Web-based Animation of Data Structures Using JAWAA

Willard C. Pierson and Susan H. Rodger*
Duke University, Durham, NC
rodger@cs.duke.edu

Abstract

JAWAA is a simple command language for creating animations of data structures and displaying them with a Web browser. Commands are stored in a script file that is retrieved and run by the JAWAA applet when the applet's Web page is accessed through the Web. JAWAA commands allow for creation and movement of primitive objects (circles, lines, text, rectangles) and data structure objects (arrays, stacks, queues, lists, trees and graphs). A JAWAA script can be generated as the output of a program written in any language.

1 Introduction

An animation of a data structure is helpful to students as an educational aid in two ways, first as an alternative view in understanding a newly presented data structure or algorithm and second as an aid in debugging a program that uses the data structure. An animation can be easier to understand and remember than a textual representation, especially when one can interact with the animation by trying different input. Furthermore, using animations to debug programs can aid in finding errors faster by seeing incorrect movement or pieces that are not connected and should be.

We have developed JAWAA (Java And Web-based Algorithm Animation), an easy-to-use, architecture independent method for creating animations of data structures run over the Web. Written in Java, the program provides an interface through which users can write animations and then display them with a Web browser that supports Java. The animations are written in a simple script language and can easily be generated as output from a program written in any programming language. One does not need to know Java in order to use JAWAA.

Using JAWAA, one can quickly animate data structures in their program. One-line commands in JAWAA can be used to create and display a data structure, followed by one-line commands to perform operations on the data structure.

By writing JAWAA in Java, users can run the animations on almost any machine and do not have to install the software locally. By leveraging the capabilities of Java and the Web, we have built a system that will provide educators the realistic option of quickly generating algorithm animations for demonstration use in the classroom or by students generating quick animations of their programs.

In Section 2 we discuss related work, and in Section 3 we give an overview of JAWAA. In Section 4 we describe the animation language, and in Section 5 we give examples of animating programming assignments. Section 6 describes the program design of JAWAA, Section 7 the user response, and Section 8 gives concluding remarks.

2 Related Work

Over the last decade a diversity of packages for algorithm animation have been developed [12]. The package Xtango [8] and its successor Samba [9] have been used by many computer science educators for a variety of topics, including operating systems[4], algorithms and data structures [7] and genetic algorithms [5]. In addition, Samba has been used by students to generate animations [9]. 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. In addition, there is no software to install to use JAWAA as animations are run over the Web.

Although the algorithm packages to date have succeeded in providing basic functionality, few have leveraged the full capabilities of the Web and Java. There is, however, a great deal of thought about the theoretical possibilities. In [6], Naps discusses how algorithm visualization could be integrated into the Web. In [3], the authors discuss ways in which the Web could be used to build courseware modules in order to aid student understanding. Among the conclusions of both papers is the assertion that Java offers many attractive features with which an algorithm animation system could be

*The work of this author is supported by the National Science Foundation's Division of Undergraduate Education through grant DUE-9555084.

built. There is currently much work being done in this area as summarized in [2] with most focusing on porting current applications to Java. There are also several Java applets currently on the Web that animate a single algorithm.

3 JAWAA Overview

JAWAA makes it easy to create and run an animation, viewing the animation over the Web using Java. To create an animation, one creates a script file and a simple applet. The script file is retrieved by the JAWAA applet when the applet's web page is accessed. The program then begins to interpret the commands line by line, carrying out the graphic task of each instruction.

The user interface is composed of an animation canvas and a panel of user controls. With these controls the user can start, stop, pause, and step through the animation. A scrollbar allows one to control the speed of the animation.

4 Animation Language

The animation language has been designed to require little programming experience. Animation scripts have a simple format of one command per line. There are two types of commands, those that handle specific objects and those that specify an action on an object. Of those that deal with specific objects there are two types of objects that can be manipulated, primitive objects, such as a circle, and intelligent objects. Intelligent objects represent data structures and have specific commands to perform operations on them.

