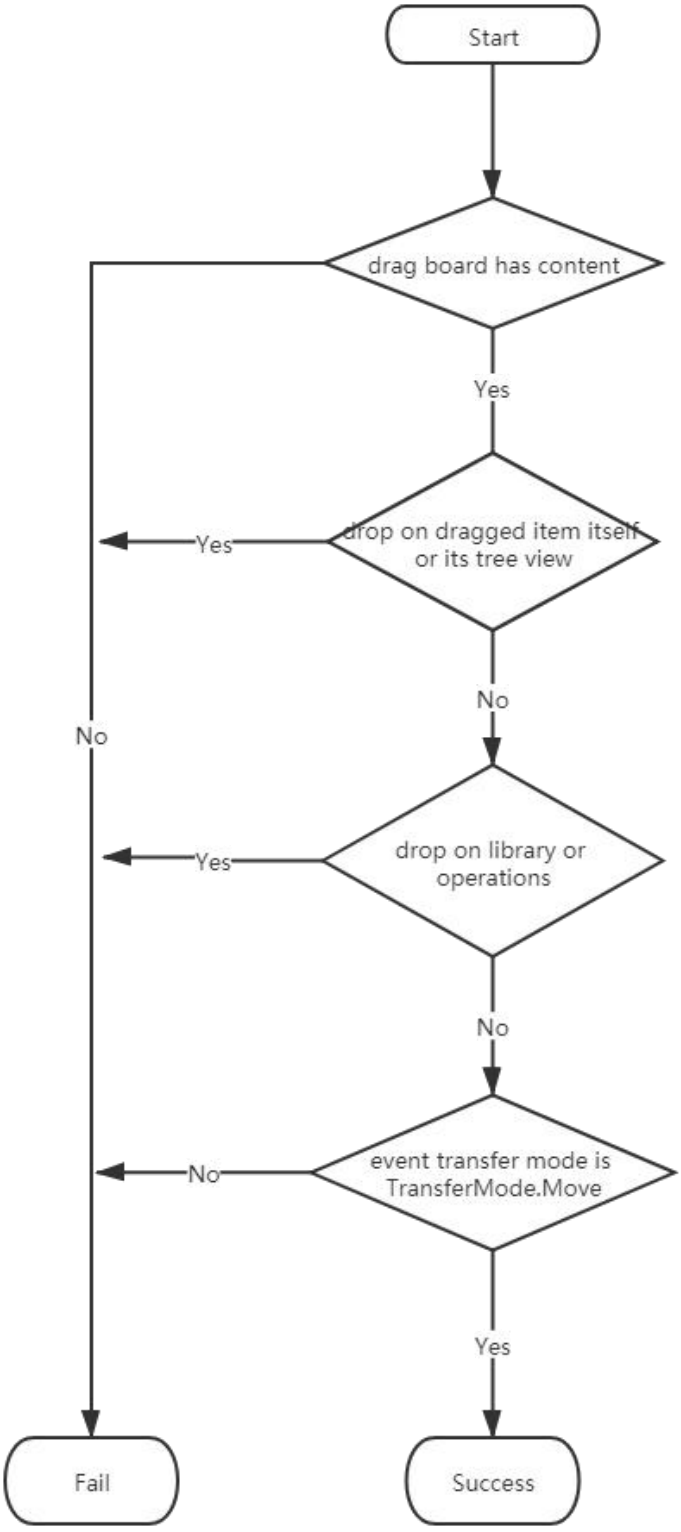


Figure 5.5 Flow chart of the method DragOver

5.2.3 Drop

If a node is actually dropped, the drop() method handles the dropped node both from the old location and to the new location. As Figure 5.6 depicts, two semaphores are needed in drop operation – `success` and `flag`. Any prosperous circumstances will flip the semaphore of `success`. Firstly, the drag board is checked whether there is content in it. Secondly, when a user drags an item from the "Result" tree view or "Operation List" tree view, the method removes the item. This is used when a user drags an item into the "Rubbish Bin. "Thirdly, when a user drags an item from the "Library" tree view, the method generates a tree item with a unique value in the "Result" tree view. This is necessary because the value of the tree item is the primary key in data storage. Fourthly, when the user drags from other tree views, normally, the "Operations" tree view uses the semaphore `flag` to check the redundancy in the "Operation List" tree view. This makes sure that no operation repeats twice in this tree view. If there is no redundancy, then add a new item. Finally, to all branches, indicate that transfer handling of this drag event was completed successfully during a DRAG_DROPPED event handler.

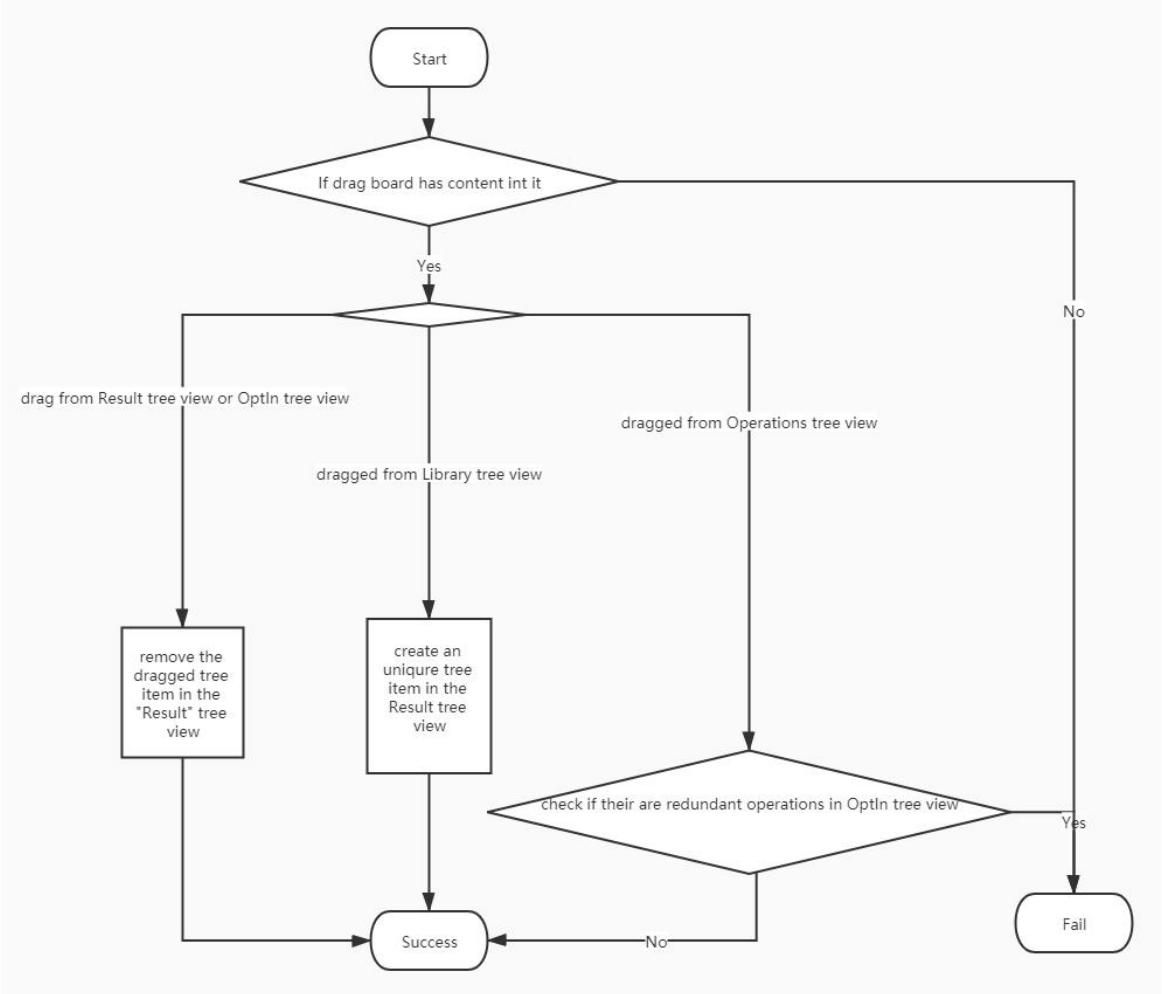
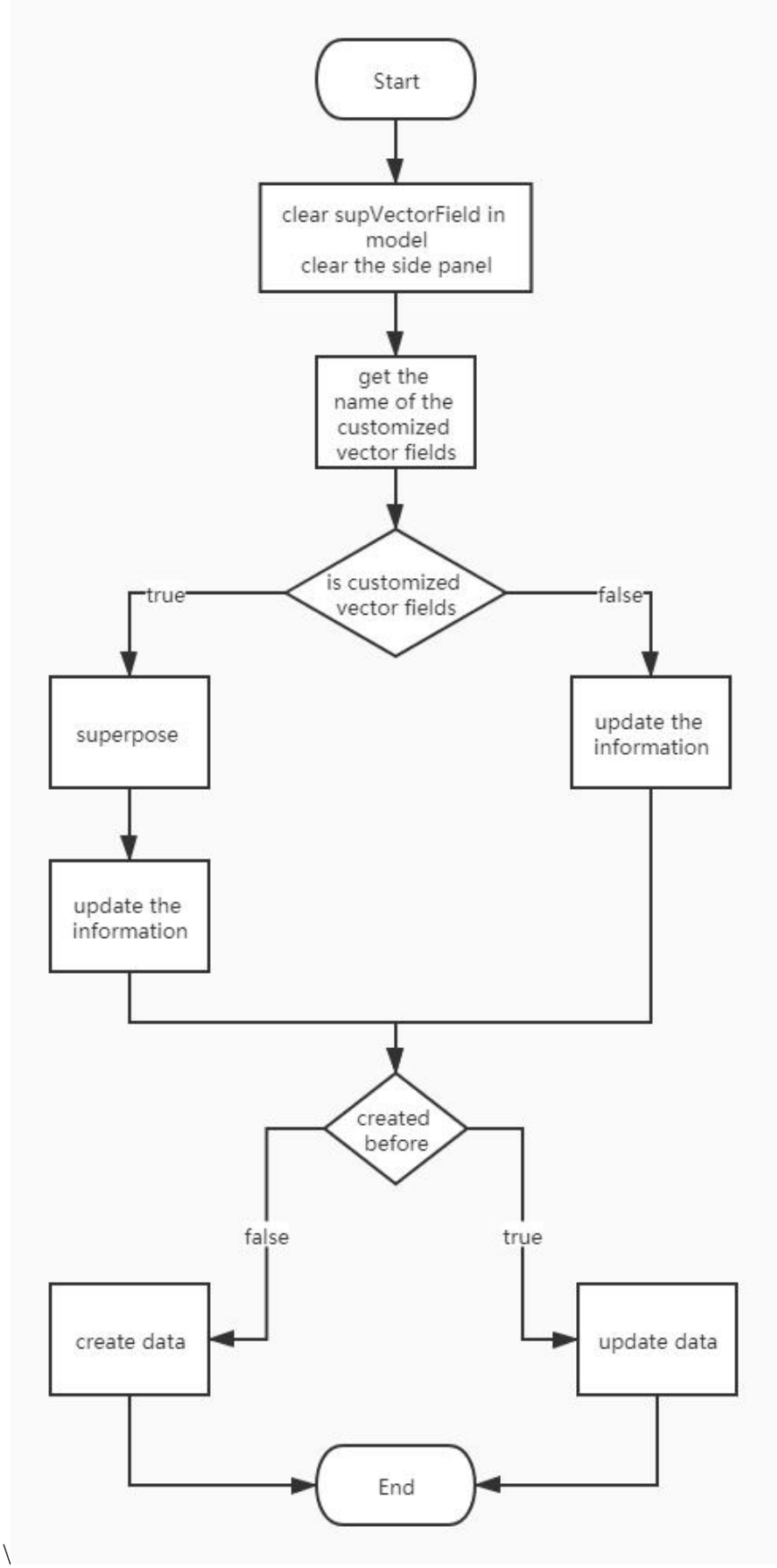


Figure 5.6 The flow chart of the drop function

5.3 Vector field superposition

In this section, the author explains how the vector fields superpose together. As Figure 5.7 depicts, when the user clicks the “Superposition” label, firstly, the `supVectorField` in the model is cleared as well as the side panel. Then, the name list of the customized vector fields is got. It is used to decide whether a certain vector field in the “Result” tree view is a customized vector field. If it is a customized vector field, it will be superposed with other customized vector fields, and then their information is updated. If it is not a customized vector field, its information is only updated. Finally, the editor decides whether the superposed vector field is created before. If it is created before, the editor updates the superposed vector field. Otherwise, the data will be created.

Since only the customized vector fields will be superposed together, getting the customized vector fields before the superposition is essential. The author creates a `calculateParentVectorField` method. The method is triggered by clicking on a customized vector field tree item. As long as one customized vector field tree item in the “Result” tree view is clicked, all children of this customized vector field will be superposed together and generate the result vector field. After that, operations will be performed on the customized vector field if its “Operation List” tree view is not empty. Then, the customized vector field is either created or updated in the data storage list.



\

Figure 5.7 Flow chart of superposition method

5.4 Save and load the vector field

In this section, the author explains how the vector fields are saved and loaded.

When the user clicks the “Save,” a `fileChooser` is generated. The user can set the saving path as the file’s parent. Then, the superposed vector field in the data storage will be saved in JSON format. The author rewrites the `save` function in the toolbox. To make it able to save the customized vector fields.

When the user clicks the “Load,” a `fileChooser` is generated. Firstly, whether the file is null will be judged. Then, the author uses the `VectorField.readInJSON` method in the toolbox. The author rewrites the `readInJson` method in the toolbox used by `VectorField.readInJSON` method. The author adds functions on parsing, so when the readInJson method parses the JSON file. A field list with the elementary vector field types and operations information will also be generated. Hence, when loading the file in the graphic user interface, the customized vector field can be successfully generated both in the “Result” tree view and in the data storage.

5.5 Vector field creation and edition

Create, Read, Update and Delete (CRUD) are the four basic operations of persistent storage in computer programming. In the editor, to create a vector field, the user clicks the vector field tree item, and then a detail pane that the user can change parameters will display. There are four elementary vector fields and customized vector fields to choose from. The author uses a semaphore to differentiate the customized and elementary ones. After the creation, the vector field will be presented by clicking on it.

If the user clicks the tree items in the “Result” tree view, the metadata’s value will be cleared first, including the `degree`, `alongX`, `alongY`, `scaleFactor`, `axisToShear`. Also, the “Operations List” tree view will also be cleared. And then, data will be queried, and metadata will then be filled again.

When the user clicks the “Save and Preview” label, the vector field type will be first checked. For the customized vector field, the changed name will firstly be updated. Then, the tree items in the “Operations List” tree view will be traversed, and then operations will perform on the customized vector field as well as the superposed vector field. After that, the data will be either updated or created. Finally, the vector field will be displayed. For the elementary vector field, the flow is more straightforward. The updated vector field will be displayed, and data will be updated or created.

