Position: Accessible Block-Based Programming: Why and How

Lauren R. Milne
Department of Mathematics, Statistics and Computer Science
Macalester College
Saint Paul, MN, USA
lmilne@macalester.edu

Richard E. Ladner
Paul G. Allen School of Computer Science and Engineering
University of Washington
Seattle, WA, USA
ladner@cs.washington.edu

Abstract—**Block-based programming environments are very popular for introducing children to programming. Unfortunately, they are not accessible to many children with visual or motor impairments. In this paper we outline why block-based environments should be made accessible for these children, describe current efforts to make environments accessible, and describe how developers can incorporate accessibility into their own block-based programming environments.**

Keywords—accessibility, motor impairments, visual impairments, aria, user interface design

I. Introduction

There is little question that block-based programming environments (BBPEs) are a huge success internationally in getting children engaged with programming and computational thinking [1]. Scratch alone has more than 40 million projects on its website [2]. Although there are BBPEs for Wonder Workshop robots and Arduino, the primary outputs for BBPEs are audiovisual animations. Several groups of children are not served well by BBPEs, namely, those who are blind and those who have mobility related disabilities. Generally, BBPEs are not screen reader accessible for blind children both in terms of creating programs and viewing their outputs. Children with limited use of their hands are not able to create and edit programs because standard interactions like drag-and-drop are not accessible to them. Generally, BBPEs are not switch accessible for children with severely limited use of their hands.

The purpose of this position paper is to encourage the creators of BBPEs to reengineer their BBPEs to be accessible in order to meet the needs of all children, including those with disabilities. Building on the 2015 position paper by Stephanie Ludi [3], we describe current research around the accessibility of BBPEs and provide suggestions for developers to implement accessibility in their own BBPEs.

II. Demographics

According to the National Center for Education Statistics (NCES, Table 204.50) about 7 million US children, ages 3 to 21, are served under the Individuals with Disabilities Education Act (IDEA), Part B. This is about 13.7% of all children in the US in this age group. Of these, about 27,000 have visual impairments. A smaller number have mobility related disabilities that hamper their use of the mouse and keyboard and may necessitate using switch access. In addition to these students under IDEA, there is another group of students with disabilities, about one million that are served under Section 504 of the Rehabilitation Act. An unknown number of these may have visual impairments or mobility related disabilities. Under these laws a school district that does not provide proper access for a child with a disability risks civil actions or law suits to remedy the lack of access. Our position on access rests on equity not possible penalties.

III. Equity

Many of us who work in K-12 computer science education believe that equity is a major goal of our work [4]. To us, equity means providing maximum opportunity and necessary supports for children to participate in computer science courses and non-school activities to all children regardless of gender, race, ethnicity, disability, and other factors.

The majority of children with disabilities are educated in mainstream classrooms (close to 95% of students with disabilities attend regular schools) [5]. Designing BBPEs to be universally accessible would allow these students to have equal access to the same curriculum that their peers are using and allow them to collaborate and share their work with their peers.

Many of the underlying concepts which make BBPEs easier for students without disabilities to learn would likely extend to children with disabilities as well. For example, it is nearly impossible to make syntax errors in BBPEs. This is because blocks are only allowed to be placed in syntactically valid locations, and there are no semicolons or braces (structural cues that can be especially challenging to keep track of with a screen reader). Additionally, having a toolbox with all the possible blocks allows one to search through valid statements and expressions in the language instead of having to remember the names and syntax. Finally, the code structure of the program is communicated spatially, which makes it easier to understand the nesting structure of the code. It may seem like this spatial information would not be useful for people with visual impairments, but current research has looked at conveying this spatial structure with screen readers through touchscreen devices [6] or keyboard navigation [7]–[9]. In fact, Schanzer et al. actually created a tool to convert text-based code into block-based code to improve navigation for blind programmers [10].

IV. Tools for Accessibility

Fortunately, there are accessibility supports in place on Windows PCs and Apple computers, as well as Android and Apple tablets, for blind and visually impaired users (screen readers and magnification) and for users with mobility related disabilities (switch access and other supports). Some of these are built into Windows PC and Apple computers and some are provided by third parties. To create an accessible native BBPE developers need to follow the accessibility guidelines and tools provided for the different operating systems and for web-based content [11]–[14].

