Web-based Graph Animation with Galant

Final Project Report

Preliminary Requirements, Design, Implementation & Testing

Dr. Matthias Stallmann

CSC 492 Team 43

Christina Albores

Vitesh Kambara

Julian Madrigal

Neha Ramesh

Minghong Zou

North Carolina State University

Department of Computer Science

04/23/2024

# Executive Summary

*Author(s): Neha Ramesh*

*Reviewer(s)/Editor(s): Christina Albores*

The project we are working on currently, is centered around a web application called Galant-js. Essentially, Galant-js facilitates algorithm visualization and step-by-step traversal via node graphs. Our sponsor, Dr. Matt Stallmann, a professor at NCSU, approached us with a pre-existing implementation of Galant-js, which had been developed by previous teams. While the basic functionalities were in place, Dr. Stallmann sought enhancements and additional features. Currently, there exists a fully functional Java-based version of Galant.

Our objective is to replicate its functionality in JavaScript, as the Java version relies on several dependencies. This transition is crucial for our sponsor, as he, along with other educators and students, aims to utilize Galant as a tool for visualizing and explaining various algorithms on different graph structures, enhancing teaching and learning experiences.

Given the multiple different functionalities requested by our sponsor, our solution consists of multiple components. We started by breaking down the requirements into nine distinct use cases, further elaborating on each to gain a better understanding of the tasks at hand. These use cases encompass automatically loading algorithm and graph examples into the application, enabling offline functionality, facilitating node and edge position and attribute editing by the user and algorithm, synchronizing graph and graph text, keyboard shortcuts, ensuring file persistence upon refresh, scaling .sgf files properly in the window, and implementing a preference panel.

These use cases are further divided into three iterations, each addressing a subset of the requirements. Currently, we have finished all our iterations successfully. Our efforts throughout the past semester were concentrated on implementing all the use cases which ranged from automatic loading of graphs and algorithms, to synchronization of graph and graph text and parsing sgf files.We dropped one use case that had to do with the offline functionality as it is too convoluted and unnecessary for us to address for this project. Attempting to implement that would have taken away from other functionalities that were more of a priority, which our sponsor agrees with.

Aside from that, all the use cases we took on at the beginning of the semester have been completed. Furthermore, we have finished testing verifying functionality with 76 unit tests that are all passing and 25 system tests that are all passing as well. As we have completed everything we wanted to complete at the beginning of the semester there is nothing left to do aside from presenting at Posters & Pies which we will be practicing.

# Project Description

## **Sponsor Background**

*Author(s): Neha Ramesh*

*Reviewer(s)/Editor(s): Christina Albores*

Our sponsor, Dr Matt Stallmann, a professor at North Carolina State University, is actively engaged in teaching and research spanning various domains, including graph algorithms and the theory of computation. For our project specifically, our focus is directed towards graph algorithms, human-computer interaction, and user experience. Graph algorithms are a main component of Galant and its use. Additionally, human-computer interaction is a rapidly growing multidisciplinary field aimed at enhancing the usability of technology and computer systems for individuals. This seamlessly intertwines with user experience, where our considerations are centered on improving computer systems for users. Dr. Stallmann uses this application in his classes to demonstrate algorithms and how they behave through the graphs, as well creating assignments that can use this application. He also creates new algorithms and this helps him implement them and see if they work how he expects them to.

*Dr Matthias Stallmann*

Graduate Professor

Location: Raleigh, NC

Email Address: mfms@ncsu.edu

Phone Number: 919-515-7978

## **Problem Description**

*Author(s): Christina Albores*

*Reviewer(s)/Editor(s): Neha Ramesh, Vitesh Kambara*

The problem presented to our team is the continuation of the development of Galant-JS, aiming to integrate all functionality from the original Java version into the current JavaScript version. Galant is a versatile tool for creating graph algorithm animations, used for both educational and research purposes. However, the current implementation requires users to have several dependencies—including Git, Apache ant, and access to a Java compiler—leading to complexity and platform-dependent behavior.

The primary need addressed by the software our team will be creating is to enhance the usability of Galant-JS, bringing it on par with the original Java version. This involves enabling users to edit and save graphs and algorithms online and offline, expanding animation features such as node movement and attribute changes, implementing infrastructure for sorting algorithms and layered graph heuristics, and providing comprehensive documentation and a flexible user interface with keyboard shortcuts.

Existing systems like the Java-based Galant offer these features but lack the convenience and accessibility of a web-based platform. Galant-JS already provides certain advantages, but it needs improvements to reach the same level of functionality as the Java version. These enhancements will not only make Galant more user-friendly but also ensure platform independence and ease of access.

**Proposed Solution & Project Goals/Benefits**

*Author(s): Minghong Zou*

*Reviewer(s)/Editor(s): Neha Ramesh*

Our solution is to update the core feature of Glant-JS implementing Javascript. This web application will be usable on Chrome, Firefox, Edge, and Safari without requiring the user to download anything onto their device. By adding the new Javascript frameworks Headless and tailwind, we have the opportunity to change the user interface UI and greatly increase the flexibility of the UI. These improvements will allow Dr. Stallmann to make better use of the algorithm visualizations in his research and his classes.Better flexibility means that we can add more visual effects, such as directly marking nodes or changing node colors. Also, We have added new shortcuts and File type support (sgf files) to allow Dr Stallman to process algorithms faster and allow him to extend the use of layered images in the future.

# Resources Needed

*Author(s):Minghong Zou*

*Reviewer(s)/Editor(s): Team*

**Table 1: Resources Table for the Galant-Js Senior Design Team Project**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Resource | Purpose | Status | Version | License |
| Previous version Code | Rebase on this version code | Obtained, found on Github | Most recent release version is by Dr. Stallmann on Jan 10 | N/A |
| NCSU Virtual Machine | Used to deploy our project, serve the web and host domains. | Obtained, Dr.Stallmann already has one | N/A | N/A |
| Permanent Domain | Use to allow public access | Obtained, Dr.Stallmann already has one | N/A | N/A |
| Github | Use to version control and team work | Obtained | N/A | N/A |
| Github Action | Use to deploy project and manage issue workflow | Obtained | N/A | N/A |
| React | Framework to create user interface | Obtained | 11.11.1 | MIT |
| Cytoscape | Display the graph and its animation | Obtained | 2.0.0 | MIT |
| Jest | Unit test | Obtained | 5.16.5 | MIT |
| Headless | Templating for a new and more flexible UI | Obtained | 1.7.18 | MIT |
| Tailwind | Styling to UI | Obtained | 3.4.1 | MIT |
| Cypress | Auto test frontend | Obtained | 13.6.6 | MIT |

# Risks & Risk Mitigation

*Author(s): Minghong Zou*

*Reviewer(s)/Editor(s): Christina Albores*

There are a few potential risks we may encounter in the course of this project. First is the remaining bugs. We inherited code from other teams, and there were some bugs in this code. This is a huge project, and it has now been determined that there are some features in this project that do not work as expected, which meant that we also needed to fix the original bugs when implementing new features, which might cause some unexpected changes. Although we have patched a large number of bugs by the end of spring 2024, there may still be some bugs that only occur in the limit case that we have not yet found and patched.

Another risk that we’re facing is the dependencies. Existing dependencies and versions of libraries/frameworks have the potential to be outdated or incompatible with some of the changes we make. We will have to either abide by these dependencies in our implementation or update them entirely to accommodate new changes if a majority of them are incompatible/outdated. Also, we cannot guarantee that these dependencies will not be subject to bugs in extreme cases or incompatibilities on untested platforms

# Development Methodology

*Author(s): Vitesh Kambara*

*Reviewer(s)/Editor(s): Christina Albores*

Our team adheres to a monthly iteration development cycle, employing the Agile Scrum methodology to organize work into fixed iterations. Within each iteration, team members rotate roles, fulfilling project requirements. Objectives, such as research, implementation, testing, and documentation, are tailored to the overarching goal. Task management utilizes a tabular format in Google Docs, outlining iteration descriptions, tasks, assigned members, and status. Tasks are assigned individually or to a group, considering scalability and difficulty, and the document is collaboratively worked on during designated class collaboration time.

Iterative focus centers on addressing tasks identified during sponsor meetings held weekly. Agendas, including project status updates and the proposed plan for the upcoming week, are shared with Dr. Stallmann. This allows sufficient time for implementation, code testing, and seeking technical assistance. Consistent messaging in Slack ensures ongoing communication of project progress. Additionally, weekly team meetings outside of class time foster collaboration and project advancement, serving as forums for discussing questions, concerns, and setbacks.

GitHub serves as our version control and project management tool. Branches are created based on task scope and acknowledging potential merge conflicts when working on the same files. If merge conflicts occur, the team collectively resolves these conflicts, valuing each member's contributions and ensuring that no valuable code is removed from the codebase. Issues are created based on the requirements and assigned based on the rotating roles of team members and their interest and comfort level in pursuing the issue. Each iteration follows a unique plan aligned with specific completion requirements. Code is merged into the main branch upon completion of key functionalities within an iteration. Integration testing, through GitHub Actions, continuously updates sponsors and teammates on the functionality of the project.

# System Requirements

## **Overall View**

*Author(s): Christina Albores*

*Reviewer(s)/Editor(s): Team*

The project aims to enhance the usability and accessibility of Galant, a graph algorithm animation tool, by developing a web-based version called Galant-js. This transition from the existing Java-based Galant to a web-based platform is driven by the need for easier deployment and platform independence.

The users of this system include researchers, educators, and students who utilize graph algorithms for classroom teaching and research purposes. Existing systems like the Java-based Galant offer these features but lack the convenience and accessibility of a web-based platform. Galant-JS already provides certain advantages, but it needs improvements to reach the same level of functionality as the Java version. These enhancements will not only make Galant more user-friendly but also ensure platform independence and ease of access.

**Key features and enhancements targeted in Galant-js include:**

1. **Automatic Loading of Algorithm and Graph Examples:** Users can select algorithm and graph examples from dropdown menus, eliminating the need to download separate files for each example.
2. **Synchronization of Graph and Graph Text:** Changes made to either the graph or text are automatically reflected in the other, ensuring consistency between the visual representation and textual description of the graph.
3. **Animation Features:** Users can animate graph algorithms, including the ability to move nodes on the graph according to the algorithm steps.
4. **Node and Edge Attributes:** Users can customize node and edge attributes such as radius, shading, and thickness to enhance visual representation.
5. **Keyboard Shortcuts:** Implementation of keyboard shortcuts allows users to perform common operations quickly, enhancing user experience and productivity.
6. **Scale .sgf Files:** Support for .sgf files and proportional scaling of coordinates based on window dimensions enables the animation of layered graph algorithms.
7. **Persistence of Algorithm and Graph Files:** Uploaded algorithm and graph files persist even after webpage refresh, ensuring users don't lose their work unintentionally.
8. **Skip to End Functionality:** Users can skip to the end of an algorithm to view the final graph result, enhancing efficiency in algorithm visualization.
9. **Preference Panel:** A preferences panel allows users to configure their preferences, such as font, text, and color, or upload a .txt file with preset preferences.

These features aim to make galant-js a user-friendly, platform-independent, and feature-rich alternative to the existing Java-based Galant tool, meeting the needs of both educators and researchers in the field of graph algorithms.

## **Functional Requirements**

# 

# *Figure 1: Use Case Diagram*

# 

# 

# UC 0A : Load in an Algorithm Example Automatically

**0A.1 Precondition**

* A researcher has started up Galant and accessed the Algorithm Editor page.
* Researcher uploads an algorithm file into the Algorithm Editor page and loads it into Galant

**0A.2 Main Flow**

* The researcher goes back into the Algorithm Editor
* The researcher clicks the example dropbox.
* The researcher selects an algorithm example.
* The algorithm example is loaded into the model, and the file name is updated to reflect the chosen algorithm [Replace selected Algorithm]
* The researcher clicks the Export button to download the file onto their computer

**0B.3 SubFlow**

[Replace selected Algorithm] If the researcher already selected one Algorithm, then they can choose another Algorithm. The filename will update and a the researcher is displayed message “Change graph to xxxx[Algorithm name]”

**0B.4 Alternative Flows**

# UC 0B: Load in a Graph Example Automatically

**0B.1 Precondition**

* A researcher has started up Galant and accesses the Graph Editor page.
* Researcher uploads a graph file from their computer and loads it into Galant

**0B.2 Main Flow**

* The researcher opens up the Graph Editor again
* The researcher goes to the example.
* The researcher selects a Graph example.
* The Graph example is loaded into the model, and the file name is updated to reflect the chosen algorithm [Replace selected Graph].
* The researcher clicks the Export button and downloads the graph onto their computer

**0B.3 SubFlow**

[Graph selected Algorithm] If the researcher already selected one Graph, then they can choose to upload another Graph. The filename will update and a pop-up window will appear and show “Change graph to xxxx[Graph name]”

**0B.4 Alternative Flows**

# UC 1: Synchronizing the Graph and Graph Text

**1.1 Preconditions**

* A researcher has started up Galant and opens up the Graph Editor
* Researcher loads in a graph file from their computer onto Galant

**1.2 Main Flow**

* A researcher makes a change on either the visual graph or the editor text (ex: changing the weight of the node).
* The researcher saves the changes.
* The change shows up in the text or the graph.
* The researcher clicks the Export button to download the modified graph from Galant onto their computer

**1.3 Subflows**

* [Simultaneous Changes] If a student makes a change on the graph and a change on the text before saving, only the first change gets implemented.
* [Save Graphs to Computer] The researcher should be able to locally save a PDF of their current graph image.

**1.4 Alternative Flows**

* [Beyond The Bounds] If a student changes the position of the node outside of the bounds of the graph editor then there will be a message saying that it cannot be moved there and the position won’t change.

# 

# UC 2: Animation features

**2.1 Preconditions**

* A researcher has started Galant.
* A graph has been loaded onto Galant.
* An algorithm has been loaded onto Galant.

**2.2 Main Flow**

* Enter a starting node which will be the starting point of the algorithm.
* Step through and back in the algorithm.
* As the algorithm sorts the nodes, the nodes should positionally move.
* The nodes should move in accordance with the algorithm and the order it is sorting them in.
* The researcher clicks the Export button to download the modified graph from Galant onto their computer

**2.3 Subflows**

* [Sorting Algorithms] When a researcher uploads a sorting algorithm file, the nodes should be positionally moving to locations and presented in the order they are currently sorted as the researcher steps through the algorithm. The researcher can manually move the nodes for these algorithms but the changes do not persist and will be reverted when they go to the next step.
* [Other Algorithms] When a researcher uploads any other type of algorithm file, the nodes should not be positionally moved by the algorithm. However, the researcher can choose to manually move the nodes and these changes will persist as they step through the algorithm and go on to the next steps.

**2.4 Alternative Flows**

* [Invalid Algorithm/Graph File] If the researcher uploads an invalid or uneditable graph file or algorithm file then the program should yield an error message detailing that the researcher cannot step through the current graph with the current algorithm - and no nodes or branches should move and be highlighted.

# UC 3: Node and Edges Attributes

**3.1 Preconditions**

* The researcher has started up Galant and opened the Graph Editor
* The researcher uploads a graph.

**3.2 Main Flow**

* Once the researcher uploads a graph there is a list of attributes in the graph that can change.
* The researcher selects an attribute to change.
* The researcher enters the change that needs to be made.
* The researcher clicks the Save button and the change will show up on the graph.
* The researcher clicks the Export button to download the modified graph from Galant onto their computer

**3.3 Subflows**

* [Changing the Shading] The researcher will click on the dropdown for the node shading and click on a color in the dropdown that they want to change it to.

**3.4 Alternative Flows**

* [Negative Radius] If the researcher types in a negative number for the radius of the node a message will pop up saying its invalid and the change won’t be made.
* [Negative Thickness] If the researcher types in a negative number for the thickness of the node/edge a message will pop up saying its invalid and the change won’t be made.

