Introduction

The **Eye Can Code** Initiative exists to address the very real challenge that blind and visually impaired young children face when following the current Computer Science (CS) curriculum.

The UK school curriculum requires children from the age of just 6 to learn the rudiments of coding. Scratch and other 'blocky' code languages exist to introduce and teach these kids coding. But the fantastically colorful drop and drag code blocks of Scratch are of no use to a child that cannot see.

So, we are left expecting kids with any kind of Visual Impairment to follow the Computer Science (CS) curriculum and learn to code without any kind of equivalent, engaging and stimulating development tools available to them as we do for sighted children. And by doing so, are putting them at a completely unnecessary disadvantage.

Figures published by the RNIB, estimate the number of children aged 16 and under with vision impairment in the UK and England is 26,000 - 26,000 school kids that are not receiving the same level of support in Computer Science as sighted kids, to meet the expectations set in the National Curriculum.

The primary objective of the **Eye Can Code** initiative is to develop a self-contained programming language and application specifically for the Visually Impaired, called **Speak**, based upon the T.E.D prototype (see article below for more information on T.E.D), which can be viewed here. **Speak** will allow anyone with any kind of visual impairment to learn, enjoy and participate in coding activities completely in line with expectations set by the National Curriculum, leveling the playing field between sighted and Visually Impaired children and most importantly simply allowing them to partake in cool and fun coding activities sighted children are fortunate to be able to take for granted.

Speak gives them that tool.

The following is an article written about the challenges blind and visually impaired children face when learning to code and the story behind T.E.D which was the impetus to developing **Speak** and launching the **Eye Can Code** Initiative:

Kids love clubs. Scouts, Brownies, Swimming, Athletics; they all offer that sense of belonging. A shared interest. A common pursuit. Social Inclusion.

Keen to find our 8-year old son a suitable extra curriculum activity, we very quickly exhausted all usual options; Archie isn't sporty and wasn't realistically going to make the team at our local football club. And he lasted just one Sunday morning taster session at Stage School, brutally crushing his Mother's dreams of one day walking the red carpet at some glitzy West End opening night. Fortunately, she did get a full refund on her frock.

But Archie likes to code. That's right – he likes to code. As educators and IT professionals, we promote the educational benefits of coding, but let's never forget; for some, as alien as it might sound to others, coding is an enjoyable hobby.

And that's how I first became involved in Code Club. With no coding clubs operating locally, I approached Archie's school and offered my services as a coding mentor if they would agree to host a club. Fantastically they enthusiastically embraced the idea and, with the support of The Code Club network, within a few weeks Deputy Head Mr. Fry and myself were running our first meeting – full of enthusiastic, likewise children. A kind of Cub Scout meeting for Geeks. In a very cool way.

But week two threw a curveball; Ted. Ted is blind. We had access to a Big Trak programmable tank, which occupied him for a while. But come week three, as the other children sat in front of their computers creating imaginative, interactive Scratch projects whilst Ted just tinkered with the Big Trak in a space we'd cleared for him on the floor, I couldn't help but think that this wasn't the inclusive club that I had set out to create. We had to do something more.

But what? And how? How does someone who is blind even code? I've been coding for around 25 years and, despite the occasional outrageous brag to the contrary, I really can't do it with my eyes closed. But then I came across an excellent article "A Vision of Coding, Without Opening your Eyes" written by Florian Beijers, a Coder from Arnhem in the Netherlands. In his enlightening piece, Florian, blind since birth, explains how all he needs to code (or access any other apps for that matter) is a Screen Reader, which does exactly what it says on the tin and reads the text on the screen back to him, in a kind of Siri/Alexa fashion.

I contacted Florian and through various exchanges, learnt a lot from him. Mainly that he's smart. He's also pretty cool; to start he has a cool alias – Zersiax, that he's known by in coding circles. I don't have an alias, and I'm not known in any circles; coding, social or otherwise. But that's an aside, and I've bitterly digressed.

