# A Tool for Data Structure Visualization and User-defined Algorithm Animation


## Tao Chen, Tarek Sobh and Abhilasha Tibrewal

### University of Bridgeport



## Abstract

In this paper, a software application that features the visualization of commonly used data structures and their associated insertion and deletion operations is introduced. In addition, this software can be used to animate user-defined algorithms. Examples illustrating the functionality of the software as a supplemental teaching tool are discussed.


## 1. Introduction

Data Structures and Algorithms is a fundamental course in Computer Science. However, many students find it difficult because it requires abstract thinking. It would be very helpful if there were a visualization tool of data structures such as arrays, queues, stacks, trees and graphs for students to experiment with. The tool would allow students to see the workings of common insert/delete operations in form of changes that take place to the corresponding data structure. Moreover, this tool would provide a simple language, by which students can write their own algorithms so that the execution of the algorithm is animated. This project is intended to create such an exploration environment, in which students can learn through experimentation. This tool can be used as an effective supplement to the traditional classroom education and textbooks for Data Structures and Algorithms courses. The software package presented in this paper has the following functionality.

a. Provides complete visualization for the widely used data structures such as array, stack, queue, tree, heap and graph.
b. Provides the animation of common operations associated with the data structures, such as inserting an element into and deleting an element from the specified data structures.
c. Provides animation of simple user-defined algorithms.

The rest of the paper is organized as follows. In Section 2, we discuss some of the currently existing tools; Section 3 entails a detailed discussion of the software package both in terms of design and functionality to bring out the pedagogical effectiveness of the tool; and Section 4 includes the concluding remarks.

## 2. Background

The development of technologies and the evolvement of the World Wide Web have influenced education. Instructional Web sites and courses on the Web have grown dramatically. Web-based courses that consist of the syllabus, assignments and lecture notes are now widely used. Instructional Web sites that are dedicated to Data Structures and algorithms can be easily found by using Search Engines. To name a few:

http://swww.ee.uwa.edu.au/~plsd210/ds/ds_ToC.html [1]

http://www.cee.hw.ac.uk/~alison/ds98/ds98.html [2]

http://www.cs.twsu.edu/~bjowens/cs300/ [3]

http://www.cs.berkeley.edu/~edith/cs270/ [4]

However, The majority of the instructional web sites explored during this project lack interactive multimedia.

One of the best sites found that does contain interactivity is a course site developed for teaching Data Structures and Algorithms in Java by the Computer Science Department of Brown University [5]. This site has a collection of applets that demonstrate some commonly used data structures such as queues, stacks, and some famous algorithms such as merge sort, quick sort, etc. However, these applets are not complete and lack a common Graphical User Interface. Another good site in interactive Data Structure visualizations is developed by Duane J. Jarc in George Washington University [6]. This site provides animations in binary Trees, graphs, and sorting algorithms. But there is no animation available for algorithms that are defined by users.

Algorithm animation is a type of program visualization that is mainly concerned with displaying the executions of computer algorithms. Lots of work has already been done in this field. For example, the XTANGO [7] and POLKA [8] systems developed by the Graphic, Visualization and Usability Center (GUV) at Georgia Tech are general-purpose animation systems, which require the user to write an algorithm in the C language and register the events that the user wants to observe during the execution of the algorithm. However, these systems are implemented on top of Unix and X11 Window system, and are not portable to other platforms. In addition, we feel they are overkill for a basic Data structures and Algorithms course.

Another algorithm animation system found is Zeus [9], which is developed by Digital Equipment Corporation's Systems Research Center. This system is a little complicated, requiring from the user lots of effort to prepare animations. It is targeted at more advanced application programmers.

Other researched software included SWAN [17], TRAKLA and TRED [14], ANIMAL [16] AND JAWAA [15]. Swan focuses on algorithm animation involving graphs and is platform dependent. TRAKLA and TRED are parts of a very powerful system with built in assessment and automatic administration of the course. The TRAKLA server is at the heart of the system. The authors, however, do not mention the availability of the system. ANIMAL is platform independent, requires no programming code, the GUI

features animation speed control, code and element highlighting and marking. It meets its objective of wider applicability such that it is not limited to algorithm or data structure animation. It is applicable for introductory courses with students form different majors rather than students with pure computer science interests.

JAWAA uses a simple command language for creating animation. JAWAA's objects are designated in a similar manner to those in Samba, one line commands for creating and moving objects. With Samba, animations are built using only primitive objects, but with JAWAA data structures can be created easily in one-line commands. JavaMy, inspite of its close resemblance to JAWAA has additional and unique characteristics of its own as described below.

JavaMy provides built in data structure visualization for common operations like inserting and deleting with respect to corresponding data structures. The algorithm animator features parallel symbolic and iconic representations in form of animation and code to make the tool pedagogically effective. Since our software is intended to aid first year Computer Science students learning Data Structures and Algorithms, ease of use becomes our main consideration. Our approach for the user-defined algorithm animation is that the user codes the algorithm in a simple language called JavaMy, which is very similar to Java. The only effort the user needs to make is to instantiate the data structures he/she wants to observe using the observable data types provided by the software. After parsing the JavaMy algorithm file, an animation frame is created and the observable data structures are added to the frame so that the user can watch the changes made to the data structures when the algorithm is executing.

