Galant Programmer’s Guide (to the Galaxy)

This document describes how to write a graph algorithm for Galant. Graph algorithm code is written using Java syntax through the Algorithm Editor panel, and requires no knowledge of classes and minimal knowledge of functions. This is handled by Galant and the programmer need only write the function itself.

**Getting Started**

Provided by Galant and prepended to the written code is a Graph object, which is accessible through the graph variable or the getGraph() function.

**Graph States**

Modifications to the graph create a new state, which is used by Galant to step forwards and backwards through steps in the animation. Each individual API, unless marked otherwise, will increment this graph state by one. If this doesn’t make sense for the algorithm (e.g. several modifications should be changed at once), the user can lock the graph state and unlock it when a set of atomic updates have been completed.

WARNING: If a state is left locked, all executed code following the lock will appear as one step in the graph animation.

**API**

**Algorithm**

|  |  |
| --- | --- |
| beginStep() | Forces the animation to consider all graph changes one step until endStep() or another beginStep() is called. |
| endStep() | Ends a step in the graph state. If no beginStep() has been called previously, does nothing. |

**Graph**

|  |  |
| --- | --- |
| isDirected() | Returns TRUE if the graph is directed, FALSE otherwise. |
| setDirected(boolean directed) | Sets the graph to directed for True and undirected for False |
| getNodes() | Gets a list of all non-deleted nodes in the latest state of the graph. |
| getEdges() | Gets a list of all non-deleted edges in the latest state of the graph. |
| getNodeById(int id) | Gets the node whose id matches the input. Returns null if the node doesn’t exist or has been deleted. |
| getEdgeById(int id) | Gets the edge whose id matches the input. Returns null if the edge doesn’t exist or has been deleted. |
| select(Node n) | Sets the specified node as the selected node in the graph and deselects all other selected nodes.  Note: If the desired effect is to leave other Nodes selected, see the Node class’s setSelected method. |
| addNode() | Adds a Node with default settings to the graph and returns a pointer to the new Node. |
| addEdge(Node source, Node target) | Adds an edge to the graph between the two specified nodes, and also stores the edge in each Node. |
| addEdge(idSource, idTarget) | Adds an edge to the graph between two nodes with specified indices, and also stores the edge in each node. |

