Alice in CS 1

Mark Vellinga
Computer Science Department
Northwestern College
Orange City, IA 51041
vellinga@nwciowa.edu

ABSTRACT

In learning to program, some students struggle with many unknowns including the syntax of the language, workings of the Integrated Development Environment (IDE), problem solving, and the use of common programming constructs. A student's success in the class hinges on their ability to manage this fight and to cope with the number of unknowns. This paper presents an alternative in using Alice, a 3D interactive environment, as an attempt to help alleviate some of these student concerns.

INTRODUCTION

Students in CS 1 have a wide array of computing backgrounds they bring with them upon entry to the course. Some students have taken a rigorous programming course, have a strong mathematics background or have a basic idea of what programming entails, while other students lack programming experience or an adequate mathematics background.

There is also a large selection of concepts to learn in CS 1. A fundamental goal of most CS 1 courses would be to improve the problem solving skills of the students. These skills would be developed through the practice of designing and writing programs. The programs would be written in a language using an integrated development environment (IDE). So before the student can work on the improvement of problem solving, they must learn the workings of the IDE and the basic syntax of a language. Along with the individual constructs of the language, the student has a number of topics to learn and understand all at the same time. The success of the student in the course quite often depends on how well they are able to develop an understanding of each one of these components. Of course, this says nothing about learning to put all of the constructs together to write a satisfactory program. There seems to be too many things for the average student to juggle at the same time to make the course effective.

A common problem then is how does a teacher help the student manage all of these variables at one time? An instructor must balance between students with varying degrees of background in mathematics and experiences with a programming language. It could very well be the case that both extremes of the range of student could excel in the field of computer science. An instructor must also balance the development of a CS 1 course with regard to the range of concepts to expand in the learning process for the students. Randy Pausch summarizes a problem in teaching computer science: Too many of these classes require students to learn the basics of computer science by creating programs that sort numbers or perform other rudimentary tasks divorced from real-world applications or interests. Students also get overwhelmed by the object-oriented approach used in many introductory classes. This approach requires them to begin mastering abstract concepts such as objects and classes immediately, in addition to programming syntax and more fundamental concepts such as types, variables and references. [8] Without the use of a CS 0 course, an instructor would need to find an effective way to communicate the world of computer science to each student. This must be done in a way to not only inform the student but also attract and retain interested students to the world of computer science.

COMMON GROUND

Beginning the course from a common ground applicable to all students would be an ideal situation. From this beginning, the class could cohere in a unified fashion. The common ground should include a number of features beginning with a mutual understanding of what is included in the goals of the course. A student's background makes a difference in the ability to do well in the first computer science course. One advantage is the ability for some students to have an idea of what programming is about. This knowledge does not guarantee excellent performance in a CS 1 course, but does seem to give a certain amount of confidence to a beginning student. It was the author's experience in more than one CS 1 course that students would leave the course stating "I had no idea this course would be like this." This would occur even though the class was informed (or maybe better to say "warned") about the content of the course. For inexperienced students, there are just too many concepts to grasp and digest as they try to understand the idea of programming. Giving the students an opportunity to slowly and confidently explore this new subject is something that would enhance their learning and understanding processes. Hearing from the instructor is one thing, but experiencing it is better. Ensuring a successful programming experience for students is critical to the success of all students.

The common ground should also include a tangible understanding of terminology. The learning process for the student should include not only the proper introduction and explanation of each individual term, but also how the term relates or interacts with other terms that have been discussed. It seems that we are building a "learning world" for our students and want to be sure that the student has all of the necessary tools and is able to use the tools appropriately. Terms such as object, class, method, function, and attributes are important enough that a beginning student should be given time to process their meaning with regard to a programming language.