## 3. Software Package

In this section, an attempt is made to put forth a comprehensive description of the software to exemplify the different capabilities of the system. A general introduction to the software shall commence the description, followed by a shifting focus on features of graphical user interface, the software's effectiveness as a data structure visualization tool and its effectiveness as a user-defined algorithm animator. The above mentioned sections include examples to further the pedagogical effectiveness of the tool.

### 3.1 Overview of the Software

The JavaMy software has been developed aiming at its use as a pedagogical tool in the fundamental data structures and algorithms course. The targeted audience thus, is the student community undertaking such a course or involved in self-study of the above concepts. The above fact makes ease of use the primary goal of this software. At the same time, the software design is competent enough to follow the "ten commandments of algorithm animation" described by Gloor [12]. Consistent design, interactivity, keeping the user engaged, incorporating symbolic and iconic representations are thus, some of the hallmarks of the software. More involved discussions on the same are included in the next section.

JavaMy is a visualization tool that can be used to supplement teaching of data structures and related algorithms. It can be used in the classroom as a teaching aid in lectures and in the lab setting to encourage exploration learning. The software functionality can be divided into two broad areas viz. the built-in data structure visualizations and the facility of user-defined algorithm animation. The observable data structures currently available in this software packages include: array, stack, queue, binary search tree, heap and graph.

The software uses JavaMy as the programming language. JavaMy has been chosen for its simplicity thus, making it suitable for the beginners. Moreover, the syntax of the language is close to programming languages such as Java and in our opinion makes it a better choice as compared to command line languages such as the one used in the JAWAA software [15].

An overview of the software would be incomplete without the implementation specifics of the software. JavaMy is implemented using Java, thus making it platform independent and portable, which goes a long way in making it our natural choice for implementation. Moreover, the AWT and SWING packages of Java provide extensive components for creating the GUI. To animate a user-defined algorithm, a lexical analyzer and parser are needed. A lexical analyzer breaks an input stream of characters into tokens. A parser reads the input tokens and converts the tokens to a Java program. There are several ways to build a lexer and parser. One possibility would be to code the lexical analyzer and parser completely from scratch, implementing all string handling and checking functions, which is a very tedious and error prone process. Another method is to find a Java parser generator, which reads a grammar specification and converts it to a Java program that can recognize matches to the grammar. After in-

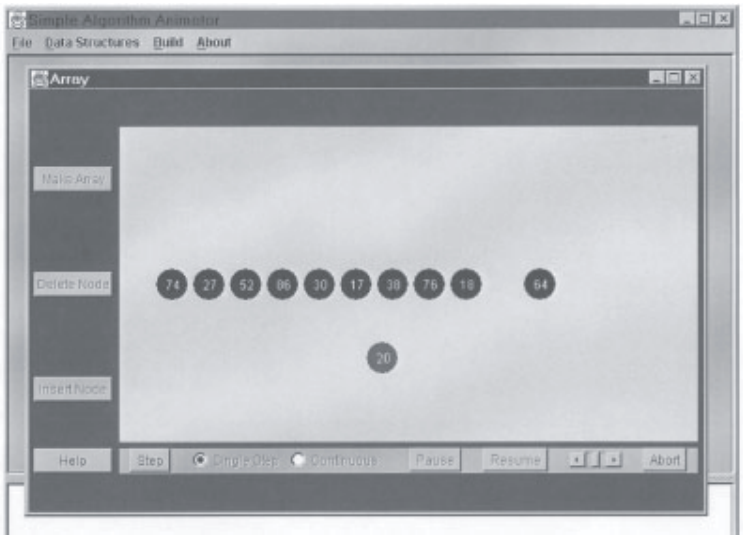


*Figure 1:* ***Insert 20 into array***

tensive search, we found that JavaCC [10], a product of Sun Microsystems is currently the most popular parser generator for use with Java applications. Consequently, it was our choice. The parser is generated by two steps: (1) Run JavaCC on the grammar input file to generate a set of Java files that implement the parser and the lexer. (2) Compile all the Java files obtained in step (1).

### 3.2 Features of the Graphical User Interface

In designing the GUI, a lot of focus has been on keeping it consistent such that the user finds it easy to use. Gloor [12] gives top priority to consistency among "the ten commandments of algorithm animation". The control interface is always at the bottom of the screen in form of a toolbar leaving the entire window for the animation, thereby, meeting the goal of emphasizing visual component. The following is a snapshot of inserting an element into an array

The toolbar at the bottom has a help button that allows the user to switch on the help feature. The help feature essentially provides the user with the function of the control over which the mouse moves. Once comfortable with the GUI features, the user can conveniently switch it off. The animations can be viewed stepwise or in a continuous manner. Moreover, the scroll bar allows the user to specify the speed of animation. The 'Pause' and 'Resume' buttons provide the users with added control such that they can view the visualization to match their pace. The user can abort a particular animation at any given point in time, which is advantageous if an algorithm is executing for a long time while the user has already understood all concepts. It goes a long way in keeping the user interested, one of "the ten commandments of algorithm animation". The above features account for a lot of interactivity built into the software. The animation specific controls (in the above figure – make array, delete node, insert node) are always included on the left side of the window.

