A JBrick: Accessible Robotics Programming for Visually Impaired Users

Stephanie Ludi(✉) and Scott Jordan

Department of Software Engineering, Department of Computer Science,
Rochester Institute of Technology, Rochester, USA
{salvse,saj1832}@rit.edu

Abstract. Despite advances in assistive technology, challenges remain in pre-college computer science outreach and university programs for visually impaired students. The use of robotics has been popular in pre-college classrooms and outreach programs, including those that serve underrepresented groups. This paper describes the specific accessibility features implemented in software that provides an accessible Lego Mindstorms NXT programming environment for teenage students who are visually impaired. JBrick is designed to support students with diverse visual acuity and who use needed assistive technology. Field tests over several days showed that JBrick has the potential to accommodate students who are visually impaired as they work together to program Lego Mindstorms NXT robots.

Keywords: Accessibility · Robotics · Visual impairment

1 Introduction

As robotics has become popular as a means for engaging pre-college students in computing and engineering [2, 15, 20], the need for accessibility persists. Robotics, such as Lego Mindstorms, are as appealing to students who are visually impaired as they are to sighted students [3, 12]. The default programming software available from Lego uses icons to represent commands. This software is not accessible, most notably in terms of screen reader compatibility. Whether for in-class activities or extracurricular outreach, the software needs to maximize accessibility in order to promote interest in computer science and related disciplines.

The underrepresented students of concern are those who are visually impaired, where the threshold is legally blind. The American Federation of the Blind defines the term "legally blind" as defined through federal law, with "central visual acuity of 20/200 or less in the better eye with the best possible correction, as measured on a Snellen vision chart, or a visual field of 20 degrees or less" [1].

The goal of the JBrick project is to devise accessible Lego Mindstorms programming software that can be used by those with or without sight. In the case of the Imagine IT workshops and future outreach, the target users are teens who are visually impaired. These teens are often novice programmers, as the focus of the outreach is to enable the

© Springer International Publishing Switzerland 2015
M. Antona and C. Stephanidis (Eds.): UAHCI 2015, Part III, LNCS 9177, pp. 157–168, 2015.
DOI: 10.1007/978-3-319-20684-4_16

participants to explore Computer Science via robotics, a common vehicle for engaging pre-college students [2, 13, 16, 20].

In this paper, we will explore the issues with making robotics programming accessible to individual with visual impairments, especially those who have little to no experience in programming. Prior work has discussed issues with using existing development software [11, 13], which at best has incomplete features and at worst is completely inaccessible due to the heavy use of graphics to depict commands and constructs. Participation in STEM (Science, Technology, Engineering, and Math) fields by the visually impaired is low in part due to inaccessible tool support, in particular at the initial, critical junctures that can encourage and captivate young people. Given that independent tool use and activity participation is needed (as opposed to reliance on a sighted person) this paper will focus on the features and design decisions that can be leveraged in other programming tools.

2 Background and Related Work

The need for JBrick is derived after evaluating exiting Lego Mind storms NXT robotics programming environments in order to ascertain at least one that was free/low-cost, accessible to the visually impaired, and was conducive to facilitating outreach/instruction for novice programmers of pre-college age.

In terms of general programming, contemporary environments such as Microsoft Visual Studio, Apple XCode and Eclipse can be made generally accessible though gaps remain. For the purposes of audience and outreach timeframe, these tools were ruled out as being irrelevant for the robotics programming or potentially daunting to young novices (Eclipse can be used for LeJOS). Many programmers who are visually impaired use text editors and command line compilers along with their assistive technology, while others also use tools such as emacspeak to help convey the nuances of code [4, 17].

