Accessible Block-Based Programming for K-12 Students with Vision Impairments.

Meenakshi Das[1], Daniela Marghitu[1], and Ayanna Howard[2]

[1] Auburn University, Auburn AL 36830, USA

[2] Georgia Institute of Technology, Atlanta GA 30332, USA

Abstract. Block-based programming applications, such as MIT's Scratch and Blockly Games, are commonly used to teach K-12 students to code. Due to the COVID-19 pandemic, many K-12 students are attending online coding camps, which teach programming using these block-based applications. However, these applications are not accessible to the Blind/Low Vision (BLV) population since they neither produce audio output nor are screen reader accessible. In this paper, we describe a solution to make block-based programming accessible to BLV students using Google's latest Keyboard Navigation and present its evaluation with four individuals with vision impairments. We distill our findings as recommendations to developers who may want to make their Block-based programming application accessible to individuals with vision impairments.

Keywords: Accessibility · Block-Based Programming · Blind/Low Vision.

1 Introduction

Block-based programming relies on visual cues and structures to convey information to users. In addition, they heavily utilize drag and drop gesture to create computer programs. This makes it inaccessible for people with visual impairments to interact with them. We reviewed the current approaches to make Block-based programming accessible and used the following research questions in guiding us to develop our solution:

1. What interaction techniques can individuals with vision impairments use to understand the structure of a block-based program?
2. What interaction techniques can individuals with vision impairments use to receive non-visual feedback from a block-based program?
3. What interaction techniques can individuals with vision impairments use to debug a block-based program?
4. What are the benefits of a screen-reader vs a non-screen reader approach?

Our contributions ultimately built upon the Keyboard Navigation feature [1] which was released recently by Blockly. Since many Block-based applications utilize the Google Blockly library as a base for their application, we concur building upon their already developed accessible keyboard navigation solution is the best standard approach.

2 Related Work

There has been substantial effort in making Block-based programming accessible to visually impaired individuals. Google first released their accessible version of Blockly which replaced the drag and drop layout with a text layout for screenreader compatibility [2]. Stephanie Ludi's work mainly focused on adding screen reader capability to Blockly, along with keyboard navigation, while maintaining its original block-based interface [2]. Google later released their own version of Blockly with Keyboard navigation for its blocks [1]. Although the blocks in this current version are navigable by keyboard, they neither produce audio output nor are accessible via screen readers. Lauren Milne and Richard Ladner created Blocks4all, an iOS touch-based tablet application which is accessible via VoiceOver [3]. Varsha Kaushik and Clayton Lewis have proposed a non-visual blocks language called Psuedospatial blocks (PB), which distorts spatial layout, and is based on T.V. Raman's idea that instead of making spatial visual data accessible to screen-readers, the focus should be on making better non-visual representations of information. Their application uses synthetic speech instead of a screen-reader for providing output, although they state it will be possible to add screen reader support to it as well [4].

3 Our solution

Custom Text-to-speech Our initial data collection shows that not all K12 students are proficient in using a screen-reader. Some of them come from low social-economic backgrounds and may not have had the financial means to learn to use a screen-reader. One of the most popular and affordable screen readers, Job Access with Speech(JAWS), also costs about 90 dollars a year. Mac computers come with a free screen-reader called VoiceOver, but a Mac itself is quite expensive. Moreover, some K-12 students might have lost their eyesight in recent times, and not be adept in using a screen-reader yet which has a steep learning curve. When Google conducted user studies with their initial Accessible Blockly approach, they found that many students were not comfortable using a screen-reader [5]. Hence, we developed a non-screen reader, self-voicing solution using synthetic speech and interactive text to speech approaches. The ability to use a screen-reader adeptly should not be a prerequisite or a barrier to learning to code, and hence we took this approach.