Moving onto the menu options, the 'File' option includes the usual functions of opening, closing, saving files and exiting the software. The 'Data Structures' menu enables the user to work with the built-in data structure visualizations. The 'Build' option provides the compile and run tools to be used in conjunction with user-defined algorithms. Finally, the 'About' menu includes information about the current version of JavaMy software.

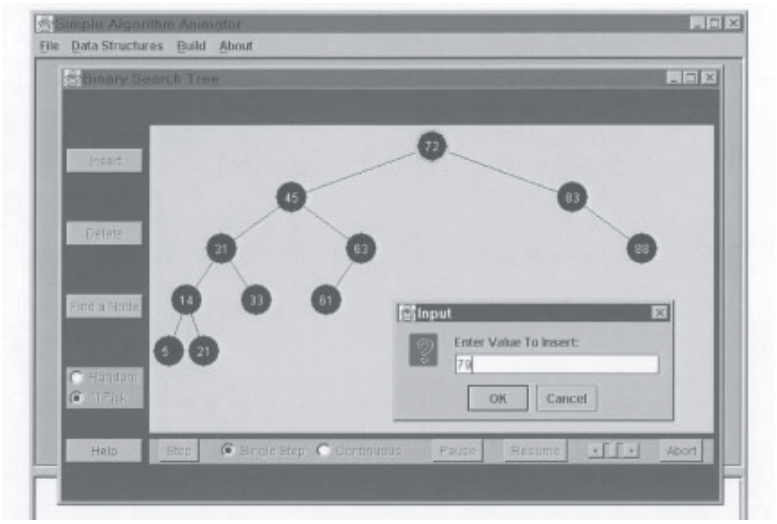


*Figure 2: Insert an element into the tree*

To relate the above features to the functionality of the software, an instance of working with built-in data structure visualization would entail choosing the same from the Data Structures menu. Animating a user-defined algorithm, on the other hand, shall sequentially step through opening the file or alternatively typing the code in a new file and saving it; compiling; and running the algorithm. In this case, the GUI meets the goal of incorporating symbolic and iconic representation in that while the left side shows the iconic animation of the algorithm, the right side displays the algorithm code for easy reference.

### 3.3 JavaMy as a Data Structure Visualization tool

An animation of a data structure is helpful to students as an alternative view in understanding a newly presented data structure. An animation can be easier to understand and remember than a textual representation, especially when one can interact with the animation [15].

The observable data structures currently available in JavaMy software package include array, stack, queue, binary search tree, heap and graph. Following is the listing of the specific operations on corresponding data structures, visualizations for which are currently available:

- Array – Make Array, Insert Node, Delete Node
- Stack – Make Stack, Push Node, Pop Node
- Queue – Make Queue, Enqueue, Dequeue
- Binary Search Tree – Insert, Delete, Find Node
- Heap – Make Heap, Delete Max Node
- Graph – Make Graph

The figure above shows a screen for inserting a node into binary search tree.

Noteworthy, is the fact that the user is allowed to choose the value to be inserted. This makes it really effective whereby the instructor can use this as a tool to visually describe the navigation of the tree to correctly insert a node as dictated by the invariant. Alternatively, the students can be assigned an exercise to use the 'Random' or 'I'll pick' feature to discover the invariant of a binary search tree by multiple runs of the software in a lab setting.

Another appropriate area could be to use the delete node from binary search tree visualization. The instructor can actively engage students while explaining the three possible scenarios in the deletion process where the node to be deleted could be a leaf, a node with one child or a node with two children. The students could then be assigned the task of coding the program for the same that could then

be tested using the algorithm animator.

Whether it is the above described scenarios or insertion/deletion into arrays, stacks, queues or heaps, JavaMy provides students with an alternative and visual perspective, which may help increase student understanding.

### 3.4 JavaMy as an Algorithm Animator

Students in computer science are constantly asked to understand dynamic processes in the form of computer algorithms. Aside from a pseudocode on computer program implementation, a higher order description for the algorithm is usually conveyed in words, perhaps with a well chosen picture or two. Unfortunately, computer code, words and individual pictures present only static descriptions, specific views or instances of a dynamic process. Perhaps, the reason why some otherwise good students have trouble understanding code examples is that they are unable to translate such static description to dynamic process in their imagination. [17]

While the previous section discussed visualization of common operations in form of the series of changes that take place to the corresponding data structures, this section elaborates on the proposition of animating user-defined algorithms involving the specified data structures. This feature of the software has a twofold usage: it can be used by instructors to aid in teaching algorithms and it can be used by students to understand how their programs work.

