# Blocks4All: Overcoming Accessibility Barriers to Blocks Programming for Children with Visual Impairments

**Lauren R. Milne, Richard E. Ladner**
Paul G. Allen School of Computer Science and Engineering, University of Washington
Seattle, WA 98195-2350
{milnel2, ladner}@cs.washington.edu

## ABSTRACT
Blocks-based programming environments are a popular tool to teach children to program, but they rely heavily on visual metaphors and are therefore not fully accessible for children with visual impairments. We evaluated existing blocks-based environments and identified five major accessibility barriers for visually impaired users. We explored techniques to overcome these barriers in an interview with a teacher of the visually impaired and formative studies on a touchscreen blocks-based environment with five children with visual impairments. We distill our findings on usable touchscreen interactions into guidelines for designers of blocks-based environments.

### Author Keywords
Accessibility; Computer science education; Visual impairments; Blocks-based programming environments.

### ACM Classification Keywords
H.5.2. Information interfaces and presentation: User Interfaces.

## INTRODUCTION
Recently there has been a big push to incorporate computer science education in K-12 classrooms. As part of this effort, blocks-based programming environments, such as Scratch [15] and Blockly [5] have become very popular [2]. These blocks-based environments use a puzzle-piece metaphor, where operations, variables and constants are conceptualized as "blocks", puzzle-pieces that can be dragged and dropped into a program. These blocks will only snap into place in a location if that placement generates syntactically correct code. Because these environments remove the syntax complexities, they can be a good choice for an introduction to programing and are used heavily in curricular materials and outreach efforts for K-12 education; for example, 60 of the 72 computer-based projects for pre-reader through grade 5 on the Hour of Code website use blocks-based environments [27].

Unfortunately, these environments rely heavily on visual metaphors, which renders them not fully accessible for students with visual impairments. As these students are already underrepresented and must overcome a number of barriers to study computer science [17,21], it is important that they have equal access to curriculum in primary schools, at the start of the computer science pipeline.

To help with this goal, we evaluated existing blocks-based environments to answer the following research question:

*RQ1: What are accessibility barriers in existing blocks-based programming environments for people with visual impairments?*

We identified five main accessibility problems, and built Blocks4All, a prototype environment where we implemented various means to overcome these barriers using a touchscreen tablet computer. We chose to implement this as an Apple iPad application, as iPads have a built-in screen reader and zoom capabilities, making them accessible for children with visual impairments. We worked with a teacher of the visually impaired (TVI) and 5 children with visual impairments who used Blocks4All to determine the usability of these techniques and answer the following questions:

R*Q2: What touchscreen interactions can children with visual impairments use to identify blocks and block types?*

*RQ3: What touchscreen interactions can children with visual impairments use to build blocks programs?*

*RQ4: What touchscreen interactions can children with visual impairments use to understand the spatial structure of blocks program code?*

Our contributions are:

(1) an understanding of accessibility barriers in blocks-based environments for children with visual impairments,

(2) design guidelines for designing blocks-based environments and touchscreen applications for children with visual impairments, drawn from interviews and formative testing with children and a TVI, and



*CHI 2018*, April 21–26, 2018, Montreal, QC, Canada

ACM 978-1-4503-5620-6/18/04$15.00
https://doi.org/10.1145/3173574.3173643

(3) the Blocks4All application itself, as the source code and application are freely available.[1]

## RELATED WORK

We discuss research in making computer science more accessible for students with visual impairments and in making touchscreen interfaces more accessible.

### Accessible Computer Science Education

Several researchers have explored making computer science education more accessible, although most the work has been for text based environments [13,14,19–21]. While these tools make text-based programming easier and more accessible for people with visual impairments; they are all designed for older people and are not as usable for young children who may still be developing literacy skills.

Researchers have also looked at developing tools to introduce programming to younger children. Thieme et al. describe the development of Torino, a physical computing language, which consists of "instruction beads" that can be joined together to create programs, including literal loops and threads [23].

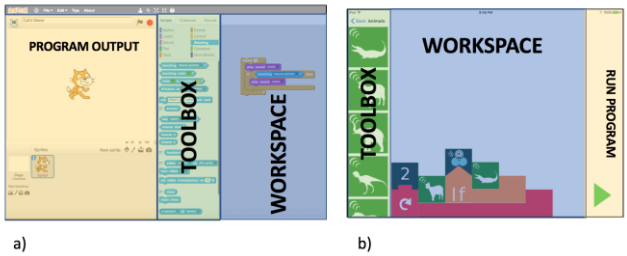


**Figure 1: Image comparing the three main components (toolbox, workspace and program output) of blocks-based environments in (a) the Scratch environment [15] and (b) a version of the Blocks4All environment. In Blocks4All, we used a robot as the accessible program output, so only needed a button on the screen to run the program. The Blocks4All environment shows a "Repeat Two Times" loop with a nested "Make Goat Noise" block and a nested "If Dash Hears a Sound, Make Crocodile Noise" statement.**

### Blocks-based Programming Environments

Blocks-based environments consist of a toolbox of blocks that can be dragged and dropped into a workspace to create a program, which can be run to produce some output. Figure 1 compares (a) the Scratch environment, one of the earliest and most popular blocks-based environments with (b) one version of the Blocks4All environment. Existing blocks-based environments are generally not accessible (discussed in the next section), but there are two exceptions to this rule. Researchers at the University of Colorado Boulder are developing a blocks programming environment [11,12] and Google has created Accessible Blockly [6]. Both environments use hierarchical menus to represent both the toolbox and workspace, which can be navigated with screen readers. These are both web-based applications designed to work well with desktop-based screen readers. Instead of designing a separate interface for visually impaired children, our goal was to provide insights into how to make existing block-based environments universally accessible. To do so, we closely mimicked existing block-based environments and used touchscreens, which we believe may be easier for young children to use.

