International Journal of Modern Engineering Research (IJMER)
www.ijmer.com Vol.3, Issue.1, Jan.-Feb. 2013 pp-350-353 ISSN: 2249-6645

A Teaching Methodology for Introductory Programming Courses using Alice

Ozgur Aktunc
Department of Engineering, St. Mary's University, San Antonio, United States

Abstract: *Enrollment numbers and retention rates have been decreasing in technology programs in United States of America, particularly for underrepresented minority groups. At St. Mary's University, we started an Engineering Summer Program (ESP) that consists of programming and robotics sessions for high school students. The programming courses offer an opportunity for many students to write a computer program for the first time in their lives. In these courses we are using a popular instructional tool for teaching programming called Alice. Alice is a 3D interactive environment that has visual and narrative aspects. Our objectives include motivating the students to start programming, keeping the frustration levels to a minimum, which is a common problem for new programmers, and increasing the students' problem solving skills. In this paper, we present a survey of the common challenges faced in introductory programming courses, the objectives of these courses, and our approach to start programming using Alice with a transition to Java, a popular object-oriented language.*

Keywords: *Introductory programming, education, teaching methodology, Alice, Java.*

I. INTRODUCTION

Introductory programming courses are part of Computer Science (CS), Computer Engineering (CE), Computer Information Systems (CIS), and many engineering curricula. Programs generally offer two to four programming courses consisting of fundamentals of programming, object-oriented programming (C++ and Java), and applications of popular languages.

The discussion of what needs to be taught in introductory programming courses has existed since the early seventies. Gries suggested that an introductory course should be concerned with three aspects of programming [1]:

- How to solve problems.
- How to develop an algorithmic solution to a problem.
- How to verify that an algorithm is correct.

Despite the advances in the availability of resources to teach programming courses, the fundamental issue of teaching problem solving using algorithms is still the main challenge for instructors. While we do not want to delve into the reasons for lack of problem solving skills in engineering and CS students, we would like to emphasize the importance of teaching problem solving. We adopt Polya's [2] four-phase process as our guideline to teach problem solving:

1. Understand the problem.
2. Devise a plan.
3. Carry out the plan.
4. Look back (reflect).

Polya's work has been adapted by many educators. Barnette et al. [3] applied Polya's four-phase process of problem solving to programming:

1. Understand the problem.
a. Knowing the boundaries of the problem.
b. Knowing the constraints of the solution.
c. Knowing what actions are allowed.

2. Devise a plan.
a. Organize thought to develop a detailed algorithm.
b. Use tools, such as outlining and pseudo code.

3. Implement the plan.
a. Carry out the steps in the algorithm.
b. Translate the problem into a language understandable by the device to be used.

4. Test the plan.
a. Did the solution yield appropriate results?
b. Can the solution be improved?

Introductory programming courses should help students gain problem solving skills while they become familiar with computers and programming languages. Palumbo [4] published a review of literature regarding the connection between

learning programming languages and problem solving skills. Palumbo discussed the transferability of the skills learned in programming courses to problem solving skills. For engineering students, learning to apply acquired skills and expertise to a new problem solving domain is a crucial benefit of programming courses.

Many techniques and tools have been used to teach programming in the last four decades. The transition from procedural languages, Fortran and Pascal, to object-oriented languages, Java and C++, has led to the increase in the number of new tools. The tools that help beginners learn to program can be classified as [5]:

- Visual programming tools.
- Narrative tools.
- Flow-model tools.
- Specialized realization tools.
- Tiered language tools.

In the following sections, we will review the challenges faced in teaching programming courses and introduce our pedagogy to overcome some of these challenges.

II. CHALLENGES OF TEACHING INTRODUCTORY PROGRAMMING COURSES

Our literature survey indicated numerous challenges instructors face while teaching programming courses [6, 7]:

- Wide variation in the students' backgrounds.
- Majority of the students find programming to be a difficult and complex cognitive task.
- Excessive amount of time spent teaching the language syntax. Spending too much time learning the syntax without context is detrimental to students' success.
- Most programming environments are also confusing as they were developed for professional software engineers, making them difficult for first year students.
- Inability to see the execution of the program before correcting the syntax and execution errors.
- Lack of motivation to learn programming, as the programming profession is considered boring by many students.