In terms of programming environments developed for visually impaired programmers, the focus is on audio depiction [19]. The Java Speak project target user is a novice programmer, in particular someone who is entering the Computer Science major at the university level. The goal of Java Speak as presented in [5] was to use audio to help depict the structure of a Java program in a manner akin to the use of color in many development environments. As of this writing, JavaSpeak evolved into a set of Eclipse plug-ins that when used with JAWS provides audio feedback for compilation, runtime status, and the program tree as well as to aid with the focus of commonly used windows [9]. JBrick plans to use audio cues to aid in code orientation and navigation in a future release. Rather than focus exclusively on the blind, JBrick seeks to serve visually impaired programmers in addition to blind programmers.

Popular robotics programming environments designed for young, novice users are often graphical in nature. Lego's own NXT-G software (often used in schools and in the FIRST Lego League competition) consists of icons that are linked together and assigned attributes. At the time of this writing, the new version is the same, as shown in Fig. 1.

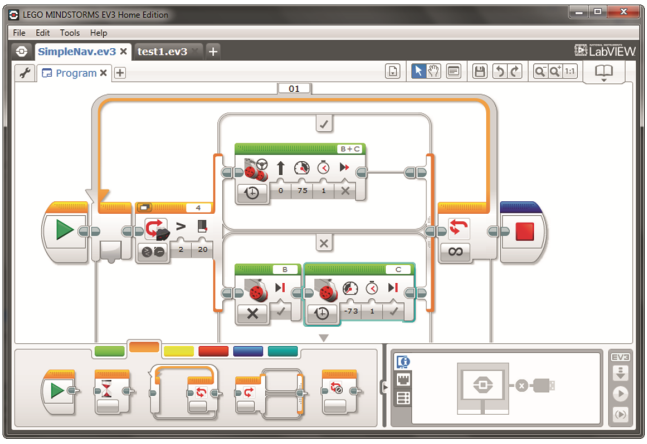

Fig. 1. Screenshot of lego NXT-G software [10]

Other visual programming environments can be found in the Microsoft Robotics Developer Studio as well as one based on Scratch [18]. As such traditional, text-based programming environments and technologies were assessed:

- LeJOS: buggy, not novice-friendly and required changing the Lego Mindstorms "brick" firmware (undesired)
- Microsoft Robotics Developer Studio's .net option: learning curve too high for the outreach, has accessibility issues
- RobotC: the IDE, which is tightly coupled to the language, has significant accessibility issues

The best solution at the time was BricxCC, developed by Hansen, and the NXC (Not eXactly C) language [7]. The evaluation for BricxCC is covered in [11, 13], and at the time the most critical need was for screen reader (JAWS) compatibility. The BricxCC software is not entirely compatible with JAWS. Assistance by a sighted person is needed at times. For example, code navigation requires assistance when the program is large as displayed code line numbers are not read. The alternative is for the user to count the lines, which can be frustrating and time consuming in a large program (at least 30 lines in this case).

In addition to the authors' ongoing work, BricxCC has subsequently been used with visually impaired students in another outreach project [3, 8] from researchers at Georgia Tech. The Georgia Tech team continues to use BricxCC though they compensate with the use of a wiimote to provide haptic feedback about the robot's status (e.g. distance to an object, distance travelled, whether the robot has bumped into something). We have taken the direction of improving the software itself in order to address accessibility gaps with BricxCC and to focus on programing rather than on the interface with the robot itself. After using BricxCC in several outreach workshops, the JBrick lead decided to design a fully accessible robotics programming environment that would also be cross-platform (Mac and Windows). Use of the NXC language remains though helper libraries are used to simplify working with the motors and sensors.

3 Software Design

JBrick is implemented in Java in order to facilitate cross-platform deployment. The NXC compiler is used, so programs implemented in JBrick and BricxCC are interoperable. Standard Java libraries have been used, including the Java Accessibility libraries.

The JBrick user interface is designed to be accessible to programmers with various degrees of vision. The screenshot of the user interface is presented in Fig. 2.