6 Test

The author firstly exports the editor in a jar file for distribution and then designs black-box test cases basing on different use cases.

6.1 Use case exemplary template

A use case is a way of specifying functional requirements and documenting the behavior of a software system. A use case is a black box that does not mention any details about the implementation of the platform but only the system's response to user input [9]. Use cases are text-based, written in casual language, and intended to be easily understood. Alistair Cockburn defines a use case as a description of a sequence of possible interactions between the system under discussion and its external actors, which are related to a specific goal [10].

As we have seen, the definition says that a use case contains a system, actors, and some interactions that should lead to a goal. The main actor - called the primary actor - is the person (or, in some cases, another system) that interacts with the system described by the use case. These interactions are called scenarios and contain all the actions needed to allow the primary actor to achieve the goal, but just as importantly, also the actions that do not successfully achieve the goal.

Alistair Cockburn proposed a template for describing use cases, which can be found on his website [10]. However, the original template included items that the author does not consider necessary for this project (e.g., priority or performance target) and creates use cases based on the following elements:

Primary Actor The person that interacts with the system.

Preconditions Certain factors need to be fulfilled to allow the primary actor to reach the goal, e.g., there needs to be a vector field to let users edit.

Success End Condition A description of the state of the system after the primary actor reached the goal, generally the purpose of the use case, e.g., the display panel is updated soon after saving.

Failed End Condition Describe the system's state after the primary actor missed the goal, e.g., the display panel remains the same if the parameters are not saved.

Main Success Scenario A detailed step-by-step description of the user interacting with the system, leads to the success end condition, e.g., describing how the user can successfully edit a vector field.

Extensions List all steps of the main scenario that can happen differently.

Sub-Variations An alternation of a step from the main scenario.

6.2 Test case 1：Create an elementary vector field

Use case

Primary Actor: User

Preconditions: The program is running

Success End Condition: The needed elementary vector field is successfully displayed and saved.

Failed End Condition: Without saving, the display panel has no change.

Main Success Scenario:

1. From the “Library” tree view, drag a “Homogeneous Vector Field” into the “Result” tree view.

2. Click on the newly created tree item.

3. Change value and click “Save and Preview.” The result will be displayed in the display pane.

4. Double click the created tree item to auto-adjust the display scope.

Extensions:

Change in Step1: drag a “Boundary Vector Field,” “Radical Vector Field,” “Tangential Vector Field.”

Test Case

See more details in Appendix A

General Feedback

As test case 1 in Appendix A shows, this use case functions well. Nevertheless, there are user evaluations on better user interaction. In the main success scenario, there are four steps. Step one and step two can emerge together. Also, step three and step four can emerge together, and then double click in step four can be reduced to one click. This feedback is discussed in the Evaluation chapter.

Screenshots

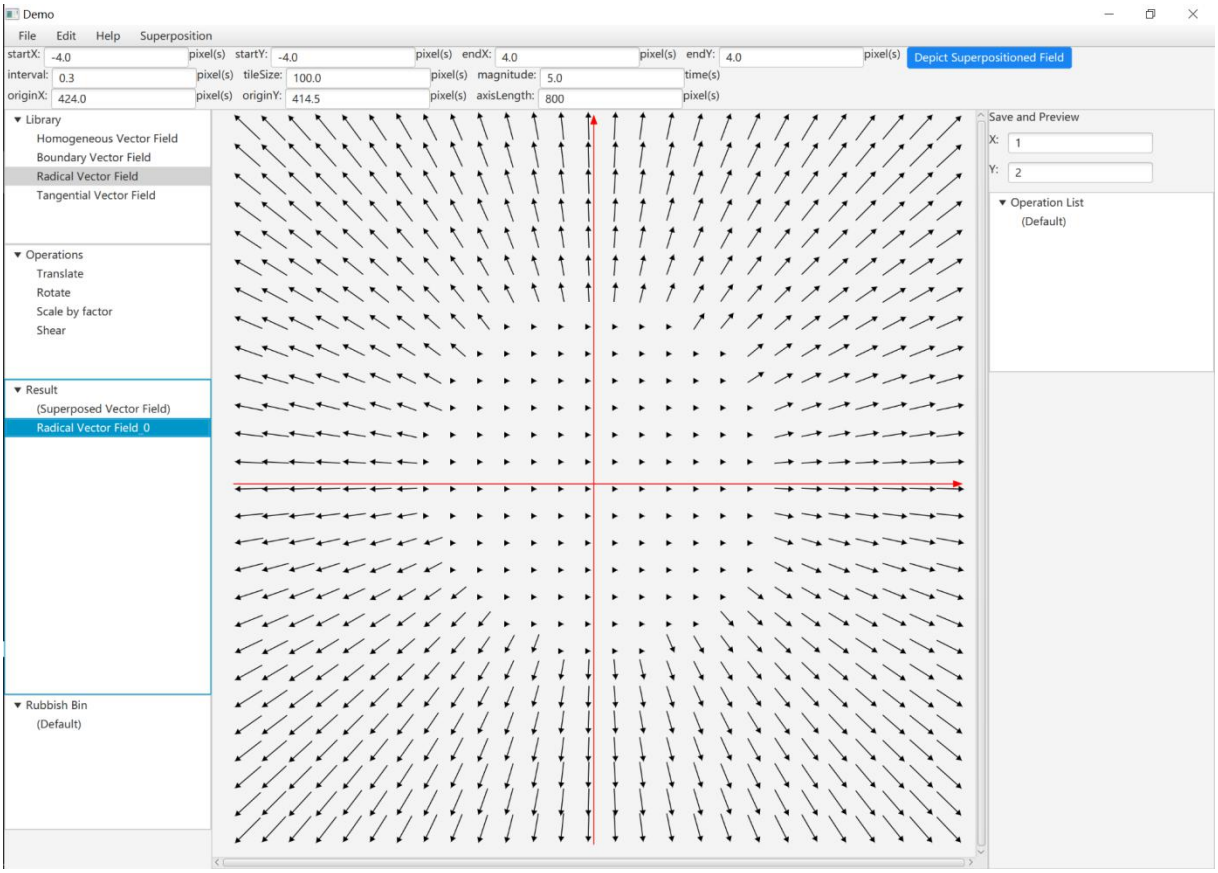


Figure 6.1 One possible outcome of Use case 1 in the success end condition (Test case 1: The user creates a radical vector field)

6.3 Test case 2：Perform operations on an elementary vector field

Use case

Primary Actor: User

Preconditions: An elementary vector field exists and displays.

Success End Condition: The vector field successfully performs the operations.

Failed End Condition: Without saving or valid data, no operation is performed.

Main Success Scenario:

1. From the “Operations” tree view, drag “Rotate” into the “Operation List” tree view in the detail pane.

2. Click on the newly generated “Rotate” tree item and change the degree value.

3. Click on “Save and Preview” to see the changes.

Extensions:

Change in Step 1: Different drag operations like “Translate, “Scale by Factor,” “Shear” into the “Operation List” tree view

Change in Step 2: Do multiple operations on the same elementary vector fields.

Sub-Variations: Change in Step 3, click on “Save and Preview” twice.

Test case

See more details in Appendix A.

General Feedback

As test case 2 in Appendix A shows, this use case functions well.

Screenshots

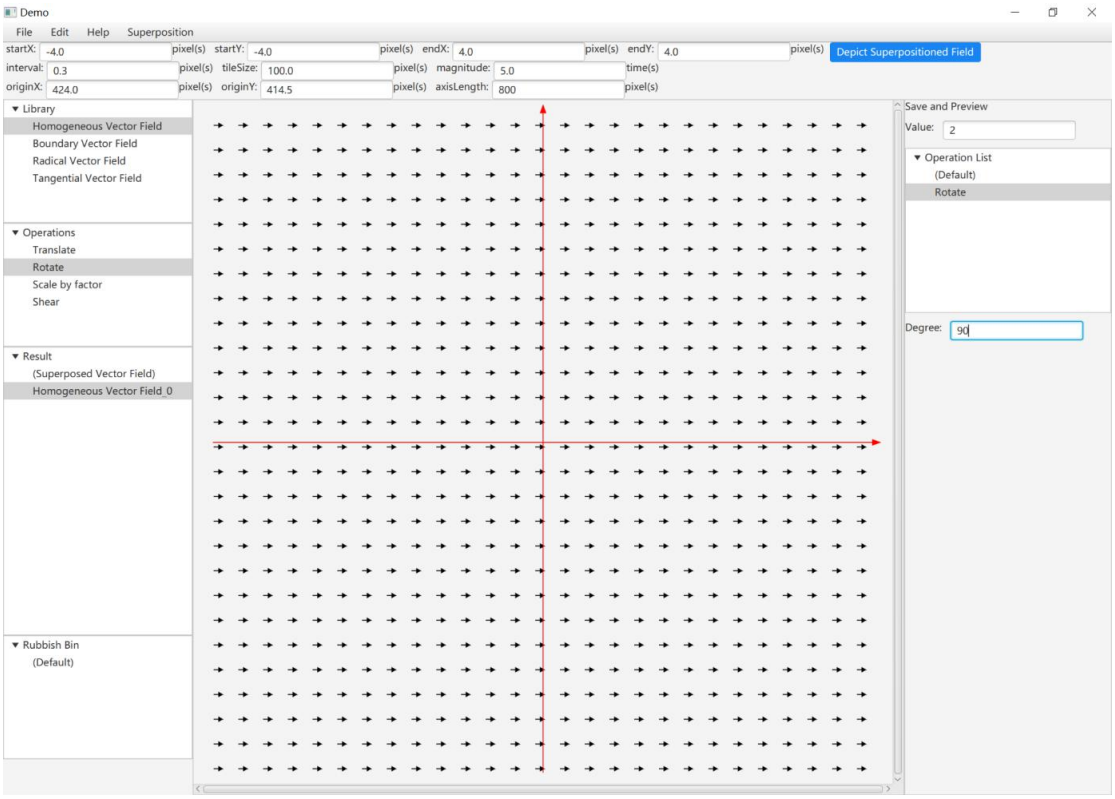


Figure 6.2 One possible outcome of User case 2 on the success end condition (Test case 2: The user performs Rotate on a homogeneous vector field with value of 2)

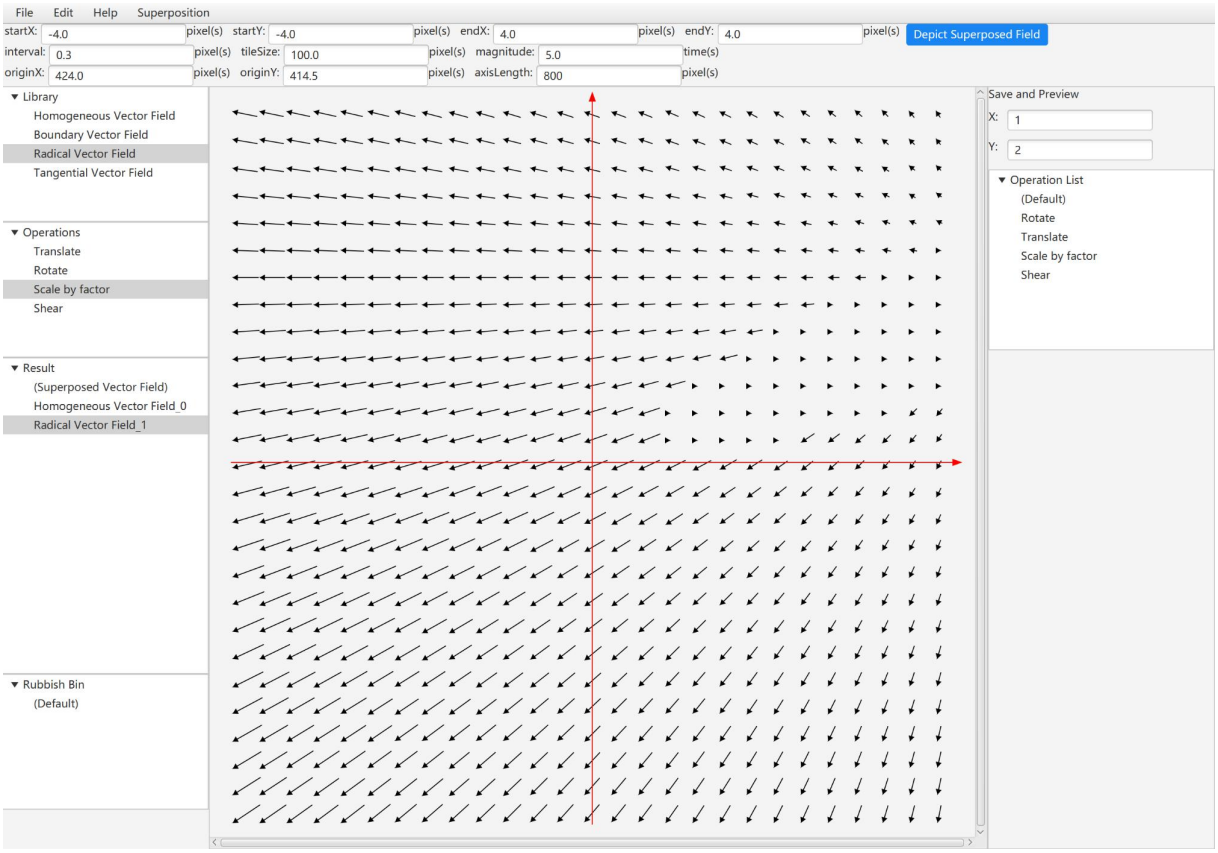


Figure 6.3 One possible outcome of Use Case 2's extensions on the success end condition (Test case 2: The user performs multiple operations on a radical vector field with x equals 1 and y equals 2)

6.4 Test case 3：Change properties of an elementary vector field

Use case

Primary Actor: User

Preconditions: An elementary vector field exists and displays.

Success End Condition: The vector field successfully performs the changes.

Failed End Condition: Without saving or valid data, no operation is performed, and the vector field is not changed.

Main Success Scenario:

1. Perform changes on operations and vector field properties.
2. Click on "Save and Preview" to see the changes.

Extensions:

Change in Step 1: Operations can be deleted.

Test case

See more details in Appendix A.

General Feedback

As test case 3 in Appendix A shows, this use case functions well.

