Using 3-D Interactive Animation To Provide Program Visualization As A Gentle Introduction To Programming[1]

Wanda Dann

Computer Science Dept.
Ithaca College
Ithaca, NY 14850
1-607-274-3602
wpdann@ithaca.edu

Stephen Cooper

Computer Science Dept.
Saint Joseph's University
Philadelphia, PA 19131
1-610-660-1561
scooper@sju.edu

Randy Pausch

Computer Science Dept.
Carnegie Mellon University
Pittsburgh, PA 15213
1-412-268-3579
pausch@cs.cmu.edu

ABSTRACT

This paper is an **extended abstract for a work in progress**. The work, as described in this paper, is a study of the use of 3D animation for program visualization in an introductory programming course for computer science majors who have gaps in their academic background. A quick summary of studies of attrition from the computer science major is presented as a background and motivation for the investigation described here. Program visualization using 3D animation allows students to view changes in the state of their virtual worlds, thereby connecting individual lines of code to the animation action. Early results of this study include a comparison of students in CS1 who have taken the course versus those students who did not.

1. BACKGROUND AND MOTIVATION

Attrition has been a longstanding problem for computer science programs in academia. Of course attrition in computer science majors occurs throughout the four years, but the largest percentage of dropouts occur during the first year of study.[11] High attrition rates over the last decade have occurred during the same period of time as most computer science departments have made the switch from imperative to object-oriented (OO) languages. Not only do OO languages require the teaching and learning of all the traditional fundamental concepts, but OO languages add the more demanding concepts of class, object, information hiding, inheritance, and polymorphism. We conjecture that a correlation exists between high attrition rates of the last decade and the switch to OO languages. Data supporting this conjecture comes from a contrast of studies from early researchers with more recent researchers. Early researchers, using imperative languages, (such as Butcher & Muth[2]; Stephens et al.[13]; Oman[7]) showed little to no relationship between prior programming experience and success in the curriculum. In contrast with those studies, later researchers Davy et al.[5] and Hagan and Markham[6], using object-oriented languages, indicate a strong relationship between prior programming experience and success in the computer science major.

Our motivation is to reduce attrition in the first year of the computer science program. We ask: "Which students are at most risk of dropping out of the major?" and "What can be done to decrease attrition?" Using historical enrollment and dropout data at our own institutions, we conclude that students who have little or no previous programming experience and who are not yet ready for calculus have an extremely high dropout rate (approximately 90%). In an effort to decrease attrition in the computer science major in the first year, we have developed a course for introducing fundamentals of object-oriented programming to these "high-risk" stu-

[1] This work is supported in part by a grant from the National Science Foundation, DUE- 0126833.

dents as a preparation for a rigorous CS1 course.[3] This pre-CS1 course is thoroughly based on the use of program visualization.

2. USING PROGRAM VISUALIZATION

The program visualization tool we choose to use is the 3D interactive animation environment, Alice (freely available at www.alice.org). This tool follows in the footsteps of Logo, Karel the Robot, and Karel++. [8, 9, 1] Other tools that might be used include Samba[12] and JAWAA[10].

The total focus of our pre-CS1 course is the use of program visualization to support the pedagogical goal: teaching fundamental concepts of object-oriented programming and problem solving. Students are assigned problem-solving tasks where the animation visualizes a solution to the problem. Importantly, the virtual world is one created entirely by the student and the program code is the student's own code. The program visualization can be incrementally developed and executed again and again, line by line.

By executing program code for animating objects in a virtual world, students view the progression of changes of state in the virtual world.[4] The smooth animations in 3D allow the student to connect individual lines of code to the animated actions of the objects. For example, a smoothly animated "move" command that evidences a bug where the object moves too far and runs off the screen is more easily interpreted than an instantaneous move, where the object "teleports" and therefore is misperceived as a "delete" operation. During the course, students develop and test a series of problem solving projects, illustrating progressively more demanding programming concepts.

