Position Paper: Towards Making Block-Based Programming Accessible for Blind Users

Stephanie Ludi
Department of Software Engineering
Rochester Institute of Technology
Rochester, USA
salvse@rit.edu

***Abstract*—Block-based programming environments are not accessible to users who are visually impaired. The lack of access impacts students who are participating in computing outreach, in the classroom, or in informal settings that foster interest in computing. This paper will discuss accessibility design issues in block-based programming environments, as well as present research questions and current design revisions being undertaken in Blockly.**

Keywords—accessibility; aria; user interface design; visually impaired

I. Introduction

Block-based systems have gained prominence in recent years as a means of introducing novices to programming. In some cases, such as MIT's Scratch, online communities exist and the tools are integrated into pre-college computer science curricula (e.g. Exploring Computer Science, CSPrinciples) [2, 4]. As these systems have become components of curricula, after-school camps, and outreach, the lack of accessibility for many students with disabilities creates an obstacle for participation in these activities that are devised to increase participation in computing. In particular users with visual impairments are generally not able to use block-based tools unless they have enough vision to view the screen comfortably.

Visually impaired individuals may have some sight or have little to no vision. Assitive technology varies according to the degree of sight a person posseses. People who can read magnified text may use screen magnification software in order to zoom in on the contents of the screen by a specified magnification factor, adjust forground and background colors, or increase the size of the cursor. These users typically use the mouse alongside the keyboard. Individuals who are blind do not use the mouse. Instead navigation is typically via the keyboard (often through keyboard shortcuts). Screen readers (and for some refreshable Braille displays) are the means to access information that is displayed on the screen. When a website or program is designed correctly, the screen reader will read the content, including menus and other navigational elements. In the case of typical block-based programming environments, the screen reader reads no content or navigation.

This paper will present accessibility issues with the design of Block-based languages, as well as research questions being explored in an ongoing project. Users with visual impairments, including blindness are the focus of this paper.

II. Block-based Programming Environment Features that affect Accessibility

Each block-based project team makes design decisions for their project, that in some cases have unforseen consequences. Like any softwear project, the team prioritizes features based on a variety of needs that are considered. Examples of these design decisions are:

A. Technology and Platform Choices

The technologies used to develop block-based systems can have a large impact on accessibility of said systems to the disabled. The impact can be significantly positive or negative.

For example, Scratch 2.0 was developed in Actionscript/Flash. A positive implication is that Scratch runs in the browser, making installation seamless and thus easy to access for many people using various operating systems. At the same time the technology selection makes accessiblity impossible. In 2010, Adobe announced accessibility suport for Flash/Flex in terms of ARIA [3]. While ARIA is supported in terms of roles and states for HTML, Flash objects and actionscript does not support (ARIA) roles and states so presenting and interacting with a system is not possible. Another issue for screen reader users, who rely on keyboard shortcuts, is that Flash can override those keyboard shortcuts.

The Lego Mindstorm block-based robotics programming environment is traditionally a desktop application that uses LabView as the underlying technology. While the software runs on the Windows and Mac operating systems, the software is not compatible with screen readers. As a result, users who are blind will not hear anything.

On the positive side, Blockly is developed in CSS, SVG, and JavaScript. As such Blockly also runs on the web browser. However Blockly does not have the same accesibility issues as it can leverage ARIA (Accessible Rich Internet Applications) specification from the W3C's Web Accessibility Initiative. Developers need to adhere to the ARIA specification as compatibilty does not happen automatically for any web application. By following ARIA, a screen reader can read the structure of the webpage, labels on buttons and menus, as well

978-1-4673-8367-7/15/$31.00 ©2015 IEEE

as graphical objects (e.g. blocks). In addition, widgets can be described (e.g. slider, treeitem), keyboard navigation can be provided, as well as clearly articulated properies for drag-and-drop, widget states, or areas of a page that can be updated over time or based on an event. [6]

B. Mouse-centric Input

Many block-based systems rely on mouse input as the primary means of accessing features, including selecting blocks and adding them to programs, selecting attributes, and running the created program. For example, one cannot use the keyboard to locate, select and place a block onto the workspace in Scratch or Blockly as they were originally designed to be used with the mouse.

