Java Applets to Reinforce Fundamental Computer Science Concepts

Michael J. Quinn
School of Electrical Engineering and Computer Science
Oregon State University

Abstract

We describe a set of 15 interactive animations developed for college freshmen taking a computer science orientation course. The purpose of using the interactive animations is to improve students' understanding of conceptual and procedural knowledge fundamental to the field of computer science. The animations have been implemented as Java applets. We explain how introducing applet-enabled activities into lectures has affected the classroom experience. We present evidence that the use of these animations has significantly improved the performance of students on exam questions.

Introduction

Computer science majors at Oregon State University take a course called Computer Science Orientation during the first term of their freshman year. The class meets for about four hours per week: three 50-minute lectures and one 80-minute recitation. The primary goal of the ten-week course is to introduce students to some of the major concepts in computer science. In this respect it resembles CS100B in Computing Curricula 2001.[1] However, the course also has secondary goals of teaching basic university survival skills, introducing career options, and giving students some experience solving problems as members of a team. An implicit, overarching goal of the course is to increase the retention rate of students enrolled as computer science majors.

In the decade since we created Computer Science Orientation, we have consistently seen a large percentage of students who do poorly on the exams. In particular, students have performed poorly when asked questions requiring them to demonstrate their procedural knowledge, such as converting binary numbers to decimal or adding integers stored in two's complement form. We hypothesize that students are not spending enough time practicing these tasks. Traditionally, we have assigned students only a few end-of-chapter review questions per week, because the class is large (typically over 100 students), and the teaching assistants do not have the time to grade a large number of homework exercises. We have looked for a way to get students more engaged with the material and to increase the amount of time they spend solving problems, without increasing the amount of grading needed.

In 2004 the College of Engineering at Oregon State University required every incoming freshman to have a laptop computer. We have taken advantage of this opportunity by creating interactive applets that enable the instructor to introduce "active learning" situations into course lectures. Students use the applets in class to answer questions posed by the instructor. When doing homework assignments, students can use some of the applets to help them check their solutions. Before exams, we hand out study sheets designed to encourage students to strengthen their conceptual and procedural knowledge by running the applets again. The applets provide interactive, animated experiences that complement the course's traditional lectures and textbook. Our hypothesis is that the availability of these applets will help students improve their performance on exams. Getting students off to a stronger start may subsequently reduce the attrition of students from computer science.

Related Work

Many efforts have been made to enhance computer science education through the use of animation and interactive software. In this section we briefly describe some of the projects that are most closely related to our work.

More than two decades ago, Ronald Baecker at the University of Toronto released the video *Sorting out Sorting*, which used animation to compare nine internal sorting algorithms.[2] Since then, three studies have concluded that passive viewing of animations is not sufficient to improve student understanding.[3,4,5] Instead, algorithm animations must be part of an active learning environment to be successful. Human explanations need to accompany the animations. Allowing students to create the data input to the animated algorithms also improves the student experience.

Peter McDonald and Vic Ciesielski have described an interactive algorithm animation tool that helps students understand state space search.[6] The DELYS software environment, created by Vassilios Degdielelis et al., helps secondary school students learn fundamental computer science concepts, including the components of a computer, the booting process, and internal representations of data.[7] David Eck has produced Java applets to accompany his introductory computer science textbook, *The Most Complex Machine: A Survey of Computers and Computing*.[8] Eck's applets illustrate data representations, logic circuits, memory circuits, the instruction execution cycle, assembly language programming, and Turing machines.

Our work is distinguished from the previous efforts by the breadth of topics covered and the number of applets developed and tested.

Project Goal

Our goal is to help students *understand and apply conceptual knowledge and procedural knowledge*.[9] Examples of freshman-level conceptual knowledge are data representations, overflow and round-off error, starvation and deadlock, and the meaning of complexity classes. Examples of procedural knowledge are adding binary numbers, negating an integer stored in two's complement form, and converting a while loop to a repeat loop.

We aim to achieve our goal through the creation and use of Java applets that illustrate these key concepts and procedures. The applets are used in two different ways in the context of the course. The first use is within the lecture. An interactive applet can provide a way to jar students out of a passive note-taking mode and get them into problem-solving mode. The second use is after the lecture. An applet can be a tool that allows students to explore, check homework solutions, and prepare for examinations.

