Touchscreen-Based Learning Technologies for Children with Visual Impairments

Lauren Ruth Milne

A dissertation

submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

University of Washington

2018

Reading Committee:

Richard E. Ladner, Chair

Steven L. Tanimoto

Andrew J. Ko

Program Authorized to Offer Degree:

Computer Science and Engineering

University of Washington

Abstract

Touchscreen-Based Learning Technologies for Children with Visual Impairments

Lauren Ruth Milne

Chair of the Supervisory Committee:
Richard E. Ladner, Professor Emeritus
Computer Science and Engineering

Many learning technologies, such as the block-based programming environments used to teach programming and many literacy applications, rely heavily on visual elements to convey information, making them inaccessible for children with visual impairments. In this dissertation, I explore ways to use touchscreens to make this material accessible: allowing children with visual impairments (aged 5-14) to access the spatial information using a combination of touchscreens, tactile and audio feedback. Because they can convey spatial information and rely on direct manipulation, touchscreens are a promising avenue of research for this population. However, there has been very little research on the use of touchscreens by children with visual impairments.

Through my research, I provide solutions to remedy this. I explore how children with visual impairments are currently using and being taught to use learning technology through

interviews with teachers of the visually impaired. I evaluated one type of learning technology (block-based programming environments) to determine how accessible they are. I created two accessible touchscreen applications that allow children with visual impairments to independently access similar content for learning to that of their sighted peers. The first is a suite of games that use the haptic and audio feedback on the touchscreen to represent Braille characters to allow children to practice reading and writing the characters. The second is a block-based programming environment to give children an introduction to computer programming.

In this dissertation, I present my research (1) to discover how children with visual impairments are currently using technologies, including touchscreens, and the accessibility problems with existing touchscreen devices and (2) on the design, development and evaluation of two pieces of touchscreen-based learning technology for children.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank my collaborators and my committee for their feedback and support for this work.

My collaborators in the project on technology use by children with visual impairments were Catherine Baker and Richard Ladner. This work was supported by the National Science Foundation under Grant No. #CNS-1440843.

My collaborators on the BraillePlay games project were Cynthia Bennett, Shiri Azenkot and Richard Ladner. This work was supported by the U.S. Department of Education, Office of Special Education Programs (Cooperative Agreement #H327A100014) and by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-1256082.

Both the accessibility challenges in block-based programming environments and design of Blocks4All were done in collaboration with Richard Ladner. This work was supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-1256082, the National Science Foundation under Grant No. #CNS-1440843, and a 2016 SIGCSE special projects grant.

Any opinions, findings, and conclusions or recommendations expressed in this dissertation are those of the author and do not necessarily reflect the views of the National Science Foundation or the Department of Education.

DEDICATION

For James and Peter, who both allowed me to work on this dissertation with good humor and a surprising amount of patience. I am so lucky to have both of you in my life. Thank you for always keeping me grounded, laughing and joyful.

GLOSSARY

TERM	DEFINITION
ASSISTIVE TECHNOLOGY	Any item, piece of equipment, or product system, whether acquired commercially off the shelf, modified, or customized, that is used to increase, maintain, or improve functional capabilities of individuals with disabilities (as defined by IDEA [133]).
BLOCK-BASED PROGRAMMING ENVIRONMENT	Computer programming in an environment where programming commands are represented as blocks: units of code that can be connected together to form programs (e.g. Scratch [107]).
BLIND	Defined in terms of function: to describe the vision of individuals who primarily use audio or Braille to access textual information.
LEARNING TECHNOLOGY	Tools that children directly interact with to support learning either in or outside a classroom.
LOW-VISION	Defined in terms of function: to describe the vision of individuals who have uncorrectable vision loss to the extent that they qualify for extra educational support (in the form of extra instruction or access to technologies) but are able to use their vision to access textual information with magnification or other aids.
REFRESHABLE BRAILLE DISPLAY	Hardware that allows a user to read output from a computer in Braille by raising and lowering small metal pins in Braille cells.
SCREEN READER	Software that interacts with the operating system on a computer in order to read aloud elements on the screen and allowing the user to navigate and select elements (e.g. JAWS [143], VoiceOver [144], TalkBack[42]).
TOOLBOX	The menu of possible blocks that can be added in a block-based environment.
VISUALLY IMPAIRED	Defined in terms of function: to describe the vision of individuals who have uncorrectable vision loss to the extent that they qualify for extra educational support (encompasses both low-vision and blindness)
WORKSPACE	The part of a block-based environment that holds the created program.

Chapter 1. INTRODUCTION

With the rise of the internet and computer, there has been a remarkable influx in the availability of digital educational content over the past 30 years [51]. This can benefit all students, but it has the potential to greatly even the playing field for students with visual impairments. This is because digital content can be more easily be accessed through multiple modalities than physical content (e.g. a digital book can be read as written text, enlarged by magnification, read aloud as audio, read as Braille on a refreshable Braille display or printed as Braille by an embosser). However, most of these educational tools have not been designed with students with visual impairments in mind, so they rely heavily on visual elements and do not interface well with assistive technologies such as screen readers.

In this dissertation, I explore some of the ways that current "mainstream" learning technologies are not accessible to children with visual impairments. I do this through interviews teachers of the visually impaired (TVIs) and through an evaluation of one type of learning technology (block-based computer programming environments). I also explore using mobile devices with touchscreens to create two pieces of learning technology: (1) the *BraillePlay* games, games that use the touchscreen of a smartphone to represent Braille characters and allow children to practice dot patterns, and (2) *Blocks4All* environment, an accessible programming environment that allows children to become familiar with some programming constructs.

1.1 MOTIVATION

I chose to design technologies for children with a wide range of visual impairments. According to the American Community Survey, there were approximately 706,400 children who reported having a visual disability in the United States in 2016 [55]. However, according to the American Printing House for the Blind (APH)[1], there were only 62,500 legally blind children in educational settings that qualified for reading materials in alternative formats (e.g. Braille, large print or audio) in 2014 [6]. Of these students, 8.6% use Braille as their primary reading medium, 9.4% use audio, 17% are pre-readers, 31% are visual readers (i.e. use print) and 34% are symbolic or non-readers. Throughout this this dissertation, I define vision in terms of functional ability to access textual information and use the following terms:

- *Blind* to describe the vision of children who primarily use audio or Braille to access textual information,
- *Low-vision* to describe the vision of children who have uncorrectable vision loss to the extent that they qualify for extra educational support (in the form of extra instruction or access to technologies), but are able to use their vision to access textual information with magnification or other aids, and
- *Visually impaired* to encompass vision that ranges from low-vision to complete blindness.

[1] The APH notes that "The specific purpose of the annual Federal Quota Census is to register students in the US and Outlying Areas who meet the definition of blindness and are therefore eligible for adapted educational materials from APH" and that the number of students who read Braille in the US cannot be determined from the data.

I chose to use touchscreen mobile devices as the output mechanism because they are:

- *Integrated with assistive technology:* they work well with assistive technology, iOS products in particular have a built-in screen reader and can be connected to Braille displays,
- *Able to convey spatial information:* they have touchscreens, which interface with screen readers so that blind children can get a better sense of how items are spatially laid out compared to traditional desktops or laptops,
- *Mainstream:* they are a mainstream technology that is widely used in educational settings (multiple school districts in Washington State have one-to-one iPads for elementary schools) and at home, and
- *Child-friendly:* they give children an easier interface than a traditional computer to learn both computing skills and how to use assistive technologies like a screen reader. These are both important skills that can be more challenging for children with visual impairments to learn, because they have to learn to an extra layer (screen reader or magnifier) on top of the technology itself.

Educational technology is broadly "the study and ethical practice of facilitating learning and improving performance by creating, using, and managing appropriate technological processes and resources" [108]. Throughout my dissertation, I use the term *learning technology* in a slightly narrower sense to mean specifically digital tools that children directly interact with to support learning either in or outside a classroom. Based on the small, heterogeneous populations and time constraints I worked with, I focused on designing pieces of technology to support children with visual impairments in practicing discrete skills (i.e. correctly entering the Braille dot patterns for a character and creating a block-based program that would make a robot drive in a square). I

focused on creating accessible interactions and measuring that the children were able to use the technology to perform similar actions to their sighted counterparts. A limitation of my work is that I did not directly measure how well my tools supported learning, although I collected qualitative feedback from the children, parents and teachers who participated in my studies about how well they thought the tools supported learning.

I chose to focus on creating learning technology to support practicing two important skills: Braille literacy and early computer programming literacy. Learning to read and write in Braille is an important skill for blind children, as it is directly correlated with academic achievement and employment [145]. Unfortunately, Braille literacy is declining, and it is estimated that only 10% of blind children are learning Braille [93]. This may be due to the fact that children with visual impairments are more likely to placed be in mainstream classrooms with sighted children and may only meet with a TVI a few times a week. This underscores the importance of creating multiple avenues for children to learn and practice Braille. As a blind child in a mainstream school may only have dedicated Braille instruction once or twice a week, it is important to create technologies that support independent practice of Braille skills.

Learning computer programming is also an increasingly important skill: children who learn programming skills today may be the creators of information technology tomorrow. Unfortunately computer science is a field that is notoriously lacking in diversity, and there are numerous challenges that make it difficult for students with visual impairments to learn to program [19]. This motivated me to develop techniques so that children with visual impairments can access block-based programming environments, and through them some of the earliest curriculum around computer science.

Although these technologies are only a small part of what is needed to master these two skills, they are a first step in allowing children with visual impairments to access independently access the digital content provided in these types of learning technologies. In this dissertation, I focus on creating learning technologies and developing touchscreen interaction techniques that provide non-visual access to the same types of information that is currently provided in a visual manner. My hope is that these techniques can be incorporated into existing learning technologies so that children with visual impairments can access existing curriculum and have the same learning opportunities as their sighted peers.

1.2 THESIS STATEMENT

The work in this dissertation supports the following thesis claims:

Children with visual impairments use mainstream touchscreen devices for learning in a variety of ways, despite having to overcome accessibility challenges in order to use them. In the context of block-based programming environments, these challenges include a reliance on visual metaphors and gesture-based interactions that do not interface well with the screen reader. To improve access to learning technologies on these devices, we can design applications that (1) use the spatial information afforded by the touchscreen to help children understand structural information such as the layout of a Braille cell or the structure of program code, and (2) use the visual, haptic and audio feedback of these devices to engage children and allow them to use these applications independently.

1.3 Research Questions and Approaches

To support my thesis, I first discuss my research into how children with visual impairments are using technology. I focused on how they are using touchscreen devices and the accessibility challenges they encounter, to answer the following research questions:

- Q1: How are children with visual impairments using touchscreen computing technologies for learning?
- Q2: What are the challenges that children with visual impairments face when using touchscreen technologies?
- Q3: What are the accessibility challenges in existing block-based programming environments for children with visual impairments?

I then explored how to overcome these challenges by creating two pieces of learning technology, answering the following research questions:

- Q4: How can we use a touchscreen device to create engaging games that children with visual impairments can use to practice reading and writing Braille characters?
- Q5: How can we use a touchscreen device to allow children with visual impairments to understand and create block-based programs?

1.3.1 *Technology use by Children with Visual Impairments*

To answer the first two research questions Q1 and Q2, I conducted interviews with six TVIs to understand how children were being introduced to and using technology in an educational setting. This provided an understanding of how children with visual impairments are currently using both mainstream and assistive technologies and the accessibility challenges they face. This also provided evidence supporting the first claim of my thesis that: *Children with visual impairments*

use mainstream touchscreen devices for learning in a variety of ways, despite having to overcome accessibility challenges in order to use them.

1.3.2 *BraillePlay Games*

In order to answer Q4, I created a suite of four word games with varying levels of difficulty, using a vibrating interface that used the spatial nature of the touchscreen to display Braille characters and reinforce Braille dot patterns for children with visual impairments who were learning to read Braille [54]. I ran a longitudinal study in the wild over four weeks with eight children with visual impairments. I found that all but one of the children could play the games independently and indicated that they understood the spatial representation of the Braille cell. They also reported that they enjoyed playing the games, although they chose to play them for only two and a half hours over the course of the study. This provides support for the third claim of my thesis: that *we can design applications that (1) use the spatial information afforded by the touchscreen to help children understand structural information such as the layout of a Braille cell or the structure of program code, and (2) use the visual, haptic and audio feedback of these devices to engage children and allow them to use these applications independently*.

1.3.3 *Accessibility Challenges in Block-Based Programming Environments*

To further explore the existing accessibility challenges in touchscreen applications, and answer Q2. I did an accessibility evaluation of a particular genre of learning technology, block-based programming environments, intended to be used as a way to introduce children to programming, answering Q3.

I identified five main accessibility problems in these environments, that broadly reflect three accessibility challenges a child with visual impairments might encounter with any touchscreen

application: (1) elements (in this case blocks) were not accessible (could not be read by the screen readers), (2) gestures (in this case drag and drop) were not compatible with screen readers, and (3) certain information (in this case program structure, type information about the blocks, and the output of the programs themselves) was conveyed only through visual metaphors that could not be accessed or understood without sight. This provides further support for the second claim of my thesis: that *in the context of block-based programming environments, these challenges include a reliance on visual metaphors and gesture-based interactions that do not interface well with the screen reader.*

1.3.4 *Blocks4All*

Based on the accessibility problems identified above, I built Blocks4All, a prototype block-based programming environment where I implemented various means to overcome these challenges using a touchscreen tablet. I worked with a TVI and five children with visual impairments who used Blocks4All to determine the usability of these techniques and answer *Q5: How can we use a touchscreen device to allow these children to understand and create block-based programs?*

I focused on exploring non-visual techniques to present information about blocks, block types, and the spatial structure of program code and interactions to replace the inaccessible drag and drop gesture. I evaluated the final version of the application with five children with visual impairments, showing that these children were able to create and manipulate block-based programs using my techniques and understand the structure of block-based programs using the touchscreen interface. This provides further support for the third claim of my thesis: *we can design applications that (1) use the spatial information afforded by the touchscreen to help children understand structural information such as the layout of a Braille cell or the structure of program code, and (2) use the*

visual, haptic and audio feedback of these devices to engage children and allow them to use these applications independently.

1.4 CONTRIBUTIONS

The contributions from this work are summarized below and include two artifact contributions (the BraillePlay games and Blocks4All environment) as well as empirical evidence on the existing accessibility challenges in touchscreen technologies and on how educational touchscreen technologies can be created to mitigate these challenges. The contributions include:

- Knowledge of how children with visual impairments are currently using information technologies and the challenges they face in using these technologies through interviews with TVIs,
- The BraillePlay games, both iOS and Android applications, as well as empirical findings and design implications from a longitudinal study that show how eight children with visual impairments interacted with games,
- An analysis of existing block-based programming environments and discussion of the accessibility challenges within them,
- The Blocks4All environment, with an exploration of multiple interaction techniques to overcome the previously identified accessibility challenges with five children with visual impairments and an evaluation of the final application with five children.

1.5 Dissertation Overview

This dissertation is divided into seven chapters which I summarize below:

- In *Chapter 2, Related Work,* I summarize research into how children with visual impairments are using technology, including touchscreens, and research in touchscreen accessibility and educational touchscreen applications.
- In *Chapter 3, Technology Use by Children with Visual Impairments,* I describe the research I conducted on how children with visual impairments are currently using technology in educational settings through interviews with TVIs.
- In *Chapter 4, BraillePlay Games,* I describe the design and evaluation of a suite of Braille-based smartphone games, which use the vibrating touchscreen interface to represent Braille characters. The work in this chapter was done in collaboration with Cynthia L. Bennett, Shiri Azenkot, Richard E. Ladner and is based on work previously published at ASSETS 2013 [88] and 2014 [90].
- In *Chapter 5, Evaluation of Existing Block-based Programming Environments,* I describe my investigation into the accessibility of block-based programming environment and testing of Tickle, an "accessible" environment with three children with visual impairments. The work in this chapter was done in collaboration with Richard E. Ladner and is based on work previously published at CHI 2018 [87].
- In *Chapter 6, Blocks4All,* I describe the design, development and evaluation of Blocks4All, an accessible block-based programming environment and provide design recommendations to make other block-based programming environments accessible. The work in this chapter was done in collaboration with Richard E. Ladner and is based on work previously published at CHI 2018 [87].

- In *Chapter 7, Contributions and Future Work*, I summarize the limitation and contributions of my work, and discuss future directions including exploring accessible programming activities.

Chapter 2. RELATED WORK

In this chapter, I discuss work that informed my research. I highlight work in four areas:

(1) *Touchscreen Accessibility*: research into making mainstream touchscreen devices more accessible for people with visual impairments,

(2) *Technology Use by Children with Visual Impairments:* research on how children (aged 0-18) are currently using both access and mainstream technologies,

(3) *K-12 Computer Science Education:* research on how computer science is taught to children, and

(4) *Accessible Computer Science*: research on both creating accessible educational tools for computer science and programming tools for blind programmers.

2.1 TOUCHSCREEN ACCESSIBILITY

In this section, I give an overview of how touchscreens are currently accessible with screen readers. I then discuss current research into non-visual input and output interactions for the phone, approaches to making the drag and drop gesture accessible and research into how people with low-vision use touchscreens.

2.1.1 *Overview*

When smartphones were first introduced, they were largely inaccessible for blind people. However, in 2009, Android and Apple released TalkBack and VoiceOver respectively, native screen readers which incorporate techniques similar to ones introduced by Kane *et al.* [57]. Currently with both TalkBack and VoiceOver, 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. Items are read aloud as users explored the screen by touch and the last item read is activated with a double tap. Since the introduction of these screen readers, smartphones have been widely adopted by people with visual impairments: a 2017 study by WebAIM found that 88% of people with visual impairments reported using screen reader on a mobile device, with 76% of them using VoiceOver and 22% using TalkBack [146]. Although these screen readers support navigating through text and menus very well, other graphical elements, custom user interface elements and gesture-based interactions (such as drag and drop) are not always accessible [49,89], for example only one of the block-based touchscreen interfaces that I evaluated in Chapter 5 was accessible.

2.1.2 *Touchscreen Input*

There has been research on improving both non-visual input and output mechanisms using touchscreens. There is a body of literature on various ways to input Braille on touchscreens [17,73,80,110], which is discussed in more detail in Chapter 4. There has also been research in using gestures to input other types of information. Kane *et al*. [60] explored the gesture preferences of people with visual impairments for input on tablet touchscreens. They compared the gestures invented by both visually impaired and sighted participants for common interaction tasks. They found that compared to sighted participants, visually impaired participants preferred gestures that relied on a virtual QWERTY keyboard (e.g. mimicking CTRL-V for copying), and gestures that used the corners or edges of screen. However, this work was done with adults with visual impairments and was done in 2011, and I have found no existing work looking at gesture preferences by children with visual impairments. As researchers have reported sighted children

can have difficulty performing certain gestures on touchscreens and their gestures are often misinterpreted [9,83,111], children with visual impairments may have different gesture preferences than adults and require a different set of design guidelines.

2.1.3 *Touchscreen Output*

Researchers have also explored many ways to help blind users make sense of output on touchscreens. Giudice *et al.* [41] used a vibro-audio interface on a tablet to help visually impaired participants explore simple on-screen elements. They found that people could identify and explore low resolution bar graphs, letters and different shapes with the same accuracy as with a traditional tactile graphic (although it took four times longer). I expand on this work by exploring if the touchscreen interface can be used to identify more complex multi-part elements (i.e. Braille characters) and more complex hierarchical information (i.e. the program structure of a block-based program).

Kane *et al.* [58] studied using different virtual information overlays to help blind users find items on large table-top touchscreens. They came up with three overlays that were well received in formative studies: edge projection (converting 2D information into linear list of targets along the edge of the screen), neighborhood browsing (increasing the size of targets and reducing empty space so that touching anywhere on the screen would result in speaking the nearest target), and voice commands. In another study, Kane *et al.* [59] used plastic tactile overlays to help users make sense of touchscreen content. In the BraillePlay study (Chapter 4), I used Android phones with small enough screens that participants did not have trouble locating elements on the screen. However, I used Kane's recommendations to place items along the edge of the screen when creating the Blocks4All environment (Chapter 6), as early tests made it clear that it was difficult to locate items on a standard-size iPad.

2.1.4 *Accessible Drag and Drop*

The drag and drop gesture is ubiquitous in applications for children, and recent work seems to indicate that children expect and prefer to use the gesture over point and click [21]. In Chapter 5, I found that the drag and drop gesture is used extensively in block-based programming environments. However, unless a developer has taken steps to provide an alternative way to perform this action, it does not work well with screen readers and is difficult for blind children to perform. I discuss my own approach to replacing drag and drop with an accessible interaction in Chapter 6, and in this section, I review the related work on making drag and drop accessible with screen readers.

I found two research projects exploring ways to make drag and drop accessible for a computer and keyboard. Somani *et al*. presented a way to make drag and drop accessible on webpages via keyboard [119]. They use a select, select, drop method where items are selected and then dropped with one key command and are navigated with another. Because there are multiple keys that can be used for different commands, the program does not switch into a selection mode when an item is selected. Instead, users can indicate whether they want to move to the next item or select an item based on which key they press. Drag and drop on web-based applications can be made accessible through a similar select, select drop mode using ARIA properties [147]. Winberg *et al*. created a version of drag and drop that gives auditory cues as items are dragged across the screen so that blind users can determine where targets are [138]. It switches between an overview and a zoom mode to help navigate a screen with many small targets. I explored using both types of drag and drop (auditory-cued and a select, select drop) in the Blocks4All environment in Chapter 6.

As of the 2017 version of the Apple iPad operating system, there are two VoiceOver methods to use 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. This method which must be augmented with audio descriptions of where you are dragging the item in order to make it truly accessible. The second is a select, select, drop method, 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 can be used to move applications on the home screen, but to get them to work within applications, developers have to do extra work. For the first method, they must provide the audio descriptions so that people know what is happening as they drag objects. For the second method they have to create a selection action and add it to a list of rotor actions available to the screen reader. It should be noted that none of the block-based environments I evaluated in Chapter 5 had implemented this option to move blocks. This could certainly replace the select, select drop interaction I implemented in Blocks4All (Chapter 6), but it introduces an extra layer of interaction in order to move the blocks. Additionally, only one of the participants who used VoiceOver in my studies was familiar with the rotor, indicating that this might be too complex an interaction for children.

2.1.5 *Touchscreen Accessibility for Low-vision*

Although people with low-vision make up the majority of people with visual impairments [148], 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 [125]. However, they found the use of zoom features (where a magnification box can be dragged around the screen) to be tedious and time consuming. They found it difficult to get contextual information when using the zoom features, as the zoomed in portion of the interface obscured parts of the rest of the screen. In my interviews with TVIs (Chapter 3), the teachers reported similarly that students with low-vision did

not like using zoom features. I took this into account when designing Blocks4All (Chapter 6) and allowed users to enlarge portions of the interface without obscuring other elements.

Crossland *et al.* [30] conducted a survey that looked broadly at how adults with visual impairments (both who were blind and had low-vision) used smartphone, tablet and e-reader devices. They found that people who were blind were equally likely as people with low-vision to own a smartphone. They found that most of their survey respondents used some type of touchscreen mobile device: (81%) of the survey participants used a smartphone and over half (51%) used a tablet. In addition to using their phones to make phone calls, send texts, access the internet and access applications, 51% of the smartphone owners used the camera and screen as a magnifier: taking pictures and then zooming in. They found that respondents with low-vision were split in what types of accessibility features they used on their smartphones: 46% used speech, 66% used large print, 43% used a big screen (either magnifying the entire screen or zooming in on a particular area), 38% changed the contrast and 26% selected different fonts. I followed up on this work in my interviews with TVIs in Chapter 3, finding how students with visual impairments also use the touchscreen devices' cameras as magnifiers, and often chose not to use the accessibility features on their phones.

2.2 Technology Use by Children with Visual Impairments

In this section, I discuss how children with visual impairments use technology. I discuss (1) the different types of assistive technology that children with visual impairments use, (2) research into how children with visual impairments are using technology at school and at home and (3) research into designing games for children with visual impairments.

2.2.1 *Assistive Technologies for Computers and Mobile Devices*

In the Individuals with Disabilities Education Improvement Act (IDEA), assistive technology is defined as "any item, piece of equipment, or product system, whether acquired commercially off the shelf, modified, or customized, that is used to increase, maintain, or improve functional capabilities of individuals with disabilities" [133]. In this section I describe the most common assistive technologies that an individual with visual impairments might use to access computers and mobile devices.

One of the most important technologies for someone who is blind to access a computer is a screen reader. This is software that interacts with the operating system which reads aloud elements on the screen and allows the user to navigate and select elements. In addition to VoiceOver and TalkBack, the screen readers designed for mobile devices discussed in Section 2.1, there are a number of screen readers designed for traditional computers. According to the WebAIM 2017 survey [146], the three most popular screen readers, in order, are: (1) JAWS for Windows, which costs over $1000 [143], (2) NVDA for Windows, a free open-source screen reader and (3) VoiceOver, which comes built-in on iOS devices and Apple computers. I used the NVDA screen reader when testing block-based applications in Chapter 5.

There are a number of hardware devices that can be used for input or output in conjunction with a screen reader. The audio output of a screen reader can be redirected to a refreshable Braille display [5], so that it can be read in Braille instead. Often these Braille displays are integrated with a Perkins chording keyboard that can be used to type in Braille, but they can also be integrated with a traditional QWERTY keyboard. There are also a number of standalone computers with Braille displays that are designed specifically for people who are blind, such as the Braille Polaris

and the BrailleNote Touch, which incorporates a touchscreen for input [149]. These computers are also expensive: the 32-Braille cell versions of both of these retail for over $5,000.

For people with low-vision, there are a number of tools that can be used to magnify the screen or change the color contrast. For most web-browsers, there are built-in text sizing and zoom controls that can be accessed on a computer via a keyboard command such as ctrl+ or on a mobile device with pinch-to-zoom. On most computers and mobile devices there is a built-in magnification window that can be found in the accessibility menu, which can be dragged around the screen like a magnifying glass (e.g. Windows Magnifier [150] , MacOS Zoom[151]). There are built-in high contrast modes on mobile devices and most computers, and you can choose to use custom style-sheets to view webpages in a web-browser. There is also a commercial product, ZoomText, a combination magnifier and reader designed for people with low-vision, and Fusion, which integrates ZoomText and the JAWS screen reader, designed for people with progressive vision loss [152].

Although people with visual impairments generally use the same input mechanisms as sighted people for computers (keyboards and the mouse if they have enough vision), it can be challenging for them to use the onscreen keyboard to type on mobile devices. This is because the keys are smaller than on a computer keyboard and are not tactile. Additionally, trying to type on a mobile device with a screen reader on is very laborious, as each key must first be found with a single tap and then selected with a double tap. Because of this difficulty, Azenkot *et al.* found that people with visual impairments are more likely than sighted people to use dictation on their mobile devices [16].

2.2.2 *Technology Use by Children with Visual Impairments*

There is up-to-date information on how adults with visual impairments are using both assistive technology such as screen readers [153] and mainstream technology such as mobile devices [30]. There are also numerous studies looking at how (mainly sighted) children are using technology [101,154,155]. However, there is not much information on how children and teenagers with visual impairments are using technology.

The largest studies that look into how children with visual impairments use technology are part of two larger longitudinal studies that broadly documented the experiences of students with disabilities in educational settings in the United States. The first was a 5-year study that followed students through elementary to high school and ended in 2005 [156] and the second was a 5-year study that followed children from preschool to elementary school and ended in 2008 [157]. These studies only reported basic information about whether children with visual impairments were using certain types of assistive technology in order to access the curriculum. Of the students without additional intellectual disabilities, they reported that 84% of the students who were blind and 14% of the students with low-vision used braille note-takers or writers to access the curriculum, and 66% of the students who were blind and 31% of the students with low-vision used computer software to access the curriculum [158]. Unfortunately, the survey results seem to indicate that students with visual impairments are underserved in terms of access to this technology. A detailed analysis of the survey data by Kelly *et al.* [62] found that "between 59% and 71% of the students with visual impairments who were most inclined to benefit from assistive technology did not have the opportunity to use assistive technology" and that students who attended received non-itinerant instruction from a TVI were much more likely to use assistive technology than those who received

itinerant instruction. The lack of current information on how children with visual impairments are using technology is a hole in the literature that I seek to remedy with my interviews in Chapter 3.

There have been smaller scale studies that look in particular at how teenagers with visual impairments are using social media. Libera *et al*. conducted focus group interviews with 14 students in Brazil on how they were using social media and mobile devices [34]. They found that the teenagers with visual impairments used social media as much as their sighted peers; however, they preferred social media applications that rely less on photos such as WhatsApp as compared Facebook. Bennett *et al*. explored this topic in depth: looking at how teenagers with visual impairments share photos on social media [25]. They conducted interviews with 14 teenagers with visual impairments. They found that the teenagers with low-vision were heavy users of photo-heavy social media platforms such as Snapchat and Instagram, and the teenagers who were blind wanted to engage with photos on social media sites such as Instagram and Facebook. They noted that the teenagers with low-vision had to use special strategies to engage with the platforms (e.g. take screen shots of shared Snapchat photos so they could explore them completely with magnification). They found that the low-vision participants used both the zooming features that were built into the applications as well as the zooming accessibility features that were built into the phones. However, this often led to usability problems, as it was hard to remember which gesture to use when switching back and forth and the gestures were sometimes misinterpreted by the applications. Interestingly, they also found that the teenagers sometimes took advantage of the ephemeral nature of Snapchat photos to take functional photos that they did not want to save (i.e. use their phone's camera as a magnifier to zoom in on some part of their environment).

There has been work exploring how assistive technology can be seen as stigmatizing for people with disabilities [114,118]. Avoiding stigma is likely to be even more important for children

in middle and high school because of the added social pressure to "fit in." In interviews with teenagers with visual impairments in Norway, researchers found that teenagers with low-vision tended to reject using assistive technologies whenever possible as they saw these technologies as stigmatizing and symbolic of dependency, but were very open toward using more "mainstream" technologies [118]. They also found that teenagers who were blind tended to be more open to using assistive technology; likely because they would be unable to access most computing technology without it. This work is in line with research by Shinohara *et al*. who conducted interview studies with 20 adults with disabilities on their assistive technology use in social settings [114]. Based on their interviews, they recommend that social acceptability should be considered when designing assistive technology and whenever possible assistive technology should be directly integrated into mainstream devices. This is likely to be even more important when designing for teenagers and middle-schoolers because of the added social pressures at those ages, and it motivated my use of mainstream mobile devices for both the BraillePlay games and the Blocks4All environment.

2.2.3 *Accessible Computer Games*

Computer-based technologies have the potential to enhance learning because they can allow for a broad range of activities that promote interaction, provide immediate feedback and allow for both independent and collaborative exploration [94]. Mobile touchscreen devices (smartphones and tablets) offer an appealing and increasingly popular platform for learning technologies for children. Mobile touchscreen devices offer advantages over traditional computers because they are mobile (people can use them anywhere) and have additional sensors (*e.g.*, a touchscreen and a vibration motor). Additionally, many children already use smartphones on a regular basis. While smartphones are generally accessible to people with visual impairments through magnifiers and screen readers, screen readers work well with text-based interfaces that use standard UI widgets.

Games, by contrast, usually have custom widgets with images rather than text and can rely on inaccessible gestures such as drag and drop. Moreover, they commonly require users to hit visual targets under time constraints, making them inaccessible. Children with visual impairments are thus unable to participate in and benefit from most of these promising educational tools

However, there are a few accessible games designed for smartphones. Both Tapbeats [64] and the Audio Flashlight [134] use the audio interface of the smartphone to be accessible. Neither game is designed for children or for educational purposes. Additionally, researchers have created recreational digital games for traditional computers as well as various other hardware platforms [12,13,27,46,86,113,142]. McElligott *et al.* [82] conducted co-design with children with visual impairments and developed several computer games with audio feedback. They emphasized the importance of using existing, mainstream platforms and designing games that can be played autonomously and cooperatively. Researchers in the TiM project [12,27] also designed accessible games that used audio feedback, but they required speakers for surround-sound and tactile overlays for a computer screen. Some researchers have laid out guidelines in building accessible games [7,15,47,82,99]. These guidelines inspired me to use a universal design approach to the design of both the BraillePlay games and the Blocks4All environment by providing both audio and visual output with high contrast for low-vision users.

2.3 K-12 Computer Science Education

In this section, I discuss how computer science is currently being taught in primary and secondary schools. As different countries are taking different approaches to introducing computer science education and my user studies are based in the United States, I focus on computer science education there. However, similar initiatives are happening worldwide. I first give a brief overview

of computer science education. I then give an overview of block-based programming environments and closely related programming environments.

2.3.1 *Overview*

There has recently been a big push to include computer science (CS) in K-12 education, and there are currently many nationwide initiatives including Code.org [159], CS For All [116], and CS for All Teachers [160] to support that goal. A 2016 Google-Gallup poll found that while only 40% of K-12 schools reported having a CS class for teaching programming (up from 25% a year ago), 76% reported that some form of CS learning is available [45]. The idea of what K-12 computer science education should look like is currently the subject of rich debate within the CS community [48,139], but many of the currently supported ideas date back to Seymour Papert's work on the LOGO programming environment for children in 1960's [100,137]. Papert suggests that early programming environments for children should have three characteristics: a low-floor (be easy to start using), wide-walls (used to do many different types of things) and a high ceiling (powerful with the ability to increase in complexity). In practice, this has meant that much of K-8 education has been taught not with traditional "professional" level programming languages and environments, but with programming environments specifically designed for children and novice programmers. These systems generally try to make it easier for children or novices to program by simplifying the act of programming in some way: making it easier to express programs and reduce syntax errors, or making it easier to understand program execution [61]. These include visual programming languages (VPLs) and their associated environments, such as Scratch [107] and Alice [40], which have a lower floor than most text-based environments because they eliminate syntax errors and allow for rapid construction of code using drag and drop interfaces. Other novice programming environments include physical computer environments and robotics kits, such as

Micro:bits [23] and Lego Mindstorms [75], which allow children to directly manipulate items. In Chapter 5 of this dissertation, I explore how these visual programming environments, specifically block-based programming environments, are not accessible for children with visual impairments. In Chapter 6, I discuss the design and evaluation of an accessible block-based programming environment used to control a robot.

2.3.2 *Block-Based Programming Environments*

Block-based programming environments and libraries, such as Scratch [78], Blockly [43], and ScratchJr [38], are a popular tool for teaching programming to primary school-aged children. They are widely used in both the Code.org [159] and the Exploring Computer Science curriculums [37]. More advanced block-based programming environments, such as Snap! [132] and Alice, have been used in curriculum for the high school AP Computer Science Principles Course [127] and introductory undergraduate computer science courses [32,40]. There have been mixed results about how effective these environments are and how well computer science knowledge transfers from block-based environments to text-based ones [1,14,72,103]. However, they have a number of features that make them appealing teaching tools for young children:

- They are composed of puzzle-piece like blocks that represent units of code (i.e. nodes of an Abstract Syntax Tree), with text, color and shapes that mirror the syntax of the language,
- Code is constructed via direct manipulation by a drag and drop interaction: blocks are dragged onto one another and snap together only if grammatically correct, making it difficult to make syntactic errors,
- Blocks are visibly nested within each other to indicate scope, and
- The code environment and output of the code are tied closely together, so that cause and effect are easy to understand.

Unfortunately, these features are highly visual in nature, meaning that block-based environments are unusable for blind children as shown in Chapter 5. My work on the Blocks4All environment in Chapter 6 focuses on exploring ways to make these same features accessible in a non-visual way. In the rest of this section, I discuss the design principles behind the three block-based programming environments, Scratch, ScratchJr, and Blockly, that had the greatest influence on the design of Blocks4All.

Scratch, one of the more popular block environments with over 20 million projects in the online repository, is an online environment designed at MIT to have a "low floor, wide walls, and a high ceiling" [78,107], and has been used as a tool to reach out to other underrepresented groups in computer science [77]. However, a case study determining if it adhered to web 2.0 guidelines for accessibility found that it is not accessible for many people with disabilities, including those with visual impairments [92]. Scratch was originally designed to be used for informal active and self-directed learning, with the goal to "make it more tinkerable, more meaningful, and more social than other programming environments," [107]. This inspires two important design goals for my own work: to make environments that support independent navigation by blind children and that allow for collaboration with low-vision and sighted children as well.

ScratchJr is a touchscreen block-based environment that was designed for younger children (5-7 years old) [38]. It has similar design goals to Scratch (low floor, high ceiling, and tinkerability), but incorporates many design ideas to make it work better for younger children. It has low number boundaries and icon-based labeling to accommodate the large amount of variability in literacy and math skills in this age group. Additionally, the designers recommend having actions that can be grasped intuitively with visible outcomes, few gestures that require

strong hand-eye coordination or fine motor skills, and a streamlined layout with a small number of blocks and menus.

Blockly is a block-based library that Google developed so that developers can easily incorporate block-based programming into their applications [43]. Unlike Scratch and ScratchJr, it is not tied to a programming environment. It uses drag and drop manipulation to create code, and color and shape indicate the syntactic properties of each block. Blockly is used in the Code.org curriculum [159].

2.3.3 *Accessible Block-Based Environments*

Lewis *et al*. did preliminary work on adapting a dataflow programming environment that they had previously worked on, Noodle [71], to create a nonvisual interface for a block-based environment [68]. This interface uses what they termed "pseudospatial" navigation using a keyboard and screen reader, where the geometry is not strictly spatial (e.g. going left and up can lead to the same place if there is a loop in the dataflow). Blocks are added by first selecting an insertion point in the workspace and then navigating through the palette to add a block. Currently, the program supports music synthesis as output.

Google has produced an accessible version of web-based Blockly [44] that uses a hierarchical HTML structure to represent blocks and block-based code. This environment also supports screen reader keyboard navigation of the code structure in the workspace and the palette, with users selecting an insertion point in the code and then pulling up a menu in the workspace. Unfortunately, it appears that the Accessible Blockly project is not currently active and neither group has reported on any user testing with their interfaces.

Although not specifically designed for people with visual impairments, hybrid text/blocks environments (e.g. Greenfoot 3, which supports the frame-based language, Stride [67,104]) are potentially more accessible programming environments than block-based programming environments [66]. I discuss them and the reasons they might be more inherently accessible in section 2.3.6.

2.3.4 *Tangible Programming*

There is work both in research [53] and in industry [65,105,124,161] on creating tangible block-based programming environments with physical blocks. These environments were not specifically designed for children with visual impairments and are currently not accessible for them (the blocks either are not able to be distinguished tactilely or the output is still completely visual). However, they seem like a promising line of research, based on the importance of tangible objects in education for students who are blind. I include four tangible programming environments in my evaluation in Chapter 5. In Chapter 6, I chose to focus on creating accessible techniques to access environments on touchscreen devices, as this allows blind children to access learning technologies using a mainstream device instead of specialized tools.

2.3.5 *Visual Programming Languages*

As hinted at by their highly visual nature, block-based programming environments use a subset of visual programming languages (VPLs), defined as "any system that allows users to specify a program in two or more dimensions" [91], implying that they might actually be more spatial than visual in nature. VPLs can be challenging to use because they tend to require a specialized IDE, programs take up a lot of screen real-estate, and they can take longer to edit than text-based languages. However, they are well-suited for certain niche areas: educational purposes (e.g. block-

based environments), to specify user interfaces, and for end-user programming (e.g. spreadsheets and LabView [162]). As Myers [91] elucidates when laying out his taxonomy of VPLs, they can be especially useful for non-expert programmers because they tend to (1) have a higher level of abstraction and de-emphasize syntax, (2) have a direct manipulation interface, as you construct a program using (often dragging and dropping) icons and other graphical objects, and (3) incorporate 2D displays of information, which can help in understanding programs (e.g. code indented by scope is much easier to understand than the same code in a single line). These are all features shared by most block-based environments, and importantly, *none of these properties are inherently visual*, so they can be presented in a non-visual way. In fact, Baker *et al*. [18] found that reintroducing 2D structure into program code read by a screen reader, by allowing users to navigate an Abstract Syntax Tree (AST) using keyboard commands, increased the speed at which blind developers were able to complete tasks with code and increased their understanding of the structure of the code. In Chapter 6, I show that a touchscreen environment with a screen reader lends itself readily to the second two features: a direct manipulation environment and a 2D display of information, which helps convey program structure.

