San Jose State University
SJSU ScholarWorks

Master's Projects | Master's Theses and Graduate Research

1-1-2010

A Framework for Active Learning: Revisited

Yue Chen
San Jose State University

Follow this and additional works at: http://scholarworks.sjsu.edu/etd_projects

Recommended Citation
Chen, Yue, "A Framework for Active Learning: Revisited" (2010). *Master's Projects.* Paper 50.

This Master's Project is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Projects by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

A FRAMEWORK FOR ACTIVE LEARNING: Revisited

A Report
Presented to
The Faculty of the Department of Computer Science
San Jose State University

In Partial Fulfillment
Of the Requirements for the Degree
Master of Science, Computer Science

By
Yue Chen
May 2010

2010

Yue Chen

ALL RIGHTS RESERVED

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

_____

Dr. Cay Horstmann

_____

Dr. David Taylor

_____

Dr. Suneuy Kim

APPROVED FOR THE UNIVERSITY

_____

Abstract

Over the past decade, algorithm visualization tools have been researched and developed to be used by Computer Science instructors to ease students' learning curve for new concepts. However, limitations such as rigid animation frameworks, lack of user interaction with the visualization created, and learning a new language and environment, have severely reduced instructors' desire to use such a tool. The purpose of this project is to create a tool that overcomes these limitations. Instructors do not have to get familiar with a new framework and learn another language. The API used to create algorithm animation for this project is through Java, a programming language familiar to many instructors. Moreover, not only do the instructors have control over planning the animation, students using the animation will also have the ability to interact with it.

Table of Contents

Abstract 4
Table of Contents 5
Index of Figures 6
Index of Listings 7
1. Introduction 8
2. Prior Work 8
  2.1 Sorting Out Sorting 8
  2.2 Algorithms in Action 9
  2.3 LaTex and Prefuse PDF Slides Creation Tool 10
  2.4 Animal 11
  2.5 Jeliot 13
  2.6 TRAKLA2 14
3. Framework Architecture 14
  3.1 Sharma's Framework 14
  3.2 This Framework 18
4. How to Use The Animation Package 19
  4.1 Components 19
    4.1.1 Node 20
    4.1.1.1 Package Node 20
    4.1.1.2 Object Node 20
    4.1.1.3 Linked list Node 20
    4.1.1.4 Label Node 21
    4.1.1.5 Note Node 21
    4.1.1.6 Variable Node 21
    4.1.2 Code 22
    4.1.3 Text 22
    4.1.4 Line 23
    4.1.5 Table 23
  4.2 Concrete Examples 24
    4.2.1 Linked List Demo 24
    4.2.2 Inheritance Demo 27
    4.2.3 Run Time Stack Demo 29
    4.2.4 Tracing a Loop Demo 31
5. Evaluation 32
  5.1 Teacher Empowerment 32
  5.2 User Interface 33
  5.3 Optimized to Task 33
6. Conclusions 35
References 36

Index of Figures

Figure 1 Sorting out sorting screenshot [9]..... 9
Figure 2 Algorithms in Action screen shot. [13] 10
Figure 3 Process of generating PDF slides utilizing LaTeX [6]. 11
Figure 4 Animal screenshot. 12
Figure 5 Jeliot screen shot..... 13
Figure 6 TRAKLA2 screen shot. 14
Figure 7 Sharma's algorithm visualization screenshot. [5] 15
Figure 8 Screenshot of the animation framework. 19
Figure 9 Object Node. 20
Figure 10 List Node. 21
Figure 11 Label Node. 21
Figure 12 Variable Node. 22
Figure 13 Table Object. 23
Figure 14 Linked-list Animation. 25
Figure 15 Inheritance demo screen shot. 27
Figure 16 Run time stack screen shot. 30
Figure 17 Tracing a loop screen shot. 31

Index of Listings

Listing 1 Original animation code [7]. 16
Listing 2 Animation code with Timing Framework. [7]. 16
Listing 3 Animation blockage implementation. 18
Listing 4 Creating Package Node. 20
Listing 5 Creating Object Node. 20
Listing 6 Creating List Node. 21
Listing 7 Creating Label Node. 21
Listing 8 Creating Note Node. 21
Listing 9 Creating Variable Node. 22
Listing 10 Creating Code Object. 22
Listing 11 Creating Text Object. 23
Listing 12 Creating Line Object. 23
Listing 13 Creating Table Object. 24
Listing 14 Create list node and group it to the appropriate package node. 26
Listing 15 Display animate effects to the node just created. 26
Listing 16 Connecting two list node. 26
Listing 17 Interaction tool creation. 26
Listing 18 Moving a node. 26
Listing 19 Loading caption into the animation. 27
Listing 20 Loading caption into the animation. 28
Listing 21 Creating three label nodes. 28
Listing 22 Package node and its usage. 28
Listing 23 Highlighting a phrase within a line of code. 29
Listing 24 Connecting between a node object and a line within a code object. 29
Listing 25 Add phrase arrow to the beginning of a line of code. 30
Listing 26 Highlighting a field node, and change a field node's value. 31
Listing 27 Display table element and draw line across the table element. 32
Listing 28 User interaction for string value. 32
Listing 29 JavaFX Code Sample. 34
Listing 30 Summary of tool comparison. 35

1. Introduction

Aside from the existing mundane textbook memorization, instructors have adopted tools such as Microsoft PowerPoint to create lecture slides with supplemental course materials for students to grasp. However static text book and power point slides tend to miss key points in explaining the algorithm. Over the decades, hundreds of visualization tools have been developed intended to help Computer Science instructors to illustrate how basic algorithms and data structures work. However, with the hundreds of visualization tools created, none have been widely utilized [2]. In fact, Shaffer et al [2] collected over 350 visualizations and tools from the internet and studied each. They stated that most of the visualization created are of poor quality or tend towards simple problems that serve no useful pedagogical purpose. The goal of this project is to create a tool that overcomes the limitations of existing visualization tools. The tool should let instructors be in control and create the animations they would like to use. The resulting animations should also be user interactive, allowing students to be participants rather than “passive observers”.

In the second section of the report, I will talk about the existing frameworks that were studied, and their advantages versus their disadvantages. In the third section, I will discuss the animation framework architecture. The fourth section will focus on the components of the animation system and its usage. User evaluation will be discussed in the fifth section. Future work and conclusions will be discussed in the final section.

2. Prior Work

Older visualization tools comprised of simple figure movement, musical notes, and narration, do serve the purpose of explaining a certain algorithm. However, there is no user interactions involved, so the user will lose interest after watching the animation for a period of time. In addition, simple actions such as revisiting a certain algorithm in the middle of the animation can be extremely difficult. With these limitations in mind, animations created later tried to avoid these deficiencies by creating animation with high resolution graphics, and incorporated user interactions. The AlgoViz Wiki set up by Shaffer et al contains a list of algorithm visualization tools created in the past decade. However, to their disappointment, most of the visualization tools created was still severely handicapped [2]. Despite the different ways the animations tried to grab a user's attention, the user still played a role of “passive observer” rather than an interactive participant. Below I will discuss several existing algorithm visualization tools.

2.1 Sorting Out Sorting

“Sorting Out Sorting” created by Ronald Baeker in 1981, is one of the first algorithms visualization ever created [2]. The animation is designed to make different sorting techniques such as insertion sort, binary sort, exchange sort and others easy to understand. The animation is composed of simple figures and letters, effects such as highlight, moving, sliding are then applied onto the objects. Throughout the animation, mono-toned musical notes are played indicating the processing stage or completion of the animation. A

narrator explains what each stage of the animation is doing. Figure 1 displays a screenshot of the animation in action.

Figure 1 Sorting out sorting screenshot [9].

In this screenshot, the performances for each of the three different sorting algorithms are being compared side-by-side in real time. This algorithm visualization is simple, detailed and easily understood. However, due to time constraints, only one problem instance can be explained for each sorting algorithm. So users can still be easily confused if the problem is modified in some degree. Moreover, there is no user interaction at all with this visualization.

2.2 Algorithms in Action

Algorithms in Action, created by Stern et al [13], explained a number of algorithms using "step-wise refinement". The visualization tool is composed of four sections: explanations, help, algorithm and the main visualization window. The explanation window gives a general overview of the algorithm as well as explanations to each individual pseudo-code line that the user might be interested in. The algorithm window displays the code for the algorithm, and its corresponding pseudo-code. Finally, the main visualization window displays the visualization of the algorithm. Figure 2 is a screenshot of the visualization during execution.

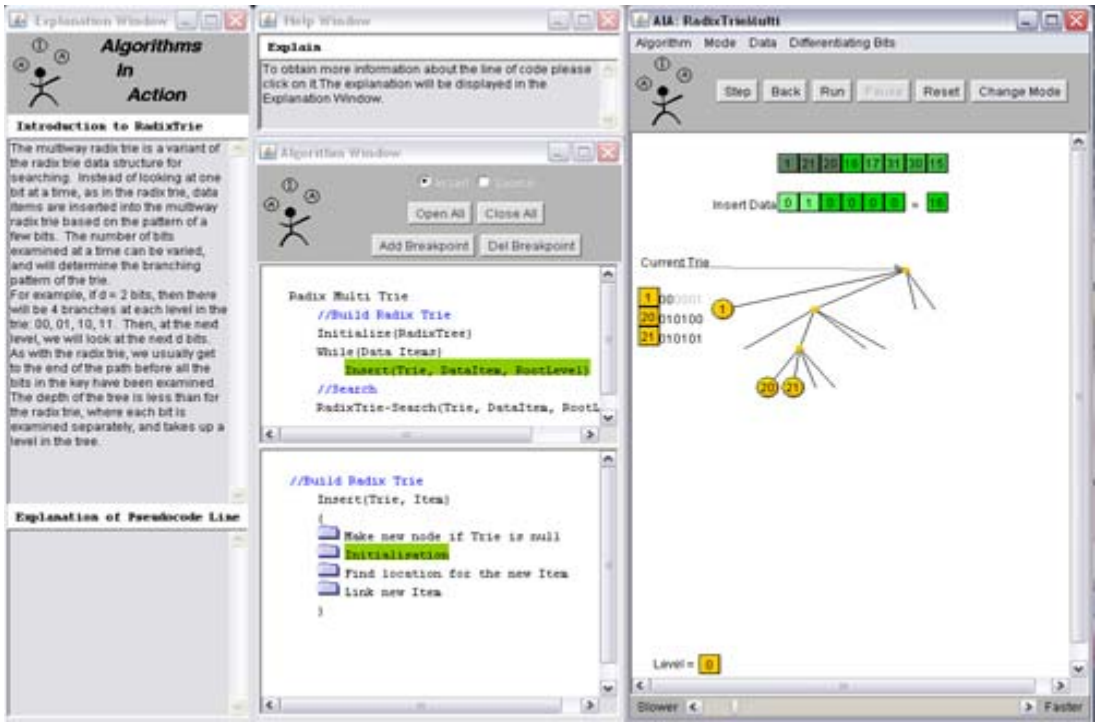


Figure 2 Algorithms in Action screen shot. [13]

