GRAPH-ENE: A WEB-BASED GRAPHING PROGRAM

By

James E. Torres

An Honors Project submitted in Partial Fulfillment

Of the Requirements for Honors in

The Department of Mathematics and Computer Science

Faculty of Arts and Sciences

Rhode Island College

2014

Acknowledgements

I would like to express my profound gratitude and appreciation to my faculty advisor and mentor, Dr. Robert Ravenscroft, who is responsible for planting the seed that culminated into the project and worked closely with me throughout the project's entirety. Without Robert's unwavering patience, exemplary guidance, help, and excitement for research and programming, the project would not have been possible.

Also, I would like to express my gratitude to Dr. Sanders, Dr. Sarawagi, Dr. McDowell and Dr. Moskol, for instilling within me the fundamental knowledge, curiosity and acumen in the field of Computer Science. Their efforts will transcend my time as an undergraduate at Rhode Island College.

I am indebted and greatly appreciative of the S-STEM Scholarship for providing the financial support I needed to continue pursuing my undergraduate studies at Rhode Island College.

I owe a great deal of thanks to Dolly Quigley and Sherry Zinn for their support over the years, and providing the supplies, advice and design ideas that went into constructing the project's poster which ended up being displayed at the Consortium for Computing Sciences in Colleges North Eastern (CCSCNE) conference at Providence College, the same year of this publication.

In addition, a thank you goes to Bill Ebeling, who was able to set up the poster at CCSCNE conference when I could not and for supplying book resources that aided in the development of the project.

Lastly, I thank my parents and siblings for their encouragement and belief in me when I couldn't believe in myself. Also, for putting up with the countless hours I stayed up late, diligently typing code.

Table of Contents

Table of Figures

1 Introduction

GRAPH-ENE is a rich internet application for building and manipulating undirected, simple graphs. It is intended for use as a classroom teaching aid, plus as a tool for students to interactively manipulate graphs for assignments. Being web based, it is portable—it can run anywhere a browser is available. Since it is interactive, it provides problem-solving capabilities that are not available using pencil and paper.

Graphs are built using the mouse. Edges and vertices can be labeled. Graphs are manipulated by dragging vertices or edges with a mouse. A highlighting tool allows for vertices, edges, labels and sub-graphs to be marked, as well as for graphs to be colored. An animation feature allows a sequence of interactions to be recorded and played back. The interface includes the ability to print, open and download the graph data, generate and download an image file, and other functionalities pertaining to graph manipulation. It currently does not support loops on a vertex or multi-edges between vertex pairs.

GRAPH-ENE is built using JavaScript, CSS and features available in HTML5. HTML 5 provides various APIs for building robust applications. Also, HTML 5 is new and evolving technology that is in various stages of implementation on various browsers. HTML implementations vary in their completeness and capabilities. For example, Internet Explorer does not permit client-side generated files to be downloaded, which does not mesh with an RIA. A Rich Internet Application (RIA for short) is a web application that has many of the same characteristics as desktop software, but is delivered over the internet as a browser-plugin, extensive use of JavaScript, or a virtual machine. Three of the most common RIA platforms are Adobe Flash, JavaFX and Microsoft Silverlight, all of which require the user to have them installed on their computer before launching the application. Since GRAPH-ENE is an RIA, built with JavaScript and HTML 5, it does not require the user to install software.

GRAPH-ENE is being tested in a Mathematics general education course. Graph problems that GRAPH-ENE is being used with include relationship modeling with graphs, paths and circuits, trees, and spanning trees. In the future, we hope to evaluate GRAPH-ENE in Computer Science courses. While we have found other tools that build and edit graphs, they are geared more for illustration or publishing, and not for teaching. We have not found a well-developed tool with the capabilities similar to those of GRAPH-ENE. Section 2 of this paper discusses the motivations and related work of GRAPH-ENE. The user interface is discussed in Section 3 and some problem applications are considered in Section 4. Section 5 provides an overview of implementation. Future work is considered in Section 6.

2 Motivations and Related Work

Before GRAPH-ENE came to fruition, the initial idea for an honors project was the creation of a program that would be used in an academic environment – some sort of tool that could help students understand concepts in Mathematics or Computer Science, or a tool to help instructors teach. Furthermore, the intent was to explore uncharted territory by trying out other programming languages and paradigms for the purpose of becoming a more adept and versatile software engineer, as well as to feed a budding curiosity.

GRAPH-ENE's primary motivation came from previous work done by Dr. Robert Ravenscroft, a professor and faculty advisor at Rhode Island College. Dr. Ravenscroft developed a prototype graphing program for a Discrete Structures course taught to first year Computer Science students at Millersville University in Millersville, Pa. Dr. Ravenscroft used the program for classroom demonstrations in the Discrete Structures course. He also used the program for class lectures in a Mathematical Foundations of Computer Science class at the University of Rhode Island. Many students used it for their assignments and gave it positive feedback. They also provided many useful suggestions for future improvements and enhancements. Intrigued by this program, we decided to re-implement the program to make it portable and to take advantages of newer technology. In order to accomplish this, we opted to use JavaScript and HTML 5 to make the program a browser based application.

A search for existing systems did not yield any with the functionality we desired for our system. We did, however, find several programs which provide some form of graph building capabilities. These programs are yFiles for HTML, Dia, and CPMP-tools.

The yFiles for HTML contains user interface controls for viewing and editing diagrams and includes layout algorithms for automatic arrangement of different graphs. It features client-side graph editing in HTML5-capable browsers, leverages both SVG (Scalable Vector Graphics) and HTML5, and provides a pure JavaScript solution that works with various frameworks and optional server components (Java/.NET) for layout and computationally-intensive tasks.

Dia is a free, open-source, vector-based drawing tool used to create flowcharts, network diagrams, circuit diagrams, and more. Dia is extensible—it supports adding new shapes by writing XML files and using a subset of SVG to draw the new shapes. Dia also allows the loading and saving of diagrams and can be scripted using the programming language known as Python.

CPMP-Tools is a suite of tools which are designed to support students with specific curriculum applications in Algebra, Geometry, Statistics and Discrete Math. The Algebra tools consist of an electronic spreadsheet and a CAS (computer algebra system) that produces tables and graphs of functions, manipulates algebraic expressions and so forth. The Geometry tools allow for interactive drawing for constructing and manipulating geometric figures and a simple object-oriented programming language for creating animation effects. The Statistics tools include tools for graphic display and analysis of data, simulation of probabilistic situations, and mathematical modeling of quantitative relationships. The Discrete Math tools allows the construction, manipulation and analyzing of vertex-edge graphs. Overall, CPMP-tools purpose is to assist students with problem solving in Mathematics.

One of the shortcomings of yFiles for HTML is that a license must be purchased for use after a limited-time trial. Dia and CPMP tools are available for free but requires users to download and install software to run. Dia and yFiles for HTML are more suited for creating and rendering diagrams for publishing purposes and neither have the kind of graph manipulation capabilities we felt were desirable for a teaching environment. The Discrete Math tool of CPMP Tools encompassed some of the capabilities we wanted in GRAPH-ENE, the main one being the ability to build graphs. However, the task of building a graph out of edges and vertices wasn't very user-friendly and the user interface had a bit of a learning curve.

3 GRAPH-ENE User Interface Experience

One of the goals with GRAPH-ENE was to have a simple interface, both for ease of use for general education students, but to also make it easy for an instructor to use in the classroom. Two primary clients of the system are general education Mathematics students and class room instructors. Both require an easy to use tool – minimal buttons, searching, reducing clicks, building and manipulating graphs.

Figure 1 on the next page shows the interface and labels show the number of the subsection that discusses each feature in detail.

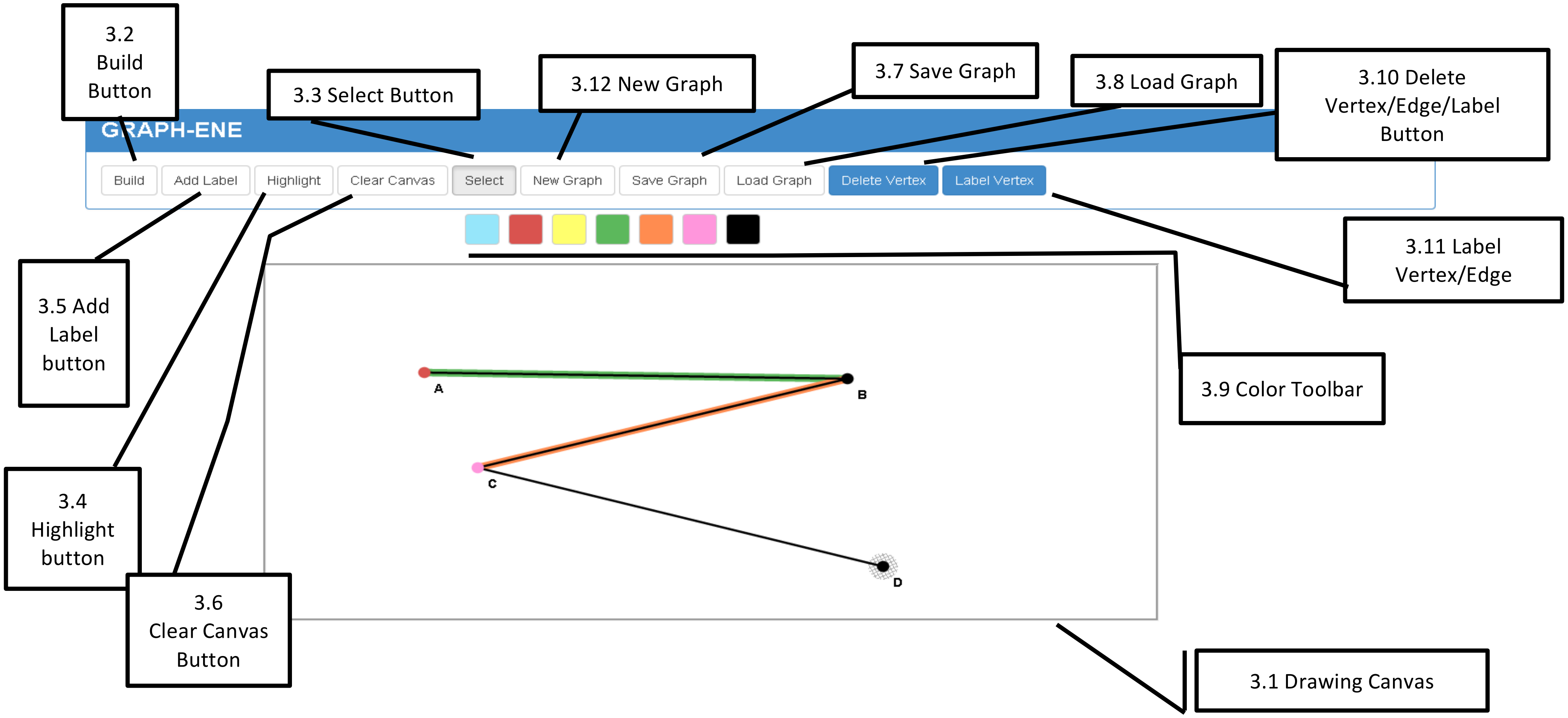


Figure 1 - Screen capture of GRAPH-ENE User Interface

3.1 Drawing Canvas

This is the region where the user builds the graph. It responds to mouse actions: presses, releases, clicks and drags. There are no modifier keys. Dragging the mouse out of bounds terminates the current action being performed.

3.2 Build

The **Build** button allows users to build graphs from vertices and edges.

i. If the user clicks the mouse without dragging, a single vertex is placed on the canvas.

ii. If the user presses the mouse, drags the mouse, then releases the mouse, an edge is created with two vertices as the end points.

iii. If the user presses on an existing vertex and drags the mouse and then releases the mouse, an edge is created from the existing vertex to a vertex at the new location.

iv. If the user drags off the canvas, the action is discarded and any selected object is placed back to its original position.

3.3 Select

The **Select** button gives users the ability to select and manipulate graph objects on the canvas.

i. If the user presses the mouse on an existing graph object, a hatch-like pattern appears on the selected object to denote it has been selected. New buttons will appear on the toolbar that will provide a few options that allow the selected object to be modified.

ii. If the user clicks on an existing graph object and drags the mouse, the graph object will track and move with the mouse pointer. If the graph object has any other graph objects attached to it, these graph objects will update in correspondence to the selected graph object's new location.

iii. If a graph object is selected and the user clicks on any of the color swatches, the color of the selected graph object will change to reflect the selected color.

3.4 Highlight

The **Highlight button** gives users the ability to highlight graph objects on the canvas in the color they choose. Yellow is the default highlight color.

i. If the user is in highlight mode and clicks on a graph object, the graph object will be highlighted.

ii. If the user clicks on a graph object that is the same color as the current highlight color, the graph object will no longer be highlighted.

iii. If the user clicks on a graph object that is of a different color than the currently selected highlight color, the graph object's highlighted color will be replaced by the currently selected highlight color.

3.5 Add Label

The **Add Label** button allows users to place labels on the canvas. These labels are independent from other graph objects but can be dragged, colored, highlighted or deleted if necessary.

i. If the user has pressed the **Add Label** button and proceeds to click on the canvas, a dialog box opens where the user is prompted to enter text that the label will display.

ii. If OK is clicked, the label will appear on the canvas at the location they clicked.

iii. If CANCEL is clicked, the action is discarded.

3.6 Clear Canvas

The **Clear Canvas** button will clear the canvas of all existing graph objects.

i. If the user clicks on the **Clear Canvas** button, the user is prompted to ensure whether they wish to clear the canvas.

ii. On pressing OK, the canvas will be cleared.

iii. On pressing CANCEL, the action is discarded.

3.7 Save Graph

Save Graph allows users to save their current graph as a downloadable text file. Browsers implement different measures when saving files based on their implementation or user preferences.

i. If the user clicks on the Save Graph button the user downloads the file as they would any web based download. Note, Internet Explorer does not allow this without a customized work-around using Flash.

