2019

THE JOURNAL ON TECHNOLOGY AND PERSONS WITH DISABILITIES

Exploring the Use of Auditory Cues to Sonify Block-Based Programs

Stephanie Ludi, Jeffrey Wang, Kavya Chapati, Zain Khoja, Alice Nguyen

Department of Computer Science and Engineering, University of North Texas

stephanie.ludi@unt.edu; jeffreywang@my.unt.edu; kavyachapati@my.unt.edu; zainkhoja7@gmail.com; aliceanguyen@gmail.com

Abstract

Visual programming languages are commonly used to help novice programmers learn introductory computer science, but accessibility is lacking. Most people who are blind use screen readers to convey the information displayed on the computer screen, but the keyboard support and the graphical blocks used in programs make block-based programming inaccessible. Source code is traditionally text-based, and thus can be conveyed by a screen reader when the user is programming in text editors or several accessible programming environments (e.g. Visual Studio, Eclipse). As part of our ongoing work to make block-based programming accessible to users with visual impairments, we explore the role that audio can have as cues to help users understand the structure of their code. In doing so we compare different types of audio cues and ascertain those which are useful and preferred by users. The participants conduct a set of tasks using a mock-up based on Google Pencil Code. Our initial feedback with three participants indicated that while speech could convey a programs contents, the use of earcons was preferred.

Keywords

Visual programming languages, accessibility, audio cues, block-based programming, visually impaired, blind, earcons, spearcons

Introduction

In computer science education, learning traditional text-based programming languages can be difficult for beginners, whether they are sighted or blind. Currently, sighted individuals also have the option to learn programming through block-based, visual programming languages (VPLs) such as Scratch (Wikipedia, 2014). The differences between the two types of programming paradigms lies in how programs are created and represented. In traditional text-based programming, languages require the programmer to construct a program via text and according to the language's syntax. VPLs in turn represent the language's components as blocks whereby the program is represented as a list-like structure of blocks, where each represents a command, structural component, or other programming construct. In order to add a block to a program, the user locates the desired block within a menu that is usually organized by block type. Additionally, the skeuomorphic representation of commands as colorful blocks (often resembling rectangles or polygons) helps sighted programmers distinguish different kinds of blocks from each other.

Block-based languages for blind or visually-impaired programmers may seem paradoxical, but the benefit of block-based languages is both sighted and visually impaired programmers – the programmer can focus on the semantics of the language rather than the syntax of the language (Ludi and Spencer, 2017). Also, block-based programming user interfaces and programs are more structured than conventional text-based programming languages. Thus, the more structured programming languages increases efficiency of navigating through programs (Baker, Milne, and Ladner, 2015). Therefore, block-based languages facilitate an easier programming experience in general due to the discrete, block-like structures, which are usually visually represented. The block-based structure's separation of information can allow

block-based languages to also be conveyed through audio (if the programming environment is designed correctly). In other words, block-based languages can allow programmers-in-training to learn the semantics of programming before they learn syntax, allowing them to focus on learning computer science concepts (Bau et al., 2015). Learning the semantics and paradigms of programming is more important for beginners because everything else in computer science stems from this. The primary feature of block-based languages reduces the need to learn syntax as it is encapsulated within the blocks, thus allowing programmers to find, drag, and drop the block into their program, as sequence of blocks. This eliminates the need to recall the syntax for a certain command, which is an essential aspect of all text-based programming languages. The missing piece is to design the features needed in the block-based programming tools to provide access to persons with visual impairments.

As blind programmers cannot view either the text or blocks, the need to audibly represent the content and structure of their programs exists. The use of screen reader persists in terms of user interface navigation and information conveyance. So far, the block-based environment we are basing our mock-up on is Pencil Code, which is open source (Pencil Code, 2015). Technically Pencil Code is a hybrid as the user can seamlessly move between a block and text representation (Bau et al, 2015; Weintrop and Holbert, 2017). The reason we chose Pencil Code as our target environment lies in its technical implementation. One of the most popular VPLs today is Scratch. However, it is currently implemented using inaccessible technologies, making screen readers are unable to access them (Resnick, et al, 2009). On the other hand, Pencil Code is used less than Scratch, but because it is written in HTML5, it is compatible with screen readers. Furthermore, its ability to switch between editing programs in blocks and text

(specifically, the Python programming language) assists learners in making a smooth transition from visual programming to traditional text-based programming (Weintrop and Holbert, 2017}.