Our introductory computer science orientation class has 29 fifty-minute lectures. About two-thirds of these lectures are devoted to introducing technical material. The rest of the lectures are spent discussing generic topics of benefit to freshmen, such as time management, group dynamics, and opportunities for international study. In order to have each technical lecture accompanied by an applet, we would need a total of about 18 applets. We were able to produce 15 applets before the class began in September 2004. The applets did not map evenly to the lecture topics. Some lectures had two relevant applets, while others had none. In the end, we used the applets in 10 lectures, or about one-third of the total number of lectures in the class.

Table 1 summarizes the goal(s) for each of the applets we have created.

Applet Descriptions

This section describes in more detail the 15 applets we have created. The consistent "look and feel" of the applets was relatively easy to ensure, since two part-time undergraduate computer science students created 14 of the applets over the summer of 2004. The applets are accessible at:

http://web.engr.oregonstate.edu/~quinn/education/.

Applet	Designed to Help the Student…
1. Boolean Operators	Understand and carry out boolean operations
2. Gates and Flip-Flops	Compute the output of logical circuits
3. Number Conversions	Convert numbers from one base to another
4. Two's Complement Addition	Understand & compute sums and overflow conditions
5. Storing Fractions	Understand representation of fractional numbers
6. Brookshear Machine	Understand instruction execution and write simple machine-language programs
7. Dining Philosophers	Understand deadlock and starvation
8. Packet Routing	Understand packet routing
9. Magnetic Disks	Understand how algorithm choice can affect embedded system performance
10. While/Repeat Loops	Understand and carry out conversions between different iterative structures
11. Searching Algorithms	Understand difference between O(lg n) and O(n) complexity classes
12. Sorting Algorithms	Understand difference between O(n lg n) and O(n^2) complexity classes
13. Parse Trees	Understand how binary trees can represent expressions and how operators can have different precedence
14. Search Trees	Understand difference between breadth-first search and depth-first search of state space trees
15. Turing Machine	Understand how Turing machines compute functions

Table 1: List of Java applets and their objectives.

1. The **boolean operators** applet focuses on binary logic operations. The user can select the number of bits in the operands (2, 4, 8, or 16) and the logical operator (AND, OR, NAND, NOR, XOR, XNOR). In demo mode the user is able to flip any bit in either operand, and the result is dynamically recomputed. In test mode, the result is fixed. The goal of the user is to change bits in one or both operands as necessary so that the result of the logical operator in each bit position matches the result shown. When the bits are correct, the applet displays a "Good Job!" message and gives the user the opportunity to create a new test case.

2. The **gates and flip-flops** applet demonstrates combinatorial circuits and flip-flops. The user can select among four different logical circuits, each with 1 to 4 gates. The user can select the type of gate at each point in the circuit. Once the gates are chosen, a truth table shows the output of the circuit for each combination of inputs. The user can click on an input bit to flip its value. When input values change, the output of the circuit is recomputed.

3. The **number conversion** applet converts positive integers from one notation to another. It supports decimal, binary, octal, and hexadecimal numbers. The user clicks on a radio button to select a notation and types a number in the selected notation. By clicking on another radio button, the user can see the value of the number in other notations.

4. The **two's complement addition** applet, shown in Figure 1, demonstrates the

addition of integers stored in two's complement format. In demo mode the student clicks on operand bits to flip their values. The result is updated dynamically. In test mode the result, the overflow indicator, and the second operand are fixed. The user flips bits of the first operand to make the sum correct. When the first operand has the correct value, a "Good Job!" message appears, along with a button giving the user the opportunity to try another test case.

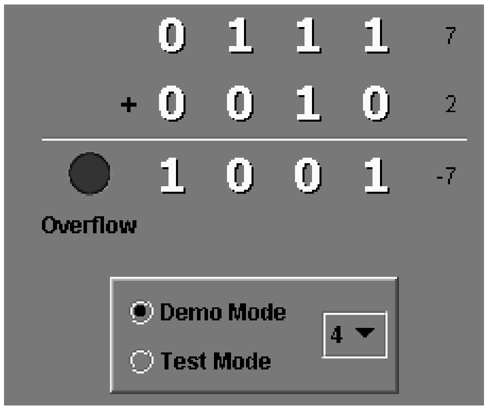

Figure 1: Screen shot of the two's complement addition applet.

5. The **storing fractions** applet is designed to help students understand the representation of floating-point numbers as a sign bit, exponent, and mantissa. Students can flip any bit in the binary representation of the floating-point number, and the applet dynamically re-computes and displays the decimal value, along with its constituent parts: the values of the sign bit, exponent, and mantissa.

