Making the Blockly Library Accessible via Touchscreen

Logan B. Caraco, Sebastian Deibel, Yufan Ma, Lauren R. Milne
Macalester College Mathematics, Statistics, and Computer Science Department
Saint Paul, MN, USA
{lcaraco, sdeibel, yma, lmilne}@macalester.edu

ABSTRACT
Block-based programming environments are a popular way to learn programming. Many of these libraries, including Scratch and MIT's App Inventor, are built on the Blockly library from Google. Unfortunately, programs built with the Blockly library are currently not accessible for people with visual impairments. We describe two designs to make the Blockly library accessible using a screen reader on a touchscreen device.

Author Keywords
Accessibility; blind users; aria; block-based programming environment; education; block languages; audio interfaces

ACM Classification Keywords
• **Human-centered computing~Accessibility technologies**

INTRODUCTION
Block-based programming environments (BBPEs) allow one to drag and drop puzzle-like "blocks" of code together to construct programs. These programs depend on visuals and gestures and are generally inaccessible for children with visual or motor impairments [7]. As computer science lessons become more commonplace in K-12 education, the issue of poorly accessible BBPEs is even more pressing.

In this paper, we describe the prototypes we are building to make the web-based Blockly library, which underlies many popular BBPEs such as Scratch [8] and the Code.org curriculum [13], accessible to as wide an audience as possible. There are concurrent complementary efforts on making these environments accessible using keyboard navigation [2,10]. We chose to focus on making Blockly accessible when using a touchscreen device and hope to have our code integrated into the core Blockly library.

Toward this goal, we designed and prototyped two models of interaction: (1) a *Hierarchical List Design*, which linearizes the block program into a navigable list, and (2) a *Spatial Design*, which preserves the original 2-D rendering of the blocks, but which annotates the blocks to make them accessible. Our code is freely available on GitHub at https://github.com/SybelBlue/macblockly.

ASSETS '19, October 28–30, 2019, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-6676-2/19/10.
https://doi.org/10.1145/3308561.3354589

RELATED WORK
The majority of the existing work on accessible BBPEs has focused on making these environments accessible through keyboard navigation by converting the block code structure into a hierarchical list that can be navigated using a screen reader [2,5,6,9,10]. There has also been work in creating accessible BBPEs that use tangible blocks [4,11]. For touch accessibility, there is our own work on Blocks4All, a BBPE designed to be accessible using a screen reader on a touchscreen device [7]. Our work on Blockly builds on this previous work and explores applying it to the more complex Blockly library. Because there is evidence that many students with visual or motor impairments learn to use the assistive technology on touchscreen devices before learning on desktop computers [1], we explore ways to make the Blockly library accessible using iPads and Android tablets.

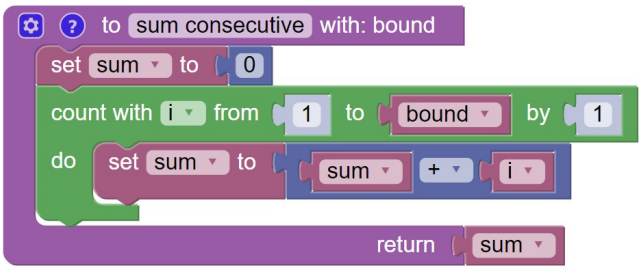
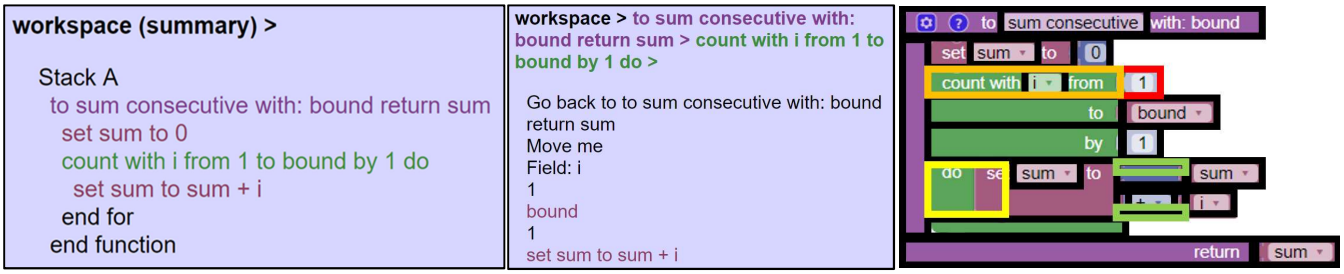


Figure 1: An unmodified Blockly function that returns the sum of natural numbers less than or equal to bound.