Screenshots

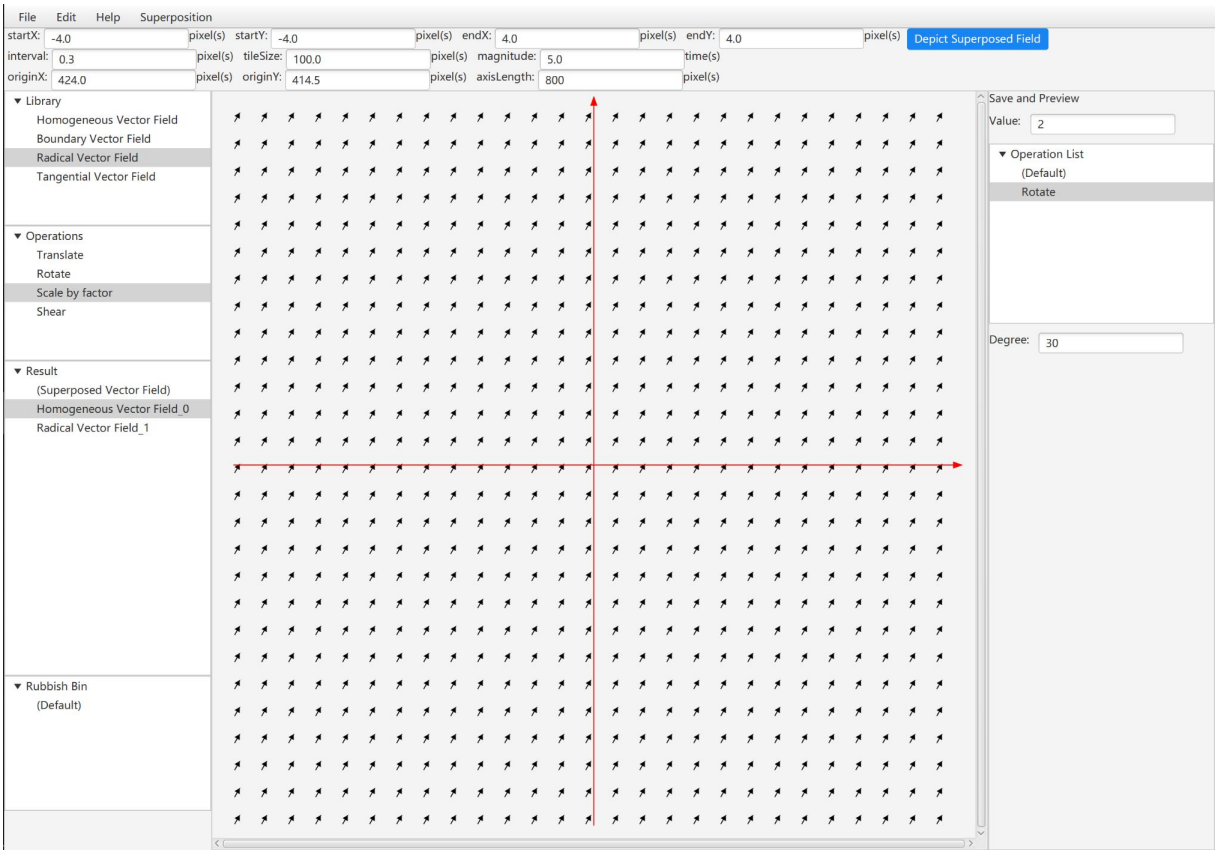


Figure 6.4 One possible precondition of Use case 3 in screenshot

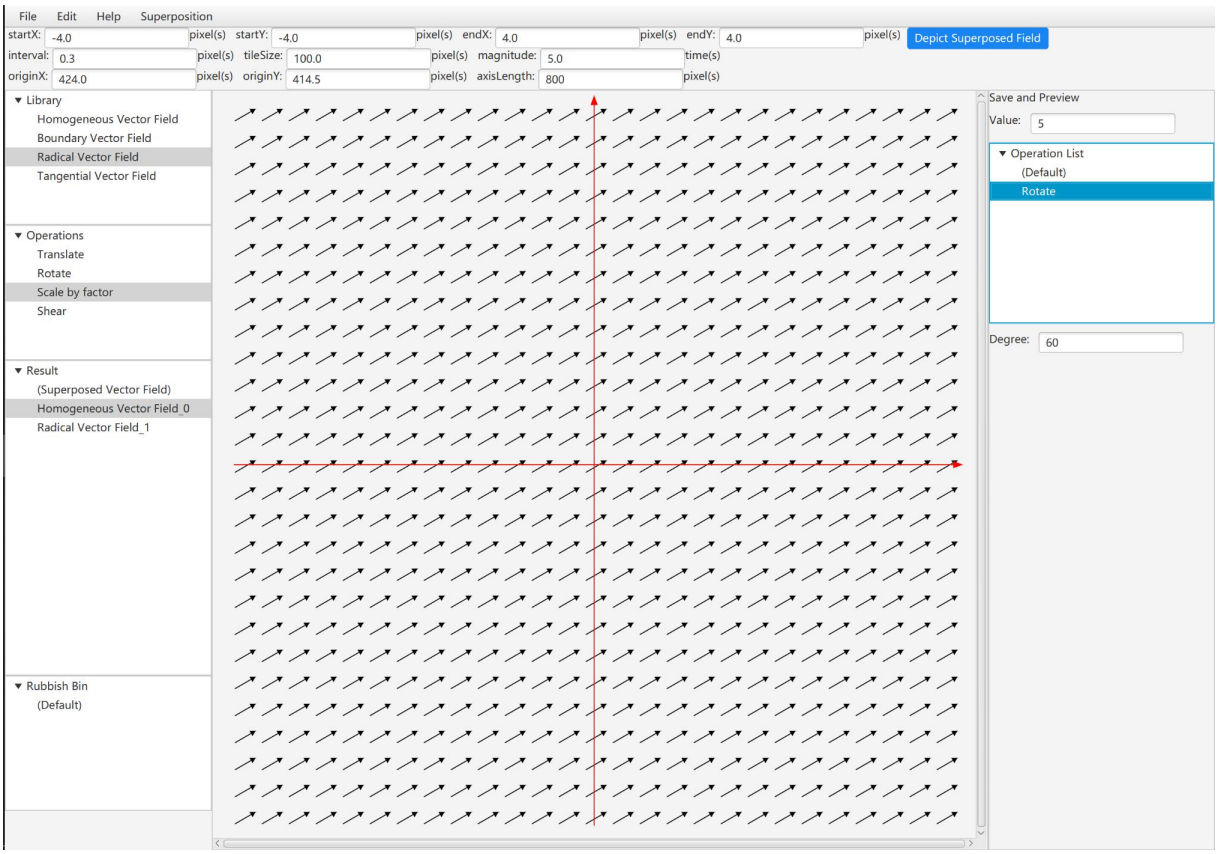


Figure 6.5 The following outcome of Use case 3 with Figure 6.4 as the precondition on success end condition (Test case 3: The user changes properties of an homogeneous vector field with value 2 and rotate 30°)

6.5 Test case 4：Delete an elementary vector field

Use case

Primary Actor: User

Preconditions: An elementary vector field exists and is superposed.

Success End Condition: The vector field is deleted.

Failed End Condition: The vector field stays the same.

Main Success Scenario:

1. Drag the vector field from the “Result” tree view into the “Rubbish Bin” tree view and click “Superposition.”

2. Click the “(Superposed Vector Field)” tree item in the “Result” tree view to verify the vector field has been deleted.

Test case

See more details in Appendix A.

General Feedback

As test case 4 in Appendix A shows, this use case functions well.

Screenshots

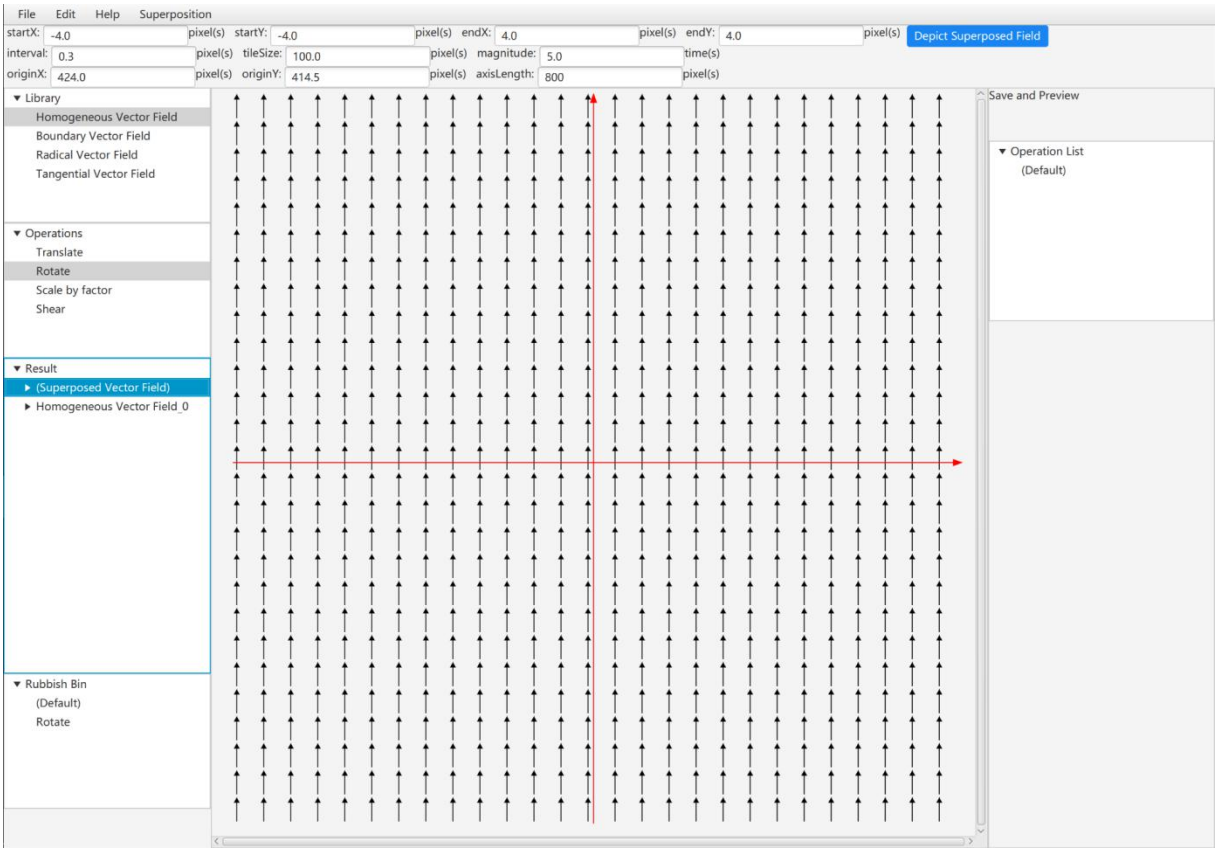


Figure 6.6 One possible precondition of Use Case 4 in screenshot

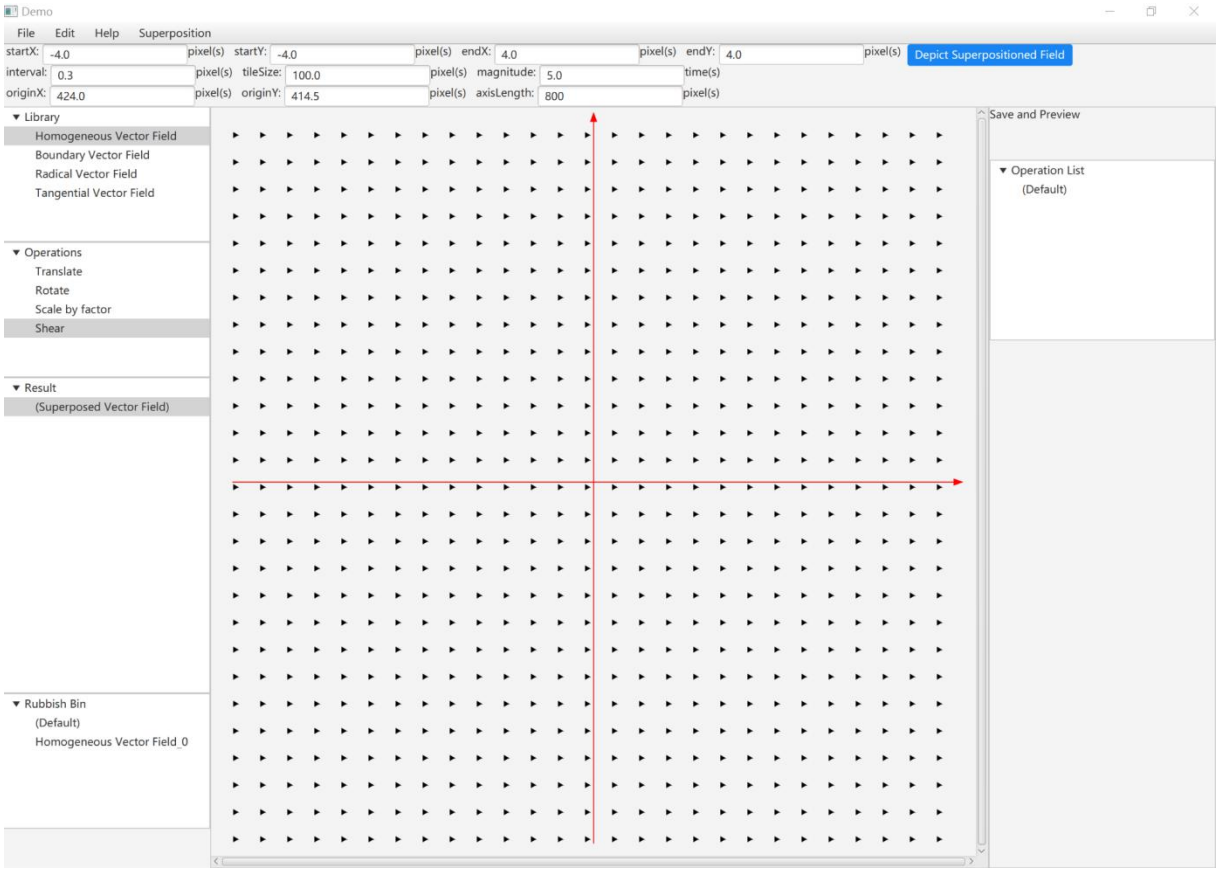


Figure 6.7 The following outcome of Use case 4 with Figure 6.6 as the precondition on success end condition (Test case 4: The user deletes a homogeneous vector field with value 5)

6.6 Test case 5：Recover an elementary vector field

Use case

Primary Actor: User

Preconditions: An elementary vector field is deleted.

Success End Condition: The vector field is recovered.

Failed End Condition: The vector field stays in the “Rubbish Bin” tree view.

Main Success Scenario:

1. Drag the vector field from the “Rubbish Bin” tree view into the “Result” tree view.
2. Click the vector field, and the vector field successfully displays.

Test case

See more details in Appendix A.

General Feedback

As test case 5 in Appendix A shows, this use case functions well.

