Development and Testing of a Graphical FORTRAN Learning Tool for Novice Programmers

A. O. Ajayi, E. A. Olajubu, and D. F. Ninan
Obafemi Awolowo University, Ile-Ife, Nigeria

anuajayi@oauife.edu.ng; emmolajubu@oauife.edu.ng; jninan@oauife.edu.ng

S. A. Akinboro
Bells University of Technology, Ota, Nigeria

akinboro2002@yahoo.com

H. A. Soriyan
Obafemi Awolowo University, Ile-Ife, Nigeria

hsoriyan@oauife.edu.ng

Abstract

To address the difficulties associated with computer programming, this article first looks at some reasons why students, especially engineering students, find programming such a daunting prospect, and it proposes a programming learning tool managed by a Deterministic Finite Automaton (DFA). The DFA machine used a graphical environment provided by Simulink to teach the FORmula TRANslator (FORTRAN) programming language to science students.

The proposed programming learning tool and the traditional method of teaching were compared and evaluated. The results of evaluation indicated that there was an improvement in learning effectiveness of the proposed learning tool.

Keywords: Graphical programming, Learning environment, Text programming, Students' failure, Algorithms.

Introduction

Computer programming is an important constituent of a computer science curriculum, but many students, especially those without a programming background, find it difficult to grasp. This is attributed to reasons that include: traditional teaching methods do not adapt well to the domains of coding and problem solving (Allison, Orton, & Powell, 2002), programming concepts are difficult to learn and teach (McCracken, Almstrum, Diaz, Guzdial, Hagan, et al., 2001; Traynor & Gibson, 2004). Poor performance of students in first year programming courses has been a growing concern in many universities (Meisalo, Suhonen, Sutinen, & Torvinen, 2002; Miliszewska & Tan, 2007; Tavares, Brzezinski, Huet, Cabral, & Neri, 2001), and not excluding Obafemi Awolowo University (OAU), Nigeria, where FORTRAN language is taught to introduce programming to students in science, engineering, and other related disciplines. Observations over the years have shown that students struggle to

Editor: Eli Cohen

pass this course, and find computer programming a daunting prospect.

Information about the difficulties faced by the students in OAU was obtained from questionnaires administered to a sample of students and interviews with lecturers involved in the teaching of the course. The major problem identified has to do with the student population. An annual enrollment of over 2,000 students makes it difficult for lecturers to teach effectively and students to comprehend in not too conducive classroom environments. Some challenging topics mentioned by students include requirements of programming syntax, arrays, subprogram, comprehending abstract terms, iteration, selection, and input/output. In addition, students find it difficult to understand the mechanics of programming. These difficulties are common to many new programming students (Carbone, Hurst, Mitchell, & Gunstone, 2001; McCracken et al., 2001; Meisalo et al., 2002; Thomas, Ratcliffe, Woodbury, & Jarman, 2002). To do well in the first programming course, students must have skills of problem solving, logical reasoning, coding, and debugging (Klassen, 2006). However, students with no previous programming experience have difficulty in comprehending the large amount of information and concepts presented in introductory programming classes. Furthermore, since programs are frequently developed using text editors, this generation of students, who are well accustomed to the graphical user interface environment, will find the command prompt environment uncomfortable. Rodger (2002) has revealed that graphics and animation are effective teaching tools to get students interested in courses. Given that capturing student's attention and interest while maintaining academic rigor is a prerequisite to teaching of any course, employing these tools in computer programming courses will reduce failure rates and enhance students' passion for programming.

Many systems designed to support learning programming abound; we give a review of some in this paper. Calloni and Bagert (1993) developed a Windows-based iconic programming language, which allow users to program with icons (blocks). This system automatically generates syntactically correct code for Pascal. However, since the conversion from icons to text is automatic, it thus encourages laziness on the part of student, and it may be difficult to ascertain whether the student is able to articulate the underlying programming concepts. Moreover, since the system is designed for high school students, it may not be applicable for complex programming problems. Guzdial and Usselman (2003) conducted a study where students are taught to program using audio and visual aids with Python. This study reported a low student dropout. Goldman (2003) reported high retention of female students in programming courses utilizing 3D graphics, virtual worlds, and sound, when JPie, a tool developed in Washington University, was used to teach students in introductory programming classes. Nevertheless, JPie is an environment for software construction using Java and is not widely adapted outside Washington University (Klassen, 2006).

Carlisle, Wilson, Humphries, and Hadfield (2005) developed a flow model environment that supports program development (flowchart), where students build programs by manipulating connections between icons. However, students who had adequate problem-solving skills to phrase a solution to a problem in terms of a flowchart found it difficult turning the flowchart into a syntactically correct program (Dunican, 2002; Kölling & Rosenberg, 2001; Sheard & Hagan, 1998).

Sanders and Dorn (2003) developed Jeroo, a narrative tool, that supports programming to tell a story. The program is small and does not require much memory to run. Jeroo focused on control structures, methods, and objects only. However, for this tool to be more effective, it must be interleaved with a primary language and taught in the first 3-4 weeks of a regular semester (Klassen, 2006). Carnegie Mellon University developed Alice, a programming environment designed to expose novice students to programming, and since its introduction, it has been successfully utilized as an introductory programming tool in various universities (Klassen, 2006; Rodger, 2002; Zaccone, Cooper & Dann, 2003). Nevertheless, Alice is not good enough to provide a solid

programming concept to students. Equally, problems solved in Alice were only in the context of 3D animations and did not cover many real world application problems.

