GRAPHICAL PROGRAMMING: A VEHICLE FOR TEACHING COMPUTER PROBLEM SOLVING

Brian T. Westphal[1], Frederick C. Harris, Jr.[2], and M. Sami Fadali[3]

Abstract *- Translating from a problem description given in a natural language to a solution expressed in a programming language requires many complex steps. Though many of these steps can be done mentally for simple problems, the process itself is important when dealing with complicated software. Expressing the process demonstrates not only the complexity of solving a particular problem but also the inherent difficulties in forcing beginners to jump from a problem description to a solution. Our experiences show that using LabVIEW and Alice as graphical foundations, with several carefully designed examples, may help students more quickly learn the process involved in computer-based problem solving than they would with traditional techniques.*

Index Terms – graphical programming, problem solving, CS1

INTRODUCTION

Introductory programming courses are typically designed to teach programming using a particular programming language. Assuming that some process skills will be picked up along the way, many instructors emphasize objects-first or syntax-first approaches in their instruction. These methods are good in that they require students to hit the ground running forcing them rapidly to start coding solutions to simple problems. However, with introductory course retention rates not as high as we would like in our department, it is clear that such an approach is overwhelming for many students.

It is generally not until much later that students begin to formally learn about the process of software design. In fact, capstone courses (such as Software Engineering, which leads into a Senior Projects course) tend to provide the only real emphasis on processes for software design and organization in an undergraduate computer science student's training. Currently at the University of Nevada, all College of Engineering students (including computer science, electrical, mechanical, and civil engineering students) are required to take the first level of introductory computer science.

For many of these students, dealing with the syntax and details of a programming language is a major obstacle to learning computer problem solving. Many or most of the students who take the introductory computer science course are not going to use the language used in the course in their future work but will instead program using software packages such as MATLAB™ or MAPLE™. For these students, teaching the process of solving problems using a computer is far more important than teaching a specific programming language. Hence, it is essential to teach students to transition through levels of abstraction to reach programming as the end result rather than teaching programming languages and syntax.

The development of graphical programming systems makes the teaching of the problem solving process even more important. In 1963 Ivan Sutherland developed Sketchpad [12], the first computer graphics application, and a new world of possibilities was opened to computer programming. It took twelve years for the next significant breakthrough in graphical programming. Pygmalion [11], as developed by David Smith, was the first icon-based programming system. This was the first system that started taking the shape of modern graphical programming systems. From 1975 to the present, work has been done in developing graphical programming systems.

Sequentially, languages such as ARK, VIPR, Prograph, Forms/3, and Cube [4] each demonstrated different possibilities for graphical programming languages Discussions of these and many more can be found in the Visual Programming Language Bibliography [14]. In the parallel and distributed programming arena there are several graphical programming tools that have been developed to help advance the programming capabilities of those learning the field. These range from development systems such as Code from the University of Texas [7], Pablo from the University of Illinois [9], to systems such as Paralex [2], Grade [6], and Trapper [10]. Many of these systems used similar iconic designs, and others incorporated graphs and connection-based constructs, leading the way for LabVIEW [3].

Developed in the late 1980s, National Instruments' Laboratory Virtual Instrument Engineering Workbench (LabVIEW) was designed to aid the development of instruments that could be run in software, rather than in expensive hardware. In this way, the virtual instrument was born, allowing scientists and engineers to develop solutions and products quickly using software and a new graphical programming language – G [3].

At the University of Nevada, Reno we are experimenting with the use of several graphical programming systems that teach beginners the process of solving a problem from beginning to end. In particular, we

[1] Brian T. Westphal, University of Nevada, Reno, Department of Computer Science, Reno, NV 89557, westphal@cs.unr.edu
[2] Frederick C. Harris, Jr., University of Nevada, Reno, Department of Computer Science, Reno, NV 89557, fredh@cs.unr.edu
[3] M. Sami Fadali, University of Nevada, Reno, Department of Electrical Engineering, Reno, NV 89557, fadali@ieee.org

0-7803-7444-4/03/$17.00 © 2003 IEEE **November 5-8, 2003 Boulder, CO**
33rd ASEE/IEEE Frontiers in Education Conference

are using LabVIEW as a means of demonstrating this process and thereby reducing the overall level of abstraction needed in solving problems using the computer. We are also using Carnegie Mellon University's Alice software, to help students transition into using a general-purpose programming language.