Screenshots

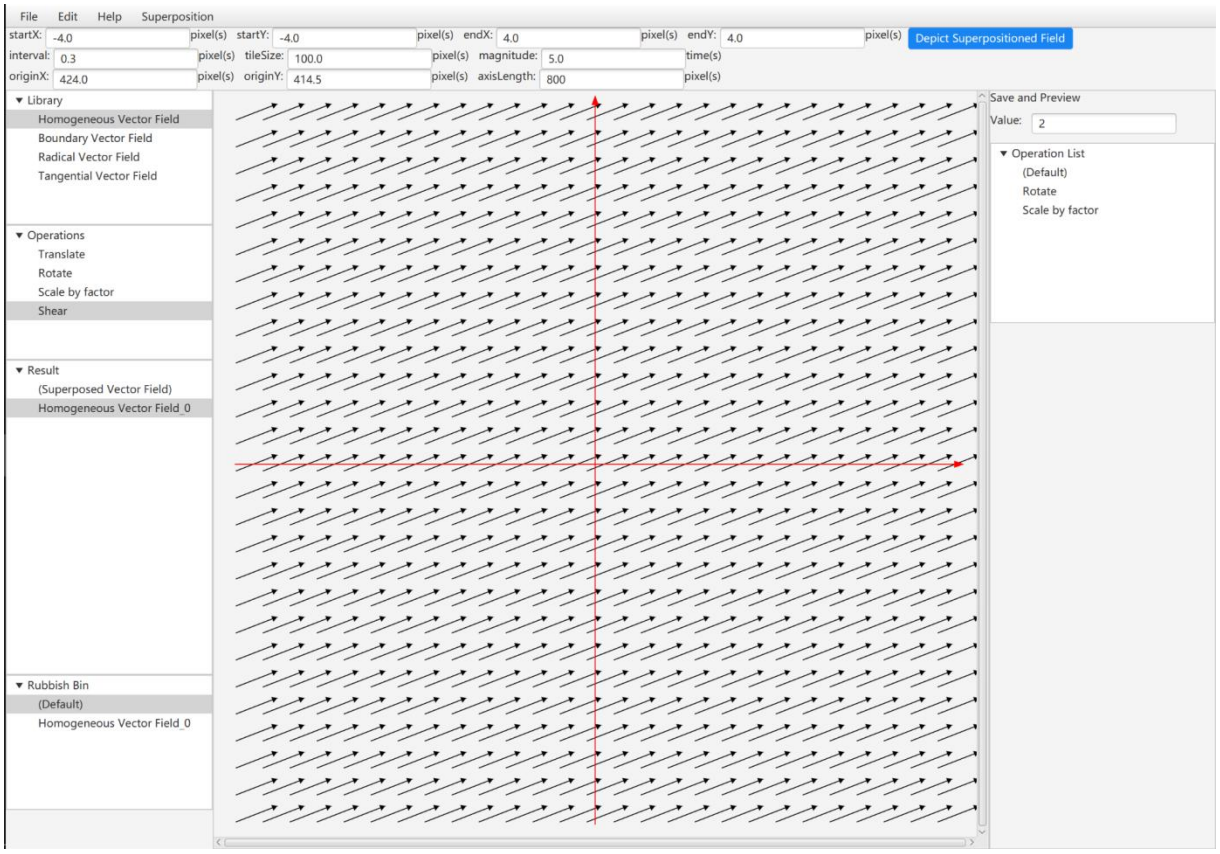


Figure 6.8 One possible outcome of Use case 5 on success end condition (Test case 5: The user recovers the deleted homogeneous vector field in Test case 4)

6.7 Test case 6: Create a customized vector field

Use case

Primary Actor: User

Preconditions: The program is running

Success End Condition: The customized elementary vector field is successfully displayed and saved.

Failed End Condition: Without superposition, clicking on the superposed vector field tree item causes no change.

Main Success Scenario:

1. Create several elementary vector fields.

2. Click on the “Superposition.”

3. Click on the “Superposed vector field” tree item, and the customized vector field is displayed.

Extensions:

Change in Step1: Load a customized vector field.

Test case

See more details in Appendix A.

General Feedback

As test case 6 in Appendix A shows, this use case functions well.

Screenshots

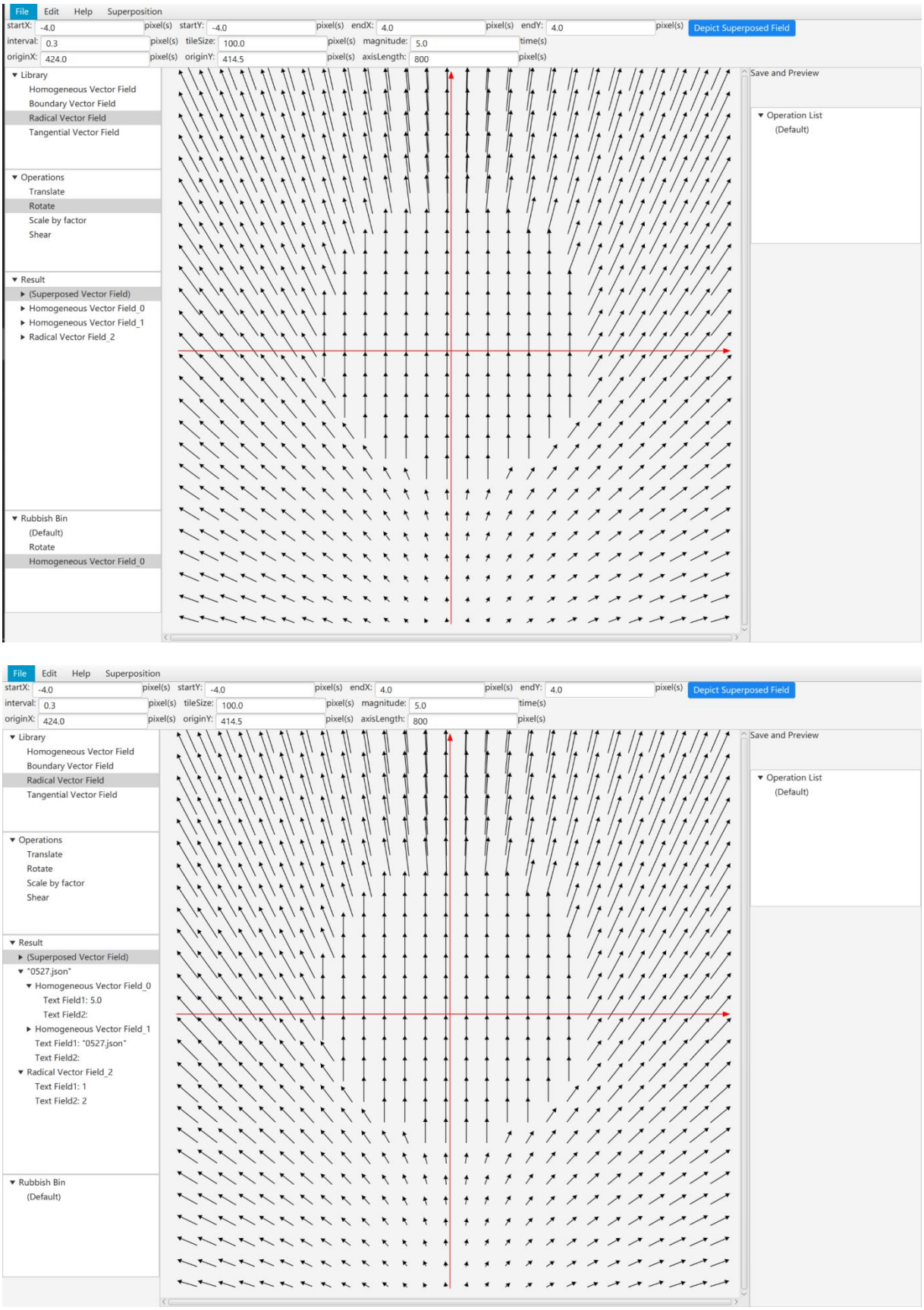


Figure 6.9 Possible outcomes of Use case 6 on success end condition (Test case 6)

6.8 Test case 7: Perform operations on the superposed vector field

Use case

Primary Actor: User

Preconditions: Vector fields are superposed together.

Success End Condition: The superposed vector field changes successfully.

Failed End Condition: The superposed vector field doesn't change.

Main Success Scenario:

1. Click on the "Superposed Vector Field."
2. Perform operations on the superposed vector field.
3. Click on the "Save and Preview" and click on the "Superposed Vector Field" again.

Sub-Variations: Change in Step 2: delete operations on the superposed vector field.

Test case

See more details in Appendix A.

General Feedback

As test case 7 in Appendix A shows, this use case functions well.

Screenshots

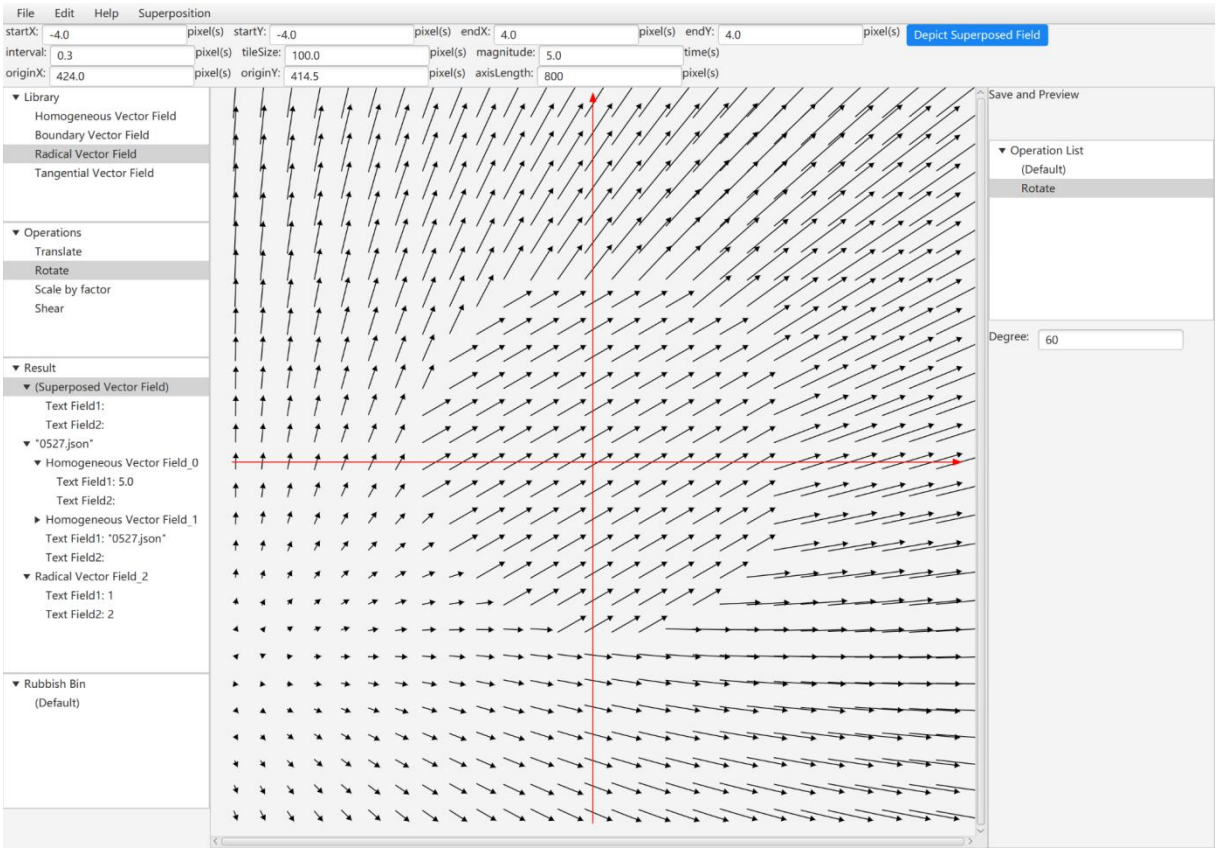


Figure 6.10 Outcome of Use case 9 on success end condition under Figure 6.9 as the precondition (Test case 7: The user performs operations on the superposed vector field with test case 6 outcome as precondition)

6.9 Test case 8：Save customized vector fields

Use case

Primary Actor: User

Preconditions: A superposed vector field exists in the “Result” tree view

Success End Condition: A JSON file is generated.

Failed End Condition: No file is generated, and the result tree view stays the same.

Main Success Scenario:

1. Click the “File” and the “Save” s in it.
2. Use file chooser to save the file.

Test case

See more details in Appendix A.

General Feedback

As test case 8 in Appendix A shows, this use case functions well.

Screenshots

Figure 6.11 A possible outcome of Use case 10 on success end condition (Test case 8: The user saves a customized vector field rotated 90 ° which is superposition of a homogeneous vector field with value of 10 and radical vector field with x equals 1 and y equals 2.)

6.10 Test case 9：Load customized vector fields

Use case

Primary Actor: User

Preconditions: The program is running.

Success End Condition: The customized vector field is added in the “Result” tree view.

Failed End Condition: The “Result” tree view stays the same

Main Success Scenario:

1. Click the “File” and the “Load” in it.
2. Use file chooser to choose the file.
3. Click all sub-vector fields and save.
4. Click the customized vector field.

Test case

See more details in Appendix A.

General Feedback

As test case 9 in Appendix A shows, this use case functions well. But, the usability is bad because the user has to click on all sub-vector fields to get the loaded customized vector field. And the information of vector fields won't update unless the "Superposition" is pressed. The author will discuss these in the Evaluation chapter.

Screenshots

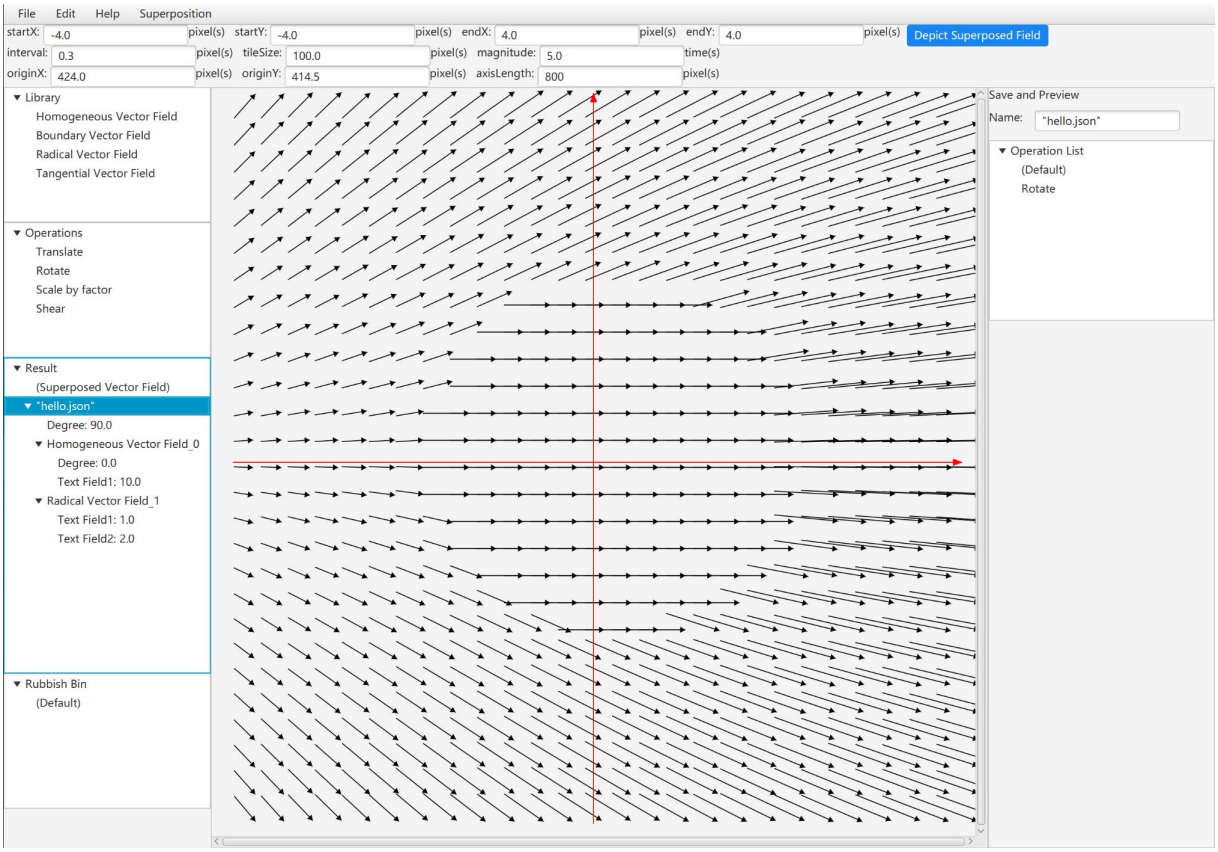


Figure 6.12 A possible outcome of Use case 9 on success condition (Test case 9: The user loads the customized vector field in test case 8)

6.11 Test case 10: Perform operations on a customized vector field

Use case

Primary Actor: User

Preconditions: A customized vector field exists.

