A Tangible Block Editor for the Scratch Programming Language

Bryson, J, Goolsby
Virginia Commonwealth University, USA
goolsbybj@vcu.edu

Hyun Woo Kim
Virginia Commonwealth University, USA
kimhw3@vcu.edu

Dianne, T.V, Pawluk
Virginia Commonwealth University, USA
dtpawluk@vcu.edu

Giovanni Fusco
Smith-Kettlewell Eye Research Institute, USA
giofusco@ski.org

ABSTRACT

We describe the early-stage development of a tangible block editor for the educational programming language Scratch that is intended to contribute to an environment that will allow blind and visually impaired (BVI) students (grades 6-12) to learn computer programming concepts alongside their sighted peers (both independently and in pairs) in mainstream classrooms. In this late breaking work, we describe our design that incorporates many of the key strategies of the Scratch visual code editor meant to promote engagement and lower hurdles to programming. Novel key elements of the design include: the strategic use of magnets and locally interlocking block edges to ensure only blocks with valid syntax can be connected, the use of telescoping tubing to specify parameter/operand location and allow their expansion for nested expressions and a block-sized-channel grid work surface that provides structure to aid BVI students in navigating and manipulating their programs.

CCS CONCEPTS

• **Human-centered computing** → Accessibility; Accessibility systems and tools; Collaborative and social computing; Collaborative and social computing systems and tools; • **Hardware** → Communication hardware, interfaces and storage; Tactile and hand-based interfaces.

KEYWORDS

Computer programming, tactile interfaces, tangible blocks, fiducial markers, blind and visually impaired (BVI) accessibility

ACM Reference Format:
Bryson, J, Goolsby, Hyun Woo Kim, Dianne, T.V, Pawluk, and Giovanni Fusco. 2021. A Tangible Block Editor for the Scratch Programming Language. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts (CHI '21 Extended Abstracts), May 08–13, 2021, Yokohama, Japan.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3411763.3451833

1 INTRODUCTION

Unfortunately programming environments being developed to increase engagement and lower hurdles to programming for sighted K-12 students, such as Scratch, are actually increasing barriers for students who are blind or visually impaired (BVIs) due to their highly visual nature. This is problematic as most BVI students are taught in mainstream schools alongside their sighted peers and where these programming environments predominate (as well as in most computer clubs and camps). Although text-based programming languages are much more accessible for BVIs through the use of screen readers and other audio interfaces [18, 20–22, 29], equal but separate is not a solution. In addition, Scratch, in particular, has thoughtfully incorporated ideas to increase engagement and ease learning [17]. These should not be abandoned but rather translated to an appropriate methodology for BVI students to benefit from as well.

Our objective is to make the Scratch environment accessible to BVI students to allow them to experience the lower barriers to programming alongside their sighted peers. In this paper, we focus on a nonvisual solution for the code editor that maintains some important characteristics of the Scratch visual code editor, including: (1) the use of code construction through action, (2) a design reducing the need to struggle with syntax, (3) a straightforward environment and (4) the ability to construct code individually and with others. Our approach is to use a set of specially designed tangible blocks that inherently implements the idea of code construction through action, while also addressing the other issues mentioned above. In this paper, we will first review related work, then present the design and development of our system and finally summarize and discuss future work.

2 RELATED WORK

There are two main approaches taken to make programming accessible to BVIs: making text-based programming languages accessible and making tangible programming languages (see [16, 29] for reviews). For the most part, approaches making text-based languages accessible have not been able to take advantage of the methods used in visual block-based languages to lower the hurdles to programming. One step in this direction, by Sanchez and Aguayo, limited and made circular the command list to choose from to decrease the emphasis on syntax [18]. However, studies on the use of tangibles in teaching have shown that they naturally capture some of the goals of visual block-based languages: encouraging engagement, excitement and collaboration, promoting discovery and participation, and making computation immediate and more accessible [7].

For tangible approaches, two main directions have been taken: active blocks with electronics embedded and passive blocks which are tracked with a camera or scanner [16]. Project Torino's product, Code Jumper, is a prominent example of active blocks: it consists of tangible "pods", containing custom printed microcontrollers, that act as programming statements and interact with additional pods with wired connections [16, 26]. However, the active blocks make it expensive to extend beyond a small set of code pieces: this is still appropriate for their targeted demographic of ages 7-11, but not for the demographic of Scratch users (ages 11-18) for which the ability to make more complex programs is desirable. Several groups have considered the use of passive blocks with camera or scanner tracking of tags for both computational (e.g., [28]) and non-computational activities (e.g., [3]). In particular, the StoryBlocks project uses tangible tiles with raised tactile symbols to construct stories by fitting tiles together like puzzle pieces [28]. We believe this type of approach can further take advantage of aspects of the visual Scratch environment to help lower barriers to programming.

