**Galant Project**

**Final Interim Project Report**

**Final Requirements, Design,**

**Implementation/Testing & Installation/Delivery**

**Dr. Matthias (Matt) Stallmann**

**CSC 492 Team 25:**

**Samy Bencherif**

**Shengdong Chen**

**Tianxin Jia**

**Ji Li**

**Qihao Lu**

**North Carolina State University**

**Department of Computer Science**

**May 5, 2021**

**Executive Summary**

(Author: Ji Li; Editor: Samy Bencherif)

Galant is a published Java based interactive tool/project for visualization of graph algorithms led by NCSU professor, Dr. Mathias Stallmann, who is interested in combinational optimization with emphasis on dealing with intractability (NP-completeness), either by looking for polynomial special cases or developing heuristics that work well in practice. Since illustrating algorithms is an abstract process without visualization and it’s time consuming for manually doing so, Dr. Stallmann has funded Galant for both educational and research purposes.

Right now the main features of Galant are well implemented including visualization of different algorithms like sorting, minimizing edge crossings, Dijkstra, etc. It also supports several graphical features for graph rendering and editing, such as assigning weight to certain edges/nodes and handling the layered graph case (nodes are assigned to different layers to indicate its hierarchy). However, the current version of Galant has multiple limitations. For the non-layered graph case, each time the display window size changes, users may incur either losing a certain part of the graph or having all graphical elements gathered at one corner. The lack of support of the auto-scaling feature worsens the user experience on galant especially when larger graphs are loaded.

Secondly, during algorithm execution, editing capability, moving nodes in particular, is banned to ensure the correctness of the algorithm. This causes inconvenience for users who wish to explore more possibilities for certain algorithm executing states. Especially under educational situations, educators may need to pause the processing algorithm so they can expand the current state of graph/algorithm into details.

Lastly, since multiple individuals and teams had worked on Galant, the current code base of Galant contains many redundant and unorganized code pieces. For example, there are two java class files sharing the same name, LayeredGraph.java, but having different functionality and dependency. Things like this unintentionally increase the structural complexity of Galant and will confuse future programmers who are planning to keep improving the Galant.

To solve the scaling issue, each graphical element should record two forms of position: the screen displaying position determining where Galant should render certain elements and the logical position used for algorithm calculation. A transform function needs to be implemented to transform those two positions one to the other. By doing so, whenever the display window size changes, Galant will adjust only the screen positions of its elements accordingly to scale. And whenever an algorithm needs data from a graph, calculation will be done by the transform function to provide appropriate logical positions.

To expand the editing capability for layered graph, a screen displaying position will need to be assigned to each layered graph element. This makes those graph elements selectable and thus, movable. Then we add a control flag to tell Galant if a particular element needs reposition or not. With this flag being set to true, Galant will reposition that particular element back to its original place in the next frame.

To simplify and reorganize the code base, we need to integrate the two redundant classes into one and clean out the duplicate code pieces in them. Then we extend the original node class to two subclasses for different types of graph -- layered node for layered graph and non-layered node for general graph. This would increase the overall readability of Galant’s code base, which will facilitate future programmers to both maintain and improve Galant.

**Current State of Project:**

The current version of our Galant project has all of the three features implemented and is stored in the master branch of our github repository. Unit tests and system test documentations are also finished and checked. And lastly, proper documentations are included in our final deliverable, including: the final project report, the installation guide, the user’s guide, and the developer’s guide. It should be convenient for anyone who is interested in our work to follow what we’ve done and how we accomplished our work.

**Project Description**

(Author: Shengdong Chen)

**Sponsor Background**

Matthias F. Stallmann is a professor of Computer Science at NCSU, co-chair, CSC strategic Planning Committee and Assistant Director of Graduate Programs (responsible for advising students in the Master’s program).

The Galant project, one of his current projects, and the one we are working on, is a host of algorithm animation programs. Primarily this project has been designed for classroom use and involves considerable overhead for the animator of the animations (an instructor or developer) --- students are passive observers. A key feature of Galant is that it simplifies the role of animators so that students can create their own animations by adding a few visualization directives to pseudocode-like implementations of algorithms. Galant provides a framework for animations in domains beyond classic graph algorithms; examples include search trees, automata and even sorting.

His primary research interests are in combinatorial optimization with emphasis on dealing with intractability (NP-completeness), either by looking for polynomial special cases or developing heuristics that work well in practice.

Current projects include:

* experimental evaluation of algorithms and heuristics for NP-hard combinatorial optimization problems,
* the theory and practical implications of NP-completeness,
* tools and testbeds for reliable and repeatable computational experiments,
* minimizing edge crossings in embeddings of graphs,
* interactive tools for visualization of algorithms and proof techniques; the most recent (and easily usable for both teaching and research) of these is Graph algorithm animation tool (Galant).
* software that promotes accessibility for blind or visually impaired users

**Problem Statement**

Dr. Stallmann is developing (or has developed) new heuristics for most of the layered graph problems and would like these to be animated as an aid to the research and to produce illustrations for publications and live demos. To that end, the following issues need to be addressed.

The first problem is that for the non-layered graph, graphs do not fit in the window when they are loaded in, and they cannot scale with the window. This problem makes the user unable to clearly observe or easily edit graphs. To be specific, if the original graph is small, it may gather at the corner of the window, or if it is huge, users may lose certain parts of the graph. With this problem being solved, graphs should fit the window when they are initially loaded in and they are scalable with the window, so that users can clearly observe graphs in any window sizes.

The second problem is that during algorithm animations, users are unable to drag nodes(drag nodes without interrupting the algorithms), this is a problem because if the graphs are complicated (lots of crossings), the animations are hard to observe. With this problem being solved, users are able to drag the nodes to focus on specific parts(nodes and edges) of the graphs during animations.

The third problem is that the structure layout of classes is inconsistent, to be specific, graphs and layered-graphs are totally different(constructed in separate classes), and the current project includes 2 classes with the same name “LayeredGraph.java” but different functions. This problem makes the project inconvenient for future developers to understand and to adjust. And the following paragraph indicates the differences between layered graph and graph.

A layered graph in mathematics is a minor variation of a graph. In a layered graph there are still nodes and edges, but in this case the nodes are assigned to bands which correspond to their vertical position called layers. There are some rules about the edge connections that can be made that are similar to that of a tree data structure.

In Galant a layered graph has the same meaning, but it was implemented using a special case and therefore has to be treated differently from the graph data structure.

**Project Goals & Benefits**

The first goal is to make graph scalable with window, the benefit of it is that the user’s time are saved because usually the purpose of enlarging or narrowing the window is to change the view of graph as well, in this case, if the graph scales with the window, the user doesn’t need to adjust the graph after changing the window size.

The second goal is to make the node movable during animation, the benefit of it is that the user can understand the graph pr algorithm better because some graphs are complicated, and the user can move some nodes away from the one she wants to focus.