The remainder of this paper is outlined as follows: Levels of Abstraction in teaching are discussed first. An overview of LabVIEW and Alice, along with some notes on using them in a classroom comes next. An overview of our study is presented next followed by our Conclusions and Future Work.

LEVELS OF ABSTRACTION

In his book, Tremblay says that in terms of current knowledge and technique, there are at least five levels of abstraction involved in "expressing algorithms"[13]. These are:

- natural languages,
- diagrams,
- flowcharts,
- algorithmic languages (pseudo code), and
- programming languages.

These levels have changed little in the past several decades.

Using only one or two of these tools for anything more than a trivial example often leads to confusion among students. Consider the following. Students whose native language is that of the instructor will have the greatest chances of understanding the basic parts of a particular lesson. Those that do not share the language may miss important details, especially in subjects like computer science where the language is somewhat beyond common vocabulary. Thus, reinforcing with diagrams will help to fill in the gaps created by language barriers. Continuing, each level of abstraction that one can add on top of natural language will help clarify both details and principals for all students.

Many instructors make use of a few of the levels of abstraction by incorporating an amount of pseudo code into their teaching. Although this is helpful to some students, it is likely confusing to many others. Students can easily become misled by the format of pseudo code, trying to use similarly structured loose syntax in their later programming assignments. There are several reasons for this. First, without use of diagrams or flowcharts, it is difficult for beginners, even with pseudo code, to communicate the flow of a program. For example, the next instruction to follow may not always be obvious from reading pseudo code. Second, without a system for verifying the correctness of a student's pseudo code, a student's difficulties with correctly expressing algorithms may go unnoticed. This leads to a trend of students completing their assignments, without necessarily caring about the correctness of their solutions. These students are often frustrated as they realize later in the course that being able to verify the correctness of a solution is as important as solving the problem.

USING LABVIEW FOR DEMONSTRATING AND VERIFYING DIAGRAMS AND FLOWCHARTS

Designed with scientists and engineers in mind, LabVIEW[3] provides a software development package for graphically constructing virtual instruments (which may communicate with physical instruments in many cases). Students and professionals alike can use it as a tool to create on-screen oscilloscopes, pressure gauges, potentiometers, and many other simulations or displays for virtual or physical devices. By making use of a diagram and flowchart-like interface, components can be placed together in a black box fashion, demonstrating some degree of program flow and interconnection through virtual wires.

The diagramming and flowcharting features of LabVIEW combined with the ability for students and instructors to run (and therefore verify) their solutions, provide an excellent foundation for developing the ideas of diagramming and flowcharting as intermediate steps of solving problems with computers. By providing a form of instant feedback, students have an obligation to correct problems discovered in their algorithms as they work. A screenshot of LabVIEW can be seen in Figure 1.

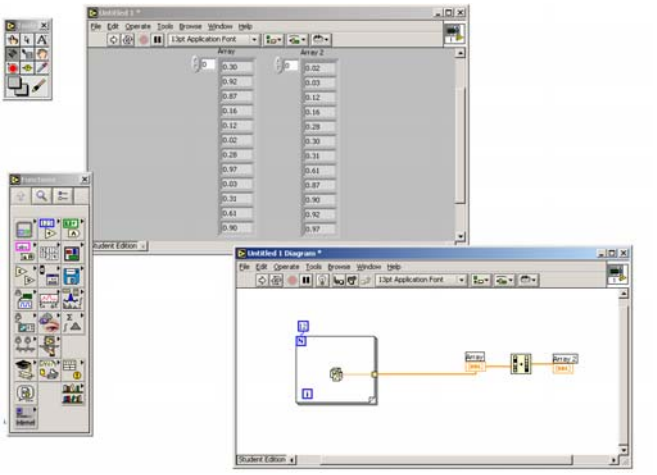

FIGURE 1
SCREENSHOT OF LABVIEW SOFTWARE

Using LabVIEW as a tool, one can demonstrate the primary concepts of solving problems using a computer, allowing students to transition into using variables, arrays, loops, and other constructs common to programming languages. As students become familiar with these concepts in the graphical world that LabVIEW offers, the instructor can begin writing ideas using pseudo code. They can start with pseudo code that is highly similar to natural language

and work into pseudo code that looks more like a programming language (with less explanation for each instruction or idea).

In our experiences, only a moderate amount of time was needed to introduce and familiarize a group of students with LabVIEW, a partial-vocabulary of computing, and several of the major constructs present in introductory computer programming courses (variables, arrays, loops, and conditional expressions). After a ten to twenty-minute tutorial, students were able to use the essential parts of the software with a moderate level of proficiency.