### Touchscreen Accessibility

With the introduction of VoiceOver [28] and Talkback [29] screen readers on iOS and Android platforms, respectively, touchscreen devices have become very popular among people with visual impairments [30]. The screen readers use an interaction technique similar to one introduced by Kane et al. [8], where a user can explore the screen with a single finger. As elements on the screen are touched they are described via speech, and they can be selected via a double tap anywhere on the screen. There are simple touchscreen multi-touch gestures to scroll down or move left and right to new screens, so that one-finger touch does not accidentally change screens. This interaction method allows a visually impaired user to understand the spatial layout of elements on the screen.

As these screen readers are built in to the touchscreen devices, they interact well with applications that use standard iOS and Android elements. However, applications with custom visual elements (such as blocks) and gesture-based interactions (such as drag and drop) are often not accessible. As of iOS 11, there are two VoiceOver methods to replace drag and drop for items on Apple touchscreen devices. The first is a double tap and hold, which allows the user to access the underlying drag gesture (and which must be augmented with audio descriptions of where you are dragging the item). The second is a select, select, drop, in which you select an item, pick the drag option out of a menu of actions and then select a location to place the item. Both of these methods work to move applications on the home screen, but to work within applications, developers have to do extra work: provide the audio descriptions for the first method and have to include the action for the second method.

Current research in touchscreen accessibility explores how adults with visual impairments input gestures and make sense of spatial information on touchscreens. Kane et al. [10] explored the gesture preferences of people with visual impairments for input on touchscreens. As design guidelines, they recommend that designers favor edges and corners, reduce demand for location accuracy by creating bigger targets and reproduce traditional spatial layouts when possible. Giudice et al. [4] used a vibro-audio interface on a tablet to help participants with visual impairments explore simple on-screen elements. They found that people could identify and explore small bar graphs, letters and different shapes, but that it was difficult for participants to move in straight line across the tablet and

[1] https://github.com/milnel2/blocks4alliOS

suggested using secondary cues for helping them to stay straight. We incorporated these design guidelines into our initial design for Blocks4All, exploring if they need to be adapted for children with visual impairments.

There has been less work on how to develop touchscreen applications for children with visual impairments. Milne et al. [18] developed Braille-based games for smartphones using the touchscreen. Their work shows that young children had difficulties with many of the gestures that are used by the screen reader and in many applications. Therefore, we chose to simplify the number of gestures used in our application.

Similarly, there has been little work on touchscreen accessibility for people with low vision. Szpiro et al. investigated how people with low vision use computing devices and found that they prefer to rely on their vision as opposed to access information aurally [22]. However, they found the use of zoom features to be tedious and time consuming, and made it difficult to get contextual information as the zoomed in portion of the interface obscured parts of the rest of the screen. As we wanted to design both for children who are blind and those who have low vision, we incorporated their findings into our design, and allow users to enlarge portions of the interface, without obscuring other elements.

## ACCESSIBILITY OF EXISTING BLOCKS-BASED ENVIRONMENTS

We evaluated 9 existing blocks-based environments, including the web-based environments of Scratch [15], Blockly [5], Accessible Blockly [6], Tynker [31], and Snap! [24] and the mobile-based environments Blockly (Android) [5], Scratch Jr (iOS) [3], Hopscotch (iOS) [32], and Tickle (iOS) [33] in March 2017.

### Methods

We evaluated the web-based applications using the NVDA screen reader on Firefox and the mobile based ones with VoiceOver on iOS or Talkback on Android. The guidelines for evaluation were drawn from the WCAG 2.0 [34], Android [35] and iOS guidelines [1]. From the guidelines five accessibility barriers emerged: (1) *Accessing Output*: is the output of the programming perceivable (related to WCAG Principle 1), (2) *Accessing Elements*: are the menus and blocks perceivable (WCAG Principle 1), (3) *Moving Blocks*: can the blocks/code be moved and edited using a screen reader (WCAG Principles 2 and 3), (4) *Conveying Program Structure*: are the relationships between programming elements perceivable (WCAG Principle 1), and (5) *Conveying Type Information*: are data types perceivable (WCAG Principle 1).

#### *Modified Android Blockly*

The majority of the environments did not allow users to access the blocks via the screen reader and used inaccessible gestures. Because Android Blockly is open source we were able to build a Modified Android Blockly that fixed trivial accessibility problems (making the blocks accessible to the screen reader and replacing drag and drop with selecting a block and then selecting where you would like to place it) to gain insights into other interactions that may be difficult. We did pilot testing with two sighted adults using TalkBack to determine if there were more accessibility problems we should design for before testing with visually impaired children. We describe both the accessibility problems we found with the original Blockly and the Modified Android Blockly.

### Accessibility Problems

We address the five accessibility barriers below.

#### *Accessing Output*

We found that Scratch, ScratchJr, Hopscotch, Tynker and both versions of Blockly had some audio output options, but the majority of the games and projects focused on visual output, such as animating avatars. The Tickle application also had audio output and could be used to control robots, giving more accessible options. Accessible Blockly is not currently part of development environment, so there is no output to evaluate on.

#### *Accessing Elements*

We found that, with the exception of Accessible Blockly and the Tickle application, it was impossible to focus on blocks in the toolbox or in the workspace using a screen reader. This is a trivial fix, but it rendered all the other environments completely inaccessible for screen reader users.