The third goal is to integrate the layered graph class with general graph class, the benefit of it is that future developers can improve the program easier because the developer is no longer to think of the differences between them, while the program is able to identify them by itself.

**Development Methodology**

we are using an iterative development approach based on the 3 goals: 1. Making the nodes scalable with window 2. Making the nodes movable during animation. 3. Getting rid of layered graph class. In this case, we are going to have 3 iterations to match our goals.

We are divided into 2 subteams to work on iteration 1 & 2 at the same time, we decided to make these 2 iterations parallel because they do not affect each other.Then we merged them together in a week. Finally, all of us will work on iteration 3.

**Challenges**

The most challenging part of the Galant project is to get familiar with existing code. The existing program contains a lot of packages and classes and each class of the most contains more than a thousand lines of code. In addition, the Galant is a project that has been updating for years and multiple teams have worked on it so that the coding styles of the code are completely different. Therefore, it is time consuming and challenging to understand the existing code. We planned to solve this by testing the code piece by piece, reading the documentation, and receiving assistance from Dr. Stallmann.

**Resources Needed**

(Author: Tianxin Jia)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Resource name | Description | Status | Version | Licensing information |
| Apache Ant | Automated software building | obtained | 1.10.10 | The Apache License Version 2.0 |
| Java | Programming language, binary format | obtained | Jdk-15.0.2 | Oracle license. |
| JUnit 4 | Unit testing framework for the Java | obtained | JUnit 4.13.2 | Eclipse Public License - v 1.0 |
| Galant | The project itself | obtained | V6.1.3 | GPLv3.  Stallmann © 2016. |

**Requirements**

(Author: Qihao Lu)

Generally, we have 3 features to implement: First, for the non-layered graph, the user shall be able to scale the window such that the graph fits into the whole window proportionally. Second, for the layered graph, the user shall be able to move the nodes during the algorithm. Third, in the coding phase, the layered graph class should integrate into graph class to build a better structure. Based on the three features, we list 8 requirements below which consist of functional requirements, non-functional requirements, and constraints.

**Functional Requirements:**

1. The system shall scale renderings of non-layered graphs to preserve the same proportions when the user resizes the animation window.
2. The layered graph nodes shall be movable during any layered graph animation.
3. The system shall reset the layered graph nodes position when the layered graph algorithm is stepped forward.
4. The system shall reset the layered graph nodes position when the layered graph algorithm is stepped backward.
5. The system shall reset layered graph nodes position when the layered graph algorithm is canceled.
6. The system shall reset layered graph nodes position when the graph window is resized by users.

**The non-functional requirements:**

1. Layered graphs shall be integrated into general graphs.

**Constraints:**

1. The code formatting should appear K&R (Kernighan & Ritchie indentation Style) for better readability.

**Design**

(Author: Ji Li, Shengdong; Editor: Qihao Lu, Samy Bencherif)

Because we are implementing this project based on our 3 goals, the design of the whole project is separated into 3 designs of features which implement our goals.

**Feature 1 --** Auto-reposition graphical elements with window resize

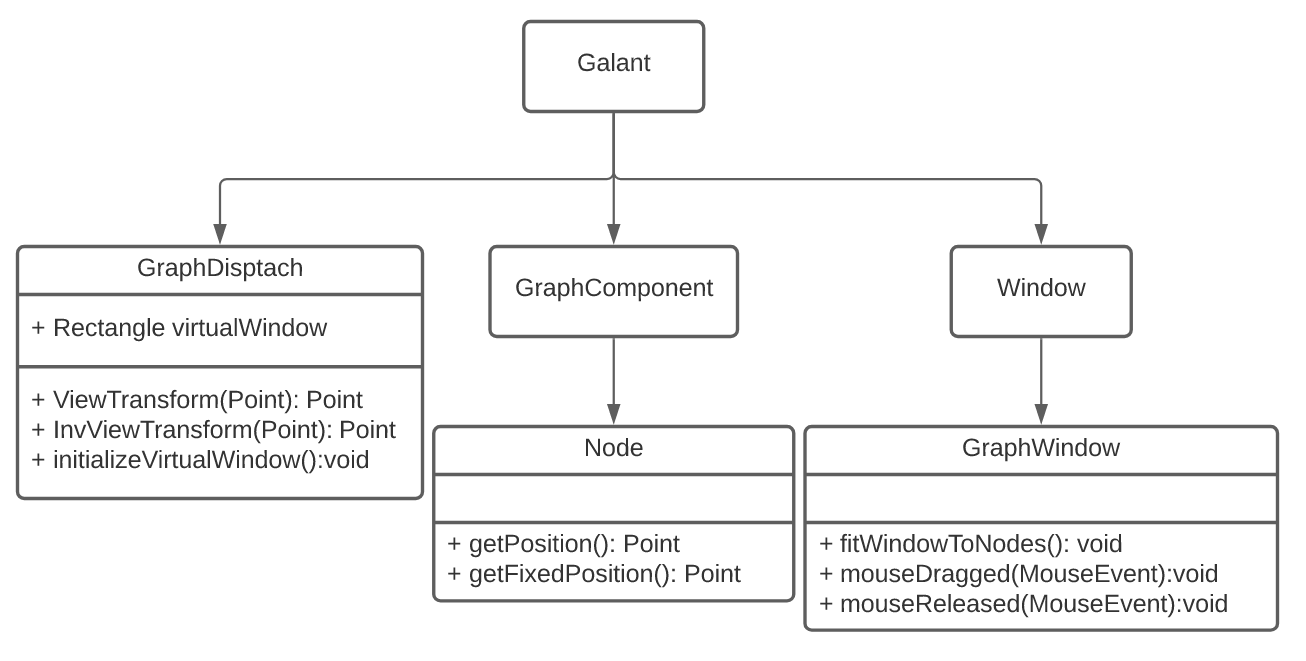


Figure 1.1 Feature 1 Design Diagram

To implement Feature 1.1, we mainly modified 4 classes in the Galant project(shown in Figure 3):

**GraphDispatch**: for managing working graphs in the editor; also used for passing information about window width and height to other classes as appropriate.

**Node**: data structure used to store information for graphical elements -- nodes

**GraphWindow:** the class controls the user interface window.

Imagine you are working on a graph such as the one pictured in Figure 2, and you would like to reduce the window size to make room for another window--or perhaps increase the window size to present an animation fullscreen. The majority of graphs used with this program would either be cropped by the edge of the screen or remain within a small portion of the window. And before scaling, graph should fit the window instead of loading with the original nodes positions from the GraphML file.

**Note**:(A GraphML file contains all the informations of a graph - nodes positions, edges between which nodes, weights, etc)

To solve this problem we added a transformation stage to the graph drawing process, labeled in blue in Figure 1.1. During this process, the initializeVirtualWindow function calculates the proper window size suits for nodes’ positions by traversing all the nodes and finding the proper width and length of window.