As a major part of the research for this paper, we set out to observe interactions that might take place in a sample lesson. Using LabVIEW as a diagramming tool, we chose to solve the problem of sorting a list of 20 random numbers. LabVIEW makes the diagramming process for this straightforward.

On the highest level of diagramming, students can easily create a system that will:

1. initialize a list of twenty random numbers,
2. sort the list, and
3. display the sorted list on the screen.

Because LabVIEW has many high-level tools such as an array sorting function, it is useful for this type of high-level diagramming. This allows students to build their system and to verify its correctness before moving on with their design. However, LabVIEW does not allow the instructor to introduce students to pseudo code. In the next section we show how this can be accomplished using Alice.

USING ALICE FOR DEMONSTRATING AND VERIFYING PSEUDO CODE

By expanding on the concrete aspects of the graphical interface of LabVIEW, students are able to continue their transition as the instructor introduces pseudo code. Because pseudo code is inherently flexible in its structure, many students find it easier to work with at first than a programming language. This is primarily because students, before gaining some experience, find it difficult to express algorithms with sufficient detail. For instance, a student has the ability to say that a list should be sorted but cannot always describe the sorting process.

Designed and implemented at Carnegie Mellon University to teach an objects-first approach to computer programming, Alice [1] allows programmers to use a pseudo code-like language to build rich three-dimensional worlds, exploring the excitement of both computer graphics and computer programming. The language used in Alice is not as loosely structured as typical pseudo code. In fact, the language is specific enough that the code can be interpreted and run, providing a level of instant feedback.

Because Alice makes use of a drag-and-drop style of programming, it is impossible for students to cause syntax errors, resolving much of the frustration for beginning programmers. The language remains in-line with many of the pseudo code ideals by leaving highly descriptive instructions and structure within the code. By using Alice as an intermediate step in learning computer problem solving, an instructor is able to transition students into reading and writing source code, allowing students to focus on developing the logic and program flow rather than worrying about syntax. Alice allows building blocks to be introduced so students begin thinking about functions. The Alice software installation provides an excellent introductory tutorial that is sufficient to demonstrate the system's basic functionality. The tutorial takes approximately twenty to forty minutes to run. A screenshot of Alice is shown in Figure 2.

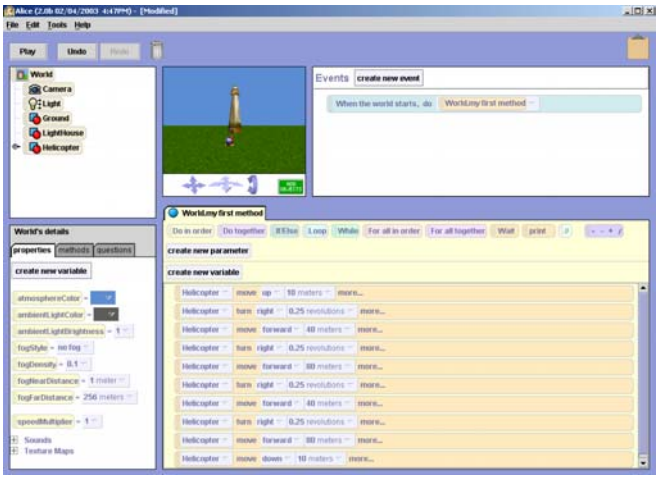

FIGURE 2
SCREENSHOT OF ALICE SOFTWARE

In the sample lesson using Alice, we asked students first to go through the tutorial (at their own pace) and then to complete a simple task: "Build a scene with a helicopter and a lighthouse. Make the helicopter take-off, fly around the lighthouse, and land near its starting point." Students worked in small groups for this assignment and completed it in less than ten minutes, most continuing to develop their scenes further. Though this was a simplistic assignment, the level of excitement allowed students to learn more than the assignment required. For instance, some students made their helicopters crash into the lighthouse and had the lighthouse tip over when it was hit. Though it may seem more like amusement, these additions involved many more complicated ideas than the original assignment did.

NOTES ON USING LABVIEW AND ALICE WHILE TEACHING

There are several less-than-ideal aspects about the LabVIEW and Alice software packages that need to be pointed out.

Using LabVIEW While Teaching

The students that had less experience with computers in general were slightly slower to pick up certain elements in the interface design. For example, the concept of connecting objects on the screen using a virtual wire could be improved by providing more direction.