3.8 Load Graph

Open Graph allows users to upload a saved graph from their computer.

i. If the user clicks on the **Open Graph** button, a dialog box will appear that allows users to locate the saved graph file they wish to load.

ii. GRAPH-ENE does not verify the source file integrity. The canvas will not render files that were not originally saved or modified through GRAPH-ENE

3.9 Color Toolbar

The **Color Toolbar** consists of a palette of colors which the user can click to change the color of a graph object. The **Color Toolbar** will appear only if certain circumstances are met which are described below.

i. If the user clicks on a graph object, while in **Select** mode, the **Color Toolbar** will appear. From here, the user may click on a color swatch and in turn, will update the currently selected graph object's color to reflect the selected swatch.

ii. When the user clicks on the **Highlight** button, the **Color Toolbar** will appear. From here, the user may click on a color swatch to set the highlight color. Once set, clicking on any graph objects will highlight them in the selected color.

iii. Clicking on a color swatch while no graph object is selected will yield no result.

iv. If the user switches from Highlight to another mode, the last selected highlight color is retained until the user goes back to **Highlight** mode and explicitly chooses a different color.

3.10 Delete Vertex/Edge/Label

The **Delete** button lets users delete the currently selected object from the canvas. This button will appear when the user has selected a graph object. Note that the button is suffixed by the type of currently selected graph object.

i. If the user has selected an isolated Vertex and clicks the **Delete Vertex** button, the Vertex is deleted from the canvas.

ii. If the user has selected a Vertex of degree 1 or more, and clicks the **Delete Vertex** button, the Vertex and the edges connected to the vertex are all deleted from the canvas.

iii. If the user has selected an Edge and clicks on the **Delete Edge** button, the Edge is deleted from the canvas. The endpoints of the deleted Edge remain intact.

iv. If the user has selected a Label and clicks on the **Delete Label** button, the label is deleted from the canvas.

3.11 Label Vertex/Edge

The **Label** button lets users attach a label to the currently selected object. This button will appear when the user has selected a graph object. Note that the button is suffixed by the type of currently selected graph object. Currently, a graph object can only have a single label.

i. If the user has selected a graph object and clicks on the **Label (object type)** button, a dialog box appears that allows the user to enter text for the Label.

ii. Once the user has entered text and clicks OK, the label will appear relative to the selected graph object.

iii. If the user clicks CANCEL, the action is discarded and the dialog box disappears.

iv. If the user does not enter text and clicks OK, an alert is thrown notifying the user that the label cannot be blank or empty.

v. If the currently selected graph object already has a Label and the **Label** button is clicked, a dialog box appears with the label's current text in the text field. The user may modify this text to their choosing.

3.12 New Graph

The **New Graph** button opens a new instance of the application in a separate browser window.

3.13 Printing

i. Regular printing for browsers does work with the current program.

ii. We have developed a prototype for printing just the canvas that has not made it into the production model of the program.

4 Problem Applications

The following screen captures are some example problems that GRAPH-ENE can be applied to:

4.1 Graph Isomorphism

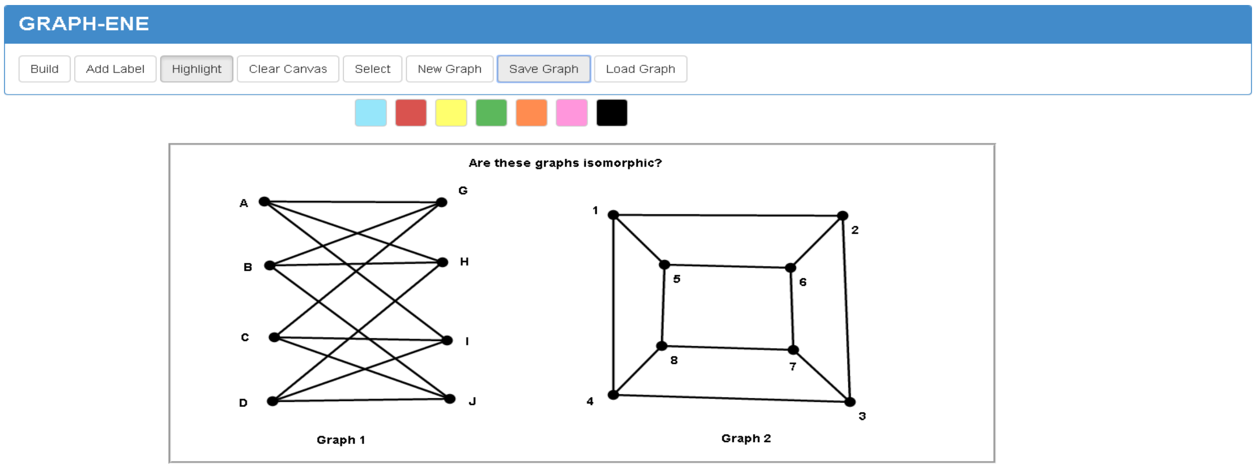


Figure 2 – Determine if both graphs are equivalent

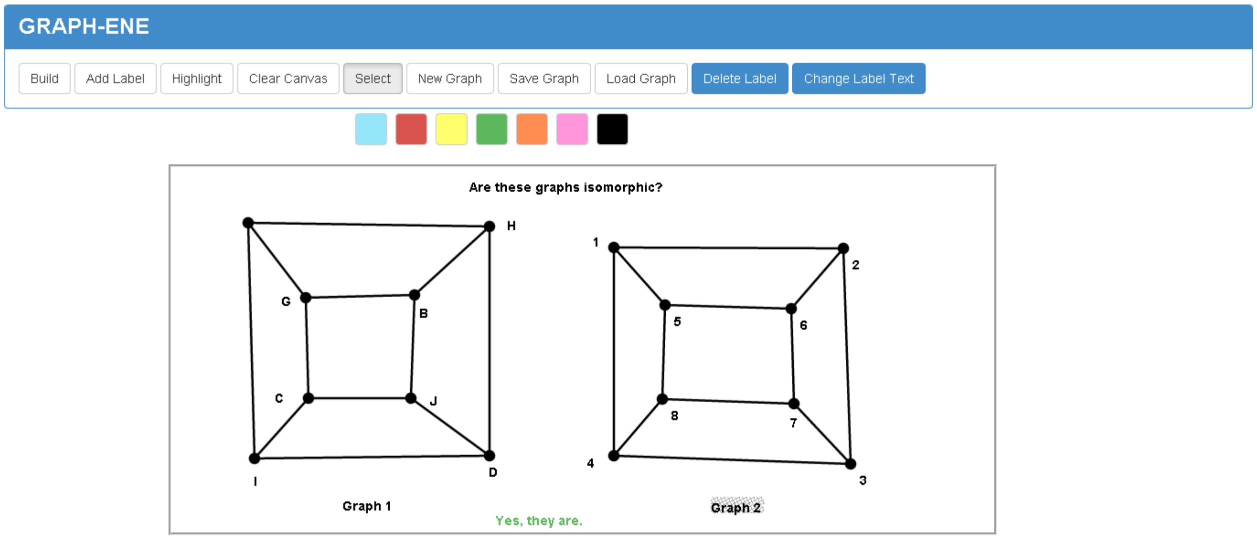
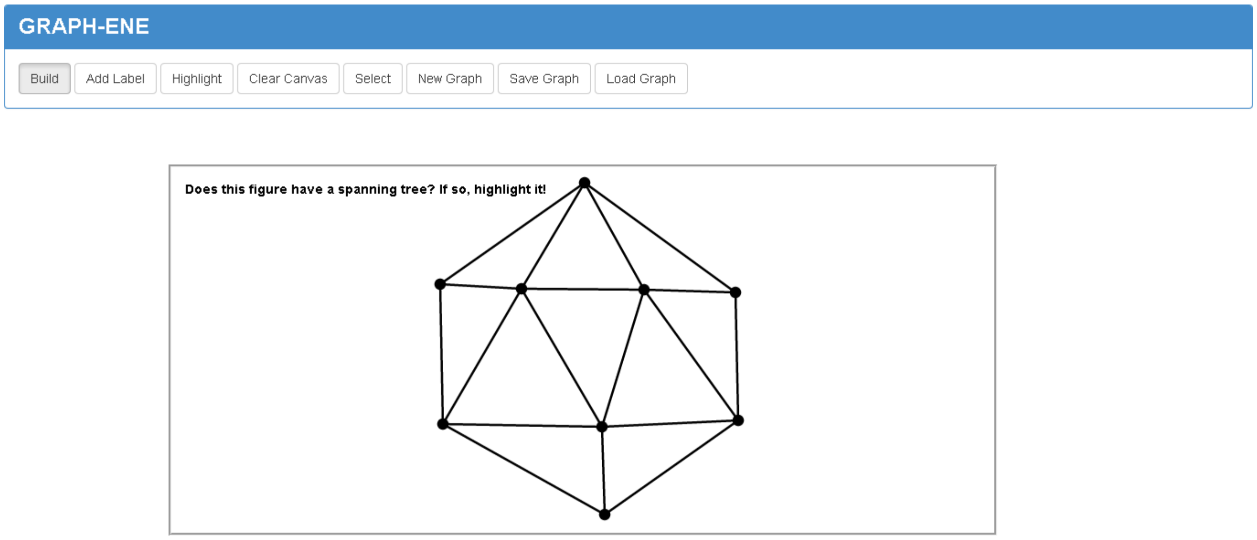


Figure 3 – By dragging vertices, Graph 1 can be transformed to equal Graph 2.

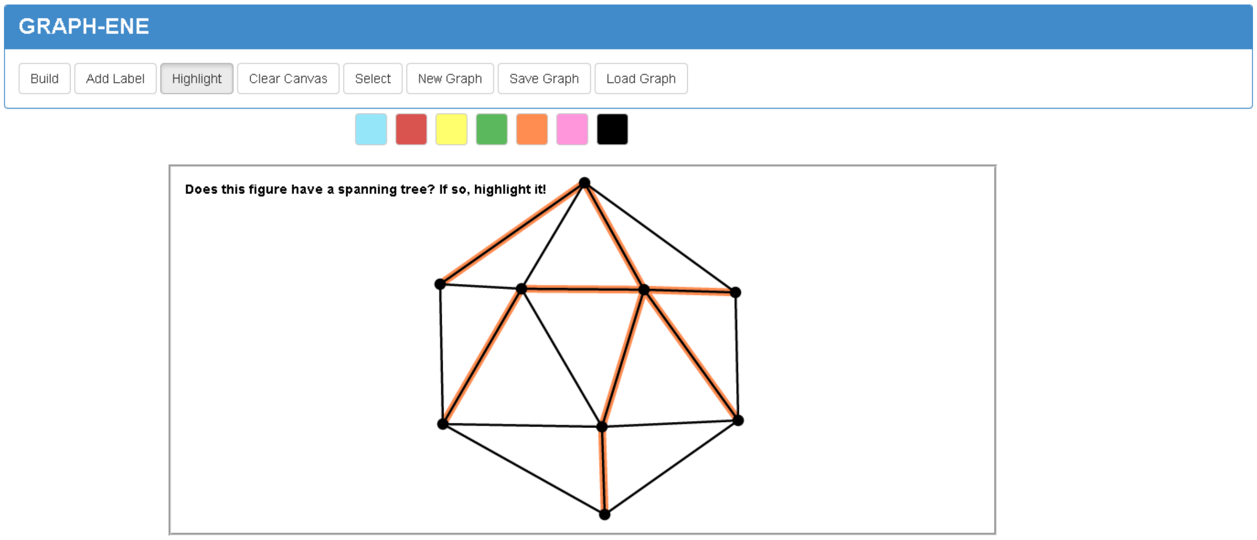
4.2 Spanning Trees

In graph theory, a spanning tree is a tree that includes all of the vertices and some or all of the edges of a graph.



Does this figure have a spanning tree? If so, highlight it!

Figure 4 – Finding a spanning tree – before highlighting.



Does this figure have a spanning tree? If so, highlight it!

Figure 5 – Using the highlight feature, a spanning tree can be highlighted.

4.3 Planar Graphing

A planar graph is a graph that can be drawn in such away that no edges cross each other.

