# Galant 2021 Developer's Guide

## About this Document

Since Galant is such a large project, we figured it would be useful to create documentation specifically about the parts of Galant we learned about during our semester developing upon it. Generally, this document contains information we learned throughout that would have been useful at the beginning. Additionally, we will try to include some insights about future plans for Galant and some very general advice about the codebase.

## Table of Contents

* [Introduction](Galant2021.md#Introduction)
* [Getting Started](Galant2021.md#Getting-Started)
* [Custom IDE](Galant2021.md#Custom-IDE)
* [Our Work](Galant2021.md#Our-Work)
* [Auto-Scale Node Positions](Galant2021.md#Auto-Scale-Node-Positions)
* [Allow Moving Layered Node during Animation/Edit Mode](Galant2021.md#allow-moving-layered-node-during-animationedit-mode)
* [Virtual Window](Galant2021.md#Virtual-Window)
* [Combining Layered Graph into existing Graph](Galant2021.md#Combining-Layered-Graph-into-existing-Graph)
* [Known Bugs](Galant2021.md#Known-Bugs)
* [Branches](Galant2021.md#Branches)
* [Futures](Galant2021.md#Futures)
* [Planning Your Implementation & Java Resources](Galant2021.md#planning-your-implementation--java-resources)
* [Subsystems](Galant2021.md#subsystems)

## Introduction

Galant is an academic and research graph visualization tool. It features a GUI editor and code editors. The GUI editor is used to display and edit graphs as well as display animations (which can be paged through like a presentation). The code editors can be used to write animations in Java or directly modify graphs in GraphML format. The tool also supports LayeredGraphs. If you are familiar with Graphs (collection of nodes and edges) then the only thing you need to know about LayeredGraphs is that each node is a member of a layer. In Galant layers are represented by vertical position, which means two nodes will have the same y-position iff they are on the same layer. In general, it seems like the highest priority for new features in Galant are ones that simplify viewing/exploring graphs as well as refactoring and simplifying the code.

## Getting Started

Galant can be made into a .jar file, which is a Java executable.

If you are willing to use Eclipse IDE it contains everything needed to build and run tests. Otherwise see the [Custom IDE](Custom-IDE) section.

## Custom IDE

This section is completely optional. If you really do not want to use Eclipse, this section will inform you how to develop Galant without it. The program [Apache Ant](https://ant.apache.org/manual/install.html) is the only program actually required to build Galant. It can be installed on Mac easily using HomeBrew. On Linux and Windows simply follow the link and extract the program into a location in your PATH. For text editing, your choice of editor will work. However, Visual Studio Code is recommended due to its Java language support (plug-in) and exceptional project searching feature. The searching feature in particular was very useful for me when learning this codebase.

I was the only person on my team who used a custom IDE (except for Dr. Stallmann) but neither of us were involved in JUnit Testing, so no one is sure how to do this without Eclipse. If you find out please include it your team's Developer's Guide.

Once you have download the program Ant and copied into /usr/local/bin (linux) or a location in your PATH env variable (windows), simply run the command ant from the folder containing Galant to build and run. You can run it again without building by running the jar directly.

## Our Work

### Auto-Scale Node Positions

This feature makes sure that the graph still fills the window whenever it is resized. We introduced the ViewTransform to the project. Right before rendering each node, it computes where the node should appear on screen. This computation is done using (1) the node's position, (2) the VirtualWindow bounds, and (3) the window size.

### Virtual Window

We also introduced the VirtualWindow to the project during this iteration. This structure is represented in the nodes' coordinate space (known as logical position). The VirtualWindow is roughly equivalent to the bounding box of (nodes\_min\_x, nodes\_min\_y) and (nodes\_max\_x, nodes\_max\_y). It is used to determine the right transformation to use in order to show all nodes within the window. Once we determine the virtual window bounds we can easily find the transformation that translate and scales it to the actual window. That same transform is then applied to each node at render time.

### Allow Moving Layered Node during Animation/Edit Mode

This change involved modifying the underlying data structure more than the previous ones did. It is important to allow movement of the layered node because it allows researchers and students to explore different possibilities while running algorithms. For example, some algorithms can move nodes around, and one common purpose is to minimize the number of times edges intersect each other. A researcher may want to move nodes in the middle of this algorithm to see if they can outperform the algorithm manually--or perhaps a student would move such a node in order to convince themselves this step is actually optimal.

Just as important as allowing the change in the first place, is undoing it whenever the user steps forward. While it may be interesting to run the algorithm from whatever state the user left it in, this functionality does not currently exist. Plus it may prove to be nonsensical in some instances, or make the program harder to use. Instead, we keep track of the original positions and reset to those positions before applying another algorithm step.

