Beyond the eye: making block-based programming languages accessible to visually impaired people

Sandro M. Rezende[1], Luciana T. Perdigão[1], Jociene M. M. Silva[1], Graziela F. Guarda[1], Sergio C. C. S. Pinto[1], Edicléa M. Fernandes[2], Gerlinde A. P. B. Teixeira[1]

[1]Programa de Pós-Graduação em Ciências, Tecnologias e Inclusão (PGCTIn) – Universidade Federal Fluminense (UFF) – Niterói – RJ – Brasil

[2]Faculdade de Educação – Universidade do Estado do Rio de Janeiro (UERJ) – Rio de Janeiro – RJ – Brasil

{sandromiranda,lucianaperdigao,jocienematheus,grazielaguarda,screspo, gerlinde_teixeira}@id.uff.br, professoraediclea.uerj@gmail.com

Abstract. *The aim of this work is to present a literature review concerning the accessibility of block-based programming languages and environments for people with visual impairments. The results show that the most popular block-based languages are built on graphical features that are not accessible for visually impaired people. However, some languages have been developed with the visually impaired learner in mind. We conclude that, in addition to the development of specific languages, other kinds of efforts have been made for the creation of accessible block-based programming environments.*

1. Introduction

Although technological resources are frequently discussed, many people still have little dimension of their relevance in everyday life. As digital technologies increasingly permeate spaces in society, it is important to develop content related to computing for k12 students. One way of creatively using these tools is through activities that involve the development of computational thinking, as well as basic concepts of computer programming to solve problems. Countries such as the United States, Canada and Israel, for example, as well as several countries in Europe, have already been implementing the teaching of skills related to digital technologies, the development of computational thinking and the introduction of computer programming in Basic Education [Ferri; Rosa 2016; Valente 2016].

However, it is essential that all learning situations are developed in a way that everyone can participate, considering the basic premises of inclusive school communities. As Artiles and Kozleski (2007, p. 357) state, "schools are about belonging, nurturing, and educating all children and youth, regardless of their differences in culture, gender, language, ability, class, and ethnicity". In this sense, meeting the different educational needs of students with and without disabilities is an important challenge that teachers must face in their pedagogical practice.

Understanding and valuing individual differences is an important aspect of an inclusive education. In the logic of inclusion, individual differences are recognized, accepted, and constitute the basis for building all pedagogical approaches. However, the implementation of inclusive didactics is not a simple task and must surpass traditional pedagogical models. In this context, it is necessary to understand what kind of knowledge is important for the teacher to plan and conduct learning activities that meet both the educational specificities of students with and without disabilities, and the multiple forms of interaction between the students, teachers, and content maters [Camargo; Nardi, 2008]. Teaching computing related contents and, more specifically, how to program, falls into this context.

More specifically, when teaching visually impaired people how to program, the choice of the programming environment that is going to be used is an issue that must be considered, since it is important that the resources enable participation and interaction for all students. Programming can be challenging to learn, and for learners with visual impairments there are additional barriers that must be considered. According to Kölling et al. (2017), making programming accessible, for example to blind programmers, is a difficult challenge.

Among the different types of programming environments that can be used in introductory programming, for educational purposes, block-based environments present relevant advantages. They allow students to develop codes by fitting blocks withpredefined expressions, and there is no need to learn complex text-based language elements, avoiding the occurrence of syntax errors. Besides, these resources make the programming activity playful and dynamic. On the other hand, since block-based languages are intrinsically visual, generally they are not accessible to visually impaired students. In this sense, in this study we proposed a systematic literature review on the accessibility of block-based programming environments for people with visual impairments, aiming to answer the following research questions: “are there block-based programming languages that are accessible to people with visual impairments?”; “If block-based programming languages that are accessible to people with visual impairments exist, what are they and how do they work?”; “What other strategies have been implemented to make block-based programming environments accessible?”

2. Method

A systematic review is a type of research that uses the literature on a given topic as the data source [Sampaio; Mancini, 2007]. This style of investigation provides a summary of the evidence related to a specific interventional strategy through the application of explicit and systematic methods of searching, critical appraisal, and synthesis of the selected information [Medeiros et al. 2015]. The research questions that guided this work were elaborated as follows: i) Are there block-based programming languages that are accessible to people with visual impairments? ii) If block-based programming languages that are accessible to people with visual impairments exist, what are they and how do they work? iii) What other strategies have been implemented to make block-based programming environments accessible?