Transform function is responsible for converting a node’s position with respect to the GraphML file (known as the logical position) into pixel coordinates suitable to be drawn on screen. Not pictured is the inverse transform function, which is used to contextualize mouse coordinates in relation to the nodes’ original logical positions. To be specific, transform function calculates the position of the node in virtual window coordinates we found in initializeVirtualWindow function, then converts it to display coordinates.

**Note**: Special case LayeredGraph has this feature already, but we needed a general solution.

**Note**: This design was preferred because it will simplify future graphical features such as pan and zoom.

**Feature 2 --** Enhancement of editing capability of Galant

(Note: this feature is specifically designed and implemented for layered graph case)

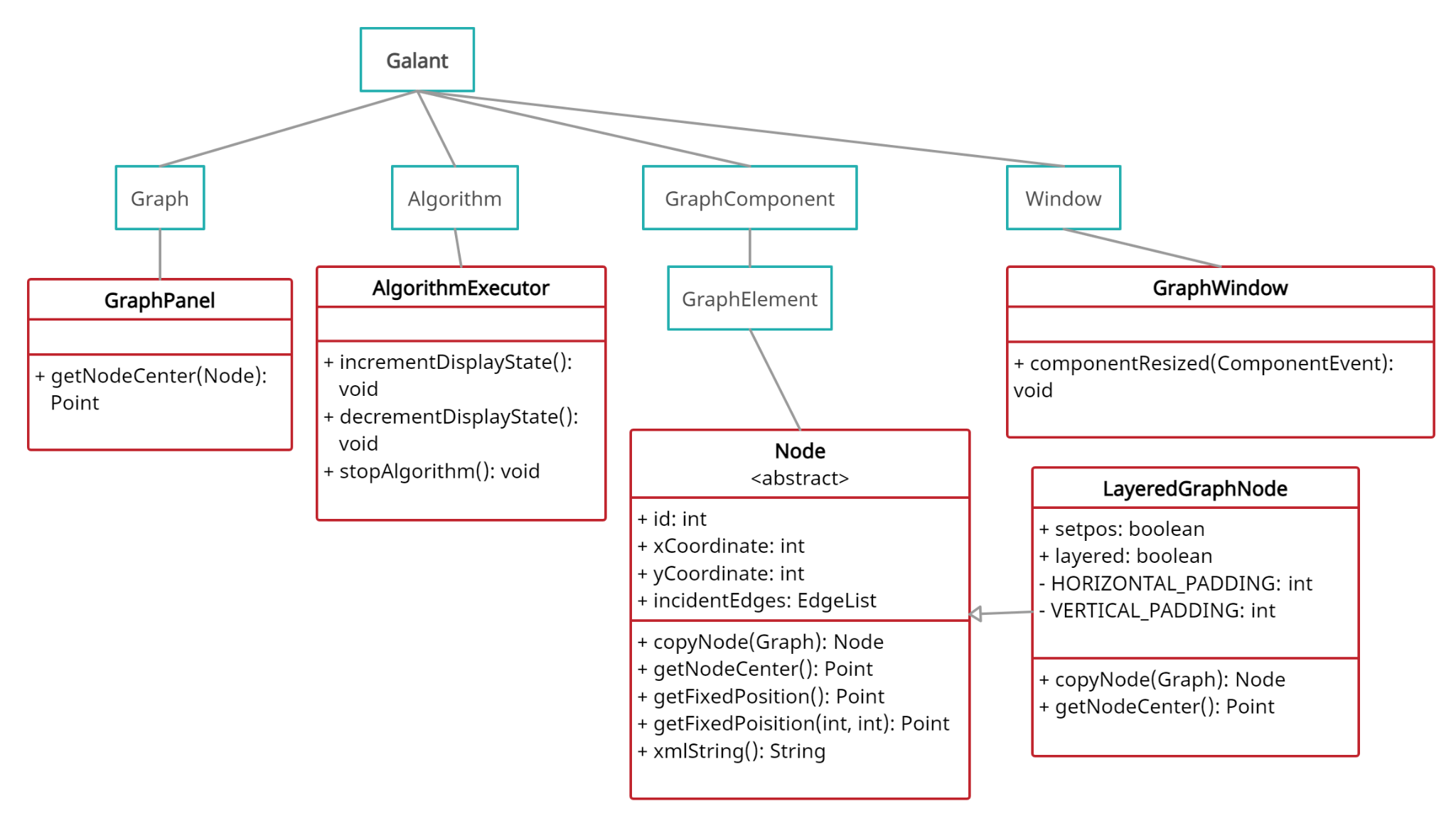


Figure 2.1. Feature 2 Design Diagram

Implementation of feature 2 mainly focuses on modifications of 4 classes (high-lighted in red):

**GraphPanel:** the class controls the graph rendering panel for the user interface.

**AlgorithmExecutor:** the class controls both the execution of algorithms and the switching of graph display state, which indicates which state of the graph should be rendered in the graph display panel.

**Nodes:** data structure used to store information for graphical elements -- nodes.

**GraphWindow:** the class controls the user interface window.

The overall design idea of implementing this feature is: first enable editing, mainly node moving, and then using a new conditional flag to control whether a node needs reposition or not. In details, the control flag called setpos was placed in layeredGraphNodes class as a field, which can be referenced by other classes easily. Corresponding codes that use setpos as a conditional check or control its state (True/False) are placed into proper existing methods, including: getNodeCenter(GraphPanel.java), incrementDisplayState(AlgorithmExecutor), decrementDisplayState(AlgorithmExecutor), stopAlgorithm(AlgorithmExecutor), and componentResized(GraphWindow). More details and the illustration of program flow will be addressed below.



Figure 3.1 Loading and Rendering Graph

Galant uses JPanel from the Java library to support its GUI feature. The JPanel built-in method paintComponent is getting called regularly as the program runs and it re-renders everything each time it gets called. We override this method to call the drawNode method to render each node which uses the getNodeCenter method to extract position information needed for rendering. As shown in figure 3.1, if setpos is **False**, getNodeCenter will be processed to calculate the position for each node, which in other words, reposition all nodes back to their original position. Thus, we use setpos flag to control this function by temporarily banning this feature to allow node editing/movement -- avoid repositioning.



Figure 3.2 Algorithm Executing State Change

As shown in Figure 3.2, when the algorithm is running, the user will be able to click the **Forward, Backward**, and **Terminate** button in the UI window. Those buttons are linking to functionalities that move algorithms into the next/previous/terminating executive state. Once they get clicked, corresponding backstage methods in AlgorithmExecutor will be called and most importantly, the setPos will be set to **False** indicating that a reposition operation needs to be done. Thus, in the next regular display screen refreshing process, the getNodeCenter method will reposition all nodes back to its original position.



