# Documentation

Preamble

I moved the graph input format subsection from the technical section to the user guide because the user needs to know what kinds of graphs they can load.

Class diagrams in this Documentation will not include most generated methods in an efford to keep the diagrams in a readable size.

User Guide

Startup

After starting the application for the first time you should see this window. If you are running this program on Linux or Mac the title bar of the window might look different but that does not matter.
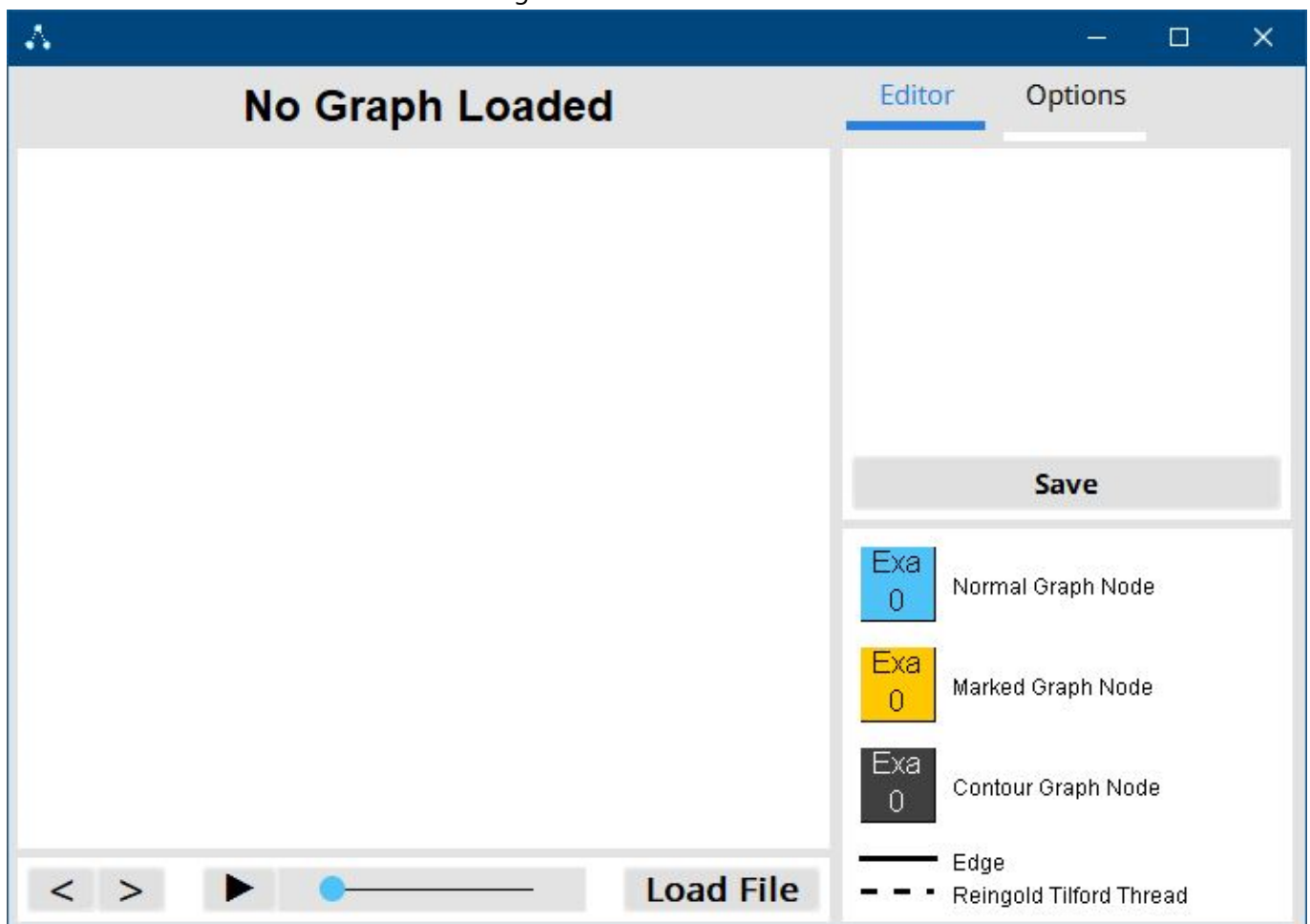
The first thing you want to do is to press the `Load File` button, this will load up the standart java swing file selector. With that you should select a text file that contains a valid graph. Do not worry though the program will remember the last file you selected so you won't have to go though all of your folder twice. Alternatively you can also type a valid graph into the editor and hit `CTRL + S` or `Save`.

Valid File Formats

A text file containing a valid graph only contains lines from a .elkt graph that do not contain { } blocks or simplified elkt edge definitions.

```
n1 -> n2
n1 -> n3

n2 -> n4
n2 -> n5

n3 -> n6
n3 -> n7
```

This is enough to describe a full binary tree with a depth of 2 in simplified elkt. Any node that appears in an edge is automatically added to the graph. However as stated above .elkt files in this format work too:

```
algorithm: Trees

layoutAlgorithm: 1

node n1
node n2
node n3
node n4
node n6
node n7
node n8
node n9
node n10
node n11
node n12

edge n1 -> n2
edge n1 -> n3

edge n2 -> n4

edge n3 -> n6
edge n3 -> n7

edge n4 -> n8

edge n6 -> n9
edge n6 -> n10

edge n7 -> n11
edge n7 -> n12
```