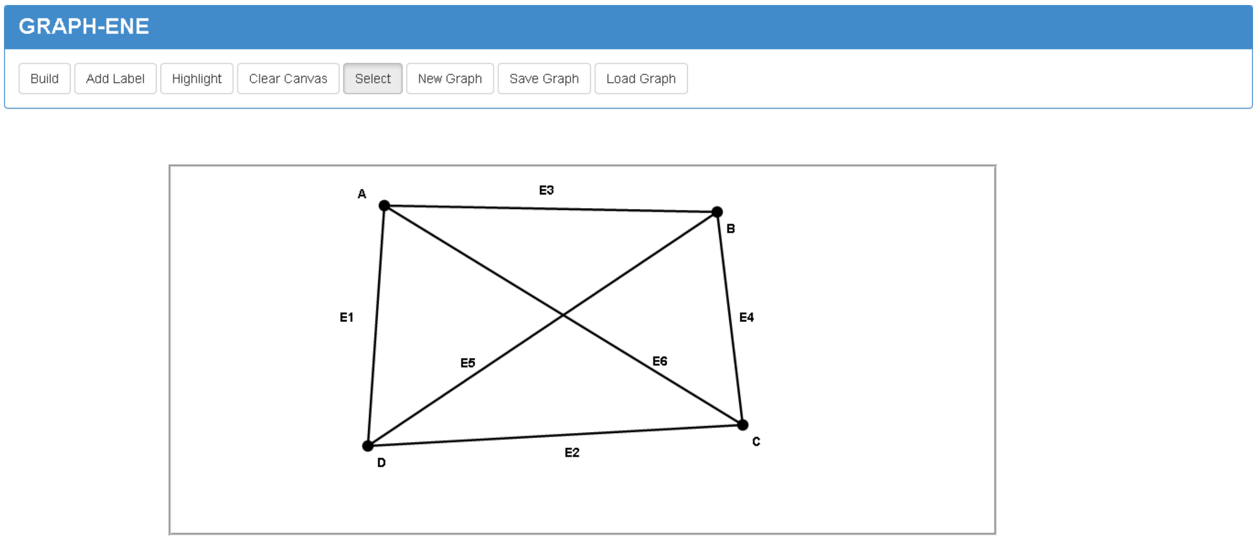


Figure 6 – Non-planar graph with crossing edges

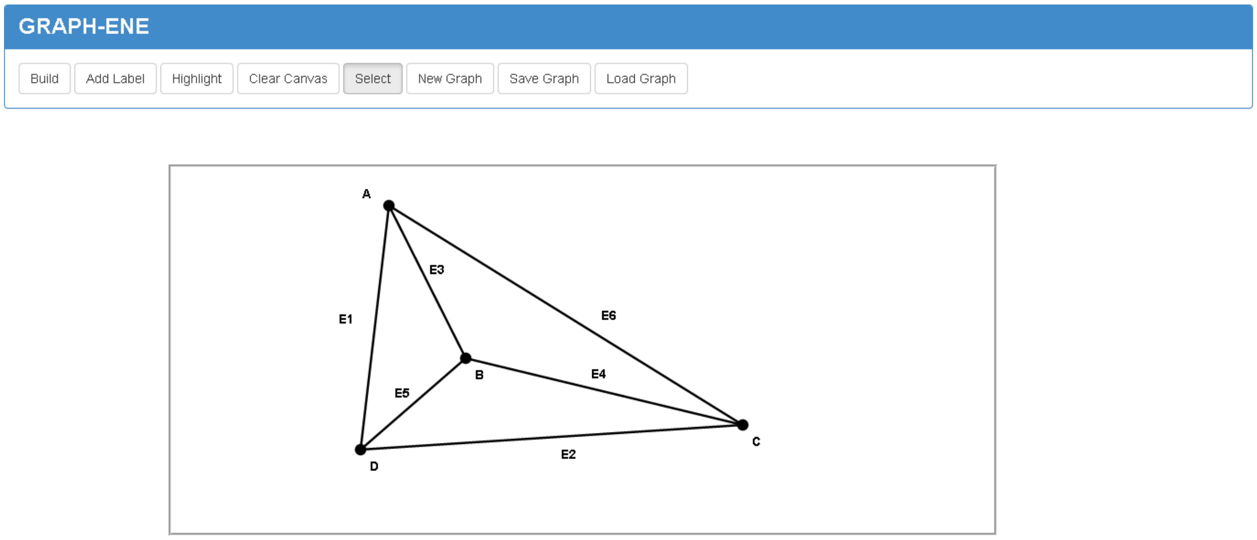


Figure 7 - Moving Vertex B to the center creates a planar graph from Figure 6

5 Implementation

GRAPH-ENE is implemented in JavaScript with HTML 5. It was developed on the Google Chrome and Mozilla Firefox browsers. We wanted to build the system and did not want to spend time dealing with cross browser issues. GRAPH-ENE uses a Model-View-Controller (MVC for short) paradigm for the purpose of keeping code readable, maintainable and manageable. The purpose of the View is to contain the objects that display information to the user. The purpose of the Model is to represent the objects or data within the system and encapsulating the code for the actions that can be performed on those objects. The purpose of the controller is to manage the interaction with the user and mediates between the Model and the View. The JavaScript Graph class implements the Model. The webpage & the HTML 5 Canvas element both comprise of the view which display the user interface and generates the output representation of a graph to the user, respectively. HTML also provides various user interface controls. The JavaScript Controller class is the Controller. This paradigm is discussed in further detail in the following subsections.

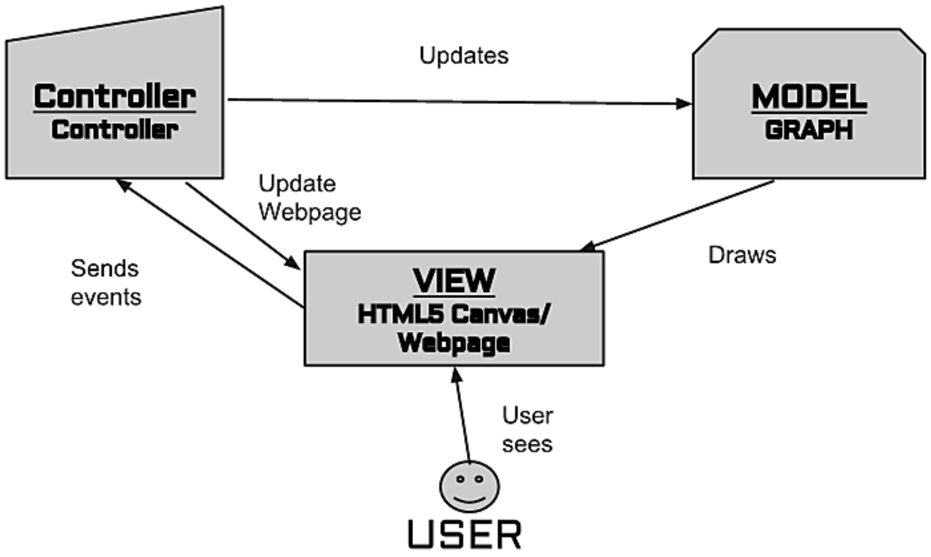


Figure 8 - Model View Controller

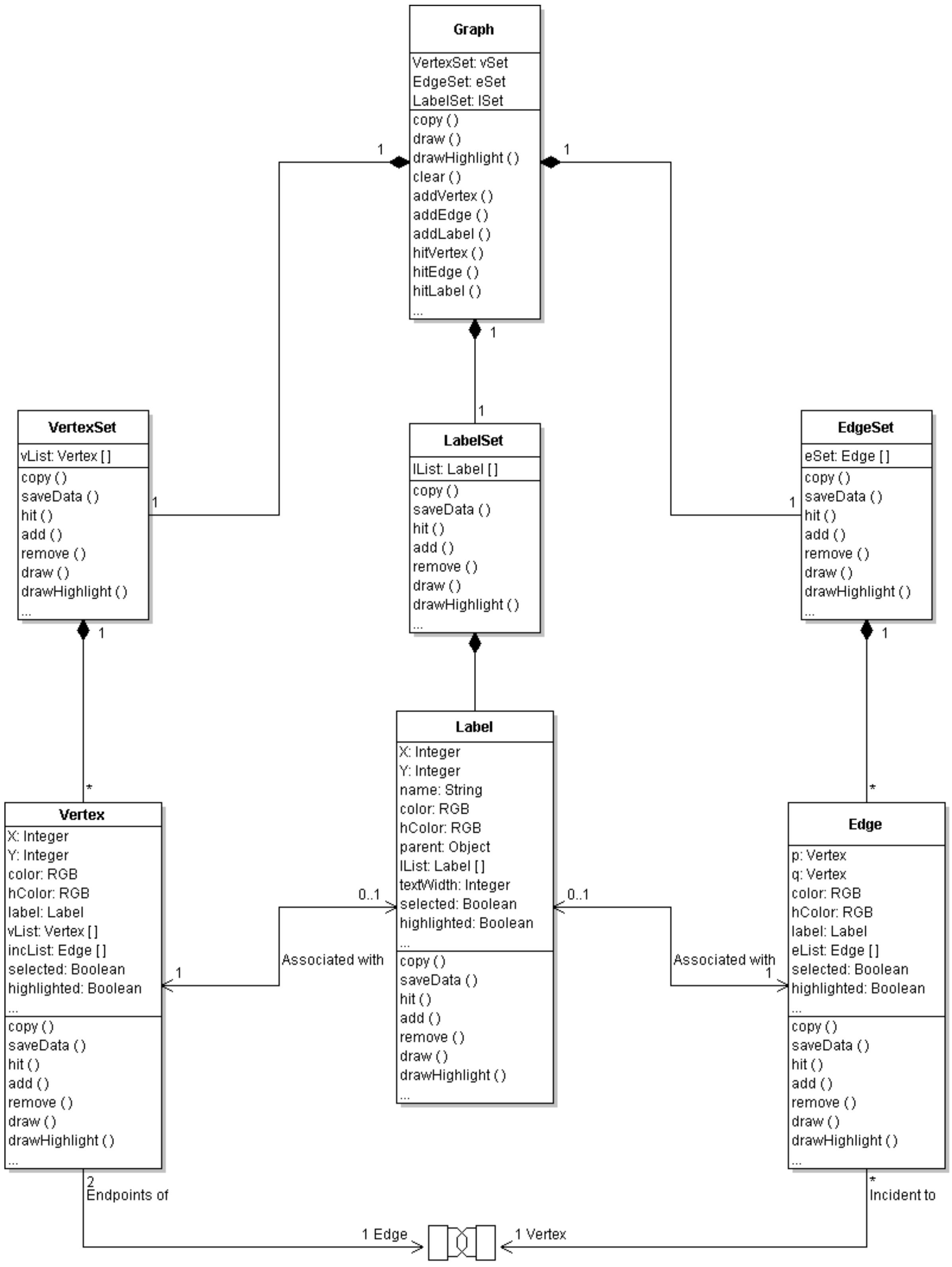


Figure 9 - Diagram of the Graph Class

5.1 The Model

A graph is mathematically defined as a set of vertices and a set of edges that connect the vertices. The Graph class models these components as well as provides labels and other attributes for customizing the graphs. The Graph class is a composite class built from the classes Vertex Set, Edge Set, and Label Set. Each of these sets is a collection of JavaScript objects: vertex objects in a Vertex Set, edge objects in an Edge set, and label objects in a Label Set. The classes Vertex, Edge, and Label define the actual objects that will be displayed on the HTML 5 canvas. The classes Vertex, Edge, and Label provide methods for manipulating the objects they represent. These methods include, but are not limited to: moving, drawing, drawing the objects selected or highlight pattern, removing or deleting, copying, and setting or getting the values of their properties.

A Vertex object consists of the following properties: its position in the form of an X coordinate and Y coordinate, color, highlight color, a list that holds edges that are incident to the vertex, a reference to the Vertex Set it belongs in, a reference to its label and Boolean properties that determine whether it is currently selected or highlighted.

An Edge object consists of the following properties: The endpoints p, and q of the edge, a reference to its label, a reference to the Edge Set it belongs in, color, highlight color, and Boolean properties that determine whether it is currently selected or highlighted.

The Label object consists of the following properties: its position in the form of an X and Y coordinate, which varies on whether or not the label has a parent, a reference to the parent object, the text the label displays, the width of the text, a reference to the Label Set it belongs to, and Boolean properties that determine whether it is currently selected or highlighted.

The classes Vertex Set, Edge Set and Label Set, are collection classes to hold sets of objects. In JavaScript they are implemented as arrays. It is important to note that JavaScript provides an object that has array-like qualities, is significantly slower than a real array, but more convenient to use. They work more like dynamic lists which can grow and shrink in size which makes the adding and removal of Graph objects easy to do.

The Graph class uses a delegation model. All manipulation of an object is done by functions on the object. The graph propagates most function requests to its member objects. One example of this is hit testing. Various actions require the user to click on a Vertex, Edge, or Label object to select the object. The Controller takes the X-Y coordinate of the mouse event and passes it to the Graph Model. The Graph then queries each of its sets, which in turn query their objects to determine if the X-Y coordinate resides within an object. If so, the object is returned to the Controller.

5.2 The View

GRAPH-ENE's View is a webpage consists of a graphical user interface complete with buttons (previously discussed in Section 3 of this paper) and a drawing canvas. In short, the view is what the user *sees* and interacts with. CSS, or Cascading Style sheets, and the Bootstrap framework are used to assist in giving the webpage a more stylish look.

The drawing canvas is the HTML5 canvas element, which generates the output or visual representation of a graph to the user. By definition, the <canvas> element in HTML 5 is, "…a resolution-dependent bitmap canvas which can be used for rendering graphs, game graphics, or other visual images on the fly. A canvas is a rectangle in your page where you can use JavaScript to draw anything you want" (diveintohtml5). The canvas is a two-dimensional grid. Coordinate (0, 0) is at the upper-left corner of the canvas. Along the X-axis, values increase towards the right edge of the canvas. Along the Y-axis, values increase towards the bottom edge of the canvas.

The canvas provides a context which is necessary in order to use the methods and properties to render drawings on the canvas. These include methods for drawing arcs, rectangles, paths, or selecting line thickness, fill styles, or stroke styles. In addition, HTML5 does allow the layering of canvases for performance increase, a technique known as double buffering, but we opted to use a single canvas element since its fast enough on the draw routine.

5.3 The Controller

The Controller gathers input from the View and converts it to commands to update the model or the view if necessary. The Controller has a state machine that monitors what events occur on both the webpage and the canvas. The state machine has 4 states it can be in. These states are: “Build”, “Highlight”, “Select”, and “Label”. When an event occurs, the Controller looks at the state and determines the appropriate course of action. The Controller manipulates the Graph Model by calling appropriate functions on the graph object to update and redraw the graph. The Controller also updates the buttons on the View to reflect the current system state.

The Controller uses JavaScript Object Notation (JSON), to upload and download the graph data. JSON serializes JavaScript objects as human-readable text to transmit them over a network. The only JSON functions needed are stringify to serialize the data and parse to restore the data. Since JSON does not validate the data, a validation function needs to be developed to handle that task. Furthermore, Internet Explorer does not allow client-side data to be downloaded directly from the browser. Instead, the data needs to be transmitted via a server, or features of the Adobe Flash plugin must be exploited to perform the download.

6 Future Work

Additional features were discussed and considered—one feature being an undo/redo button, which could be an entire project on its own due to the complexity it would have based on our current system. Another feature was having a side bar that would provide more detailed information and properties of a selected graph object. We are also looking at incorporating the capability to draw curved lines, multi-edges and loops.

The User Interface needs future research to determine what is effective and what is not effective, which may require testing in a classroom and retrieving feedback from teachers and students. Also, after running some tests on a classroom projector, we came across display resolution issues which sparked the idea of having a desktop mode and a presentation mode for GRAPH-ENE in order to alleviate issues dealing with different resolution displays.

The list of ideas is growing, but below are a few things to be added/tweaked:

- Auto-labeling.
- Attributes such as size and line style.
- Allow the 'New' button to produce common graph types.
- Animation has been prototyped and must be made more robust.
- Code refactoring.
- Ability to verify data integrity for saved files.

7 Conclusion

GRAPH-ENE was a collaborative effort with previous research of the project advisor Dr. Ravenscroft. We were able to successfully implement a web-based graphing tool. GRAPH-ENE provides users the means to build graphs using vertices edges and labels, incorporates working features such as highlighting, coloring, deletion and dragging, and allows for the loading, saving and printing of graphs. GRAPH-ENE can be used as a learning tool by students and a classroom demonstration tool by faculty. Its capabilities allow it to be used for many different graphing problems.

The web is an ever evolving and rapidly changing environment. Developing true cross-platform applications is a time-consuming challenge. GRAPH-ENE is currently working on Firefox and Google Chrome. Implementation of HTML 5 varies across platforms, particularly in Internet Explorer, which is lacking in some features needed to fully implement GRAPH-ENE at this time. As HTML 5 matures and browser technologies evolve, we should be able to implement a more powerful and robust tool across all platforms.