2.3.6 *Structured Editors*

More broadly, block-based programming environments are a type of structured editor, where the unit of composition is not a character or word (although most block-based environments allow character by character editing of string literals) but a node in an AST [35]. Many of the features that are often present in VPLs that make them useful for beginner programmers—higher levels of abstraction and lack of syntax errors—are also built into structured editors. However, structured editors are not necessarily visual, which makes them useful for providing design guidelines for a non-visual environment. These editors have been around since at least the 1980's. An early

example was the Cornell Program Synthesizer [126], which was used for introductory computer science courses at Cornell and other universities. The Program Synthesizer allowed users to insert statements using keyboard commands, creating syntactically correct templates with placeholders, which indicated whether they could be replaced with other statements or with typed expressions or assignments. This created a hybrid model, where AST nodes could be completely entered only in syntactically correct spaces with a single keyboard command, and expressions were typed in. Additionally, the high-level statements allowed the program to be easily traced during runtime—the position of the cursor showed where the program was currently executing—which is mirrored by the runtime highlighting of blocks today. One element of the Program Synthesizer that informed one of my initial designs of a Blocks4All is the use of "invisible placeholders" before and after each statement, which can be navigated to using the enter command (the placeholders are visible if there's nothing else there). In block-based environments these are implicitly present as blocks can be added in between elements (and they are more explicit in the current form of Blockly which has small gaps in between each block). In my initial design, I made these explicit as the starting point for adding blocks to help to simplify navigation.

More recently, Ko *et al*. introduced the Barista framework which can be used to create "hybrid" structured editors, which still represent the code as an AST (allowing for invalid token nodes within it), but which allow for text-based editing of both the statements and expressions in visual representation of the tree [66]. These editors can use the structure of the code to provide contextual information to programmers and detect or prevent syntax errors but allow more low-level editing and use keyboard commands in addition to drag and drop to move and create code.

Closely related frame-based editors are also "hybrid" structured editors. These editors in particular are designed to combine the best of block-based and text-based programming editors

and were proposed to be a bridge between from a block to a text-based environment [67]. In Greenfoot 3, the editor which supports the frame-based language Stride, the syntactically-correct structured frames of code (i.e. “blocks” or nodes in the AST) are created using keyboard commands and can be moved using keyboard commands or drag and drop. Expressions are then directly typed within the frames [67]. Although I have not tested these environments with a screen reader, they are more inherently accessible because they use keyboard commands instead of drag and drop to move and place blocks, and the structure of the AST could be interpreted by the screen reader in the same way as any other structured tree. Additionally, the text-based expressions would likely be more readable with a screen reader than the block-based expressions, which can be hard to access with a screen reader.

2.3.7 *Touchscreen Programming*

With the rise of mobile devices, researchers are starting to create programming environments for touchscreen devices. A group of researchers at Microsoft Research created TouchDevelop [20,130], a mobile application used to design mobile applications, designed for older students and hobbyists. I evaluated the accessibility of this application along with other block-based environments in Chapter 5. Like the Cornell Program Synthesizer, TouchDevelop uses a structured editor for statements, and a less-structured editor for inserting expressions, which relies on auto-completion to insert complete tokens. It uses a text-based language, like BASIC, but it is limited to specific syntax and predefined blocks. It is designed to be easy to use on a touchscreen: a statement can be selected with a finger press, then a keyboard pops up with expressions that can be added within the statement, as well as buttons to add statements before and after. The application does not rely on drag and drop to add code from a palette to the workspace, instead the

user selects a spot to edit in the workspace and then selects an expression or statement to add from a list.

Another code editor designed for touchscreens is RefactorPad [106]. Unlike TouchDevelop, which uses a structured editor so that text can quickly be entered using a touchscreen, RefactorPad focuses on using gestures to edit and modify text-based source code, as opposed to entering it. The researchers used a guessability study to design RefactorPad, where they showed the output of a common refactoring task and asked participants to supply a gesture. The most common gestures were then presented as design guidelines. Although many of the refactoring tasks are likely to be used in a block-based environment, such as selecting, duplicating and moving code, the actual gestures recommended either conflict with common screen reader gestures or seem specific to sighted users (e.g. the undo/redo gesture is drawing a circle), so would not be useful in a non-visual application.

2.4 Accessible Computer Science

In this section, I discuss (1) broadly how STEM education is made accessible for students with visual impairments, (2) approaches to making computer science education more accessible and (3) efforts to make non-block-based programming environments accessible.

2.4.1 *Accessible STEM Education*

Students with visual impairments and other disabilities are underrepresented in all STEM (science, technology, engineering and math) fields, including computer science [95]. One reason for this is that much of the current computer science curriculum and many of the tools are not accessible. Researchers are looking to increase the accessibility of computer science with this population. Studying the use of audio assistive technology for students with visual impairments in STEM

fields, Nees and Berry [96] found that the majority of students with visual impairments went to mainstream schools and participated in activities with their sighted classmates, signaling a need for educational curriculum that is accessible to both. While they found that there has been widespread adoption of text-to-speech for presenting digital text, there is a need for research in making other items accessible, such as graphics and text, which are often present in STEM textbooks and on standardized educational tests.

2.4.2 *Accessible Computer Science Education*

Much of the research in computer science has looked at using audio to increase access to tools and curriculum by students with visual impairments. Sánchez and Aguayo [112] created the Audio Programming Language (APL), an audio-based programming language and associated environment to teach students with visual impairments how to program. Stefik *et al.* [122] designed a programming environment, Sodbeans, a plug-in for NetBeans, which relies on audio cues to convey runtime and debugging information. They also developed Quorum, an accessible programming language, and curriculum that was accessible for students with visual impairments. In observations of students at a school for the blind, the authors found that students did best when introduced to concepts through objects they could physically manipulate, with more hands-on projects than lecturing, and when they were able to work at their own pace. This indicates that accessible block-based environments such as Scratch, which encourage hands-on projects and working independently, might be useful with this population.

Although most of the research involves using audio to develop curriculum and tools for students with visual impairments in computer science, a few studies have looked at incorporating tactile information. Kane and Bigham [56] ran a four day workshop with high school students with visual impairments using twitter data to create a 3D printed visualization of fake crisis data. They

found that the students enjoyed using the 3D printed graphics. However, the authors also found that the students spent a great deal of time making impromptu "music" using VoiceOver audio and seemed to enjoy getting immediate feedback as opposed to the delayed gratification of a 3D printed graphic. Ludi *et al.* [75,76] looked at using tangible output for teaching programming and designed a development environment and curriculum to teach robotics to high school students using the Lego Mindstorms robots. Because the traditional block-based IDE for the Lego robots is Robolab, which is not accessible with screen readers, the authors developed a text-based language and environment for the students to use instead. 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 [128]. In line with this work showing that robots are an accessible and tangible way to learn programming and that immediate feedback is motivating for blind students, I used the Dash robot [141] for output in designing the block-based programming environment in Chapter 6.

2.4.3 *Accessible Programming Environments*

In addition to developing curriculum to teach programming, there has been research into creating better tools for blind programmers and understand their needs and the challenges they are likely to encounter. A number of studies have found that blind programmers have a difficult time understanding where they are in code in terms of scope without visual cues [2,3,56,112], and at least two projects have looked at solutions to this problem. Baker *et al*. [18] created an eclipse plug-in that allowed programmers to navigate a modified AST to help understand the structure of their code, and Stefik *et al.*[121] injected verbal cues to indicate how deeply nested a statement is during runtime execution of the code. In Blocks4All, I used the spatial layout of the code on the touchscreen to help make the scope of nested statements more clear.

Chapter 3. TECHNOLOGY USE BY CHILDREN WITH VISUAL IMPAIRMENTS

In this chapter I answer the following research questions: *Q1: How are children with visual impairments using touchscreen computing technologies for learning?* And *Q2: What are the challenges that children with visual impairments face when using touchscreen technologies?* I present evidence that supports the first claim of my thesis: Children with visual impairments use mainstream touchscreen devices for learning in a variety of ways, despite accessibility challenges that they must overcome to use them. This work was done in collaboration with Catherine Baker and Richard Ladner.

3.1 INTRODUCTION

Prior work has shown that individuals with disabilities can feel self-conscious about using assistive technologies or accessibility features on mainstream devices [114,118]. They can also feel frustrated by the usability of these devices [125], which can lead them to abandon these devices [102]. Recent work suggests that the social issues can have an even bigger impact on teenagers and tweens than adults because of the added social pressures in middle and high school [25,118]. However, there is little work looking at how the social and usability challenges impact the use of technology by teenagers and children with visual impairments in educational settings.

Many individuals with visual impairments have their first introduction to assistive technology in school. With the advent of the computer, students have begun to use a wide variety of computer-based assistive technology (e.g. screen readers or optical character recognition to read text) in order to access information. Because of laws like IDEA [133], they can receive additional support and training on this technology from TVIs while in primary and secondary school to ensure that they

are able to access educational curriculum to the best of their ability. However, to the best of my knowledge, the last look at how children with visual impairments are using technology in educational settings was a set of surveys completed in 2005 and 2008 [157,158]. These surveys looked broadly at how students receiving special education services performed at school and documented the services they received, through surveys with school staff and interviews with parents. However, although the surveys asked if the students used assistive technology, they did not include very detailed information about what types of technology the students used or how often. Additionally, as touchscreen-based tablets have become more popular in educational settings and the accessibility on them have improved greatly since then, I was interested on expanding on and updating this work focusing more on the technologies used by the students.

I sought to gain a better understanding of how children with visual impairments are using both mainstream and assistive technologies in educational settings through semi-structured interviews with six TVIs. I and one other researcher interviewed teachers who worked with students ages 5-21 from both residential and mainstream schools. I sought to answer the following questions:

- TQ1: What factors do TVIs take into consideration when selecting and teaching technology for the student to use?
- TQ2: What factors do TVIs think affect the students' use or non-use of technology?

I was interested in exploring these questions from the perspective of the TVIs, as they work with the children to select and train them on technologies they will use. I was interested broadly in what technologies students with visual impairments are using in educational settings and what challenges they encounter in using these technologies. I wanted to know what role teachers can play in alleviating some of the challenges to use technology.

I found that all of the teachers interviewed taught touchscreen mobile devices, the iPad in particular. In some school districts, all students (including sighted students) in a certain age group were automatically given an iPad. In other districts, teachers used iPads especially with students with visual impairments. This is in part because the accessibility features, such as the screen reader, zoom functions and color contrast, work well on the iPad, but also because of the other features on the iPad. For example, students can use the camera as a video magnifier or the built-in sensors on the phone to help with orientation and mobility. However, the teachers did mention some problems they encountered when trying to get students to use the touchscreen devices. First, the students did not want to stick out. Even when using a mainstream device such as an iPad, a student might feel uncomfortable if they are the only one using it. Students, especially those with low vision, sometimes opt not to use the accessibility features such as zoom or color contrast if they feel it makes them stand out. Additionally, teachers noted some usability problems with the devices: the screen reader gestures were hard for some of their students to learn and some applications were not accessible with the screen reader.

In this chapter, I provide a better understanding of how children with visual impairments are using technology in educational settings and the roles that TVIs play in selecting and training students on these technologies. I found that students with visual impairments face a number of challenges in using these technologies, as they are complicated to learn, often have usability problems and often have bugs. Based on these challenges, I discuss the limitations of current technologies and describe three areas of opportunities for design. This can be used to inform future work on designing learning technologies.

3.2 METHOD

To answer my research questions, I, along with another researcher, conducted semi-structured interviews with TVIs to determine how children were using and being trained to use technology at school.

Table 3.1. Teachers of the Visually Impaired who were interviewed

Participant	Gender	Job Type	Job Description
P1	F	Itinerant (Small School District)	Meets one-on-one with students to teach everything from self-advocacy skills to learning to use different types technology. Works itinerantly to support students in one school district (grades K-12 and transition ages 18-21).
P2	F	Residential	Teaches a STEM subject and focuses on college prep and reading tactile graphics, etc... Works at a residential school for the blind (grades 6-12 and transition).
P3	M	Itinerant (Multiple School Districts)	Provides consultative and direct instruction services on using assistive technology to TVIs to many schools. Works itinerantly to support TVIs and students (grades K-12 and transition).
P4	F	Residential	Works one-on-one and in small groups with students on learning technology. Meets students both in and outside of the classroom. Works at a residential school for the blind (grades 6-12 and transition).
P5	F	Itinerant (Large School District)	Meets one-on-one with students to teach expanded core curriculum (technology, travel skills, independent living skills). Works itinerantly to support students in one school district (grades K-12).
P6	M	Itinerant (Large School District)	Meets one-on-one with students to teach expanded core curriculum (technology, travel skills, independent living skills). Works itinerantly to support students in one school district (grades K-12).

3.2.1 *Participants*

I recruited six teachers (two male, four female) from my contacts within the community, word of mouth and snowball sampling. I drew from the population of approximately 90 TVIs in Washington State [84]. Approximately 84% of students with visual impairments are educated in their local schools [6]; because of the low incidence of visual impairments, there are generally very few students with visual impairments in a single school. Therefore, the majority of these teachers work itinerantly, traveling between mainstream public schools in a single district to work with students. According to a 2007 report, 59% of the TVIs in Washington State work in the state's

school districts as either itinerant or resource room teachers, 23% work with students at the state residential school for the blind and 8% are based at the residential school but do outreach and provide support across the entire state. The remaining TVIs are contract workers or work for the Department of Blind Services or Washington Sensory Disability Services [24]. I interviewed teachers from each of these populations: three itinerant teachers, two teachers who work primarily with students at a residential school for the blind, and one who is based at the residential school, but travels to provide assistive technology support to TVIs across the state. Of the itinerant teachers, two worked in a large school district where there were multiple TVIs and one worked in a small school district (<10,000 students), in which she was the only TVI.

I started recruitment through my existing contacts in the community. I contacted three teachers via email asking if they were interested in doing the interview with me. During the course of these initial interviews, the teachers recommended that that I reach out to the remaining three teachers as they were very knowledgeable about assistive technology. This means the TVIs in my interviews likely had more expertise on assistive technology than the general population of TVIs. All of the teachers that were contacted agreed to be interviewed. Teachers were paid $35/hour for completing the study.

3.2.2 *Procedure*

I conducted semi-structured interviews that ranged in length from 20 to 60 minutes. I audio-recorded the interviews for data collection purposes. Participants were asked broadly about their work, what kinds of technology they use with students, how they chose to introduce these technologies and what types of problems they encounter pertaining to the technologies (Appendix A). My collaborator and I formulated the interview questions based on our existing knowledge and prior work on (1) how both adults and children with visual impairments use technology

[125,146,156,157], (2) how students with visual impairments are taught to use technology [8,28] and (3) barriers that adults and children with visual impairments encounter when using technology [25,114,118,125]. We narrowed down the questions using our overarching research questions to ensure the interviews were not longer than an hour.

3.2.3 *Analysis*

The audio-recorded interviews were professionally transcribed. We used grounded theory to analyze the interviews. During data collection and analysis of the first two of interviews, my collaborator and I brainstormed and developed codes (Appendix B). Then we independently coded up the rest of the interviews, meeting to discuss any disagreements or new codes. When we disagreed on a code, we discussed the issue until we reached agreement. We continued data collection and analysis until we reached saturation: that is, no new strategies for introducing technology, technology use patterns or challenges emerged. After coding all the interviews, we developed themes using axial coding [123].

3.3 FINDINGS

3.3.1 *Technology Use and Selection*

The TVIs described a number of technologies that they used with their students, including both mainstream tools (e.g. computers and tablets) and accessibility features (e.g. screen readers). I summarize the different types of technology in Table 3.2.

Table 3.2. Types of Assistive Technology Described in Interviews

Technology	Description	Examples
Refreshable Braille Display	Display for screen reader output that uses small metal pins that pop up to create Braille characters. Often integrated with a Perkins (Braille) or QWERTY keyboard.	Brailliant Series by Humanware [163]
Braille Notetaker	Stand-alone computer (with similar computing power to a smartphone) with Braille display.	Braille Polaris, BrailleNote Touch [149]
Video Magnifier/ Closed-Circuit Television	Video camera that projects a magnified image onto a display (e.g. computer or television screen).	Various stand-mounted and handheld cameras work with screen or head-mounted displays
Screen Reader	Software that reads aloud elements on the screen and allows user to navigate and select elements.	JAWS [143], ChromeVox [164], NVDA [165], VoiceOver [144], TalkBack [42]
Magnification Tools	Features that can be used to adjust text sizing, text color or zooming. Magnification windows that can be dragged around the screen like a magnifying glass.	Mainstream (Pinch-to-zoom, ctrl+) and accessibility features (Windows Magnifier [150] and MacOS Zoom [151])
Magnification Software with Screen Reader	Stand-alone software that combines magnification tools with some screen reader features.	ZoomText (magnification features with limited screen reader) and Fusion (combination of ZoomText and JAWS) [152]

3.3.1.1 Types of Technology Used

I found that the TVIs train their students on a wide variety of technologies. In terms of hardware, students with visual impairments use iPads, sometimes connected to a Bluetooth keyboard, as well as laptop or desktop computers. Students who are blind might use a Braille display connected to their computer or even a stand-alone computer with a Braille display instead of a screen (called a Braille Notetaker). However, because these devices are expensive, they are limited to students with

high academic ability. P1 noted that they have to "triage" in determining how to buy AT for her students. She said that "you make sure that your highest need person...by highest need (I mean) academic rigor (combined) with cognitive ability. that person he gets everything he can." Similarly, P2 noted that even at a residential school for the blind, cost is a consideration in purchasing the specialized Braille Notetakers and that they generally only give them to seniors who are headed to college.

Students with low-vision also use video magnifiers. Multiple teachers (P1, P3 and P4) note that these are one of the more challenging pieces of technology to get students to use, mostly for social reasons. According to P3:

> *The other assistive technology... they're always going to use the Braille...but people will get a low-vision magnifier and because of social reasons, for whatever reasons, they won't use the video magnifiers.*

In terms of software, the blind students learn whichever screen reader matches the type of device they are using: ChromeVox for Chromebooks, VoiceOver for Apple products, and JAWS or NVDA for Windows PCs (P4 noted that they had tried to use the less popular WindowEyes and Narrator screen readers in the past but were no longer using them). Oftentimes the screen reader dictates the device that the TVI teaches the students: P2, P5 and P6 all stated that the superior accessibility features on the iPad mean that they use that tablet with their students instead of an Android tablet. P2 explained that "the school just teaches VoiceOver because it's much better."

The low-vision students use a variety of magnification software on their tablets and computers. These include (1) the built-in "mainstream" features (e.g. pinch-to-zoom on a tablet, ctrl+ in a web-browser), (2) the built-in "accessibility" features (e.g. the accessibility magnifiers which zoom in on a portion of the screen or the color contrast) or (3) additional magnification

software such as ZoomText or Fusion (which combines the magnification of ZoomText with the JAWS screen reader). Three of the TVIs (P1, P5 and P6) noted that the built-in "accessibility" features for low-vision are generally not good and their students prefer to use the built-in "mainstream" features or the additional magnification software. In the words of P6:

> *We used to teach the magnification that's built in like Windows and stuff. It's pretty awful. To be honest, it's really awful….you set it on a little window of magnification that zooms in on part of the screen and it's hard to keep track of where your mouse is, you can make the mouse a little bigger but not that big...The little squares, those things don't really, I don't think I've ever met anybody that likes that….and I don't think it's any better on Macs either.*

Finally, many of the TVIs (P1, P2, P4, P6) expressed that many of their students seem to prefer using more mainstream devices. In the words of P2, "If we could teach them normal technology that's better. Voiceover is just part of an iPad, Siri is just part of an iPad, right, there's no special program that was downloaded." This may be due in part to a desire on the part of the students to fit in.

3.3.1.2 Mobile Touchscreen Devices

All six of the teachers mentioned that they used mobile touchscreen devices (iPads in particular) with their students for multiple reasons. One of the main benefits is that the built-in accessibility of iPads is extremely good, making it better for students with disabilities. P1 said, while she wants to migrate her blind elementary school students to Chromebooks to match the district, they are currently mostly using iPads because the screen reader works better and interfaces more easily

with Braille displays. When asked if the use of iPads was particular to the students with visual impairments or was reflective of the technology the sighted children were using, P5 replied:

> *No, it's very particular to students with visual impairment, it's a very useful technology, I mean if you have a low-vision student they can take pictures of things on the board and enlarge it. They can take pictures of materials in their book and there's software now that you can take pictures of a math page and enlarge it and actually work on it in an app in the iPad. There's typing programs if the font isn't very big you can make a black background with white lettering. There's just a lot of amazing apps and iPad is portable.*

P1 echoed this and noted how her low-vision students repurposed the iPad, a mainstream device, to take the place of physical magnifiers (and the old-fashioned practice of note-taking):

> *High schoolers are really big on that, you know they will be in a math class, they will walk up to a white board and take a picture and they won't have to take notes. Yeah, they love it. and they will do it to read too, like they will take a picture of a worksheet and then zoom in on it. It's really cool. iPads are life changing.*

Both P5 and P6 noted that the portability of the iPads and iPhones made it useful both to replace video magnifiers and to help with orientation and mobility training. P5 said it was useful even for her elementary school students because they had to move around to different classrooms. Because of the portable nature of the iPad, the TVIs found they were useful beyond the classroom and would use them for mobility purposes as well. P6 mentioned that his students use travel applications with point-by-point directions or applications that let them know about nearby stores and intersections.

Many of the teachers said that iPads are particularly useful for younger students or students with additional disabilities, especially those with motor impairments. According to the APH, students with multiple disabilities make up approximately 50-55% of students with visual impairments [6]. P3 noted that the Switch Access (used by individuals with motor impairments to operate the iPad with an adaptive switch, such as a large button or a "sip and puff" straw)[166] is very good on the iPad and interfaces well with the VoiceOver screen reader. Because of this, he mainly uses the iPads with his students that have additional impairments. P5 and P6 explained that many times younger children (elementary and preschool aged children) have exposure to mobile touchscreen devices. P5 said:

> *These days even at the preschool level many of them have had exposure to touchscreens: like tablets, iPads, their parents' phone. I'm finding that a lot because I do all the preschool check-ins for public school, so I see every single student that comes in and most of them do have early exposure to those technologies.*

P6 stated that they often start young children with iPads and then transition them to computers when they learn to type: "A lot of times younger kids will start with iPads, we try to get them typing pretty early too, so then once they kind of start typing then we can get them on computers." P2 noted one reason it can be easier for blind students to learn to use an iPad is because they have do not have to learn the hundreds of key commands that they would have to on a laptop because the iPad is gesture-based.

One of the challenges both the blind and their low-vision students encounter in learning to use the touchscreens is how to explore the screen. The students with visual impairments have to remember to explore the whole screen with the magnification turned on. In the words of P1, "they either zoom with the pinch and enlarge or they actually set in settings to make it large all the time

and then of course they have to swipe back and forth and up and down all the time to see the big picture." For students who are blind, the TVIs teach specific search strategies to ensure they explore everything on the screen. For example, P1 teaches a search pattern where they move systematically move back and forth across the screen starting in the top left corner. She said that "without a doubt in my mind, they are using their visual cortex... they know that certain apps are down there on the bottom, they know where the apps are, and they have a visual mental map for sure." However, even with this mental model, she noted there are still challenges for the students:

> *The problem is that it's so easy for, especially for a totally blind person, to get disoriented when you are just in space... I can use my eyes and go straight to the app. The minute I close my eyes, I start to go like this <veering downward> and I don't know that I'm doing it… I think I am going across, but I'm veering. -P1*

The TVIs mentioned a few other challenges for using the iPad. Some children have difficulty performing the VoiceOver gestures (double tap to select or twist to engage the rotor). There are also usability problems (e.g. certain features that move focus, like the spellcheck or find, do not work well with screen readers).

3.3.1.3 Matching the Technology of their Peers

The teachers teach their students a similar set of technologies regardless of whether they are at a residential school for the blind or a mainstream school; however, they do try to match the technology of the school district when possible. In P1's school district, this means the students use Chromebooks at the elementary and middle school level and then iPads for high school. However, she noted that because of accessibility problems with integrating Braille displays and ChromeVox (the screen reader for Chromebooks) her blind elementary school-aged students are using iPads

instead. At the residential school for the blind, P2 and P4 teach at, the middle schoolers receive iPads and Bluetooth keyboards and the high schoolers receive PC laptops. However, P2 notes that they tweak what technology is available to students to match their personal needs:

> *Many of the students have BrailleNotes or other Braille note taking devices. Those decisions little tweaks and twerks are decided by the technology team. Some of our high schoolers have iPads, some of our 8th graders end up with a laptop, it just kind of depends on what they're using it for and what will fit the best.*

Two of the TVIs (P1, P6) mentioned that matching the technology of the other students in the class makes it easier for students with visual impairments to collaborate with their peers. P1 described a situation in which a blind child could collaborate with a sighted peer or teacher while using an iPad or a computer:

> *Everything is right there on the screen and so a sighted person just reads the screen and the blind person reads the Braille display…. It's really nice for teachers too because they can see the kids work as they work. Yeah and the same thing with a MacBook Pro. The other kids can use the keyboard to answer stuff and the blind kids can use either the keyboard or the Braille display. What a difference from years ago you know, it's much easier.*

P6 echoed the idea that having students using the same technology makes it easier for them to collaborate and noted that they usually will "train" the sighted student, so they have some familiarity with the assistive technology.

3.3.1.4 Other Factors in Selecting Technology for a Student

In determining what technology a student should use, P4, who determines the technology needs of students at a residential school for the blind, follows a set framework: "you consider the student, the environment, the task and then finally the tool that you're going to use." For example:

> *Let's say we have a student who will, let's start with a Braille student first, they're still learning the Braille code, then a device like a Braille Note would be extremely helpful for them because they would be practicing the reading and writing of Braille all throughout the day….So it would make sense for them to be on a device like that versus having them learn keyboarding and a laptop. -P4*

She elaborated on how the environment affects the choice of tool, and that they might chose different tools to support a student in a school environment versus a work environment or for learning orientation and mobility. P2 mentioned that different classes help reinforce different skills. For example, students are more likely to practice typing in English and social studies classes where they have to type essays as opposed to math classes.

Cost was cited as a concern by four of the TVIs, although the concern level varied based on what type of school they taught at. P4 explained that cost was less of an issue at the residential school: "we are lucky here we do get the tools and stuff the students need usually, and so financial stuff isn't an issue." P2 concurred "if we can hand it to them it's no longer a financial barrier...if…they learn to use it and they can prove they need it for future times, blind services will buy it for them." However, P2 noted that she advocates for her students to learn a laptop and refreshable Braille display instead of the all-in-one notetakers (e.g. BrailleNote) because the notetakers are so expensive and would be challenging to replace, even if they originally receive one for free. In terms of software, both P1 and P6 noted that they preferred teaching their students

to use the free or built-in screen readers (e.g. NVDA and VoiceOver) to JAWS because of the expense.

The TVIs mentioned a number of other factors that they consider when selecting technology: the student's academic ability, vision and personal preferences. As might be expected, students who are blind generally require more technology and training than students with low-vision. Students with higher academic ability are under more pressure to keep up with the general education curriculum, and so there is more pressure to ensure they have access to all the best assistive technology. The TVIs also keep in mind personal preferences of the students when selecting technology. These range from an aversion to synthesized speech to a strong desire to use only familiar technology. P5 noted the importance of considering a whole student when picking out their technology:

> *I really try to meet people where they're at, you know, and like I try to figure out, like ... what is doable...because she spent the last three years learning voiceover on the iPad and she doesn't ever use it, so I'm not going to do that with her.*

3.3.1.5 Technology Skills and Experience

The TVIs in my interviews thought that often their students do not have as much exposure to technology as their sighted peers. When asked about what kind of experience her students had with technology, P2 said:

> *If they're a young middle schooler, which is when we get a lot of our kids, not as much technology as your average middle schooler. Most of our average middle schoolers out there have a cell phone, have an iPad, have a, have a, have a, I would*

say 50% of our kids don't... so learning how to navigate any of that is taught as like a new skill in middle school.

Three of the TVIs (P2, P4, P5) highlighted learning typing as one skill that is both especially important and challenging to teach students with visual impairments. P2 said "This year we had a big group of 6th graders and none of them knew how to type. You could give them a number pad and they can't dial home." She further noted, "they have been really resistant to learning to type they just want to dictate everything or ask Siri everything." P5 said that it's very important for her students to learn to type as often students with low-vision have bad handwriting, so she starts teaching them to type as early as possible. Both P2 and P4 noted that traditional typing programs are not accessible, and while accessible versions exist, they tend to be less engaging and fun.

3.3.2 *Technology Reinforces Disability Identity*

In my interviews, a common theme that arose is the relationship between the technology the students are using and their disability identity. While for some students this is a positive relationship (I can do something others cannot, and I am proud of it), for many of the students the relationship is negative. These associations between the technology and disability identity can be both external and internal. For some students, the external perceptions are the largest factor in their acceptance or rejection of technology. They do not want other students to perceive them as being different, having a disability. For other students, the internal perceptions are the largest factor in acceptance or rejection of the technology. These students do not perceive themselves as having a disability and therefore do not need the assistive technology.

3.3.2.1 External Disability Identity

For many students, the resistance to using assistive technology comes from managing other students' perceptions of their disability. They do not want other students to notice their disability. P6 noted, "I mean a lot of kids who are afraid to stick out especially like in middle school and stuff, you know, and they start not wanting to use their devices and everything because they don't want to stick out with different everything." P1 thought that this is particularly true for students with low-vision: "Especially kids with quite high acuities, relatively high vision acuity and their vision is stable, nothing is getting worse, those kids for sure they like to pass."

Because these students do not want technology that will draw attention to their disability, they will resist using technology that is different from other students. When given the choice for magnification, many students will select the built-in technology that is used by both visually impaired and sighted students over technology that would be specific to someone with a visual impairment. For the iPad, this means that students are less resistant to pinch-to-zoom over using the magnifier box:

> *Oh, everybody's doing it [using pinch to zoom] all the time. And that and you know they can see their sighted peers doing it too and then it's just no big deal. Whereas if you've got that magnified box, you're going to stand out. Partially sighted kids tend to be quite guarded about their peers knowing. -P1*

For technologies that do not have an unobtrusive option for the student to use, they may choose to just go without access. For instance, there is a technology that allows a teacher's screen to be transmitted to the student's iPad so that they can see it more easily. However, P1 often found that the students would not use the technology: "They would rather not see and just imagine." In order for this software to work, the teacher must explicitly invoke it. If the teacher forgets to turn it on,

then the student has to ask for it to be turned on, thereby drawing additional attention to their disability, which may be why they choose to go without.

The resistance to technology is not just limited to assistive technology. Mainstream technology can still make students stand out to their peers if they are the only one using it:

> *I mean I've had students who don't like to use iPads because they are the only one that has an iPad, you know. I think if all the kids are using something then they just have an accessibility feature going on, they don't mind that as much. But, as soon as it's something that only they're using, a lot of times they don't like that. So, like I said, if a kid is the only one that has an iPad that kid won't want to use it because like I said, they don't want to stick out. - P6*

3.3.2.2 Internal Disability Identity

While some students are resistant to use technology due to perceptions of others, some students are also resistant as the technology they are provided does not match their internal disability identity. Many technologies are associated by the students with specific levels of vision impairment. P4 stated: "With JAWS I think there's some people that think it's so hard to use and that they don't have the experience with it, and they also know that the people who use JAWS don't have vision or very usable vision and they don't want to perceive themselves that way." When students do not believe that their visual condition matches the technology they are being taught, they are resistant to learning it as they don't believe they need that technology.

For students that have a degenerative condition, this can be particularly important as they technologies they are learning will likely not match their current level of vision. Many TVIs will

begin teaching the students the technology the students will need as their condition worsens before they reach the state that they need it. However, this can be very hard on the student:

> *It's just the emotional, mental, I mean they're young and it's hard to grasp that their vision will be getting worse and there's the denial, there's you know, all that kind of stuff that goes with it. If you say JAWS, the students here know, they know who the JAWS users are and right now they know how different they are from the JAWS users in terms of their vision. -P4*

If they are in denial about their vision worsening, then they will not be open to learning the technologies that are not currently needed but will be needed in the future and take time to learn.

The association of technology with disability identity is not always negative. There are some students who are proud of the skills they have gained from the technology they use. They see that they have unique skills that allow them to do things that other students cannot do:

> *It's interesting and those are usually my older students who have kind of come to terms with it, they're finding some pride in their identity as, 'Okay this is part of who I am and it's not a bad thing, it makes me unique, I can do something most other people can't, yep, I can read in the dark.' I am also a Girl Scout leader ... and we went camping. 'Alright lights off at midnight!' Okay, they all had Braille books, they could read. - P2*

3.3.2.3 TVI Framing of Technology

Due to the issues related to the associations between technology and disability identity discussed above, I found that the TVIs will often frame their introduction of the technology or skills to combat some of the negative associations.

For instance, when teaching a student who uses magnification software the keyboard commands for actions such as closing an application, one TVI mentioned trying to link the less common keyboard commands with those that are commonly used by many people (e.g. the copy and paste keyboard command):

> *So, I'll say to the kids hey you know, a lot of the fully sighted kids would just love to know these commands and they'll wonder you know, if you start using these really fluently, they're going to think you are super cool. How did you do that, you did that so fast, right. … Yeah so, I try to get value from them on that score. It's just faster and easier and your peers will love it because you will be fast. -P1*

Since some students have been resistant to learning these skills, potentially due to their impressions that they are only used by someone who has a disability, the TVI has framed the skills as mainstream technology skills, instead of assistive technology specific skills.

Other TVIs have run into issues with students who have degenerative vision condition that will get worse. They associate certain assistive technologies with different levels of vision. This can be an issue when the TVI is trying to introduce the technology that they will need when their vision worsens. Many technologies are associated with specific levels of vision, so some students are resistant to technology which doesn't match their current vision level, even if they will need it in the future. As students do have these associations between certain technologies and vision levels, some TVIs will focus avoid mentioning JAWS, while still ensuring they learn the skills they will need to use it in the future:

> *I haven't talked to them about ZoomText, haven't talked to them about JAWS, have just gone directly to Fusion, because right now it's the magnification they need. So,*

> *voice is there too, and they're learning all the JAWS stuff without even knowing that that's what they're kind of learning… JAWS is the screen reader that runs Fusion so they're learning all the JAWS commands and they don't kind of even know it, which is kind of a good thing because they oftentimes have a hard time with this transition. -P4*

By using a technology like Fusion, which combines ZoomText and JAWS into a single product, the TVIs are avoiding the resistance to JAWS. Fusion blends the magnification software and the screen reader, which means it does not have the same associations with only being needed by students with a certain level of vision like JAWS does.

3.3.3 *Challenges in Learning Assistive technology*

Another theme that came up repeatedly in my interviews was that the assistive technology was challenging to learn and that coming up with motivating examples to get the students to learn was an important part of the teachers' jobs. I found that the technology was hard to learn both because it introduced extra layers to learn on top of whatever content the student was trying to access and because a lot of the technology was buggy or not user-friendly.

3.3.3.1 Learning Extra Layers

Both P1 and P2 repeatedly emphasized that their students had to take additional steps and learn additional skills in order to access the same materials as sighted children. In describing one student, P1 pointed out:

> *He had to learn how to keyboard, he had to learn how to listen, and he had to learn how to separate what I call everybody commands, you know standard keyboard*

> *commands like command-q for quit, command-w for closing a window...so separating the everybody commands from VoiceOver commands.*

She noted that her blind students have to learn how to listen to their screen readers and still participate in group work or listen to their teachers. P2 noted that it was hard to learn many academic skills and even harder to learn them in conjunction with assistive technology:

> *I use MathType that translates into LaTeX that will then emboss (it) with so much less work on my end, but it's teaching them yet another code to be able to do their math.... It's just one more layer to learn before you can learn, it's just one more thing that slows our kids down, just one more step.*

Any of her students who use Braille would also have to learn a typesetting language (such as LaTeX) if they wanted to write math equations that a non-Braille reader could understand. While this could be a useful skill to have, it also can be a barrier when they are just trying to learn math.

3.3.3.2 Usability Problems and Bugs

All of the TVIs mentioned encountering usability problems and bugs with the assistive technology. These challenges included curricular materials that could not be accessed with the assistive technology: web-based programs that were missing alternative text or labels, or testing materials that did not allow for magnification.

Some of the problems were bugs or usability issues with the assistive technology itself. P1 mentioned usability problems with the combination of Chromebooks and ChromeVox:

> *Last year district tech, a nice person, announced that she was going to help me achieve accessibility (on Chromebooks) and I said, "I would love your support" and*

we worked for three hours and just couldn't get in. It's just we'd get you know 85% there and then roadblock. Horrible.

P1 developed strategies to shield her students from some of the worst of the usability problems. When describing why she had not obtained a BrailleNote Touch for one of her students, she said:

We know it's going to be buggy, so give it two years and then cause I'm not going to take a fourth grader and make him alpha test. It'd be different if he was an adult, but no, I'm not going to do that to him…. I just want him to have confidence.

She noted that the usability problems were particularly problematic for children:

With a little fourth grader, entering fourth grade, you can imagine, I just want him to have confidence…. [When] they can't navigate, it's depressing for them, they can't count on it. It's hard for us as teachers too to say, 'Hey this is going to work and then it doesn't.'

3.3.3.3 Little Support for Independent Exploration

In part because of the usability and social issues, many of the teachers thought that the assistive technologies (especially for the high need students) do not support independent exploration very well and are not easy to learn without additional support. When asked if the students were likely to use the technology on their own, P5 replied, "It's not going to happen because it's just, yeah, I don't see it. I just think about all my students, like my Gen. Ed. students..., It's just not going to happen, they don't do it." P4 thought that a recent policy change at her school in which students were given iPads before receiving any formal training meant that they often learned "bad habits"

or shortcuts which meant they were more resistant or had trouble learning the more efficient accessibility features:

> *They're not perhaps learning the most efficient and effective ways to do things...Particularly for the iPad with Siri, they just rely on Siri to do most things…(They use) the on-screen keyboard instead of the Bluetooth keyboard, which is impacting their typing skills because they're most often using just their one hand and one finger to find the different keys ...So that's definitely the downside to just having the students have it and they learn on their own and explore on their own. In some ways that's great it's student generated, but they're kind of getting set in some of those ways too.*

3.3.3.4 Technology Support

One of the factors that can impact a student's success in learning different technologies, is the level of support they receive. P5 notes that if the student doesn't have support in learning the technologies, "it's just not going to happen." This support comes primarily from the TVIs but can also come from the students' other teachers or their parents.