And combinations of the two:
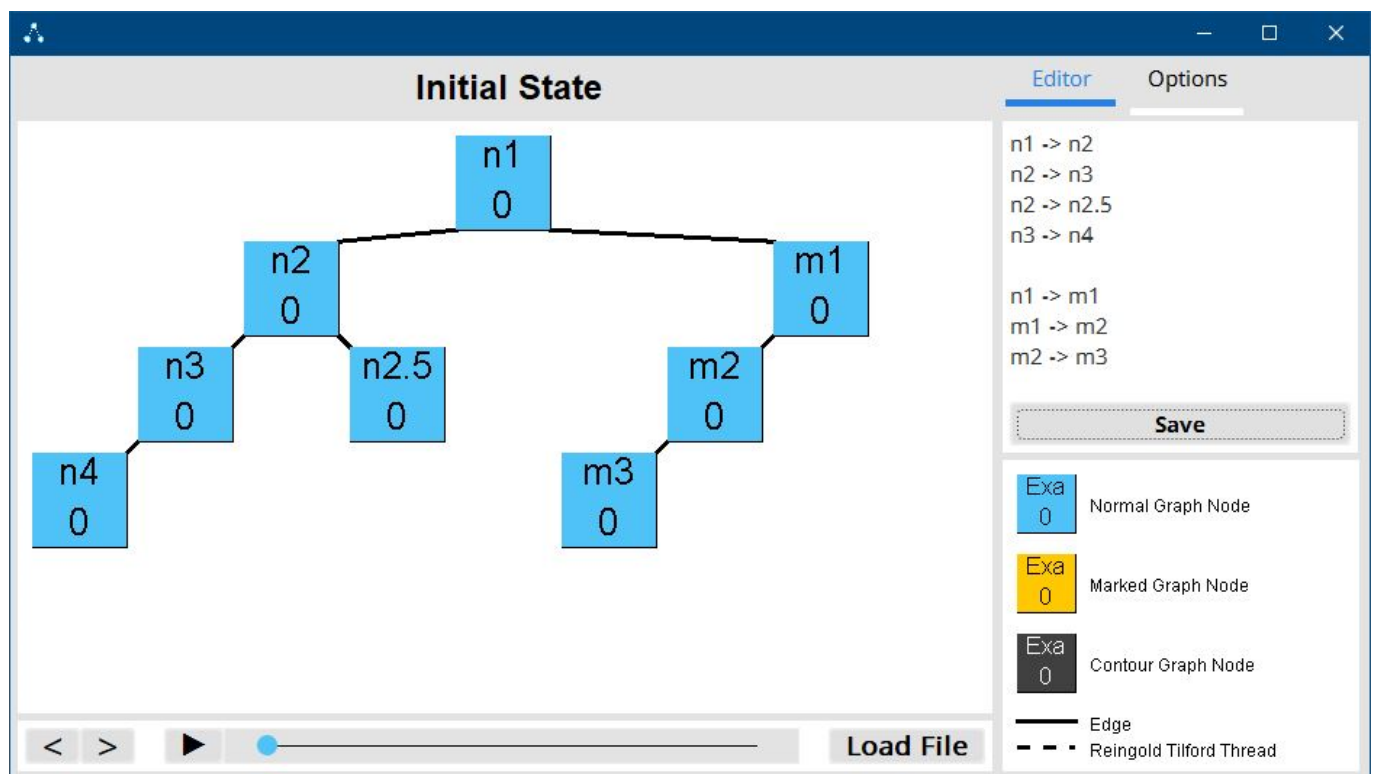
```
algorithm: Trees

layoutAlgorithm: 1

node n1
node n2

edge n1 -> n2
edge n1 -> n3

n2 -> n4
```
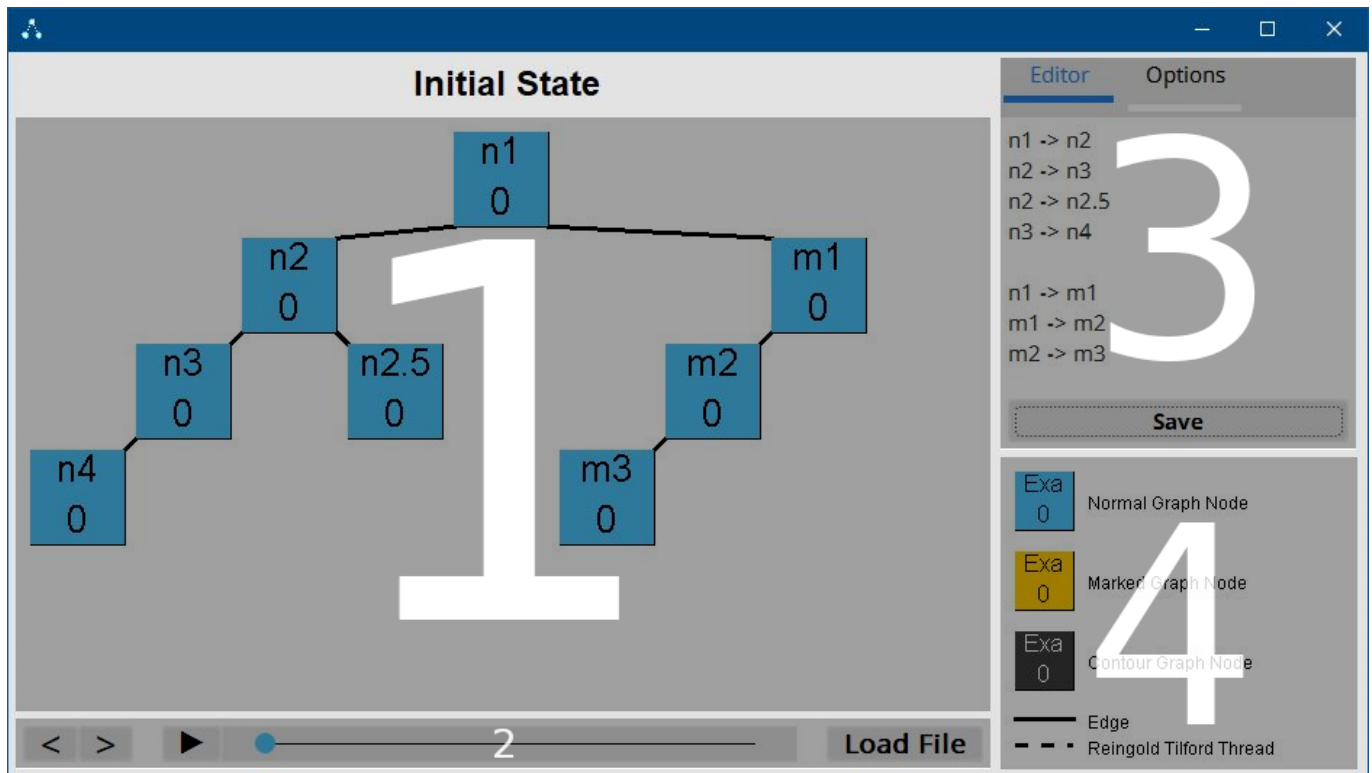
UI

After you loaded a graph the window should look something like this:

As you can see the UI is seperated into 4 main panes:

Pane 1

The first pane is the drawPane, surprisingly the graph is drawn here. More interestingly however is the stateLabel above it that displays information about the current State. Remember to keep an eye on it while stepping through the algorithm states using the UI elements in pane 2.

Pane 2

The second pane contains 3 groups of UI elements that are seperated by empty spaces.
To the very left we got arrow buttons that allow us to step though the animation states frame by frame in each direction. They act like a ring buffer so trying to move backwards on the initial frame will result in landing in the last frame.
In the middle there are a play/pause button and a slider that is misused as a progress bar for the animation. It works very similar to YouTube video bar.
To the right there is the load button that we already used to load this graph.

