Developing Algorithmic Thinking With Alice

Stephen Cooper
Computer Science Dept., Saint Joseph's University
Philadelphia, PA 19131 USA

Wanda Dann
Computer Science Dept., Ithaca College
Ithaca, NY 14850 USA

and

Randy Pausch
Computer Science Dept., Carnegie Mellon University
Pittsburgh, PA 15213 USA

Abstract

Rapid change in information technology motivates a corresponding evolution in our definition of computer literacy. One recent movement is toward Fluency with Information Technology, a key-concepts approach to computer literacy that includes algorithmic thinking. Algorithmic thinking is used to describe one methodology for solving problems. We introduce Alice, a 3-dimensional animation tool. Alice is an emerging technology that provides a learning environment that may be helpful in developing algorithmic thinking. We present our instructional experience with Alice and demonstrate a possible use of Alice to support the development of algorithmic thinking.

Keywords: Algorithmic thinking, emerging technology, problem solving, Alice

1. INTRODUCTION

The rapid pace at which information technology evolves is often a catalyst for change in our definition of computer literacy. In consideration of this fact, the influential National Research Council's (NRC) Committee on Information Technology Literacy recently called for an educational focus on Fluency with Information Technology (FITness).(Fluent 1999) The concept of FITness goes well beyond the conventional curricula for computer literacy. FITness is advised as a lifelong learning process for all Americans and thus spans all levels of academic institutions. FITness demands a basic understanding of fundamental concepts of information technology and communication as well as skill in problem solving. While the NRC claims FITness is a universal need, FITness concepts are of particular importance to Information Systems students. Information technology and communication concepts and skills form a foundation for later mastery of the Information Systems curriculum. The NRC committee recommends a pedagogical approach that incorporates ten key information technology concepts. In this paper, we concentrate on one of the primary concepts of importance to Information Systems educators: algorithmic thinking.

A problem often faced by information systems educators is that many students entering their first programming courses are not prepared to think algorithmically. In their high school and previous college studies, students are generally exposed to numeric computation. However, they often are not sufficiently able to develop a formal step-wise algorithm for solving a given problem. Skills have not been adequately gained in areas such as word problem solving in mathematics and the formal specification of a problem. Students may not have learned how to 1) state a problem clearly, 2) break the problem down into a number of well-defined smaller problems, and 3) devise a step-by-step solution to solve each of the sub-tasks. These three steps are critical for developing programs in both imperative and object-oriented languages. These skills are crucial to ensure the student's success in a first programming class. In this paper, we present a software tool that might be used by information systems educators to provide a learning environment for developing algorithmic thinking. We present preliminary observations from a pilot program that is still in its early stages. The working hypothesis of the pilot project is that the software may be a useful tool for supporting the learner in acquiring skills in problem solving and algorithmic thinking. While our early data is promising, a definitive analysis will take time. A follow-up to this paper will present the quantitative results of pre and post tests (e.g., SAT-9 and PSSA exam scores), interviews, and other measurements.

What is Algorithmic Thinking?
Algorithmic thinking is considered to be one of the key information technology concepts that enable people to become FIT. The members of the NRC committee define algorithmic thinking. They state that the "… general concepts of algorithmic thinking, [include] functional decomposition, repetition (iteration and/or recursion), basic data organizations (record, array, list), generalization and parameterization, algorithm vs. program, top-down design, and refinement. Note also that some types of algorithmic thinking do not necessarily require the use or understanding of sophisticated mathematics." (Fluent 1999)

Previous Work
This paper presents Alice as a tool for supporting the development of algorithmic learning for the beginning programmer. The use of animation as a tool to assist students with learning to program is not a new idea (Brown 1988, Naps 1986, Shu 1988, Stasko 1998). For example, Shu (Shu 1988) (a researcher in visualization) considers programming to require both parts of the brain, and focuses on the need to involve the artistic half – expressing the need to involve pictures in the process. van Dam, (van Dam 2000) another strong proponent of visualization and visual tools in learning to program, points out that "60% of our neurons are located in the visual cortex."(van Dam 2000) Attempts in algorithm animation, e.g. XTANGO (Stasko 1992) and BALSA, (Brown 1988) have been developed with the idea of incorporating visualization into the learning process. Most of these attempts have been targeted at CS1 students and at students in higher levels.

One of the most successful program simulation software packages, used for program visualization, has been *Karel, The Robot* (Pattis 1981). The Karel software has been used as a gentle introduction to programming at both high school and college levels for many years. Karel is well-known for setting the stage, preparing students for success in learning Pascal and in learning to solve problems. Karel is a programmable software robot. The robot executes a sequence of commands that the user has written as part of a program, moving about on a two-dimensional grid. As students attempt to have Karel implement their algorithms, they see Karel carry out their instructions on screen. There is a direct, and easily observable, link between each instruction of the student's algorithm and what the Karel robot does during program execution. We believe that Alice can be used to follow Karel's tradition with a 3-D, animated environment where students can create their own virtual worlds.