Two areas where this feature can be gainfully employed are common application of the data structures and algorithm comparisons. Some examples of data structure applications are balanced symbol checking and conversions of infix expressions to postfix expressions and viceversa that are essential parts of operator precedence parsing algorithm. Sorting is one of the most commonly studied concepts and there are many sorting algorithms to be studied. The algorithm animator can be effectively used to animate the different algorithms to provide students with a dynamic perspective of the various algorithms such that the students can have a more gainful insight as to how one algorithm compares to another. The animations can also be used to teach the concepts of time complexity of various algorithms and analysis how one algorithm compares to others in the same genre. Some areas where the instructor can effectively use this facet of the software are in comparing sorting algorithms, comparing the various tree traversals, depth first search vs. breadth first search in relation to graphs.

We include two detailed examples to illustrate both the areas described above. Balanced symbol checking algorithm illustrates a stack data structure application and the depth first search and breadth first search in a graph provide a comparison study of two related algorithms.

## Balanced Symbol Checking

A balanced symbol checker is a tool to help debug compiler errors, which checks whether symbols are balanced. In other words, whether every "{" corresponds to a "}", every "[" to a "]", every "(" to a ")", and so on. The basic algorithm is stated as follows:

Make an empty stack. Read tokens until the end of the input file. If the token is an opening symbol, push it onto the stack. If it is a closing symbol and if the stack is empty, report an error. Otherwise, pop the stack. If the symbol popped is not the corresponding opening symbol, then report an error. At the end of the file, if the stack is not empty, then report an error.

The above algorithm coded in JavaMy is shown in the following program:

```
/* Balanced Symbol checker is used to check whether every { corresponds to a }, every [ to a ], every ( to a ). And the sequence [()] is legal, but [(]) is wrong.
 */

public static void main(String arg[])
{
  String input = "{[([([()]}";
  char c, match;
  String errmsg;

  MyArray in = new MyArray(AnimatorFrame.ARRAY_POSITION,
        input.length());
  MyStack pendingTokens = new MyStack(AnimatorFrame.STACK_POSITION,
  0);

  for (int i=0; i<input.length(); i++)
  {
    in.setValue(input.charAt(i),i);
  }

  for (int i=0; i<input.length(); i++)
  {
    c = in.getChar(i);
    switch(c)
    {
      case '(':
      case '{':
      case '[':
          pendingTokens.push(c);
          break;

      case ')':
      case '}':
      case ']':
          if (pendingTokens.isEmpty())
          {
 System.out.println("Extraneous "
 + c +" found");
          }
          else
          {
                    match =
pendingTokens.topChar();
          pendingTokens.pop();
          if (match == '(' && c != ')' ||
             match == '{' && c != '}' ||
             match == '[' && c != ']' )
          {
           errmsg = "Found \"" + c + "\"
      does not match \""+match+"\"";
  JOptionPane.showMessageDialog(
  new JPanel() , errmsg,"Error",
   JOptionPane.ERROR_MESSAGE);
      }
          }
          break;
vdefault:
       break;
      }
    }

    while (!pendingTokens.isEmpty())
    {
     match = pendingTokens.topChar();
     pendingTokens.pop();
     errmsg = "Unmatched \"" +
match +"\"";
      JOptionPane.showMessageDialog(
new
JPanel(),      errmsg,      "Error",
JOptionPane.ERROR_MESSAGE);
    }
  }
```

The program starts with multiple-line comments, which document programs and improve program readability. The comment notation in JavaMy is the same as Java. Multiple-line comments are delimited with /* and */, and single-line comments are delimited with //. Following comments is simply a blank line. Blank lines, space characters and tab characters are known as white-space. Such characters are used to make the program easier to read. They are ignored by the parser. The bold line indicates the beginning of the real code of the algorithm. The three lines following the opening parentheses declare normal variables as in Java, using the data type provided by the Java programming language. The next six bold lines instantiated two observable data structures that will show on the animation frame. The first one is an array, which is used to hold the input string, that is, the string to be checked. The second one is a stack, which is used to hold the opening symbols. Here, MyArray and MyStack are used. Both of the constructors of MyArray and MyStack take two parameters. One is the Position parameter, which is used to decide the location of the data structure on the animation frame. Another parameter is the size of the array or stack. The rest of the code is the same as Java. Class MyArray and MyStack provides most of the commonly used methods, for example, setters and getters for setting and getting the values of the elements in the array, respectively, push(), pop() and methods for peeking the top element on the stack, etc. Details of those methods are described in the documentation generated by Javadoc.

After parsing and compiling the algorithm successfully, we can run the animation as described in subsection 3.2.1. The resulting animation frame is shown in Figure 3.

As can be seen from the resulting animation frame, there is a specific error message, on the right are the contents of the stack and the row at the top are the popped out elements. Just above the error message is the input string where the current input under processing is highlighted. On the right side of the screen is the algorithm whose animation occurs on the left.

This example could be used by instructors to supplement their lecture. The algorithm could be executed in the stepwise mode where the instructor could actively engage students in asking them to come up with the next step orally and then hitting the step button on the GUI explaining why the students answer was right or wrong as the case may be. At the same time, students could animate the algorithm to further their understanding of the algorithm at a pace suitable to their needs.