Pane 3

The Editor

This pane contains tabs, the default shown tab is the editor tab which contains a text editor that shows the source code of the currently loaded graph. Hitting CTRL + S while the tab is focused or the Save button will save the changes you made in that editor to the loaded text file and reload it. Don't move or delete a text file that is currently loaded.

Also noteworthy is a simple syntax highlighter that highlights the currently marked node and contour.
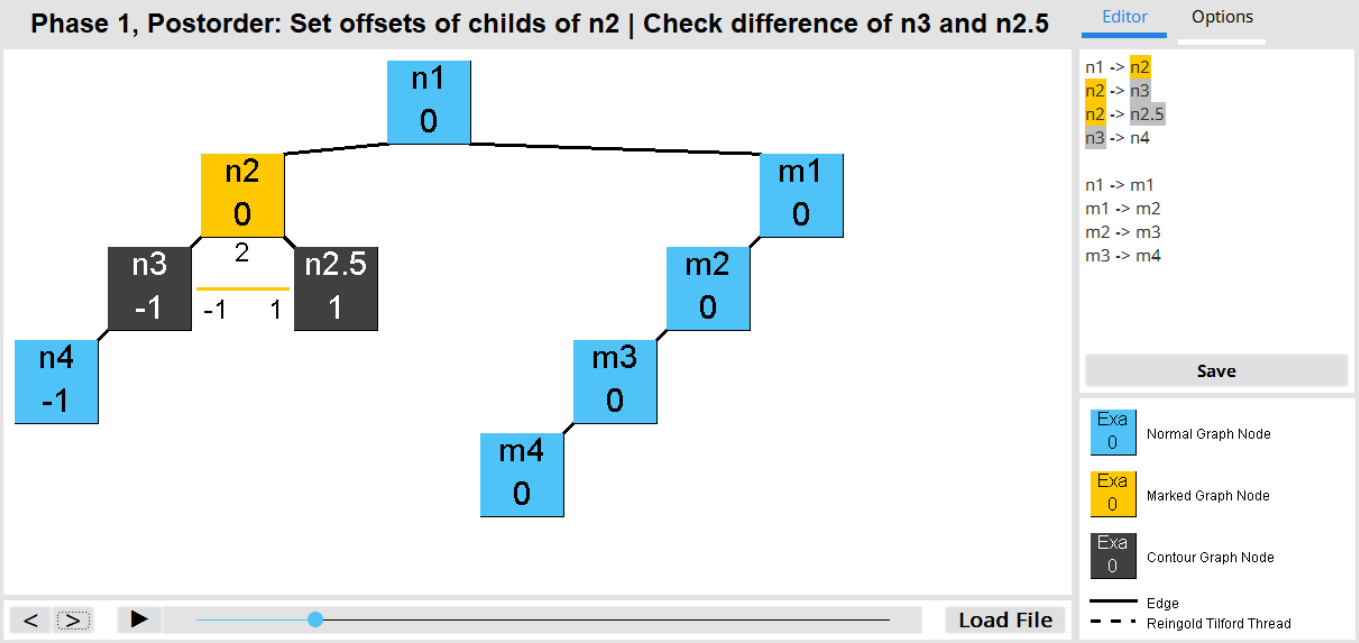
The Options Menu
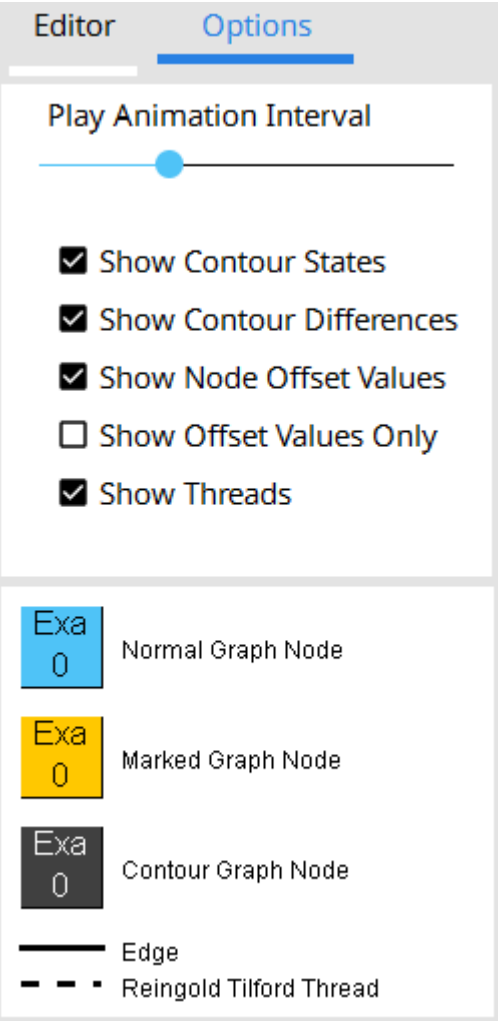
The options menu currently contains an animation interval slider and 4 level of detail options.

The first one shows all animations states that show how the contour was calculated. The second option shows all states that show the distance check on each contour layer.
The third option shows the xOffset values that the algorithm saves for each node. They are usually displayd

below the nodes name. The fourth only shows the offset value for each node. The fifth option hides threads which are displayed as dashed lines.

Some of these options disable other options. Disabling `Show Node Offset Value` and enabling `Show Offset Values Only` doesn't make any sense. Therefore some options disable others.
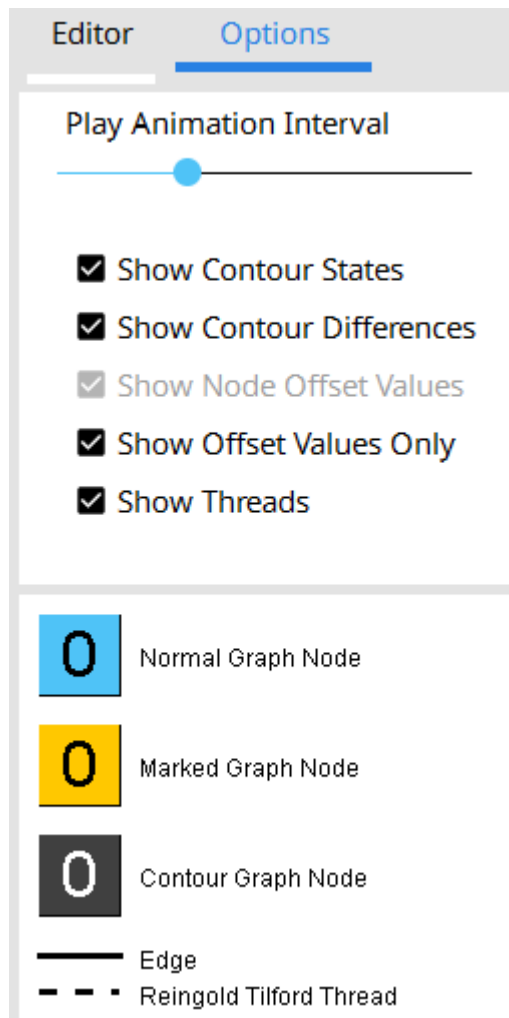
The legend also shows changes in the options.