Pencil Code is comprised of three different sections. The first section is block selection, which is where a user can choose blocks to add to their program. Each block represents a certain command. For example, a user wishes to use the command “move forward.” The second section is the program, where each “line of code” (in text-based programming language terms) is a code block, and a block represents a certain command. If the user added the “move forward” block to the program, then a turtle would move forward. This turtle indicates the position of the program on the output grid, which is the third section of Pencil Code. Its origin is at the middle of the grid. The turtle lies on the grid and starts at the origin facing upwards. It can be controlled with various commands from the program that make the turtle move around the grid, wear something, change in size, or draw something with a pen as it moves. This grid is two dimensional. Having the pen enabled at the start of a program can allow the movement of the turtle to draw various geometric shapes in different colors and different thicknesses. While the scope of this paper uses the general design layout of Pencil Code in a mock-up that simulates a block-based environment, future work will focus on other user experience aspects (e.g. navigation, keyboard support).

The scope of this paper concerns the role that auditory cues that can serve to help in the understanding of the structure of code via code navigation – a persistent challenge to persons who are blind (Albusays, Ludi and Huenerfauth, 2017). Three different types of auditory cues were compared: speech (from the screen reader), earcons, abstract musical sounds; and spearcons, sped-up speech. Our research questions about the effectiveness of these auditory cues are:

1. What is the most effective usage of each type of auditory cue, in this experiment speech, earcons, and spearcons, in code traversal? Which one is most intuitive? Most pleasing?
2. What is the most effective auditory cue for block identification?
3. What is the optimal placement of auditory cues to facilitate navigation of block-based code?

Related Work

Two areas of interest in this paper relate to the use of audio cues in user interfaces, as well as in the task of programming. In the early stages of earcons, Brewster, et. al investigated the effectiveness of each individual property of sound in order to determine which properties were most distinguishable from each other. They determined that differing musical timbres were more effective than varying simple tones. However, there was no statistical difference between the effectiveness of musical earcons compared to "simple" earcons (i.e. sine waves, square waves, sawtooth, and a combination of the three). Furthermore, musicians were no better at recognizing different timbres than non-musicians. However, this experiment reveals that musicians were indeed better at recognizing pitch than non-musicians. The author attributes musicians' pitch training to their higher accuracy in pitch recognition. Therefore, to create earcons accessible to everybody, they should have differing timbres.

The structure of code can be likened to a hierarchy of commands and related structures. The Yalla, Pavani and Walker study explores how visual menus are currently designed and function, then extrapolating proposed methods to represent these menus through auditory cues. The two main menu types are simple and hierarchical: the former would simply be a listing of choices (items), while the latter is a tree-like structure and has multiple levels (depth) and items

at each level (breadth). For visual hierarchical menus, the optimal balance of depth versus breadth depends on the screen size of the screen used to display the menu. To help make menus more accessible, this paper initially explores systematic ordering. Alphabetical ordering is the most intuitive, while ordering by frequency is another plausible option. However, beyond order, there are bigger issues associated with traversal of menus: feedback of where they are and faster text-to-speech of the menu's content are needed. Two earcon schemes are proposed to determine traversal of a specific level. The first is to incrementally increase the pitch as menu traversal proceeds. The second uses two beeps per item hovered over, one for the location and another reference beep for the last beep; as the first beep sounds increasingly similar to the other, the traversal of a certain menu is getting closer to the end. For depth traversal, there are three proposed earcon methods. The first proposal has the top level playing one sound, with each level below that playing its unique sound added to the top level's sound, and so on. The second proposal is similar to the first, except each level depth plays the same sound. The third proposal retains unique sounds for every single level but does not combine previous levels' sounds like the first. These varied proposals offer a variety of choices for optimizing accessibility of menu usability. We considered these options as we designed our audio cues.