Other approaches with user integrated development environments to support learning programming include BALSAII (Brown, 1988), Xtango (Stasko, 1992), Jeliot 2000 (Levy, Ben-Ari, & Uronen, 2003), BlueJ (Kolling, Quig, Patterson, & Rosenberg, 2003), and Jhavé (Naps, 2005), visual C++ builder, NetBean, and Eclipse. These tools lacked the intuitive nature of a "point and click" interface (Klassen, 2006). Although, many computer science programs are using these environments in introductory programming courses, they still add unnecessary level of complexity for students to master (Cooper, Dann, & Pauch, 2003).

Despite many approaches designed to aid learning, programming learning problems still continued to be widely reported (Cooper et al., 2003; Miliszewska & Tan, 2007; Zaccone et al., 2003). Truly, different approaches are still necessary to make computational environments effectively used to assist students in learning programming skills. The framework presented in this study examines a situation in which the main problem confronting science and engineering students is their inability to construct algorithms and create solutions to those problems. We introduced a FORTRAN programming learning approach using Simulink (a graphical programming environment) to construct algorithms that is managed by a DFA machine. Simulink is Matlab add-on software, which enables block programming and simulation of different algorithms.

A graphical environment or iconic programming is the most convenient environment to describe algorithms visually using block diagrams (Ichikawa & Hirakawa, 1990). This form is much closer to engineers and scientists than other types. The proposed approach focuses essentially on using graphical environments where students design, run, correct algorithms, and subsequently convert such algorithms to FORTRAN programs. Graphical programming tools (Labview, Simulink, OPnet, RSoft, etc.) are increasingly used to simulate a design or model a process in engineering and technology disciplines due to their ease of implementation and because they allow a natural, intuitive interaction with the system under simulation. This study uses Simulink for two reasons. Firstly, being an industry standard, the engineering and science students will have valuable experience in a tool they will eventually use in their later employment. Secondly, many blocks from Simulink library and Toolboxes (Neural network, Fuzzy logic, Stateflow, Communications, and so on) are available for use in other areas of their engineering education.

From the literature reviewed (Guzdial & Usselman, 2003; Rodger, 2002; Sanders & Dorn, 2003), we discovered that there has not been any reported attempt at applying Simulink as teaching aids for any text-based programming language. Interest in learning computer programming is declining, and as a result a number of potential students, even those with the slightest interest in computer programming, are been discouraged, the consequence of which is high dropout from programming courses (Meisalo et al., 2002). Therefore, in accordance with Cilliers, Calitz, and Greyling (2005), it is the authors' opinion that the most effective methodology for learning programming skills lies within the bounds of graphical representations of algorithms. The uniqueness of the study is that it investigates the use of the proposed learning tool in a university setting to aid the process of teaching and learning computer programming. A comparison of the proposed learning approach with the traditional teaching approach is also made. The study will also foster the understanding of the potential of using a graphical environment to aid the learning process.

Method

The Course

Computer programming I (CSC 201) is the first programming course offered in the second year to all students from the faculties of Science, Technology, and Environmental Design and Man-

agement in OAU, Nigeria. The course involved computer programming using FORTRAN. The FORTRAN 77 compiler was the programming choice for the course before the 2000/2001 academic session, but burdened by high failure rate recorded from past years, certain drastic measures were taken to salvage the situation. For instance, the use of a different textbook with a slightly modified curriculum was employed in the 2001/2002 academic session, with Lahey FORTRAN 95 Compiler replacing the unfriendly Fortran 77 environment, which forced students to type correctly all syntactically correct statements before running their programs. The Lahey FORTRAN 95 Compiler provided a friendlier environment by supplying part of the syntactically correct statements, with students supplying the rest. This feature is meant to reduce the number of syntax errors generated from erroneous program. Also in the 2002/2003 academic session, the large class size (average of 2400 students), identified as one of the main causes of failure, was divided into 3 sub classes, each not more than 700 students. Another measure introduced was tutorial classes, manned by fourth year and postgraduate students from the department of Computer Science and Engineering. This is to serve as a source of support to students since they prefer to seek help from fellow students rather than from lecturers (Pascarella & Terenzini, 2005). All these measures did little to improve the situation. In fact, not much success was recorded in terms of reducing failure rate, and this undoubtedly, necessitated the need for a new learning approach.

Proposed Programming Learning Approach

In the proposed learning approach, DFA machine manages the process of learning FORTRAN programming using Simulink. The proposed programming learning approach involves two different activities. Firstly, the design of algorithms to problems using a visual representation in Simulink, where graphical symbols that represent the algorithms building blocks (inputs, outputs, selections, repetitions, procedures, etc) are used, and secondly, the subsequent conversion of the running algorithms to FORTRAN code.