4.1 Primitive Objects

Primitive objects in JAWAA are lines, circles, text, rectangles, and polygons. Shown below is an example of a *circle* command to draw a red-filled circle with a blue outline at position (30,20) and with radius 60. The circle is given an identifier name, c1, so that it can be refered to later, for example to change its color or to move it to another position.

```
circle c1 30 20 60 blue red
```

4.2 Arrays

Arrays are created by using the *array* command and specifying the contents of the cells and the orientation of the array, either vertical or horizontal. Shown below is an example to create an array named *words* containing three words. The array is drawn horizontally at position (50,60), with black outline and red background.

```
array words 50 60 3 "hello" "my" "world"
            horz black red
```

Array cells are manipulated on an individual level as if they were rectangles. For instance if an array was named A, then A[2] would refer to the 3rd entry in A (array indexing begins at 0). This allows each cell to be accessed directly instead of creating commands specific to array cells.

4.3 Stacks and Queues

There are three commands specific to the use of stacks. The *stack* command declares a stack and must contain information about the name of the stack, the color, the position, the size, and the entries. The *push* command pushes a string on the stack and the *pop* command pops the top item off the stack. Queues have three commands, a *queue* declaration, *enqueue*, and *dequeue*.

4.4 Graphs

Graphs in JAWAA can be drawn with node placements specified by the user or by an automated layout chosen from two algorithms. To build a specified layout graph, the *node* and *connectNodes* commands are used. The *node* command specifies the location and size of a node. Once the nodes are placed, the *connectNodes* command can be used to create an edge between two nodes. The *connectNodes* command takes as parameters the names of the two nodes and a boolean which indicates whether to animate the connection or to immediately place the edge on the screen. For example, shown below is a *node* command to create a node named *n1* located at position (30,20) with diameter 10. The outline of the node is black with its interior light gray.

```
node n1 30 20 10 black lightGray
```

Node *n1* could be connected to another node *n3* with an edge labeled *e9* with the command shown below.

```
connectNodes e9 n1 n3 black false
```

One can also use JAWAA's automated graph layout capabilities to draw a graph. The *graph* command is used to create the graph and takes as parameters a list of the connections in the graph and the layout algorithm to be used. One algorithm draws the nodes of the graph in a circle.

A second heuristic algorithm uses a cost function to assess the quality of the graph[10]. The idea is to place nodes individually on the graph, to compute the cost of each placement and then place the node at the position with the least cost. Possible placements for nodes are found with a template that gives locations around the node's neighbors and the barycenter[1] of its neighbors [11]. The cost function rates the graph on three criteria: the distance between all nodes, the distance between the node's neighbors, and the number of edge crossings. The graph with the most distance between nodes, the least distance between neighbors, and the least number of crossings will receive the lowest (best) cost. This graphing style takes significantly longer to run than the circle style to draw a graph. Figure 1 shows a graph layout using this algorithm, and Figure 2 shows the change in layout after one edge (and one node) was added to the graph. Notice most of the nodes have changed their position.

[1] The barycenter is the "average" location of the nodes, or the average of each node's x and y coordinate

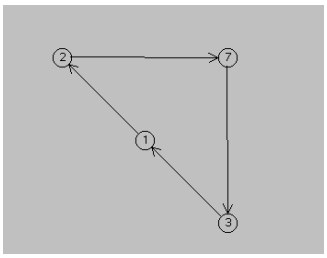

Figure 1: Drawing a Graph

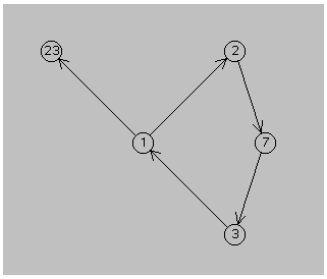

Figure 2: Adding an Edge to the Graph

There are two other commands that can be used with graphs once the graph has been created. The *addNode* command places an unconnected node on the graph. The *addEdge* command adds the connection to the graph along with either node if either have not been previously placed.