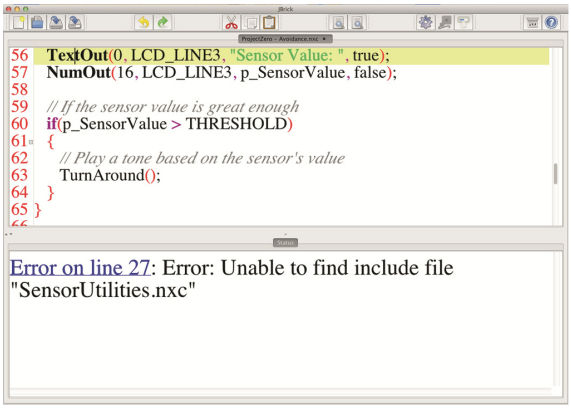

Fig. 2. Screenshot of JBrick showing larger font size and color changes

In addition to compatibility with screen readers, refreshable braille displays, and magnification software, the user interface itself was designed to accommodate both sighted and low vision users.

3.1 Code Display (Visual and Audio)

Accurate and legible depiction of the user's source code are critical to a programming environment. In the case of JBrick such depiction must be visually and audibly (when screen readers are used).

Reading code accurately means conveying the alphanumeric text and the punctuation since the NXC language uses punctuation in the same manner as the C language. Using JAWS for testing, code is read completely when the screen reader option to read punctuation is selected. As such any issue is that of the screen reader and is out of scope for JBrick directly. In addition to the code itself, line numbers are also read (see Sect. 3.3 for more details).

Programmers who are visually impaired have diverse needs and so a spectrum of accommodations are available for display customization. While some individuals may use magnification software (e.g. ZoomText), others have needs that preclude such software. For example, some programmers use accessibility options from with the operating system (e.g. screen resolution adjustment or selecting a larger pointer). Others may need enlarged text but may be frustrated by having to continuously move the magnified screen

around when reading. JBrick provides enlarged icons for those who use the toolbar, but the ability to change the font and its size as well the coloration of text and the background was designed to aid both visually impaired and sighted programmers. These features are shown in Fig. 2. To further aid the programmer the current line of code that the cursor is on is highlighted so that it stands out.

3.2 Keyboard and Mouse Feature Execution

Drop down menus and a toolbar containing large icons exists in JBrick for use by those who wish to use them. Both the menus and icons are accessible by mouse and keyboard. Both the number of menus and icons are fewer than those in the BricxCC software, which streamlines the graphical user interface and aids users who need to listen to those features names.

Navigation can also be accomplished with the keyboard as the sole input device. Keyboard support for UI navigation is present for JAWS users. Commands such as compile, download, save, and tab navigation can be accessed through keyboard shortcuts. Before user testing, these shortcuts were tested to ensure that they do not interfere with the myriad of JAWS keyboard shortcuts. While conventions for tasks such as Open and Save follow the platform conventions (Windows, Mac), domain specific tasks such as compile and download to robot are simple, single key commands (function keys).

3.3 Line Numbering

The presence of on-screen line numbering has existed in programming environments for years. Line numbers facilitates locating the line of code that corresponds to a compiler error or to simply aid in locating to a line of code. In BricxCC line numbering can be activated but a screen reader does not read them.

In JBrick, line numbering is not only visible but as the font size is changed by the user the corresponding size of the line number also changes. The line numbers are displayed in a visually separated column just of the left of the code so that the numbers are not confused with source code content. JBrick manages line numbering so that they can be displayed, read by a screen reader, and displayed on a printout. When read by a screen reader, the line number is read as “line n” rather than merely the number so that the user understands that the number is to signify the line and not just a value that may be in the code itself.

3.4 System Feedback (Visual and Audio)

Feedback informs the user of the current state of a task/process or if action is needed to move a process forward. In robotics programming, interaction with the external hardware necessitates such feedback. During the programming workflow, the user must be informed of the status of the following:

- Is the robot (Lego Mindstorms) hardware connected to and recognized by the computer?

- Was the compilation successful or not? If not successful, then what are the errors and what line(s) do the errors correspond to?
- Has the program been downloaded to the robot successfully?