As our team's project seeks to improve the coding skills of blind and visually-impaired users via audio cues, the study by Murphy, Bates and Fitzpatrick explored how auditory displays assisted users in improving the understanding of mathematics (which can also be complex and require an understanding the whole and the component parts). In the study, 56 users took a survey in which they heard auditory cues and presented alternatives to these sounds based on the action associated with the sound. This study also attempted to use auditory cues that were not speech-related simply because speech-related cues are known to be a larger burden on the user,

who must process the speech information before recognizing the action. The study utilized sounds to represent brackets (a short beep-like earcon with glissandos to symbolize opening and closing), fractions (spearcon ‘frac’ played from left to indicate beginning and right to indicate end), and superscripts/subscripts (spearcon ‘sup’ manipulated by changes in bass and treble frequency). Results showed that visually-impaired persons were significantly more likely to be able to “read” the equations with the auditory cues that the sighted group. Additionally, words “begin” and “fraction” had high recognition rates while the word “end” was confused with “in” and “and”. “Superscript” was also confused with “pseudo-script” and “subscript,” indicating that using spearcons for similar words may not be the best choice when using auditory displays in mathematics. In order to sonify mathematical notation, Murphy, et. al used a combination of novel auditory cues, spearcons, and binaural spatialization were created and used. Of particular note is their use of brackets. An opening bracket was represented by an upwards glissando in the left ear and a closing bracket was represented by the opposite, a downwards glissando in the right ear. This utilization of binaural spatialization was especially effective among participants (70% recognition) because it was intuitive. The spearcon “frac” was used to denote a fraction. A binaurally spatialized spearcon were used for superscript and subscript: “sup” and “sub”, respectively. However, these spearcons were confused with other words and recognition was only at 50%. Perhaps a less ambiguous choice of spearcons would have increased recognition. However, binaural spatialization, when intuitively implemented, is very effective.

Stefik, Hundhausen, and Smith et al. presented work on a tool called Sodbeans based on NetBeans IDE for (text-based) Java programming, to help convey certain information to students who are blind. Sodbeans includes a custom screen reader, talking debugger and programming language called Hop. The tool used audible cues to convey programming concepts to blind

students. These cues were designed to be browsed in a hierarchical tree manner, to support navigation. In a study by Stefik, Hundhausen, and Patterson, the usability of audio cues was explored to show the lexical scoping relations between textual coding statements in a program. The relationship between statements was found to be dynamic, and different cues were played when a change in scope was detected. The cues were short the end goal was to integrate them into a larger programming tool via an assistive plug-in. Our work expands on their work in audio cues but explores applicability in block-based programming.

Simulated Block-Based Environment's Design

A low-fidelity mock-up based on Pencil Code was created to allow for feedback of the use of auditory cues with sample blocks, as shown in Figure 1. Our mock-up emulates the layout of Google's Pencil Code as closely as possible. We used Twitter Bootstrap for structuring the layout and implementing responsive design into our prototype. The left side of the screen is for block selection, while the right side is where code navigation occurs. Block selection is comprised of two menus: categories and blocks. Pencil Code organizes each block into eight categories. Only one category's blocks are listed at a time. To select a block, a user would click on that block's category and then find their block within the second menu. The visual user interface was intentionally plain as the audio perspective was the focus of the experiment. This approach is due to the fact that Pencil Code itself is not accessible so the considerable effort involved to facilitate that would not be done in time for this study.

Three types of auditory cues for potential inclusion in the study were devised: speech, earcons, and spearcons. The control is the speech-generated by the participants' screen readers, since participants will be familiar with listening to information via screen reader for this experiment. Earcons were created by team members via Apple's GarageBand. An earcon was

designed to represent ‘identify block’ for the purpose of working with blocks in the workspace, as well as in the block selection menu. Earcons were created for identify category, identify block, select category, select block, open nesting, and close nesting. The Studio percussion sound kit in GarageBand was used for the first four, while a digital piano in GarageBand was used for the last two. While each of the identification earcons were comprised of one note, the select category earcon had two notes and the select block earcon had three notes. Earcons to represent the nesting of code utilized binaural spatialization, which was implemented with Audacity. Binaural spatialization is the practice of selectively playing audio through a certain channel to convey meaning (Murphy, Bates and Fitzpatrick, 2010). In other words, playing audio through only the left ear or right ear. A goal is to utilize binaural spatialization in order to intuitively convey nesting information to the user if it is shown to be effective. The open nesting earcon was panned to the left, meaning it sounded on only the left audio channel, while the close nesting earcon was panned to the right. This may be intuitive because Western texts are written and read from left to right so the mental model would hold. Spearcons were created by feeding words into Apple's Speech Synthesis API and then saving these to AIFF files (later being converted to MP3 files) through an AppleScript program written by one of the authors.