A. *Screen readers*

A screen reader is a software application that allows a user to read and navigate a computer screen without vision. They can be used on both computers and touchscreen-based smartphones or tablets. Screen readers can come built-in on devices (e.g. VoiceOver for macOS and iOS, TalkBack for Android) or are available through third-party developers (e.g. JAWS and NVDA for Windows PCs). Someone using a screen reader can use the keyboard or touch to change the focus of a screen reader to different elements on the screen. The items that are focused on by the screen reader are either read aloud or read on a refreshable Braille display. If the focused item is interactive (e.g. a button, link, adjustable control) the user can interact with it via various keyboard commands or touch gestures. This means it is important that any application or document provide structured information that can easily be navigated by the screen reader (e.g. headers in a webpage or document) and clearly give information about what interactive elements are (e.g. button or slider), as well as give an accessible way to interact with them.

B. *Magnification Software*

For people with visual impairments who still have limited vision, there are magnification tools. For most operating systems (including tablets), there are built-in zoom controls that allow one to zoom in using all or part of the screen. There are also third-party tools such as ZoomText, which gives the user a great deal of flexibility in setting the magnification and has screen reading capabilities [15], and Fusion, which is a combination of the JAWS screen reader and ZoomText magnifier [16]. To ensure an application works for someone who has low vision, developers should ensure that the application works with these magnification tools, that there is sufficient color contrast and that text-size can be increased.

C. *Switch Control and Keyboard Navigation*

There are many types of technology that can be used to allow someone with motor impairments to use computers [17]. Many of the devices allow users to interact with a computer or tablet using a standard or adaptive keyboard in place of a mouse. As keyboard navigation can be used by both people who are blind and those who have motor impairments, it is important for developers to create alternatives for interactions that rely on mouse or touch input (e.g. drag and drop). Someone with very limited use of their hands might use different types of single-switch controls to interact with a computer or tablet. For example, a commonly used switch is a large button that someone with limited hand mobility could press with their head. This type of switch control allows a user to give limited (potentially only binary) input to a computer, so it works with software on the computer to use this input to select or interact with elements on the screen. For example, on iOS, you can set up switch control to cycle over the "focusable" elements on the screen, and you can use the switch to select an item when it is in focus. To make an application work with switch control, it is important for developers to ensure that the elements on the screen can be read by the Accessibility API, and that they are presented in a navigation order that makes sense.

Support from the National Science Foundation Grant nos. CNS-1738252 and CNS-1738259.

V. PROGRESS ON ACCESSIBLE COMPUTING ENVIRONMENTS

A. *Born Accessible*

There has been some notable progress over the past ten years in making computer science more accessible, but very little on making BBPEs accessible. One approach to accessibility in programming environments is to make it accessible from the get-go, that is, make the environment what we call "born accessible." Examples of born accessible programming environments are Apple's Swift Playgrounds [18], Microsoft Research's Code Jumper (formally Torino) [19], [20], and StoryBlocks [21]. Swift Playgrounds has a hybrid programming environment that includes a drag-and-drop method and text-entry method for creating and editing programs. The output is an animation, but can be audio-described for a blind user. Code Jumper is a physical environment for creating programs using pods that connect together. The output for Code Jumper is strictly auditory: music, stories, and jokes. StoryBlocks is a BBPE that uses physical blocks to create and edit programs. The program created with blocks is captured with a camera and interpreted to yield a working program on a separate computer.

The Quorum programming language [22] is a text-based language that was born accessible. Its syntax and semantics have been tested for learnability using randomized controlled experiments. As a result, the syntax is far less arcane than most programming languages, making it less intimidating when first encountered. For example, the language has no curly braces, no semicolons, and key words and phrases use a simple vocabulary. A new IDE for Quorum will be announced in fall 2019 that is screen reader accessible.

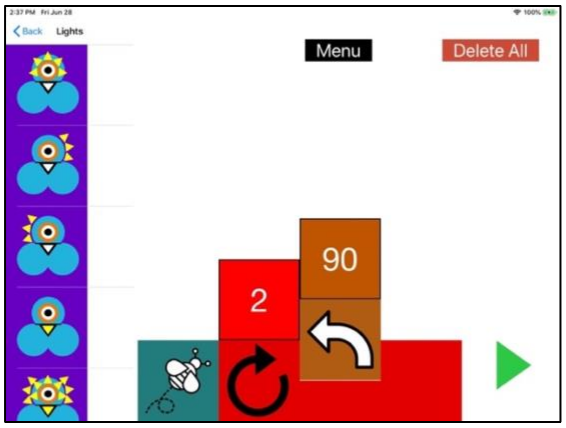

Figure 1: Blocks4All Programming Environment. The code reads: Make Buzzing Noise. Then the robot will turn left 90 degrees 2 times. On the left side of the screen, the menu of blocks to change the robot's lights is shown.