Finally, the common ground should also include an understanding of the primary programming constructs including loops, conditions, and sequential processing. Having an idea of basic programming constructs is also an advantage for the beginning student. Separating the notions of programming constructs and programming is crucial in learning to program. Giving the students the ability to understand individual constructs allows them to build a program from the constructs, or as Soloway has stated "putting the pieces together". With this confidence in basic constructs, the beginning student can figure out which to use and how to coordinate them in a program. Without this background, many students are not prepared to solve problems in the way needed for a computer program. Giving the students an opportunity to learn the constructs and effectively use them in their programs helps to develop this understanding.

ALICE

The programming environment that students use for their programs plays a significant role in the laying of the common ground. The ability to animate program execution and to visualize the program is a key component in support of the common ground. The animation of program execution can be used to help the student put the pieces together, while visualization is one approach to assist the student in finding out what task each piece can be expected to perform and how the pieces work together to perform the overall task of solving the problem at hand. [4]

Alice, which is freely available from http://www.alice.org, is such an environment. Alice is a 3D Interactive Graphics Programming Environment built by the Stage 3 Research Group at Carnegie Mellon University under the direction of Pausch. It is an object-oriented language that uses an animation and storyboard approach to teach students the fundamental ideas behind object oriented programming without requiring an undue focus on syntax. The Alice environment uses a drag-and-drop editor for composing object-oriented programs so that the syntax errors that so frustrate beginning programmers are eliminated. In addition, you'll get immediate visual feedback on the effect of your program instructions since the Alice environment encourages programmers to experiment and to explore three-dimensional programming techniques in an interpreted, object-oriented environment. Alice's primary design goal is to be accessible to people who don't necessarily possess a great deal of mathematical training or graphics programming experience, without unduly tying the hands of the expert programmer.

A screen capture of the interface is shown in Figure 1.

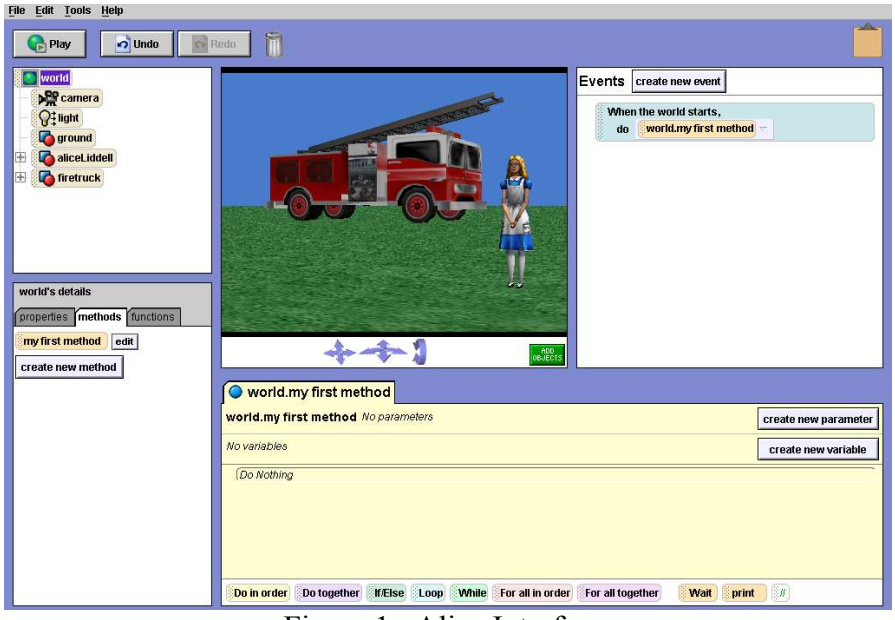

Figure 1: Alice Interface.

