Using Hands-On Visualizations to Teach Computer Science from Beginning Courses to Advanced Courses

Susan H. Rodger[*]
Department of Computer Science
Duke University
Durham, NC 27705
rodger@cs.duke.edu

Abstract

We have developed visual and interactive tools to be used by instructors to teach computer science and by students to learn concepts in a visual and hands-on manner. We describe two such tools and their use in computer science courses, JAWAA for easy creation of animations and JFLAP for experimenting with different types of automata and grammars.

1 Introduction

Students learn best by seeing several representations of a concept, including textual, visual, and animated. Textbooks can provide the textual and some visual representations. Software tools provide a visual and animated view. However, observing is not enough. In order to check their understanding of a concept, students must be able to interact with the concept in some way, receiving feedback.

There have been many attempts at creating visualizations of computer science concepts in textbooks. These textbooks take care of the textual representation, and possibly some part of the visual representation. For example, algorithms textbooks illustrate via snapshots the execution of sorting algorithms. Sedgewick [11] represents numbers as columns of different heights, and shows snap shots of the movement of the columns starting in random order until the columns are sorted by heights. As another view, he shows a 2-d matrix where the ith column represents the number with value i which is shown as a black pixel somewhere in the column. All numbers are sorted when the i in column i appears in row i, thus forming a diagonal line on the main diagonal. These snapshots allow the student to study the animation of one example, but only that example.

As another example, automata theory textbooks visualize automata by showing transition diagrams. However it is tedious to trace an input string through an automaton, especially if the automaton is nondeterministic.

There are many animations of computer science concepts available on the web. For example, the sorting animations in Sedgewick's textbook described above appear as Java applets on several web pages, mostly the results of student projects. However, many such animations do not allow the user to interact with the animation. Only one data set is used, and the same animation plays over and over again.

Software tools are needed to view the animated representation, and to further experiment with the concepts. We have been experimenting with visual and interactive tools [7] in several levels of computer science courses from the non-major course, to the introductory programming courses (CS 1 and CS 2), to the upper-level undergraduate course in automata theory and formal languages. We describe two such tools and their use in computer science courses, JAWAA for easy creation of animations and JFLAP for experimenting with concepts in automata theory.

In Section 2 we describe our requirements in creating tools. We describe the tool JAWAA in Section 3 and its use in Section 4. We describe the tool JFLAP in Section 5 and its use in Section 6. In Section 7 we describe additional tools for animation and for use along with JFLAP. Finally we conclude with future work in Section 8.

2 Successful Interactive Visualization

For a successful interactive visualization, students should be in control of the visualization and should receive feedback immediately. Control includes changing the input, changing the speed, allowing movement forward and backward and the ability to reproduce the result. Students should be able to create their own input and receive feedback on their input immediately.

[*]The work of this author is supported by the National Science Foundation through grants NSF DUE-9752583.

3 JAWAA for Animation

JAWAA [6] is a scripting language that can be the output of a program. It is designed to easily run over the web with no installation. One can enter JAWAA commands in a file (either by hand or as output from a program), modify a templated HTML file, and easily generate an animation that runs on a web page.

With JAWAA one can create simple primitives such as circles, rectangles, polygons, lines and text, and move them around singly or in a group. With advanced features of JAWAA, one can create arrays, stacks, queues, trees and graphs easily. The animation has a speed control bar and the ability to pause or restart the animation.

4 Using JAWAA in Courses

We describe how we have used JAWAA in several computer science courses at Duke University.

4.1 JAWAA in Non-Majors Freshmen Seminar

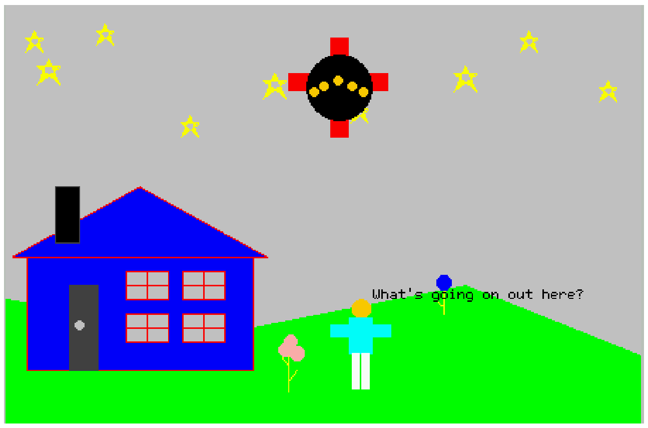


Figure 1: JAWAA Outer Space

The non-majors course CPS 49S [8], Animation and Virtual Worlds, is a seminar for freshmen to learn animation and computer science concepts. Several tools are used in this course to create 2-D and 3-D worlds, and such tools must be easy to use by nonprogrammers, as programming experience is not a prerequisite.

