Using Voice as an Input Modality

Zane Johnston
Kennesaw State University

Contents

Using Voice as an Input Modality

Background Information

Computer applications allow for users to perform a wide range of complex tasks and actions. Unfortunately, many computer applications employ graphical user interfaces (GUIs) to achieve their high levels of functionality. While the text and images used in GUIs allow for these more complex actions to be performed, they rely on the users' ability to use inputs, such as the mouse and keyboard, to effectively make use of the GUIs full potential. This reliance on dexterity causes users with mobility impairments to have difficulty using these applications, barring them from activities in which computer use is a necessity and limiting their opportunities for employment and skill development.

Voice as an input modality offers a way for users with mobility impairments to interface with applications that use a GUI. In this paper, I will discuss the vocal user interface (VUI), which is an interface that takes voice commands and performs actions normally achieved through the use of a mouse or a keyboard, Myna, a VUI built with Java for the Scratch 1.4 initial programming environment (IPE) (Wagner et al, 2012). I will also make an argument throughout this paper that voice is a viable option as an input modality and is something that should be further explored.

Scratch is both a development environment and a block based programming language, which is a language that is broken up into blocks removing the need for one to memorize a syntax and instead allowing one to focus on the logic of the language (Scratch, 2017). Created by the Massachusetts Institute of Technology, Scratch is meant to help children to begin learning how to program by dragging and dropping code blocks to form meaningful structures (2017). Myna runs parallel but separately from Scratch, so no information is shared between the two

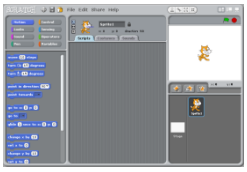
applications (Wagner et al, 2012). While this implementation does allow the ideas and concepts of Myna to be adapted and applied to other applications in general making the production of other VUIs possible, it does add some problems when attempting to achieve the full functionality of the underlying GUI application.

My work with Myna particularly involves Myna's inability to scale with screen resolution leaving some components unreachable at resolutions other than the default 1366px x 768px resolution; to recognize, scroll, and track a scrollbar making some code blocks unreachable and long programs impossible; and to navigate dialogs and prompts created by Scratch making file retrieval not feasible. The next section will describe solutions to these problems.

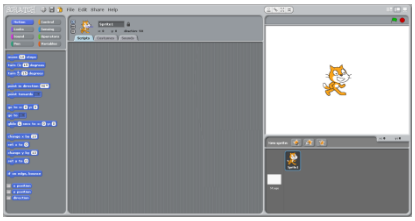
Addressing Myna's Problems

Adjusting Component Mapping for Screen Resolution

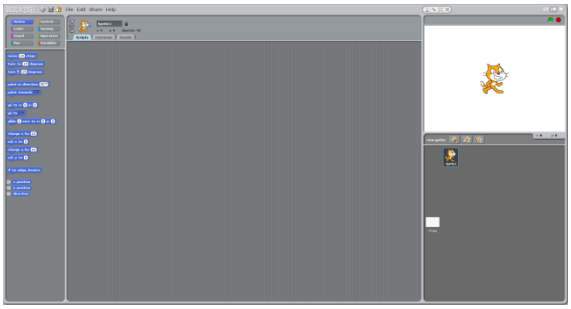
Myna was originally implemented for a system with a screen resolution of 1366px x 768px (Wagner et al, 2012); the use of Myna outside of this screen resolution would result in it being unable to successfully act upon user commands, particularly commands that required actions on components on the right side of the screen. Essentially, if the screen resolution was smaller than the default resolution, Myna would go off the screen in attempting to reach some components, while Myna would not go far enough on the screen if the screen resolution was larger than the default. These problems occur because Scratch attempts to adjust itself to the screen resolution by contracting and expanding the editor area, in turn moving components to the right of the editor. Figure 1 shows a comparison of three different screen resolutions of Scratch; the editor area changes in all three – smaller in Figure 1a, default in Figure 1b, and larger in Figure 1c – while the stage only changes from its default large state to small in Figure 1a.



a) 800px x 600px, editor area is much smaller and stage is in small state



b) 1366px x 768px, default



c) 1920px x 1080px, editor area is much larger, stage stays the same

Figure 1. Scratch Screen Resolution Comparison

In order to adjust for changes in the screen resolution, each of the components on the right of the editor will need to be repositioned. In order to do this, a formula is applied:

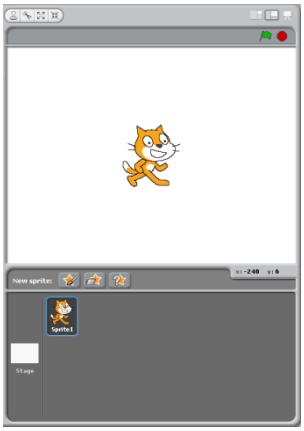
newX = newScreenWidth – (defaultScreenWidth – oldX)

newY = newScreenHeight – (defaultScreenHeight – oldY)

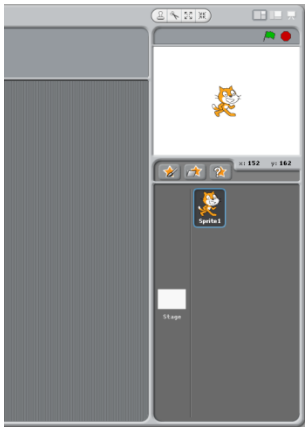
These formulas determine how far from the right, in the case of X, or from the bottom, in the case of Y, that a component is on the default screen resolution. This value is then subtracted out of the new screen resolution in order to determine the adjusted position. In the case of Myna, however, the Y formula is not used often due to most components not moving up or down when Scratch adjusts the editor area.

