International Journal of Computer Science Education in Schools, April 2018, Vol. 2, No. 2
ISSN 2513-8359

Making Programming Accessible to Learners with Visual Impairments: A Literature Review

Alex Hadwen-Bennett[1]
Sue Sentance[1]
Cecily Morrison[2]

[1]King's College London
[2]Microsoft Research Cambridge

DOI: 10.21585/ijcses.v2i2.25

Abstract

Programming can be challenging to learn, and for visually impaired (VI) learners, there are numerous additional barriers to the learning process. Many modern programming environments are inaccessible to VI learners, being difficult or impossible to interface with using a screen reader. A review of the literature has identified a number of strategies that have been employed in the quest to make learning to program accessible to VI learners. These can be broadly divided into the following categories; auditory and haptic feedback, making text-based languages (TBLs) accessible, making block-based languages (BBLs) accessible and physical artefacts. A common theme among the literature is the difficulty VI learners have in gaining an understanding of the overall structure of their code. Much of the research carried out in this space to date focuses on the evaluation of interventions aimed at VI high-school and undergraduate students, with limited attention given to the learning processes of VI learners. Additionally, the majority of the research deals with TBLs, this is despite the fact that most introductory programming courses for primary learners use BBLs. Therefore, further research is urgently needed to investigate potential strategies for introducing VI children in primary education to programming and the learning processes involved.

Keywords: visual impairments, programming education, physical programming, special needs

1. Introduction

The introduction of computing into the national curriculum for England in 2014, brought with it the requirement for primary school children to be taught the basic concepts of programming from the age of 5 (Department for Education, 2014). Programming can be challenging to learn and, for visually impaired (VI), learners there are numerous additional barriers to the learning process. Many modern programming environments are inaccessible to VI learners, being challenging or impossible to interface with using a screen reader (Baker et al., 2015; Stefik et al., 2011) and user interfaces often employ highly graphical depictions (Ludi, 2013). Kane & Bigham (2014) identified the following criteria for the development of environments in which VI children can learn to program:

- "Programming tools must be accessible to the student and must work with the assistive technology that he or she uses."
- "The student must be provided with programming tasks that hold their interest and provide encouraging feedback." (Kane & Bigham, 2014, p. 257).

This literature review sets out to provide an overview and discussion of the different strategies that have been employed in order to make learning programming accessible to VI learners. Additionally, areas that require further research will be identified and discussed.

2. Methodology

This review examines literature from peer-reviewed sources, published between 2000 and November 2017. Studies were identified by searching research databases, in addition to citation tracing. The following databases

were searched: ACM Digital Library, Taylor and Francis, IEEE, Eric and Wiley Online Library. The search terms "visually impaired", "programming" and "education" were initially used, followed by additional searches employing alternative search terms with similar meanings, an overview of these terms is given in Table 1.

Search Term	Alternatives
Visually Impaired	Blind, visual impairments
Programming	Coding, software development
Education	Learning, learners, school

Table 1 Summary of Search Terms

Once a short list of articles was formed, the following criteria were used to decide whether the articles should be included in the literature review:

- Papers were included if they had an educational focus, however a small number of other papers were retained in order to provide contextual information.
- Papers were included if they were included in a peer-reviewed academic publication.
- Papers published since 2000 were included. One exception was made for a paper that is frequently cited and therefore provides contextual information.

Upon further examination of the literature, four main themes emerged; making text-based languages accessible, making block-based languages accessible, physical artefacts as well as auditory and haptic feedback. Each of these themes is explored in turn in the following sub-sections. An overview of the literature cross-referenced by theme is also provided in appendix A.

3. Overview of Literature

3.1 Making Text-Based Languages Accessible

3.1.1 Accessibility of Programming Environments

A survey of experienced VI developers has demonstrated that many programming environments are either not fully compatible with screen readers or challenging to navigate solely using auditory feedback alone, this makes them inaccessible to many VI programmers (Albusays & Ludi, 2016). For example Eclipse features a number of tabbed windows, which can be accessed through keyboard shortcuts, however this is a time consuming process when relying on auditory feedback (Cheong, 2010). Additionally, the BricxCC and Robot C programming environments, which are both designed for programming Lego Mindstorms robots, are not fully compatible with JAWS (a popular screen reader) (Ludi, 2013). Although Visual Studio (2010) is technically accessible, no sound is generated to indicate when the user switches between tabs (Stefik et al., 2011).