Figure 3.3 Node Dragging

Since the node editing/movement is now enabled for all cases and setPos remains **True**, users will be able to move nodes to desired positions and the moved nodes will remain there. (setPos = True bans the reposition feature, as shown in Figure 3.3)



Figure 3.4 Display Window Resizing

For window resizing (Figure 3.4), the componentResized method will get called and it will set setPos to **False**. Similar mechanism will then be processed as illustrated earlier for Figure 3.2 -- node reposition will occur and setPos will then be turned back to **True** again.

**Feature 3 --** Simplify and Reorganize Code Base

The goal for feature 3 is to simplify and re-organize the code base for Galant. There are two major design ideas that we plan to follow:

1. There are two classes sharing the same name, LayeredGraph.java, in the code base, each has unique functionality and is independent of the other one. However, both classes are essential for performing layered graph functionality. We think it would be much easier for future programmers to read and understand Galant if we can merge those two classes into one and give it a proper dependency relationship. As shown in figure 4.1, we’ve merged two LayeredGraph,java classes into one and assigned it as a subclass extending the graph class. All core functionalities of the original two classes have now remained in the new LayeredGraph.java class.

(Note: there are too many algorithm related methods to be shown in the UML diagram below, but those methods can always be viewed in Galant’s code base)

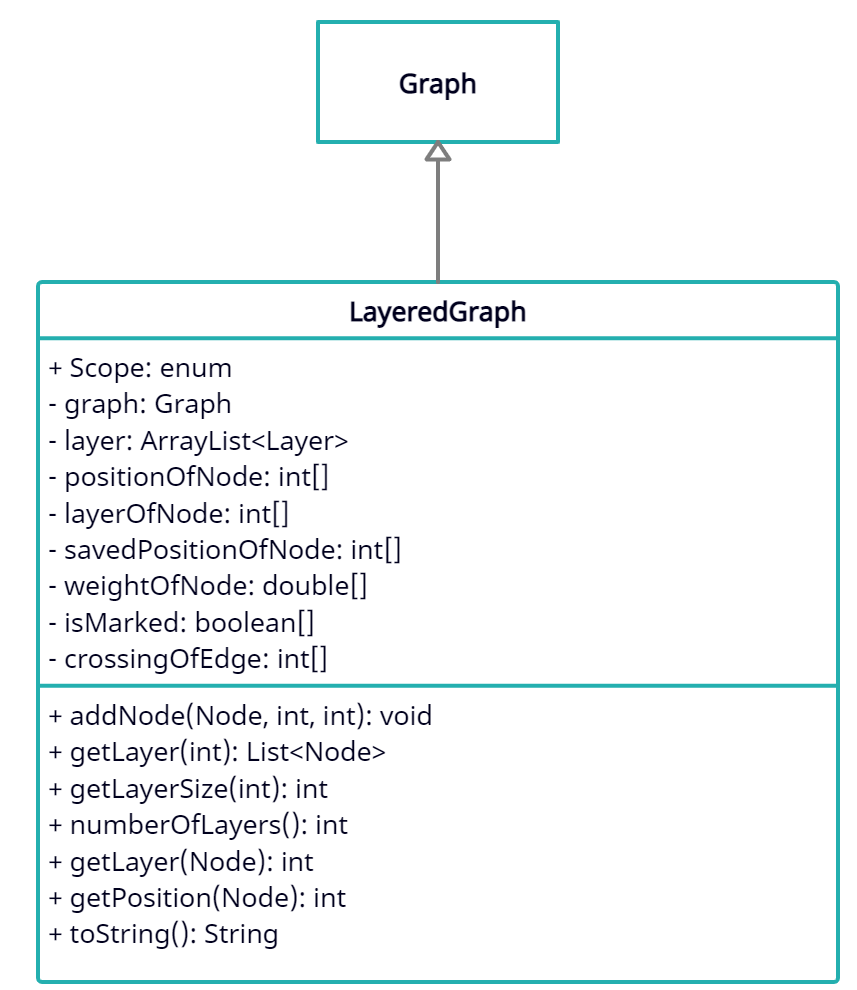


Figure 4.1 UML Diagram for LayeredGraph

1. As mentioned earlier, Galant is mainly dealing with two graph cases, layered graph and non-layered graph. However, there’s only one node class in Galant to deal with both graph cases. This brings too many codes into a single class and a lot of redundant conditional checks need to be done in the case of determining which function a particular method should perform based on the current graph type. We think it could be better organized if we divide the current node class into two subclasses. That way, we can clean out most of the conditional check and increase the readability of our code base by breaking the giant node class into two separate cases. As shown in figure 4.2, we’ve made node class abstract and extended it to two subclasses. The abstract node class contains only commonly shared methods now and each subclass will have more feature based methods in their own places.

