ADAPTIVE TUTORS FOR DEVELOPING VISUAL REASONING SKILLS IN PROGRAMMING COURSES

Rika Yoshii

Computer Science and Information Systems Department, California State University San Marcos, California, U.S.A.

Keywords: Visual reasoning, Programming, Adaptive, Mastery.

Abstract: This position paper argues that students who have difficulty programming often exhibit difficulty in building correct semantic representation of program statement effects. We argue for the need to develop and use tutoring software to help students develop visual reasoning skills in programming. Adaptive tutoring systems will allow students to work privately at their own pace. Example tutoring system designs incorporating visualization, adaptive learning, conversational tutoring and mastery learning are presented.

1 INTRODUCTION

As early as the late 80's, difficulties students face in solving algebra word problems have been discussed widely. Studies have been published in cognitive science journals indicating students' problems in building correct semantic representation of the situations depicted in word problems (Hall, 1989a)(Hall, 1989b). The same type of difficulty has been discussed in the area of programming (Buchananand, 1995). Many students cannot envision correct semantic representation of

- the problem statements, and/or
- programming constructs,

and they fail to pass beginning programming classes. But what are we doing to help students develop visual reasoning skills?

One possibility is to use example diagrams in lectures and force students to use those diagrams to explain their program parts. This approach works well with students who have little or no ability to form their own diagrams, but it will inhibit creativity in other types of students. To help only those who are having trouble forming semantic representation, tutoring software to help students in this area will be valuable, allowing those students to practice privately at their own pace. Adaptive tutoring software availability is especially important in beginning programming classes with a large number of students, making individualized instruction difficult.

We have developed two tutoring systems in second language learning, incorporating the visualization approach. DaRT helps students develop visual understanding of English articles such as "a" versus "the" (Yoshii, 1998). DeCEN helps students develop visual understanding of countable versus non-countable nouns (Slott, 2005), (Yoshii, 2010). Both systems have been used with actual students and produced good results. We believe that the same can and should be done for programming classes.

In this paper, we will describe a sequence of findings and projects at California State University, San Marcos (CSUSM) leading to our current project for creating a highly interactive adaptive tutor to help students visualize effects of C++ statements.

2 VISUAL REASONING FOR ENTRY LEVEL MATHEMATICS

Entry Level Mathematics was the first area in which we assessed the student difficulty in forming semantic representation of situations. In our first programming course for computer science majors, students often expressed difficulty writing a small program for solving simple mathematical problems. After giving a quiz each semester to students entering the first programming class, we realized that all students who missed the following problem drew incorrect diagrams to depict the described

Yoshii R. (2010).
ADAPTIVE TUTORS FOR DEVELOPING VISUAL REASONING SKILLS IN PROGRAMMING COURSES.
In *Proceedings of the 2nd International Conference on Computer Supported Education*, pages 364-367
DOI: 10.5220/0002860703640367

situation:

> "Trains A and B are at the same station. Train A goes to the east at X miles per hour. Train B goes to the west at Y miles per hour. The trains left the station at the same time and travelled for Z hours. What is the distance between Train A and Train B?"

Students with incorrect answers drew diagrams showing the trains moving in the same direction or towards each other.

This experiment was repeated again this year in both the first and second programming classes. In both classes, 35% of students had incorrect diagrams and thus equations. These students were not having problems with the programming task but were having problems coming up with the correct equations because they could not visualize the situation correctly. This result echoes what was discussed in the papers (Hall, 1989a) and (Hall, 1989b). Unfortunately, appropriate remedial instruction for the 35% of the students cannot be provided within the context of the programming classes.

3 VISUAL REPRESENTATION OF LINKED LISTS

The second area in which we discovered the same difficulty was with linked lists in the second programming course. After two lectures showing diagrams for adding and deleting nodes from linked lists, the students were given an assignment to draw the effects of C++ statements for manipulating linked lists before they were allowed to write a linked list class. An example problem is:

> "Why can't we do the following to add a node to the rear? Draw the effect. Rear = new Node;
> Rear->Next = Rear;"

Every semester, more than 50% of the class had great difficulty drawing correct pictures, and the same group of students had difficulty coding the linked list class. With a large number of students, it was not possible to sit with each student to individually analyze their diagrams.

4 CPP TRACER TUTOR

The students in the first programming course who had problems with Entry level Mathematics problems were also resistant to the explanations and exercises that were typically successful in leading others to an visual understanding of machine operation, and the ability to understand their own programs. Instead of relating their programs to the sequences of machine actions they cause, the students seemed to memorize a few stock "boilerplate" program fragments for the problems they had seen. They were unable to analyze why a program works or does not, seeing no connection to the resulting actions by the machine.

With these students, simply using a debugger did not help since they needed repeated explanations with examples. But they were reluctant to ask for help and preferred to practice without being noticed by other students. Therefore, we developed a tutoring system, Cpp Tracer Tutor (Yoshii, 2003). The goal of the Tutor is to help students develop and practice their reasoning skills using visual representations of executing programs. The following are its main features. The latter three of these features were adopted from the Irvine-Geneva strategy (Bork, 1992)(Bork 2001):

- *Visual representations of programs*: the Tutor provides a live media visualization of the actions taken by the machine in executing a running program.
- *Conversational tutoring*: the Tutor provides frequent interactions with free-form student answers.
- *Adaptive learning*: the Tutor uses student answers to provide appropriate hints and to select the next exercise.
- *Mastery learning*: the Tutor does not let the student move onto a different program example until the student shows no difficulty understanding the current one.