These formulas do not account for position changes relative to another component's size changing, only the screen resolution's size changing. In Myna's case, the stage area, where the sprites and other images are placed, is a component that affects the positions of other components based on its size. Depending on the screen resolution and the user's settings, the size of the stage may vary between full, about 480px x 360px, and small, about 240px x 180px. The components above the stage move left and right based on the stage's size, while the components

beneath the stage move both up and down and left and right; this movement is illustrated in Figure 2. In order to compensate for something like this, further adjustment by some constant value is needed, which in this case is about 240px for X and 180px for Y. However, these values only need to be applied if the stage is outside of its default state, which is full size; otherwise the formulas above are sufficient.



a) Default, large stage



b) Small stage, the components surrounding the stage shifted in response

Figure 2. Movement of components relative to the stage state.

This solution was implemented in Myna by creating a ScreenResolution class that provides methods for repositioning each component as each component is used – not all components are repositioned at the start of Myna. The class first begins by comparing the current screen resolution with the default screen resolution to determine whether components need to be repositioned. If it is found that the current screen resolution is the same as the default screen resolution, the reposition method exits on call, making no changes to any component; however, if it is found that the current screen resolution is in any way different than the default screen resolution, the method then goes into a hierarchy of conditional statements.

The next condition checks the state of the stage, whether the stage is full or small. If it is found that the stage is full, the default state, no changes are made on behalf of the stage, but a condition is checked to see if the component is on the right side of the screen. If it is found that the component is on the right side of the screen, the formula above is applied to the X values of the component's positions. Then it must be determined whether or not the component is located at or beyond 768px vertically – in Scratch's case, these components will move up or down based on Scratch adjusting the editor area as the screen resolution changes; the above formula for the Y position adjustment is applied to the component if it is found to meet this condition.

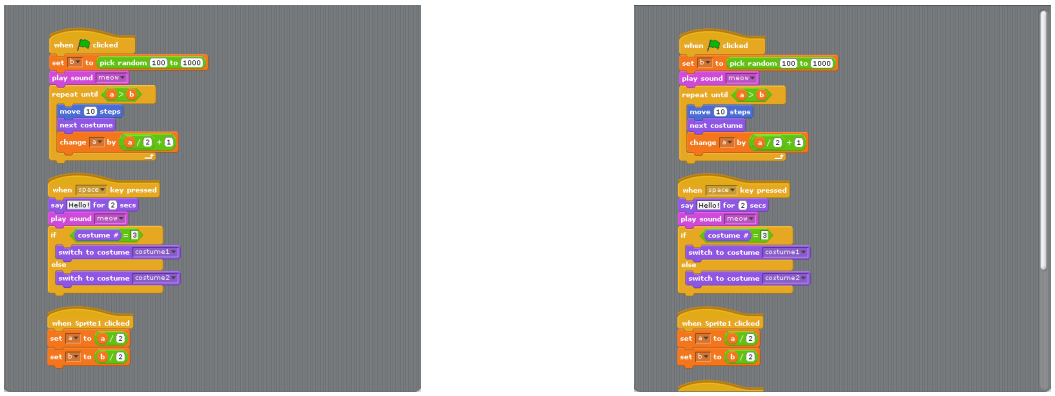
If the stage is found to be in the small state, the method progresses almost exactly as it would normally; however, the component is checked to determine whether or not its position is dependent on the stage's size. If the condition is true, the constant values mentioned above are used – 240px are added to the X position in order to move the component right and 180px are subtracted from the Y position in order to move the component up. If the condition is false, no further changes are made to the component aside from applying the formulas.

Moving, Tracking, and Updating the Scrollbar

The scrollbar poses a twofold problem: first it must be determined whether or not a container has a scrollbar and then the scrollbar has to be moved the correct amount in order to move the components within the container the desired amount.

In order to determine whether or not a container has a scrollbar, the dimensions of the container and of the elements within the container must be tracked. Throughout the process the dimensions of the container are kept the same while the dimensions of the components are updated as components are added or removed from the container. At any point where the dimensions of the components are equal to or greater than the dimensions of the container, there

is a scrollbar; if the width of the components are greater than or equal to the width of the container, a horizontal scrollbar exists, while if the height of the components is greater than or equal to the height of the container, a vertical scrollbar exists.



a) Elements have not surpassed the container, no scrollbar.

b) Element height is beyond the editor, vertical scrollbar.

Figure 3. Visual of a scrollbar becoming necessary in the editor area.

After a scrollbar has been determined to exist in one of the directions, the amount that the scrollbar needs to be moved to scroll the container elements a desired amount can be found using:

$$\text{scrollAmount} = \frac{\text{ContainerDimension}}{\text{ElementDimension}} \times \text{desiredElementMovement}$$

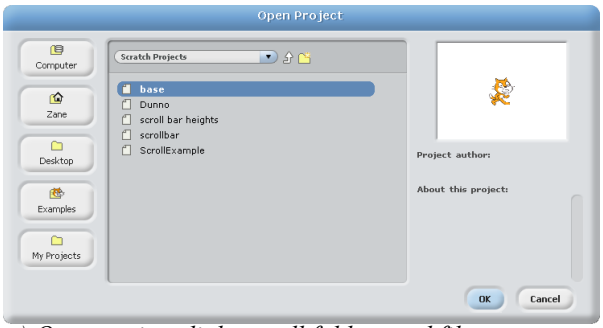
This formula uses a ratio of the aforementioned dimensions of the container and the contained elements and then multiplies the desired element movement amount to determine how much the scrollbar needs to be moved. Once the scroll has occurred, the positions of the element within the container will need to be updated using the value of desiredElementMovement. Moving in the opposite direction, such as left on a horizontal scrollbar or up on a vertical scrollbar, would use the same formula.