### Combining Layered Graph into existing Graph

This was our main refactoring task for our run. This was a great opportunity for automatic testing, since our objective is to not make any semantic changes. We completed this task by creating an abstract class called "Graph" and two subclasses LayeredGraph and NonLayeredGraph. Please understand that "Non-Layered Graph" is not a type of graph in any mathematical sense. That is simply our umbrella term for all types of graphs that are not Layered Graphs. It includes Directed Graphs, Weighted Graphs, etc. Simply our team and the ones before us, did not need to make a programmatic distinction for any type of graph except Layered.

## Known Bugs

1. Creating nodes

When the user creates a new graph and creates new node on the edge of the graph window, the node will not display on the window, and if the user try multiple times or scale the window, Galant will crash. This issue may come from the virtual window we used in feature 1, and only happens when creating new nodes in a new graph. This issue is not happening on any pre-created graph.

This is likely because we often performed our manual tests on precreated graphs to save time. If you find yourselves doing the same thing, we advise you to test edge cases even if they do not seem relevent to your changes. You don't have to do it everytime, but you should do it sometimes (especially towards the end).

If the virtual window is a source of problems in the future, but you would still like to leverage some of our team's work in your revision, please take note that vwin (virtual window) is only one part of the scalable-nonlayered iteration. The other component is called viewTransform. Whenever it is time to draw a node on screen the render location is the node's logical position with the viewTransform applied. You can use the viewTransform to displace or "scale" all nodes at the same time. The virtual window is used to determine what extent these transformations should be made in order to match the resized window. In the next section we will talk about ways you can phase out virtual window or use it your own implementations.

1. Import field in all layered graph

Since we have integrated the layered graph with the graph class, we do not have to import layeredgraph class in every layered graph algorithm. Thus, each time before we run layered graph algorithm, we have to delete the import field in the algorithm.

## Branches

This repository has a handful of branches--this is partially a result of how we are encouraged to work within SDC. Now, especially considering that Dr. Stallmann has his own git repository where he keeps official versions of Gallant, at the time of writing none of the extra branches are still needed in the student repository. Please make sure you are in a student repository and delete all branches except for master and development. Please note that master has more commits than development in 2021, because we never used the development branch. Instead we created our own branch that we used for development called "Iteartion\_III\_LayeredGraph\_Intergration". This branch was merged directly to master. Please merge master into development before you proceed and do all of your semester work in development or branches that you create.

## Futures

It is likely that Dr. Stallmann will be thinking about some of these features in the future.

#### 1. Panning

Panning is when the user is able to move the entire view of graph. This is helpful when viewing and editing.

Luckily this feature will be much easier to implement using the VirtualWindow. Simply adjusting the x and y coordinates of VirtualWindow will causing panning. The next consideration is user input. For that we recommend using JPanel mouse events.

#### 2. Zoom

There are two sides to this feature. Firstly, zoom allows the user to focus on a specific region of a graph or show the entire graph at once. With this feature implemented we no longer need to ensure all nodes are always on screen--which is the source of some of our known bugs. Instead, the user can intuitively manage which nodes need to be on screen. Together with panning, this feature will make Galant's editor have similar navigation to Google Maps or Photoshop.

This feature can also leverage the virtual window, in this case the width and height values should be either decreased to zoom in or increased to zoom out. To keep the view centered you will also want to adjust x and y values.

### Planning Your Implementation & Java Resources

Some of the contents of the Galant codebase are specific to the codebase such as GraphWindow and GEditorPane, but even those structures are usually build off of Java content. To understand Galant as best you can, you will need a mixture of searching around the code & docs, talking to Dr. Stallman, and using Java resources.

Galant is not really organized into subsystems, but at this scale your implementations will likely only touch one or two. For example if you were tasked to implement a feature that allows you to drag images into the graph editor and have them display next to any active graph you would have interact with mouse input and rendering systems. Firstly, you should ask youself how Galant already works with mouse input and rendering. Once you have some clarity on the matter, you should consider whether or not there is a perfectly sensible and straightforward way to implement your feature using Galant's existing functionality. Make sure to look thoroughly as Galant has a lot of systems and features already. If you don't find a good match, you will need to consult the Java manual, and use it to extend Galant's base functionality in order to make sense for your implementation.

Keep in mind, Galant is entirely built off of standard Java objects. The UI uses [JPanel](https://docs.oracle.com/javase/7/docs/api/javax/swing/JPanel.html). So the Javadocs will give you all the relevant information concerning mouse movement, rendering, concurrency, filesystem access, or anything else you might need.

### Subsystems

Like I said in the previous section, Galant is not really organized in subsystems, but if I had to name some anyway, here they are. There is definitely some overlap, but these categories roughly describe the different parts of the program. Likely each of your iterations will touch on about two of these subsystem (and maybe become one in the future).

* Code Execution
* Animation Playback
* Graph Editing
* Text Editing
* Mouse Events
* Virtual Window / View Transform
* UI Buttons
* Graph Drawing
* Intelligent Rearrange
* Filesystem Access

Remember this is not official terminology, yet. I am making it up right now, but I think it will help you to plan out your project. If you want to talk to Dr. Stallmann about subsystems, please clarify what you are talking about (unless your team is from 2023 or later).