Exploring why novice programmers switch between text and blocks in a dual-modality coding environment

Nathan Holbert, Teachers College, Columbia University, holbert@tc.columbia.edu
David Weintrop, UChicago STEM Education, University of Chicago, dweintrop@uchicago.edu

Abstract: Block-based, graphical programming environments are increasingly becoming the way that novices are being introduced to the practice of programming and the field of computer science more broadly. An open question surrounding the use of such block-based tools is how well they prepare learners for using more conventional text-based programming tools. In an effort to address this transition, new programming environments are providing support for both block-based and text-based programming. In this paper, we present findings from a study investigating how learners use such an environment. Our analysis investigates what modality learners choose to work in, and why and when learners move from one representation to other. We conclude with a discussion of design implications and future directions for this work.

Introduction

In response to the growing interest in learning to program, many introductory programming tools are being designed to be 'low-threshold'—meaning they are intuitive, welcoming, and appeal to diverse audiences. One such approach is block-based programming tools (Figure 1).

Despite widespread adoption, open questions remain about the block-based modality within the larger realm of Computer Science (CS). More specifically, it is unclear whether such tools prepare students for future learning opportunities and how best to transition learners from block-based introductory tools to more conventional text-based languages (Powers et al., 2007). One proposed solution involves the creation of dual-modality tools that allow learners to seamlessly shift back-and-forth between block-based and textual representation (Bau et al., 2015; Dann et al., 2012; Homer & Noble, 2014; Matsuzawa et al., 2015). While recent work has offered insight into supports offered by block-based environments, and in how learners transition from blocks to text, less is known about the particular conceptual resources mobilized by each representation.

In this paper, we use Pencil Code (Bau et al., 2015), a programming environment that allows learners to switch between block- and text-based representations of code, to investigate these questions. Specifically, we explore *why* and *when* learners move from one modality to the other. This paper contributes to our knowledge of how novices make use of block-based programming tools and advances our understanding of the design affordances of the modality.

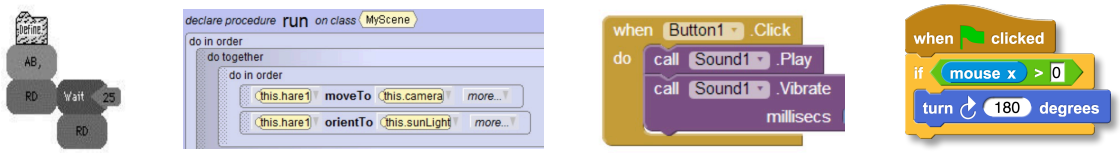

Figure 1. Four examples of block-based programming languages.

Prior Work

Block-based programming environments are designed to ensure novice programmers have early successes by providing visual cues as to how and where commands can be used. Lead

by the popularity of tools like Scratch (Resnick et al., 2009), Alice (Cooper, Dann, & Pausch, 2000), and Blockly (Fraser, 2013), block-based tools are becoming the standard approach for the design of introductory programming tools (Duncan, Bell, & Tanimoto, 2014).

The popularity of block-based tools has led to their incorporation into formal CS education settings (Astrachan & Briggs, 2012; Goode, Chapman, & Margolis, 2012). However, their use has seen mixed results, with some studies reporting successes (Armoni et al., 2015; Dann et al., 2012), while others questioning the suitability of such environments in preparing learners for future CS studies (Meerbaum-Salant, Armoni, & Ben-Ari, 2011; Powers et al., 2007). Further, studies exploring learning in each modality suggest differences with respect to programming comprehension (Lewis, 2010; Weintrop & Wilensky, 2015b), program generation (Price & Barnes, 2015) as well as with perceptions of the power and authenticity of each modality (Weintrop & Wilensky, 2015a).

Methods

In this paper we explore when and why learners of varying experience switch between block-based and text-based modalities in Pencil Code (Bau et al., 2015), an environment that allows users to switch freely between textual or block-based representations of their program (Figure 2). Participants were drawn from two populations of novice programmers who worked with Pencil Code in a formal learning context. One population, recruited from a highly diverse high school class designed to introduce students to computational thinking, consisted of 13 girls (8 freshmen, 2 8th grade, 2 sophomores, and 1 junior). The girls spent three 100-minute classes working through a series of activities designed to introduce them to the basics of computer programming (including concepts like loops and variables).