GRAPH-ENE was developed to be used. GRAPH-ENE will remain at Rhode Island College for future development and classroom use

Appendix A – Source Code

A-1 GRAPH-ENE.html

```
<!DOCTYPE HTML>
<html>
  <head>
          <meta charset="UTF-8"/>
    <html lang="en">
          <title>GRAPH-ENE: A Rich Internet Application for Graph Manipulation</title>
          <link rel="stylesheet" href="css/canvasStyle.css">
    <link rel="stylesheet" href="css/bootstrap.css">
    <!--<script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.0/jquery.min.js"></script>-->
    <script type="text/javascript" src="jscript/jquery-1.11.0.min.js"></script>
    <script type="text/javascript" src="jscript/Graph.js"></script>
          <script type="text/javascript" src="jscript/Controller.js"></script>
    <script type="text/javascript" src="jscript/bootstrap.js"></script>
  </head>
  <body>
      <div class="panel panel-primary">
        <div class="panel-heading">
          <h1 id="headertitle" class="panel-title"><strong>GRAPH-ENE</strong></h1>
        </div>

        <div class="panel-body">
                             <!--- Main Menu --->
          <div class="btn-group-horizontal" id="defaultMenu">
            <button id="build"  class="btn btn-default" type="button">Build</button>
            <button id="label"  class="btn btn-default" type="button">Add Label</button>
            <button id="hlight" class="btn btn-default" type="button">Highlight</button>
            <button id="clear"  class="btn btn-default" type="button">Clear Canvas</button>
            <button id="select" class="btn btn-default" type="button">Select</button>
            <!--<button id="clearH" class="btn btn-default" type="button">Clear all
highlights</button>-->
          </div>
                             <!--- Vertex Menu --->
          <div class="btn-group-horizontal" id="vertexMenu">
            <button id="vtxdelete" class="btn btn-primary" type="button">Delete
Vertex</button>
            <button id="vtxlabel" class="btn btn-primary" type="button">Label Vertex</button>
          </div>
                             <!--- Edge Menu --->
          <div class="btn-group-horizontal" id="edgeMenu">
```

```
              <button id="edgedelete" class="btn btn-primary" type="button">Delete
Edge</button>
              <button id="edgelabel" class="btn btn-primary" type="button">Label Edge</button>
            </div>
                                <!--- Label Menu --->
            <div class="btn-group-horizontal" id="labelMenu">
               <button id="labeldelete" class="btn btn-primary" type="button">Delete
Label</button>
               <button id="labelchange" class="btn btn-primary" type="button">Change Label
Text</button>
            </div>

          <div class="btn-group-horizontal" id="saveloadMenu">
              <button id="newgraph" class="btn btn-default" type="button">New Graph</button>
              <button id="download" class="btn btn-default" type="button">Save Graph</button>
                                 <!--<input type="file" id="Upload" name="file" />-->
              <span class="btn btn-default btn-file">
                 Load Graph
                 <input type="file" id="Upload" name="file"/>
              </span>
          </div>
        </div>
      </div>
                                <!--- Color Menu --->
     <div id="color-toolbar" class="btn-toolbar">
       <button id="color-swatch-blue" class="btn btn-default color-swatch"
type="button"></button>
       <button id="color-swatch-red" class="btn btn-default color-swatch"></button>
                  <button id="color-swatch-yellow" class="btn btn-default color-
swatch"></button>
       <button id="color-swatch-green" class="btn btn-default color-swatch"></button>
       <button id="color-swatch-orange" class="btn btn-default color-swatch"></button>
                  <button id="color-swatch-pink" class="btn btn-default color-swatch"></button>
       <button id="color-swatch-black" class="btn btn-default color-swatch"></button>
     </div>

     <div id="canvasDIV">
       <canvas id="gCanvas" height="450" width="900">Your current browser does not support
Canvas! Use Firefox or Google Chrome.</canvas>
     </div>

  </body>
</html>
```