#### *Moving Blocks*

All the applications except Accessible Blockly relied on drag and drop to move the blocks from the toolbox to the workspace. Although it is technically feasible for drag and drop to work with screen readers (e.g. with VoiceOver on iOS, you can double tap and hold to perform an underlying gesture), extra information must be provided to place the blocks. Without the extra information, the user will be unaware of the current location of the block and where their intended target is in relation to their current location.

The Tickle application augments drag and drop by providing an audio description of the current location of the block as the blocks are dragged across the screen. It does not work with Apple's version of select, select, drop. With Accessible Blockly, blocks are placed in the workspace by selecting the place in the workspace where you would like the block to go, and then selecting the block you would like to place there from a menu that pops up. We tested both the Accessible Blockly and the Tickle method for placing and moving blocks in the workspace in the initial design for Blocks4All.

#### *Conveying Program Structure*

When testing the Tickle application and the Modified Android Blockly application with TalkBack, we found it was very difficult to determine the structure of the program. When blocks were nested inside of each other, as in a 'for' loop or 'if' statement, it was impossible to tell where the

nesting ended and which blocks were outside. Additionally, it was difficult to locate blocks on the screen and difficult to navigate in a straight line (similar to what Kane et al. [9] found) either down between block statements or horizontally to find nested blocks. In Accessible Blockly, program structure is conveyed in a hierarchical list, with nested statements contained as sublists under their containing statement. In our initial designs, we chose to represent this hierarchy aurally and spatially.

*Conveying Type Information*

In all the applications, apart from Accessible Blockly, type information was conveyed visually via the shape and color of the block. This is not accessible via screen reader and due to the size of the blocks not very accessible for people with low vision. Some applications (e.g. ScratchJr) avoid the need for types by having drop down menus to select certain typed options (e.g. having a menu with the number of times a repeat loop will repeat as opposed to introducing a type for numbers). Accessible Blockly explicitly names the type needed at each location (e.g. Boolean needed) and the type of each block is listed in the toolbox.

## DESIGN EXPLORATION

In designing our environment, we wanted to create an environment that was *independently* usable by and *engaging* for people who are *sighted*, who have *low vision* and who are *blind*, making only changes that could be adopted by existing blocks-based environments. Additionally, we wanted to support the features of blocks-based environments that make them suitable for young children: (1) a lack of syntax errors due to blocks that are units of code, which can only fit in places that are syntactically correct, (2) code creation using a menu of blocks that relies on recognition instead of recall, and (3) clear hints about the type and placement of blocks, which in traditional blocks-based environments is conveyed via shape and placement of blocks.

We chose to develop Blocks4All on a touchscreen device, and specifically an iPad, for multiple reasons: (1) they are popular among people with visual impairments and have a state-of-the-art screen reader that is built-in [30], (2) they are easy to use and popular for children in educational contexts (iPads are actually provided to every child in some school districts), (3) many blocks-based environments are already available as touchscreen applications [3,33], and (4) touchscreens used with a screen reader allow for spatial exploration of the screen, which could be useful for conveying program structure.

### Initial Design

The initial design of Blocks4All was based on prior research and our own design exploration. We summarize our different approaches to overcome the five accessibility barriers we identified.

*Accessing Output*

We created a blocks-based environment that can be used to control a Dash robot [26], as this makes for tangible output that is very accessible for children with visual impairments. We included commands to move the robot (e.g. "Drive Forward/Backward", "Turn Right"), for the robot to make sounds (e.g. "Bark like a Dog", "Say Hi"), as well as repeat and 'if' statements.

*Accessing Elements*

In our design, blocks can be accessed using VoiceOver, which provides the name, the location and the type of the block, (e.g. "Drive Forward, Block 2 of 4 in workspace, operation"). We also give hints on how to manipulate the blocks (e.g. "double tap to move block"). Blocks in the workspace are placed along the bottom of the screen to help with orientation, as it is easy for blind users to "drift off course" when tracing elements on a touchscreen [9,21].

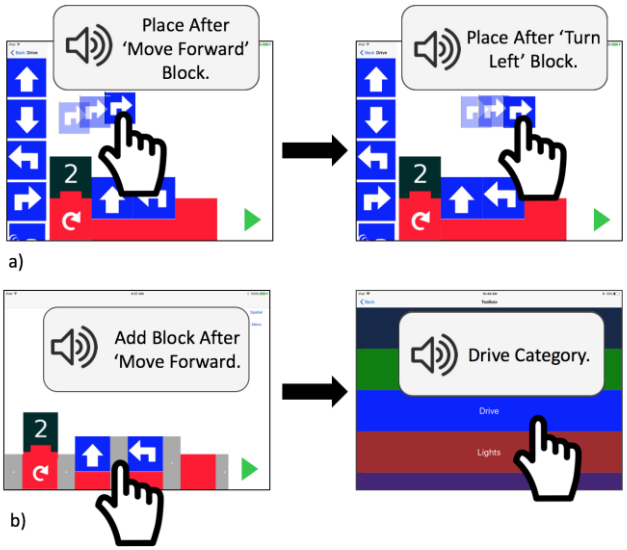


**Figure 2. Two methods to move blocks: (a) audio-guided drag and drop, which speaks aloud the location of the block as it is dragged across the screen (gray box indicates audio output of program) and (b) location-first select, select, drop, where a location is selected via gray "connection blocks", then the toolbox of blocks that can be placed there appears.**

*Moving Blocks*

We initially explored two methods to move blocks: (1) ***audio-guided drag and drop***, with a similar set-up to traditional blocks-based environments with the toolbox on the left side of the screen and which gives feedback about where in the program a block is as it is dragged across the screen (e.g. "Place block after Move Forward Block") (Figure 2a), and (2) ***location-first select, select, drop***, where a location is selected in the workspace via "connection blocks," which represent the connecting points of the block (analogous to the jigsaw puzzle piece tabs in the visual version), and then a full screen menu pops up with the toolbox of blocks that can be placed at that location (Figure 2b). This is slightly different from traditional blocks-based environments, in which you first select the block and then the location to place it, but it is logically similar to the method used in Accessible Blockly, although the physical manifestation is different.