3 TANGIBLE BLOCK EDITOR DESIGN OVERVIEW

The design approach worked to choose the most effective means to address the requirements itemized in the introduction based on the visual Scratch environment, previous work with tangible environments for BVIs, knowledge of tactile and haptic perception, and stakeholder involvement.

3.1 Stakeholder Involvement

During the development process, feedback from BVIs and their teachers/rehabilitation professionals was sought. The two main methods were: obtaining feedback from BVI high school students about the tangible block design and working with blind teachers of the blind to develop the tangible block and tactile surface designs. The first involved presenting tangible block prototypes to seventeen BVI high school students during the 2019 Learning Excellence in Academics Program (LEAP) summer program at Virginia Commonwealth University. The second involved several meetings with Michael Fish, lead technology instructor, and Domonique Lawless, orientation and mobility instructor, at the Rehab Center of the Virginia Department for the Blind and Visual Impaired (DBVI).

3.2 System Overview of Editor

The tangible block editor is designed as an alternate, BVI accessible code editor for Scratch that can be used by both BVI and sighted students, individually or in arbitrary pairs, to create programs of small to moderate size. The target demographic is students from grades 6 to 12 in computer education class settings and at home. An approach using passive pieces was chosen due to the desired maximum allowable program size (which would be cost prohibitive with active pieces) and age group (i.e., old enough to follow a design workflow) [16].

The main components of the design (Figure 1) are the tangible code blocks themselves for which a physical "palette" is created to hold and organize the different types of code blocks when not in use. Our initial version of the tangible block editor focuses on the motion commands, control statements, operators, variables and simple event blocks from the Scratch environment, although eventually all commands will be included to allow accessibility to any Scratch program created. The code assembly workspace is a structured surface used to facilitate navigation and assembly of the code blocks. A small part of the workspace, adjacent to the physical palette, is dedicated to speaking the block name and parameter requirements within that area.

The tangible code editor is meant as an alternative editor for Scratch and we intend to map the tangible blocks into the visual Scratch editor (where it enters the Scratch environment) in real-time. For this, the tangible code blocks are identified and tracked using markers placed on the bottom of individual pieces. A Logitech web camera mounted in the center of the support frame underneath the clear work surface determines each code block's identity and spatial location in real-time. A below surface rather than an above surface camera was chosen to avoid occlusion of the blocks from camera view by users' arms and bodies. A second, above surface Logitech web camera will potentially be used to track the user's fingers as part of a navigation assistance subsystem.

By acting as an interface for Scratch, users will be able of produce programs of various complexities from simple looping examples to creating robust computer games that can be shared online [17].

4 TANGIBLE ELEMENTS

4.1 Tangible Code Blocks

We have observed several features in the Scratch visual code blocks that make them easy to use, promote program concepts and reduce the emphasis on syntax. For easy selection of the appropriate code block, blocks are grouped into categories, which are color coded, and command labels are short and meaningful. Blocks are sized for easy manipulation and snap together appropriately when the syntax is valid. All code blocks with parameters have a specific "slot" for each parameter, with a label, to ensure the correct number of parameters are used. The shape of the slot ensures that only a valid parameter type (i.e., logical versus numeric) is used. Another important concept is that these parameters are "expandable" to allow for nested expressions not just explicit variables and literals. If and repeat code blocks have similar vertical expansion abilities.

4.1.1 Overall Block Design. The prototype tangible blocks (e.g., Figure 2) were 3D-printed on an Ultimaker 3 using PLA. The general shape of a basic block is a 1" by 1" square, which is easily manipulated by the user and allows over 450 blocks to fit into a rectangular area within the average 11-year old's arm span [30]. To identify the code block category, both the overall block color and the texture along the bottom edge of the block are provided. Texture is used rather than block shape (e.g., [28]) as it is easier and quicker to process tactually [31], as well as distinct in nature from our actual code block command symbols (Figure 2, right). In addition, uniform blocks, in contrast to those varying in shape and/or size, make it easier to construct a structured work surface to facilitate program management. The final blocks will have distinct, bright colors to aid students with low vision. Colors will match those in the visual editor to aid sighted students moving between virtual and tangible environments. The textures chosen were selected from an experimentally derived texture palette [11].