One approach that has been taken to address the inaccessibility of programming environments is the use of a standard text editor alongside a screen reader (Bigham et al., 2008; Cheong, 2010; Kane & Bigham, 2014). A drawback of this approach is the loss of debugging tools that are standard in most modern programming environments. Tools have also been developed to improve the accessibility of programming environments, for example the Wicked Audio Debugger (WAD) was developed to work with the popular Visual Studio programming environment to assist VI programmers with the debugging process (Stefik et al., 2007).

An alternative strategy is the development of accessible programming environments. An example is JavaSpeak, which was developed as a tool to assist VI undergraduate students learn how to program in Java (Francioni & Smith, 2002; Smith et al., 2000). It is based on the concept of EmacSpeak (Raman, 1996), which has a speech interface aimed at experienced programmers. Unlike EmacSpeak, JavaSpeak is designed for undergraduate students that are learning to program, enabling them to experience their code at different granularities. The development process of the JavaSpeak environment has been described, however there is no evidence of evaluation of the tool in use.

More recently, the JBrick programming environment was developed to make the programming of Lego Mindstorms robots accessible (Ludi, 2013). The NXC language (Not eXactly C) has been used in outreach programs along with the BricxCC programming environment to enable VI learners to program Lego Mindstorms robots (Dorsey et al., 2014; Ludi & Reichlmayr, 2011). However, the BricxCC programming environment is not

fully compatible with JAWS (a popular screen reader). JBrick was developed as an alternative to BricxCC, it is compatible with common screen readers and braille displays, enables code to be easily located by line number and provides both audio and visual feedback (Ludi et al., 2014).

3.1.2 Accessibility of Programming Languages

Another important consideration is the choice of programming language; many commonly used languages, such as C and Java, make extensive use of non-alphanumeric characters such as brackets and curly braces, which can be challenging to work with using a screen reader. Additionally, the complex syntax of many languages can make typing mistakes more likely and debugging more challenging. Languages such as Ruby, which use mainly text and limit the number of non-alphanumeric symbols are preferable as they are less likely to cause problems with screen readers (Kane & Bigham, 2014). In their study, Kane and Bigham also considered Python, as it meets most of the previously mentioned criteria, however it also uses white space which could be confusing when used with a screen reader. During the course of their study, which took place over a week and involved 12 VI learners, Kane and Bigham found that the students were successful in writing programs in Ruby, however the mispronunciation of some of the terms by the screen reader caused minor challenges.

There are text-based languages that have been designed specifically for VI users, for example the APL (Audio Programming Language) for example, was developed by VI learners for VI learners (Sánchez & Aguayo, 2006). APL features a reduced set of commands which can be accessed and selected through a circular command list, with no requirement to memorise commands. The results of a small usability study of APL indicate that the language enables learners to understand programming concepts and apply them.

In 2011, Stefik et al. conducted an exploratory study to evaluate the accessible programming environment Sodbeans, along with the Hop programming language, which they developed. Sodbeans is aimed at middle and high school students and makes use of audio cues for navigation along with an auditory debugger for the Hop programming language. The findings from the evaluation indicate an increase in learner self-efficacy after participation in a programming workshop that employed Sodbeans and Hop.

The Hop programming language was developed further, becoming Quorum, a language designed for all, while still being accessible to VI learners (Stefik et al., 2011). The development of Quorum was informed by empirical studies investigating the intuitiveness of the syntax of different languages and the accuracy rates of novice programmers using them (Stefik & Siebert, 2013).

3.1.3 Code Navigation

A common theme that occurs among the literature is the difficulty VI learners have navigating their code and understanding the overall structure when using a screen reader (Bigham et al., 2008; Kane & Bigham, 2014; Ludi et al., 2014). This can often result in learners inserting code in the incorrect position. There are steps that can be taken to mitigate these difficulties; in order to gain a better understanding of their position in the code, learners can be encouraged to move the text cursor in order to hear the characters read out. In addition, learners can also be provided with code samples in braille to help them develop an understanding of the overall structure of the code.

The challenge of navigating the code and understanding its structure was considered during the development of StructJumper, a plugin for the Eclipse programming environment which enables VI users to navigate through a program written in Java (Baker et al., 2015). StructJumper generates a tree that is made up of the nested structures contained within the program, this enables the user to easily jump between each nested structure in the code. The participants that took part in a small-scale evaluation of StructJumper found that it helped them speed up their navigation through the code.