6. The **Brookshear machine language** applet emulates the simple eight-bit computer described in the course textbook, *Computer Science: An Overview* by J. Glenn Brookshear. Students can type in machine language programs and watch them execute. The applet provides students with a view of the program counter, all 16 registers, and all 256 memory locations. It highlights the memory and register locations that are accessed or modified as each instruction executes.

7. The **Dining Philosophers** applet demonstrates the concepts of process starvation and deadlock in the context of the classic Dining Philosophers problem. In test mode the student must create a deadlock among the five processes.

8. The **packet routing** applet demonstrates a layered approach to transferring data across large networks. In the applet window, the user sees a network of computers. Each computer has a unique IP address. When the user clicks on Demo, the applet begins a data transfer. A three-packet message is transported from the source computer to a randomly chosen destination computer. In test mode, the user must first examine the contents of the message header to find the IP address of the destination computer, then route the message by clicking on intermediate computers along the path from the source to the destination.

9. The **magnetic disks** applet creates an animation that compares two different algorithms for deciding the order in which file accesses should be handled. The first-come, first-served algorithm puts access requests in a queue. The Elevator algorithm puts access requests in a priority queue based on the direction of motion of the disk's read head. The user clicks on file icons to add requests to both queues, then observes the performance difference of the two algorithms.

10. The **while/repeat loop conversion** applet tests the user's ability to convert a while loop into a repeat loop or vice versa. The user is able to select among a set of pre-arranged examples. For each example, the left window contains the original loop. The user types the converted loop into the right window. A "Check" button allows the user to check to see if the conversion is correct.

11. The **searching algorithms** applet, shown in Figure 2, compares sequential search with binary search. The user clicks on Start to watch both algorithms execute a single search. A slider bar allows the user to select the number of elements being searched. By clicking on Demo, the user initiates a series of searches with a varying numbers of elements. A graph at the bottom of the applet displays the average number of comparisons required by each algorithm.

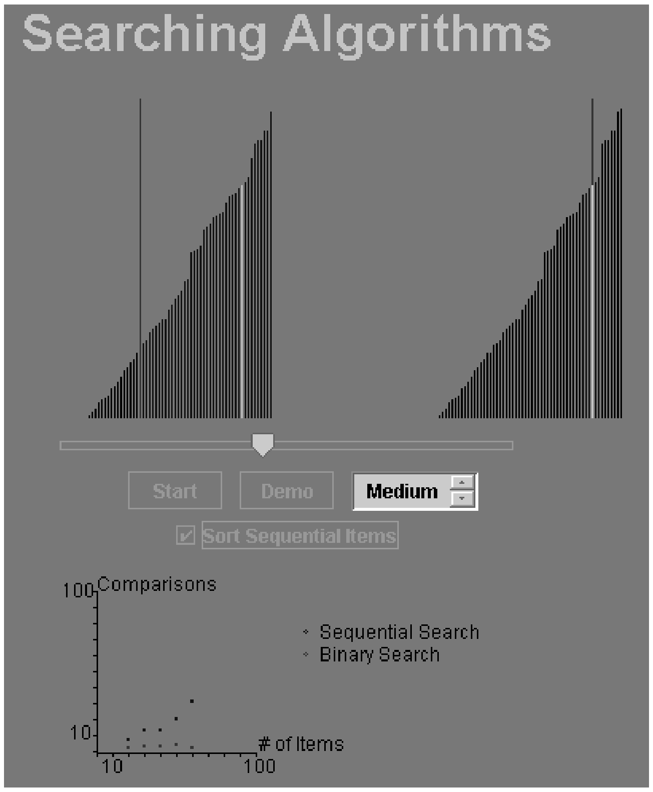

Figure 2: Screen shot of the searching algorithms applet.

12. The **sorting algorithms** applet, shown in Figure 3, is designed to help student understand the significant difference between the complexity classes O(n^2) and $O(n \lg n)$ as n increases. The applet implements an animation that “races” two algorithms the user chooses from the following list: quicksort, selection sort, insertion sort, merge sort, and bubble sort. The student may use a slider bar to select the number of elements to be sorted (between 10 and 100).

13. The purpose of the **parse trees** applet, shown in Figure 4, is to help students understand how arithmetic expressions are represented as binary trees. The student inputs an expression that may contain integer and floating-point values, parentheses, and the operators +, -, *, and /. When the student clicks the Parse button, the applet displays the parse tree. The operators * and / have higher precedence than + and -.