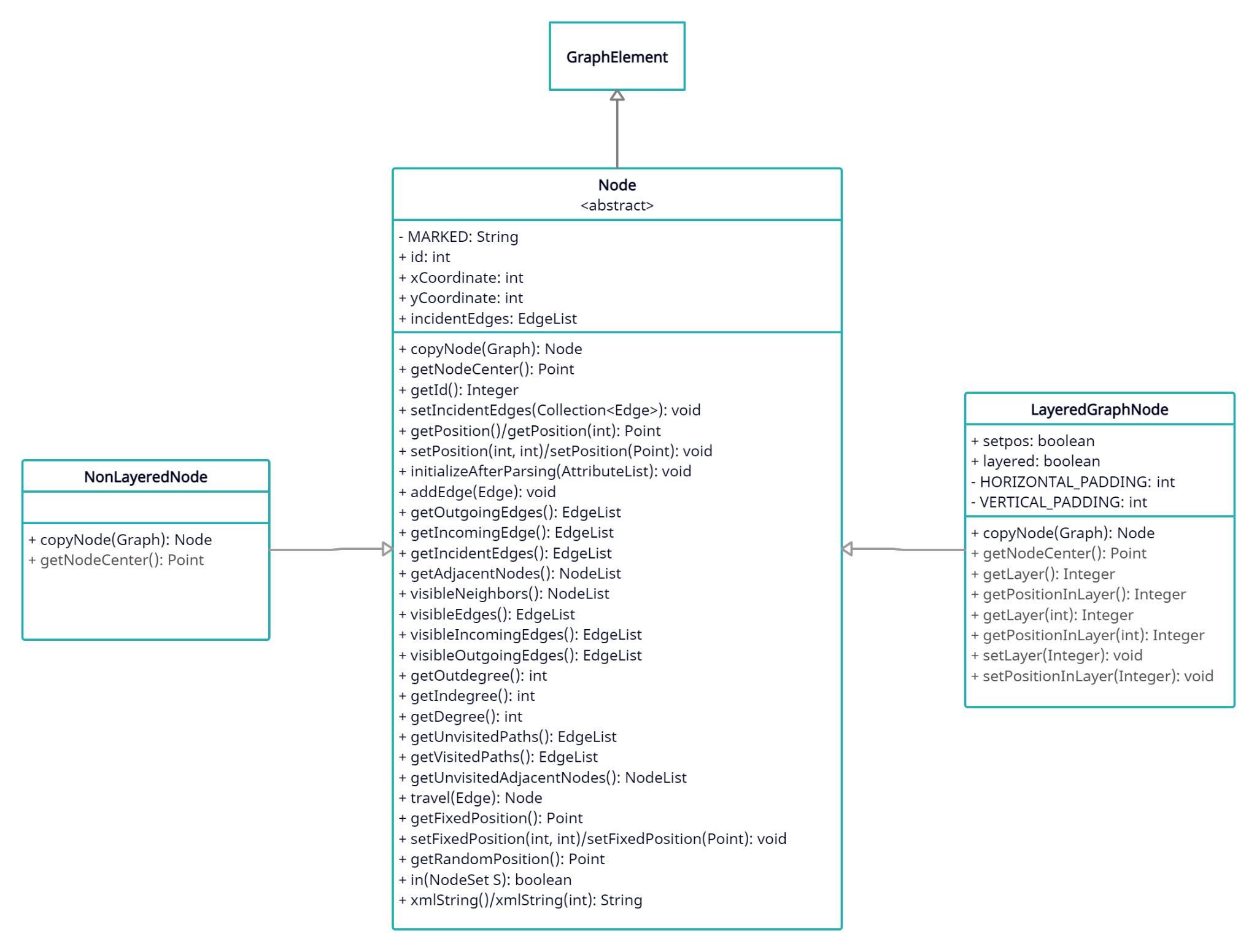


Figure 4.2 UML Diagram for Node

**Implementation**

(Author: Shengdong Chen)

**Iteration definition**

***Iteration 1 (feature 1) - Scaling problem(Samy and Shengdong - Feb. 22nd to March14th)***

This iteration will make the graphs scale with the window.

***Iteration 2 Moving problem(Ji, Tianxin and Qihao - Feb. 22nd to March 14th)***

This iteration will make nodes movable even during animation.

***Iteration 3 Integration of LayeredGraph and Graph (All team members - March 22nd to April 19th)***

This iteration will delete the layered graph class and the general graph class should contain its features.

**Security considerations**

Galant allows users to execute code in order to make animations. The programming is done in a super-set of Java. At the time of writing, we are not sure if there are any mechanisms to prevent malicious code execution. As a result we must advise users to only execute algorithms from trusted sources.

We believe our additions to the Galant codebase do not increase the attack surface available to a malicious animation program. However, Galant user programs (henceforth: algorithms) have access to seemingly all parts of the Galant codebase. This is evident in the algorithms that import the LayeredGraph class. Part of our work is to remove the need for such an import, which would make it easier to ban imports in the future--wherein most security vulnerabilities are accessible.

Our work has been mostly in the domain of user interface, and as a result we did not need access to system critical resources. One exception, however, is the feature that causes the window to enlarge when a node is moved to the edge. In an attack, this could be used to make the user’s desktop difficult or impossible to use. To ameliorate this risk, the window will only be affected by events generated using the mouse cursor and not by algorithms moving nodes.

We are unsure if algorithms can import any Java module or if they are confined to the Galant repository. If algorithms can import any module, they can create arbitrary harm--much like standalone computer programs. If they can only import Galant modules, it is still likely that an exploit related to windowing can be used to slow down or crash the user’s desktop.

**Project Folder structure**

Galant is an existing program with thorough structure, so we are modifying the existing classes to implement our goals instead of creating new ones. The list below displays all the files we edited.

src/edu/ncsu/csc/Galant/

GraphDispatch.java

src/edu/ncsu/csc/Galant/gui/window/

GraphWindow.java

src/edu/ncsu/csc/Galant/algorithm/

AlgorithmExecutor.java

src/edu/ncsu/csc/Galant/algorithm/code/

CodeIntegrator.java

src/edu/ncsu/csc/Galant/graph/parser/

GraphMLParser.java

src/edu/ncsu/csc/Galant/graph/gui/editor/

GTabbedPane.java

src/edu/ncsu/csc/Galant/graph/component

Graph.java Layer.java LayeredGraph.java

Node.java NonLayeredNode.java LayeredGraphNode.java

src/edu/ncsu/csc/Galant/test/

TestViewTransform.java

**Project Configuration**

We do not have any configuration settings in the Galant.

**Test Plan & Results**

(Authors: Ji Li, Samy Bencherif; Editor: Qihao Lu)

Galant is a relatively complex project that has been published for years, and its structure is complicated, which contains many lower coding levels. These lower coding levels refer to any classes that relate to computing algorithms or the representation of graph elements, those are the parts we don’t want to break. Thus, we avoid interfering with those when designing and implementing the new features, which in other words, to contain the already existing codes as much as we can.

Again, enhancing the user experience with better UI and integrating redundant classes (LayeredGraph.java) are the key focuses for this project, which won’t cause executional differences in the lower coding levels but will certainly change the performance in the higher UI level, which will be shown to users. As a conclusion, for the testing strategy, since we will be mainly testing the UI features, we decided to focus on system testing and to use unit testing as a supplement.

On the other hand, the existing JUnit tests in Galant are out of date, but the overall correctness of Galant’s features can be guaranteed by Dr. Stallmann. By Dr. Stallmann’s suggestion we are allowed to omit those old test cases and to create our own tests that are only relevant to the parts of Galant we modified.

In detail, The test plans are broken down into feature based cases and each case will focus on its own domain. Thus, the tests should be specialized and credible enough to guarantee the correctness of each implemented feature.

For feature 2, since all of the implementations are at UI level, system testing should be enough for feature 2 by Dr. Stallmann’s suggestion. Feature 1 is also mostly in UI level, but there is one new calculation function which provides transformation between screen position and logical position of a node, so JUnit test is useful in this case.

After finishing up the 2 features above, we have our feature 3 to test, which is integrating LayeredGraph into the common graph. Our goal is to make sure nothing will break after the integration, and all the functions work correctly, including the 2 new features we implemented. Thus, we first do all the tests in feature 1 and 2 to make sure nothing is wrong. Then, based on our sponsor Dr. Stallmann’s suggestion, there are 4 algorithms in the top priority we want to test, so we make additional tests to ensure the system is in good condition.

Overall, our tests are all passed and no failing cases related to our implemented features.