In this visualization, the user can either manually step through the animation, or let the animation run to completion on its own. For every stage of the animation execution, the main visualization window will display the tree structure with some comments labeled on the side. The algorithm window highlights the current line of code that is been executed. By selecting a line of code in the algorithm window, an explanation of the line's purpose will be displayed in the explanation window. Aside from the default set of data values used for this animation, user can also enter a desired set of data values they are interested in or let the data values be randomly generated. Though user interaction is included into the animation, it is still severely limited. Since the tool only deals with pre-determined algorithms, instructors have no way of adding in their own content. Students do not have the opportunity to solve the problem on their own without the temptation of viewing the answers. In this example, the data values for the problem instance is modifiable, however the way the animation is presented to the user is fixed. This can be a serious problem when there are certain sections of the animation which needs more explanations than the other.

2.3 LaTex and Prefuse PDF Slides Creation Tool

Erkan et al [6] designed a data structure visualization utilizing LaTeX and Prefuse. Their project focused on algorithm visualization with no animation involved. They argue that since most instructors use presentation slides to show how an algorithm works and find it effective, it would be good to develop a tool that can generate detailed slides automatically. Their hypothesis was, if the slides were detailed enough, students should be able to review the material ahead of class and reverse engineer it to understand the material beforehand. This will significantly reduce instructors' efforts in teaching a new algorithm or

data structure, and maximize students' productivity. Figure 3 displays the general idea for their project.

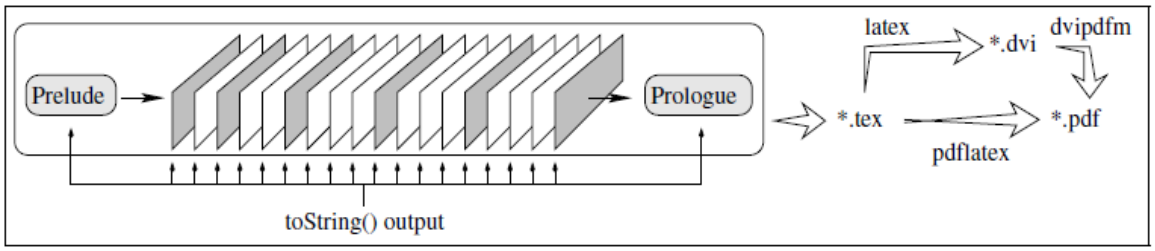


Figure 3 Process of generating PDF slides utilizing LaTeX [6].

The list of slides in the image represents the complete output that will be generated. The gray slides represent the “well defined” stages of the data structure/algorithm, such as the before and after of a particular algorithm function call. Any intermediate modifications to the algorithm or any object for a data structure object during the execution of the function are represented on the white colored slides. To generate a customizable PDF PowerPoint slide for particular data structure/algorithm visualization, a set of API functions can be called. The Java Reflection API can also be used to get information regarding the status of an object for the data structure/algorithm during execution. A major advantage of this project is that instructors now have the flexibility of deciding which parts of the algorithm deserves more attention, and provide appropriate information for it. A concern might be, with algorithms that are complicated and cumbersome in detail, the number of slides created will be too large to be of any real practical use.

2.4 Animal

Animal is a tool similar to the one discussed above. However, instead of automatically generating a large number of PDF presentation slides, Animal is a Java applet that animates the objects used to illustrate a particular algorithm or data structure. Figure 4 displays the application screenshot during execution.

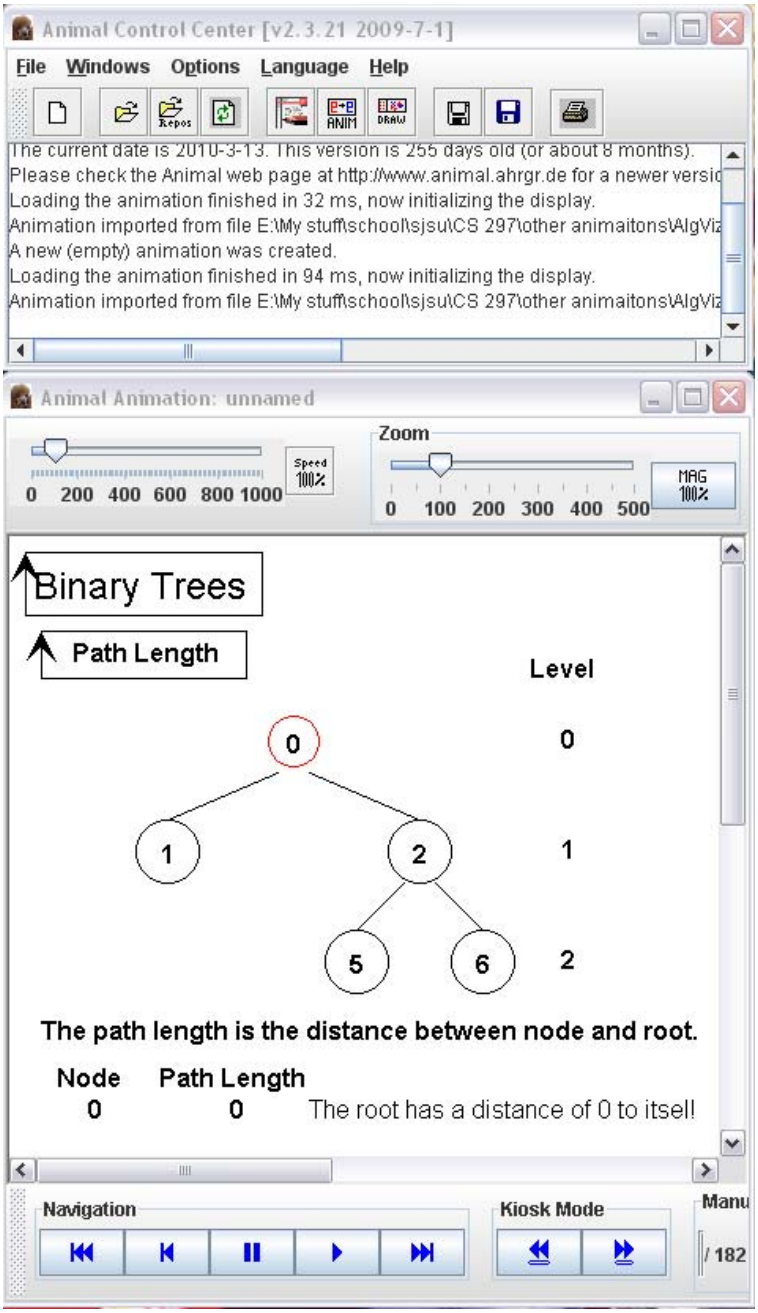


Figure 4 Animal screenshot.

Animal contains several windows. Two windows opened on default are the Control and Animation windows. The control window contains tools and options to control an existing animation or create a new animation. The animation window displays the actual animation that's been created. Animations are created either through the drawing window or scripting using AnimalScripts. Compared to the previous two algorithm visualization tools, a major advantage of this tool is that it gives instructors control in creating the animations they want. However, creating animation using this tool requires learning the new API for AnimationScripts or learning how to draw animation objects with the drawing window. Due to the lack of user interaction with the animation being generated, [2] claims that it is simply another tool that is equivalent to Microsoft PowerPoint.

2.5 Jeliot

One of the high-end algorithm visualization tool created is Jeliot [10]. This tool takes in a working program and visually displays the execution sequence. It is divided into two stages, after the user enters the working program. It is required to compile the program with the built-in compiler for the tool to understand what the program is doing. Then the user can select “Play” to visually see the execution of the program. Figure 5 shows a screen shot of Jeliot during execution.