Zersiax learnt to code at just 10-years old, almost by accident, when he stumbled across an online HTML tutorial written by an older, also blind, student. Using his Screen Reader, he followed the tutorial and taught himself to code.

See; I told you he was smart.

But scarily, in coding terms, 10-years-old is now practically ancient. The UK curriculum is asking children aged just 6 or 7 to learn the rudiments of coding. This is of course where Scratch and other 'blocky' code languages come into their own. But here's the snag; screen readers only read text. The fantastically colorful drop and drag code blocks of Scratch aren't text, but images, and a screen reader simply can't work in this instance. As Zersiax explains "This makes getting started for a kid in this age group trickier than it should be".

But even with Screen Reader capabilities, there's a further challenge; "You need to find some way of giving these kids that same instant feedback sighted people get with Scratch almost morphing in front of their eyes when they build things" adds Zersiax.

This is an issue. Currently we are expecting kids like Ted to follow the Computer Science (CS) curriculum and learn to code without making equivalent, engaging and stimulating development tools available to them as we do for sighted children. And we are putting them at a completely unnecessary disadvantage.

Let's put things into some perspective - figures published by the RNIB, estimate the number of children aged 16 and under with vision impairment in the UK and England is 26,000. That's 26,000 school kids that are not receiving the same level of support in Computer Science as sighted kids, to meet the expectations set in the National Curriculum. How is that fair?

What's so disappointing about this, is that visual Impairment or blindness clearly isn't an obstacle to becoming a great coder (I cite here the super smart and super cool Zersiax as one such example). The new CS curriculum was introduced in part to tackle an industry skill shortage, and yet it's inadvertently eliminating 26,000 potential new coders. And Coding isn't a bad job for anyone.

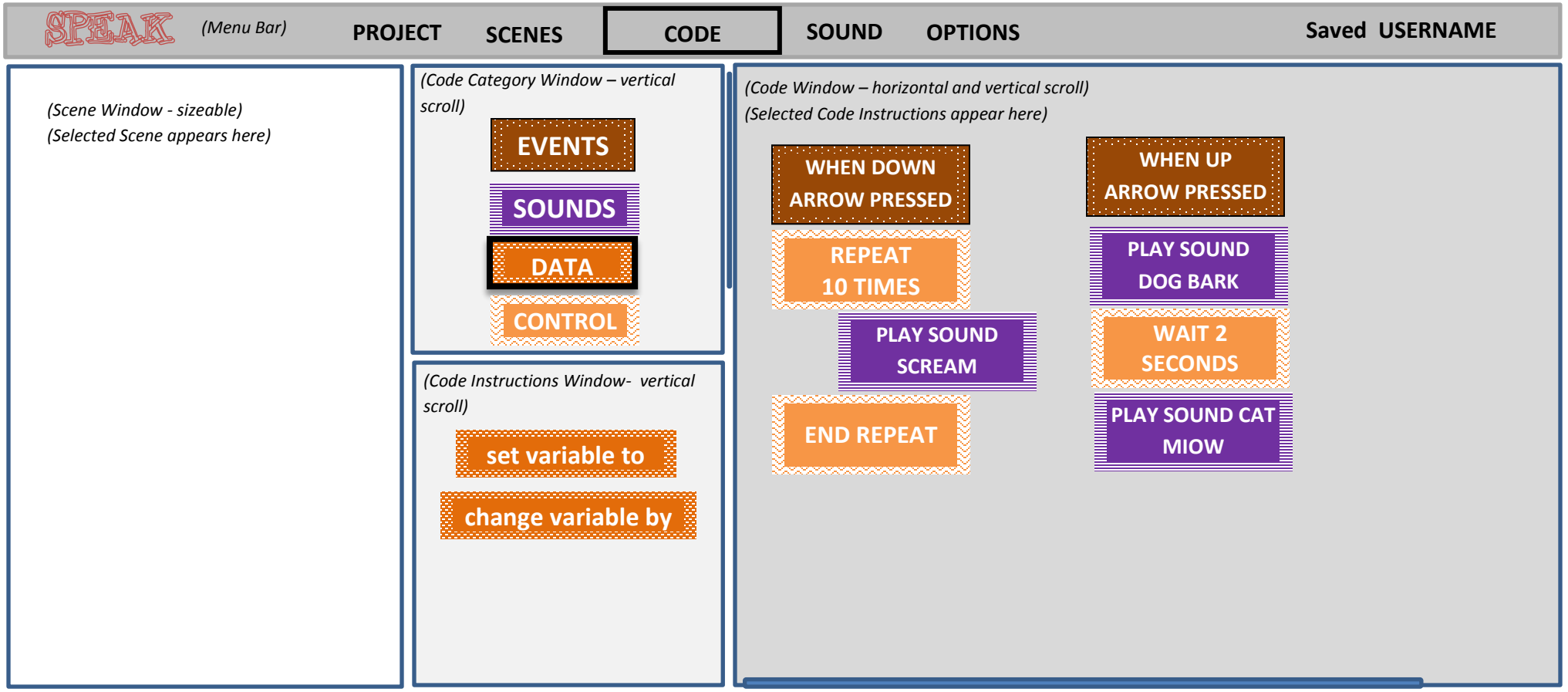
But moreover, and putting education and career aspirations to one side, we are excluding children from participating in a fun, creative and engaging activity that other kids can just take for granted. If we are truly to embrace "CS for All", and we should, then this needs addressing.