For Myna, this solution is implemented with two scrollbar classes, EditorScrollbar and CommandScrollbar, each for the two containers that have a scrollbar. Both classes contain variables for the location where the cursor goes to click and drag the scrollbar and the potential

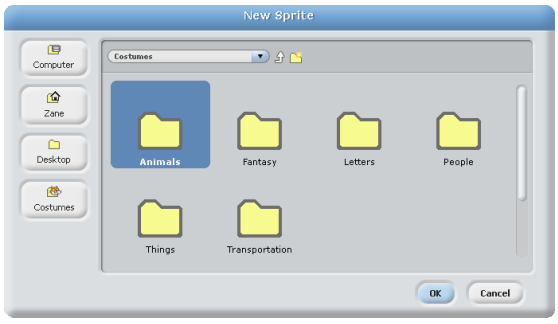
movement in any direction. The dimensions of the containers, the editor and the command palette respectively, are also stored in these classes and are updated if the screen resolution causes them to change from the default dimensions. Methods for calculating the scroll potential using the method described in the first part of this section and the resetting of the scrollbar when components that reset Scratch's scrollbar are also included. The method for scrolling is contained within the CommandExecutor class and implements the formula for determining the scroll amount; this command is executed when the user says "scroll editor up/down/left/right" or "scroll command up/down/left/right".

Navigating Prompts and Dialogs

As mentioned before, Myna is completely separate from Scratch, so no communication of the state of Scratch or the positions of elements are provided to Myna from Scratch; this means that Myna has no way of knowing that a dialog has been created, much less the positions of the elements within that dialog. In the case of a file explorer dialog, the situation is further complicated due to the fact that the names of files and directories are user defined, so names cannot be accurately premapped to components for voice commands. Figure 4 presents examples of such dialogs.



a) Open project dialog – all folders and files are user defined



b) New sprite from file dialog – default folders and sprites, but user may add their own

Figure 4. Example of dialogs with user defined elements not suitable for premapping names.

In order to allow for the user to navigate these dialogs and prompts, the user must be allowed to navigate it his or herself – normal voice commands cannot be used. This can be accomplished through seamless navigation (Wagner et al, 2012) by giving the user control over the mouse movements and allowing them to move the cursor to the desired element within the dialog.

The current implementation of this solution is to have commands in place for the user to specify movement in any direction – up, down, right, and left – in which the cursor moves in the specified direction some set amount, which is currently 40px. Commands are also provided for positioning the cursor in regions of the screen, such as top left, bottom right, center, center right, etc. While this solution works and does allow users to navigate the dialogs created by Scratch successfully, the set amount for the cursor movement can lead to the movement being tedious, with multiple uses of this command needed to get to a target location, and imprecise, with the cursor over- or undershooting the target location with no way to land on the location exactly.

Alternative solutions that may help to alleviate these problems are being explored. In particular, one solution is for the user to specify a direction for movement and then to give a stop command once the cursor has reached the desired location; however, this solution could be problematic if there is any delay between a user giving a command and the command being acted upon, which would lead to the cursor movement to be inaccurate and hard to control. Another solution is for the user to specify both a direction and a movement amount. For example, the user could say “move cursor right 30” or “move cursor left 100” in order to vary the movement for the context; however, a problem with this solution is that it may be difficult for a user to understand how much a given value of movement will move the cursor, but this is something that will become easier as the user becomes accustomed to using the application.

User Testing

In order to gain feedback on these solutions and on Myna in general, user tests with 10 able-bodied users with no prior experience with Myna or Scratch were conducted from January to February 2017. Of the users, three were graduate students within the sciences, six were undergraduate Computer Science majors, one was an undergraduate honors student, and all ranged from 20 to 24 years of age. Each user was allowed time to get accustomed to Scratch and Myna with a 15 minute training session in which they were provided instruction from an expert user and a sheet of the most common voice commands. Once the user tests began, the sheet containing the commands was taken away and the user had to rely on memory and a help menu within Myna.

The user tests consisted of each user completing three benchmark programs with a mouse and keyboard, then completing three more benchmark programs with Myna, and finally completing a survey detailing the user's impressions and thoughts on the solutions. With each successive program in Myna, the screen resolution was gradually decreased from the default resolution for program 1 to 1280px x 720px resolution in program 2 and finally to 1024px x 600px resolution in program 3. Observations on the user's performance and Myna's ability to perform actions were also documented. The documents used for the user studies can be found in Appendix A through F. All user tests were conducted on a HP Envy m6 laptop with Windows 10, Scratch 1.4, with Myna running in the Eclipse IDE, and an Insignia™ NS-PAUM50 USB microphone.

Table 1 presents the average response from the users to five statements regarding the ease of use and functionality of the solutions. It is important to note that these questions and statements are meant to reflect how the user perceived the action – the results may not reflect

what actually happened during the tests. For example, some users marked that Myna did not always adjust appropriately to the screen resolution; however, there were no observed instances in which Myna attempted to access a component that was not repositioned correctly. Despite this, the solutions were well received overall with the users finding the commands to be predictable and the functions to work as expected; however, some users found that the fixed movement amount for the cursor movement commands negatively affected the usability of the implemented solution. Many users suggested replacing the current implementation with the alternative methods mentioned in the above section.

Table 1

User Responses to Survey Statements

Statement	Average User Response
Myna appropriately adjusted to the screen resolution	4.73 (Always)
Scrolling the editor and command palette worked	3.91 (Most of the time)
The commands for scrolling were	2 (Somewhat predictable)
The cursor movement commands moved the cursor	3.55 (Enough but sometimes too much)
The cursor movement commands were	1.18 (Predictable)

The observations taken during the user tests focus on the ability of the user to become accustomed to Scratch, which includes learning the locations and functions of the code blocks and the names of particular components within Scratch; the ability of the user to learn the voice commands in Myna; and the ability of Myna to do the requested actions properly.

The data from these observations suggest that the greatest hurdle for usability is the voice recognition system, which is affected by the environment in which the user is working and the microphone that the user is using. The average occurrence of each observation over the three Myna programs per user is presented in Table 2, but the data regarding the observations for the xy-location being incorrect on screen and the scrollbars not scrolling properly are not included because these events were not observed in any of the trials. As users became accustomed to using Myna's commands and to finding the correct blocks in Scratch, the amount of errors in voice commands and Scratch commands decreased; this is something that is good to see as it helps to show that the commands are predictable enough so that the users can learn the commands fairly easily as they progress. The low amount of needing to start over is reassuring as it means that Myna is working properly enough so that any errors or incorrect actions by Myna are oftentimes not detrimental enough to ruin the user's program to an unrecoverable state; the fact that most of the trials that were repeated were only repeated due to the microphone not being responsive also encourages this perception.