3.1.4 Other Considerations

It is also important to consider that the level of vision among VI learners will vary considerably, as will their preferred assistive technologies (Bigham et al., 2008; Ludi et al., 2014). Experience with assistive technologies may also vary. Bigham et al. (2008) found that students that were already proficient in the use of a screen reader were the most successful. Another factor that can impact on progress of VI learners is their familiarity with keyboard layout, with typing skills also being identified as an important skill for learning to program in a text-based language (Ludi, 2013; Ludi et al., 2014).

Another factor to be considered is the accessibility of tools designed to create graphical user interfaces (GUIs), as existing tools that are employed to generate GUIs are either not accessible or very challenging to use for VI learners. In order to address this issue Siegfried (2006) developed a scripting language to enable VI programmers to produce Visual Basic Forms. More recently, Konecki (2014) developed GUIDL, a tool that enables VI learners to create GUIs for their programming projects. GUIDL was evaluated by a small group of adult novice programmers who found they were able to use the tool to successfully create GUIs that could be used in their own programs.

Although there are a number of studies focusing on teaching VI learners to program in a text-based language (TBL), these mainly focus on high school and undergraduate students. The following section will look at the accessibility of (BBLs), which are targeted at students in primary school.

3.2 Making Block-Based Languages Accessible

When learning how to program a significant amount of time is spent learning the syntax of a specific language; this can potentially hinder the development of an understanding of the core programming concepts. BBLs such as Scratch (Maloney et al., 2010) enable learners to develop programs by snapping blocks together, removing the need for them to learn the complex syntax of a TBL.

BBLs are intrinsically visual and are therefore not accessible to most VI learners. There is a need for an alternative to BBLs such as Scratch (Koushik & Lewis, 2016; Ludi, 2015). One such alternative is Noodle, a programming system for creating sound and music that has program elements which can be inserted and arranged purely using keyboard commands (Lewis, 2014). The concept of Noodle is promising; however, it does not appear to have been trialed with learners and the language used in the audio feedback is not appropriate for primary school children. This makes it an unsuitable choice for the introduction of programming to young VI children.

Ludi (2015) and her team have been working on making the Blockly language accessible to VI learners. The language that Ludi and her team are developing will enable navigation purely by keyboard and also incorporate audio cues in order to communicate the level of nesting. Following on from the work on Noodle, Lewis has been working with Koushik in the development of another accessible Blockly-based language called the Pseudospatial Blocks (PB) language (Koushik & Lewis, 2016). Pseudospatial refers to the distorted nature of the geometry of movement. In PB the learner selects an insertion point using the keyboard and they can select the program element they want from a filtered list; the program elements are filtered by syntactic category. Koushik and Lewis (2016) argue that PB has advantages over visual languages for all learners as invalid program blocks for a given space are filtered out.

The Lady Beetle and World of Sounds programming environments are alternatives to BBLs that were developed in order to introduce young VI children to the basic concepts of programming (Jašková & Kaliaková, 2014). The Lady Beetle programming environment enables the learner to select single word commands, without having to type them. These commands control the movement of a beetle across a grid. As the beetle moves, the coordinates of the current square are read out. World of Sounds, on the other hand, enables learners to create simple programs that produce sequences of sounds.

The development of these accessible BBL alternatives is a promising step forward in the quest to find an accessible alternative to block-based languages, however they could still present learners with difficulties gaining an understanding of the overall structure of their code when using a screen reader. The table shown in appendix A demonstrates that there is still some way to go for BBL research to catch up with TBLs.

3.3 Physical Artefacts

3.3.1 Programmable Devices

The physical nature of programmable devices such as robots make them a common tool for the teaching of introductory programming and it is has been shown to be just as appealing to VI learners (Ludi, 2013). When teaching computing with robotics, the robots can either be pre-assembled or learners can be required to build their own robots as part of the learning process. This has its own challenges, particularly for VI learners.

Dorsey Rayshun, Chung Hyuk, & Howard (2014) conducted an evaluation of four educational robotics kits during a series of summer workshops, which investigated their suitability for use with VI learners. In each workshop the VI learners were paired with a sighted buddy and tasked with building robots using the various kits.

The LEGO Mindstorm RCX was found to be the easiest for VI learners to work with, requiring the least support from their sighted buddies.