Creating animations with JAWAA is one of the units in this course, lasting one week and a half. Students create a JAWAA animation by typing a JAWAA script into a file. Creating an animation in this way is easy for them to do, but a bit tedious. Furthermore, students use only the simple primitives in JAWAA.

Students begin work together in pairs by following a tutorial that guides them in creating a simple animation. Next they work on specific projects in pairs. One project is a traffic intersection with vehicles and stoplights. Another project is a sorting algorithm animation. They are given columns of different heights in random order and must sort them using an algorithm they create. Finally they work individually on a JAWAA animation for homework. Figure 1 shows a student JAWAA project. In this animation the space ship flies in, pulls the bush up and takes it, and a person walks out of the house into the yard.

4.2 JAWAA in CS 1

Duke's CS 1 course, CPS 6, teaches programming using C++. In this course JAWAA is used by hiding the details of JAWAA in classes. Students use a C++ class that has already been written for them and JAWAA is automatically output into a file that students can load onto a web page.

For example, students learn about a hot air balloon class that they can control by making it ascend, cruise, and descend. In an early assignment when they are learning about classes, they can create the balloon and JAWAA output is automatically written to a file when they run their program. They can put this file on a web page and see the balloon move. Later when they learn about functions returning values, loops and randomness, they can create a hot air balloon race. The race has several balloons that randomly move one to four steps each. Students can continually check the distances of each balloon to decide which balloon wins the race.

4.3 JAWAA in CS 2

Duke's CS 2 course, CPS 100, teaches data structures using C++. In this course students write JAWAA commands as output in their programs. With JAWAA students can create both the simple primitives and data structures easily. All the projects in the course have data structures, so students can pick out a particular data structure to animate.

Figure 2 shows a CPS 100 project illustrating how arrays are shown in JAWAA. This figure shows the Josephus problem after three passes. In the Josephus problem, the magic number chosen is 5. Starting from 0, every fifth person is pulled out of the array. The first person pulled out is erich, then laura, then fritz.

To implement the Josephus problem, there are three arrays created, one for numbers labeled 0 to 7 to show the indices of the arrays, one array for players' names and one array for tracking the order the players were pulled out of the array. For example, laura is in slot 1 and was pulled out after erich who is in slot 4.

The declaration of the array for players' names is shown below. The array's leftmost top corner is at location

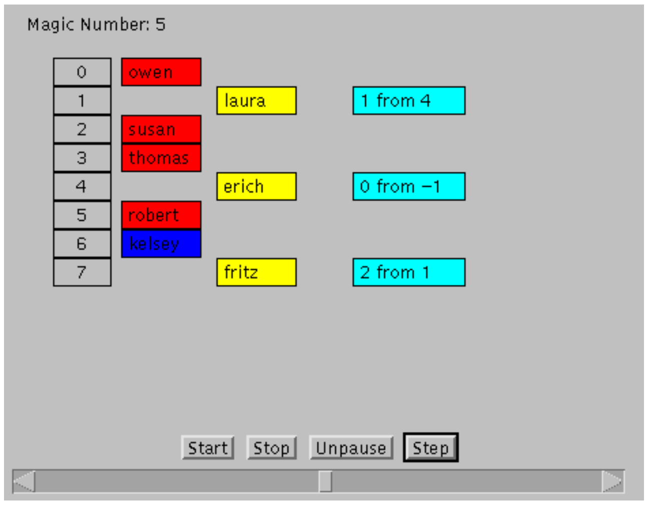


Figure 2: JAWAA Josephus Problem

(80,30), there are 8 positions in the array, the array is vertical, the inside color of the array is red, and the outline color of the array is black.

```
array Players 80 30 8 owen laura susan thomas
    erich robert kelsey fritz vert black red
```

After declaring an array, the individual positions can be referenced. Thus, stepping through the array can be animated easily by changing the inside color of a position to a different color when examining it, and back to its original color when finished. The first 10 lines of the code below illustrates stepping through 5 positions in the array, changing the color to blue when it examines a particular position in the array. Another feature of JAWAA is that array positions (rectangle) can be moved. In the code below the last two lines move the position with erich in it to the right and change the background color to yellow.