DFA is a 5-tuple (Q,∑,T, s, F) where Q is the number of states, ∑ is the alphabet of the encoded language, T is a transition function, s is the start state, and F is a set of final or accepting states (Bongard & Lipson, 2005). Given a sentence, which is made up of string of symbols taken from the alphabet ∑, and beginning at the start state s, the first symbol is extracted from the sentence, and based on that symbol the sentence transits to a new state as indicated by the transition function T. A DFA follows the transition dictated by the current symbol, the current state, and the state transition function. After a state transition, the next symbol is then extracted from the sentence, and based on the transition function T, the sentence transits to a new state. This process stops when all the symbols in the sentence have been exhausted. If the last state visited is a member of F, then the sentence belongs to the language otherwise, it does not belong to the language. This is mathematically stated as:

Let M be a DFA such that M = (Q, Σ, T, s, F), and $X = x_0x_1 \ldots x_{n-1}$ be a string over the alphabet Σ. M accepts the string X if a sequence of states, $r_0, r_1, \ldots, r_n$, exists in Q with the following conditions:

$$r_0 = s$$
$$r_{i+1} = T(r_i, x_i), \text{ for } i = 0, \ldots, n-1$$
$$r_n \varepsilon F.$$

The first condition indicates that the machine starts in the start state s. The second condition says that given each character of string X, the machine will transit from state to state according to the transition function (T). The last condition says that the machine accepts X if the last input of X causes the machine to halt in one of the accepting states. Otherwise, the machine rejects the string.

Traditionally, compilers and other language-processing systems used DFA to recognize symbols. Other areas where DFA has demonstrated its applicability include measuring the complexity of a dynamical system (Schittenkopf, Deco, & Brauer, 1997), real-time detection of denial of service in computer attacks (Branch, Bivens, & Chan, 2002), epileptic spike recognition in electroencephalogram, speech-processing and signal-processing systems. Finally, wide variety of controllers in finite-state systems used it to track their current states (Keshri, Sinha, Hatwal, & Das, 2008; Webber, 2007).

Transition diagram is one of the notations used to describe automata. Figure 1 is a transition diagram depicting the states involved in the proposed pedagogical approach, with directed arcs showing the valid moves a student can make in the process of acquiring FORTRAN programming skills. Student transits from state to state based on the learning objective (transition function) specified. Upon the completion of the study materials in each state, students are given coursework exercises to evaluate their level of competence. The questions posed consisted of the basic concepts learned in the current state, and they varied from multiple-choice questions to written (theory and practical). The DFA confirms that learning has taken place if a student starts from the start state, and moving from state to state, each time following the arrow corresponding to the next task to be done until the final state (either state 4 or 7, shown in Figure 1) is reached, which signifies that knowledge has been acquired.

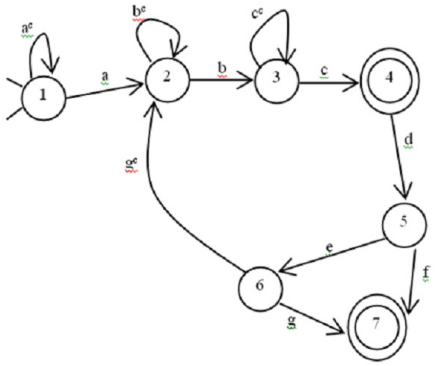

Figure 1: States in the proposed learning approach.

Legend

States	Activity	Arc	Meaning
1	Identify basic Simulink blocks	a	Student is able to identify blocks
		a^c	Student is still unable to identify blocks
2	Convert Simulink instruction to FORTRAN syntax	b	Student is able to convert Simulink statement to FORTRAN syntax
		b^c	Student is still unable to convert Simulink statement to FORTRAN statement
3	Convert Simulink code examples to FORTRAN equivalents	c	Student is able to generate a syntactically correct text equivalent of the Simulink program
		c^c	Student is still unable to generate syntactically correct text equivalent of a Simulink program
4	Compile and run converted FORTRAN program examples given in state 3	d	Present students with new programming problems
5	Present new programming problems to students	e	Student is able to create Simulink program for the problem given
6	Convert Simulink code to FORTRAN code	f	Student is able to write FORTRAN program
7	Write code in text-based notation, compile and run code	g	Student is able to generate a syntactically correct FORTRAN equivalent of the Simulink program
		g^c	Student is still unable to generate syntactically correct program

For instance, a typical coursework exercise to find the mean of three identifiers (a, b, and c) is as depicted in Figure 2, where a student is expected to connect the blocks to form a Simulink program and to arrange the texts in the Simulink Model Window to form the equivalent FORTRAN code. The purpose of this is to help students gain experience and obtain confidence in those topics. A satisfactory completion of all the activities in a state would suggest that the student has garnered sufficient experience and exposure in a given module and, therefore, should move on to the next state, otherwise the study materials in either the current state or the previous states is repeated. A brief discussion of the states follows. The first (1) state is "Identify basic Simulink blocks", where students are required to recognize and identify basic blocks needed to write Simulink programs, as well as to know their functions. Few of these blocks with names and functions are given in Figure 3. The second (2) state is "Convert Simulink instruction to FORTRAN syntax", where students learn how to convert block statement in Simulink to its equivalent in FORTRAN. For example, Figure 4 depicts some Simulink statements and their equivalents in FORTRAN. The third (3) state is "Convert Simulink code examples to FORTRAN equivalents", where students use concepts learnt in the previous two states to convert Simulink program examples to FORTRAN codes. The proposed learning approach used in this study included several program examples in Simulink, which can be used as teaching aids, from simple to more complex ones, with study materials tailored towards FORTRAN programming language. A number of such examples close to fifty (50) and drawn from available exercises in some programming textbooks (C++, FORTRAN, Pascal, etc) were incorporated into every topic, especially programming concepts such as sequence, iteration, selection, and modular programming. A student learning FORTRAN programming can perform the tasks specified in states 1 to 4, to convert the Simulink code examples to FORTRAN. The inclusion of study examples of well-written codes in Simulink follows Kölling and Rosenberg's (2001) recommendations. Figure 5 depicts a Simulink program example and its equivalent in FORTRAN. The fourth (4) state is "Compile and run the converted FORTRAN program". This state involves using a FORTRAN compiler to compile, build, and run the manually generated FORTRAN code. The FORTRAN code as generated by one of the students used in this study is as shown in Figure 6. The student copied the text instruction from the Simulink Model Window, pasted it into the programmer's editor (Notepad), and compiled it using Essential Lahey FORTRAN 95 compiler. The compilation result (command prompt window) is also shown in Figure 6. "Present new programming problems to students" is the fifth state. In this state, the students develop solutions to new programming problems, either by first creating a Simulink code, which is later converted to FORTRAN or by proceeding straightaway to state 7 to write the FORTRAN code for the problem.