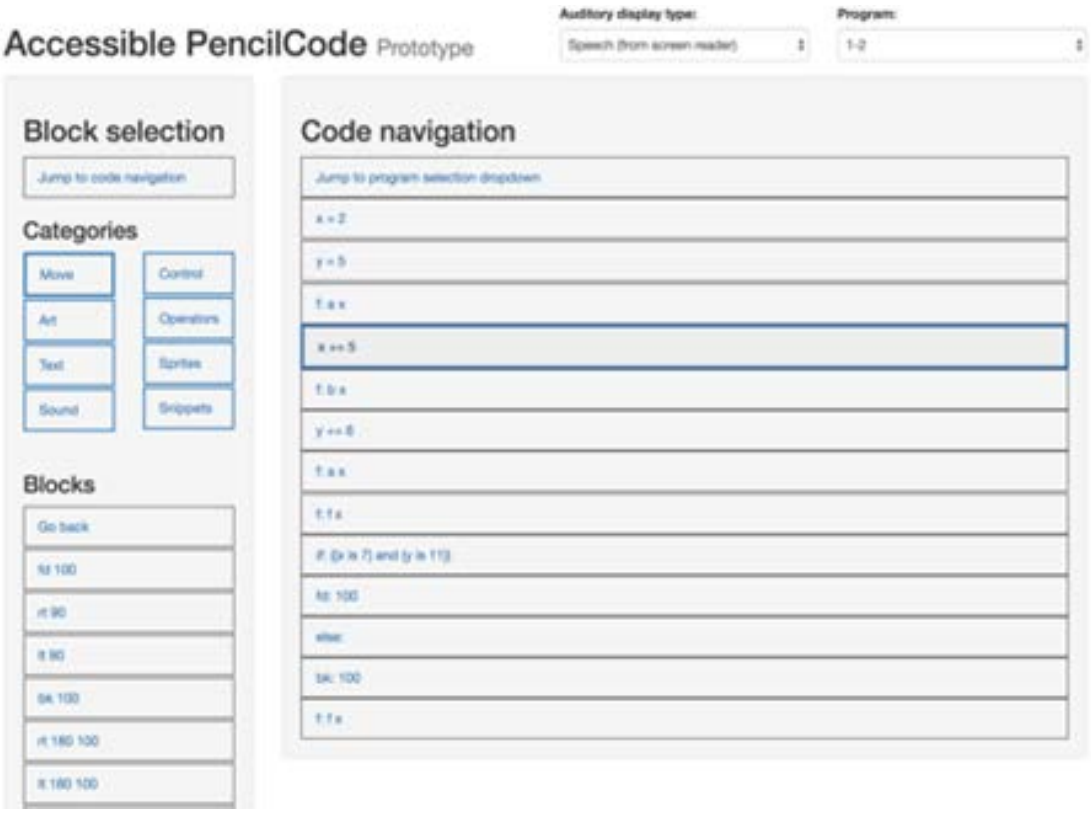

Fig. 1. Screenshot of Our Pencil Code-Based Mock-Up.

These auditory cues were then integrated into a mockup of Pencil Code, shown in Figure 1. The mock-up implements the block selection and code navigation sections of Pencil Code, but does not include the output grid because its output would not be able to be sonified to blind or visually-impaired programmers. Instead, we intend to have programmers imagine the changes made by the program to the grid for this experiment.

Code navigation is comprised of the source code for a program, which in Pencil Code is an ordered listing of each constituent code block. To navigate the code, the participant focuses onto the first code block and advances through the program by pressing tab. The focused element has a thicker border to clearly show where the focus is on. Each time the focus changes to another block, the newly-focused block is sonified through one of the three auditory cues. For speech and spearcons, the name of the block and its parameter's values are sonified. Since it

would be unfeasible to assign a unique sound to every single block, the identify block sound is played when the selected auditory cue is earcon, and in order to hear the block name, the user presses the slash (/) key on their keyboard to hear the description of the block through speech. To hear the nesting level of the currently-focused block, the user would press the period (.) key. (Nesting is a concept in computer science that applies to structures such as if statements, for and while loops, functions, etc.) Such a diverse array of sounds was then tested on the prototype to be evaluated for accuracy, efficiency, and likeability.

Our mock-up uses JavaScript, including the jQuery JavaScript library, and an audio handling library called Howler.js. Additionally, this prototype also uses the Web Speech API in a limited fashion. While the Web Speech API is experimental, it is supported by all recent versions of popular browsers Chrome, Firefox, and Safari, as of July 2017. To ensure screen reader support, this prototype makes use of WAI-ARIA specifications. Spearcons use pre-generated, sped-up audio files that represent a category name or block name. For command parameters, which are user-defined, they are sonified on the spot by the Web Speech API because it would not be feasible to pre-record the infinite possibilities of words and numbers.

