An Accessible Blocks Language: Work in Progress

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

ABSTRACT
Block languages are extensively used to introduce programming to children. They replace the complex and error prone syntax of textual languages with simple shape cues that show how program elements can be combined. In their present form, blind learners cannot use them, because they rely on graphical presentation of code, and mouse interactions. We are working on a nonvisual blocks language called Pseudospatial Blocks (PB), that supports program creation using keyboard commands with synthetic speech output. It replaces visual shape cues for language syntax, the key feature of block languages, with filtering of program elements by syntactic category.

CCS Concepts

• **Human-centered computing~Accessibility systems and tools** • Human-centered computing~Auditory feedback • **Software and its engineering~Visual languages**

Keywords
Accessible Interfaces; Barriers to programming; Learning to program

1. VISUAL PROGRAMMING ENVIROMENTS ARE INACCCESSIBLE TO BLIND USERS
Block languages like Scratch, Snap, MIT App Inventor, Microsoft Block Editor for BBC, and many more are among the most popular platforms for introducing children to the world of programming. Program elements are presented as blocks, which have tabs and sockets that show how they can be fit together, so that learners don't have to master the complex syntax of textual languages. However, non-sighted users cannot see these shape cues (nor can they perform the drag and drop mouse interactions used to assemble the blocks.)

A number of researchers have responded to the need to offer the potential conceptual benefits of block languages in a form accessible to blind learners. 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, a tool with which many blind users are already familiar. Stephanie Ludi [4] has proposed a related approach, also based on Blockly, which will also support visual access, an advantage in many learning situations where blind and sighted learners work together. The Bootstrap system (http://www.bootstrapworld.org/) is being extended to support a blocks language made accessible by a screen reader (Emmanuel Schanzer, personal communication, July 15, 2016). Richard Ladner (personal communication, June 1, 2016) has proposed using a touchscreen to permit learners to explore and operate on blocks by touch. Like Ludi's work, these approaches may allow the same system to be used by blind and sighted learners. There are also projects aimed at coding with tangible, physical blocks; see e.g. http://cubescoding.com/, https://www.primotoys.com/, and https://projectbloks.withgoogle.com.

ASSETS '16, October 23-26, 2016, Reno, NV, USA
ACM 978-1-4503-4124-0/16/10.
http://dx.doi.org/10.1145/2982142.2982150

2. PSEUDOSPATIALITY – BEYOND THE SCREEN READER
Our approach to creating an accessible blocks language is based on the ideas of T.V. Raman, who suggests that many visual tasks can be replaced by nonvisual ones ([6], [7]; see also [2].) In Raman's thinking, one should create nonvisual presentations of content that work well in themselves, rather than seeking to make visual representations accessible, as a screen reader does.

Screen reader navigation is serial and hierarchical. Commands are provided to read all material, or to read only elements at a given level, or to skip to elements of a specified type, which provides some flexibility, but the underlying structure is still constrained. Some designers of applications for blind users have moved outside this structure to support navigation in a virtual two dimensional space, with operations provided that move right to left or up and down (for example, the TWBlue Twitter client, http://twblue.es/).

A navigation scheme can be *pseudospatial*, rather than simply spatial, if the geometry of movement is distorted. We describe here 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. The blocks in both areas, and parts within blocks, are arranged "vertically". The arrangement is pseudospatial in that a step "left" from a block in the workspace will reach the same location in the toolbox, regardless of the "vertical" position of the block in the workspace, and there are other distortions of actual space. Lewis [3] presented work on a nonvisual dataflow language with a similar liberal use of two-dimensional space.

Schiff and colleagues (see e.g. [9]) have shown that people navigating virtual environments have no greater difficulty learning environments that are distorted or impossible geometrically or even topologically than realistic ones (in a topologically impossible environment one can move from the outside to the inside of a closed contour without crossing the contour, for example.) We conjecture that distortions in the geometry of PB will not be an issue.

3. SHAPE CONSTRAINTS IN PB

As mentioned earlier, the distinctive feature of blocks languages is the use of block shape to indicate how blocks can legally be assembled. For example, a block representing a statement has a different shape from a block representing an expression. Statement blocks can fit together to make sequences of statements, but expression blocks cannot; similarly, expression blocks can fit into holes representing the inputs to operations, while statement blocks cannot fit there. How can these constraints on block placement be handled in a system with no visual presentation?

In PB, one constructs a program by selecting an insertion point, where a new block is to be placed. The system then prompts the user to select the desired block from a list of candidates, in the toolbox of available blocks. But the list of candidates is filtered, so that only legal candidates can be selected. For example, in the situations just compared above, if one selects an insertion point in a sequence of statements, only statement blocks are presented as candidates, while selecting an insertion point where an expression is legal gives only expression blocks as candidates.

Arguably, this selection scheme has advantages over shape-based constraints, in two ways. First, shapes have difficulty indicating when two or more different kinds of blocks can be used. For example, in Scratch, blocks that represent numeric quantities are shown as rounded shapes, and character strings are rectangles. But a rounded shape can legally be placed in a rectangular hole, even though it does not "fit". The numeric quantity is converted to a string, in such a situation, but there is no visual cue that this will happen until it is tried. Using filtered selection, this is no problem: all legal blocks are offered as candidates.

A second issue with shapes is that each kind of block that has distinctive fit constraints has to have a distinctive shape. But it seems that only so many shapes are workable, so some distinctions among blocks may go unmarked. For example, in the demo language at https://blockly-demo.appspot.com/static/demos/code/index.html, list blocks and number blocks have the same shape, but some list operations cannot be applied to numbers, and so number blocks will not always fit where their shape suggests. Distinctive colors are used for list and number blocks, which helps, but only for sighted users.

This situation is unproblematic for filtered selection. For example, PB has blocks representing sounds and numbers; in many situations only one or the other can be inserted, and the filtered selection process offers only legal candidates.

4. IMPLEMENTATION AND STATUS

The implementation of PB is based on the Blockly library (https://developers.google.com/blockly/; [1]), 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, converts this to XML, and then uses Blockly to generate JavaScript code. PB communicates with users via synthetic speech; it will also be possible support screen reader users by using an ARIA active region to generate responses via the screen reader.

4.1 Status of PB

PB supports a small number of statement types, for arithmetic, control structure, and sound. Sounds and operations on them are supported by two kinds of blocks. First there are sound statement blocks that simply play fixed, associated sounds. As statements, these blocks can be formed into sequences to play desired melodies. Second, there are sound expression blocks, that represent sounds (as lists of samples) but do not play them; they are played by providing them as input to a "play sound" statement block. Sound expression blocks support generating musical notes, attenuating or amplifying a sound (by multiplying the samples by a given number), concatenating two sounds, or mixing them, that is, playing the sounds simultaneously. As discussed earlier, the filtered selection mechanism enforces that (for example) only sounds and not numbers can be mixed. We believe PB is nearing readiness for user testing, which is obviously crucial. We expect many changes to be driven by user input.

5. ACKNOWLEDGMENTS

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 help in our use of Blockly, and for 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 block languages at MIT in April, 2015, that provided the inspiration for our work.

6. REFERENCES

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

[2] 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.

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

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

[5] 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.

[6] Raman, T.V. (1996) Emacsspeak—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.

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

[8] Zetzsche, C., Wolter, J., Galbraith, C., & Schill, K. (2009). Representation of space: image-like or sensorimotor?. *Spatial Vision*, *22*(5), 409-424.