It is difficult to propose a simple solution to overcome these challenges. We will summarize some of the ideas proposed in the literature, followed by our own approach to improve teaching methods.

One of the first issues that an instructor has to make a decision about is the choice of a programming language. Teaching the syntax of the language chosen is one of the pedagogical objectives of the instructor. However, instructors should place a higher priority on improving students' problem solving and design skills. We agree with the perspective of Al-Imamy et al. [8] that the choice of programming language is less important than the issues of what to teach, how to teach it, and finding the balance between knowledge of the syntax, design skills, and problem solving skills.

The majority of the software community agrees that object-oriented programming is a good tool for teaching the fundamentals of programming. Object-oriented paradigm supports concepts, such as well-structured programming, modularization, and program design. It also supports techniques, such as programming in teams, maintenance of large systems, and software reuse, which are vital in today's business world. We believe that object-oriented programming should be taught first in the introductory programming courses. The reason for doing this is the issue of paradigm shift. Learning to program in an object-oriented style seems to be very difficult after being used to programming in a procedural style. Research shows that becoming familiar with an object-oriented language, having started out with procedural programming, may take from 6 to 18 months [9]. On the other hand, students do not seem to have as much difficulty understanding object-oriented principles when they learn them first.

Programming is central to the computing curricula. CS, CE, and CIS curricula all require instruction in computer programming. Many students in these programs perceive programming as difficult and socially isolating. Students' psychological barriers towards programming are as challenging to overcome as the technical issues we mentioned earlier. Programming courses must be redesigned to be more suitable for the new generation of students who have grown up with computers, gaming, and interactive user interfaces [10]. Students get discouraged when "programming the solution of a problem" is harder for them than "working it out in their head". Current programming environments are not designed to help the students to transform their algorithmic solution to programs.

Another issue that higher education institutes face is that women and minorities are underrepresented in programming and computer related jobs and schools. To improve their representation, new approaches are needed to teach programming courses and increase these groups' confidence in their knowledge as it influences their perseverance [11]. Whether taken at a high school or an undergraduate institution, the first programming course has a lasting effect on students. A positive course experience leaves students with good programming habits, the ability to learn on their own, and a favorable impression of programming as a profession [12].

Engineering or computer science summer programs available to middle and high school students have successfully introduced programming to these groups. Evidence shows that students participating in these programs have increased confidence in their programming ability, understand basic programming concepts, and understand the relationship between algorithms and programs [13]. The author of this paper started an initiative at St. Mary's University to have an engineering summer program in 2010 offering programming and robotics courses to high school students. This summer program introduces the engineering profession to underrepresented groups in conjunction with teaching programming and robotics.

III. OUR PEDAGOGY FOR TEACHING INTRODUCTORY PROGRAMMING COURSES

One of the most popular instructional tools for teaching programming is called Alice. Alice is a 3D interactive environment that has visual and narrative aspects. Alice is an open-source software written in Java. It supports creating animations and building virtual worlds through a user interface where the user can drag and drop basic programming blocks to create programs (Figure 1). Alice allows students to choose from many objects, such as animals or buildings, and create their virtual world based on these predefined objects. Students can create their programs by dragging the statements/expressions into the main window instead of having to write code. This mechanism helps to avoid syntax errors that are known to discourage new programmers. The interface allows students to change the properties of their objects and create their own methods and functions (Figure 2). Students can also create animations and make their virtual world interactive using keyboard and mouse inputs. The output of the program can be easily accessed anytime, without the need for compiling, by using the play button. This brings up an animation window where the student can watch his/her animation at different speeds. The visual output allows the students to see both the results of their program immediately and the building blocks of the program.