Pane 4

The fourth pane contains a legend for the graph drawing in the first pane. If this UI element does it's job it should be self explanatory.

The Software behind the UI

Dependencies

The project is written using Java 1.8 and Maven. I imported and used the Maven packages `org.eclipse.elk.graph`, `org.eclipse.elk.alg.common` and `material-ui-swing`. The latter one contains a Java Swing Look and Feel that was inspired by Googles material design.
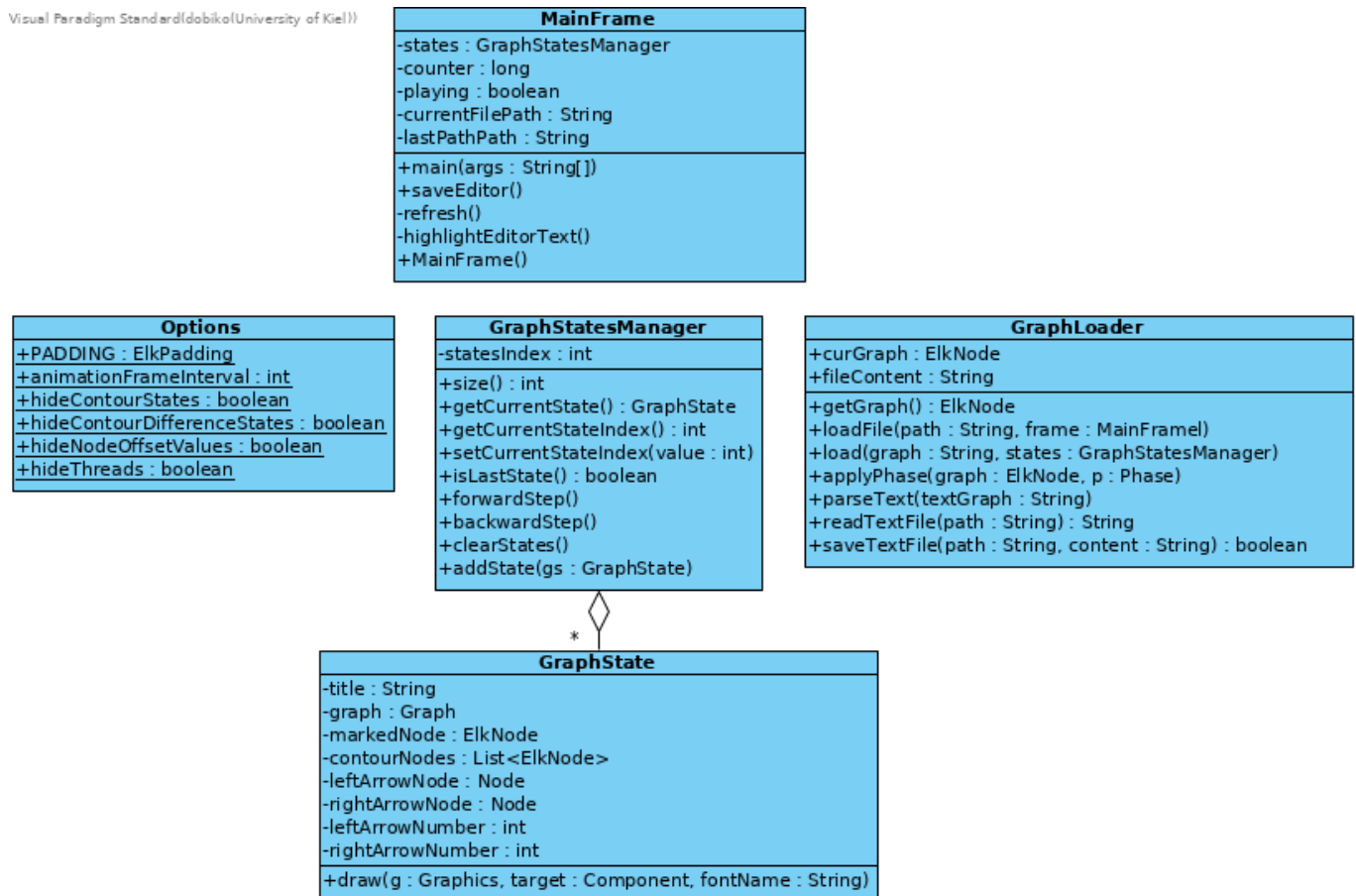
Program Architecture

The code of this project is split into 3 packages:

- graph.drawing.RTProject
  - Contains classes that are specific to this project
- helper
  - Contains classes that support other calculations, most of the classes in this package have been imported from the last assignment
- phases
  - Contains some layout phase classes from the last assignemnts, most notably a modified version of the RT layout phase that

graph.drawing.RTProject

Visual Paradigm Standard(dobiko(University of Kiel))

MainFrame
-states : GraphStatesManager
-counter : long
-playing : boolean
-currentFilePath : String
-lastPathPath : String

+main(args : String[])
+saveEditor()
-refresh()
-highlightEditorText()
+MainFrame()

Options
+PADDING : ElkPadding
+animationFrameInterval : int
+hideContourStates : boolean
+hideContourDifferenceStates : boolean
+hideNodeOffsetValues : boolean
+hideThreads : boolean

GraphStatesManager
-statesIndex : int

+size() : int
+getCurrentState() : GraphState
+getCurrentStateIndex() : int
+setCurrentStateIndex(value : int)
+isLastState() : boolean
+forwardStep()
+backwardStep()
+clearStates()
+addState(gs : GraphState)

GraphLoader
+curGraph : ElkNode
+fileContent : String

+getGraph() : ElkNode
+loadFile(path : String, frame : MainFramel)
+load(graph : String, states : GraphStatesManager)
+applyPhase(graph : ElkNode, p : Phase)
+parseText(textGraph : String)
+readTextFile(path : String) : String
+saveTextFile(path : String, content : String) : boolean

*

GraphState
-title : String
-graph : Graph
-markedNode : ElkNode
-contourNodes : List<ElkNode>
-leftArrowNode : Node
-rightArrowNode : Node
-leftArrowNumber : int
-rightArrowNumber : int

+draw(g : Graphics, target : Component, fontName : String)

MainFrame

This class was with the exception of the code inside some events generated using the WindowBuilder Plugin and its attributes are the GUI elements shown in the last chapter.