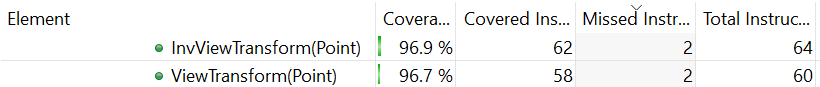


Figure 5. Junit Test Coverage

**Feature 1:**

|  |  |
| --- | --- |
| **TestID** | testVertexGraphLoading |
| **Description** | Preconditions:   * Run Galant  1. **Click** file > open in the text editor window and go to Research > Vertex-Cover to open the *test.graphml* to load the testing Vertex graph. |
| **Expected Result** | A graph with 5 nodes and 5 edges is displayed on the graph window. |
| **Actual Result** | The graph shows up correctly, with 5 edges and 5 nodes. |

|  |  |
| --- | --- |
| **TestID** | testAlgorithmRun |
| **Description** | Preconditions:   * Run Galant * Research > Vertex-Covers > *test.graphml* is loaded  1. **Click** file > open in the text editor window and go to Research > Vertex-Cover to open the *dominance.alg* to load algorithm. 2. **Click** compile and run 3. **Click** forward button until the algorithm finish(the forward button turns gray) |
| **Expected Result** | All the nodes are on the original position, node 1 and 2 turn to gray, and each node has been assigned a weight value. Node 1 gets 4, node 2 gets 1, node 3, 4, and 5 get 0. |
| **Actual Result** | All the nodes are in the original position, node 1 and 2 are gray, and the weight on each node is correct. |

|  |  |
| --- | --- |
| **TestID** | testResizeAlgorithm |
| **Description** | Preconditions:   * Run Galant * Research > Vertex-Covers > *test.graphml* is loaded * Research > Vertex-Covers > *dominance.alg* is loaded  1. **Click** compile and run in the algorithm window 2. **Click** forward button 3. **Click** the edge of the graph window and **Toggle** the graph window edge to reduce the window size by 50%. |
| **Expected Result** | All mathematical objects and labels should be repositioned to stay within the window. The objects should maintain relative positions, and the graphical elements themselves are not expected to scale. The overall graph is reduced by 50%. |
| **Actual Result** | The relative positions are the same for all the nodes and edges, the overall graph is reduced by 50%. |

|  |  |
| --- | --- |
| **TestID** | testDragResizedVertexGraphLeft |
| **Description** | Preconditions:   * Run Galant * Research > Vertex-Covers > *test.graphml* is loaded  1. **Click** the edge of the graph window and **Toggle** the graph window edge to reduce the window size by 50%. 2. **Click** to drag node 1 to the left of node 0. |
| **Expected Result** | Node 1 should be moved to the left of node 0, and all the edges of node 1 should follow node 1. |
| **Actual Result** | Node 1 is on the left of 0, the edges are still on node 1. |

|  |  |
| --- | --- |
| **TestID** | testDragResizedVertexGraphRight |
| **Description** | Preconditions:   * Run Galant * Research > Vertex-Covers > *test.graphml* is loaded  1. **Toggle** the graph window to reduce the window size by 50%. 2. **Click** to drag node 1 to the right of node 0. |
| **Expected Result** | Node 1 should be moved to the left of node 0, and all the edges of node 1 should follow node 1. |
| **Actual Result** | Node 1 is on the left of 0, the edges are still on node 1. |

|  |  |
| --- | --- |
| **TestID** | testDragNodeOutWindow |
| **Description** | Preconditions:   * Run Galant * Research > Vertex-Covers > *test.graphml* is loaded  1. **Click** to drag node 1 to one edge of the window and try to drag out of the window. 2. **Click** on a blank space in the graph window until the node is back in the visible area. |
| **Expected Result** | The node comes back to the window. |
| **Actual Result** | Node 1 is back to the visible area. |

|  |  |
| --- | --- |
| **TestID** | testNonLayerGraphLoading |
| **Description** | Preconditions:   * Run Galant  1. **Click** file > open in the text editor window and go to Research > P-Median to open the *2019-example.graphml* to load the testing non layered graph |
| **Expected Result** | A non layered graph with 16 nodes should be loaded and displayed on the graph window, each of the nodes shall have a weight. |
| **Actual Result** | The graph shows up correctly, with 16 nodes and weights |

|  |  |
| --- | --- |
| **TestID** | testNonLayerAlgorithmLoading |
| **Description** | Preconditions:   * Run Galant * Research > P-Median > *2019-example.graphml* is loaded  1. **Click** file > open in the text editor window and go to Research > P-Median to open the *1-median.alg* to load algorithm. 2. **Click** compile and run |
| **Expected Result** | Forward, backward, and return buttons should appear on the bottom of the animation window. |
| **Actual Result** | All three buttons are listed correctly. |

|  |  |
| --- | --- |
| **TestID** | testNonLayerNodeDragging |
| **Description** | Preconditions:   * Run Galant * Research > P-Median > *2019-example.graphml* is loaded * Research > P-Median > *1-median.alg* is loaded  1. **Click** on node 1and hold the mouse. 2. **Drag** node 1 to the top of node 2 and release the mouse. 3. **Click** on the forward button. |
| **Expected Result** | The node should be able to be dragged, and the node shall keep the dragged position when the algorithm is forwarded. |
| **Actual Result** | Node 1 is moved to the top of node 2, and will keep the dragged position. |

|  |  |
| --- | --- |
| **TestID** | testNonLayerResize |
| **Description** | Preconditions:   * Run Galant * Research > P-Median > *2019-example.graphml* is loaded * Research > P-Median > *1-median.alg* is loaded  1. **Click** on node 1and hold the mouse. 2. **Drag** node 1 to the top of node 2 and release the mouse. 3. **Resize** the animation window |
| **Expected Result** | The node should be able to be dragged, and the node shall **not** return to its original position when the window is resized. |
| **Actual Result** | Node 1 is moved to the top of node 2, and keeps the modified position when resized. |

|  |  |
| --- | --- |
| **TestID** | testNonLayerReturn |
| **Description** | Preconditions:   * Run Galant * Research > P-Median > *2019-example.graphml* is loaded * Research > P-Median > *1-median.alg* is loaded  1. **Click** on the forward button. 2. **Drag** node 1 to the top of node 2 and release the mouse. 3. **Click** on return button(red cross) |
| **Expected Result** | The node should be able to be dragged, and the node shall not return to its original position when the return button is clicked. |
| **Actual Result** | Node 1 is moved to the top of node 2, and keeps the modified position when the return button is clicked. |

|  |  |
| --- | --- |
| **TestID** | testNonLayerNodeOutScreen |
| **Description** | Preconditions:   * Run Galant * Research > P-Median > *2019-example.graphml* is loaded * Research > P-Median > *1-median.alg* is loaded  1. **Click** on the forward button. 2. **Drag** node 1 until node 1 is out of screen and release the mouse. 3. **Resize** the window until node 1 is on the screen. |
| **Expected Result** | The node should be able to be dragged and is out of screen, and the node should display on the modified position when the window is resized. |
| **Actual Result** | Node 1 is moved out of screen, and keeps the modified position when the window is resized. |