The primary support comes from the TVIs, who may have to learn the technology independently. P2 said about learning AT:

> *It's not something you learn in a TVI program, … you don't really get like a TVI degree in AT, it's not a thing…. It's a little splattering but it's the school you went to, did they have things for you to play with? Did they actually teach you how to use it or did they just say, 'Hey these are things that are available'? It depends on where you went to college how much AT training you got.*

Many of the TVIs I talked to were constantly working to stay up-to-date on the technology. Many manufacturers are willing to come train people on the technology as they hope to sell it, but most of the TVIs learn about the technology independently or from colleagues:

> *We are a department that's very passionate. People are constantly going to conferences, constantly reading things, constantly educating themselves and so I think that's how a lot of things happen: where we have an issue and then someone reads a blog and then suddenly, we also communicate a lot and so some of them will bring like a technology solution on to the team and then five of us will have students that can benefit from that." -P5*

Gaining this knowledge can be challenging for the TVIs as they have many things they need to stay up to date on beyond just the technology. P2 noted that one challenge is:

> *Time to do everything because I'm also staying up on my standards. I'm doing new programming languages every time people turn around. So, there's that aspect and then there's Special Ed. law you have to stay up on and oh they're going to make us learn how to administer this kind of state test and oh there's so many aspects of staying up-to-date as a teacher.*

While the TVIs are an excellent support for the students who are learning these technologies, they have limited time with the students. Because of this, P5 said that he also tries to work with the students' other teachers:

> *I also try to communicate with teachers ... like this is a student's needs and that kind of helps sometimes getting them more support on-site, because I just go and see her 30 minutes a week…Really integrating technology into our students lives can*

> *sometimes be really challenging especially when we don't have someone from our department on-site.*

Many of these students' teachers are not familiar with the technology, so they cannot provide support a student needs without additional training themselves.

This is also true for parents. Many times, the parents are not familiar with what technology is best for the students. The level of support that parents are able to provide can vary. When talking with a TVI who worked at a residential school for the blind (P4), she mentioned that the parent's involvement with the technology is minimal. Most often, the impact she sees from the parents is when they purchase technology for the students: "Unfortunately often times they just buy stuff for their child, and it's like, 'Oh I wouldn't have bought that, they might have bought an Android tablet when it might have been better to have bought an iPad.'"

The TVIs who work with students in mainstream schools have more involvement with the parents than those at the residential schools. P1 noted:

> *I will do trainings for the parents, like come in during the school day, and we'll have your kid sitting here and he'll show you what he knows, and you can ask questions and I'll help answer them. But again, for some of them it's a really steep learning curve. I do have expectations that they will support their child, but my expectations are not always met.*

P5 said that while she tries to keep the families informed with what technology the student is learning and will often send technology home, she doesn't push the parents to work on specific skills with their children unless they specifically ask for that.

3.3.3.5 Lack of Motivation

P2 and P4 both mentioned that it could be hard to motivate students to learn some of the technical skills such as typing either because the students did not see how the skill would be useful for them or because there were not accessible and fun programs to teach it. However, the teachers explained that if the students were properly motivated they picked up the technology much more quickly, and a lot of this motivation comes from interacting with the same pieces of popular culture that their sighted peers are interacting with (e.g. Facebook or YouTube). P2 said,

> *So, we're just teaching some kids iPads. They're all obsessed with YouTube. Fabulous, go find a YouTube video that you think I want to watch and learn how to share it and email it to me. I had guys jumping into blue Jell-O, I had kittens, I had puppies, I had religious music, I had … like yeah okay, this is a middle school class. This is what I was teaching in a middle school class. You know, and when we have those moments it's so cool because so much of their lives isn't normal.*

3.4 DISCUSSION

The teenage and childhood years are a formative time, when individuals are learning who they are. It is important to design technology that both allows students to have access to educational information, but also does not make them feel uncomfortable or bring emotional distress when they use it. Children with visual impairments often have to learn an extra layer of technology in order to interact with digital material. It is important to make this extra layer as easy to use and easy to learn as possible, so they can focus on learning educational content. Based on this, I discuss some of the limitations of current assistive technology and opportunities for design below.

3.4.1 *Design for Mainstream Technology*

In agreement with previous work [114], I found that technology reinforces disability identity. This could be positive, as some students relished having skills and using technology that their sighted peers could not use. It could also be negative, as many students felt using different technology made them stand out or made them confront their own disability. Interestingly, I found that teachers would frame their introduction to the technology to combat some of the negative associations. I do not believe this has been reported on in previous work. In general, mainstream technology such as iPads seemed to have less strong identity associations and perhaps less stigma. I believe that designing mainstream technology so that it can support individuals with a wide range of vision and making accessibility features on technology unobtrusive is especially important for children and teenagers, who might not yet be comfortable with their disability and are still forming their identity.

3.4.2 *Increase Support for Independent Exploration*

I found that many of the technologies and the accessibility features were not very easy to learn and the students required a lot of support from parents and teachers in order to successfully learn to use them. When the students did learn to use technologies such as the iPad on their own, they often learned inefficient ways of doing things. While Szpiro *et al*. found this was important for adults with visual impairments, [125] I think there are design opportunities around making the features more discoverable and easier to learn for children. Teachers also noted that it is hard to motivate students to learn some skills (e.g. typing) that they need in order to fully access the technology, in part because there are few fun, accessible games to teach these skills. This indicates a need for technology that is learnable, as well as a need for fun and motivating ways to teach it.

3.4.3 *Design Technology that Supports Students with Progressive Vision Loss*

Teachers were enthusiastic about technology such as the Fusion software, which combines the JAWS screen reader with the ZoomText magnification software and can be used by individuals with low-vision as well as those who are blind. The teachers noted that this software was especially useful when working with students whose vision was worsening over time. Oftentimes, the students had a great deal of emotional stress associated with the vision loss. Having the students learn software that would work for them as their vision change is easier for them both emotionally (as they do not have to deal with the emotional ramifications of switching to a tool for a "blind" person) and cognitively (as they do not have to learn two separate pieces of technology). I believe that designing technologies to support individuals, and especially children, with progressive vision loss is an under-researched area that deserves more consideration. As far as I know, this has not been reported on in the HCI literature.

3.5 LIMITATIONS

All of the teachers that I interviewed were based in one state and their experiences may not broadly represent teachers' experiences elsewhere. My sample size for the interviews was small and because we were referred to and recruited teachers with special knowledge of assistive technology, their students likely have a higher dependence on technology. I learned about the technology experiences of these children indirectly through their teachers; an interesting next step would be to conduct interviews with the children themselves and observe how they are currently using technology. Additionally, as I specifically asked about barriers and touchscreen mobile devices, my results have a high emphasis on those topics.

3.6 SUMMARY

I conducted interviews with six TVIs to determine how children with visual impairments are using technology in educational settings and how teachers are training them on these technologies. I found that teachers are training their students on a wide variety of both mainstream (e.g. iPads and computers) and assistive technologies (e.g. low-vision magnifiers and screen readers). The teachers keep a number of factors in mind when choosing technology for their students. They (1) try to match the technology of the rest of the students in the school, (2) choose technology that fits the individual student (based on visual ability, academic ability, vision and personal preference), and (3) choose technology that fits the tasks and environments the students need it for. The teachers report that many of their students had less exposure to technology than their sighted peers and often had challenges learning some of the skills (like typing) needed to access these technologies.

All of the teachers used touchscreen mobile devices (mainly iPads) with certain students. Different teachers noted that they used them specifically with low-vision students, younger students or students with additional impairments. The students use the iPads in many different ways: in place of stand-alone video magnifiers, for information about the surrounding environment for orientation and mobility, or simply as computers.

I found that students face many challenges in trying to use the technologies. They have to learn an extra layer in order to access the same material as their sighted peers and often the technologies have usability problems and bugs. Based on these challenges, I discussed the limitations of current technologies and describe three areas of design opportunities: (1) designing for mainstream technologies, (2) increasing support for independent exploration of these technologies and (3) designing technology that supports students with progressive vision loss.

Chapter 4. BRAILLEPLAY GAMES

As discussed in the related work and previous chapter, mobile devices became increasingly popular among people with visual impairments with the release of screen readers for both Android and Apple mobile devices in 2009. However, because of the flat screen of these devices, it is impossible to read or write Braille characters without an external Braille display. Additionally, in 2014, at the time that the research in this chapter was done, I could find no Braille games for the smartphone that were accessible for children with visual impairments. Because of this, in this chapter, I seek to answer *Q4: How can we use a touchscreen device to create engaging games that children with visual impairments can use to practice reading and writing Braille characters?*

The 2014 ASSETS paper [90] on this work has been cited by a number of other papers on creating accessible educational games [63,70,109,128,140], including a set of Braille-based games using a similar interface called "GBraille" [11] and work on collaborative programming environments, which took heed of the design recommendation to design for both children with and without visual impairments. The work in this chapter was done in collaboration with Cynthia L. Bennett, Shiri Azenkot, Richard E. Ladner and is based on work previously published at ASSETS 2013 [88] and 2014 [90].

4.1 INTRODUCTION

In this chapter, I present *BraillePlay*, a suite of educational smartphone games for blind children. In creating these games, my goal was to design games that (1) were accessible and engaging to blind children and (2) promoted Braille concepts. I focused on Braille because in recent years there has been a sharp decline in Braille literacy. According to the National Federation of the Blind, only 10 percent of blind children in the US are learning Braille despite the fact that Braille literacy

has been strongly linked with a higher education level, a better chance of employment, and a higher income for blind adults [93]. I designed the games for children who are 5 years old or older, so they can learn Braille concepts at the same age that their sighted peers are learning to read.

Figure 4.1. The VBraille interface for "reading" and "writing" Braille characters (left) and a menu from the VBHangman game that is based on the word game Hangman (right).

The BraillePlay suite includes four applications that use the VBraille interface [54,88] through which the user can identify (*i.e.*, "read") and enter (*i.e.*, "write") Braille characters (see Figure 4.1). The BraillePlay games are intended to help children learn and memorize Braille character encodings, which is a critical aspect of learning how to read Braille. Two of the games simulate Braille flashcards, testing children's ability to identify and enter characters while the other two incorporate the VBraille interface into word games. The games have speech output with high-contrast but minimal graphics, large print, and gestures that follow accessible design guidelines [57,60].

To evaluate the games, I conducted a longitudinal study in the wild with eight blind children (ages 5 – 8). I asked participants to play instrumented versions of the games at least four times a

week for four weeks. Throughout the study, I surveyed and interviewed participants and their parents. I found that most participants were able to play the games independently with minimal training. Several of the children found the games enjoyable and were motivated to play them for long periods of time, and several of the parents felt that the games helped their children learn Braille.

In summary, my contributions include:

- The BraillePlay games, a set of iOS and Android applications, the code for which is available in an online repository[2],
- Empirical findings from a longitudinal study that show how children interacted with the games,
- Design implications for accessible smartphone games for children, and
- A discussion of lessons learned that provide guidance for researchers interested in conducting studies with children in the wild.

The work in this chapter supports the third claim of my thesis: *to improve access to learning technologies on these devices, we can design applications that (1) use the spatial information afforded by the touchscreen to help children understand structural information such as the layout of a Braille cell or the structure of program code, and (2) use the visual, haptic and audio feedback of these devices to engage children and allow them to use these applications independently*.

4.2 Related Work

To the best of my knowledge, BraillePlay games are the first educational smartphone games designed specifically for blind children. In addition to the work on educational touchscreen

[2]https://code.google.com/p/mobileaccessibility/

applications described in Chapter 2, my work draws from prior work on representing Braille on smartphones and educational games for sighted children.

4.2.1 *Representing Braille on Smartphones*

Since BraillePlay teaches Braille concepts, I discuss several approaches to representing Braille on smartphones. People commonly read Braille on specialized devices called Braille displays [5] that display a row of characters at once. However, Braille displays cost hundreds or thousands of dollars.

BraillePlay games use Jayant *et al.*'s VBraille [54], a method of reading Braille characters on a smartphone. VBraille displays one character on the screen at a time. The screen is divided into three rows and two columns like a Braille cell and the phone vibrates when the user touches regions that correspond to the raised dots in the current character. Jayant *et al.* argue that VBraille can be a useful output method for deaf-blind mobile device users, since VBraille only provides haptic feedback. When using VBraille in the games, I added speech output that tells the user which dot she is touching. I also created a VBraille interface for entering characters, where the user double taps regions of the screen to raise the corresponding dots. Al-Qudah *et al.* [4] also present a method where characters are output through haptic feedback only. They use a vibration pattern for each character that is based on its Braille representation.

There has also been much work on Braille-based methods for input on a smartphone. BrailleTouch [39], BrailleSketch [73], Perkinput [17], and TypeInBraille [80] use multi-touch input to enter characters, and these touchscreen Braille input methods have even been incorporated into the BrailleNote Touch Notetaker [149]. BrailleSketch [73], which allows a user to input a Braille character by sketching a Braille character using a gesture that goes through all the dots included in the character. These methods are likely to be difficult for young children who are not

as coordinated or dexterous as adults. BrailleType [97] is similar to the VBraille input method I use in BraillePlay. Unlike VBraille, BrailleType users single tap the regions of the screen to raise dots. BrailleType is likely to be faster than VBraille but also more error-prone, especially for children who are learning Braille. Since this work has been published, a set of mobile phone games using an interface called GBraille, which also uses the spatial layout of the touchscreen to represent Braille characters, was also created and evaluated in a study with children with visual impairments [11].

4.2.2 *Educational Games*

There is a body of literature enumerating the potential benefits of using digital games for education [31,36,79]. Games can be valuable learning tools because they are engaging and motivate users to improve their gameplay skills. However, it is difficult to isolate the learning effects of games, and there is little direct evidence in the literature that games teach certain concepts [36], and results have been mixed in studies measuring the empirical evidence of learning using educational games [50,74].

There has been a great deal of research on the use of smartphones for education by sighted children. In a study by PBS and Joan Ganz Cooney Center, they found that preschool and kindergarten children in the United States both have access to and are adept at using smartphones [167]. After evaluating an educational application for teaching literacy skills, they recommend designing to match developing motor skills and avoiding timing-reliant gestures such as tap and hold. In a gesture elicitation study, Rust *et al*. [111] found that children generated similar "standard" gestures to adults, but that they used fewer symbolic (e.g. drawing the letter "A") gestures. In designing the BraillePlay games, I attempted to adhere to these guidelines and keep the gestures as simple as possible.

There are few educational games designed for people with visual impairments. Song *et al.* [120] developed two audio-based learning games on TeacherMate, an inexpensive mobile device designed for people in developing countries. They found that children enjoyed playing their games individually and in groups, working together on challenging parts of the game. Sánchez *et al.* [113] also developed an accessible game on a custom hardware platform called MOVA3D, which teaches children orientation and mobility skills. Although there have been a few studies on mobile games for children with visual impairments [11,64], I found no guidelines about gestures preferred by children with visual impairments.

4.3 BRAILLEPLAY GAMES

4.3.1 *Design Principles*

I designed the BraillePlay games to be (1) educational, (2) accessible, (3) accommodate a variety of skill levels, and (4) available for mainstream devices.

4.3.1.1 Educational

I created games to promote Braille literacy, which is in rapid decline despite the fact that it is important for success in both school and the workplace [93]. Early exposure to Braille concepts is crucial for blind children [98]. Just like their sighted peers, blind children must begin learning reading and writing concepts in preschool, and there are a variety of games and avenues to aid with this. To read Braille, children must both know the Braille dot patterns and be able to discern these dot patterns tactilely. Because it can be difficult and frustrating for children to discern the patterns using the small standard Braille cell, the Braille alphabet is often taught using over-sized stand-ins including plastic eggs in egg cartons and tennis balls in muffin tins [98].

The VBraille interface differs from standard Braille in two ways: it is much larger, and it uses vibration instead of raised bumps to represent the characters. Because of these differences, it cannot help children develop the tactile sensitivity needed to read a standard Braille cell. However, VBraille can help children learn Braille encodings, a critical component of learning how to read. Teachers already use large representations of Braille characters that they make out of egg cartons and muffin tins.

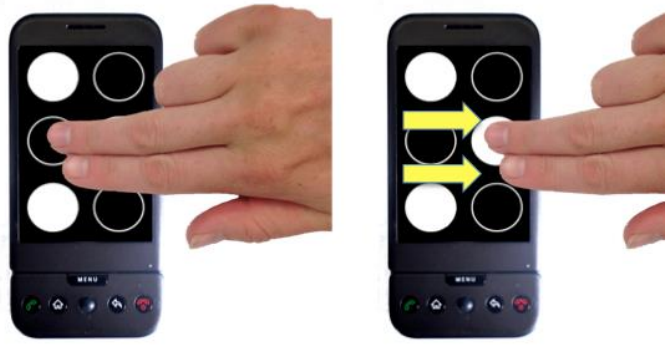


Figure 4.2. A two-finger swipe to the right allows users to submit a character entered in VBWriter or move to the letter input screen in VBReader.

4.3.1.2 Accessible

To be accessible, the games had to be (1) age appropriate for children aged 5 years old and older and (2) accessible for people who are sighted, low-vision, and blind.

To accommodate the developing motor skills of young children, I used simple gestures such as single and double taps and swipes (Figure 4.2). When possible, I gave users the ability to use either the smartphone's keypad or a touchscreen gesture to complete an action. For example, users can either double tap on the touchscreen or use numbers 1-6 on the keypad to raise or lower dots in VBWriter. I chose a large representation of Braille so that young children could easily distinguish between dots in nearby rows or columns. I also used a representation that has the same

spatial layout as a Braille cell, as opposed to a purely temporal representation like the one developed by Al-Qudah *et al.* [4], making it easier for young children to make the connection to Braille characters. Additionally, the BraillePlay games do not rely on timing. Instead they are self-paced, and children can hear information about the state of the game as many times as they want.

To ensure the games were accessible for people with all levels of vision, I used high contrast and large fonts as well as audio and haptic feedback. I also relied on simple gestures and VoiceOver-like interaction techniques with menus (single tap to hear an option and double tap to select it) in line with earlier accessibility work [57,60].

4.3.1.3 Accommodate Different Skill Levels

I wanted to design games that worked for children 5 years old and older and accommodated varying levels of Braille, vocabulary, and spelling skills. I thus created simple flashcard games (VBReader and VBWriter) for people just learning Grade 1 Braille characters, and more complex word games (VBHangman and VBGhost), which require a larger vocabulary and spelling ability. I hoped that children would begin using VBReader and VBWriter and be motivated to "graduate" to the more complex word games.

4.3.1.4 Available on Mainstream Devices

Finally, I wanted to design games that could be played using a mainstream device instead of specialized assistive technology. As smartphones become increasingly pervasive in our culture, the ability to use a smartphone is, in itself, a crucial skill. It is therefore important that blind children gain exposure and experience using smartphones just like their sighted peers.

4.3.2 *Description of Games*

I designed four BraillePlay games: VBReader, VBWriter, VBHangman and VBGhost. In the year (August 2013-August 2014) that the Android and iOS versions of the games were available in the Google Play store and the Apple App Store, there were 1946 downloads of the Android games and 4,471 downloads of the iOS games. There are minor differences between the two platforms, and in this paper, I describe the Android versions, since those were the games used in the longitudinal study. See Appendix D for more details on the games and gameplay.

4.3.2.1 VBReader

VBReader is a simple flashcard game in which a user identifies VBraille characters. A VBraille character is presented on the screen. When the user determines which character is presented, she uses a two-finger swipe to the right (Figure 4.2) or presses the trackball to load the character entry screen. Using the menu button, the user can choose between three options to enter a character: (1) a Qwerty keyboard, which presents a Qwerty keyboard in landscape orientation, which can be navigated with a VoiceOver-like gestures, (2) a two-column alphabetical keyboard, navigated in the same way, or (3) a tapping progression through the alphabet, in which a user can cycle through the alphabet with a single tap to move to the next letter and a double tap to select a letter. After the user selects a letter, the application tells the user whether the input was correct or not and presents a new VBraille letter. If the user exits the application, VBReader tells the user how many characters were identified correctly out of the total number: “You entered 5 out of 10 characters correctly!”

4.3.2.2 VBWriter

VBWriter is another simple flashcard game in which a single character is spoken aloud (“Enter A as in Alpha”), and the user must input that character using the VBraille interface. The user can

press the menu button to hear the letter again. After raising the desired dots in the VBraille screen, the user can submit the character with a two-finger swipe to the right or a tap on the trackball. If the character was entered correctly, the application congratulates the user and presents anther character. If the letter was not entered correctly, the application tells the user what letter was entered and displays the correct letter on the VBraille interface. When the user exits the game, VBWriter announces the number of correctly entered characters out of the total number attempted.

4.3.2.3 VBHangman

VBHangman is based on the word game Hangman. In Hangman, the user must determine what a word is, given the length of the word and a limited number of guesses as to which letters are in the word. With each guess, the user is told whether the letter is in the word, and, if it is, where the letter is in the word. For example, if the word was "banana" and a user guessed the letter "A," she would be told "blank-A-blank-A-blank-A." Hangman is typically played between two people.

In VBHangman, the user plays against the "computer." At the start of a game, the user chooses the length of the word she would like to play using a menu. The smartphone randomly selects a word of that length, then displays the main menu (Figure 4.1). In the main menu, the user can choose to hear (1) the word so far ("blank-A-blank-A-blank-A"), (2) the number of trials left ("You have four trials left"), (3) the letters guessed so far ("Letters guessed: 'A' as in alpha, 'C' as in Charlie"), or (4) the instructions for the game. The menu also includes an option to enter another letter. The user enters letters through the VBraille interface and uses a two-finger swipe to the right to submit her guess. She is given a chance to verify her guess: "You entered the letter 'B' as in bravo, swipe right with two fingers if that is correct." After each guess, the game will tell her if the letter is in the word: "Good guess, 'B' is in the word. The word so far is 'B-A-blank-A-blank-A.'" If the user successfully completes the word, the game will congratulate her and read the

completed word aloud. If the user uses up all her guesses and does not correctly identify the word, the game informs her that she has lost and tells her what the word was.

4.3.2.4 VBGhost

VBGhost [88] is a slightly more complicated game, based on the word game Ghost, in which users take turns adding letters onto a word fragment. If a user adds a letter that spells out a complete word then she loses (*e.g.*, if the current word fragment is "gam," the user will lose if she adds the letter "e" because she will spell "game"). A user also loses if she adds a letter to the word fragment that makes the word fragment invalid (*e.g.*, if she adds the letter "z" to the word fragment "gam," "gamz" is no longer the start to any word).

In VBGhost, the user indicates whether she wants to play the game against the smartphone or against a co-located friend. If she plays against the smartphone, the user is taken to the VBraille screen where she can enter the first letter of the game. After entering a letter, the user swipes with two fingers to the right, and the game tells her (1) if she entered a non-alphabetic character (she is presented with a blank VBraille screen again), (2) if the letter creates an invalid word fragment or completes a word: "I am sorry, you lost, "gamz" is an invalid word fragment" (she is then presented with the main menu screen), or (3) if the letter is a valid play. If the letter is a valid play (part of a valid word fragment), the smartphone selects a letter to add to the word fragment (the smartphone will sometimes complete a word and therefore lose at this point). If the smartphone loses the game by completing a word, it informs the user of her victory and moves to the main menu screen. Otherwise, the phone adds a letter to the word fragment.

In the multi-player mode, users pass the smartphone back and forth to enter letters. After one user adds a letter to the word fragment, the other user can choose to (1) enter her own letter, (2) listen to the word fragment so far or (3) challenge the previous letter entered by her opponent as

either completing a word or creating an invalid word fragment. The smartphone checks whether the challenge is correct and announces which user won based on the outcome of the challenge. I chose to include a more complex, multi-player game in the suite of BraillePlay games to facilitate collaborative learning.

4.4 EVALUATION

To evaluate BraillePlay, I conducted a longitudinal study with eight blind children and their parents over a four-week period. Participants were recruited from across the United States and Panama. I chose to perform a longitudinal study in the wild because I wanted to observe gameplay patterns over time that reflected natural use. I conducted the study remotely because there are few blind children locally that were willing to participate in the study.

I sought to answer three research questions:

- Are BraillePlay games *accessible* to blind children?
- Are BraillePlay games *engaging* for blind children?
- Are BraillePlay games *effective* teaching tools?

Table 4.3. Participants in BraillePlay Longitudinal Study

ID	Age	Gender	Degree of Vision	Braille Knowledge	Note
P1	6	F	Light perception	Grade 1	-
P2	6	F	Low-vision	Some contractions	-
P3	5	F	No functional vision	Some letters	-
P4	6	M	No functional vision	Some letters	Language delays
P5	8	M	Light perception	Some contractions	-
P6	7	F	Light perception	Some letters	English as a second language
P7	7	F	No functional vision	Some letters	Cerebral palsy
P8	6	M	Low-vision	Some letters	Language and motor delays

4.4.1 *Participants*

I recruited eight sets of participants (children and parents) for this study from around the United States and Panama (Table 4.3) through email lists related to raising and caring for blind children. I required that the children (1) be between the ages of 5 and 13 and (2) they identify as either blind or low-vision. The mean age of the child participants was 6 years old (age range was 5 to 8). There was a wide range in the children's knowledge of Braille, as some children were learning how to read single letters, while others were learning Grade 2 contractions. Participants were compensated with a $50 Amazon gift card.

4.4.2 *Apparatus*

I used five Android G1 phones with three games: VBReader, VBWriter and VBHangman (Figure 4.1). I did not include VBGhost because of the age range and literacy level of the participants: most participants were learning individual letters and had not progressed yet to spelling words. I included VBHangman for the participants with higher literacy skills, but the mechanics of VBGhost seemed too complex for the participants. I installed two versions of each game: one for the parent and one for the child, because I wanted to have one instrumented version that the children played exclusively ("child version") and one version that the parents could use to become familiar with the games ("parent version"). The parent and child versions of the games were identical, except that the child versions were instrumented, and the parent versions were labeled as such (*e.g.*, "Parent VBReader" vs. "VBReader"). Logged information from the child version was sent to the researchers' server when the phone was connected to the Internet and saved to a log file on the phone otherwise.

4.4.3 *Procedure*

Throughout the study, I conducted a preliminary survey, weekly semi-structured interviews, and a final exit interview and survey with each set of participants (Appendices C-H). The preliminary survey included questions about demographic information about the child along with information about the child's vision, siblings, and Braille knowledge. Each family was then sent an Android phone as described above. I instructed the families to take a few days to become familiar with the applications and the phones, using the parent version of the applications. After the orientation period, I conducted 30-minute training sessions over a video-conferencing system with each family to answer questions. The participants were then instructed to play the games for 30-minute sessions four times a week for four consecutive weeks. I encouraged families to maintain a weekly average of two hours of gameplay, but they were flexible in how they did this due to differing attention spans and schedules. Because of the varying developmental stages and experience with Braille, I did not define how much I wanted the participants to play each game, and, instead, hoped each participant would play the games best suited to their ability and preference. At the end of the study, I conducted a final set of 30-minute semi-structured interviews with the parents and children.

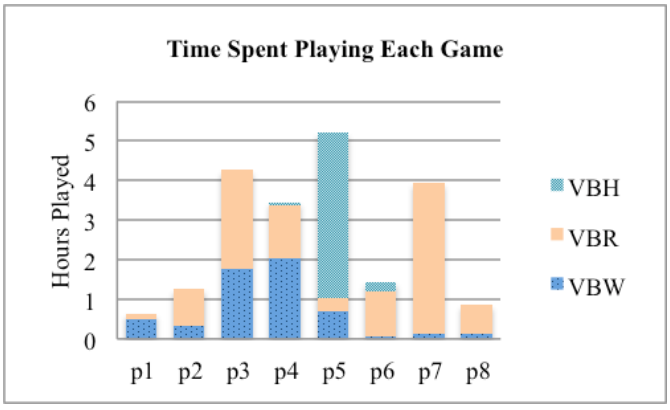


Figure 4.3. Time children spent playing VBHangman (VBH), VBReader (VBR) and VBWriter (VBW).

4.4.4 *Design and Analysis*

I transcribed and coded the interviews and preliminary surveys. For the quantitative data, I explored two measures: accuracy and time to enter each letter. For both VBReader and VBWriter, accuracy for each letter was binary: either the user entered the letter correctly or not. To measure the length of time to enter each letter, I measured the period of time between the time the letter was presented to the user (either loaded onto the VBraille interface for VBReader or spoken aloud in VBWriter) until the time the user double-swiped to submit a character in VBWriter or selected an answer from the menu in VBReader. For analysis, I excluded extreme outliers.

Our analysis did not include entry times and accuracy measures from VBHangman. I could not compute accuracy rates because there was no way to determine what letter the participant *intended* to enter in VBHangman. I had no reliable measure of entry time because the time spent in the character entry screen included the time spent *thinking* about what letter to enter.

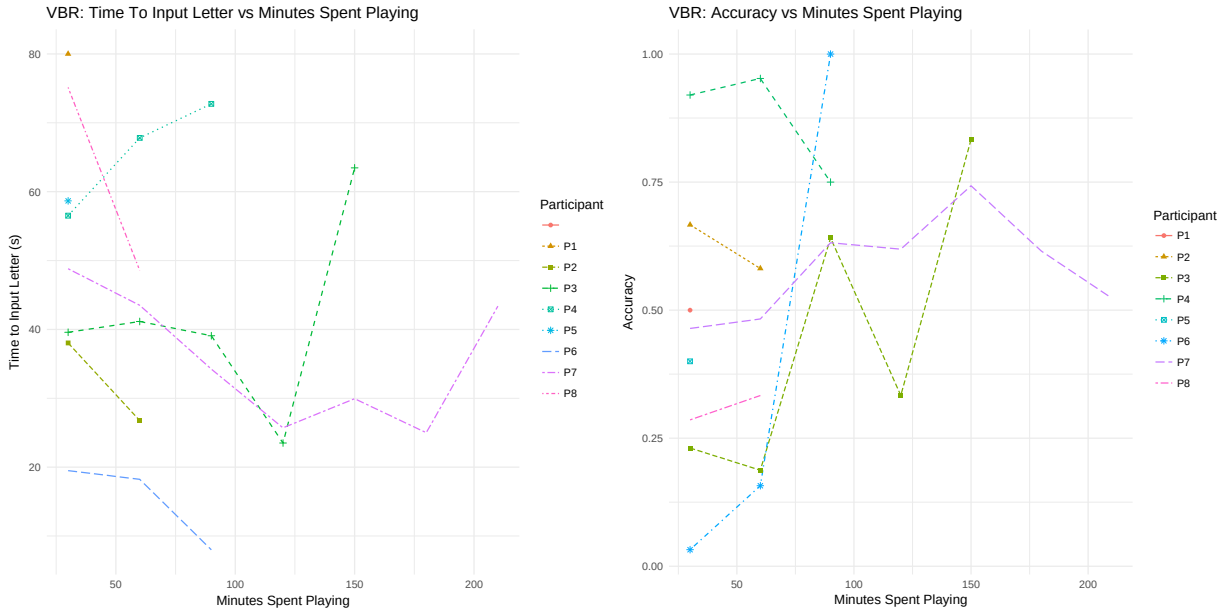


Figure 4.4. Gameplay patterns for VBReader (VBR). The left plot shows the average time to enter a letter for and the right plot shows accuracy rates for participants over time. Accuracy rates and letter entry are computed for every 30 minutes of gameplay. There is high variability and no trends.

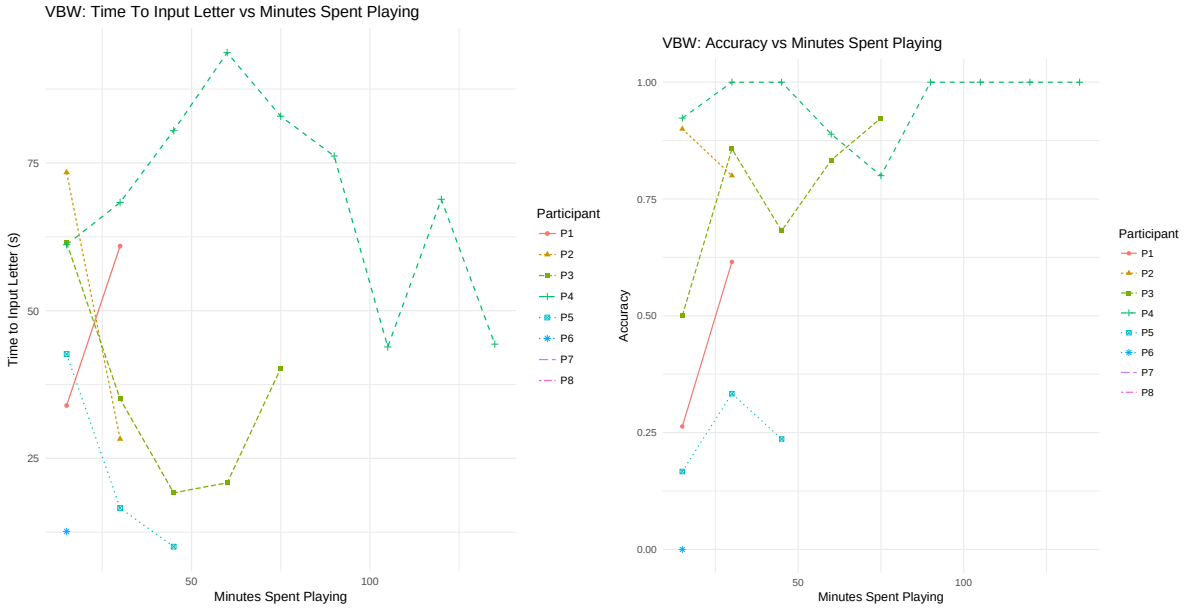


Figure 4.5. Gameplay patterns for VBWriter (VBW). The left plot shows the average time to enter a letter for and the right plot shows accuracy rates for participants over time. Accuracy rates and letter entry are computed for every 15 minutes of gameplay. There is high variability and no trends.

4.5 FINDINGS

4.5.1 *Gameplay Patterns*

The children collectively played the games for 21 hours (Figure 4.3), in which they read over 360 letters in VBReader and entered over 598 letters in VBWriter. The total time a child spent playing the games ranged from 37 minutes to 5 hours and 12 minutes (Figure 4.3).

Entry time and accuracy varied widely for the children (Figure 4.4, Figure 4.5). I could not find any visible trends in the data. I believe the variability is likely due to the differing ages, skill levels, abilities, and knowledge of Braille of the children, as well as differences in the amount of parental supervision children received while playing the games.

P4 (age 6) achieved the highest accuracy rates, entering characters correctly 96% of the time with VBReader and 90% with VBWriter. P4 did not play VBHangman. Interestingly, P4 also had the longest average entry rates on both VBReader and VBWriter, with an average of 63.3 seconds to identify a character on VBReader and 67.6 seconds to enter a character on VBWriter. P6 (age 7) had the lowest accuracy rates, with an average accuracy of 11% on VBReader and 8% on VBWriter. She took on average of 18.6 seconds to identify letters on VBReader and 12.6 seconds to enter letters on VBWriter. Her parent explained that she had difficulty understanding the screen reader since she was learning English as a second language. P5 (age 8) was the only child who played VBHangman consistently. He was the oldest child in the study and had relatively advanced Braille skills.

4.5.2 *Interviews and Observations*

With the exception of P8 who had difficulty playing the games due to motor impairments, parents and children were enthusiastic about the games in the initial and exit interviews. In the preliminary

surveys, I found a strong desire for and interest in accessible smartphone games. Five out of eight children either had their own smart device (either an iPod Touch or an iPad) or used their parent's smartphone to play games, listen to music, or listen to audiobooks. Three of those five parents mentioned that they had not found any accessible Braille applications. Four parents said that they had difficulty motivating their child to learn Braille concepts, signaling a need for fun Braille-based games.

From the exit interviews with the children, I found that most of the children were able to interact autonomously and enjoyed playing with the BraillePlay games. Seven children were able to play the games by themselves, while P8 needed assistance because of motor impairments. P1 – P7 reported that they had no problems holding the phone, using buttons or exploring the screen with one finger. However, three participants reported having some difficulty with the two-finger swipe, indicating that multi-touch gestures may not be suitable for children. Most importantly, all seven children (except for P8) reported liking the games. Additionally, these seven children reported that they understood the representation of the Braille cell using the VBraille interface.

Interestingly, I found slightly different results from the interviews with the parents: only six parents felt that their child was able to consistently play the games without help after the first few sessions of the study. P1's parent reported that her child had "problems getting the double tap to work" and the "reading with one finger on the screen was too confusing," although P1 reported that she was able to play the games autonomously. When asked what they thought could improve the games or make them more engaging, the parents had multiple suggestions. Four of the parents (parents of P4, P5, P6, P8) suggested improving the voice, as the text-to-speech voice on Android was in the words of P8's parent "awful" and "electronic sounding." Two parents (P1 and P3) suggested making more advanced versions of the games that included Braille contractions and two

parents (P3 and P6) suggested adding more visual interest (e.g. pretty colors or cute animals) for children with low vision. Finally, two parents (P6 and P7) recommended adding more interesting audio such as a "kid's voice talking" or an "applause of hooray type thing with every correct answer and an 'oh try again' or something like it for each incorrect answer."

I also found evidence that some children were able to learn Braille concepts with the Braille games. Three children thought that the games helped them learn Braille letters. Additionally, three parents reported that the games helped their child improve his or her ability to read and write Braille letters. P5's parent elaborated:

> *It helped him to remember that there are six dots in a cell as opposed to a whole shape and to remember which dots made up the letters. I think he loved having something on the phone that he could do in Braille.*

Three other parents reported that they were not sure if the games helped their children in their current stage in development but thought it would help their child at some point. The parent of P3 said,

> *I'm not sure if it helped her specifically… it was an exciting app for me as a parent…I think at some point in her development, it could help a lot.*

The parents of the two participants (P1 and P8) who did not think the BraillePlay games were or would be helpful stated that they had difficulty getting their children to play the games. The parent of P1 said, "No, she didn't play them enough." Figure 4.3 shows that P1 spent little time playing the games. Interestingly, I found that the BraillePlay games helped some of the parents learn Braille. Of the six parents who played the games with their child, four said that they learned Braille characters from the games (the other two already knew all of the characters).

4.6 DISCUSSION

4.6.1 *Research Questions*

Using both the gameplay patterns and information from the interviews, I was able to answer the three research questions:

4.6.1.1 Are BraillePlay games *accessible* to blind children?

For the most part, I found that the children were able to play the games autonomously. P8 had difficulty navigating the games due to a motor impairment. P1 had trouble understanding the vibration output of the phone and performing some of the more complex gestures such as the double tap or two-finger swipe. This highlights the challenges of designing for this particular population: screen readers for touchscreen devices tend to rely heavily on two-fingered and timed gestures, but these gestures can be very challenging for children to perform.

In the exit interviews, all the parents thought that their children were able to understand the concept of the vibrating dots representing the raised dots of a Braille cell. This is interesting as the children were able to "localize" the dots even though the entire phone vibrated and were able to understand 2D information through the haptic and audio feedback. In general, the simple design of the games (audio menu with the VBraille interface) was able to be understood and navigated by the children. This, along with Giudice *et al.*'s work on making graphical information accessible [41], is one of the first examples of using the spatial layout of a touchscreen to convey 2D structured information through haptic and audio feedback. One thing that, surprisingly, turned out to be important was including the high contrast visual element, as many of the children had some vision and wanted to use it. I found that only one child regularly played VBHangman; in interviews with the parents, it appeared that this was because the children were not yet fluent enough in Braille to spell out words and were instead focused on learning the Braille character patterns.

4.6.1.2 Are BraillePlay games *engaging* for blind children?

I found that most of the children enjoyed playing the BraillePlay games, but they did not play them for as long as I had hoped. The children each played the games for an average of 2.6 hours over the course of the study; this is significantly lower than the encouraged 8 hours. The parents reported that interest in the games dwindled at the end of the study for many of the children, indicating that I did not design the games to have the optimal balance of challenge and success needed for game flow [79]. This is likely because two of the three games used in the study were the simple flashcard games. The only child that consistently played VBHangman was the oldest child in the study (P5) and also logged the most total hours playing the games (Figure 4.3).