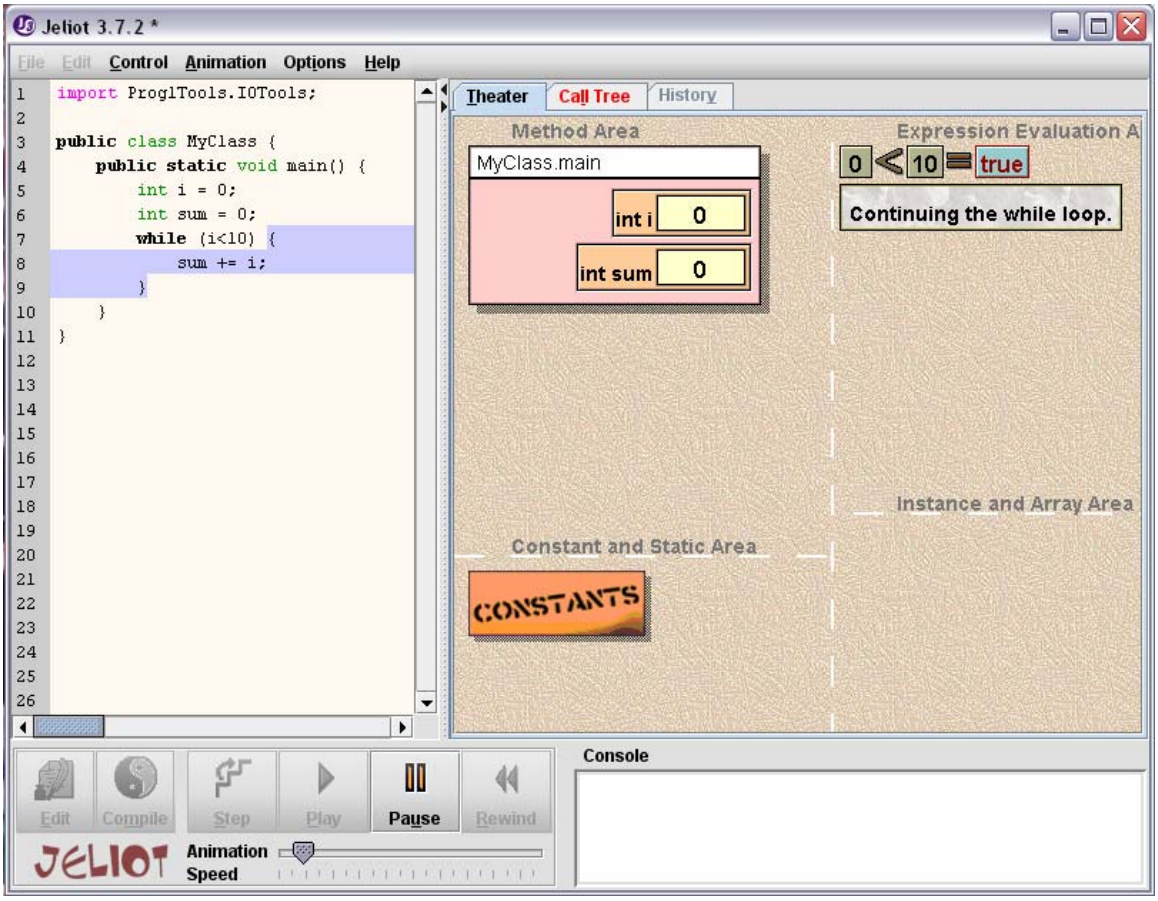


Figure 5 Jeliot screen shot.

Each line of code will be highlighted upon execution. The animation panel is then separated into several sections. The method area displays the program stack during execution. Expression evaluation is the scratch paper for the current executing statement. The constants and static area is the virtual box that creates constants needed during execution. The instance and array area shows behavior of arrays or objects for the program during execution. The tool is designed and implemented with usability in mind. Apart from the existing visualization tools, Jeliot requires very little effort in learning the tool. Instructors can simply pass in the program they would like to visualize and the tool will take care of the rest. This tool is user friendly and results are self explanatory. Arguable dis-

advantages might be lack of user interaction; as well as the resulting animation might not be as effective for programs with complicated structure.

2.6 TRAKLA2

TRAKLA2 [11] incorporated user interaction and a grading system with the animations. For each of the built-in algorithm, TRAKLA2 is able to take in user inputs and provide feed-back depending on the number of correct and incorrect moves made. The application is divided into several sections: instruction section and general working area. Figure 6 shows a screen shot of TRAKLA2 during execution.

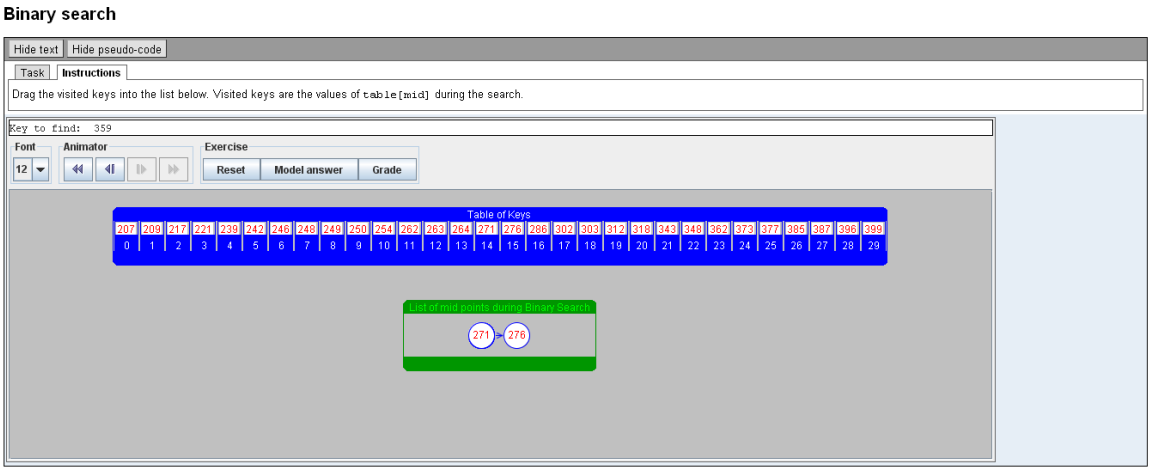


Figure 6 TRAKLA2 screen shot.

By following the given instructions, the user is asked to predict the next move that will be performed by the algorithm. Major advantages of TRAKLA2 are its visualization and ability to grade student's understanding for an algorithm through hands-on exercises and a large library of problems. Disadvantages for the application might include insufficient control for instructors to create animations that will be a better fit for their teaching style, as well as a rigid grading framework.

3. Framework Architecture

For this project, I implemented an algorithm visualization tool that can interact with users. It also gives control to instructors, allowing them to create their own custom animations, without the requirement of re-learning a new language, or becoming familiar with a new environment.

3.1 Sharma's Framework

Sharma [5] from San Jose State University created an animation framework targeting instructors teaching first year computer science students. This framework is designed to allow instructors to create animations through a set of simple Java API function calls. His API is able to create animations without requiring the instructors to learn another new tool or language. The animator UI contains four major components; label, animation

panel, button panel and direction panel. Instructors can specify the problem instance that is currently been displayed in the label panel. The animation panel is where the animation takes place. The button panel contains the actions that students need for interaction with the animation. The blue area underneath the buttons gives a brief description for the button selected. The direction panel gives detailed explanations regarding the animation's progress and hints for students regarding the next move. Figure 3.1 shows a screenshot of the animator during animation:

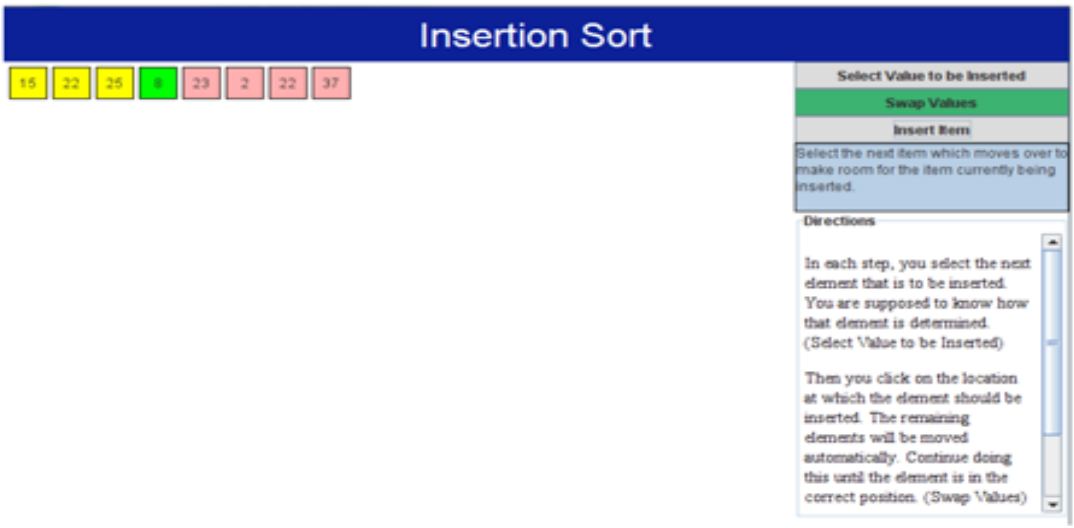


Figure 7 Sharma's algorithm visualization screenshot. [5]

In this example, the framework displays an interactive animation for the insertion sort algorithm. Upon execution, the framework generates an array of eight randomly selected integers. Students are expected to perform a series of actions to correctly sort the given array. The actions are done by selecting the buttons displayed in the button panel. At every stage of the algorithm execution, the animation halts to wait for the student's button selection. Each selection will then be validated, and with the correct selection, the animation will proceed to the next stage.

In the above image, the first three yellow-colored nodes indicate they have been correctly sorted. The student's next task would be to sort the next element in the array. The correct moves would be selecting the button "Swap Values" and the node 8. Again indicated by the color in the image above, both "Swap Values" button and node 8 are colored green, indicating they are currently selected. At this point, the animation halts and waits for the student's next move to indicate the correct location for the node to be inserted to. If the student is lost through the process, and unsure what the next step should be, the direction panel displays the detailed explanations of how this animation should be used. If the student is unsure which button to select, an explanation for each button is also displayed within the button panel. The explanation will appear upon selection of a button from the panel.

The main focus for this framework's architectural design is to create a simple set of API calls that can be used to easily create animations for any algorithm [5]. It should also be flexible enough for future tool extensions. This framework is built with Java, the JUNG

framework, and the Timing framework. JUNG stands for Java Universal Network/Graph Framework. It is a Java library which can perform analysis and visualization of data structures in the form of a graph or a network [3]. The JUNG framework provides a layout for graph representations.

The Timing framework helps make all the animations possible. The Timing framework library is divided into three basic components, the basics, interpolation and triggers. Two of the three compositions have been utilized in Sharma's animator framework. They are the basic and the trigger compositions. The basic composition is the logic which makes all dynamic effects, animations or time-based events on a Java application possible.

To demonstrate the level of simplicity of this framework, below is a set of code shown on the Timing framework website to compare the difference. First, let's consider the code for the original animation creation:

```
class MyActionListener extends ActionListener {
    public void actionPerformed(ActionEvent e){
        // Here's where all the logic goes:  check the event time and
        // see whether to reverse or end the animation based on the
        // time elapsed
    }
}

resolution = 30;          // update every 30 ms
begin = 50;               // wait for 50 ms before starting
ActionListener myActionListener = new MyActionListener();
Timer timer = new Timer(resolution, myActionListener);
timer.setInitialDelay(begin);
```

Listing 1 Original animation code [7].