The research was organized based on an adaptation of the systematic review checklist proposed by Medeiros et. al. (2015), as shown in Table 1. To select the papers, the inclusion criteria were: a) availability of the paper; b) written in English, Spanish or Portuguese; c) reference to block-based languages or programming; d) reference to visually impaired people; e) reference to accessibility. In turn, the exclusion criteria were: a) not about block-based programming; b) the paper quotes visually impaired people but does not present accessible block-based programming languages; c) only mentions accessibility other than visual; d) not full papers (abstracts, poster or conference presentations); e) in French.

3. Results and discussion

The searches returned the amount of 86 results. After the application of the inclusion and exclusion criteria 11 publications were selected for further analysis: 6 journal articles, 2 conference papers, 2 thesis, and 1 dissertation (Table 2). To answer the research questions, the publications were analyzed to identify the degree of accessibilityof blockbased programming languages for visually impaired people, and if such resources exist, what are they, how do they work and what strategies have been implemented to make block-based programming environments accessible.

Riazy et al. (2020) analyzed 17 web-based introductory programming environments, as well as their usability for visually impaired people. To perform the analysis, the authors created six categories where they checked if: 1) all of the images had alternative text descriptions; 2) the font size could be enlarged without endangering the structure and make-up of the website; 3) the website was navigable using exclusively the keyboard; 4) there was a contrast of at least 1:3 for important items (text, graphics); 5) the language of the website could be automatically detected using a web service, which may be necessary for reading software; and 6) all viewers could display and process the website correctly. Most of the environments analyzed were widely used graphical blockbased programming languages, such as Scratch, Snap!, OpenRoberta and MIT App Inventor. The authors concluded that none of the 17 analyzed graphical programming environments were suitable for blind and visually impaired people including the widely used graphical programming environments based on drag-and-drop blocks used in Scratch and Scratch-like IDEs. The authors place that "apart from missing alt-text descriptions, the sites were not operable using only the keyboard".

Alternatively, in other studies it was possible to identify that less popular languages have been developed aiming to provide accessibility to visually impaired people. We also identified other types of efforts that have been made aiming to make block-based languages accessible. These were divided into five categories, as follows.

3.1. Development of new accessible block-based languages

Hadwen-Bennett et al. (2018, 2019) highlighted the fact that many modern programming environments such as block-based languages are intrinsically visual and therefore are inaccessible to visually impaired learners, as they are difficult or impossible to use with a screen reader. The authors performed a literature review to identify strategies that have been employed to make programing learning accessible to visually impaired learners and were able to identify some alternatives in this context. Their results fell into four categories: physical artefacts, auditory and haptic feedback, making text-based languages accessible, making block-based languages accessible.

Firstly, they presented Noodle, a programming system that allows the creation ofsound and music that has program elements which can be inserted and arranged using keyboard commands. Nevertheless, the authors mentioned that the language used in the audio feedback is not appropriate for primary school children, which makes it an unsuitable choice for the introduction of programming to young visually impaired children [Hadwen-Bennett et al., 2018, 2019].

Another language mentioned by Hadwen-Bennett et al. (2018, 2019) was the Pseudospatial Block (PB) language, in which the learner can select an insertion point using the keyboard and choose the program element they want from a list that is filtered by a syntactic category. One of the advantages of the Pseudospatial Blocks for all learners is the fact that invalid program blocks for a given space are filtered out of the list. The authors also mentioned The Lady Beetle and World of Sounds, two programming environments that were developed to introduce young visually impaired children to basic concepts of programming. According to them, The Lady Beetle programming environment enables the learner to select single word commands through the movement of a beetle across a grid, without having to type the word, in such a way that as the beetle moves, the coordinates of the current square are read out. In its turn, World of Sounds allows learners to create simple programs that produce sequences of sounds.

In accordance, Milne (2018) also stressed that commonly used block-based programming environments, are inaccessible for visually impaired children as they heavily rely on visual elements to convey information. Considering this fact, she explored ways to allow children with visual impairments aged 5 to 14 years to access the spatial information in programming environments. In this scenario, with the aim of developing a touchscreen device with a combination of tactile and audio feedback that allowed children to understand and create block-based programs, she built Blocks4All,a blockbased environment for the Apple iPad that basically requires the use of the touchscreen and VoiceOver (a screen reader built into Apple operating systems). The environment explores non-visual features to present information about blocks, block types, the spatial structure of program code and specific interactions that replace the drag and drop gestures that are inaccessible to visually impaired people.