Success End Condition: The customized elementary vector field is successfully changed.

Failed End Condition: The vector field has no change.

Main Success Scenario:

1. Click on the customized vector field.

2. Add operations on the customized vector field.

3. Click “Save and Preview.”

Sub-Variations: Change in step 2: Delete operations on the customized vector field.

Test case

See more details in Appendix A.

General Feedback

As test case 10 in Appendix A shows, this use case failed. Because after deleting the operations from the customized vector field, the customized vector field is not updated.

Screenshots

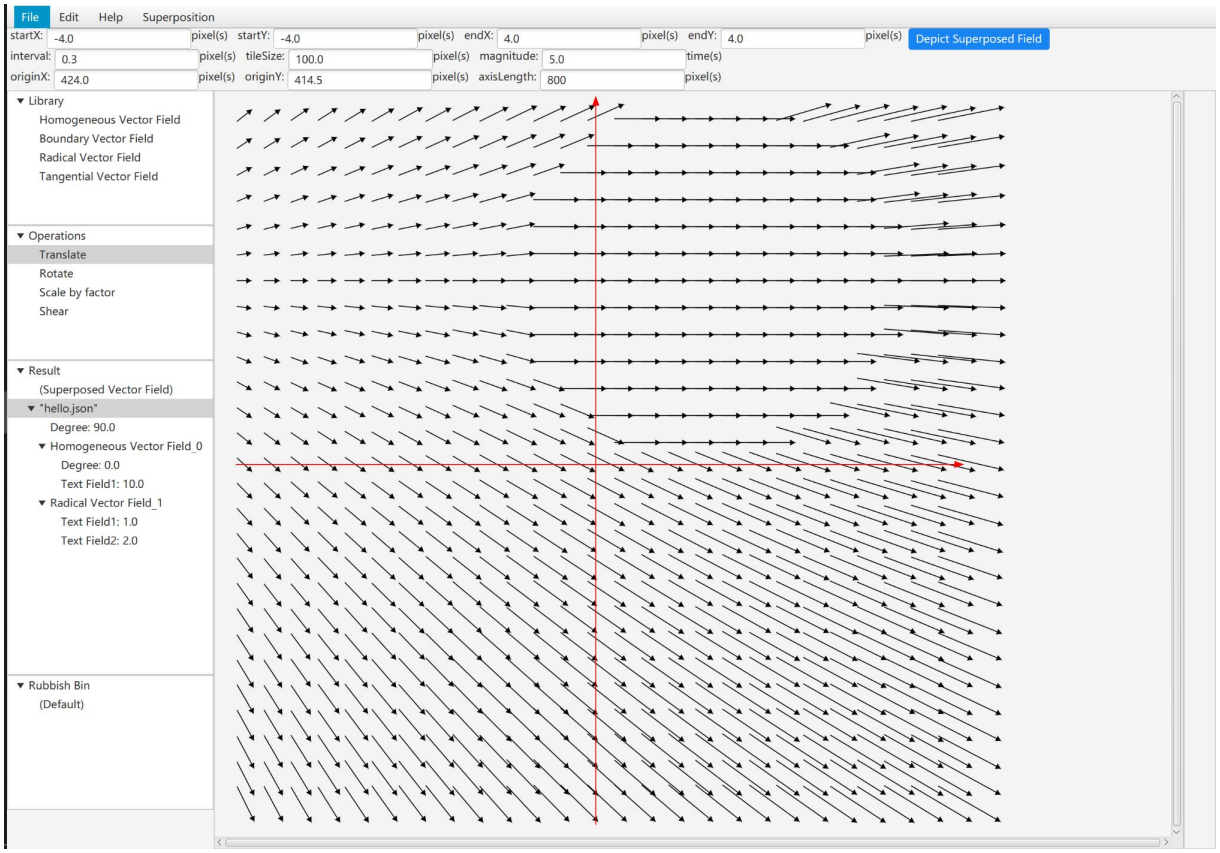


Figure 6.13 Outcome of Use case 10 on success end condition under Figure 6.12 as the precondition (Test case 10: The user performs translate on the customized vector field in test case 8)

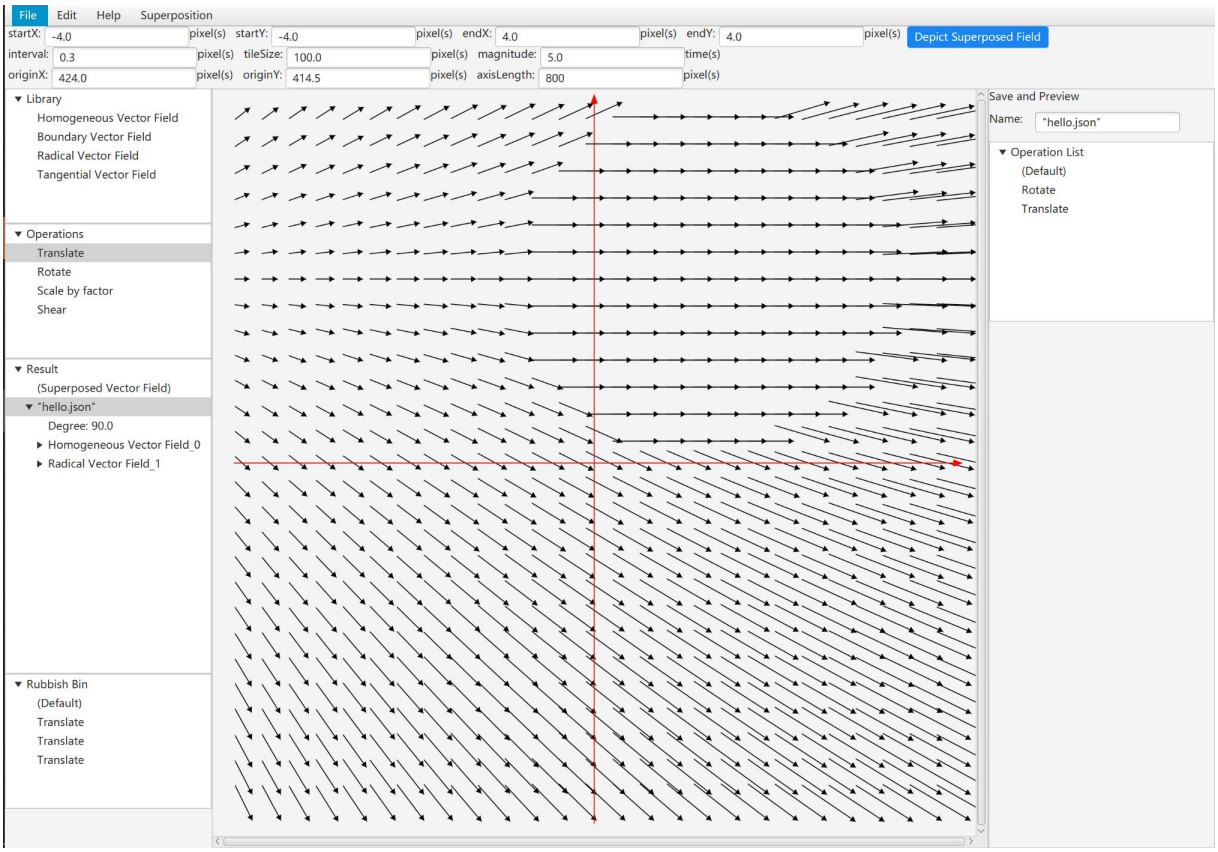


Figure 6.14 Outcome of Use case 10 on fail end condition in Test case 10's sub variation (Test case 10: The user deletes "translate" from the customized vector field as Figure 6.13 as precondition)

6.12 Test case 11: Change sub-vector fields' properties of a customized vector field

Use case

Primary Actor: User

Preconditions: A customized vector field exists.

Success End Condition: The customized elementary vector field is successfully changed.

Failed End Condition: The vector field has no change.

Main Success Scenario:

1. Click on the sub-vector fields and change the sub-vector fields' properties of the customized vector field and save them.

2. Click on the customized vector field again.

Test case

See more details in Appendix A.

General Feedback

As test case 11 in Appendix A shows, this use case functions well.

Screenshots

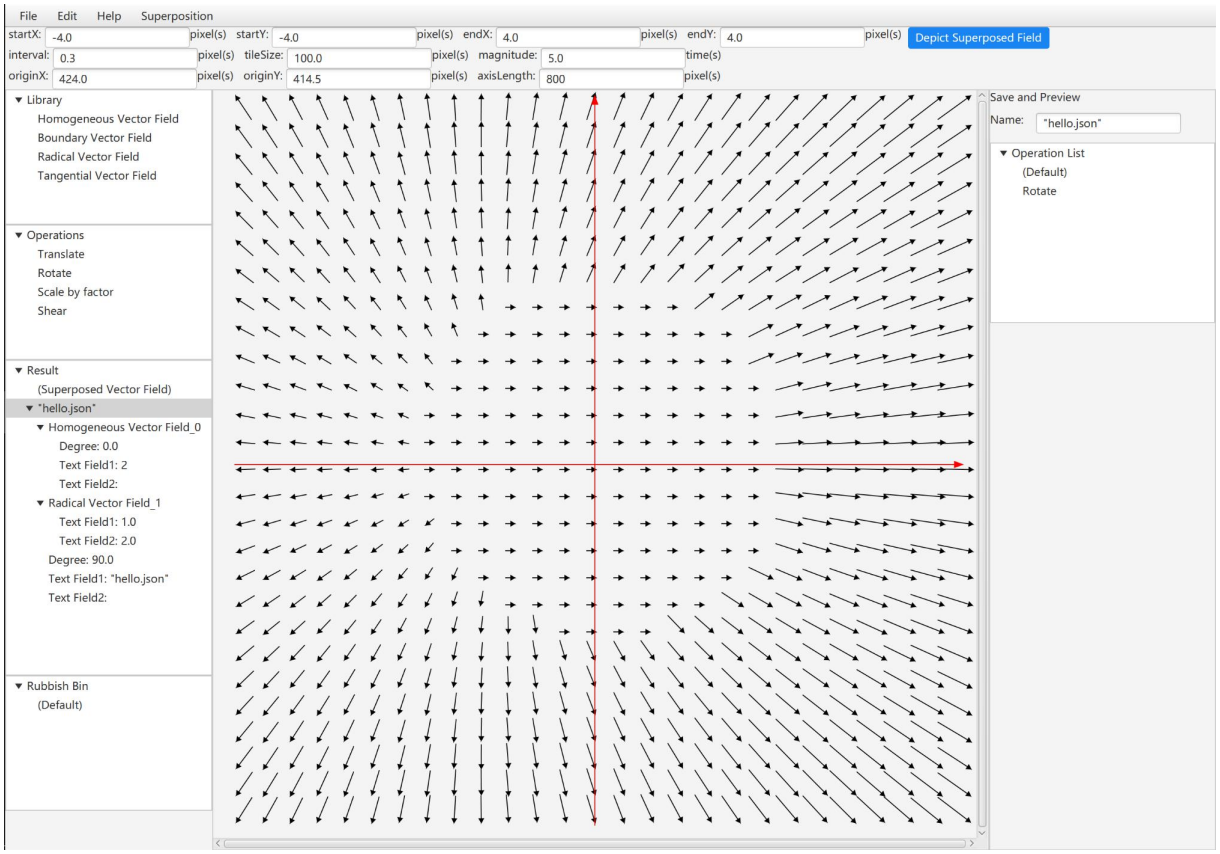


Figure 6.15 A possible outcome of Use case 11 on success end condition (Test case 11: The user changes the customized vector field in test case 8's sub-vector fields' properties)

6.13 Test case 12: Change the name of a customized vector field

Use case

Primary Actor: User

Preconditions: A customized vector field exists.

Success End Condition: The name of the vector field is changed.

Failed End Condition: The name of the vector field has no change.

Main Success Scenario:

1. Click on the customized vector field.
2. Change the name of the vector field.
3. Click on “Save and Preview.”

Test case

See more details in Appendix A.

General Feedback

As test case 12 in Appendix A shows, this use case functions well.