Methodology and Design

The experiment was conducted remotely via Skype, with persons who were either blind or whose vision was low enough that a screen reader was used on a regular basis. One at a time, the participants performed tasks that would be needed to select blocks to add to a program, as well as to navigate an existing program. The code navigation tasks were completed one at a time, randomized by each exercise type.

At the onset of the experiment, our team conducted a sound check training exercise to confirm that the subject would be able to accurately hear the subsequent auditory cues. Also, we verbally provided the participants with a short briefing regarding the definition of speech, earcons, and spearcons, as well as instructions on how to navigate through the prototype website, so that the user could comprehend the model's basic functions before proceeding to the documented tasks.

In addition to our data logging and observations, the participants provided feedback via SurveyMonkey due to its accessibility for screen readers. At the beginning of each participant trial, we completed the initial checks of the participant using a compatible browser (Firefox, Chrome, or Safari), had stereo audio (two channels) and was wearing headphones, ensured that the participant was in a closed environment and had the survey and prototype website pulled up, and instructed the participant to enable screen sharing so that we may record their computer's screen and the participant's voice with QuickTime.

Each participant completed exercises 2 sets of code navigation exercises (hereafter referred to as Exercise 3 and 4). The first exercise consisted of the participant interaction with 3 source code files where each file used a different auditory cue. The 3 files were completed in a random order for each participant.

In the first set of exercises (Exercise 3), each program's contents was presented with either speech only, earcons + speech, or spearcons + speech. The sonification method was linked to each program and did not change per participant. In other words, the source code that used spearcons + speech for one participant was the same for the other participants – it was the order of the files that differed. The participants were asked to determine what each program does (their "perceived function"). For instance, a valid answer might be "this program drew a

triangle." The time needed for a participant to identify each participant's perceived function of the program was recorded to determine which auditory type was most effective in conveying the source code's contents as a means of assisting in the determination of the program's functionality. The perceived function was then compared to the actual function of the program to ascertain the accuracy, as well as determine whether the participant needed assistance. After all three programs were conducted, the participants were asked to rank the auditory methods according to the following criteria: from most useful to least useful, easiest to most difficult, and most to least pleasant, in order to determine which auditory icon was most intuitive and most pleasant for them.

In the second set of exercises (Exercise 4), the participants were given another set of three programs, one at a time, where each program was presented with one of the 3 auditory methods. This time, the participants were given a program to navigate through on their own but the participants were told what the program was supposed to do. In this set of exercises each program contained a bug. In each program, there was one code block in the program that, if changed, would allow the program to run as intended. Participants were asked to identify the erroneous block. We again recorded the time needed to identify this incorrect block and which block they identified. We then compare the block they identified as the problematic block to the actual problematic block. The participants' accuracy and time needed were used to determine how effectively they were able to identify the erroneous block with a specific auditory cue. After all three source code files were completed the participants were asked to rank the auditory methods from most useful to least useful, easiest to most difficult, and most to least pleasant.

After both sets of exercises, the participants were asked to offer feedback about the description of nesting levels within the program, whether the descriptions of the blocks or actions needed to change, if nesting should be conveyed by speech or earcons, the pleasantness of the musical timbres used to create the earcons, the overall quality of the audio, and how it should be changed.

Results and Discussion

Three participants with varying degrees of traditional (text-based) programming experience and visual impairment completed our experiment in about one hour and a half. One participant is visually impaired though used a screen reader. One participant is completely blind, and another participant is completely blind but was formerly sighted. Two participants use JAWS, Mozilla Firefox, and Windows, while another participant uses VoiceOver, Chrome, and Mac.

The participants first completed block selection exercises and a training task before beginning the two code navigation sets of exercises. Block comprehension for both exercises 3 and 4 was very good, as all participants were able to interpret the given cues as blocks. In exercise 3, while all participants were able to identify the program sonified with speech, only 1 participant was able to correctly identify the program sonified with earcons and no one was able to identify the program sonified with spearcons. We suspect this is because the programs created for earcons and spearcons incorporated the concept of functions, which was not immediately intuitive to the programmers.