GraphStatesManager

This class manages a list of GraphStates and the index of the currently displayed one.

GraphState

This class holds all the information nessecary to visualize a state in the RT algorithm. The left and right arrow attributes may be confusing, they contain all the nessecary information to draw the line that symbolises the check of the contour difference.

The class also contains the draw method which draws this state to a target component. I filtered the numerous constructors and the arguments for the draw methods to keep the diagram readable.
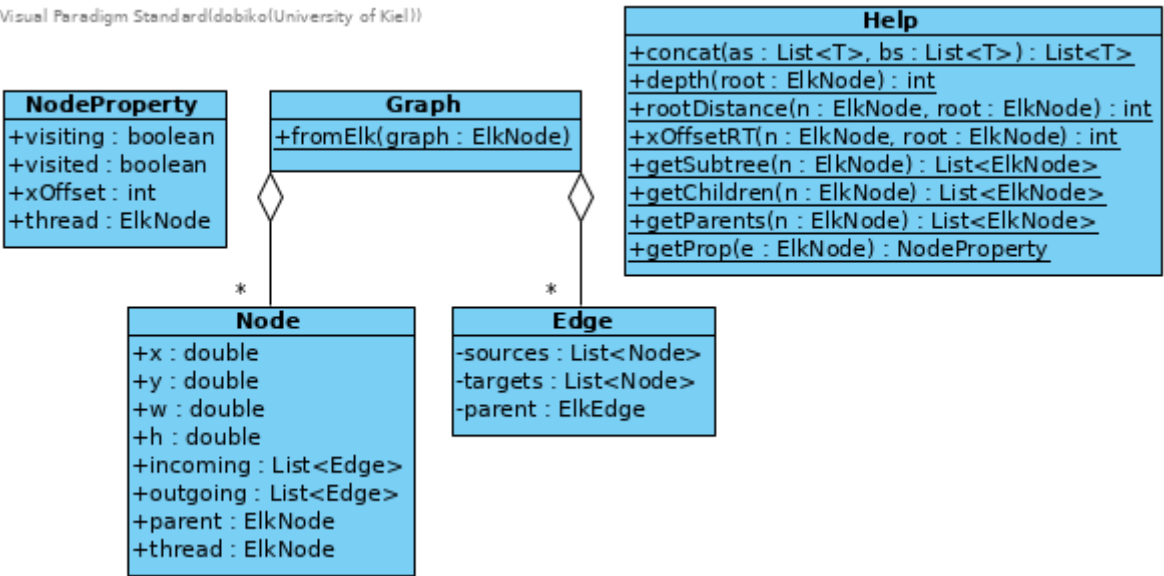
GraphLoader

This class contains the nessecary methods to read, parse, check and update the GraphStates in the GraphStatesManager.

Options

This class contains final and non final options for the program. The non final ones can be changed in the options tab in the GUI.

helper

Visual Paradigm Standard(dobiko(University of Kiel))

NodeProperty
+visiting : boolean
+visited : boolean
+xOffset : int
+thread : ElkNode

Graph
+fromElk(graph : ElkNode)

Help
+concat(as : List<T>, bs : List<T>) : List<T>
+depth(root : ElkNode) : int
+rootDistance(n : ElkNode, root : ElkNode) : int
+xOffsetRT(n : ElkNode, root : ElkNode) : int
+getSubtree(n : ElkNode) : List<ElkNode>
+getChildren(n : ElkNode) : List<ElkNode>
+getParents(n : ElkNode) : List<ElkNode>
+getProp(e : ElkNode) : NodeProperty

Node
+x : double
+y : double
+w : double
+h : double
+incoming : List<Edge>
+outgoing : List<Edge>
+parent : ElkNode
+thread : ElkNode

Edge
-sources : List<Node>
-targets : List<Node>
-parent : ElkEdge

Graph

This class is used to clone an Elk Graph using the fromElk() method so a GraphState can get a copy of the graph.

Node

This class mirrors an ElkNode and additionally also contains a RT thread pointer.

Edge
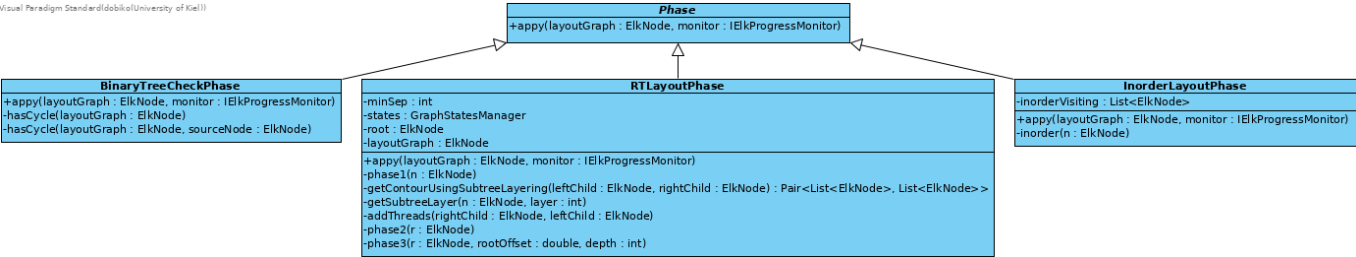
This class mirrors an ElkEdge.

Help

This class contains a lot of helpful michellenious methods.

NodeProperty

An instance of this class is saved for every ElkNode and contains needed additional information for the BinaryCheckPhase and the RTLayoutPhase.

phases

Phase

This is the superclass for all phase classes in this package. It defines an apply method that gets a reference to an ElkNode that contains a graph and a ProgressMonitor.

BinaryTreeCheckPhase

This is a Phase that throws an Exception if the apply method is called on a graph that isn't a binary tree. This Phase is applied before the other Phases in GraphLoader.load().

InorderLayoutPhase

This is a Phase that layouts a graph using the inorder algorithm.

RTLayoutPhase

This is a layout phase that layouts the graph using a RT implementation. However this phase is modified to additionally create GraphState instances of the current progress and add them to the GraphStatesManager.