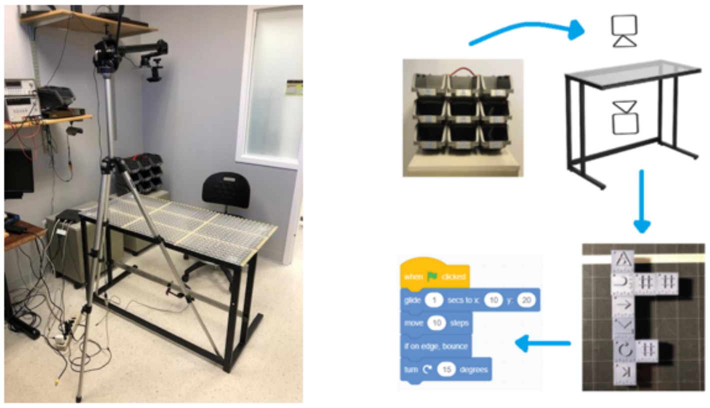

Figure 1: Left: Tangible workspace consisting of the program assembly area on the table top and the code block organizer to the right of the user (who would sit in the chair). Both cameras are shown, one above and one beneath. Right: Diagram of workflow of system: Top left, users take appropriate blocks out of organization bins; Top right, as blocks are placed on workspace, they are visual track by associated web cameras; Bottom right, user creates Scratch program with tactile blocks; Bottom left, positions and placement of tactile blocks are translated in digital Scratch GUI.

Code block commands are represented by simple and intuitive raised relief symbols in a high contrast color of either black or white (depending on the block color), with a raised orientation marker in the upper left-hand corner used to orient the piece (Figure 2, left). Tactile symbols were chosen as a representation (similar to e.g., [28]), rather than text or Braille, as Roman letters are very difficult to perceive tactually and most people (BVI or sighted) do not know Braille. Previous work on tactile character recognition [11] and feedback from our stakeholders were used in creating symbols that were legible and intuitive.

All blocks also have magnets (carefully selected in strength) embedded in them on their top and bottom sides (Figure 2, right), where correctly orienting the polarity for the top and bottom sides of the block allow pieces to snap together if correctly aligned and placed close together. Feedback from our stakeholders suggested that this provided a stronger sense of blocks "belonging together" than using interlocking edges alone. The magnets also facilitate the movement of whole multi-line program chunks on the workspace and provide resistance to the inevitable disturbance by tactile exploration and manipulation of the assembled code by the hands (although this contrasts with the experiences in [16] with Magnets construction blocks). Although snap-fits are a cost effective alternative, they wear easily and can become ineffective with use.

The back of each block has a raised square "peg" which allows for each piece to slide along the channels of the tactile surface of the workspace (see Section 4.2). These pegs are also recessed to accept (and protect from damage) a small visual orientation marker, used to track block movement (see Section 5.1).

4.1.2 Blocks with Parameters. The design for blocks with parameters incorporates the design ideas for the visual Scratch blocks: (1) a "slot" is provided for each parameter, (2) blocks or block expressions cannot physically be attached in a slot if they do not belong, and (3) slots can expand to allow for nested expressions. These ideas were implemented in the tangible code editor by a block assembly with physical, expandable slots for parameters that can only be inserted when valid (e.g., Figure 4). The slots are created and made expandable through the use of copper telescoping tubing. Copper was chosen due to its much larger modulus of elasticity compared to plastics and even aluminum: this was important as more compliant materials caused bending, resulting in sticking when the tubing was expanded and contracted. Other methods tried, such as using tangible reels [5] or physical string or cord [26, 28], had a hard time clearly showing the relation between the parameters and the code command, especially for nested expressions. Syntax is enforced using the location of interlocking juts and notches rather than connection shape (as in visual Scratch). This choice was for two reasons: (a) connection shape is not easily discernible by touch and (b) magnets can be used to enhance the use of location to define syntax but not the use of connection shape.

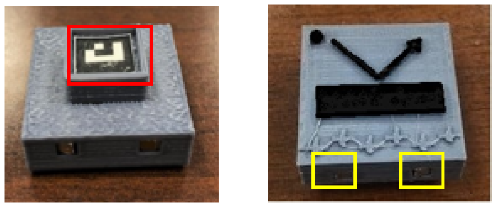

Figure 2: Tangible code block for "If on edge, bounce". Left: facing the back surface. Right: facing the top surface; yellow boxes highlight the imbedded magnets used in the blocks and the red box highlights the orientation marker.