14. The **search trees** applet compares two exhaustive search methods (breadth-first search and depth-first search) on a 2-by-3 sliding tile puzzle (a simplified version of Loyd’s famous 15-puzzle). With a radio button, the user can switch between the two kinds of search. Clicking on Step causes the search to examine a single node of the state space tree. Clicking on Run animates the search. A status window unveils the depth of the search, the number of nodes examined so far, and the number of unexamined nodes.

15. The **Turing machine** applet animates the actions of a Turing machine. Currently, there are two different functions to choose from. However, the user is allowed to move the read/write head and initialize the contents of the tape. The user may single-step through the simulation or allow the machine to run automatically.

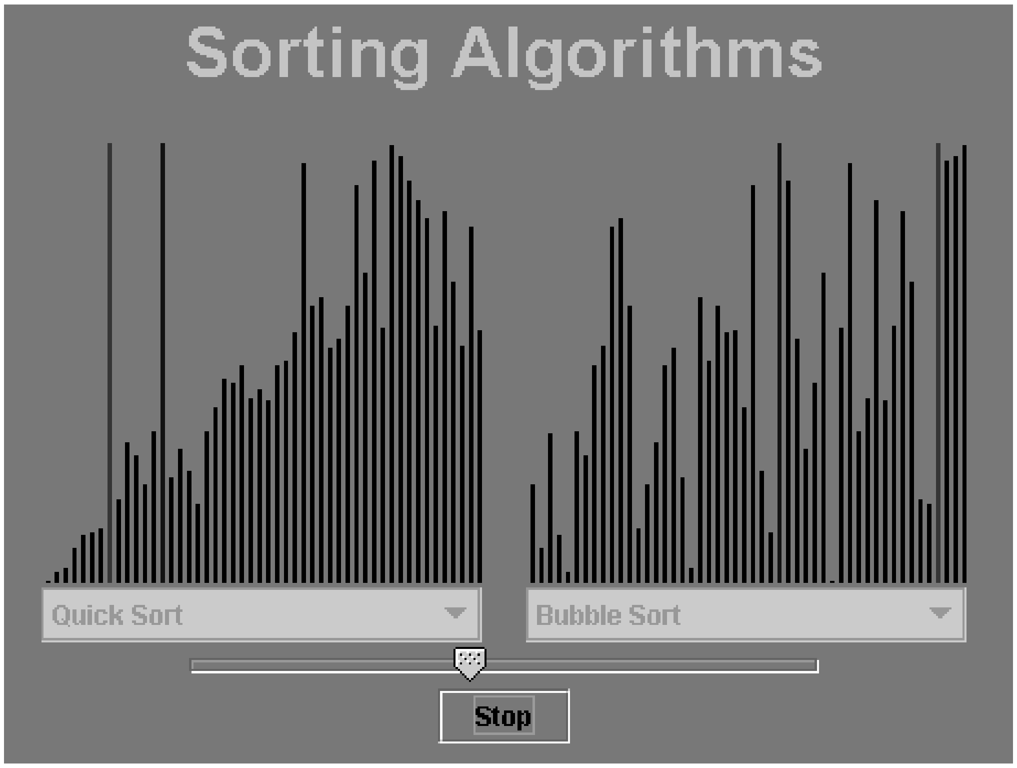

Figure 3: Screen shot of the sorting algorithms applet.

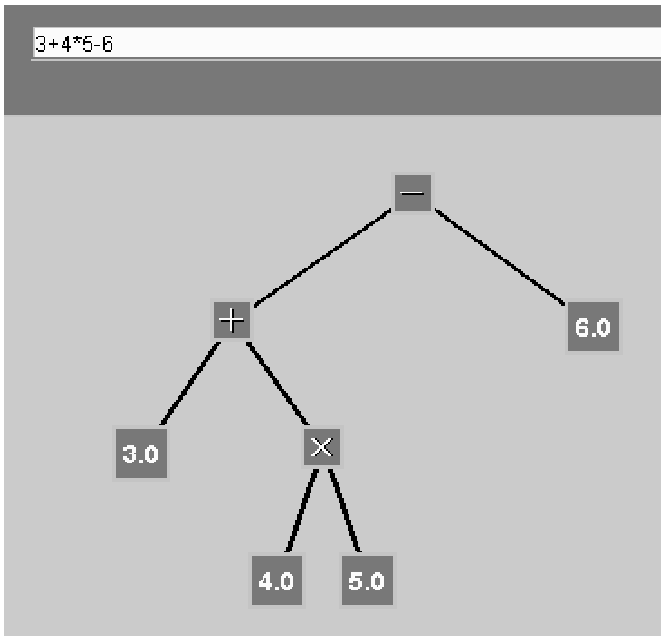

Figure 4: Screen shot of the parse trees applet.