3.2. Development of block programming editors and libraries

According to Pasternak et al. (2017, p. 21), Blockly is “an open-source library that makes it easy to add block based visual programming to an app”. It is a Google project, first released in May 2012, that was designed to be flexible and support a large set of features for different applications. The Blockly library adds an editor to an app that represents coding concepts such as variables, logical expressions and loops as interlocking blocks, and outputs syntactically correct code in a programming language of the user’s choice [Google, 2020].

As a library, Blockly is not a language, but allows developers to create their own block-based languages, providing a grammar and a representation for programming that developers can use in their apps. Pieces of code are represented by blocks, which maybe dragged around the screen. However, Blockly does not provide a full vocabulary of blocks or a runtime environment. Developers need to integrate Blockly with some form of output, build their vocabulary, and decide how the generated code will run [Pasternaket al., 2017, p. 21].

Using Blockly as a model, Ludi and Spencer (2017) discussed accessibility design issues in block-based programming environments, focusing on the programming workflow. The authors mention that Blockly is often used as a framework for the design of other block-based languages and activities. They redesigned Blockly’s user interface in such a way that the current drag-and-drop resources for creating block-based programs was preserved, while adding additional features such as keyboard interface, screen reader compatibility and appearance customization to increase accessibility to visually impaired users. Among their goals we draw attention to: providing a unique identifier for each block to enable visual and audio-based understanding of blocks; allowing the user to change the highlight color of blocks, connection points of blocks, or the workspace that minimize visual discomfort and improve readability and discernibility; allowing the keyboard to navigate between blocks and within a block; and providing audio cues in order to reinforce the level of nesting.

In turn, Schanzer et al. (2018) argued that sighted programmers can use visual elements such as syntax highlighting, bracket matching and auto-indenting to help them understand the general structure of a program and keep track of the abstract syntax tree. However, these elements are of little help for visually impaired people. On the other hand, Baker et al. (2015, cited in Schanzer et al., 2018) pointed out that there is an improvement in comprehension when visually impaired people can navigate the structure of the program instead of its syntax, which gives block-based languages an advantage since they allow the representation of the code’s tree structure. In this perspective, the authors developed the UncleGoose toolkit, which creates a block programming environment for different languages. Such an environment provides a block editor that uses standard drag and drop conventions that can be done using keyboard navigation. Besides, spoken feedback is also provided in a way that the description of a block is separated from its visual or textual syntax, providing a third representation beyond texts and blocks.

3.3. Use of auditory cues to sonify block-based programs

As mentioned before, another strategy explored in some of the analyzed studies was the use of auditory cues. Ludi et al. (2016) investigated if auditory cues used in a visual programming environment were effective in aiding users in navigating and understanding source codes, and which types of cues are the most effective. The authors conducted an experiment with seven visually impaired programmers, using a modified version of Blockly. Along with the editor, the study employed the use of auditory cues, specifically earcons (abstract sounds, usually musical, that are used to represent objects or ideas) and spearcons (sounds generated from sped up speech). The experiment consisted of three trials using a different type of auditory cue for each. The participants used their typical screen readers to conclude the tasks, which involved finding a specific block in a pregenerated source of code using the available auditory cues, listening to an audio description of a set of source code, and then writing down their interpretation of the code’s structure. When asked how useful the participants felt over the auditory cues for navigation and comprehension, they rated that speech was the most useful auditory cue, followed by spearcons and then earcons.

In a later study Ludi et al. (2019) compared the three types of audio cues (speech, earcons and spearcons) to identify the most useful and the preferred by the users. In the experiment, three visually impaired participants conducted a set of tasks using a mockup based on Google Pencil Code and gave their opinions over how each auditory cue should sound. The results showed that all the participants were able to identify blocks through the three auditory cues. However, contrary to results presented by the earlier study [Ludi et al., 2016], the participants of this study found that earcons were the most useful auditory cues, followed by speech and then spearcons.

3.4. Development of audio-based programming languages with block-based elements

Damsma and Norgaard (2018) highlighted the lack of accessible coding programs with the relevant and essential specifications for children. The authors identified the accessibility issues of the existing barriers in non-accessible programs regarding the user interface and the coding output. They emphasized the fact that block-based programming languages use an interface with graphical elements, which are user friendly for sighted children but create accessibility challenges for students who are blind. In this perspective, they presented Sonokids, a fully accessible iPad coding app specifically designed to support visually impaired children to acquire basic coding skills. Among the app's objectives, the authors say that the idea is "to enable young children who are blind to start to participate equally in learning how to code", and to achieve this "by developing an innovative, engaging, interactive learning experience on iPad, which offers high levels of accessibility and usability for young children who are blind, by way of audio".