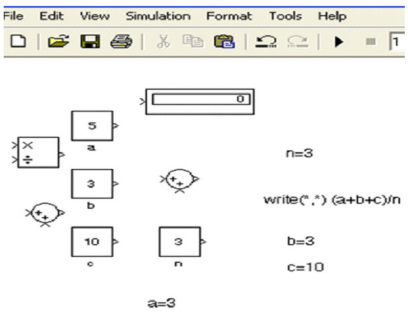

Figure 2: Sample Exercise.

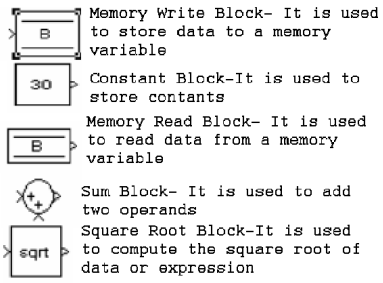

Figure 3: Some Simulink blocks.

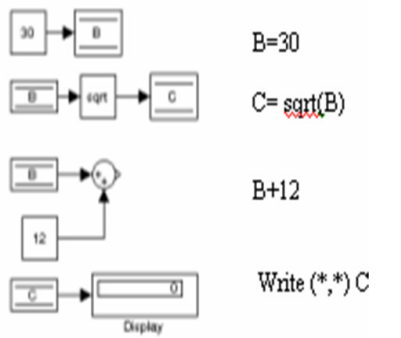

Figure 4: imulink statements with equivalents in FORTRAN

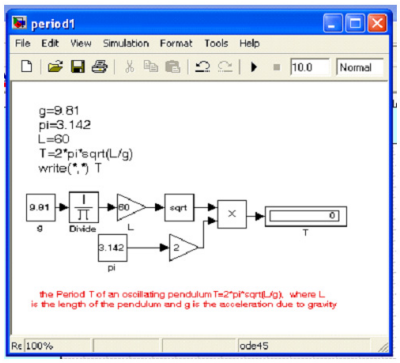

Figure 5: Simulink and FORTRAN codes

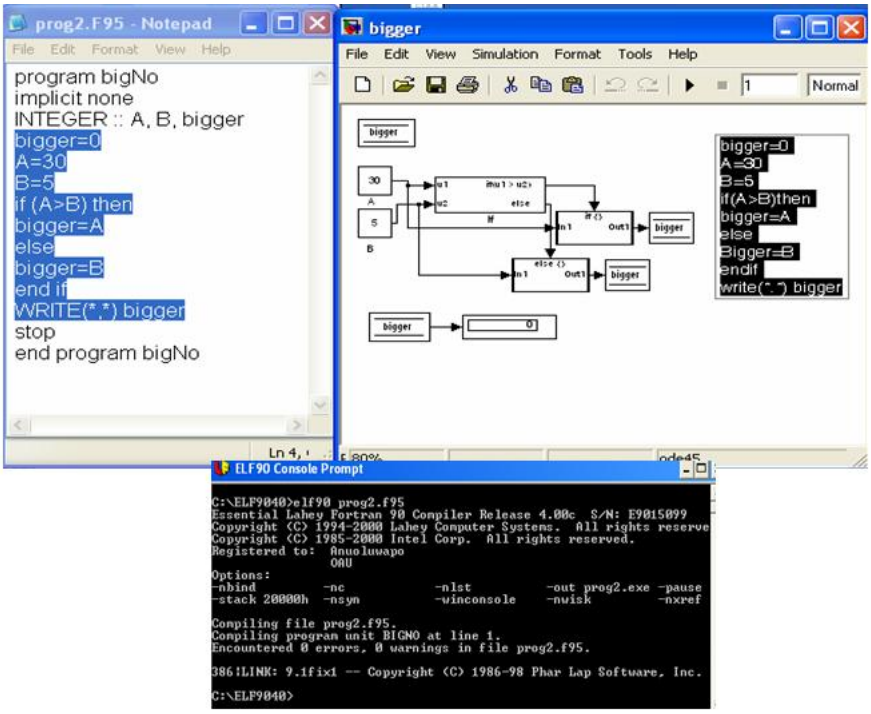

Figure 6: Text code in Simulink copied to an Editor with compilation results

The sixth (6) state is "Convert Simulink code to FORTRAN equivalent". In this state, students use the concepts learnt in states 1 to 4 to come up with a syntactically correct FORTRAN equivalent of the Simulink code developed. The last state is the "Write code in text-based notation, compile and run code" (state 7). Here, the student is assumed to have mastered the techniques involved in writing correct programs in FORTRAN, the student now develops program instructions in FORTRAN using a text editor, compiles and subsequently runs the code to obtain results.