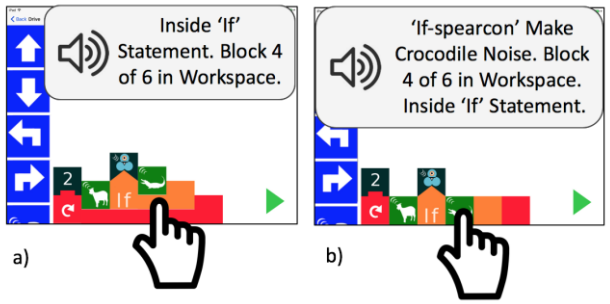


**Figure 3. Two methods to indicate the spatial structure of the code: (a) a spatial representation with nested statements placed vertically above inner blocks of enclosing statements, and (b) an audio representation with nesting communicated aurally with spearcons (shortened audio representations of words).**

*Conveying Program Structure*

We tested two methods to indicate the spatial structure of the code. The first is a ***spatial representation*** with repeat loops and conditionals represented with both a start and an end block and nested statements placed vertically above special inner blocks of their enclosing statements (Figure 3a). Navigating with VoiceOver, users can determine if or how deeply nested a statement is, by counting the number of "Inside _" blocks below it. The second is an ***audio representation*** with start and end blocks for the enclosing statements. When nested blocks are selected, nesting is communicated aurally with spearcons: short, rapidly spoken words, in this case "if" and "repeat" [25] (Figure 3b).

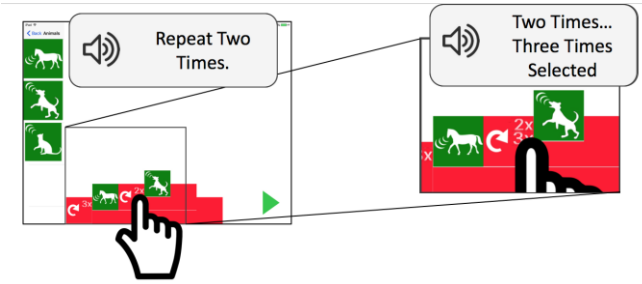


**Figure 4. The first method to access different block types: embedded typed blocks, accessed from a menu embedded within each block (e.g. "Repeat 2/3 times")**

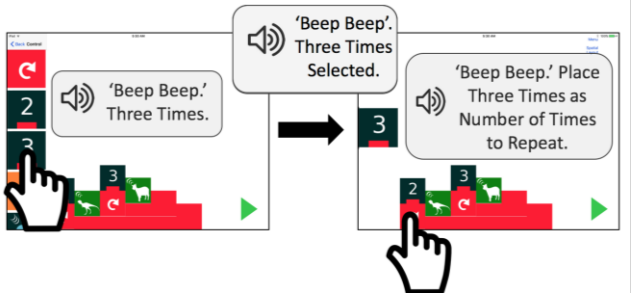


**Figure 5. The second method to access different block types: *audio-cue typed blocks*, when a typed block in the toolbox and the blocks in the workspace that accept it play the same distinct audio cues.**

*Conveying Type Information*

Our prototype application supports three types of blocks: (1) operations (e.g. "Drive Forward"), (2) numbers (used in conjunction with repeat loops and as distances for driving blocks), and (3) Boolean statements (e.g. "Hears Sound" and "Senses Obstacle" used in conjunction with 'if' statements). We explored two methods to access these different block types. The first is ***embedded typed blocks*** within operation blocks. These can be accessed from a menu embedded within each block (e.g. "Drive Forward *10/20/30*") (Figure 4). To access these menus with VoiceOver, you select the containing block and then swipe up or down to cycle through the options. This is similar to the approach taken in the ScratchJr [3] and the Dash robot [26] applications. The second is ***audio-cue typed blocks*** (Figure 5). In this method, when a number or Boolean block is selected with VoiceOver in the menu, it plays a distinct audio cue (two short low-pitched notes for numbers and two short high-pitched notes for Booleans), and the workspace blocks play matching audio cues for the types they can accept ('if' statements play two short high-pitched notes as they can accept Booleans). This information is also conveyed visually with the shapes of the blocks: 'if' statements have a triangular top tab and Booleans have an inverted triangle to indicate they fit together, while repeat statements have a rectangular top tab and numbers have an inverted rectangle (Figure 5). The visual approach is similar to traditional blocks-based environments (e.g. Blockly [5] and Scratch [15]), but the tabs are larger to accommodate users with low vision.

## DESIGN OF FORMATIVE STUDY AND INTERVIEW

### Interview with Teacher of the Visual Impaired

To collect feedback on our initial designs, we conducted a semi-structured interview with a teacher of the visually impaired (TVI), who works with elementary school-aged children in a local district. We asked about her work teaching children how to use technology and the assistive tools (screen readers, zoom, connectable braille displays, etc.…) on touchscreen devices. We also asked her to provide feedback on our different designs, and we include her feedback in the discussion below on our design.

### Participants with Visual Impairments

We recruited 5 children (3 male) aged 5-10 with visual impairments through our contacts with teachers of the visually impaired for a formative study (Table 1). Two of the children (P4, P5) used VoiceOver exclusively (P4 has some residual sight), one (P2) relied on sight and occasionally used VoiceOver to explore new blocks, and two children relied entirely on sight (P1, P3), holding the application close to their faces to read.