The feedback for thee areas must be provided both visually and with audio. The audio (and some visual) feedback was lacking in BricxCC for these aspects of the workflow. In order to run a program, the program must be successfully compiled and then downloaded to the robot. JBrick must recognize the robot before a program can be downloaded onto the robot. Upon starting JBrick the robot is selected, but given that the robot generally needs to be disconnected and moved elsewhere to run the program, it is possible to forget to reconnect the robot or the robot may have shut down during programming in order to save battery. Such occurrences require the programmer to know that the robot is not on or connected to the computer. JBrick will alert the programmer if the robot is not detected and alerts them to this whereupon the user can then go and correct the issue. In this Find Brick feature, the programmer can clearly see or hear what the status is and when the status changes (to being detected).

A successful compilation generates a text message that is read by the screen reader, as well as a tone. In JBrick, compiler errors are displayed in their own pane and these errors are read by the screen reader. When a compiler error is displayed or read, the corresponding line in the code is highlighted and the programmer can initiate the line being read to them. This feature expedites the need to find the line that the error corresponds to.

Once a program compiles successfully and the robot is detected, it is critical that the programmer knows when the program has downloaded successfully. If there is ambiguity, then when the programmer tries to run the program there will be confusion as to whether the error was in downloading the program or in the program itself (especially when a new version of a program is being tested). When the program is downloaded successfully, the user is notified with a tone. If the program is not successful, then an error alert is provided to indicate a problem (such as a brick detection issue).

4 Evaluation

Initial evaluation with JBrick is outlined in [11]. Long-term JBrick field tests were conducted during Summer 2013. Ten participants with visual impairments ranging from moderate vision impairments to completely blind used JBrick between 3-4 h per day over the course of 4 days. The participants were teens participating in a Computer Science exploration program. The participants were self-selected and their programming experience varied with only 3 having any experience (1 of the 2 had enough functional vision to use the graphical Lego NXT-G software). For the purposes of the exploration workshop novice programmers were encouraged to participate.

Participants worked in teams of 2-3 for the duration of the activity. Groupings were random within gender. The overview of the field test consisted of assistive technology setup, training in the NXC language and JBrick via a tutorial, and applying NXC via JBrick to solve a challenge for the duration of the activity.

Experiments. Before the tutorial started, any assistive technology was configured for the participants. 2 participants used refreshable Braille displays, 6 participants used screen readers, and 3 used screen magnification software and 1 adjusted screen resolution and appearance in the Windows Display preferences. All teams had at least one monitor so that the team could see the their work and headphones were used by participants who used screen readers.

During the first day, the participants were asked to go through the NXC programming tutorial with a provided Lego robot for each team that required each participant team to add sensors as needed. The tutorial has been used in prior workshops [11]. Display options were modified for those who need customized views. The adjustments included: *Increase font size, Change colors (for text, background, current line highlight), and Change cursor size.*

Participant groups worked at their own pace with members of the project team floating to answer questions and offer guidance as needed (e.g. how to use the robot, add sensors). The tutorial walked the participants through the NXC language in small chunks, integrating the programming workflow into the lessons. Practice with using compiler – working with error messages, finding line with errors.

Participants entered commands, constructs, expressions, and variables throughout the tutorial, which consists of 3 sections. The programming workflow of design, implementation (including compilation and downloading), testing, and evaluation was practiced many times during the tutorial and subsequent activities. All teams completed the tutorial within 4 h. If teams finished early, mini challenges were added in order for the team to apply their skills and experiment with the NXC language and the Lego Mindstorms robot.

Each session (day) lasted for 4 h and within each team each participant used JBrick directly for at least an hour, with the remaining time being used to accomplish tasks such as solve a problem, design/revise/test a programming solution or design/revise the robot itself. Participants who were particularly confidant with typing tended spend more time in JBrick so that the team could test a solution faster. Time to complete a task was not measured as typing speeds and the complexity of programs disallowed any comparison. Assistance was provided if the participants had difficulty with JBrick, programming, or robot building at any point during the activity. In addition, the team floated between the groups to see how they were doing from an instructional standpoint, as well as to observe the use of JBrick.

