WCLTA 2010

Troubleshooting assessment: an authentic problem solving activity for it education

David C. Webb a *

aUniversity of Colorado at Boulder, 249 UCB, Boulder, Colorado 80309, USA

Abstract

To evaluate the effectiveness of an instructional unit for game design and computer programming, we designed an authentic assessment with five troubleshooting scenarios. This assessment was completed by 24 middle grades students (age 12 – 14 years) after 10 hours of instruction using a visual programming environment. Students successfully completed most of the tasks in 45 minutes. Results from the Troubleshooting Assessment demonstrated that students developed sufficient fluency with programming to be able to apply their knowledge to new problems. These results suggest that troubleshooting scenarios can be used to assess student fluency in computer programming and computer-based problem solving.

© 2010 Published by Elsevier Ltd. Open access under CC BY-NC-ND license.

Keywords: authentic assessment; computational thinking; computer programming; game design; problem solving; STEM education; technology-based assessment

1. Introduction

Calls for investment in mathematics and science education in the United States and other countries are often argued on the basis of promoting innovation, design, and production of new knowledge. Yet, in spite of these calls, there are few opportunities for young students to innovate, design, and produce using computers. Most students complete their compulsory educations as users, not designers, of computer products. This leads to the prevailing perception that technology is a tool for computation, game play, word processing, or social networking.

If students are given opportunities to design games and simulations, the products they create can provide evidence of their knowledge of the programming interface. However, when computer instruction is enacted by teachers in ways that constrain the programming of computer projects to a stepwise cookbook approach to computer projects, this can create the illusion that students understand more about computer programming and design than their products might suggest. Is the end product simply a result of following instructions, or do students really understand the relationship between the programmed arguments and the product?

1.1. Assessing student learning with new interfaces

Computer programming has been viewed as an instructional context that promotes broad educational goals, such
as problem solving and logical reasoning, often using a variety of tactile and visual representations. These goals are in addition to specific computer science objectives that include understanding the importance of programming syntax and recognizing the constraints and affordances of computer software.

Assessing student learning in rapidly developing technology-based fields is confounded by differences among software interfaces. Simple differences in visual programming interfaces can challenge young children (and adults!) when they try to apply previously learned programming skills with new software. It can often take hours for students to become acquainted with new commands and to understand the relationship between the commands and screen-based outputs that might be used in a gaming environment or science simulation (e.g., the design, placement, and control of animated agents). Because evidence of computer programming fluency is inextricably tied to the software interface, student learning should be assessed in situations where their programming methods, strategies and design decisions can be observed.

1.2. Authentic assessment

According to Newmann, Secada and Wehlage (1995), authentic assessments promote disciplined inquiry, construction of knowledge and have value beyond school. Disciplined inquiry is represented by sustained, deep inquiry into the processes and concepts of the discipline. When assessed authentically, a student has the opportunity to synthesize knowledge and processes to create new strategies or products. Incorporating tasks that have value beyond school (such as tasks that mirror work accomplished by adults in the discipline) help students make connections to the world beyond the classroom. Giving students work that is similar to what is expected of experts in that field, in this case by game designers and computer programmers, is an overarching theme in the authentic assessment literature (Wiggins, 1998).

Administration of pre-unit assessments for software-dependent instruction is problematic as student fluency requires some degree of familiarity with the software. Therefore, end-of-unit assessments for evaluation purposes make the most sense. In instructional sequences where time is limited (e.g., < 10 hours), activities that offer opportunities for assessment and new student learning are desirable. For computer programming and game design, troubleshooting is an activity that fits the three criteria for authentic assessment: promoting disciplined inquiry, constructing knowledge, and value beyond school. It is worth noting that these three criteria are, in fact, descriptions of opportunities for student learning. This dual purpose of authentic assessment – for both learning and assessment – means that it can be used as an embedded assessment, a goal oriented instructional activity where students may not even notice that they are completing a summative assessment.

1.3. Troubleshooting defined

Jonassen and Hung (2006) described troubleshooting as “among the most common types of problem solving” (p. 78). They argue that problems well-suited for troubleshooting:

- …usually possess a single fault state…;
- have known solutions with easily interpreted success criteria…;
- rely most efficiently on experience-based rules for diagnosing most of the cases…;
- require learners to make judgments about the nature of the problem;
- and vary significantly in terms of system complexity and dynamicity… (p. 78-9).