# UC 4: Keyboard Shortcuts

**4.1 Preconditions**

* A researcher has started up the Galant and opened the Graph Editor and Algorithm Editor
* A researcher had loaded a graph and algorithm into the respective editors

**4.2 Main Flow**

* The researcher selects/uploads their graph and algorithm.
* The researcher is prompted to enter the starting node.
* The researcher uses any of the listed keyboard shortcuts to change their view of the graph/step through the algorithm.
* The specific component will show up on the graph.
* The researcher clicks the Export button to download the modified graph from Galant onto their computer

**4.3 Subflows**

**4.4 Alternative Flows**

* [Wrong Keyboard Shortcut] If the student presses an invalid keyboard shortcut then nothing will show up on the graph.

# 

# UC 5: Scale .sgf files in Algorithm Editor

**5.1 Preconditions**

* A researcher has started up the Galant
* The researcher has a valid .sgf file they want to upload

**5.2 Main Flow**

* The researcher uploads the .sgf file to the algorithm editor.
* The algorithm is successfully loaded into Galant.
* The researcher is able to scale coordinates horizontally and vertically proportional to window dimensions.
* The researcher clicks the Export button to download the modified graph from Galant onto their computer

**5.3 Subflows**

* [S1] The researcher can edit the .sgf file and save it.

**5.4 Alternative Flows**

* [E1] If the .sgf is invalid then the researcher is prompted to upload another file.

# UC 6: Algorithm/Graph Files Persist on Webpage Refresh

**6.1 Preconditions**

* The student has opened Galant and uploaded a graph file and algorithm file.
* They are viewing, editing, and using a Graph and/or an Algorithm loaded into Galant.

**6. 2 Main Flow**

* The researcher reloads their browser.
* The previous graph and algorithm shall remain loaded in the application and not be reset.
* The researcher clicks the Export button to download the modified graph from Galant onto their computer

**6.3 Subflows**

* [No Files Loaded] If no files were loaded into Galant then once the browser is reloaded then the application will reload and load in/show the example Graph and Algorithm files as intended in UC0.

**6.4 Alternative Flows**

# UC 7: Skip to End Functionality

**7.1 Preconditions**

* The researcher has the Galant system running.
* The researcher opens up the Graph Editor and Algorithm Editor
* The researcher has a valid algorithm and graph loaded into Galant.

**7.2 Main Flow**

* The researcher is able to skip to the end of the algorithm to see the end result of the graph.
* The researcher clicks the Export button to download the modified graph from Galant onto their computer

**7.3 Subflows**

* [At End of Algorithm] If the researcher is already at the end of the algorithm and tries to skip to the end, nothing changes.

**7.4 Alternative Flows**

* [E1] There is an error within the algorithm or graph, an error message via pop-up is provided to the researcher.
* [E2] If the algorithm runs forever, a hard limit is reached and the algorithm will be automatically stopped.

# UC 8: Preference Panel

**8.1 Preconditions**

* The researcher has the Galant system running and opens up the Graph Editor and Algorithm Editor (optional) to load in a Graph and Algorithm (optional)

**8.2 Main Flow**

* The researcher can edit the preferences panel in which to configure their researcher preferences (Font, text, color,...etc).
* The researcher preferences will be applied automatically after saving.

**8.3 Subflows**

* [S1] The researcher can upload a .txt file with preferences listed. This will be saved as future default preferences.
* [S2] The researcher can edit preferences after a default preferences file has been loaded in.
* The researcher clicks the Export button to download the modified graph from Galant onto their computer

**8.4 Alternative Flows**

* [E1] If the uploaded preferences file is invalid, the researcher receives an error message via pop-up box.

## **Non-Functional Requirements**

[NFR1] The system must have ADA standard text fonts, colors, and sizes.

[NFR2] All changes to the code shall be annotated with noted documentation on GitHub.

[NFR3] Program must be web-based and run on multiple browsers including: Google Chrome, Google Chromium, Mozilla Firefox, and Brave.

## **Constraints**

[C1] The software must be implemented using JavaScript. The current JavaScript implementation uses React and Cytoscape for researcher interaction and graph drawing, respectively.

# Design

*Author(s): Vitesh Kambara, Julian Madrigal*

*Reviewer(s)/Editor(s): Team*

## The design for Galant-JS is intended to emulate the original Java version of the application Galant. This transition process of Galant from a Java-based application to a JavaScript application has been in progress for three semesters now and our design focuses on enhancements, modifications, and improvements desired by Dr. Stallmann to be made to previous semesters’ implementations. Our design changes consist of updating the High-Level Design, refactoring the Low-Level Design, and revising the GUI Designs of previous semester teams. Each of these changes to our design is intended to fulfill our finalized set of requirements for Galant-JS to transform it into a more flexible and functional application for students and researchers. Developers and researchers looking into Galant-JS can look at these designs and their documentation to better understand the application on an internal level.

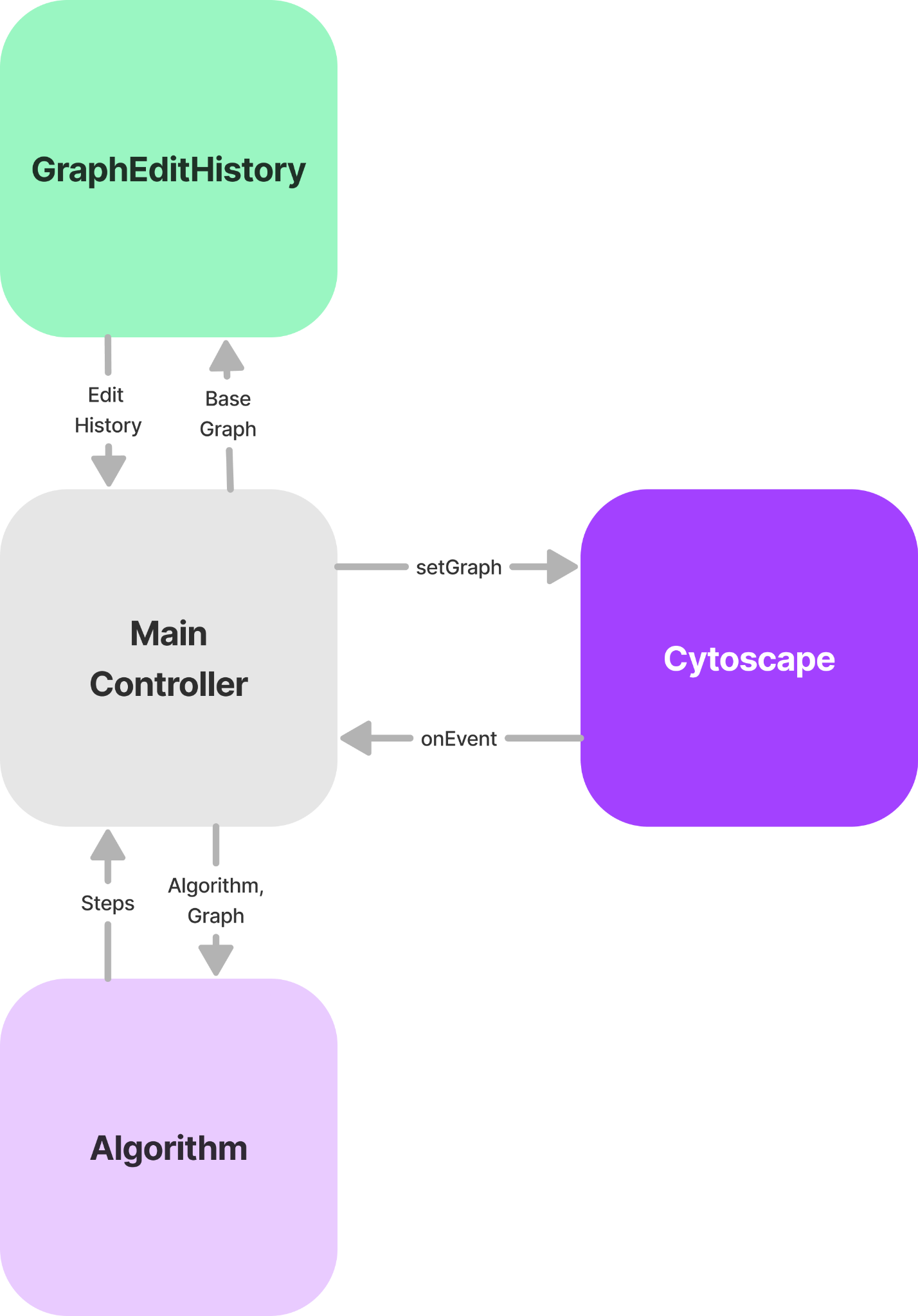
## **High-Level Design**

Since the project is a continuation of a previous project from the Fall 2023 Stallmann Team, many of our changes were based on their final design of galant-js. For reference, Figure 1.1 contains the high-level design of galant-js at the end of the Fall 2023 team’s project.



*Figure 1.1: Fall 2023 Final High-level Design*

During implementation of our graph edit feature, we encountered some issues that made it difficult to implement the feature without a redesign of how graph state is handled and set into Cytoscape. Since AlgorithmHandler handled setting the graph into Cytoscape, the options were to either write the edit feature as part of AlgorithmHandler, or bring the graph state into main. In our updated High-level design in Figure 1.2, we decided on the latter which would allow us more control over the view.

*Figure 1.2: Spring 2024 StallmannTeam’s High Level Design*

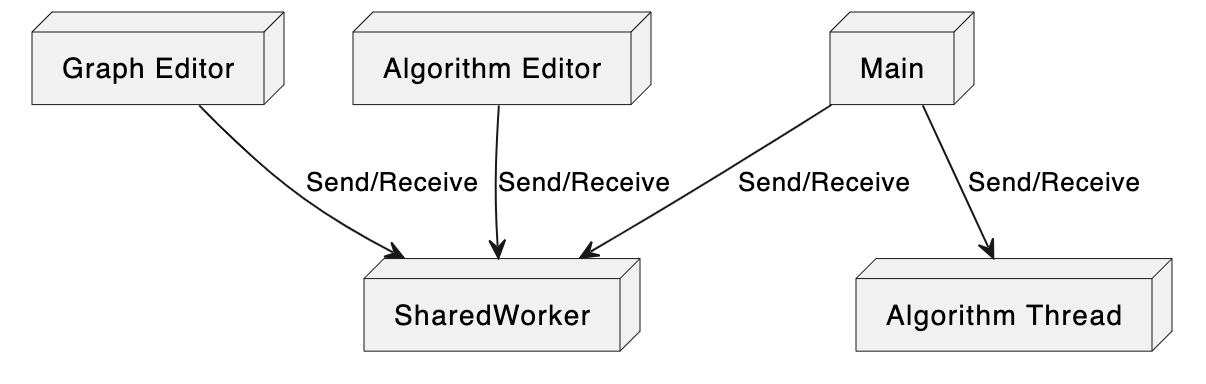
In our updated design, MainController bears the responsibility of determining where to fetch the displayed state from, and sending the graph to Cytoscape for rendering. Having MainController manage the graph view allows us to easily implement multiple ‘modes’ that can be switched between. One of the modes, ‘Algorithm’ will be given algorithm code and a base graph, which it can then use to generate steps, each represented as a GraphSnapshot. These steps can then be accessed by the main controller so that the algorithm step can be displayed if the application is currently in algorithm mode. In algorithm mode, a UI overlay is displayed, which gives the user the ability to change the current step that is set in the algorithm, and therefore Cytoscape’s graph view.

The GraphEditHistory addition to the design was based on a feature requested by Dr. Stallmann. Stallmann requested the ability for the user to change the graph by interacting with the UI, specifically Cytoscape. Furthermore, Dr. Stallmann also wanted the ability to undo and redo between the changes, which would require the application to keep history of previous changes. Creating a new component rather than reading state from Cytoscape was preferable to keep state of changes, along with the fact that data was difficult to retrieve and transform from Cytoscape once it has been passed into Cytoscape. We decided to limit reading data from Cytoscape to only reading events such as right clicking on elements. UI Components such as menus would then be displayed that would apply changes directly to the stored graph state, and create a new history item. Similar to Algorithm, the MainController can read the history of edits made by the user.

The logic behind determining mode in the MainController is relatively simple. By default, edit mode can be entered anytime initially. At the point that an algorithm is loaded, MainController will revert any changes made in edit mode, and switch to algorithm mode. During this context switch, the algorithm is initialized and generates the first step. The user is displayed the UI overlay, and is given the ability to step through the algorithm. Edit mode cannot be entered thereafter, and requires the user to completely remove the algorithm to return edit mode functionality.

**Low-Level Design**

Since Galant-JS is primarily a frontend application, the low level design focuses on communication between components and webpages. Much of the communication between components is achieved through the use of React contexts. Furthermore, in order to communicate between windows, the use of browser APIs such as Service Workers and Shared Workers must be utilized. Therefore, the following outlines such communications and data that may be passed between components or windows.



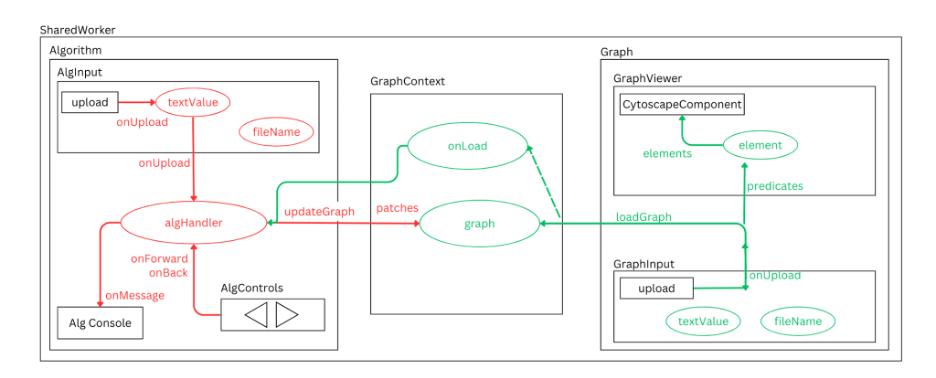
*Figure 2.1: SharedWorker Design*

The SharedWorker design in Figure 2.1 was created by the Fall 2023 Stallmann Team, which were tasked with creating support for multiple windows to display the graph editor and algorithm editor separately. In order to implement multi-window support, the team needed a method to communicate between the editors and the main view. The solution was [SharedWorker](https://developer.mozilla.org/en-US/docs/Web/API/SharedWorker), which provides the ability to send messages between the windows. The GraphEditor is responsible for parsing the graph text and sending the graph object to the main window via SharedWorker. This message is accompanied by message type ‘graph-init’ to describe its payload. AlgorithmEditor is responsible for sending the algorithm code as-is to Main, which will be responsible for running the code. This message is accompanied by the type ‘algo-init’ to describe its payload. SharedWorker serializes these messages, which restricts the content you can pass using the SharedWorkerAPI. Regardless of these constraints, SharedWorker is a requirement for multi-window support, and its existing functionality sufficed and required little modification. The only modification was to allow GraphEditor the ability to listen for updates provided by Main. As part of graph edit mode, the user should be able to save the graph, along with its text representation. This requires us to send the graph back to the editor, which can overwrite the previous tab, or create a new graph tab.