4.5 Trees

The creation of trees is similar to the creation of graphs. Trees can be created by specifying the position of each node or by an automated process. The *tree* command will create a binary tree based on a list of connections. The tree is drawn with a recursive algorithm developed to apportion space according to the width of a subtree. This results in a tree that makes the most use out of the drawing area. The *addEdge* command can also be used with trees to add new nodes to the tree. Figure 3 shows the automatic drawing of a tree and Figure 4 shows the same tree after several more elements have been added to the tree. Note the change in position of nodes 5 and 9 to make room for the additional nodes.

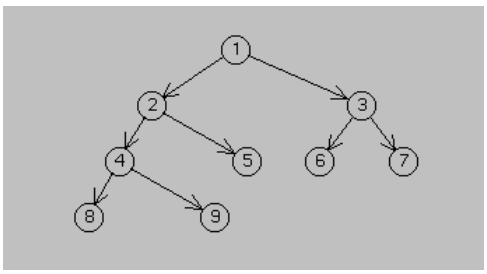

Figure 3: Drawing a Tree

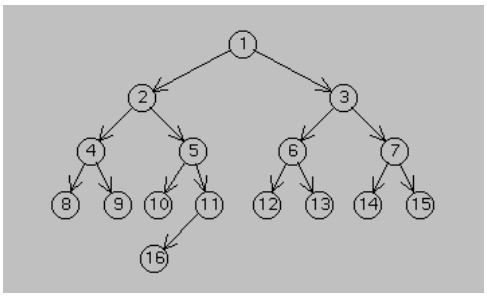

Figure 4: The Tree Several Steps Later

4.6 Action Commands

Action commands are different than the previous commands in that they act on a variety of different object types. For example, the command *moveRelative* will move an object a specified distance from its current location. The command *changeParam* will change a given parameter to a given value. For example, the command shown below changes the background color of the second entry of array A to green.

```
changeParam A[1] bkgrd green
```

There are other commands to delete objects, to pause the animation for a specified amount of time, and to group a list of objects together so that they can be treated as one object.

5 Examples using JAWAA

We describe the setup for running JAWAA and give three examples to show how JAWAA can be used to animate a programming assignment. Any programming assignment that uses a data structure that JAWAA supports can be easily modified to produce JAWAA commands for animating the data structure. Small amounts of data that can fit reasonably in a window work best and can aid in understanding an algorithm.

The setup for running JAWAA is easy. In each assignment, students copy a templated web page and change one line to the path of their public web directory. JAWAA output is written to a file in this directory. To see the animation, students just run their program to generate new output and then reload the web page.

We describe two assignments in the fall of 1997 in the courses CPS 100 and CPS 100E, data structures courses at Duke. In the first assignment, students write a program to find word ladders. (A word ladder is a list of words in which consecutive words differ by one letter.) Figure 5 shows the final view of an animation of a word ladder program, animating an array and a queue. The animation begins by storing the words to consider for ladders (the dictionary) in an array with the pointer fields empty. Two words are given, start and goal, and the object is to change the start word to the goal word through a ladder. The ladder is found in the following manner. The start word is placed in the queue. Repeatedly a

word is dequeued and becomes the current word. Any word in the array that has not yet been placed in the queue and is one letter different than the current word is enqueued and its link is set to the current word's position in the array. If the goal word is found, the ladder can be produced by following the links in the array, from the goal word to the start word.