Using the VBraille interface to build more difficult word games or include contractions in the future might lead to more engaging games. Additionally, based on feedback from the parents, the games could be made more engaging by improving the voice of the text-to-speech, as well as making the visual and audio effects more interesting and fun. For this population, it would be interesting to examine how to increase engagement through the audio and haptic feedback, such as adding fun sound effects and allowing the children to change around the voice of the screen reader.

Although I did not include VBGhost in the study, P5 expressed interest in playing more word games and might have enjoyed the more difficult game. My results are similar to findings from prior work. A study with sighted children playing a literacy game over a period of two weeks found that children only played for 10 minutes at a time and interest dwindled after a few days of playing [29]. In future studies, I recommend a shorter playing period and focusing on making the audio effects fun and engaging.

4.6.1.3 Are BraillePlay games *effective* teaching tools?

Evidence from the final interviews with parents and children suggests that some of the children learned from the games, but these findings are only preliminary. Three of the parents thought that the games helped their children learn Braille, three were not sure and two thought they did not help. In general, it is difficult to evaluate learning, and similar studies on educational games often have mixed results when trying to show evidence of learning [36,50,52,74,99].

The BraillePlay games seem best suited for slightly older children than most of the participants (7 - 8 years old). The three sets of parents who thought the games helped their child learn Braille concepts were the parents of P5, P6 and P7, the oldest children in the study. Surprisingly, the games seemed to be effective teaching tools for sighted adults, as all four of the six parents who did not know Braille but who played the games with their children reported that they learned Braille characters from playing the games.

4.6.2 *Design Implications*

Based on my study, I distill implications for the design of educational games for blind children.

- Design for collaborative play: I found that many of the participants engaged in collaborative play with their sighted siblings and parents during the study. Games should be designed for collaborative play to engage children and allow their parents and sighted peers to also learn Braille concepts and identify with blind children.
- Design for blind, low-vision, and sighted children: Parents of children with low-vision in the study suggested using more exciting graphics. Games should be designed for children with all degrees of vision by presenting information in multiple ways: using audio, visual and haptic feedback. This will both encourage sighted siblings and peers to play and make the games more appealing for children with some functional vision.

- Design for developing motor skills: I found that many of the children had difficulty using multi-finger gestures, such as a two-finger swipe (used to submit letters), so applications for games should include single finger gestures.

4.7 LIMITATIONS

This study had many limitations. First, it was a small sample size, based on the small population of academic Braille readers [6]. There was a wide variation in abilities of the children, also reflecting the diverse population of children with visual impairments, many of whom have additional disabilities. The subjects were recruited online, so there might be some bias in that they come from somewhat tech-savvy families. Additionally, measurement of learning outcomes was outside the scope of this study. Because of this and the small sample size, most of the data comes from post-study interviews, in which the participants may have responded positively to please the researcher [33,131].

4.8 SUMMARY

These results show that there is a compelling need and desire for educational games for blind children: both parents and children were excited about the BraillePlay games, and many participants mentioned that they were not aware of any accessible Braille-based games for the smartphone. The longitudinal study indicates that you can create accessible ways to interact with Braille characters on a touchscreen, and that this interface can be used and understood by children. I found mixed evidence on whether the children truly found the games engaging. Although most of the children reported that they enjoyed playing the games, they only played them for an average of 2.5 hours over a four-week period. However, many additional word games can be developed with the VBraille interface, maintaining children's interest as they continue to improve their

Braille skills. This work demonstrates that there is great potential for fun and educational games for blind children on mainstream devices.

Chapter 5. ACCESSIBILITY CHALLENGES IN BLOCK-BASED PROGRAMMING ENVIRONMENTS

Based on my research with children with visual impairments in evaluating the BraillePlay games in Chapter 4, I found that these was a lot of interest in and excitement about accessible applications on touchscreen devices from both parents and children. However, the participants in my studies were frustrated with how inaccessible many of the applications were on these devices. Because of this, I decided to explore what challenges these children face in using applications on touchscreen devices. In this chapter, I seek to answer *Q2: What are the challenges that children with visual impairments face when using touchscreen technologies?* I answer this question by looking at a particular type of learning technology: block-based programming environments, which are used as a way to introduce children to programming. More specifically, I conducted an evaluation of existing block-based programming environments, to answer *Q3: What are the accessibility challenges in existing block-based programming environments for children with visual impairments?* The work in this chapter was done in collaboration with Richard Ladner and is based on work previously published at CHI 2018 [87].

5.1 INTRODUCTION

Recently there has been a push to incorporate computer science education in K-12 classrooms. As part of this effort, block-based programming environments, such as Scratch [78] and Blockly [43] have become very popular [22]. These block-based environments use a puzzle-piece metaphor, where operations, variables and constants are conceptualized as *blocks*, puzzle-pieces that can be dragged and dropped from a *toolbox* (a menu of blocks) into the *workspace* (the editor that holds the actual program code) to create 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 block-based environments [168].

Unfortunately, these environments rely heavily on visual metaphors, which render them not fully accessible for students with visual impairments. As these students are already underrepresented and must overcome a number of challenges to study computer science [85,122], it is important that they have equal access to curriculum in primary schools, at the start of the computer science pipeline.

I evaluated 26 existing block-based environments using a screen reader to determine what are the accessibility challenges in existing block-based programming environments for children with visual impairments. I found that in the majority of the environments it was impossible to even select the elements or move them with a screen reader. To explore accessibility problems in the touchscreen environment beyond simply accessing and moving the elements, I modified the Android version of Google's Blockly so that the elements were accessible and movable by screen reader and evaluated it alongside the other environments.

Because of my interest in touchscreen-based learning technologies, I also tested the block-based environment of the Tickle iOS application (the only accessible mobile environment) with three children with visual impairments. This environment is advertised as being the only block-based application that is accessible with VoiceOver, the built-in screen reader on Apple phones [169]. I tested with one child who used a screen reader and two children with low-vision to determine how usable the environment was for them. I found that one child was able to use his

vision to successfully use the Tickle application, but the other two children (one who was functionally blind, the other who had low-vision) were not able to use the environment.

Through these evaluations, I identified five main accessibility problems in block-based programming environments:

(1) The output of the programming environments is generally visual in nature and not accessible,

(2) Blocks are not accessible (cannot be read by the screen readers and often do not use high contrast),

(3) The drag and drop gesture to move the blocks is not accessible with screen readers,

(4) Program structure is not discernable with screen readers, and

(5) Type information is conveyed only through visual metaphors that could not be accessed or understood without sight.

This provides further support for the second claim of my thesis: *In the context of block-based programming environments, these challenges (that children with visual impairments must overcome) include a reliance on visual metaphors and gesture-based interactions that do not interface well with the screen reader.* The work in this chapter was done in collaboration with Richard E. Ladner and is based on work previously published at CHI 2018 [87].

5.2 Screen Reader Evaluation

5.2.1 *Method*

I reviewed 26 block-based programming environments meant for introducing children to programming in March 2017. Environments were selected from Google searches of "block-based programming environments" and "blocks programming languages", the related literature, as well as online lists of programming environments for children [129,170]. Applications that were text-based or seemed directed toward an older audience were not evaluated (e.g. Android AppInventor [171], Actimator [172] and Stencyl [173]).

I evaluated a mixture of web-based, desktop, mobile, and tangible applications. The web-based environments were:

- Scratch [78],
- Blockly [43],
- Accessible Blockly [44],
- Tynker [174],
- Snap! [132],
- Pencil Code [175],
- Gamefroot [176],
- Waterbear [177],
- Spherly [178] and
- Touch Develop [20,130].

The mobile-based environments were:

- Blockly (Android) [43],
- Scratch Jr (iOS) [38],
- Hopscotch (iOS) [179],
- the Wonder Workshop Blockly App (iOS) [141],
- Daisy the Dinosaur (iOS) [180],
- Tynker (iOS) [174],
- PocketCode (Android) [181] and
- Tickle (iOS) [169].

The desktop-based environments were:

- AgentSheets [183],
- Alice [184],
- Looking Glass [185],
- Kodu [186] and
- Squeaky eToys [187].

The tangible environments were

- Osmo [162],
- Kibo [65,125], and
- Cubetto [105].

The web-based and desktop applications were evaluated with the Firefox browser on a Lenovo Flex 3 Laptop computer with the open-source NVDA (Non-visual Desktop Access) screen reader, version 2017.1. The speech viewer was turned on to visualize the spoken audio. The mobile applications were evaluated on an iPad air version 10.1.1 with VoiceOver and a Samsung Galaxy View tablet with TalkBack.

The guidelines for evaluation were drawn from the WCAG [188], Android [189] and iOS guidelines [10]. However, these guidelines only require a minimum level of accessibility and do not necessarily require an interface to be truly usable with a screen reader. For example, according the WCAG 2.0 Guideline 1.2, time-based media is only required to have captions or descriptions if it is prerecorded, which would make it unnecessary to make the programming output of most these environments accessible [188]. Because of this, I evaluated the environments according to a more functional and blocks-specific set of guidelines, which emerged as I conducted the evaluations:

(1) *Accessing Output*: is the output of the programming perceivable (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).

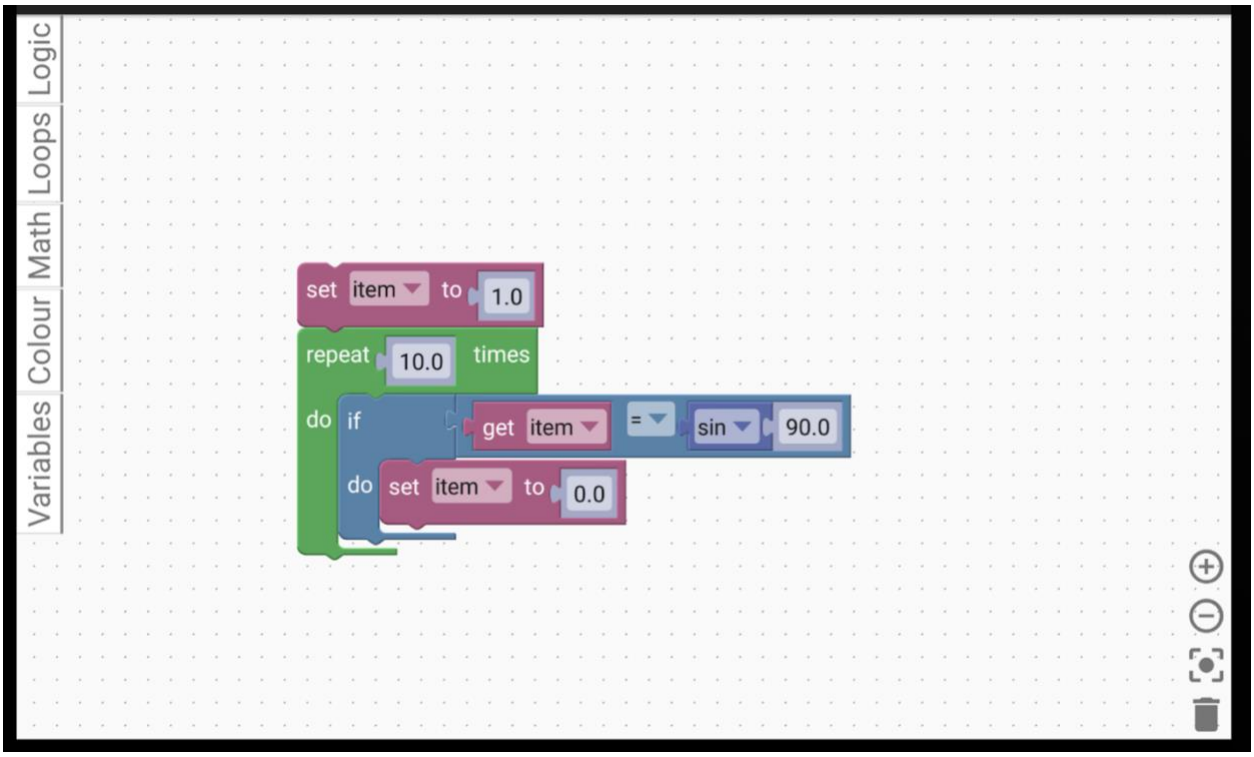


Figure 5.1. The Modified Android Blockly interface. The toolbox can be access by clicking the buttons on the left. This program will set item to zero (and will check if item is equal to sin(90) 10 times).

5.2.2 *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 (Figure 5.1) is open source, I was able to build a Modified Android Blockly that fixed certain 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. In the Findings, I describe the accessibility problems I found with both the original Android Blockly and the Modified Android Blockly.

Table 5.4. Results of Screen Reader Accessibility Evaluation of Block-Based Programming Environments

MOBILE-BASED BLOCKS	Accessible Output	Accessible Elements	Accessible Actions	Convey Program Structure	Convey Type Information	Version
Android-Based Blockly	**Yes** (audio options)	No	No	No	No	Evaluated 2/11/17
Modified Android Blockly	**Yes** (audio options)	**Yes**	**Yes**	No	No	Evaluated 3/17
ScratchJr	**Yes** (audio options)	No	No	No	No	Evaluated 2/1/17
Wonder Workshop Blockly App	**Yes** (robot)	No	No	No	No	Evaluated 2/1/17
Hopscotch	No	No	**Yes**	**Yes**	No	Evaluated 2/11/17
Daisy the Dinosaur (hopscotch)	No	**Yes**	No	No	No	Evaluated 2/11/17
Tickle	**Yes** (audio options)	**Yes**	**Yes** (drag and drop with audio cues)	No	No	Evaluated 2/26/17
Tynker	**Yes** (audio options)	No	No	No	No	Evaluated 2/11/17
PocketCode (Android)	**Yes** (audio options)	**Yes**	No	No	No	0.9.27
WEB-BASED BLOCKS						
Web-Based Blockly	**Yes** (audio options-Blockly games)	No	No	No	No	Evaluated 2/14/17
Scratch	**Yes** (audio options)	No	No	No	No	2
Accessible Blockly	N/A (not part of programming environment)	**Yes**	**Yes**	**Yes**	**Yes**	Evaluated 2/14/17
Tynker	**Yes** (audio options)	No	No	No	No	Evaluated CandyQuest activity
Snap!	No	No	No	No	No	4.0.10
Pencil Code	**Yes** (audio options)	No	No	No	No	Evaluated 2/17/17
Gamefroot	**Yes** (audio options)	No	No	No	No	Evaluated 2/17/17
Waterbear	**Yes** (audio options)	No (can access fields)	No	No	No	Evaluated 2/18/17
Spherly	**Yes** (robot)	No	No	No	No	Evaluated 2/18/17
Touch Develop	**Yes** (audio options)	No	No	No	No	0.0.638.2632 Web App on Windows and iOS

DESKTOP NON-BLOCKS						
AgentSheets	No	No	No	No	No	4
Alice	No	No	No	No	No	3.3.0.0
Looking Glass	No	No	No	No	No	2015.11.5
Kodu	No	No	No	No	No	1.4.164.0
Squeaky eToys	**Yes** (audio options)	No	No	No	No	5
TANGIBLE						
Osmo	No	No	**Yes**	**Yes**	**Yes**	Evaluated 2/1/17
Cubetto	**Yes** (robot)	**Yes**	**Yes**	N/A (does not support complex structure)	**Yes**	Evaluated from online image and description
Kibo	**Yes** (robot)	No	**Yes**	No	No	Evaluated from online image and description

5.2.3 *Findings*

I address the five accessibility challenges encountered in the applications below.

5.2.3.1 Accessing Output

I found that 14 of the 26 environments had some audio output options, but the majority of the games and projects focused on visual output, such as animating avatars. Accessible Blockly is not currently part of development environment, so there was no output to evaluate on. The Tickle, Spherly, Kibo and Cubetto applications could be used to control robots, making them more accessible. This was by far the most accessible feature that I evaluated.

5.2.3.2 Accessing Elements

There were four mobile applications that allowed you to access the blocks themselves (Daisy the Dinosaur, Tickle, PocketCode and Modified Android Blockly), although it was still not possible to construct code using either Daisy the Dinosaur or PocketCode. With the exception of Accessible Blockly, it was impossible to focus on blocks in the toolbox or in the workspace using a screen

reader on any of the desktop or web-based applications. This can be fixed by adding tags that can be read by the screen reader, but it rendered all the other environments completely inaccessible for screen reader users. Surprisingly, of the tangible environments, only Cubetto had blocks that could be understandable to blind children as it has accessible plastic pieces that can be distinguished from each other based on shape. The Kibo blocks and Osmo pieces are not accessible (although Braille labels could be added to make them accessible).

5.2.3.3 Moving Blocks

Most of the digital applications except Accessible Blockly and the modified version of Android Blockly relied on drag and drop to move the blocks from the toolbox to the workspace. Although it is technically feasible for the drag and drop gesture 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. This effectively makes drag and drop inaccessible using screen readers. The Touch Develop application did not use drag and drop. Instead, when you touched a location in the code, a menu would pop up that contained elements that could fit in that location. This has the potential to be an accessible interaction, but unfortunately the menus did not work with the screen readers I tried (VoiceOver and NVDA). Hopscotch also used a menu to construct code, so the action was accessible. Unfortunately, it did not allow you to fully access each of the blocks (e.g. it read some components of the blocks out loud, but not all of them so “set color to purple” read “set color” and you were unable to change the color), so it was not accessible. It was possible to move and construct code for all the tangible environments without vision.

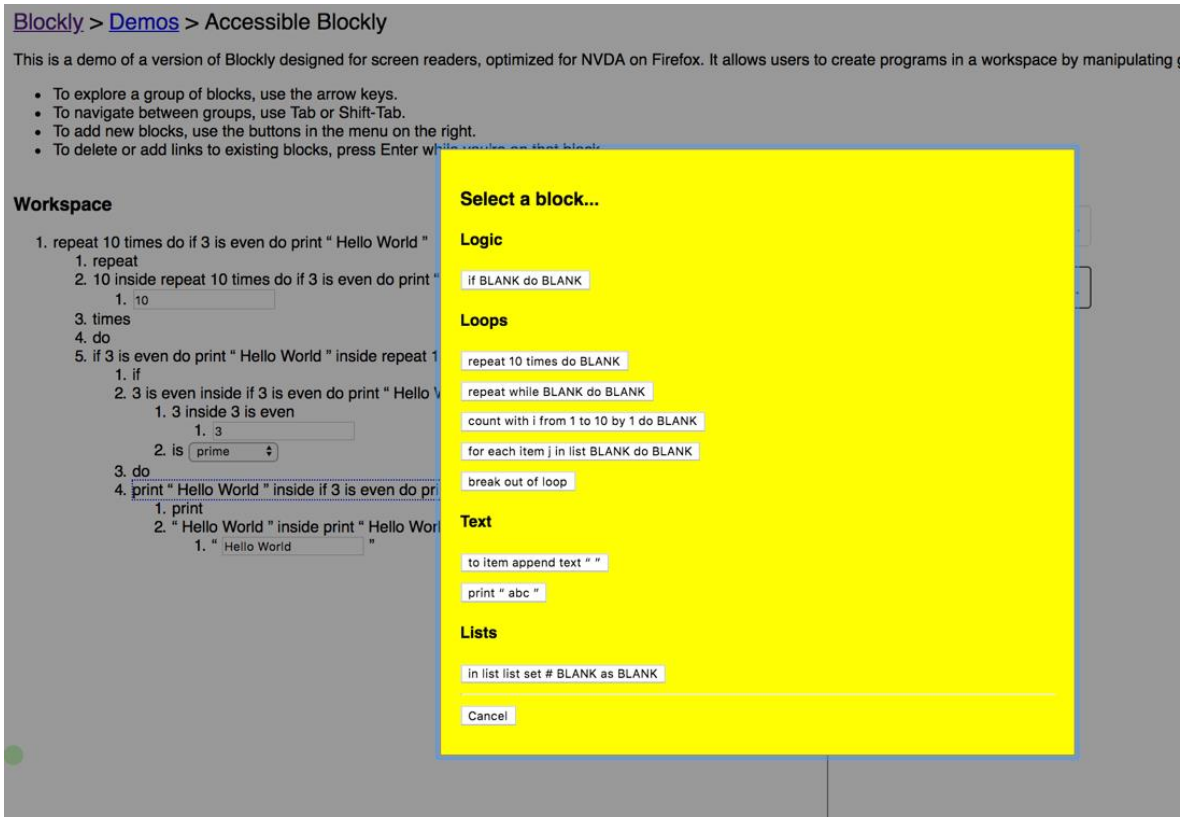


Figure 5.2. The "Create a new block group…" menu which pops up in Accessible Blockly, so you can add a block to the workspace. This is essentially the "toolbox" and does not require drag and drop.

In Accessible Blockly, blocks were not added to the workspace via drag and drop. Instead blocks were added to the workspace by selecting the "Create new block group" button or a "create link" button within an existing block group. Blocks were then selected from a menu (Figure 5.2). This is a form of "select, select, drop," in which you first select a location in the workspace to add a block and then select a block that you would like to add.

In the Modified Android Blockly application, blocks were also moved via select, select, drop: you would select a block in the toolbox and then select a block in the workspace you would like to place it either before or after. This could still be a challenging action to do with a screen reader, as you have to find the location in the workspace where you want to place their blocks. 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 via the rotor.

5.2.3.4 Conveying Program Structure

Only the Hopscotch application, the Osmo tangible blocks and Accessible Blockly were able to convey information about program structure with the screen reader. This was because with the other blocks programs it was impossible to determine whether statements were nested inside or outside of a control statement like a repeat loop or conditional statement without sight. The screen readers gave no clues about program structure. With the Hopscotch application, there were "end" statements at the end of each loop or conditional. The "end" statements did not state which control structure they were closing, however that could be determined by the programmer by navigating up to find the closest open statement. With the Osmo blocks, program structure is conveyed by having blocks physically moved to the right if they are nested inside a repeat forever loop.

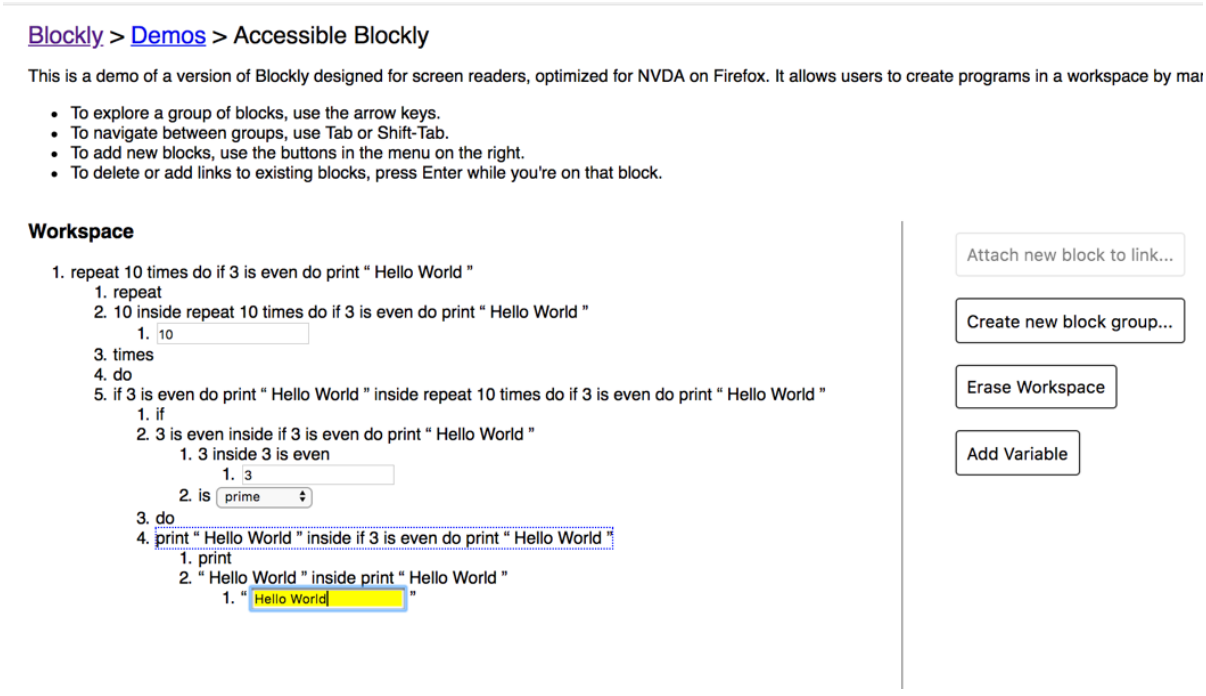


Figure 5.3. A program written in Accessible Blockly. It will print "Hello world" ten times. Note that there is a small bug: "3 is even" should say "3 is prime".

In Accessible Blockly, program structure is conveyed in a hierarchical list, with nested statements contained as sub lists under their containing statement (Figure 5.3). All of the information to determine program structure is communicated in an unambiguous manner. In the Modified Android Blockly application, it is impossible to determine the structure of a program with nested blocks. When blocks are nested inside of each other, as in a 'for' loop or 'if' statement, it is impossible to tell where the nesting ends, and which blocks are outside. Additionally, it would be difficult to locate blocks on the screen and difficult to navigate in a straight line (similar to what Kane *et al.* [58] found) either down between block statements or horizontally to find nested blocks. The fact that the blocks are different sizes could also make it difficult to understand the program structure.

5.2.3.5 Conveying Type Information

In the digital applications, apart from Accessible Blockly and the Tickle Application, type information, if it was conveyed at all, was conveyed visually via the shape and color of the block. This is not accessible via screen reader. 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). Both Osmo and Cubetto conveyed type information through the shapes of their tangible blocks, so they were accessible.

In the Modified Android Blockly application, I did not explicitly include type information in the hints for VoiceOver, but this likely could be added. In Accessible Blockly the type needed at each location is explicated named (e.g. Boolean needed) and when adding a block, only blocks of that type appear in the toolbox.

In the Tickle application, type was mostly conveyed through shapes and color, but with VoiceOver on, the category of the block was announced, (e.g. “Control block, repeat”). However, there was no audio information to convey things like which blocks were Boolean statements that could fit inside conditionals.

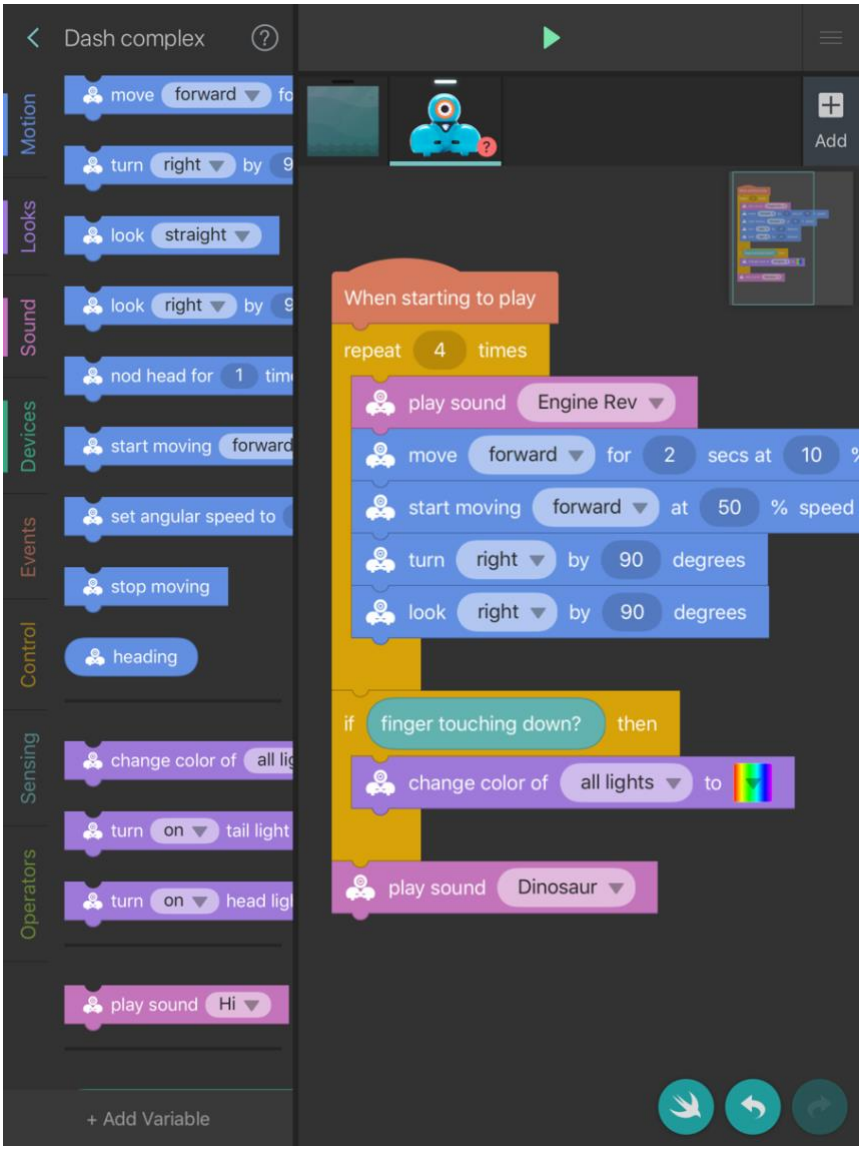


Figure 5.4. A program written with the Tickle Application. Toolbox is on the left, workspace on the right. It will play the “Engine Rev” sound, move forward two times, turn right and then look right four times. Then if you are touching the iPad it will change all the light colors. Finally, it will make a dinosaur sound.

5.3 TICKLE EVALUATION

Tickle was the only unmodified touchscreen application that both allowed you to access and move blocks with a screen reader. It is advertised as being accessible with VoiceOver on its website. I wanted to determine how usable it is for children with visual impairments, and to get an idea of how usable children with low-vision would find a traditional block-based environment. I planned to test it with a number of children with visual impairments. However, I only ended up testing it with three children, as they uncovered a number of usability issues.

5.3.1 *Apparatus*

I evaluated Tickle (Figure 5.4) using an iPad running iOS 10.3.3. This evaluation was conducted in February and March of 2018, a year after the initial evaluations of the other environments. One participant used VoiceOver in the evaluation and the other two did not.

Table 5.5. Participants who Evaluated the Tickle Application

Participant	Age	Gender	Level of Corrected Vision	Previous VoiceOver Experience	Used Screen Reader in Evaluation	Previous Experience with Blocks4All
P1	8	Female	20/150, difficulty focusing	Experience with VoiceOver but prefers to use vision with iPad.	Occasionally	Participated in 4 Blocks4All sessions before evaluating Tickle application (P1 in Table 6.6 and in Table 6.7).
P2	10	Male	20/400	Uses iPad with both VoiceOver and Zoom.	No	Participated in 4 Blocks4All sessions before evaluating Tickle application (P2 in Table 6.6 and in Table 6.7).
P3	10	Male	Has color and light perception, but no real usable vision	Very familiar with VoiceOver, often uses with Braille display on iPad.	Yes	Participated in a Blocks4All evaluation session before evaluating Tickle application (P3 in Table 6.7).

5.3.2 *Participants*

I evaluated it with three children with visual impairments (Table 5.5). Both P1 and P2 chose not to use VoiceOver to evaluate the application, although I turned on VoiceOver to read some of the blocks names to P1. P3 used VoiceOver to explore the environment. All three of the children had previous experience with block-based environments as they had participated in the design and/or evaluation of the Blocks4All environment (described in Chapter 6).

5.3.3 *Method*

The children were asked to complete tasks that included constructing programs from scratch, modifying existing programs and describing existing programs (Appendix I). For all three types of tasks, they used a combination of control structure blocks (e.g. if statements or repeat loops) and operation blocks (e.g. "Play sound 'Engine Rev'"). The sessions were video-recorded and analyzed for usability challenges.

5.3.4 *Findings*

P2 was able to complete the tasks on the Tickle interface using his vision. Neither P1 nor P3 was able to complete any of the tasks due to a number of usability problems. I describe these findings in further detail below.

5.3.4.1 Accessing Elements

In the Tickle application, P2 was able to read the names of the blocks by holding the iPad close to his face without VoiceOver. P1 attempted to do the same, but she was unable to read any of the blocks names, although she could identify the color. For P1, I turned on VoiceOver, so she could hear the name of the blocks. Unfortunately, as both she and P3 discovered, although VoiceOver could access the blocks, the way it read them out was buggy. It would not read the parts of the

block that were contained as drop-down menus (e.g. in the turn “right” by “90” degrees block, the “right” option is part of a drop-down menu and the block reads “Motion block. Turn by degrees. End of block”). It was difficult for P3 to locate the blocks within the workspace without sight, and when asking him to describe an existing program structure, I had to guide his hand to the program.

5.3.4.2 Moving Blocks

P2 was able to use his sight to move the blocks without VoiceOver and was able to place them where he wanted them to go. P1 was also able to move the blocks without VoiceOver. P3 was not able to perform the drag and drop gesture with VoiceOver on. As mentioned in the previous section, Tickle augments the standard drag and drop gesture with an audio description of where the block currently is in the program if VoiceOver is on. The gesture can be accessed with VoiceOver on by doing a double tap and hold. P3 was able to drag blocks using the audio-guided drag and drop, but he was not able to place blocks where he wanted them to go. This was because VoiceOver’s reading of the labels in the workspace lagged behind the movement of his fingers, so he would “drop” the items too late and they would not connect to the existing program in the workspace. He also had trouble finding the existing program within the workspace with VoiceOver.

5.3.4.3 Conveying Program Structure

Even if his hand was guided to an existing program, P3 found it difficult to read blocks by moving his finger down the screen with VoiceOver. In his words, “Maybe it would kind of work for people who can see, but for blind people, they kind of need something big enough to put their fingers on. Like the slightest motion could make your finger move off it.” He was also unable to determine which statements were nested inside of a repeat loop. P1 did not attempt to describe the program structure of existing programs as she was frustrated with the interface before reaching this point.

P2 had a little trouble determining whether blocks were inside or outside of repeat statements but was able to accurately describe the rest of the program structure.

5.3.4.4 Conveying Type Information

As mentioned in Section 5.2.3.5, Tickle conveys type information in both color and shape, and both P1 and P2 were able to distinguish the shapes from each other. With VoiceOver on, you are told what category the block is from (e.g. a repeat block was from the "control category"), but do not have information to indicate where a block might fit. In any case, P3 did not try to determine type information, as he had enough trouble simply accessing the elements.

5.4 LIMITATIONS

There are a number of limitations to my evaluations. First, the desktop and web applications were evaluated with NVDA on the Firefox browser (for the web applications) on a machine running Windows. This was due to cost as access to NVDA is free whereas a subscription to JAWS, the most popular screen reader for Windows machines, is over $1,000 [144]. Accessibility can vary widely across screen reader, browser and operating system combinations, so it is possible that some combination for a particular environment would be more accessible. However, for an environment to be truly accessible, it seems reasonable to ask that should be usable with the end user's preferred combination of screen reader and browser. Additionally, the environments were evaluated by a screen reader novice user, and it is possible that an expert user would have been able to access more features. The second set of evaluations with the Tickle application was done with only three children, as I quickly determined the application was not accessible enough to continue testing.

5.5 SUMMARY

This work demonstrates that there is a need for accessible introductory programming environments that are actually usable by elementary-school aged children. I found that the majority of the environments surveyed were not accessible. Only four of the 26 environments met the bare minimum for accessibility, in that a blind child should both be able to select each block and be able to move the block to build a program. These four environments were (1) the Cubetto tangible blocks application (which was accessible because it used tangible blocks with different shapes), (2) the Modified Android Blockly environment (which was only accessible because I modified it to be so), (3) Accessible Blockly (which was built specifically to work well with screen readers) and (4) the Tickle application (which was built with VoiceOver in mind). However, there are caveats to the accessibility of these applications. While Accessible Blockly is accessible with a screen reader, it appears that there are no longer developers working on it. Both the Tickle application and Modified Android Blockly (environments built for sighted children that were also programmed to work with screen readers) were difficult to use with the screen reader. This is because the blocks were challenging to find in the workspace without sight, the complex blocks were not read correctly by the screen reader, and it was impossible to determine program structure using the screen reader.

Chapter 6. BLOCKS4ALL

In the evaluation of existing introductory programming environments detailed in Chapter 5, I found five main accessibility problems that made these environments difficult, if not impossible for children with visual impairments to use. This is unfortunate, especially as block-based environments, in particular, have been touted as a way to broaden participation in programming [22,77]. They are the environments used in the majority of the hour-of-code activities, which serve as an introduction to programming for millions of students [169], and their inaccessibility sends a message to children with visual impairments that computer science may not be for them. So, in this chapter, I seek to answer research question *Q5: How can we use a touchscreen device to allow children with visual impairments to understand and create block-based programs?*

6.1 INTRODUCTION

In my evaluation of existing programming introductory environments, I identified five main accessibility challenges that made them difficult or impossible to use for children with visual impairments. In order to address these problems, I built Blocks4All, a prototype environment where I implemented various means to overcome these challenges using a touchscreen tablet computer. I 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. I worked with a TVI and five children with visual impairments who used Blocks4All to determine the usability of these techniques and answer the following questions:

- BQ1: What touchscreen interactions can children with visual impairments use to identify blocks and block types?

- BQ2: What touchscreen interactions can children with visual impairments use to build blocks programs?
- BQ3: What touchscreen interactions can children with visual impairments use to understand the spatial structure of blocks program code?

I incorporated the feedback from the five children into a final design, which I then evaluated with five children (three of whom had participated in the formative study). In the evaluation study, the children were asked to independently complete four sets of tasks: two sets that required the children to build and then modify a blocks program and two sets that required the children to state the structure of an existing program with complex program structure. In the evaluation study, I found that children were able to independently use the application, although they needed help to understand how to execute the underlying programming concepts. They were also able to describe the program structure of existing programs and enjoyed using the applications to program a robot. This supports the final claim of my thesis: *To improve access to learning technologies on these devices, we can design applications that (1) use the spatial information afforded by the touchscreen to help children understand structural information such as the layout of a Braille cell or the structure of program code, and (2) use the visual, haptic and audio feedback of these devices to engage children and allow them to use these applications independently*. The work in this chapter was done in collaboration with Richard E. Ladner and is based on work previously published at CHI 2018 [87].

The contributions from this work include (1) design guidelines for designing block-based environments and touchscreen applications for children with visual impairments, drawn from

interviews and formative testing with children and a TVI, and (2) the Blocks4All environment itself, as the source code and application are freely available.[3]

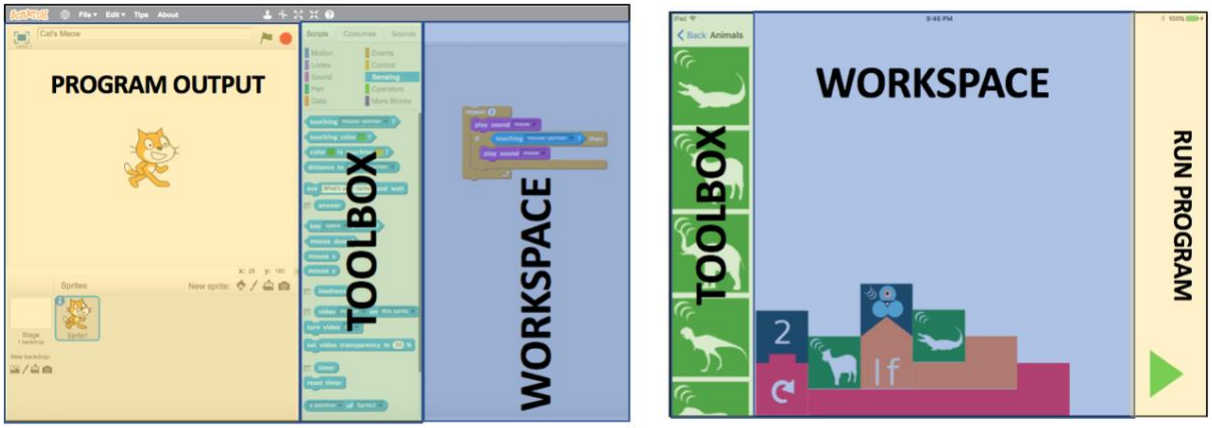


Figure 6.1. Image comparing the three main components (toolbox, workspace and program output) of block-based environments in (a) the Scratch environment [78] and (b) a version of the Blocks4All environment. In Blocks4All, I used a robot as the accessible program output, so only needed a button on the 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.