Our proposed programming learning approach introduces students, in particular novices, to basic FORTRAN programming concepts using friendly and graphical environments provided by Simulink with a view to making programming attractive to students. Using Simulink, one drags and drops blocks from standard libraries and connects them via data lines. Once the block diagram is prepared, the program can be run without compilation. Its use in a classroom setting can serve to demonstrate a multitude of basic and advanced concepts of most programming courses to enhance students' understanding of such concepts considerably.

The advantages of Simulink are that programming time is reduced considerably, and that the ease with which such a program can be modified encourages further tinkering, which undoubtedly is a good way for students to learn, develop their motivation, curiosity, and creativity.

The approach used in this study is straightforward because it assists students to come up with a running algorithm in Simulink, which is effortlessly converted to FORTRAN, as compared to turning a static flowchart to a program.

The objectives of the proposed learning tool are that (a) students learn computer programming with FORTRAN language using a structured based approach, and (b) students are able to write computer programs (in Simulink and FORTRAN) to solve varied science and engineering related problems. The learning materials comprised in the application include the following topics: Simulink overview, Simulink library and blocks, Data and variable types in Simulink and FORTRAN, input/output instructions (in Simulink and FORTRAN), assignment statement, and implicit functions (in Simulink and FORTRAN). Others include conversion of Simulink program to FORTRAN program, control statements (logical IF, block IF, ELSE IF), loops (for statement, Do loop) in Simulink and FORTRAN, Arrays, and Modular programming.

Participants

The research was conducted in OAU, Ile-Ife, Nigeria in two sessions, the first experiment (Experiment 1) was conducted immediately after the second (Rain) semester of the 2007/2008 session (during holiday) when the newly enrolled students had completed their first year and had no prior knowledge of programming. The purpose of the first experiment is to test students in the preliminary aspect of FORTRAN language. The second experiment (Experiment 2) that tests advanced concepts in FORTRAN, was conducted in the first (Harmattan) semester of the 2008/2009 academic session, during their second year, when they had started their first programming experience. The sample total was 22, comprising of the science and engineering undergraduate students, with grade point averages (GPAs) between 2.80 and 4.20 [mean (M) = 3.35, standard deviation (SD) = 0.42] out of five (5) possible points that can be obtained in a semester. The sample was grouped into two (Group A and Group B), with students' academic ability (indicated by their GPA) evenly distributed between the groups. Group A had 10 students assigned to it while Group B had the remaining 12 students. The sample size was restricted as a result of the limited resources available in the laboratory to cater for large number of students without belaboring few academic staff in the department.

Materials and Research Design

The study compared the proposed learning approach with the traditional teaching approach to teach FORTRAN programming, specifically to find out the effect (if any) of the use of the proposed learning tool to teach FORTRAN programming concepts. The proposed programming learning approach was administered on Group A, while the traditional form of teaching where an instructor explains and demonstrates key concepts using whiteboards and projections was applied on Group B.

The research question in the study is thus: can the proposed learning approach help students learn how to program in FORTRAN? The two applications were identical in terms of embedded learning objectives and learning materials and differed only in that one used a graphical environment managed by a DFA machine to teach programming concepts, while the other used the traditional approach of teaching. Any differences in learning outcomes and appeal to students between the two applications could, thus, be attributed to the graphical environment factor.

Two experiments (Experiment 1 and Experiment 2) were used in this study to explore the effects of the type of application used (proposed learning tool, and traditional form of teaching) on students' achievement, as measured by a knowledge test in programming concepts. Two groups (Group A and Group B) were used as mentioned in the previous section. We formulated the hypothesis of the study with the probability of rejecting a true null hypothesis at 5% ($\alpha = 0.05$) as:

H_0: $\mu_1 = \mu_2$: There will not be a significant difference in the achievement in computer programming between Group A and Group B.

The alternative hypothesis was stated thus:

H_1: $\mu_1 \neq \mu_2$: There will be a significant difference in the achievement in computer programming between Group A and Group B.

Since the test is 2-tailed, we used $\alpha/2 = 0.025$, with the critical region given as:

Critical region: $t_{0.025}(20) = 2.085962$. This tabulated t-value (t-tab) was obtained by looking up t-value at 20 degree of freedom and α=0.05. The decision rule was defined as:

Reject null hypothesis if t-calculated (t cal) $\geq$ 2.086 (i.e. if t cal $\geq$ | t-tab|).

Furthermore, after the completion of the intervention, an interview method was used to elicit the views of the treatment group.

Procedure and Data Analysis

The first experiment was conducted immediately after the 2007/2008 academic session, during the long holiday for a period of three weeks. The students who volunteered to participate in the experiment were introduced to the preliminaries of FORTRAN programming. Specifically, teaching them the following topics: (a) data and variables types, (b) arithmetic operations and statements, and (c) input and output statements. The two groups (Group A and Group B) were taught the introductory programming concepts for the first two weeks by the same lecturer using same course materials but different methods. The proposed learning tool was used on Group A while the traditional form of teaching was applied on Group B. At the end of the application, the students were evaluated in the third week to determine their level of competency in introductory FORTRAN programming. The questions posed varied from the simple true/false to multiple-choice questions.