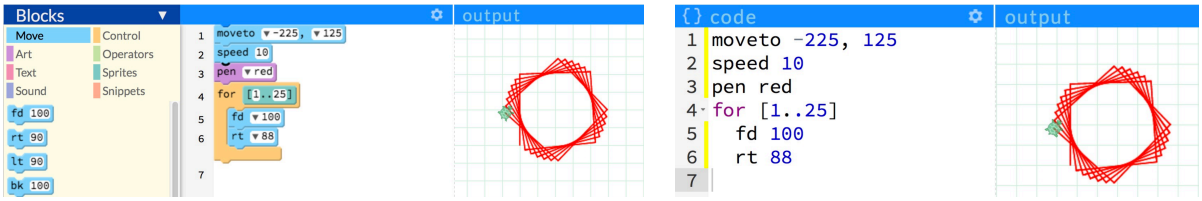

Figure 2. The two interfaces of Pencil Code: text (right) and blocks (left).

The second population that participated in this study includes four girls and six boys enrolled in a graduate level course on the design of educational learning environments (mean age of 29) taught by the first author. Students used Pencil Code as part of a "create a quilt" activity where each student wrote a program to visually represent some characteristic of themselves that, in the end, were "stitched" together with other programs into a larger "quilt." Students began the assignment on the second class of the semester and completed the entire assignment outside of class sessions over one week.

To answer our research questions about why and when students switch between modalities, all actions performed in the Pencil Code environment were logged. These logs provide a unique identifier for the user, a timestamp, the modality being used, the action performed, and the complete program. Using these logs, we can identify when and where the user switched modalities during the course of construction.

Motivations for Shifting Modality

Dual-modality environments provide an opportunity to investigate and understand the affordances provided by block-based interfaces relative to isomorphic text-based tools by

looking at both when shifts occur, as well as what happens immediately after the shift. Two different groups of programmers were evaluated using the Pencil Code environment to complete a basic programming assignment. Despite one group being composed of high school students and the other graduate students, both groups overwhelmingly used the block representation when coding (92% and 91% respectively).

To determine ways block-based representation might help learners overcome programming challenges, we focused on log events that were captured when students toggle the interface from text to blocks. The logs contained 217 instances of this transition, each containing a snapshot of the program from each modality. By comparing the two events, we were able to determine the specific programmatic changes made after switching from text to the blocks modality. We then coded the changes made using a scheme that identified the type of change and blocks involved.

After transitioning from text-to-blocks, there are a number of next steps learners could take, including adding a new command, deleting some portion of the program, moving blocks within the program, or simply returning to the text modality. While learners shifted to the block representation for a variety of reasons, our analysis indicates that 65.4% of these events were to add commands to their program. Two-thirds (67.1%) of these code block additions involved adding a block-type that had not been previously used in that program. The high frequency of the addition of previously unused block after a text-to-blocks transition indicates the block representation supported learners in adding new, never-before-used commands to their programs. On the one hand, this suggest learners may be using "drawer" afforded by the block-based modality to browse the available set of commands. Alternatively, users may be relying on the block representation to avoid accidental syntax errors (Maloney et al., 2010). Finally, it is possible that some students may simply prefer dragging-and-dropping commands into their programs over the act of typing, which suggests that ease-of-composition motivated the transition.

Given that students frequently transition from text to blocks in order to add a new type of block to their program, we can gain insight into what commands are challenging or have difficult syntax by analyzing the types of new blocks that were added. Looking at the block type added after a text-to-blocks transition, the most frequently added blocks were from the *movement* (30.5%), *art* (21.6%), and *control* (18.0%) categories. As the assignments asked students to write code to move a turtle to create visually interesting patterns, it's perhaps not surprising that movement and art blocks were frequently used. While control blocks were used slightly less frequently, we found that 86.7% of the time one was added; it was added for the first time. First-add move blocks and art blocks were 62.8% and 63.9% respectively. The reliance on the block representation to add control blocks, which include commands like repeat, if/else, and while, further indicates the value of blocks in overcoming syntax challenges. However, as these commands also involve complex non-linear processes, the block representation may also be providing a conceptual support as learners attempt to incorporate these complex ideas into their programs.