**Graph View Design**

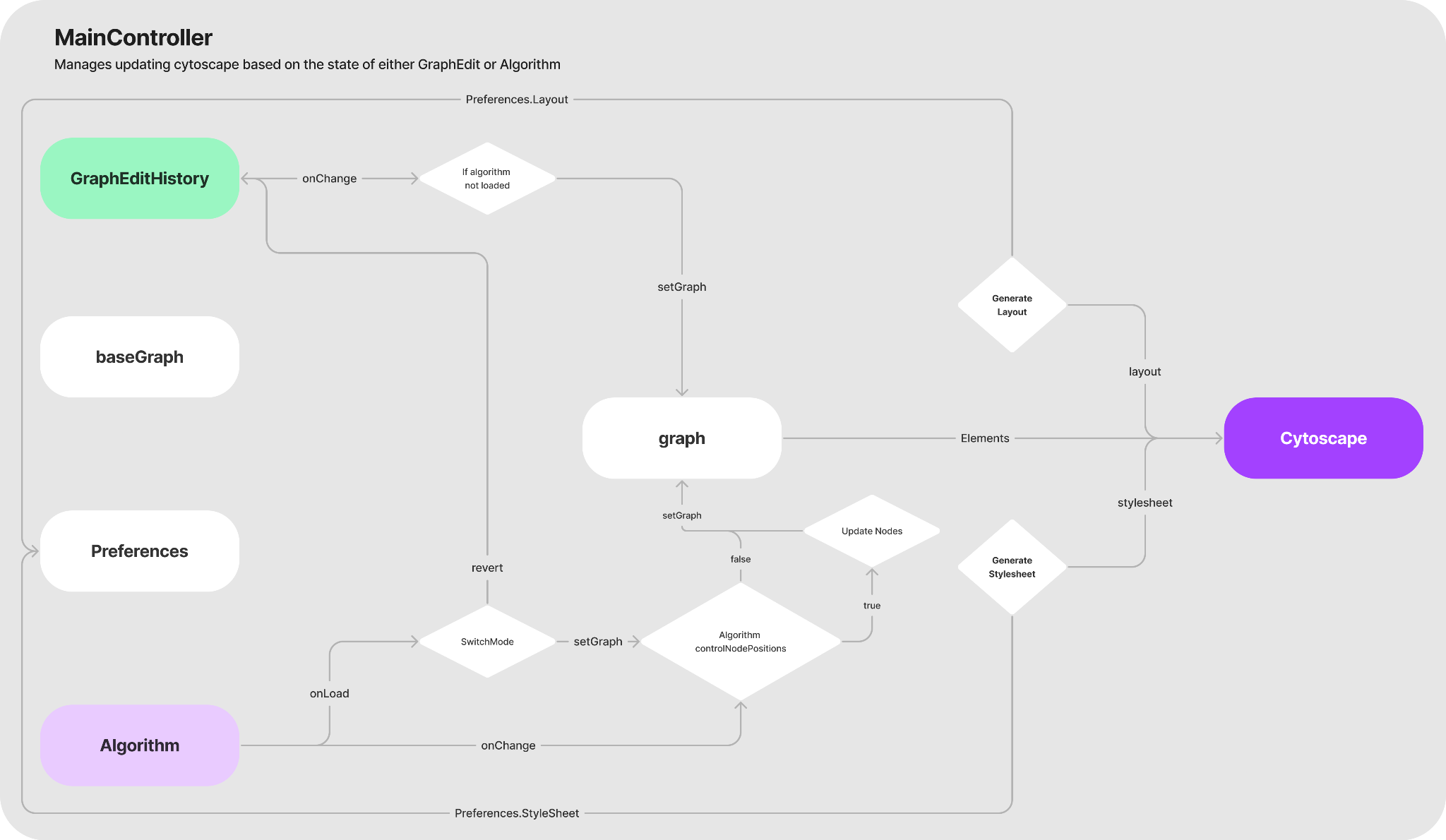


*Figure 2.2: Spring 2023 Stallmann Team Low Level Backend Design*

**

*Figure 2.3: Spring 2023 Stallmann Team Low-Level Frontend Design*

The Spring 2023 Stallmann team and Fall 2023 Stallmann team classified the sections of the graph view as frontend and backend. The ‘backend’ code was still front-end code running in the client’s browser. This backend code consisted of utilities and classes that helped manage the complexity of their graph and algorithm state. During our adaptation of the backend design, many of the elements of the original design remained the same. Our frontend design, however, has drastically changed during our project timeline. Figure 2.4 and Figure 2.5 below show our updated low-level designs. However, these designs are not sectioned as ‘frontend’ and ‘backend’, but instead build on the high-level design’s components and their functionality.

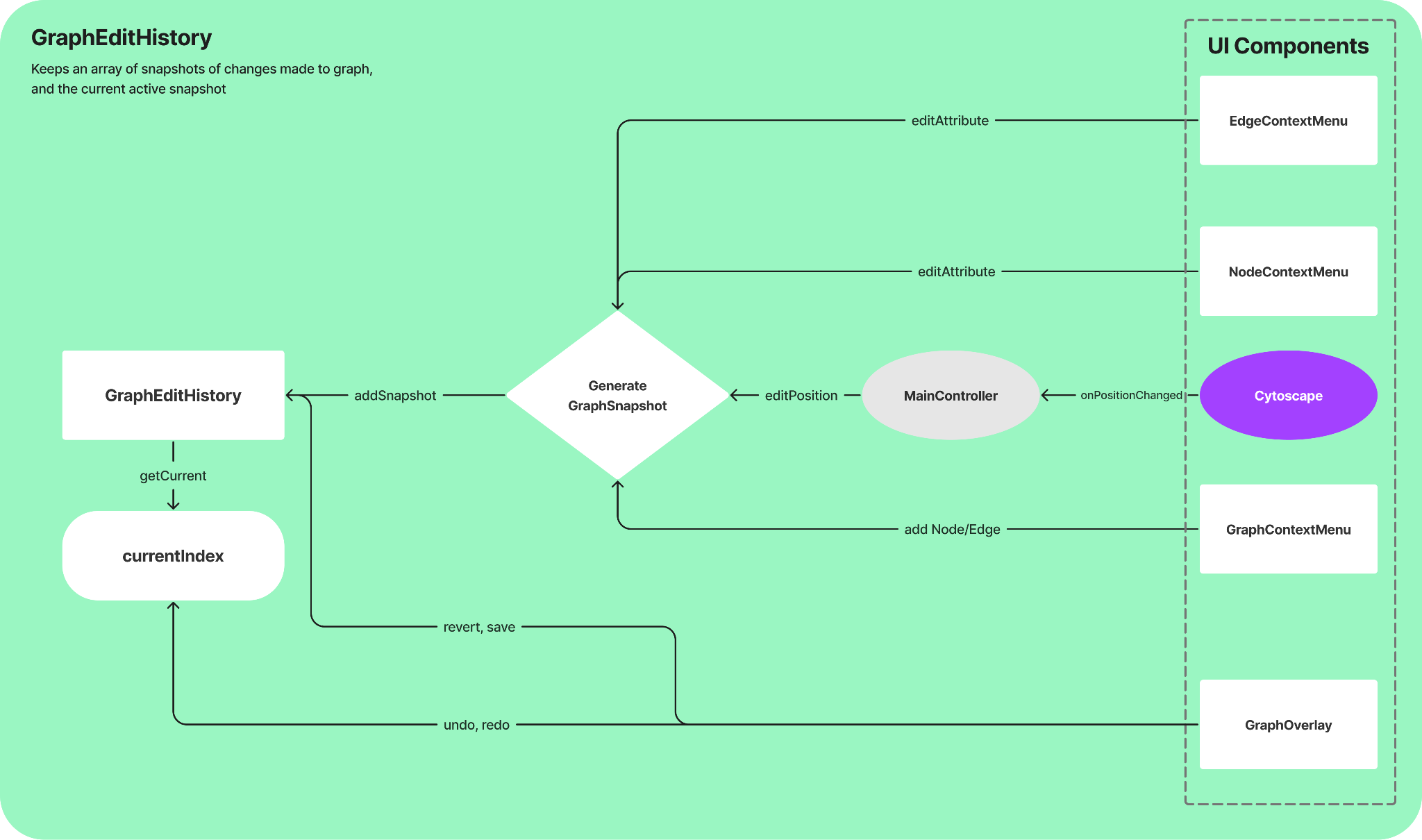
*Figure 2.4: Spring 2024 Stallmann Team Low-Level Design: MainController*

The MainController is responsible for handling the graph view, and which graph is displayed is displayed in Cytoscape. It does this by determining which mode, algorithm mode or edit mode, and fetching the current graph of the mode. The elements of this graph are then used in the rendering of Cytoscape. Along with the graph’s elements, a stylesheet and a layout is generated for Cytoscape. During the generation of these, both layout and stylesheet fetch and consider users preferences from the Preferences object.

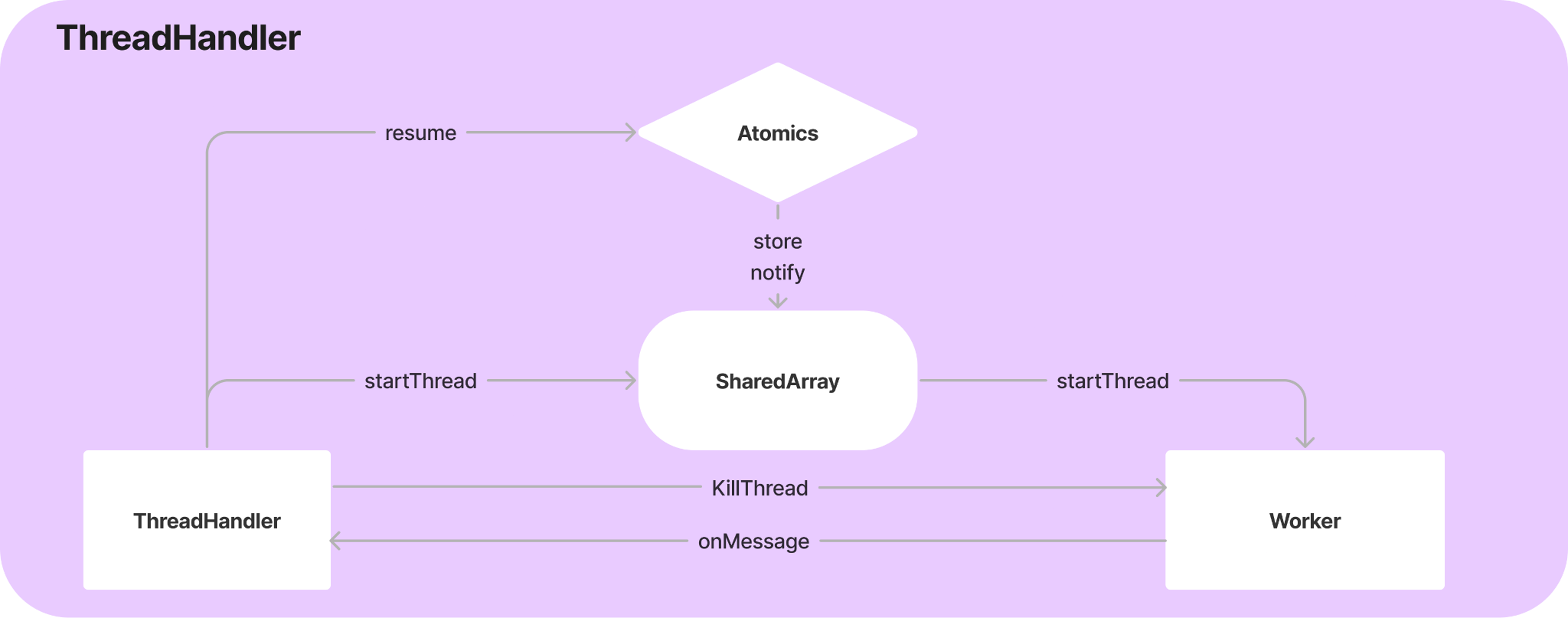
This Preferences object is part of a React context called ‘GraphContext’. This context contains baseGraph, Preferences, and graph, along with services. This context is accessible by descendants of the MainController, which is implemented in index.js of the GraphView page.

By default, when the Graph view page is loaded, an empty graph is loaded. At this initial state, the user can enter edit mode. If the user loads a new graph while in edit mode, any changes will be reverted and the loaded graph will replace the current graph. Lastly, if an algorithm is loaded at any time, the context is switched to algorithm mode. If the edit mode was previously active, any changes are reverted and the algorithm uses the original graph.

One notable feature of MainController is the logic behind applying graph changes based on several variables. For example, during the implementation of node movements, Dr. Stallmann requested two different behaviors between steps. In one instance, an algorithm doesn’t move nodes. In that case, then the correct behavior is to allow the user to move nodes between steps and have the positions persist. In the instance that an algorithm *can* move nodes, then the correct behavior would be for the nodes to ‘snap back’ into their original position between steps. This required extraction of node positions from Cytoscape in the instance of algorithms that don't move nodes. That way those same positions can be applied to the subsequent step and its graph.

*Figure 2.5: Spring 2024 Stallmann Team Low Level Design: GraphEditHistory*

GraphEditHistory stores the state of the edits made to the graph while in edit mode. Components can add snapshots, undo, redo, and revert to GraphEditHistory. These methods are usually applied by the user via the UI overlay. MainController queries Cytoscape for events such as right clicking the graph, nodes, and edges. The appropriate UI component context menu is then displayed to the user. Changes made in the UI are then applied to the GraphEditHistory, by adding a new snapshot. The currentIndex is then updated, and the MainController will update the graph view with the current snapshot.



*Figure 2.6: Spring 2024 Stallmann Team Low Level Design: ThreadHandler*

ThreadHandler exists as part of Algorithm and is a class that is instantiated and used by Algorithm, which passes along the algorithm code, initial graph, and onMessage callback. ThreadHandler is responsible for providing methods for things such as starting, resuming, and killing the thread. Moreover, ThreadHandler listens for messages from Worker and relays them back via the onMessage callback. At the point that ThreadHandler initializes the Worker, it passes along messages for SharedArray, the algorithm code, and the base graph. The SharedArray, along with [Atomics](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Atomics) which are used to resume or yield the thread. Worker will then be responsible for creating the graph, and running the code, which can manipulate the graph and send messages back to ThreadHandler.

*Figure 2.7: Spring 2024 Stallmann Team Low Level Design: Algorithm*

The Algorithm class exists as the main component for managing the algorithm and its state. During its construction, it creates the ThreadHandler from Figure 2.6 and begins waiting for responses by ThreadHandler.

Algorithm also utilizes a class named StepBuilder, which keeps a list of steps, which are basically changes to the graph, and then builds a final GraphSnapshot which represents the next algorithm step.

The algorithm also defines a message handler, which will complete certain tasks depending on the message from the algorithm. For example, ‘rule’ adds a rule, or graph edit, to the step builder. The message ‘step’ defines the completion of rules, and a new algorithm step can be created from StepBuilder. The message ‘completion’ sets the algorithm's completed attribute to true. Lastly, ‘prompt’ and ‘error’ prompt the UI for either input or to inform about an algorithm error.

AlgorithmStepControls defines a React Component that includes buttons that allow the user to step forward or backward in the algorithm. If stepping forward requires generation of a new step, Algorithm calls resume on the thread, and returns the new step when available.