The interface reflects a world that contains objects. Figure 1 shows the objects in the world in the upper left object tree, the initial scene in the upper center, a list of events in this world in the upper right, a code editor in the lower right, and details about the world and its objects in the lower left. The overlapping window tabs in the world's details allow for the examination of individual properties or characteristics, dragging methods or actions into the code editor, and the use of functions to gain information about the state of an object. Examples of built-in functions include “distance to y” and “is within x meters of y” which return special information about the objects in question. [5]

Each object encapsulates its own private properties such as height, width, and location and has its own member methods such as “move” and” “turn”. Figure 1 contains an initial scene that includes a fire truck and Alice herself. Once the virtual world is initialized, the program code is created using a drag-and-drop smart editor. Using the mouse, an object is dragged into the editor where drop-down menus allow the student to select from primitive methods that send a message to the object. The words in each line of code cannot be edited and only syntactically correct dragging and dropping can occur. Hence, there are no syntax errors; students always produce programs that run without compile errors. Students can also write their own user-defined methods and functions, which are automatically added to the drop-down menus. [4]

ALICE LANGUAGE CONSTRUCTS

Decisions

Decisions are supported in Alice. For example in Figure 2, if the distance between Alice and the fire truck is less than 25 meters, the student sees Alice turn right 2 revolutions. Otherwise, Alice will jump up and down.

This example illustrates some important aspects of using Alice. The actual code produced is done automatically rather than worrying about the particulars of punctuation. The net result is a gain from this reduction in complexity. Students are able to focus on the if-else construct itself and the terminology, rather than dealing with the frustration of parentheses, commas, and semicolons. It also should be noted that an option is available in Alice to display the code in Java form, so the transition to Java syntax at a later date will not be completely new.

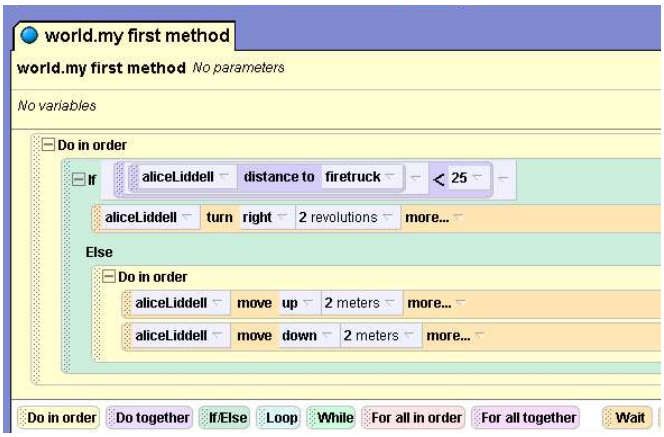

Figure 2: If-Else construct in Alice.

Looping

The possibility exists for constructing the traditional *while* loop or a loop mechanism similar to a *for* loop. In this example (Figure 3a), Alice will move toward the fire truck as long as she is not within 10 meters of it. Figure 3b shows the layout of a *for* loop in Alice.

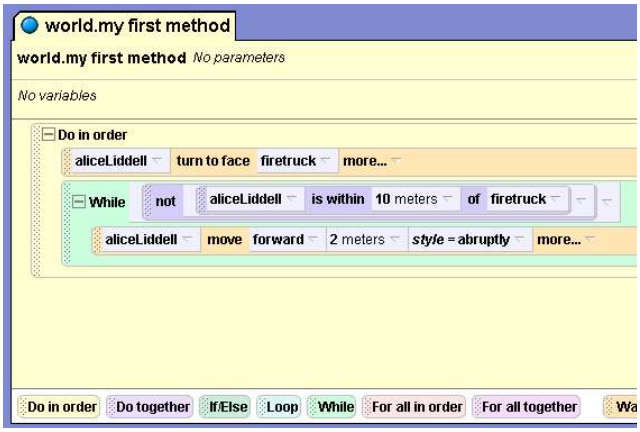

Figure 3a: While loop in Alice.

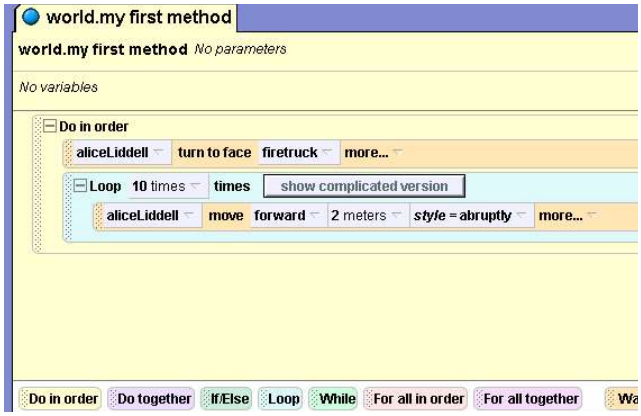

Figure 3b: Alice's for loop.