The created block assemblies are illustrated for the PLUS operator and the AND operator in Figure 3. For both operator assemblies, we have bounding blocks, which could be thought of as parentheses in text-based code and which are implicit in the visual blocks of Scratch. The main block is the operator, with the appropriate number of slots added for the given operator. For the tangible blocks, the location of the notches(s) on the left side of a parameter slot (indicated in red in Figure 3) indicate what type of expression resultant

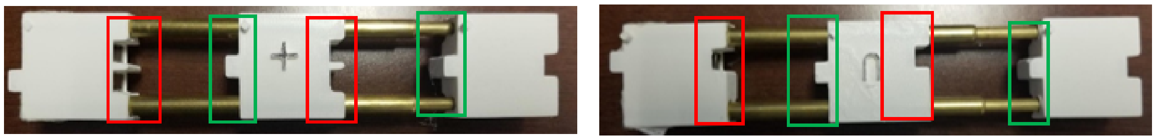

Figure 3: Left: PLUS operator with expandable slots, Right: AND operator with expandable slots. Both: telescoping tubes appear as parallel sets of brass tubing between each block in the expandable elements. Note that neither assembly includes the texture and color for Operator blocks.

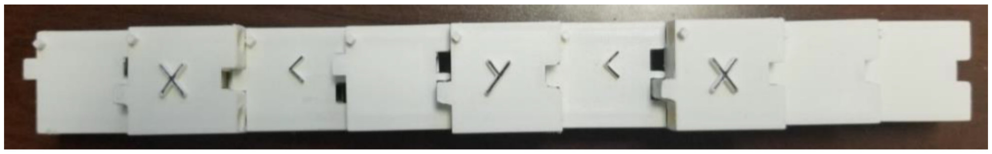

Figure 4: (x < (y < x)), correct expression. Note that the resultant of (y < x) is a logical value.

is accepted: the top notch is for logicals and the bottom notch is for regular numbers. The notch on the right-hand side of a slot is positioned so that the command cannot be completed without a parameter (indicated in green in Figure 3). Again, magnets are used for a "snap effect", this time placed directly in the juts and notches of the interlocking connections. The telescoping used in the parameter slots (e.g., Figure 4) currently allows 2 levels of nesting.

4.1.3 Variables and Literals. Variables and literals are more difficult to represent in a tangible interface as compared to a virtual one as they must be represented physically despite being any of an infinite number of values/names. Multi-digit combination "locks" were considered for literals, however the feasibility and cost of producing a tangible version did not seem realistic. Using a Braille label maker and sticking the created number on a blank block is possible but would require a method for the editor to automatically interpret the created value. However, for our initial development we will only allow the use of variables (which also encourages good programming practice).

Variables are represented by blocks with their name in raised relief (e.g., "x" and "y" in Figure 4), with duplicate blocks provided to allow multiple instances of each variable in a program. We will work with our stakeholders to select 10 easily discernible names/symbols for our variables, and provide up to 10 instances of each. Users will be able to assign initial values and give their variables meaningful descriptions within a specific area of the tangible workspace. In the future, we will explore using user generated Braille labels on blank blocks.

4.2 Tangible Workspace

An important concept of the visual Scratch workspace is that is designed to promote tinkering by: (a) being able to make changes to code while it is running, (b) creating parallel threads by simply creating parallel stacks of blocks, and (c) the ability to leave extra blocks or stacks around in case they are needed later [17]. In order to create moderately sized programs and provide for the latter two functions of tinkering, a sufficiently large workspace is needed. However, the use of a tangible environment by BVIs presents a unique problem as the spatial field of view through touch is significantly more limited than vision. This means that the conceptual organization of the workspace in the user's mind must be done through sequential contact with the hands, which is slow and cognitively very demanding [31].

Educators of BVIs have suggested that a confined space be used for programming [32]. However, feedback from our stakeholders suggested that this was insufficient. They felt it was hard to structure lines of code in an empty workspace and wanted some structure. They also thought that a surface structure that facilitated the connection of code blocks would be beneficial. A tangible workspace (Figure 1) was created with a structured surface to address these concerns and an organizational method for code blocks being stored off the surface in bins.