Retaining spatiality The Individuals with Disabilities Education Act(IDEA) requires that students with disabilities receive education in the least restrictive environment, i.e alongside their non-disabled peers. We make this possible by using a spatial approach to convey information, instead of non-visual approaches which distort spatiality. Although non-visual approaches have been shown to have immense learning benefits for blind users [6], they are not very useful if blind and non-blind individuals want to collaborate and work together. In a research conducted on access overlays for blind users [7], it was shown that, "Access techniques that distort or remove spatial information may reduce users' spatial understanding and memory"; furthermore, this distortion makes it more difficult for a blind person to collaborate with sighted peers. Peer programming is a common method through which not only students, but adults at software companies code. Our approach maintains a spatial yet accessible organization of the block-based coding platform, making it easier for sighted and non-sighted peers to collaborate and learn programming together.

Auditory Cues Auditory Cues such as Earcons are distinctive sounds that convey certain information. In studies for teaching robotics to BLV K-12 students, earcons were shown to improve learning of coding concepts [8]. In addition, using non-speech audio such as earcons can greatly reduce the cognitive burden that comes with sole speech output [9]. Blockly already features some auditory cues such as a *click sound* to denote the deletion of a block. Research has shown that sounds produced from different spatial locations are easier to distinguish [10]. For example, when you play video games, you can hear certain audio sounds from only the left/right ear of the headphones or certain sounds can feel "nearer" than others. Audio software enables this process by specifying audio coordinates in a 3D space. This type of binaural spatialization has been effective in math notation feedback and its use has also been investigated in Pencil Code (a blockbased coding platform) with positive outcomes [11]. We used this technique to assist students to understand the opening and closing of nested code blocks. For example, a *click sound* is heard through the left and right side of the earphones respectively to denote the opening and closing of nested blocks.

4 Technical Details

Blockly fires an event for almost every change on its workspace [12]. All events contain properties that provide further information about that event. This was the key in developing the voice functionality as we could listen to specific event details on the workspace and convey them via speech to the user. Blockly currently has three cursors in their Keyboard Navigation functionality to navigate through the blocks. We chose the line cursor with few modifications since it closely mimicked a text editor - with the ability to move up and down and next and previous lines of code.

Adding Text to Speech Text to speech was added for the following:

1. Toolbox: As the user navigates the toolbox through its various categories and blocks, multiple events are fired. The workspace listens to these events and uses the Web Speech API to communicate the details of that event to the user. In this case, it would be the name of a category or a block.
2. Block connections: Blocks have several connections such as an input connection, block connection, previous connection, next connection, and field connections. Fig 1 shows the different types of connections a block may have. Specific values from these connections were listened to and sent to Web Speech API, and then ultimately conveyed to the user.

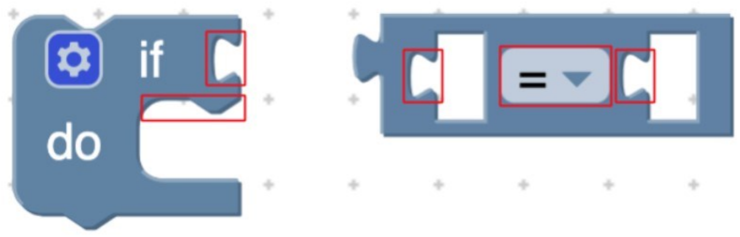

Fig.1. Image from Google Blockly's Keyboard Navigation Page which shows different connections of an *if-do* and *logic-compare* block.

3. Marking a connection: In order to connect a block to an existing block onthe workspace, a connection has to be marked on the latter. This can be done through navigating to the desired connection and pressing the *Enter* key. This leads to the connection color flickering between red and blue. Since there is no audio feedback for this feature, we listened to this marking event and sent the spoken speech *Location marked* to the Web Speech API.
4. Connecting two blocks: After marking the desired connection on a block ona workspace, the user proceeds to add a block from the toolbox to connect to that block. We added spoken speech which conveyed the *moved* block that was connected to the *parent* block on the workspace. For example, if the *repeat block* was already on the workplace, and the *print block* was to be inserted from the toolbox to the *repeat block's do connection*, the feedback would say, *Print block connected to the repeat block*. This way the user knows whether the block was inserted into the place it was intended to. If the two blocks are incompatible and cannot be connected, the feedback says, *This block can not be inserted at the marked location.*

Creating and Firing Custom Events At some of the places which required a voice feedback, there was no event present. An example of the location is the dropdown options of a field on a block. In Fig 2, a user was able to navigate up and down the dropdown list; however, no event was fired. Hence, there was no capability to add the voice feedback. After consulting with the Google Blockly team on their forum, we created our own custom events at the needed locations. In our case, for the Fig 2 scenario, we added an UI event to go up and down the dropdown field in the core Blockly code, fired and listened to the event, which ultimately allowed us to add the voice feedback .