Screenshots

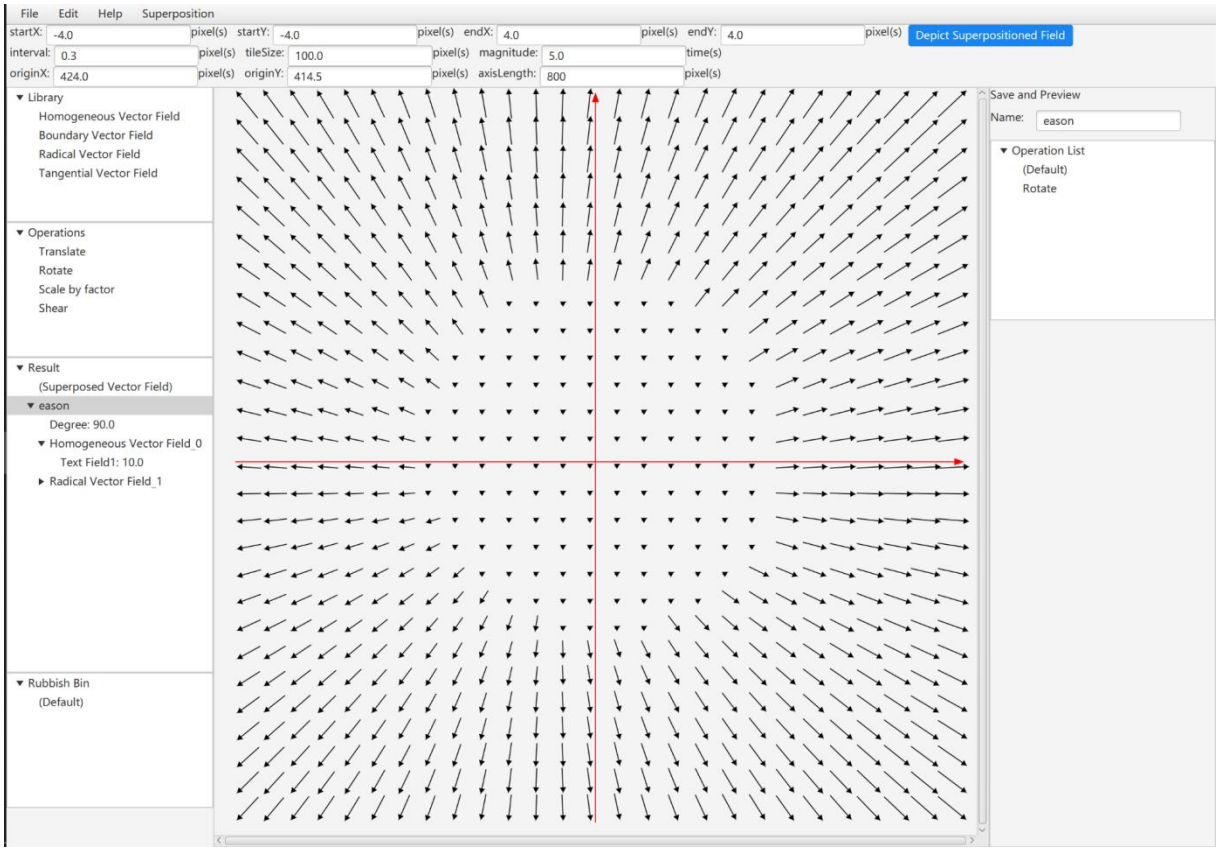


Figure 6.16 A possible outcome of Use case 11 on success end condition (Test case 12: The user changes the name of the customized vector field in test case 8)

6.14 Test case 13：Delete a customized vector field

Use case

Primary Actor: User

Preconditions: A customized vector field exists.

Success End Condition: The vector field is deleted.

Failed End Condition: The vector field stays the same.

Main Success Scenario:

1. Drag the vector field from the “Result” tree view into the “Rubbish Bin” tree view.

Test case

See more details in Appendix A.

General Feedback

As test case 13 in Appendix A shows, this use case functions well.

Screenshots

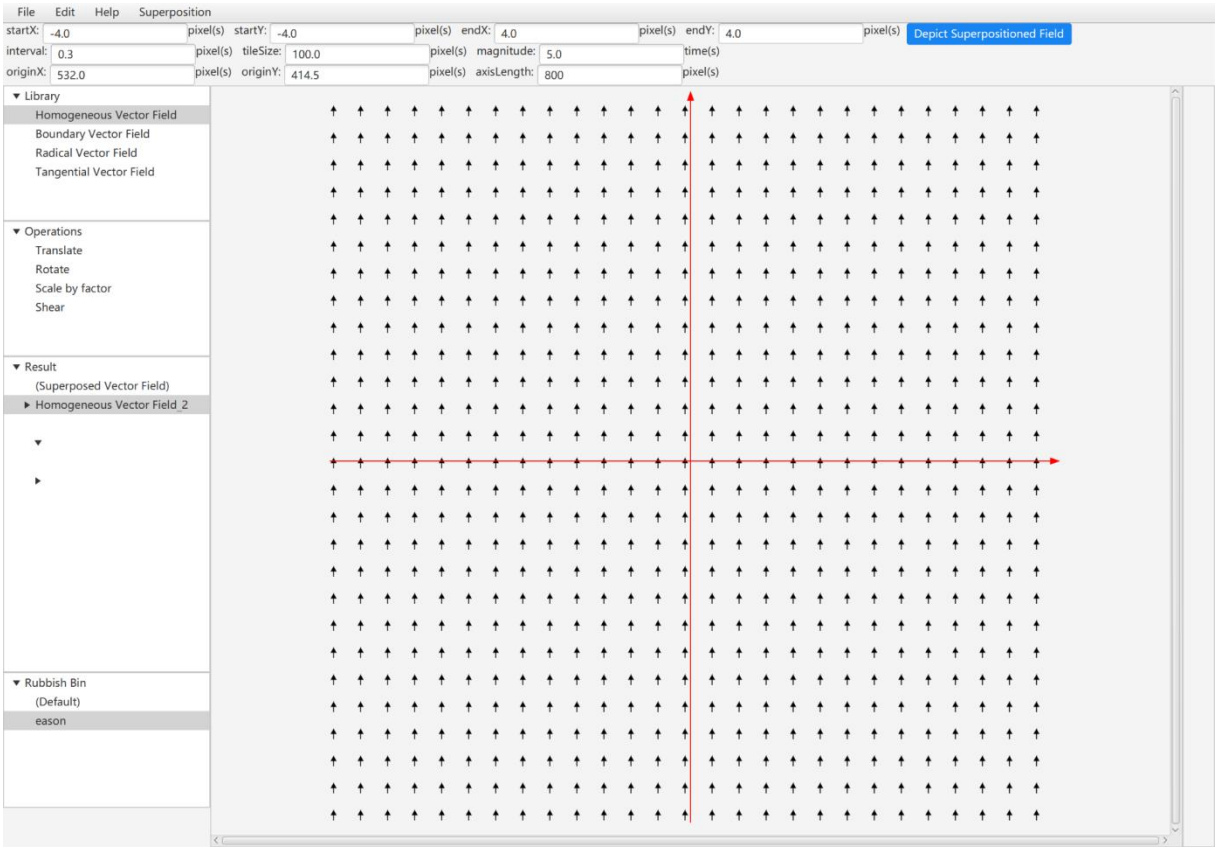


Figure 6.17 A possible outcome of Use case 14 on success end condition (Test case 13: The user deletes the customized vector field in test case 12)

6.15 Test case 14：Recover a customized vector field

Use case

Primary Actor: User

Preconditions: A customized vector field is deleted.

Success End Condition: The vector field is recovered.

Failed End Condition: The vector field stays in the “Rubbish Bin” tree view.

Main Success Scenario:

1. Drag the vector field from the “Rubbish Bin” tree view into the “Result” tree view.
2. Click the vector field, and the vector field successfully displays.

Test case

See more details in Appendix A.

General Feedback

As test case 14 in Appendix A shows, sub-vector fields are missing.

Screenshots

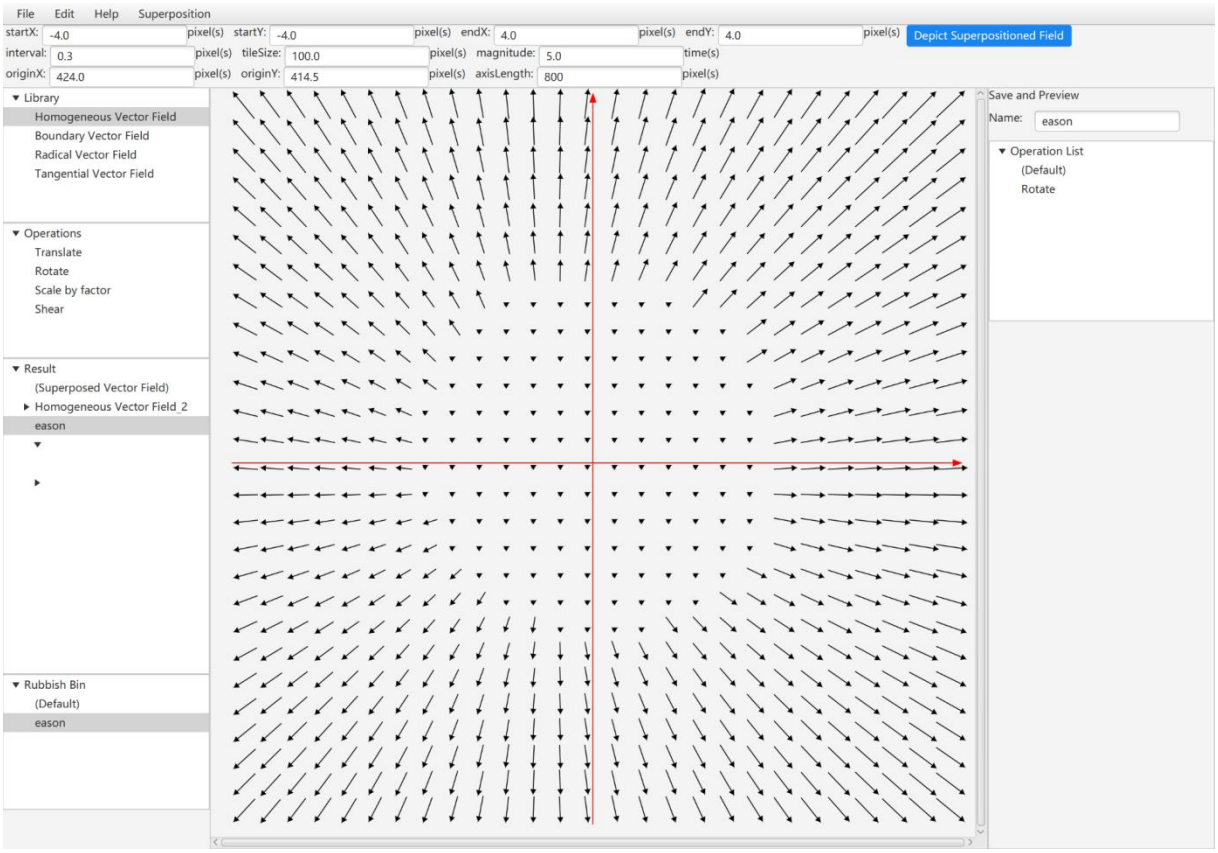


Figure 6.18 A possible outcome of Use case 14 on success end condition (Test case 14: The user recovers the customized vector field in test case 13)

6.16 Test case 15：Zoom in/out the superposed vector fields

Use case

Primary Actor: User

Preconditions: A vector field is superposed.

Success End Condition: The vector field is refreshed.

Failed End Condition: The vector field remains the same.

Main Success Scenario:

1. Click on the vector field
2. Change the parameters in `hbox5`
3. Click the “Depict Superposed Field.”

Test case

See more details in Appendix A.

General Feedback

As test case 15 in Appendix A shows, this use case functions well.

Screenshots

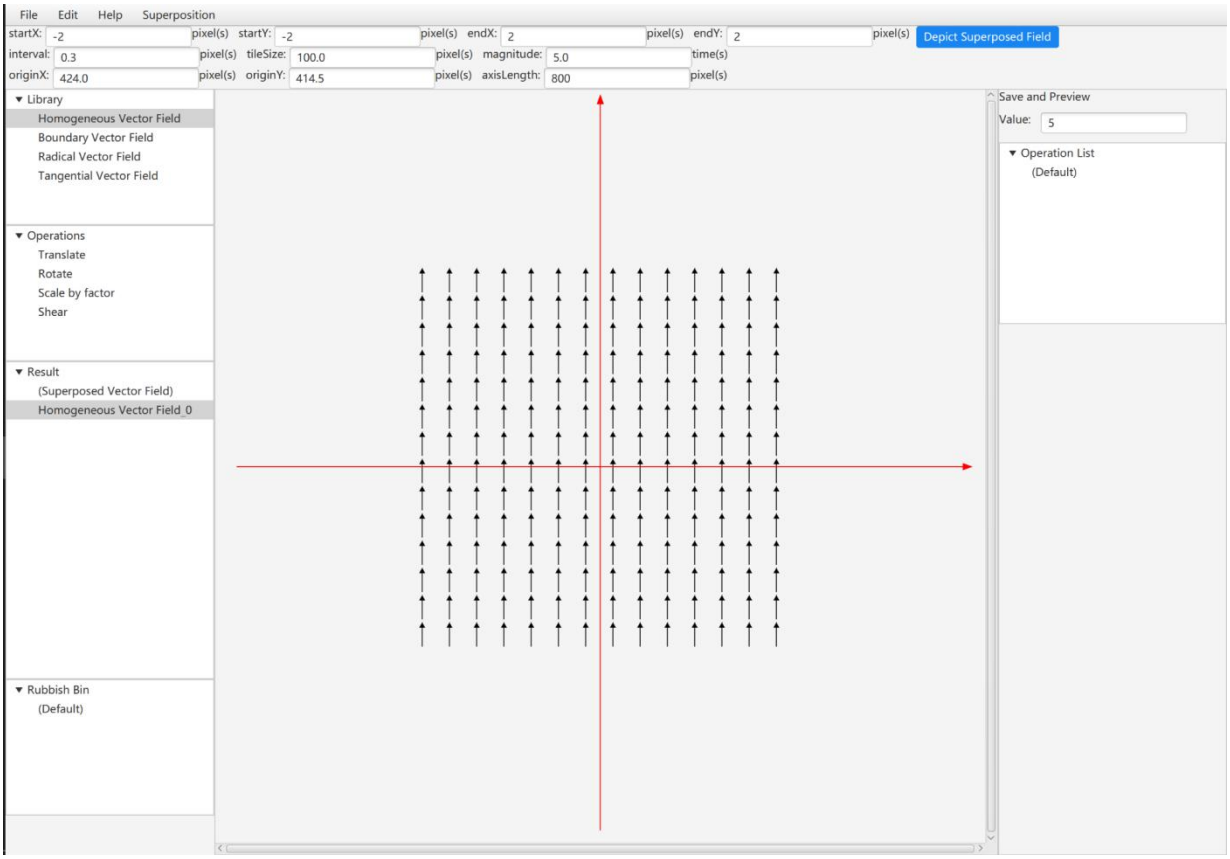


Figure 6.19 A possible outcome of Use case 15 on success end condition (Test case 15: The user changes how a superposed vector field is displayed with a homogenous vector field whose value is 5 as example)

7 Evaluation

After testing with 15 test cases, 13 test cases passed, and 2 test cases failed. Moreover, many usability issues were exposed in the program. As well as some features that were not able to be covered in the design, the authors will explain them in this chapter.

7.1 Usability

An essential component of the GUI design is to ensure high usability. According to Jakob Nielsen [6], usability has five quality components:

Learnability: How easy is it for users to accomplish basic tasks the first time they encounter the design?

Efficiency: Once users have learned the design, how quickly can they perform tasks?

Memorability: When users return to the design after a period of not using it, how easily can they reestablish proficiency?

Errors: How many errors do users make, how severe are these errors, and how easily can they recover from the errors?

Satisfaction: How pleasant is it to use the design?