DESIGN
We describe how our interfaces can be interacted with via touch on a tablet with a touchscreen screen reader. Android tablets and iPads come with built-in screen readers (TalkBack [14] and VoiceOver [3], respectively), which can be used to explore the "focusable" elements on the screen by either touching where the focusable elements are displayed on the screen or by iterating over them. This is achieved via keyboard accessible gestures that allow users to select, scroll, or swipe through certain user-selected types of elements.

Without a screen reader, our version of Blockly functions identically to the original. We wrote in JavaScript without the use of any additional external libraries. We followed the WCAG 2.0 AA standards, labeling items with appropriate ARIA descriptions and roles [12]. We tested our designs on the Android and iOS operating systems with the Safari and Chrome browsers.



(A) (B) (C)

Figure 2: (A) Hierarchical List Design: The main navigation list displaying the same workspace as Figure 1. (B) The block-specific view after selecting the "count with i from 1 to bound by 1" block. (C) Spatial Design: The rectangles show the interactable fields labeled with the text on the block/field (e.g. the orange box reads "count with i from", and the red reads "1, editable field") or contextual information (e.g. the yellow reads "inside for block", the green read "left-" and "right-parenthesis").

Hierarchical List Design

The *Hierarchical List Design* linearizes the workspace into a navigable list that mirrors the 2-D workspace.

Reading

The main navigation list is designed to allow users to quickly read a workspace of blocks (Figure 2A). At the highest hierarchical level (the workspace), the main navigation list is a complete summary of all blocks. The list is placed on the edge of the screen to make it easier to find non-visually, and list items extend across the width of the list so they can be quickly scanned by touch. Block list items are organized into sub-lists using the nesting structure of the blocks, and all blocks are part of lettered 'stacks'. The ARIA-label of a block item is constructed from data pulled from the block.

Editing

Upon selecting a block list item, the list navigates to a block-specific view (Figure 2B) and focuses on the single block represented by the item. In this view, (1) the parent navigation bar (a breadcrumb trail displaying the current scope of the code) is updated according to the location of the block, (2) the option to move the block is displayed, (3) block list items which navigate to the block's parent and children are shown, and (4) all block-specific options are shown.

Block-specific views contain the full set of controls for editing a block's properties. These options show any warnings the block may be visually indicating. They also include any controls to mutate the block (e.g. add "else/else-if" statements to "if" blocks). Lastly, the options provide ARIA-labeled toggles to change any dropdown or text field values embedded in the block.

Moving

Movement is designed to mirror the drag and drop method. The user first selects a block in the block-specific view and then selects a destination. Connection items only appear after a move is begun and are visible anywhere the block may be placed. They appear on the hierarchical level and position in the list where the resulting move would place the block.

Spatial Design

The *Spatial Design* preserves the 2-D block structure of the workspace. The layout is almost identical to Blockly's current layout, with rectangles overlaid on each row and editable field of a block. Each of these rectangles is given an ARIA-label to make it interactable with a screen reader.

Reading

In order to read the code, the user can either touch a block (useful if they have some vision) or swipe through the blocks and fields via screen reader. After a block is focused, the user can swipe to the editable fields in that row (Figure 2C). The "swiping" order of the blocks and fields corresponds to how they would be read from top-to-bottom, left-to-right. To improve the navigability of the workspace, we added an "End of Block" row to any block with nested statements and added "left parenthesis" and "right parenthesis" annotations around any logic or arithmetic operator blocks.