6.2 Related Work

I discussed much of the related work on block-based programming environments and touchscreen accessibility in Chapter 2. In this section, I discuss in further detail: (1) design guidelines pulled from prior work on creating accessible touchscreen applications that I found particularly important in designing Blocks4All (2) existing approaches to creating accessible block-based environments and how my design differs and (3) approaches to evaluating learning in block-based environments and how that contrasts with what I chose to evaluate in my design.

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

6.2.1 *Accessible Touchscreen Design Guidelines*

In Kane *et al.*'s work on understanding gesture preferences of blind adults on touchscreen devices, the researchers found that it was hard for blind adults to find targets on the screen and had some trouble with time-based gestures [60]. They recommend that designers favor edges and corners, reduce demand for location accuracy by creating bigger targets and limit time-based gesture processing. My design takes these into account with large targets and blocks aligned to the edges of the screen. Additionally, I tried to limit the number of gestures I used in my application.

Giudice *et al.* [41] conducted a study looking at whether blind participants could understand simple bar graphs, letters and shapes on a tablet with vibration and audio. They found that participants were able to understand the information presented on the vibro-audio tablet as well as they could on a tactile graphic. This is promising as it indicates that simple code structure and block shapes might be understandable using a touchscreen. They did note that it was difficult for participants to move in straight line across the tablet and suggested using secondary cues for helping them to stay straight. This matched the early results from piloting a prototype of Blocks4All and led to my design decision to place blocks on the bottom of the screen (Figure 6.1).

6.2.2 *Accessible Block-Based and Visual Environments*

Block-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 6.1 compares (a) the Scratch environment, one of the earliest and most popular block-based environments with (b) the final version of the Blocks4All environment.

Existing block-based environments are generally not accessible (as discussed in Chapter 5), but there are two exceptions to this rule: work by Lewis *et al.* [68,71] and Google's Accessible Blockly [44] (described in more detail in Chapter 2). 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, my goal was to provide insights into how to make existing block-based environments universally accessible. To do so, I closely mimicked existing block-based environments and used touchscreens, which I believe may be easier for young children to use.

Looking more broadly at visual programming languages (section 2.2), Siegfried [116] created a scripting language to enable blind people to work in the Visual Basic environment. Visual Basic is a visual environment where users can create graphical user interface (GUI) forms by dragging and dropping components and controls in a form and then specifying details about them. In Siegfried's accessible solution, placement of GUI elements in done in a text-based file; the screen is divided into either three rows or three columns, and items sequentially added to a specified row or column. Although these constraints result in less freedom for the developer, it makes the forms much easier to specify and covered most use cases for the forms. I used this idea of constraining the spatial layout of blocks in the design of Blocks4All; blocks can no longer be placed anywhere on the screen, but only on the edge so they are easier to find.

Although not designed for children with visual impairments, Wagner *et al*. created a Scratch-based environment that can be navigated completely by voice, intended for children with motor impairments [136]. This provides a potential solution to the input problem (selecting and dragging a block) that blind children face when using a block-based programming environment, but another solution is also needed to help blind children understand what is on the screen and the structure of the program. I chose to use a touchscreen environment to facilitate this.

6.2.3 *Evaluation of Block-Based Programming Environments*

Because using block-based programming environments in educational contexts is fairly new, there are still a number of open questions about how effective they are, what concepts they teach well, and how well computer science knowledge transfers from block-based environments to text-based ones [14,72,103]. In a study of sixth-grade students in a summer enrichment program, Lewis found that after a week of programming in either Scratch or Logo (a text-based introductory programming environment), students who programmed with Scratch performed better on an assessment of their understanding of conditionals, but not loops (although conditionals were evaluated after five days, and loops after only two) [72]. She also found that students who learned Logo were more confident in their programming ability than those who learned Scratch. However, it's not clear if these differences are due to the blocks or text-based representations of the environment, or simply the different syntax and keywords.

Price *et al.* [103] evaluated two groups of novice middle school programmers as they completed an hour of code activity in outreach program, where each group used a version of their programming environment that supported only blocks or text so that differences in performance could be attributed to that. They found that the different interfaces did not affect attitudes or perceived difficulty of the task, but that students using the blocks interface spent less time off task and completed more activities.

Armoni *et al.* [14] found that students who learned first learned Scratch in a middle school computer science class took less time to learn new topics, had fewer learning difficulties and had a better level of understanding of most topics (loops, though not conditionals or variables) in Java and C# high school programming classes. Additionally, those students had higher self-efficacy and motivation. In general, there is enough support and curriculum for block-based environments

that it important that blind children to have access to them. Measuring learning outcomes was outside the scope of my own evaluation. Instead I focused on studying the usability of the interface, making sure that children with visual impairments are able to access the same types of curriculum and educational tools as their sighted peers.

6.3 Design Exploration

In designing the environment, I 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 block-based environments. Additionally, I wanted to support the features of block-based environments that make them suitable for young children:

- A lack of syntax errors due to blocks that are units of code, which can only fit in places that are syntactically correct,
- Code creation using a menu of blocks that relies on recognition instead of recall, and
- Clear hints about the type and placement of blocks, which in traditional block-based environments is conveyed via shape and placement of blocks.

I chose to develop Blocks4All on a touchscreen device, and specifically an iPad, for multiple reasons:

- They are popular among people with visual impairments and have a state-of-the-art screen reader that is built-in [154].
- They are easy to use and are often used by children in educational contexts (iPads are actually provided to every child in some school districts and are often provided as assistive technology for children with visual impairments both because of the screen reader and because they can be used to magnify things by taking a picture and zooming in).

- Many block-based environments are already available as touchscreen applications [38,170], and
- Touchscreens used with a screen reader allow for spatial exploration of the screen, which could be useful for conveying program structure.

6.4 INITIAL DESIGN

The initial design of Blocks4All was based on prior research and my own design exploration. I summarize my different approaches to overcome the five accessibility challenges I identified in Chapter 5.

6.4.1 *Accessing Output*

I created a block-based environment that can be used to control a Dash robot [142], as this makes for tangible output that is very accessible for children with visual impairments. I 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.

6.4.2 *Accessing Elements*

In my 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”). I 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 [58,123].

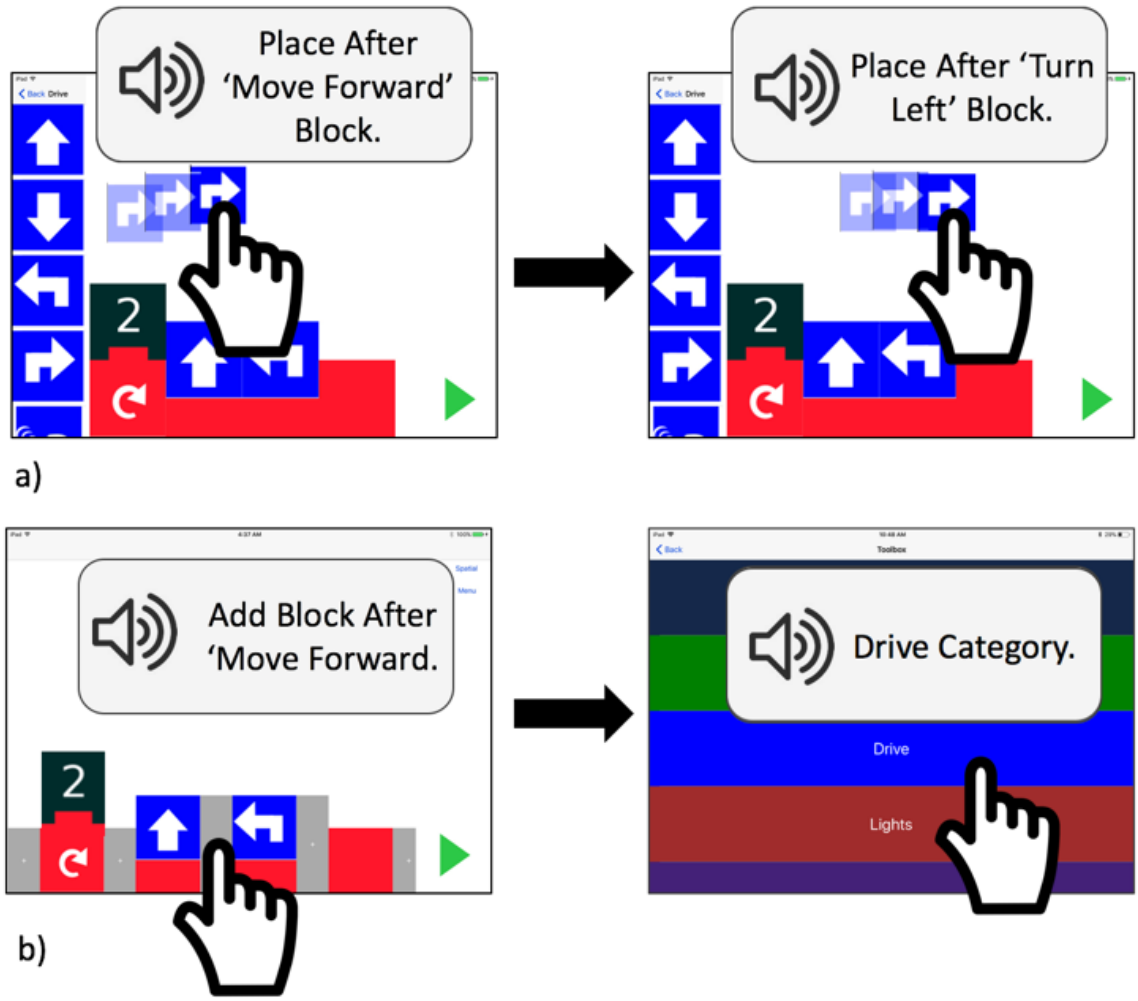


Figure 6.2. Two methods to move blocks: (a) audio-guided drag and drop, which speaks aloud the location of the block as it is dragged across the screen (gray box indicates audio output of program) and (b) location-first select, select, drop, where a location is selected via gray "connection blocks", then the toolbox of blocks that can be placed there appears.

6.4.3 *Moving Blocks*

I initially explored two methods to move blocks: (1) ***audio-guided drag and drop***, which had a similar set-up to traditional block-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 6.2a), 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 6.2b). This is slightly different from traditional block-based environments, in which you first select the block and then the location to place it. It is logically similar to the method used in Accessible Blockly, although the physical manifestation is different.

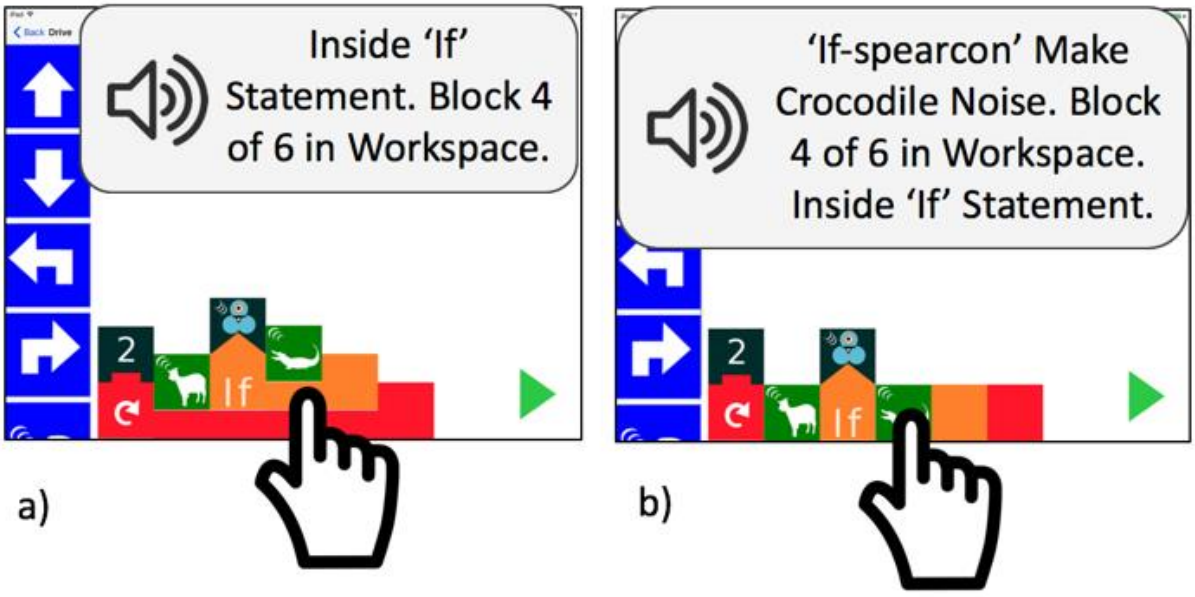


Figure 6.3. Two methods to indicate the spatial structure of the code: (a) a spatial representation with nested statements placed vertically above inner blocks of enclosing statements, and (b) an audio representation with nesting communicated aurally with spearcons (shortened audio representations of words).

6.4.4 *Conveying Program Structure*

I 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 6.3a). 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” [137] (Figure 6.3b).

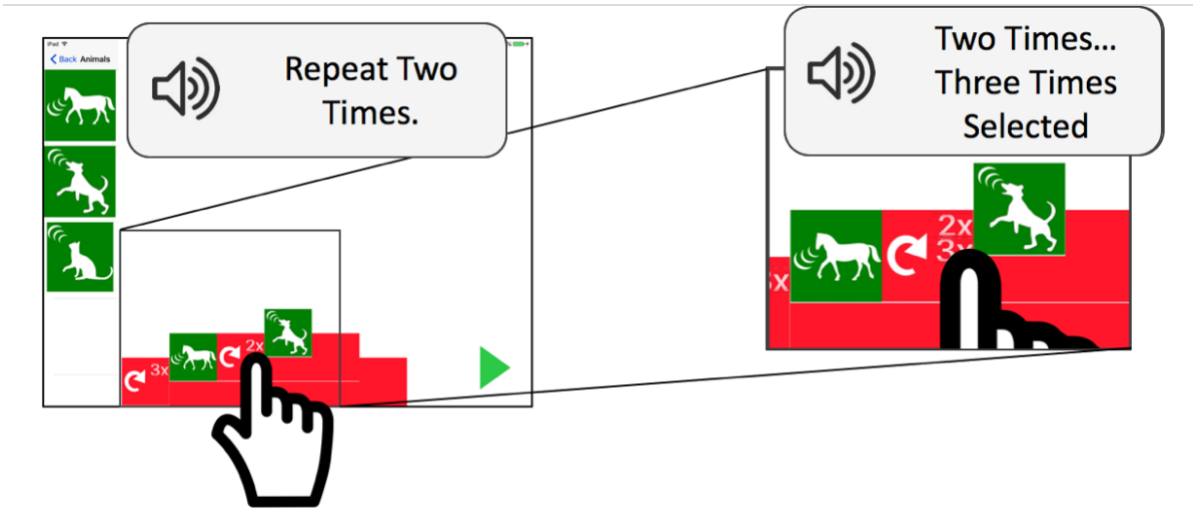


Figure 6.4. The first method to access different block types: embedded typed blocks, accessed from a menu embedded within each block (e.g. "Repeat 2/3 times")

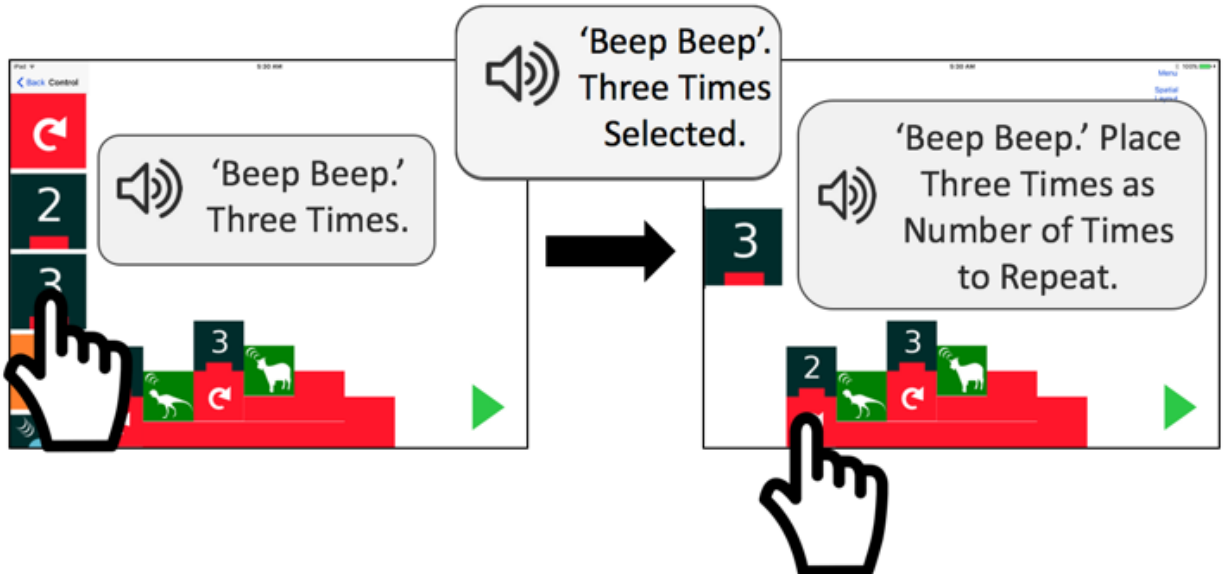


Figure 6.5. The second method to access different block types: *audio-cue typed blocks*, when a typed block in the toolbox and the blocks in the workspace that accept it play the same distinct audio cues.

6.4.5 *Conveying Type Information*

The 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). I 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 6.4). 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 [38] and the Dash robot [142] applications. I only prototyped the ***embedded typed blocks*** with numbers in the repeat and drive forward statements.

The second is ***audio-cue typed blocks*** (Figure 6.5). 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: Booleans have an orange triangle on the bottom to indicate they fit with orange ‘if’ statements, while numbers have a red rectangle at the bottom to indicate they fit with the red repeat statements (Figure 6.5). The visual approach is similar to traditional block-based environments (e.g. Blockly [43] and Scratch [142]), but the tabs are larger to accommodate users with low-vision. These blocks can only be placed as conditions of either repeat or if statements in the workspace, which is communicated aurally with VoiceOver (i.e. “3 times cannot be placed here”).

6.5 FORMATIVE STUDY

6.5.1 *Interview with Teacher of the Visual Impaired*

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

Table 6.6. Participants in Blocks4All Design Study

Participant	Age	Gender	Evaluation Sessions	Level of Corrected Vision	Previous Technology Use	Used Screen Reader in Evaluation	Used Blocks4All in a Later Study
P1	8	Female	4	20/150, difficulty focusing	Lots of experience with screen readers, uses iPads and tablets.	No	Yes. (P1 in Table 5.5 and in Table 6.7).
P2	8	Male	4	20/80-20/100	Uses iPad at school as assistive technology device with both VoiceOver and Zoom.	Occasionally	Yes. (P2 in Table 5.5 and in Table 6.7).
P3	5	Male	1	20/100	Uses iPads at home for games.	No	No
P4	10	Male	3	20/400	Uses VoiceOver on iPad and iPhone. Uses Apple computer, Braille, CCTV, Kindle.	Yes	Yes. (P4 in Table 6.7).
P5	9	Female	3	Totally blind, no light perception	Uses iPad at school with VoiceOver and refreshable Braille reader and Braille Note.	Yes	No

6.5.2 *Participants with Visual Impairments*

I recruited five children (3 male) aged 5-10 with visual impairments through my contacts with TVIs for a formative study (Table 6.6). 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.

6.5.3 *Method*

The children participated in one to four 60-90 minute sessions in which they programmed a Dash robot [142] using an iPad running iOS 10.3.3.

In each session, I introduced them to the interfaces using cardboard cutouts with Braille and large lettering to indicate the different application elements. For the first two 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 [142], 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, I introduced repeat loops, and the children used the two different methods for conveying program structure. In this session, I had tasks that required repeat loops (e.g. "Have Dash drive in a square using only one forward and one turn left block"). In the third or fourth sessions (depending on the participant), I introduced 'if' statements, and the children used two different methods for accessing type information. The two different methods for accessing type information were not counterbalanced in these later sessions. P1 and P2 used the ***embedded typed blocks*** in session 3 and the ***audio-cue typed blocks*** in session 4, and P4 and P5 used both in session 3. In these later sessions, I had tasks that required using different modifiers for repeat loops, drive blocks and

conditionals (e.g. “Have Dash ‘Say Hi’ if he hears a sound”). As I was gathering usability information about the interfaces and not testing how well the children understood the concepts, I provided any verbal help the children needed to figure out how to accomplish the tasks in all of the sessions.

6.5.4 *Analysis*

For each task, I video-recorded the children and evaluated the interfaces for usability issues. At the end of each session, I asked for feedback on the different interfaces. In the final session, I 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 [81]. 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, I report only summaries of my findings on the usability of the interfaces.

6.5.5 *Findings*

I report on the feedback from the formative study and interview, and the resulting changes to the design of Blocks4All. I discuss the findings in terms of the accessibility problems from Chapter 5.

6.5.5.1 Accessing Output

I 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”, I 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. I did not explore any other accessible programming outputs in the study, but parents of participants recommended adding the option of recording or typing your own audio for the robot to speak in future prototypes.

6.5.5.2 Accessing Elements

To answer to the first part of *BQ1: What touchscreen interactions can children with visual impairments use to identify blocks and block types?* I found that children in my study could access blocks in my 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. I elaborate on my 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 [145].

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 my 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. The children with some vision were all able to distinguish the icons for the different blocks from one another. The parents of P1, P3 and P4 noted that their children were not able to read 12-point size text on iPad, so they thought the icons were easier to use for their children.

Design Modifications: Based on the feedback of the children with some vision, I added high contrast white space between the blocks in the toolbox and made the blocks resizable 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.

6.5.5.3 Moving Blocks

In answer to *BQ2: What touchscreen interactions can children with visual impairments use to build blocks programs?* I explored multiple techniques to move blocks and found that (1) all the children in my 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. I elaborate on my 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 thought that it would be better to have more of the items on the screen at the same time, so they are easier to access with the item chooser.

Design Modifications: I created a hybrid of the 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 one and stated that they preferred this "hybrid" method to either of the two original methods.

6.5.5.4 Conveying Program Structure

In answer to *BQ3: What touchscreen interactions can children with visual impairments use to understand the spatial structure of blocks program code?* I found that children seemed to understand program structure using both the **spatial** and **audio representations** I explored, and all the participants who had an opinion preferred the **spatial representation**.

Initial Findings: The participants were able to understand both the ***spatial*** and ***audio representations*** of the program structure for singly nested repeat loops in Session 2. 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). However, it should be noted that these tasks were fairly simple: they only involved one level of nesting and only involved repeat loops (e.g. "Have Dash drive in a square" required them to put a drive forward and a turn left block inside a repeat block). 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 blind children 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 he thought that the audio presentation "looked nicer". The spatial representation has the added benefit that it has visual cues to better convey the structural information to children with some vision.

Design Modifications: After the second session, I used the ***spatial representation*** to convey program structure. I modified the order that VoiceOver reads the blocks when swiping through the elements in the page so that it focuses on the contained block first followed by the "Inside _" blocks (e.g. "Drive Forward Block" followed by "Inside Repeat Two Times").

6.5.5.5 Conveying Type Information

In answering the second part of *BQ1: What touchscreen interactions can children with visual impairments use to identify blocks and block types?* I 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. And while the participants were able to build programs using the different types of blocks, it is unclear whether they understood the type differences. I elaborate on these findings below.

Initial Findings: All participants, other than P3 who did not participate past session 1, 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. I 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 (including an empty workspace), indicating that the visual cue was not enough. Additionally, although the children found the number blocks 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.

Design Modifications: Although it was more difficult for the children to grasp, I 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, I 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.

6.5.5.6 Other Feedback

I received positive feedback on my interfaces. The children liked the all the interfaces: all five 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 (2, SD=1.75). All the children chose to continue playing with the interfaces after completing the tasks.

6.6 Final Design

This section describes the design of Blocks4All after making the modifications described previously.

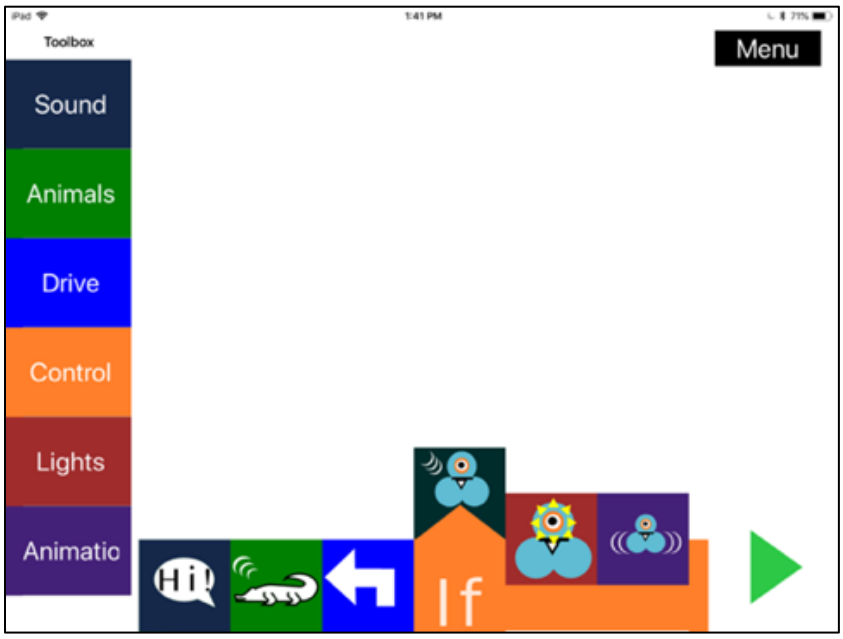


Figure 6.6. The categories in the final version of Blocks4All: sound, animals, drive, control, lights, animation. An example block from each category is in the workspace.

6.6.1 *Accessing Output*

In the final version of Blocks4All, the application is still used to program the Dash robot, to do tasks that would be perceivable by children with a wide range of visual impairments. The final version contains six categories of blocks (Figure 6.6):

- Sound (example blocks: “Say Hi”, “Makes a confused noise”),
- Animals (example blocks: “Make a crocodile noise”, “Make a dinosaur noise”),
- Drive (example blocks: “Drive forward”, “Turn left”),
- Control (example blocks: “Repeat _ times”, “If _”),
- Lights (example block: “Turn on left ear light”),
- Animation (example blocks: “Dance”, “Wiggle”).

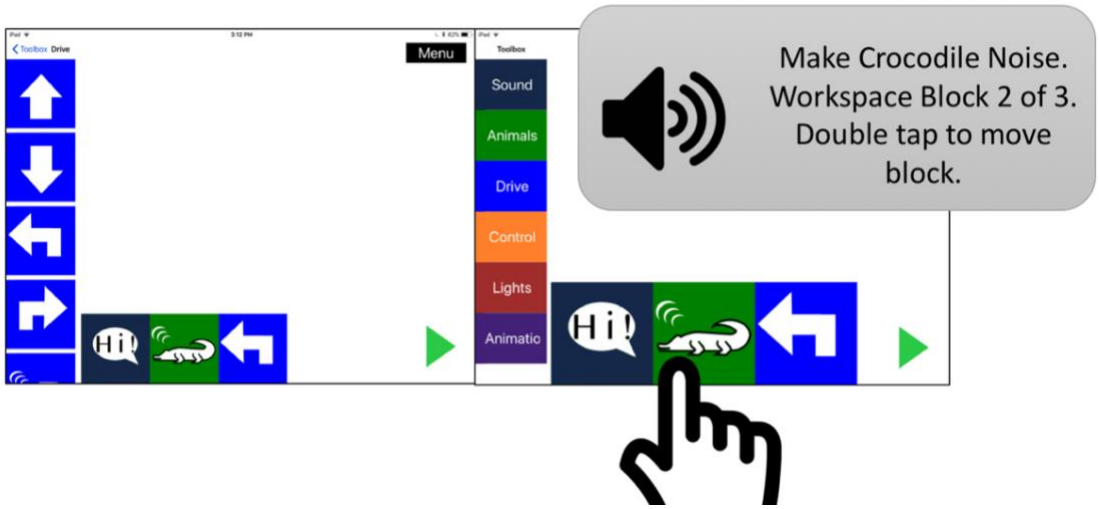


Figure 6.7. Blocks in the workspace are resizable in the final version of Blocks4All. When a block is touched in the workspace or toolbox with screen reader on, it announces the name, location and how to move the block (see gray callout).

6.6.2 *Accessing Elements*

Accessing the blocks remained largely unchanged from the first version to the final version of the Blocks4All design. Blocks use high-contrast white icons on a dark background to make it easier for children with low-vision to use them. There is white space between the blocks in the toolbox to make it easy to distinguish between the blocks, and the blocks are resizable in the final version (Figure 6.7). When accessed with a screen reader, the blocks read the name of the block (e.g. “Make crocodile noise”), followed by the location (e.g. “Workspace block 2 of 3), and how to move it (“Double tap to move block”). The blocks are all located at the bottom of the screen to make them easier to find without sight.

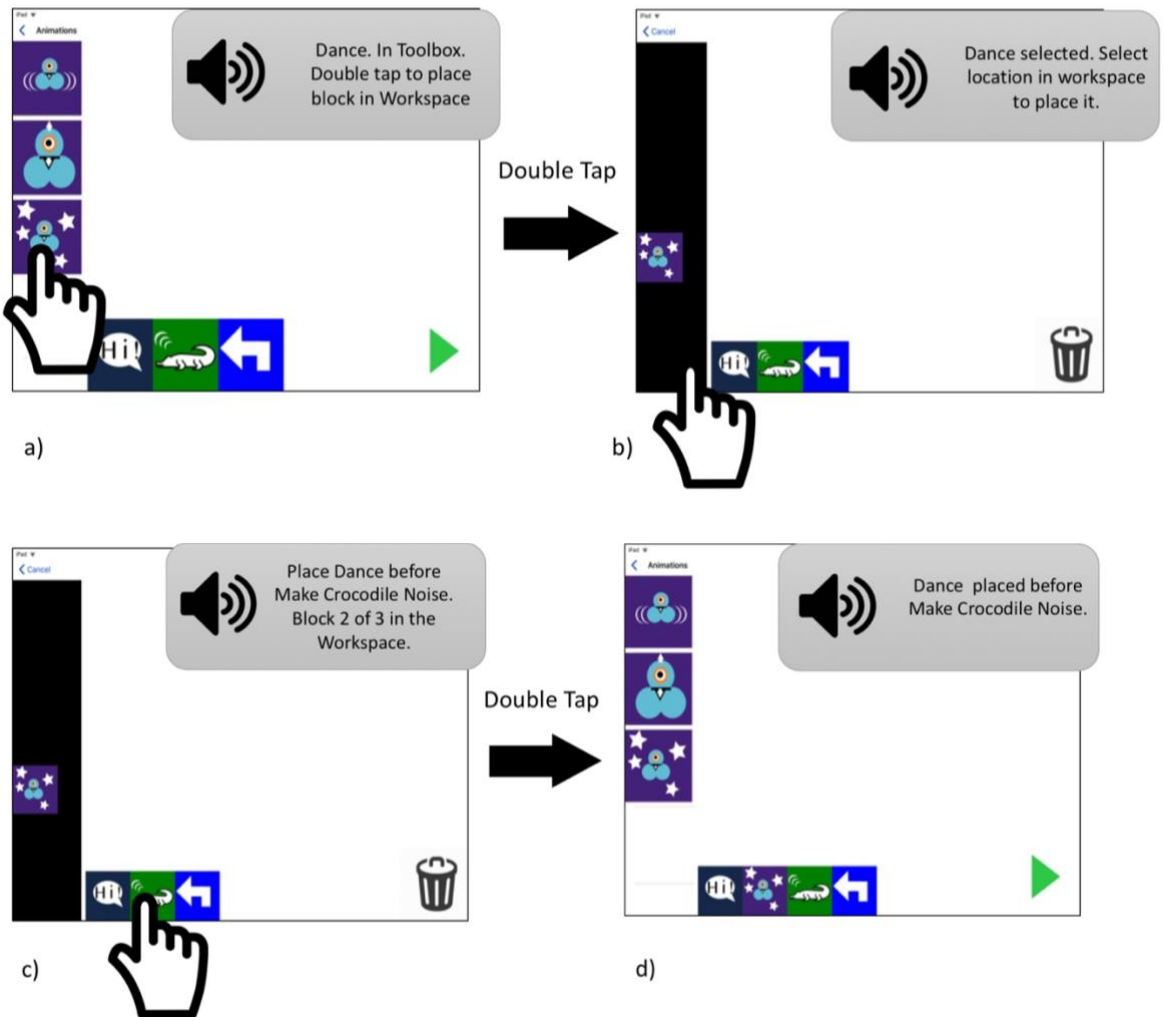


Figure 6.8. Blocks are moved via blocks-first select, select, drop. (a) A block is first selected from the toolbox via double tap, (b and c) the application then switches into "selection mode", in which the toolbox menu is replaced with information about the selected block, (c and d) when a location is selected in the workspace, the block is moved there.

6.6.3 *Moving Blocks*

In the final design, blocks are moved via ***blocks-first select, select, drop***. Blocks are selected from the toolbox (either with or without the screen reader on) (Figure 6.8a). The application then switches into "selection mode," where the toolbox contains information about the selected block (Figure 6.8b). In this mode, the user can place the block by selecting a location (Figure 6.8c).

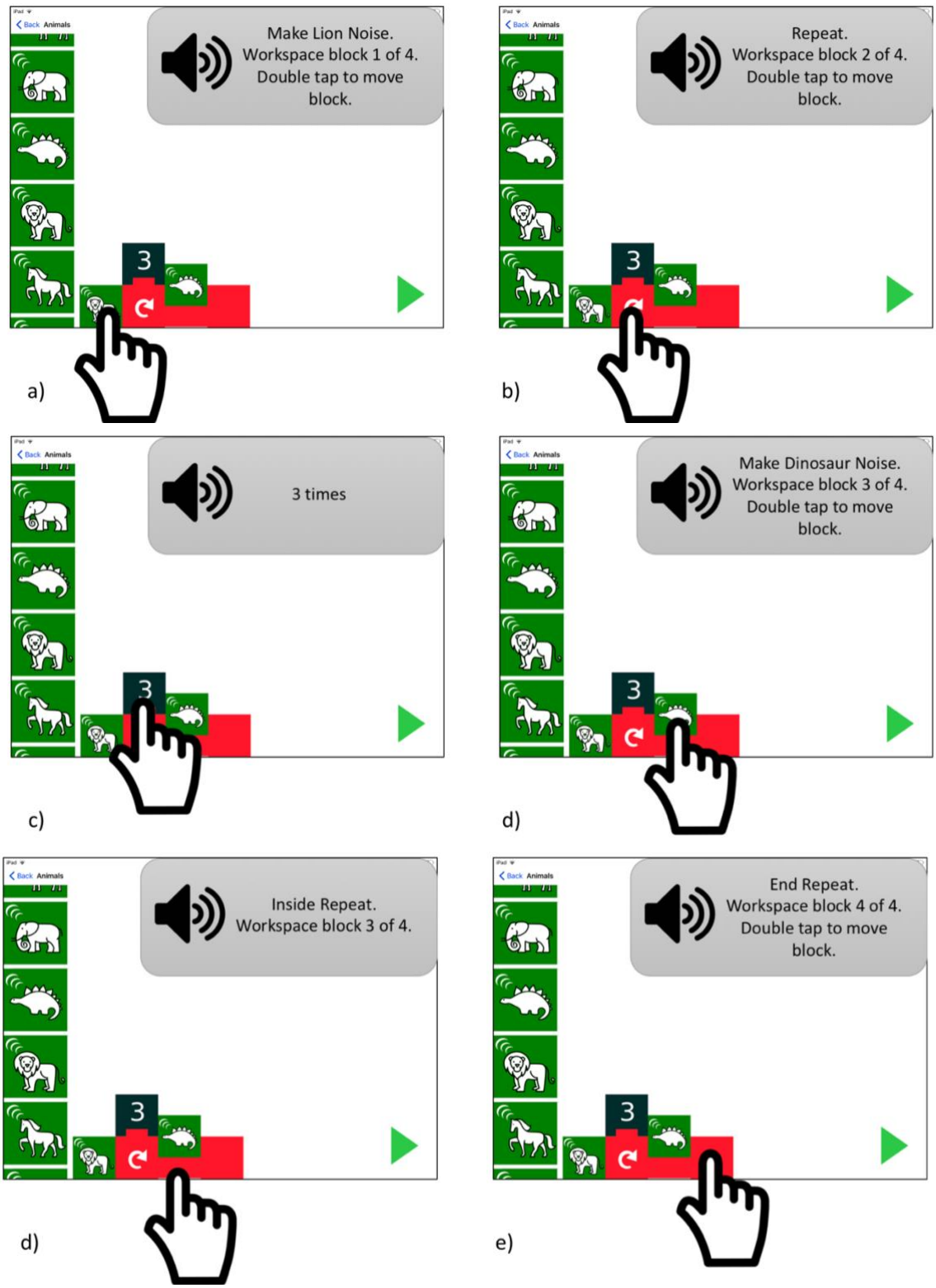


Figure 6.9. Spatial representation of program structure. A repeat loop has an opening block ("repeat") with a modifier ("3 times"), inner blocks ("inside repeat") and a closing block ("end repeat). Gray boxes indicate what the screen reader reads for each block.

6.6.4 *Conveying Program Structure*

In the final design, program structure is conveyed through *spatial representation* (Figure 6.9). A nested statement is opened by the nesting block (e.g. “Repeat”), located directly above the nesting block is any modifying block (e.g. “3 times”), then any nested blocks are located to the right of the nesting block and are above an “inside” block (e.g. a block that readers “Make dinosaur noise” is directly above one that reads “inside repeat” in
Figure 6.9). Finally, any nested statement ends with an “End” block (e.g. “End Repeat 3 times). When swiping through the elements with VoiceOver on, the blocks are read left to right and then top to bottom.

