Integrating Intelligent Feedback into Block Programming Environments

Thomas W. Price
North Carolina State University
890 Oval Drive
Raleigh, NC 27606, USA
twprice@ncsu.edu

ABSTRACT

Block Programming Environments (BPEs) are becoming popular tools for introducing novices to programming, due in part to their connection with students' interests in games, apps and stories. This has led to increasing use of BPEs outside of classroom settings, where knowledgeable instructors are not always available. Intelligent Tutoring Systems (ITSs) can keep students on track in the absence of instructors by providing hints and warnings to students in need of help. Further, data-driven techniques can generate this feedback automatically from previous students' attempts at a problem. This research focuses on the integration of this data-driven, ITS-style feedback into a modern BPE and the evaluation of its impact.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computers and Information Science Education—*Computer Science Education*.

Keywords

Block programming, Intelligent Tutoring Systems, Hints.

1. PROGRAM CONTEXT

I am a second year PhD student in the Computer Science department at North Carolina State University, working in the Center for Educational Informatics. I have completed my coursework, as well as the first milestone of my program, the written preliminary exam. As part of the exam, I designed and conducted an experiment to compare the effects of textual and block programming interfaces on middle school students' performance and self-efficacy as they completed an introductory programming assignment. I am currently preparing to propose my dissertation research on providing data-driven, intelligent feedback in Block Programming Environments. I have begun initial research and plan to present my proposal in the spring of 2016 and defend my dissertation in spring of 2017.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author(s). Copyright is held by the owner/author(s).
ICER'15, August 9–13, 2015, Omaha, Nebraska, USA.
ACM 978-1-4503-3630-7/15/08.
DOI: http://dx.doi.org/10.1145/2787622.2787748.

2. CONTEXT AND MOTIVATION

The need for more Computer Science (CS) teachers in K12 schools is well established, with programs such as the NSF's CS10K initiative seeking to train 10,000 high school teachers. Despite these efforts, many students still have little to no access to formal CS education. For these students, the best opportunities to learn CS are informal learning environments, such as after school programs (e.g. [4]) and websites like Code.org that offer self-paced lessons. In these settings, especially in the absence of a trained instructor, it is important that students learn CS in a way that is engaging and supportive.

Block Programming Environments (BPEs), such as Alice, Scratch, MIT App Inventor and Snap are popular tools for introducing CS in informal learning settings [4, 9]. These environments make programming easier by removing syntax errors and the need to memorize procedure names, while allowing students to write code that connects with their interests, such as games, apps and stories. BPEs have been positively evaluated in formal and informal learning settings [2, 4] and have been suggested as tools for broadening participation in Computer Science (CS) [3, 4]. However, these environments generally offer little support to students who get stuck and do not know how to proceed or debug their program. If no instructor is available in these situations, students may give up or lose interest in CS.

Intelligent Tutoring Systems (ITSs) offer a solution to this problem by providing struggling students with automated feedback and guidance. This often takes the form of hints when students are stuck, or warnings if they perform an incorrect step [8]. In fact, this feedback can be generated from past students' solutions to a problem, using data-driven techniques that avoid the need for expensive, hand-authored expert models [7]. This data-driven feedback, in the form of hints, has been successfully generated in many domains, including programming [6].

My research seeks to integrate data-driven, ITS-style feedback into a modern BPE and to evaluate the effect of this feedback on student performance and self-efficacy, in the absence of an instructor.

3. BACKGROUND & RELATED WORK

BPEs have been evaluated in a number of contexts. Dann et al. [2] showed that Alice 3 had great educational benefit in an introductory college CS course, where students later transitioned to a more traditional Java curriculum. Students who took the Alice-to-Java course performed on average a full letter grade higher on the final exam (which was in Java)

than historical students taking the Java-only version of the course. Maloney et al. [4] show that BPEs can also be successful in informal settings. They describe the use of Scratch in an urban after-school center, where students used it voluntarily and frequently, despite having no formal instruction. Students picked up some CS concepts, with about half of their programs employing loops and user interaction.

While BPEs generally lack built-in, intelligent feedback, programming ITSs have provided this support for many years. The ACT Programming Tutor [1] uses a Cognitive Model, consisting of hundreds of expert written, language-specific production rules, which model correct (and buggy) actions within the tutor. These rules are used to interpret student actions, recognize mistakes and offer hints. The Hint Factory [7] offers a data-driven alternative to these hand-constructed models, using correct solutions from previous students to build a graphical model of how students can progress through a given problem. This model is used as the basis for hint-generation, directing students in need of help down correct solution paths. The Hint Factory has been extended to work in the domain of programming [6], where a larger space of possible solutions to any given problem makes the task more difficult.