A number of studies have been conducted, which investigate outreach programs designed to increase participation of VI students in computing using robotics (Dorsey et al., 2014; Ludi, 2013; Ludi et al., 2014; Ludi & Reichlmayr, 2011). The findings of these studies indicate that after the workshops the confidence level of the students in programming improved, as did their desire to take computing in school or pursue it as a career.

3.3.2 Physical Programming Languages

Most systems used in physical computing, whilst being physical themselves are still programmed using a GUI on a computer. In physical programming languages (PPLs), commands are represented by physical objects which can be joined together to create programs. The Tern PPL uses wooden blocks that can be joined together in order to construct programs. A webcam is used to convert physical into digital code (Horn & Jacob, 2007a, 2007b). Tern was initially evaluated over the period of one week with nine sighted children. The children used Tern to program robots, not all of them were able to understand the effect of their programs on the robot. This may be partially down to the delay between code creation and execution as it has to be converted to digital code using a webcam connected to a computer.

The physical nature of physical programming languages means they have the potential to be a powerful learning tool for VI children, howeverTern itself is not accessible. On the other hand there is Torino, a physical programming language that is designed to be inclusive of VI learners (Thieme et al., 2017). Torino features pods which can be joined together to create programs that produce sound and music. Each pod features dials, which act as parameters and enable the learner to change the sound sample or note and the duration. The physical nature of Torino programs could potentially enable the learner to gain an overall of the structure of the whole program.

3.3.3 3D Models

It is common practice for computing teachers to use diagrams, graphics or animations to illustrate programming concepts such as data structures, “most tools used to teach data structures, algorithmic thinking and basic programming are visually oriented” (Papazafiropulos et al., p. 491). While assistive technologies enable VI learners to access information, they are unable to present a complex concept in a simple form in the same way a visual representation can.

3D models can be used to represent abstract concepts in a way that is accessible to VI learners. As part of their research Stefik et al. (2011) interviewed teachers in one school for VI children and found that where possible new concepts should be introduced through the use of physical objects. In response to this, they developed ‘manipulatives’ for teaching key programming concepts, such as variables. Jašková & Kaliaková (2014) used a tactile table consisting of a 10x10 grid to teach VI children how to write simple algorithms. The children were given the task to write a sequence of commands in a text editor that guided a bee to follow a pre-set path through the tactile grid. The learners would simulate the execution of the program by moving the bee with their hands.

With the advent of 3D printers, 3D models have become much easier to produce. Papazafiropulos et al. (2016) used 3D printed models in a small feasibility study to teach concepts such as data structures and algorithms to VI children. The model they used features cylinders of varying heights, with the height representing the value of the element. The cylinders slot into a tray which represents the array. It was used to teach how sorting and searching algorithms could be applied to arrays.

3D printing was also used by Kane & Bigham (2014) as part of a week-long programming workshop, in which children produced code to generate physical visualizations of data. They found that the ability to generate and print their own tactile maps was extremely engaging for the children, however, the speed of 3D printing was a limitation as they had to be printed overnight. They also identified the need for universal tools that can be used to easily create tactile graphics.

Lego provides a quick and simple method of producing basic 3D models for use in the teaching of programming concepts to VI learners. Capovilla et al. (2013) discovered this when they employed Lego models in the teaching of sorting and searching algorithms to a small group of adult VI learners. Once the learners had familiarized themselves with the algorithms using the Lego models, they were then asked to solve sorting and searching tasks in a spreadsheet. All participants were able to complete the assigned tasks.

3.4 Auditory and Haptic Feedback

Sounds that vary in tone and pitch can be used to indicate the different states of a physical object or virtual representation, as can haptic feedback in the form of vibrations. PLUMB EXTRA (EXploring data sTRuctures using Audible Algorithm Animation) was developed to enable VI undergraduate students to access simulations of algorithms designed to manipulate data structures (Calder et al., 2007). It is based on PLUMB, a system designed to enable VI learners navigate graphs (Calder et al., 2006). The PLUMB EXTRA system enables learners to explore the state of data structures at any point using a series of audio cues. In the Calder et al. (2007) study, the development of the system is described; however, the evaluation of the system is limited.

During a series of workshops, Dorsey et al. (2014) made use of different piano notes and vibrations in a Wii remote in order to indicate the different states of a robot while navigating a maze. The results are this study indicate that if sufficient haptic and auditory feedback is provided, VI learners are able to perform tasks that are considered to be highly visual.