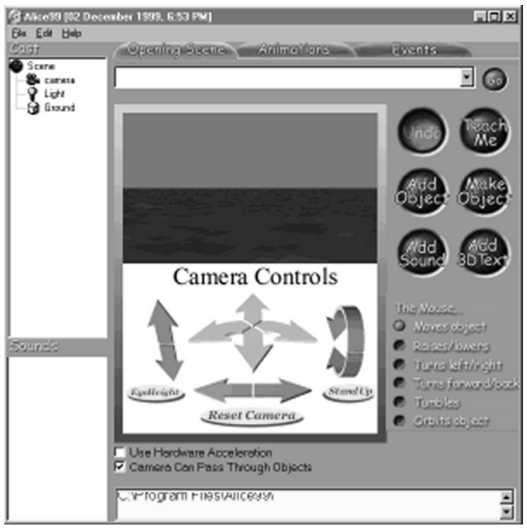
Instructional Experience
We used Alice as an instructional tool for two summer sessions at Ithaca College in a special Summer College program. The goal of the course was to provide an opportunity for students to learn the fundamental concepts of problem solving. Our observations of students working with Alice are the basis of viewpoints presented in this paper. The authors' textbook for programming in Alice (a draft copy may be found at *www.ithaca.edu/wpdann/alice1298*) is a reflection of experiences gained from working with these students. The true success of our approach will not be fully known until these students enter college and their performance in other problem solving classes (such as in mathematics and in computer science) can be compared with comparable students who did not work with Alice. A first course in visual programming using Alice will be offered in this upcoming fall semester at Ithaca College for full time college students. Discussions have begun about doing the same at Saint Joseph's University.

What is Alice?
Alice (*www.alice.org*) is a 3D animation tool that provides for the creation of interactive, animated 3D worlds. Alice is a 3D Interactive Graphics Programming Environment built by the Stage 3 Research Group at Carnegie Mellon University under the direction of Randy Pausch (Pausch 1995). The Alice application is freely available and may be downloaded from the Alice web site. The goal of the Alice project is to make it easy to develop interesting 3D environments and to explore the new medium of interactive 3D graphics.

Alice is primarily a scripting and prototyping environment that allows the user to build virtual worlds and write simple programs to animate objects (e.g., animals and vehicles) in those worlds. Objects in Alice can move, spin, change color, make sounds, react to the mouse and keyboard, and more. By writing simple scripts, Alice users can control object appearance and behavior. During script execution, objects may respond to user input via mouse and keyboard. Each action is animated smoothly over a specified duration. In the rest of this paper, we will describe the Alice programming environment, and then propose how Alice may also be used to assist in supporting the development of algorithmic thinking for novice programmers.

2. CREATING ANIMATIONS IN ALICE

Populating a Virtual World
An Alice animation begins with an opening scene, created by populating a virtual world with objects. Hundreds of pre-made 3D objects are available for selection from the Alice archives. A 3D object can also be constructed using third party commercial tools. The latest version of Alice incorporates a 3D modeling program, Teddy, built by Takeo Igarashi (Igarashi 1999) at the University of Tokyo. Teddy allows the programmer to create primitive 3D objects in a manner as easy as using a 2D paint and drawing application.



☐ Use Hardware Acceleration
☑ Camera Can Pass Through Objects
C:\Program Files\Alice99\

Figure 1. Alice Interface

Animation
Once the opening scene is set up, the next step is to plan and write a program for animating interactions between the objects and each other and between the objects and the virtual world in which they reside. Alice defines an assortment of built-in actions. In general, actions can be subdivided into two categories: those that tell an object to perform a motion and those that change the physical nature of an object. Motion actions include moving objects within the world (e.g. **Move**), rotating them about their 3D axes (e.g. **Turn** and **Roll**), and pointing at other objects (e.g. **PointAt**). Commands that change the physical nature of objects include object destruction (**Destroy**), dynamic object creation (e.g., **AddObject**), object resizing (**Resize**), and making objects visible/invisible (e.g. **Hide** and **Show**). While it is beyond the scope of this paper to detail Alice's actions (the actions listed above are a subset), we will illustrate some of the actions in the next section on developing algorithmic thinking with Alice.

3. ALICE AS A TOOL FOR ALGORITHMIC THINKING

As an illustration of using Alice to support the development of algorithmic thinking, the following is a sample lesson we have used with students in our classes. Of note is the fact that this problem is used early in the course and we use some trial and error as part of the process of learning how to develop an algorithm. Our first step is to clearly state the problem.