During the implementation of algorithm node movements, it was necessary to differentiate an algorithm that moves nodes from one that doesn’t move nodes. Furthermore, we needed a way that the algorithm could set this configuration, as well as a way for the graph view to access these configurations. The solution was to utilize an AlgorithmConfiguration class, which allows you to apply and retrieve configuration options. This allows an algorithm to define configuration options using:

Algorithm.configure({

controlNodePositions: true

};



and allows retrieving options using:

console.log(anAlgorithm.configuration.controlNodePositions)  
  
// prints true

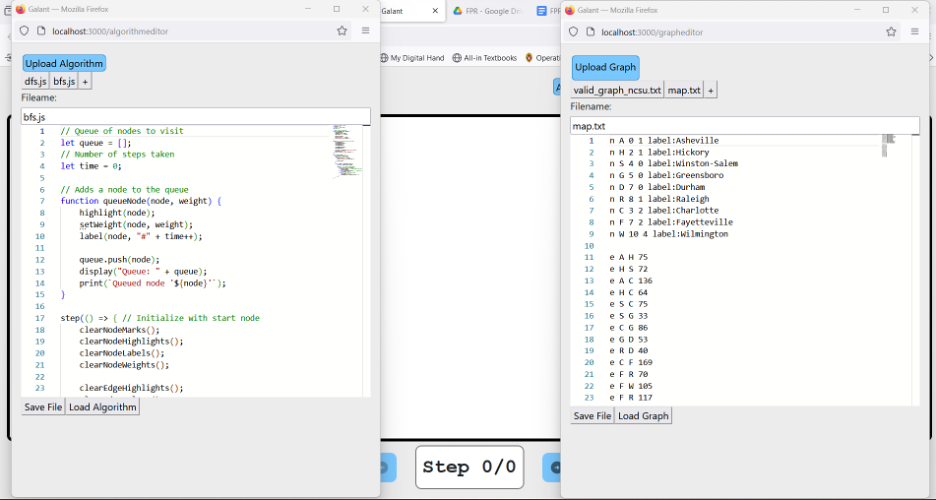


## **GUI Design**

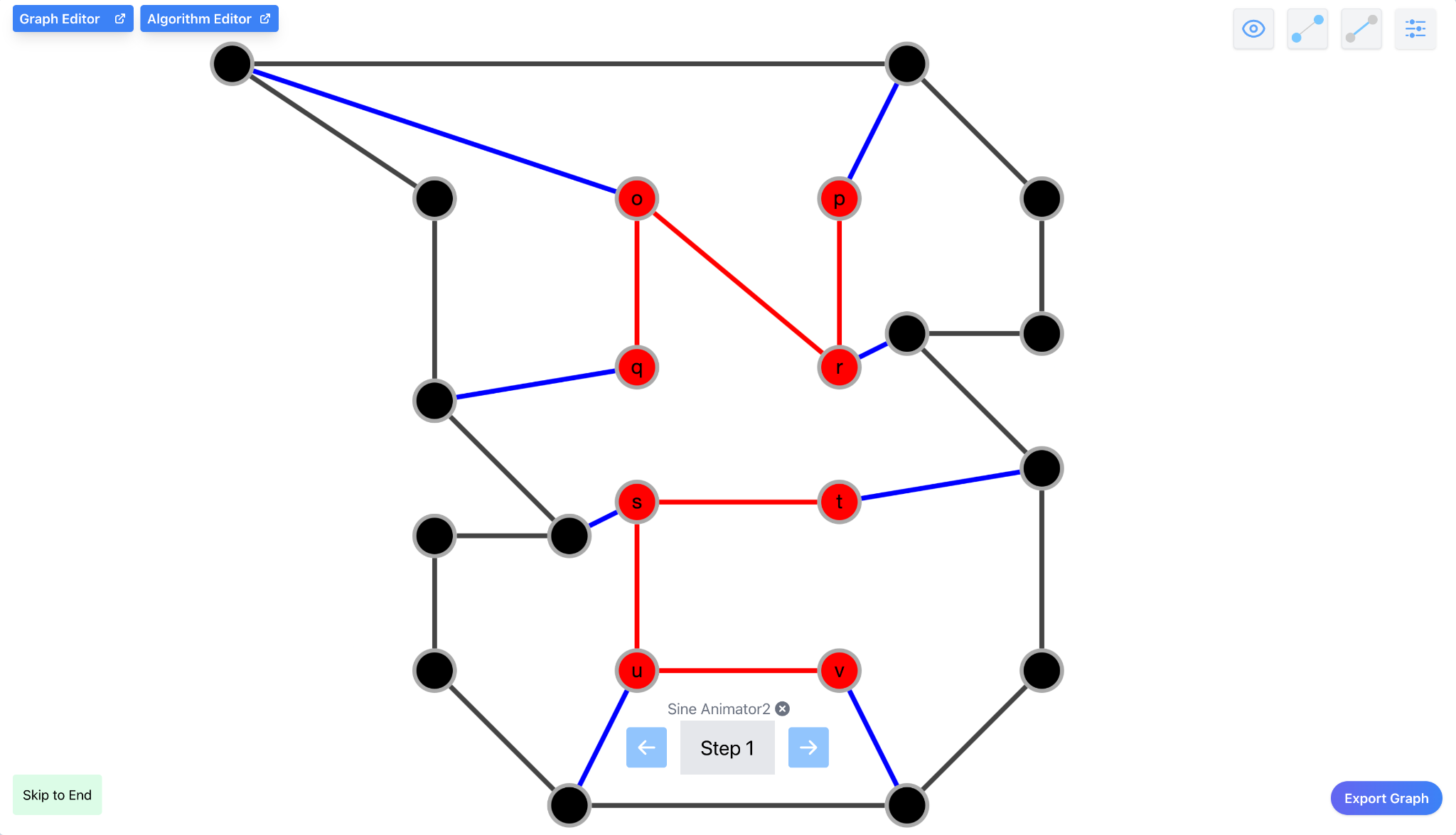
The GUI design of Galant entails the use of shared React components that utilize various libraries. TailwindCSS is the primary use for styling throughout GalantJS. This is a new addition introduced, and previous teams primarily used plain SCSS along with bootstrap to handle styling. HeadlessUI was also introduced to handle creation of components such as dropdowns, popovers, and other basic components. This library also allows implementation of keyboard navigation with ease.



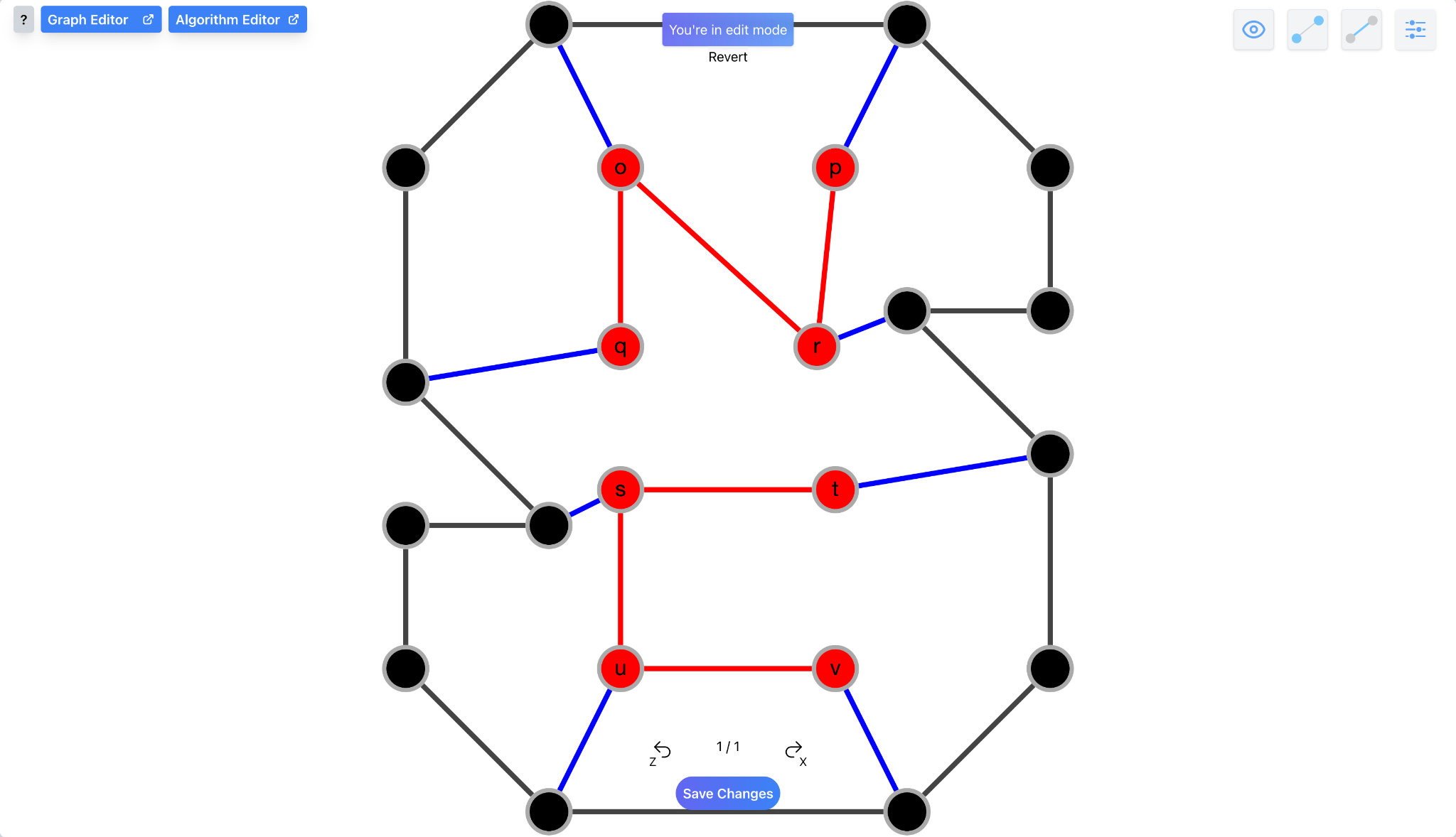
*Figure 3.1: Fall 2023 Stallmann Team Final GUI Design*

*Figure 3.2: Fall 2023 Stallmann Team Final GUI Design: Editor*

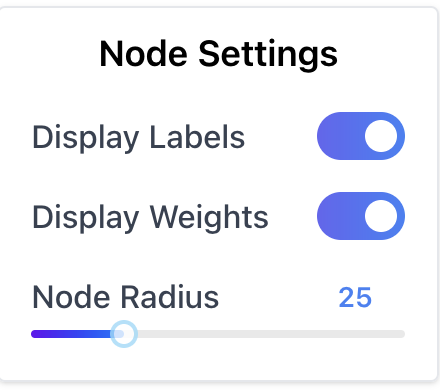
The GUI design in 3.1 and 3.2 are the GUI designs that we started with. The Graph View uses a side panel which contains the preferences a user could set for the graph. This previous design was suboptimal as the graph view was frequently obscured. Stallmann stated his preference to maximize the screen space for the graph view. The editor views also have a suboptimal user experience for various reasons. Firstly, a user could not load in example graphs without having downloaded the file elsewhere. Secondly, tabs were visually unappealing and renaming a file did not automatically rename the tab. Lastly, if the user refreshed an editor page, any content would be lost if the user had not previously saved the files to their local machine.

*Figure 3.3: Spring 2024 Stallmann Team GUI Design: Algorithm Mode*

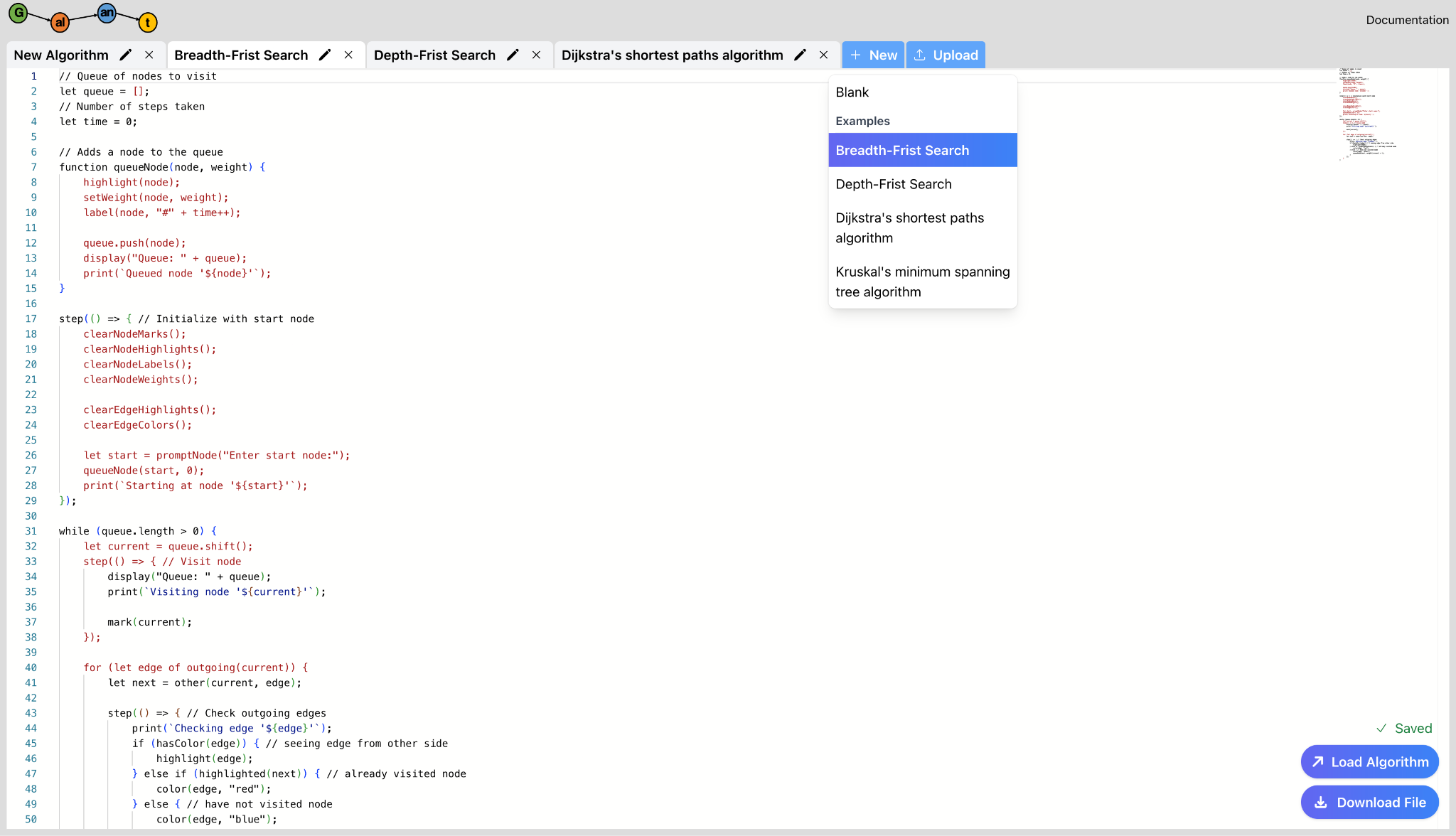
The updated GUI design in Figure 3.3 are the changes that have been made to the graph view page while in algorithm mode. Notable changes are replacing the side panel for preferences for action buttons overlapping the graph view. These buttons open individual menus that allow the user to make changes to the graph, its nodes, edges, and other default preferences. The second change was having the step controls absolutely positioned and overlaid on the graph view. This further increases the screen height of the graph view, which is something Stallmann preferred. During our final iteration, an addition of a Skip-To-End functionality was implemented. One notable addition is the use of Export Graph. This was a request of Dr. Stallmann’s that allows the user to download the current snapshot of the graph, with any modifications the graph has made.

*Figure 3.4: Spring 2024 Stallmann Team GUI Design: Edit Mode*

During the implementation of UC2, which involved allowing the user to modify the graph from Cytoscape, required a new GUI for edit mode. Our objective was to keep a minimal profile and reduce space wasted, while informing the user of edit mode and current functionality. This resulted in Figure 3.4, which includes a “You’re in edit mode” message, along with buttons for the user to undo/redo and save/revert changes.

*Figure 3.5: Spring 2024 Stallmann Team GUI Preference Menu Design*

In Figure 3.5, the panel displayed various actions to manipulate the graph. This design resulted in much of the graph-view being hidden. Furthermore, these buttons didn’t represent the current context of a setting. For example, the ‘toggle node labels’ would not change appearance based on whether the node labels were active or inactive. This resulted in the need for a better and more intuitive UI. Figure 3.6, is the result of the updated GUI components. Multiple dropdowns include the settings that previously existed in the traditional panel. Moreover, the representations of actions are now more intuitive. For example, switches replace the toggle buttons, which display the current status of the toggle. Furthermore, a slider now complements the existing input field, which allows for a better user experience.



*Figure 3.6: Spring 2024 Stallmann Team GUI Editor Design*

The old editor of galant-js had many shortcomings that made for a frustrating user experience. For example, tabs were not well-represented, and made it difficult for the user to switch between editor files. Moreover, the editor was small and did not take full advantage of the screen space. This resulted in Figure 3.6, which includes a nicer UI overall. This includes the new tab components, along with having load and download buttons be an overlay on top of the editor. Lastly, a new dropdown was included for UC0, which allows the user to load existing examples into the editor.