In demonstrating loops, the LabVIEW interface is unclear for new users in visualizing the structure of a loop. While the overall ability to demonstrate program flow using LabVIEW is reasonable, the loop figures are ambiguous and are not readily apparent.

It is important to point out these design flaws to students, especially those that will be using programming languages in the future, because program flow is a key part of developing logical skills for students.

Using Alice While Teaching

Alice is an excellent system for demonstrating pseudo code development and verification. However, because it is based on Java, it can be noticeably sluggish at times. There are currently many bugs in the software that cause it to crash or cause erroneous video effects from time to time. Other than being frustrating in these minor ways, with up-to-date hardware (especially in way of the video card) the software runs smoothly most of the time.

DEMOGRAPHICS, SURVEY, AND INTERACTIONS

This research study was conducted with a small sample of students at the University of Nevada, Reno. Because we have had so many students enroll in our CS I courses over the years we have a good understanding of what the typical student's attitudes were, so we did not feel the need to conduct this study with a control group.

Along with a sample lesson using the tools that have been proposed above, the study also included a short pre and post-survey designed to record students' perceptions of the ideas presented in the study. Figures 3 and 4 represent the survey that was conducted both before and after the sample lesson. Following are highlights from the survey and study interactions.

Our first set of students was selected at random from the hall in the College of Engineering. This group was composed of typical students who had been in Computer Science I. We were surprised to find that the students were more enthusiastic toward working with the LabVIEW and Alice software (this is especially the case for Alice) than originally anticipated. Several of the students requested that the software be left on the systems after our sessions so that they could continue to experiment with them. On a scale from 0 to 10 (10 being the most enjoying), students rated the lesson 8.89. The instant feedback and increased use of senses (including audio in the case of Alice) may have contributed to this enjoyment rating. This level of enthusiasm, even over the short term of a single lesson, demonstrates the level of enthusiasm that students may be expected to have during a semester.

Many of the students made these or similar remarks, demonstrating their enthusiasm towards the lesson: "It opened my eyes in terms of approaching programming problems and seems like it is a much better method to teach CS I", "I like your ideas for changing the CS teaching methods", and "my interest has gone up."

The second group of students that was selected was a lab section of students currently in a Computer Science I course. These students were given the same lesson format as the first group, and the results were amazingly similar. It is significant to note that all of the students in both groups stated that this type of learning (either as a pre-cursor to a Computer Science I type course, or as a replacement) would be highly beneficial in the overall learning experience for an engineering student.

1. Age _____
2. Major _____
3. Minor _____
4. Have you taken CS201 (or equivalent) before? YES NO
5. If yes, rate the level of difficulty involved in initially jumping into programming (i.e. how difficult was it to start with), 0-10 (10 is the hardest): _____
6. If no, please give the primary reason for which you have not taken CS201.
_____
7. Have you heard, from other people, that CS201 is a difficult course? YES NO
8. Rate your level of experience with general computer use, 0-10 (10 being very experienced):
_____

The next two questions are optional but will help correlate the level of difficulty of CS201 across all levels of students.

9. **Optional**, if you have taken CS201, what grade did you receive? _____
10. **Optional**, what is your current overall GPA? _____
11. As an engineering student (if applicable), rate your abilities towards general problem solving, 0-10 (10 being an expert problem solver): _____
12. As an engineering student (if applicable), rate your abilities towards computer-based problem solving, including your abilities to find and correct problems with computer hardware and/or software, as well as to solve problems that require some level of computer programming, 0-10 (10 being an expert problem solver): _____
13. If possible (if you do not know what is meant by any of the following phrases, please skip the part of the question), rate the differences you perceive in the following, 0-10 (10 being highly different):
    a. natural language (spoken or written) vs programming language _____
    b. natural language vs diagrams _____
    c. natural language vs pseudo code _____
    d. diagrams vs pseudo code _____
    e. pseudo code vs programming language _____

FIGURE 3
SURVEY PART I – GIVEN BEFORE THE SAMPLE LESSON

1. Rate your level of enjoyment for the session today, 0-10 (10 being highly enjoyable): _____
2. Have any of your perceptions of computer science changed?
_____
3. If possible (if you do not know what is meant by any of the following phrases, please skip the part of the question), rate the differences you perceive in the following, 0-10 (10 being highly different):
    a. natural language (spoken or written) vs programming language _____
    b. natural language vs diagrams _____
    c. natural language vs pseudo code _____
    d. diagrams vs pseudo code _____
    e. pseudo code vs programming language _____
4. If you have not already taken CS201, do you feel less anxiety for taking it after this session?
_____

FIGURE 4
SURVEY PART II – GIVEN AFTER THE SAMPLE LESSON

CONCLUSIONS

LabVIEW and Alice provide excellent tools to be used as stepping-stones in teaching an introductory computer programming and/or computer-based problem-solving course. Though these systems are not perfect tools, they are powerful and exciting to use. Students enjoy using them and, as a result, may continue to learn above and beyond the requirements of the classroom and/or lab.