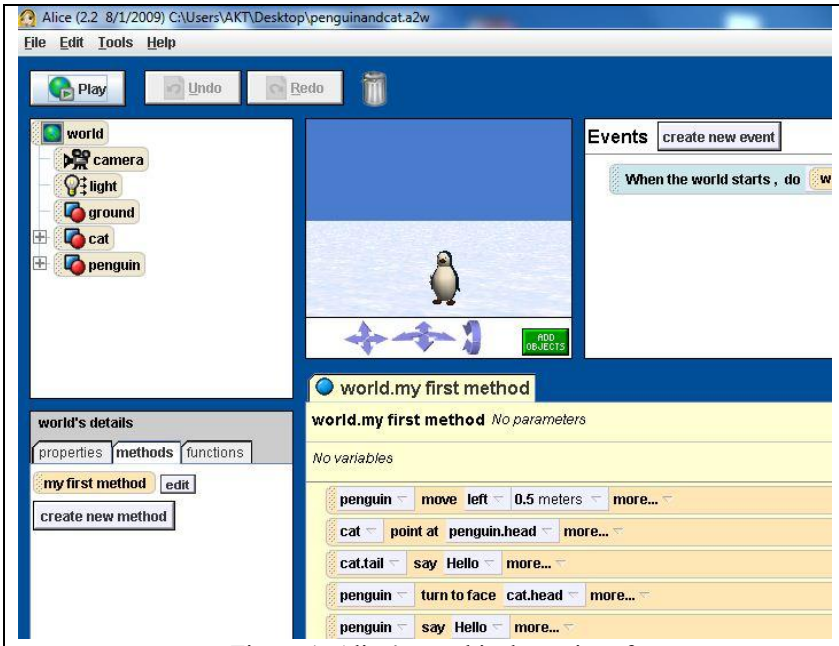

Figure 1. Alice's graphical user interface

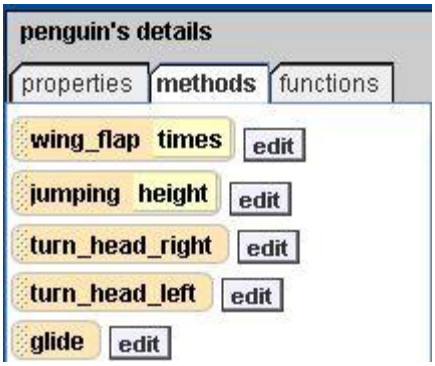

Figure 2. Alice's method editing interface

It is well-known in education that the use of visual stimuli increases student comprehension significantly. A program's internal state and flow of execution can be represented visually by Alice [14]. The animations where characters play out a scene or a virtual world where objects respond to mouse and keyboard inputs are more appealing than text-based programs. Alice is very helpful for instructors that teach the basics of object-oriented paradigm. Students can add objects to their virtual world, add or modify properties of these objects, and create their own methods. Previous research showed that using Alice sustains student interest and involvement [15, 16]. Alice gives the students an intuitive feel for object-oriented concepts that makes the transition to object-oriented languages, such as Java and C++, easier.

We follow an inquiry-based approach where students receive basic instructions to use Alice and are expected to identify the procedures and do the implementation themselves. Programming workshops should not be strictly cookbook type practices where students have step-by-step instructions on how to select objects and choose the methods for the objects. The cookbook type instruction may result in students leaving the workshop with animations and stories without any idea of their significance and the underlying concepts. Evidence shows that students' research skills, problem solving abilities, and self-confidence in programming improve significantly with inquiry-based instruction [17].

IV. FUTURE STUDY

Our Engineering Summer Program at St. Mary's University provides us with invaluable data regarding the use of Alice software. We hold surveys for each session to record the students' opinions. We plan to do a comparative study based on our ESP survey results and our programming course surveys collected from our freshman courses. We also have plans to incorporate Alice into our beginning programming course at the freshman level. Our approach will begin with teaching object-oriented basics using Alice, followed by a transition to Java language. In our software engineering program at St. Mary's University, this introductory programming course is followed by an object-oriented design course and a higher level Java and applications course.

