Panel: Future Directions of Block-based Programming

Neil C. C. Brown
School of Computing
University of Kent
Canterbury, CT2 7NF
UK
nccb@kent.ac.uk

Jens Mönig
Communications Design
Group (CDG), SAP Labs
1025 Westwood Blvd.
Los Angeles, CA, USA 90024
jens@moenig.org

Anthony Bau
Phillips Exeter
Academy
NH, USA
dab1998@
gmail.com

David Weintrop
(moderator)
Northwestern University
Evanston, IL, USA 60208
dweintrop@
u.northwestern.edu

1. SUMMARY (DAVID WEINTROP)

Blocks-based programming is becoming the way that learners are being introduced to programming and computer science. Led by the popularity of tools like Scratch, Alice, and Code.org's Hour of Code activities, many new programming environments and initiatives are employing the blocks-based modality. This trend can be seen in the growing number of classroom computer science curricula incorporating blocks-based environments into their materials. Despite this rise in use, many open questions remain surrounding blocks-based programming. In this panel, we discuss the current state of blocks-based programming environments, review what we know about learning with blocks-based tools, and look to the future, discussing what form next-generation blocks-based, or blocks-inspired, programming environments might take.

Research looking at blocks-based programming is revealing that modality matters: that the representations used to present programming concepts affect learners' conceptual understanding [6], programming practices [3], and perceptions of programming and computer science [5]. This panel brings together leading designers and researchers looking to advance graphical, blocks-based programming through new, innovate designs. The panel will open with a review of current research literature on learning with blocks-based programming and then continue with presentations of three recently designed blocks-based programming environments (Greenfoot 3, GP, Pencil Code), each of which look to push the boundaries of the approach in different directions. These short presentations will frame the discussion of pertinent questions facing designers and educators who use blocks-based programming environments. As part of the panel discussion, the following questions with be explored:

Why do blocks-based programming environments work? Blocks-based tools weave together a myriad of features to produce accessible, engaging programming environments. The color and shape of commands, the organization and easily navigable way blocks are displayed, and the drag-and-drop composition mechanism all contribute to supporting novices in writing successful programs. Drawing on research and our own experiences, we discuss why the current generation of blocks-based tools look the way they do, and what are the key features of successful block environments.

What are the challenges associated with designing blocks-based editors? In the creation of a programming environment, countless design decisions are made, including aspects of the interface and layout, questions of language design, and consideration of how the blocks-based programming component will interact with the runtime environment. In answering this question, we discuss the tensions and trade-offs inherent in creating blocks-based programming environments, and discuss elusive features we have yet to figure out how best to incorporate into blocks-based tools.

What are the challenges we face in bringing blocks-based programming into the classroom? With this question we explore technological, pedagogical, and perceptual challenges of teaching with blocks-based programming in formal settings. This includes confronting the perception that blocks are not "real" programming and other hurdles towards adoption of blocks-based tools in formal contexts.

What are the pros and cons of blocks-based programming (especially relative to text-based alternatives)? Research is revealing that the blocks-based modality can support learners' conceptual understanding, help novices advance more quickly, and engage underrepresented populations with computer science. At the same time, text still retains some advantages over blocks. In addressing this question we reflect on the current state of blocks-based environments and discuss how we as designers, researchers, and educators might address the current limitations.

What are the gaps between blocks- and text-based programming? How can they be bridged by design or pedagogy? A major open question is if, and how, concepts and practices learned in introductory blocks-based environments transfer to more conventional text-based languages. Research has reported both successful and unsuccessful transfer. With this question we explore the gaps that exist between the two modalities and discuss potential design solutions to address them.

What is the future of blocks-based programming? This final question looks to the future, thinking about what blocks-based programming might look in the near and not-so-near future. As part of this question we discuss the long-term role of blocks-based programming and whether or not we think blocks-based (or blocks-inspired) environments will supplant text-based programming.

These questions, along with questions from the audience, will be explored as part of this panel session.

SIGCSE '16 March 02-05, 2016, Memphis, TN, USA

ACM ISBN 978-1-4503-3685-7/16/03.

DOI: http://dx.doi.org/10.1145/2839509.2844661

2. NEIL BROWN (GREENFOOT 3)

Neil Brown, Michael Kölling and Amjad Altadmri are designing the new Greenfoot 3 "frame-based" editor [1]. Frame-based editing is intended to combine the best parts of block-based programming and text-based programming. Frames are very similar to blocks, but crucially, keyboard entry is supported by use of a "frame cursor" which occupies the space between frames and allows single keypress insertion of new frames. The cursor also allows for easy selection and manipulation of frames in a similar way to text. For example, holding shift while moving the cursor creates a frame selection, which can be cut/copied/pasted. This keyboard support is accompanied by many further usability improvements to better support intermediate and experienced users.