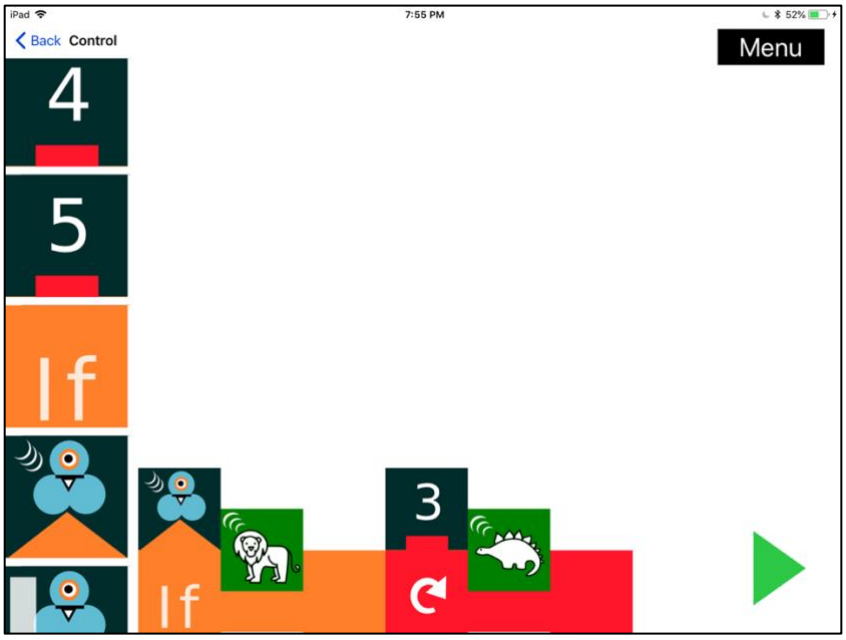


Figure 6.10. A program containing the three types supported by Blocks4All: numbers, Booleans and operations. There is an “If Dash hears a sound” block with a “Make lion noise” nested inside. This is followed by a “Repeat 3 times” block with a nested “Make dinosaur noise” block inside. The orange triangle on the bottom of the “hears a sound” block that matches the orange of the “if” statement, and the red rectangle on the bottom of the “three times” block that matches the red “repeat” block. The “if” statements and conditions make two short high-pitched notes when touched with VoiceOver on to indicate that they fit together. The “repeat” blocks and numbers make two low-pitched notes.

6.6.5 *Conveying Type*

In the final design, type is conveyed through both shape and color to benefit children with low-vision and sound for children who use screen readers. There are three supported types in Blocks4All: (1) numbers which modify repeat loops, (2) Booleans which modify if statements, and (3) operations (Figure 6.10). The numbers have a red rectangle on the bottom of the block to indicate that they fit in the red repeat block. They make two short low-pitched notes in the toolbox which matches the low-pitched notes that the repeat blocks make to indicate they fit. The Boolean conditions have an orange triangle at the bottom of the block to indicate they fit the orange "if" statements. They make two short high-pitched notes, which match the sound that "if" statements make.

6.7 EVALUATION

I ran an evaluation to determine the usability of the final design of Blocks4All. I was particularly interested in whether the design allowed children with a wide range of visual impairments to overcome the five accessibility challenges identified in Chapter 5: (1) access output, (2) access elements, (3) move blocks, (4) understand program structure, and (5) understand type, so the evaluation tasks were designed to test the participants on these aspects. I also particularly wanted to test the application with more children who were primarily screen readers, and I was able to recruit two more screen readers to evaluate the final application.

6.7.1 *Method*

The children participated in one 30-60 minute sessions in which they programmed a Dash robot [142] using an iPad running iOS 10.3.3. If the participants were unfamiliar with Blocks4All ($P_2$3

and $P_2$5), they were given an approximately 15-minute introduction to the interface, otherwise they were given a five-minute review of the interface.

The children were then asked to complete four tasks on each of the interfaces. In Tasks 1 and 2, the participants were asked to build and then modify a blocks program, addressing if they could (1) access output, (2) access elements, (3) move blocks and (5) understand type. Task 1 was a simple task that consisted of placing three operation blocks in the workspace and then moving one of them to a different place in the program. Task 2 was a more complex task in which the participants created a program with a repeat statement that was nested inside of a conditional statement with step by step instructions. In Tasks 3 and 4, participants were asked to describe the program structure of an existing blocks program with nested statements, addressing if they could (2) access elements, and (4) understand program structure.

Table 6.7. Participants in Blocks4All Evaluation Study. $P_2$3 and $P_2$5 are given subscripts to distinguish them from P3 and P5 of the formative study.

Participant	Age	Gender	Level of Corrected Vision	Previous Technology Use	Used Screen Reader in Evaluation	Used Blocks4All Before
P1	8	Female	20/150, difficulty focusing	Lots of experience with screen readers, uses iPads and tablets.	No	Yes. (P1 in Table 5.5 and in Table 6.6).
P2	8	Male	20/80-20/100	Uses iPad at school as assistive technology device with both VoiceOver and Zoom.	No	Yes. (P2 in Table 5.5 and in Table 6.6).
$P_2$3	10	Male	Color and light perception but no real usable vision	Uses an iPad almost daily for school work and homework. Uses the iPad in conjunction with a refreshable Braille display for reading digital books.	Yes	No, but also tried Tickle Application (P3 in Table 5.5).
P4	10	Male	20/400	Uses VoiceOver on iPad and iPhone. Uses Apple computer, Braille, CCTV, Kindle.	Yes	Yes. (P4 in Table 6.6).
$P_2$5	10	Female	No usable vision	Uses iPod touch for music, computer and BrailleNote.	Yes	No

6.7.1.1 Participants

I worked with five children (3 male) aged 8-10 with visual impairments (Table 6.7). Three of the children (P1, P2, P4) took part in the formative study for Blocks4All, which took place 5 months before the start of the evaluative study. I use subscripts to distinguish between $P_2$3 and $P_2$5 who participated in this study and P3 and P5 who participated in the formative study. The children were recruited through my contacts with TVIs. Three of the children ($P_2$3, P4, $P_2$5) used VoiceOver exclusively during the evaluation (P4 has some residual sight). The other children relied entirely on sight. Of the participants that relied on sight, none used the magnification tool on the iPad. Instead they chose to hold the application close to their faces to read.

6.7.1.2 Analysis

I measured the time it took to complete the task and (for Tasks 3 and 4) the accuracy in completing the task. Accuracy was not computed for Tasks 1 and 2, as the children were able to complete all the actions required for the tasks. For these two tasks, the participants received any verbal help from the researcher that they needed to complete the tasks. I describe the help that they received on Tasks 1 and 2 and report any on other usability problems that they encountered in the qualitative findings.

For Tasks 3 and 4, children were asked to read out which blocks were in the workspace and in what order. Accuracy was determined on a 7-point scale, using a string-matching algorithm. Participants lost a point for each operation needed to convert their guess into the correct answer. The operations were (1) insertion (they added a block that was not there), (2) deletion (they did not find a block), (3) substitution (they thought one block was another, e.g. thinking a block was a repeat 2 times block instead of a repeat 3 times block) and (4) transposition (they swapped the

order of two blocks). If participants made more than seven mistakes, they would receive a score of 0. The 7 points corresponded to the 7 blocks in the workspace (including the "end" blocks for both "if" and "repeat" statements).

At the end of the session, I asked children to rate the amount of fun they were having on a 5-point Likert scale. I also asked them what they found hard or difficult about using the application. Finally, I asked the children questions from the System Usability Scale [26]. The questions were changed to match the terminology from the study (e.g. "I thought the system was easy to use" became "I thought Blocks4All was easy to use"), and I elaborated on the some of the wording if the children found it difficult to understand (e.g. "I found Blocks4All very cumbersome to use," cumbersome became "awkward" or "took a lot of work to use.")

Table 6.8. Time (Minutes: Seconds) to Complete Tasks in Blocks4All

Participant	T1	T2	T3	T4
P1	2:03	6:04	0:13	0:30
P2	1:26	6:46	0:19	0:26
$P_2$3	7:41	4:58	0:56	0:46
P4	3:30	4:47	0:25	0:42
$P_2$5	6:26	10:58	1:10	2:21

6.7.2 *Quantitative Findings*

6.7.2.1 Time to Complete Tasks

I found that all the participants were able to complete each of the tasks in under 11 minutes per task (Table 6.8). It should be noted that participants did not stay completely on task for many of the tasks, and I did not actively discourage them from taking detours if they were excited to discover what a particular block did. This meant that they often added extra blocks to the workspace even if it was not part of the task or pressed play multiple times if they thought the task was funny. I considered Tasks 1 and 2 (the build and modification tasks) to be "finished" after

they had successfully added and moved the blocks that were listed as part of the program and then then removed them all from the workspace. I considered Tasks 3 and 4 (the understanding tasks) to be "finished" after the children had finished reading the program, and after they had answered any follow-up questions needed to clarify their answer (i.e. if they did not answer how many times a repeat loop repeated, I asked them).

6.7.2.2 Accuracy on Tasks

All the children were able to accurately read what blocks were in the workspace for Task 3 (i.e. received a score of 7). In Task 4, both P1 and P4 made the same mistake and received a score of 6 (all other participants received a score of 7). They thought that the "Say Cool" block would be repeated twice, although it was placed outside the repeat loop and inside the conditional. Interestingly, P4 used a screen reader during the evaluation, and P1 used her sight.

Table 6.9. Response to the System Usability Scale. 5-Point Likert scale (1 is strongly agree, 5 is strongly disagree). The overall score was calculated by subtracting the odd-numbered response from 5 and subtracting 1 from the even-numbered responses and then multiplying by 2.5).

	P1	**P2**	**P$_2$3**	**P4**	**P$_2$5**
I think that I would like to use Blocks4All frequently.	1	2	1	2	2
I found Blocks4All unnecessarily complex.	5	4	4	4	4
I thought Blocks4All was easy to use.	1	2	1	2	3
I think that I would need the support of a technical person to be able to use Blocks4All	4	3	5	4	5
I found the various functions in Blocks4All were well integrated.	1	1	1	2	2
I thought there was too much inconsistency in Blocks4All. (unexpected)	5	5	5	3	4
I would imagine that most people would learn to use Blocks4All very quickly.	3	1	1	2	3
I found Blocks4All very cumbersome to use. (awkward)	5	5	5	4	4
I felt very confident using Blocks4All	1	3	1	2	2
I needed to learn a lot of things before I could get going with Blocks4All.	5	5	5	3	5
Overall Score	92.5	82.5	97.5	70	75

6.7.2.3 Post-Session Interview

In the post-session interview, the participants gave the interface the average score of 83.5 on the System Usability Scale. This indicates that they found the system to be usable, as any score above 80 ranks in the top 10% of scores (Table 6.9) [190].

6.7.3 *Qualitative Findings*

I describe the verbal assistance the participants needed to complete Tasks 1 and 2 and report on other usability problems. I describe the findings in terms of the five accessibility barriers from Chapter 5.

6.7.3.1 Accessing Output

All of the children were able to access the robot output. Many of the children used the robot to debug their programs. For example, P1 noticed that she placed the "dance" block outside of the repeat loop when she meant to place it inside, after running the robot and noticing that he only danced once on Task 2. She then made the change to the program.

6.7.3.2 Accessing Elements

All of the children were able to access the elements. If they had low-vision, they were able to distinguish the blocks from one another, although they were not always sure what the icons meant (although they could find out by getting the robot to perform the action). All of the children who used screen readers were able to find and hear the blocks in the workspace and the toolbox. However, P_23 had to be reminded to check at the bottom of the screen for blocks that were in the workspace in Task 1. He had no more problems with this after the first reminder. P_25 had to be reminded where the "back" button was to access the rest of the toolbox in the first step of Task 1. She was able to access this button for the rest of the session without problems.

6.7.3.3 Moving Elements

All of the participants were able to move blocks from the toolbox to the workspace and within the workspace. They were also all able to clear the workspace by placing blocks in the trash. One usability issue that came up is that everything inside a repeat loop is moved with the repeat loop. This is not communicated well with the screen reader. Both P1 and P25 were surprised when they accidentally threw out everything inside a repeat loop when discarding the repeat loop.

6.7.3.4 Conveying Program Structure

The participants were able to describe the program structure of existing programs in Tasks 3 and 4. However, two participants, P1 and P4, were incorrect about the nesting level of a block in Task 3. P1 did not use a screen reader and P4 did. Both participants seemed to understand their mistake after running the program on Dash. P4 noted "Oh, so he only said cool once." When asked why, he replied "cause it's lower…it can't actually be there *(inside repeat two times)* because there's a space there. It's inside the if." P1 and P25 also needed help in building up the complex program structure in Task 2, in particular they needed making sure that things they wanted to be repeated were inside the repeat loops.

6.7.3.5 Conveying Type

All of the participants needed extra coaching on Task 2 when building a program that used the Boolean type for a conditional statement and the Number type for a repeat statement. The conditional statements seemed to be the most confusing. For all the participants in Task 2, I walked them through adding an "if" statement to the workspace and what the default condition ("false") meant (highlighting a problem with audio interfaces, P25 thought it was "falls"). I then walked them though adding the "hear voice" condition and the rest of the task.

Some participants also had trouble in Task 2 with creating repeat loops. When asked to make something repeat 3 times, both $P_2$3 and P4 tried to add a 3 times block without having a repeat block inside the workspace. Participants also had some trouble finding the Boolean conditions that would fit inside the conditional statement. Both P1 and P2 (not screen reader users) had to be reminded to look for the golden triangle at the bottom of the condition. It was unclear if the sound effects helped the screen reader users to find the conditions. This was reflected in their comments about what was difficult about the interface at the end of the session. P2 noted, “I don't really like the conditions. I don't think they are that fun to use. They are fun sometimes, but they are not fun all the time.” And $P_2$5 said, “Changing condition was kind of hard. It was kind of hard to find the condition that you wanted I knew what I wanted to do. It was just hard to do.”

6.7.3.6 Post Session Interview

All of the children had positive things to say about the application in the post-session interview, and they all reported that they thought the tasks were a 5 or “really fun” on a Likert fun scale. When P1 was asked about her favorite part, she said “"Everything." And when asked can you narrow it down? She replied, "No cause it's true I liked everything.” When asked about his favorite part, P2 said, “I like the driving. I kind of like that it's a little puzzle." $P_2$3 said, “It was a little complicated but not too much. It wasn't overwhelming, and it was really fun too. It was really fun. I really enjoyed programming. It was interesting to figure out how the repeat loops and stuff worked and how you could move the blocks.” $P_2$5 said, “It's really, really cool, I think more people should get to play with it.”

6.7.4 *Discussion*

I found the two quantitative measures from my evaluative study, time and accuracy, were not the most descriptive measures for this type of study. The children often wanted to play around with the interface while doing tasks, meaning that the time measured was not always reflective of the time it would take them to do the tasks. Also, I did not want children to be discouraged while performing the tasks as this was their first introduction to programming, so I provided hints to support the Tasks 1 and 2, which made it difficult to report on accuracy. I found that most of the participants did not need hints on Task 1 (creating and modifying a simple sequential program) but did need hints on Task 2 (creating and modifying a program that had a conditional block and a repeat loop). Task 2 was a much more difficult task and most of the confusion was around the conditional statements. This was true even for the three participants who had encountered conditional statements before (P1, P2, P4). This confusion may simply be an artifact of the complexity of these types of statements or the fact that I did not explain conditionals well, but it remains to be seen if a better non-visual metaphor can be found to replace the puzzle-piece metaphor of traditional block-based environments.

For the most part, participants were able to describe the program structure of existing programs. However, the tasks did not involve blocks that were nested more than two blocks high. It is possible that more complex programs would have made it more challenging for participants to find the blocks with the screen reader. Also, the participants were only asked to describe what the blocks were in the workspace, so they did not necessarily show that they understood what a program would do. I found that the children were able to independently physically interact with the application; however, for the more complicated programming concepts (repeat and conditional statements) they needed explicit verbal instructions to construct the programs.

There are also a number of limitations to my study and evaluation of the environment (limitations of this study are discussed further in Section 6.6). I did not do a comparison to other interfaces in my study as I could not find an accessible environment that was usable by this population. Because of this, many of my findings (especially around enjoyment) might be about the Dash robot and certainly my findings about the usability of the interface may be specific to the language that I created and the way that I taught it. I found that the children were able to understand a simple hierarchical structure using the spatial layout and audio feedback describing the position of the blocks. This could generalize to understanding a (shallow) abstract syntax tree, (AST) and therefore more broadly to other programming languages. The interaction technique of building the AST using the toolbox to select, select and drop a node in the AST (similar to other block-based editors) could be universally applied to any programming language, and currently the same interaction is used to build expressions, although the programming language used in Blocks4All does not currently support complex expressions. One area that I have not explored, and which limits the expressiveness of Blocks4All is there is no way to input data, assign variables or create functions. It would be interesting to explore how to do this well in a non-visual manner for children who have not yet learned to type; perhaps through speech or gestures.

6.8 Design Guidelines

Based on these studies, I distill design guidelines for designers to make both block-based environments and touchscreen applications usable by children with visual impairments. In particular, I 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.

6.8.1 *Make Items Easily Locatable and Viewable on Screen*

In agreement with previous work [126], I found that the children in my 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, I made the blocks resizable instead, so children could see the blocks without occluding other parts of the application.

I found it was important to locate elements close to the edge of the screen, so they could be found without vision, as the participants found it difficult to navigate "floating" segments of code. Kane *et al*. [58] followed a similar guideline when designing interactions for large touch interfaces, and I found it equally important for a standard-size iPad. I recommend making elements resizable and high contrast and locating elements close to the edge of the screen to make them findable.

6.8.2 *Reduce the Number of Items on Screen*

Simple interfaces are easier to use for children in general, and I found it was especially important to reduce the number of focusable items on the screen for both the 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, I found it was important to not have multiple screens needed to perform an interaction. For example, in the ***location-first select, select, drop*** method used in the formative evaluation, 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.

6.8.3 *Provide Alternatives to Drag and Drop*

I 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 I 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 my knowledge, this is a novel finding, although it aligns well with work by Szpiro *et al*. [126] 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. I 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. Select, select drop has the added benefit that it works well with Switch Control, making it usable by children with motor impairments. Blocks4All is accessible with Switch Control on the iPad.

6.8.4 *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. I also found that when I 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 block-based programming environments accessible to children with visual impairments (Accessible Blockly [44] and work at the University of Colorado, Boulder [68,71]), have focused on creating an interface that works well with a screen reader, but without many visual elements. These interfaces would be less appealing and likely more difficult to use for children with some vision.

Dash, the talking robot I used in my study, was very popular with all of the 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 block-based programming, which rely heavily on visual output for programming tasks (e.g. programming the avatars in Scratch or ScratchJr [38,78]). In the future, I 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.

6.8.5 *Design for Collaboration*

Although I planned to do the design and evaluation sessions with only one child at a time, I found that often the children wanted to play with the application simultaneously with other participants. In particular, P1 and P2 participated in the sessions in the same house and at the end of each session would build (usually very large) programs together. Additionally, both P$_2$3 and P4 had younger brothers who were sighted that attended the sessions. They would often join in after the session was through to build massive programs. Other block-based programming environments [38] are designed to allow for collaboration, and it is important to design environments for children with visual impairments that can be used collaboratively with sighted peers to ensure these children are fully included and develop necessary teamwork skills. One advantage to using a mainstream touchscreen device is that it is an interface that can be used by both children who are sighted and those who are blind.

6.9 LIMITATIONS

There are several limitations of my studies. Because of the difficulty in recruiting from this population, I had a small number of participants, with a wide range of ages and abilities, making

it difficult to make comparisons between participants. In the formative study, I explored only a small number of design possibilities to make these environments accessible and were constrained by my desire to only make changes that could be adopted by all designers of block-based environments. In my evaluation, I did not measure learning outcomes, but focused on the usability of the interfaces. Many of the children who participated in the evaluative study had also participated in the formative studies. In both studies, I provided verbal direction and hints to the children when creating and modifying programs. So, while the children were physically independently accessing the program, it remains to be seen if they could fully access the programs without adult help. Additionally, most of my measures were self-reported responses by the children, and it has been shown that participants, especially those with disabilities, artificially report positively in user studies in order to please the researcher [132].

6.10 Future Work

There are still open questions I 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. There is also a lot of room to explore different modes of output for these programs: incorporating music, storytelling or the different sensors such as the gyroscope or microphone on the smart device.

6.11 Summary

I conducted an evaluation of current block-based environments and found five accessibility challenges. I designed multiple techniques to overcome these challenges and conducted a formative study to evaluate these techniques with five children with visual impairments. I distilled the findings from this study into a final design, which I evaluated with five children with visual

impairments, who were able to write and describe programs. I found that the children enjoyed interacting with Blocks4All and were able to physically independently interact with the applications, although they needed verbal instruction to create complex programs. Based on these studies, I provide a set of design guidelines and novel interaction touchscreen techniques; including (1) evidence that the drag and drop interaction (even with audio guidance) is not accessible for many children with low vision as well as those who are blind, and an accessible alternative in select, select, drop with a "selection mode" that allows children to reference what has been selected and which gives verbal and sound guidance on where items can be placed, and (2) evidence that children with visual impairments can understand and build a hierarchical structure (such as a block-based program) based on audio feedback on the spatial location of blocks that are built-up from the bottom of the screen. The contributions from these studies are the Blocks4All environment and design guidelines for designing block-based environments for children with visual impairments.

Chapter 7. CONTRIBUTIONS AND FUTURE WORK

In this chapter, I discuss how my work supports my thesis statement and limitations of my work. Finally, I discuss future directions for this research and broader implications of my thesis.

SUPPORT OF THESIS STATEMENT

I made the following thesis claims:

Children with visual impairments use mainstream touchscreen devices for learning in a variety of ways, despite having to overcome accessibility challenges in order to use them. In the context of block-based programming environments, these challenges include a reliance on visual metaphors and gesture-based interactions that do not interface well with the screen reader. To improve access to learning technologies on these devices, we can design applications that (1) use the spatial information afforded by the touchscreen to help children understand structural information such as the layout of a Braille cell or the structure of program code, and (2) use the visual, haptic and audio feedback of these devices to engage children and allow them to use these applications independently.

My work in Chapter 3 supports the first claim that children with visual impairments use mainstream touchscreen devices for learning in a variety of ways, despite having to overcome accessibility challenges in order to use them. In interviews with teachers, I found that children with visual impairments use touchscreens in a variety of contexts. Children with low-vision use the camera as magnification aid to make in-world information more accessible and also use the accessibility features on the device to access educational applications. Children who are blind often learn to use touchscreen devices earlier than traditional computers, and this is often their first

introduction to screen readers. However, when using these devices, the children have to overcome usability problems and their own fear of standing out.

My work in Chapter 5 supports the second claim that in the context of block-based programming environments, these challenges include a reliance on visual metaphors and gesture-based interactions that do not interface well with the screen reader. I did an evaluation of 26 block-based programming environments. I found that most of the environments relied on the drag and drop gesture, which was difficult if not impossible to use with a screen reader, and on visual metaphors to convey type information and program structure, which were not accessible without sight.

My work in Chapters 4 and 6 supports the third claim that to improve access to learning technologies on these devices, we can design applications that (1) use the spatial information afforded by the touchscreen to help children understand structural information such as the layout of a Braille cell or the structure of program code, and (2) use the visual, haptic and audio feedback of these devices to engage children and allow them to use these applications independently. To support this claim, I built the BraillePlay games and the Blocks4All environment. With one exception, the children were able to use both applications independently. They were also able to understand the structural information that was presented spatially on the touchscreen. With the BraillePlay games, this took the form of the touchscreen being split into six regions to represent the six dots that can make up a Braille character. With Blocks4All, this took the form of the program structure being conveyed through the spatial layout of the blocks on the bottom of the screen. With the BraillePlay games, the children reported that they enjoyed using the games and in interviews I found they were excited to have a Braille-based game that they could play on a

smartphone. With the Blocks4All environment, the children reported that they enjoyed programming and using the application.

7.1 IMPLICATIONS FOR DESIGN

In designing the two touchscreen-based learning technologies and evaluating them with children with visual impairments, I started the designs with a number of guiding principles. I also developed a number of design guidelines that I found particularly important in working with this population based on insights from my studies. I outline these guidelines below.

7.1.1.1 Design for Mainstream Technology

Previous work has found that there is often a stigma associated with assistive technologies [115] and in my interviews with TVIs, I found that many children, especially those with some usable vision, did not want to use specialized assistive technologies. They felt that using specialized technologies made them stand out, so they preferred to use mainstream technologies. In my evaluations for Blocks4All and BraillePlay, I found that children and parents were excited about practicing these skills on a smartphone or iPad. Designing for a mainstream device, such as an iPad or smartphone, has the added benefit that the device is usable by children both with and without vision, so it is possible to use the applications collaboratively.

7.1.1.2 Design for Collaborative Play

In my evaluations for both BraillePlay and Blocks4All, I found that sighted siblings and parents wanted to play the games as well. As learning to work and play collaboratively is an important skill for children to develop and helps children to learn [69], it is important to design learning technologies that support this. In the BraillePlay study, I only designed one game, VBGhost, specifically as a multiplayer game, but found in the post-study interviews that parents and siblings

took turns and played the other games with the participants. In designing the Blocks4All environment, I designed an environment that would allow for more collaborative play. This meant that I conveyed information (such as the type of block or the program structure) though both vision and audio and made an environment that could be used both with and without a screen reader. In my evaluation, although I planned to work with only one child at a time, I found that children often wanted to play the games simultaneously and would build programs together during the free play time. This guideline is in line with recent work on creating collaborative learning environments to support children who are neurodiverse [118] and who have visual impairments [129]. However, it is worth emphasizing, as often researchers explore creating separate interfaces when designing learning technologies for children with disabilities.

7.1.1.3 Convey Information in Multiple Ways

I found in the interviews with the TVIs, as well as during the evaluation of my applications, that it is important to design applications for children with visual impairments that convey information in as many ways as possible: visually, tactilely, and aurally. Conveying information in multiple ways supports children with low-vision who prefer to use their sight, children with additional disabilities (such as deafness or motor impairments) and allows children with a wide range of vision to use the applications collaboratively. In my interviews with TVIs, I also found that applications that support both children with low-vision and those who are completely blind can be important tools to use with children who have degenerative conditions. The teachers found that tools like Fusion, which combines the JAWS screen reader with ZoomText, were more acceptable to students who were losing their vision and had associated emotional distress.

The majority of children with visual impairments have some usable vision[6]. I found during my studies that if children have vision they prefer to use it. This means that when designing an

application for children with visual impairments it is important to incorporate visual information as well as audio information. To fully accommodate children with low-vision, I designed both Blocks4All and the BraillePlay games with high contrast colors and large, clear shapes as well as with audio cues.

Finally, as many children with visual impairments have additional disabilities [6], it is important to design technologies that convey information in such a way that it is accessible for them. I tried to keep this in mind when designing both the BraillePlay games and the Blocks4All environment. As noted in Chapter 4, the VBraille interface used in the BraillePlay games is accessible to a child who is deaf-blind. Additionally, the dots can be raised or lowered using a keyboard as well as with gestures to support children with motor impairments. Additionally, the Blocks4All environment works with Switch Access on the iPad, so it is accessible for children with motor impairments who use switches.

7.1.1.4 Provide Alternatives to Gestures

In my evaluations of BraillePlay and Blocks4All and in my interviews with TVIs, I found that many children had a challenging time using gestures on a touchscreen. In the BraillePlay games, some children had difficulty with the two-finger swipe to enter letters. In Blocks4All, even children with some vision had difficulty with the drag and drop gesture, and none of the children were able to successfully use the gesture with VoiceOver on. Because of this, the final design of Blocks4All used select, select drop instead. Some children also had difficulty with some of the more basic gestures. One child in the formative studies had difficulty with the timing of double tap (so I ended up using a Bluetooth keyboard), and a teacher in my interviews in Chapter 3 noted that one of her students had similar trouble with double tap and accessing the rotor. Notably, only one child in the Blocks4All study was familiar with the rotor gesture. As Apple's version of select,

select drop relies on the rotor, this might indicate it should not be relied upon in applications developed for children.

The finding that children often have difficulty with gestures is not new, as previous studies have found it to be true with sighted children [29], but it is especially important to keep in mind with this population. In designing for blind adults, it can be very useful to rely on gestures as an input mechanism as sight is not required to make a gesture [60,73]. However, I have found that in designing for blind children, gestures should be used sparingly, and a keyboard work-around should be available to replace any gesture.

7.2 LIMITATIONS

There are a number of limitations in the studies on the design and evaluation of Blocks4All and the BraillePlay games. In both studies, as well as the interviews described in Chapter 3, there was a small sample size and a wide variation in the abilities of children, some of whom had additional disabilities. These limitations are reflective of both the small population of academic Braille readers in primary and secondary schools (approximately 6,000 in the United States), and the diversity of the population of children with visual impairments [6]. Additionally, in all of my studies, most of the subjects were recruited online through email lists and social media posts. This means that the children are more likely to come from families that are fluent in technology.

In both of my studies, I did not measure learning outcomes even though I was evaluating learning technologies. This was partially due to the small and heterogenous set of participants. I also did not do any comparisons to existing interfaces, as I could not find suitable ones for either learning Braille on the smartphone or learning block-based programming. Instead I focused on how measuring how usable the interfaces were. Many of my quantitative measures were self-

reported responses by the children, and though I emphasized that I wanted honest feedback, it is likely that the children responded positively to please the researcher [33,132].

7.3 FUTURE WORK

In my dissertation research, I designed and evaluated novel touchscreen-based learning technologies for children with visual impairments. In doing this research, I found many opportunities for future research, and I highlight a few areas below.

7.3.1 *How Children with Visual Impairments Use Technology*

In outlining my interviews with TVIs and descriptions of how children in my user studies interacted with my prototypes, I have presented preliminary work in getting an understanding of how children with visual impairments use technology. However, considering the last survey into how these children are using technology was concluded in 2008 [157], this is an area that needs updated study. I am in the process of distributing a survey to parents to get a better idea of how children are using technology in the home. I also plan to interview children to see how they interact with existing technology.

7.3.2 *Designing for Collaboration*

I found in the evaluations for both the Blocks4All environment and the BraillePlay games that sighted siblings wanted to collaborate and play as well as the participants of the study. Although some researchers have begun to design interfaces that can be used by collaboratively by children both with and without visual impairments [129]; oftentimes, interfaces for children with visual impairments are designed as separate interfaces (e.g. Accessible Blockly). In the future, more

research is needed on designing interactions to ensure that children with visual impairments are able to access the same technologies as their sighted peers.

7.3.3 *The Value of Spatial Representation*

Both Blocks4All and the BraillePlay games presented information spatially with the touchscreen. This seemed to help with understanding, and the children indicated in post-session interviews that they understood the spatial information presented on the touchscreen. As tactile tools are so important for this population, it would be interesting to directly measure how important this spatial representation is for understanding when you do not have vision. While Giudice *et al*. [41] found that adults with visual impairments could understand certain types of information such as bar graphs and shapes on a vibro-audio tablet interface, it would be interesting to see if this translates to more complex hierarchical information. This could be explored in terms of the work presented in this thesis: as an explicit comparison in program understanding between a Blocks4All interface that is navigated spatially via touch and one that is navigated "pseudo-spatially" via a keyboard. However, I believe this could also be explored more broadly outside of block-based environments and would be useful to inform future work in designing touchscreen applications for children with visual impairments.

7.3.4 *More Complex Accessible Block-Based Programming*

While the work on Blocks4All is a start at making block-based environments accessible via a screen reader, there are still a number of open questions that must be answered in order to make all of the features of existing block-based environments accessible. One such question is how to accommodate multiple "threads" of program code, as the Blocks4All environment only allows for a single thread along the bottom of the screen. Another is how to navigate more complicated

hierarchies of nested code (for example conditionals that take multiple Boolean statements or operators). There are also open questions about improving the usability of the interface for people with visual impairments: how we can best integrate other means of input such as speech-based commands or gestures? Finally, there is a lot of room to explore different modes of non-visual output for these programs: are there ways to incorporate the sensors, audio and haptic output of these smart devices to make programming challenges that are fun with or without vision?

7.4 Concluding Remarks

In this dissertation, I describe my work to create touchscreen-based learning technologies for children with visual impairments. Through my research, I found that mobile devices with touchscreens are a powerful tool to create these technologies because they allow one to convey information through multiple modalities: spatial information can be conveyed via touch and audio, and visual, haptic and audio feedback can be used to provide more information. Additionally, I found using mainstream devices reduced stigma that might be associated with using traditional assistive technology and made it easier for children with visual impairments to collaborate with sighted peers and siblings.

However, I found that there is a need to improve existing learning technologies on these devices, as many of them are not accessible. The two pieces of learning technology that I created are only small part of what is needed. Ideally, the design outlines I provide and the interactions I explored can be used to make existing literacy and programming applications more accessible, so that blind children can have access to some of the rich curriculum that has been created around literacy and programming. In particular I found that when designing for children with visual impairments, designers need to provide alternatives to gestures. As most prior work on touchscreen accessibility for visual impairments has been done with adults and screen readers can rely heavily

on gestures because they can be used without sight, this is an important novel consideration. Additionally, I found that it is important to convey information in multiple ways, including visually, as most children with visual impairments have some sight. Many of the existing accessible learning technologies for children with visual impairments have bare-bones visual interfaces making them less appealing and potentially less usable for many children with visual impairments. I have found that there is a great need for future work in this area: both exploring how children with visual impairments are using technology and in creating and improving learning technologies, such as block-based programming environments.

BIBLIOGRAPHY

1. Efthimia Aivaloglou and Felienne Hermans. 2016. How Kids Code and How We Know: An Exploratory Study on the Scratch Repository. In *Proceedings of the 2016 ACM Conference on International Computing Education Research* (ICER '16), 53–61. https://doi.org/10.1145/2960310.2960325
2. Khaled Albusays and Stephanie Ludi. 2016. Eliciting Programming Challenges Faced by Developers with Visual Impairments: Exploratory Study. In *Proceedings of the 9th International Workshop on Cooperative and Human Aspects of Software Engineering* (CHASE '16), 82–85. https://doi.org/10.1145/2897586.2897616
3. Khaled Albusays, Stephanie Ludi, and Matt Huenerfauth. 2017. Interviews and Observation of Blind Software Developers at Work to Understand Code Navigation Challenges. In *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility* (ASSETS '17), 91–100. https://doi.org/10.1145/3132525.3132550
4. Zakaria Al-Qudah, Iyad Abu Doush, Faisal Alkhateeb, Eslam Al Maghayreh, and Osama Al-Khaleel. 2014. Utilizing Mobile Devices' Tactile Feedback for Presenting Braille Characters: An Optimized Approach for Fast Reading and Long Battery Life. *Interacting with Computers* 26, 1: 63–74. https://doi.org/10.1093/iwc/iwt017
5. American Foundation for the Blind. Refreshable Braille Displays - Browse Results - American Foundation for the Blind. Retrieved February 15, 2018 from http://www.afb.org/prodBrowseCatResults.aspx?CatID=43
6. American Printing House for the Blind. APH — Distribution of Eligible Students Based on the Federal Quota Census of January 6, 2014. Retrieved April 11, 2018 from http://www.aph.org/federal-quota/distribution-2015/
7. Gavin Andresen. 2002. Playing by ear: Creating blind-accessible games. *Gamasutra Article*.
8. Andrew Leibs. Learn Why iPad Is a Great Learning Tool for Visually Impaired Students. *Lifewire*. Retrieved July 18, 2018 from https://www.lifewire.com/ipad-ideal-tool-blind-visually-impaired-198768
9. Lisa Anthony, Quincy Brown, Jaye Nias, Berthel Tate, and Shreya Mohan. 2012. Interaction and Recognition Challenges in Interpreting Children's Touch and Gesture Input on Mobile Devices. In *Proceedings of the 2012 ACM International Conference on Interactive Tabletops and Surfaces* (ITS '12), 225–234. https://doi.org/10.1145/2396636.2396671
10. 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
11. Maria C. C. Araújo, Antônio R. S. Silva, Ticianne G. R. Darin, Everardo L. de Castro, Rossana M. C. Andrade, Ernesto T. de Lima, Jaime Sánchez, José Aires de C. Filho, and Windson Viana. 2016. Design and Usability of a Braille-based Mobile Audiogame Environment. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing* (SAC '16), 232–238. https://doi.org/10.1145/2851613.2851701
12. Dominique Archambault and Damien Olivier. 2005. How to make games for visually impaired children. In *Proceedings of the 2005 ACM SIGCHI International Conference on*

Advances in computer entertainment technology, 450–453.

13. Dominique Archambault, Roland Ossmann, Thomas Gaudy, and Klaus Miesenberger. 2007. Computer games and visually impaired people. *Upgrade* 8, 2: 43–53.
14. Michal Armoni, Orni Meerbaum-Salant, and Mordechai Ben-Ari. 2015. From Scratch to "Real" Programming. *Trans. Comput. Educ.* 14, 4: 25:1–25:15. https://doi.org/10.1145/2677087
15. Matthew T. Atkinson, Sabahattin Gucukoglu, Colin HC Machin, and Adrian E. Lawrence. 2006. Making the mainstream accessible: redefining the game. In *Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames*, 21–28.
16. Shiri Azenkot and Nicole B. Lee. 2013. Exploring the Use of Speech Input by Blind People on Mobile Devices. In *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility* (ASSETS '13), 11:1–11:8. https://doi.org/10.1145/2513383.2513440
17. Shiri Azenkot, Jacob O. Wobbrock, Sanjana Prasain, and Richard E. Ladner. 2012. Input Finger Detection for Nonvisual Touch Screen Text Entry in Perkinput. In *Proceedings of Graphics Interface 2012* (GI '12), 121–129. Retrieved January 10, 2017 from http://dl.acm.org/citation.cfm?id=2305276.2305297
18. Catherine M. Baker, Lauren R. Milne, and Richard E. Ladner. 2015. StructJumper: A Tool to Help Blind Programmers Navigate and Understand the Structure of Code. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (CHI '15), 3043–3052. https://doi.org/10.1145/2702123.2702589
19. Catherine Marie Baker. 2017. Understanding and Improving Blind Students' Access to Visual Information in Computer Science Education. Retrieved April 9, 2018 from https://digital.lib.washington.edu:443/researchworks/handle/1773/40540
20. Thomas Ball, Sebastian Burckhardt, Jonathan de Halleux, Micha\l Moskal, Jonathan Protzenko, and Nikolai Tillmann. 2015. Beyond Open Source: The TouchDevelop Cloud-based Integrated Development Environment. In *Proceedings of the Second ACM International Conference on Mobile Software Engineering and Systems* (MOBILESoft '15), 83–93. Retrieved February 13, 2017 from http://dl.acm.org/citation.cfm?id=2825041.2825057
21. Wolmet Barendregt and Mathilde M. Bekker. 2011. Children May Expect Drag-and-drop Instead of Point-and-click. In *CHI '11 Extended Abstracts on Human Factors in Computing Systems* (CHI EA '11), 1297–1302. https://doi.org/10.1145/1979742.1979764
22. 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
23. BBC. *Micro:bit*. Retrieved from http://microbit.org/
24. Robert J. Beadles Jr. 2007. *Identification of Statewide Services and Best Educational Practices for Washington State Students Who are Blind, Visually Impaired, and with Multiple Disabilities: A Study of the Washington State School for the Blind and Related Statewide Services*. VI RehaB Consulting.
25. Cynthia L. Bennett, Jane E, Martez E. Mott, Edward Cutrell, and Meredith Ringel Morris. 2018. How Teens with Visual Impairments Take, Edit, and Share Photos on Social Media. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (CHI '18), 76:1–76:12. https://doi.org/10.1145/3173574.3173650
26. John Brooke. 1996. SUS-A quick and dirty usability scale. *Usability evaluation in*

industry 189, 194: 4–7.