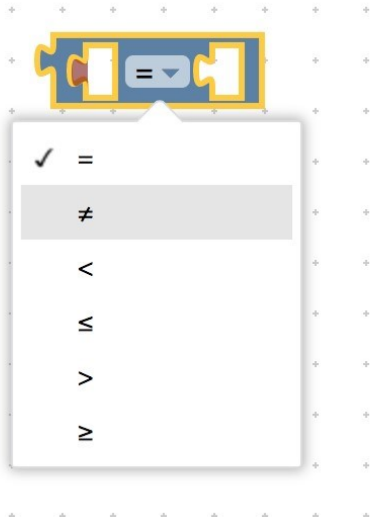

Fig.2. Image from Google Blockly which shows the dropdown options of a *logic compare* block.

Adding Keyboard Shortcuts Some features of Blockly such as accessing the Tooltip, deleting a block, are not accessible via the keyboard. A Tooltip in Blockly is a user interface element which provides more information on what a block does when you hover over it with a mouse. Hence, after consulting with the Blockly team on their forum, we added custom keyboard shortcuts for these features so a blind user could access them via the keyboard. After selecting a block, the user would have to press *CTRL + T* to access the tooltip and press the *DELETE* key to delete a block. In addition to this, we also added a shortcut to get the text representation of a block, including its nested children. For example, on pressing *CTRL + I* on the outer block in Fig 3, the user would hear, *if (count = 3) do print " Hello "*.

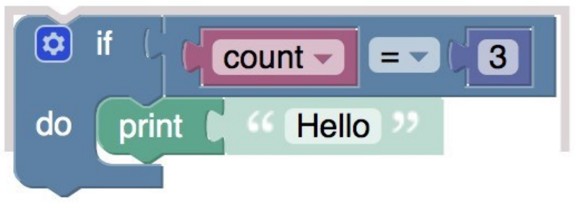

Fig.3. Image from Google Blockly of a block of code which says *if (count = 3) do print " Hello ".*

Adding Auditory cues via Binaural Spatialization To signify the opening and closing of a nested block, we made use of binaural spatialization, i.e., directing audio through the left or right audio channel. This was developed using the StereoPannerNode interface of the Web Audio API which provides the capability of panning an audio stream through the left or right. For example, in Fig 4, when the cursor is on *print block's top connection*(as denoted with a red line), a beep is heard through the left audio channel to denote opening of the nested block. Similarly, when the cursor is on *print block's bottom connection(as denoted with a blue line)*, a beep is heard through the right audio channel to denote closing of the nested block.

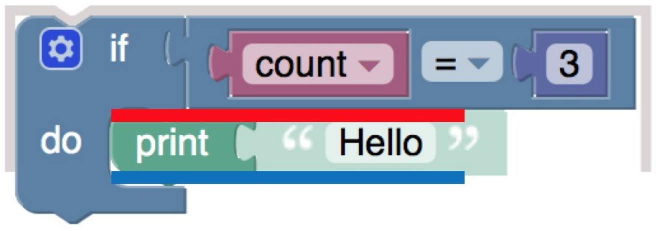

Fig.4. Image from Google Blockly of a block of code which says *if (count = 3) do print " Hello ". Print block's* top connection is marked with red, and bottom connection marked with blue.

Accessing Output The focus of this work is on Accessible input, i.e making block navigation, creating, inserting and moving blocks accessible to users with vision impairments. For the purposes of testing, we added output in the form of print statements. Participants were asked to write code which printed some text depending upon the logic of the code. This is explained further in the evaluation section. We are currently experimenting adding voice feedback to outputs of code in a robotic simulation as well. This involves the use of auditory cues and audio descriptions to let user know of the robot's movement and position in a simulation.