**Feature 2:**

|  |  |
| --- | --- |
| **TestID** | testGraphLoading |
| **Description** | Preconditions:   * Run Galant  1. **Click** file > open in the text editor window and go to Research > Layered-Graphs > Graphs to open the *pyramid.graphml* to load the testing layered graph 2. **Check** result |
| **Expected Result** | A layered graph with 4 layers should be loaded, 4 nodes on the first layer, 3 nodes on the second layer, 2 nodes on the third layer, and one on the fourth layer. |
| **Actual Result** | The graph shows up correctly, with 4 layers and 10 nodes. |

|  |  |
| --- | --- |
| **TestID** | testAlgorithmLoading |
| **Description** | Preconditions:   * Run Galant * Research > Layered-Graphs > Graphs > *pyramid.graphml is loaded*  1. **Click** file > open in the text editor window and go to Research > Layered-Graphs > Algorithms to open the *mce.alg* to load algorithm. 2. **Click** compile and run |
| **Expected Result** | Forward, backward, and return buttons should appear on the bottom of the animation window. |
| **Actual Result** | All three buttons are listed correctly. |

|  |  |
| --- | --- |
| **TestID** | testNodeDragging |
| **Description** | Preconditions:   * Run Galant * Research > Layered-Graphs > Graphs > *pyramid.graphml is loaded* * Research > Layered-Graphs > Algorithms > *mce.alg is loaded*  1. **Click** on node 1and hold the mouse. 2. **Drag** node 1 to the top of node 2 and release the mouse. 3. **Click** on the forward button. |
| **Expected Result** | The node should be able to be dragged, and the node should return to its original position when the forward button is clicked. |
| **Actual Result** | Node 1 is moved to the top of node 2, and returns to its original position when forward. |

|  |  |
| --- | --- |
| **TestID** | moveNodeToTheLeft |
| **Description** | Preconditions:   * Run Galant * Research > Layered-Graphs > Graphs > *pyramid.graphml is loaded* * Research > Layered-Graphs > Algorithms > *mce.alg is loaded*  1. **Left click** the node with number 5 2. **Hold** to drag and move the node towards the node with number 4 3. **Release** it goes beyond the node 4 4. **Check** result |
| **Expected Result** | Node 5 should be placed to the left of node 4, and the position of all other nodes should be the same. |
| **Actual Result** | Node 5 is on the left of node 4, other nodes all in the original position. |

|  |  |
| --- | --- |
| **TestID** | moveNodeToTheRight |
| **Description** | Preconditions:   * Run Galant * Research > Layered-Graphs > Graphs > *pyramid.graphml is loaded* * Research > Layered-Graphs > Algorithms > *mce.alg is loaded*  1. **Left click** the node with number 5 2. **Hold** to drag and move the node towards the node with number 6 3. **Release** it goes beyond the node 6 4. **Check** result |
| **Expected Result** | Node 5 should be placed to the right of node 6, other nodes all in the original position. |
| **Actual Result** | Node 5 is on the right of node 6, other nodes all in the original position. |

|  |  |
| --- | --- |
| **TestID** | moveNodeDownwards |
| **Description** | Preconditions:   * Run Galant * Research > Layered-Graphs > Graphs > *pyramid.graphml is loaded* * Research > Layered-Graphs > Algorithms > *mce.alg is loaded*  1. **Left click** the node with number 5 2. **Hold** to drag and move the node towards the node with number 9 3. **Release** it goes beyond the node 9 4. **Check** result |
| **Expected Result** | Node 5 should be placed to the bottom of node 9, other nodes all in the original position. |
| **Actual Result** | Node 5 is on the bottom of node 9, other nodes all in the original position. |

|  |  |
| --- | --- |
| **TestID** | testResize |
| **Description** | Preconditions:   * Run Galant * Research > Layered-Graphs > Graphs > *pyramid.graphml is loaded* * Research > Layered-Graphs > Algorithms > *mce.alg is loaded*   .   1. **Click** on node 1and hold the mouse. 2. **Drag** node 1 to the top of node 2 and release the mouse. 3. **Resize** the animation window |
| **Expected Result** | The node should be able to be dragged, and the node should return to its original position when the window is resized. |
| **Actual Result** | Node 1 is moved to the top of node 2, and returns to its original position when resized. |

|  |  |
| --- | --- |
| **TestID** | testReturn |
| **Description** | Preconditions:   * Run Galant * Research > Layered-Graphs > Graphs > *pyramid.graphml is loaded* * Research > Layered-Graphs > Algorithms > *mce.alg is loaded*   .   1. **Click** on the forward button. 2. **Drag** node 1 to the top of node 2 and release the mouse. 3. **Click** on return button(red cross) |
| **Expected Result** | The node should be able to be dragged, and the node should return to its original position when the return button is clicked. |
| **Actual Result** | Node 1 is moved to the top of node 2, and returns to its original position when the return button is clicked. |

|  |  |
| --- | --- |
| **TestID** | testNodeOutScreen |
| **Description** | Preconditions:   * Run Galant * Research > Layered-Graphs > Graphs > *pyramid.graphml is loaded* * Research > Layered-Graphs > Algorithms > *mce.alg is loaded*  1. **Click** on the forward button. 2. **Drag** node 1 until node 1 is out of screen and release the mouse. 3. **Resize** the window. |
| **Expected Result** | The node should be able to be dragged and is out of screen, and the node should return to its original position when the window is resized. |
| **Actual Result** | Node 1 is moved out of screen, and returns to its original position when the window is resized. |

**Feature 3:**

We want to make sure this function does not break previous functions, so all the previous test cases should be tested again. Then we have additional tests below:

|  |  |
| --- | --- |
| **TestID** | testBayCennterDragging |
| **Description** | Preconditions:   * Run Galant * Research > Layered-Graphs > Graphs > *pyramid.graphml is loaded* * Research > Layered-Graphs > Algorithms > *baycentrer.alg is loaded*  1. **Click** on node 1and hold the mouse. 2. **Drag** node 1 to the top of node 2 and release the mouse. 3. **Resize** the animation window 4. **Click** on the forward button until all the nodes are the same as before running the algorithm. 5. **Click** cancel button. |
| **Expected Result** | The node should be able to be dragged, and the node should return to its original position when the window is resized. And the graph goes back to the original position before the algorithm when the cancel button is clicked. |
| **Actual Result** | Node 1 is moved to the top of node 2, and returns to its original position when resized and when the algorithm is canceled.. |

|  |  |
| --- | --- |
| **TestID** | testModBaryDragging |
| **Description** | Preconditions:   * Run Galant * Research > Layered-Graphs > Graphs > *pyramid.graphml is loaded* * Research > Layered-Graphs > Algorithms > *mod\_bary.alg is loaded*  1. **Click** on node 1and hold the mouse. 2. **Drag** node 1 to the top of node 2 and release the mouse. 3. **Resize** the animation window 4. **Click** on the forward button until all the nodes are the same as before running the algorithm. 5. **Click** cancel button. |
| **Expected Result** | The node should be able to be dragged, and the node should return to its original position when the window is resized. And the graph goes back to the original position before the algorithm when the cancel button is clicked. |
| **Actual Result** | Node 1 is moved to the top of node 2, and returns to its original position when resized and when the algorithm is canceled.. |

|  |  |
| --- | --- |
| **TestID** | testSiftingDragging |
| **Description** | Preconditions:   * Run Galant * Research > Layered-Graphs > Graphs > *pyramid.graphml is loaded* * Research > Layered-Graphs > Algorithms > *sifting.alg is loaded*  1. **Click** on node 1and hold the mouse. 2. **Drag** node 1 to the top of node 2 and release the mouse. 3. **Resize** the animation window 4. **Click** on the forward button until all the nodes are the same as before running the algorithm. 5. **Click** cancel button. |
| **Expected Result** | The node should be able to be dragged, and the node should return to its original position when the window is resized. And the graph goes back to the original position before the algorithm when the cancel button is clicked. |
| **Actual Result** | Node 1 is moved to the top of node 2, and returns to its original position when resized and when the algorithm is canceled.. |