A-2 Graph.js

```
/*jslint browser: true*/
/*jslint devel: true */


/*****************************************
            Graph.JS

Holds all the functions for creating & drawing the parts of a Graph object. These
include vertices, edges & labels, each with a draw routine for highlighting,
selecting & rendering the object onto the HTML 5 canvas.

NOTE: This is the most recent version as of April 27th 2014.
*****************************************/

//(function () { JS Lint stuff to make code pretty.
   //"use strict";

/*****************************************
Name: Label(lSet, name, x, y, parent, ctxt)
Description: Creates a Label object.
Parameter(s): 'name' refers to the Label name, 'x' and 'y' refer to its position.
            'parent' is a reference to the object that a Label object is
            attached to. 'ctxt'is a reference to the object that provides
            methods & properties for drawing on the canvas.
*****************************************/
function Label(lSet, name, x, y, parent, ctxt) {
   if (parent) { // tests whether Label has parent object.
      this.x = 10; // place Label 10 horizontal pixels from parent.
      this.y = 10; // place Label 10 vertical pixels from parent.
   } else { // Label with no parent.
      this.x = x;
      this.y = y;
   }

   this.name = name;
   this.parent = parent;
   this.color = "black";
   this.hColor = "#FFFF6D";
   this.lSet = lSet;
   this.highlighted = false;
   this.selected = false;
   this.textWidth = ctxt.measureText(this.name).width + 20;
}
```

```
/******************************************
Name: Label.copy(label, lSet)
Description: Copies a label object or initializes using label fields.
Parameters:
            label: label raw data to copy to label object
             lSet: labelSet object that will contain the new label
*******************************************/
Label.copy = function (label, lSet) {
        this.x = label.x;
        this.y = label.y;
        this.name = label.name;
   this.parent = null;     // parent will set this
        this.color = label.color;
        this.highlighted = label.highlighted;
        this.hColor = label.hColor;
        this.textWidth = label.textWidth;
        this.lSet = lSet;
        this.select = false;
};

Label.copy.prototype = Label.prototype;

/******************************************
Name: label.prototype.saveData()
Description: Creates a data set for downloading and saving
Parameters: N/A.
*******************************************/
Label.prototype.saveData = function () {
        var data = {};
        data.x = this.x;
        data.y = this.y;
        data.name = this.name;
        data.parent = null;  // need field, not value
        data.color = this.color;
        data.highlighted = this.highlighted;
        data.hColor = this.hColor;
        data.textWidth = this.textWidth;
        return data;
};

/******************************************
Name: Label.prototype.setParent(p)
Description: sets parent for label
Parameters: 'p' is the parent that the label is set to have.
*******************************************/
```

```
Label.prototype.setParent = function (p) {
        this.parent = p;
};

/*****************************************
Name: Label.prototype.getIndex()
Description: Gets a label's index position in the lList
Parameter(s):  N/A.
*****************************************/
Label.prototype.getIndex = function () {
        return this.lSet.getIndex(this);
};

/*****************************************
Name:Label.prototype.remove
Description: Removes a label from the list of labels.
Parameter(s): N/A.
*****************************************/
Label.prototype.remove = function () {
   if (this.parent) { // if this label has a parent, let parent know its null.
      this.parent.Label = null;
   }
   this.lSet.remove(this);
};

/*****************************************
Name: Label.prototype.move
Description: Moves a label to the specified location.
Parameter(s): 'dx' & 'dy' refer to the amount of pixels
            to move this Label by.
*****************************************/
Label.prototype.move = function (dx, dy) {
   this.x += dx;
   this.y += dy;
};

/*****************************************
Name: Label.prototype.getText
Description: Returns a label's text.
Parameter(s): N/A
*****************************************/
Label.prototype.getText = function () {
   return this.name;
};

/*****************************************
```

```
Name: Label.prototype.changeText
Description: Changes the text of an existing label.
Parameter(s): 'txt' refers to the new text this Label will now display.
*******************************************/
Label.prototype.changeText = function (txt) {
   this.name = txt;
};

/*******************************************
Name: Label.prototype.setColor
Description:
Parameter(s): 'colorChosen' refers to the color the user specified.
*******************************************/
Label.prototype.setColor = function (colorChosen) {
   this.color = colorChosen;
};

/*******************************************
Name: Label.prototype.setHlight
Description: Sets a highlight color to a Label.
Parameter(s): "colorChosen" is the color chosen by the end-user. It will consist
            of an RGB value passed from the Controller class.
*******************************************/
Label.prototype.setHlight = function (colorChosen) {
   this.hColor = colorChosen;
};

/*******************************************
Name: Label.prototype.toggleHighlight
Description: Toggles highlighting on a Label.
Parameter(s): N/A.
*******************************************/
Label.prototype.toggleHighlight = function (clr) {
   if (this.highlighted === true &&  this.hColor === clr) {
      this.highlighted = !this.highlighted;
   } else {
      this.highlighted = true;
      this.hColor = clr;
   }
};

/*******************************************
Name: Label.prototype.toggleSelect
Description: Toggles select pattern on and off on label.
Parameter(s): N/A.
*******************************************/
```

```
Label.prototype.toggleSelect = function () {
   this.selected = !this.selected;
};

/*****************************************
Name: Label.prototype.draw
Description: Draws the Label onto the canvas.
Parameter(s): 'ctxt'is a reference to the object that provides
            methods & properties for drawing on the canvas.
*****************************************/
Label.prototype.draw = function (ctxt) {
   var loc = {x: 0, y: 0};

   if (this.parent) {
      loc = this.parent.getTextLocation();
   }

   ctxt.font = "bold 14px Arial";
   ctxt.textBaseline = "top";
   ctxt.fillStyle = this.color; // this must be done first before filltext; otherwise color wont work.
   ctxt.fillText(this.name, (loc.x + this.x), (loc.y + this.y));
};

/*****************************************
Name: Label.prototype.drawHighlight
Description: If highlighted property is true, draws the Highlight onto this Label.
Parameter(s): 'ctxt'is a reference to the object that provides
            methods & properties for drawing on the canvas.
*****************************************/
Label.prototype.drawHighlight = function (ctxt) {
   var loc, pattern;

   if (!this.highlighted) {
      return;
   }

   loc = {x: 0, y: 0};

   if (this.parent) {
      loc = this.parent.getTextLocation();
   }

   //pattern = Label.hColor.getPattern(this.hColor);
   ctxt.save();

   ctxt.strokeStyle = this.hColor;
   loc.x += this.x;
   loc.y += this.y;

   ctxt.beginPath();
   ctxt.lineCap = "round";

   ctxt.moveTo(loc.x, loc.y);
   ctxt.lineTo(loc.x + this.textWidth, loc.y + 5);
   ctxt.lineTo(loc.x, loc.y + 10);
   ctxt.lineTo(loc.x + this.textWidth, loc.y + 14);

   ctxt.stroke();
   ctxt.restore();
};

/*****************************************
Name: Label.prototype.drawSelect
Description: If selected property is true, draws the select pattern onto this Label.
Parameter(s): 'ctxt'is a reference to the object that provides
              methods & properties for drawing on the canvas.
*****************************************/
Label.prototype.drawSelect = function (ctxt) {
   if (!this.selected) { return; }

   var loc = {x: 0, y: 0};

   if (this.parent) {
      loc = this.parent.getTextLocation();
   }

   loc.x += this.x;
   loc.y += this.y;
   //ctxt.strokeStyle = selectPattern;
   ctxt.lineWidth = 8;

   ctxt.beginPath();

   ctxt.moveTo(loc.x, loc.y);
   ctxt.lineTo(loc.x + this.textWidth, loc.y);
   ctxt.lineTo(loc.x, loc.y + 10);
   ctxt.lineTo(loc.x + this.textWidth, loc.y + 10);

   ctxt.stroke();
};
```

```
/****************************************
Name: Label.prototype.hit
Description: Determines if a hit was made on this Label.
Parameter(s): 'x' & 'y' refer to the mouse event position;
        for 'ctxt', see note at top of document.
****************************************/
Label.prototype.hit = function (x, y, ctxt) {
  var dy, loc;

  loc = {x: 0, y: 0};

  if (this.parent) {
    loc = this.parent.getTextLocation();
  }

  loc.x += this.x;
  loc.y += this.y;

  //this.txtwidth = ctxt.measureText(this.name).width;
  dy = y - loc.y;
  return (loc.x <= x && x <= loc.x + this.textWidth) && (0 <= dy && dy <= 15);
};

/****************************************
Name: LabelSet
Description: Creates an empty array that will hold Label objects added to the canvas.
Parameter(s): N/A.
****************************************/
function LabelSet() {
  this.lList = [];
}

/****************************************
Name: LabelSet.copy(lSet)
Description: Copies a label list and it's Label members
Parameters: 'lSet' refers to raw LabelSet data to copy to LabelSet object.
****************************************/
LabelSet.copy = function (lSet) {
  var me = this;
  this.lList = lSet.lList.map( function(lbl) { return new Label.copy(lbl, me); } );
}

LabelSet.copy.prototype = LabelSet.prototype;

/****************************************
Name: LabelSet.prototype.saveData()
```

```
Description: Creates a data set for downloading/saving and returns it.
Parameters: N/A.
******************************************/
LabelSet.prototype.saveData = function(){
        return { lList: this.lList.map( function (l) {console.log(l);return l.saveData(); } ) };
};

/******************************************
Name: LabelSet.prototype.getIndex()
Description: gets index for a label in the labelSet
Parameters: label : label to locate
******************************************/
LabelSet.prototype.getIndex = function (label) {
        return this.lList.indexOf(label);
};

/******************************************
Name: LabelSet.prototype.add
Description: Adds a Label to the list of labels.
Parameter(s): 'name' represents the Label name; 'x' & 'y' represent the
            coordinates of the Vertex; 'parent' is a reference to the
            object that this label is attached to; 'ctxt'is a reference to
            the object that provides methods & properties for drawing on the canvas.
******************************************/
LabelSet.prototype.add = function (name, x, y, parent, ctxt) {
   this.lList.push(new Label(this, name, x, y, parent, ctxt));
};

/******************************************
Name: LabelSet.prototype.remove
Description: Removes a label from the list of labels.
Parameter(s): 'aLabel' refers to the label passed in to be removed.
******************************************/
LabelSet.prototype.remove = function (aLabel) {
   var i = this.lList.indexOf(aLabel);

   if (i >= 0) {
      this.lList.splice(i, 1);
   }
};

/******************************************
Name: LabelSet.prototype.get
Description: Returns the Label at specified index.
Parameter(s): 'i' refers to the index of the Label in the array.
******************************************/
```

```
LabelSet.prototype.get = function (i) {
   return this.lList[i];
};

/*****************************************
Name: LabelSet.prototype.getSize
Description: Returns the size of the Label list.
Parameter(s): N/A.
*****************************************/
LabelSet.prototype.getSize = function () {
   return this.lList.length;
};

/*****************************************
Name: LabelSet.prototype.draw
Description: Draws Labels onto the canvas.
Parameter(s): 'ctxt'is a reference to the object that provides
            methods & properties for drawing on the canvas.
*****************************************/
LabelSet.prototype.draw = function (ctxt) {
   var i;
   for (i = 0; i < this.lList.length; i += 1) {
      this.lList[i].draw(ctxt);
   }
};

/*****************************************
Name: LabelSet.prototype.drawHighlight
Description: Calls drawHighlight on each Label in the list.
Parameter(s): 'ctxt'is a reference to the object that provides
            methods & properties for drawing on the canvas.
*****************************************/
LabelSet.prototype.drawHighlight = function (ctxt) {
   var i;
   for (i = 0; i < this.lList.length; i += 1) {
      this.lList[i].drawHighlight(ctxt);
   }
};

/*****************************************
Name: LabelSet.prototype.drawSelect
Description: Calls drawSelect on each Label in the list.
Parameter(s): 'ctxt'is a reference to the object that provides
            methods & properties for drawing on the canvas.
*****************************************/
LabelSet.prototype.drawSelect = function (ctxt) {
    var i;
    for (i = 0; i < this.lList.length; i += 1) {
        this.lList[i].drawSelect(ctxt);
    }
};

/*****************************************
Name: LabelSet.prototype.hit
Description: Calls the hit method on each Label in the list.
Parameter(s): 'x' and 'y' refer to where the mouse event occurred.
            'ctxt'is a reference to the object that provides
            methods & properties for drawing on the canvas.
*****************************************/
LabelSet.prototype.hit = function (x, y, ctxt) {
    var i;
    for (i = 0; i < this.lList.length; i += 1) {
        if (this.lList[i].hit(x, y, ctxt)) {
            return {object: this.lList[i], index: i};
        }
    }
    return null;
};

/*****************************************
Name: Vertex
Description: Creates a Vertex object.
Parameter(s): 'vSet' is a reference to the VertexSet; 'x' and 'y' refer to the
            position of the Vertex.
*****************************************/
function Vertex(vSet, x, y) {
    this.x = x;
    this.y = y;
    this.Label = null;
    this.color = "black";
    this.hColor = null;
    this.incList = [];
    this.vSet = vSet;
    this.selected = false;
    this.highlighted = false;
}

/*****************************************
Name: Vertex.copy(vertex, vSet, lSet)
Description: copies a vertex object or initializes using vertex fields
Parameters:
        vertex: Vertex raw data to copy to vertex object
```

```
	vSet: VertexSet that will contain the new vertex
	lSet: LabelSet that contains the labels
*****************************************/
Vertex.copy = function(vertex, vSet, lSet) {
	this.x = vertex.x;
	this.y = vertex.y;
	this.Label = null;
	this.color = vertex.color;
	this.hover = vertex.hover;
	this.incList = [];
	this.vSet = vSet;
	this.selected = false;
	this.highlighted = vertex.highlighted;
	this.hColor = vertex.hColor;
	if ("label" in vertex) { this.Label = lSet.get(vertex.label); this.Label.setParent(this); }
};

Vertex.copy.prototype = Vertex.prototype;

/*****************************************
Name: Vertex.prototype.saveData()
Description: Ceates a data set for downloading and saving
Parameters: N/A.
*****************************************/
Vertex.prototype.saveData = function(){
	var data = {};
	data.x = this.x;
	data.y = this.y;
	if ( this.Label ) {
		data.label = this.Label.getIndex();
	}
	data.color = this.color;
	data.hover = this.hover;
	data.highlighted = this.highlighted;
	data.hColor = this.hColor;
	return data;
};

/*****************************************
Name: Vertex.prototype.getIndex()
Description: Returns the index of the Vertex using this method.
Parameters:
*****************************************/
Vertex.prototype.getIndex = function(){
	return this.vSet.vList.indexOf(this);
};
```

```
/*****************************************
Name: Vertex.prototype.remove
Description: Removes a Vertex.
Parameter(s): N/A.
*****************************************/
Vertex.prototype.remove = function () {
   var Edge, temp;
   temp = this.incList.slice(0);

   for (Edge in temp) {
      if (temp.hasOwnProperty(Edge)) {
         temp[Edge].remove();
      }
   }

   if (this.Label) { // If this Vertex has a label, remove.
      this.Label.remove();
   }

   this.vSet.remove(this); // Remove 'this' Vertex from Vertex set.
};

/*****************************************
Name: Vertex.prototype.setLabel
Description: Sets a Label onto a Vertex object.
Parameter(s): 'aLabel' refers to the Label object this Vertex will be a parent of.
*****************************************/
Vertex.prototype.setLabel = function (aLabel) {
   this.Label = aLabel;
};

/*****************************************
Name: Vertex.prototype.getLabel
Description: Returns the label object of a Vertex
Parameter(s): N/A.
*****************************************/
Vertex.prototype.getLabel = function () {
   return this.Label;
};

/*****************************************
Name: Vertex.prototype.addAdjacent
Description: Adds specified Edge to incident list of this Vertex.
Parameter(s): 'anEdge' is the Edge being pushed into the incident list.
*****************************************/
```

```
Vertex.prototype.addAdjacent = function (anEdge) {
   this.incList.push(anEdge);
};

/*****************************************
Name: Vertex.prototype.removeAdjEdge
Description: Removes specified Edge from this Vertex incident list.
Parameter(s): 'anEdge' is the Edge to be removed from incident list.
*****************************************/
Vertex.prototype.removeAdjEdge = function (anEdge) {
   var i = this.incList.indexOf(anEdge);
   if (i >= 0) {
      this.incList.splice(i, 1);
   }
};

/*****************************************
Name: Vertex.prototype.setColor
Description: Sets a color to a Vertex; default is black.
Parameter(s): "colorChosen" is the color chosen by the end-user. It will consist
          of an RGB value passed from the Controller class.
*****************************************/
Vertex.prototype.setColor = function (colorChosen) {
   this.color = colorChosen;
};

/*****************************************
Name: Vertex.prototype.setHlight
Description: Sets a highlight color to a Vertex.
Parameter(s): "colorChosen" is the color chosen by the end-user. It will consist
          of an RGB value passed from the Controller class.
*****************************************/
Vertex.prototype.setHlight = function (colorChosen) {
   this.hColor = colorChosen;
};

/*****************************************
Name: Vertex.prototype.draw
Description: Draws the Vertex on the canvas.
Parameter(s): 'ctxt'is a reference to the object that provides methods &
           properties for drawing on the canvas.
*****************************************/
Vertex.prototype.draw = function (ctxt) {
   ctxt.beginPath();
   ctxt.arc(this.x, this.y, 5, 0, 2 * Math.PI);
   ctxt.fillStyle = this.color;
```

```
  ctxt.strokeStyle = this.color;

  ctxt.fill();
  ctxt.stroke();
};

/***********************************************
Name: Vertex.prototype.drawHighlight
Description: If Vertex highlight property is true, draws a highlight over that Vertex object.
Parameter(s): 'ctxt'is a reference to the object that provides methods &
              properties for drawing on the canvas.
***********************************************/
Vertex.prototype.drawHighlight = function (ctxt) {
  if (!this.highlighted) {
    return;
  }

        //var pattern = Vertex.hColor.getPattern(this.hColor);
  var pattern = this.hColor;
        ctxt.save();

        ctxt.fillStyle = pattern;
        ctxt.strokeStyle = pattern;

        ctxt.beginPath();
  ctxt.arc(this.x, this.y, 5, 0, 2 * Math.PI);
  //ctxt.fillStyle = this.hColor;
  //ctxt.strokeStyle = this.hColor;
  ctxt.fill();
  ctxt.stroke();

        ctxt.restore();
};

/****************************************
Name: Vertex.prototype.drawSelect
Description: If selected property is true, draws select pattern on this Vertex.
Parameter(s): 'ctxt'is a reference to the object that provides methods &
              properties for drawing on the canvas.
****************************************/
Vertex.prototype.drawSelect = function (ctxt) {
  if (!this.selected) { return; }

  //ctxt.fillStyle = selectPattern;
  //ctxt.strokeStyle = selectPattern;
  ctxt.lineWidth = 10;
```

```
    ctxt.beginPath();
    ctxt.moveTo(this.x + 10, this.y);
    ctxt.arc(this.x, this.y, 10, 0, 2 * Math.PI, false);
    ctxt.closePath();

    ctxt.fill();
    ctxt.stroke();
};

/*****************************************
Name: Vertex.prototype.toggleHighlight
Description:
Parameter(s): N/A.
*****************************************/
Vertex.prototype.toggleHighlight = function (clr) {
    if (this.highlighted === true && this.hColor === clr) {
        this.highlighted = !this.highlighted
    } else {
        this.highlighted = true;
        this.hColor = clr;
    }
};

/*****************************************
Name: Vertex.prototype.toggleSelect
Description: Toggles select pattern on and off on Vertex.
Parameter(s): None.
*****************************************/
Vertex.prototype.toggleSelect = function () {
    this.selected = !this.selected;
};

/*****************************************
Name: Vertex.prototype.getTextLocation
Description: Returns the text location of this Vertex.
Parameter(s): N/A.
*****************************************/
Vertex.prototype.getTextLocation = function () {
    return {x: this.x, y: this.y};
};

/*****************************************
Name: Vertex.prototype.hit
Description:
Parameter(s): 'x' & 'y' are the mouse-event coordinates used to test a hit.
```

```
*****************************************/
Vertex.prototype.hit = function (x, y) {
   var dx, dy;

   dx = this.x - x;
   dy = this.y - y;

   return (dx * dx + dy * dy) <= 10 * 10;
};

Vertex.prototype.move = function (dx, dy) {
   this.x += dx;
   this.y += dy;
};

/*****************************************
Name: VertexSet
Description: Creates an empty list object that will hold the vertices of the Graph object.
Parameter(s): None.
*****************************************/
function VertexSet() {
   this.vList = [];
}

/*****************************************
Name: VertexSet.copy(vSet, lSet)
Description: Copies a VertexSet and its Vertex members
Parameters:
         vSet: VertexSet raw data to copy to new VertexSet object
         lSet: LabelSet with Labels for Vertices
*****************************************/
VertexSet.copy = function(vSet, lSet){
      var me = this;
      this.vList = vSet.vList.map( function(vrtx){ return new Vertex.copy(vrtx, me, lSet); } );
}

VertexSet.copy.prototype = VertexSet.prototype;