### Methods

The children participated in one to four 60-90 minute sessions in which they programmed a Dash robot [26] using an iPad running iOS 10.3.3.

| Participant | Age | Gender | Evaluation Sessions | Level of Corrected Vision | Previous Technology Use |
|---|---|---|---|---|---|
| P1 | 8 | Female | 4 | 20/150 | Lots of experience with screen readers, uses iPads and tablets. |
| P2 | 8 | Male | 4 | 20/80-20/100 | Uses iPad at school as assistive technology device with both VoiceOver and Zoom |
| P3 | 5 | Male | 1 | 20/100 | Uses iPads at home for games. |
| P4 | 10 | Male | 3 | 20/400 | Uses VoiceOver on iPad and iPhone. Uses Apple computer, Braille, CCTV, Kindle. |
| P5 | 9 | Female | 3 | Totally blind, no light perception | Uses iPad at school with VoiceOver and refreshable Braille reader and Braille Note. |

**Table 1. Participant Details.**

In each session, we introduced them to the interfaces using cardboard cutouts with Braille and large lettering to indicate the different application elements. For each of three sessions, the children used two interfaces (counterbalanced between children) to complete four simple tasks for each interface and then had free time to program the robot to do whatever they pleased. These tasks were modeled after the tasks in the hour of code for the Dash robot [26], but were modified to work for children with low vision (e.g. "Have Dash drive forward, turn right and drive forward to run into an obstacle").

In the first session, the children tried the two methods for moving blocks. Based on his age and shorter attention span, P3 only participated in this first session. In the second session, we introduced repeat loops, and the children used the two different methods for conveying program structure. In this session, we had tasks that required single and double nested repeat loops (e.g. "Have Dash drive in a square using only one forward and one turn left block"). In the third and fourth sessions, we introduced 'if' statements, and the children used the different methods for accessing type information. P1 and P2 used the ***embedded typed blocks*** in session 3 and the ***audio-cue typed blocks*** in session 4, and P3 and P4 used both in session 3. In these later sessions, we had tasks that required using different conditions for repeat loops and conditionals (e.g. "Have Dash 'Say Hi' if he hears a sound").

### *Measures*

For each task, we video-recorded the children and evaluated the interfaces for usability issues. We also measured the time it took to complete the task and signs of excitement (exclamations, laughing and smiling) and tiredness/boredom (yawns, frowns) [7]. At the end of each session, we asked for feedback on the different interfaces. In the final session, we asked the children questions from the scales for interest/enjoyment, perceived competence and effort/importance from Intrinsic Motivation Inventory to determine how engaged and motivated they felt during the programming activities [16]. The questions were trivially changed to match the tasks from the study (e.g. "I think I am pretty good at this activity" became "I think I am pretty good at programming robots"). Based on the small number and wide range in ability and age of participants, we report only summaries of our findings on the usability of the interfaces.

## RESULTS OF FORMATIVE STUDY AND INTERVIEW

We report on the feedback from the formative study and interview, and the resulting changes to the design of Blocks4All.

### Accessing Output

We chose to use the Dash robot as the output for the programming tasks. All five of the children greatly enjoyed using the robot, and three of the five children asked to be photographed with the robot. All the children, even those with very little or no functional vision were able to hear the robot and follow its movements by placing a hand on it. In order to make it clear to the children when they successfully completed a task such as "Have Dash move in a square", we created towers out of wooden blocks that the robot would knock down for each segment of the task (e.g. in each corner of the square). All of the children thought this was quite fun. We did not explore any other accessible programming outputs in the study, but would like to add the option of recording or typing your own audio for the robot to speak in future prototypes.

### Accessing Elements

To answer to the first part of R*Q2: What touchscreen interactions can children with visual impairments use to identify blocks and block types?* we found that children in our study could access blocks in our prototype application most easily when the blocks (1) were aligned along the bottom of the screen, (2) were resizable, (3) were separated by white space, and (4) could be accessed with both VoiceOver and through a keyboard. We elaborate on our findings below.

#### *Initial Findings*

All the children could focus on the blocks in Session 1; however, P5 had difficulty selecting blocks using the standard double tap gesture with VoiceOver, so for later sessions, she used a Bluetooth connected keyboard connected to the iPad to do the selection. The keyboard was

used with "Quick Navigation" on in conjunction with VoiceOver to focus on items in the application using left and right arrows and to select items by using the up and down arrows simultaneously [36].

None of the children with some vision (P1, P2, P3, and P4) used the zoom function on their iPad to view the blocks, instead they relied on VoiceOver or held the iPad within inches of their faces. In our interview with the TVI, she reported that often children had difficulty using the zoom feature because they had to switch between getting an overview of the application to focusing in on an element.

*Design Modifications*

Based on the feedback of the children with some vision, we added high contrast white space between the blocks in the toolbox and made the blocks resizable in both the toolbox and the workspace for all participants after session 1. This allowed children with some sight to see the details on the blocks, but still left the application layout the same so that they could get an overview of the program.

### Moving Blocks

In answer to *RQ3: What touchscreen interactions can children with visual impairments use to build blocks programs?* we explored multiple techniques to move blocks and found that (1) all the children in our study could use the select, select, drop method, (2) none of the children could not use the ***audio-guided drag and drop*** method with VoiceOver and (3) all the children preferred first choosing a block to move as opposed to a location to move it to when using the ***select, select, drop*** method. We elaborate on our findings below.