From the second day onward, the teams worked at their own pace to design and program their robot to complete a challenge. The challenge was devised to encourage collaboration and creativity so each team's solution was different. The challenge was for the robot to aid in a search and rescue mission to locate and guide people (designated by small cubes of foam) out of an area that was impacted by a natural disaster. Incremental design was used as the environment was made more complex (e.g. line following, a more complex route with obstacles) to maintain spectrum of challenge for the participants. To devise a solution, each team had to examine the environment, design their robot and then their programmatic solution. Each team needed to use at least one sensor and at least two motors in the robot design. Program design required the use of constants, motor and sensor commands, variables, expressions, audio, and at least one if or while construct. More advanced students used subroutines.

In order to construct the program, each team used the following JBrick features multiple times: file management, cut/copy and paste text within and between files, Undo, navigate the program file to enter or revise code including the use of line numbers (enlarged or read by the screen reader), compile and download code onto the robot, locate compiler errors (line numbering, also the cursor is placed and the line highlighted for the first error), revise desired screen display preferences (font, size, text and background color, highlight color, cursor), and use of keyboard shortcuts to access JBrick features and navigation.

The research team provided guidance when needed (in all aspects of the challenge and in use of JBrick), and teams could help each other as well. On the fifth day of the event, the teams showed their solutions to the overall group, the research team and to students who were participating in other discipline activities.

After Day 4, we conducted a semi-structured interview with each team using questions that were designed to capture the participant's preferences, as well as open-ended questions to capture their likes and dislikes about the JBrick software and the programming activity. The JBrick-related questions will be presented in this paper. Those questions included:

1. What did you like about using JBrick to program the robot?
2. What would you change about using JBrick?
3. How well were you able to customize JBrick so that you could use it (e.g. screen reader, font size, etc.)?

The participants were also asked to respond to five statements on a 5-point Likert scale (1 is strongly disagree, 5 is strongly agree) in the form of a short online survey. The statements were:

S1. The line numbering enabled me to locate the line of code with the error *faster* than if I did not have the line numbering displayed/read to me.
S2. The line highlighting/focus and cursor positioning enabled me to locate the line of code with the error *faster* than if I did not have this feature.
S3. I was *confidant* in being able to follow the structure of the programs I created.
S4. I am *satisfied* with using JBrick to program the Lego Mindtorms robot.
S5. I would *recommend* using JBrick to a visually impaired friend who was interested in programming a Lego Mindstorms robot.

In addition to the written responses for the above questions and statements, observation notes were taken during the activity. The participants' programs were also saved.

5 Results and Discussion

All participants used JBrick to complete the tutorial and activities. Post activity feedback will be divided between participants who read large print and those who read Braille or electronic files (and thus used the screen reader). In addition the data was also examined by the level of programming experience. The data is summarized in Table 1.

The participants were asked their level of agreement that line numbering helped them locate code faster when locating compiler errors, as well as their level of agreement that the line highlighting and cursor placement had in locating code when fixing compiler errors. In terms of line numbering, the mean for blind participants is 4.4 while the mean for the visually impaired participants is 5. When looking at the responses by programming experience, the mean for the four experienced programmers is 5 while the mean for the six novice programmers is 3.67. In terms of the line highlighting/focus and cursor placement, the mean for blind participants is 3.6 while the mean for visually impaired participants is 4.2. The means for experience programmers and novice programmers are 4.5 and 3.5 respectively.