Along with adding new blocks, other actions students took after toggling include deleting existing blocks (15.6%), moving existing blocks to new locations in the program (13.3%), or other events like toggling back to text or temporarily removing commands from the program (a combined 5.7% of post-toggle actions). While these actions were less frequent, and potentially less revealing than the patterns we found for adding new blocks, patterns within these actions do point to further affordances of the block representation. For example, when students transitioned

from text-to-blocks and then proceeded to move code in their program, 60.7% of the time the move included shifting the scope of the moved code. In the blocks interface, shifting scope means moving the block into, or out of a block that has a nesting shape, such as a conditional or iterative block. Here the visual depiction of scope afforded by the blocks may provide a conceptual resource for learners as they attempt to leverage the non-linearity of code.

Discussion

To understand how the block-based modality supported novice programmers as they authored their programs, we analyzed the contents and changes of program snapshots that occurred when users shifted from the text modality to blocks. This analysis revealed that when students that coded in the text modality returned to the block representation, they did so mostly to add new code. When student made this move, they were usually adding commands that they had not yet used—commands they were unfamiliar with—in their program. The fact that 65.4% of the blocks added were being added for the first time suggests that the block modality supported users in finding new commands for use in their program. Furthermore, of the command types added, complex control blocks (such as for loops and if statements) were often added (86.7%) for the first time during this text to blocks shift. This reliance upon the block-based modality may indicate that users were either unable to or hesitant to add commands that may introduce syntax errors in their programs, which can be particularly tricky for control commands, or that the blocks may serve a conceptual function as learners attempt to incorporate non-linear commands into their programs.

Along with illuminating patterns in how novices learn to program in dual-modality programming environments, this work also has potential design implications for the creation of low-threshold/high-ceiling programming environments. On the low-threshold end of the spectrum, the finding that students transition to the block-based modality to add a new command or to change the scope for existing commands, suggests that including similar features into text-environments, such as a "drawer" of possible/useful code that can easily be added to in-process programs, could be useful for novices just becoming comfortable with text-based programming (Kölling, Brown, & Altadmri, 2015). As for ensuring a high-threshold, our findings suggest a dual-modality design means at the least, blocks don't restrict the text-based programmer, and at the best, blocks provide new opportunities for exploration and experimentation to enhance or extend text-based programs. By allowing learners to choose which modality they want to work in, novices who need additional support can leverage the various scaffolds designed into block-based tools, while students with more experience or who are particularly eager to learn text-based coding can do so. Further, the dual modality approach gives learners control of their own learning experience, deciding for themselves about what scaffolds they want or when they might need more support.

Conclusion

As interest in learning to program continues to grow, new interfaces and tools are being designed to make the practice engaging and accessible. In this paper, we explore how novice programmers use Pencil Code, a tool that provides text and block-based representations of code, to understand how access to both modalities impacts programming practices. By studying how and when student move back and forth between block-based and text-based interfaces, we advance our understanding of the affordances of the tools for helping novices. Our findings indicate novice programmers leveraged the opportunity to switch between modalities to overcome challenges throughout their programming experience. Regardless of the modality chosen, all students were able to fully participate in the course and complete the programming

activities, showing the effectiveness of the dual modality approach for welcoming and supporting novices, while also keeping more experienced programmers engaged. By looking at how and when students chose to shift modalities, we can gain insight into the types of supports each modality offers. While much of the discussion in introductory programming as been in trying to understand which is better for learners – blocks or text, this paper shows the answer may be: why not both?

REFERENCES

Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From Scratch to "Real" Programming. *ACM Transactions on Computing Education (TOCE)*, *14*(4), 25:1-15.

Astrachan, O., & Briggs, A. (2012). The CS principles project. *ACM Inroads*, *3*(2), 38–42.