4. Discussion

This review has demonstrated the dominance of TBLs in the literature, this is despite the fact that in primary computing education BBLs are most prevalent, as highlighted by the recent Royal Society Report (The Royal Society, 2017). According to the national curriculum (Department for Education, 2014), all children in England should learn the basic concepts of programming from the age of 5. However, the inherent inaccessibility of BBLs, along with their widespread use in primary computing lessons can lead to VI learners being excluded from programming lessons. Initial steps have been taken towards making BBLs accessible to VI learners, however there is still a long way to go and more research is needed.

Research relating to the use of TBLs with VI learners has identified the difficulty learners can have in gaining an understanding of the overall structure of their code as can they only listen to one line of code at a time, putting a heavy reliance on short term memory. Even though it has been shown that it is possible to make BBLs accessible to VI learners, this difficulty could still present a barrier for learners. PPLs, on the other hand, could potentially enable VI learners to develop an understanding of the structure of the code through touch, as long as the individual blocks or elements used in the PPL are physically different. Therefore, the use of PPLs with VI learners needs to be investigated in terms of learning processes and possible benefits.

The literature relating to TBLs has identified a number of potential challenges for VI learners in addition to possible strategies to overcome them. This research can be used to inform the teaching of programming to high-school VI learners, however more research is still required. If VI learners are successfully introduced to programming in primary school through PPLs or accessible BBLs, they will enter high-school understanding the basic concepts. This could potentially smooth the transition to TBLs and as a result possibly reduce the significance of some of the challenges currently associated with TBLs. This highlights the urgent need for research into strategies for making programming accessible to primary VI learners.

5. Conclusion

Much of the research carried out in this space to date focuses on the development of interventions and their impact on student perceptions and engagement, with limited attention given to the pedagogy of teaching programming to VI learners. This is certainly an area that warrants further research.

Currently the most popular languages for introductory programming in primary schools in the UK are block-based (The Royal Society, 2017), which are currently not accessible to VI learners. Therefore, there is a need for further investigation into potential accessible alternatives to BBLs, PPLs are a promising candidate given their potential to enable learners to gain an understanding of the overall structure of their code.

6. Summary

A range of studies have investigated ways in which learning text-based languages can be made accessible to VI learners (Bigham et al., 2008; Dorsey et al., 2014; Kane & Bigham, 2014; Ludi, 2013; Ludi et al., 2014; Ludi & Reichlmayr, 2011; Smith et al., 2000; Stefik et al., 2011), however, these have focused mainly on high school

and undergraduate students. Block-based languages have also been examined, with the aim of making them accessible to VI learners (Koushik & Lewis, 2016; Lewis, 2014). Pseudospacial Blocks (PB) is a promising development, which is more suited to the needs of VI learners in primary education. It should be noted however, that it could be challenging for learners to gain an understanding of the overall structure of their code when using PB, as is the case with text-based languages.

Physical artefacts can be employed to engage sighted and VI learners alike, the use robotics is one such example (Dorsey et al., 2014; Ludi, 2013; Ludi et al., 2014; Ludi & Reichlmayr, 2011). The drawback of this approach is that it currently still relies on TBLs, bringing with them their own complications, which have based discussed previously. PPLs, on the other hand have the potential to be a powerful tool in the teaching of programming to VI learners in primary education, combining the physical with the facility to gain an understanding of the overall structure of a program.

3D models (Kane & Bigham, 2014; Papazafiropulos et al., 2016; Stefik et al., 2011) along with auditory and haptic feedback (Calder et al., 2007; Dorsey et al., 2014) have been shown to be useful aids in the teaching process, however they cannot be used to teach programming in isolation and need to be combined with other strategies.

7. Guidelines

Drawing on the literature, a set of guidelines has been produced for educators and developers working with VI learners. It should be noted, however that these guidelines are based on the literature that is currently available and may change as the field develops and more evidence is gathered.

1. Accessible physical programming languages may be a suitable alternative to block-based languages when introducing young VI children to programming.
2. Simple programming concepts can be taught to young VI children using 3D artefacts, for example writing an algorithm to move a bee in a tactile grid.
3. When teaching with text-based programming languages, the choice of language is important. Either choose a language that is specially designed for VI learners, or a general-purpose language with simple syntax and limited use of non-alphanumeric characters, for example Ruby.
4. Ensure you choose a programming environment that is fully accessible and easy to navigate using a screen reader. If an appropriate environment is not available, a plain text editor can be used, although the lack of debugging tools can be challenge.
5. Abstract concepts that are usually taught using visual representations can often be effectively taught to VI learners using 3D artefacts. For example, teaching data structures using different sized cylinders that slot into a tray.
6. VI learners often struggle to gain an overall understanding of the structure of code written in text-based languages, one support strategy is to provide example code in Braille (for braillists).
7. Choosing an appropriate theme for programming activities can make them accessible and engaging for VI learners. For example, tasks that involve programming a physical device, such as a robot can be very engaging. However, it is important to provide positional information for the robot in non-visual forms, this can include the use of auditory and haptic feedback.