Table 2

Average Occurrence of Observation Per User

Observations	Average Occurrence
Participant stated incorrect vocal command	1.03
Participant started incorrect Scratch command	0.53
Voice recognition was inaccurate	3.47
Number of times needing to start over	0.3

Another interesting observation made during the user tests is how long it took each user to complete both the mouse and keyboard programs and the Myna programs. Presented in Table 3, the average completion times suggest that using Myna takes about double the time of using standard manual inputs. While doubling the time for a user using the keyboard and mouse may seem suboptimal, it is expected that Myna will take longer and, because the users being targeted by Myna do not have other options for interacting with applications otherwise, the time disparity is something that is likely going to need to be accepted regardless. However, this does not mean that the time disparity should not be lessened if possible. When comparing the average completion times and the average times per line, it is important to note that these users are beginners with both Scratch and Myna having no prior experience with either. It is also important to remember that the Myna completion times are being increased by the currently implemented cursor movement command that many users have found to be tedious and imprecise; implementing the alternative method where the user specifies the movement amount would likely greatly decrease the time needed to complete the programs in Myna.

Table 3

Average Time To Complete Programs

Time to Complete	Average Time (s)
With mouse	87.03
With Myna	176.63
With mouse (per line of code)	16.50
With Myna (per line of code)	36.92

User tests with users with mobility impairments are still being scheduled and, while none have been administered so far, it is hoped that their feedback and insight will greatly help in both evaluating Myna and improving its functionality. These users are the target audience, so their feedback is especially valuable.

Based on the feedback received so far and the fact that most users seemed to enjoy using Myna despite having difficulty in the beginning, it seems that Myna and other VUIs do have a future in user interfacing. While there are definitely areas in which voice can be improved upon, particularly the reliability of speech recognition, voice as an input modality does seem to be viable.

Significance of Myna

The solutions to these problems are not only applicable to Myna; any developer who is making a VUI or is attempting to add functionality to their application using positional data can use them. Because Myna is implemented without communicating with Scratch or making use of any API, the solutions and implementations are general enough that they can be taken and adapted to any other application. While Myna's initial goal is to allow children with mobility impairments to have the opportunity to learn programming through Scratch and enter the world of Computer Science, the overarching goal is to allow all users with mobility impairments greater access to computer applications, not just those that focus on Computer Science education.

Other Uses of Voice as an Input Modality

Other developers and researchers have implemented ways of using voice as an input modality differently than through a VUI. This section describes those methods, user impressions, and functionality data where possible.

The Vocal Joystick

The developers of the Vocal Joystick were not satisfied with what most speech recognition systems offered, arguing that their focus on syllables and words was suboptimal for interacting with computers and did not allow for more subtle control for complex actions, so they created a voice-based interface that uses the energy, pitch, and vowel quality of a user's voice to detect user activity, control the speed, and to control the direction of a mouse cursor (Bilmes et al, 2005). The Vocal Joystick is modularly designed so that others can customize what libraries and noises are used and adapt the functionality to their own systems. There are also several different forms of the system, one that offers cursor movement in the four directions, up/down/left/right; one that allows the user to vary their pitch and loudness more; one that offers movement in the four main directions as well as the in-between diagonal directions; and one that includes all directions and allows for more variance in voice input (2005). Figure 5 illustrates the vowel sound-direction mapping used for the Vocal Joystick.

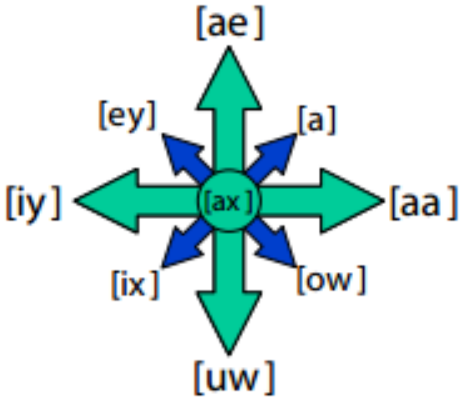


Figure 5. Vowel sound-direction mapping (Bilmes et al, 2005).

A user study was held by the developing team with able bodied users in order to obtain preliminary data about the functionality and ease of use of the Vocal Joystick. With no prior knowledge of the interface, seven users attempted two tasks; the first was to navigate to a

particular web page from a specified starting page by following links, and the second was to navigate to a specified school campus on a map after starting zoomed out to a view of the United States (2005). For each task, the user was given an initial instruction by an experienced user. Then completion time, the amount of false positive clicks, the amount of missed clicks, and the amount of user errors were recorded. Users were also given a 14 question survey detailing their impressions and experiences using the system (2005). It was found that the system seemed to work better for the female users than for the male users; this was attributed to the fact that the system was initially calibrated by a female voice. The time to complete the tasks with the Vocal Joystick compared to using the mouse showed that the Vocal Joystick requires a significant amount of more time than with the mouse – about 4 to 9 times more time; however the developers are confident that improvements can be made (2005). Overall the user impressions were good; the users seemed to enjoy the system and the only major complaint was that more practice time would have likely increased the user's performance (2005).

Another user study with the Vocal Joystick was conducted over the course of two and a half weeks by Harada et al. However, instead of focusing on the usability of the interface, this study focused on the ability of the users to learn to use the interface and how these users learned (Harada et al, 2009). The study was conducted with nine participants, five of whom were physically impaired, over ten sessions in which the users were introduced to the system, were tested on their ability to recall the vowel mappings, were allowed to practice with the tool, and were given two types of tasks (2009). The first type of task was the target acquisition task, which involved a practice round similar to the link phase the developers of the Vocal Joystick conducted in their tests and a round in which the user is asked to repeatedly click two bars separated by a set distance (2009). The next phase was the steering task, in which the user first

practiced by drawing figure eights with the cursor and then went on to navigate through circular tunnels that varied in their width and radii (2009). It was found that all users could recall the sounds that were required for specific directions; however, some users found it difficult to actually create the vowel sounds, but users did seem to improve in creating the sounds with each passing session (2009). Through comparing the performance of some users with the Joystick to their performance with their preferred manual input device, it was found that the Joystick has the potential to approach the effective functionality of manual devices without needing to actually do any manual manipulation (2009). Overall the researchers concluded that the Joystick has great potential in the area of computer accessibility and that many of the users showed promising potential to learn how to use the system despite facing some difficulties at the beginning (2009).