**Task Plan**

(Author: Tianxin Jia)

|  |  |  |  |
| --- | --- | --- | --- |
| **Item** | **Owner(s)** | **Due Date** | **Status** |
| Ground Rule | All | Jan. 23 | Complete |
| 1st Sponsor Meeting | All | Jan. 29 | Complete |
| January Peer Eval | All | Feb. 3 | Complete |
| 2nd Sponsor Meeting | All | Feb. 5 | Complete |
| Iteration plan | All | Feb. 12 | Complete |
| 3rd Sponsor Meeting | All | Feb. 12 | Complete |
| OPR 1 | Samy | Feb. 16 | Complete |
| 4th Sponsor Meeting | All | Feb. 17 | Complete |
| First Draft of IPR | All | Feb. 19 | Complete |
| 5th Sponsor Meeting | All | Feb. 26 | Complete |
| February Peer Eval | All | Mar. 1 | Complete |
| 6th Sponsor Meeting | All | Mar. 5 | Complete |
| OPR 2 | Ji (Intro/High lvl Design Overview/IT II high/low lvl design/Test plan/Challenge)  Tianxin (Proj Recap/Prob Stmt/IT I high/low lvl design/Demo/Contribution/Action Steps) | Mar. 11 | Complete |
| 7th Sponsor Meeting | All | Mar. 12 | Complete |
| Iteration 1: Requirement #1 Auto-scaling graphical elements | Samy (making the graph scalable, fixed mouse interruption, border fixed )  Shengdong (Junit test, system test) | Mar. 14 | Complete |
| Iteration 2: Requirement #2 Allowing editing during animation/algorithm execution | Tianxin (allow node moving and reset position)  Ji (reset position whenever algorithm moves to its next executive state)  Qihao (Check if algorithm move node/test) | Mar. 14 | Complete |
| Iteration 2.5  integrate changes from both teams and test | All | Mar. 19 | Complete |
| 8th Sponsor Meeting | All | Mar. 19 | Complete |
| Final IPR | All | Mar. 19 | Complete |
| 9th Sponsor Meeting | All | Mar. 26 | Complete |
| 10th Sponsor Meeting | Qihao, Tianxin, Shengdong | Apr. 2 | Complete |
| Draft of Poster | All | Apr. 2 | Complete |
| March Peer Eval | All | Apr. 2 | Complete |
| OPR 3 | Qihao(Proj Recap/Iteration sum/IT3 Intro/IT2 Recap/Contribution/Action Steps)  Shengdong(Intro/Demo/IT1 Recap/Testing/Challenge) | Apr. 8 | Complete |
| 11th Sponsor Meeting | Ji, Qihao, Tianxin, Shengdong | Apr. 9 | Complete |
| 12th Sponsor Meeting | Ji, Qihao, Tianxin, Shengdong | Apr. 16 | Complete |
| Iteration 3:  Requirement #3 Integration of layeredGraph class | Tianxin (Move LayeredGraph into Graph Component package. Add NonlayeredNode & Layered Node that inherit Node. Add getNodeCenter)  Ji (Integrate redundant layer inner class in LayeredGraph.java to layer.java. Reorganize nonLayeredNode and LayeredNode classes. Resolve bugs/conflicts caused by refactoring codes) | Apr. 19 | Complete |
| Poster Day Project Title & Short Description | Ji, Qihao | Apr. 23 | Complete |
| 13th Sponsor Meeting | All | Apr. 23 | Complete |
| Finalize Poster Day Posters | All | Apr. 29 | Complete |
| Poster Day | Shengdong(Background/Terminology/Demo for IT I)  Tianxin(Problem Statement)  Samy(Proposed Solution)  Qihao(Test/Challenge/Demo for IT II)  Ji(Design & Implementation) | Apr. 30 | Complete |
| Sponsor Final Presentation | All | May. 7 | Complete |
| Installation Manual | Qihao | May. 7 | Complete |
| User's Guide | Ji, Tianxin | May. 7 | Complete |
| Developer's Guide  Open | Samy, Qihao | May. 7 | Complete |
| Written Final Reports | All | May. 7 | Complete |

**Team Contact Information**

Ji Li, [paulli990202@gmail.com](mailto:paulli990202@gmail.com) ([jli77@ncsu.edu](mailto:jli77@ncsu.edu)), +1 (919) 308-2411 ( or +86 138-8849-8153)

Qihao Lu, [kocoru8@gmail.com](mailto:kocoru8@gmail.com), [qlu4@ncsu.edu](mailto:qlu4@ncsu.edu), +1 (919) 884-8884

Samy Bencherif, [sbenche2@ncsu.edu](mailto:sbenche2@ncsu.edu) / [SamyBencherif73@gmail.com](mailto:SamyBencherif73@gmail.com) / [samy@programmer.net](mailto:samy@programmer.net) , +1 (336) 473-7303

Shengdong Chen, [schen42@ncsu.edu](mailto:schen42@ncsu.edu)/ [shengdongc0@gmail.com](mailto:shengdongc0@gmail.com) , +1 (919) 749-9393

Tianxin Jia, [tjia2@ncsu.edu](mailto:tjia2@ncsu.edu) / [samjia971@gmail.com](mailto:samjia971@gmail.com) / [sam\_jia97@126.com](mailto:sam_jia97@126.com), +1 (919) 840-8490/+86 180-9897-8848

**Suggestions for Future Teams**

Read the design document and Developer guide first. Start with Galant.java, see how the project starts and runs, then check which file/class you should edit. Use Visual Studio Code to globally search through the codebase. It is commented such that this is a fast way to discover what files and sections you need.

Ask Dr. Stallmann technical and design questions, for example:

1. For most clients it’s about *what* to do, but with Dr. S you should probably also ask *how* to do it. He’s very familiar with the program, having written most of it.
2. Do not wait for sponsor meetings to ask questions. Post them in Slack/Email as well.

There are a lot of packages and classes, but don’t be afraid, usually you don’t need to edit all of them together. If you are working in the gui editor, check the following files: GraphPanel.java, GraphDispatch.java, Graph.java, Node.java.