References

Albusays, K., & Ludi, S. (2016). Eliciting programming challenges faced by developers with visual impairments. In *Proceedings of the 9th International Workshop on Cooperative and Human Aspects of Software Engineering - CHASE '16* (pp. 82–85). Austin, TX, USA. https://doi.org/10.1145/2897586.2897616

Baker, C. M., Milne, L. R., & Ladner, R. E. (2015). StructJumper. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems - CHI '15* (pp. 3043–3052). New York, New York, USA: ACM Press. https://doi.org/10.1145/2702123.2702589

Bigham, J. P., Aller, M. B., Brudvik, J. T., Leung, J. O., Yazzolino, L. a., & Ladner, R. E. (2008). Inspiring blind high school students to pursue computer science with instant messaging chatbots. *ACM SIGCSE Bulletin*, *40*(1), 449. https://doi.org/10.1145/1352322.1352287

Calder, M., Cohen, R. F., Lanzoni, J., Landry, N., Skaff, J., Calder, M., … Skaff, J. (2007). Teaching data structures to students who are blind. In *Proceedings of the 12th annual SIGCSE conference on Innovation*

and technology in computer science education - ITiCSE '07 (Vol. 39, p. 87). New York, New York, USA: ACM Press. https://doi.org/10.1145/1268784.1268811

Calder, M., Cohen, R. F., Lanzoni, J., & Xu, Y. (2006). PLUMB: An interface for Users who are Blind to Display, Create, and Modify Graphs. In *Proceedings of the 8th international ACM SIGACCESS conference on Computers and accessibility - Assets '06* (p. 263). New York, New York, USA: ACM Press. https://doi.org/10.1145/1168987.1169046

Capovilla, D., Krugel, J., & Hubwieser, P. (2013). Teaching Algorithmic Thinking Using Haptic Models for Visually Impaired Students. In *2013 Learning and Teaching in Computing and Engineering* (pp. 167–171). IEEE. https://doi.org/10.1109/LaTiCE.2013.14

Cheong, C. (2010). Coding without sight: Teaching object-oriented java programming to a blind student. In *Eighth Annual Hawaii International Conference on Education* (pp. 1–12). Hawaii International Conference on Education. Retrieved from http://researchbank.rmit.edu.au/view/rmit:13231

Department for Education. (2014). The national curriculum in England - Framework document. Department for Education. Retrieved from https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/381344/Master_final_national_curriculum_28_Nov.pdf

Dorsey, R., Chung, H. P., & Howard, A. (2014). Developing the Capabilities of Blind and Visually Impaired Youth to Build and Program Robots. In *28th Annual International Technology and Persons with Disabilities Conference*. San Diego: California State University, Northridge. Retrieved from http://scholarworks.csun.edu/handle/10211.3/121965

Francioni, J. M., & Smith, A. C. (2002). Computer science accessibility for students with visual disabilities. In *Proceedings of the 33rd SIGCSE technical symposium on Computer science education - SIGCSE '02* (Vol. 34, p. 91). New York, New York, USA: ACM Press. https://doi.org/10.1145/563340.563372

Franqueiro, K. G., & Siegfried, R. M. (2006). Designing a scripting language to help the blind program visually. In *Proceedings of the 8th international ACM SIGACCESS conference on Computers and accessibility - Assets '06* (p. 241). New York, New York, USA: ACM Press. https://doi.org/10.1145/1168987.1169035

Horn, M. S., & Jacob, R. J. K. (2007a). Designing tangible programming languages for classroom use. In *Proceedings of the 1st international conference on Tangible and embedded interaction - TEI '07* (p. 159). New York, New York, USA: ACM Press. https://doi.org/10.1145/1226969.1227003