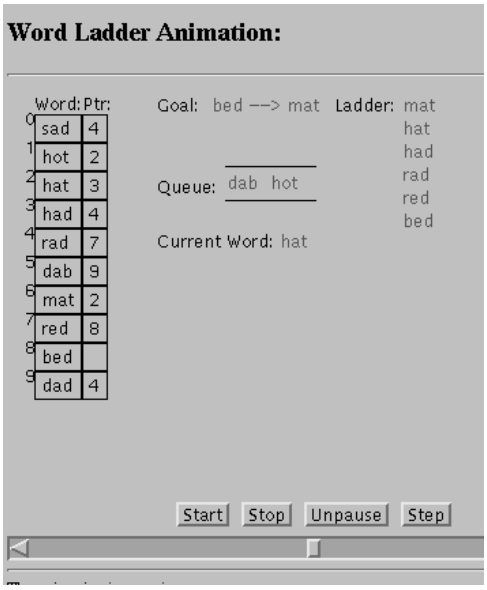

Figure 5: Finding WordLadders

In a second assignment, students animate the Josephus problem. *N* people sit around a circle and a hot potato is passed around. Every *M*th person the potato reaches is eliminated. This repeats until one person is left, the winner. In this animation, the people are represented in an array, the passing of the potato by blinking a slot in the array, and the elimination of a person, by pulling the person out of the array.

In the course CPS 140, a formal languages course at Duke, students write a program for an LR(1) parser. The students are given a parse table and write the code for parsing a string, which includes the parsing stack and accessing the parse table. Figure 6 shows how a student created an animation of parsing a string by adding JAWAA output to his program. He animated the parsing stack, the current position in the string and the movement through the parse table (by highlighting the current entry). Although there is not a JAWAA command for a 2-dimensional array, one can easily be animated by generating an array for each row (or for each column). The parse table in Figure 6 was generated by creating an array for each row, named a1, a2, etc. The highlighted position (0,a) in the table is refered to as a2[1].

6 Program Design

There are four modules in the JAWAA program design. When the JAWAA applet is loaded, execution starts in the AnimClass module. This module controls the execution of

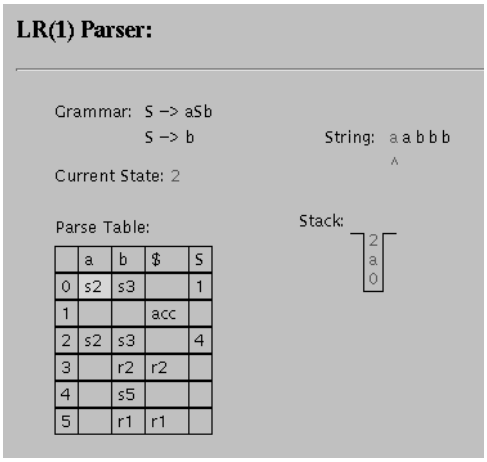

Figure 6: LR parsing

the animation and receives user commands through the user interface. The Animation class contains the routines that draw and update the screen. The Interpreter class reads the commands from the user specified animation script and calls the appropriate methods in the Animation class to carry the commands out. Finally, the GraphicStore class stores the graphic objects in memory.

6.1 AnimClass

The AnimClass module drives all execution that takes place in the program. When the program is run, the AnimClass constructor method is called to declare instances of the other classes and setup the layout of the user interface. When the user clicks the start button, AnimClass initiates the drawing of the animation. The program first retrieves the animation from the specified URL. The program then uses Java's multithread capability to spawn a new thread, which is responsible for running the animation. This is to prevent the drawing of the animation to affect the program's interaction with the user. The two threads communicate with each other through shared variables.

The animation itself is run by a method in the new thread. In this method, a loop repeats until the animation is finished. Each iteration of the loop calls a method of the Interpreter module, which reads and executes the next line of the animation script. Next the screen is redrawn so that the actions of the command are displayed. The method then goes to sleep for a period of time based on the user's speed selection.

6.2 Animation

The Animation module has two parts: methods that deal with the drawing of objects on the screen and methods that create and manipulate the objects behind the scene. The first part is used when the screen is redrawn. Screen redrawing is accomplished by double-buffering, which involves using two images, one held in memory and one that is displayed on the

screen. To redraw the screen, a new image containing each graphic object is created and then placed over the image currently on the screen, creating a flicker-free animation.

The second part of the Animation module handles the manipulation of the graphic objects. Every time a command is processed, the Interpreter calls methods from this class to carry out the desired action. For example, there are methods to create objects and there is a method to move an object. Once these methods have finished carrying out the command's instructions, the Interpreter will return to the main loop in AnimClass where the screen is redrawn.