27. Aurélie Buaud, Harry Svensson, Dominique Archambault, and Dominique Burger. 2002. Multimedia Games for Visually Impaired Children. In *Proceedings of the 8th International Conference on Computers Helping People with Special Needs* (ICCHP '02), 173–180. Retrieved January 10, 2017 from http://dl.acm.org/citation.cfm?id=646269.758896
28. Brian M. Celusnak. Teaching the iPhone with VoiceOver Accessibility to People with Visual Impairments. Retrieved July 18, 2018 from https://files.eric.ed.gov/fulltext/EJ1114771.pdf
29. Cynthia Chiong and Carly Shuler. 2010. Learning: Is there an app for that. In *Investigations of young children's usage and learning with mobile devices and apps. New York: The Joan Ganz Cooney Center at Sesame Workshop*, 13–20.
30. Michael D. Crossland, Rui S. Silvia, and Antonio F. Macedo. 2014. Smartphone, tablet computer and e-reader use by people with vision impairment. *Ophthalmic and Physiological Optics* 34, 5: 552–557. https://doi.org/10.1111/opo.12136
31. Cynthia Chiong, C. Shuler. Learning: Is there an app for that? Investigations of young children's usage and learning with mobile devices and apps. *The Joan Ganz Cooney Center at Sesame Workshop*. New York. Retrieved February 15, 2018 from https://dmlcentral.net/resources/learning-is-there-an-app-for-that-investigations-of-young-children-s-usage-and-learning-with-mobile-devices-and-apps/
32. Wanda Dann, Dennis Cosgrove, Don Slater, Dave Culyba, and Steve Cooper. 2012. Mediated Transfer: Alice 3 to Java. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (SIGCSE '12), 141–146. https://doi.org/10.1145/2157136.2157180
33. Nicola Dell, Vidya Vaidyanathan, Indrani Medhi, Edward Cutrell, and William Thies. 2012. "Yours is Better!": Participant Response Bias in HCI. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '12), 1321–1330. https://doi.org/10.1145/2207676.2208589
34. Bianca Della Líbera and Claudia Jurberg. 2017. Teenagers with visual impairment and new media: A world without barriers. *British Journal of Visual Impairment* 35, 3: 247–256. https://doi.org/10.1177/0264619617711732
35. Véronique Donzeau-gouge. Programming environments based on structured editors: The MENTOR experience. Retrieved February 13, 2017 from http://www.academia.edu/22643369/Programming_environments_based_on_structured_editors_The_MENTOR_experience
36. Michael Eagle. 2009. Level up: a frame work for the design and evaluation of educational games. In *Proceedings of the 4th International Conference on Foundations of Digital Games*, 339–341.
37. Exploring Computer Science. What is ECS? Retrieved from http://www.exploringcs.org/
38. 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
39. Brian Frey, Caleb Southern, and Mario Romero. 2011. Brailletouch: mobile texting for the visually impaired. In *International Conference on Universal Access in Human-*

Computer Interaction, 19–25.

40. Ryan Garlick and Ebru Celikel Cankaya. 2010. Using Alice in CS1: A Quantitative Experiment. In *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education* (ITiCSE '10), 165–168. https://doi.org/10.1145/1822090.1822138
41. 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
42. Google Inc. Get started on Android with TalkBack - Android Accessibility Help. Retrieved September 5, 2017 from https://support.google.com/accessibility/android/answer/6283677?hl=en
43. Google Inc. Blockly. Retrieved from https://developers.google.com/blockly/
44. Google Inc. Accessible Blockly. Retrieved from https://blockly-demo.appspot.com/static/demos/accessible/index.html
45. Google Inc. & Gallup Inc. 2016. *Trends in the State of Computer Science in U.S. K-12 Schools*. Retrieved from http://goo.gl/j291E0
46. Dimitris Grammenos, Anthony Savidis, Yannis Georgalis, and Constantine Stephanidis. 2006. Access invaders: Developing a universally accessible action game. In *International Conference on Computers for Handicapped Persons*, 388–395.
47. Dimitris Grammenos, Anthony Savidis, and Constantine Stephanidis. 2009. Designing Universally Accessible Games. *Comput. Entertain.* 7, 1: 8:1–8:29. https://doi.org/10.1145/1486508.1486516
48. Shuchi Grover, Stephen Cooper, and Roy Pea. 2014. Assessing Computational Learning in K-12. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education* (ITiCSE '14), 57–62. https://doi.org/10.1145/2591708.2591713
49. William Grussenmeyer and Eelke Folmer. 2017. Accessible Touchscreen Technology for People with Visual Impairments: A Survey. *ACM Trans. Access. Comput.* 9, 2: 6:1–6:31. https://doi.org/10.1145/3022701
50. Erik Harpstead, Brad A. Myers, and Vincent Aleven. 2013. In search of learning: facilitating data analysis in educational games. In *Proceedings of the SIGCHI conference on human factors in computing systems*, 79–88.
51. Ted S. Hasselbring and Candyce H. Williams Glaser. 2000. Use of Computer Technology to Help Students with Special Needs. *The Future of Children* 10, 2: 102–122. https://doi.org/10.2307/1602691
52. Todd Hoffman. Can Smartphones Makes Kids Smarter? *Education.com*. Retrieved May 8, 2014 from http://www.education.com/magazine/article/smartphones-kids/
53. Felix Hu, Ariel Zekelman, Michael Horn, and Frances Judd. 2015. Strawbies: Explorations in Tangible Programming. In *Proceedings of the 14th International Conference on Interaction Design and Children* (IDC '15), 410–413. https://doi.org/10.1145/2771839.2771866
54. Chandrika Jayant, Christine Acuario, William Johnson, Janet Hollier, and Richard Ladner. 2010. V-braille: haptic braille perception using a touch-screen and vibration on mobile phones. In *Proceedings of the 12th international ACM SIGACCESS conference on Computers and accessibility*, 295–296.

55. K. Lisa Yang and Hock E. Tan Institute on Employment and Disability, Cornell University. 2017. Disability Statistics from the American Community Survey (ACS). *Ithaca, NY: Cornell University Yang-Tan Institute (YTI)*. Retrieved April 11, 2018 from http://www.disabilitystatistics.org/reports/acs.cfm?statistic=1
56. Shaun K. Kane and Jeffrey P. Bigham. 2014. Tracking @Stemxcomet: Teaching Programming to Blind Students via 3D Printing, Crisis Management, and Twitter. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (SIGCSE '14), 247–252. https://doi.org/10.1145/2538862.2538975
57. 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
58. 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
59. Shaun K. Kane, Meredith Ringel Morris, and Jacob O. Wobbrock. 2013. Touchplates: Low-cost Tactile Overlays for Visually Impaired Touch Screen Users. In *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility* (ASSETS '13), 22:1–22:8. https://doi.org/10.1145/2513383.2513442
60. 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
61. Caitlin Kelleher and Randy Pausch. 2005. Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers. *ACM Comput. Surv.* 37, 2: 83–137. https://doi.org/10.1145/1089733.1089734
62. Stacy M. Kelly. 2009. Use of Assistive Technology by Students with Visual Impairments: Findings from a National Survey. *Journal of Visual Impairment & Blindness* 103, 8: 470–480.
63. Marwa Khan and Sahar Bayoumi. 2015. Multimedia as a Help for Children with Special Learning Needs. In *2015 International Conference on Cloud Computing (ICCC)*, 1–5. https://doi.org/10.1109/CLOUDCOMP.2015.7149647
64. Joy Kim and Jonathan Ricaurte. 2011. TapBeats: Accessible and Mobile Casual Gaming. In *The Proceedings of the 13th International ACM SIGACCESS Conference on Computers and Accessibility* (ASSETS '11), 285–286. https://doi.org/10.1145/2049536.2049609
65. KinderLab Robotics. 2014. KIBO. *KinderLab Robotics*. Retrieved February 24, 2017 from http://kinderlabrobotics.com/kibo/
66. Andrew J. Ko and Brad A. Myers. 2006. Barista: An Implementation Framework for Enabling New Tools, Interaction Techniques and Views in Code Editors. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '06), 387–396. https://doi.org/10.1145/1124772.1124831
67. Michael Kölling, Neil C. C. Brown, and Amjad Altadmri. 2015. Frame-Based Editing:

Easing the Transition from Blocks to Text-Based Programming. 29–38. https://doi.org/10.1145/2818314.2818331

68. 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
69. Marjan Laal and Seyed Mohammad Ghodsi. 2012. Benefits of collaborative learning. *Procedia - Social and Behavioral Sciences* 31: 486–490. https://doi.org/10.1016/j.sbspro.2011.12.091
70. Barbara Leporini and Eleonora Palmucci. 2017. An Inclusive Educational Game Usable via Screen Reader on a Touch-screen. *SIGACCESS Access. Comput.*, 119: 3–9. https://doi.org/10.1145/3167902.3167903
71. 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
72. Colleen M. Lewis. 2010. How Programming Environment Shapes Perception, Learning and Goals: Logo vs. Scratch. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (SIGCSE '10), 346–350. https://doi.org/10.1145/1734263.1734383
73. Mingzhe Li, Mingming Fan, and Khai N. Truong. 2017. BrailleSketch: A Gesture-based Text Input Method for People with Visual Impairments. In *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility* (ASSETS '17), 12–21. https://doi.org/10.1145/3132525.3132528
74. Conor Linehan, Ben Kirman, Shaun Lawson, and Gail Chan. 2011. Practical, appropriate, empirically-validated guidelines for designing educational games. In *Proceedings of the SIGCHI conference on human factors in computing systems*, 1979–1988.
75. 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
76. 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
77. John H. Maloney, Kylie Peppler, Yasmin Kafai, Mitchel Resnick, and Natalie Rusk. 2008. Programming by Choice: Urban Youth Learning Programming with Scratch. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education* (SIGCSE '08), 367–371. https://doi.org/10.1145/1352135.1352260
78. 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
79. Marc Prensky. 2005. Computer games and learning: Digital game-based learning. *Handbook of computer game studies* 18: 97–122.
80. Sergio Mascetti, Cristian Bernareggi, and Matteo Belotti. 2011. TypeInBraille: A Braille-based Typing Application for Touchscreen Devices. In *The Proceedings of the 13th International ACM SIGACCESS Conference on Computers and Accessibility* (ASSETS

’11), 295–296. https://doi.org/10.1145/2049536.2049614

81. 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
82. Joanne McElligott and Lieselotte van Leeuwen. 2004. Designing Sound Tools and Toys for Blind and Visually Impaired Children. In *Proceedings of the 2004 Conference on Interaction Design and Children: Building a Community* (IDC ’04), 65–72. https://doi.org/10.1145/1017833.1017842
83. Lorna McKnight and Daniel Fitton. 2010. Touch-screen Technology for Children: Giving the Right Instructions and Getting the Right Responses. In *Proceedings of the 9th International Conference on Interaction Design and Children* (IDC ’10), 238–241. https://doi.org/10.1145/1810543.1810580
84. Craig Allan Meador. 2015. Meeting the needs of visually impaired students in Washington state: an exploratory study of the working conditions that affect teachers of the visually impaired. Washington State University. Retrieved July 17, 2018 from https://research.libraries.wsu.edu:8443/xmlui/bitstream/handle/2376/5467/Meador_wsu_0251E_11319.pdf;sequence=1
85. Sean Mealin and Emerson 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
86. Daniel Miller, Aaron Parecki, and Sarah A. Douglas. 2007. Finger Dance: A Sound Game for Blind People. In *Proceedings of the 9th International ACM SIGACCESS Conference on Computers and Accessibility* (Assets ’07), 253–254. https://doi.org/10.1145/1296843.1296898
87. Lauren R. Milne. 2017. Blocks4All: Making Block Programming Languages Accessible for Blind Children. *SIGACCESS Access. Comput.*, 117: 26–29. https://doi.org/10.1145/3051519.3051525
88. Lauren R. Milne, Cynthia L. Bennett, and Richard E. Ladner. 2013. VBGhost: A Braille-based Educational Smartphone Game for Children. In *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility* (ASSETS ’13), 75:1–75:2. https://doi.org/10.1145/2513383.2513396
89. Lauren R. Milne, Cynthia L. Bennett, and Richard E. Ladner. 2014. The Accessibility of Mobile Health Sensors for Blind Users. *29th Annual International & Persons with Disabilities Conference*.
90. 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
91. Brad A. Myers. 1990. Taxonomies of visual programming and program visualization. *Journal of Visual Languages & Computing* 1, 1: 97–123. https://doi.org/10.1016/S1045-926X(05)80036-9
92. Natalia G. Monjelat, Marisa A. Cenacchi and Patricia S. San Martín. Programming for All? Tools and Accessibility: A Case Study. *Revista Latinoamericana de Inclusión Educativa* 12, 1: 213–227.
93. National Federation of the Blind Jerrigan Institute. 2009. *The Braille Literacy Crisis in*

America: Facing the Truth, Reversing the Trend, Empowering the Blind. Retrieved from https://www.nfb.org

94. National Research Council. 2000. *How people learn: Brain, mind, experience, and school: Expanded edition*. National Academies Press.
95. National Science Foundation. 2017. *Women, Minorities, and Persons with Disabilities in Science and Engineering*. Retrieved February 24, 2017 from https://www.nsf.gov/statistics/2017/nsf17310/digest/introduction/
96. Michael A. Nees and Lauren F. Berry. 2013. Audio assistive technology and accommodations for students with visual impairments: Potentials and problems for delivering curricula and educational assessments. *Performance Enhancement & Health* 2, 3: 101–109. https://doi.org/10.1016/j.peh.2013.08.016
97. João Oliveira, Tiago Guerreiro, Hugo Nicolau, Joaquim Jorge, and Daniel Gonçalves. 2011. BrailleType: unleashing braille over touch screen mobile phones. In *IFIP Conference on Human-Computer Interaction*, 100–107.
98. Myrna R. Olson and Sally S. Mangold. 1981. *Guidelines and games for teaching efficient braille reading*. American Foundation for the Blind.
99. Roland Ossmann and Klaus Miesenberger. 2006. Guidelines for the Development of Accessible Computer Games. In *Proceedings of the 10th International Conference on Computers Helping People with Special Needs* (ICCHP'06), 403–406. https://doi.org/10.1007/11788713_60
100. Seymour Papert. 1980. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, Inc., New York, NY, USA.
101. Pew Research Center. 2015. *Teens, Social Media & Technology Overview 2015: Smartphones facilitate shifts in communication landscape for teens*. Retrieved April 13, 2018 from http://assets.pewresearch.org/wp-content/uploads/sites/14/2015/04/PI_TeensandTech_Update2015_0409151.pdf
102. Betsy Phillips and Hongxin Zhao. 1993. Predictors of assistive technology abandonment. *Assistive technology: the official journal of RESNA* 5, 1: 36–45. https://doi.org/10.1080/10400435.1993.10132205
103. Thomas W. Price and Tiffany Barnes. 2015. Comparing Textual and Block Interfaces in a Novice Programming Environment. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research* (ICER '15), 91–99. https://doi.org/10.1145/2787622.2787712
104. Thomas W. Price, Neil C.C. Brown, Dragan Lipovac, Tiffany Barnes, and Michael Kölling. 2016. Evaluation of a Frame-based Programming Editor. In *Proceedings of the 2016 ACM Conference on International Computing Education Research* (ICER '16), 33–42. https://doi.org/10.1145/2960310.2960319
105. Primo Toys. Cubetto: Screenless coding toy for girls and boys aged 3-6. Retrieved from https://www.primotoys.com/
106. Felix Raab, Christian Wolff, and Florian Echtler. 2013. RefactorPad: Editing Source Code on Touchscreens. In *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (EICS '13), 223–228. https://doi.org/10.1145/2494603.2480317
107. 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
108. Rhonda Robinson, Michael Molenda, and Landra Rezabek. 2016. Facilitating Learning. *Association for Educational Communications and Technology*.
109. Antonio Rodríguez, Imma Boada, and Mateu Sbert. 2017. An Arduino-based device for visually impaired people to play videogames. *Multimedia Tools and Applications*: 1–23. https://doi.org/10.1007/s11042-017-5415-1
110. Mario Romero, Brian Frey, Caleb Southern, and Gregory D. Abowd. 2011. BrailleTouch: Designing a Mobile Eyes-free Soft Keyboard. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services* (MobileHCI '11), 707–709. https://doi.org/10.1145/2037373.2037491
111. Karen Rust, Meethu Malu, Lisa Anthony, and Leah Findlater. 2014. Understanding Childdefined Gestures and Children's Mental Models for Touchscreen Tabletop Interaction. In *Proceedings of the 2014 Conference on Interaction Design and Children* (IDC '14), 201–204. https://doi.org/10.1145/2593968.2610452
112. 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
113. Jaime Sánchez, Mauricio Sáenz, and Jose Miguel Garrido. 2010. Usability of a multimodal video game to improve navigation skills for blind children. *ACM Transactions on Accessible Computing (TACCESS)* 3, 2: 7.
114. Kristen Shinohara and Jacob O. Wobbrock. 2011. In the Shadow of Misperception: Assistive Technology Use and Social Interactions. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI '11)* (CHI '11), 705–714. https://doi.org/10.1145/1978942.1979044
115. Robert M. Siegfried. 2006. Visual Programming and the Blind: The Challenge and the Opportunity. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education* (SIGCSE '06), 275–278. https://doi.org/10.1145/1121341.1121427
116. Megan Smith. 2016. Computer Science For All. *whitehouse.gov*. Retrieved February 16, 2017 from https://obamawhitehouse.archives.gov/blog/2016/01/30/computer-science-all
117. Kiley Sobel, Kyle Rector, Susan Evans, and Julie A. Kientz. 2016. Incloodle: Evaluating an Interactive Application for Young Children with Mixed Abilities. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (CHI '16), 165–176. https://doi.org/10.1145/2858036.2858114
118. Sylvia Söderström and Borgunn Ytterhus. 2010. The use and non-use of assistive technologies from the world of information and communication technology by visually impaired young people: a walk on the tightrope of peer inclusion. *Disability & Society* 25, 3: 303–315. https://doi.org/10.1080/09687591003701215
119. Rucha Somani, Jiahang Xin, Bijay Bhaskar Deo, and Yun Huang. 2014. Building Keyboard Accessible Drag and Drop. In *Proceedings of the 16th International ACM SIGACCESS Conference on Computers & Accessibility* (ASSETS '14), 289–290. https://doi.org/10.1145/2661334.2661342
120. Donggil Song, Arafeh Karimi, and Paul Kim. 2011. Toward designing mobile games for visually challenged children. In *e-Education, Entertainment and e-Management (ICEEE), 2011 International Conference on*, 234–238.
121. 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

122. 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
123. Anselm Strauss and Juliet M. Corbin. 1990. *Basics of qualitative research: Grounded theory procedures and techniques.* Sage Publications, Inc, Thousand Oaks, CA, US.
124. Amanda Sullivan, Mollie Elkin, and Marina Umaschi Bers. 2015. KIBO Robot Demo: Engaging Young Children in Programming and Engineering. In *Proceedings of the 14th International Conference on Interaction Design and Children* (IDC '15), 418–421. https://doi.org/10.1145/2771839.2771868
125. 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
126. Tim Teitelbaum and Thomas Reps. 1981. The Cornell Program Synthesizer: A Syntax-directed Programming Environment. *Commun. ACM* 24, 9: 563–573. https://doi.org/10.1145/358746.358755
127. The College Board. 2018. Adopt Ready-to-Use Curricula | AP Central – The College Board. Retrieved July 20, 2018 from https://apcentral.collegeboard.org/courses/ap-computer-science-principles/classroom-resources/curricula-pedagogical-support
128. 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
129. Alfred Thompson. Programming With Blocks. Retrieved February 25, 2017 from http://blog.acthompson.net/2012/12/programming-with-blocks.html
130. Nikolai Tillmann, Michal Moskal, Jonathan de Halleux, and Manuel Fahndrich. 2011. TouchDevelop: Programming Cloud-connected Mobile Devices via Touchscreen. In *Proceedings of the 10th SIGPLAN Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software* (Onward! 2011), 49–60. https://doi.org/10.1145/2048237.2048245
131. Shari Trewin, Diogo Marques, and Tiago Guerreiro. 2015. Usage of Subjective Scales in Accessibility Research. In *Proceedings of the 17th International ACM SIGACCESS Conference on Computers & Accessibility* (ASSETS '15), 59–67. https://doi.org/10.1145/2700648.2809867
132. UC: Berkley. *Snap! Build your own blocks*. Retrieved from https://snap.berkeley.edu/
133. United States. 2004. *Individuals with Disabilities Education Improvement Act*.
134. Luis Valente, Clarisse Sieckenius de Souza, and Bruno Feijó. 2008. An exploratory study on non-visual mobile phone interfaces for games. In *Proceedings of the VIII Brazilian Symposium on Human Factors in Computing Systems*, 31–39.
135. Amber Wagner, Ramaraju Rudraraju, Srinivasa Datla, Avishek Banerjee, Mandar Sudame, and Jeff Gray. 2012. Programming by Voice: A Hands-free Approach for Motorically Challenged Children. In *CHI '12 Extended Abstracts on Human Factors in*

Computing Systems (CHI EA '12), 2087–2092. https://doi.org/10.1145/2212776.2223757
136. 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
137. Uri Wilensky and Seymour Papert. 2010. Restructurations: Reformulations of knowledge disciplines through new representational forms. *Constructionism*.
138. Fredrik Winberg and Sten Olof Hellstrom. 2003. Designing Accessible Auditory Drag and Drop. In *Proceedings of the 2003 Conference on Universal Usability* (CUU '03), 152–153. https://doi.org/10.1145/957205.957235
139. Jeannette M. Wing. 2006. Computational Thinking. *Commun. ACM* 49, 3: 33–35. https://doi.org/10.1145/1118178.1118215
140. Pinata Winoto and Tiffany Y. Tang. 2015. Sensory Substitution to Enable the Visually Impaired to Play an Affordable Wearable Mobile Game. In *Adjunct Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers* (UbiComp/ISWC'15 Adjunct), 193–196. https://doi.org/10.1145/2800835.2800915
141. Wonder Workshop. *Blockly iOS Application for Dot and Dash Robots*. Retrieved from https://www.makewonder.com/apps/blockly
142. Bei Yuan and Eelke Folmer. 2008. Blind Hero: Enabling Guitar Hero for the Visually Impaired. In *Proceedings of the 10th International ACM SIGACCESS Conference on Computers and Accessibility* (Assets '08), 169–176. https://doi.org/10.1145/1414471.1414503
143. JAWS®Pro Screen Reader. *Freedom Scientific eStore*. Retrieved April 25, 2018 from https://store.freedomscientific.com/products/jaws-pro-screenreader
144. iOS VoiceOver Gesture, Keyboard & Braille Shortcuts | AxS Lab. Retrieved September 16, 2017 from http://axslab.com/articles/ios-voiceover-gestures-and-keyboard-commands.php
145. NBP - All About Braille: The Need for Braille. Retrieved May 5, 2018 from https://www.nbp.org/ic/nbp/braille/needforbraille.html
146. WebAIM: Screen Reader User Survey #7 Results. Retrieved May 8, 2018 from https://webaim.org/projects/screenreadersurvey7/
147. Dev.Opera — Accessible Drag and Drop Using WAI-ARIA. Retrieved May 9, 2018 from https://dev.opera.com/articles/accessible-drag-and-drop/
148. WHO | Vision impairment and blindness. *WHO*. Retrieved April 11, 2018 from http://www.who.int/mediacentre/factsheets/fs282/en/
149. Humanware - BrailleNote Touch 32 braille notetaker / tablet - Blindness - Low Vision Aids for Macular Degeneration. Retrieved April 15, 2018 from https://store.humanware.com/hus/braillenote-touch-32.html
150. Use Magnifier to see items on the screen - Windows Help. Retrieved May 9, 2018 from https://support.microsoft.com/en-us/help/11542/windows-use-magnifier
151. macOS Sierra: Zoom pane of Accessibility System Preferences. Retrieved May 9, 2018 from https://support.apple.com/kb/PH25741?locale=en_US
152. ZoomText. Retrieved May 9, 2018 from https://www.zoomtext.com/
153. WebAIM: Screen Reader User Survey #6 Results. Retrieved February 16, 2017 from http://webaim.org/projects/screenreadersurvey6/#mobile
154. The Common Sense Census: Media Use by Kids Age Zero to Eight 2017 | Common

Sense Media. Retrieved February 17, 2018 from https://www.commonsensemedia.org/research/the-common-sense-census-media-use-by-kids-age-zero-to-eight-2017

155. *Common Sense Census: Media Use by Tweens and Teens*. Retrieved April 13, 2018 from https://www.commonsensemedia.org/sites/default/files/uploads/research/census_researchreport.pdf
156. Special Education Elementary Longitudinal Study (SEELS) | SRI International. Retrieved April 12, 2018 from https://www.sri.com/work/projects/special-education-elementary-longitudinal-study-seels
157. National Center for Special Education Research (NCSER) Projects and Programs: PEELS. Retrieved April 12, 2018 from https://ies.ed.gov/ncser/projects/peels/
158. SEELS Info & Reports: Behind the Label. Retrieved May 9, 2018 from https://www.seels.net/info_reports/visual_impairment.htm
159. Code.org: Anybody can Learn. *Code.org*. Retrieved June 16, 2017 from https://code.org/
160. About CS for All Teachers | CS for All Teachers. Retrieved February 16, 2017 from https://csforallteachers.org/about
161. Osmo | Play Beyond The Screen. Retrieved February 24, 2017 from https://www.playosmo.com/en/
162. LabVIEW System Design Software - National Instruments. Retrieved February 24, 2017 from http://www.ni.com/labview/
163. Humanware - Home - Low Vision Aids for Macular Degeneration. Retrieved July 13, 2018 from http://humanware.com/en-usa/products/blindness/braille_displays
164. ChromeVox. Retrieved July 12, 2018 from http://www.chromevox.com/
165. NV Access. *NV Access*. Retrieved July 12, 2018 from https://www.nvaccess.org/
166. Use Switch Control to navigate your iPhone, iPad, or iPod touch. *Apple Support*. Retrieved July 14, 2018 from https://support.apple.com/en-us/ht201370
167. Learning: Is there an app for that? Investigations of young children's usage and learning with mobile devices and apps | DMLcentral. Retrieved February 17, 2017 from http://dml2011.dmlhub.net/node/4496.html
168. Hour of Code. *Code.org*. Retrieved September 3, 2017 from https://hourofcode.com/learn
169. 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/
170. Alternatives to Scratch - Scratch Wiki. Retrieved February 25, 2017 from https://wiki.scratch.mit.edu/wiki/Alternatives_to_Scratch
171. MIT App Inventor | Explore MIT App Inventor. Retrieved April 15, 2018 from http://appinventor.mit.edu/explore/
172. Actimator! - Actimator. Retrieved April 15, 2018 from https://www.actimator.com/
173. Stencyl: Make iPhone, iPad, Android & Flash Games without code. Retrieved April 15, 2018 from http://www.stencyl.com/
174. Coding for Kids. *Tynker.com*. Retrieved September 5, 2017 from https://www.tynker.com/
175. Pencil Code. Retrieved September 5, 2017 from https://pencilcode.net/
176. Browse | Gamefroot. Retrieved April 12, 2018 from https://gamefroot.com/browse-games/
177. Waterbear: Welcome. Retrieved April 12, 2018 from http://waterbearlang.com/

178. Spherly: Programmatic Sphero Control. Retrieved April 12, 2018 from http://outreach.cs.ua.edu/spherly/
179. Hopscotch - Learn to Code Through Creative Play. Retrieved September 5, 2017 from https://www.gethopscotch.com/
180. Daisy the Dinosaur on the App Store. Retrieved April 12, 2018 from https://itunes.apple.com/us/app/daisy-the-dinosaur/id490514278?mt=8
181. Pocket Code Website. Retrieved April 12, 2018 from https://share.catrob.at/pocketcode/
182. Coding for Kids | Computer Programming | AgentSheets. Retrieved April 12, 2018 from http://www.agentsheets.com/
183. Alice – Tell Stories. Build Games. Learn to Program. Retrieved April 12, 2018 from http://www.alice.org/
184. Looking Glass. Retrieved April 12, 2018 from https://lookingglass.wustl.edu/
185. Kodu | Home. Retrieved April 12, 2018 from https://www.kodugamelab.com/
186. squeakland : home of squeak etoys. Retrieved April 12, 2018 from http://www.squeakland.org/
187. How to Meet WCAG 2.0. Retrieved February 25, 2017 from https://www.w3.org/WAI/WCAG20/quickref/
188. Accessibility Developer Checklist | Android Developers. Retrieved February 25, 2017 from https://developer.android.com/guide/topics/ui/accessibility/checklist.html
189. MeasuringU: Measuring Usability with the System Usability Scale (SUS). Retrieved April 25, 2018 from https://measuringu.com/sus/

APPENDIX A

Interview Script for TVIs on Technology Use by Children with Visual Impairments

Pre-Interview

Thank you for agreeing to talk with us today. You should have received the consent form with information about this study via email. Before we begin, do you have any question?

Is it all right if I record this interview?

START RECORDING

Introduction

- Can you tell me about your role working with visually impaired children?
    - What ages are the children you work with?
    - What types of school are you working in?
    - How often do you meet with each of your students?
- When you start working with your students, what experience with technology have they typically already had?
- What mainstream devices do you teach?
    - Desktop/laptop computers
    - Mobile Devices
        - Android/Apple
- What sort of programs do you teach them to use (word processors, email, educational games, etc...)
- What assistive technologies do you teach? How proficient do you feel at each one?
    - Screen readers? Which ones?
    - Magnification (software and hardware: zoom, dynamic text resizing)
        - differences mainstream (ctrl +, vs those under accessibility menu: iOS zoom, Windows magnifier)
    - Peripherals (Braille displays, Braille notetakers, external QWERTY keyboard)
    - Color contrast
    - Switch access
- What kind of problems do you encounter with the AT? How well integrated are they with the rest of the technology and websites that your students use in school?
    - Are there ever cases where a VI child uses a different technology (either app or hardware) than their non-VI peers in class?
        - If so, how are you or the teacher finding the alternative?
        - How commonly does this occur?

How they use assistive tech

- Can you describe the process for determining what assistive technology a student needs? What are the main factors affecting your decision?
- If a child's needs will change over time (e.g. they have a degenerative condition), how do you determine when to introduce the technologies needed for when their vision worsens?

- What interaction do you have with the parents and what role do you expect them to have in a student's learning of assistive technology?
- When the class is using technology (iPads, computers, etc.) and you are not working with the student at that time, what kind of support is the student able to receive from their instructor?

Barriers

- Are there times when your students choose not to use AT?
    - What factors do you believe influence their decision not to use AT?
    - Have you noticed any differences between students with low-vision vs those who are blind?
- How do your student's peers react to their use of AT?
- Does your student ever collaboratively share their devices with a non-VI peer? How do they manage their needs for AT when collaborating with the peer?
- What do you think is the biggest challenge you encounter in trying to get students to use assistive technology?

Wrap up:

- Is there any other information that would be useful for us to know about children with visual impairments technology use?

Thank you for your time. I am going to stop the recording and then I will collect your information for the gift card.

APPENDIX B

Code Book for Analysis of TVI Interviews on Technology Use by Children with Visual Impairments

Code	Descriptions	Examples
Pref MT/MT+	Preference towards Mainstream technologies	
Pref AT/AT+	Preference towards Assistive Technologies	
Repurpose	Repurposing Mainstream Technologies for AT	Using camera to zoom
Match	Matching technology of peers	District uses Chromebook, so VI students use Chromebook
AT->MT	Make access methods seem more mainstream	Comparing keyboard commands to ctrl+ and saying other students will want to learn, take a picture of the board so they don't have to take notes
exp-	Unmet expectations	Company says product accessible, but users find its not
exp+	Met expectations	if Bruce from WSSB if he says they’re accessible. it is. he doesn’t mislead us.
resist	Resistance to technology	Resistant to type, prefer dictation
motivation	General motivation / fun	No fun games to teach typing skills
restrict	Restrictions in technology	School only teaches apple, not android
Cost	Mentions of options being expensive	Braille notetakers 6,000 dollars, if break can't just buy a new one
Identity+	Pride in identity -	Proud can do something others can't
Identity-	Not accepting or hiding disability	Using AT that won't be noticed, not accepting that blind or that vision will worsen
TechSupport+	Support using tech	
TechSupport-	Lack of support in using tech/not trained	Teachers don't know, so can't help
Familiarity	Preference to what they already know	
Access-	Lack of access	don't have access to devices (e.g. don't have cell phone)
Usability+	Usable	
Usability-	Not Usable	problems, bug with software, not well integrated, challenges in understanding (hard to get context and focus with magnifier)
Support+	TVI spends lots of time with kid (non-tech related support)	
Support-	TVI spends little time with kid (non-tech related support)	

Collaboration+	Able to collaborate with other peers	
Collaboration-	Not able to collaborate	
Strategy	Strategies for learning/using technology	
DreamTech	Technology the TVI wishes was around	
MultipleDis	multiple disabilities	
dominant	The dominant technology	JAWS is most commonly used
parent+	parents support kids learning of tech	
parent-	parents can't/don't support kids learning of tech	
skill+	having the expected skills	
skill-	not having the expected skills or learning bad habits	
ability	academic ability	A kid going to college will focus on tech needs for college
framing	TVI presents the technology in a specific way to motivation kids to use the technology	TVI telling kid using fusion, not mentioning JAWS even though it's the combo of JAWS and ZoomText
layer	extra steps to learn non-tech concept	
exploration	learning technologies on own	

APPENDIX C

BraillePlay Preliminary Survey

This survey is intended to be completed by the parent(s) of a blind child aged 5 to 12 years old who would like to participate in the study of how V-Braille mobile applications can be used by blind children to learn Braille.

As a participant, your child will be asked to use simple Braille-learning applications on a smart phone during a one-month time period. (Smart phones with these applications already installed will be provided to participants for use during the study.) The applications let the child practice reading and writing Braille. They use vibration to represent Braille letters and use text-to-speech technology to provide verbal responses to the child's interactions with the phone's touchscreen. Participants are asked to interact with the applications for at least 30 minutes (combined time) a minimum of four times a week. So, the expected minimum time commitment is two hours per week. While we encourage participants to use the applications as much as possible, ultimately, how often and how long your child uses the applications is completely up to your child.

Non-personal information about how your child uses the applications (e.g., how long (s)he played, what patterns of dots were touched on the screen, the accuracy of your child's attempts to write Braille letters, etc.) will be recorded on the phone's SD card and sent to the project team for analysis. This information is used to determine which aspects of these applications work well for teaching Braille to children and which aspects of the applications need to be improved. Parents of participants are asked to supervise their children when they are using the applications (particularly for younger children). Parents are welcome to try out the applications themselves, although if they do so, they are asked to use "non-logging" versions of the applications that will not record the parent's usage results. (For the success of the project, the recorded information needs to be representative of children's usage only.)

Thank you,
The V-Braille Project Team

Please provide the following information about the Parent(s)

1) Your name: (required)

2) Your address (*optional*):

3) Your phone number:

4) Your email address: (required)

Please provide the following information about the <u>Child</u>

1) Name: (required)

2) Gender:

- Female
- Male

3) Age in Years (*5 - 12*) plus additional Months (*0 - 11*):

Years: (required – response must be one- or two-digit integer)

Months: (required – response must be one- or two-digit integer)

4) What is the extent of your child's Braille knowledge?

- None
- Letters
- Grade 1 Braille (letters, numbers, punctuation)
- Some Contractions
- Grade 2 Braille (part- and whole-word contractions)
- Other:

5) At present, how interested is your child in learning Braille?

- Not interested at all
- Shows a little interest
- Shows moderate interest
- Very interested

6) In what grade is your child at school as of Spring 2011 (*if applicable*)?

7) If in school, is your child... (*check all that apply*)

- ☐ in a blind program with other blind children
- ☐ in a main-stream classroom
- ☐ Other:

8) Since the term "blind" can have different meanings to different people, how would you describe (*in your own words or in medical terms*) your child's vision level / ability to see?

9) At what age did your child begin to experience vision difficulties? (*That is, was your child blind since birth, is this a degenerative condition that began at a certain age, etc.*)

10) If your child has taken standard vision tests, what were the results (*e.g., 20/20, 20/80, 20/200, etc.*)?

11) Does your child have any other disabilities? If so, what are they?

12) Does your child have any siblings? If so, please list their ages and genders.

13) If applicable, please describe any disabilities that any other family members have:

14) What types of technology does your child use and what is his/her level of familiarity with each technology? (*Technology examples: computer, cell phone, Braille display, Braille notetaker, game console, iPod, PDAs, electronic books, screen readers, etc.*)

15) What types of games does your child enjoy playing (currently or in the past)? (*check all that apply*)

- ☐ Board games
- ☐ Thinking/deduction games
- ☐ Computer video games
- ☐ Computer audio games
- ☐ Games on smart phones
- ☐ Console games (such as X-Box, Nintendo, PlayStation)
- ☐ Tactile games / games that involve manipulating physical objects (such as blocks, Legos)
- ☐ Games involving physical activity or movement
- ☐ Other:

APPENDIX D

BraillePlay Study Instructions

VBRAILLE STUDY
University of Washington Winter 2013

TABLE OF CONTENTS

GETTING STARTED: VBRAILLE STUDY

VB stands for Vibrational Braille. We have two applications that test Braille reading and writing with single characters- just like “flashcards” and two slightly more complicated word games. Currently, these are all Grade 1 Braille characters, and we are not including punctuation or numbers. We ask that when the parent (or anyone other than the child) is using the phone, to use the Parent versions of the applications, which are clearly marked. It is important that the child uses the Child version. We want to make sure to know where the data is coming from and not mix up information from the parent and the child when looking at the results.

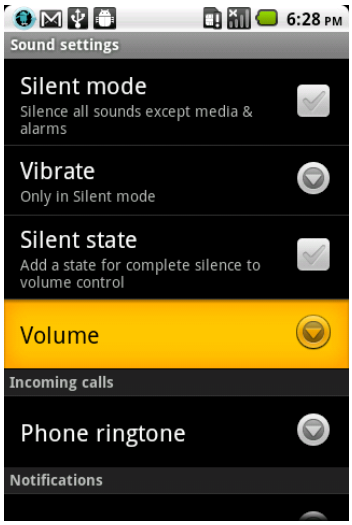
GETTING FAMILIAR WITH THE PHONE:

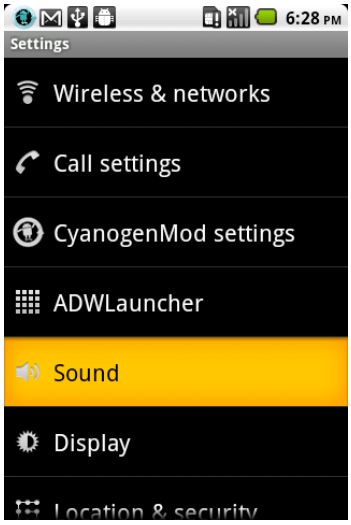
The phone has volume buttons on the top left edge, a camera button on the bottom right edge, and 4 tactile buttons on the bottom of the screen along with a trackball. Hold the phone vertically (in Portrait orientation) with the buttons at the bottom.

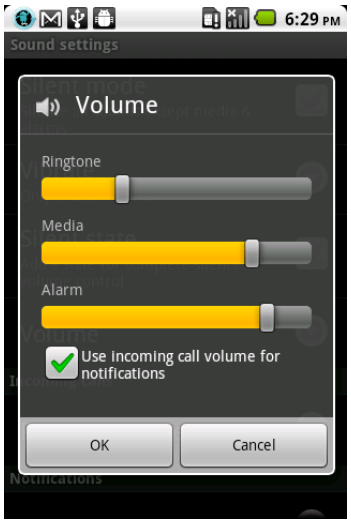
Get familiar with the Android phone and how to access its applications.
On your Android phone, we have installed VBWriter, VBReader and VBHangman. For each of

them, we have a Parent and a Child version. We have created a shortcut to the child versions on the screen to the “left” of the home screen. You can access both the Parent and the Child versions through the applications menu.

These applications use speech synthesis (Text-to-Speech) to speak instructions and screen information to the user. This is built into the applications, so TalkBack does not need to be on to play the game If you do use TalkBack, menu instructions will repeat. When starting the application, check the media volume setting on your phone to make sure it is loud enough to hear. To get to this setting, go to the **Setting**s menu, and then click on **Volume** and then click on **Sound**. Adjust the 'Media' volume accordingly.