Exercise set 2 (Exercise 4) investigated how effectively the audio cues could be used for troubleshooting bugs in code. In each program, 2 of the 3 participants correctly identified the problematic block that was causing the given program to deviate from its intended functionality.

The failed identification may stem from two reasons: these programmers may not have extensive troubleshooting experience and these programs were not written by the participants.

The participants offered mixed opinions over how each auditory cue should sound. However, their performance shows that they are able to use each auditory cue quite well. Our participants found earcons were the most useful throughout both exercises, followed by speech and then spearcons. Two participants chose earcons as the most useful auditory cue (Figures 3 and 6), while all three participants chose earcons as the most pleasant auditory cue out of the three (Figures 4 and 7).

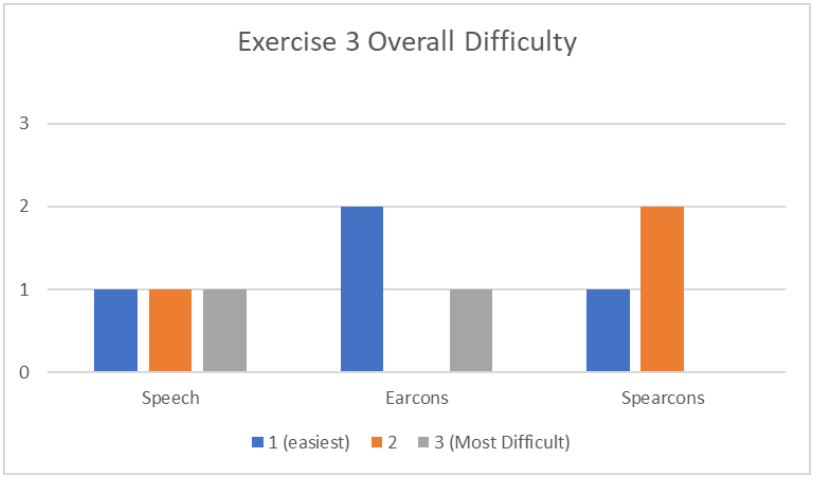

Fig. 2. Perceived Overall Difficulty of Each Auditory Cue as Presented in Exercise 3.

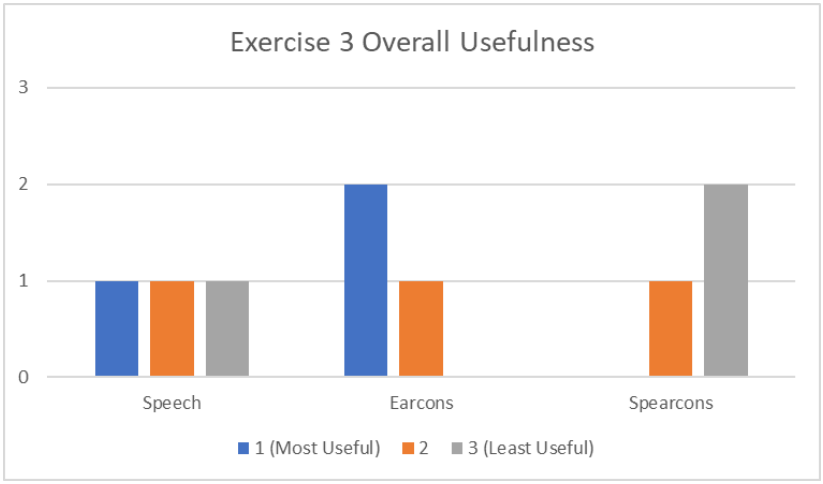

Fig. 3. Perceived Overall Usefulness of Each Auditory Cue as Presented in Exercise 3.