4. PROBLEM, GOALS & METHODS

My hypothesis is that we can adapt the Hint Factory to generate data-driven, on-demand hints for students working in BPEs, and that these hints will improve students' performance and self-efficacy in the environment. Specifically, my goal is to integrate this functionality into the SNAP[1] programming environment, with a focus on supporting open-ended tasks such as creating games, apps and stories.

As a concrete example, imagine a student working on simple game, similar to Whack-a-Mole, which requires the use of variables, loops, conditionals and calls to a simple API to control the position of sprites onscreen. The student can make the mole appear, but is having trouble randomizing its location. After requesting a hint, the student receives a message instructing her to replace her hard-coded coordinates with a call to a `random` function.

I will start by instrumenting the SNAP environment and collecting detailed snapshots from students, who will also serve as a control group, as they complete an open-ended programming activity without instructor assistance. I will work to adapt the Hint Factory to the complexities of this open-ended domain, drawing on previous work [6], and generate hints from the previously collected data. I will integrate these hints into SNAP and collect data from a second group of students completing the same activity with hints available. I will compare the two groups' performance in the environment, as measured by completion of the activity's subgoals, and their change in self-efficacy regarding computing, measured by pre- and post-surveys.

5. DISSERTATION STATUS

I have collected a preliminary dataset from students working with both block and textual programming interfaces on an open-ended problem. I have compared their performance, and confirmed that the block interface does improve student performance. I have explored the application of existing data-driven techniques to this dataset and identified areas that will require extension [5]. I am currently using these findings to explore possible modifications to the Hint Factory algorithm, and preparing for the initial data collection.

I am in the initial stages of drafting my dissertation proposal, but have not yet started writing the dissertation itself. I intend to complete the initial data collection, analysis and proposal by spring of 2016. The hint generation and evaluation should be completed by the fall of 2016 and the document itself should be finished by spring of 2017.

From this DC I would like guidance on the design of the programming activity, the appropriate setting for data collection, and if there are additional performance measures I should consider collecting. I am also interested in what additional feedback and guidance the programming environment should ideally provide to students, besides on-demand hints.

6. EXPECTED CONTRIBUTIONS

Practically, this work seeks to improve BPEs, common tools for introducing novices to programming, by increasing their accessibility and efficacy outside of a formal learning environment. The data-driven techniques explored here will be important progress toward automated hint generation in ill-defined domains, such as open-ended programming. This work is an important first step towards integrating two successful tools for learning: ITSs and BPEs.

7. REFERENCES

[1] A. T. Corbett. Cognitive Mastery Learning in the ACT Programming Tutor. Technical report, 2000.

[2] W. Dann, D. Cosgrove, and D. Slater. Mediated transfer: Alice 3 to java. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 141–146, 2012.

[3] C. Kelleher, R. Pausch, and S. Kiesler. Storytelling alice motivates middle school girls to learn computer programming. *Proceedings of the SIGCHI conference on Human Computer Interaction*, 2007.

[4] J. Maloney, K. Peppler, Y. Kafai, M. Resnick, and N. Rusk. Programming by choice: urban youth learning programming with scratch. *ACM SIGCSE Bulletin*, 40(1):367–371, 2008.

[5] T. W. Price and T. Barnes. An Exploration of Data-Driven Hint Generation in an Open-Ended Programming Problem. In *Workshop on Graph-Based Data Mining held at Educational Data Mining (EDM)*, 2015, forthcoming.

[6] K. Rivers and K. Koedinger. Automatic generation of programming feedback: A data-driven approach. In *The First Workshop on AI-supported Education for Computer Science (AIEDCS 2013)*, 2013.

[7] J. Stamper, M. Eagle, T. Barnes, and M. Croy. Experimental evaluation of automatic hint generation for a logic tutor. *Artificial Intelligence in Education (AIED)*, 22(1):3–17, 2013.

[8] K. Vanlehn. The behavior of tutoring systems. *International Journal of Artifical Intelligence in Education*, 16(3):227–265, 2006.

[9] A. Wagner, J. Gray, J. Corley, and D. Wolber. Using app inventor in a K-12 summer camp. In *Proceeding of the 44th ACM Technical Symposium on Computer Ccience Education*, 2013.

[1] snap.berkeley.edu