In the above example, the user would like to create an animation, where an image is screened ten times, for a duration of 500 milliseconds each time. The image will also be updated on the screen every 30 milliseconds. In the above code, the user creates this animation using the original javax.swing.Timer package. As shown, the Timer class takes an integer for update intervals, and an ActionListener object to perform the animation actions. Thus, every time the user would like to create an animation, he/she would have to create a new ActionListener for each action the animation will perform. Now let's consider the following code which utilizes the new Timing framework.

```
repeatCount = 10;         // do the animation 10 times
duration = 500;           // each animation lasts 500 ms
resolution = 30;          // update very 30 ms
begin = 50;               // wait for 50 ms before starting
repeatBehavior = Reverse; // reverse at the end of each animation
endBehavior = Hold;       // hold final position
Animation anim = new Animation(repeatCount, duration, resolution, begin, repeatBehav-
ior, endBehavior);
```

Listing 2 Animation code with Timing Framework. [7]

In this case, the user only has to create an `Animation` object with the desired parameter values. There is no need to create additional and individual ActionListener objects for each Timer object as described above. Another major advantage of the Timing framework is that the original animation implements its speed according to specific machines at that time [7]. This is a problem because with the advancement in technology, old animations created will not behave in the same way the new machines compared to the original intended machine. The Timing framework solves this problem by basing the animation's speed on general elapsed time instead of environment depended timing event's frequency [7].

The Triggers composition is the feature that starts and stops the animation, so instead of the user implementing separate listeners for each animation type and manually keeping track of the starting and stopping of that animation. The Triggers composition wraps around various types of listeners and does the listening and starting and stopping of the animation for the user.

Sharma's framework itself is composed of several packages: algorithm, effects, engine, graph, and tools. The algorithm package contains Java animation files that will be created by instructors for each algorithm problem. The effects package contains all the animating effects that can be used on object nodes, edges or any other animation objects. The engine package ties together all the core components and framework to create animations. The graph package contains files for each animation object, such as nodes and edges. The tools package contains a list of button that can be used to allow student interaction with the animation.

Sharma also implemented a feature that will block the animation and wait for user input. Each of the buttons will be represented by a `SelectTool` object and the user interaction will be done by the `InteractionControl` class. During the animation execution, the correct tool will be expected by the function call `selectTool`. This function does several things such as displaying the tool tips in the direction panel and blocks all animation thread by setting the interaction flag to be true. Once the correct tool has been selected, the animation will continue to be blocked to wait for the correct parameter selection. The code displayed in Listing 3 shows the implementation of the blockage.

```
setWaitingInteraction(true);
while (r == null) {
    if ((mode & (AUTO | SHOW_STEP)) != 0) {
        Object tmp = null;
        if ((step & FIRST_STEP)!=0) {
            toolSelectAnimation(expTool);
            …
        if (tmp != null) {
            r = new PickedItemInfo(true,expTool,tmp);
        }
        else {
            hideCursor();
            synchronized(sync) {
                try {
```

```
                sync.wait();
            } catch (InterruptedException e) {      }
        }
        if (isInteractive()) {
            r = new PickedItemInfo(false,expTool,lastSelectedItem);
        ...
for(InteractionListener l : tmp) {
    l.itemSelected(r);
```

Listing 3 Animation blockage implementation.

In the above code, the execution will not exit the while loop if the item picked is null. Once an item has been selected, the function `itemSelected` will check for correctness. The algorithm will exit when the correct item has been selected, else the blockage will continue.

3.2 This Framework

Despite the advantages of Sharma's animation framework, there are still several concerns. First of all, the main purpose for this animation framework is to let instructors create animation without requiring them to learn a new language or a new tool. Sharma's framework is implemented in pure Java which is known to most computer science instructors. However, its dependency on the JUNG framework still creates a learning curve for instructors. This might even hinder the instructor's willingness to try creating any animations. Secondly, despite the fact that Sharma's animation framework is flexible enough to create animation for different problem instances, it is still rigid in showing different solutions to a particular problem. Computer science algorithms such as linked list do have a certain restrictive order of how the pointers should connect to each other among a list of nodes. However the order of a student selecting a node first or action button first should not be fixed.

Sharma's framework is based on the "student is the director" metaphor. The students using the animation created by instructors have the power to choose what will happen next in the animation for a certain algorithm. The students can direct the animation's execution through the usage of buttons. However this does not work so well. Deciding which button or action to select next is not always directly related to understanding a particular algorithm. For this project, button/action selection has been eliminated. This framework relies on the instructor to prompt for feedback when it is pedagogically beneficial.

Both Sharma's framework and this framework do not support multiple solution animation. Currently the purpose of this project is to create a framework that produces API to create an animation to be closely related to the natural code for the algorithm, with just a few `expect` and `doTogether` function calls intermingled. To accomplish multi-solutions for an algorithm can be a cumbersome task for the instructors.

For this animation framework, the Timing framework is kept for animation implementation, but it is transparent to animation implementers. The JUNG framework has been removed. Instead the animation objects were implemented with Java Graphics2D objects. In Sharma's framework, limited and restricted animation building blocks are created such

as Vertex, Edge, Graph, Network, List and Tree. In order to create a desired algorithm animation, instructors would have to first find out which structures (Graph, Network, List or Tree), that is a best fit for the problem. For this animation framework, all animation objects are stand alone entities. They are not bound to a list of rigid structures. Thus to create any algorithm animation, instructors can choose the animation objects that will best serve the purpose. For a complete list of objects available please refer to the following section.

4. How to Use The Animation Package

I created an API for producing structures that are common in CS education; see Figure 8.

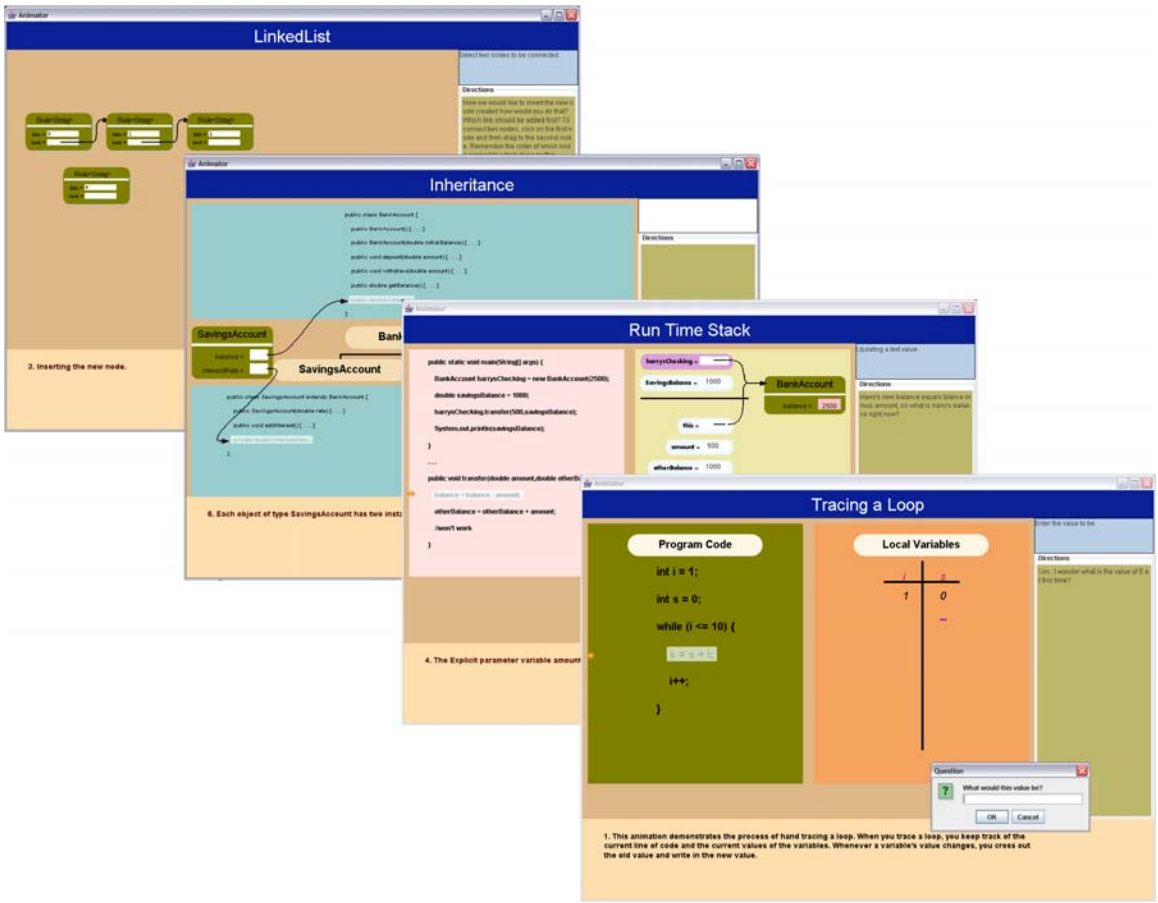


Figure 8 Screenshot of the animation framework.

Each of the components is explained in the components sub-section. More detailed examples will be introduced in the sub-section that follows.

4.1 Components

4.1.1 Node

A node object is extended to several different node objects: package node, object node, linked list node, label node, note node and variable node. Each node has its own characteristics and functionality. Below I will provide a detailed description for each.

4.1.1.1 Package Node

The package node is used to group all other objects together to be displayed in an orderly fashion. It takes several parameters such as x and y coordinates, width and height, as well as color. Four methods a package node uses to group all the other objects together are: `addNodeList`, `addCode`, `addText`, and `addLine`. The package node can be both invisible and visible as desired.

```
Color color = Color.BLACK;
PackageNode pn = new PackageNode(10,10,820,220,color);
pn.addCode(code);
pn.addNodeList(labelList1);
pn.addNodeList(labelList2);
pn.addTable(table);
```

Listing 4 Creating Package Node.

4.1.1.2 Object Node

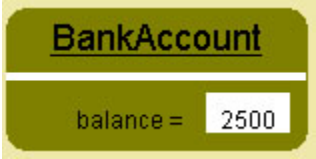


Figure 9 Object Node.

The object node is used to represent an object with a list of parameters. Each object node takes a string for the name of the node, as well as its desired coordinates to be displayed on screen. A user can then add a field node to the object node to represent the parameters of that object. The field node is composed of the name of the parameter and its string value. The string value can also be an empty string. Adding the field node can be done through the method call `addFieldNode`. An unlimited number of field nodes can be added to the object node. A sample of creating the object node is show below,

```
ObjectNode obj = new ObjectNode(label,50,50);
obj.setCoverNodeColor(176, 224, 230);
obj1.addFieldNode("balance = ", "");
obj1.addFieldNode("interestRate = ","");
```

Listing 5 Creating Object Node.