It is interesting to see developers and researchers attempting to make use of syllables and otherwise meaningless sounds for different functionalities in user interfacing. It seems that many want to focus on using spoken language as the primary method of interfacing, but they never consider the amount of control that the nuances of small sounds can give a user. In the case of the Vocal Joystick, using vowel sounds is particularly interesting due to the amount of control and fluidity speaking vowels has, which allows unique and potentially continuous movement in curves and other paths that are not as easily obtained through using spoken language commands. While I do not think it is the way I want to implement cursor movement in Myna, this method of control would likely be very beneficial to applications needing the fluid, subtle control that it offers.

VoiceCode

VoiceCode is an application that allows for programming with the use of natural language, in particular it offers code dictation, code navigation, and error correction (Désilets et

al, 2006). Instead of requiring the user to dictate each piece of a programming language's syntax, such as curly braces, semicolons, and indentations, VoiceCode interprets natural language and then converts it to the programming language. The application considers the context special keywords, such as "if", considering them a part of a variable name if the context does not suit keyword being used (2006). Navigation of the code is handled in several different ways; navigating through an if statement or some other type of structure can be done through commands like "then" and "next", while navigating through several lines of code can be achieved by navigating through the symbols that are a part of the syntax such as "next comma" or "previous semicolon" (2006). Error correction is handled by allowing the user to tell the system whether or not the rendered code is correct or correct, but written improperly, allowing for the use to correct errors in syntax and variables' names (2006).

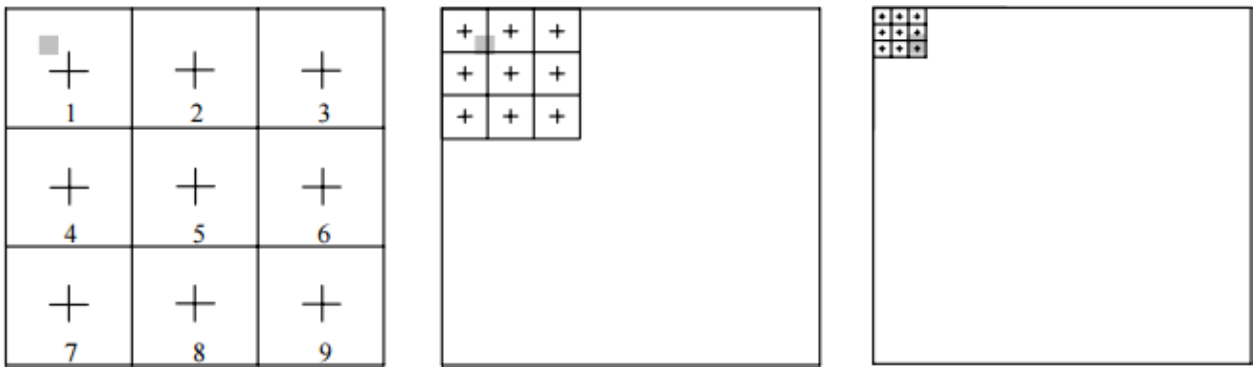
While no user studies of the system were provided in this article, it does seem that VoiceCode would have a lot of potential in allowing those with mobility impairments easier access to structured programming languages. The way that VoiceCode allows spoken language to be easily translated into programming language is almost awe-inspiring – from the perspective of a Computer Science major it can be difficult at times to explain the structure of one's code to one's colleagues, so speaking code to a computer sounds daunting at first. However, this shows that voice has a lot of potential uses in interfacing, especially in the Computer Science field.

Cursor Control with a Grid

Dai et al rejected the use of direction based cursor movement, claiming that it is too inefficient to warrant using despite its ability to allow the cursor to move anywhere on the screen, so they developed a system in which a grid marks regions of the screen (2004). As the user calls for the cursor to be moved to one segment of the grid, the grid recursively shows itself within the

selected segment allowing for the user to continue to specify regions within that segment (2004). This process continues until the user has narrowed down the cursor location precisely enough to click on the desired target location (2004).

In an attempt to find the most efficient solution, two grid systems were developed – one in which cursors were placed in every cell of a 3 x 3 grid and each successive grid recursively and another in which only once cursor exists within the 3 x 3 grid (2004). The major difference in the two grids is how the user interacts with them. The grids operate by allowing the user to zoom in by specifying the cell that contains the target location in each iteration; once the cursor has reached a precise enough location, the user only needs to give the order to click. (2004) Figure 6 shows a mockup of the grid system with cursors in each cell of the 3 x 3 grid.



a) The original 3 x 3grid with cursors in the center of each cell

b) The grid zoomed in one time, showing another 3 x 3 grid

c) The grid zoomed in a second time.

Figure 6. A 3 x 3 grid system with a cursor in each cell (Dai et al, 2004)

User tests were conducted with 24 students to determine the efficiency of the two grid systems. Each participant was given 50 tasks in which they were asked to select objects of varying distance and size for training (Dai et el, 2004). For the actual testing, the users were asked to select squares of different sizes using their specified grid type, to select 10 squares of random size, and then to fill out a survey. Using the average error rates and average times to

reach the objects for each object for each grid type, it was found that the 9-cursor grids had faster access times but the 1-cursor grids have less average errors (2004). The results from the survey showed that the users were generally happy with the functionality, speed, and efficiency of the grid systems, but the results were slightly in favor of the 9-cursor grids (2004).