The problem: Imagine a world that looks something like the one shown in Figure 2. In this animation, we want the snowman to move forward to the stool. We try several ideas in thinking through a solution to this problem. It is important to help the student understand that part of the problem solving process involves coming up with an idea and trying it out. We are not discouraged with failures…that is part of the learning process. Once we have solved part of the problem, we may get an idea of how to go about solving more of it. Over time, our ideas gradually evolve to more complex, more effective, methods of solving the problem.

The solution - approach 1: Since we do not know how far away the snowman is, we can issue several `snowman.Move(forward)` commands. The snowman eventually runs into, and then through, the stool, say after **5** `snowman.Move(forward)` commands. The results are displayed in Figure 3.

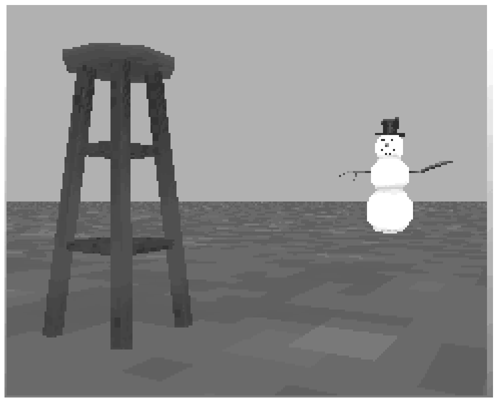


Figure 2. Initial Scene for Sample Lesson

The snowman moves right through the stool, as if the stool weren't even there! Having one figure move through another and appear on the other side with no damage is not considered good animation technique -- unless some special effect is desired in a ghost story.

So, what can be done? Well, the student can observe the animation and note that the snowman should have stopped after 4 moves. Then, a program can be written as follows:

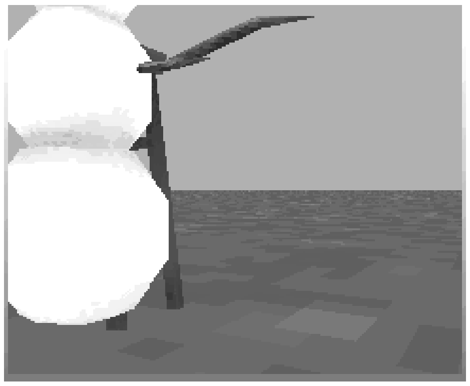


Figure 3. Snowman Intersects Stool

```
snowmove =
snowman.Move(forward, 1)
loop(snowmove, 4)
```

Running the program then produces the output shown in Figure 4.

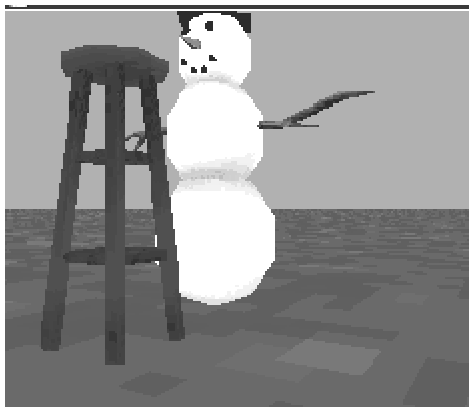


Figure 4. Snowman Meets Stool

As the students work with this problem, they realize that others in the classroom are setting their loops to different numbers of repetitions. The obvious reason is that the distance the snowman is from the stool in the opening scene effects the number of times the move must be repeated. So, we consider what if the snowman is only two units away from the stool? Or what if it is 27 units away?

The problem is refined: No matter how far the snowman is from the stool, we want the snowman to move to the stool and stop before intersecting the stool.

The solution - approach 2: At this point, Alice's *DistanceTo* command is discussed in terms of returning the distance (how far away) one object is from another,. An algorithm is planned:

1. Issue the DistanceTo command to find out the distance.
2. Loop the appropriate number of times to move the snowman close to the stool.

The program code is entered and tested with the snowman in various positions, facing the stool.

```
snowmove =
snowman.Move(Forward, 1)
loop(snowmove,
snowman.DistanceTo(stool) - 1)
```

We set up the loop so that it executes `snowman.DistanceTo(stool)` **`-1`** times, because if we looped `snowman.DistanceTo(stool)` times, the snowman would run into the stool!

4. OBSERVATIONS

As stated in section 1, a definitive study is underway. From our preliminary experimentation using Alice with Novice programmers, however, we have a few observations. In particular, Alice has several features that make it suitable for new programmers.

Immediate feedback: Students are immediately able to see how their animated programs run. Most programming languages either require users to first compile their programs, and/or require them to issue some sort of output statement to see the results of their programs. The highly visual feedback allows the student to relate the program "piece" to the animation action. We believe the immediate feedback may lead to an understanding of the actual functioning of different programming language constructs.