Processing

Alice can execute instructions in sequential order, with a *Do in order* block, and concurrently with a *Do together* block. The default is a *Do in order* so all of the examples shown involving a *Do in order* could have gone without the construct.

Methods and parameters

Figure 4 shows 2 methods (*myFirstMethod* and *speaks*), the method calls and the parameters. The method *myFirstMethod* is a method similar to the *main* method in Java. It is automatically created by Alice and is also the method that will be called when the program is run. Also note that the user-defined method *speaks* is now part of the other methods that belong to the Alice object as shown on the left of Figure 4. This modified object could be saved with its method *speaks* to be inherited by another world.

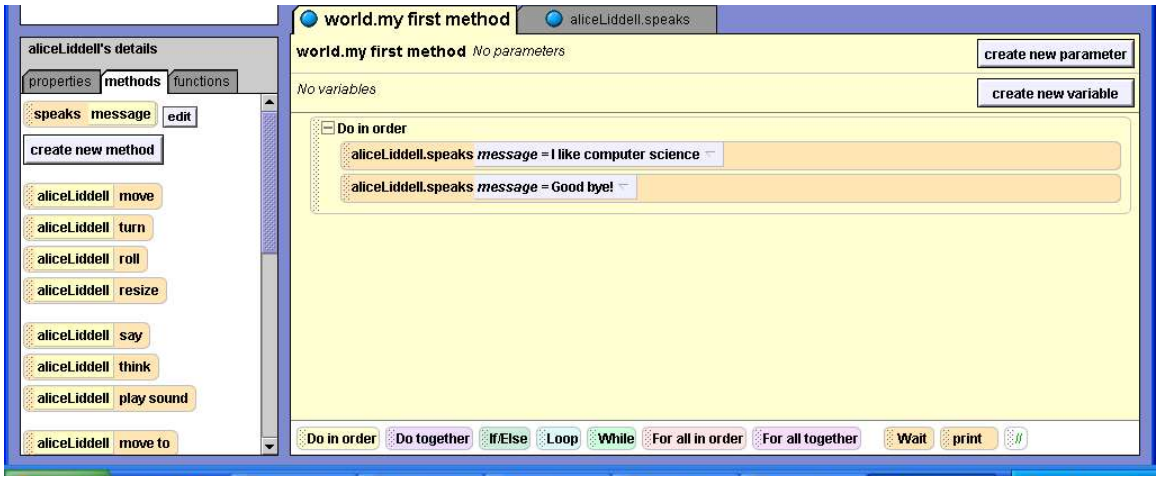

Figure 4: Method example in Alice (*myFirstMethod* and *speaks*).

COURSE PLAN

It was the author's goal to experiment with this idea of common ground. The intent was to use Alice for the first 3 weeks of a CS 1 course. The expectation was that this exposure would strengthen the students' transition to learning Java.

It was the thinking of this author that using Alice would make it easier to learn Java in some aspects because of the ideas the students would have about key terms like *object*, *class* and *method*. With Alice the students would experience practical hands-on building of their own worlds which included placing and coordinating objects. Since every 3D graphic in Alice is an object, the idea of *object* and *class* would be an easy concept to build up and reinforce.

By using Alice, the students would be gently informed about the world of programming and fairly insured of success in creating their own programs; therefore the confidence level of the students would be enhanced. The students would also gain in familiarity, so the transition to Java would not be a completely new experience. The use of Alice's visualization of the world also would be very helpful with the development of a student's understanding with regard to what their program is actually doing.