Bau, D., Bau, D. A., Dawson, M., & Pickens, C. S. (2015). Pencil Code: Block Code for a Text World. In *Proceedings of the 14th International Conference on Interaction Design and Children* (pp. 445–448). New York, NY, USA: ACM. http://doi.org/10.1145/2771839.2771875

Begel, A. (1996). *LogoBlocks: A graphical programming language for interacting with the world.* Electrical Engineering and Computer Science Department. MIT, Cambridge, MA.

Brown, N. C. C., Mönig, J., Bau, A., & Weintrop, D. (2016). Future Directions of Block-based Programming. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (pp. 315–316). New York, NY, USA: ACM. http://doi.org/10.1145/2839509.2844661

Cooper, S., Dann, W., & Pausch, R. (2000). Alice: a 3-D tool for introductory programming concepts. *Journal of Computing Sciences in Colleges*, *15*(5), 107–116.

Dann, W., Cosgrove, D., Slater, D., Culyba, D., & Cooper, S. (2012). Mediated transfer: Alice 3 to Java. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education* (pp. 141–146). ACM.

Dorn, B., & Elliott Tew, A. (2015). Empirical validation and application of the computing attitudes survey. *Computer Science Education*, *25*(1), 1–36. http://doi.org/10.1080/08993408.2015.1014142

Duncan, C., Bell, T., & Tanimoto, S. (2014). Should Your 8-year-old Learn Coding? In *Proceedings of the 9th Workshop in Primary and Secondary Computing Education* (pp. 60–69). New York, NY, USA: ACM. http://doi.org/10.1145/2670757.2670774

Fraser, N. (2013). *Blockly*. https://developers.google.com/blockly/: Google.

Goode, J., Chapman, G., & Margolis, J. (2012). Beyond curriculum: the exploring computer science program. *ACM Inroads*, *3*(2), 47–53.

Homer, M., & Noble, J. (2014). Combining Tiled and Textual Views of Code. In *IEEE Working Conference on Software Visualisation (VISSOFT)* (pp. 1–10). Victoria, BC: IEEE. http://doi.org/10.1109/VISSOFT.2014.11

Kölling, M., Brown, N. C. C., & Altadmri, A. (2015). Frame-Based Editing: Easing the Transition from Blocks to Text-Based Programming. In *Proceedings of the Workshop in Primary and Secondary Computing Education* (pp. 29–38). New York, NY, USA: ACM. http://doi.org/10.1145/2818314.2818331

Lewis, C. M. (2010). How programming environment shapes perception, learning and goals: Logo vs. Scratch. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (pp. 346–350). New York, NY.

Maloney, J. H., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, *10*(4), 16.

Matsuzawa, Y., Ohata, T., Sugiura, M., & Sakai, S. (2015). Language Migration in non-CS Introductory Programming through Mutual Language Translation Environment. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (pp. 185–190). ACM Press. http://doi.org/10.1145/2676723.2677230

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2011). Habits of programming in Scratch. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education* (pp. 168–172). Darmstadt, Germany: ACM.

Powers, K., Ecott, S., & Hirshfield, L. M. (2007). Through the looking glass: teaching CS0 with Alice. *ACM SIGCSE Bulletin*, *39*(1), 213–217.

Price, T. W., & Barnes, T. (2015). Comparing Textual and Block Interfaces in a Novice Programming Environment (pp. 91–99). Presented at the ICER '15, ACM Press. http://doi.org/10.1145/2787622.2787712

Resnick, M., Silverman, B., Kafai, Y., Maloney, J., Monroy-Hernández, A., Rusk, N., … Silver, J. (2009). Scratch: Programming for all. *Communications of the ACM*, *52*(11), 60.

Weintrop, D., & Wilensky, U. (2015). To Block or Not to Block, That is the Question: Students' Perceptions of Block-based Programming. In Proceedings of the 14th International Conference on Interaction Design and Children (pp. 199–208). New York, NY, USA: ACM.

Weintrop, D., & Wilensky, U. (2015b). Using Commutative Assessments to Compare Conceptual Understanding in Block-based and Text-based Programs. In Proceedings of the Eleventh Annual International Conference on International Computing Education Research (pp. 101–110). New York, NY, USA: ACM.