The frame-based language in Greenfoot 3 is named Stride. Stride is very similar to Java, with a near-identical semantic model, and allows full use of Java libraries. Where most block based systems have a finite set of commands, all of which are shown on a potentially overwhelming palette, Stride has a single, generic "method call" frame which can be used to call any Java method, without complicating the set of available frames. A classic code completion interface helps users select available methods.

Neil and his colleagues believe that blocks-based programming, as enhanced in their frame-based programming, has the potential to surpass text-based programming in terms of usability – that by adding keyboard support and further improvements for program manipulation, readability and navigation, it is possible to bring together the best of blocks- and text-based programming into a single unified system.

3. JENS MÖNIG (GP AND SNAP!)

Jens Mönig, John Maloney, and Yoshiki Ohshima are developing GP, a portable, extensible, general purpose blocks programming language for casual programmers [4]. The GP class library and programming environment are written in GP itself, so all code can viewed, edited, and debugged as blocks. So far, the GP developers have used conventional text editors on the textual representation of GP code as a way to bootstrap the system, but we hope to develop techniques to allow us – and advanced GP users – to evolve and extend the GP system using only blocks-based editing tools.

Blocks languages offer many advantages to the beginner or "casual" programmer. They eliminate syntax issues, allow the user to work with logical program chunks, provide affordances such as drop-down menus, and leverage the fact that recognition is easier than recall. However, as users gain experience and start creating larger programs, they encounter two inconvenient properties of pure blocks languages: blocks take up more screen real-estate than textual languages and dragging blocks from a palette is slower than typing.

We will describe two techniques that address these problems in GP. The first technique, which addresses the screen-real estate and code readability (and "skim-ability") issues, is to switch to an alternate block rendering style that removes block outlines and colors, changes fonts, and shrinks the vertical and horizontal spacing. The result looks and reads like text (in the same amount of screen space), yet retains the underlying structure and affordances of blocks.

The second technique explores ways to input and edit blocks using only the keyboard. This effort was initially inspired by an interest in making GP accessible to users with visual or physical impairments, but we quickly realized that keyboard-based block editing benefits all GP users. Like Greenfoot's frame editor, GP's keyboard block editor introduces a "block input cursor" that can be moved around using navigation keys. To insert a new block at the cursor, the user starts typing any word that appears on the desired block. As they type, the system shows a menu of the top five matching blocks. It usually takes just a few keystrokes to select a given block from GP's current palette of approximately 250 blocks. Keyboard entry avoids the overhead of selecting a category in the blocks palette, finding the block in that category, dragging it out, and dropping it into the correct location – thus saving a great deal of time.

4. ANTHONY BAU (PENCIL CODE)

Anthony Bau is developing Droplet, a new block editor used in David Bau's Pencil Code environment and Code.org's Applab. Droplet is a block-based editor that can load and edit code in text languages. This provides many of the advantages of blocks (a palette, less typing, assistance with syntax, and a visualization of the program) while allowing students to use text language runtimes and participate in text language communities.

Currently, block languages generally fall short of text languages in expressiveness and in the maturity of their developer communities. There is also a disconnect in students' minds between block programming and their perception of "real" programming [2]. These problems could be solved by creating more advanced block languages; however, they could also be solved by bringing the benefits of block languages to existing text languages. Droplet is designed to do the latter, by allowing students to work in blocks with real-world text programs in mainstream languages, and to switch between blocks and text at any time.

Even as visual languages improve, Anthony believes that text languages will always be a staple of computer science education. Block editing for text programs, like Droplet, will be a way for students and professionals to take advantage of the simplicity of blocks while learning to code in text.

5. REFERENCES

[1] M. Kölling, N. C. C. Brown, and A. Altadmri. Frame-based editing: Easing the transition from blocks to text-based programming. In *WiPSCE '15*, pages 29–38, 2015.

[2] C. M. Lewis. How programming environment shapes perception, learning and goals: Logo vs. scratch. In *SIGCSE '10*, pages 346–350, 2010.

[3] O. Meerbaum-Salant, M. Armoni, and M. Ben-Ari. Habits of programming in scratch. In *ITiCSE '11*, pages 168–172, 2011.

[4] J. Mönig, Y. Ohshima, and J. Maloney. Blocks at your fingertips: Blurring the line between blocks and text in GP. In *Blocks and Beyond workshop*, 2015.

[5] D. Weintrop and U. Wilensky. To block or not to block, that is the question: Students' perceptions of blocks-based programming. In *IDC '15*, pages 199–208, 2015.

[6] D. Weintrop and U. Wilensky. Using commutative assessments to compare conceptual understanding in blocks-based and text-based programs. In *ICER '15*, pages 101–110, 2015.