So what about Ted? What Scratch does have is an excellent sound library. To engage and include him in the Code Club, I wondered if it would be possible to develop something using Scratch to leverage that sound library along with some of the key Scratch code categories (Events, Control etc.). The goal was to give Ted a real coding experience, the same as everyone else. It of course had to have voice prompts and use the keyboard as little as possible; just the arrow keys to navigate and the space bar to select an option.

The result was a somewhat rudimentary Scratch project I eponymously dubbed 'Talk Enabled Development' (or T.E.D for short). It's basic, uses my irritating, far from dulcet voice for prompts and probably contains more bugs than an ant's nest, but until the big guys get fully onboard with this and consider accessibility with their application builds, it'll have to do. Most importantly, Ted is now enjoying writing code. Mission accomplished.

A major part of the Eye Can Code initiative is to develop a self-contained programming language and application called Speak, loosely based upon the T.E.D prototype.

Illustration: Proposed UI:



General Overview and Navigation

It's imperative that screen navigation is intuitive for the visually impaired user. The user can navigate all options just using the arrow keys and space bar (later version will allow these keys to be configurable). Voice prompts are given on each item so the user knows where they are placed. Depending on configuration, these may be short prompts or a more detailed explanation (see **Options** for more details).

The Tab key will work the same as the Right Arrow Key.

The Enter Key will work the same as the Space Bar.

It will also be possible to point and click on the items using the mouse. Voice prompts will be given when mouse cursor is hovered over the item.

Where options are audibly listed for selection (for example variable values), users with touch typing skills will also be able to directly enter any valid value, via the keyboard. When entering any value from the keyboard, the value is audibly read back to the user, e.g. if entering '103' the user would hear 'one-zero-three'.

All images must have detailed, descriptive alternative text (ALT Text) attributed to it as screen readers used by the Visually Impaired will read these to the user.

Upon logging on, navigation is initially placed on the **PROJECT** item on the menu bar.

For keyboard control, the following rules apply:

- **Main Menu Bar**
  - **Right (and Tab) Key**
    - Moves control to the next item on the right
    - Control wraps to first menu item, when last menu item reached.
  - **Left Key**
    - Moves control to the next item on the left
    - Control wraps to last menu item, when first menu item reached.
  - **Down Key**
    - Moves control to the first item on the submenu. If on menu item **CODE**, control moves to the **Code Categories** window
  - **Up Key**
    - No affect

- **Space Bar**
  - Moves control to the first item on the submenu. If on menu item **CODE**, control moves to the **Code Categories** window