Visualization

GUI
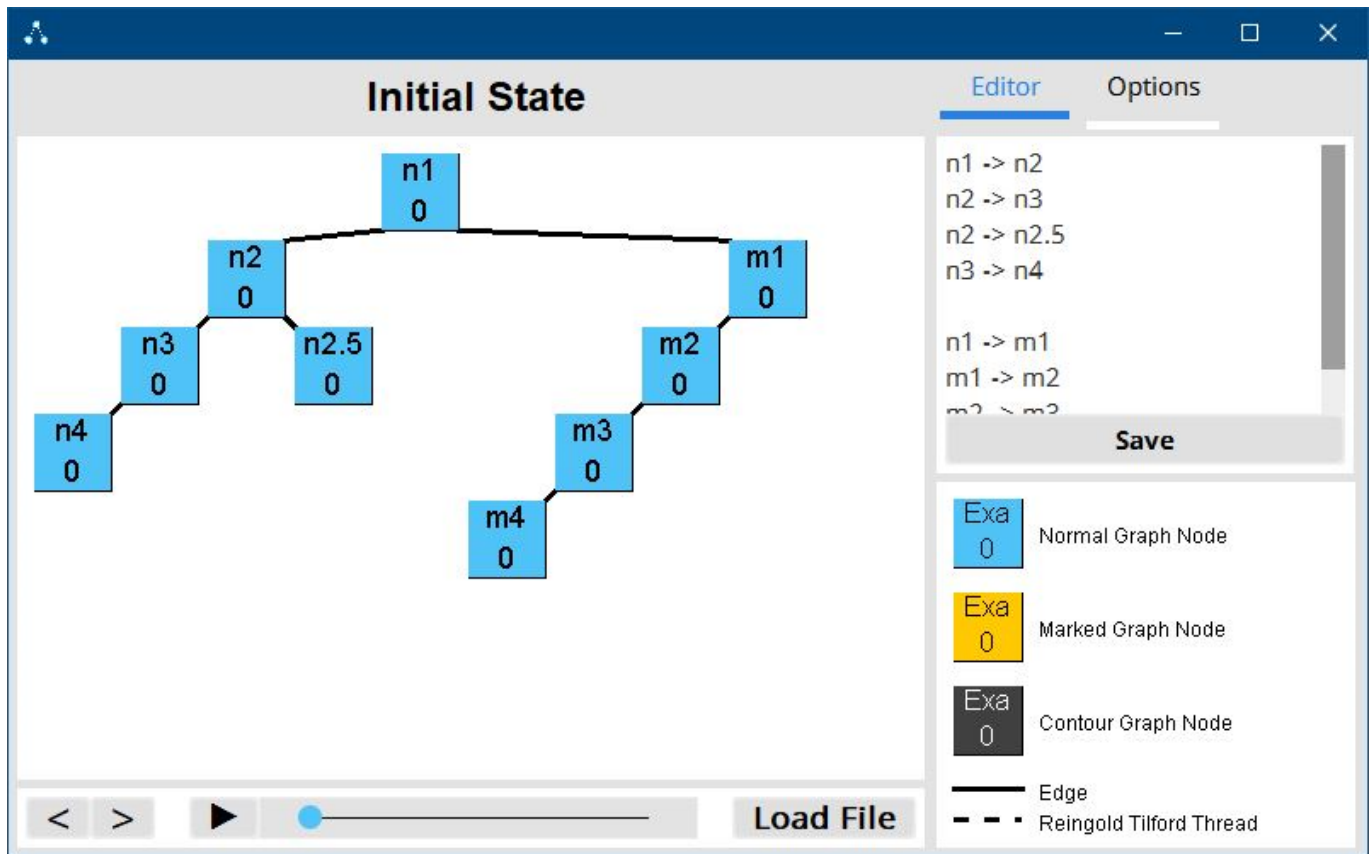
The left side of the GUI is designed similarly to a generic video player with the play plause button and the progress bar showing the progress through the video. However we also need buttons to step through the animation frame by frame. Similarly to sony vegas' gui I seperate the video player controls from the frame by frame buttons.

The right side of the GUI contains additional information like the legend, similarly to YouTubes info panel that i enabled on certain videos. The information displayed here is always important so there is no option to hide it.

Within the editorTab there is some padding around the editorPane because the text would otherwise start directly next to the background and no other text editor does that.
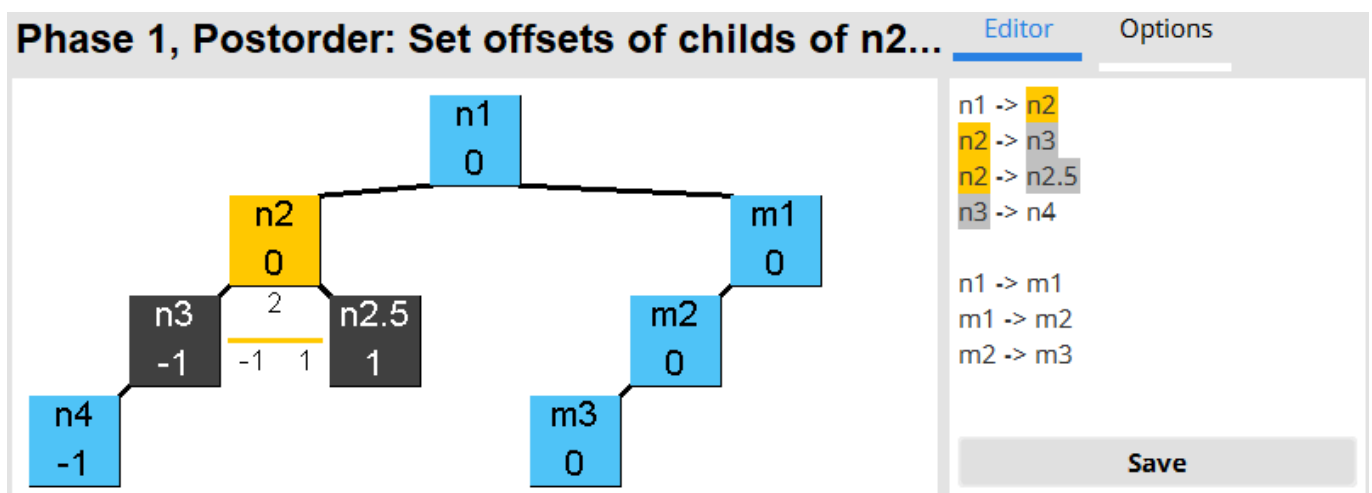
Filtering
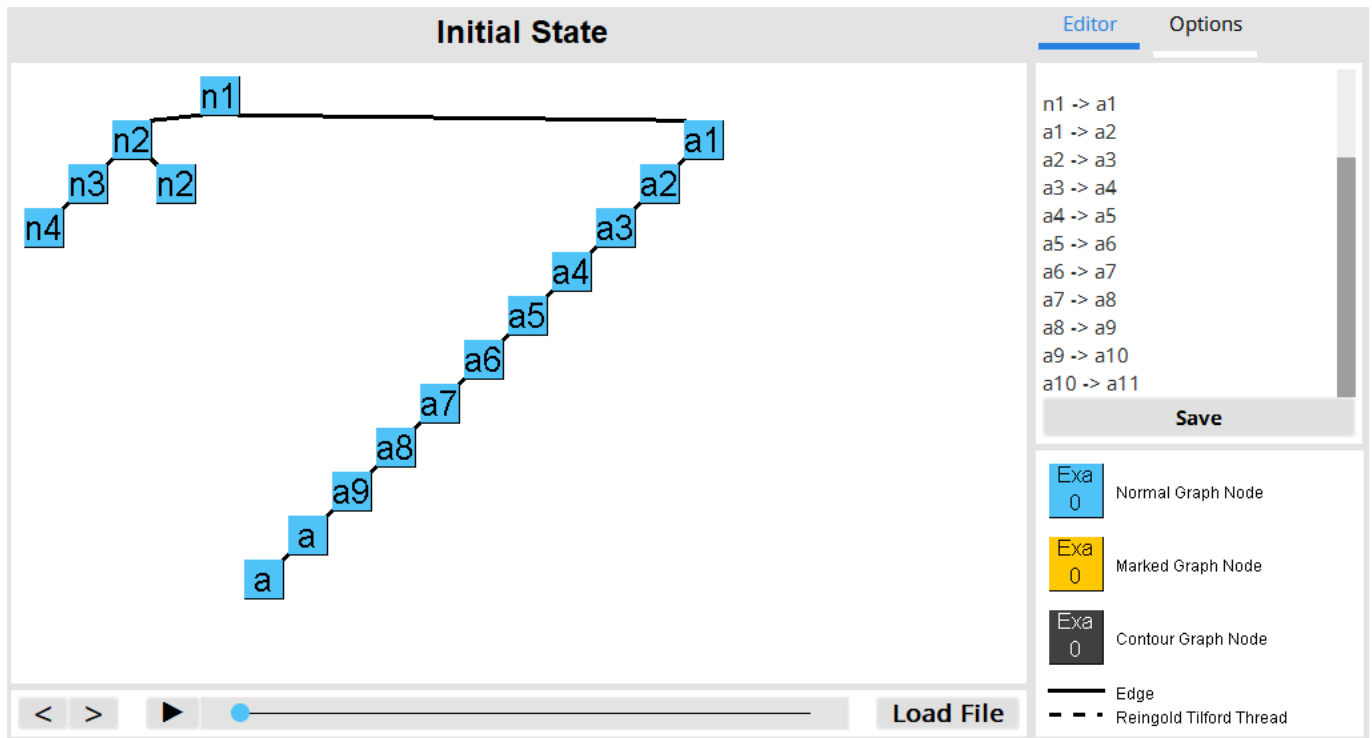
If there is not enough space for the Node name and its offset value to be drawn the program will draw the node name only to keep it readable.