One born accessible BBPE, called Blocks4All, has been developed and studied recently by the authors (Fig. 1) [6]. The environment is implemented for iPads, with the output being commands to a Dash robot [23]. We chose the iPad because its popular VoiceOver screen reader that already familiar to many blind children. We chose a robot output because it is tactile and auditory (moves and makes sounds). The programming interface replaces inaccessible drag-and-drop with a more accessible way to create and edit programs.

B. *Made To Be Accessible*

It can be challenging to reengineer an existing BBPE to make it accessible. For example, Stephanie Ludi started a project several years ago to make a programming environment for Blockly accessible [7]. Blockly is a library from Google for building beginner-friendly block-based programming languages, which can be integrated into programming environment. Her approach, aimed at the standard desktop Blockly programming environment, is to define keyboard shortcuts to help create, edit, and navigate Blockly programs. These shortcuts must be compatible with standard screen reader shortcuts. The project is still on-going. Other examples educational environments that were not born accessible, but are in the process of becoming so, are Bootstrap [10] and PHET [24]. Both of these efforts have been major undertakings. The Blockly team at Google is working toward incorporating keyboard navigation in their next release and also plan to add screen reader compatibility, thereby making any BBPEs built using the library much closer to being accessible. Additionally, Schanzer et al. have released a toolkit that can be integrated into any programming environment built on the CodeMirror library to generate a navigable, accessibility-enabled abstract syntax tree view of code [10].

VI. Making an Accessible Environment

The first step in making an accessible environment is to adhere to the current standards for accessibility: web-based BBPES should follow the WCAG 2.0 AA standards [14], and native tablet applications should follow the Apple [12] and Android [13] accessibility guidelines. At a minimum, this means that they should ensure (1) that all the elements are perceivable by technology such as a screen reader (e.g. blocks should be properly labeled and (2) that it is possible to perform all operations using a keyboard (e.g. they should provide an alternative to drag and drop). These changes should make their BBPE more accessible for someone with motor impairments or some vision. To ensure that the BBPE is fully accessible for someone who is blind, it is important to make sure that the screen reader navigates the environment in an understandable manner and gives good feedback. This can be challenging, and we describe some of the challenges that we have encountered below in our work.

A. *Blockly Touchscreen Environment*

In earlier work on Blocks4All, a born-accessible touchscreen environment [6], we discuss some of the challenges that exist around making BBPEs accessible for people with visual and motor impairments. We focus on the three main ones:

1. accessing elements (e.g. accessing the blocks and input fields),
2. presenting structural information about the blocks (e.g. the nesting structure of the program), and
3. moving blocks

Below, we briefly discuss our current work on making touchscreen interaction with the Blockly library more accessible, and our approaches to overcoming these problems. We chose to focus on the interaction with Blockly on a touchscreen device based on our previous work with Blocks4All and to complement other research [7]. We discuss some of the challenges we have encountered “adding on” accessibility to an existing web-based environment.

B. *Accessing Elements*

In Blockly, blocks are represented as Scalable Vector Graphics (SVG) elements. These can be made accessible to screen readers with WAI-ARIA tags; however, the irregular shape of the blocks leads to some challenges using this approach. First, selecting an SVG element with a screen reader on leads to a pointer event in the “center” of the element. In the case of structural blocks such as for loops or conditional statements, this will often result in selecting a contained block or empty space. Second, the irregular shape of blocks makes it difficult to find blocks and determine their relationship without sight. Because of these challenges, we are experimenting with using other elements to represent the blocks (e.g. rectangles overlaid on top of the blocks or list elements placed elsewhere which can more easily be navigated with a screen reader).

C. *Presenting Structural Information*

The structure of blocks-based programs is traditionally conveyed via the spatial layout of the blocks (e.g. contained blocks are placed inside other blocks). Other researchers have explored using a screen reader with a keyboard to navigate the hierarchical structure much like a hierarchical list [7], [10]. We are exploring ways to explore this hierarchy using a touchscreen; however, the irregular shapes of the blocks make it difficult to understand complex code. Once again, we are exploring ways to change the representation of the blocks presented to the screen reader to make this easier.

D. *Moving Blocks*

The drag and drop technique traditionally used to construct block-based programs is challenging to perform with a screen reader. In our and other researchers’ work [6], [7], this is replaced with a two-part process, in which you select (1) a block and (2) a connection point in order to place the block. This means that connection points (i.e. the puzzle tabs) must be made “selectable” with a screen reader. Additionally, it has to be clearly communicated by the screen reader when a block or connection point has been selected.

VII. Conclusion

In this position paper, we argue that BBPE developers should consider making their environments accessible, and we present current research on making these environments accessible. As we acknowledge that this can be challenging, we highly recommend hiring people who know how to do this, as opposed to using just consultants. Additionally, the people with the most expertise around these access technologies are people with disabilities. We recommend adding people with disabilities to your team.