The programming language concepts used in building 3D animations in Alice include:

- Action commands (such as move, turn, roll, and resize) cause state transformations in the students' virtual worlds. These commands are similar to assignment statements in more traditional programming languages. Mutable variables can be avoided, as objects maintain their own properties.
- Behavior methods allow students to group action commands into one named instruction, to define their own commands. The concept of a behavioral method is similar to the procedure concept in many other programming languages (or a function that just performs side effects, not returning anything).
- Functions (implemented as questions), decisions, expressions, collections, and repetitions are all similar to any other programming language. But, in a 3D environment students can easily see the consequences of their execution.
- Recursion is used for repeated animated actions.

Object-oriented programming concepts are integral to the 3D program visualization. All 3D objects are obviously objects and have behaviors, state, and identity. These objects (e.g., animals and vehicles) populate the world as seen in the onscreen visualization. By writing simple code, object appearance and behavior is controlled. Likewise, events and interaction are a natural part of the visualization. Students write programs to make object actions respond to mouse and keyboard input. For example, students build simple flight simulators and space invader kinds of games.

A great advantage of using 3D animation to create program visualizations in the Alice environment is that concurrency is trivial to understand. Some actions must occur simultaneously while others must occur in sequence. An ice skater that turns and moves simultaneously can skate around a hole in the ice, while the same actions in sequence cause the skater to fall into the icy water.

3. RESULTS

Ithaca College (IC) and Saint Joseph's University (SJU) have conducted this course over two semesters under different experimental conditions. At IC, the course was offered concurrently with the CS1 course. At SJU, the course was a stand-alone class that preceded CS1. A study of the effectiveness of this course in its different manifestations is currently underway. Early results of this study are based on a comparison of students in CS1 who have taken the course versus those students (with similar math and computing backgrounds) who did not take the course. Results include an evaluation of whether this course provides sufficient "prior programming experience" as discussed by Davy, et al.[5] Particular aspects include student performance in CS1 and student level of confidence in CS1, along with an analysis of retention. Early results, not ready for publication at this writing, will be summarized in presentation at this workshop.

4. REFERENCES

[1] Bergin, J., Stehlik, M., Roberts, J., & Pattis, R. Karel++, A gentle introduction to the art of object-oriented programming. New York: Wiley, 1997.

[2] Butcher, D. & Muth, W. Predicting the success of freshmen in a computer science major. Communications of the ACM, 28(3), (1985), 263-268.

[3] Cooper, S., Dann, W., & Pausch, R. Using Animated 3D Graphics To Prepare Novices for CS1, to appear in Computer Science Education Journal, 2002.

[4] Dann, W., Cooper, S., & Pausch, R. Making the connection: programming with animated small worlds. Proceedings of the 5^{th} Annual Conference on Innovation and Technology in Computer Science Education, Helsinki, Finland, (2000), 41-44.

[5] Davy, J.D., Audin, K., Barkham, M., & Joyner, C. Student well-being in a computing department. Proceedings of the 5^{th} Annual Conference on Innovation and Technology in Computer Science Education, Helsinki, Finland, (2000), 136-139.

[6] Hagan, D. & Markham, S. Does it help to have some programming experience before beginning a computing degree program? Proceedings of the 5^{th} Annual Conference on Innovation and Technology in Computer Science Education, Helsinki, Finland, (2000) 25-28.

[7] Oman, P. Identifying student characteristics influencing success in introductory computer science courses. AEDS, 19(2-3), (1986), 226-233.

[8] Papert, S. MindStorms: children, computers, and powerful ideas. New York: Basic Books, 1980.

[9] Pattis, R. (1981). Karel the robot. New York: Wiley.

[10] Rodger, S. Introducing computer science through animation and virtual worlds. Proceedings of the 33^{rd} SIGCSE Technical Symposium, Cincinnati, (Feb. 2002), 186-190.

[11] Seymour, E. and Hewitt, N. *Talking about leaving.* Boulder : Westview Press, 1997.

[12] Stasko, J.T. Using student-built algorithm animations as learning aids. Proceedings of the 28^{th} SIGCSE Technical Symposium, San Jose, (1997), 25-29.

[13] Stephens, L., Konvalina, J., & Teodoro, E. Procedures for improving student placement in computer science. Journal of Computers in Mathematics and Science Teaching, 4(3), (1985), 46-49.