Work in Progress: A Nonvisual Interface for a Blocks Language

Varsha Koushik and Clayton Lewis
Department of Computer Science
University of Colorado, Boulder
vasr6678@colorado.edu,
clayton.lewis@colorado.edu

Abstract
Visual programming systems, including blocks languages like Scratch (Maloney et al., 2010) are widely used to introduce children and other learners to programming, but in their present form they cannot be used by blind people. Following up work on the nonvisual dataflow language Noodle (Lewis, 2014), and building on the Blockly language system (https://developers.google.com/blockly/; Fraser, 2015) we are building a nonvisual interface to a blocks language. We present a pseudospatial interface for the language, similar to that of Noodle, and compare it with other approaches to accessibility for blocks languages.

1. Introduction
There is a surge of interest in introducing programming to children, through initiatives like the BBC micro:bit in the UK (https://www.microbit.co.uk/), and the international Hour of Code program (https://hourofcode.com/us). Some of the most popular platforms for this work are *blocks languages*, in which program elements are presented as blocks, somewhat like jigsaw puzzle pieces. Blocks have tabs and sockets that suggest how the elements can be assembled. The constraints in program grammar are conveyed as rules of fitting shapes together, rather than as syntax rules, so that the often-frustrating syntax errors in textual languages are eliminated. Examples of blocks languages include Scratch (https://scratch.mit.edu/), Snap (http://snap.berkeley.edu/), MIT App Inventor (http://appinventor.mit.edu/), Microsoft Block Editor for the BBC micro:bit (https://www.microbit.co.uk/create-code), and many more.

Unfortunately these languages can't be used by learners who can't see, an issue with visual languages generally. While one might think that using visual languages is an inherently visual activity (as the name suggests), in fact there are approaches to delivering the conceptual benefits of these languages, that is, the logical structure and meaning of programs, in nonvisual representations. Some of these approaches derive from the work of T.V. Raman, who has suggested that many, if not all, visual tasks can be substituted by nonvisual ones (Raman, 1996; Raman and Gries, 1997; see also Lewis, 2013.) Lewis presented work on a nonvisual dataflow language, Noodle, based on Raman's ideas, at PPIG 2014 (Lewis, 2014).

2. Approaches to accessibility for blocks languages

2.1. Screen reader access
A common approach to accessibility for blind users is support for screen reader programs, software tools that render textual material displayed on a computer screen as synthesized speech, and support interaction with common controls and widgets. A team at Google is creating Accessible Blockly, (https://blockly-demo.appspot.com/static/demos/accessible/index.html), a system that presents blocks and blocks programs as HTML structures that can be read by a screen reader. This approach has the advantage that some blind users are skilled screen reader users, and are thus already familiar with all the controls and conventions needed to navigate and manipulate the HTML. Stephanie Ludi (2015) has proposed a related approach, also based on Blockly, that will also support visual access.

Schanzer's Bootstrap system (http://www.bootstrapworld.org/) is being extended to support a blocks language made accessible by a screen reader in a different way (Emmanuel Schanzer, personal communication, July 15, 2016). The Bootstrap language is normally presented textually in a Web page, in a form that a screen reader can access. By using alternative CSS style rules these textual programs can be presented as structures of blocks, and can be edited in that form (though the system does not use shape constraints to guide program creation.)

2.2. Beyond the screen reader

As Raman (1996) points out, the screen reader approach starts with a presentation of content intended for sighted people, and seeks to provide nonvisual access to that presentation. But it may be possible to provide a different presentation, nonvisually, that is superior to the visual presentation, in some respects. For example, Lewis (2014) suggested that a nonvisual dataflow language can make it easier to trace relationships between program elements than the lines used in visual presentations.

This argument supports the investigation of presentations that are not based on presentations for sighted users, but it does not rule out the use of screen readers. The representation presented by the screen reader can be structured differently from that shown to sighted users. That is, the screen reader can be used as a means of navigating and manipulating a representation that need not be the same as what sighted users work with. This is true of Accessible Blockly, in that its HTML representation includes many controls that are not presented to sighted users.