Observations noted that blind participants had different levels of skill with using screen readers and the keyboard layout. As such familiarity with JAWS shortcut commands had an effect in the number of attempts needed to activate the desired command (e.g. read the current line). Familiarity with the keyboard had a more significant impact as there were many times when the novice programmers who are blind would add extraneous characters, thus adding to the defect correction task. The addition of the extra characters required more of a need to use the line numbering feature than the participants who were visually impaired. The visually impaired participants immediately were able to see the relationship between line numbering and the compiler errors, and when larger programs were created late in the activity. The visually

Table 1. Overview of participant feedback for the 5 Likert-scaled statements

ID	Degree of Vision	Programming Experience	S1	S2	S3	S4	S5
1	Blind	No	4	2	3	3	3
2	Blind	No	4	3	3	4	4
3	Blind	No	4	4	4	4	4
4	Visually Impaired	Yes	5	4	5	5	5
5	Visually Impaired	Yes	5	5	5	5	5
6	Blind	Yes	5	5	5	5	5
7	Visually Impaired	No	5	4	4	4	5
8	Visually Impaired	No	5	4	4	4	5
9	Visually Impaired	No	5	4	5	5	5
10	Blind	Yes	5	4	4	4	4
		Mean (Blind)	4.4	3.6	3.8	4	4
		Mean (Visually Impaired)	5	4.2	4.6	4.6	5
		Mean (Experience)	5	4.5	4.75	4.75	4.75
		Mean (No Experience)	3.67	3.5	3.83	4	4.33

impaired participants adjusted the font size during the tutorial to ensure that the line numbers were clearly discernable. The participants with programming experience were already familiar with line numbering; a participant mentioned having the line highlighted as a positive feature (in addition to being able to change the colors of text, background).

When asked the degree of confidence in following the structure of their code, blind participants have a mean of 3.8 while visually impaired participants have a mean of 4.6. In terms of programming experience, novice programmers have a mean of 3.83 while experience programmers have a mean of 4.75. The means between both pairings are similar, indicating that there are issues with navigating the code that will need further study such as using audio differences (e.g. pitch, earcons) to aid in code orientation and navigation.

Observations of the blind participants, having sped up the reading of text sometimes became lost during code orientation in terms of constructs such as if/then and repeat blocks especially during when trying to fix errors. Of particular issue was nested if/then statements. Part of the issue was that some participants were not as familiar with the use of punctuation such as braces and brackets where locating them on the keyboard was an additional issue. Participants with vision were able to see the code layout (e.g. spacing, blank lines for clarity), however there was sometimes confusion between the curly braces and the brackets by some participants (on screen and on the keyboard). Both groups of students needed occasional assistance from the instructors though, the novice blind programmers needed extra assistance from the team or from teammates when confusion arose over how to find or fix a bug in code.

Participants were then asked to rate their level of agreement in terms of how satisfied there were in using JBrick to program their robot. The mean for blind participants is 4.0 while the mean for visually impaired participants is 4.6. The mean for novice programmers is 4.0 while the mean for experienced programmers is 4.75. For the final statement, participants were asked to what degree they would recommend JBrick to a visually impaired friend who was interested in programming a Lego Mindstorms robot. Blind participants had a mean of 4.0 while visually impaired participants had a mean of 5. In terms of programming experience, novice programmers have a mean of 4.33 while experience programmers have a mean of 4.75. Overall the experience was positive for both novice and experienced programmers, as well as both blind and visually impaired participants. This is partly due to the generally open-ended nature of the activity where participants get to decide on the robot design and the program designs along the way, including learning from mistakes and redesigning aspects as needed. In addition, for 8 out of 10 participants the opportunity to work with a robot was completely new. Thus the overall experience may have influenced the results. Regardless, the challenges that the blind participants had using JBrick did not severely impact their impressions of the software to the extent of recommending it to a peer. Some students either copied their programs to a flash drive or asked for copies of their programs to take with them even though they did not have access to a robot of their own.

6 Conclusions and Future Work

JBrick has been successful in terms of providing an accessible foundation in Lego Mindstorms NXT programming for pre-college students with visual impairments. In the context of outreach, the software allows students to collaborate with one another as well as with sighted peers during robotics activities that build technology skills.