Finally, students would be exposed to common programming constructs. Using Alice to lay the groundwork for the syntax that the students will see again later in the course when using Java would enable a better understanding of the fundamental constructs used.

OBSERVATIONS

The course plan has been carried out for only 1 semester so the conclusions are purely speculative. Given the course plan and rationale for trying Alice in CS 1, the students seemed to appreciate the intent of using Alice. An informal end-of-the-course survey inquiring about the benefits of Alice with regard to learning to program in Java showed that most students, whether they did well in the course or not, thought Alice helped contextualize classes and objects.

The students thought their experience with Alice helped them better understand loops and conditionals in Java. Given the introduction with Alice it was easy for the students to understand and use nested conditionals and nested loops. The use of Alice's visualization tools to "see" what the program is doing was also a help to the student.

Students were positive about their brief introduction to methods and parameter passing in Alice. Discussing methods in Alice helped sell the idea that solving a problem by breaking it up into smaller parts is a good thing. The students found making a method better by using parameters to allow for communication between the calling statement and the method being called made good, practical sense to the student.

As mentioned by Cooper, Dann and Pausch in discussing the merits of Alice: "A strength of our approach is also a weakness." [4] Students do not develop that good old-fashioned, detailed sense of syntax. The experience of working through compiler errors looking for a missing semi-colon or the misspelling of a key word is a lost opportunity.

Another negative is similar to other problems with software packages in that the learning curve for manipulating Alice and methods is not a daunting task, but it does take time. A student may have to spend more time than 3 weeks in the use of Alice in order to become very proficient with the product.

Finally, student retention was not improved from this 1 semester experiment. Students still seemed to have the same types of experiences as in past years regarding programming. It was the author's perception at times during the semester that using Alice was helpful but also distracting. The distraction may be a perception that is caused by the author's years of experience in teaching Java in a more traditional way. But the inclusion of Alice seemed to give the CS 1 course a softer side with less rigor and intensity.

CONCLUSIONS

The author believes the experiment was a good one and worth trying again, primarily for the reason of this quote from Bennedsen and Caspersen: The idea of revealing the programming process is not new. Anyone with a reasonable intelligence and some grasp of basic logical and mathematical concepts can learn to program; what is required is a way to demystify the programming process and help students to understand it, analyze their work, and most importantly gain the confidence in themselves that will allow them to learn the skills they need to become proficient. [2]

When the CS 1 course is offered next, the author will continue to use Alice at the beginning of the course, although possibly for a shorter period of time.

REFERENCES

[1] Alice v. 2.0, www.alice.org, retrieved March 8, 2006.

[2] Bennedsen, J., Caspersen, M., Revealing the programming process, *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, 37, (1), 176-190, 2004.

[3] Carlisle, M., Wilson, T., Humphries, J., Hadfield, S., RAPTOR: A visual programming environment for teaching algorithmic problem solving, *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, 37, (1), 176-180, 2004.

[4] Cooper, S., Dann, W., Pausch, R., Teaching objects-first in introductory computer science, *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, 35, (1), 191-195, 2003.

[5] Cooper, S., Dann, W., Pausch, R., Alice: A 3-D tool for introductory programming concepts, *Consortium for Computing in Small Colleges*, 15, (5), 108-117.

[6] Dann, W., Cooper, S., Pausch, R., *Learning to Program with Alice*, Upper Saddle River, NJ: Pearson Prentice Hall, 2006.

[7] Gross, P., Powers, K., Evaluating assessments of novice programming environments, *Proceedings of the 2005 International Workshop on Computing Education*, 99-110, 2005.

[8] More than fun and games: New computer science courses attract students with educational games, www.microsoft.com/presspass/features/2005/sep05/09-12CSGames.mspx, September 12, 2005

[9] Ramalingam, V., LaBelle, D., Wiedenbeck, S., Self-efficacy and mental models in

learning to program, *Proceedings of the 9th ITiCSE Conference on Innovation and Technology in Computer Science Education*, 36, (3), 171-175, 2004.