CONNECTING TO WI-FI

If possible, we would like to collect usage data over the phone’s wi-fi connection. In order to connect to a Wi-Fi network, go to the **Settings** menu and open the **Wireless & networks** option. Click the **Wi-Fi** button to turn on the Wi-Fi, you may need to open **Wi-Fi settings** and click **Add Wi-Fi network**. If it is a secure network, you’ll need to enter the password.

VBREADER (CHILD READER and PARENT READER)

Now we are ready to see how the applications work. The first application is VBReader- you will see this on the phone as Child Reader and Parent Reader.

WHAT THE APPLICATION DOES:

VBReader allows the user to practice reading Braille letters using vibration to represent raised dots. It is an application suitable for blind, low-vision and sighted users who wish to learn Braille letters.

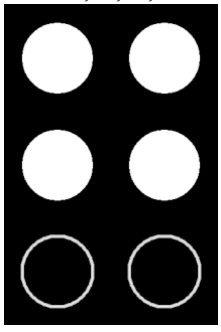
The touchscreen is divided evenly into six regions, each of which contains one of the six dots found in the standard Braille cell. The dots are numbered with dots 1 through 3 down the left side of the screen and dots 4 through 6 down the right side. When a region is touched, its number is spoken.

Braille letters are randomly displayed, one at a time, on the screen using both visual and tactile cues to indicate the presence of the raised dots that compose the Braille symbol for the letter. Visually, the dots that would be raised in a paper-based Braille symbol are shown as large, solid white dots and dots that are not raised are shown as unfilled circular outlines. Tactually, a dot that is raised (filled) will vibrate when touched. (More accurately, touching the region containing a dot triggers the haptic device so that the entire phone vibrates.) Empty (non-raised) dots do not.

To help distinguish one row of dots from another, the vibration frequency between rows of dots is varied -- meaning the feel of the vibration in each row differs from those of the other two rows. To maximize the benefit of these differences, players should begin their touch in the upper left-hand corner of the screen and drag down the column before moving to the second column and repeating the downward dragging motion.

HOW TO USE THE APPLICATION:

1) Start the application by tapping on it in the Applications menu. After the welcome greeting is spoken, the application will choose and display a letter (below you see the representation for the letter 'G'- dots 1, 2, 4, and 5).



2) To hear spoken instructions for the application, swipe down with two fingers (see diagram at the end of the document if you want more detailed instructions on how to do this).
3) Use your finger to navigate to each of the six dots, noting by the presence (or absence) of vibration whether or not a dot is raised. Dot numbers are spoken when touched to help you navigate to each dot. Try to identify which Braille letter is being displayed based on where the vibrating dots are located within the Braille cell.
4) To enter your answer, click on the trackball or swipe right with two fingers (see end of document for more detailed description on how to swipe).
5) You are then presented with an onscreen touch keyboard. As you feel around the screen, you will hear which letter you are on. Letters A through M are on the left side, and letters N to Z are on the right side. See below:

A	N
B	O
C	P
D	Q
E	R
F	S
G	T
H	U
I	V
J	W
K	X
L	Y
M	Z

6) To enter the letter you are guessing, find the letter with your finger (you will hear it say the letter out loud) and then double tap anywhere on the screen to enter that letter as your answer. You will then be told whether or not your guess was correct.
7) The letter will be redisplayed on the screen after you guess. When you are ready for the next letter, do a two-finger right swipe or press down on the trackball. When the new letter is displayed, a "Ready" cue is spoken.
8) To interrupt and stop instructions, single tap anywhere on the screen.
9) To use another input method instead of the two column keypad, press the menu button at any time. The menu will display three options:
    a. Qwerty Keypad: input method is a Qwerty keypad in landscape orientation. Touch and drag finger to hear letter options. Double tap to select last heard letter
    b. Simple Alpha Keypad: This is the default two column keypad described above.

c. Tap Hold Input: Tap to move through the letters in the alphabet one by one. Double tap to select last heard letter.

Press the Back button (not the Home button) to hear how many of the letters you read correctly and then exit the application.

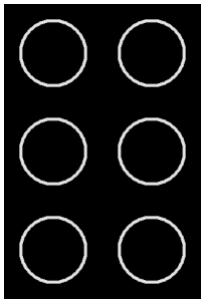
VBWRITER (CHILD WRITER AND PARENT WRITER)

WHAT THE APPLICATION DOES:

VBWriter allows the user to practice creating Braille letters by double-tapping where dots in the Braille letter should be raised. It is an application suitable for blind, low-vision and sighted users who wish to practice writing Braille letters.

The touchscreen is divided evenly into six regions, each of which contains one of the six dots found in the standard Braille cell- the same layout as with the previous application, VBReader. The dots are numbered with dots 1 through 3 down the left side of the screen and dots 4 through 6 down the right side. When a region is touched, its number is spoken.

All dots start out empty -- an empty dot is the same as a non-raised dot in a paper-based Braille cell. Solid (or filled) dots represent raised dots and a dot can be filled in by double-tapping on it. Once a dot is filled, it will vibrate when touched. (More accurately, touching the region containing a dot triggers the haptic device so that the entire phone vibrates.) Filled dots can be made empty by double-tapping on them. Empty dots do not vibrate.



HOW TO USE THE APPLICATION:

1) Start the application by tapping on it in the Applications menu.
2) After the welcome greeting is spoken, the application will tell you which letter to enter. You can press the menu button to repeat the letter.
3) Use your finger to navigate to the dots that should be raised -- dot numbers are spoken when touched to help you navigate to the desired dot. After you hear the number you want, double tap in that same location. The application will respond verbally that the dot has been added and the dot will begin vibrating when touched. (Note: the dot will be

added in the location where the double tap occurs. The regions are fairly large so unless your finger is right on the border area between regions, you should be able to add dots accurately to the dot number last spoken.)

4) If you need to remove a dot, navigate to it and double tap. The application will respond that the dot has been removed and the dot will no longer vibrate when touched.
5) When you are finished entering the dots in the current letter, swipe right with two fingers (or press down on the trackball or center of the directional keypad) to find out if you entered the Braille symbol correctly. If you did, you will hear "that is correct!", and the game will move on to the next letter.
6) If you entered the letter incorrectly, the application will attempt to find a match for the dots you entered and tell you what letter you wrote. It will then vibrate the correct dots on the screen so that by touching them, you can determine which dots should have been raised and lowered. When you are ready to move on to the next letter, swipe right with two fingers (or press down on the trackball or center directional key) again.

VBHANGMAN (CHILD VBHANGMAN AND PARENT BVHANGMAN)

WHAT THE APPLICATION DOES:

VBHangman allows the user to play the word game hangman in V-Braille. The smartphone randomly selects a word of a given length, and the user has to guess what letters are in the word (with only nine chances to incorrectly guess a letter). The player must input her letters in V-Braille.

The game is played with a combination of the VBWriter input screen and accessible menu screens, which you can touch and drag down to hear options and double tap to select the last heard option.

HOW TO USE THE APPLICATION:

1) Start the application by selecting the VBHangman application from your phone’s application menu.
2) Hold the phone vertically (in Portrait orientation) with the buttons at the bottom.
3) You will briefly be presented with a loading screen that has a hangman icon. You will automatically move to the main menu screen
4) On the main menu, some brief instructions for how to use the menu are spoken, and you can choose the length of the word you would like to play. In order to make the selection, touch and drag your finger down the screen to hear the options and then double tap anywhere on the screen to select the last heard option.
5) After selecting the length of word that you would like to use, you will move on to the current game menu. Once again to make selections from this menu, touch and drag your finger down the screen to hear the options and then double tap anywhere on the screen to select the last heard option. The options you can choose from are: **Word**, **Trials Left**, **Guessed Letters**, **Enter Letter** and **Instructions**.
6) If you select **Instructions**, the phone will read some brief instructions about how to navigate through the menu and the enter letter screens.

7) If you select **Word**, the phone will spell out the word with blanks for the letters that you have not guessed yet (initially all letters in the word will be blanks).
8) If you select **Trials Left**, the phone will tell you the number of incorrect guesses you have left before you lose the game.
9) If you select **Enter Letter**, you will be taken to the Enter Letter screen, where you can enter a guess for a letter, swipe down with two fingers for instructions at any time. On the Enter Letter screen:
    a. You will be presented with a screen that represents a blank Braille cell.
    b. Pressing the back or menu buttons will take you back to the current game menu without guessing any letters.
    c. Use your finger to navigate to the dots that should be raised -- dot numbers are spoken when touched to help you navigate to the desired dot. After you hear the number you want, double tap in that same location.
    d. The application will respond verbally that the dot has been added and the dot will begin vibrating when touched. (Note: the dot will be added in the location where the double tap occurs. The regions are fairly large so unless your finger is right on the border area between regions, you should be able to add dots accurately to the dot number last spoken.)
    e. If you need to remove a dot, navigate to it and double tap. Again, the application will respond that the dot has been removed and the dot will no longer vibrate when touched.
    f. Alternatively, you can raise or lower dots using the numbers 1 - 6 on the phone's keypad. You can also add entire letters with the keypad.
    g. When you are finished entering the dots in the current letter, swipe with two fingers to the left or right to find out if you entered a letter that is in the word.
    h. If you did not correctly enter a Braille letter, the phone will tell you and you will once again be given a blank Braille cell.
    i. If the letter is not in the word, you will be taken back to the current game screen, and the phone will tell you the number of incorrect guesses you have left.
    j. If the letter is in the word, the game will read it to you. Swipe right with two fingers if that is what you meant to enter. Double tap to enter a different letter.
    k. If you confirm that you entered the correct word, you will be taken back to the current game screen, and the phone will spell out the word with blanks for the unguessed letters.
10) To exit the application, press the **Back** button repeatedly (through the current game screen, main menu screen and loading dictionary screen), or press the **Home** button (If you exit using the **Home** button, you will be taken back to the current game when you restart the application).

REMINDERS AND TIPS:

To hear spoken instructions for the application, swipe down with two fingers (see below).
To interrupt and stop instructions, single tap anywhere on the screen.
You can use two-finger right swipe in place of the trackball to move to the next letter.

In VBWriter, you can press the Menu button to repeat the name of the letter to enter.

Press the Back button to hear how many letters you got right and then exit the application. In VBHangman, you must hit the back button an extra time to exit the loading screen.

TWO-FINGER DOWNWARD SWIPE: TWO-FINGER RIGHT SWIPE:

This can be difficult to get the hang of. It works best if the fingers are a little bit apart and aligned vertically (as in image on right) as opposed to horizontally (as in image on left).

CONTACTS:

If you are having technical issues or any general questions, please don't hesitate to contact either Cynthia Bennett (206-221-3147) or Lauren Milne (505-220-5996). You can also send questions via email to bennec3@uw.edu or milnel2@cs.washington.edu.

APPENDIX E

BraillePlay Experience Sampling Questions for Parents

Application usage questions

What did you have to help your child with this week when (s)he played with the applications?

Does your child seem to enjoy playing with the applications? Why or why not?

Do you think using these applications are helping your child improve his/her ability to read and write Braille letters?

Describe your child's experience relative to the following features of the applications:

- Recognizing raised dots – was your child able to recognize the phone's vibration as representing a raised dot and keep track of all the dots that (s)he felt vibrating?

- Screen orientation – did your child get confused about the orientation of the Braille letter because (s)he turned the phone different directions in her/his hand?

- Spoken instructions -- was your child able to understand the spoken instructions? If not, how could they be improved?

- Spoken dot numbers -- did your child prefer to have the dot numbers spoken when (s)he touched the different regions of the screen or did (s)he turn the speech off?

- Double-tapping to raise or lower dots -- was your child able to raise or lower the dots easily by double-tapping or did (s)he has problems tapping in the correct region?

- Using buttons – did your child have any problems finding or using the phone buttons that helped control the games (i.e. back button to leave, menu button to repeat the letter to enter, center dpad button to enter answer/move to next letter)? Did your child use the keypad numbers to enter Braille letters instead of double tapping?

Were there other aspects of the games that
- caused your child difficulties?
- were easy for your child?

Did you spend any time using the applications yourself?
If so,
- did you learn Braille by using them?
- what did you like and dislike about them?

Do you have any suggestions for making these applications more appealing and/or usable?

Experience Sampling Questions for Younger Children

Did you have fun playing the games on the phone?

What was your favorite part about them?

Would you like to keep playing them after today?

V-B-Reader Questions

Did you learn to read any new Braille letters by playing the games?

If child answers yes, ask...

- Can you tell me some of the letters you learned?
- Have you tried reading any of those letters on paper? Would you like to [try/show me] now?

V-B-Writer Questions

Was it easy to make new dots start vibrating by double tapping on the screen?

General Questions

Could you understand what the phone was saying when it spoke to you?

Can you show me how you use the games now?

APPENDIX F

BraillePlay Application Data Collection

Log Entry Codes and Qualifiers

Code	Stands for	Description	Qualifier	Description
as	Application Status	Gives run state of the application	Started	Application has been started.
			Paused	Application has been paused.
			Resumed	Application has been resumed.
dc	Dot Change	Indicates that the user added or removed a dot from her input.	+[number]	A dot was added in the region indicated by the number.
			-[number]	A dot was removed from the region indicated by the number.
dn	Dot Numbers	Indicates that the speaking (by the application) of dot numbers as they are touched was turned on or off.	TurnedOff	Speaking of dot numbers was turned off.
			TurnedOn	Speaking of dot numbers was turned on.
kp	Key Press	Indicates that the user pressed a key or button on the phone.	MenuKey, , ,	The menu key was pressed.
			EnterKey	The center dpad key or trackball was pressed.
			BackKey	The back key was pressed.
			[keyCode]	The key represented by the given key code was pressed.
lp	Long Press	Indicates that the user touched and held his finger in one location for at least three seconds.	InstructionsSpoken	The long press triggered the speaking of application instructions.
rt	Region Touched	Indicates which region of the screen was touched.	[number 1 - 6]	The number of the region that was touched.
sc	Screen Contact	Tells when the contact between the user's finger and the screen changed.	FingerDown	The user set her finger down on the screen.
			FingerUp	The user lifted his finger off the screen.

ss	Symbol Selected	Indicates which letter the application randomly selected to display.	[letter]:[dot pattern]	The dot pattern is a six-digit binary string where where 1 represents a dot that vibrates. and digit sequence equates to the six dots of the Braille cell. The letter is the alphabetic letter that the dot pattern represents in Braille.
ua	User Answer	Indicates that the user pressed the enter key to input an answer.	[letter]:[dot_pattern]	Shows the dot pattern that the user entered. If the pattern matches a Braille letter, that letter is shown before the colon.
			?:[dot_pattern]	If the user's dot pattern does not match any Braille letters, a question mark is shown before the colon.
ws	Word Selected	Indicates that the hangman game has selected a new word	[word]	

Example of Log Entry for a session:
(*format*: device id, app abbrev, session id, round number, datetime, event code, event qualifier)

A0000015FD4D20 VBR 110125132106 0 2011-01-25 22:10:38:235 as Started
A0000015FD4D20 VBR 110125132106 1 2011-01-25 22:10:38:418 ss b:101000
A0000015FD4D20 VBR 110125132106 1 2011-01-25 22:10:41:041 sc FingerDown
A0000015FD4D20 VBR 110125132106 1 2011-01-25 22:10:41:167 rt 2
A0000015FD4D20 VBR 110125132106 1 2011-01-25 22:10:41:502 dn TurnedOff
A0000015FD4D20 VBR 110125132106 1 2011-01-25 22:10:41:891 sc FingerUp
A0000015FD4D20 VBR 110125132106 1 2011-01-25 22:10:45:121 sc FingerDown
A0000015FD4D20 VBR 110125132106 1 2011-01-25 22:10:45:507 rt 3
A0000015FD4D20 VBR 110125132106 1 2011-01-25 22:10:48:356 rt 1
A0000015FD4D20 VBR 110125132106 1 2011-01-25 22:10:49:201 rt 2
A0000015FD4D20 VBR 110125132106 1 2011-01-25 22:10:49:593 sc FingerUp
A0000015FD4D20 VBR 110125132106 1 2011-01-25 22:10:52:375 kp BackKey
A0000015FD4D20 VBR 110125132106 1 2011-01-25 22:10:55:688 as Paused

APPENDIX G

BraillePlay Application Parent Exit Survey

The following questions are designed to get your child's and your feedback about both of your experiences using the V-Braille applications on a smart phone. Please discuss these questions with your child, as appropriate, to get his/his input. Feel free to write on the back of the page or attach additional pages if there is not enough room for your answer. You can skip any questions you do not wish to answer. Your feedback will be used to improve these applications.

Thank you,
V-Braille Research Team

Braille questions

How long has your child known Braille?
How did he/she learn Braille?
How long did it take to learn Braille? Can you describe the overall experience?
Does your child use more audio or more Braille?
What devices does your child use on a day-to-day basis?
Are there any problems with current Braille learning systems that you have noticed, which could possibly be improved?

Application usage questions

Did you have to help your child with setting up the applications? Finding buttons? Remembering how to use the applications?
Did your child want to play these applications or was it more of a chore to get him/her to play them? Did (s)he stay interested in playing them for long?
Did your child prefer one application? If so, why?
Do you think using these applications helped your child improve his/her ability to read and write Braille letters?

Describe your child's experience relative to the following features of the applications:

- Recognizing raised dots – was your child able to recognize the phone's vibration as representing a raised dot and keep track of all the dots that (s)he felt vibrating?
- Screen orientation – did your child get confused about the orientation of the Braille letter because (s)he turned the phone different directions in her/his hand?
- Spoken instructions -- was your child able to understand the spoken instructions? If not, how could they be improved?
- Spoken dot numbers -- did your child prefer to have the dot numbers spoken when (s)he touched the different regions of the screen or did (s)he turn the speech off?
- Double-tapping to raise or lower dots -- was your child able to raise or lower the dots easily by double-tapping or did (s)he have problems tapping in the correct region?

- Using buttons – did your child have any problems finding or using the phone buttons that helped control the games (i.e. back button to leave, menu button to repeat the letter to enter, center dpad button to enter answer/move to next letter)? Did your child use the keypad numbers to enter Braille letters instead of double tapping?

Were there other aspects of the games that

- caused your child difficulties?
- were easy for your child?

APPENDIX H

BraillePlay Application Child Exit Interview

Did you enjoy using the V-B-Reader and V-B-Writer applications? What did you like about them?

Did you like one application more than the others? If so, why?

If you had these applications on your mom or dad's phone, would you want to play them even though the study is over?

V-B-Reader Questions

Were you able to read Braille letters on paper before you began playing V-B-Reader?

- If no, did V-B-Reader help you to learn your Braille letters? What letters do you know? Have you tried reading those letters on paper since using V-B-Reader? How did you do? (Have Braille letters on a page in case they haven't tried so they can try it during the interview.)

- If yes, were you also able to read the letters using V-B-Reader? How hard or easy was it to get used to thinking of vibrating dots in V-B-Reader as being the same as raised dots on paper? Do you think using V-B-Reader helped you improve your ability to read Braille?

Do you think this game would be more fun if before it told you the name of the letter, you could select the letter name from a list and then see if you got it right?

V-B-Writer Questions

When you wanted to raise or lower a dot, were you able to double tap in the right place to get the dot you wanted?

Did you try using the keypad to raise and lower dots instead of double tapping? Which way was easier for you?

Did you know you could press the menu button to repeat the name of the letter to write? Did you try this out? How did it go? If child had difficulties, ask: is there a different button you would like to press or motion to make on the screen to repeat letter names?

Did you know how to write Braille letters before you began playing the game? Do you think playing the game helped you get better at writing Braille letters?

General Questions

Dexterity:

- Was it hard to hold the phone and touch the screen at the same time?
- Was it hard to remember which way to hold the phone?
- Was it hard to just use one finger on the screen to feel for vibrations?
- Was it hard to find and push the right buttons?
- Was it hard to swipe with two fingers to enter the letters?

Sequence:

- Did you have any trouble remembering which button to press next?

Speech:

- Could you understand what the phone was saying when it gave you instructions?
- Did you know that you could stop the instructions by touching the screen? Did you know you could repeat the instructions by touching and holding the screen?
- Did you like hearing the dot numbers when you touched the different dots or was it confusing?

Did you have a hard time remembering where the dots were supposed to be? Was it hard to remember which dot numbers were vibrating and which ones weren't?

Can you show me how you use the games now?

Do you think these games would be more fun if they were changed a little bit? What kinds of changes do you think would make you want to play them more often?

APPENDIX I

Tickle Evaluation Study Tasks

TASK SET 1: DRAG AND DROP
Place *When starting to play* at beginning
Add *Move Forward for 2 secs at 50% speed*
Play sound hi after *move forward*
Move *play sound* to right after *when starting play*
Move all blocks to trash (not when starting play)

TASK SET 2: ACCESSING TYPE INFO/MODIFYING REPEAT BLOCKS
Add *if zero than...* block
Add any condition that fits (If they can't find one add *Reflectance in front of Dash* block)
Add *play sound Hi* inside
Add *repeat 10* times
Change repeat condition to 4 times
Add *turn right by 90 degrees*

TASK SET 3: UNDERSTANDING PROGRAM STRUCTURE
Blocks: *when play, repeat 3x, move forward, turn right, end repeat, repeat 2x, end repeat, play lasers*
Which blocks are in the workspace and in what order?
Find the block that makes him turn right
Find block that is inside repeat 3 times
Find repeat condition that is done too few times if you want the robot to go in a square
Find the block that is outside the repeat loop, but shouldn't be

TASK SET 4: UNDERSTANDING PROGRAM STRUCTURE
Blocks*: if 1 = 1, repeat 2x, cat sound, end repeat, horse sound, hi sound, end if*
Which blocks are in the workspace and in what order?
Find the block that is sequentially in the wrong order (hi at end)
Will he say make horse noise more than once? is it inside repeat?
Switch if condition to something else

APPENDIX J

Blocks4All: Parental Preliminary Survey for both Formative and Evaluative Studies

Question 1.
How old is your child (years and months)?

Question 2.
What is your child's gender?

Question 3.
What grade is your child in?

Question 4.
To the best of your knowledge, what is the extent of your child's knowledge of computer science, programming or computational skills?

Question 5.
How interested is your child in learning about computer science?
Select one... Not interested at all Shows a little interest Shows moderate interest Very interested

Question 6.
If in school, is your child... (check all that apply)

At a school for the blind

In a blind program with other blind children

In a mainstream classroom

Other:

Question 7.
Since the term "blind" can have different meanings to different people, how would you describe (in your own words or in medical terms) your child's vision level / ability to see?

Question 8.
At what age did your child begin to experience vision difficulties? (That is, was your child blind since birth, is this a degenerative condition that began at a certain age, etc.)

Question 9.
If your child has taken standard vision tests, what were the results (e.g., 20/20, 20/80, 20/200, etc.)?

Question 10.
Does your child have any other disabilities? If so, what are they?

Question 11.

How familiar is your child with touchscreen devices? Have they used a smartphone (iPhone) or a tablet (iPad) before? If so, how often do they use it? and what for?

Question 12.
Does your child use any other types of technology and what is his/her level of familiarity with each technology? (Technology examples: computer, cell phone, Braille display, Braille notetaker, game console, iPod, electronic books, screen readers, etc.)

APPENDIX K

Blocks4All Formative Study Materials: Tasks, Child and Parent Interview Questions

Session 1: Introduction to Programming and Constructing Simple Programs (Drag and Drop vs Placeholder)

Introduction

- **CONSENT AND ASSENT FORMS**
- **SET UP VIDEO RECORDER AND TIMER**
- Introduce robot, idea of blocks, toolbox (categories), workspace, play button, and practice
- Go over VoiceOver commands

Drag and Drop Interface

- Find the
    - toolbox
        - navigate menu
            - single tap to read categories
            - double tap to select a category
            - single tap to read blocks
            - back button to go to different category
        - select item from menu, drag into workspace
            - tap and hold, wait for sound, move to right
            - down, up, down, hold
    - workspace
        - find item that you dragged into workspace
            - touch, can also swipe left and right
        - move more items into workspace
        - drag over to the right until you hear **trash can**
            - double tap and drag
        - move items around in workspace
    - play button
        - find the play button and double tap to make robot go!
- Putting blocks together to control a robot
    - select from toolbox, place in workspace, press play

Task Set 1

1. Let's get Dash's engine going, press PLAY to run the code! Since the "Start Engine" Block is the first block in the workspace, the engine will run whenever you touch the PLAY button.
2. Now let's build the code ourselves. Get Dash to move forward and knock over the dominos by selecting the "Drive Forward" block from the Drive menu and placing it at after Start Engine. Don't forget to press PLAY to run your program.
3. Now let's make Dash drive forward, turn left and then forward again by selecting the forward and turn left blocks from the Drive menu and adding them in the correct order! Then he will knock over both sets of dominos!
4. Delete the "Turn Left" block so that Dash just drives forward twice instead and knocks over the domino blocks.

5. Free for all! Make Dash do whatever you would like using the Sounds, Drive, Movement and Control Blocks.

Questions

1. Are you having fun? really fun, kind of fun, ehh, kind of not fun, really not fun
2. What was hard about that? Did you find anything difficult about dragging and dropping the blocks to add them to the workspace?
3. What did you find easy about it?
4. Is there anything you would change about this method to make it easier or more fun?

Placeholder Interface

- Start in
    - workspace
        - find the add block button
        - double tap to open
    - toolbox
        - navigate menu
            - single tap to read categories
            - double tap to select a category
            - single tap to read blocks
            - back button to go to different category
        - select item from menu
            - double tap
            - will be placed where you initially selected
    - find item that you dragged into workspace
        - touch, can also swipe left and right
        - notice add button on either side, use to move more items into workspace
        - double tap to move **to trash** or move blocks
    - play button
        - find the play button and double tap to make robot go!
- Putting blocks together to control a robot
    - find place to add in workspace, double tap to add blocks, press play

Task Set 2

1. Let's get Dash talking, press PLAY to run the code! Since the "Say Hi" Block is connected to the "When Start" block, Dash will say Hi whenever you touch the PLAY button.
2. Get Dash to bark like a dog after he says "Hi" by selecting the "Make Dog Noise" block from the Sounds menu and adding it after the "says hi" button. Don't forget to press PLAY to run your program.
3. Now let's make Dash say hi, make a cat noise, and then bark like a dog, by selecting the "Make Cat Noise" blocks and adding them in the correct order! (Middle)
4. Delete the "Bark like a dog noise" so Dash just Says Hi and meows like a cat.
5. Free for all! Make Dash do whatever you would like using the Sounds, Drive and Control Blocks.

Questions

1. Are you having fun?
2. What was hard about that? What did you find difficult about using the add blocks button to add blocks to the workspace?

3. What did you find easy about it?
4. What would you change to make this easier or more fun?

More End of Session Questions

1. What was your favorite part?
2. What was your least favorite part?
3. How did you like the version of the game with dragging and dropping vs the version where you add blocks from the menu?
    a. Why?
4. What would you change?
5. What did you find difficult about programming?
6. What did you find easy about programming today?
7. Are you interested in doing more programming?
8. Have you learned any computer science before (programmed robots, code.org, hour of code)? How much experience do you have with computer science?
9. Have you used a smartphone (iPhone) or a tablet (iPad) before? If so, how often do you use it? and what for?

End of Session Questions for Parent/Teacher:

1. What do you think your/the child had the most difficulty with when using this version?
2. What do you think your/the child found easiest when using this version?
3. How did you like this version of the game compared to previous versions?
    a. Why?
    b. What would you change?
4. Any feedback on how to improve this interface?

Session 2: Showing Complex Structure with Repeat Blocks (Spatial vs Audio Layout)
Introduction

- **SET UP VIDEO RECORDER AND TIMER**
- Review
    - Robot: Dash
    - blocks
    - toolbox (categories)
        - Drive
        - Control
        - Sound
        - Movement
    - workspace
    - play button
- Putting blocks together to control a robot
    - select from toolbox, place in workspace, press play
- VoiceOver
    - Finger to explore the screen, elements will read aloud, pause and more information will be given
    - Double tap to select elements

- ***Repeat blocks***
    - What if you want the robot to say hi over and over again? How would you do it?
    - Add repeat blocks
        - any blocks inside these blocks are repeated as many times as the repeat block says

Review

- Find the
    - toolbox
        - navigate menu
        - select item from menu, drag into workspace
    - workspace
        - find item that you dragged into workspace
        - move more items into workspace
        - drag over to the right until you hear trash can
        - move items around in workspace
    - play button
        - find the play button and double tap to make robot go!
- Putting blocks together to control a robot
    - select from toolbox, place in workspace, press play

Audio Interface

- Repeat: Let's make Dash bark three times
    - Set-up:
        - Repeat 3 times -> Make Dog Noise -> End Repeat
    - Explore workspace
        - Where is the make dog noise block?
        - notice that block is inside a begin and an end repeat block
            - also there is an inside repeat block and you can follow up to find what block is inside (make dog noise)
- Select a repeat block from the control blocks
    - what does it look like in the workspace?
        - two blocks
            - start block, end block
    - what happens when we run code?
        - nothing, telling Dash to repeat nothing
    - Let's add something more
        - add block inside repeat
        - add block outside repeat

Task Set 1

1. Now let's make Dash start his engine, drive forward, turn right and then forward again by selecting those blocks from the Drive menu and adding them in the correct order! Then he will knock over both sets of dominos!
2. Let's get Dash to drive in a square and knock over all the dominos. Place the following blocks in the workspace: drive forward, turn right, drive forward, turn right, drive forward, turn right, drive forward

3. Let's get Dash to do a circle by making him turn right four times without all that code! Use a "Repeat 4 times" block from the control menu and place a "Turn Right" block inside of it.
4. Let's make Dash drive in a square. Place a "Drive Forward" block after the "Turn Right" block inside of the repeat statements.
5. Free for all! Make Dash do whatever you would like using the Sounds, Drive and Control Blocks.

Questions

1. Are you having fun?
2. What did you find difficult about figuring out which blocks were going to be repeated? Can you find the blocks inside the repeat statements?
3. What did you find easy about it?
4. What would you change about this method to make it easier or more fun?

Spatial Interface

- Repeat: Let's make Dash Start Engine three times
    - Set-up:
        - Repeat 3 times -> Start Engine -> End Repeat
    - Explore workspace
        - Where is the start engine noise block?
        - notice that block is inside a begin and an end repeat block
            - also there is an inside repeat block and you can follow up to find what block is inside (start engine)
- Select a repeat block from the control blocks
    - what does it look like in the workspace?
        - two blocks
            - start block, end block
    - what happens when we run code?
        - nothing, telling Dash to repeat nothing
    - Let's add something more
        - add block inside repeat
        - add block outside repeat

Task Set 2

1. Let's make Dash meow like a cat, make a horse noise and then meow again, by selecting the "Make Cat Noise" and "Make Horse Noise" blocks and adding them in the correct order!
2. Let's get Dash to "Say Hi" and "Make Cat Noise" 4 times. Place the following blocks in the workspace: say hi, make cat noise, say hi, make cat noise, say hi, make cat noise,

. Set-up:

3. Let's get Dash to meow by making him meow four times without all that code! Use a "Repeat 4 times" block from the control menu and place a "Make Cat Noise" block inside of it.
4. Let's make Dash meow and say hi multiple times. Place a "Say Hi" block after the "Make Cat Noise" block inside of the repeat statements.
5. Free for all! Make Dash do whatever you would like using the Sounds, Drive and Control Blocks.

Questions

1. Are you having fun?
2. What was hard about that? What did you find difficult about figuring out the repeat blocks?
3. What did you find easy about it?
4. What would you change to make this easier or more fun?

End of Session Questions

1. What was your favorite part?
2. What was your least favorite part?
3. How did you like the two different versions of the interface?
4. Why?
5. What would you change?
6. What did you find difficult about programming?
7. What did you find easy about programming today?
8. Are you interested in doing more programming?
9. Have you learned any computer science before (programmed robots, code.org, hour of code)? How much experience do you have with computer science?
10. Have you used a smartphone (iPhone) or a tablet (iPad) before? If so, how often do you use it? and what for?

End of Session Questions for Parent/Teacher:

1. What do you think your/the child had the most difficulty with when using this version?
2. What do you think your/the child found easiest when using this version?
3. How did you like this version of the game compared to previous versions?
    a. Why?
    b. What would you change?
4. Any feedback on how to improve this interface?

Intrinsic Motivation Inventory (IMI) for enjoyment (scale of 1 to 5) (asked in final session)

1. While I was working on programming, I was thinking about how much I enjoyed it
2. I found my playing of programming to be very interesting
3. Doing programming was fun
4. I enjoyed doing programming very much
5. I thought programming was very boring
6. I thought doing programming was very interesting
7. I would describe programming as very enjoyable
8. I think I am pretty good at solving programming
9. I think I did pretty well at programming compared to others
10. I am satisfied with my performance at programming
11. I feel pretty skilled at the game
12. After working at programming for awhile, I felt pretty competent

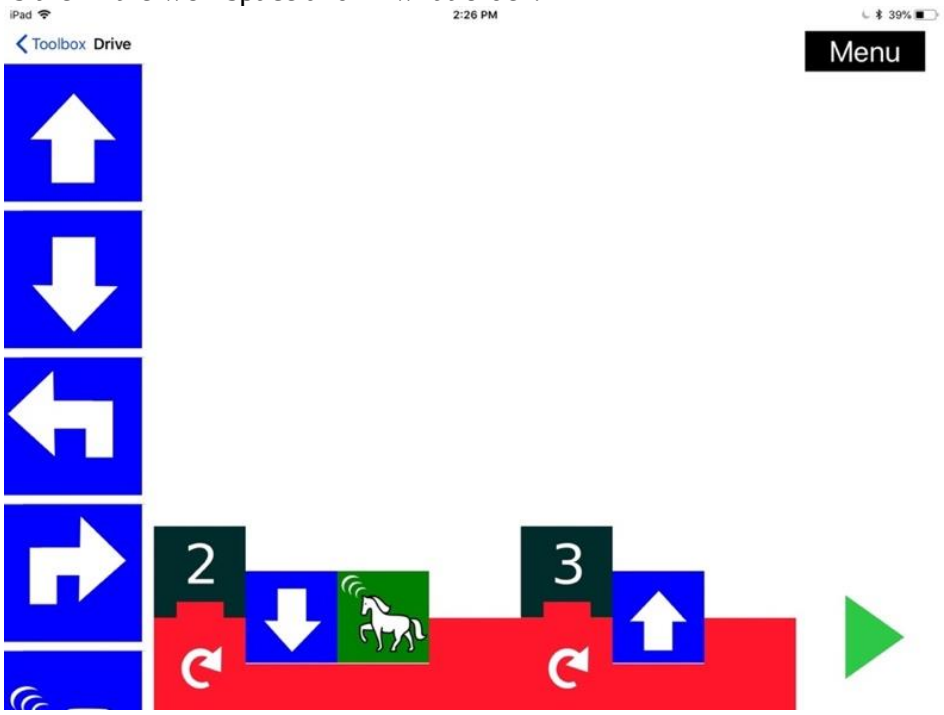
APPENDIX L

Blocks4All Evaluative Study Materials: Tasks and Child Interview Questions

TASK SET 1: DRAG AND DROP (can substitute in blocks of the same category)
Place *block1* in the workspace and press play (e.g. favorite animal noise, goat noise)
Place *block2* before it (e.g. move forward, make dog noise)
Place *block3* in between first two (e.g. move backward, make lion noise)
Move *block1* in between the other two
Move to trash

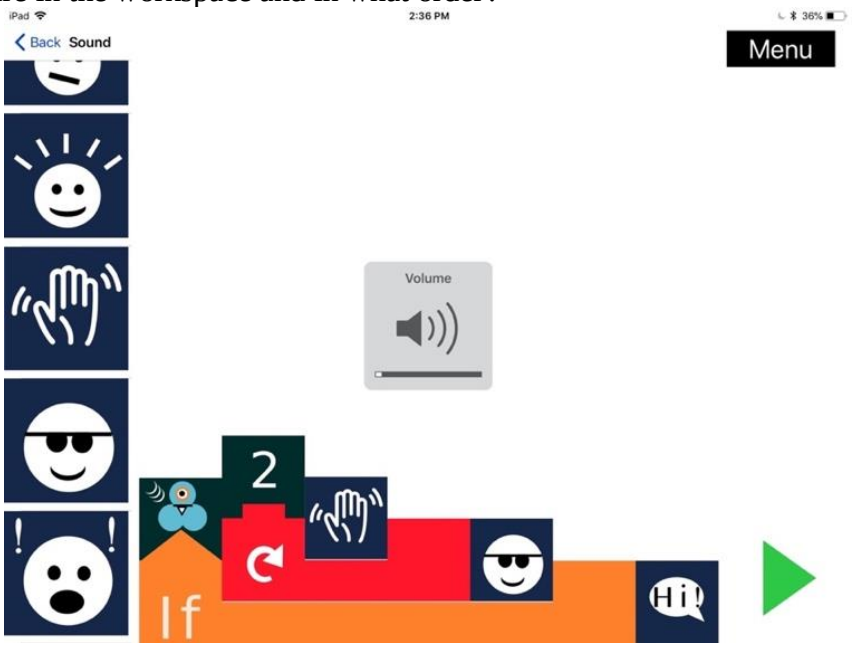
TASK SET 2: ACCESSING TYPE INFO/MODIFYING REPEAT BLOCKS (can substitute in blocks of the same category)
Make Dash do *block1* (e.g. drive backward, say hi, make random noise) if he hears a voice
-Add if block
-Add hear voice (or other condition that fits)
-Add *block1* inside if
Now make him do *block2* (e.g. dance party, say hi, make random noise) 3 times inside if he hears a voice (only use one copy of *block2*)
-Add repeat block inside if
-Change repeat condition to 3 times
-Add *block2* inside repeat

TASK SET 3: UNDERSTANDING PROGRAM STRUCTURE
Which blocks are in the workspace and in what order?



TASK SET 4: UNDERSTANDING PROGRAM STRUCTURE

Which blocks are in the workspace and in what order?



Children Post Session Interview Questions

Are you having fun? (scale from 1,5)
What is difficult about using the interface?
What is easy about using the interface?
What would you change?
Favorite part?
Least favorite part?

System Usability Scale [190]

When a SUS is used, participants are asked to score the following 10 items with one of five responses that range from Strongly Agree to Strongly disagree:

1. I think that I would like to use Blocks4All frequently.
2. I found Blocks4All unnecessarily complex.
3. I thought Blocks4All was easy to use.
4. I think that I would need the support of a technical person to be able to use Blocks4All.
5. I found the various functions in Blocks4All worked together well
6. I thought there was too much inconsistency in Blocks4All.
7. I would imagine that most people would learn to use Blocks4All very quickly.
8. I found Blocks4All very cumbersome to use.
9. I felt very confident using Blocks4All.
10. I needed to learn a lot of things before I could get going with Blocks4All.

VITA

Lauren Milne is a graduate student at the Paul G. Allen School of Computer Science at the University of Washington, advised by Richard Ladner. She is an NSF graduate research fellow and her research interests include making touchscreen applications for children with visual impairments and making programming more accessible in general. She graduated magna cum laude from Carleton College in 2008 with a B.A. in Physics and spent three years after graduating working as a personal care assistant to a young man with an intellectual disability, leading to her interest in accessibility. She also spent a year doing crazy science experiments with elementary school children as a Mad Scientist, leading to her interest in designing educational tools for children. She is passionate about making STEM education more accessible for underrepresented groups. Outside of academics, she enjoys hiking, biking, skijoring, triathlons and roller derby.