# Implementation

## **Iteration Definition & Current Status**

*Author(s): Minghong Zou*

*Reviewer(s)/Editor(s): Team*

* **Iteration 0** (Finished): Draft requirements, review original team design, update high level, low level and GUI designs, analyze existing features, fix existing bugs.
* **Iteration 1** (Finished): Let Galant-JS automatically load existing examples instead of downloading and uploading (UC0). Synchronizing the Graph and Graph Text, allow users to edit graph files by editing graph windows. (UC1). Also, since we were unable to complete the original UC1 without refactoring most of the code, we dropped the original UC1 with our sponsor’s approval.
* **Iteration 2** (Finished): Allow the user to have the ability to move nodes on the graph (UC2). Update node attribute to represent radius, shading, thickness (UC3). Create shortcut system, allow users to use shortcut to Node Weight Toggle, Node Label Toggle, Edge Weight Toggle, Edge Label Toggle(UC4).
* **Iteration 3** (Finished): Update the upload system to allow .sgf file in algorithm editor (UC5). Store current status and persist it when the web page refreshes (UC6). Allow the user skip steps and directly get an answer if the step is less than 250 steps, else skip to 250 steps.(UC7). Allow users to set their own preference and store them (UC8).

Our current status is that we have finished iteration 0, iteration 1, iteration 2 and iteration 3. The functions work well; currently, we implemented all use cases except UC1, wrote a unit test for it, and passed the final handoff. We deleted UC1 since it seems unlikely that it can be done within the existing time schedule; it requires refactoring most of the code. Dr. Stallman has already deployed our version of Galant-JS to the NCSU Server.

## **Security Considerations**

*Author(s): Christina Albores*

*Reviewer(s)/Editor(s): Team*

With web applications there are always possibilities of XSS attacks and DDOS attacks. After discussion of these potential security issues with our sponsor, he recommended that we not spend time addressing them during our development this semester. He explained to us that the risks would mainly be for the client when they are loading information into the application. As these risks would mainly be pushed to the client, our sponsor informed us that he did not require any new security features to be implemented by our team this semester. Our team brought this recommendation to the attention of the technical advisors so that there was adequate documentation that we addressed security considerations with our sponsor.

However, we do have to be careful in giving access to JavaScript to the user. This means a user could potentially run code that damages their personal browser/machine/etc. We mitigate this by only running their code within a worker thread which has no DOM access. However, there could be vulnerable worker configurations that allow DOM which may be within the Cytoscape plugins the previous team used.

During the development of this project, we had minimal amount of user verification needed and therefore issues of confidentiality, integrity, privacy should be minimized. As a baseline for security, the server that hosts the application has adequate protections and logging of requests by using a standard web server program with built-in security measures. With these security measures in place for the application itself, the security risks should be minimized during our development.

**Confidentiality**:

Confidentiality ensures that sensitive information is only accessible to authorized users. There is currently no sensitive information that is being stored on Galant-JS, so there is no need to put any user restrictions in place. However, if in the future a login process is implemented, that is when problems could arise with securing sensitive information.

**Integrity**:

Integrity ensures that data remains accurate and unaltered during storage and transmission. Without proper safeguards, malicious actors could tamper with data, compromising its integrity. The data they could potentially tamper with would be the example graph and algorithm json that is stored in the cloud. The most damage they could do would be changing the contents of it, but there is no threat of code injection since the code is never run on the site. To address integrity concerns, future teams could implement data validation mechanisms to detect and prevent unauthorized modifications.

**Availability**:

Availability ensures that the application and its resources are accessible to authorized users when needed. Potential availability issues could arise from system failures, network outages, or malicious attacks (DDoS). If our sponsor wanted to mitigate these issues in the future, he could implement a failover mechanism to automatically switch to a backup system when the primary system fails.

**Identification & Authentication**:

Identification and authentication mechanisms are used for verifying the identity of users and ensuring that only authorized individuals can access the application. As of right now, there is no login system for Galant-JS. However, if login capability is implemented in the future, weak authentication mechanisms could lead to unauthorized access. To strengthen identification and authentication, future teams could implement strong password policies for login purposes.

**Accountability**:

Accountability ensures that actions taken within the application can be traced back to the responsible party. Without proper accountability measures, it becomes difficult to identify and mitigate security incidents. As of now Galant-JS does not have any logging records/capabilities.

**Privacy**:

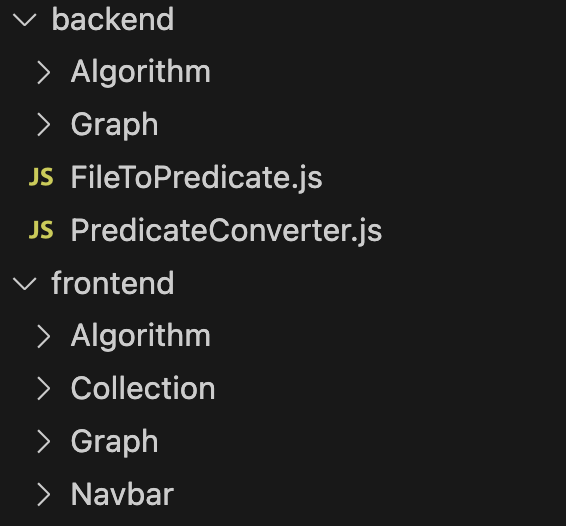
Privacy concerns arise when personally identifiable information (PII) or sensitive data is collected without proper consent. Although there is currently no PII for Galant-JS end users, future teams should use privacy-by-design principles if they are to be handling sensitive information.

## **Project Folder Structure**

*Author(s): Minghong Zou*

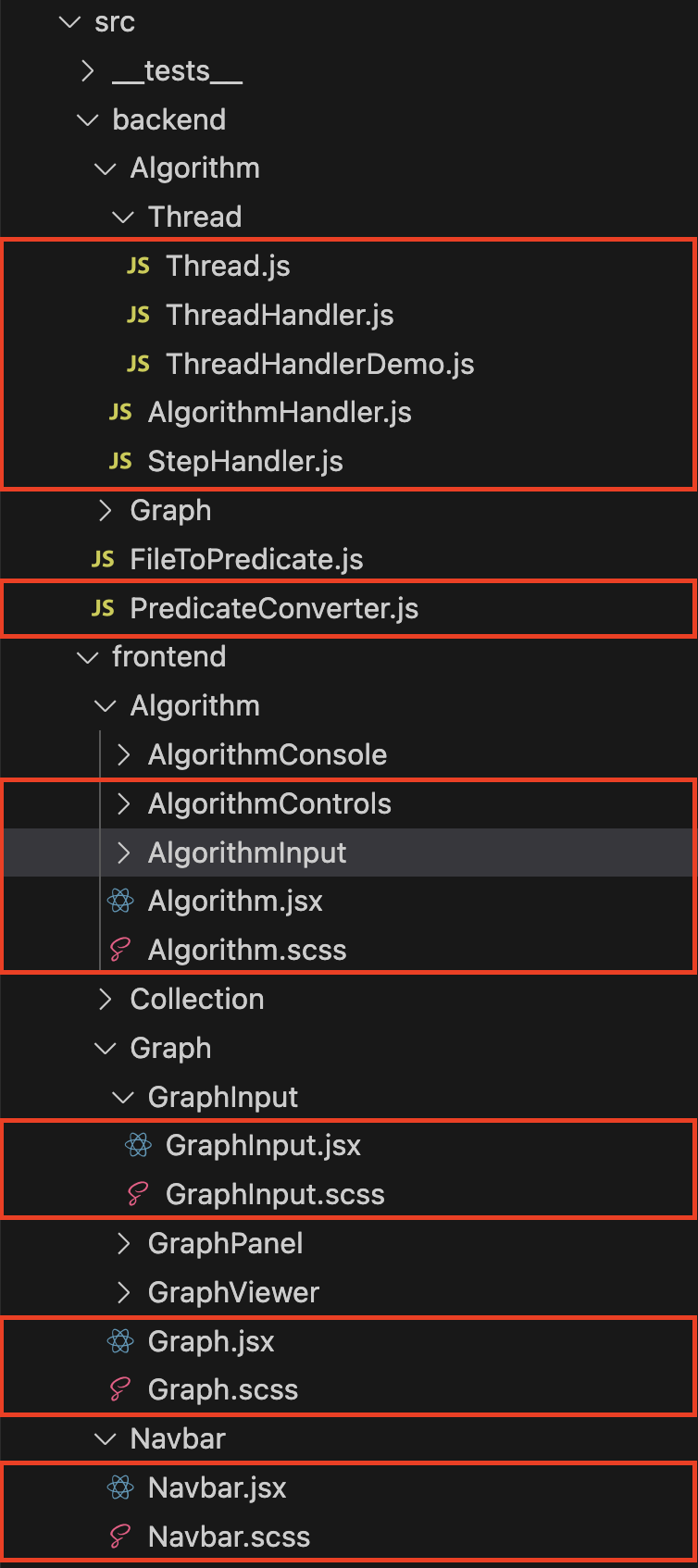
*Reviewer(s)/Editor(s): Christina*

Currently, our project folder structure is organized into two main sections: Pages and Utils. The Pages directory encompasses the front-end components. Within this section, the front-end components—AlgorithmEditor, GraphEditor, and GraphView—are housed in separate folders, mirroring their placement on the respective pages. On the other hand, the back-end functionalities reside in the Utils folder. Here, the Backend directory holds classes and functions vital for the system's internal logic, such as file-to-object type conversions and integration with the algorithm system. Additionally, the Utils folder houses utilities shared across multiple components. Our unit tests, which are automatically executed, are stored in the \_\_tests\_\_ and frontend\_tests folders, as provided by Jest. For manual front-end testing, corresponding tests are located within their relevant front-end folders.



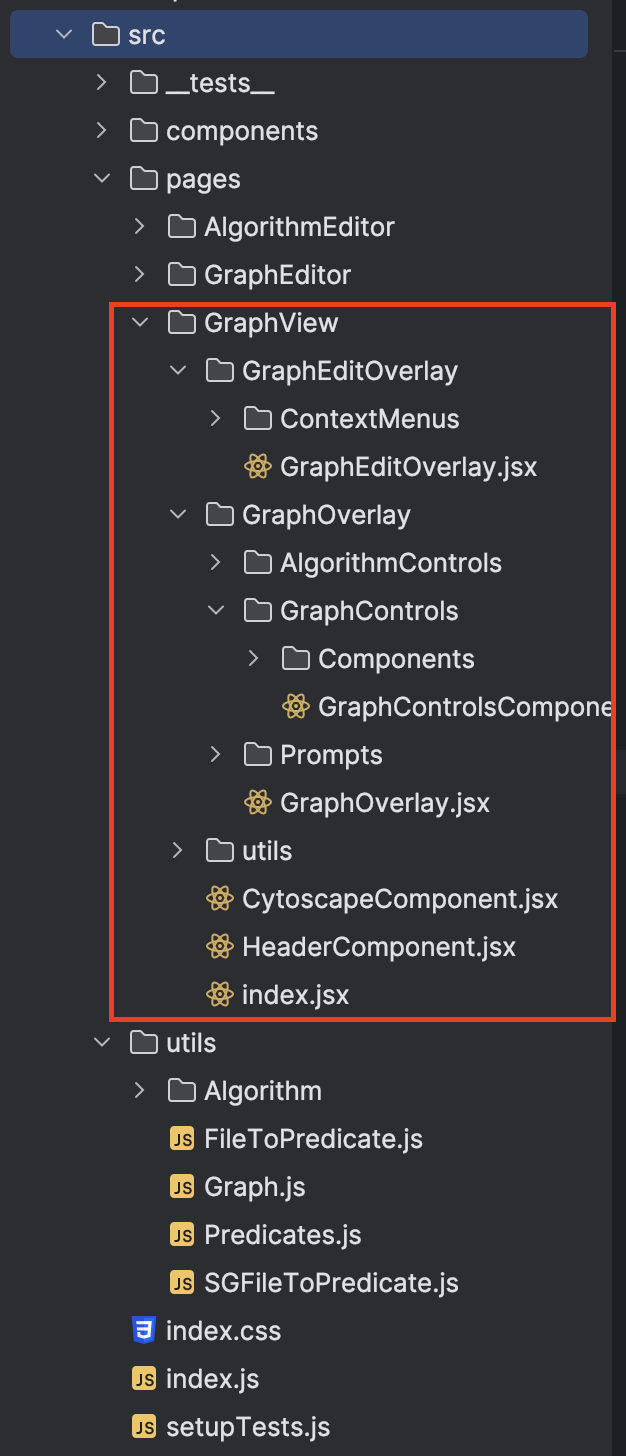
*Figure 4.1: Previous Folder Structure*

The initial design involved organizing the code into separate front-end and back-end folders, with the front end handling presentation and the back end managing calculations.



*Figure 4.2: Previous Folder Structure*

The previous folder structure, as seen above in Figure 4.2, led to frustrations in grasping its intricacies. The file organization was categorized into backend/frontend and algorithm/graph, posing challenges in locating specific files. For instance, dependencies for the main page, the graph view, were dispersed across multiple locations.



*Figure 4.3: Updated Folder Structure*

The updated design in Figure 4.3 shows organizing files into a "pages" classification, such as the GraphView page containing related components and utilities. This restructuring improves component searchability and visualization of dependencies, achieved by categorizing files by pages and replacing "Backend" with "utils," stored closer to their corresponding components.

## **Project Configuration/Settings**

*Author(s): Neha Ramesh*

*Reviewer(s)/Editor(s): Team*

In our project setup, configuration is stored within the 'config' folder. A significant portion of our configuration revolves around Jest, a JavaScript testing framework utilized extensively across various components of the project. Within the Jest configuration, several files play essential roles: 'babel transform,' 'css transform,' and 'file transform' JavaScript files. These transformers serve to adapt different elements of JavaScript, such as CSS, Babel, and files, ensuring compatibility with Jest testing.

Additionally, a file named 'createEnvironmentHash.js' handles configuration for file caching, a feature essential for offline usage. Alongside this, several other configuration files contribute to our setup. These include 'env.js,' which establishes environment variables tailored for the React App environment, 'httpConfig.js,' responsible for configuring HTTPS settings for a Node.js application, and 'modules.js' and 'path.js,' which respectively configure modules and file paths essential for the codebase and project.



*Figure 5.1: Fall 2023 Stallmann Team Babel Transform Configuration File*

In our project, Babel, transforms JavaScript code to ensure compatibility, particularly for Jest testing. It handles various elements such as CSS and files, adapting them to meet the requirements of our testing framework. Additionally, Babel configures file caching through 'createEnvironmentHash.js' for offline usage, alongside managing environment variables, HTTPS settings, modules, and file paths crucial for the project's functionality.



*Figure 5.2: Fall 2023 Stallmann Team Config Folder Setup*

Our project's configuration setup, housed in the 'config' folder, revolves around Jest, a key JavaScript testing framework. Within Jest configuration, files like 'babel transform,' 'css transform,' and 'file transform' ensure compatibility by adapting JavaScript elements such as CSS and files.

# Testing

*Author(s): Vitesh Kambara*

*Reviewer(s)/Editor(s): Team*

## **Overall View**

The types of testing that would be appropriate and performed on Galant-JS would be Unit Testing to ensure that code is performing as expected without any errors or failures, and System Testing to ensure that the system is satisfying all the expected requirements and use cases. The combination of these two methods of testing allows for both verification and validation of the application.