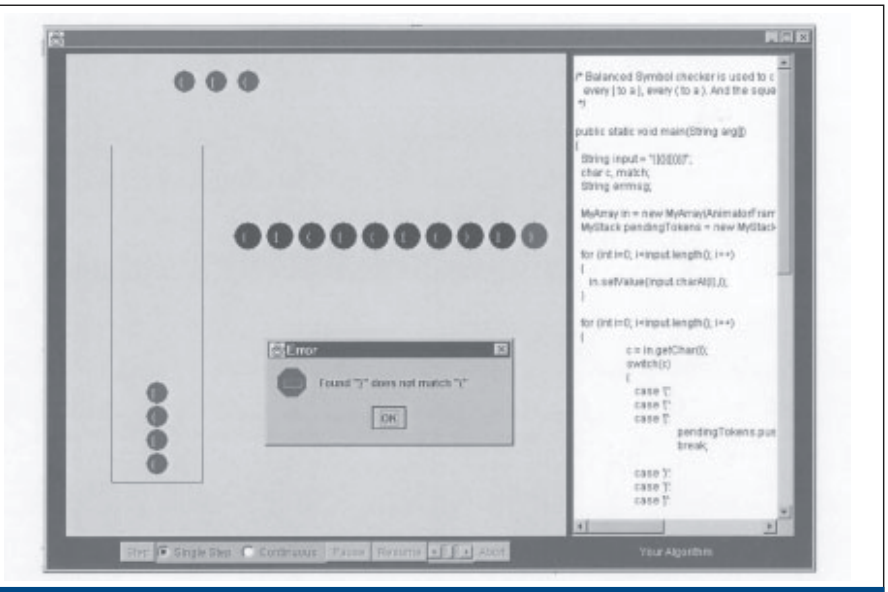


*Figure 3: **Animation Frame of Balanced** Symbol Checking*

## Breadth First Search and Depth First Search Algorithms

Breadth-first Search Algorithm

The breadth-first search algorithm takes a graph and a vertex in the graph known as the source, and visits (performs functions on) each node that can be reached from the source by traversing the edges. In doing so, it is easy to determine which vertices can be reached from the source. The algorithm for breadth-first search from a source vertex s in a graph g is as follows:

```
enqueue the source vertex;
repeat
  dequeue u;
  perform any relevant operations on u;
  enqueue all the neighbors of u;
until the queue is empty
```

The algorithm coded in JavaMy is:

```
// Breadth First Search
public static void main(String args[])
{
  final Position GRAPH_POSITION =
new
        Position(80, 80);
  MyGraph myGraph = new MyGraph(
GRAPH_POSITION, 4, 8, true);
    DrawableString label = new
DrawableString(
new Position(20, 390),
"Visited Nodes(Breadth First):");
  DrawableString traversalList = new
DrawableString(new Position(20, 420));
  label.setColor(Color.blue);
  traversalList.setColor(Color.red);
  myGraph.makeGraph(2,2);
  myGraph.init();
  // search the graph
  int depth = 0;
  int current = myGraph.initSearch(false);
  Vector nextQueues[] = new
Vector[myGraph.getNumOfNodes()];
  nextQueues[depth] = new Vector();
  Position positions[] = new Position[1];
  positions[0] =
myGraph.nodePosition(current);
  myGraph.circle.moveTo(positions);
  myGraph.traceAndMark(current,
traversalList);
  myGraph.setNexts(current,
nextQueues[depth]);
  nextQueues[++depth] = new Vector();
  while (!myGraph.empty(
nextQueues[depth - 1]))
  {
    current = myGraph.getNext(
nextQueues[depth - 1]);
    positions[0] =
```

```
myGraph.nodePosition(current);
   myGraph.circle.moveTo(positions);
   myGraph.traceAndMark(current,
traversalList);
   myGraph.setNexts(current,
nextQueues[depth]);
   if (nextQueues[depth - 1].size() == 0)
    nextQueues[++depth] = new Vector();
  }
  myGraph.circle.hide();
}
```

This example demonstrates usage of the observable data structure MyGraph. Some snapshots of the animation are shown in Figures 4-6.

In the snapshots, the algorithm code occurs on the right hand side of the screen for easy reference. The top part on the left side shows the graph configuration at a given point in time. The nodes that have been visited are marked by a cross and at the bottom the visited nodes are added to the list in order of visitation (in this case breadth first).

Depth-first Search Algorithm

Depth first search is another way of traversing graphs, which is closely related to a preorder traversal of a tree.

The algorithm coded in JavaMy:

```
// Depth First Search
public static void main(String args[])
{
  final Position GRAPH_POSITION =
new
Position(80, 80);
  MyGraph myGraph = new MyGraph(
GRAPH_POSITION, 4, 8, true);
  DrawableString label = new
DrawableString(new Position(20, 390),
"Visited Nodes(Depth First):");
  DrawableString traversalList = new
        DrawableString(new Position(20,
420));
  label.setColor(Color.blue);
  traversalList.setColor(Color.red);
  myGraph.makeGraph(2,2);
  myGraph.init();
  int current = myGraph.initSearch(true);
  // search the graph
  depthFirstSearch(myGraph,
 current,traversalList);
  myGraph.arrow.hide();
}
private static void
depthFirstSearch(MyGraph myGraph, int
current,DrawableString traversalList)
{
```