Enjoyment: The students we have observed found Alice extremely fun to use. They enjoyed building 3D animated worlds. They tended to spend significantly more time than was expected (as part of the assignments) in their efforts to build "realistic" animated sequences. For example, in order to make a bunny hop it is necessary to sequentially and simultaneously move several different parts of the bunny's body. The more time students spend in the animation, the more realistic the animation becomes.

State: It is, in general, not necessary to use traditional variables when writing programs in Alice. The only objects (variables) are those objects with which the students populate their worlds. Students can immediately "see" the results of changing their objects as the objects move around and interact with each other in their worlds. Additionally, eliminating the use of variables allows the students to spend more time focussing on their comprehension of various programming constructs.

Collaborative learning: We have found that students work well within small groups as they attempt to design and implement their animated worlds. Making a motion appear realistic often requires input from several students, and the projects developed in small teams have been significantly stronger than those developed individually. Students take pride in their work and send copies of Alice worlds to friends via email, or post them on the web.

Natural language: The Alice language is fairly similar to English. All instructions have an object (noun), an action (verb), and optional parameters (the adjectives and adverbs). An instruction such as **Bunny.Move(Forward, 1, Duration=3)** is fairly clear as to its intention. Namely, that the Bunny should move a distance of one unit in the forward direction, and that the movement should take three seconds to complete.

Objects: Alice has a strong *object-oriented flavor*. We did not teach students about object-oriented programming or its concepts, per se. However, students are exposed to such concepts as information hiding (they do not know how the bunny is moved forward one unit, just that it occurs as a result of their action statement). They also see a form of inheritance (by making a bunny a child of a horse, moving the horse results in the bunny moving as well). Perhaps most significantly, objects and the actions (methods) performed on them are the key component to creating animations!

5. CONCLUSION

Alice is an easy to learn and use 3D development environment that allows the user to build virtual worlds and animate objects within those worlds. We believe that building animations with Alice provides a natural set of problems to solve and an environment that supports teaching and developing algorithmic thinking in solving those problems. It is too early to draw conclusions on any carryover effect from using Alice with novice programmers. Additional work and follow-up data collection and analysis are on-going.

6. ACKNOWLEDGEMENT

Alice and the Stage3 Research Group are sponsored by DARPA, NSF, Intel, Chevron, Advanced Network & Services, Inc., Microsoft Research, PIXAR, and NASA. The Stage 3 Research Group includes Steve Audia, Dennis Cosgrove, Adam Fass, Andrew Faulring, Cliff Forlines, Caitlin Kelleher, Shawn Lawson, Daniel Maynes-Aminzade, Jeff Pierce, Dan Moskowitz, Jason Pratt, Dave Stern-Gottfried, and Desney Tan.

7. REFERENCES

Brown, M.H. 1988, *Algorithm Visualization*. Cambridge, MA: M.I.T. Press.

June 1999. *Fluent With Information Technology by the National Research Council*., National Academy Press. Also available electronically at: http://www.nap.edu/html/beingfluent/

Igarashi, T., Matsuoka, S., and Tanaka, H. August 1999, "Teddy: a sketching interface for 3D freeform design." *Proceedings of the SIGGRAPH 1999 Annual Conference on Computer Graphics*, Los Angeles, Ca., 409 - 416.

Naps, T.L. Chair, June 1996, "Working Group on Visualization. An Overview of Visualization: its Use and Design". in *Proceedings of the Conference on Integrating Technology into Computer Science Education*. Barcelona, Spain, 192-200.

Pattis, R. 1981, *Karel the Robot*. New York: John Wiley & Sons.

Pausch, R., Burnette, T., Capeheart, A.C., Conway, M., Cosgrove, D., DeLine, R., Durbin, J., Gossweiler, R., Koga, S., White, J. May 1995, "Alice: Rapid Prototyping System for Virtual Reality". *IEEE Computer Graphics and Applications*.

Rodger, S.H. June 1996, "Integrating Animations Into Courses". in *Proceedings of the Conference on Integrating Technology into Computer Science Education*, Barcelona, Spain, 72-74.

Shu, N.C., 1988, *Visual Programming*. New York: Van Nostrand Reinhold Co.

Stasko, J.T., Dominque, J., Brown, M. and Price, B., eds. 1998, *Software Visualization, Programming as a Multimedia Experience*. Cambridge:MIT Press.

Stasko, J.T. 1992, "Animating Algorithms with XTANGO". *SIGACT News,* 23, 67-71.

vanDam, Andries. March 8-12, 2000, *Exploratories: From Algorithm Animations and Interactive Illustrations to Explorable Microworlds*. Keynote Address. SIGCSE Technical Symposium, Austin, Texas.