Horn, M. S., & Jacob, R. J. K. (2007b). Tangible programming in the classroom with tern. In *CHI '07 extended abstracts on Human factors in computing systems - CHI '07* (p. 1965). New York, New York, USA: ACM Press. https://doi.org/10.1145/1240866.1240933

Howard, A. M., Chung Hyuk Park, & Remy, S. (2012). Using Haptic and Auditory Interaction Tools to Engage Students with Visual Impairments in Robot Programming Activities. *IEEE Transactions on Learning Technologies*, *5*(1), 87–95. https://doi.org/10.1109/TLT.2011.28

Jašková, Ľ., & Kaliaková, M. (2014). Programming Microworlds for Visually Impaired Pupils. In G. Futschek & C. Kynigos (Eds.), *Proceedings of the 3rd international constructionism conference*. Vienna. Retrieved from http://constructionism2014.ifs.tuwien.ac.at/papers/2.7_2-8251.pdf

Kane, S. K., & Bigham, J. P. (2014). Tracking @stemxcomet. In *Proceedings of the 45th ACM technical symposium on Computer science education - SIGCSE '14* (pp. 247–252). New York, New York, USA: ACM Press. https://doi.org/10.1145/2538862.2538975

Konecki, M. (2014). GUIDL as an Aiding Technology in Programming Education of Visually Impaired. *Journal of Computers*, *9*(12), 2816–2821. https://doi.org/10.4304/jcp.9.12.2816-2821

Koushik, V., & Lewis, C. (2016). An Accessible Blocks Language. In *Proceedings of the 18th International ACM SIGACCESS Conference on Computers and Accessibility - ASSETS '16* (pp. 317–318). New York, New York, USA: ACM Press. https://doi.org/10.1145/2982142.2982150

Lewis, C. (2014). Work in Progress Report: Nonvisual Visual Programming. In *Proceedings of the 25th Psychology of Programming Annual Conference (PPIG 2014)*. Retrieved from www.ppig.org

Ludi, S. (2013). Robotics Programming Tools for Blind Students. In *28th Annual International Technology and Persons with Disabilities Conference*. San Diego: California State University, Northridge. Retrieved from http://scholarworks.csun.edu/handle/10211.3/121968

Ludi, S. (2015). Position paper: Towards making block-based programming accessible for blind users. In *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)* (pp. 67–69). IEEE. https://doi.org/10.1109/BLOCKS.2015.7369005

Ludi, S., Ellis, L., & Jordan, S. (2014). An accessible robotics programming environment for visually impaired users. In *Proceedings of the 16th international ACM SIGACCESS conference on Computers & accessibility - ASSETS '14* (pp. 237–238). New York, New York, USA: ACM Press. https://doi.org/10.1145/2661334.2661385

Ludi, S., & Reichlmayr, T. (2011). The Use of Robotics to Promote Computing to Pre-College Students with Visual Impairments. *ACM Transactions on Computing Education*, *11*(3), 1–20. https://doi.org/10.1145/2037276.2037284

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch Programming Language and Environment. *ACM Transactions on Computing Education*, *10*(4), 1–15. https://doi.org/10.1145/1868358.1868363

Papazafiropulos, N., Fanucci, L., Leporini, B., Pelagatti, S., & Roncella, R. (2016). Haptic Models of Arrays Through 3D Printing for Computer Science Education. In *International Conference on Computers Helping People with Special Needs* (pp. 491–498). Springer, Cham. https://doi.org/10.1007/978-3-319-41264-1_67

Raman, T. V. (1996). Emacspeak---direct speech access. In *Proceedings of the second annual ACM conference on Assistive technologies - Assets '96* (pp. 32–36). New York, New York, USA: ACM Press. https://doi.org/10.1145/228347.228354

Remy, S. L., & L., S. (2013). Extending access to personalized verbal feedback about robots for programming students with visual impairments. In *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility - ASSETS '13* (pp. 1–2). New York, New York, USA: ACM Press. https://doi.org/10.1145/2513383.2513384

Sánchez, J., & Aguayo, F. (2005). Blind learners programming through audio. In *CHI '05 extended abstracts on Human factors in computing systems - CHI '05* (p. 1769). New York, New York, USA: ACM Press. https://doi.org/10.1145/1056808.1057018

Sánchez, J., & Aguayo, F. (2006). APL: Audio Programming Language for Blind Learners. In K. Miesenberger, J. Klaus, W. L. Zagler, & A. I. Karshmer (Eds.), *Computers Helping People with Special Needs. ICCHP 2006. Lecture Notes in Computer Science* (4061st ed., pp. 1334–1341). Springer, Berlin, Heidelberg. https://doi.org/10.1007/11788713_192