- **Sub-Menu Bar**
  - **Right (and Tab) Key**
    - Moves control back to the next item on the right on the **Main Menu**
    - Control wraps to first menu item, when last menu item reached
  - **Left Key**
    - Moves control to the next item on the left on the **Main Menu**
    - Control wraps to last menu item, when first menu item reached.
  - **Down Key**
    - Moves control to the next item on the submenu
    - Control wraps to first submenu item, when last first submenu item reached.
  - **Up Key**
    - Moves control to the previous item on the submenu
    - Control returns to main menu bar, when first submenu item reached (does not wrap).
  - **Space Bar**
    - Selects that item. If appropriate, opens options list

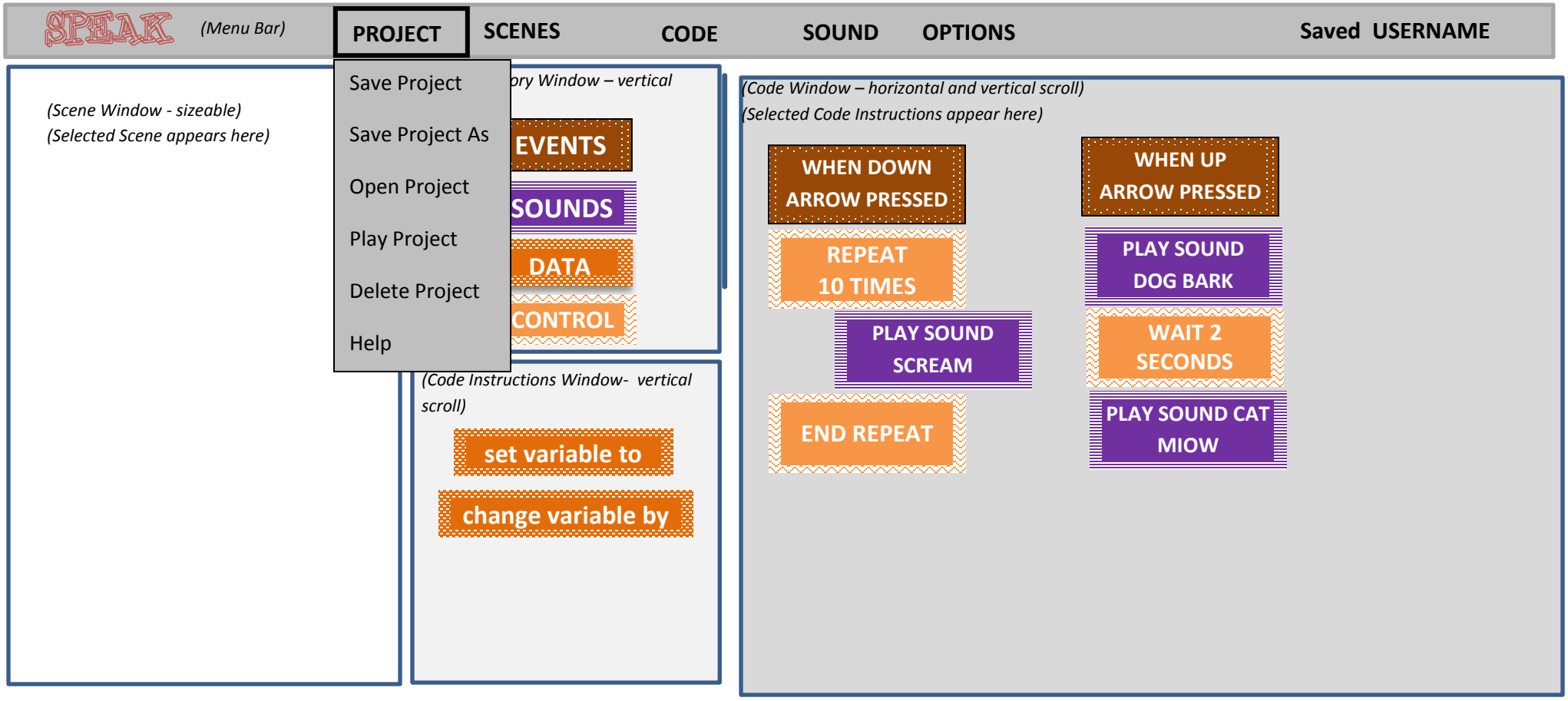
- **Code Category Window**
  - **Right (and Tab) Key**
    - Moves control to the **Code Window**
  - **Left Key**
    - No affect
  - **Down Key**
    - Moves control to the next code category
    - Control wraps to first code category, when last first code category reached.
  - **Up Key**
    - Moves control to the previous code category
    - Control returns to main menu bar, when first code category is reached (does not wrap).
  - **Space Bar**
    - Selects that code category. Opens the Code Block list for that category