I find this implementation of voice as an input modality particularly interesting because it applies to one of the problems I sought to fix with Myna and it is one of the solutions proposed to me by some of the users in the user tests. However, while many users had suggested the grid, none had thought of having the grid zoom in recursively for the extra precision. In hindsight, I would have considered this method for implementing cursor movement for dialog navigation, but would have likely been put off by the clutter caused by the grid being on the screen; I would likely still choose to go the direction I went if only to avoid having the underlying GUIs view covered in visual noise by the grid.

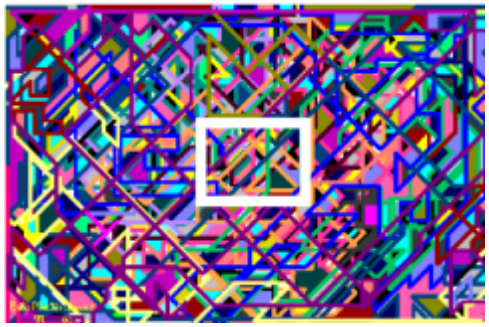
VoiceDraw

In 2007, Harada et al sought to create an application that would allow those with mobility impairments to express themselves through the creation of art (Harada et al 2007). Making use of the Vocal Joystick, the application allows the user to move a brush along a canvas in a path defined by the vowel sounds the user makes with varying stroke thickness based on the user's volume (2007). It allows for art creation without manually making a stroke with a mouse or any other manual input. Settings can be changed by issuing the same vowel sounds that are used for brush movement and the same "ch" sound that is used for placing down the brush; stroke thickness, speed, and color can be edited in this way (2007).

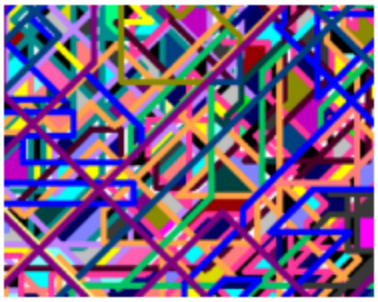
There are several interesting implementations within VoiceDraw. In particular, VoiceDraw allows for a user to incrementally erase a brush stroke instead of always erasing an

entire brush struck after making an error; this is accomplished by issuing an erase command and then using the vowel sound “a”, which signifies up, until the correct amount of the stroke is removed (Harada et al 2007). This feature was made possible by the developers implementing the brush strokes as a collection of scalable vectors (2007).

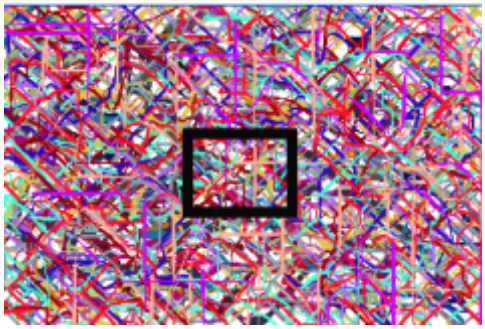
In order to evaluate the effectiveness of VoiceDraw, the developers had an artist with mobility impairments perform two tasks. The first task was for the artist to paint a predefined scene using the software he currently used, Microsoft Paint and Dragon Dictate, and then also with Voice Draw (Harada et al, 2007). The next task gave the artist complete artistic freedom to create whatever he wanted; Harada et al watched and made observations as the artist worked. Figure 7a shows the work created at some earlier point by the artist using Microsoft Paint; it took about nine hours to complete (2007). Figure 7c shows the piece created in task 2, which was inspired by 7a; however, figure 7c took only three hours with VoiceDraw (2007).



a) A piece created by an artist using Microsoft Paint



b) Zoom in on a



c)Another piece created by the same artist as a, but created with VoiceDraw

d) Zoom in on c

Figure 7. A comparison of two artworks, one created with Microsoft Paint and the other with VoiceDraw, both created by the same artist (Harada et al, 2007).

Comparing the two pictures in Figure 7, Harada et al felt that the application was an overall success because the artist was able to use it to make a piece of art, achieving the goal of being able to express oneself using only voice (2007). The artist that was used in the consultation for VoiceDraw's design and who performed the two evaluation tasks expressed his satisfaction with the application and its ability to allow him to produce different art than he was capable of making with Microsoft Paint, particularly that the lines varied in size and direction more than they used; the artist was also able to memorize and produce the vowel sounds necessary for the brush movements despite only have a relatively short amount of exposure to the application, which suggests that learning how to use the application should not be a large hurdle for users (2007).The developers were also happy to see that the artwork created with VoiceDraw took a significantly less amount of time to create than the artwork created in Paint – almost one third the amount of time (2007). They also allowed children and their parents to create artworks of their own after a few minutes of training; the artworks, while minimal, showed that the application was usable by all ages even with only a few minutes of exposure to the functionality (2007).

I find VoiceDraw to be an exciting application that showcases the potential of using voice as an input modality. Everyone should have some way to express his or herself, and, thanks to applications like these, eventually users with mobility impairments will have many different applications for artistic expression. It is also nice to see the Vocal Joystick being applied to other types of problems, even though the brush movement is essentially just another form of cursor movement.

Conclusion

While a small sample, Myna and the other types of interfaces discussed in this paper that use voice as an input modality show that voice is a flexible medium; any noise from spoken language to monosyllabic grunts can be used to interface with a system and allow for functionalities ranging from moving a cursor to generating another language. Despite having some difficulties during user testing, many users seemed to be open to using voice, to enjoy using voice, and to be able to learn the commands with some practice. The major problem with these interfaces does not seem be that the functionality or usability is not good for enough, but instead that voice recognition is not where it needs to be in order for a user to consistently use one of these interfaces effectively despite good, quiet workspaces and quality microphones. Users with mobility impairments need a way to interface with computer applications, and while it definitely has some ways to go, voice as an input modality has a lot of potential to be the chosen input modality for these users.

References

Bilmes, J.A., Li, X., Malkin, J., Kilanski, K., Wright, R., Kirchhoff, K., Subramanya, A., Harada, S., Landay, J.A., Dowden, P., and Chizeck, H. The Vocal Joystick: A voice-based human-computer interface for individuals with motor impairments. In Proc. HLT/EMNLP 2005, ACL (2005), 995-1002.