VIII. Acknowledgments

Our work on the Blockly library is funded by a CS-ER grant from Google LLC and NSF grant number CNS-1738252.

References

[1] D. Bau, J. Gray, C. Kelleher, J. Sheldon, and F. Turbak, "Learnable Programming: Blocks and Beyond," *Commun ACM*, vol. 60, no. 6, pp. 72–80, May 2017.

[2] "Scratch - Imagine, Program, Share." [Online]. Available: https://scratch.mit.edu/. [Accessed: 27-Jun-2019].

[3] S. Ludi, "Position paper: Towards making block-based programming accessible for blind users," in *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*, 2015, pp. 67–69.

[4] Richard E. Ladner and Maya Israel, "'For all' in 'computer science for all,'" *Commun. ACM*, vol. 59, no. 9, pp. 26–28, Aug. 2016.

[5] U.S. Department of Education, National Center for Education Statistics, "Digest of Education Statistics, 2017," NCES 2018-070.

[6] L. R. Milne and R. E. Ladner, "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*, New York, NY, USA, 2018, pp. 69:1–69:10.

[7] S. Ludi and M. Spencer, "Design Considerations to Increase Block-based Language Accessibility for Blind Programmers Via Blockly," *J. Vis. Lang. Sentient Syst.*, vol. 3, no. 1, pp. 119–124, Jul. 2017.

[8] V. Koushik and C. Lewis, "An Accessible Blocks Language: Work in Progress," in *Proceedings of the 18th International ACM SIGACCESS Conference on Computers and Accessibility*, New York, NY, USA, 2016, pp. 317–318.

[9] "Accessible Blockly Demo." [Online]. Available: https://blockly-demo.appspot.com/static/demos/accessible/index.html. [Accessed: 27-Jan-2019].

[10] E. Schanzer, S. Bahram, and S. Krishnamurthi, "Accessible AST-Based Programming for Visually-Impaired Programmers," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, New York, NY, USA, 2019, pp. 773–779.

[11] Microsoft, "Developing accessible applications – Windows app development." [Online]. Available: https://developer.microsoft.com/en-us/windows/accessible-apps. [Accessed: 27-Jun-2019].

[12] "Accessibility - Apple Developer." [Online]. Available: https://developer.apple.com/accessibility/. [Accessed: 27-Jun-2019].

[13] "Accessibility overview," *Android Developers*. [Online]. Available: https://developer.android.com/guide/topics/ui/accessibility. [Accessed: 27-Jun-2019].

[14] w3c_wai, "Web Content Accessibility Guidelines (WCAG) Overview," *Web Accessibility Initiative (WAI)*. [Online]. Available: https://www.w3.org/WAI/standards-guidelines/wcag/. [Accessed: 27-Jun-2019].

[15] "ZoomText Magnifier - zoomtext.com." [Online]. Available: https://www.zoomtext.com/products/zoomtext-magnifier/. [Accessed: 27-Jun-2019].

[16] "Fusion Professional - zoomtext.com." [Online]. Available: https://www.zoomtext.com/products/zoomtext-fusion/. [Accessed: 27-Jun-2019].

[17] "WebAIM: Motor Disabilities - Assistive Technologies." [Online]. Available: https://webaim.org/articles/motor/assistive. [Accessed: 27-Jun-2019].

[18] "Swift Playgrounds - Apple." [Online]. Available: https://www.apple.com/swift/playgrounds/. [Accessed: 27-Jun-2019].

[19] "Code Jumper." [Online]. Available: https://codejumper.com/. [Accessed: 27-Jun-2019].

[20] C. Morrison *et al.*, "Torino: A Tangible Programming Language Inclusive of Children with Visual Disabilities," *Human–Computer Interact.*, vol. 0, no. 0, pp. 1–49, 2018.

[21] V. Koushik, D. Guinness, and S. K. Kane, "StoryBlocks: A Tangible Programming Game To Create Accessible Audio Stories," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, New York, NY, USA, 2019, pp. 492:1–492:12.

[22] "The Quorum Programming Language." [Online]. Available: https://quorumlanguage.com/. [Accessed: 27-Jun-2019].

[23] "Wonder Workshop | Home of cue, Dash and Dot, robots that help kids learn to code." [Online]. Available: https://www.makewonder.com/. [Accessed: 27-Jun-2019].

[24] T. L. Smith, C. Lewis, and E. B. Moore, "A Balloon, a Sweater, and a Wall: Developing Design Strategies for Accessible User Experiences with a Science Simulation," in *Universal Access in Human-Computer Interaction. Users and Context Diversity*, 2016, pp. 147–158.