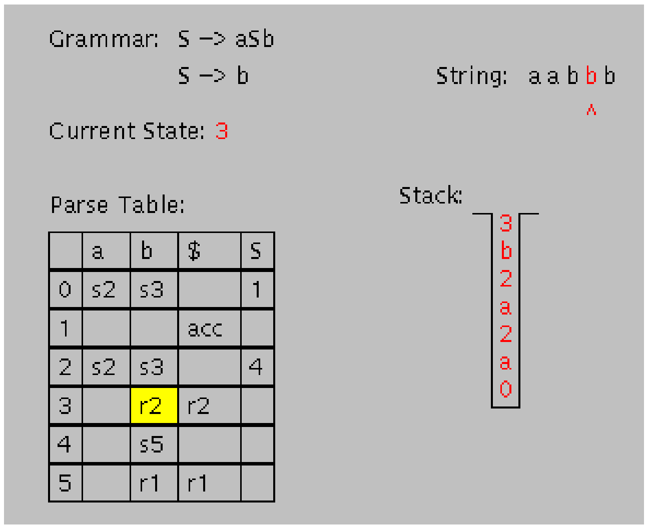
```
changeParam Players[0] bkgrd blue
changeParam Players[0] bkgrd red
changeParam Players[1] bkgrd blue
changeParam Players[1] bkgrd red
changeParam Players[2] bkgrd blue
changeParam Players[2] bkgrd red
changeParam Players[3] bkgrd blue
changeParam Players[3] bkgrd red
changeParam Players[4] bkgrd blue
changeParam Players[4] bkgrd red
moveRelative Players[4] 70 0 true
changeParam  Players[4] bkgrd yellow
```

With JAWAA's features of data structures, students can generate an animation involving data structures quickly. For example, in the Josephus problem above the student used one line to declare a filled array. If the animation was created using only simple primitives, over 15 lines of JAWAA code would have been needed. In Figure 3 a stack is shown on the right. One JAWAA command creates an empty stack, one JAWAA command animates the push of an item onto the stack and one JAWAA command animates the pop of an item from the stack. Alternatively, many lines of code would be needed to animate the stack from simple primitives.

4.4 JAWAA in Other CS courses

Other courses can use JAWAA in two ways. One to create animations of concepts to study, and two to animate a part of their program. We show examples of both ways below for CPS 140, the automata theory course at Duke.

As an example of an animation to study a concept, Figure 3 shows an animation of an LR parse table. The animation shows the parsing of the string aabbb. As the parsing proceeds, the current state is listed, the current position in the table is highlighted in yellow, and the stack contains the current symbols and state numbers. Currently the parsing is in state 3 and a reduce by rule 2 is about to happen, meaning items will be popped from the stack.



	a	b	$	S
0	s2	s3		1
1			acc	
2	s2	s3		4
3		r2	r2	
4		s5		
5		r1	r1	

Figure 3: JAWAA LR Parser

For an example of using JAWAA with a programming project, in CPS 140 students study a new simple language and write an interpretor for it, studying in detail the phases of the interpretor and how they relate to automata theory. Figure 4 shows a student's animation of a CAMO program. CAMO is a simple language in which you can declare cat, mice and mouse holes and move the cat and mice around. The figure shows one cat in yellow (the lightest color) being chased by a large number of mice that all came out of the mouse hole (the darkest color). With JAWAA students can see their program execute.