Many users with visual impairments rely on the keyboard to access software. As such, keyboard-focused commands and shortcuts are key to making interaction possible. Systems must be designed in order to utilize the keyboard as input in terms of menu navigation, programming, and accessing various panes in the programmng environment (e.g. changing focus to access specific information). In addition, the keys used to access features and information needs to be consistent with said standards and not conflict with keyboard shortcuts used by screen readers. In order to provide appropriate access to both sighted and visually impaired students, designing the system to allow for interaction via the mouse or keyboard is needed.

C. Feedback

User feedback in block-based programming environments are often visual in nature (e.g. a visual change on the workspace, pop-up messages, dialog boxes) without an audio feedback mechanism. Examples include a successful compilation or incompatible blocks that the user tries to connect together. Providing associated audio-based feedback can take various approaches depending on the nature of the feedback.

Students who use screen readers need to have all content including errors and any status messages provided audibly. Audio-based feedback can be in the form of speech or sound. As a pane or dialog box gets focus, the error or status text can be read (assuming it is programmed to enable a screen reader to access the text). Other feedback may be in the form of sounds (e.g. an audio icon or earcon) that correspond to meaning. An example of an audio icon is the sound of crumpling paper when a file is moved to the trashcan [1]. An example of an earcon is a tone or chord that is abstract in terms of the sound itself, but it is given meaning according to the association such as a deep sound may correspond to an unsuccessful download of the program to the robot [5]. The author is leveraging related work has been conducted to assess the use of audio cues in programming for programmers, though the study was conducted in a traditional, text-based programming environment [5].

D. Block-based Programs Created by the User

Each block-based program is designed for a particular purpose. Scratch programs can be in the form of animations or games. Lego Mindstorms NXT-G programs allow a robot to move and interact with its environment. Blockly programs can be translated into other languages such as JavaScript or Dart (though derivitives have been designed for specific domains such as music). As such, when a system is designed to be accessible, the devised programs themselves should be accessible to their users.

The audio capabilities of many block-based systems are limited, whether it be the ability for a robot to play a tone or recorded sound or for a Kodu game to play a sound effect when an event occurs. The audio capabilities, including the ability for a form of audio description when an animation is played or dialog is displayed is needed. Tapping into the location attributes or dialog text to enable compatibility with a screen reader is possible in many technologies (especialy JavaScript, Java, C++, C#).

III. Research Questions

Using the Blockly platform, the author and her team is re-engineering the system to enable use by users who are visually impaired. As such, our preliminary research questions are:

- Do visually impaired users want access to a block-based programming environment?
- How can Blockly be made accessible to the visually impaired, while at the same time remain usable to sighted users?
- What features will both visually impaired and sighted users appreciate?

IV. Design Implications

Modifying Blockly to provide access to users who are visually impaired is underway. The following sections provide a high-level view of the modifications to the system. The team uses only the JavaScript libraries that Blockly has originally used. The only exception to be made may be due to specific audio feedback that is being studied at this time. This addition would be a single JavaScript library.

A. Program Presentation

The visual and audio-based presentation of blocks and their content is fundamental. The key design revisons are:

- Allowing the user to change the highlight color for the current block or the connection point of a block in the workspace to improve discernability.
- Allowing the user to change the color of the workspace in order to minimize visual discomfort and improve readability.
- Enabling the blocks to be read on the workspace and in the toolbox (the menu where blocks are chosen)
- Visually linking blocks with any associated comments, where the user can jump from the block to the comment and back as desired.
- Providing a unique identifier with each block to enable visual and audio-based understanding of blocks.