4.2.1 Surface. The structured surface consists of a grid of channels (Figure 1) that the tangible code blocks fit in and can slide along. Program code elements (blocks and block assemblies) are made in a standard size or integer units of the standard size to enable movement within the channels. The created grid of channel grooves results in the bottom "post" of each block (Figure 2) being restricted to movement within the grooves, which maintains alignment of a block along a row or column while it is moving. The standardized size of the grid (and blocks) allows adjacent blocks to be connected across the channels in both horizontal and vertical directions. The channeled grid is meant to prevent accidental movement of the code blocks when reading the program with the hands and to facilitate alignment when assembling adjacent pieces. The grid organization is also intended to facilitate the user's recall of where they placed blocks in the workspace. The structured surface was created from a sheet of transparent acrylic 23.5 inches by 48 inches, which translates into 23 by 48 code blocks. Horizontal and vertical channels were ground into the top surface of the sheet using a CNC End Mill. Tolerances were chosen to allow the blocks to slide in horizontal and vertical directions in the channels, while preventing the blocks from rotating freely.

4.2.2 Palette Organization. Given the large number of coding commands, a method to organize and easily retrieve tangible code blocks is essential. A preliminary organizational prototype for the tangible block palette uses stackable storage bins (Figure 1). Each category of code blocks is in a different row, with one type of code block per bin (labelled with a plate that contains the front face of the block).

4.2.3 Block Management Area. Again, given the large number of different coding commands used in the tangible editor, as well as their representation solely by tactile symbols, a sub-area of the workspace (beside the palette organizer) is used to provide audio descriptions of any blocks or block systems placed in it. This is also where we expect to assign variable values, although currently variables are only represented symbolically in the editor.

5 SOFTWARE DESIGN

5.1 Real-time Marker Information Extraction and Translation

The identity and location of the code blocks on the workspace are determined using the OpenCV library to track ArUco markers on the back of the blocks (Figure 2) via the camera positioned below the clear workspace surface (Figure 1). The OpenCV detection algorithm provides the relative distances between the markers it tags. Markers placed at known distances from each other on the workspace surface (one in each corner) are used to determine absolute position of each code block. To date, the detection method has not failed to detect a code block's id and location (x,y,θ) in spite of the structure of the workspace surface which interferes with its optical clarity to some degree. Current work is now focused on using the extracted data in real-time to create a homomorphic representation in the visual Scratch editor through Blocky keystroke commands (as Blocky underlies the visual Scratch code editor).

5.2 Navigation Assistance Subsystem

An important consideration for BVI students is: how do they, when working on a program, interact with a teacher (who may be referring to a highly visual lesson plan at the front of the room) or other students (both BVI or sighted, who may be working with them such as in paired programming)? What tools need to be provided for communication? Even if a sighted person is close enough to "pull" a BVI student's hand to what they want to show, it is more desirable for BVI students to have agency over themselves. However, haptic exploration of the workspace without vision is slow. The navigation assistance subsystem is predicated on the belief that guidance inherently built into the tangible editor system itself will be useful and inoffensive to BVI students. Currently we are testing this belief with a variety of different audio feedback algorithms. If desirable, a functional subsystem will be developed.

6 DISCUSSION

In this paper we presented a prototype tangible block editor that can be used as a code editor for Scratch by both BVI and sighted students. The design approach preserves the constructivist approach of Scratch by using tangible code blocks to enact code construction through action. The approach also preserves the low barriers to programming by implementing puzzle piece style fits and snap to connections for syntax, and expandable parameter slots for nested expressions. Modifications were also made to accommodate differences in haptic and visual perception, particularly the difficulty of navigating space haptically, without vision: In particular, a channeled grid-based surface was created and automated audio navigation assistance proposed. These new design components are currently being assessed for their usability and usefulness. We should also acknowledge the limitations in the design, which are primarily due to using a physical rather than a virtual environment: the size of the programs that can be constructed has a limit, nesting of operator expressions is currently restricted to two levels deep, and variables are not easily created and assigned. For the most part, we do not expect these limitations to be too restrictive for our aim in improving educational access to Scratch for BVIs.