4.1.1.3 Linked list Node

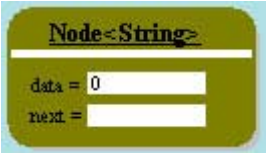


Figure 10 List Node.

The linked list node is used in the creation of a linked list. Currently, only a single linked list is supported. The linked list node is composed of two field nodes, one for the string data that is stored in this list node, the other is for the edge used to connect to the next list node in the list. The constructor for the linked list node is shown below. It requires two parameters, the name of the node, and the string data value for the data that is stored in the node.

```
ListNode node1 = new ListNode("Node<String>","0");
```

Listing 6 Creating List Node.

4.1.1.4 Label Node

Figure 11 Label Node.

The label node is used to create labels in the animation. They are one of the simplest forms of a node type. To create a label node, the user needs to provide the string value for the label that will be displayed on screen, as well as its desired x and y coordinates. If the user passes in 0 for x, then it will automatically be mapped to the middle of the package node. The user can also change the font of the label to be displayed, and the color for the label node itself. An example of the creation of the label node is displayed below.

```
LabelNode label1 = new LabelNode("Program Code",0,50);
```

Listing 7 Creating Label Node.

4.1.1.5 Note Node

The note node is used to display notification in the animation. For example, if there is a section of the animation that the user would like to put emphasis on by pointing it out and displaying a description line, then the note node will be the ideal choice. The difference between the note node and a label node is that the note node can display a sentence instead of a phrase.

```
note = new NoteNode("Insert header",30,40);
```

Listing 8 Creating Note Node.

4.1.1.6 Variable Node

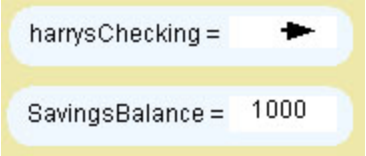


Figure 12 Variable Node.

The variable node is used to represent a single entity. For example, it can represent a variable with edge pointing to another node or code. It can also be used to represent a variable with a string data value. Two constructors are available to create a variable node. Both constructors take a string value for the node's variable name, its desired x and y co-ordinates, a string width limit and an alignment side. A user can align the variable node either on their left side or right side by simply passing in the string "left" or "right". The difference between the two constructors is that for the variable node that has a string value for its variable, an extra parameter is needed for its string content. The string width is needed to indicate the maximum width allowed for the string content of the variable node. Sample code is displayed below:

```
VariableNode funcInfo = new VariableNode(“harrysChecking”,” 1000”,x,y,50,
 “right”);
VariableNode funcInfo = new VariableNode(“balance”, x, y, 50,” right”);
```

Listing 9 Creating Variable Node.

4.1.2 Code

The purpose of this animation system is to help first year students learn the basics of computer science. Thus it is critical that the animation system is able to display the code that is being studied. To display code, a code object can be used. The code object will align itself in the package node it is assigned to. However, the user will need to specify how wide they would like to display the code. The code object takes in lines of code as an array list of strings. Some special features can be applied to the code object such as highlight a line or a phrase. Sample code is displayed below.

```
Code code = new Code(width);
ArrayList<String> c = new ArrayList<String>();
c.add("int i = 1;");
c.add("int s = 0;");
c.add("while (i <= 10) {");
c.add("    s = s + i;");
c.add("    i++;");
c.add("}");
code.addCodes(c);
/* code is displayed automatically */
```

Listing 10 Creating Code Object.

4.1.3 Text

Besides the code display, it is also crucial to display text values during the animation. One simple example would be to display the values of a variable throughout the iterations

of a while or for loop. Displaying regular text is easy; simply create a text object and pass in a string value, and the desired text coordinates. The user can also set the desired font and color for the text. Sample code is displayed below.

```
Font font = new Font("Dialog",Font.ITALIC,20);
Text it = new Text("i",ix,y,x2-x1);
it.setFontColor(font,Color.RED,BGBOX2C);
doInOrder(drawText(text1,i));
```

Listing 11 Creating Text Object.

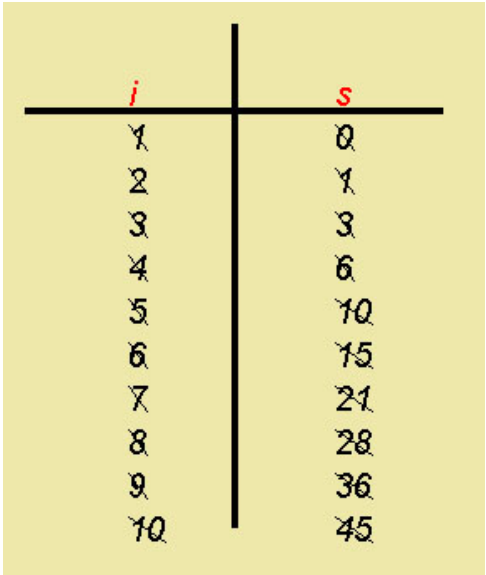
4.1.4 Line

To create a line object, the user needs to define the desired x and y coordinates and the width and height of the line to be displayed. Sample code is displayed below.

```
Line line1 = new Line(x,y,height,width);
line1.setColor(Color.BLACK);
doInOrder(drawLine(line1));
```

Listing 12 Creating Line Object.

4.1.5 Table



i	*s*
1	0
2	1
3	3
4	6
5	10
6	15
7	21
8	28
9	36
10	45

Figure 13 Table Object.

Despite the fact that users can use line and text objects to display a table, the coordinate's alignment for each text element in the table can be a tedious task. Thus for the convenience of the instructors, a table object is implemented. To create a table in an animation, the user will first need to call the table constructor and then pass in the table elements in columns. Each column of the table will be represented as a string array. Users can add-in as many columns as they desire. By default, the table's coordinates are set to the center of the package node they are assigned to. But the user can also assign desired coordinates for the table created. Below is a sample code used to create a table object.

```
String[] column1 = {"i","1","2","3","4","5","6","7","8","9","10"};
String[] column2 = {"s","0","1","3","6","10","15","21","28","45","55"};
private Table table = new Table();
table.addColumn(c1);
table.addColumn(c2);
table.setTableFont(table_f);
```

Listing 13 Creating Table Object.

To create an algorithm animation, the user should first select the objects they would like to use. Then they can apply desired effects to each of the objects to create the animation. Detailed examples will be given in the following section.

4.2 Concrete Examples

To create a desired animation, the user will first need to decide which components are needed to represent the problem at hand. Attributes for each of the components will then need to be considered and set. The animation is created by deciding the order in which components are displayed. Other special effects can be applied to each object to make the animation attractive. User interaction can also be inserted into the animation with simple function calls. In the following sub-sections, four different animations will be discussed to show how to build an animation in detail.

4.2.1 Linked List Demo

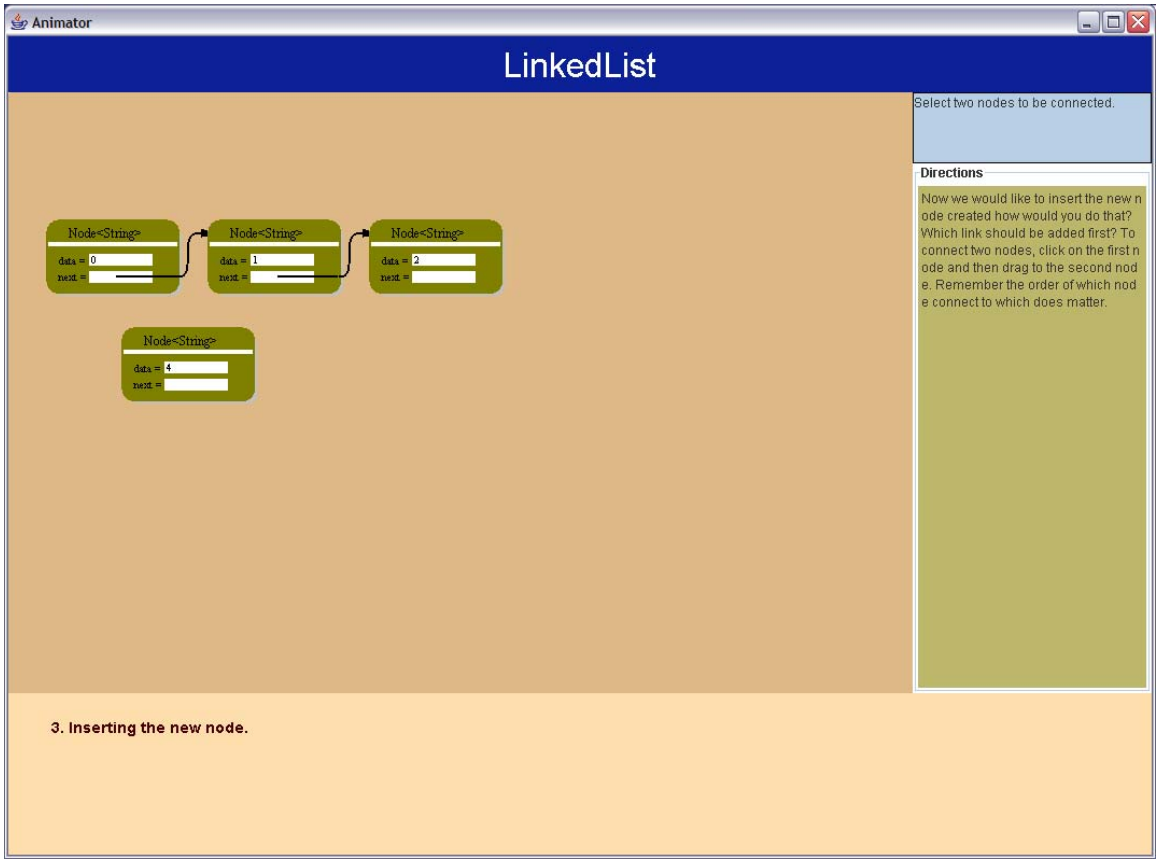


Figure 14 Linked-list Animation.

In this example, I will demonstrate how to create an animation that displays inserting a node into a linked list. Figure 12 displays a screen shot of the animation created.

Inserting a node in the beginning or end of a linked list is not complicated. Inserting a node in the middle is where things get tricky. This example shows the user how a link list node should be properly inserted into the linked-list.