The second experiment was conducted in the Harmattan Semester 2008/2009 academic session. Students in Group A were taken through the entire curriculum of the course (CSC 201) as approved by the University senate, using the proposed learning approach. Students in Group B were also taken through the same topics as contained in the approved curriculum but using the traditional form of passing instruction. The University Semester Examination (2008/2009 Session) for the course CSC 201 was used to evaluate the performance of the students in each group in programming. Students' performance in CSC 201 course was measured in terms of (a) the results of prescribed theory and practical examination, and (b) continuous assessment, which constituted 40% of the measured performance. Examination questions comprised both objectives and theory parts, and the scores obtained in the examination by students in each group were noted and recorded.

To investigate the potential differences between Group A and Group B in terms of the students' ability to learn programming concepts effectively using the proposed programming learning approach, the scores obtained by students in each group in the two experiments conducted were analyzed for equality of means using t-test, with the level of significance set at 0.05.

Results and Discussion

The evaluation of research results was studied by comparing the performance of students (Group A and Group B) in the two experiments (Experiment 1 and Experiment 2). In comparing the results between the two groups, we found that there was statistically significant difference between the groups in the two experiments, and thus we reject the null hypothesis.

There was statistically significant difference in the performance of students in Experiment 1 ($t(20) = 6.06$, $p < 0.001$). For the inference, t-tab=2.086, and t cal=6.06, and since t cal > t-tab, we reject the null hypothesis with 5% probability of Type I error and say that the performance of students in Group A and Group B differed. This is easily noticed from the mean score obtained from students in Group A that made significant performance using the proposed learning tool. The mean score obtained ($M = 73.90$, $SD = 5.59$) by the students in Group A was higher than that obtained by the students in Group B ($M = 57.08$, $SD = 7.13$).

Similarly, in the second experiment ($t(20) = 6.11$, $p<0.001$), since 6.11 > 2.086, there was statistically significant difference in the performance of students in Experiment 2, we therefore reject the null hypothesis again, and say that the performance of students in Group A and Group B differed. The mean score obtained ($M = 73.60$, $SD = 5.89$) by students in Group A was higher than that obtained by students in Group B ($M = 47.92$, $SD = 12.11$). Thus, from the two experiments conducted, students made significant learning gains using the proposed learning tool, we therefore conclude that there is significant difference in the achievement in computer programming between Group A and Group B. Table 1 summarizes the findings.

Table 1: t-Test: Two-Sample Assuming Equal Variances

	Experiment 1		Experiment 2	
	Group A	Group B	Group A	Group B
Mean	73.90	57.08	73.60	47.92
Variance	31.21	50.81	34.71	146.63
Standard Deviation	5.59	7.13	5.89	12.11
Observations	10	12	10	12
Pooled variance	41.99		96.27	
Hypothesized Mean Difference	0		0	
df	20		20	
t cal	6.06		6.11	
p(T<=t)	P<0.001		P<0.001	
T Critical two tail	2.086		2.085	

When comparing the performance of students in Group B in Experiment 1 and Experiment 2, it was discovered that there was difference in their performance (t cal=2.260, t-tab=2.074, p=0.034), with students' mean score in Experiment 1($M = 57.08$, $SD = 7.13$), better than what

was obtained in Experiment 2 (M = 47.92, SD = 12.11). The reason that may be attributed to this is that in Experiment 1, preliminary FORTRAN programming concepts were introduced to students, which resulted in their above average performance as compared to the complex concepts (entire curriculum) introduced to them in Experiment 2.

As a descriptive measure, individual interview was used to obtain attitude information from students in Group A. The students were queried regarding their feelings. The students' remarks were highly positive regarding their experience on the use of the proposed learning approach, which had further inspired their interest in programming.

Absolutely, the study had demonstrated that the proposed learning tool was more effective in promoting science and engineering students' knowledge of computer programming with students having better performance compared with those taught using the traditional method. It can therefore be concluded that graphical languages can be exploited as learning environments in universities to aid the teaching of programming to science and engineering students, as deduced from this study; they can considerably improve knowledge of programming concepts in the learning process. The findings obtained in this study seem to support the outcomes of some prior studies (Baldwin, & Kuljis, 2000; Ichikawa & Hirakawa, 1990; Rebelo, Marcelino, & Mendes, 2005), which emphasized the feasibility and usefulness of graphical language in contributing to increased academic achievement.

Conclusion

Low problem solving skill, as observed by the authors, is usually the cause of failure and frustration of a number of students in introductory programming courses. Text-based programming language has the major drawback that it makes beginners to lose sight of the problem solving aspects of using algorithms at the inception of language syntax. The cognitive advantage, which graphical methodologies provide over textual ones have been empirically established (Glinert & Tanimoto, 1984; Ichikawa & Hirakawa, 1990; Scanlan, 1989). Graphical programming environments allow the user to create programs by connecting together graphical icons representing different functions. This environment will help students, in particular engineering students, who tend to learn better from visual presentations, because the graphical nature of the program will make the structures easier to comprehend.