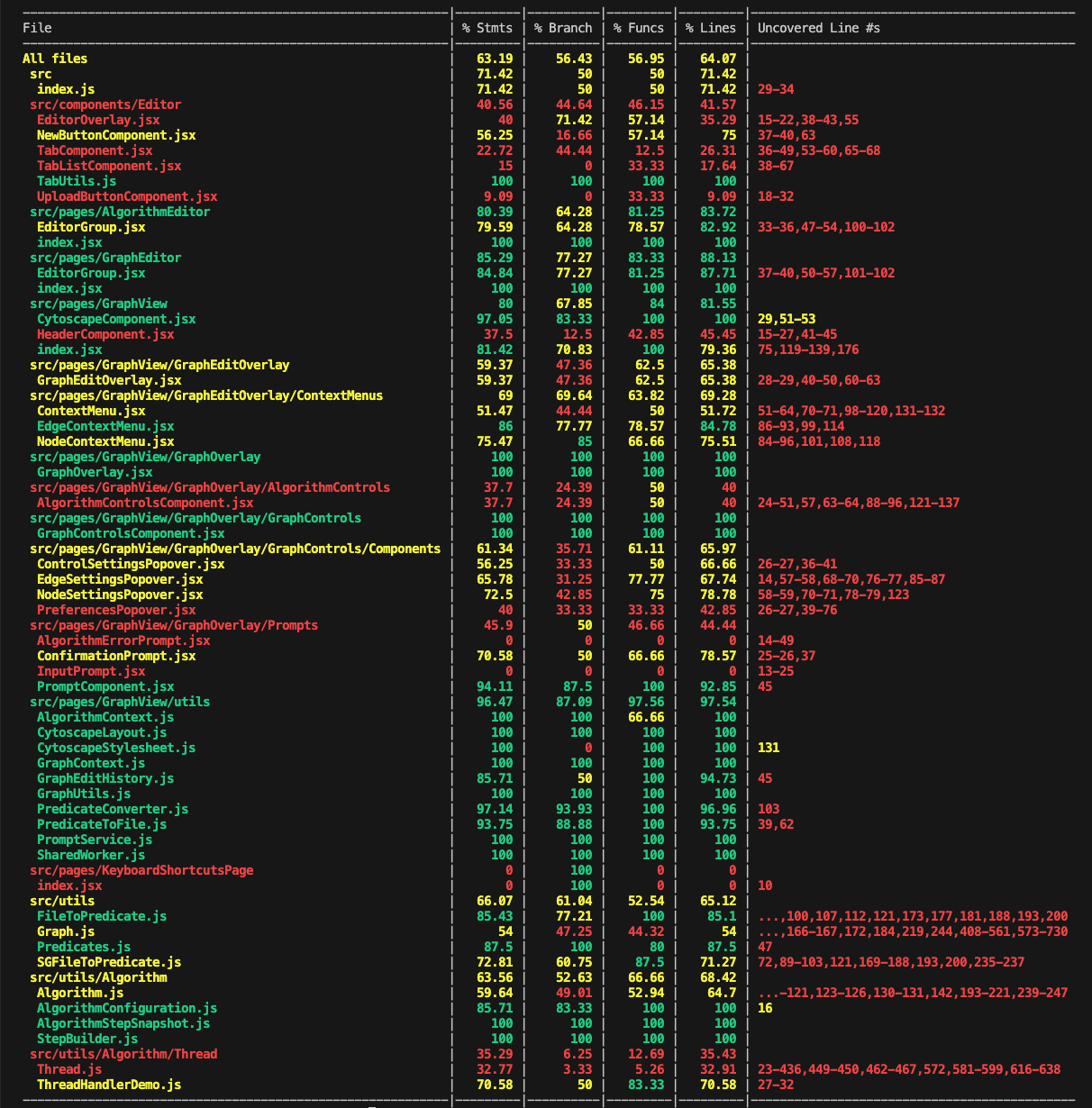
## **Unit Testing**

Our team employed Jest, a JavaScript unit testing framework, to write all unit tests for Galant-JS. Jest was the framework of choice by previous teams working on Galant-JS, and we found that it would be convenient to continue using this framework so that we could build off existing tests and not have to rewrite them.

In our preliminary coverage analysis of the Fall 2023 team’s tests, we found that most tests were failing as their implementation of Shared Workers caused regression amongst the previous tests written by the Spring 2023 team. Among the passing tests and the corresponding testable components, we found that the Fall 2023 team maintained a minimum of 70% statement coverage and 50% branch coverage throughout their tests. As of the time that this IPR is being written, we have only been able to perform minimum testing or testing coverage improvements on Galant-JS, as we have been preoccupied with refactoring the codebase. Our refactoring of the codebase has led to us changing up a lot of previous functionality, project structure, and classes which has caused numerous unit tests to regress and fail and our coverage to ultimately decrease. Since refactoring absorbed a lot of our time initially we made the decision to hold off on redesigning and rewriting unit tests until after refactoring was completed. Dr. Stallmann took no issue with this and it gave us the time and flexibility to fully revamp Galant-JS so that the application would start meeting current requirements and expected behaviors. We did have a thorough System Test Plan that contained plenty of blackbox and acceptance testing to help guide our development and ensure the validity of our application throughout this time.

Due to our refactoring of critical components in Galant-JS, testable components such as ThreadHandler, AlgorithmHandler, and StepHandler failed all tests as some of these components have been removed from the current implementation while some have undergone major changes. In the second half of the semester, we eventually re-wrote or replaced all of these components and re-implemented their tests so that they would begin passing their tests again. However, we did not have enough time to properly implement unit tests for every single component we wrote, and therefore some tests are not meeting our coverage requirements.

Looking at our coverage requirements, we aimed to maintain a threshold of 70% statement coverage and 50% branch coverage used by previous semester teams and make improvements wherever possible. Below are some pictures detailing the final unit testing coverage report of the application (Figure 6.1).





*Figure 6.1: Jest Testing report*

Current coverage results indicate that 76 of the 76 unit tests are passing and all 9 testing suites are passing tests. Lots of code, functions, properties, and components had testing adding to them as the semester progressed. As a team, we planned to implement newer tests that provided better coverage for newly made or refactored components and also for the new use cases that have been implemented, and we did just that. However, future teams may find that some components in the application could still use further testing and may have to implement them for proper verification of the application.

## **Acceptance Testing**

We have devised a System Test Plan to conduct a series of acceptance tests on Galant-JS to ensure the proper functionality of the application and that the intended enhancements are being made without any regression. To do this we went through our requirements for Galant-JS use-case by use-case and determined the appropriate behaviors of the application that should result after successful implementation of the requirement. Structuring our requirements as use cases was particularly useful in this case as it allowed us to more clearly visualize, define, and understand the expected behaviors of the application, and this gave us a better idea of what our System Tests should look like and what situations - acceptance scenarios - they should aim to replicate. The Table 2 below shows our current system test plan entailing our test IDs, descriptions, expected results, and actual results thus far.