7.2 Usability issues in the program

This program could be enhanced in the future in terms of usability, and the next generation of researchers could design questionnaires to evaluate the usability of the program based on Nielsen's model. The authors identify three areas of usability that need improvement, which is explained in detail in the following sections.

7.2.1 Double click on result tree view's tree items

Users often need to double-click on the tree items in the "Result" tree view to get the vector field to adjust to the center of the display panel automatically. This is because the vector field is drawn based on the stacking and translation of the tiles. Only when all the tiles to be displayed have been previously drawn and tiled across the board can the program calculate the size of the drawn vector field and then set the origin of the vector field to the center of the board.

Figure 7.1 A mosaic work that used to explain the reasons for this user-unfriendly usability [13]

As shown in Figure 7, the program is like completing the creation of the entire mosaic, calculating the size of the display frame needed for the work, and finally reconstructing them one by one on the appropriate wall.

To avoid this lack of usability, the author presents a hypothesis that a simple solution would be to perform two creations with one keystroke but avoiding the concurrent operations of Java on the program. This operation is to be verified by later researchers.

7.2.2 After the loading, the customized vector field is not calculated immediately

The problem arises because each time a customized vector field is loaded, each elementary vector field in the vector field must be saved in a single pass, and then the customized vector field must be clicked to be calculated.

The author presents two hypotheses. The first solution to this problem is to rewrite the `calculateParentVectorField` method in MyController.java so that each time a customized vector field is clicked, the value of its sub-vector field is first calculated before the custom vector field is calculated. Second, rewrite the `readInJson` method in VectorField.java so that the values of all sub-vector fields are calculated each time before they are loaded. The authors believe that both approaches can solve this problem. The exact implementation is to be implemented by future researchers.

7.2.3 Information of the vector fields in the "Result" tree view

This problem arises because of the confusion in the author's understanding of logic flow when designing the program. Due to the limitation of my ability, I was not able to write and use `giveInfo` functions in MyView.java well. This is to be solved by the next researcher with a refactoring of this function based on a fully functional implementation.

7.2.4 Operations performed on the customized vector fields failed to delete

This problem occurs because the author failed in saving the data, although the user can manually set the operation parameters to zero to achieve the same purpose. The author thinks that a judgment should be added to the drag and drop function to set the metadata corresponding to the operation deleted to be null if the operation is detected to be placed in the "Rubbish Bin" tree view. This would allow the data to be successfully updated and the deletion of operations for the customized vector field to be completed.

7.3 More features

The following is a list of features that the author would like to accomplish but was not able to do so due to the limitations of the current stage of capacity and time. The author hopes they will give clues to future researchers.

7.3.1 Vector field copy by value

The author only implements copying when the elementary vector field is created and cutting when the vector field is edited. This is because, in the toolbox, vector fields cannot be copied by value. Therefore, the author judges that copying a created vector field in the font-end is not possible. The author suggests that future researchers can first build a copy-by-value function in the toolbox to implement this more user-friendly feature.

7.3.2 Operations unimplemented

Constrained by time, the authors have only partially implemented the operation. In the toolbox, there are still operations such as mask, scale, etc., and their required properties such as shape, along which axis, etc., that need to be implemented by future researchers.

7.3.3 Recover a customized vector field from the “Rubbish Bin” tree view

While performing black-box testing of the program, the author finds that when trying to recover deleted customized vector fields from the trash. The sub-vectors of the vector field were lost, and the author believes that the sub-vectors are still stored in the data storage just need future researchers to rewrite the Recover function to recover them. The author hesitates whether this is a usability flaw or a feature that has not been implemented. However, since this issue caused a test case to fail, the author is more inclined to say that this is an unimplemented feature.

7.3.4 The customized vector field’s sub-vector field is a customized vector field

This is the greatest difficulty that the author believes future researchers need to address urgently. Limited by software engineering capabilities, the author is not able to come up with a good solution to this problem.

The problem may not be significant at edit time, but there is a problem of missing operation information when the user tries to save a customized vector field that has operations performed on its customized sub-vector field. The customized vector field after loading is a customized vector field only consisting of elementary vector fields.

8 Summary and Outlook

The author has generally completed the design, implementation and testing of a graphical tool based on the superposition of multiple vector-fields. However, there are still many unrealized features and usability aspects that need to be improved as mentioned in the Evaluation chapter. The author believes that due to the modular programming, later researchers can more easily improve on the authors' program.

The target user of this editor wants to use the program to plan coursing on board. Therefore, the editor is based on the JavaFX framework, using the MVC pattern and with well-documented comments, to make the project suitable for long-term use in the Autosail project. The author believes that through the efforts of generations of programmers this project will become more and more mature and stable.

Acknowledgments

I cannot express enough thanks to my supervisor - Prof. Dr. Ole Blaurock, for his continuous support and instructions. I offer my sincere appreciation for the learning and practicing opportunities provided by him. Prof. Blaurock not only helped me to revise my thesis but also guided me to build the structure, and selflessly shared his insights, gave me timely feedback, and assisted me in the design, implementation, and testing of the program. Prof. Blaurock also took care of me in my life. He helped me transition from being a student to a researcher and programmer more quickly. His rigorous approach to academics and life is something I can continue to learn from.

I also want to thank my parents, Mr. Chen, and Mrs. Yan. They support my studying in Germany. From Lübeck to Hamburg, they support my beautiful exotic experience even during the global pandemic period.

Finally, to my caring, loving, and supportive girlfriend, Yishun. My completion of this project could not have been accomplished without encouragement from you. I hope to see you this summer in Chicago.

9 Appendix A

9.1 Test Case 1: Create an elementary vector field

Description

The user can create an elementary vector field.

The user creates a homogeneous vector field

Test Case					
Steps	**Step Description**	**Data /Value**	**Expected Result**	**Actual Result** (if different from expected)	**Successful/ Failed**
1	The user drags a "Homogeneous Vector Field" into the "Result" tree view		New tree item "Homogeneous Vector Field_0" appears in the Result tree view		Successful
2	The user clicks on the newly created tree item.		The side panel appears.		Successful
3	The user changes value and clicks "Save and Preview."	2	The homogeneous vector field is drawn.		Successful
4	The user double clicks the created tree item to the auto-adjust scope and makes it display in the		The homogeneous vector field displays in the middle.		Successful

	middle.				
			Test Case Status	*Successful*	

The user creates a boundary vector field

Test Case					
Steps	**Step Description**	**Data /Value**	**Expected Result**	**Actual Result** (if different from expected)	**Successful/ Failed**
1	The user drags a "Boundary Vector Field" into the "Result" tree view		New tree item "Boundary Vector Field_0" appears in the Result tree view		Successful
2	The user clicks on the newly created tree item.		The side panel appears.		Successful
3	The user changes value and clicks "Save and Preview."	2	The boundary vector field is drawn.		Successful
4	The user double clicks the created tree item to the auto-adjust scope and makes it display in the middle.		The boundary vector field displays in the middle.		Successful
			Test Case Status	*Successful*	

The user creates a radical vector field

Test Case					
Steps	**Step Description**	**Data /Value**	**Expected Result**	**Actual Result** (if different from expected)	**Successful/ Failed**
1	The user drags a "Radical Vector Field" into the "Result" tree view		New tree item "Radical Vector Field_0" appears in the Result tree view		Successful
2	The user clicks on the newly created tree item.		The side panel appears.		Successful
3	The user changes X, Y and clicks "Save and Preview."	X->1 Y->2	The Radical vector field is drawn.		Successful
4	The user double clicks the created tree item to the auto-adjust scope and makes it display in the middle.		The radical vector field displays in the middle.		Successful
			Test Case Status	*Successful*	

The user creates a tangential vector field

Test Case					
Steps	**Step Description**	**Data /Value**	**Expected Result**	**Actual Result** (if different	**Successful/ Failed**

				from expected)	
1	The user drags a “Tangential Vector Field” into the “Result” tree view		New tree item “Tangential Vector Field_0” appears in the Result tree view		Successful
2	The user clicks on the newly created tree item.		The side panel appears.		Successful
3	The user changes X, Y and clicks “Save and Preview.”	X->1 Y->2	The tangential vector field is drawn.		Successful
4	The user double clicks the created tree item to the auto-adjust scope and makes it display in the middle.		The tangential vector field displays in the middle.		Successful
			Test Case Status	*Successful*	

9.2 Test case 2: Perform operations on an elementary vector field

Description

The user can perform operations on an elementary vector field.

The user performs Rotate on a homogeneous vector field with value of 2

Test Case					
Steps	**Step Description**	**Data**	**Expected Result**	**Actual Result** (if different	**Successful/**

		/Value		from expected)	Failed
1	The user drags a “Rotate” into the “OptIn” tree view		New tree item “Rotate” appears in the “OptIn” tree view		Successful
2	The user clicks on the newly created tree item.		Textfield for Degree appears.		Successful
3	The user changes value and clicks “Save and Preview.”	30	The homogeneous vector field is rotated 30°.		Successful
4	The user clicks the “Radical Vector Field_0” again, and clicks the “Save and Preview”		The homogeneous vector field stays the same.		Successful
			Test Case Status	*Successful*	

The user performs Translate on a radical vector field with x equals 1 and y equals 2

Test Case					
Steps	**Step Description**	**Data /Value**	**Expected Result**	**Actual Result** (if different from expected)	**Successful/ Failed**
1	The user drags a “Translate” into the “OptIn” tree view		New tree item “Translate” appears in the “OptIn” tree view		Successful

2	The user clicks on the newly created tree item.		Textfield for Along X Axis, Textfield for Along Y Axis appears.		Successful
3	The user changes value and clicks "Save and Preview."	Along X Axis-> 1 Along Y Axis->2	The radical vector field is translated.		Successful
			Test Case Status	*Successful*	

The user performs scale by factor on a homogeneous vector field with value of 2

Test Case					
Steps	**Step Description**	**Data /Value**	**Expected Result**	**Actual Result** (if different from expected)	**Successful/ Failed**
1	The user drags a "Scale by factor" into the "OptIn" tree view		New tree item "Scale by factor" appears in the "OptIn" tree view		Successful
2	The user clicks on the newly created tree item.		Textfield for Scaling factor appears.		Successful
3	The user changes value and clicks "Save and Preview."	Scaling factor ->2	The homogeneous vector field is scaled by 2.		Successful

	Test Case Status	*Successful*	

The user performs shear on a homogeneous vector field with value of 2

Test Case					
Steps	**Step Description**	**Data /Value**	**Expected Result**	**Actual Result** (if different from expected)	**Successful/ Failed**
1	The user drags a "Shear" into the "OptIn" tree view		New tree item "Shear" appears in the "OptIn" tree view		Successful
2	The user clicks on the newly created tree item.		Textfield for Shearing factor appears.		Successful
3	The user changes value and clicks "Save and Preview."	Shearing factor ->2	The homogeneous vector field is sheared.		Successful
			Test Case Status	*Successful*	

The user performs multiple opeartions on a radical vector field with x equals 1 and y equals 2

Test Case					
Steps	**Step Description**	**Data /Value**	**Expected Result**	**Actual Result** (if different from expected)	**Successful/ Failed**

1	The user drags a “Shear” into the “OptIn” tree view		New tree item “Shear” appears in the “OptIn” tree view		Successful
2	The user clicks on the newly created tree item.		Textfield for Shearing factor appears.		Successful
3	The user changes value and clicks “Save and Preview”	Shearing factor ->2	The radical vector field is sheared.		Successful
4	The user drags a “Translate” into the “OptIn” tree view		New tree item “Translate” appears in the “OptIn” tree view		Successful
5	The user changes value and clicks “Save and Preview.”	Along X Axis-> 1 Along Y Axis->2	The radical vector field is translated.		Successful
6	The user clicks on the newly created tree item.		Textfield for Along X Axis, Textfield for Along Y Axis appears.		Successful
7	The user drags a “Rotate” into the “OptIn” tree view		New tree item “Rotate” appears in the “OptIn” tree view		Successful
8	The user clicks on the newly created tree item.		Textfield for Degree appears.		Successful
9	The user changes value and clicks “Save and	30	The radical vector field is rotated 30°.		Successful

	Preview.”				
10	The user drags a “Scale by factor” into the “OptIn” tree view		New tree item “Scale by factor” appears in the “OptIn” tree view		Successful
11	The user clicks on the newly created tree item.		Textfield for Scaling factor appears.		Successful
12	The user changes value and clicks “Save and Preview.”	Scaling factor ->0.5	The radical vector field is scaled by 0.5.		Successful
			Test Case Status	*Successful*	

9.3 Test case 3: Change properties of an elementary vector field

Description

The user can change properties of an elementary vector field.

The user changes properties of an homogeneous vector field with value 2 and rotate 30°

Test Case					
Steps	**Step Description**	**Data /Value**	**Expected Result**	**Actual Result** (if different from expected)	**Successful/ Failed**
1	The user changes the value to 5 and clicks “Save and	5	The direction vectors in homogeneous vector field becomes 5.		Successful

	Preview"				
2	The user clicks on the "Rotate" tree item in the "OptIn" tree view		Textfield for Degree appears.		Successful
3	The user changes degree and clicks "Save and Preview."	60	The homogeneous vector field is rotated 60°.		Successful
			Test Case Status	*Successful*	

The user deletes operations of a homogeneous vector field with value 5 and rotate 30°

Test Case					
Steps	**Step Description**	**Data /Value**	**Expected Result**	**Actual Result** (if different from expected)	**Successful/ Failed**
1	The user drags the "Rotate" tree view in the "OptIn' tree view into the "Rubbish Bin" tree view		No operations in the "Operation List" tree view		Successful
2	The user clicks "Save and Preview."		The homogeneous vector field with value of 5 is displayed.		Successful
			Test Case Status	*Successful*	

9.4 Test case 4: Delete an elementary vector field

Description

The user can delete an elementary vector field.

The user deletes a homogeneous vector field with value 5

Test Case					
Steps	**Step Description**	**Data /Value**	**Expected Result**	**Actual Result** (if different from expected)	**Successful/ Failed**
1	The user drags the "Homogeneous Vector Field_0" tree item into the "Rubbish Bin" tree view		The "Homogeneous Vector Field_0" tree item disappers.		Successful
2	The user clicks on the "Superposition" and clicks on "(Superposed Vector Field)"		The unit vector field appears.		Successful
			Test Case Status	*Successful*	

9.5 Test case 5: Recover an elementary vector field

Description

The user can recover an elementary vector field.

The user recovers the deleted homogeneous vector field in Test case 4

Test Case					
Steps	**Step Description**	**Data**	**Expected Result**	**Actual**	**Successful/**

		/Value		Result (if different from expected)	Failed
1	The user drags the “Homogeneous Vector Field_0” tree item into the “Result” tree view		The “Homogeneous Vector Field_0” tree item appears in the “Result” tree view.		Successful
2	The user clicks on the “Homogeneous Vector Field_0” tree item		The homogeneous vector field appears.		Successful
			Test Case Status	*Successful*	

9.6 Test case 6: Create a customized vector field

Description

The user can create a customized vector field.