Subsequently, Damsma and Norgaard (2018) attempted to provide easy accessto the pieces of code and easy navigation of code sequencing and editing, with no need to type any text or drag and drop codes. In this perspective, the app provides pieces of code that can be selected in an accessible panel through finger gestures. In addition, the authors argued that in most other programs, it is common that the audio output that accompanies the graphical output is not as meaningful, engaging nor as motivating as the visual effects are for sighted children. Therefore, they aimed to offer audio description, audio effects and audio alerts that were truly meaningful and could provide fun engagement for visually impaired children.

Inspired by the widely used block-based programming language Scratch, Quach (2019) developed Codi, a computer program that children can talk to in order to create, modify, and explore computer programs. Although Codi is not a block-based language, we included this paper as it has a conversational audio-based interface that was designed for a screenless experience and was inspired by voice assistants. This program presents new possibilities and reinforces the potential of programming through conversation. Besides, its virtual machine is heavily influenced by the Scratch Virtual Machine, and both share the same targets, threads, runtime, and sequencer.

Quach (2019) also states that, in the development of the program, she first attempted to implement screen reader compatibility for block-based programming. However, in that case visually impaired children would have to follow a series of steps in the learning process: firstly, they would have to learn how to use the screen reader to access a web browser on a computer; then they would have to learn how to navigate the code editing interface; next, it would be necessary to build an understanding of the blocks; and finally, they would have to learn programming concepts by building with the blocks. In her own words. Considering the amount of work that would have to be done by the child, the author concluded that screen reader compatibility alone was not enough for this audience to get started. On the other hand, as visual languages can lower the barriers in the programming process for sighted children, a conversational audio-based interface could be positive for children with visual or motor impairments. In this perspective, Codi integrates audio cues that support coding conversationally, providing information and suggestions when the child asks for them. This allows children to express themselves through the projects.

3.5. Development of tangible programming languages with block-based elements

The development of tangible programming languages (physical programming languages) was identified by Hadwen-Bennett et al. (2018) as one of the main strategies that have been employed in the quest of making the programming learning process accessible to visually impaired learners. As the authors state, "in physical programming languages (PPLs), commands are represented by physical objects which can be joined together to create programs", which makes them particularly promising in terms of accessibility to visually impaired people as they can be explored by touch.

Although the analysis of tangible programming languages is not the scope of this study, Angelo (2018) proposes a block-based system formed by tangible pieces with Braille characters for introductory programming activities for blind and sighted children. In the proposed system, based on the Scratch interface, virtual blocks were replaced by physical pieces with different shapes, so that tactile recognition and auditory responses were provided. The Scratch programming principles were maintained, with the difference that the digital blocks were replaced by tangible parts that could be moved manually, instead of using the mouse. This physical programming environment provided integration with the Scratch platform. According to the author, the proposed system recognizes tangible parts arranged in a defined area, generates a listwith the parts in the order they are organized in the area, creates the digital blocks in Scratch according to the order of the parts, executes the program and emits sounds in response to certain interactions with the user. This project represents an alternative to overcome limitations related to digital block-based languages for visually impaired children, as it enables the exploration of block programming through tactile manipulation and, in addition, allows the created code to be executed on the computer.

Finally, it is important to emphasize the existence of other types of efforts that may be not primarily aimed at promoting accessibility to people with visual impairments but can still benefit them. This is the case of the work of Kölling et al. (2017), which did not meet the inclusion criteria for analysis in this work, as it does not represent a specific strategy that has been implemented to make block-based programming environments accessible, but still presents benefits for people with visual impairments. The authors argue that block-based editors have become very popular in introductory programming classes since they offer many advantages for beginner programmers, such as elimination of syntax errors and availability of instructions for visual selection. On the other hand, text-based systems are preferred among proficient programmers or expert users due to usability and productivity related advantages. In thisperspective, they discuss the framebased editing paradigm. Although it does not represent a block-based language, it combines advantages of block-based and text-basedsystems. The authors discuss the implementation of such a system for a java-like language called Stride.