Initial Phase

In the initial state the graph is layouted using the inorder algorithm because it tends to layout graphs too widely which gives us enough space to visualize the stages of the RT algorithm.

Phase 1

In the first phase of the RT algorithm we step through the graph in postorder, check contours and set the offset values for each node except the root.

Firstly to make clear in what order we stepping through the graph currently and what phase we are in this information is always displayed in the stateLabel before the :.

To show what node we are looking at in the current step the node is marked in an orange color. Due to the fact that this color is a warning color that should make it easy to see that this node is important in the current

step.

To emphasize nodes that belong to the contours that are currently checked, those nodes are drawn kind of negatively. The negative color of cyan is unfitting though so I used gray. That gray tone also fits well to the orange.

The layer of the currently checked contour difference is displayed by a line between the two nodes. Below that line at each side the actual contour value is drawn.

Due to the fact that the contour number and the threads might overlap the background of the contour number is cleared. Overlaps between thread lines and contour difference lines aren't so confusing because they are drawn in different colors.

When the difference line is drawn the dv value of the algorithm is drawn below the marked node to visualize that it's the horizontal difference between the two subtrees under it.
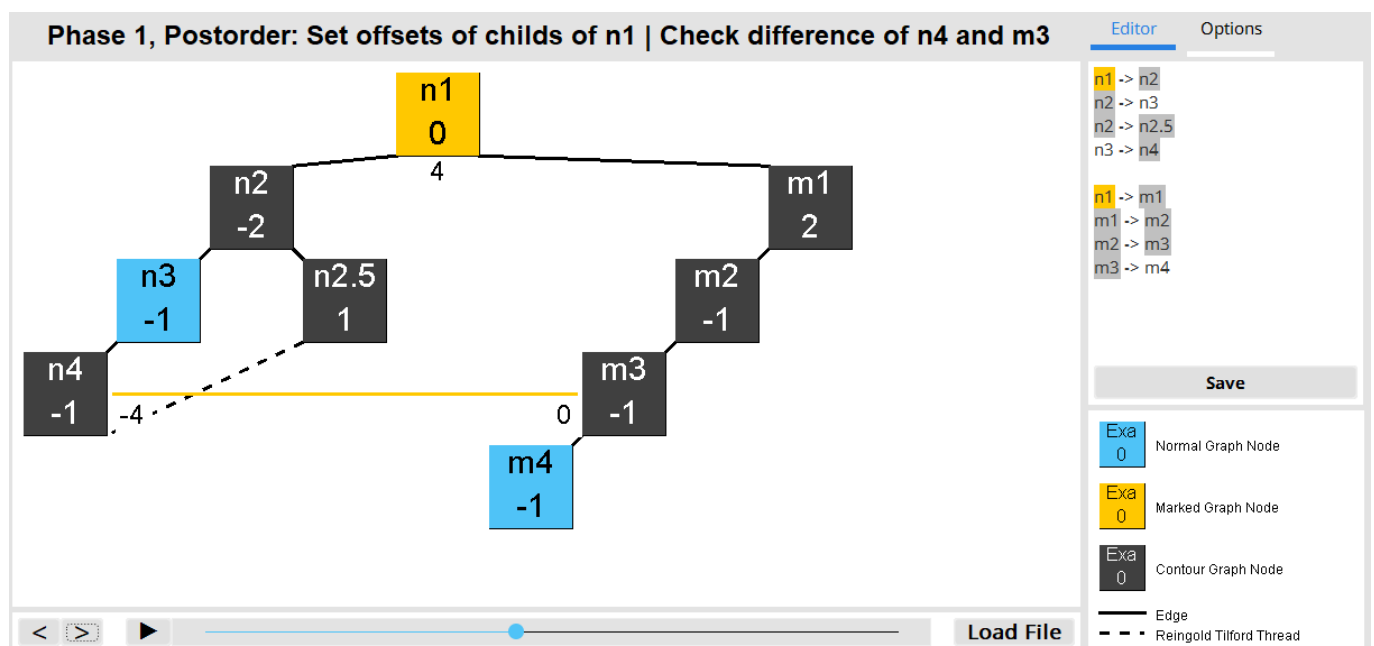

Threads that are created during this phase are displayed as dashed lines similarly to how they were drawn in the paper from Reingold and Tilford that first proposed the RT algorithm. Unlike edges however that are drawn from and to the edges of the nodes rectangle that are the closest to each other, the threads are routed always from and to the lower sides of the rectangle to prevent overlaps.