The Tutor, written as a Java applet is available via the Internet, allowing students to use it privately at their own pace. Currently, the Tutor supports five different types of example C++ programs that are often seen in the first programming course: 1) a for-loop with a constant as its upper limit, 2) a for-loop with the upper limit pre-supplied by the user, 3) a while-loop that continuously take the user's input until a sentinel value is reached, 4) a while-loop that terminates when a sentinel value is read or a certain number of iterations has been done, and 5) a while-loop that terminates when a sentinel value is read or the sum of input values reaches a certain number. The Tutor's design permits addition of more example programs without making major changes to the system.

Assessment in five different laboratory sections of the first programming course indicate that the students had an average improvement of 15% in points between the pre and post-tests on reading and writing C++ loops after using the system for only two hours.

5 THE NEXT STEP : VISUAL TUTOR FOR C++ STATEMENTS

A problem with Cpp Tracer Tutor is that it assumes the students already understand the effects of individual statements found in loops. With students who have problems with algebra word problems, training in visual reasoning must start earlier. Another problem is that the students are simply observing the visualization of actions. The students need to be trained to visualize the actions.

Therefore, we are currently designing a system for helping students visualize the effects of individual C++ statements. As with the Cpp Tracer Tutor, the tutorial style will be based on the Irvine-Geneva theory emphasizing:

- *Conversational tutoring*
- *Adaptive learning*
- *Mastery learning*

The system will start off by showing correct visual representation of C++ statements using many examples. In the tutorial section, the student will be given a C++ statement. By clicking and dragging icons the student must create the visual understanding of that statement. The system will categorize wrong answers and record them so that the student can be given an appropriate sequence of exercises. The system will give appropriate conversational hints and take the student back to the same question. When the answer is finally correct, the system will give the same type of question again to verify student understanding. For example:

`Declaration` `int A;`
The student has to select a memory box and add labels for the type and name.

`Input` `cin >> A;`
The student has to select the correct box and drag the user input into the box.

`Output` `cout << A;`
The student has to select the correct box and copy the value to the output stream.

`Assignment` `A = A + 1;`
The student has to select the correct box and change the value.

`Loops` Each part of a loop will be highlighted in turn in the execution order, and the student is asked to show its effect.

In advanced tracks, arrays, stacks, queues and linked lists will be included.

One of the goals of the project is to create it based on a language-independent framework so that the tutor can be used for learning any programming language. Once the system is complete, we plan to 1) assess its usability with actual students and teachers, and 2) assess its effectiveness with our students and students at local community colleges.

6 SUMMARY

We have presented an acute need for developing tutoring systems to help students develop visual reasoning skills in order to succeed in programming classes. Out findings in beginning programming classes at CSUSM show that many students cannot envision correct semantic representation of the word problem statements, and/or programming constructs.

We have shown two example tutor designs incorporating visual reasoning, adaptive learning, conversational tutoring and mastery learning. With these types of tutoring systems, students in large classes can work at their own pace at home to develop their visual reasoning skills.

We also believe that such systems should be developed for Entry Level Mathematics classes to train students to visualize problem situations before they come to programming classes.

REFERENCES

Bork, A. et al., 1992. The Irvine-Geneva Development System. In R. Aiken (Ed.), *Education and Society Vol. II*, North Holland: Elsevier Science Publisher.

Bork, A, Gunnarsdottir, S., 2001. *Tutorial Distance*

Learning: Rebuilding our Educational System, Kluwer Academic Publishers.

Buchanan, R., Farrand, P., 1995. Can simulations help students understand programming concepts: a case study.http://www.ascilite.org.au/conferences/melbourne95/smtu/papers/buchanan.pdf.

Hall, R., Kibler, D., Wenger, E., & Truxaw, C., 1989a. Exploring the episodic structure of algebra story problem solving. *Cognition and Instruction, 6*(3), 223-283.

Hall R., 1989b. Qualitative diagrams: supporting the construction of algebraic representations in applied problem solving. In D. Bierman, J. Breuker & J. Sandburg (Eds.), *Artificial intelligence and education, Proceedings of the 4th International Conference on AI and Education* (116–122). Amsterdam, Netherlands: IOS.

Slott, K., Yoshii, R., 2005. "DeCEN and Tutor Writer: Making an Interactive CALL Tutor Available as a Java Applet", *Proceedings of the ED-MEDIA 2005 Conference, June 2005*.

Yoshii, R., 1998. "DaRT: A CALL System to Help Students Practice and Develop Reasoning in Choosing English Articles." *CALICO Journal, vol. 16, no 2., pp. 121-155, Fall 1998.*

Yoshii, R., Milne, A., 2003. "The C++ Tracing Tutor: Visualizing Computer Program Behavior for Beginning Programming Courses", *Proceedings of the ED-MEDIA 2003 Conference, June 2003.*

Yoshii, R., Pasrija, N., 2010. "TWIGy: The Tutor Writer Input file Generator to Help Create an Interactive Individualized Tutor that Runs Over the Internet", *Proceedings of the SITE 2010 Conference., March 2010.*