The user creates a customized vector field.

Test Case					
Steps	**Step Description**	**Data /Value**	**Expected Result**	**Actual Result** (if different from expected)	**Successful/ Failed**
1	The user drags the “Homogeneous Vector Field” tree item into the “Result” tree view		The “Homogeneous Vector Field_0” tree item appears in the “Result” tree view.		Successful
2	The user changes the value and	5	A homogeneous vector field is drawn.		Successful

	clicks the “Save and Preview”				
3	The user drags the “Homogeneous Vector Field” tree item into the “Result” tree view		The “Homogeneous Vector Field_1” tree item appears in the “Result” tree view.		Successful
4	The user changes the value and clicks the “Save and Preview”	2	A homogeneous vector field is drawn.		Successful
5	The user drags the “Radical Vector Field” tree item into the “Result” tree view		The “Radical Vector Field_2” tree item appears in the “Result” tree view.		Successful
6	The user changes the value and clicks the “Save and Preview”	X->1 Y->2	A radical vector field is drawn.		Successful
7	The user clicks the “Superposition” and clicks the “(Superposed Vector Field)”		The customized vector field appears		Successful
			Test Case Status	*Successful*	

The user loads a customized vector field which consists of the first two elementary vector fields of the customized vector field before and adds a radical vector field with x equals 1 and y equals 2.

Test Case

Steps	Step Description	Data /Value	Expected Result	Actual Result (if different from expected)	Successful/ Failed
1	The user loads a customized vector field.		A customized vector field appears in the "Result" tree view		Successful
2	The user drags the "Radical Vector Field" tree item into the "Result" tree view		The "Radical Vector Field_2" tree item appears in the "Result" tree view.		Successful
3	The user changes the value and clicks the "Save and Preview"	X->1 Y->2	A radical vector field is drawn.		Successful
4	The user clicks the "Superposition" and clicks the "(Superposed Vector Field)"		The customized vector field appears		Successful
			Test Case Status	*Successful*	

9.7 Test Case 7: Perform operations on the superposed vector field

Description

The user can perform operations on the superposed vector field.

The user performs operations on the superposed vector field with test case 6 outcome as precondition

Test Case

Steps	Step Description	Data /Value	Expected Result	Actual Result (if different from expected)	Successful/ Failed
1	The user clicks on the "(Superposed Vector Field)" tree item		The superposed vector field appears.		Successful
2	The user drags "Rotate" in the "Operation List" tree view and clicks the "Rotate", changes the degree, clicks "Save and Preview" , and then clicks on the "(Superposed Vector Field)" tree item again	60	The superposed vector field rotates 60°		Successful
			Test Case Status	*Successful*	

The user deletes operations on the superposed vector field before.

Test Case					
Steps	**Step Description**	**Data /Value**	**Expected Result**	**Actual Result** (if different from expected)	**Successful/ Failed**
1	The user clicks on the "(Superposed Vector Field)" tree item		The superposed vector field appears.		Successful

2	The user drags “Rotate” from the “Operation List” to “Rubbish Bin” tree view and clicks “Save and Preview” , and then clicks on the “(Superposed Vector Field)” tree item again		The superposed vector field doesn’t rotate 60° any more.		Successful
			Test Case Status	*Successful*	

9.8 Test Case 8: Save customized vector fields

Description

The user can save customized vector field.

The user saves a customized vector field rotated 90 ° which is superposition of a homogeneous vector field with value of 10 and radical vector field with x equals 1 and y equals 2.

Test Case					
Steps	**Step Description**	**Data /Value**	**Expected Result**	**Actual Result** (if different from expected)	**Successful/ Failed**
1	The user clicks on the “File” and then clicks on the “Save”		A File chooser appears.		Successful
2	The user chooses where to store the file and types in	hello	A hello.json file is generated.		Successful

	the file name then clicks “Save”				
			Test Case Status	*Successful*	

9.9 Test Case 9: Load customized vector fields

Description

The user can load customized vector field.

The user loads the customized vector field in test case 8

Test Case					
Steps	**Step Description**	**Data /Value**	**Expected Result**	**Actual Result** (if different from expected)	**Successful/ Failed**
1	The user clicks on the “File” and then clicks on the “Load”		A File chooser appears.		Successful
2	The user chooses the file and loads it.	hello.json	A “hello.json” tree item is generated in the “Result” tree view.		Successful
3	The user clicks on all loaded vector field’s sub-vector fields and clicks “Save and Preview”		All sub-vector fields are generated.		Successful
4	The user clicks on the loaded vector field		The loaded customized vector field displays.		Successful
			Test Case Status	*Successful*	

9.10 Test Case 10: Perform operations on a customized vector field

Description

The user can perform operations on a customized vector field.

The user performs translate on the customized vector field in test case 8

Test Case					
Steps	**Step Description**	**Data /Value**	**Expected Result**	**Actual Result** (if different from expected)	**Successful/ Failed**
1	The user clicks on the customized vector field in the "Result" tree view		The customized vector field appears.		Successful
2	The user drags "Translate" in the "Operation List" tree view, clicks on the "Translate", changes the parameters, and clicks "Save and Preview".	X->1 Y->2	The customized vector field updates.		Successful
			Test Case Status	*Successful*	

The user deletes "translate" from the customized vector field as Figure 6.13 as precondition

Test Case					
Steps	**Step Description**	**Data /Value**	**Expected Result**	**Actual Result** (if different	**Successful/ Failed**

				from expected)	
1	The user clicks on the customized vector field in the "Result" tree view		The sub-vector field appears.		Successful
2	The user drags the "Translate" from the "Operation List" into "Rubbish Bin" tree view and clicks "Save and Preview"		The customized vector field is not translated any more.		Successful
3	The user clicks on the customized vector field.		The customized vector field doesn't update.		Fail
			Test Case Status	*Fail*	

9.11 Test Case 11: Change sub-vector fields' properties of a customized vector field

Description

The user can change sub-vector fields' properties of a customized vector field.

The user changes the customized vector field in test case 8's sub-vector fields' properties

Test Case					
Steps	**Step Description**	**Data /Value**	**Expected Result**	**Actual Result** (if different from expected)	**Successful/ Failed**

1	The user clicks on the homogeneous field of the customized vector field in the "Result" tree view		The sub-vector field appears.		Successful
2	The user changes the value and clicks "Save and Preview".	2	The homogeneous vector field updates.		Successful
3	The user clicks on the customized vector field.		The customized vector field updates.		Successful
			Test Case Status	*Successful*	

9.12 Test Case 12: Change the name of a customized vector field

Description

The user can change the name of a customized vector field.

The user changes the name of the customized vector field in test case 8.

Test Case					
Steps	**Step Description**	**Data /Value**	**Expected Result**	**Actual Result** (if different from expected)	**Successful/ Failed**
1	The user clicks on the customized vector field in the "Result" tree view		The sub-vector field appears.		Successful
2	The user changes the name and	eason	The customized vector field's name updates.		Successful

	clicks “Save and Preview”.				
			Test Case Status	*Successful*	

9.13 Test Case 13: Delete a customized vector field

Description

The user deletes a customized vector field.

The user deletes the customized vector field in test case 12.

Test Case					
Steps	**Step Description**	**Data /Value**	**Expected Result**	**Actual Result** (if different from expected)	**Successful/ Failed**
1	The user drags the customized vector field from the “Result” tree view into the “Rubbish Bin” tree view		The customized vector field disappears in the “Result” tree view.		Successful
			Test Case Status	*Successful*	

9.14 Test Case 14: Recover a customized vector field

Description

The user recovers a deleted customized vector field.

The user recovers the customized vector field in test case 13.

Test Case					
Steps	**Step Description**	**Data /Value**	**Expected Result**	**Actual Result** (if different	**Successful/ Failed**

				from expected)	
1	The user drags the customized vector field from the "(Superposed Vector Field)" in the "Result" tree view		The customized vector field disappears in the "Result" tree view.	The sub-vector field of the customized vector field misses.	Fail
			Test Case Status	*Fail*	

9.15 Test Case 15: Zoom in/out the superposed vector fields

Description

The user can change how the superposed vector field is displayed.

The user changes how a superposed vector field is displayed with a homogenous vector field whose value is 5 as example.

Test Case					
Steps	**Step Description**	**Data /Value**	**Expected Result**	**Actual Result** (if different from expected)	**Successful/ Failed**
1	The user changes the parameters in hbox5. And clicks "Depict Superposed Field"		The superposed vector field is drawn again.		Successful
			Test Case Status	*Successful*	

10 Appendix B

Appendix B states the application programming interfaces (API) from the toolbox. Many of them are referenced in the thesis with the font format `Consolas`.

10.1 Elementary vector fields

Functions	Methods	Parameters
Construct a border vector field	`public BorderField(double value, Proportionality proportionality)`	`@param value` value of the field `@param proportionality` proportionality of the value to the distance from the border
	`public BorderField()`	
Construct a homogenous vector field	`public HomogenousField(double value)`	`@param value` value of the field
	`public HomogenousField()`	
Construct a superposed vector field with all values being zero	`public SuperposedField()`	
Construct a radial vector field	`public RadialField(double value, double radius, Proportionality proportionality)`	`@param value` value of the field `@param radius` radius of the obstacle `@param proportionality` proportionality of the value to the distance from the obstacle
	`public RadialField()`	

Construct a tangential vector field	`public TangentialField(double value, double radius, Proportionality proportionality)`	`@param value value of the field`
		`@param radius radius of the obstacle`
		`@param proportionality proportionality of the value to the distance from the obstacle`

	`public TangentialField()`	

10.2 Operations

Get the value of direction vector given the location vector	`public Vector2D getFunctionValueOptimized(double x, double y)`	`@param x x coordinate of location vector`
		`@param y y coordinate of location vector`
		`@return direction vector value`

	`public Vector2D getFunctionValue(double x, double y)`	`@param x x coordinate of location vector`
		`@param y y coordinate of location vector`
		`@return direction vector value`

Rotate vector field around origin point	`public void rotate(double degree)`	`@param degree rotate to this angle`
Rotate vector field around a specific	`public void rotate(double degree, Vector2D rotationPoint)`	`@param degree rotate to this angle` `@param rotationPoint point to rotate around`

point		
Scale a specified area (shape)	`public void scaleArea(Shape shape, double factor)`	`@param shape shape to scale` `@param factor scaling factor`
Shear by specified axis	`public void shear(double factor, Axis axis)`	`@param factor shearing factor` `@param axis axis to shear by`
Mask a specified shape	`public void mask(Shape shape)`	`@param shape shape to mask`
Translate by specified values	`public void translate(double alongXaxis, double alongYaxis)`	`@param alongXaxis translation value along x axis` `@param alongYaxis translation value along y axis`
Scale by factor	`public void scale(double factor)`	`@param factor scaling factor`

10.3 File Operations

Save vector fields as Json file	`public void save()`	
	`public void save(String fileName)`	`@param fileName name of json file`

Export vector field as csv file	`public void export(Vector2D intervalXaxis, double samplingDistanceX, Vector2D intervalYaxis,double samplingDistanceY)`	`@param intervalXaxis evaluation interval on x axis`
		`@param samplingDistanceX mesh size of the grid on x axis`
		`@param intervalYaxis evaluation interval on y axis`
		`@param samplingDistanceY mesh size of the grid on`

		y axis
	`public void export(Vector2D intervalXaxis, double samplingDistanceX, Vector2D intervalYaxis, double samplingDistanceY, String fileName)`	@param intervalXaxis evaluation interval on x axis @param samplingDistanceX mesh size of the grid on x axis @param intervalYaxis evaluation interval on y axis @param samplingDistanceY mesh size of the grid on y axis @param filename Name of csv file
	`public void exportUsingOpt(Vector2D intervalXaxis, double samplingDistanceX, Vector2D intervalYaxis, double samplingDistanceY, String fileName)`	@param intervalXaxis evaluation interval on x axis @param samplingDistanceX mesh size of the grid on x axis @param intervalYaxis evaluation interval on y axis @param samplingDistanceY mesh size of the grid on y axis @param filename Name of csv file
Reading in a json file that represents a field.	`public static VectorField readInJSON(String file)`	@param file file path and field @return a vectorfield specified by the json field

Bibliography

[1] Manuela Kastner. “Entwurf, Implementierung und Test einer anwendungsspezi-fischen Toolbox fur die Modellierung und Superposition von zweidimensionalen Vektorfeldern in Java”. Informatik/ Softwaretechnik, B.Sc. TH Lübeck. 2018.

[2] Christian Weber.“Analyse und Verbesserung der Funktionalität und des Laufzeitverhaltens einer anwendungsspezifischen Toolbox für die Modellierung und Superposition von zweidimensionalen Vektorfeldern in Java”. TH Lübeck. 2020.

[3] Alexander Ostermann Michael Obergouggenberger. “Vektorwertige Funktionen in zwei Veraenderlichen”. In: Analysis fuer Informatiker: Grundlagen, Methoden, Algorithmen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 201–207. isbn: 978-3-540-89823-8. doi: 10.1007/978-3-540-89823-8_16. url: https://doi.org/10.1007/978-3-540-89823-8_16.

[4] Oliver Vornberger. 2010. url: http://www-lehre.inf.uos.de/~cg/2010/PDF/ kap-06.pdf.

[5] Andreas Filler. “Matrizen”. In: Elementare Lineare Algebra: Linearisieren und Koordinatisieren. Heidelberg: Spektrum Akademischer Verlag, 2011, pp. 227–256. isbn: 978-3-8274-2413-6. doi: 10.1007/978- 3- 8274- 2413- 6_6. url: https://doi.org/10.1007/978-3-8274-2413-6_6.

[6] Jakob Nielsen’s Alertbox, August 25, 2003: Usability 101: Introduction to Usability https://www.nngroup.com/articles/usability-101-introduction-to-usability/

[7] Jakob Nielsen. P72. Usability Engineering. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993. isbn: 0125184050.

[8] Alan Cooper. “The perils of prototyping”. In: Visual Basic Programmers Journal (1994).

[9] Raffael Pancheri, 2015. " Design and Implementation of a Graphical The user Interface for Elektra". Elektra Iniitiative. https://www.libelektra.org/ftp/pancheri2015gui.

[10] Alistair Cockburn. Writing Effective Use Cases. 1st. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000. isbn: 0201702258.]

[11] Kn0412. /Arrow.java. https://gist.github.com/kn0412/2086581e98a32c8dfa1f69772f14bca4

[12] Reenskaug, Trygve; Coplien, James O. (20 March 2009). “The DCI Architecture: A New Vision of Object-Oriented Programming”. Artima Developer. Archived from the

original (html) on 23 March 2009. Retrieved 3 August 2019. More deeply, the framework exists to separate the representation of information from The user interaction

[13] Asram, Photographie personnelle prise au musée archéologique de Sousse; la mosaïque représente le triomphe de Neptune, https://fr.wikipedia.org/wiki/Fichier:Sousse_neptune.jpg