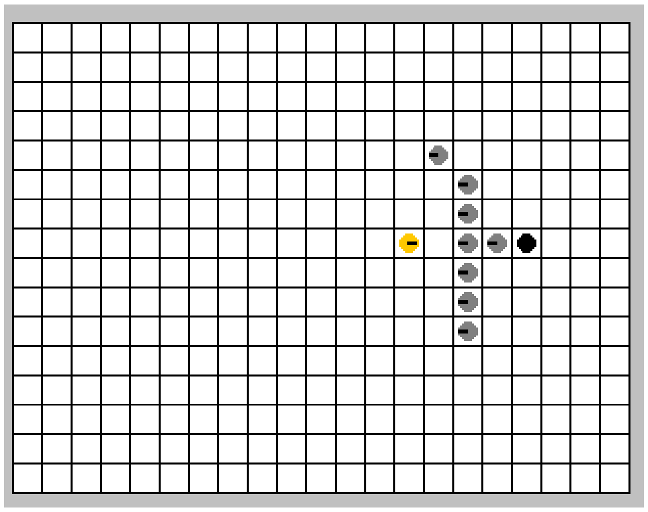


Figure 4: JAWAA Cat Vs Mouse

5 JFLAP for Automata Theory

JFLAP [4, 3] is a tool for interacting with automata theory concepts. With JFLAP one can create a finite automaton, pushdown automaton, and 1 or 2-tape Turing machine, drawn as a transition diagram. The user can enter input strings and watch the execution, even for nondeterministic machines. In addition, one can explore the conversion of several different forms including NFA to DFA, DFA to minimum state DFA, NFA to regular grammar, regular grammar to NFA, regular expression to NFA, NFA to regular expression, NPDA to CFG, and CFG to NPDA. JFLAP has been designed with general definitions and can be used with most automata theory textbooks such as [5].

Without JFLAP, students create their automata on paper and turn them in, usually with errors in them. It is too tedious to trace through to check for correctness. With JFLAP, their automata come to life.

6 Using JFLAP in Courses

JFLAP is used at Duke in CPS 140, the automata theory and formal languages course. It could also be used in the early phases of a compiler course. We describe how JFLAP can be used in several ways during lecture, and later by students outside of lecture.

6.1 Instructors Demo JFLAP

Instructors can demo JFLAP to students in lecture to show them how to use JFLAP. Demoing can show students the ease of use and its functionality, including moves they may not figure out on their own. As an example, in earlier years when JFLAP was not demoed in class, students learned it on their own for homework. Many students would turn in awful looking drawings, because they did not realize that states could be moved after they are created.

6.2 Instructors Solve Problems With the Class

Instructors can solve problems using JFLAP during lecture with input from the students. A typical example is giving the students an automaton and to ask them if it is correct or not, and if not to fix it. Figure 5 shows an incorrect Turing machine for recognizing the language $\{a^n b^n c^n \mid n > 0\}$. Students are asked during lecture to take a few minutes to trace the machine to see if it is correct. Tracing is easier and shorter than asking them to write the complete machine during class. Almost all students will say the machine is correct. The instructor can then run the machine on an input string suggested by a student such as **aaabbbccc**, which accepts. At this point the instructor can ask the class if there are strings that the Turing machine does not work correctly for, and then run the TM on the suggested strings. The instructor can also suggest the string *aaabbbbccc*, a string not in the language, run the TM and show the TM accepts it! Figure 6 shows a snapshot of this run. Here students can see that the TM is not checking to see if there are extra b's or c's. Students can be given a few minutes to determine how to fix the machine. A volunteer can suggest fixes which the instructor can implement and try on the previous input string.

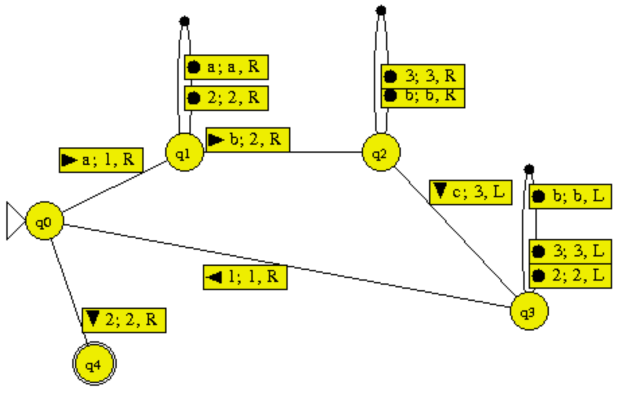


Figure 5: Incorrect JFLAP Turing machine for $a^n b^n c^n$

6.3 Instructors Experiment with Difficult Concepts

Instructors can use JFLAP to experiment with difficult concepts. Students have difficulty creating a nondeterministic pushdown automaton (NPDA) that is not deterministic. An example is to ask them to write an NPDA for $\{ww^R|w \in \Sigma^+\}, \Sigma = \{a, b\}$ during class. The majority of them will be stumped. They want to find the middle first, which is impossible since this language is not a deterministic context-free language. With some