2.3. Pseudospatiality

Screen reader navigation of HMTL is serial and hierarchical. That is, one moves up and down a list of elements, some of which are sublists that can be entered and navigated as well; the sublists can contain subsublists, and so on. Commands are provided to read all material, or to read only elements at a given level, to skip to elements of a specified type, and more. While this system is quite flexible, some designers of applications for blind users have moved outside this structure to support navigation in a virtual two dimensional structure, with operations provided that move right to left or up and down. For example, the TWBlue Twitter client (http://twblue.es/) uses arrow keys to navigate an invisible two-dimensional array of content. Noodle (Lewis, 2014) uses a somewhat similar scheme. (It is sometimes suggested that spatial arrangement would be meaningless for blind users, but this is simply not true. Spatial relationships are fundamental in a vast range of nonvisual activities, including walking around, reaching for objects, and so on.)

In systems of this kind feedback for navigation and manipulation is provided by synthetic speech. For example, the system will speak a description of a content element when the user moves to it. The synthetic speech can be provided by the system itself (so-called *self voicing* applications) or, if the user normally uses a screen reader, the speech can be produced by the screen reader (by placing data in a “live region” that the screen reader monitors.)

A navigation scheme can be *pseudospatial*, rather than simply spatial, if the geometry of movement is distorted. In TWBlue, moving “left” from a content element might take one to the top of the column to the left, rather than to an element directly to the left. In Noodle, following an edge “to the right” might go to an element that could also be reached by moving “up”, if there is a loop in the dataflow.

We describe below Pseudospatial Blocks (PB), a blocks language presentation based on arrow key navigation. In PB a toolbox of blocks is to the “left” of a workspace (as is common); blocks in either area, and parts within blocks, are arranged “vertically”. The arrangement is pseudospatial in that (for example) moving one step “down” from a block, or moving one step “down” in a list of parts of the same block, would not reach the same location.

2.4. Tactile presentation

Richard Ladner (personal communication, June 1, 2016) has proposed using a touchscreen to permit learners to explore and operate on blocks by touch. Because many blind children are familiar with touchscreen devices and the screen reader programs used with them, this presentation should be easy for children to learn to use. More radically, it may be possible to create blocks languages with physical blocks; see e.g. http://cubescoding.com/ and https://www.primotoys.com/ for beginnings.

2.5. Presenting fit constraints

As mentioned earlier, a distinctive feature of blocks languages is the use of shape fitting to indicate how program elements can legally be put together. For example, a block that produces a value has a different shape from a block representing a statement; two statement blocks fit together to form a sequence, but a value block will not fit there. How can this information be presented nonvisually?

In Ladner's tactile system shape information will still be presented, but will be detected by touch, as children explore the outlines of the blocks with their fingers, receiving auditory feedback when they touch the block. Ladner proposes to enlarge the blocks to make it easier to explore their outlines.

In Accessible Blockly and PB shape information is not presented as such. Rather, the systems enforce the placement rules that the shapes would convey: a block of a given kind can only be placed where it is legal (see more below on PB's treatment of his issue.) In this respect these designs both follow Raman's suggestion that presentations for blind users should not be aimed at conveying the information presented visually to sighted users, but should focus on the underlying content.

3. Pseudospatial Blocks (PB)

3.1 Design and implementation

We based the design of PB on an analysis of user tasks in Scratch (the most widely used blocks language). We aimed to support these tasks, without requiring a screen reader, using common keyboard commands: arrow keys to navigate, space to select, ENTER to open (or move inside) a complex entity, and period to operate. Responses to all commands are provided in synthetic speech.

Like Scratch, PB presents a toolbox of program elements, navigated by up and down arrows. Selecting a block causes a copy to be added at an insertion point in the workspace. Moving to the right takes one into the workspace, where up and down arrows allow one to move among the blocks in the program. Pressing the period key executes the program. For each key press the user receives audio feedback about the current location in the program, delivered as synthesized speech.

Some blocks, for example a block for repeating a sequence of operations, have statement blocks nested within them. To create such a structure the user navigates to the outer block and then presses ENTER; the up and down arrows then move among the elements of the block that can be edited, including the place where nested statements can be inserted. Pressing ENTER at this point establishes the insertion point there, so that selecting a statement block in the toolbox will add a nested statement.

When choosing a block to insert, the user is only presented with block choices that are legal for insertion at the current insertion point. For example, only statement blocks, and not expressions, can be inserted within a repeat block. This distinction is enforced in visual blocks languages by shape constraints. Similar operations support other forms of editing. For example, if a statement includes a dropdown menu of alternatives (such as musical notes), one ENTERs that statement, navigates to the dropdown, ENTERs that, navigates the alternatives, and selects the desired one.