Dai, L., Goldman, R., Sears, A., & Lozier, J. (2004, October). *Speech-based cursor control: A study of grid-based solutions*. Paper presented at the ACM SIGACCESS conference on Computers and Accessibility, Atlanta, GA, USA. doi: 10.1145/1029014.1028648

Désilets, A., Fox, D. C., & Norton, S. (2006). VoiceCode. CHI '06 extended abstracts on Human factors in computing systems - CHI EA '06. doi:10.1145/1125451.1125502

Harada, S., Wobbrock, J. O., & Landay, J. A. (2007). Voicedraw: a hands-free voice-driven drawing application for people with motor impairments. *Proceedings of the 9th international ACM SIGACCESS conference on Computers and accessibility - Assets '07*. doi:10.1145/1296843.1296850

Harada, S., Wobbrock, J., Malkin, J., Bilmes, J., & Landay, J. (2009, April). Longitudinal study of people learning to use continuous voice-based cursor control. Paper presented at the ACM SIGCHI International Conference on Human Factors in Computing Systems, Boston, MA, USA. Doi: 10.1145/1518701.1518757

Scratch. (2017). Retrieved February 20, 2017 from Massachusetts Institute of Technology: http://scratch.mit.edu.

Shaik, S., Corvin, R., Sudarsan, R., Javed, F., Ijaz, Q., Roychoudhury, S., . . . Bryant, B. (2003). SpeechClipse. *Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange - eclipse '03*. doi:10.1145/965660.965678

Wagner, A., Rudraraju, R., Datla, S., Banerjee, A., Sudame, M., & Gray, J. (2012). Programming by voice: a hands-free approach for motorically challenged children. *Proceedings of the 2012 ACM annual conference extended abstracts on Human Factors in Computing Systems Extended Abstracts - CHI EA '12*. doi:10.1145/2212776.2223757

Appendix A

SIGNED CONSENT FORM

Title of Research Study: Myna Updated Feature and Usability Study

Researcher's Contact Information: Zane Johnston, 706-844-2601, zjohnst2@students.kennesaw.edu

Introduction

You are being invited to take part in a research study conducted by **Zane Johnston** of Kennesaw State University. Before you decide to participate in this study, you should read this form and ask questions about anything that you do not understand.

Description of Project

The purpose of the study is to evaluate your experience using a Vocal User Interface, called Myna, to program Scratch (a block-based programming language) by voice.

Explanation of Procedures

Your participation in this research study consists of the following:

1. Learning how to use Myna;
2. Writing three specific programs in Scratch using traditional methods (mouse);
3. Writing three specific programs in Scratch using Myna (voice); and
4. Completing a post-survey regarding your experience.

Time Required

The experiment should take less than one hour from beginning to end.

Risks or Discomforts

Participation in this research study is completely voluntary.

Benefits

While there are no direct benefits to you for participating in this research study, the researcher may learn about the effectiveness of programming by voice in a block language opposed to using the mouse and keyboard.

Compensation

There will be no compensation for participating in this research study.

Confidentiality

The results of this participation will be anonymous. Please do not include your name on the survey to ensure that it remains anonymous.

Inclusion Criteria for Participation

In order to participate in this study, you must be at least 18 years of age.

Signed Consent

I agree and give my consent to participate in this research project. I understand that participation is voluntary and that I may withdraw my consent at any time without penalty.

_____

Signature of Participant or Authorized Representative, Date

_____

Signature of Investigator, Date

PLEASE SIGN BOTH COPIES OF THIS FORM, KEEP ONE AND RETURN THE OTHER TO THE INVESTIGATOR

Research at Kennesaw State University that involves human participants is carried out under the oversight of an Institutional Review Board. Questions or problems regarding these activities should be addressed to the Institutional Review Board, Kennesaw State University, 585 Cobb Avenue, KH3403, Kennesaw, GA 30144-5591, (470) 578-2268.

Appendix B

Myna Commands

1. Change Command Menu – “Motion”, “Control”, “Looks”, “Sensing”, “Sound”, “Operators”, “Pen”, “Variables”
2. “Drag and Drop _____” – Drags the element into the editor region and drops it.
3. “Drag _____” – Drags the element.
4. “Drop after _____” – Drops the element after a specified position.
5. “Drop before _____” – Drops the element before a specified position.
6. “Drop in _____” – Drops the element inside of another elements at the specified position.
7. “Play” – Executes the program if it contains a “When Clicked” block.
8. “Edit _____ at _____” – Edits the given parameter (1, 2, 3) of the element at the specified position.
9. “Number _____” – Types the stated number. “Number two” will type “2’. For multiple numbers, such as 29, say, “Number two, Number three”.
10. “Letter _____” – Types the stated letter. “Letter a” will type “a”. For multiple letters, such as ac, say, “Letter a, Letter c”.
11. “Clear” – Deletes the program and clears the screen.
12. “Undo” – Undoes the last action.
13. “Pause” – Pause the voice recognition system.
14. “Resume” – Resume the voice recognition system.
15. “Click _____” – Clicks the specified element. If no element is specified, the click occurs at the cursor’s current location.
16. “Double click” – A double click is performed at the cursor’s current location.
17. “Scroll _____ _____” – Scroll the editor or command menu in a direction. “Scroll editor down” will scroll the editor down. “Scroll command down” will scroll the active command palette down.
18. “Move cursor _____” – Move the cursor in a given direction. “Move cursor right” will move the cursor to the right.
19. “Place cursor _____ _____” – Place the cursor in a specific location of the screen. The first parameter is the vertical location (top, center, bottom) and the second parameter is the horizontal location (left, center, right). For putting the cursor in the center of the screen only one parameter is used, “center”.

Appendix C

Myna Updated Feature and Usability Study

This instrument will be used to record each observation during user testing and will be completed by the researcher.