*Initial Findings*

All of the children had difficulty with the ***audio-guided drag and drop*** method. Neither of the children (P4, P5) that used VoiceOver could perform the VoiceOver-modified drag and drop gesture. The children that relied on sight with the iPad held close to their faces (P1, P3) found the drag and drop gesture difficult to perform as well, because moving the iPad to see the blocks interfered with the gesture. The ***location-first select, select, drop*** method worked well, and the children were able to complete all the tasks. However, P5 found that switching to a new screen to select a block after selecting a location was confusing with VoiceOver. In the post-session interviews, P2 and P4 expressed that they liked the idea of selecting a block first and then a location (as in the drag and drop interface) better. Also, the TVI noted that many VoiceOver users use the item chooser where you can search for an element on the current screen by name, making it faster to use an interface if you do not have to switch between screens to access items.

*Design Modifications*

We created a hybrid of our two methods: ***blocks-first select, select, drop***, where blocks are selected from the toolbox on the left side of the screen. Then the application switches into "selection mode", where the toolbox menu is replaced with information about the selected block, and a location can be selected in the workspace. All the participants who participated in two or more sessions (P1, P2, P4 and P5) used this method after session 1 and stated that they preferred this "hybrid" method to either of the two original methods.

### Conveying Program Structure

In answer to *RQ4: What touchscreen interactions can children with visual impairments use to understand the spatial structure of blocks program code?* we found that children were able to understand program structure using both the ***spatial*** and ***audio representations*** we explored, and most participants preferred the ***spatial representation***.

*Initial Findings*

The participants were able to understand both the ***spatial*** and ***audio representations*** of the program structure. P1, P2, P4, and P5 could complete all tasks with both representations (P3 did not attempt to use either as he only participated in the first session). Both P1 and P4 preferred the spatial representation, and P4 noted that he did not pay attention to the spearcons when using VoiceOver in the audio representation. The TVI noted that many children with visual impairments use a spatial mental model to remember information, so she thought the spatial interface would help with recall. P2 thought the spatial representation was easier to use, but thought that the audio presentation "looked nicer".

*Design Modifications*

After the second session, we used the ***spatial representation*** to convey program structure. We modified the order that VoiceOver read the blocks in the spatial representation so that it focused on the contained block first followed by the "Inside _" blocks (e.g. "Drive Forward Block" followed by "Inside Repeat Two Times").

### Conveying Type Information

In answering the second part of R*Q2: What touchscreen interactions can children with visual impairments use to identify blocks and block types?* we found that children were able to use both methods to select blocks, but that the ***audio-cue typed blocks*** need better cues for children who are not using VoiceOver. We elaborate on these findings below.

*Initial Findings*

All participants, other than P3 who did not attempt it, were able to use both methods of selecting typed blocks. P2 and P5 had some difficulty scrolling through the menus to select the ***embedded typed blocks***, but both were able to do so after some practice. We found that the children who used VoiceOver (P4, P5) with the ***audio-cue typed blocks*** had an easier time determining where Boolean and number statements could fit, as they received the audio cues from VoiceOver and could listen for it as they chose where to place the blocks. The children who did not use VoiceOver (P1, P2) often tried to place the typed blocks in places where they could not go, indicating that the visual cue was

not enough. Additionally, although the children found the number blocks quite easily in the toolbox, it took some trial and error for them to find the Boolean type blocks, likely because they were less familiar with that type. This was not a problem with the ***embedded typed blocks*** as the typed blocks were contained inside the conditional and repeat blocks and were not in the toolbox menu.

*Design Modifications*

Although it was more difficult for the children to grasp, we plan to use the ***audio-cue typed blocks*** in the future, as this method allows for more flexibility in creating blocks and programs and can more easily accommodate creating more complex statements. However, we plan to add better visual and audio cues when VoiceOver is not on, such as highlighting the blocks where a typed block will fit and using an error sound to indicate if a block cannot be placed in a location.

### Other Feedback

We received positive feedback on our interfaces. The children liked the all the interfaces: all 5 reported that they thought the tasks were a 5 or "really fun" on a Likert fun scale after using each interface. Using the Intrinsic Motivation Inventory with a 5 point Likert scale, participants rated completing the tasks on the interfaces high on the scales for interest/enjoyment (4.71, SD=0.32), perceived confidence (4.45, SD=0.44), and low for pressure/tension (1, SD=1.75). All the children chose to continue playing with the interfaces after completing the tasks.

## DESIGN GUIDELINES

Based on our formative studies, we distill design guidelines for designers to make both blocks-based environments and touchscreen applications at large usable by children with visual impairments. In particular, we focus on guidelines to make applications more usable by both children with low vision and children who are blind, as the former are largely understudied.

*Make Items Easily Locatable and Viewable on Screen*

In agreement with previous work [22], we found that the children in our study with low vision did not like to use the zoom function, as it made it harder to interact with items and occluded other elements in the application. Based on feedback from early sessions, we made the blocks resizable instead, so children could see the blocks without occluding other parts of the application.

In our pilot testing of the Modified Android Blockly application with TalkBack, we found it was important to locate elements close to the edge of the screen, so they could be found without vision, as our participants found it difficult to navigate "floating" segments of code. Kane et al. [9] followed a similar guideline when designing interactions for large touch interfaces, and we found it equally important for a standard-size iPad.

We recommend making elements resizable and high contrast and locating elements close to the edge of the screen to make them findable.

*Reduce the Number of Items on Screen*

Simple interfaces are easier to use for children in general, and we found it was especially important to reduce the number of focusable items on the screen for both our visual and audio interfaces. For children with low vision, having fewer items on the screen made it easier to identify and remember what the different elements were, and for children who were blind, having fewer items made it harder to "get lost" while navigating with VoiceOver. However, we found it was important to not have multiple screens needed to perform an interaction. For example, in our ***location-first select, select, drop***, some children found it difficult to switch between screens to select blocks, and having multiple screens makes it more challenging for VoiceOver users to use the search feature.