Instructors of beginning computer science classes need to develop a plan to allow students to transition into computer programming, by focusing on computer-based problem solving. LabVIEW provides the fundamental support for diagramming and flowcharting that an instructor can use to ensure that at each stage in the solution design process, a student is working along the right lines. Unlike the traditional methods of working with diagrams on paper or not requiring diagrams at all, LabVIEW allows students to verify their work as they continue, keeping the motivation to solve the problem correctly.

As an exciting and dynamic tool for the design of graphical worlds, Alice helps ease students into programming by providing a syntax-error-free environment in which students can work to develop and verify in-depth pseudo-code-like solutions. This helps to lead the transition into programming languages, as instructors begin to explain the differences between pseudo code and actual code.

We look forward to bringing these graphical programming tools into use at the beginning of the semester in a CS I course. Our preliminary results have shown that they should be very beneficial to enhancing the student's comprehension. These types of tools should help stimulate students that are typically uninterested in a traditional programming course due to the fact that they feel it is not essential to their job. They will also allow the students to learn how to start the problem solving that they will need on the job, no matter what their major.

ACKNOWLEDGMENT

We would like to give special thanks to the instructors and students that participated in this research study.

REFERENCES

[1] "Alice: Free, Easy, Interactive 3D Graphics for the WWW", Retrieved March 10, 2003, from http://www.alice.org/.

[2] Babaoglu, O., Alvisi, L., Amoroso, A., Davoli, R., Giachini, L. A., "Paralex: An Environment for Parallel Programming in Distributed Systems", *Proceedings of the ACM international Conference on Supercomputing*, July, 1992.

[3] Bishop, R. H., *LabVIEW Student Edition 6i*, Prentice Hall, Upper Saddle River, NJ, 2001.

[4] Boshernitsan, M., Downes, M., "Visual Programming Languages: A Survey", Retrieved March 14, 2003, from http://www.cs.berkeley.edu/~maratb/cs263/paper/paper.html

[5] Cunningham, S., Shiflet, A. B., "Computer Graphics in Undergraduate Computational Science Education", *SIGCSE Technical Symposium on Computer Science Education*, pp. 372-375, ACM Press, Reno, NV, 2003.

[6] Kacsuk, P., et al., "A Graphical Development and Debugging Environment for Parallel Programs", *Parallel Computing*, Vol. 22, pp. 1747-1770, 1997.

[7] Newton, P., Browne, J.C., "The CODE 2.0 Graphical Parallel Programming Language", *Proceedings of the ACM International Conference on Supercomputing*, July, 1992.

[8] Pausch, R., et al., "A Brief Architectural Overview of Alice, a Rapid Prototyping System for Virtual Reality", *IEEE Computer Graphics and Applications*, 1995.

[9] Reed, D. A., Aydt, R. A., Madhyastha, T. M., Noe, R. J., Shields, K.A., et al. "An overview of the Pablo performance analysis environment", Technical report, University of Illinois, Urbana, Illinois 61801, November 1992.

[10] Scheidler, C., Schafers, L., "TRAPPER: A Graphical Programming Environment for Industrial High-Performance Applications", *Proceedings of PARLE'93: Parallel Architectures and Languages Europe*, Munich, Germany, 1993.

[11] Smith, D. C., "PYGMALION: A Creative Programming Environment", *Ph.D. dissertation, Stanford University*, 1975.

[12] Sutherland, I. E., "SKETCHPAD, A Man-Machine Graphical Communication System", *Proceedings of the Spring Joint Computer Conference*, pp. 329-346, Spartan Books, Baltimore, MD, 1963.

[13] Tremblay, J. P., "Algorithms", *Introduction to Computer Science: An Algorithmic Approach*, pp. 18-23, McGraw-Hill, 1989.

[14] "Visual Programming Languages Bibliography", Retrieved March 31, 2003, from http://cs.oregonstate.edu/~burnett/vpl.html