7 FUTURE WORK

The most immediate next step is to complete a fully functional tangible block editor and perform an objective assessment of its usability and ability to engage BVI students. However, it is not the only component of Scratch that needs to be made accessible for BVIs. At the very least, a method to execute (and follow the execution of) the developed code is critical for learning how to program. We are exploring several different approaches: 1) the use of mini robots as dynamic interaction objects (similar to Ducasse's work [6]), 3D sound (similar to computer action games), descriptive audio and a combination of all three. In doing so, we believe one of the most critical properties of any approach is that it needs to be interchangeable with Scratch's stage (where code is animated visually). This, as well as to be developed web access tools, will be critical to attain the social, shareable (in class and online) aspect of Scratch programs, which is an important facilitator of student engagement.

ACKNOWLEDGMENTS

This work is based upon work supported by the National Science Foundation under grant no: 17-422-42.

REFERENCES

[1] Edith K. Ackermann. 2004. Constructing Knowledge and Transforming the World. In *A Learning Zone of One's Own: Sharing Representations and Flow in Collaborative Learning Environments*. IOS Press, Washington, DC, 2004, 17-35.
[2] Mark S. Baldwin, Gillian R. Hayes, Oliver L. Haimson, Jennifer Mankoff, and Scott E. Hudson. 2017. The Tangible Desktop: A Multimodal Approach to Nonvisual Computing. ACM Trans. Access. Comput. 10, 3, Article 9 (August 2017), 28 pages.
[3] Sébastien Cuendet, Jessica Dehler-Zufferey, Giulia Ortoleva, and Pierre Dillenbourg. 2015. An integrated way of using a tangible user interface in a classroom. International Journal of Computer-Supported Collaborative Learning 10, 2 (2015), 183–208.
[4] Son Do-Lenh, Patrick Jermann, Amanda Legge, Guillaume Zufferey, and Pierre Dillenbourg. 2012. TinkerLamp 2.0: Designing and Evaluating Orchestration Technologies for the Classroom. Lecture Notes in Computer Science 21st Century Learning for 21st Century Skills (2012), 65–78. Jon M. Kleinberg. 1999. Authoritative sources in a hyperlinked environment. J. ACM 46, 5 (September 1999), 604–632.
[5] Julie Ducasse, Marc Macé, Marcos Serrano, and Christophe Jouffrais. 2016. Tangible Reels : Construction and Exploration of Tangible Maps by Visually Impaired Users. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (2016).
[6] Julie Ducasse, Marc Macé, Bernard Oriola, and Christophe Jouffrais. 2018. BotMap: Non-Visual Panning and Zooming with an Actuated Tabletop Tangible Interface. ACM Trans. Comput.-Hum. Interact. 25, 4, Article 24 (September 2018), 42 pages.
[7] Rabia Jafri, Asmaa Mohammed Aljuhani, and Syed Abid Ali. 2015. A Tangible Interface-based Application for Teaching Tactual Shape Perception and Spatial Awareness Sub-Concepts to Visually Impaired Children. Procedia Manufacturing 3 (2015), 5562–5569.

[8] Shun Kakehashi, Tatsuo Motoyoshi, Kenfichi Koyanagi, Toru Ohshima, and Hiroshi Kawakami. 2013. P-CUBE: Block Type Programming Tool for Visual Impairments. 2013 Conference on Technologies and Applications of Artificial Intelligence (2013).

[9] Arthur I. Karshmer and Ken Paap. 2010. AutOMathic Blocks: Supporting Learning Games for Young Blind Students. Lecture Notes in Computer Science Computers Helping People with Special Needs (2010), 459–465.

[10] Sébastien Kubicki, Marion Wolff, Sophie Lepreux, and Christophe Kolski. 2015. RFID interactive tabletop application with tangible objects: exploratory study to observe young children' behaviors. Personal and Ubiquitous Computing 19, 8 (2015), 1259–1274.

[11] Jack M. Loomis. 1990. A model of character recognition and legibility. Journal of Experimental Psychology: Human Perception and Performance 16, 1 (1990), 106–120.

[12] Stephanie Ludi, Mohammed Abadi, Yuji Fujiki, Priya Sankaran, and Spencer Herzberg. 2010. JBrick. Proceedings of the 12th international ACM SIGACCESS conference on Computers and accessibility - ASSETS '10 (2010).

[13] Stephanie Ludi and Tom Reichlmayr. 2011. The Use of Robotics to Promote Computing to Pre-College Students with Visual Impairments. ACM Transactions on Computing Education 11, 3 (2011), 1–20.

[14] Stephanie Ludi, Lindsey Ellis, and Scott Jordan. 2014. An accessible robotics programming environment for visually impaired users. Proceedings of the 16th international ACM SIGACCESS conference on Computers & accessibility - ASSETS '14 (2014).