Participant ID: _____

Date: _____

Observer: _____

Notes/observations during experimental opportunity:

Observation	Program One	Program Two	Program Three
Participant stated incorrect vocal command ("Drag" vs. "Drag and Drop")			
Participant state incorrect Scratch command ("Go to x" vs. "Go to xy")			
Voice recognition was inaccurate (Participant said the correct commands, but Myna reacted incorrectly)			
The xy-location on the screen was incorrect			
The scrollbar did not scroll properly or elements were not updated properly			
Time to complete program			
Number of times needing to start over			
Other			

For any survey question on which the participant responded in a way that contradicts the observed software usage, how does the participant explain the contradiction?

a. Survey question

b. Participant response

c. Observed behavior

d. Participant's reasoning regarding response and observed behavior

Appendix D

Myna Testing Instructions

To begin this study, you will create three programs using the mouse. You may use this time to become familiar with Scratch and the functions of the many GUI elements and draggable components.

Exercise #1:

1. To begin the program, we want to use the "when clicked" control block. This means that when the green flag is clicked, the program will run.
2. First, let's play a sound.
3. The Cat should move steps using the default number of steps.
4. The Cat should think.
5. Remove the previous action.
6. The Cat should say.
7. The Cat should go to x y using the default xy location.
8. Execute (play) the program.
9. Clear the screen for the next program.

Exercise #2:

1. The Cat should move steps.
2. The Cat should turn degrees clockwise.
3. The Cat should move steps.
4. Change the parameters for move steps so that the Cat takes 5 steps.
5. Add the "when clicked" event block to the beginning of the program.
6. The Cat should go to x y.
7. Edit the parameters of go to x y so that x = 5 and y = 10.
8. Execute (play) the program.
9. Clear the screen for the next program.

Exercise #3:

1. To begin the program, we want to use the "when clicked" control block.
2. Play a sound.
3. The Cat should turn degrees clockwise.
4. The Cat should move steps.
5. Edit the turn degrees clockwise parameter so that the cat turns 45 degrees clockwise.
6. The Cat should think.
7. The Cat should go to the default location.
8. Remove the previous command.
9. Execute (play) the program.

Appendix E

Myna Testing Instructions

Please take fifteen minutes to experiment with using Myna. If you have any questions, please feel free to ask them during this time. Once the experimental time is over (and you feel comfortable with using Myna), you will begin creating the programs described below. During which time, I will be unable to answer your questions. Your performance will be observed to document the time taken to complete each exercise as well as how Myna performs. At the end of the experiment, you will be asked to complete a survey regarding how you feel about Myna.

At any time, you can say, "help" to see a list of vocal commands. You can also start over by saying "clear." You can also pause the voice recognition by saying "pause."

Exercise #1 – 1366 x 768 Resolution:

1. To begin the program, we want to use the "when clicked" control block. This means that when the green flag is clicked, the program will run.
2. First, the Cat should move steps using the default number of steps.
3. The Cat should think.
4. Undo the previous action.
5. The Cat should "meow".
6. The Cat should turn 15 degrees clockwise.
7. The Cat should go to x y using the default x y location.
8. Execute (play) the program.
9. Clear the screen for the next program.

Exercise #2 – 1280 x 720 Resolution:

For this program we are going to make a cat and a bee race each other.

1. The Cat should move steps.
2. Change the parameters for move steps so that the Cat takes 20 steps.
3. Add the "when clicked" event block to the beginning of the program.
4. The Cat should change costume.
5. Get the Bee from a file.
6. Add the "when clicked" event block to the Bee's scripts.
7. The Bee should move steps.
8. Change the parameters for move steps so that the Bee takes 25 steps.
9. Execute (play) the program.
10. Clear the screen for the next program.

Exercise #3 – 1024 x 600 Resolution:

1. Scroll down the control panel.
2. Scroll up the control panel.
3. Drag and drop the “when clicked” control block.
4. Scroll down the motion panel.
5. The Cat should go to x y using the default parameters.
6. The Cat should say “meow”.
7. The Cat should go to the next costume.
8. Make the stage large.
9. Add a dragon sprite from file.
10. Drag and drop the “when clicked” control block for the new sprite.
11. The new sprite should rotate counterclockwise using the default parameters.
12. Execute the program.

Appendix F

Myna Updated Feature and Usability Study

The purpose of this survey is to evaluate the effectiveness of the adjustments made to Myna. Your feedback is greatly appreciated.

Specific Features

The following questions concern the way specific features work within Myna. Please circle the number that best represents your response to each question.

1. I found that Myna appropriately adjusted to the screen resolution:

1	2	3	4	5
Never	Occasionally	Half of the time	Most of the time	Always

2. I found that scrolling the editor and command palette worked:

1	2	3	4	5
Never	Occasionally	Half of the time	Most of the time	Always

3. The commands for scrolling were:

1	2	3	4	5
Predictable	Somewhat predictable	Neither predictable nor unpredictable	Somewhat unpredictable	Unpredictable

4. I found that the cursor movement commands moved the cursor:

1	2	3	4	5
Too much	Sometimes enough, sometimes too much	Just enough	Sometimes enough, sometimes too little	Too little

5. I found that the cursor movement commands were:

1	2	3	4	5
Predictable	Somewhat predictable	Neither predictable nor unpredictable	Somewhat unpredictable	Unpredictable

Did you experience any bugs/errors with the features mentioned above? **Yes | No**
If yes, please enumerate them as well as any steps that may lead to their replication.

For any of the features mentioned above, do you have any suggestions for improving their quality or fixing any errors? **Yes | No**

If yes, please explain.

Demographic Information

All of the below questions concern your demographic information. For each question, please circle the response that best describes you.

1. Age:

20-24 25-29 30-34 35-39 40+

2. Gender:

Male Female

3. Race:

Asian, Asian American, or Pacific Islander	Black or African American	Native American	South, Latin, or Central American or other Hispanic	White or Caucasian	Other: _____

4. Current Year in University:

Freshmen Sophomore Junior Senior Graduate Student