Before creating the animation, the user need to first decides what and how many components will be needed to create this animation. In this example, linked list nodes will be needed to create a linked list demo. For the basics of creating a linked list node object, please refer to sub-section 4.1.1.3. To create a linked list node, user would have to decide on the nodes' data values as well as their coordinates. Once the node has been created, they were grouped together into a package node for printing. Code is displayed in Listing 4.2.1.1.

```
PackageNode pn1 = new PackageNode(0,0,0,0,BGBOXC);
ListNode node1 = new ListNode("Node<String>","0");
ListNode node2 = new ListNode("Node<String>","1");
node1.init(xloc, yloc);
lnode.add(node1);
pn1.addNodeList(lnode);
packNode1.add(pn1);
VisualizationPanel.getMe().setPackageNode(packNode1);
```

Listing 14 Create list node and group it to the appropriate package node.

To create the animation, the user needs to decide the order for the node to be displayed on screen for the animation. This example started with displaying the first original three linked list nodes on screen. To display a node, the user can call `displayNode` method, which takes the node to be displayed as a parameter. The user can display the node together or in order, using the two functions `doInOrder` and `doTogether`. The code is shown below.

```
doTogether(displayNode(node3),glow(node3));
```

Listing 15 Display animate effects to the node just created.

After creating two list nodes, the next thing to do is connect them. First two of the existing three nodes on screen will be connected. To connect two nodes through an edge, the user needs to create the edge for the two nodes first. Then the animation function `linkNodeNode` will be called to draw the edge on screen.

```
ListNodeEdge node1edge1 = node1.connectNode(node2);
doInOrder(linkNodeNode(node1edge1,node1,node2));
```

Listing 16 Connecting two list node.

After connecting all three existing nodes on screen, a fourth node is created to be inserted in-between the first and second node. Inserting a new node into the list requires knowledge on how to reference the nodes to be connected. This is where instructors would like students to perform the task to check their understandability toward the problem. To enter user interaction, an action needs to be created and selected. The instructor should also pass in the correct objects that the tool is expecting. Listing 16 displays the tool creation and usage.

```
TwoNodeTool select = new TwoNodeTool();
addTool(select);
selectThisTool(select);
select.expectSelectTwo(node1,node4);
```

Listing 17 Interaction tool creation.

In this case, `node1` and `node4` are expected. If the student fails to do this, two options are available. First they can try to select two nodes to be connected again, or give up and the animation will automatically display the correct move.

After correctly inserting the new node, it has to be moved into the list. Moving the node will require shifting the other two nodes to the right. To do that, the user can call the `moveNode` method. This function takes in three parameters: first the node to be moved. Second, the original coordinates of the node to be moved. And lastly the x and y difference for the node to be moved.

```
doInOrder(moveNode(node4,node4.getNodeLoc(),difx,dify));
```

Listing 18 Moving a node.

To display the caption, the user needs to prepare a story.txt file. This file should consist of a list of captions that will be used. Each line of the caption needs to be numbered from 1 to n for easy access. The `StatusPanel` object will load the file and display one line at a time as desired. Below is a simple example of how to load and display a line of caption into the animation.

```
StatuPanel.getMe().loadCaption(capF,capC,captionfilename);
doInOrder(drawCaption(1,2));
```

Listing 19 Loading caption into the animation.

4.2.2 Inheritance Demo

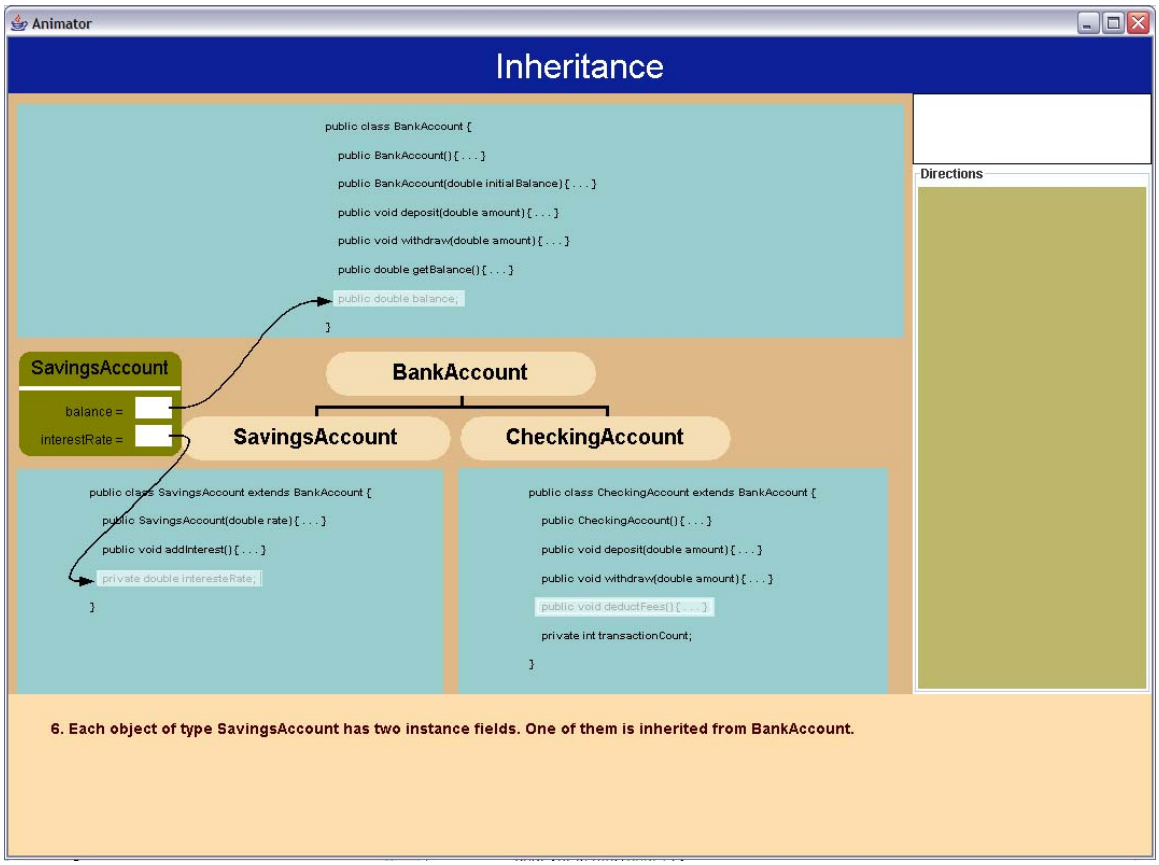


Figure 15 Inheritance demo screen shot.

Components that are needed to create this animation are: code, line, label node, object node, caption and package node. In this example, different objects are needed in comparison to the previous example. To better organize the objects to be displayed, package nodes are utilized.

First the caption is loaded into the animation through the `loadCaption` method as described below. This is done in the same fashion as in the previous example.

```
StatusPanel.getMe().loadCaption(capF,capC,captionfilename);
doInOrder(drawCaption(1,2));
```

Listing 20 Loading caption into the animation.

Creating the label nodes can be done through calling its constructors. Label nodes are grouped together for better organization.

```
NodeList list = new NodeList();
double x = 420-250/2;
LabelNode label1 = new LabelNode("BankAccount",x,240);
list.add(label1);
x = 410-250;
LabelNode label2 = new LabelNode("SavingsAccount",x,300);
list.add(label2);
LabelNode label3 = new LabelNode("CheckingAccount",420,300);
list.add(label3);
```

Listing 21 Creating three label nodes.

In this example, three package nodes are used. Each of the package nodes in this example will have their own code object. One advantage of the package node is that, it can also display an object which does not strictly belong to its area. So the three label nodes and the two object nodes in this example can be assigned to any of the package nodes. Code shown below:

```
/* creating three package nodes */
PackageNode pn1 = new PackageNode(10,10,820,220,BGBOXC);
PackageNode pn2 = new PackageNode(10,350,400,200,BGBOXC);
PackageNode pn3 = new PackageNode(420,350,400,200,BGBOXC);
/* assigning each package node with objects */
pn1.addCode(code1,10,10);
pn1.addNodeList(list);
pn2.addCode(code2,10,350);
pn2.addNodeList(funcList);
pn3.addCode(code3,420,350);
/* group the package node together */
packNode1.add(pn1);
packNode1.add(pn2);
packNode1.add(pn3);
/* pass to visualization panel for printing */
VisualizationPanel.getMe().setPackageNode(packNode1);
```

Listing 22 Package node and its usage.

Code objects that are created by calling its constructor as shown in subsection 4.1.2. In this example, two animation effects are introduced: highlighting a code phrase and connecting a node edge to a line of code. Highlighting a phrase within a line of code can be done through a code object's `addPhraseBox` method. For this method, the user would pass in the occurrences of the phrase that appeared in the code. For example, if there are two "this" strings in a line of code, and the user would like to highlight the first "this", then 1 should be entered as the first argument. The second parameter is the line number for the line of code that has the phrase to be highlighted. The third parameter is the string of phrase to be highlighted. To move the highlight box, the `updatePhraseBox` method

should be called. The same arguments are needed, with one extra parameter for the code object that contains the phrase which needs to be highlighted. The code is shown below:

```
code1.addPhraseBox(1,4,"public void deposit(double amount) { . . . }");
code1.addPhraseBox(1,5,"public void withdraw(double amount) { . . . }");
code1.addPhraseBox(1,6,"public double getBalance() { . . . }");
code2.addPhraseBox(1,1,"extends BankAccount");
doTogether(updatePhraseBox(code2,1,1,"extends BankAccount"),
                updatePhraseBox(code3,1,1,"extends BankAccount"));
```

Listing 23 Highlighting a phrase within a line of code.

Connecting a node object with a code object can be done in similar fashion as connecting two node objects together. First the user needs to create a `ListNodeEdge` object, by calling the node's `connectCode` method. The user can use this method to specify the source node and the target node that will be connected. The first parameter this method takes is the source node. The second parameter specifies the target node. To display the animation, the function `linkNodeCode` can be used. Parameters needed are the object node that will initiate the connection, the string name of the field node of the object node, the code object to be connected to and its line number.

```
ListNodeEdge obj1edge1 = obj1.connectCode("balance = ", code1);
ListNodeEdge obj1edge2 = obj1.connectCode("interestRate = ", code2);
ListNodeEdge obj2edge1 = obj2.connectCode("balance = ", code1);
ListNodeEdge obj2edge2 = obj2.connectCode("transactionCount = ", code3);
doInOrder(linkNodeCode(obj1edge1,obj1,"balance = ",code1,7),
                  linkNodeCode(obj1edge2,obj1,"interestRate = ",code2,4));
```

Listing 24 Connecting between a node object and a line within a code object.