In a frame-based editor, program components are represented by frames, using both graphical and textual elements. In the case of the frame-based editor developed for Stride, for example, "the editor uses some graphical elements (shapes and colors) to present aspects where graphics have advantages over characters […], however, the presentation maintains the look of a program as essentially a textual, if colored, document" [Kölling et al. 2017, p. 44].

Regarding accessibility, the authors state that frame-based editing presents different features that contribute to making them more accessible than existing block-based editors. Besides, based on initial studies with middle-school students and experienced programmers, the authors concluded that the proposed editing system has clear advantages for both novice and expert programmers, improving program representation and error avoidance for beginners, as well as speeding up program manipulation for experts. The participants also reported more readability and flexibility in the insertion of new codes or deletion of existing codes [Kölling et al. 2017]. Although it was not mentioned whether there were visually impaired people among the participants, framebased editing are within the possibilities to improve programming environment accessibility.

4. Conclusion

In the educational field, the objective of developing computational thinking is not to transform all children into software developers, but to provide tools that allow students to understand and act in the digital culture in which they are inserted. From the perspective of inclusive education, the provision of these tools involves planning and conducting learning activities that meet the educational specificities of students in their individuality, as well as the multiple forms of interaction between students, teachers and content. In this context, when planning learning situations for the development of computer programming skills for groups where there are people with visual impairments, the choice of the programming environment that will be used is crucial, as the resources used must enable participation and interaction of all those involved.

Learning to program is not a simple task, and for visually impaired people the challenge is even greater since many of the existing programming environments are not accessible. To make the programming learning process simpler, new languages that have less complex syntax have been developed over time, offering graphic and playful features to help young learners to program. Block programming languages fall into this context, since instead of requiring users to type textual elements to build the code, they allow them to select predefined structures from lists presented on the screen, and interconnect them through blocks. Consequently, it is possible to avoid many syntax errors that are typical in text-based languages and prevent programs to run correctly.

On the other hand, as block-based languages are essentially visual and commonly use drag and drop features that are operationalized using the mouse, they are usually not accessible to visually impaired people. To address this issue and enable the exploration of these languages benefits in the programming learning process of visually impaired learners, several researchers and programmers have been looking for solutions to make them accessible. The literature review allowed us to identify some of the effortsthat have been made in this direction.

From the analysis of the articles, it was possible to identify several block-based programming languages that were developed considering design aspects to provide accessibility to visually impaired people. Projects such as Noodle, Pseudospatial Blocks, The Lady Beetle, World of Sounds and Blocks4All are some examples of block-based languages built with a focus on accessibility, which illustrates that it is possible to develop accessible and playful programming environments aimed for young learners. Efforts to build block programming editors with accessibility elements have also been identified, as well as attempts to modify and extend existing editors. In special the Blockly library, that aims to carry out improvements that enable the creation of new accessible block-based languages. We also identified attempts to insert different types of audio cues to give meaning to components of the source code in block-based programs built using Blockly, with promising results.

The association of block-based elements with other types of languages, such as audio-based, text-based or tangible languages was another strategy identified. This type of approach makes it possible to explore the advantages of different languages and overcome the limitations of block-based environments. In addition, in these cases other senses such as hearing and touch gain emphasis in the construction and understanding of the codes, which contributes to giving meaning to the learning process of visually impaired people.

It is worth mentioning that although it is possible to find accessible block-based programming environments, they still represent a minority and present limited possibilities. Block-based languages are increasingly being used in different learning situations due to their pedagogical potential. However, the most popular languages, suchas Scratch, still do not meet the accessibility requirements to enable visually impaired people to participate effectively in such learning contexts. Just as it is important that developers continue working to provide accessibility to block-based environments, it is essential that teachers of visually impaired students seek to understand the potentials, benefits, difficulties, as well as the limitations of the different programming environments and based on this evaluation, find the best type of language to work with their visually impaired students.

Finally, it is possible to raise a reflection about the role of the visually impaired people in the development of more accessible block programming languages. This approach faces the blind person not only as a user, but also as a developer of technologies, and brings the necessity of including blind people in the systems development teams. Lastly, it allows the creation of programming languages based on the blind’s perception of the world and on the artifacts that blind people produce and use.

References

Angelo, I. M. (2018) Recomendações para o desenvolvimento de ambientes de programação inclusivos para crianças cegas. Universidade de São Paulo, São Paulo, 127 p. Dissertação de mestrado.

Artiles, A.; Kozleski, E. (2007) Beyond convictions: interrogating culture, history, and power in inclusive education. Language Arts, 84, n. 4, mar, p. 351-358.