[15] Muhanad S. Manshad, Enrico Pontelli, and Shakir J. Manshad. 2012. Trackable Interactive Multimodal Manipulatives: Towards a Tangible User Environment for the Blind. Lecture Notes in Computer Science Computers Helping People with Special Needs (2012), 664–671.

[16] Cecily Morrison, Nicolas Villar, Anja Thieme, Zahara Ashktorab, Eloise Taysom, Oscar Saladin, Daniel Cletheroe, Greg Saul, Alan F. Blackwell, Darren Edge, Martin Grayson, and Haiyan Zhang. 2018. Torino: A Tangible Programming Language Inclusive of Children with Visual Disabilities. Human–Computer Interaction 35, 3 (2018), 191–239.

[17] Mitchel Resnick *et al.* 2009. Scratch: Programming for All. Communications of the ACM 52, 11 (2009), 60–67.

[18] Jaime Sánchez and Fernando Aguayo. 2005. Blind learners programming through audio. CHI '05 extended abstracts on Human factors in computing systems - CHI '05 (2005).

[19] Bertrand Schneider and Paulo Blikstein. 2018. Tangible User Interfaces and Contrasting Cases as a Preparation for Future Learning. Journal of Science Education and Technology 27, 4 (2018), 369–384.

[20] Ann C. Smith, Joan M. Francioni, and Sam D. Matzek. 2000. A Java programming tool for students with visual disabilities. Proceedings of the fourth international ACM conference on Assistive technologies - Assets '00 (2000).

[21] Andreas M. Stefik, Christopher Hundhausen, and Derrick Smith. 2011. On the design of an educational infrastructure for the blind and visually impaired in computer science. Proceedings of the 42nd ACM technical symposium on Computer science education - SIGCSE '11 (2011).

[22] Andreas Stefik and Richard E. Ladner. 2015. Introduction to AccessCS10K and Accessible Tools for Teaching Programming. Proceedings of the 46th ACM Technical Symposium on Computer Science Education (2015).

[23] Debbie Denise Reese, Dianne T.v. Pawluk, and Curtis R. Taylor. 2016. Engaging Learners Through Rational Design of Multisensory Effects. Emotions, Technology, and Design (2016), 103–127.

[24] Detection of ArUco Markers. Retrieved January 11, 2021 from https://docs.opencv.org/master/d5/dae/tutorial_aruco_detection.html.

[25] Guidelines and Standards for Tactile Graphics. Retrieved January 11, 2021 from http://www.brailleauthority.org/tg/web-manual/index.html.

[26] Morrison, Cecily, Nicolas Villar, Alex Hadwen-Bennett, Tim Regan, Daniel Cletheroe, Anja Thieme, and Sue Sentance. "Physical Programming for Blind and Low Vision Children at Scale." Human–Computer Interaction (2019): 1-35.

[27] Lúcia Abreu, Ana Cristina Pires, and Tiago Guerreiro. 2020. TACTOPI: a Playful Approach to Promote Computational Thinking for Visually Impaired Children. The 22nd International ACM SIGACCESS Conference on Computers and Accessibility (2020).

[28] Varsha Koushik, Darren Guinness, and Shaun K. Kane. 2019. StoryBlocks: A Tangible Programming Game To Create Accessible Audio Stories. In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (Glasgow, Scotland Uk) (CHI '19). ACM, New York, NY, USA, Article 492, 12 pages.

[29] Alex Hadwen-Bennett, Sue Sentance, and Cecily Morrison. 2018. Making Programming Accessible to Learners with Visual Impairments: A Literature Review. *International Journal of Computer Science Education in Schools* 2, 2 (2018), 3–13.

[30] R.M. Malina. 1973. Selected body measurements of United States children 6 through 11 years. *The American Journal of Clinical Nutrition* 26, 11 (1973), 1265–1265.

[31] Susan J. Lederman and Roberta L. Klatzky. 1997. Relative availability of surface and object properties during early haptic processing. *Journal of Experimental Psychology: Human Perception and Performance* 23, 6 (1997), 1680–1707.

[32] Ana Cristina Pires, Filipa Rocha, Antonio José de Barros Neto, Hugo Simão, Hugo Nicolau, and Tiago Guerreiro. 2020. Exploring accessible programming with educators and visually impaired children. *Proceedings of the Interaction Design and Children Conference* (2020).