Using 3D Animation Programming in a Core Engineering Course Seminar

Richard Zaccone
Bucknell University
Computer Science Department
Lewisburg, PA 17837
Email: zaccone@bucknell.edu

Stephen Cooper
Saint Joseph's University
Computer Science Department
Philadelphia, PA 19131
Email: scooper@sju.edu

Wanda Dann
Ithaca College
Computer Science Department
Ithaca, NY 14850
Email: wpdann@ithaca.edu

***Abstract*— A core engineering course plays a vital role in the curricula of schools of engineering. At Bucknell University, Exploring Engineering (ENGR100) is the core engineering course where first year students are introduced to the study and practice of engineering and presented with overviews of specific engineering disciplines. In this paper, we describe an innovative approach for the 3-week programming seminar component of ENGR100. This approach makes use of a 3D animation tool for program visualization. Program visualization is used to introduce traditional programming concepts in a short time frame. The primary goal of this approach is to provide: a firm foundation for the novice programmer, a challenge for the experienced student, and a collaborative project experience for all students. A description of the first trial results of this innovative approach is included in the paper.**

***Index Terms*— CS1, Alice, programming, visualization.**

I. Introduction

In the early 1990s the National Science Foundation's Engineering Education Coalitions Program funded six coalitions [1]. These NSF-funded coalitions, such as the Foundation Coalition [2], have stimulated change in Engineering Education curricula at colleges and universities across North America. The innovations often include the introduction and/or redesign of freshman level courses to encourage creative, hands-on design projects, multidisciplinary teaming, and use of computer technology. One course that has been added to many curricula is a core engineering course designed to give freshmen a realistic overview at what engineering is all about as well as develop foundation skills such as an understanding of modeling and application of problem-solving skills. Part of the reason for a first-year core engineering course is an effort to improve retention of engineering majors without the side effect of "watering down" traditional courses.

At Bucknell University, Exploring Engineering (ENGR100) is the core engineering course where first-year students are introduced to the study and practice of engineering. The first few weeks of the semester are devoted to specific engineering discipline overviews. Following the discipline overviews, ENGR100 students select and participate in two choices from several three-week seminars. One of the seminars that students may select is an introduction to computer programming. From the perspective of the core course, the computer programming seminar provides a general understanding of what programming is, an opportunity to apply problem-solving skills in designing and writing computer programs, and experience in collaborative projects.

From a Computer Science perspective, the major goals of the three-week programming seminar go beyond that of the overall core course. The seminar is expected to prepare students for going on to an object-oriented CS1 course. It is also important to note, however, that the Computer Science department prefers not to duplicate the first three weeks of CS1. This preparation provided in the seminar is of particular importance to students who have little or no previous programming experience. The seminar offers a means to "level the playing field" for these students. In addition, it is hoped that the seminar will attract some undecided students to computing majors.

The educational problem faced by the instructor of a computer programming component of a core engineering course is the extremely limited time frame (three weeks at Bucknell). The challenge is to create a firm understanding of object-oriented programming concepts and at the same time provide experience with a team project. The greatest jeopardy is that students who have little or no previous programming experience will be unable to gain enough knowledge and skill with fundamental concepts to allow them to reasonably succeed in a rigorous CS1 course. At the same time, it is equally important that students who have previously studied programming are not lost to boredom.

In this paper, we describe an innovative approach for the programming seminar component of a core engineering course. This approach is designed to address the need for developing a foundation in object-oriented concepts in a very short time frame while avoiding duplication of the first three weeks of CS1. Simultaneously, this approach enables both the novice and experienced programmer to work on collaborative projects with a high level of interest. This approach makes use of a 3D animation programming environment named Alice (freely available at [3]).

II. The Software

Alice is a rapid prototyping environment for 3D object behavior, designed to enable novice programmers to develop interesting 3D animations. 3D models of objects are added to

Fig. 1. An initial skater scene

a virtual world to set up an initial scene. In Figure 1, a skater, a second person, and a sign have been added to a virtual world (as have a hill, several trees, an ice patch, and a house). It is important to note that the concept of "object" is reified by the objects in the world. Objects can be turned 360 degrees to provide a visual perception of height, width, and depth of a real-world object. Thus, the object-oriented sense of an object is given some (virtual) reality.