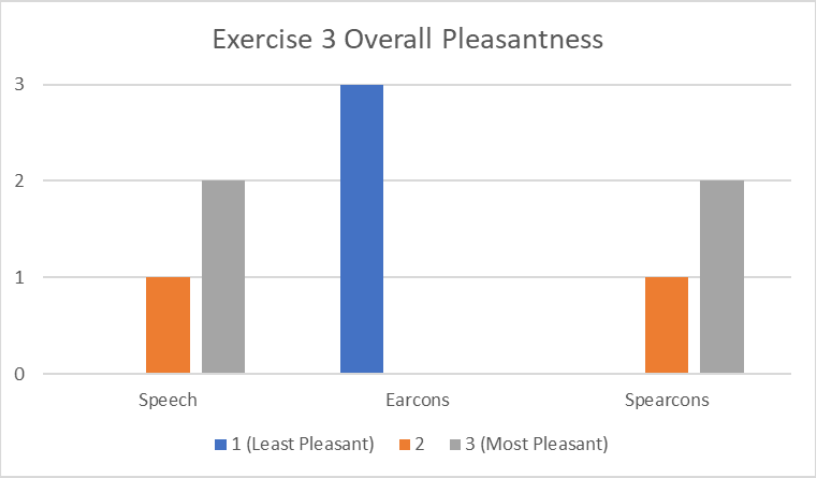

Fig. 4. Perceived Overall Pleasantness of Each Auditory Cue as Presented in Exercise 3.

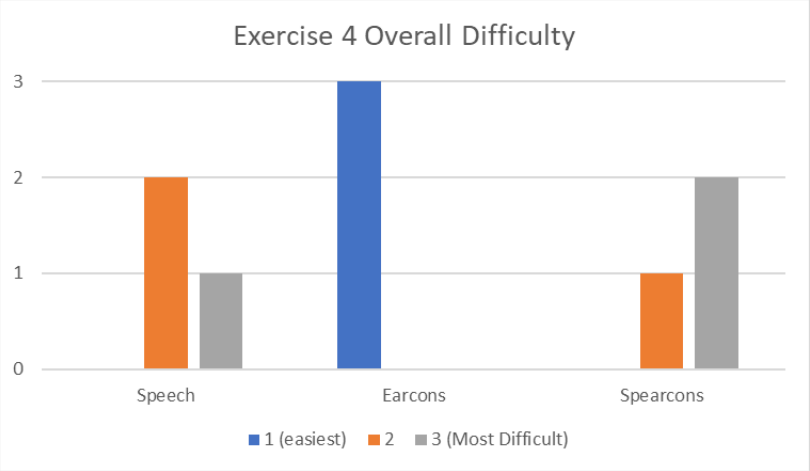

Fig. 5. Perceived Overall Difficulty of Auditory Cues as Presented in Exercise 4.

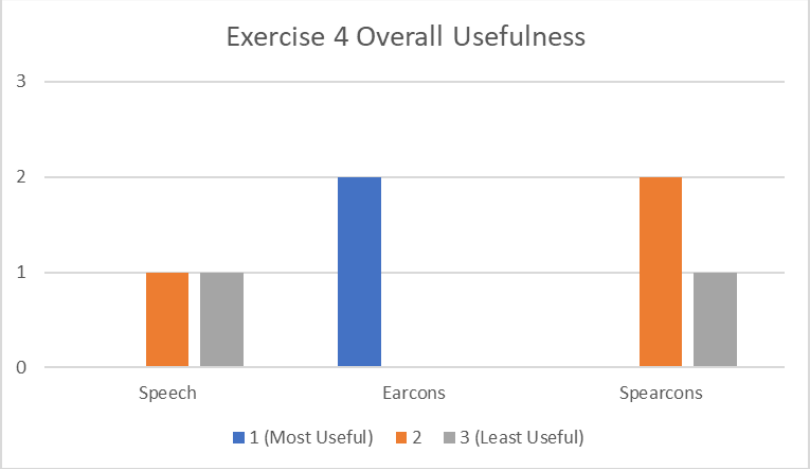

Fig. 6. Perceived Overall Usefulness of Each Auditory Cues as Presented in Exercise 4.

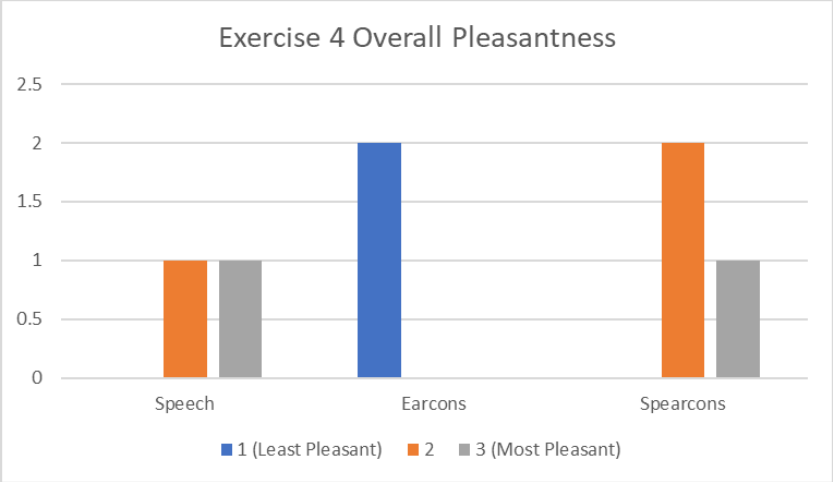

Fig. 7. Perceived Overall Pleasantness of Each Auditory Cue as Presented in Exercise 4.