Experience Using the Applets

Computer science has traditionally been one of the more popular majors at Oregon State University. Typically we have had a single lecture section for all of the computer science freshmen. With class sizes usually at 100 or more, it is not practical for the instructor to keep students engaged by calling on them to answer questions. Our hope was that the use of the applets could keep students out of passive, note-taking mode and allow them to apply the ideas being introduced in the lectures.

We found that some of the applets worked much better than others in achieving this aim. For example, the two's complement applet worked well. The functionality of the applet is easy to grasp. After introducing the applet, it was easy to think of questions for students to answer. Here are two examples. What is the fewest number of bits you can set to 1 to cause an overflow? How can the computer automatically detect an overflow?

The sorting algorithms applet was also well received by the students. It dramatically illustrates the difference between an $O(n \lg n)$ sort such as quicksort and an $O(n^2)$ sort such as bubblesort. It, too, was amenable to interesting questions, such as "Under what circumstances does insertion sort run as quickly as quicksort?" and "If quicksort and merge sort are both *O(n* lg *n*), why does quicksort finish faster than merge sort?"

Unfortunately, other applets came across more like static presentations and did not engage the students as fully. The Dining Philosophers and search trees applets were interesting to watch for a few minutes, but they did not prove to be rich sources of thought-provoking questions.

We found that adding an applet-oriented activity to a 50-minute lecture significantly changed the amount of technical material we could cover. We taught the class in a large lecture hall, and we presented most of the technical material on the blackboard. When we began an applet-oriented activity, students would fire up the applet on their laptop computers. The instructor would also run the applet, which would be displayed on a large screen at the front of the classroom. It took 3-5 minutes to adjust the lighting, lower the screen, engage the instructor's computer with the projection system, and initiate the applet. The actual coverage of the applet, including the time students spent answering the instructor's questions, usually filled 5-7 minutes. If the applet was presented in the middle of the lecture, another 2 minutes were spent shutting down the computer, raising the screen, and adjusting the lights. In other words, about 20-25% of the lecture was spent on the applet-oriented activity. We found that use of the applets reduced the amount of technical material covered per lecture. As a result, we covered fewer topics than we did in prior years.

Evaluation

Our hypothesis was that using interactive animations to reinforce important concepts and procedures would improve the performance of the students on the examinations. To test our hypothesis, we have compared the performance of the freshman computer science orientation class in 2004 versus the similar class in 2002. (The 2003 class was not a suitable candidate for a control group, because a different textbook was used that year.)

Besides the availability of Java applets, the following variables that could also affect the performance of a class on examination questions: the size of the class, the quality of the students in the class, the location of the class, the instructor, the book, and the particular questions asked. The 2002 freshman orientation class is an excellent control group, because these variables are either identical or very close. The greatest difference between the two years is the size of the freshman class: 128 in 2002 versus 92 in 2004. However, in 2002 the class was divided into two lecture sections. The larger of these lecture sections, with 81 students, is smaller than the size of the single lecture in 2004. About one third of the freshmen in 2002 enjoyed a much smaller computer science orientation class with less than 50 students in it. If a smaller class size is beneficial, the advantage goes to the control group. Table 2 summarizes the differences between Computer Science Orientation as offered in 2002 and 2004.

Comparing the goal of each applet with the course learning objectives, we found that four of the learning objectives were addressed by at least one of the applets. Each of these learning objectives was evaluated by a series of questions on the midterm or final examination. For each objective, we gave the 2002 exam questions to the 2004 class to see if student performance would improve. Table 3 summarizes our results.

For each of the four learning objectives, we used the two-sample t-test to compare the average grades earned by the students in the two classes. In the case of determining the output of simple circuits, the mean grades of the two classes were not significantly different (p-value = 0.22). This is not surprising, since the mean exam grade earned by the students before the introduction of the applets was already above 90%. For the other three learning objectives,

Variable	Year	
	2002	2004
Size of class	128	92
Mean high school GPA	3.43	3.38
Lecture size(s)	81, 47	92
Room(s)	Cordley 1109, Covell 216	Cordley 1109
Instructor	Quinn	Quinn
Book	Brookshear, 7th edition	Brookshear, 8th edition
Questions asked	Identical questions for each learning objective	

Table 2: Comparison between the control group (2002 class) and the test group (2004 class). Every variable is identical or very close.