**Node**

|  |  |
| --- | --- |
| isSelected():boolean | Returns True if the node is selected in the latest state and False otherwise. |
| setSelected(boolean selected) | Sets the node as selected. In contrast with the Graph’s select function, this does not deselect other selected nodes in a graph.  Note: Selected nodes will be displayed with a red ring, overriding the default color set for a node. |
| isVisited():boolean | Returns the true if the Node has been marked as visited and False otherwise. |
| setVisited(boolean mark) | Sets the Node’s visited/marked property to the input value.  Note: Marked nodes’ interior color changes from white to light grey. |
| isMarked():boolean | Returns the true if the Node has been marked and False otherwise.  Note: Exhibits the same behavior as isVisited() |
| mark() | Sets the Node’s visited/marked property to True  Note: Marked nodes’ interior color changes from white to light grey. |
| getWeight():float | Returns the floating point weight of the current node. The default weight is 0. |
| setWeight(float) | Sets the weight of the current node. |
| getUnvisitedPaths():List<Edge> | Returns a List<Edge> object of all the Node’s incident edges connecting to unvisited Nodes in the latest state of the graph. |
| getVisitedPaths():List<Edge> | Returns a List<Edge> object of all the Node’s incident edges connecting to visited Nodes in the latest state of the graph. |
| getUnvisitedAdjacentNodes():List<Node> | Returns a List<Node> object of all the Node’s adjacent Nodes whose visited properties are False in the latest state of the graph. |
| getIncidentEdges():List<Edge> | Returns a List<Edge> object of all the edges incident to this node. |
| getIncomingEdges():List<Edge> | Returns a List<Edge> object of all the edges directed into this node if the graph is directed. If the graph is undirected all incident edges are returned. |
| getOutgoingEdges():List<Edge> | Returns a List<Edge> object of all the edges directed out of this node if the graph is directed. If the graph is undirected all incident edges are returned. |
| travel(Edge e):Node | Returns the other Node endpoint of the specified edge. Returns null if neither of the edge’s endpoints is this Node. If e is a loop, returns this Node. |
| getId():int | Returns the numerical id of the node. This will not change during Algorithm execution unless explicitly set in the user code. |
| getColor():String | Gets the default color of the Node in the format ‘#RRGGBB’ |
| setColor(String color) | Sets the default color of the Node. This color will correspond to the colored ring around a node and will be the default color when the node is not selected.  A valid color input should be of the form ‘#RRGGBB’ |
| getLabel():String | Returns the String label associated with the Node |
| setLabel(String label) | Sets the label associated with the Node |
| getPosition():Point | Gets the position of the node on the coordinate plane. Currently requires the algorithm to import java.awt.Point |
| getX():int | Returns the x-coordinate of the node. |
| getY():int | Returns the y-coordinate of the node. |
| setPosition(Point p) | Sets the position of the Node on the coordinate plane. |
| setPosition(int x, int y) | Sets the position of the Node on the coordinate plane. |
| equals(Node n):boolean | Returns true if the given Node points to the same Node as the current Node and false otherwise. |
| setStringAttribute(String:key, value) | Stores a String in the node under the specified key. This will overwrite any String, Integer, or Double attribute already set with the same key. |
| getStringAttribute(String key):String | Gets the String value associated with the specified key. Returns null if the key doesn’t exist or it exists but its value is not a String. |
| setIntegerAttribute(String:key, Integer value) | Stores an Integer in the node under the specified key. This will overwrite any String, Integer, or Double attribute already set with the same key. |
| getIntegerAttribute(String key):Integer | Gets the Integer value associated with the specified key. Returns null if the key doesn’t exist or it exists but its value is not an Integer. |
| setDoubleAttribute(String key, Double value) | Stores a Double in the node under the specified key, This will overwrite any String, Integer, or Double attribute already set with the same key. |
| getDoubleAttribute(String key):Double | Gets the Double value associated with the specified key. Returns null if the key doesn’t exist or it exists but its value is not a Double. |

**Edge**

|  |  |
| --- | --- |
| isSelected():boolean | Returns True if the edge is selected in the latest state and False otherwise. |
| setSelected(boolean selected) | Sets the edge as selected. In contrast with the Graph’s select function, this does not deselect other selected edges in a graph.  Note: Selected edges will be displayed with a red line, overriding the default color set for a node. |
| getWeight():int | Returns the Integer weight of the current edge. The default weight is 0. |
| setWeight() | Sets the weight of the current edge. |
| getSourceNode():Node | Returns the source Node of the current Edge.  Note: This has no additional meaning for undirected graphs. This method will return whatever is stored as the “source” attribute in the graphml. |
| setSourceNode(Node n) | Sets the source Node of the current Edge.  Note: This has no additional meaning for undirected graphs. This method will set the “source” attribute in the graphml. |
| getDestNode():Node | Returns the destination Node of the current Edge.  Note: This has no additional meaning for undirected graphs. This method will return whatever is stored as the “destination” attribute in the graphml. |
| setDestNode(Node n) | Sets the destination Node of the current Edge.  Note: This has no additional meaning for undirected graphs. This method will set the “destination” attribute in the graphml. |
| getOtherEndpoint(Node n):Node | Returns the other endpoint of an edge, or NULL if the provided node is not either endpoint in the specified edge. |
| getId():int | Returns the unique ID of the edge. This will not change during Algorithm execution unless explicitly set in the user code. |
| getColor():String | Gets the default color of the Node in the format ‘#RRGGBB’ |
| setColor(String color) | Sets the default color of the Edge. This color will corresponds to the highlighting on the edge’s path and will be the default color when the node is not selected.  A valid color input should be of the form ‘#RRGGBB’ |
| getLabel():String | Returns the String label associated with the Node |
| setLabel(String label) | Sets the label associated with the Node |
| equals(Edge e):boolean | Returns true if the given Edge points to the same Node as the current Edge and false otherwise. |
| setStringAttribute(String:key, value) | Stores a String in the edge under the specified key. This will overwrite any String, Integer, or Double attribute already set with the same key. |
| getStringAttribute(String key):String | Gets the String value associated with the specified key. Returns null if the key doesn’t exist or it exists but its value is not a String. |
| setIntegerAttribute(String:key, Integer value) | Stores an Integer in the edge under the specified key. This will overwrite any String, Integer, or Double attribute already set with the same key. |
| getIntegerAttribute(String key):Integer | Gets the Integer value associated with the specified key. Returns null if the key doesn’t exist or it exists but its value is not an Integer. |
| setDoubleAttribute(String key, Double value) | Stores a Double in the edge under the specified key, This will overwrite any String, Integer, or Double attribute already set with the same key. |
| getDoubleAttribute(String key):Double | Gets the Double value associated with the specified key. Returns null if the key doesn’t exist or it exists but its value is not a Double. |