This article discusses difficulties experienced by novice programming students, especially science and engineering students, and it introduced a learning framework in an attempt to address the problem using a graphical environment in the form of Simulink, and DFA, which manages the learning process to teach FORTRAN programming to novices. Knowing fully well that undergraduate instructors are often extremely busy, devoting a considerable part of their time in developing Graphical User Interface (GUI) educational software, which usually requires a larger investment time, may seem unrealistic. Simulink is adopted in this study because of the availability of numerous basic and advanced concepts of most programming courses, which can be used to demonstrate students' understanding of programming concepts considerably.

The evaluation results of the proposed learning tool and the traditional method to teach FORTRAN programming to beginners was presented, an outcome of which was an improvement in learning effectiveness. This preliminary result has thus inspired the authors to develop online course materials for the proposed learning approach, which will be uploaded on a Learning Management System, recently acquired by the University, to create an environment where teaching of FORTRAN programming to a large number of students will be achieved with a relatively low specialized human effort. It is our belief that the students' understanding of programming concepts will be enhanced considerably, and by extension, this tool will provide students the valuable background in modeling and simulation, motivate them to perform better in other programming

courses (C++, Java, COBOL), and finally, improve their modeling skills in engineering courses such as signals processing, computer communications, and control.

In the future, we hope to increase the sample size to involve as many students as possible to review the effectiveness of the proposed learning method, and to measure other variables that influence the learning behaviour.

References

Allison, I., Orton P., & Powell, H. (2002). A virtual learning environment for introductory programming. *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences, Loughborough University, UK*, 48-52.

Baldwin, L. P., & Kuljis, J. (2000). Visualisation techniques for learning and teaching programming. *Journal of Computing and Information Technology*, *8*(4), 285–291.

Bongard, J., & Lipson, H. (2005). Active coevolutionary learning of deterministic finite automata. *Journal of Machine Learning Research*, *6*, 1651–1678.

Branch, J. W., Bivens, A. & Chan, C. Y. (2002). Denial of service intrusion detection using time dependent deterministic finite automata. *The First Annual Walter Lincoln Hawkins Graduate Research Conference*, Troy, New York, October 17, 2002.

Brown, M., (1988). Exploring algorithms using BALSA-II. *IEEE Computer*, *21*(5), 14-36.

Calloni, B. A., & Bagert, D. J. (1993). BACCII: An iconic syntax-directed system for teaching procedural programming. *Proceedings of the 31st ACM Southeast Conference, Birmingham, UK*, 177-183.

Carbone, A., Hurst, J., Mitchell, I., & Gunstone, D. (2001). Characteristics of programming exercises that lead to poor learning tendencies: Part II. *ACM SIGCSE Bulletin*, *33*(3), 93-96.

Carlisle, M. C., Wilson, T. A. Humphries, J.W., & Hadfield, S. M. (2005). RAPTOR: a visual programming environment for teaching algorithmic problem solving. *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education, St. Louis, Missouri, USA*, 176-180.

Cilliers, C., Calitz, A., & Greyling, J. (2005). The effect of integrating an iconic programming notation in CS1. *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, Monte de Caparica, Portugal*, 89-93.

Cooper, S., Dann, W., & Pauch, R. (2003). Using animated 3D graphics to prepare novices for CS1. *Computer Science Education Journal*, *13*(1), 3-30.

Dunican, E. (2002). Making the analogy: Alternative delivery techniques for first year programming courses. *Proceedings from the 14th Workshop of the Psychology of Programming Interest Group, Brunel University, UK*, 89-99.

Glinert E. & Tanimoto S. (1984). Pict: An interactive graphical programming environment. *IEEE Computer*, *17*(11), 7-25.

Goldman, K.J. (2003). A demonstration of JPie: An environment for live software construction in Java. *Proceedings of the 18th annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, Anaheim, CA, USA*, 74 - 75.

Guzdial, M. & Usselman, M. (2003). Media computation course for non-majors. *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education, Thessalonika, Greece*, 104-108.

Ichikawa, T., & Hirakawa, M. (1990). Iconic programming: where to go? *IEEE Software*, *7*(6), 63-68.

Keshri, A. K., Sinha, R. K., Hatwal, R., & Das, B. N. (2008). Epileptic spike recognition in electroencephalogram using deterministic finite automata. *Journal of Medical Systems*, *33*(3), 173-179.

Klassen, M. (2006). Visual approach for teaching programming concepts. *Proceedings of the 9th International Conference on Engineering Education, San Juan, Puerto Rico*, 23-28.

Kölling, M., & Rosenberg, J. (2001). Guidelines for teaching object orientation with Java. *ACM SIGCSE Bulletin*, *33*(3), 33-36.

Kolling, M., Quig, B., Patterson, A., & Rosenberg, J. (2003). The BlueJ system and its pedagogy. *Journal of Computing Science Education*, *12*(4), 249–268.

Levy, R. B., Ben-Ari, M., & Uronen, P. A. (2003). The Jeliot 2000 program animation system, *Computers & Education*, *40* (1), 1–15.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y.B., Laxer, C. Thomas, L., Utting, I., & Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin*, *33*(4), 125-140.

Meisalo, V., Suhonen, J., Sutinen, E., & Torvinen, S. (2002). Formative evaluation scheme for a web-based course design. *Proceedings of the 7th Annual Conference on Innovation and Technology in Computer Science Education, University of Aarhus, Denmark,* 130-134.