- **Code Window**
  - **Right (and Tab) Key**
    - Moves control to the first code block (Event) on the next section of code listed under an Event (if any exists), on the right.
  - **Left Key**
    - Moves control to the first code block (Event) on the previous section of code listed under an Event (if any exists) on the left. If no exists, it control moves back to the **Code Category** Window.
  - **Down Key**
    - Moves control to the next code block
    - Control wraps to first code block, when last code block reached.
  - **Up Key**
    - Moves control to the previous code block
    - Control wraps to last code block, when the first code block reached.
  - **Space Bar**
    - Opens the Code Block edit options for that code block

- **Code Window – Edit Options**
  - **Right (and Tab) Key**
    - No Affect
  - **Left Key**
    - Moves control back to the code block that was previously selected in the **Code** Window.
  - **Down Key**
    - Moves control to the next edit option
    - Control wraps to first edit option, when last edit option reached.
  - **Up Key**
    - Moves control to the previous edit option
    - Control wraps to last edit option, when the first edit option is reached.
  - **Space Bar**
    - Selects the Code Edit option
    - Processes the option (e.g. Delete, Insert, Edit).
    - Control returns to the Code Window once edit is complete (note that Edit will open a further option list – navigation should follow similar rules as for **Code Window – Edit Options**).

Menu Bar

Project Menu



- ## Save Project

  Saves the current project using the current filename.
  Project is also auto-saved periodically.
  Projects save to a default Speak project folder, to avoid user having to browse/select folders.
  If the current project does not have an associated filename, the Save Project As logic is invoked (see below).

- ## Save Project As

  Saves the current project with a specific filename. Because users may find it difficult type in a specific filename, Speak will automate a new, preset filename (e.g. project1, project2 etc.).

- ## Open Project

  Opens previously saved projects. The user can step through a list of all previously saved projects, with Audible prompt of each project filename.

- ## Play Project

  Switches Speak to Play Mode to execute the current project. Play Mode is required so that the keys used in Code Mode to navigate menu options, can now be used execute code as an Event.
  Upon switching to Play Mode, the Scene Window will auto maximize.
  Pressing the Space Bar will exit Play Mode back to Code Mode and reset the Scene Window to the standard size.