5 Evaluation

We evaluated our solution with four participants of whose visual impairments ranged from low vision to total blindness. Two of these were experienced programmers, while the other two were low or inexperienced in programming. The was done to get a variety of feedback. Table 1 below shows participant's demographic data.

The participants were given some simple warm-up exercises followed by a combination of some, all or related tasks below :

Table 1. Participant details

Participant	Age	Gender	Level of Corrected Vision	Screen Reader Proficiency
P1	18-22	Female	20/200 to 20/400 severe low vision	Moderate
P2	27-35	Male	Total Blindness with Light Perception	Expert
P3	14-17	Female	20/70 to 20/160: moderate low vision	Low
P4	14-17	Male	more than 20/1,000: near total blindness	Moderate

1. Write code using the *if-do block* which print's *Hello world* if the value of variable *test* is equal to *3*.
2. Write code using the *repeat while block* which prints *Hello world* until the value of a variable named *apple* reaches *10*.
3. Debugging: The following code in Fig 5 is supposed to print *Hello world* until the value of *count* reaches *3*. Find the bug on one line.
4. Debugging: The following code in Fig 6 is supposed to print *Hello world* until the value of *count* reaches *3*. Find the bug on one line.

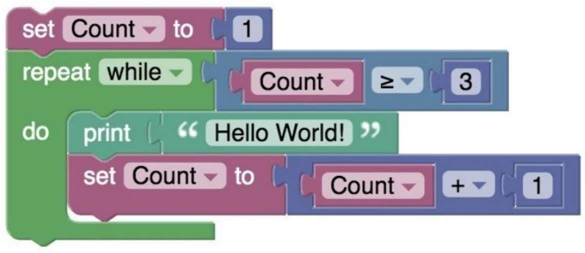
```
set Count to 1
repeat while Count ≥ 3
do  print "Hello World!"
    set Count to Count + 1
```

Fig.5. A block of code with the following text representation: *set Count to 1, repeat while (Count greater than or equal to 3) do print " Hello World! " , set Count to (Count + 1), end of do.*

Participant 1: Experienced Programmer working as a Software Engineer in Industry.

Observation: Participant at first was trying to connect two blocks without marking a connection on the workspace. Due to this, the blocks were simply inserted into the workspace and not connected to the desired block. She tried to move the block from the workspace and connect to the desired block, however current functionality only allows blocks to be connected via toolbox insertion. After this understanding, she was successfully able to complete all the tasks. However, she did not realize that binaural spatialization was used for the nested blocks until specifically told about it. We concur she was still able to debug the statements

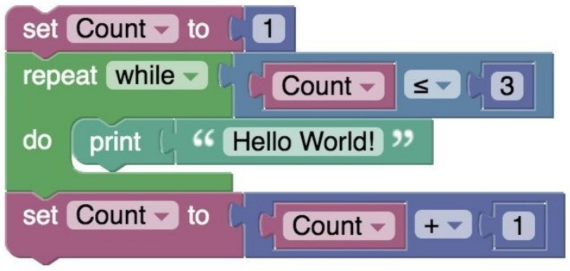

Fig.6. A block of code with the following text representation: *set Count to 1, repeat while (Count lesser than or equal to 3) do print “ Hello World! ” end of do, set Count to (Count + 1).*

successfully due to her having light perception and thus using it to understand the nested structure.

Feedback: Participant utilized the audio feedback most of the time with light perception guiding her to understand the structure of the code. As an experienced programmer who does not write code in linear fashion, the major frustration of the participant was not being able to move blocks across the workspace. As a person with moderate screen reader proficiency, she wanted a capability to increase speed of the audio feedback and more options to control the voice and verbosity. The participant liked the navigation and controls and found them easy to work with.

Participant 2: Experienced Programmer working towards a PhD in Computer Science.

Observation: When the participant tried to create a variable, there was no audio feedback provided as to whether the application was ready to take the variable name as input. This is due to the fact that when one clicks on the *create variable button* in Blockly, a JavaScript alert box pops up asking to type the variable name.