Phase 2
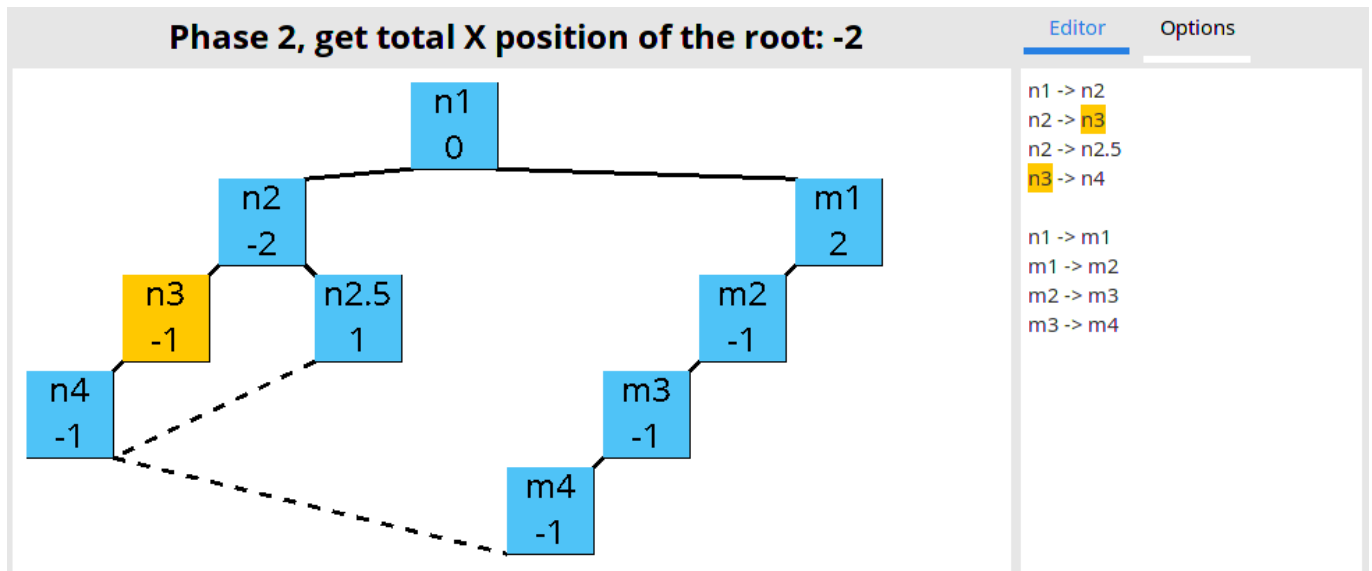
This phase is a lot simpler than the first one. All we do is finding out the roots X position within the grid by stepping through its left children recursively. We mark the node we are currently visitig again and show the currently accumulated X position in the title.

Phase 3
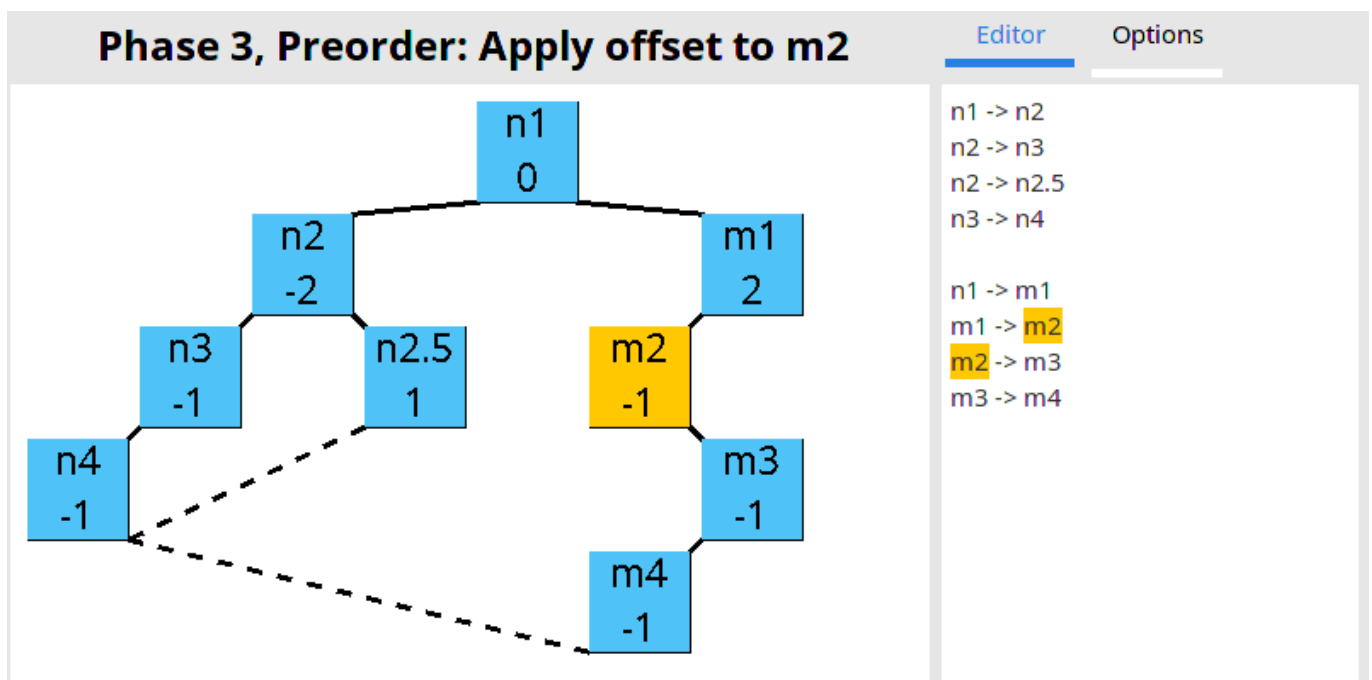
In this phase we step through the graph in Preorder and apply the offsets we calculated earlier. The node we are currently processing is marked again and the order is displayed in the title.