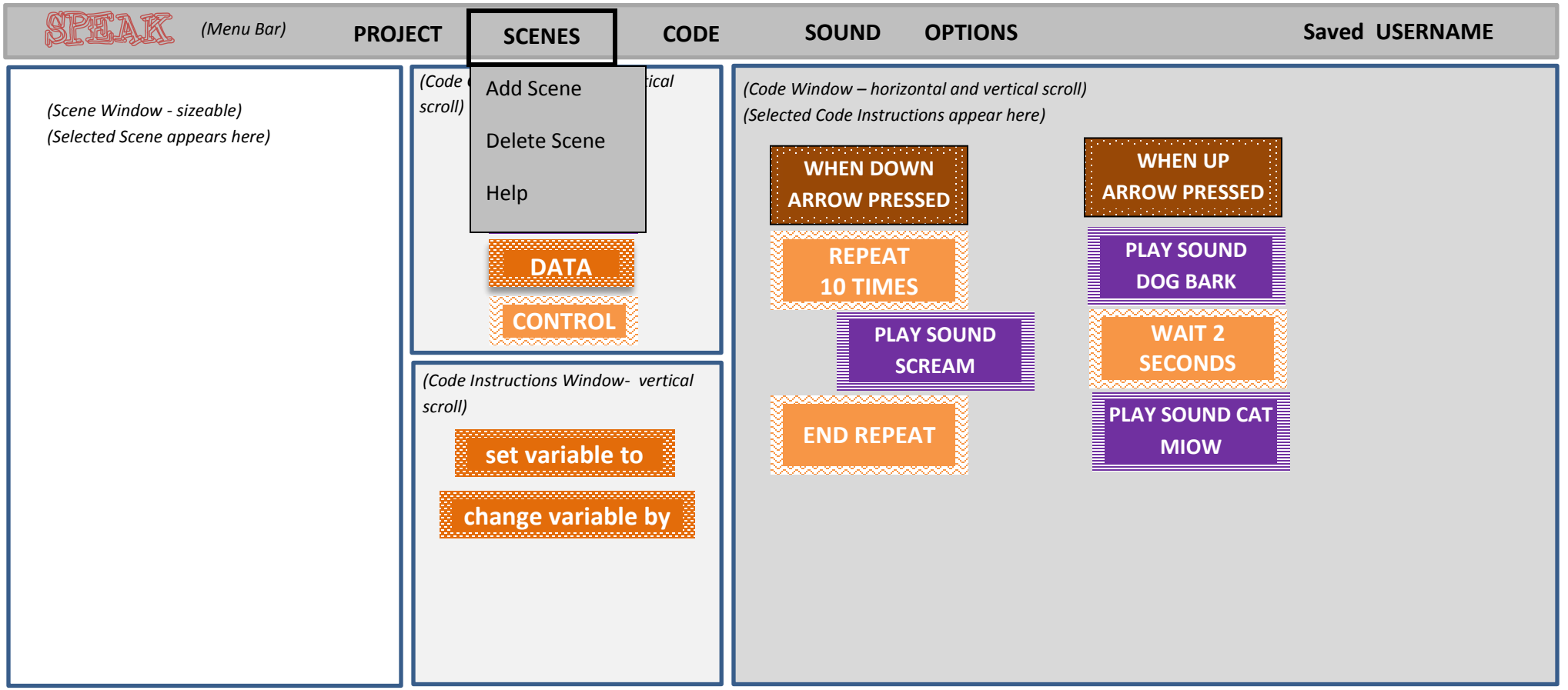
- ## Delete Project

  Audibly lists all saved projects to allow for deletion. Deleting a project makes that preset project name available to use again.
  Before deleting, user is asked to confirm deletion by pressing the space bar again.

- ## Help

  Provides audible help on the Project Category.

Scene Menu



Scenes are just a static background. They serve no real purpose other than to set the scene and provide a relative background to the project. Future versions of Speak will allow for Backgrounds and Objects (Sprites) similar what is available in Scratch.

When selecting or on when hovering the mouse over, a detailed description will be given describing the scene, e.g. "A stage with musicians and a singer. There is a drummer and drum kit at the back, a woman playing a guitar on the right hand-side, and at the front there is a male singer wearing jeans and a T shirt holding a microphone"