Once objects are constructed and the scene is set up, students write programs to control the appearance/behavior of objects and to make objects respond to mouse and keyboard input. Students use the Alice drag-and-drop editor to write their own programs for animating the objects. The editor has drag-and-drop tiles that are used to put together common programming language constructs (e.g., *if*, *while*). The program tiles provide constructs in a C++/Java-like syntax and are equivalent in expressive power to these languages. The intention of a drag-and-drop editor is to protect students from making frustrating syntax errors. The editor only allows students to drag and drop program components to syntactically correct locations. For example, only a construct of type boolean may be dropped into the condition of a *while* loop. We note that syntax is still a part of what students are learning, but the frustrations of novice programmers in learning to enter code is reduced. A much more detailed description of the Alice programming environment is provided in [4] or [5].

As students write their own programs, an incremental approach to implementing and testing their programs can be used. That is, students are able to test out individual commands and methods to ensure they perform as expected. This enables students to see the relationship of the program construct to the animation action performed by method execution. A simple example is illustrated in Figure 2, where a student has written a method for animating a skate action for the ice skater object.

The skate method incorporates the use of a parameter (how many steps to skate forward), a loop (similar to the *for* loop in Java), as well as sequential and simultaneous execution of code. Specifically, the skater pushes off first with her right thigh and then with her left thigh. At the same time, she is also gliding forward. (And, to make the motion more realistic, at the same time she also slides slightly to her left, indicating the effect of pushing off with her right leg, and then to her right.) Finally, this skate method takes a parameter of type integer, and loops that many times, so she will skate forward the appropriate number of strides.

III. Conducting the Seminar

The programming seminar component of the ENGR100 course consisted of three one hour lectures and a two hour lab per week over a 3-week period. Students learned to write programs illustrating the following programming concepts: methods (at the world-level as well as at the class-level), repetition (*for* and *while* loops), decisions (*if*), concurrency, objects, class inheritance, encapsulation, events and behaviors. The following examples illustrate how we used Alice to teach CS1 topics.

objects	We discussed objects on the first day of class by showing objects such as rabbits and boats to the students. We even talked about state in a meaningful way because the state is something that the students can see.
methods	By the third lecture we began learning how to write methods. The students learned how to write a method that makes an object (with legs) walk and turn. This led naturally to a discussion of loops since marching (walking back and forth) is just a repetition of the walk method.
parameters	Supplying a count to the "march" method was an effective way to teach the usefulness of parameters.
inheritance	The students learned inheritance by extending the capabilities of an object so that their extended object could perform actions the parent object did not know how to do.
events	Objects in Alice can respond to events such as key presses and mouse clicks. We devoted a portion of a lecture to this topic, but most students had already figured it out on their own.
if	Students learned logical expressions and conditional execution by having an ice skater make different moves depending where she is on the ice. Other objects were used in student programs.

This is an impressive list of topics for a 3 week seminar. We felt that we were able to introduce and build a basic understanding of each of these topics because the program visualization in Alice made these concepts seem natural and easy to understand (see Results section below).

In addition to the day-to-day discussions and programming assignments, a complete programming project was assigned on the first day of class and was due by the end of the seminar. Students worked on their project in groups of three.