We inquired about the intuitiveness of the nesting earcons' binaural spatialization. Two participants remarked that they liked it, but they did not realize the nesting earcons were binaurally spatialized until we surveyed them after the exercises were completed. So the earcons were useful but the binaural spatialization was not apparent. Each program was also interpreted with varying accuracies. In program 3-2, which included function definitions and calls, accuracy sharply decreased. We believe this may have to do with the complexity of the code since not all of the participants were familiar with the concept.

Conclusion and Future Work

The experiment has shown promising results: all of our participants were able to identify blocks through speech as well as with earcons and spearcons as auditory cues. Also, most participants noted that earcons are desirable as well. The study needs to be expanded to include a more refined set of source code samples, as well as the need to increase the number of participants. The small sample size is a limitation of the study, but as the study will be replicated as effort on the project continues.

In the future, further developments of these auditory cues can be applied to the actual Pencil Code visual programming environment, allowing for us to bring visual programming languages to blind and visually-impaired programmers. Previously, only sighted programmers could take advantage of the powerful pedagogical potential that visual programming languages offer. With the advancement of these auditory cues, they can later be implemented into Pencil Code so that blind and visually-impaired individuals are afforded the same educational tools with which sighted programmers facilitate their advancement of knowledge.

Works Cited

Albusays, Khaled, Ludi, Stephanie, and Huenerfauth, Matt. 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). ACM, New York, NY, USA, 91-100. DOI: https://doi.org/10.1145/3132525.3132550

Baker, Catherine M., Milne, Lauren R., and Ladner, Richard E., 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). ACM, New York, NY, USA, 3043-3052. DOI: https://doi.org/10.1145/2702123.2702589

Bau, David. et al., 2015. Pencil Code: Block Code for a Text World. Proceedings of the 14th International Conference on Interaction Design and Children (New York, NY, US), 445–448.

Brewster, Stephen A., Wright, Peter C., Edwards, Alistair D. N., 1992. A detailed investigation into the effectiveness of earcons. In Proceedings of ICAD '92 (Santa Fe Institute, Santa Fe) Addison-Wesley. Addison-Wesley, Boston, MA, USA, 471-498. DOI=10.1.1.65.7635

Ludi, Stephanie and Spencer, Mary. 2017, Design considerations to increase Blockly-based Language Accessibility for Blind Programmers Via Blockly, Journal of Visual Languages and Sentient Systems, Volume 3, DOI: reference number: 10.18293 , P. 119-124

Murphy, E., Bates, E., and Fitzpatrick, D., 2010. Designing auditory cues to enhance spoken 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, 75- 82. DOI=10.1145/1878803.1878819

Pencil Code, Pencil Code Homepage. Retrieved on July 2016. Available: https://pencilcode.net/

Resnick, Mitchel. et al., 2009. Scratch: Programming for All. Communications of the ACM. 52, 11, 60.

Stefik, Andreas, Hundhausen, Christopher, and Smith, Derrick. 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). ACM, New York, NY, USA, 571-576. DOI=http://dx.doi.org/10.1145/1953163.1953323

Stefik, Andreas, Hundhausen, Christopher, and Robert. "An Empirical Investigation into the Design of Auditory Cues to Enhance Computer Program Comprehension." The International Journal of Human-Computer Studies, vol. 69, pp. 820-838, 2011.

Weintrop, David and Holbert, Nathan, 2017. From Blocks to Text and Back: Programming Patterns in a Dual-Modality Environment. In Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17). ACM, New York, NY, USA, 633-638. DOI: https://doi.org/10.1145/3017680.3017707

Wikimedia. Screenshot from Scratch 2.0 showing a Hello World solution, 2014. Retrieved August 10, 2017, from Wikimedia Commons: https://commons.wikimedia.org/wiki/File:Scratch_2.0_Screen_Hello_World.png

Yalla, Pavani, and Walker, Bruce N., 2007. Advanced Auditory Menus. Georgia Institute of Technology.