Siegfried, R. M. (2006). Visual programming and the blind: The Challenge and the Opportunity. In *SIGCSE '06 Proceedings of the 37th SIGCSE technical symposium on Computer science education* (Vol. 38, pp. 275–278). Houston, Texas: ACM. https://doi.org/10.1145/1124706.1121427

Siegfried, R. M., Diakoniarakis, D., Franqueiro, K. G., & Jain, A. (2005). Extending a scripting language for visual basic forms. *ACM SIGPLAN Notices*, *40*(11), 37. https://doi.org/10.1145/1107541.1107547

Smith, A. C., Francioni, J. M., & Matzek, S. D. (2000). A Java programming tool for students with visual disabilities. In *Proceedings of the fourth international ACM conference on Assistive technologies - Assets '00* (pp. 142–148). New York, New York, USA: ACM Press. https://doi.org/10.1145/354324.354356

Stefik, A., Alexander, R., Patterson, R., & Brown, J. (2007). WAD: A Feasibility study using the Wicked Audio Debugger. In *15th IEEE International Conference on Program Comprehension (ICPC '07)* (pp. 69–80). IEEE. https://doi.org/10.1109/ICPC.2007.42

Stefik, A., Hundhausen, C., & Smith, D. (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* (p. 571). New York, New York, USA: ACM Press. https://doi.org/10.1145/1953163.1953323

Stefik, A., & Siebert, S. (2013). An Empirical Investigation into Programming Language Syntax. *ACM Transactions on Computing Education*, *13*(4), 1–40. https://doi.org/10.1145/2534973

Stefik, A., Siebert, S., Stefik, M., & Slattery, K. (2011). An empirical comparison of the accuracy rates of novices using the quorum, perl, and randomo programming languages. In *Proceedings of the 3rd ACM SIGPLAN workshop on Evaluation and usability of programming languages and tools - PLATEAU '11* (p.

3). New York, New York, USA: ACM Press. https://doi.org/10.1145/2089155.2089159

The Royal Society. (2017). *After the reboot: computing education in UK schools*. Retrieved from https://royalsociety.org/~/media/policy/projects/computing-education/computing-education-report.pdf

Thieme, A., Morrison, C., Villar, N., Grayson, M., & Lindley, S. (2017). Enabling Collaboration in Learning Computer Programing Inclusive of Children with Vision Impairments. In *Proceedings of the 2017 Conference on Designing Interactive Systems - DIS '17* (pp. 739–752). New York, New York, USA: ACM Press. https://doi.org/10.1145/3064663.3064689

Appendix A: Literature Cross-Referenced by Theme

	Making Text-Based Languages Accessible	Making Block-Based Languages Accessible	Physical Artefacts	Auditory and Haptic Feedback
(Bigham et al., 2008)	x			
(Kane & Bigham, 2014)	x		x	
(Cheong, 2010)	x			
(Smith et al., 2000)	x			
(Francioni & Smith, 2002)	x			
(Ludi & Reichlmayr, 2011)	x		x	
(Dorsey et al., 2014)	x		x	x
(Ludi, 2013)	x		x	
(Ludi et al., 2014)	x		x	
(Sánchez & Aguayo, 2005)	x			
(Sánchez & Aguayo, 2006)	x			
(Andreas Stefik, Siebert, et al., 2011)	x			
(Andreas Stefik, Hundhausen, et al., 2011)	x		x	
(Andreas Stefik & Siebert, 2013)	x			
(Konecki, 2014)	x			
(Baker et al., 2015)	x			
(Siegfried, Diakoniarakis, Franqueiro, & Jain, 2005)	x			
(Franqueiro & Siegfried, 2006)	x			
(Siegfried, 2006)	x			
(A. Stefik et al., 2007)	x			
(Lewis, 2014)		x		
(Ludi, 2015)		x		
(Koushik & Lewis, 2016)		x		
(Thieme et al., 2017)			x	
(Papazafiropulos et al., 2016)			x	
(Capovilla et al., 2013)			x	
(Calder et al., 2007)				x
(Calder et al., 2006)				x
(Howard, Chung Hyuk Park, & Remy, 2012)	x		x	x
(Jašková & Kaliaková, 2014)		x	x	x
(Remy & L., 2013)			x	