4.2.3 Run Time Stack Demo

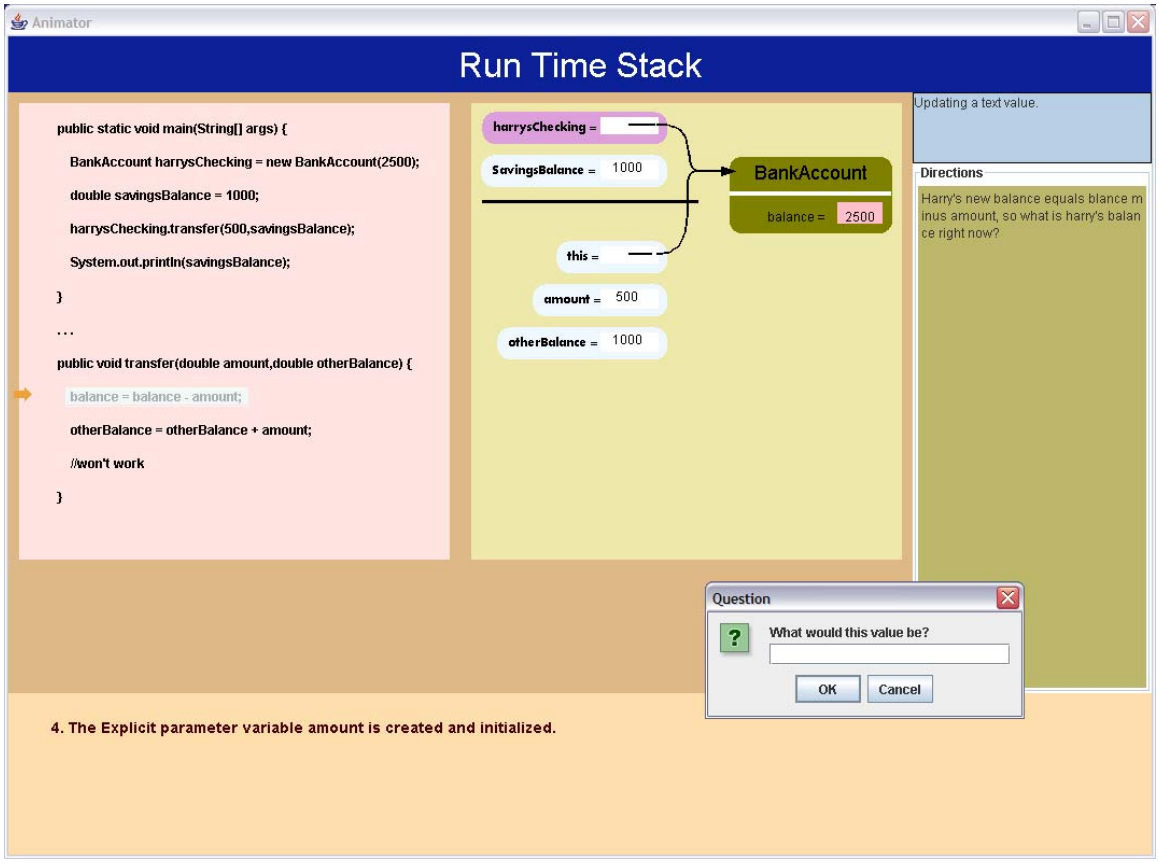


Figure 16 Run time stack screen shot.

This animation is created to demonstrate how a variable's value changes throughout the program execution. Components that are needed for this animation are code object, line object, variable node and object node. For details on loading captions please refer to the previous two examples. For details on the Code object creation, please refer to sub-section 4.1.2.

Three new features worth mentioning here are: the code phrase arrow, the coloring of the variable node and the field node of the object node. It is also possible to change the value of a field node for an object node. Usually, besides highlighting a certain variable or a line of code, it is also convenient to have an indicator to show the execution sequence of a program. The phrase arrow is used for instructors to point at the beginning of a line of the code object. By moving the arrow up and down, the instructor can show the execution sequence of a program or simply put emphasis on a particular line of code. To add an arrow to the code object, the method `addPhraseArrow` can be called. One parameter is needed to indicate which line the arrow will stop at. Once the arrow is added to a particular line of code, it does not matter which order the arrow moves. The sample code is shown below.

```
code.addPhraseArrow(4);
doInOrder(updateArrow(code,4));
```

Listing 25 Add phrase arrow to the beginning of a line of code.

Changing the background color of a variable node or a field node of the object node can be done through two effects function call, `highLightField` and `highLightNode`. The `highLightField` method takes an object node and the field string name as arguments. It needs to know which field node of which object node will be highlighted. The `high-LightNode` method takes the variable node that will be highlighted as an argument. The sample code is shown below:

```
doInOrder(highLightField(obj,"balance = "), highLightNode(var1),
                changeFieldNodeValue(obj,"balance = ","2000"));
```

Listing 26 Highlighting a field node, and change a field node's value.

4.2.4 Tracing a Loop Demo

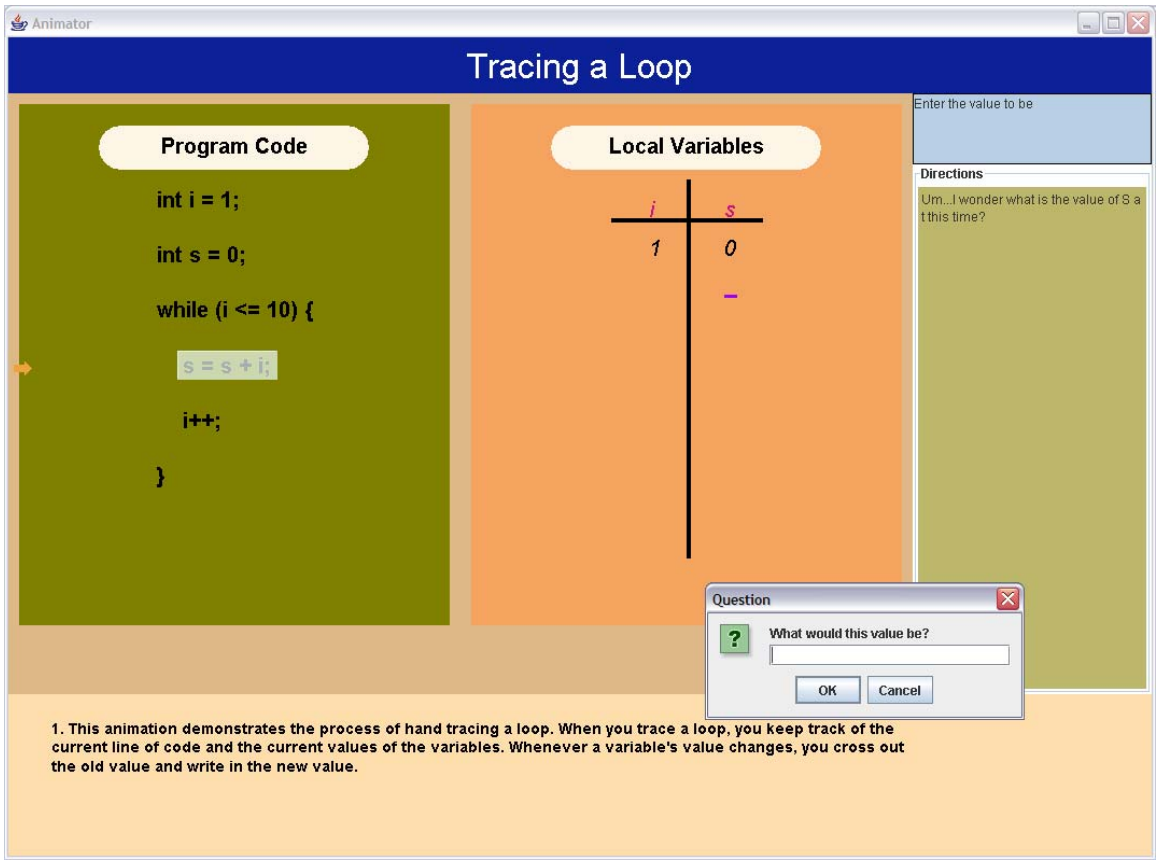


Figure 17 Tracing a loop screen shot.

Animations can also be created with text objects. The purpose of this animation is to show the value change for a variable through out a while loop. Several objects are needed, label node, code, text and line. For each of the objects' creation please refer to the components section.

The variable `i` will be incremented from 0 to 10 and the variable `s` will be incremented by value `i` in every iteration. The code is displayed first on the left hand side of the panel, then moving the arrow and highlight box to indicate the execution process of the pro-

gram. On the right hand side of the panel, two lines are drawn to create a table for both variable `i` and `s`. For each iteration, values for variable `i` and `s` need to be updated. In this animation, variable `i` is updated automatically, while variable `s` requires the user's interaction.

Initially, creation of the table for this animation is done through text and line objects. However, due to the complication of coordinates of each table element, table objects were created. For details regarding the creation of the table object, please refer to subsection 4.1.5. Two features worth mentioning are drawing the line to cross a text, and incorporating user interaction for the next correct value for variable `s`. Displaying a text object can be done through the simple function call `drawTableText`. Drawing a line over the text to cross out the value can be done through the function call `drawTableTextLine`. Sample code is shown below.

```
doInOrder(drawTableText(table,"s",es),drawTableTextLine(table,"s",old_es));
```

Listing 27 Display table element and draw line across the table element.

After displaying the objects required for the animation and the initial value for variable `i` and `s`, the animation halts and waits for the user to select the correct tool. A window pop-up will prompt the user for the next correct value for variable `s`. If the correct value is entered, the animation will move onto the next iteration and prompt the user for the animation tool and value again, until the end of execution. Sample code can be seen below.

```
enterValue(select,es);
```

Listing 28 User interaction for string value.