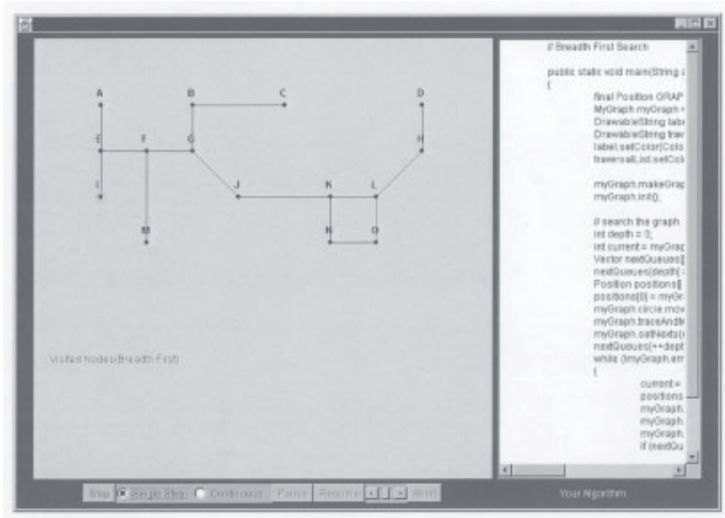

*Figure 4: **Graph to be searched***

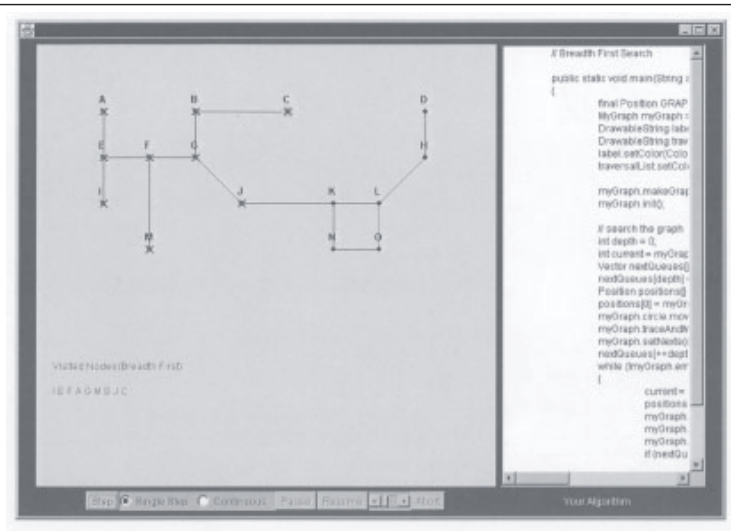

*Figure 5: **Breadth-first Search in progress***

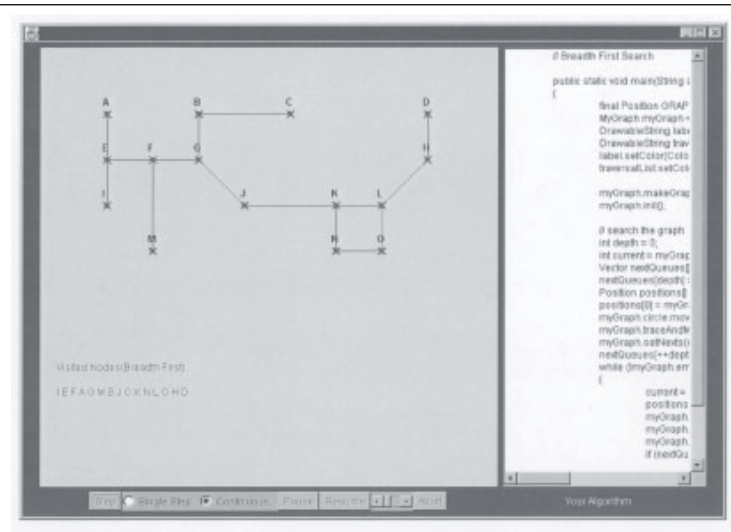

*Figure 6: **Breadth-first Search done***

```
  if (!myGraph.marked[current])
  {
    myGraph.arrow.setDirection(
myGraph.nodePosition(current), true);
   Position positions[] = new Position[1];
    positions[0]=
myGraph.nodePosition(current);
    myGraph.arrow.moveTo(positions);
    myGraph.traceAndMark(current,
traversalList);
    Vector nextQueue = new Vector();
            myGraph.setNexts(current,
nextQueue);
    while (!myGraph.empty(nextQueue))
    {
     int next =
myGraph.getNext(nextQueue);
     depthFirstSearch(myGraph, next,
traversalList);

myGraph.arrow.setDirection(
 myGraph.nodePosition(current), false);
     positions[0] =
myGraph.nodePosition(current);

myGraph.arrow.moveTo(positions);
     myGraph.traceAndMark(current,
traversalList);
    }
  }//end if
}
```

This example uses the observable data structure MyGraph and the helper class DrawableString. The resulting animation is shown in Figures 7-9.