/*****************************************
Name: VertexSet.prototype.saveData()
Description: Creates a data set for downloading and saving
Parameters:
*****************************************/
VertexSet.prototype.saveData = function(){
         return { vList: this.vList.map( function (v) { return v.saveData(); } ) };
};
```

```
/*****************************************
Name: VertexSet.prototype.remove
Description: Removes specified Vertex from the Vertex list.
Parameter(s): 'aVertex' is the Vertex to be removed.
******************************************/
VertexSet.prototype.remove = function (aVertex) {
   var i = this.vList.indexOf(aVertex);

   if (i >= 0) {
      this.vList.splice(i, 1);
   }
};

/*****************************************
Name: VertexSet.prototype.add
Description: Adds a Vertex to the array of vertices.
Parameter(s): 'x' & 'y' represent the coordinates of the Vertex.
******************************************/
VertexSet.prototype.add = function (x, y) {
   this.vList.push(new Vertex(this, x, y));
};

/*****************************************
Name: VertexSet.prototype.get
Description: Returns a Vertex from specified index
Parameter(s): 'i' refers to the subscript of the array object.
******************************************/
VertexSet.prototype.get = function (i) {
   return this.vList[i];
};

/*****************************************
Name: VertexSet.prototype.getSize
Description: Returns the size of the Vertex list.
Parameter(s): N/A.
******************************************/

VertexSet.prototype.getSize = function () {
   return this.vList.length;
};

/*****************************************
Name: VertexSet.prototype.draw
Description: Calls the draw routine on each Vertex in the set.
Parameter(s): 'ctxt'is a reference to the object that provides methods &
```

```
            properties for drawing on the canvas.
******************************************/
VertexSet.prototype.draw = function (ctxt) {
   var i;
   for (i = 0; i < this.vList.length; i += 1) {
      this.vList[i].draw(ctxt);
   }
};

/******************************************
Name: VertexSet.prototype.drawHighlight
Description: Calls the highlight routine on each Vertex in the set.
Parameter(s): 'ctxt'is a reference to the object that provides methods &
            properties for drawing on the canvas.
******************************************/
VertexSet.prototype.drawHighlight = function (ctxt) {
   var i;

   for (i = 0; i < this.vList.length; i += 1) {
      this.vList[i].drawHighlight(ctxt);
   }
};

/******************************************
Name: VertexSet.prototype.clearHlight
Description: Calls clearHlight on all Vertices.
Parameter(s): N/A.

******************************************/
VertexSet.prototype.clearHlight = function () {

};

/******************************************
Name: VertexSet.prototype.drawSelect
Description: Calls the drawSelect function on all vertices in the list.
Parameter(s): 'ctxt'is a reference to the object that provides methods &
            properties for drawing on the canvas.
******************************************/
VertexSet.prototype.drawSelect = function (ctxt) {
   var i;

   for (i = 0; i < this.vList.length; i += 1) {
      this.vList[i].drawSelect(ctxt);
   }
};
```

```
/*****************************************
Name: VertexSet.prototype.hit
Description:  Tests for a hit on a vertex. Returns Vertex & its index in the
              list if successful.
Parameter(s): 'x' & 'y' refer to coordinates of the mouse event.
*****************************************/
VertexSet.prototype.hit = function (x, y) {
   var i;

   for (i = 0; i < this.vList.length; i += 1) {
      if (this.vList[i].hit(x, y)) {
         return {object: this.vList[i], index: i};
      }
   }
   return null;
};

/*****************************************
Name: Edge
Description: Creates an Edge object.
Parameter(s): 'v1' & 'v2' are vertices on which the Edge is connected to.
*****************************************/
function Edge(eSet, v1, v2) {
   this.p = v1;
   this.q = v2;
   this.Label = null;
   this.eSet = eSet;
   this.color = "black"; // Black is default color.
   this.hColor = null;
   this.selected = false;
   this.highlighted = false;
   //v1.addAdjacent(this);
   //v2.addAdjacent(this);
   this.p.addAdjacent(this);
   this.q.addAdjacent(this);
}

/*****************************************
Name: Edge.copy(edge, eSet, vSet, lSet)
Description: copies an edge object or initializes using edge fields
Parameters:
         edge: Edge raw data to copy to new edge object
         eSet: EdgeSet that will contain the new vertex
         vSet: VertexSet that contains the vertices
         lSet: LabelSet that contains the labels
```

```
*****************************************/
Edge.copy = function(edge, eSet, vSet, lSet){
        this.p = vSet.get(edge.p);
        this.q = vSet.get(edge.q);
        this.Label = null;
        this.eSet = eSet;
        this.color = edge.color;
        this.selected = false;
        this.highlighted = edge.highlighted;
        this.hColor = edge.hColor;
        this.p.addAdjacent(this);
        this.q.addAdjacent(this);
  console.log(edge);
        if (edge.label) { this.Label = lSet.get(edge.label); this.Label.setParent(this); }
}

Edge.copy.prototype = Edge.prototype;

/****************************************
Name: edge.prototype.saveData()
Description: creates a data set for downloading and saving
Parameters:
*****************************************/
Edge.prototype.saveData = function(){
        var data = {};
        data.p = this.p.getIndex();
        data.q = this.q.getIndex();
        if ( this.Label ) {
                data.label = this.Label.getIndex();
        }
        data.color = this.color;
        data.highlighted = this.highlighted;
        data.hColor = this.hColor;
        return data;
}

/****************************************
Name: Edge.prototype.remove
Description: Removes this edge from the edge set & its adjacent vertices
Parameter(s): N/A.
*****************************************/
Edge.prototype.remove = function () {
  this.p.removeAdjEdge(this);
  this.q.removeAdjEdge(this);
  this.eSet.remove(this);
};
```

```
/****************************************
Name: Edge.prototype.setLabel
Description: Sets a Label onto an Edge
Parameter(s): 'aLabel' refers to the Label object this Edge will be a parent of.
*****************************************/
Edge.prototype.setLabel = function (aLabel) {
   this.Label = aLabel;
};

/****************************************
Name: Edge.prototype.getLabel
Description: Returns the Label object of this Edge.
Parameter(s): N/A.
*****************************************/
Edge.prototype.getLabel = function () {
   if (this.Label) {
      return this.Label;
   } else {
      return null;
   }
};

/****************************************
Name: Edge.prototype.draw
Description: Draws an Edge onto the canvas.
Parameter(s): 'ctxt' refers to the drawing context of the canvas.
*****************************************/
Edge.prototype.draw = function (ctxt) {
   ctxt.beginPath();
   ctxt.moveTo(this.p.x, this.p.y);
   ctxt.lineTo(this.q.x, this.q.y);
   ctxt.lineWidth = 3.5;
   ctxt.fillStyle = this.color;
   ctxt.strokeStyle = this.color;
   ctxt.stroke();
};

/****************************************
Name: Edge.prototype.drawHighlight
Description: Draws the highlight over the edges.
Parameter(s): 'ctxt' refers to the drawing context of the canvas.
*****************************************/
Edge.prototype.drawHighlight = function (ctxt) {
   var pattern;
   if (!this.highlighted) {
      return;
   }

   //pattern = Edge.hColor.getPattern(this.hColor);

   ctxt.save();

   //ctxt.lineWidth = 12;
   ctxt.fillStyle = this.hColor;
   ctxt.strokeStyle = this.hColor;

   ctxt.beginPath();
   ctxt.moveTo(this.p.x, this.p.y);
   ctxt.lineTo(this.q.x, this.q.y);

   ctxt.fill();
   ctxt.stroke();

   ctxt.restore();
};

/****************************************
Name: Edge.prototype.drawSelect
Description: If selected property is true, draws the select pattern onto this Edge.
Parameter(s): 'ctxt' is a reference to the drawing context of the canvas.
****************************************/
Edge.prototype.drawSelect = function (ctxt) {
   if (!this.selected) {
      return;
   }

   ctxt.lineWidth = 12;

   ctxt.beginPath();
   ctxt.moveTo(this.p.x, this.p.y);
   ctxt.lineTo(this.q.x, this.q.y);

   ctxt.fill();
   ctxt.stroke();
};

/****************************************
Name: Edge.prototype.getTextLocation
Description: Returns location where an Edge's label will be placed.
Parameter(s): N/A.
```

```
*******************************************/
Edge.prototype.getTextLocation = function () {
   return {x: (this.p.x + this.q.x) / 2, y: (this.p.y + this.q.y) / 2};
};

/*******************************************
Name: Edge.prototype.toggleHighlight
Description: Toggles on & off the highlighting of an Edge.
Parameters: N/A.
*******************************************/
Edge.prototype.toggleHighlight = function (clr) {
   if (this.highlighted === true && this.hColor === clr) {
      this.highlighted = !this.highlighted
   } else {
      this.highlighted = true;
      this.hColor = clr;
   }
};

/*******************************************
Name: Edge.prototype.toggleSelect
Description: Toggles select pattern on and off on an Edge.
Parameter(s): None.
*******************************************/
Edge.prototype.toggleSelect = function () {
   this.selected = !this.selected;
};

/*******************************************
Name: Edge.prototype.hit
Description:
Parameters: 'x' & 'y' are coordinates to test against for a hit on the Edge.
*******************************************/
Edge.prototype.hit = function (x, y) {
   var dx, dy, ddx, ddy, C, D, denom;

   dx = this.q.x - this.p.x;
   dy = this.q.y - this.p.y;
   denom = Math.sqrt((dx * dx) + (dy * dy));

   ddx = 10 * dy / denom;
   ddy = -10 * dx / denom;

   denom = ddx * dy - dx * ddy;

   C = ((this.p.x - x) * ddy + (y - this.p.y) * ddx) / denom;
   D = ((this.p.x - x) * dy + (y - this.p.y) * dx) / denom;

   return 0 <= C && C <= 1 && -1 <= D && D <= 1;
};

/*************************************
Name: Edge.prototype.move
Description: Moves an edge to a new location.
Parameter(s): 'dx' & 'dy' refer to the mouse event coordinates to move the edge.
*************************************/
Edge.prototype.move = function (dx, dy) {
   this.p.move(dx, dy);
   this.q.move(dx, dy);
};

/****************************************
Name: Edge.prototype.setColor
Description:
Parameter(s):
****************************************/
Edge.prototype.setColor = function (colorChosen) {
   this.color = colorChosen;
};

/****************************************
Name: Edge.prototype.setHlight
Description: Sets the highlight color of an Edge.
Parameter(s): 'ahColor' is a RGB value passed in from the Controller class.
****************************************/
Edge.prototype.setHlight = function (ahColor) {
   this.hColor = ahColor;
};

/****************************************
Name: Edge.prototype.clearHlight
Description: Removes the highlight of an edge by setting value to false.
Parameter(s): N/A.
****************************************/
Edge.prototype.clearHlight = function () {
   this.highlighted = false;
};

/****************************************
Name: EdgeSet
Description: Creates an empty array that will hold the Edges of the canvas.
Parameter(s): None.
```

```
*****************************************/
function EdgeSet() {
   this.eList = [];
}

/*****************************************
Name: EdgeSet.copy(eSet, vSet, lSet)
Description: copies an EdgeSet and its Edge members
Parameters:
         vSet: VertexSet to copy
         lSet: LabelSet with Labels for Vertices
*****************************************/
EdgeSet.copy = function(eSet, vSet, lSet){
      var me = this;
      this.eList = eSet.eList.map( function(edg){ return new Edge.copy(edg, me, vSet, lSet); } );
}

EdgeSet.copy.prototype = EdgeSet.prototype;

/*****************************************
Name: EdgeSet.prototype.saveData()
Description: creates a data set for downloading and saving
Parameters:
*****************************************/
EdgeSet.prototype.saveData = function(){
         return { eList: this.eList.map( function (e) {return e.saveData(); } ) };
}

/*****************************************
Name: EdgeSet.prototype.remove
Description: Removes the Edge from the EdgeSet at the specified sub
Parameter(s): 'Edge' refers to the Edge specified to be removed.
*****************************************/
EdgeSet.prototype.remove = function (Edge) {
   var i = this.eList.indexOf(Edge);

   if (i >= 0) {
      this.eList.splice(i, 1);
   }
};

/*****************************************
Name: EdgeSet.prototype.add
Description: Adds an Edge to the array of Edges.
Parameter(s): 'p' and 'q' refer to the  end vertices of the Edge.
*****************************************/
```

```
EdgeSet.prototype.add = function (p, q) {
   this.eList.push(new Edge(this, p, q));
};

/******************************************
Name: EdgeSet.prototype.get
Description: Returns the Edge at the specified index.
Parameter(s): 'i' refers to the index of the Edge in the array.
******************************************/
EdgeSet.prototype.get = function (i) {
   return this.eList[i];
};

/******************************************
Name: EdgeSet.prototype.getSize
Description: Returns the size of the Edge list.
Parameter(s): N/A.
******************************************/
EdgeSet.prototype.getSize = function () {
   return this.eList.length;
};

/******************************************
Name: EdgeSet.prototype.draw
Description: Draws the Edges of the array onto the canvas.
Parameter(s): 'ctxt' refers to the drawing context of the canvas.
******************************************/
EdgeSet.prototype.draw = function (ctxt) {
   var i;
   for (i = 0; i < this.eList.length; i += 1) {
      this.eList[i].draw(ctxt);
   }
};

/******************************************
Name: EdgeSet.prototype.drawHighlight
Description: Draws the highlighting for any Edge that has it.
Parameter(s): 'ctxt' refers to the drawing context of the canvas.
******************************************/
EdgeSet.prototype.drawHighlight = function (ctxt) {
   var i;

   for (i = 0; i < this.eList.length; i += 1) {
      this.eList[i].drawHighlight(ctxt);
   }
};
```

```
/*****************************************
Name: EdgeSet.prototype.drawSelect
Description:
Parameter(s): 'c'txt
*****************************************/
EdgeSet.prototype.drawSelect = function (ctxt) {
   var i;

   for (i = 0; i < this.eList.length; i += 1) {
      this.eList[i].drawSelect(ctxt);
   }
};

/*****************************************
Name: EdgeSet.prototype.hit
Description: Tests each Edge in the EdgeSet for a hit.
Parameter(s): 'x' & 'y' refer to the coordinates of the mouse event.
*****************************************/
EdgeSet.prototype.hit = function (x, y) {
   var i;

   for (i = 0; i < this.eList.length; i += 1) {
      if (this.eList[i].hit(x, y)) {
         return {object: this.eList[i], index: i};
      }
   }
   return null;
};

/*****************************************
Name: EdgeSet.prototype.clearHlight
Description: Clears highlighting on edges.
Parameter(s): N/A.
*****************************************/
EdgeSet.prototype.clearHlight = function () {
   var i;
   for (i = 0; i < this.eList.length; i += 1) {
      this.eList[i].clearHlight();
   }
};

/*****************************************
Name: Graph
Description: Creates a Graph object consisting of a set of vertices, Edges & Labels.
Parameter(s): N/A.
```

```
*****************************************/
function Graph() {
   this.eSet = new EdgeSet();
   this.lSet = new LabelSet();
   this.vSet = new VertexSet();
   this.selectPattern = null;
}

/*****************************************
Name: Graph.copy(aGraph)
Description: Copies a Graph object or initializes using graph fields.
Parameters:
	aGraph: Graph to copy.
*****************************************/
Graph.copy = function(gr){
	this.lSet = new LabelSet.copy(gr.lSet);
	this.vSet = new VertexSet.copy(gr.vSet, this.lSet);
	this.eSet = new EdgeSet.copy(gr.eSet, this.vSet, this.lSet);
   this.selectPattern = null;
}

Graph.copy.prototype = Graph.prototype;

/*****************************************
Name: Graph.prototype.saveData()
Description: creates a data set for downloading and saving
Parameters:
*****************************************/
Graph.prototype.saveData = function(){
	return {
		lSet: this.lSet.saveData(),
		vSet: this.vSet.saveData(),
		eSet: this.eSet.saveData()
	};
}

/*****************************************
Name: Graph.prototype.setSelectPattern
Description: Takes the created select pattern and passes reference to Graph variable.
Parameter(s): 'aPattern' is the pattern being used to designate a selected object.
*****************************************/
Graph.prototype.setSelectPattern = function (aPattern) {
   this.selectPattern = aPattern;
}

/*****************************************
```

```
Name: Graph.prototype.addLabel
Description: Adds a Label to the Graph object
Parameter(s): 'name' refers to the Label name; 'x' & 'y' are the coordinates of the Label.
******************************************/
Graph.prototype.addLabel = function (name, x, y, parent, context) {
   this.lSet.add(name, x, y, parent, context);
};

/******************************************
Name: Graph.prototype.addVertex
Description: Adds a Vertex to the Graph object.
Parameter(s): 'x' and 'y' refer to the coordinates of the Vertex being added.
******************************************/
Graph.prototype.addVertex = function (x, y) {
   this.vSet.add(x, y);
};

/******************************************
Name: Graph.prototype.addEdge
Description: Adds an Edge to the Graph object
Parameter(s): 'p' & 'q' are the end vertices the Edge is connected to.
******************************************/
Graph.prototype.addEdge = function (p, q) {
   this.eSet.add(p, q);
};

/******************************************
Name: Graph.prototype.getLabel
Description: Returns the Label at the given index
Parameter(s): 'index' refers to the position of the Label in the list.
******************************************/
Graph.prototype.getLabel = function (index) {
   return this.lSet.get(index);
};

/******************************************
Name: Graph.prototype.getVertex
Description: Returns the Vertex at the given index.
Parameter(s): 'index' refers to the position of the Vertex in the list.
******************************************/
Graph.prototype.getVertex = function (index) {
   return this.vSet.get(index);
};

/******************************************
Name: Graph.prototype.getEdge
```

```
Description: Returns the Edge at the given index.
Parameter(s): 'index' refers to the position of the Edge in the array.
******************************************/
Graph.prototype.getEdge = function (index) {
   return this.eSet.get(index);
};

/******************************************
Name: Graph.prototype.getSizevSet
Description: Returns the size of this Graph's Vertex set.
Parameter(s): N/A.
******************************************/
Graph.prototype.getSizevSet = function () {
   return this.vSet.getSize();
};

/******************************************
Name: Graph.prototype.getSizelSet
Description: Returns the size of this Graph's Label set.
Parameter(s): N/A.
******************************************/
Graph.prototype.getSizelSet = function () {
   return this.lSet.getSize();
};

/******************************************
Name: Graph.prototype.getSizeeSet
Description: Returns the size of this Graph's Edge set.
Parameter(s): N/A.
******************************************/
Graph.prototype.getSizeeSet = function () {
   return this.eSet.getSize();
};

/******************************************
Name: Graph.prototype.draw
Description: Draws Graph object by calling draw methods on its Vertex & Edge sets.
Parameter(s): 'ctxt' refers to the drawing context of the canvas.
******************************************/
Graph.prototype.draw = function (canvas, ctxt) {
   ctxt.save();
   ctxt.clearRect(0, 0, canvas.width, canvas.height);
   ctxt.restore();

   ctxt.save();
   ctxt.lineWidth = 8.5;
  /* ctxt.fillStyle = "#8CE6FF";
   ctxt.strokeStyle = "#8CE6FF";*/

   // draw highlight first
   this.vSet.drawHighlight(ctxt);
   this.eSet.drawHighlight(ctxt);
   this.lSet.drawHighlight(ctxt);
   ctxt.restore();

   // draw select pattern next
   ctxt.save();
   ctxt.fillStyle = this.selectPattern;
   ctxt.strokeStyle = this.selectPattern;
   this.vSet.drawSelect(ctxt);
   this.eSet.drawSelect(ctxt);
   this.lSet.drawSelect(ctxt);
   ctxt.restore();

   // draw objects last
   ctxt.save()
   this.eSet.draw(ctxt);
   this.vSet.draw(ctxt);
   this.lSet.draw(ctxt);
   ctxt.restore();
};

/****************************************
Name: Graph.prototype.hitVertex
Description: Tests whether a Vertex was selected.
Parameter(s): x & y refer to the coordinates where the mouse event occurred.
****************************************/
Graph.prototype.hitVertex = function (x, y) {
   return this.vSet.hit(x, y); // Returns the selected Vertex; otherwise null.
};

/****************************************
Name: Graph.prototype.hitEdge
Description: Tests whether an Edge was selected.
Parameter(s): x & y refer to the coordinates where the mouse event occurred.
****************************************/
Graph.prototype.hitEdge = function (x, y) {
   return this.eSet.hit(x, y); // Returns the selected Edge; otherwise null.
};

/****************************************
Name: Graph.prototype.hitLabel
```

```
Description: Tests whether a Label was selected.
Parameter(s): x & y refer to the coordinates where the mouse event occurred.
*****************************************/
Graph.prototype.hitLabel = function (x, y, ctxt) {
   return this.lSet.hit(x, y, ctxt); // Returns the selected Label; otherwise null.
};

/*****************************************
Name: Graph.prototype.clear
Description: Clears the canvas & Graph at users request.
Parameter(s): canvas is the canvas object; ctxt is the canvas drawing context.
*****************************************/
Graph.prototype.clear = function (canvas, ctxt) {
   ctxt.clearRect(0, 0, canvas.width, canvas.height);
   this.eSet = new EdgeSet();
   this.vSet = new VertexSet();
   this.lSet = new LabelSet();
};

/*****************************************
Name: Graph.prototype.clearHlight
Description: Clears all the higlighting from the graph.
Parameter(s): N/
*****************************************/
Graph.prototype.clearHlight = function () {
   if (this.getSizeeSet() > 0) {this.eSet.clearHlight();}

   if (this.getSizevSet() > 0) {this.vSet.clearHlight();}

   if (this.getSizelSet() > 0) {this.lSet.clearHlight();}
};
//}());
```