5. Evaluation

Due to the different teaching styles and personal preferences many algorithm visualization tools developed have not been widely utilized [1]. Reasons for the under utilization of the tools are many. In order to develop a good algorithm visualization tool, three things should take into consideration: teacher empowerment, student interaction and familiar framework.

5.1 Teacher Empowerment

Ben-Ari et al [1] did a study on what are instructors looking for within a software tool that is designed to help them in making learning experiences better and easier. As a result, they came up with two important factors. First, an instructor's interest in utilizing a tool relies heavily on the “integrating the tools with other learning materials”. Second, it also depends on “addressing the role of the teacher in the use of software by the students”.

From their year-long study, it was shown that students using the visualization tool perform better compare to the students who do not. Thus it is the instructor's responsibility

to expose students to any new tools that might help them in achieving an academic course. However, they found that the number of instructors that are open for innovative ideas is inconsistent to the number of instructors who are actually doing it. Ben-Ari et al believe that the major reason for this inconsistency is due to the inability for instructors to accept changes in their routine procedures. They called such behavior “centrality of the teacher”, where experienced instructors refused to accept any changes that might jeopardize their role of authority in teaching the subject that they are most familiar to.

Developers devoted tremendous amount of time on the “technical capabilities” of the tool, while ignoring how it will integrate with the rest of the learning materials [1]. Numerous visualization tools such as Jeliot did not get widely utilized because they are standalone applications. They take an existing problem instance and solve it in a way that tool developers see it, without thinking how the instructor might like it to be done. Because of this, “centrality of the teacher” is being taken away, and instructors will mostly likely avoid a tool that will replace them in a classroom. The application created for this project solved this problem by leaving instructors in charge. A list of available APIs is created and instructors are able to create the animations that best fit their needs.

5.2 User Interface

Taylor et al [19] did research comparing two different types of user interactions in animation learning tools. Their study shows that algorithm visualization with user interaction incorporated produce better results compared to pure animation. The first type of interaction they talked about is “passive” interaction. This type of tool allows user to interact with the animation in its control level. The user is able to adjust the animation speed, stop or restart the animation or rearranging the animation objects/data values as desired. This type of user interaction usually does not involve intelligent interaction with its user. The second type of interaction they defined is “predictive” interaction. Different from the first type of user interaction, “predictive” interaction allows the user to predict the next move for a particular algorithm in its execution. With the appropriate feedback, this achieves the “in-class effect”.

For their study, with the selected group of the students, they let them use both types of interactive animation tools but each with a different order. For example, group A will try “passive” interaction first then “predictive” interaction next, while group B will try “predictive” interaction first and “passive” interaction next. The result is pretty clear that students who have seen “predictive” interaction first perform much better than students who used “passive” interaction tool first. More importantly, they argue that students who have seen animation with some type of interaction out performs students who does not use animation at all or use animation without any types of interaction.

For this project, instructors have control over creating the animation they would like to use. They can also incorporate user interaction at desired locations throughout the animation. Freedbacks is also given to students after each move.

5.3 Optimized to Task

In researching for the language to use to create the animation API, two well known dynamic animation generation tools, JavaFX [20] and Flash are studied. Flash is yet another popular tool for creating fancy animation with minimal effort. However a domain specific language such as JavaFX is not sufficient enough to create sophisticated and reusable animation quickly. Figure 17 is the code that displays an animation which draws node objects and connects edges between them.

```
def newnode = Rectangle {
                effect: DropShadow {};
                translateX: 400,
                translateY: 120
                width: 50,
                height: 50
                arcHeight: 20
                arcWidth: 20
                fill: Color.BEIGE
            };
def path = Path {
    elements: [
        MoveTo{x: 130          y: 420 },
        CubicCurveTo { controlX1: 220  controlY1: 390
                            controlX2: 220  controlY2: 456
                                    x: 310          y: 420 },
        CubicCurveTo { controlX1: 400  controlY1: 390
                            controlX2: 460  controlY2: 490
                                    x: 525          y: 420 },
         CubicCurveTo { controlX1: 590  controlY1: 355
                            controlX2: 640  controlY2: 455
                                    x: 730          y: 420 },
    ]
};
def anim = PathTransition {
        path: AnimationPath.createFromPath(path)
        orientation: OrientationType.ORTHOGONAL_TO_TANGENT
        node: newnode
        interpolate: Interpolator.LINEAR
        duration: 8s
        repeatCount: Timeline.INDEFINITE
  };
```

Listing 29 JavaFX Code Sample.

In the above code, a user would have to define each component that will be used for the animation. This severely limits reusability of the code and causes the resulting code to be bulky and hard to manage. The same animation can be accomplished with eight lines of code by using my APIs created for this project.

To do an animation in Flash that moves and glows objects, redirects arrows or crosses out text can be a tedious and repetitive task. Though creating animation can be done through buttons and tools, users would need to learn scripting and programming using Flash to

effectively produce these animations. Moreover, creating animation with Flash requires some level of design skill and creativity. Instructors usually do not have and do not wish to spend too much time in designing the visual effects of the animation they are creating. For this project, instructors can use the readily made animation objects without worrying about the animation's visual effects.

From the research, listing 27 displays the overall levels of usability between this framework, JavaFX and Flash.

	Algorithm Learning Framework	JavaFX	Flash
Reusability	**High**	Medium	Low
Tool requirement	**Low**	Medium	High
Language framework	**Low**	High	High
Lines of code	**Low**	High	High
Visual Effects	**Medium**	High-Medium	High

Listing 30 Summary of tool comparison.

The visual effects created by this application might not be as good as animation created by domain specific tools such as JavaFX and Flash, but I believe advantages such as high reusability, low tool requirements, no need to learn a new language and fewer lines of code are needed, will make this tool more attractive to instructors.

This project is designed with instructors in mind and how to better address their needs. It is not a stand alone application. Instructors are able to create the visualization they desire, they can also choose to incorporate student interaction if they like. Creating animation with this API can be done with a language that is familiar to most instructors. However, depending on the personal preference and requirements, this project will not satisfy all instructors need. For example, instructors who prefer minimum effort to create the animation, writing code to create the animation may still be discouraged.

6. Conclusions

Over the past decade, hundreds of visualizations have been researched and developed. Only a few instructors are actually using these tools in their daily routines. Reasons are many; some believe that the visualizations have bad quality; while others argue that visualizations created are for problems that are too simple to provide any real value. Recent studies shown that many tools either ignore instructor's needs or incorporate limited user interactions.

The purpose of this project is to avoid these limitations and create a framework that will be user friendly to most instructors. Three factors that are important when creating an active learning framework involves: teacher empowerment, user interaction and optimized to task. Sharma's framework incorporated these three factors in some form. For this project, I have rewritten majority of the framework structure to better achieve the effectives based on these three requirements. First of all, this framework continued to use Java and eliminated the existing JUNG framework. This has significantly reduced instructor's ef-

fort and time in learning this tool that might be new to them. Secondly, by rewriting Sharma's framework, creating an animation is now much more flexible and more time can be spent on the actual design of the animation. Lastly, by eliminating the usage of buttons, student interaction is simplified.

Though limitations of existing tools have been addressed in this project, there are some features that have been sacrificed, for example, features such as automatically generating algorithm animation from pseudo-code. This is inevitable due to different instructors have different needs and preferences. It is almost impossible to cover all of them. However, the framework is sufficiently general that it can be enhanced to provide for animations that will be of interest to other instructors.

References

[1] Mordechai Ben-Ari, Ronit Ben-Bassat Levy, "We Work So Hard and They Don't Use It: Acceptance of Software Tools by Teachers", ITiCSE'07, June 23-27, 2007. Dundee, Scotland, United Kingdom.

[2] Clifford A. Shaffer, Matthew Copper, Stephen H. Edwards, "Algorithm Visualization: A Report on the State of the Field", SIGCSE'07, March 7–10, 2007, Covington, Kentucky, USA.

[3] http://jung.sourceforge.net/, Mar. 1, 2008.

[4] http://www.sun.com/software/javafx/index.jsp, 2007.

[5] Sean Sharma, "A Framework for Active Learning", Master Thesis report, May 2008.

[6] Ali Erkan, T.J. VanSlyke, Timothy M. Scaffidi, "Data Structure Visualization with LaTeX and Prefuse", ITiCSE'07, June 23–27, 2007, Dundee, Scotland, United Kingdom.

[7] https://timingframework.dev.java.net, 2007.

[8] http://wiki.algoviz.org/AlgovizWiki/, 2010.

[9] http://video.google.com/videoplay?docid=3970523862559774879#, 1981.

[10] http://cs.joensuu.fi/jeliot/, 2007.

[11] http://wiki.algoviz.org/AlgovizWiki/Trakla2, 2009.

[12] http://wiki.algoviz.org/AlgovizWiki/Animal, 2000.

[13] http://wiki.algoviz.org/AlgovizWiki/AlgorithmsInAction, 1999.

[14] Raghvinder S. Sangwan, James F. Korsh, Paul S. LaFollette, Jr, “A System for Program Visualization In the Classroom”, SIGSCE 98 Atlanta GA USA

[15] Michael T. Helmick, “Interface-based Programming Assignments and Automatic Grading for Java Programs”, ITiCSE’07, June 23–27, 2007, Dundee, Scotland, United Kingdom.

[16] Thomas Naps, Guido RoBling, Jay Anderson, eta., “Evaluating the Educational Impact of Visualization”, Report of the Working Group on Evaluating the Educational Impact of Visualization"

[17] David Carlson, Mark Guzdial, Colleen Kehoe, Viren Shah, John Stasko, “WWW Interactive Learning Environments for Computer Science Education”, SIGCSE ’96 2/96 Philadelphia, PA USA

[18] Duane J. Jarc, Michael B. Feldman, Rachelle S. Heller, “Assessing the Benefits of Interactive Prediction Using Wed-based Algorithm Animation Courseware”, SIGCSE 2000 3/00 Austin, TX, USA

[19] David Scot Taylor, Andrei F. Lurie, Cay S. Horstmann, Menko B. Johnson, Sean K. Sharma, Edward C. Yin, “Predictive vs. Passive Animation Learning Tools”, SIGCSE’09, March 3-7, 2009, Chattanooga, Tennessee, USA

[20] http://javafx.com/, 2010