The implementation of PB is based on the Blockly library (https://developers.google.com/blockly/; Fraser, 2015), an extremely flexible and widely used platform for creating blocks languages. Blockly supports an xml representation of blocks programs, and can generate code for these in a number of languages. This support makes it easy to create new interfaces like Accessible Blockly or PB. PB is implemented as a Web application that builds a JSON (JavaScript Object Notation) representation of programs, and then uses Blockly to generate JavaScript code.

3.2 Status of PB

PB is a running proof of concept program that supports a small number of block types, most of which produce sounds, adequate to create very simple melodies. We have focussed on programs for producing sounds for the obvious reason that the results are easy for blind learners to understand, whereas common visual activities, such as producing animations or turtle graphics, are themselves inaccessible, apart from the accessibility of the language used to specify them. Music may also have conceptual benefits as an application domain for learners, with rich structure; see e.g. Blackwell and Collins, 2005; Shapiro et al., 2016.

Currently, hand coding is required to support new blocks, even when these have been defined for other Blockly applications, but in future we hope to be able to generate the representations needed by PB automatically from Blockly's representations. With that improvement in hand we plan to support a wider range of operations on sound. We will then undertake user testing and iterative redesign.

4. Discussion

Creating accessible blocks languages may have impact in a number of ways, beyond the obvious value of supporting blind children learning to code. As Jamal Mazrui (remarks at M-Enabling Summit, June 2012) and others have observed, making programming more accessible to people with disabilities will allow them to satisfy their technology needs themselves, rather than relaying on other people to understand and respond to their requirements. Nonvisual programming interfaces may be more broadly useful in allowing people to program on small screen devices, something difficult today.

In pursuing these aims we should cast a wide net, as represented by the three approaches described above. We are far from understanding the tradeoffs in this design space, so we should pursue, and compare, multiple lines of work. Here are just some of the important issues we need to understand:

How important is it that interfaces used by blind children can also be understood by their sighted classmates? Screen-reader based approaches, using representations very similar to those provided for sighted users, may have an edge there, as may Ludi's design. How important is it to support people who are blind but do not use screen readers? This is common today among older people who become blind, but may less often be an issue for children. When and how can nonvisual interfaces be superior to visual ones? In particular, can the use of shapes to express constraints on the assembly of program elements be improved upon, by providing easy access to elements that fit a selected insertion point? Can non-visual interfaces support increasingly complex programs, as learners mature?

5. Acknowledgements

We thank Sina Bahram for telling us about TWBlue, and for suggestions for structuring pseudospatial interfaces; Richard Ladner for sharing his proposal for tactile support for blocks languages; Neil Fraser and Madeeha Ghori of Google for their support for our use of Blockly, and sharing their work on Accessible Blockly; Ben Shapiro and Annie Kelly for advice on platforms; and Andy Stefik and Brian Harvey for convening a workshop on accessibility of Blockly at MIT in April, 2015, that provided the inspiration for our work. We thank the PPIG reviewers for helpful suggestions.

6. References

Blackwell, A., & Collins, N. (2005). The programming language as a musical instrument. *Proceedings of PPIG05* (Psychology of Programming Interest Group), 3, 284-289.

Fraser, N. (2015). Ten things we've learned from Blockly. In *Blocks and Beyond Workshop (Blocks and Beyond), 2015 IEEE* (pp. 49-50). IEEE.

Lewis, C. (2013) Pushing the Raman principle. In *Proceedings of the 10th International Cross-Disciplinary Conference on Web Accessibility* (W4A '13). ACM, New York, NY, USA, Article 18, 4 pages.

Lewis, C. (2014). Work in Progress Report: Nonvisual Visual Programming. In B. duBoulay and J. Good (Eds.) *Proc. PPIG 2014 Psychology of Programming Annual Conference*, 25th Anniversary Event. Brighton, England, 25th-27th June 2014.

Ludi, S. (2015). Position paper: Towards making block-based programming accessible for blind users. *Blocks and Beyond Workshop (Blocks and Beyond), 2015 IEEE*, Atlanta, GA, 2015, 67-69.

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

Raman, T.V. (1996) Emacspeak-- A speech interface. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '96), Michael J. Tauber (Ed.). ACM, New York, NY, USA, 66-71.

Raman, T.V. and Gries, D. (1997.) Documents mean more than just paper! *Mathematical and Computer Modelling*, Volume 26, Issue 1, July, 45-53.

Shapiro, R.B., Kelly, A., Ahrens, M., and Fiebrink, R. (2016) BlockyTalky: A Physical and Distributed Computer Music Toolkit for Kids. In Proc. NIME'16, July 11-15, 2016, Griffith University, Brisbane, Australia.