V. CONCLUSION

Alice is undoubtedly a useful tool for teaching introductory programming. Alice is currently being used successfully in numerous undergraduate and high school programs. However, the instructors should define the learning objectives of the course based on the object-oriented paradigm, not on Alice alone. Focusing exclusively on one tool would distract both the instructor and the students from the main objectives of a programming course. Alice is limited in terms of the applications that can be created, mainly animations and virtual worlds. Alice should be used to teach the basic concepts of programming and these concepts should be applied to high level languages, such as Java, that do not have the limitations of Alice. The transition from Alice to Java allows students to compare and repeat these fundamental concepts and strengthen their understanding. The combination of Alice and Java gives students a solid programming background and prepares them for higher level programming courses.

References

[1] D. Gries, What should we teach in an introductory programming course? Proceedings of the Fourth SIGCSE Technical Symposium on Computer Science Education, New York, NY, 1974, 81-89.
[2] G. Polya, How to solve it, 2nd edition (Princeton, NJ: Princeton University Press, 1957).
[3] N. D. Barnette, W. D. McQuain, and M. A. Keenan, Programming in C++, course notes, Computer Science Department, Virginia Tech University, 1999.
[4] D. Palumbo, Programming language/problem solving research: A review of relevant issues, Review of Educational Research, Vol. 60, 1990, 65-89.
[5] K. Powers, P. Gross, S. Cooper, M. McNally, K. J. Goldman, V. Proulx, and M. Carlisle, Tools for teaching introductory programming: what works?, SIGCSE Bulletin, Vol. 38, 2006, 560-561.
[6] A. Azad and F. Kohun, Considerations for Selecting a Programming Language to Teach Perspective Teachers, Alice Symposium, Duke University, Durham, NC, 2009.
[7] D. Goulet and D. Slater, Alice and the introductory programming course: An invitation to dialogue, Information Systems Education Journal, Vol. 7, 2009.
[8] S. Al-Imamy, J. Alizadeh, and M. A. Nour, On the development of a programming teaching tool: The effect of teaching by templates on the learning process, Journal of Information Technology Education, Vol. 5, 2006.
[9] M. Kolling, The problem of teaching object-oriented programming, part 1: Languages, Journal of Object-Oriented Programming, Vol. 11, No. 8, 1999, 8-15.
[10] B. McKenzie, Introductory Programming with Alice as a Gateway to the Computing Profession, Proceedings of the 23rd Annual Conference for Information Systems Educators, 2006.
[11] B. L. Wellman, J. Davis, and M. Anderson, Alice and robotics in introductory CS courses, the Fifth Richard Tapia celebration of diversity in computing conference: intellect, initiatives, insight, and innovations, 2009, 98-102.
[12] M. O. Pendergast, Teaching introductory programming to IS students, Journal of Information Technology Education, Vol. 5, 2006, 491-515.
[13] B. Moskal, D. Lurie, and S. Cooper, Evaluating the effectiveness of a new instructional approach, Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education, 2004, 75-79.
[14] D. Parsons, Programming osmosis: Knowledge transfer from imperative to visual programming environments, 20th Annual Conference of the National Advisory Committee on Computing Qualifications, 2007.
[15] T. Wang, W. Mei, S. Lin, S. Chiu, and J. M. Lin, Teaching programming concepts to high school students with Alice, Proceedings of the 39th IEEE International Conference on Frontiers in Education Conference, 2009, 955-960.
[16] T. Daly, Using introductory programming tools to teach programming concepts: A literature review, The Journal for Computing Teachers, 2009.
[17] P. Brickman, C. Gormally, B. Hallar, and N. Armstrong, Effects of Inquiry-based Learning on Students' Science Literacy Skills and Confidence, International Journal for the Scholarship of Teaching and Learning, Vol. 3, No. 2, 2009.