Moving forward, further study is needed in order to improve the user experience for novice programmers who are blind. Some issues such as screen reader skill is outside the scope of JBrick. However additional features will be explored and added in order to provide students the ability to navigate code using audio cues, as well as debugging (a tool to help work through coding and logic defects). Remaining work will be completed in order to provide Mac OSX support. In addition, JBrick will be revised to accommodate the new version of Lego Mindstorms (EV3), as changes to the compiler are likely.

Acknowledgements. JBrick is supported by the National Science Foundation (#1240809). Thanks to students who have contributed to this project, as well as the participants who have provided feedback.

References

1. American Foundation for the Blind: Statistics and Sources for Professionals. (2006). from American Federation for the Blind: http://www.afb.org/section.asp?SectionID=15&DocumentID=1367Fröhlich Accessed March 1 2008
2. Cannon, K., Panciera, K., Papanikolopoulos, N.: Second annual robotics camp for underrepresented students. In: Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, pp. 14–18, Scotland (2007)
3. Dorsey, R., Park, C.H., Howard, A.: Developing the capabilities of blind and visually impaired youth to build and program robots. J. Technol. Persons Disabil. **1**(1), 55–67 (2014)
4. Emacspeak Homepage. from Emacspeak Homepage: http://emacspeak.sourceforge.net/ Accessed April 15 2014
5. Francioni, J., Smith, A.: Computer science accessibility for students with visual disabilities. In: Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education, pp. 91–95, USA (2002)
6. Georgia Tech. Technology, Engineering, and Computing Camp. 2010. from Georgia Tech: http://www.coe.gatech.edu/diversity/wietec.php Accessed January 15 2010
7. Hansen, J.: BricxCC Command Center Homepage. 2007 from BrixcCC: http://bricxcc.sourceforge.net/ Accessed January 20 2010
8. Howard, A., Park, C.H., Remy, S.: Using haptic and auditory interaction tools to engage students with visual impairments in robot programming activities. IEEE Trans. Learn. Technol. **5**(1), 87–95 (2012). doi:10.1109/TLT.2011.28
9. JavaSpeak Project Homepage. from: http://cs.winona.edu/cscap/javaspeak.html Accessed April 30 2014
10. Lego NXT-G EV3 screenshot. from Curious-on-Hudson Homepage: http://curiousonhudson.com/class-details.php?id=223 Accessed April 20 2014
11. Ludi, S.: Robotics Programming Tools for Blind Students. J. Technol. Persons Disabil. **1**(1), 77–89 (2014)

12. Ludi, S., Reichlmayr, T.: Developing inclusive outreach activities for students with visual impairments. In: Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education, pp. 439–443, USA (2008)
13. Ludi, S., Reichlmayr, T.: The use of robotics to promote computing to pre-college students with visual impairments. ACM Trans. Comput. Educ. **11**(3), 1–20 (2011)
14. Marghitu, D.: Computer Literacy Academy for Children Homepage. from Auburn University 2008. https://fp.auburn.edu/comp1000/SummerOutreach/index.html Accesed January 22 2010
15. National Federation of the Blind. 2009. Youth Slam 2009 Homepage. from National Federation of the Blind: http://nfbyouthslam.org/ Accesed January 17 2010
16. National Science Foundation. 2010. Recent Awards for the Broadening Participation in Computing Program from National Science Foundation: http://www.nsf.gov/awardsearch/ Accessed January 22 2010
17. Rarnan, T.V.: Emacspeak - Direct Speech Access. In: Proceedings of Assets 1996, pp. 32–36, Vancouver April 11–12 (1996) http://cs.cornell.edu/home/raman/emacspeak/publications/assets-96.htrnl
18. Scratch Modification for NXT (Enchanting). Scratch Wiki from Scratch Wiki: http://wiki.scratch.mit.edu/wiki/Enchanting Accessed March 25 2014
19. Smith, A., Francioni, J., Matzek, S.: A java programming tool for students with visual disabilities. In: Proceedings of Assets 2000, USA (2000)
20. USFIRST.org. FIRST Homepage. from USFIRST: http://www.usfirst.org Accessed July 12 2009