In consistency with the breadth first search, the algorithm code occurs on the right hand side of the screen for easy reference. The top part on the left side shows the graph configuration at a given point in time. The nodes that have been visited are marked by a cross and at the bottom the visited nodes are added to the list in order of visitation (in this case depth first).

The above two algorithm animations can be used in a pedagogically effective way. They can be used to provide a dynamic view of the two algorithms enabling the students to get a clear understanding of the underlying differences of the two methods of graph traversals aimed at achieving the same end result. The instructor could use the stepwise feature to reinforce the taught concepts by asking the students the sequence of visitation at each step. More thought provoking questions such as which traversal method would be more suitable under what con-

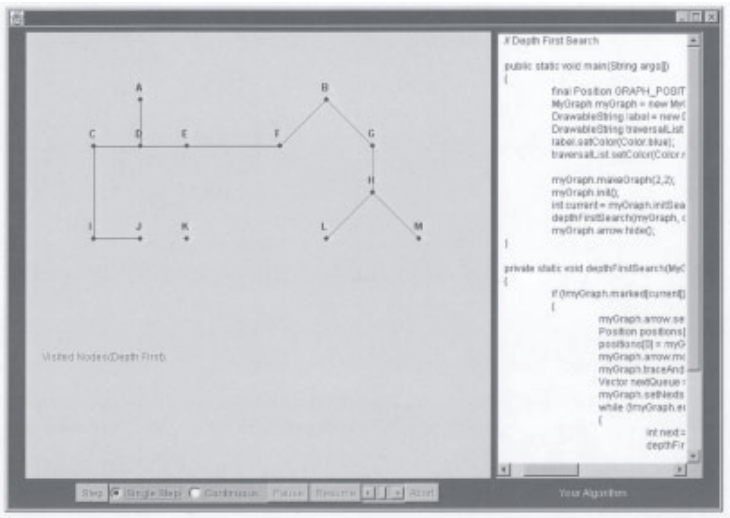

*Figure 7*: **Graph to be Depth-first searched**

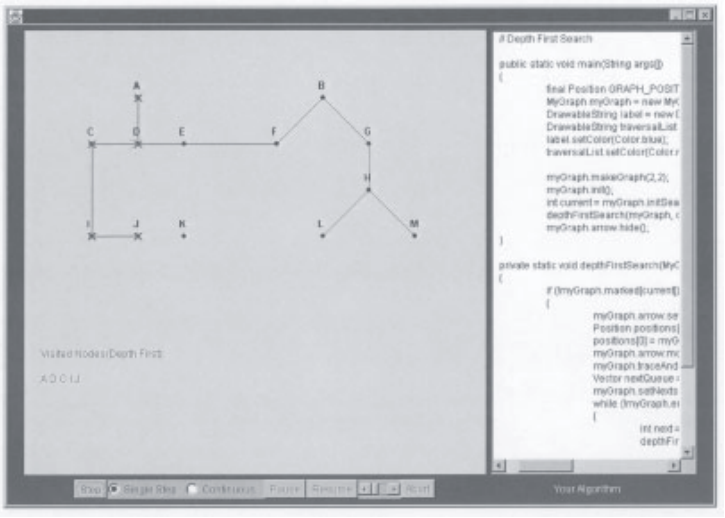

*Figure 8*: **Depth-first Search in progress**

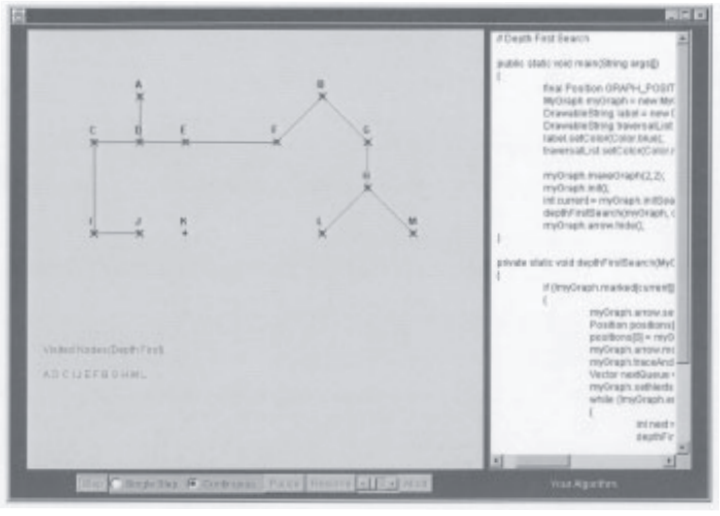

*Figure 9*: **Depth-first Search Done**

ditions could follow.

## 4. Conclusions and future works

In this paper, we present a visualization tool designed to aid first-year computer science students learn Data Structures and Algorithms. This tool not only lets students visualize the commonly used data structures, but also allows students to write their own algorithms in a Java similar language - JavaMy, and observe the execution of the algorithms.

The effectiveness of the tool is largely dependent on its usage. It is designed to be used to supplement traditional instruction not supplant it. The pedagogical effectiveness of the tool can be brought out by the proper use of the software by the instructors in their lectures as well as encouraging students to use it as an aid to further their understanding of the related concepts. Creative assignments could be designed around the software to enhance its effectiveness.