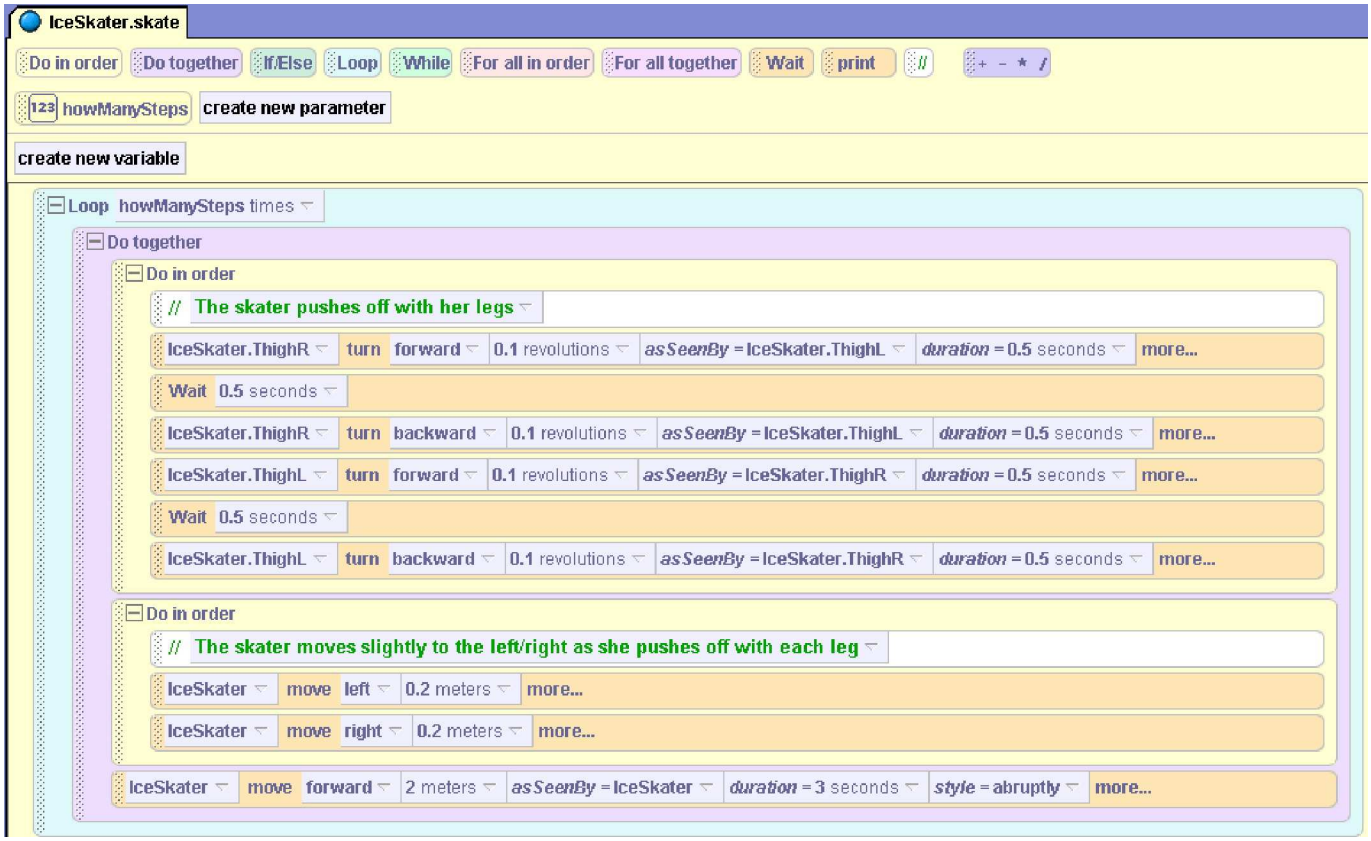

Fig. 2. The ice skater's skate method

The project was to create a computer animation (a more involved program than the in-class exercises and programs). The project animation specifications were that the animation must last at least 30 seconds and must also contain at least three methods, two of which use parameters in a meaningful way. The students had no restrictions on the subject of their animations.

IV. Results

In the first trial of this innovative programming seminar, the instructor of the seminar kept a log of observations of student participation and questions as well as his own teaching experience. (These observations were recorded as email to two of the authors of this paper.) At the end of the seminar, students were asked to complete a survey in which they provided feedback on the seminar. The instructor did not see student feedback until grades had been submitted. This section summarizes the instructor's comments and student reactions.

Approximately two-thirds of the students coming into this seminar had little or no prior programming experience. Despite their lack of experience, students were able to understand a wide range of fundamental programming concepts (outlined in the previous section of this paper). Student understanding of the concepts was evident in their ability to apply these concepts to a nontrivial project in just 3 weeks. In addition, students were introduced to even more advanced CS2 topics such as inheritance. The wide span of topics and the high-interest, 3-dimensional, game-like interface kept the more experienced students interested and challenged.

From a pedagogic standpoint, three weeks is a very limiting time for introducing programming concepts in any meaningful way. The Alice programming environment helped us overcome the time limitation by providing program visualization of objects and a drag-and-drop editor. Because programming in Alice is so visual, we were able to talk about objects on the first day. Objects were something that they could see on the screen. Alice's drag-and-drop editor had the students constructing scenes for their projects in the very first lab. By the second lab, students already had a significant portion of their animation project working, and by the end of the seminar, all of the students had completed their animation projects.