**Macros**

Macros abstract away some Java code and provide a simpler interface specific to Galant and graph algorithms.

When an algorithm is compiled, macros are converted into Java code for the Java compiler. This is similar to C’s preprocessor, although macros are defined within Galant rather than in the algorithm code itself.

**Note:** macros are not yet able to ignore text in comments or strings.

Now there are three macros:

**for\_adjacent**, **for\_outgoing,** and **for\_incoming** all with the same basic syntax (shown for **for\_adjacent** only):

**for\_adjacent** corresponds to getIncidentEdges()

**for\_outgoing** corresponds to getOutgoingEdges()

**for\_incoming** corresponds to getIncidentEdges()

Usage: for\_adjacent(*node*, *edge*, *adjacentNode*) {*code\_block*}

Parameters:

*node*: the name of a variable of type Node. This is the starting node.

*adjacentNode*: a variable name. Within the code block, this can be used to refer to the current adjacent node as a Node object.

*edge*: a variable name. Within the code block, this can be used to refer to the current incident edge as an Edge object. *edge* connects *node* and *adjacentNode*.

*code\_block*: a block of code that is executed for each adjacent node / incident edge of *node*. The curly braces are required.

**for\_nodes** – iterates over all nodes in the graph.

Usage: for\_nodes(*node*) *code\_block*

Parameters:

*node*: a variable name. Within the code block, this can be used to refer to the current node as a Node object.

*code\_block* a block of code that is executed for each node in the graph.

**for\_edges** – iterates over all edges in the graph.

Usage: for\_edges(*edge*) *code\_block*

Parameters:

*edge*: a variable name. Within the code block, this can be used to refer to the current edge as an Edge object.

*code\_block*: a block of code that is executed for each edge in the graph.

**function** – creates a function that can be called later.

Functions are objects (of type Function), and so they can be assigned to variables and passed to other functions. (Note: this is not properly implemented at the moment (and probably isn’t very high priority).)

Usage: function *[return\_type]* *name*(*params*) {*code\_block*}

Parameters:

*return\_type* (optional): a type. If the function returns a value, this should indicate the type of value returned.

If no value is returned, this is not necessary.

*name*: a variable name. Used to identify and call the function.

*params*: a comma-separated list of variable names, including types (e.g., "int i, String str").

Can be referenced from within the code block. May be empty, if there are no parameters.

*code\_block*: a block of code that is executed when the function is called. The curly braces are required.

**Calling a function** – calls a function created by **function**.

Usage: *name*(*args*)

Returns a value of the type defined by *return\_type* in **function**.

*name*: the same as *name* in **function**.

*args*: values of the types determined by *params* in **function**, which are passed to the function.

**Notes**

The keyword bool can be used in place of boolean if desired.

**Imports**

A set of classes, shown below, are imported by default into the algorithm. If a user requires an additional import, it may be specified at the top of the algorithm.

java.util.LinkedList

java.util.Queue

edu.ncsu.csc.Galant.algorithm.Algorithm

edu.ncsu.csc.Galant.graph.component.Graph

edu.ncsu.csc.Galant.graph.component.Node

edu.ncsu.csc.Galant.graph.component.Edge

edu.ncsu.csc.Galant.algorithm.code.macro.Function

edu.ncsu.csc.Galant.algorithm.code.macro.Pair