Research on the use of troubleshooting scenarios as assessment events has typically focused on adults in medical fields (e.g., Mislevy et al, 1999), professionally oriented undergraduate majors (e.g., Darabi, Nelson & Palanki, 2007), and vocational programs (e.g., Frezzo, Behrens & Mislevy, 2010). Use of troubleshooting scenarios for training and assessment is also found in descriptions of flight simulations and military training. Troubleshooting often involves the analysis of common problems encountered in technical support of machines, software, and systems (Williamson et al, 2004). While prevalent in studies of adult learners, electronic searches for research on the
use of troubleshooting tasks with young students with ERIC, PsychLit and Web of Science were unsuccessful. At the very least, the use of troubleshooting activities with middle grades students is rare in peer-reviewed research.

Troubleshooting is an authentic assessment for computer programming courses since it emulates the type of work expected of game designers and computer programmers, requires the application and synthesis of knowledge and skills to find a solution to a problem worth solving, and involves some degree of self-assessment on the part of students to determine when the goals are satisfied.

2. Approach and Method

This evaluation study was completed in conjunction with a five-week implementation of AgentCubes in two middle schools (i.e., ages 12-14) in the Rocky Mountain region of the United States. The settings in which these studies were conducted were afterschool settings involving instructors with extensive expertise in programming and outreach experiences to support student use of technology. This study focused on documenting the impact of student use of AgentCubes on identifiable learning objectives. These objectives included the development of student fluency in information technology such as algorithmic thinking and programming, engaging in sustained reasoning, and managing problems in faulty situations (NRC, 1999). These objectives have also been cited in proposed definitions for computational thinking (Wing, 2006).

2.1. AgentCubes as a programming environment

What distinguishes AgentCubes from other programming environments used with middle grades students is the emphasis on three-dimensional world design, the range of programmable agent behavior, and the ability to determine outcomes of agent interaction within that world. To program agent behavior effectively, students must consider aspects of visualization, probability, scale and proportion. Logic, symbolic notation, inductive and deductive reasoning also underlie the programming syntax in AgentCubes.

The AgentCubes software used during the conduct of this study was a beta-version not publically available at the time of this writing. The precursor to AgentCubes, AgentSheets, included a similar visual programming interface and commands that had been extensively field tested with middle school students. The main difference between AgentSheets and the newer AgentCubes is that the agent interactions in AgentSheets are in a 2D world whereas AgentCubes offers renderings and agent interaction in a 3D world (Ioannidou, Repenning & Webb, 2009).

2.2. Research context

Forty students from two middle schools in the Rocky Mountain region of the United States were recruited for a pilot study of AgentCubes. Instruction occurred during five weekly two-hour sessions scheduled after school in each school’s computer lab. Students recruited from the two schools were of similar ethnic and economic diversity as the urban (Jefferson MS) and suburban (Memorial MS) neighborhoods within which the schools were situated. With respect to gender representation, 19 boys and 21 girls started the study.

A survey administered during the first session found that 23 of the 40 students (58%) reported having their own computer at home. When comparing computer ownership between students at the two middle schools, 13 of 20 students at North Middle School reported having their own computer compared to 10 of 20 students at Centennial Middle School reported owning their own computer.

The first four sessions of instruction focused on student design of a working version of Frogger, a popular arcade game from the 1980s (e.g., http://en.wikipedia.org/wiki/Frogger). The general flow of the sessions involved an overview of the agent world and programming interface, the creation of agents (e.g., frog, trucks, road), and an introduction to common methods to control agent behavior. By the end of the third session, most of the students had created a working version of Frogger. The first half of the fifth session was devoted to assessing what students had learned about game design and computer programming with AgentCubes. Of the original 40 students, only 24 (16 males, 8 females) attended the final session due to family commitments or extra-curricular conflicts.
2.3. Assessment design problem

The assessment design problem that led to the development and use of troubleshooting scenarios was: How can student learning of new software be assessed when students enter the study with no prior experience with the software? Since AgentCubes was a new interface for all participating students and since instructional contact time with students was limited to less than 10 hours, we were reluctant to use pre-assessment tasks during first session involving programming, which could frustrate young students and have a negative impact on their attitudes towards future sessions. Instead, we opted to design a culminating assessment during the final session requiring students to draw upon what they had learned about game design and programming to resolve five problematic scenarios:

1) One type of car was not moving on the highway
2) On one side of the highway, cars were stacking up (they did not disappear from the screen)
3) The movement for the frog in world-view perspective (2D) was incorrect
4) The movement for the frog in first person perspective (3D) was incorrect
5) There were not enough turtles being generated for the frog to make a successful crossing of the river

Students were informed that they were a game designer and had to fix a program with at least five problems. Students were not told what the problems were, so they needed to identify the problem, locate the problematic agent and its related behavior procedures, and correct the program code for the agent. Students were given 45 minutes to “fix the Frogger program.” Students were required to complete the activity on their own and could only ask the instructors questions of clarification.

3. Results

Throughout the troubleshooting session almost all students demonstrated sustained engagement and persistence in resolving the problems. All students were able to identify at least three of the problems with the program and resolve the problem by reprogramming agent behavior. There was only one student, a male student from Memorial MS, who was not able to fix any of the troubleshooting scenarios.

The typical response routine of most students during this activity was: identify the agent that was causing the problem (e.g., car, frog, etc.); click on the problematic agent to bring up the programming screen; scroll through the different procedures for the agent to identify procedure that needed to be fixed; change the problematic procedure; replay the Frogger program to observe if the changed procedure resulted in the desired behavior; if the behavior was not resolved, try a different program code or procedure until the agent demonstrates the desired behavior.

The car generation and frog movement in first person (3D view) were the most challenging scenarios to identify and correct. The programming required to resolve these two issues was less intuitive and required greater abstraction of ways to control agent behavior. However, the results for these two scenarios show that the majority of students were able to correctly troubleshoot the problem. The car generation problem was fixed by 88% of the students (81% male, 100% female), and the 3D frog movement problem was corrected by 75% of the students (88% male, 50% female). The turtle generation scenario was the last problem addressed, and for many students the problem was too subtle to even identify. Turtles were being generated and were moving across the river, there were just not enough turtles for the frog to successfully cross the river. So the turtle generation problem was really one of improving playability and not a major problem with agent behavior. Even so, 42% of the students (50% male, 25% female) were able to identify and solve the turtle generation problem. Six students (25%) went beyond the scope of the activity and improved the program in other ways that were unintended, such as using the graphics tool to change agents so they would be easier to see in first person view.

4. Conclusion

The results suggest that troubleshooting scenarios are worthwhile authentic assessments that can be used to
document student fluency in IT education. Even beyond the assessment purpose, if computer programming is a goal for students the curriculum should provide opportunities to engage in troubleshooting routines, either by design or implicitly supported as an expected part of working with new software. Furthermore, in the search for a working definition and processes that exemplify computational thinking, it is essential to include awareness of troubleshooting as an expected aspect and benefit of computer-based design processes in these discussions. The benefit of designing games, simulations, and products with computers is the rapid and regular feedback offered to the designer as the product is being developed and improved. Understanding the relationships between the commands, programming syntax and the resulting product is critical to design and problem solving with computers.

We observed that students successfully completed most of the tasks and were fully engaged in troubleshooting behavior for up to 45 minutes, even though participants in this study were sixth and seventh grade students with no prior experience with the interface. This suggests that the troubleshooting activity is an appropriate authentic assessment to use with middle grades students, is accessible to a wide range of learners, and may be included relatively early in an instructional sequence (i.e., after only 8 hours of instruction). Very few students demonstrated any outward signs of frustration or reluctance to improve agent behavior, suggesting that this type of assessment activity is worth considering in the instructional programs for other design software.

Designing an assessment to reveal what students have learned about computer programming or new software interfaces necessarily relies more on how students can apply what they have learned rather than showing learning gains between a pre- and post-assessment. Given students lack of familiarity with the AgentCubes interface and the relatively brief contact time with the students, it was important to design an activity that could be motivating to students, have some instructional value, and serve as an assessment of student fluency with computer game design.

Acknowledgements

This work was supported by the National Science Foundation (NSF) under Grant IIP 0712571. Any opinions, fundings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

References