**Table 2: System Test Plan Table**

|  |  |  |  |
| --- | --- | --- | --- |
| Test ID | Test Description | Expected Results | Actual Results |
| createBlankGraphs (UC0B) | Preconditions:   1. User has launched Galant-JS. 2. User has opened the graph editor window.   Steps:   1. User clicks the 'New’ button. 2. User selects ‘Blank’ from the dropdown menu. 3. User clicks the 'New’ button. 4. User selects ‘Blank’ from the dropdown menu. | * New tabs ‘Blank’ and ‘Blank1’ are added to the tabs list. * The monaco editor frame is empty as there is no text in the graph file. | SUCCESS   * New tabs ‘Blank’ and ‘Blank1’ are added to the tabs list. * The monaco editor frame is empty as there is no text in the graph file. |
| createPresetGraph (UC0B) | Preconditions:   1. User has launched Galant-JS. 2. User has opened the graph editor window.   Steps:   1. User clicks the 'New’ button. 2. User selects ‘Small Graph’ from dropdown menu | * A new tab called ‘Small Graph’ is added and selected in the tabs list. * The monaco editor frame contains:   n 0 1 1  n 1 2 1  n 2 4 1  n 3 2 2  n 5 3 2  n 7 1 3  n 8 3 3  n 9 4 3  e 0 3  e 0 5  e 1 3  e 2 3  e 2 5  e 3 8  e 3 7  e 5 7  e 5 9 | SUCCESS   * A new tab called ‘Small Graph’ is added and selected in the tabs list. * The monaco editor frame contains:   n 0 1 1  n 1 2 1  n 2 4 1  n 3 2 2  n 5 3 2  n 7 1 3  n 8 3 3  n 9 4 3  e 0 3  e 0 5  e 1 3  e 2 3  e 2 5  e 3 8  e 3 7  e 5 7  e 5 9 |
| loadGraphSuccess (UC0B) | Preconditions:   1. User has launched Galant-JS. 2. User has opened the graph editor window. 3. User clicks the 'New’ button. 4. User selects ‘Small Graph’ from the dropdown menu.   Steps:   1. User clicks the 'Load Graph’ button. | * The graph was loaded successfully. * The graph view on the main page displays the ‘Small Graph’. | SUCCESS   * The graph was loaded successfully. * The graph view on the main page displays the ‘Small Graph’. |
| createBlankAlgorithm (UC0A) | Preconditions:   1. User has launched Galant-JS. 2. User has opened the algorithm editor window.   Steps:   1. User clicks the 'New’ button. 2. User selects ‘Blank’ from the dropdown menu. 3. User clicks the 'New’ button. 4. User selects ‘Blank’ from the dropdown menu. | * New tabs ‘Blank’ and ‘Blank1’ are added to the tabs list. * The monaco editor frame is empty as there is no text in the graph file. | SUCCESS   * New tabs ‘Blank’ and ‘Blank1’ are added to the tabs list. * The monaco editor frame is empty as there is no text in the graph file. |
| createPresetAlgorithm (UC0A) | Preconditions:   1. User has launched Galant-JS. 2. User has opened the algorithm editor window.   Steps:   1. User clicks the 'New’ button. 2. User selects ‘Breadth-First Search’ from dropdown menu | * A new tab called ‘Breadth-First Search’ is added and selected in the tabs list. * The monaco editor frame contains:   // Queue of nodes to visit  let queue = [];  // Number of steps taken  let time = 0;  // Adds a node to the queue  function queueNode(node, weight) {  highlight(node);  setWeight(node, weight);  label(node, "#" + time++);  queue.push(node);  display("Queue: " + queue);  print(`Queued node '${node}'`);  }  step(() => { // Initialize with start node  clearNodeMarks();  clearNodeHighlights();  clearNodeLabels();  clearNodeWeights();  clearEdgeHighlights();  clearEdgeColors();  let start = promptNode("Enter start node:");  queueNode(start, 0);  print(`Starting at node '${start}'`);  });  while (queue.length > 0) {  let current = queue.shift();  step(() => { // Visit node  display("Queue: " + queue);  print(`Visiting node '${current}'`);  mark(current);  });    for (let edge of outgoing(current)) {  let next = other(current, edge);  step(() => { // Check outgoing edges  print(`Checking edge '${edge}'`);  if (hasColor(edge)) { // seeing edge from other side  highlight(edge);  } else if (highlighted(next)) { // already visited node  color(edge, "red");  } else { // have not visited node  color(edge, "blue");  queueNode(next, weight(current) + 1);  }  });  }  } | SUCCESS   * A new tab called ‘Breadth-First Search’ is added and selected in the tabs list. * The monaco editor frame contains:   // Queue of nodes to visit  let queue = [];  // Number of steps taken  let time = 0;  // Adds a node to the queue  function queueNode(node, weight) {  highlight(node);  setWeight(node, weight);  label(node, "#" + time++);  queue.push(node);  display("Queue: " + queue);  print(`Queued node '${node}'`);  }  step(() => { // Initialize with start node  clearNodeMarks();  clearNodeHighlights();  clearNodeLabels();  clearNodeWeights();  clearEdgeHighlights();  clearEdgeColors();  let start = promptNode("Enter start node:");  queueNode(start, 0);  print(`Starting at node '${start}'`);  });  while (queue.length > 0) {  let current = queue.shift();  step(() => { // Visit node  display("Queue: " + queue);  print(`Visiting node '${current}'`);  mark(current);  });    for (let edge of outgoing(current)) {  let next = other(current, edge);  step(() => { // Check outgoing edges  print(`Checking edge '${edge}'`);  if (hasColor(edge)) { // seeing edge from other side  highlight(edge);  } else if (highlighted(next)) { // already visited node  color(edge, "red");  } else { // have not visited node  color(edge, "blue");  queueNode(next, weight(current) + 1);  }  });  }  } |
| loadValidAlgorithmSuccess (UC0A) | Preconditions:   1. User has launched Galant-JS. 2. User has opened the algorithm editor window. 3. User clicks the 'New’ button. 4. User selects ‘Breadth-First Search’ from the dropdown menu.   Steps:   1. User clicks the 'Load Algorithm' button. | * A new tab called ‘Breadth-First Search’ is added and selected in the tabs list. * In the Graph view on the main page the algorithm stepper is displayed with the name ‘Breadth-First Search’. The Step is at 0. * The monaco editor frame contains:   // Queue of nodes to visit  let queue = [];  // Number of steps taken  let time = 0;  // Adds a node to the queue  function queueNode(node, weight) {  highlight(node);  setWeight(node, weight);  label(node, "#" + time++);  queue.push(node);  display("Queue: " + queue);  print(`Queued node '${node}'`);  }  step(() => { // Initialize with start node  clearNodeMarks();  clearNodeHighlights();  clearNodeLabels();  clearNodeWeights();  clearEdgeHighlights();  clearEdgeColors();  let start = promptNode("Enter start node:");  queueNode(start, 0);  print(`Starting at node '${start}'`);  });  while (queue.length > 0) {  let current = queue.shift();  step(() => { // Visit node  display("Queue: " + queue);  print(`Visiting node '${current}'`);  mark(current);  });    for (let edge of outgoing(current)) {  let next = other(current, edge);  step(() => { // Check outgoing edges  print(`Checking edge '${edge}'`);  if (hasColor(edge)) { // seeing edge from other side  highlight(edge);  } else if (highlighted(next)) { // already visited node  color(edge, "red");  } else { // have not visited node  color(edge, "blue");  queueNode(next, weight(current) + 1);  }  });  }  } | SUCCESS   * A new tab called ‘Breadth-First Search’ is added and selected in the tabs list. * In the Graph view on the main page the algorithm stepper is displayed with the name ‘Breadth-First Search’. The Step is at 0. * The monaco editor frame contains:   // Queue of nodes to visit  let queue = [];  // Number of steps taken  let time = 0;  // Adds a node to the queue  function queueNode(node, weight) {  highlight(node);  setWeight(node, weight);  label(node, "#" + time++);  queue.push(node);  display("Queue: " + queue);  print(`Queued node '${node}'`);  }  step(() => { // Initialize with start node  clearNodeMarks();  clearNodeHighlights();  clearNodeLabels();  clearNodeWeights();  clearEdgeHighlights();  clearEdgeColors();  let start = promptNode("Enter start node:");  queueNode(start, 0);  print(`Starting at node '${start}'`);  });  while (queue.length > 0) {  let current = queue.shift();  step(() => { // Visit node  display("Queue: " + queue);  print(`Visiting node '${current}'`);  mark(current);  });    for (let edge of outgoing(current)) {  let next = other(current, edge);  step(() => { // Check outgoing edges  print(`Checking edge '${edge}'`);  if (hasColor(edge)) { // seeing edge from other side  highlight(edge);  } else if (highlighted(next)) { // already visited node  color(edge, "red");  } else { // have not visited node  color(edge, "blue");  queueNode(next, weight(current) + 1);  }  });  }  } |
| uploadPreviousAlgorithm (UC0A) | Preconditions:   1. User has launched Galant-JS. 2. User has opened the algorithm editor window.   Steps:   1. User clicks the Upload button. 2. User selects ‘bfs.js’ from the local system. | * A new tab named ‘bfs.js’ is created and holds the text. * The monaco editor frame contains:   // Queue of nodes to visit  let queue = [];  // Number of steps taken  let time = 0;  // Adds a node to the queue  function queueNode(node, weight) {  highlight(node);  setWeight(node, weight);  label(node, "#" + time++);  queue.push(node);  display("Queue: " + queue);  print(`Queued node '${node}'`);  }  step(() => { // Initialize with start node  clearNodeMarks();  clearNodeHighlights();  clearNodeLabels();  clearNodeWeights();  clearEdgeHighlights();  clearEdgeColors();  let start = promptNode("Enter start node:");  queueNode(start, 0);  print(`Starting at node '${start}'`);  });  while (queue.length > 0) {  let current = queue.shift();  step(() => { // Visit node  display("Queue: " + queue);  print(`Visiting node '${current}'`);  mark(current);  });    for (let edge of outgoing(current)) {  let next = other(current, edge);  step(() => { // Check outgoing edges  print(`Checking edge '${edge}'`);  if (hasColor(edge)) { // seeing edge from other side  highlight(edge);  } else if (highlighted(next)) { // already visited node  color(edge, "red");  } else { // have not visited node  color(edge, "blue");  queueNode(next, weight(current) + 1);  }  });  }  } | SUCCESS   * A new tab named ‘bfs.js’ is created and holds the text. * The monaco editor frame contains:   // Queue of nodes to visit  let queue = [];  // Number of steps taken  let time = 0;  // Adds a node to the queue  function queueNode(node, weight) {  highlight(node);  setWeight(node, weight);  label(node, "#" + time++);  queue.push(node);  display("Queue: " + queue);  print(`Queued node '${node}'`);  }  step(() => { // Initialize with start node  clearNodeMarks();  clearNodeHighlights();  clearNodeLabels();  clearNodeWeights();  clearEdgeHighlights();  clearEdgeColors();  let start = promptNode("Enter start node:");  queueNode(start, 0);  print(`Starting at node '${start}'`);  });  while (queue.length > 0) {  let current = queue.shift();  step(() => { // Visit node  display("Queue: " + queue);  print(`Visiting node '${current}'`);  mark(current);  });    for (let edge of outgoing(current)) {  let next = other(current, edge);  step(() => { // Check outgoing edges  print(`Checking edge '${edge}'`);  if (hasColor(edge)) { // seeing edge from other side  highlight(edge);  } else if (highlighted(next)) { // already visited node  color(edge, "red");  } else { // have not visited node  color(edge, "blue");  queueNode(next, weight(current) + 1);  }  });  }  } |
| uploadPreviousGraph (UC0B) | Preconditions:   1. User has launched Galant-JS. 2. User has opened the graph editor window.   Steps:   1. User clicks the Upload button. 2. User selects ‘unweighted\_10.txt’ from the local system. | * A new tab named ‘unweighted\_10.tx’ will show on the Graph Editor tab list. * The monaco editor frame contains:   # unweighted, based on a Delaunay triangulation  # if directed, all nodes are reachable from node 1  n 1 50 20  n 2 30 20  n 3 5 60  n 4 35 5  n 5 20 50  n 6 10 30  n 7 25 30  n 8 50 60  n 9 40 35  n 10 5 40  e 1 9  e 1 8  e 1 4  e 4 2  e 2 6  e 2 7  e 2 9  e 3 10  e 3 8  e 5 3  e 4 9  e 4 6  e 5 7  e 5 6  e 10 5  e 5 8  e 6 10  e 6 7  e 7 8  e 7 9  e 8 9 | SUCCESS   * A new tab named ‘unweighted\_10.tx’ will show on the Graph Editor tab list. * The monaco editor frame contains:   # unweighted, based on a Delaunay triangulation  # if directed, all nodes are reachable from node 1  n 1 50 20  n 2 30 20  n 3 5 60  n 4 35 5  n 5 20 50  n 6 10 30  n 7 25 30  n 8 50 60  n 9 40 35  n 10 5 40  e 1 9  e 1 8  e 1 4  e 4 2  e 2 6  e 2 7  e 2 9  e 3 10  e 3 8  e 5 3  e 4 9  e 4 6  e 5 7  e 5 6  e 10 5  e 5 8  e 6 10  e 6 7  e 7 8  e 7 9  e 8 9 |
| synchronizingGraphAndText  (UC1) | Preconditions:   * User has launched Galant * User has opened graph editor window   Steps:   * User clicks ‘New’ button from graph editor * User selects ‘Small Graph’ from dropdown menu * User moves the second node to the right one unit on the graph * User hits save | * A new tab ‘Small Weighted Graph’ is added and selected in the tabs list * The monaco editor frame contains before the change:   n 0 1 1  n 1 2 1  n 2 4 1  n 3 2 2  n 5 3 2  n 7 1 3  n 8 3 3  n 9 4 3  e 0 3  e 0 5  e 1 3  e 2 3  e 2 5  e 3 8  e 3 7  e 5 7  e 5 9   * The monaco editor frame contains after the change:   n 0 1 1  n 1 3 1  n 2 4 1  n 3 2 2  n 5 3 2  n 7 1 3  n 8 3 3  n 9 4 3  e 0 3  e 0 5  e 1 3  e 2 3  e 2 5  e 3 8  e 3 7  e 5 7  e 5 9 | SUCCESS   * A new tab ‘Small Weighted Graph’ is added and selected in the tabs list * The monaco editor frame contains before the change:   n 0 1 1  n 1 2 1  n 2 4 1  n 3 2 2  n 5 3 2  n 7 1 3  n 8 3 3  n 9 4 3  e 0 3  e 0 5  e 1 3  e 2 3  e 2 5  e 3 8  e 3 7  e 5 7  e 5 9   * The monaco editor frame contains after the change:   n 0 1 1  n 1 3 1  n 2 4 1  n 3 2 2  n 5 3 2  n 7 1 3  n 8 3 3  n 9 4 3  e 0 3  e 0 5  e 1 3  e 2 3  e 2 5  e 3 8  e 3 7  e 5 7  e 5 9 |
| algorithmMoveNodes (UC2) | Preconditions:   * User has launched Galant-JS * User has loaded dijkstra.js algorithm file into Algorithm Editor   Steps:   * User clicks ‘New’ button from graph editor * User selects ‘Small Graph’ from dropdown menu * User goes to the homepage * User clicks the Step Forward button on the algorithm | * A new tab ‘Small Weighted Graph’ is added and selected in the tabs list in Graph Editor * A new tab ‘Dijkstra’ is added and selected in the Algorithm Editor * The nodes on the graph should be moving with the algorithm representing the pattern in which Djikstra’s algorithm is sorting them in. | SUCCESS   * A new tab ‘Small Weighted Graph’ is added and selected in the tabs list in Graph Editor * A new tab ‘Dijkstra’ is added and selected in the Algorithm Editor * The nodes on the graph should be moving with the algorithm representing the pattern in which Djikstra’s algorithm is sorting them in. |
| runDiskAlgorithm  (UC3) | Preconditions:   * User has launched Galant-JS * User has loaded the NCSU graph file into Graph Editor   Steps:   * User clicks ‘New’ button from algorithm editor * User selects ‘disk animator’ from dropdown menu * User steps through to step 30 * User clicks export graph * User opens the download of the graph text | * The graph itself looks like a disk * The exported graph text looks like this:   n a 103 94 color:black  n b 178 80 color:black  n c 239 60 color:black  n d 280 36 color:black  n e 299 8 color:black  n f 294 -21 color:black  n g 264 -47 color:black  n h 214 -70 color:black  n i 146 -87 color:black  n j 66 -98 color:black  n k -19 -100 color:black  n l -103 -94 color:black  n m -178 -80 color:black  n n -239 -60 color:black  n o -280 -36 color:red  n p -299 -8 color:red  n q -294 21 color:red  n r -264 47 color:red  n s -214 70 color:red  n t -146 87 color:red  n u -66 98 color:red  n v 19 100 color:red  e a b  e a c  e b d  e d e  e c f  e e g  e g h  e f i  e i j  e j k  e h l  e k m  e l n  e m n  e q o color:red  e o r color:red  e r p color:red  e t s color:red  e s u color:red  e u v color:red  e i s color:blue  e v n color:blue  e u m color:blue  e t h color:blue  e r g color:blue  e f q color:blue  e a o color:blue  e p b color:blue | SUCCESS   * The graph itself looks like a disk * The exported graph text looks like this:   n a 103 94 color:black  n b 178 80 color:black  n c 239 60 color:black  n d 280 36 color:black  n e 299 8 color:black  n f 294 -21 color:black  n g 264 -47 color:black  n h 214 -70 color:black  n i 146 -87 color:black  n j 66 -98 color:black  n k -19 -100 color:black  n l -103 -94 color:black  n m -178 -80 color:black  n n -239 -60 color:black  n o -280 -36 color:red  n p -299 -8 color:red  n q -294 21 color:red  n r -264 47 color:red  n s -214 70 color:red  n t -146 87 color:red  n u -66 98 color:red  n v 19 100 color:red  e a b  e a c  e b d  e d e  e c f  e e g  e g h  e f i  e i j  e j k  e h l  e k m  e l n  e m n  e q o color:red  e o r color:red  e r p color:red  e t s color:red  e s u color:red  e u v color:red  e i s color:blue  e v n color:blue  e u m color:blue  e t h color:blue  e r g color:blue  e f q color:blue  e a o color:blue  e p b color:blue |
| runSineAlgorithm  (UC3) | Preconditions:   * User has launched Galant-JS * User has loaded the NCSU graph file into Graph Editor   Steps:   * User clicks ‘New’ button from algorithm editor * User selects ‘sine animator’ from dropdown menu * User steps through to step 30 * User clicks export graph * User opens the download of the graph text | * The graph itself looks like a disk * The exported graph text looks like this:   n a 0 -18 color:black  n b 50 -38 color:black  n c 100 -49 color:black  n d 150 -48 color:black  n e 200 -35 color:black  n f 250 -13 color:black  n g 300 11 color:black  n h 350 33 color:black  n i 400 47 color:black  n j 450 49 color:black  n k 500 40 color:black  n l 550 20 color:black  n m 600 -4 color:black  n n 650 -28 color:black  n o 700 -44 color:red  n p 750 -50 color:red  n q 800 -44 color:red  n r 850 -26 color:red  n s 900 -3 color:red  n t 950 21 color:red  n u 1000 40 color:red  n v 1050 50 color:red  e a b  e a c  e b d  e d e  e c f  e e g  e g h  e f i  e i j  e j k  e h l  e k m  e l n  e m n  e q o color:red  e o r color:red  e r p color:red  e t s color:red  e s u color:red  e u v color:red  e i s color:blue  e v n color:blue  e u m color:blue  e t h color:blue  e r g color:blue  e f q color:blue  e a o color:blue  e p b color:blue | SUCCESS   * The graph itself looks like a disk * The exported graph text looks like this:   n a 0 -18 color:black  n b 50 -38 color:black  n c 100 -49 color:black  n d 150 -48 color:black  n e 200 -35 color:black  n f 250 -13 color:black  n g 300 11 color:black  n h 350 33 color:black  n i 400 47 color:black  n j 450 49 color:black  n k 500 40 color:black  n l 550 20 color:black  n m 600 -4 color:black  n n 650 -28 color:black  n o 700 -44 color:red  n p 750 -50 color:red  n q 800 -44 color:red  n r 850 -26 color:red  n s 900 -3 color:red  n t 950 21 color:red  n u 1000 40 color:red  n v 1050 50 color:red  e a b  e a c  e b d  e d e  e c f  e e g  e g h  e f i  e i j  e j k  e h l  e k m  e l n  e m n  e q o color:red  e o r color:red  e r p color:red  e t s color:red  e s u color:red  e u v color:red  e i s color:blue  e v n color:blue  e u m color:blue  e t h color:blue  e r g color:blue  e f q color:blue  e a o color:blue  e p b color:blue |
| changeNodeColor  (UC4) | Preconditions:   * User has launched Galant-JS * User has loaded the NCSU graph file into Graph Editor   Steps:   * User clicks goes into the graph editor * User changes node a color to purple by typing in purple after the color: like so color:purple * User clicks load graph * User goes back to the main screen | Node a is now colored in purple | SUCCESS  Node a is now colored in purple |
| changeNodeShape  (UC4) | Preconditions:   * User has launched Galant-JS * User has loaded the NCSU graph file into Graph Editor   Steps:   * User clicks goes into the graph editor * User changes node g shape to rectangle by typing in shape:rectangle after the color attribute * User clicks load graph * User goes back to the main screen | Node g is now a rectangle | SUCCESS  Node g is now a rectangle |
| changeNodeAlgorithm(UC4) | Preconditions:   * User has launched Galant-JS * User has loaded the NCSU graph file into Graph Editor * User has opened algorithm Editor   Steps:   * User pastes this algorithm into a new tab in the editor:   const nodeIds = getNodes();  for (const nodeId of nodeIds) {  color(nodeId, "yellow")  setBackgroundOpacity(nodeId, 0.7)  }   * User loads the algorithm * User steps through until step 43 | * Every node in the graph is now yellow | SUCCESS   * Every node in the graph is now yellow |
| redoUndoShortcuts  (UC5) | Preconditions:   * User has launched Galant-JS * User has loaded the NCSU graph file into Graph Editor   Steps:   * User moves node b to the side * User clicks the z key on the keyboard * User clicks the x key on the keyboard | Node b snaps back into its original place after hitting the z key and at the bottom it says 0/1. After hitting the x key the node moves into the place that the user moved it into first and at the bottom it says 1/1 for changes. | SUCCESS  Node b snaps back into its original place after hitting the z key and at the bottom it says 0/1. After hitting the x key the node moves into the place that the user moved it into first and at the bottom it says 1/1 for changes. |
| preferenceShortcuts  (UC5) | Preconditions:   * User has launched Galant-JS * User has loaded the NCSU graph file into Graph Editor   Steps:   * User clicks the c key on the keyboard * User clicks the n key on the keyboard * User clicks the e key on the keyboard * User clicks the p key on the keyboard | After pressing the c key, the controls panel opens up.  After pressing the n key the node settings panel opens up.  After pressing the e key the edge settings panel opens up.  After pressing the p key the preference settings panel opens up. | SUCCESS  After pressing the c key, the controls panel opens up.  After pressing the n key the node settings panel opens up.  After pressing the e key the edge settings panel opens up.  After pressing the p key the preference settings panel opens up. |
| loadSgfFile(UC6) | Preconditions:   * User has launched Galant-JS * User has opened the Graph Editor   Steps:     * Click upload the upload tab * Upload ex\_10.sgf file * User clicks load graph | The graph looks like the orientation is flipped aka its longer on the x axis than the y axis | SUCCESS  The graph looks like the orientation is flipped aka its longer on the x axis than the y axis |
| editSgfFile(UC6) | Preconditions:   * User has launched Galant-JS * User has opened the Graph Editor and uploaded ex\_10.sgf file   Steps:     * User moves node 6 to the side in edit mode * User save changes * User goes to graph editor | When looking at the coordinates of the sgf file in the graph editor one coordinate of node 6 should change and the orientation of the graph should remain the same. | SUCCESS  When looking at the coordinates of the sgf file in the graph editor one coordinate of node 6 should change (mine changed the second coordinate to 122) and the orientation of the graph should remain the same. |
| persistonRefresh  (UC7) | Preconditions:   * User has launched Galant-JS * User has loaded the NCSU graph file into Graph Editor   Steps:   * User refreshes the page | The graph persists on the page after refreshing. | SUCCESS  The graph persists on the page after refreshing |
| finiteSkiptoEnd  (UC8) | Preconditions:   * User has launched Galant-JS * User has loaded the small graph file into Graph Editor   Steps:   * User goes to the algorithm editor and loads in the breadth first search algorithm * User steps through 10 steps after entering starting node 3 * User clicks skip to end button | At the bottom it says step 27/27 and all the nodes and edges are highlighted. | SUCCESS  At the bottom it says step 27/27 and all the nodes and edges are highlighted. |
| infiniteSkiptoEnd  (UC8) | Preconditions:   * User has launched Galant-JS * User has loaded the small graph file into Graph Editor   Steps:   * User goes to the algorithm editor and loads in the sine animator algorithm * User steps through 10 steps * User clicks skip to end button | Algorithm goes through 250 steps and stops at step 260. | SUCCESS  Algorithm goes through 250 steps and stops at step 260. |
| preferencePanel(UC9) | Preconditions:   * User has launched Galant-JS * User has loaded the small graph file into Graph Editor   Steps:   * User opens the preference panel by clicking on the icon with the sliders or clicking p on the keyboard * User clicks on the default node color and chooses red | The nodes of the graph are all the red color that was chosen on the preference panel | SUCCESS  The nodes of the graph are all the red color that was chosen on the preference panel |

These tests have been used and referred to throughout the implementation of our use cases to validate our implementation. Our implementation of Galant-JS is currently able to pass System Tests 1 - 23 and this indicates that we have implemented Use Cases 0A, 0B, 2, 3, 4, 5, 6, 7, 8, and 9 in such a way that no regression is occurring and the intended functionality for these use cases is being achieved.