*Provide Alternatives to Drag and Drop*

We found that it is important to provide alternatives to drag and drop. Children were not able to use VoiceOver to perform an audio-guided drag and drop, and we also found that children with low vision had difficulty with drag and drop. They held the iPads close to their faces, making it difficult to both drag the block and see what was going on. To the best of our knowledge, this is a novel finding, although it aligns well with work by Szpiro et al. [22] who found that adults with low vision preferred to move closer to screens instead of using zoom functions. Drag and drop could also pose difficulties for children with motor impairments. We found that select, select, drop worked well instead, but it was important to make it clear non-visually that the application had switched to a "selection" mode and to make it clear what the currently selected item was.

*Convey Information in Multiple Ways*

In designing an application that can be used by children with a wide range of visual impairments, it is essential to convey information in multiple ways. Using spatial layouts with both visual and audio cues helped all of the children understand program structure. We also found that when we did not provide enough visual cues (e.g. cues about where ***audio-cue typed blocks*** fit with VoiceOver off), the children had difficulty using the application. Other approaches to making blocks programming environments accessible to children with visual impairments (Accessible Blockly [6] and work at the University of Colorado, Boulder [11,12]), have focused on creating an interface that works well with a screen reader, but without many visual elements, which would be less appealing and likely more difficult to use for children with some vision.

Dash, the talking robot we used in our study, was very popular with all of our participants and is accessible for children who are blind, have low vision or are sighted. This is in contrast to most current applications that use blocks-based programming, which rely heavily on visual output for

programming tasks (e.g. programming the avatars in Scratch or ScratchJr [3,15]). In the future, we believe designers should consider how to incorporate more physical and auditory output for programming tasks, which would make their applications more broadly appealing, and which is particularly important for children with visual impairments.

## LIMITATIONS AND FUTURE WORK

There are several limitations of our study. Because of the difficulty in recruiting from this population, we had a small number of participants, with a wide range of ages and abilities, making it difficult to make comparisons between participants. We also explored only a small number of design possibilities to make these environments accessible and were constrained by our desire to only make changes that could be adopted by all designers of blocks-based environments. There are still open questions we would like to explore: how to navigate more complicated hierarchies of nested code, how to accommodate multiple "threads" of program code, and how to best incorporate speech-based commands or other gestures. We also plan to do a formal evaluation of Blocks4All.

## CONCLUSION

We conducted an evaluation of current blocks-based environments and found five accessibility barriers. We designed multiple techniques to overcome these barriers and conducted a formative study to evaluate these techniques with five children with visual impairments. We distilled the findings from this study into a final design, which we plan to evaluate formally, and a set of design guidelines for designers of these applications.

## ACKNOWLEDGMENTS

This work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-1256082 and a SIGCSE special projects grant.

## REFERENCES

1. Apple. 2012. Accessibility Programming Guide for iOS. Retrieved February 25, 2017 from https://developer.apple.com/library/content/documentation/UserExperience/Conceptual/iPhoneAccessibility/Introduction/Introduction.html#//apple_ref/doc/uid/TP40008785
2. David Bau, Jeff Gray, Caitlin Kelleher, Josh Sheldon, and Franklyn Turbak. 2017. Learnable Programming: Blocks and Beyond. *Commun. ACM* 60, 6: 72–80. https://doi.org/10.1145/3015455
3. Louise P. Flannery, Brian Silverman, Elizabeth R. Kazakoff, Marina Umaschi Bers, Paula Bontá, and Mitchel Resnick. 2013. Designing ScratchJr: Support for Early Childhood Learning Through Computer Programming. In *Proceedings of the 12th International Conference on Interaction Design and Children* (IDC '13), 1–10. https://doi.org/10.1145/2485760.2485785
4. Nicholas A. Giudice, Hari Prasath Palani, Eric Brenner, and Kevin M. Kramer. 2012. Learning Non-visual Graphical Information Using a Touch-based Vibro-audio Interface. In *Proceedings of the 14th International ACM SIGACCESS Conference on Computers and Accessibility* (ASSETS '12), 103–110. https://doi.org/10.1145/2384916.2384935
5. Google. Blockly. Retrieved from https://developers.google.com/blockly/
6. Google. Accessible Blockly. Retrieved from https://blockly-demo.appspot.com/static/demos/accessible/index.html
7. Libby Hanna, Kirsten Risden, and Kirsten Alexander. 1997. Guidelines for Usability Testing with Children. *interactions* 4, 5: 9–14. https://doi.org/10.1145/264044.264045
8. Shaun K. Kane, Jeffrey P. Bigham, and Jacob O. Wobbrock. 2008. Slide Rule: Making Mobile Touch Screens Accessible to Blind People Using Multi-touch Interaction Techniques. In *Proceedings of the 10th International ACM SIGACCESS Conference on Computers and Accessibility* (Assets '08), 73–80. https://doi.org/10.1145/1414471.1414487
9. Shaun K. Kane, Meredith Ringel Morris, Annuska Z. Perkins, Daniel Wigdor, Richard E. Ladner, and Jacob O. Wobbrock. 2011. Access Overlays: Improving Non-visual Access to Large Touch Screens for Blind Users. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology* (UIST '11), 273–282. https://doi.org/10.1145/2047196.2047232
10. Shaun K. Kane, Jacob O. Wobbrock, and Richard E. Ladner. 2011. Usable Gestures for Blind People: Understanding Preference and Performance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '11), 413–422. https://doi.org/10.1145/1978942.1979001
11. 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
12. 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
13. Stephanie Ludi, Mohammed Abadi, Yuji Fujiki, Priya Sankaran, and Spencer Herzberg. 2010. JBrick: Accessible Lego Mindstorm Programming Tool for Users Who Are Visually Impaired. In *Proceedings of the 12th International ACM SIGACCESS Conference on Computers and Accessibility* (ASSETS '10), 271–272. https://doi.org/10.1145/1878803.1878866
14. Stephanie Ludi and Tom Reichlmayr. 2011. The Use of Robotics to Promote Computing to Pre-College Students with Visual Impairments. *Trans. Comput. Educ.* 11, 3: 20:1–20:20. https://doi.org/10.1145/2037276.2037284