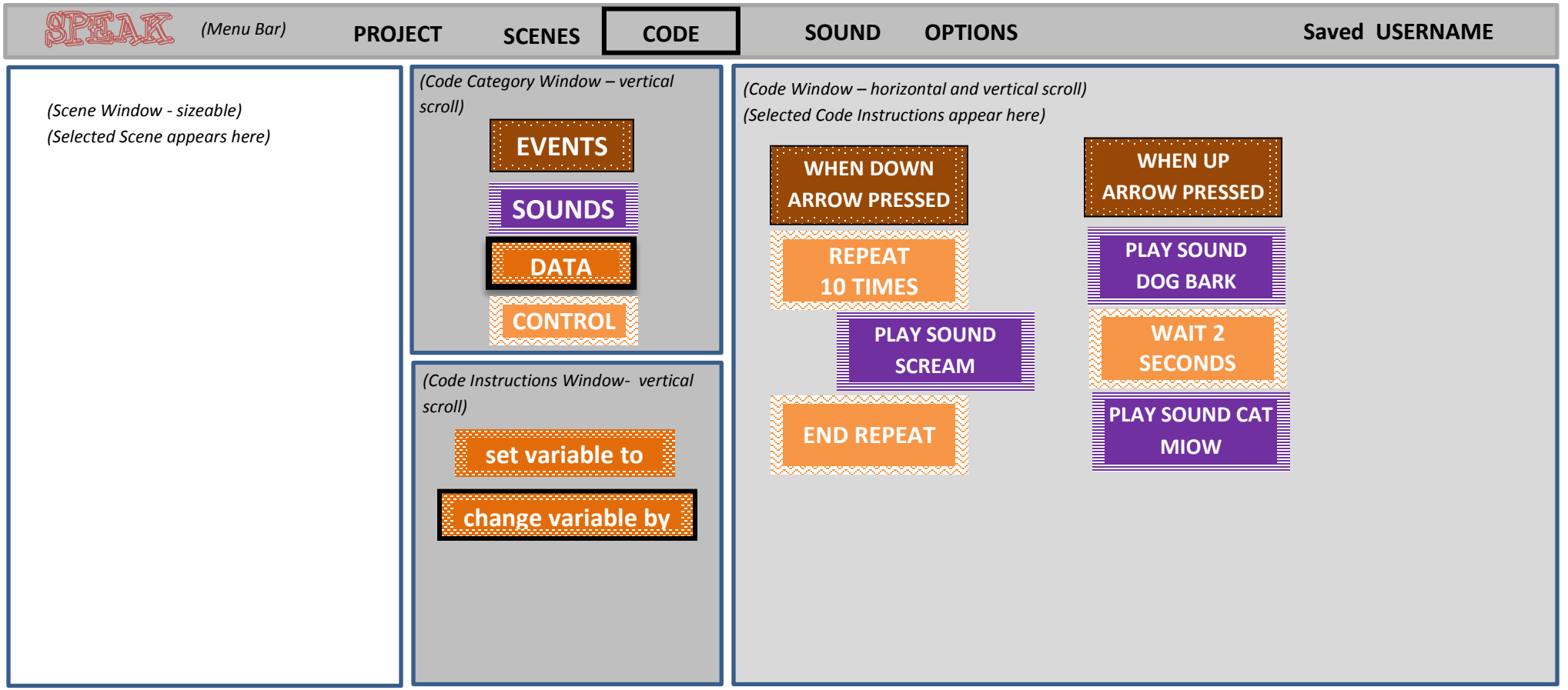
Note that the description could be shortened by configuring the short/long voice dialogue option (see Options for more details).

The selected Scene will appear in the Scene window. When the project is played, this auto maximizes the Window. It auto resets to standard size once the user the user presses the Space Bar to exit Play Mode and return to Code Mode.

- ## Add Scene
  Audibly lists and describes all Scenes which come as standard with Speak, to allow for selection.

- ## Delete Scene
  Deletes the current displayed Scene (note it only removes it from the Scene Window – the scene is still stored on Speak)

Code Menu



Allows the user to add and edit the code instruction blocks to the Code Window, building up their project.

Unlike the other menu items, selecting the **CODE** menu item does not open a sub-menu but instead passes control to the static **Code Category Window.** The user is then able to navigate between the various Code Categories listed in the window. Each category is spoken to the user. Pressing the space bar selects this category and allows the user to then audibly browse the various code instructions related to the chosen category.

The space bar (or double mouse click) is used to select a specific code instruction block. When a code block is selected, the block auto moves to the code window and snaps into place under the last placed code block. Selecting a new Event block starts a new code instruction list.

Pressing the right (or tab) key whilst on any category, switches control to the Code Window. From here the user can move up and down any previously places code blocks, or right (or tab) key again to the next section of code listed under an Event (if any exists).