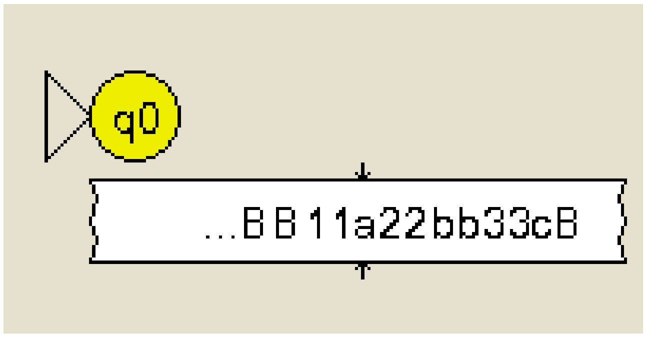


Figure 6: JFLAP Turing machine run on *aaabbbbccc*

help in guiding them, the class can create the NPDA solution, shown in Figure 7.

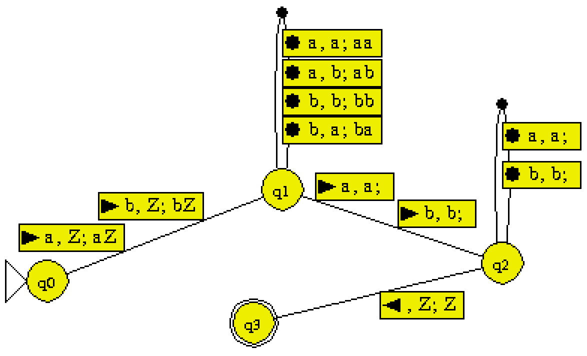


Figure 7: JFLAP NPDA for ww^R

Most students will not believe this NPDA is correct without some experimentation. There are three ways to run an input string. The first way to run it is to select the *Fast run*. JFLAP runs the NPDA on the input string and prints a message when it completes (or a message that it is taking a long time and prompts whether or not to continue). In running the above NPDA on the string a^{16} a message stated that the string was accepted, 89 configurations were generated, and the path of acceptance was of length 18. This gives the instructor a chance to explain how the NPDA runs and that there are several configurations being tried at the same time. The second way to run the NPDA is the *Step run* which shows all the current configurations at any time (up to a limit of 15). By stepping through and showing how to control the run (the user can freeze configurations for later expansion), students can experience the nondeterminism. Figure 8 shows a snapshot of the run on a^{16} in which 8 configurations are currently active and one configuration cannot proceed any further. Each configuration shows the current state, input symbols remaining, and current stack.

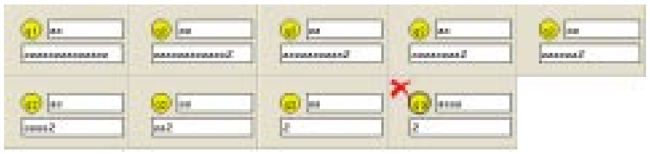


Figure 8: JFLAP run of NPDA for ww^R

6.4 Instructors Work Examples of Transformations

Instructors can give an example of a transformation either before or after studying a formal proof. For example, one proof students study is that every NPDA can be converted to a context-free grammar (CFG) that represents the same language. The instructor can start JFLAP and choose an NPDA, such as the one in Figure 7, and then step through the steps of converting that NPDA into the equivalent grammar.

6.5 Instructors Relate Concepts From Other Courses

Instructors can use JFLAP to show how automata relate to other computer science courses, by studying some of the same concepts. For example, with JFLAP one can revisit the problem of developing test data, in this case to test out an automaton. A third run feature is *Multiple run* mode, a window in which one can enter up to 10 input strings and JFLAP will run on all of them providing accepts or rejects. Another example is to study the running time of one and two-tape Turing machines. With JFLAP one can run them and see how long (how many steps) they take to figure out the running time.