Learning Objective	Exam Question Grade Mean (Std. Dev.)	
	2002	2004
Determine output of simple circuits	91.4% (16.6%)	94.1% (13.2%)
Represent binary, decimal, hexadecimal integers	81.8% (19.8%)	92.2% (12.8%)
Add two's complement integers	63.2% (26.6%)	81.4% (23.1%)
Write simple machine language program	56.6% (15.4%)	78.8% (28.7%)

Table 3: Mean exam question grades for four course learning objectives.

the mean exam grade earned by the class that had exposure to the Java applets was significantly higher than the mean grade achieved by the 2002 class (p-value < 0.001).

Summary

We have used Java applets to create a set of interactive animations designed to promote active learning opportunities for students taking a computer science orientation course. Our goal is to improve these students' understanding of conceptual and procedural knowledge fundamental to the field of computer science.

The use of these animations has added variety to the traditional "chalk talk" lectures. On the other hand, because an applet-oriented activity consumes 20-25% of a lecture, we have had to reduce the amount of material presented in a single lecture. That has led to a reassessment of the importance of each concept presented during the lectures.

Short-term results are positive. We have evidence that the use of these animations has significantly improved the performance of students on exam questions related to the course's learning objectives. At this point it is too early to see if these changes will help reduce the percentage of students who drop out of the computer science major.

Acknowledgements

Three undergraduate computer science majors at Oregon State University - E. Rogan Creswick, Stacy Brock, and Gary Hertel - wrote the Brookshear machine language applet. The Turing machine applet was adapted from a Java applet originally created by Suzanne Skinner (http://ironphoenix.org/tril/tm/). The rest of the applets were implemented by Casey Manion and Robert Ottaway, computer science undergraduates at Oregon State University. This work was supported by a grant from the William and Flora Hewlett Foundation through its Engineering Schools of the West Initiative.

Bibliography

1. The Joint Task Force on Computing Curricula, IEEE Computer Society, Association for Computing Machinery, *Computing Curricula 2001, Computer Science Volume* (December 15, 2001).

2. R. Baecker, “Sorting Out Sorting: A Case Study of Software Visualization for Teaching Computer Science,” in *Software Visualization: Programming as a Multimedia Experience*, MIT Press (1998), pp. 369-381.

3. J. Stasko, A. Badre, and C. Lewis, “Do Algorithm Animations Assist Learning? An Empirical Study and Analysis,” in *Proceedings of ACM INTERCHI ’93 Conference on Human Factors in Computing Systems, Understanding Programming* (1993), pp. 61-66.

4. A. Lawrence, A. Badre, and J. Stasko, “Empirically Evaluating the Use of Animations to Teach Algorithms,” in *Proceedings of the 1994 IEEE Symposium on Visual Languages* (1994), pp. 48-54.

5. C. Kehoe, J. Stasko, and A. Taylor, “Rethinking the Evaluation of Algorithm Animations as Learning Aids: An Observational Study,” Technical Report GIT-CVU-99-10, Georgia Institute of Technology (1999).

6. P. McDonald and V. Ciesielski, “Design and Evaluation of an Algorithm Animation of State Space Search Methods,” *Computer Science Education* 12, 4 (2002), pp. 301-324.

7. V. Dagdilelis, G. Evangelidis, M. Satratzemi, E. Vassilios, and C. Zagouras, “DELYS: A Novel Microworld-based Educational Software for Teaching Computer Science Subjects,” Computers & Education 40 (2003), pp. 307-325.

8. D. Eck, “Labs and Applets for *The Most Complex Machine,*” http://math.hws.edu/TMCM/java/index.html (June 2004).

9. L. W. Anderson and D. R. Krathwohl, editors, *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom’s Taxonomy of Educational Objectives, Complete Edition*, Longman, New York, NY (2001).

Biographical Information

Dr. Michael J. Quinn is a professor of computer science at Oregon State University. He helped create Oregon State University’s freshman computer science orientation course in the mid-1990’s, and he has been actively involved in teaching and improving it ever since. He is author or co-author of seven textbooks in the areas of parallel computing, computer ethics, and computer concepts.