A-3 Controller.js

```
/*jslint browser: true*/
/*jslint devel: true */
/*jslint plusplus: true */
/*global Label, Vertex, Edge, toggleMenusOff, getLabelName, Graph, Controller */
/*****************************************
            Controller.js

The Controller class is the intermediary of communication between the Graph and
the user interface.

NOTE: This is the most recent version as of April 27th, 2014.
*****************************************/

var ctrl;

window.onload = function () {
   var bBtn, hBtn, sBtn, lblBtn, cBtn, vtxDelBtn, vtxLblBtn,
      edgDelBtn, edgLblBtn, lblDelBtn, lblchgBtn, clrToolBar,
                clrSwatches, selSwatch, currBtn, ewgButton, gDownloadButton,
      fileUploadSelect, cHlightBtn;

   ctrl = new Controller('gCanvas'); // creates a new Controller instance.

   /*cHlightBtn = document.getElementById('clearH');
   cHlightBtn.onclick = function () {
      ctrl.graph.clearHlight();
      ctrl.graph.draw(ctrl.canvas, ctrl.context); // Refactor this so that call is not done explicitly.
   };
        */
        // build Btn
        bBtn = document.getElementById('build');
        bBtn.onclick = function () {
                ctrl.changeState('BUILD');
      toggleMenusOff();
      turnOffBtn(currBtn);
      turnOnBtn(this);
      currBtn = this;
      console.log(this);
        };

        // label Btn
        lblBtn = document.getElementById('label');
        lblBtn.onclick = function () {
      //lblBtn.classList.toggle("active", true);
```

```
                ctrl.changeState('LABEL');
    toggleMenusOff();
    turnOffBtn(currBtn);
    turnOnBtn(this);
    currBtn = this;
    $(clrToolBar).fadeOut('slow');
        };

        // highlight Btn
        hBtn = document.getElementById('hlight');
        hBtn.onclick = function () {
                ctrl.changeState('HIGHLIGHT');
    toggleMenusOff();
    turnOffBtn(currBtn);
    turnOnBtn(this);
    currBtn = this;
    $(clrToolBar).fadeIn('slow');
        };

        // Clear Canvas button.
        cBtn = document.getElementById('clear');
        cBtn.onclick = function () {
                var confirm;
    confirm = ctrl.clearCanvas();
    if (confirm) {
        toggleMenusOff();
        $(clrToolBar).fadeOut('slow');
    }
    cBtn.blur();
        };

        // select Btn
        sBtn = document.getElementById('select');
        sBtn.onclick = function () {
                ctrl.changeState('SELECT');
    turnOffBtn(currBtn);
    turnOnBtn(this);
    currBtn = this;
        };

// new graph button
        newgButton = document.getElementById('newgraph');
        newgButton.onclick = function () {
    newgButton.blur();
                var newWin = window.open(decodeURI(document.URL),'_blank');
        };
```

```
// Save Graph button.
        gDownloadButton = document.getElementById('download');
        gDownloadButton.onclick = function(){
    var blob = new Blob([JSON.stringify(ctrl.graph.saveData())], {type: 'text/plain'});
    var url = URL.createObjectURL(blob);

    var a = document.createElement('a');
    a.href = url;
    a.download = 'p.txt';
    a.textContent = '';
    document.body.appendChild(a);
    a.click();
    gDownloadButton.blur();
        }

// Load Graph button.
        fileUploadSelect = document.getElementById('Upload');
        fileUploadSelect.addEventListener(
                'change',
                function(evt) {
    ctrl.changeState('SELECT');
    toggleMenusOff();
    turnOffBtn(currBtn);
    turnOnBtn(sBtn);
    currBtn = sBtn;
                        var file = evt.target.files[0]; // FileList object
                        var reader = new FileReader();

                        // Closure to capture the file information.
                        reader.onload = (function(theFile) {
                                return function(e) {
                                        var temp = new Graph.copy(JSON.parse(e.target.result));
                                        ctrl.graph = temp;
                                        //ctrl.graph.
                                        ctrl.graph.setSelectPattern(ctrl.selectPattern);
                                        ctrl.graph.draw(ctrl.canvas, ctrl.context);
                                        //bButton.click();
                                };
                        })(file);
                        // Read in the image file as a data URL.
                        reader.readAsText(file);
        fileUploadSelect.blur();
                },
                false
        );
        // Btns specific to vertices

        // delete vertex Btn.
        vtxDelBtn = document.getElementById('vtxdelete');
        vtxDelBtn.onclick = function () {
                ctrl.remove();
    $(clrToolBar).fadeOut("slow");
    toggleMenusOff();
        };

        // label vertex Btn.
        vtxLblBtn = document.getElementById('vtxlabel');
        vtxLblBtn.onclick = function () {
    ctrl.LabelObj();
};


// delete Edge Btn.
edgDelBtn = document.getElementById('edgedelete');
edgDelBtn.onclick = function () {
                ctrl.remove();
    $(clrToolBar).fadeOut("slow");
    toggleMenusOff();
};
// label Edge Btn.
edgLblBtn = document.getElementById('edgelabel');
edgLblBtn.onclick = function () {
                ctrl.LabelObj();
};

// delete label Btn.
lblDelBtn = document.getElementById('labeldelete');
lblDelBtn.onclick = function () {
    ctrl.remove();
    $(clrToolBar).fadeOut("slow");
    toggleMenusOff();
};

lblchgBtn = document.getElementById('labelchange');
lblchgBtn.onclick = function () {
    ctrl.LabelObj();
};

currBtn = bBtn; // On load, Build is first mode.
```

```
   turnOnBtn(bBtn);

         // color toolbar & swatches
   clrToolBar = document.getElementById('color-toolbar');
         clrSwatches = document.getElementsByClassName('color-swatch');

   for (var i = 0; i < clrSwatches.length; i += 1) {
      clrSwatches[i].addEventListener("click", getSwatch, false);
   }

   function getSwatch(event) {
      selSwatch = window.getComputedStyle(document.getElementById(event.target.id),
"background-color");
      ctrl.setColor(selSwatch.backgroundColor);
   }
};

/*****************************************
Name: turnOnBtn
Description: Changes button to display that its currently active.
Parameter(s): 'selectedBtn' is a reference to the current button.
*****************************************/
var turnOnBtn = function (selectedBtn){
   var selbtn;
   selbtn = selectedBtn;
   selbtn.classList.toggle("active", true); //
};

/*****************************************
Name: turnOffBtn
Description: Resets button display to inactive.
Parameter(s): 'previousBtn' is a reference to the previous active button.
*****************************************/
var turnOffBtn = function (previousBtn) {
   var prevbtn;
   prevbtn = previousBtn;
   prevbtn.classList.toggle("active", false);
};

/*****************************************
Name: toggleMenuOn
Description: Causes specific menu to be displayed.
Parameter(s): 'type' is a string that determines what menu to display based
         on which canvas object was selected.
*****************************************/
var toggleMenuOn = function (type) {
         switch (type) {
         case 'Vertex':
                 document.getElementById('vertexMenu').style.display = "inline-block";
                 break;
         case 'Edge':
                 toggleMenusOff();
                 document.getElementById('edgeMenu').style.display = "inline-block";
                 break;
         case 'Label':
                 toggleMenusOff();
                 document.getElementById('labelMenu').style.display = "inline-block";
                 break;
   }

   $('#color-toolbar').fadeIn("slow");
};

/*****************************************
Name: toggleMenusOff
Description: Toggles off canvas object menus.
Parameter(s): N/A.
*****************************************/
var toggleMenusOff = function () {
         document.getElementById('edgeMenu').style.display = "none";
         document.getElementById('labelMenu').style.display = "none";
         document.getElementById('vertexMenu').style.display = "none";
};

/*****************************************
Name: Controller
Description: Handles interactions with the HTML 5 canvas object.
Parameter(s): 'canvasID' is the id of the HTML 5 canvas object.
*****************************************/
function Controller(canvasID) {
         this.STATE = 'BUILD'; // Initial state set to build mode.
         this.mouseData = { track: false, x: 0, y: 0, dx: 0, dy: 0, sx: 0, sy: 0 }; // Keeps track of the
position of the mouse.
   this.graph = new Graph(); // Creates a new Graph object for this controller.
         this.canvas = document.getElementById(canvasID); // The HTML canvas object.
   this.context = this.canvas.getContext("2d");

   this.selectPattern = new CreateSelectPattern(this.context);
   this.graph.setSelectPattern(this.selectPattern);
   this.currentHColor ="#FFFF6D";
         this.startVertex = null;
         this.canvasObj = null;
         this.newLabel = null;

         // The following are event listeners placed on the canvas object,
         // which then call prototype wrappers for handling mouse events.
         this.canvas.addEventListener('mousedown', function (event) {
                 this.CanvasMouseDownHandler(event);
         }.bind(this), false);

         this.canvas.addEventListener('mousemove', function (event) {
                 this.CanvasMouseMoveHandler(event);
         }.bind(this), false);

         this.canvas.addEventListener('mouseup', function (event) {
                 this.CanvasMouseUpHandler(event);
         }.bind(this), false);

   this.canvas.addEventListener('mouseout', function (event) {
     this.CanvasOutOfBounds(event);
   }.bind(this), false);

         this.canvas.addEventListener('contextmenu', function (event) {
                 this.CanvasRightClickHandler(event);
         }.bind(this), false);

}

/******************************************
Name: Controller.prototype.changeState
Description: Changes the internal state of the Controller object
Parameter(s): 'aState' is a string passed in from a Btn object.
******************************************/
Controller.prototype.changeState = function (aState) {
   if (this.STATE === 'SELECT' && this.canvasObj) {
     this.canvasObj.object.toggleSelect();
     this.graph.draw(this.canvas, this.context);
     this.canvasObj = null;
   }

   this.STATE = aState;
};

/******************************************
Name: Controller.prototype.remove
Description: Removes selected object from the canvas.
Parameter(s): N/A.
******************************************/
```

```
Controller.prototype.remove = function () {
        if (this.canvasObj) {
                this.canvasObj.object.remove();
                this.graph.draw(ctrl.canvas, ctrl.context);
                this.canvasObj = null;
        }
};

/****************************************
Name: Controller.prototype.setColor
Description: Sets the color or highlighted color of the selected object.
Parameter(s): 'color' is a reference to the RGB value of the selected color button.
****************************************/
Controller.prototype.setColor = function (color){
   this.context.save();
   if (this.canvasObj) {
           if (this.STATE === 'SELECT') {
                   this.canvasObj.object.setColor(color);
      }
   }

   if (this.STATE === 'HIGHLIGHT') {
      this.currentHColor = color;
      console.log("selected " + this.currentHColor);
   }
   this.graph.draw(this.canvas, this.context);
   this.context.restore();
};

/****************************************
Name: Controller.prototype.LabelObj
Description: Binds a label to a canvas object.
Parameter(s): N/A.
****************************************/
// REFACTOR START.
Controller.prototype.LabelObj = function () {
   var labeltxt, objLabel;

   if(this.canvasObj.object instanceof Label){
      labeltxt = this.replaceLabelTxt(this.canvasObj.object)
      if (labeltxt) {
         this.canvasObj.object.changeText(labeltxt);
      }
   } else {
      objLabel = this.canvasObj.object.getLabel(); // Take current object's Label; store reference.
      if (objLabel) { // check if object has Label
```

```
          // if true, call method to replace the Label's text.
          labeltxt = this.replaceLabelTxt(objLabel);
          if (labeltxt) {
             objLabel.changeText(labeltxt); // Change label's text to display new text.
          }
       } else { // 'aLabel' is null, thus make a new label for the object.
          labeltxt = this.getLabelTxt(); // Call method to prompt user for new Label's text.
          if (labeltxt) {
             this.graph.addLabel(labeltxt, this.canvasObj.object.x, this.canvasObj.object.y,
                         this.canvasObj.object, this.context);
             this.canvasObj.object.setLabel(this.graph.getLabel(this.graph.getSizelSet() - 1));
          }
       }
    }
    this.graph.draw(this.canvas, this.context);
// REFACTOR END
};

/****************************************
Name: Controller.prototype.getLabelName
Description: Helper function that prompts user for a new Label's text, then
          returns the text string.
Parameters: N/A.
****************************************/
Controller.prototype.getLabelTxt = function () {
          var ltext;

    ltext = prompt("Please enter text for this label");

    if (ltext) {
       ltext = ltext.trim();
    }

    if (ltext !== null && ltext.length === 0) {
       alert("Label cannot be blank!");
       ltext = prompt("Please enter text for this label");

       if (ltext !== null) {
          ltext = ltext.trim();
       }
    }

          return ltext;
};

/****************************************
```

```
Name: Controller.prototype.replaceLabelTxt
Description: Helper function that prompts user for a Label's text.
Parameter(s): 'aLabel' is a reference to a label attached to a graph object.
*****************************************/
Controller.prototype.replaceLabelTxt = function (aLabel) {
        var ltext, currentLabelTxt;

        currentLabelTxt = aLabel.getText();
        console.log(currentLabelTxt);

  ltext = prompt("Please enter text for this label", currentLabelTxt);

  if (ltext) {
    ltext = ltext.trim();
  }

  while (ltext !== null && ltext.length === 0) {
    alert("label cannot be blank!");
    ltext = prompt("Please enter text for this label", currentLabelTxt);

    if (ltext !== null) {
      ltext = ltext.trim();
    }
  }

        return (ltext || currentLabelTxt);
};

/*****************************************
Name: Controller.prototype.clearCanvas
Description: Clears the canvas.
Parameter(s): N/A.
*****************************************/
Controller.prototype.clearCanvas = function () {
  var isConfirm;
  isConfirm = confirm('Are you sure you want to clear the canvas?');

  if (isConfirm) {
    this.graph.clear(this.canvas, this.context);
  }
  return isConfirm;
};

/*****************************************
Name: Controller.prototype.hitTest
Description: Tests whether an object on the canvas was hit. If true, return
```

```
        the object.
Parameter(s): 'x' & 'y' are the mouse coordinates.
******************************************/
Controller.prototype.hitTest = function (x, y) {
        return (this.graph.hitVertex(x, y) || this.graph.hitEdge(x, y) || this.graph.hitLabel(x, y,
this.context));
};

/******************************************
Name: Controller.prototype.CanvasRightClickHandler
Description: Prevents default right-click context-menu from being displayed.
Parameter(s): 'event' represents a mouse right-click event captured from the canvas.
******************************************/
Controller.prototype.CanvasRightClickHandler = function (event) {
        event.preventDefault();
        // Add code here to display a new context-menu with canvas object options
        // such as "Add vertex", "Add highlighting", etc.
};


/******************************************
Name: CanvasOutOfFocus
Description: Handles mouse down click event on the canvas (user presses left-click)
Parameter(s): 'event' represents a mouse-down event on the canvas object.
******************************************/
Controller.prototype.CanvasOutOfBounds = function (event) {
/*    switch (this.STATE) {
    case 'BUILD':
        break;
    case 'SELECT':
      if(this.canvasObj){
        this.canvasObj.object.move((this.mouseData.sx - this.mouseData.x), (this.mouseData.sy -
this.mouseData.y));
                this.mouseData.track = false;
        this.graph.draw(this.canvas, this.context);
      }
        break;
    }*/
};

/******************************************
Name: CanvasMouseDownHandler
Description: Handles mouse down click event on the canvas (user presses left-click)
Parameter(s): 'event' represents a mouse-down event on the canvas object.
******************************************/
Controller.prototype.CanvasMouseDownHandler = function (event) {
        this.mouseData.track = true; // Begin tracking mouse movement.
        //this.x = event.clientX - this.rect.left; // X-coordinate where mouse event occurred on
canvas.
        //this.y = event.clientY - this.rect.top; // Y-coordinate where mouse event occurred on
canvas.
    this.x = event.clientX - this.canvas.getBoundingClientRect().left;
    this.y = event.clientY - this.canvas.getBoundingClientRect().top;

        this.mouseData.x = this.x; // Save the current x position.
        this.mouseData.y = this.y; // Save the current current y position.
    this.mouseData.sx = this.x; // Save the current x position.
        this.mouseData.sy = this.y; // Save the current current y position.

        var hit = this.hitTest(this.mouseData.x, this.mouseData.y); // Test if a canvas object was
hit.
        switch (this.STATE) {
    case 'BUILD':
        if (hit && hit.object instanceof Vertex) { // Testing whether a graph object was hit on the
canvas.
        this.startVertex = hit.object;

        } else { // No graph object hit, then add a vertex at current position.
            this.graph.addVertex(this.mouseData.x, this.mouseData.y);
            this.startVertex = this.graph.getVertex(this.graph.getSizevSet() - 1);
        }
        break;
    case 'HIGHLIGHT':
                this.canvasObj = hit;
        break;
    case 'LABEL':
        break;
    case 'SELECT':
        if (hit) {
            console.log(hit);
            if (this.canvasObj) {
                console.log(this.canvasObj);
                if (this.canvasObj.object.selected === true) { // Change to isSelected function
                    this.canvasObj.object.toggleSelect();
                }
            }
            console.log(hit);
            this.canvasObj = hit;
            this.canvasObj.object.toggleSelect();
        } else {
            if (this.canvasObj) {
                if (this.canvasObj.object.selected) {
```

```
                this.canvasObj.object.toggleSelect();
                this.canvasObj = null;
            }
          }
        }
        break;
    }
        this.graph.draw(this.canvas, this.context);
};

/*****************************************
Name: CanvasMouseMoveHandler
Description: Handles mouse move events on the canvas (user is holding down left-click).
Parameter(s): 'event' refers to the mouse move event.
*****************************************/
Controller.prototype.CanvasMouseMoveHandler = function (event) {
        if (!this.mouseData.track) { // Stops tracking and returns when user releases left-click.
                return;
        }
    this.x = event.clientX - this.canvas.getBoundingClientRect().left;
    this.y = event.clientY - this.canvas.getBoundingClientRect().top;

        switch (this.STATE) {
    case 'BUILD': // the following begins drawing an edge from starting vertex until a mouse up
event is detected.
        this.graph.draw(this.canvas, this.context);
        this.context.beginPath();
        this.context.moveTo(this.mouseData.x, this.mouseData.y);
        this.context.lineTo(this.x, this.y);
        this.context.lineWidth = 3.5;
        this.context.stroke();
        break;
    case 'SELECT':
        if (this.canvasObj !== null) {
            this.canvasObj.object.move((this.x - this.mouseData.x), (this.y - this.mouseData.y));
            this.mouseData.x = this.x;
            this.mouseData.y = this.y;
        }
        this.graph.draw(this.canvas, this.context);
        break;
    }
};

/*****************************************
Name: CanvasMouseUpHandler
Description: Captures a mouse-click release; based on current Controller state,
            perform necessary method calls.
Parameter(s): 'event' refers to the mouse up event.
*****************************************/
Controller.prototype.CanvasMouseUpHandler = function (event) {
            //this.x = event.clientX - this.rect.left;
            //this.y = event.clientY - this.rect.top;
    this.x = event.clientX - this.canvas.getBoundingClientRect().left;
    this.y = event.clientY - this.canvas.getBoundingClientRect().top;

            var hit = this.hitTest(this.x, this.y);

            switch (this.STATE) {
    case 'BUILD':
        if (hit !== null && hit.object instanceof Vertex) {
            if (hit.object !== this.startVertex) {
                this.graph.addEdge(this.startVertex, hit.object);
            }
        } else if (0 <= this.x && this.x <= this.canvas.width && 0 <= this.y && this.y <=
this.canvas.height) {
            this.graph.addVertex(this.x, this.y);
            this.graph.addEdge(this.startVertex, this.graph.getVertex(this.graph.getSizevSet() - 1));
        }
        break;
    case 'HIGHLIGHT':
                        if (this.canvasObj) {
            //this.canvasObj.object.setHlight(this.currentHColor);
                                    //if (this.canvasObj.object.toggleHighlight) {
            this.canvasObj.object.toggleHighlight(this.currentHColor);
            this.graph.draw(this.canvas, this.context);
            //}
                        }
        this.canvasObj = null;
        break;
    case 'LABEL':
        this.newLabel = this.getLabelTxt();
        if (this.newLabel) {
                                    this.graph.addLabel(this.newLabel, this.x, this.y, null, this.context);
                        }
        this.mouseData.track = false;
        this.canvasObj = null;
        break;
    case 'SELECT':
        if (this.canvasObj) {
            toggleMenusOff();
            if (this.canvasObj.object instanceof Vertex) {
                toggleMenuOn('Vertex');
        } else if (this.canvasObj.object instanceof Edge) {
                                toggleMenuOn('Edge');
                        } else if (this.canvasObj.object instanceof Label) {
                                toggleMenuOn('Label');
                        } else {
            return;
        }
      } else {
        $('#color-toolbar').fadeOut('slow');
        toggleMenusOff();
      }
      break;
  }

        this.newLabel = null;
        this.startVertex = null;
        this.mouseData.track = false;
  this.graph.draw(this.canvas, this.context);
};

/*****************************************
Name: CreateSelectPattern
Description: Draws a pattern off screen that is then used to signify a selected object.
Parameter(s): 'ctxt' refers to the drawing context of the canvas.
*****************************************/
var CreateSelectPattern = function (ctxt) {
  var tempCanvas, tempContext, tempData, selectData, i, j, k;

        tempCanvas = document.getElementById("tempCanvas");
        tempCanvas = document.createElement('canvas');
        tempContext = tempCanvas.getContext('2d');
        tempData = tempContext.getImageData(0, 0, 8, 8);
        selectData = [ [1,0,0,0,1,0,1,1], [1,0,0,0,1,1,0,0], [0,1,1,1,0,1,0,0], [1,1,0,0,0,1,0,0],
                [0,0,1,0,0,0,1,1], [0,0,1,0,1,1,1,0], [0,0,1,1,0,0,0,1], [1,1,0,1,0,0,0,1] ];

        tempCanvas.height = 8;
        tempCanvas.width = 8;

        k = 0;
        for (i = 0; i < 8; i += 1) {
                for (j = 0; j < 8; j += 1) {
                        if ( selectData[i][j] === 1 ) {
                                tempData.data[k++] = 0xb0;
                                tempData.data[k++] = 0xb0;
                                tempData.data[k++] = 0xb0;
                                tempData.data[k++] = 0xff;
                }
                else{
                        tempData.data[k++] = 0;
                        tempData.data[k++] = 0;
                        tempData.data[k++] = 0;
                        tempData.data[k++] = 0x0;
                }
            }
        }

        tempContext.putImageData(tempData,0,0);

        return ctxt.createPattern(tempCanvas, "repeat");
};
```