6.6 Students using JFLAP

Students can use JFLAP outside of class for homework and for additional study problems they create. There are the obvious problems such as building an automaton and creating a set of test data, and stepping through one of the many transformations included in JFLAP. Students can also use JFLAP to study properties. For example, given two DFAs, students should create the DFA that represents the intersection of the two, and create test data to determine if they have been successful. One more use of JFLAP is that students can load files from lecture to recreate the instructor's examples presented in class.

7 Other Tools for Hands-On Visualization

There are several other tools for creating animations with scripts. Samba [12] has been used to create animations for many computer science topics including CS 1, CS 2, algorithms, and operating systems. AnimalScript [9] is a scripting language that includes a list element primitive and many other features.

We mention several other tools that can be used for teaching automata theory with hands-on visualization. Pâté [4] is a tool for parsing restricted and unrestricted grammars and a grammar transformer from a context-free grammar to Chomsky Normal Form. Given a grammar and an input string, an exhaustive search parser shows either the textual derivation or the parse tree. This tool can be used in conjunction with JFLAP in the example above in which JFLAP converts an NPDA to a CFG. The resulting CFG can then be tested with input strings to convince the students it recognizes the same input strings as the NPDA.

JeLLRap [7] is an instructional tool for building LL(1), LL(2) and LR(1) parse tables. JeLLRap guides the user through several steps in constructing a parse table. When the table is complete, the user can enter an input string and step through the building of the parse tree. JFLAP is a Java version of LLparse and LRparse [10].

Other tools include Turing's World[1], a tool for experimenting with Turing machines as building blocks, and Snapshots of the Theory of Computing [2], a hypertextbook for the web.

8 Future Work

We are currently working on a new version of JAWAA that will be easier to create animations by novices and by instructors of a class. To create an animation one will be able to create two JAWAA files, one using an editor to setup the animation, and then a second file created from a program to finish the animation We are also updating JFLAP to include some additional transformations.

Acknowledgements Thanks to the many students who worked on these tools, including Anna Bilska, Thomas Finley, Eric Gramond, Ted Hung, Pretesh Patel, Willard Pierson, Magda Procopiuc, Tavi Procopiuc, and Jason Salemme.

References

[1] Barwise, J., and Etchemendy, J. *Turing's World.* CSLI, 1993.

[2] C. Boroni, F. Goosey, M. G., and Ross, R. Engaging students with active learning resources: Hypertextbooks for the web. *Thirty-second SIGCSE Technical Symposium on Computer Science Education* (2001), 65–69.

[3] Gramond, E., and Rodger, S. H. Using jflap to interact with theorems in automata theory. *Thirtieth SIGCSE Technical Symposium on Computer Science Education* (1999), 336–340.

[4] Hung, T., and Rodger, S. H. Increasing visualization and interaction in the automata theory course. *Thirty-first SIGCSE Technical Symposium on Computer Science Education* (2000), 6–10.

[5] Linz, P. *An Introduction to Formal Languages and Automata, Third Edition.* Jones and Bartlett, 2001.

[6] Pierson, W., and Rodger, S. H. Web-based animation of data structures using jawaa. *Twenty-ninth SIGCSE Technical Symposium on Computer Science Education* (1998), 267–271.

[7] Rodger, S. Visual and interactive tools. www.cs.duke.edu/∼rodger/tools/tools.html.

[8] Rodger, S. H. Introducing computer science through animation and virtual worlds. *Thirty-first SIGCSE Technical Symposium on Computer Science Education* (2002), 186–190.

[9] Rossling, G., and Freisleben, B. Animalscript: An extensible scripting language for algorithm animation. *Thirty-second SIGCSE Technical Symposium on Computer Science Education* (2001), 70–74.

[10] S. Blythe, M. J., and Rodger, S. H. Llparse and lrparse: Visual and interactive tools for parsing. *Twenty-fifth SIGCSE Technical Symposium on Computer Science Education* (1994), 208–212.

[11] Sedgewick, R. *Algorithms.* Addison Wesley, 1988.

[12] Stasko, J. Using student-built algorithm animations as learning aids. *Twenty-eighth SIGCSE Technical Symposium on Computer Science Education* (1997), 25–29.