In other words, this is not a part of the Blockly workspace, hence there were no audio feedback provided. We guided the participant to creating the variable in this case. This could have been mitigated if his screen-reader was on to read information outside of the workspace or if synthetic speech was added for this particular instance.

Another instance where we guided the participant was while changing the value of a math block. A value of math block can be changed by navigating to its field and pressing the *Enter Key* which leads to the current value being selected and thus can be changed. However, no audio feedback was provided after pressing the *Enter Key*. Hence, we guided the participant in this scenario as well.
In the current cursor, a user can navigate up and down lines/connections of code using keys W and S respectively and can navigate in and out a line of code by using keys A and D respectively. However, if reached the start or end of a block using keys A and D, the cursor will automatically reach the first element of the block on new line, but *not* the new line's top or input connection. In other words, no block could be marked between those two lines. This caused some confusion to the user as he did not know whether a new line had started. Fig 7 illustrates this issue. If a user keeps pressing the D key starting from the *if element* on line 1, they will eventually reach the *do element* as marked by orange square in the figure. However, access to the *do connection* is needed to insert a block between *if and do*, as illustrated by a black line in the figure. Using the D key does not reach this *do connection*. Constraining how a user navigates between lines can fix this issue. The participant completed all given tasks.

Feedback: The participant liked the workflow of marking and inserting blocks. He also liked the binaural spatialization to understand the structure of the code once we explained what they meant. The participant said they would have liked more feedback as to how the toolbox was arranged with a more hierarchical description.

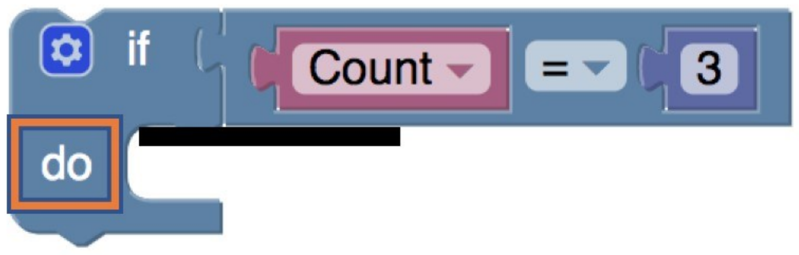

Fig.7. A block of code with the following text representation: *set Count to 1, repeat while (Count lesser than or equal to 3) do.*

Participant 3: High School Student with minimal programming experience.

Observation: The participant seemed to use a combination of mouse, zoom function and keyboard since she had moderate low vision. The participant experienced a bit of learning curve trying to understand the different connections and markings but once got used to it, was able to write code using blocks. She also had some understanding issues with the cursor as did participant 2. She did understand when a new line had started due to her moderate vision. However, same as participant 2, needed some help to navigate to the input connection of the next line as pressing the D key moved to the first element on the next line, and not its input connection.

Feedback: The participant said the software was easy to understand and the tasks helped her understand the basics of computer programming. She preferred having this synthetic voice than a screen reader because she felt that screen readers repeat words and read out unnecessary information. This can also be due to the fact she had low screen reader proficiency. She also mentioned she would have liked a custom zoom feature to zoom on individual blocks.

Participant 4: High School Student with no programming experience.

Observation: Due to technical difficulties arising from a Bluetooth Keyboard, we shifted our testing approach to a purely audio based one. Using our audio feedback solution, we went over a couple programs and asked the participant to answer what the program printed. The programs contained *if-do, logic compare and repeat-while blocks.* After the participant got used to do the synthethic speech, he was able to answer most of the questions correctly. However, this was purely based on the researcher operating the keyboard and the participant only listening to the audio feedback. The participant had keyboarding skills and we concur with practice the participant should be able to use the keyboard commands successfully as well.

Feedback: We did not get any feedback on the Keyboard Navigation. However, the participant mentioned that they believed with practice they could get a strong understanding of the application.