15. John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The Scratch Programming Language and Environment. *Trans. Comput. Educ.* 10, 4: 16:1–16:15. https://doi.org/10.1145/1868358.1868363
16. Edward McAuley, Terry Duncan, and Vance V. Tammen. 1989. Psychometric Properties of the Intrinsic Motivation Inventory in a Competitive Sport Setting: A Confirmatory Factor Analysis. *Research Quarterly for Exercise and Sport* 60, 1: 48–58. https://doi.org/10.1080/02701367.1989.10607413
17. S. Mealin and E. Murphy-Hill. 2012. An exploratory study of blind software developers. In *2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 71–74. https://doi.org/10.1109/VLHCC.2012.6344485
18. Lauren R. Milne, Cynthia L. Bennett, Richard E. Ladner, and Shiri Azenkot. 2014. BraillePlay: Educational Smartphone Games for Blind Children. In *Proceedings of the 16th International ACM SIGACCESS Conference on Computers & Accessibility* (ASSETS '14), 137–144. https://doi.org/10.1145/2661334.2661377
19. Jaime Sánchez and Fernando Aguayo. 2005. Blind Learners Programming Through Audio. In *CHI '05 Extended Abstracts on Human Factors in Computing Systems* (CHI EA '05), 1769–1772. https://doi.org/10.1145/1056808.1057018
20. Andreas Stefik, Christopher Hundhausen, and Robert Patterson. 2011. An Empirical Investigation into the Design of Auditory Cues to Enhance Computer Program Comprehension. *Int. J. Hum.-Comput. Stud.* 69, 12: 820–838. https://doi.org/10.1016/j.ijhcs.2011.07.002
21. Andreas M. Stefik, Christopher Hundhausen, and Derrick Smith. 2011. On the Design of an Educational Infrastructure for the Blind and Visually Impaired in Computer Science. In *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education* (SIGCSE '11), 571–576. https://doi.org/10.1145/1953163.1953323
22. Sarit Felicia Anais Szpiro, Shafeka Hashash, Yuhang Zhao, and Shiri Azenkot. 2016. How People with Low Vision Access Computing Devices: Understanding Challenges and Opportunities. In *Proceedings of the 18th International ACM SIGACCESS Conference on Computers and Accessibility* (ASSETS '16), 171–180. https://doi.org/10.1145/2982142.2982168
23. Anja Thieme, Cecily Morrison, Nicolas Villar, Martin Grayson, and Siân Lindley. 2017. Enabling Collaboration in Learning Computer Programing Inclusive of Children with Vision Impairments. In *Proceedings of the 2017 Conference on Designing Interactive Systems* (DIS '17), 739–752. https://doi.org/10.1145/3064663.3064689
24. UC: Berkley. *Snap! Build your own blocks*. Retrieved from https://snap.berkeley.edu/
25. Bruce N. Walker, Amanda Nance, and Jeffrey Lindsay. 2006. Spearcons: speech-based earcons improve navigation performance in auditory menus. Retrieved September 10, 2017 from https://smartech.gatech.edu/handle/1853/50642
26. Wonder Workshop. *Blockly iOS Application for Dot and Dash Robots*. Retrieved from https://www.makewonder.com/apps/blockly
27. Hour of Code. *Code.org*. Retrieved September 3, 2017 from https://hourofcode.com/learn
28. Vision Accessibility - Mac - Apple. Retrieved September 5, 2017 from https://www.apple.com/accessibility/mac/vision/
29. Get started on Android with TalkBack - Android Accessibility Help. Retrieved September 5, 2017 from https://support.google.com/accessibility/android/answer/6283677?hl=en
30. WebAIM: Screen Reader User Survey #6 Results. Retrieved February 16, 2017 from http://webaim.org/projects/screenreadersurvey6/#mobile
31. Coding for Kids. *Tynker.com*. Retrieved September 5, 2017 from https://www.tynker.com/
32. Hopscotch - Learn to Code Through Creative Play. Retrieved September 5, 2017 from https://www.gethopscotch.com/
33. Tickle: Program Star Wars BB-8, LEGO, Drones, Arduino, Dash & Dot, Sphero, Robots, Hue, Scratch, Swift, and Smart Homes on your iPhone and iPad. *Tickle Labs, Inc.* Retrieved February 27, 2017 from https://www.tickleapp.com/
34. How to Meet WCAG 2.0. Retrieved February 25, 2017 from https://www.w3.org/WAI/WCAG20/quickref/
35. Accessibility Developer Checklist | Android Developers. Retrieved February 25, 2017 from https://developer.android.com/guide/topics/ui/accessibility/checklist.html
36. iOS VoiceOver Gesture, Keyboard & Braille Shortcuts | AxS Lab. Retrieved September 16, 2017 from http://axslab.com/articles/ios-voiceover-gestures-and-keyboard-commands.php