B. Program Creation

Blockly requires the user to click on a block in the toolbox and place it in the desired place on the workspace using a mouse. Using the mouse, blocks can be moved as desired and attributes and parameters are set by selecting fields and entering a value via text or selection. This is typical of block-based programming environments. To make Blockly accessible, the following revisions are being implemented:

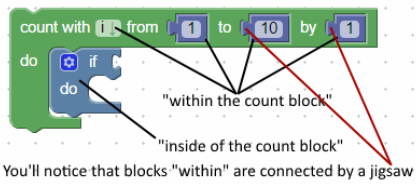
- The toolbox menu can be accessed and navigated via the keyboard.
- The desired block can be selected in the toolbox via a keyboard or mouse and it will appear at the insertion point on the workspace.
- Blocks can be moved within a program with a keyboard or mouse.
- Valid blocks that can be connected to a specified block are conveyed to the user visually and audibly.

C. Code Navigation

While program creation is critical, one tends to program incrementally in terms of adding to the features or complexity of the program, as well as fixing defects. As such, the need to enable visually impaired users to be able to navigate their code is critical. The features to facilitate the navigation of code are:

- Use of the keyboard to navigate between blocks and within a block (e.g. to fields), an shown in Figure 1.

FIG 1. Annotated View of a block with vertical and a nested block.



You'll notice that blocks "within" are connected by a jigsaw

- Provide each block (including vertical blocks) with a unique identifier to provide a visual and auditory structure for the program. The identifier is presented in the tree view (described in the next section).
- The use of audio cues in order to reinforce the level of nesting. A comparitive study is currently underway.
- Each block as well as each block part (e.g. field or inner block) can be read as a single block or in the program as a whole, depending on user need. When vertical blocks are read such as the 'count with' block presented above, the entire line can be read without all of the block identifiers if desired.

D. Comment Presentation

Program documentation is an afterthought to many programmers, but it is a feature that is in many block-based programming environments. Providing comments can help when a teacher wants to prepare lessons with starter programs or when a student may work on a program over time. While comments are text-based, activating the ability to add a comment requires the mouse in Blockly.

As part of redesigning the user interface, the user will have a box on the side of their screen to view comments in a tree view. The screen reader reads the comments as desired. The tree view will be updated automatically, and the structure will match that of the program using the hierarchy of identifiers associated with each block. In addition, a line will connect the current block with the comment line in the tree view.

Since the blocks in the workspace have their own identifying prefixes assigned to them (e.g. 1.1, 1.2) and the comment tree view will display the block prefix followed by that blocks comment if it has a comment. They will also have an info box that tells more information about the block they are currently on in a larger font for a better understanding of where they are in the structure. The user will be able to jump between a comment they have highlighted and a block in the workspace.

V. NEXT STEPS

The initial version of accessible Blockly should be completed during Fall 2015. The results of an audio feedback study that assessed the impact of various types of audio feedback modalities for code navigation and the understanding of nesting will be used in implementing code-based audio feedback during code navigation, in conjunction with the option for screen reader use, if needed. The accessibility features will be also studied in order to compare the usability impact for users with and without sight in order to ascertain what value may be found for sighted users as well as those who are visually impaired.

REFERENCES

[1] Dingler, T., Lindsay, J., & Walker, B. N. (2008). Learnability of sound cues for environmental features: Auditory icons, earcons, spearcons, and speech.Proceedings of the International Conference on Auditory Display (ICAD 2008), Paris, France (24-27 June).

[2] Goode, J. (2011, Summer). Exploring computer science: An equity-based reform program for 21st century computing education. *Journal for Computing Teachers*. Retrieved from http://www.iste.org/store/magazines-and-journals/downloads/jct-downloads.aspx

[3] Kirkpatrick, A. (2010) Adobe Accessibility / Flash Player and Flex Support for IAccessible2 and WAI-ARIA. Retrieded from: http://blogs.adobe.com/accessibility/2010/03/flash_player_and_flex_support.html

[4] Maloney, J., Resnick, M., Rusk, N., Silverman, B., Eastmond, E. (2010). The Scratch Programming Language and Environment. ACM Transactions on Computing Education, November 2010

[5] Stefik, A., Hundhausen, C., & Patterson, R.. An Empirical Investigation into the Design of Auditory Cues to Enhance Computer Program Comprehension. The International Journal of Human-Computer Studies, 69 (2011) 820-838.

[6] W3C-WAI (2014). WAI-ARIA Overview. Retrieded from: http://www.w3.org/WAI/intro/aria