Because of the time limitation, only the most commonly used data structures are implemented in this version of the software package, which include arrays, stacks, queues, binary search tree, binary heap, priority queue and undirected graph. There are two ways to add more observable data structures to this software such as directed graph, weighted graph, AVL tree, Red Black Tree, AA- tree, splay tree, hash table, etc. One way is to implement these data structures in the software. Another approach would be to develop and implement a mechanism for the software package to recognize the user-defined observable data structures, and leave the implementation to the user. This approach will allow users to use their own observable data structures, hence add more flexibility to the software.

Another possible future enhancement for the software is to highlight the executing command line of the user-defined algorithm file. This would help the user to better follow the execution of the algorithm.

## 5. References


1. Morris, John, "Programming Languages and Data Structures", http://swww.ee.uwa.edu.au/~plsd210/ds/ds_ToC.html

2. Cawsey, Alison, "Data Structures and Algorithms",http://www.cee.hw.ac.uk/~alison/ds98/ds98.html

3. Owens, Brad "CS300 Data Structures and Algorithms I", http:/www.cs.twsu.edu/~bjowens/cs300/

4. Cohen, Edith "CS270: Combinatorial Algorithms and Data Structures", http://www.cs.berkeley.edu/~edith/cs270/

5. Goodrich, Michael T. and Tamassia, Roberto, "Data Structures and Algorithms in Java", http://www.cs.brown.edu/courses/cs016/book/

6. Jarc, Duane J., "Interactive Data Structure Visualizations", http://www.seas.gwu.edu/~idsv/idsv.html

7. The Graphics, Visualization & Usability (GVU) Center at Georgia Tech, "XTango", http://www.cc.gatech.edu/gvu/softviz/algoanim/xtango.html

8. The Graphics, Visualization & Usability (GVU) Center at Georgia Tech, "Polka", http://www.cc.gatech.edu/gvu/softviz/algoanim/xtango.html

9. System Research Centers (SRC) at Compaq Computer Corporation, "Algorithm Animation at SRC", http://www.research.compaq.com/SRC/zeus/home.html

10. Sun Microsystems, "JavaCC – The Java Parser Generator", http:/www.metamata.com/javacc/

11. Dann, W., Cooper, S. and Pausch, R.(2001). Using Visualization to Teach Novices Recursion. Proceeding on Integrating Technology into Computer Science Education, 109-112.

12. Gloor, P.A. (1998). User Interface Issues for Algorithm Animation. In Stasko, J., Dominigue, J., Brown, M.H. and Price, B. A.(Eds.), Software Visualization(pp.145-152). Cambridge, Massachusetts: The MIT Press.

13. Jarc, D.J., Feldman, M.B. and Heller, R.S. (2000). Assessing the Benefits of Interactive Prediction using Web-based Algorithm Animation Courseware. Proceeding of 31st SIGCSE Technical Symposium on Computer Science Education, 377-381.

14. Korhonen, A. and Malmi, L. (2000). Algorithm Simulation with Automatic Assessment. Proceedings on Integrating Technology into Computer Science Education, 160-163.

15. Pierson, W.C. and Rodger, S.H. (1998). Web-based Animation of Data Structures Using JAWAA. Proceeding of 29th SIGCSE Technical Symposium on Computer Science Education, 267-270.

16. Rößling, G., Schüler, M. and Freisleben, B. (2000). The ANIMAL Algorithm Animation Tool. Proceeding on Integrating Technology into Computer Science Education, 37-40.

17. Shaffer, C.A., Heath, L.S., Yang, J. (1996). Using the Swan Data Structure Visualization System for Computer Science Education. Proceeding of 27th SIGCSE Technical Symposium on Computer Science Education, 140-144.

**Tarek M. Sobh** received the B.Sc. in Engineering degree with honors in Computer Science and Automatic Control from the Faculty of Engineering, Alexandria University, Egypt in 1988, and M.S. and Ph.D. degrees in Computer and Information Science from the School of Engineering, University of Pennsylvania in 1989 and 1991, respectively. He is currently the Dean of the School of Engineering at the University of Bridgeport, Connecticut; the Founding Director of the Interdisciplinary Robotics, Intelligent Sensing, and Control (RISC) laboratory; a Professor of Computer Science and Computer Engineering; and the Chairman of the Prototyping Technical Committee of the IEEE Robotics and Automation Society.

**Tao Chen** was a graduate student in University of Bridgeport's Computer Science and Engineering Department pursuing an M.S. degree in computer science from Fall 1999 – Fall 2000. She obtained her M.S. degree in December 2000. Picture not available.

**Abhilasha Tibrewal** received her B.Sc. and M.Sc. in Home Science with honors in Textile and Clothing from Lady Irwin College, Delhi University, India in 1993 and 1995, respectively, and M.S. in Education and M.S. in Computer Science from University of Bridgeport, CT, USA, in 2000 and 2001 respectively. She is currently employed as Visiting Assistant Professor of Computer Science and Engineering at University of Bridgeport. She is member of ACM, ASEE and the honor societies of Phi Kappa Phi and Upsilon Pi Epsilon.