6.3 Interpreter

The Interpreter class reads in each line of the animation script and calls the necessary methods in Animation to fulfill each command. To read in a line from the script the Interpreter uses a Java supplied class called StreamTokenizer, which takes an input stream and returns discrete words and numbers. The Interpreter then takes each number or word and stores them as strings in an array. Based on the first word of the line, which is always the command, the Interpreter calls a function to process the command line. This function then arranges the parameters and calls the appropriate method in Animation.

6.4 GraphicStore

Objects on the screen are stored in memory by storing information about each object in a separate instance of a class called GraphicObject. The GraphicStore module defines the GraphicObject class and handles the storage of objects using a hash table class provided with Java. However, the order of object creation is also needed and stored in an array, so the first objects created in the animation are visually below objects created later.

7 Availability

A Beta version of the JAWAA software package was released in late December 1996. In fall of 1997 we are currently using JAWAA in the Data Structures courses (CPS 100 and CPS 100E) and the Algorithms course (CPS 130) at Duke. One user in Australia has developed a suite of animations for use in an undergraduate Data Structures class and stated that he found "JAWAA to be an exceptional tool for animating algorithms." Through his feedback, we have refined the package. JAWAA is currently available on `http://www.cs.duke.edu/~rodger`

8 Conclusion

JAWAA is a command language for creating animations of data structures and algorithms. JAWAA commands can be added as output to any program to quickly generate an animation. When used in conjunction with traditional teaching methods, JAWAA can provide students with an alternative and visual perspective, which may help increase their understanding.

References

[1] A. Badre, C. Lewis, and J. Stasko, Empirically Evaluating the Use of Animations to Teach Algorithms, *Proceedings of the 1994 IEEE Symposium on Visual Languages*, p. 48-54, 1994.

[2] C. Boroni, F. Goosey, M. Grinder, R. Ross and P. Wissenbach, WebLab! A Universal and Interactive Teaching, Learning, and Laboratory Environment for the World Wide Web, *Twenty-eighth SIGCSE Technical Symposium on Computer Science Education*, p. 199-203, 1997.

[3] D. Carlson, M. Guzdial, C. Kehoe, V. Shah and J. Stasko, WWW Interactive Learning Environments for Computer Science Education, *Twenty-seventh SIGCSE Technical Symposium on Computer Science Education*, p. 290-294, 1996.

[4] S. Hartley, Animating Operating Systems Algorithms with Xtango, *Twenty-fifth SIGCSE Technical Symp. on Computer Science Education*, p.344-348, 1994.

[5] D. Jackson and A. Fovargue, The Use of Animation to Explain Genetic Algorithms, *Twenty-eighth SIGCSE Technical Symposium on Computer Science Education*, p. 243-247, 1997.

[6] T. Naps, Algorithm visualization served off the World Wide Web: why and how, *Proc. on Integrating Technology into Computer Science Education*, p.66-71, 1996.

[7] S. Rodger, An Interactive Lecture Approach to Teaching Computer Science, *Proceedings of the Twenty-sixth SIGCSE Technical Symposium on Computer Science Education*, p.278-282, 1995.

[8] J. Stasko, Tango: A Framework and System for Algorithm Animation, *IEEE Computer*, p.27-39, 1990.

[9] J. Stasko, Using Student-Built Algorithm Animations as Learning Aids, *Twenty-eighth SIGCSE Technical Symp. on Computer Science Education*, p. 25-29, 1997.

[10] D. Tunkelang. A practical approach to drawing undirected graphs. Technical Report, Carnegie Mellon University, June 1989.

[11] HiroYuki Watanbe, Heuristic graph displayer for g-base. *International Journal of Man-Machine Studies*, p. 287-302, 1989.

[12] J. Wilson and R. Aiken, Review of animation systems for algorithm understanding, *Proceedings on Integrating Technology into Computer Science Education*, p.75-77, 1996.