6 Recommendations for Improvements

We propose working on the following to improve the experience informed by our usability studies described above:

1. Personalization: Add controls to control the speed and verbosity of synthetic speech so users with different levels of audio comprehension have that flexibility.
2. Moving Blocks: Currently, the cursor will not move to a block unless theblock does not have a previous or next connection(which is unlike in case of most blocks), and hence moving blocks on workspace is not possible. We will modify the cursor to add this functionality.
3. Improved Audio Feedback: We will add audio feedback to the *creating variable* experience as explained in Observation data of Participant 2. Other ways to fix that experience could be :
   (a) move the create variable workflow to the Blockly workspace, as is in Open Roberta [13].
   (b) ask users to turn on screen-reader so content outside Blockly workspaceis accessible to them.

   We will also make granular aspects of the workspace accessible. For example, adding audio feedback after pressing the *Enter Key* to change the math block value. We will also add audio feedback on some hierarchical information of the toolbox. An example of that could be: *Logic, category 1 of 9...*
4. Improved Cursor: We will constrain how the user can navigate through theblocks so that they are not able to go next or previous lines of code without using their respective keys specifically. This should fix most of the navigation issues such as not knowing if new line of code has begun as found in usability testing.
5. Other: We hope to add screen-reader support to blocks itself so experienced screen-reader users can benefit. This can be done by using the Aria Live region to communicate where the cursor is. Stephanie Ludi and team has already done work on this [2]. In addition to this, we will add a zoom feature to assist users with low vision.

After we make the following changes above, we will re-evaluate our solution with more users with vision impairments.

7 Acknowledgments

We sincerely thank the Google's Blockly team for answering our technical questions on their public Blockly forum. This work is made possible due to a NSF Grant 1842092.

References

1. “Keyboard Navigation — Blockly.” Google Developers, https://developers.google.com/blockly/guides/configure/web/keyboard-nav. Accessed 10 Feb. 2021.
2. Ludi, Stephanie, and Mary Spencer. ”Design Considerations to Increase Block-basedLanguage Accessibility for Blind Programmers Via Blockly.” Journal of Visual Languages and Sentient Systems 3.1 (2017): 119-124.
3. Milne, L. R., & Ladner, R. E. (2018, April). Blocks4All: overcoming accessibilitybarriers to blocks programming for children with visual impairments. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (pp. 1-10).
4. Koushik, Varsha, and Clayton Lewis. “An Accessible Blocks Language: Work inProgress.” Proceedings of the 18th International ACM SIGACCESS Conference on Computers and Accessibility, Association for Computing Machinery, 2016, pp. 317–18. ACM Digital Library, doi:10.1145/2982142.2982150.
5. Google/Blockly-Experimental. 2019. Google, 2020. GitHub, https://github.com/google/blockly-experimental.
6. Lewis, C. (2014). Work in Progress Report: Nonvisual Visual Programing. In B.duBoulay and J.Good (Eds) Proc. PPIG 2014 Psychology of Programming Annual Conference, 25th Anniversary Event. Brighton, England, 25th -27th June 2014.
7. Kane, S. K., Morris, M. R., Perkins, A. Z., Wigdor, D., Ladner, R. E., & Wobbrock,J. O. (2011, October). 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 (pp. 273-282).
8. R. Dorsey, C.H. Park, A. Howard “Developing the Capabilities of Blind and VisuallyImpaired Youth to Build and Program Robots,” Journal on Technology and Persons with Disabilities, Vol. 1, pg. 57-69, 2014.
9. Brewster SA. Using non-speech sound to overcome information overload. Displays1997; 17: 179-189.
10. Murphy, E., Bates, E., and Fitzpatrick, D. Designing auditory cues to enhancespoken mathematics for visually impaired users. In Proceedings of the 12th International ACM SIGACCESS Conference on Computers and Accessibility, ASSETS ’10, ACM (New York, NY, USA, 2010), 75–82.
11. S. Ludi, J. Wang, K. Chapati et al., “Exploring the use of Auditory Cues to sonifyBlock-Based Programs”, Journal on Technology and Persons with Disabilities, Vol. 7, pg. 1-21, 2019.
12. Events — Blockly. Google Developers,https://developers.google.com/blockly/guides/configure/web/events, Accessed 10 Feb. 2021.
13. Open Roberta Lab. https://lab.open-roberta.org/. Accessed 10 Feb. 2021.