Miliszewska, I., & Tan, G. (2007). Befriending computer programming: A proposed approach to teaching introductory programming. *Proceedings of the Informing Science and Information Technology Education Joint Conference, Ljubljana, Slovenia*, 278-289.

Naps, T. (2005). Jhavé – Supporting algorithm visualization. *IEEE Computer Graphics and Applications*, *25*(5), 49-55.

Pascarella, E. T., & Terenzini, P. T. (2005). How college affects students: A third decade of research. San Francisco: Jossey-Bass.

Rebelo, B., Marcelino, M. J., & Mendes, A. J. (2005). Evaluation and utilization of SICAS – A system to support algorithm learning. *Proceedings of 8th IASTED International Conference on Computers and Advanced Technology in Education. Oranjestad, Aruba*, 153-158.

Rodger, S. (2002). Introducing computer science through animation and virtual worlds. *Proceedings of Technical Symposium on Computer Science Education SIGCSE'02, Cincinnati, Kentucky,* 186-190.

Sanders, D., & Dorn, B. (2003). Classroom experience with Jeroo. *Journal of Computing in Small Colleges*, *18*(4), 308-316.

Scanlan, D. (1989). Structured flowcharts outperform pseudocode: An experimental comparison. *IEEE Software*, *6*(5), 28-36.

Schittenkopf, C., Deco, G., & Brauer, W. (1997). Finite automata-models for the investigation of dynamical systems. *Information Processing Letters*, *63* (3), 137-141.

Sheard, J., & Hagan, D. (1998). Experiences with teaching object-oriented concepts to introductory programming students using C++. *Proceedings of Technology of Object-Oriented Languages and Systems, Beijing, China*, 310-319.

Stasko, J. (1992). Animating algorithms with XTANGO. *SIGACT News*, *23*(2), 67-71.

Tavares, J., Brzezinski, I., Huet, I., Cabral, A., & Neri, D. (2001). "Having coffee" with professors and students to talk about higher education pedagogy and academic success. *Proceedings of the 24th International HERDSA Conference, Newcastle, Australia*.

Thomas, L., Ratcliffe, M., Woodbury, J., & Jarman, E. (2002). Learning styles and performance in the introductory programming sequence. *ACM SIGCSE Bulletin*, *34*(1), 33-37.

Traynor, D., & Gibson, P. (2004). Towards the development of a cognitive model of programming; a software engineering approach. Retrieved December 10, 2008, from http://www.cs.nuim.ie/~pgibson/Research/Publications/E-Copies/PPIG04.pdf

Webber, A. (2007). Formal language: A practical introduction. Washington: Franklin, Beedle & Associates.

Zaccone, R., Cooper, S., & Dann, W. (2003). Using 3D animation programming in a core engineering course seminar. *Proceedings of the 33rd ASEE/IEEE Frontiers in Education Conference, Boulder, Colorado, US*, 14-17.

Biographies

Dr. **Anuoluwapo Ajayi** holds a Ph.D. degree in Computer Science from the Obafemi Awolowo University (OAU), Ile-Ife, Nigeria. He is a lecturer in the department of Computer Science and Engineering, OAU, Ile-Ife. His research interests include Artificial Intelligence and Programming Languages Techniques. He is a member of International Association of Engineers (IAENG), Nigerian Computer Society (NCS) and IEEE Computer society. He has published scientific articles in several journals of international repute.

Dr. **Emmanuel Olajubu** has a Ph.D. degree in Computer Science from the Obafemi Awolowo University (OAU), Ile-Ife, Nigeria. He is a member of Nigerian Computer Society, Computer Professional Registration Council of Nigeria, and International Association of Engineers (IAENG). He has over 10 years of experience in teaching and research in OAU, Ile-Ife. He has published in reputable journals and learned conferences. His research interest is Computer Communication and Networking with emphasis on Network Fault Management and Mobile Agent Systems.

Deborah Ninan obtained her B.Sc. honours in Applied Physics (Electronics) from the University of Lagos, M.Ed. in Educational Evaluation from the University of Ibadan, PGD and M.Sc. degrees in Computer Science from the Obafemi Awolowo University (OAU), Ile-Ife, where she is currently an academic staff and a Ph.D. student. Her research area is the development of Virtual Reality Model for Higher Institutions in Developing countries. She is a member of Nigeria Computer Society (NCS) and Third World Organization for Women in Science (TWOWS).

Solomon Akinboro holds a B.Tech in Computer Engineering from Ladoke Akintola University of Technology, Ogbomoso, Nigeria, and M.Sc. in Computer Science, from Obafemi Awolowo University Ile-Ife, Nigeria. He is a member of the Nigeria Society of Engineers (NSE) and the Nigerian Institute of Electrical and Electronic Engineering (NIEEE). He is currently a staff of Bells University of Technology Ota, Ogun State, Nigeria. His research interests are Data Communication and Networking, Export system, and Artificial Intelligence.

Dr. **Abimbola Soriyan** (Ph.D. Obafemi Awolowo University) is a currently a Reader and the Deputy Director of Distance Learning Center, Obafemi Awolowo University, Ile-Ife, Nigeria. She has over 17 years of experience in teaching and research. Her interests include information systems development with emphasis on healthcare. She has published a good number of journal and learned conference articles, and made useful contributions to books