Editing

Once the user navigates to an editable field or connected block, they can edit its content by selecting it. In order to improve selectability of blocks with a screen reader on, we use Blockly's external inputs option to organize connected blocks vertically in rows rather than in-line. Fields are edited via either user keyboard input or dropdown list, depending on the field type. Although dropdown lists are selectable with a screen reader, they are challenging to use without sight. We are working to improve their usability.

Moving

This is a work-in-progress. We plan to have users move blocks via a "select-select-drop" method. After selecting a block, a list of valid connection points for that block will be added to the 2-D representation of the code in the workspace. The user can then select one of the points to move the block.

CONCLUSION

We present two interfaces which were designed to give the Blockly library inheritable accessible features to allow users to manipulate and read code swiftly. We plan to test these prototypes with students with visual impairments.

ACKNOWLEDGMENTS

Thanks to Professor Paul Cantrell and the Blockly Team at Google for their valuable input. This project was funded by a CS-ER grant from Google LLC.

REFERENCES

[1] Catherine M. Baker, Lauren R. Milne, and Richard E. Ladner. 2019. Understanding the Impact of TVIs on Technology Use and Selection by Children with Visual Impairments. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems - CHI '19*.

[2] Google Inc. Accessible Blockly. Retrieved from https://blockly-demo.appspot.com/static/demos/accessible/index.html

[3] Google Inc. Android Accessibility Overview. *Android Developers*. Retrieved June 27, 2019 from https://developer.android.com/guide/topics/ui/accessibility

[4] Varsha Koushik, Darren Guinness, and Shaun K. Kane. 2019. StoryBlocks: A Tangible Programming Game To Create Accessible Audio Stories. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (CHI '19), 492:1–492:12. https://doi.org/10.1145/3290605.3300722

[5] Varsha Koushik and Clayton Lewis. 2016. An Accessible Blocks Language: Work in Progress. In *Proceedings of the 18th International ACM SIGACCESS Conference on Computers and Accessibility* (ASSETS '16), 317–318. https://doi.org/10.1145/2982142.2982150

[6] Clayton Lewis. 2014. Work in Progress Report: Nonvisual Visual Programming. *Psychology of Programming Interest Group*. Retrieved from http://users.sussex.ac.uk/~bend/ppig2014/14ppig2014_submission_5.pdf

[7] Lauren R. Milne and Richard E. Ladner. 2018. Blocks4All: Overcoming Accessibility Barriers to Blocks Programming for Children with Visual Impairments. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (CHI '18), 69:1–69:10. https://doi.org/10.1145/3173574.3173643

[8] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. 2009. Scratch: Programming for All. *Commun. ACM* 52, 11: 60–67. https://doi.org/10.1145/1592761.1592779

[9] Emmanuel Schanzer, Sina Bahram, and Shriram Krishnamurthi. 2019. Accessible AST-Based Programming for Visually-Impaired Programmers. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (SIGCSE '19), 773–779. https://doi.org/10.1145/3287324.3287499

[10] Stephanie Ludi. 2015. Position paper: Towards making block-based programming accessible for blind users. In *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*, 67–69. https://doi.org/10.1109/BLOCKS.2015.7369005

[11] Nicolas Villar, Cecily Morrison, Daniel Cletheroe, Tim Regan, Anja Thieme, and Greg Saul. 2019. Physical Programming for Blind and Low Vision Children at Scale. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems* (CHI EA '19), INT003:1–INT003:4. https://doi.org/10.1145/3290607.3313241

[12] w3c_wai. Web Content Accessibility Guidelines (WCAG) Overview. *Web Accessibility Initiative (WAI)*. Retrieved June 27, 2019 from https://www.w3.org/WAI/standards-guidelines/wcag/

[13] Code.org Computer Science Curriculum for Grades K-5. *Code.org*. Retrieved July 8, 2019 from https://code.org/student/elementary

[14] Vision Accessibility - iPad. *Apple*. Retrieved July 8, 2019 from https://www.apple.com/accessibility/ipad/vision/