In order to automate the execution of these tests and obtain detailed coverage reports on the front-end verification of the application, we configured Cypress to automate our system tests. Cypress is an open-source end-to-end testing framework designed for modern web applications, and it is specifically built to simplify the process of testing by providing an efficient way to write and execute user-level tests and report on the coverage and functionality of tests on the application. Last semester’s team used Cypress as well for user-level testing and automation, but as the testing in our project had to undergo significant changes, it took us a while to begin configuring this framework for our project but we ultimately set it up, as seen in the screenshot below (Figure 6.2), and we made consistent use of Cypress in the second half of the semester to validate the functionality of Galant-JS.

# 

*Figure 6.2: Cypress Current Progress*

# Cypress, an open-source end-to-end testing framework, is being implemented in our project to automate front-end tests and generate comprehensive coverage reports. Although configuring Cypress for our project required time due to significant changes, we managed to successfully set it up and utilized it consistently in the second half of the semester to enhance testing efficiency and quality. Some new features and components we added, such as KeyboardShortcutsPage and AlgorithmControls, still need more coverage and testing to ensure validation of these functionalities.

# Task Plan

*Author(s): Christina Albores*

*Reviewer(s)/Editor(s): Team*

**Table 3: Team Member Contact Information**

|  |  |  |
| --- | --- | --- |
| **Name** | **Email** | **Phone Number** |
| Christina Albores | ccalbore@ncsu.edu | 704-577-0871 |
| Julian Madrigal | jmadrig@ncsu.edu | 336-710-2893 |
| Vitesh Kambara | vkkambar@ncsu.edu | 919-200-8968 |
| Minghong Zou | mzou4@ncsu.edu | 919-561-4095 |
| Neha Ramesh | nramesh3@ncsu.edu | 704-904-1055 |

**Table 4: Spring 2024 Stallmann Team Weekly Roles**

|  |  |  |  |
| --- | --- | --- | --- |
| **Week** | **Team Leader** | **Technical Officer** | **Documentation Officer** |
| 0 | Minghong | Neha | Christina |
| 1 | Julian | Christina | Minghong |
| 2 | Vitesh | Minghong | Julian |
| 3 | Neha | Julian | Vitesh |
| 4 | Christina | Vitesh | Neha |
| 5 | Minghong | Neha | Christina |
| 6 | Julian | Christina | Minghong |
| 7 | Vitesh | Minghong | Julian |
| 8 | Neha | Julian | Vitesh |
| 9 | Christina | Vitesh | Neha |
| 10 | Minghong | Neha | Christina |
| 11 | Julian | Christina | Minghong |
| 12 | Vitesh | Minghong | Julian |
| 13 | Neha | Julian | Vitesh |

**Table 5: Fall 2023 Stallmann Team Weekly Roles - Sponsor Meeting**

|  |  |  |
| --- | --- | --- |
| **Meeting** | **Facilitator** | **Scribe** |
| 1 | Julian | Vitesh |
| 2 | Vitesh | Neha |
| 3 | Neha | Christina |
| 4 | Vitesh | Minghong |
| 5 | Minghong | Julian |
| 6 | Julian | Vitesh |
| 7 | Christina | Neha |
| 8 | Neha | Christina |
| 9 | Christina | Minghong |
| 10 | Minghong | Julian |
| 11 | Julian | Vitesh |
| 12 | Vitesh | Neha |
| 13 | Neha | Christina |
| 14 | Christina | Minghong |
| 15 | Julian | Vitesh |
| 16 | Vitesh | Neha |

The following task plans shown below provides a list of all development activities and their completion date. Project task tracking occurred throughout the duration of the semester and scheduled work items and tasks were marked with the current status of development. As we worked through the iterative development process we updated this task plan accordingly.

**Iteration 0: (Jan.23 - Feb. 5)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Item** | **Owner(s)** | **Due Date** | **Status** |
| Ground Rules | Christina, Team | 1/18 | Completed |
| Initial Sponsor Meeting | All | 1/23 | Completed |
| Drafting Preliminary Requirements | All | 1/30 | Completed |
| Weekly Sponsor Meeting #1 | All | 1/30 | Completed |
| Merge sd-2023-8 branch into the main branch on the Stallmann repo | All | 1/30 | Completed |
| Finish the IPR draft report | All | 2/1 | Completed |
| UC0 Research | Minghong, Christina | 2/1 | Completed |
| UC1 Research | Julian, Vitesh | 2/1 | Completed |
| UC2 Research | Neha | 2/1 | Completed |
| Created Cytoscape playground repo | Julian | 2/1 | Completed |
| Weekly Sponsor Meeting | All | 2/6 | Completed |
| Setup last team’s github actions | Christina, Vitesh, Julian | 2/6 | Completed |
| OPR1 Draft | Vitesh,Team | 2/8 | Completed |

**Iteration 1: (Feb. 5- Feb. 26)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Item** | **Owner(s)** | **Due Date** | **Status** |
| OPR1 | Vitesh, Team | 2/8 | Completed |
| Refactor Graph Editor | Julian | 2/15 | Completed |
| Refactor Algorithm Editor | Christina | 2/15 | Completed |
| UC0 | Julian | 2/10 | Completed |
| ~~UC1~~ | ~~Christina, Neha~~ | ~~2/26~~ | ~~In Progress~~ |
| Testing for UC2:  Algorithm.js (NR)  GraphEditHistory.js (CA) | Christina, Neha | 2/26 | Completed |
| UC2: Implementing  Algorithm, StepBuilder, GraphEditHistory, Context Menus, EditOverlay | Julian | 2/26 | Completed |
| UC2: Implemented PredicateToFile.js | Minghong | 2/26 | Completed |
| UC2: Save Graph text to Editor on ‘Save’ press | Vitesh | 2/26 | Completed |
| Testing: See which tests we might need to change/add. Research Jest. | Vitesh | 2/26 | Completed |
| System Test Plans | Team | 2/10 | Completed |
| Document what backend design changes we will be making for the syncing component.  Editor Graph View: Edit State: Undo, Redo | Team | 2/10 | Completed |
| Finish documenting code base | Team | 2/29 | Completed |
| IPR: Executive Summary | Neha | 3/8 | Completed |
| IPR: Sponsor Background | Neha | 3/8 | Completed |
| IPR: Problem Description | Christina | 3/8 | Completed |
| IPR: Proposed Solution & Project Goals/Benefits | Minghong | 3/8 | Completed |
| IPR:Resources Needed | Minghong | 3/8 | Completed |
| IPR: Risks & Risk Mitigation | Minghong | 3/8 | Completed |
| IPR: Development Methodology | Vitesh | 3/8 | Completed |
| IPR: System Requirements | Christina | 3/8 | Completed |
| IPR: Low-Level Design | Julian | 3/8 | Completed |
| IPR: High-Level Design | Julian | 3/8 | Completed |
| IPR: GUI Design | Julian | 3/8 | Completed |
| IPR: Implementation: Iteration Definition & Current Status | Minghong | 3/8 | Completed |
| IPR: Implementation: Security Considerations | Christina | 3/8 | Completed |
| IPR: Implementation: Project Folder Structure | Minghong | 3/8 | Completed |
| IPR: Implementation: Project Configuration/Settings | Neha | 3/8 | Completed |
| IPR: Overall View | Vitesh | 3/8 | Completed |
| IPR: Unit Testing | Vitesh | 3/8 | Completed |
| IPR: Acceptance Testing | Vitesh | 3/8 | Completed |
| IPR: Other Testing | Vitesh | 3/8 | Completed |
| IPR: Task Plan | Christina | 3/8 | Completed |
| OPR2 | Julian, Christina | 2/28, 3/7 | Completed |
| Sent Email to Dr. Stamann for OPR2 invite | Christina | 2/27 | Completed |

**Iteration 2: (Feb. 27- Mar. 27)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Item** | **Owner(s)** | **Due Date** | **Status** |
| Finish documenting code base | Team | 3/27 | Completed |
| Testing for UC2:  Algorithm.js (NR)  GraphEditHistory.js (CA) | Christina, Neha | 3/6 | Completed |
| OPR2 | Julian, Christina | 3/7 | Completed |
| IPR: Development Methodology | Vitesh | 3/8 | Completed |
| IPR: Implementation: Project Configuration/Settings | Neha | 3/8 | Completed |
| IPR: Unit Testing | Vitesh | 3/8 | Completed |
| IPR: Acceptance Testing | Vitesh | 3/8 | Completed |
| IPR: Other Testing | Vitesh | 3/8 | Completed |
| Update System Test Plan | Neha, Christina | 3/8 | Completed |
| IPR Final Review | Team | 3/8 | Completed |
| Iteration 3 Review | Team | 3/8 | Completed |
| OPR3 | Neha, Minghong | 4/2 | Completed |
| Rough Draft of P&P Poster Due by 11:59 pm via Submission System | Team | 3/24 | Completed |
| Times Available for OPR 3 Prep | Neha, Minghong | 3/25-29 | Completed |
| Optional Draft OPR 3 Slides Due by 11:59 pm via submission system | Neha, Minghong | 3/27 | Completed |
| UC 3: Animation features | Julian | 3/27 | Completed |
| UC 4: Node and Edges Attributes | Christina | 3/27 | Completed |
| UC 5: Keyboard Shortcuts | Vitesh | 3/27 | Completed |
| Testing UC 3, 4, 5 | Team | 3/27 | Completed |
| Test Algorithm (New) (add algorithmConfiguration to this test) <https://docs.google.com/document/d/199tVHDlFh2NUmVNBK9zQDoe9VOT0casg_nefR0uYi6s/edit?usp=sharing> | Neha | 3/27 | Completed |
| TestPredicateConverter (Update) | Minghong | 3/27 | Completed |
| GraphSnapshot (New) + GraphUtils (New, util functions for manipulating the graph) | Christina | 3/27 | Completed |
| Look into how to combine cypress and Jest coverage | Julian | 3/27 | Completed |
| Review UC2 and test functionality | Team | 3/27 | Completed |
| Merge Branches into development | Team | 3/27 | Completed |
| User Guide | Christina, Neha | 3/27 | Completed |
| Dev Guide | Julian, Vitesh, Minghong | 3/27 | Completed |
| Update Use Cases | Vitesh | 3/27 | Completed |

**Iteration 3: (Mar. 28 - Apr.23)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Item** | **Owner(s)** | **Due Date** | **Status** |
| UC 4: Node and Edges Attributes | Christina | 3/27 | Completed |
| UC5 Additions/Fixes | Vitesh | 4/9 | Completed |
| Test Algorithm (New) (add algorithmConfiguration to this test) <https://docs.google.com/document/d/199tVHDlFh2NUmVNBK9zQDoe9VOT0casg_nefR0uYi6s/edit?usp=sharing> | Neha | 4/23 | Completed |
| GraphUtils (New, util functions for manipulating the graph) applyPositions, roundPositions, extractPositions, transformPositions | Christina | 4/23 | Completed |
| Graph.js Testing | Vitesh | 4/23 | Completed |
| Thread.js Testing | Julian | 4/23 | Completed |
| Dev Guide | Christina, Julian | 3/27 | Completed |
| User Guide | Christina, Neha | 3/27 | Completed |
| Cytoscape-node-html-label bug | Julian | 4/23 | Completed |
| Review Wake Tech IPR. Everyone needs to ask one question. | Team | 4/4 | Completed |
| Update Poster from Feedback | Christina | 4/4 | Completed |
| UC8 Skip to end + add test to algorithm | Vitesh | 4/15 | Completed |
| Installation Guide | Team | 4/16 | Completed |
| UC6 .sgf files + Testing | Neha, Minghong | 4/15 | Completed |
| Final Sponsor Handoff sides (Tuesday 4/16 during regular time) | Team | 4/15 | Completed |
| Finalize Poster (submit final version via sub system) | Team | 4/15-17 | Completed |
| Update documentation in codebase | Neha | 4/22 | Completed |
| Update System Test Plan | Neha | 4/18 | Completed |
| Final code and documents are due 4/23/24 | Team | 4/22 | Completed |
| Neha FPR Section Update  - Executive Summary - Sponsor Background - Implementation: Project Folder Structure | Neha | 4/22 | Completed |
| Vitesh FPR Section Update - Development Methodology - Overall View - Unit Testing - Acceptance Testing - Other Testing - Future Team Suggestions | Vitesh | 4/22 | Completed |
| Minghong FPR Section Update - Proposed Solution and Project Goals/Benefits - Resources Needed - Risks and Mitigation | Minghong | 4/22 | Completed |
| Julian FPR Section Update - Low level design  - High level design - GUI Design | Julian | 4/22 | Completed |
| Christina FPR Section Update - Problem Description  - System Requirements  - Security Considerations  - Task plan | Christina | 4/22 | Completed |
| Update UI to have “? Help” button | Vitesh | 4/18 | Completed |

# 

# 

# Future Team Suggestions

*Author(s): Vitesh Kambara*

*Reviewer(s)/Editor(s): Team*

* Reduce reliance of useEffect() for data transfer, especially for graph/SetGraph. Some drawbacks of this include the inability to determine when a new graph has loaded. This results in the need to use timeouts to estimate completion of loading to auto-fit. It may be better to use callbacks to resolve this potential issue.
* Reduce reliance of third party extension cy-node-html-label and use Cytoscape’s functionality for labelling of nodes and edges. While this reduces possible customization, it is unneeded and introduces bugs as the extension is unmaintained.
* For the sake of implementing more layered graph algorithms in GalantJS in the future, Dr. Stallmann would like to see algorithms having more control over the layout of the graph and moving nodes in the Graph as opposed to the graph file/input. So it may be necessary to migrate some of these functions responsible for the movement of nodes from Graph.js into Algorithm.js.
* Lots of new keyboard shortcuts were implemented this semester, but a few other handy ones that a future team could implement would be a shortcut in the Algorithm/Graph Editors for Uploading a document (could use key ‘U’), a shortcut in the Algorithm/Graph Editors for clicking the User Guide button to view documentation, and Dr. Stallmann may have a few more shortcuts he would like as he uses this version of the application more and more. Also, there is one shortcut that currently does not work on Google Chrome but works on Firefox and other browsers, in the Algorithm/Graph Editor, clicking ‘S’ should download the current file pulled up in the Editor but this shortcut doesn’t seem to work on Google Chrome as of right now.

**Appendices**

Our below appendices provide supplemental information to different sections of our IPR. The Appendices are organized in alphabetical order with numbers delimiting multiple files under the same appendix (i.e., Appendix E1).

**Appendix A: UC1 (Algorithms Use and View Offline) Removal**

Since we were unable to complete the original UC1 without refactoring most of the code, we dropped the original UC1 with the professor's consent.

# UC 1: Algorithms Use and View Offline

**1.1 Preconditions**

* A researcher has successfully loaded the Galant-JS application including the graph view, graph editor, and algorithm editor.
* A researcher subsequently uses the application offline and has uploaded a Graph in the Graph Editor

**1.2 Main Flow**

* The researcher uploads an algorithm file.
* The researcher loads the algorithm.
* The appropriate Algorithm will be loaded into the Graph Viewer to use and step through even without an Internet connection.

**1.3 Subflows**

**1.4 Alternative Flows**

* [E1] If the algorithm is unable to be loaded then a Timeout Error page will pop up on the screen telling the researcher that the algorithm took too long to load/didn’t load and the application took too long to respond.