A-4 canvasStyle.css

```
#canvasDIV {
   margin-left: auto;
   margin-right: auto;
   position: absolute;
   overflow: hidden;
   top: 175px;
   left: 200px;
}

#gCanvas {
   /*display: inline-block;*/
   display: block;
   margin-left: auto;
   margin-right: auto;
   border-width: 3px;
   border-style: groove;
   /*background: #F2F2F2;*/
   background: #FFFFFF;
   display: block;
}

#defaultMenu {
         display: inline-block;
}

#defaultMenu .btn:focus {
   outline: 0;
}

#saveloadMenu{
   float: right;
}

#headertitle {
   font-size: 24px;
}

.color-swatch {
   width: 35px;
   height: 35px;
}

.color-swatch:focus {
   background-color: #FFFFFF;
}

#clear:focus{
   /*background: transparent;*/

}


#color-toolbar {
	display: none;
   position: relative;
   top: -15px;
   right: 150px;
   text-align: center;
}

#color-swatch-blue {
   /*background-color: #428bca;*/
   background-color: #96E6FA;
   /*background-color: #009999;*/
}

#color-swatch-red {
   background-color: #d9534f;
}
#color-swatch-yellow {
	background-color: #FFFF0D;
}

#color-swatch-green {
   background-color: #5cb85c;
	/*background-color: #D0FA5A;*/
}

#color-swatch-orange {
  /*  background-color: #f0ad4e; */
  background-color: #FF8C50
}

#color-swatch-pink {
	background-color: #FF96DC;
}

#color-swatch-black {
   background-color: #000000;
}
```

```
#vertexMenu {
        display: none;
        position: relative;
}

#edgeMenu {
        display: none;
        position: relative;
}

#labelMenu {
        display: none;
        position: relative;
}


.btn-file {
   position: relative;
   overflow: hidden;
}
.btn-file input[type=file] {
   position: absolute;
   top: 0;
   right: 0;
   min-width: 100%;
   min-height: 100%;
   font-size: 999px;
   text-align: right;
   filter: alpha(opacity=0);
   opacity: 0;
   outline: none;
   background: white;
   cursor: inherit;
   display: block;
}

.context-menu {
   display: none;
   z-index:1000;
   position: absolute;
   background-color:#EEE;
   border: 1px solid #888;
   overflow: hidden;
   width: 120px;
   white-space:nowrap;
   font-family: sans-serif;
}

.contextMenu li {
   padding: 5px;
}

.contextMenu li:hover {
   background-color: #DDD;
   cursor: pointer;
}

body {
   background: #FFFFFF;
}

::selection {
   color: #000000;
}
```