Camargo, E.; Nardi, R. (2008) O emprego de linguagens acessíveis para alunos comdeficiência visual em aulas de Óptica. Revista Brasileira de Educação Especial, v.14, n. 3, p. 405-426.

Damsma, P.; Norgaard, J. (2018) Audio Based Coding: An Innovative Approach to Accessible Coding for Children who are Blind. Journal of the South Pacific

Educators in Vision Impairment, v. 1, n. 1, p. 49-58.

Ferri, J; Rosa, S. S. (2016) Como o Ensino de Programação de Computadores Pode Contribuir Com a Construção de Conhecimento na Educação Básica Uma Revisão Sistemática da Literatura. CINTED-UFRG.

Google (2020) Introduction to Blockly. Available at: https://developers.google.com/blockly/guides/overview. Accessed: 20 January 2021.

Hadwen-bennett, A.; Sentance, S.; Morrison, C. (2018) Making Programming Accessible to Learners with Visual Impairments: A Literature Review. International Journal of Computer Science Education in Schools, v. 2, n. 2, p. 3-13.

Hadwen-bennett, A.; Sentance, S.; Morrison, C. (2019) Cómo conseguir que la programación sea accesible a estudiantes con discapacidades visuales: examen de la bibliografía. Integración: Revista sobre ceguera y deficiencia visual, n.74, p.127-150.

Kölling, M.; Brown, N. C. C.; Altadmri, A. (2017) Frame-Based Editing. Journal of Visual Languages and Sentient Systems, v. 3, p. 40–67.

Ludi, S.; Wang, J.; Chapati, K.; Khoja, Z.; Nguyen, A. (2019) Exploring the Use of Auditory Cues to Sonify Block-Based Programs. Journal on Technology and Persons with Disabilities, v. 7, p. 1-21.

Ludi, S.; Simpson, J.; Merchant, W. (2016) Exploration of the Use of Auditory Cues in Code Comprehension and Navigation for Individuals with Visual Impairments in a Visual Programming Environment. In: Proceedings of the 18th International ACM SIGACCESS Conference on Computers and Accessibility. New York, NY, USA: Association for Computing Machinery, p. 279-280.

Ludi, S.; Spencer, M. (2017) Design Considerations to Increase Block-based Language Accessibility for Blind Programmers Via Blockly. Journal of Visual Languages and Sentient Systems, v. 3, n. 1, p. 119-124.

Medeiros, I. L et al. (2015) Revisão sistemática e bibliometria facilitadas por um Canvas para visualização de informação. Revista Brasileira de Design da Informação, v. 12, n. 1, p. 93-110.

Milne, L. (2018) Touchscreen-Based Learning Technologies for Children with Visual Impairments. University of Washington, Washington. 225 p. Dissertation. Available at: https://digital.lib.washington.edu:443/researchworks/handle/1773/43021. Accessed: 6 april 2022.

Pasternak, E.; Fenichel, R.; Marshall, A. N. (2017) Tips for creating a block language with blockly, 2017 IEEE Blocks and Beyond Workshop (B&B), Raleigh, NC, USA, p. 21-24.

Quach, T. (2019) Agent-based programming interfaces for children supporting blind children in creative computing through conversation. Massachusetts Institute of Technology. Thesis. Available at: https://dspace.mit.edu/handle/1721.1/123155. Accessed: 6 april 2022.

Riazy, S.; Weller, S.; Simbeck, K. (2020). Evaluation of Low-threshold Programming Learning Environments for the Blind and Partially Sighted: In: Proceedings of the

12th International Conference on Computer Supported Education. Prague, Czech Republic: SCITEPRESS - Science and Technology Publications, p. 366-373.

Sampaio, R. F., Mancini, M. C. (2007) Estudos de Revisão Sistemática: Um Guia Para Síntese Criteriosa Da Evidência Científica. Revista Brasileira de Fisioterapia, São Carlos, v. 11, n. 1, jan./fev., p. 83-89.

Schanzer, E.; Bahram, S.; Krishnamurthi, S. (2018) Building an accessible block environment: multi-language, fully-accessible AST-based editing in the browser. BLOCK+ 2018, Boston, Massachusetts, nov, p. 2.

Valente, J. A. (2016) Integração do pensamento computacional no currículo da educação básica: diferentes estratégias usadas e questões de formação de professores e avaliação do aluno. e- Curriculum, São Paulo, v. 14, n. 6, p. 34.