Alice supports collaborative programming and makes project coordination easy. The students found it easy to work together as a team because they could plan the animation of several different objects and assign the development of different objects to different team members. Each team member developed their own object independently and then integrated it with the other objects in the overall animation for the project.

Alice offers many advantages to novice programmers. The drag-and-drop editor encourages students to experiment. They can simply drag behaviors into an object and see what happens. Because there is no compile-link-run cycle, the visual feed-

back is immediate. The programs that students created were equivalent in expressive power to those written in a Java-like language. In addition to these benefits, the students were also freed from syntax concerns, to which students in most CS1 courses devote a significant amount of energy.

Overwhelmingly, students thought that programming with Alice was a fun experience. (This was despite the fact that we were using a pre-beta version of Alice2 that was somewhat unstable and crashed periodically. The latest version of Alice2 is far more stable.) In the survey distributed at the end of the seminar, a Lickert 5-point scale was used to solicit feedback. The average rating to a question regarding whether the seminar had increased their interest in Compute Science was 3.73. When asked whether they would recommend this seminar to other students, the average response was 4.05.

V. Alternative Approaches

Before deciding to use Alice, we considered several other possibilities. Below is a summary of our considerations.

A. Squeak

Perhaps the most appealing alternative was Squeak [6] [7]. Squeak is an open source implementation of Smalltalk-80 written in itself. Squeak has all the elegance and simplicity of Smalltalk with a good number of multimedia extensions [8]. The multimedia features of Squeak make programming fun and Squeak has a very simple syntax that is easy to learn. Despite its simplicity, we decided that there was still too much overhead for a three week seminar.

B. C, C++, Java

C++ and Java have their roots in C and it shows. We decided that the amount of syntactical overhead in these languages would make it difficult for novice programmers to do anything meaningful in just 3 weeks.

C. Visual Basic

VB was a potential alternative. While it is quite easy to create GUIs in VB, it was too feature laden to be digestible in three weeks. It also tends to encourage/allow poor programming techniques, such as the requirement of using global variables (required by the event handling mechanism) and, the allowance of not declaring variables (unless a compiler option is set).

VI. Conclusion

The programming seminar component of the ENGR100 core engineering course was successfully implemented using 3D animation for program visualization. Alice proved to be a very effective tool for introducing fundamental principles of programming. Program visualization and a drag-and-drop editor made it possible to introduce object-oriented topics including inheritance in just 3 weeks. We believe this introduction will lay a foundation for further study in CS1. The students found Alice to be a fun way to learn and expressed an increased interest in computer science.

Acknowledgment

This work was partially supported by NSF grant DUE-0126833.

References

[1] H. R. Coward, C. P. Ailes, and R. Bardon. (2000, July) Progress of the engineering education coalitions. [Online]. Available: http://www.nsf.gov/pubs/2000/nsf00116/nsf00116.pdf

[2] (2003) The Foundation Coalition website. [Online]. Available: http://www.foundationcoalition.org/

[3] (2003) The alice website. [Online]. Available: http://alice.org/

[4] S. Cooper, W. Dann, and R. Pausch, "Alice: A 3D tool for introductory programming concepts," in *Proceedings of the 5th Annual CCSC Northeastern Conference 2000, Ramapo, NJ*, 2000.

[5] ——, "Using animated 3D graphics to prepare novices for CS1," *Computer Science Education Journal*, 2003, (to appear).

[6] (2003) The squeak website. [Online]. Available: http://squeak.org/

[7] D. Ingalls, T. Kaehler, J. Maloney, S. Wallace, and A. Kay, "Back to the future: The story of squeak, a practical smalltalk written in itself," in *ACM SIGPLAN Notices, Proceedings of the 1997 ACM SIGPLAN conference on Object-oriented programming systems, languages and applications*, vol. 32, no. 10, oct 1997, pp. 318–326. [Online]. Available: http://users.ipa.net/~dwighth/squeak/oopsla_squeak.html

[8] M. Guzdial, *Squeak: Object-Oriented Design with Multimedia Applications.* Prentice-Hall, 2001.