Pressing the Space Bar on any previously placed code blocks opens the options list to allow editing, deleting or inserting of code blocks:

- **Edit**
  Edit the selected code block (i.e. amend a value). Only available of Code Blocks that have editable values.

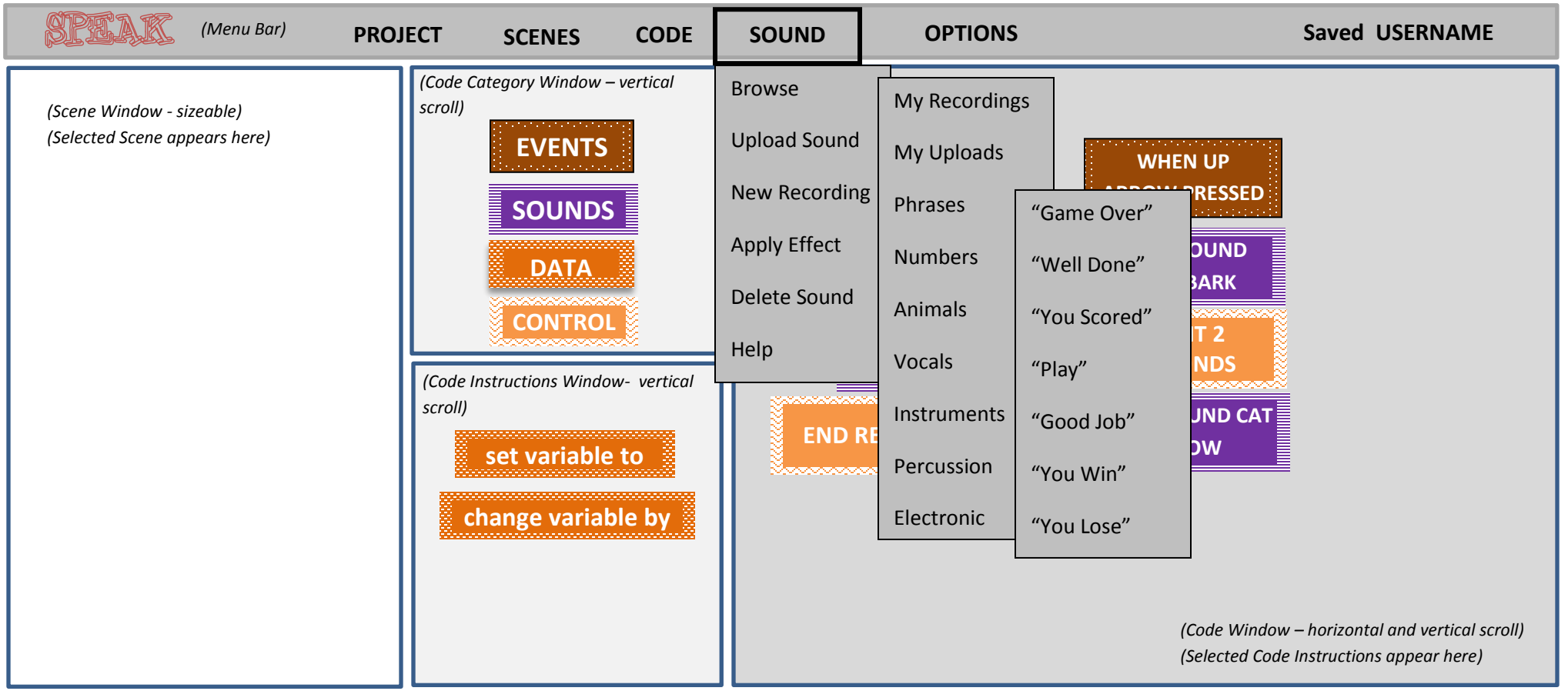
- **Insert New After this Block**
  Inserts a space between the selected Code Block and the next one. The next Code Block added is inserted into this space.

- **Delete**
  Deleted the selected Code Block.

Full details on the code instruction blocks available under each Code Category, are detailed in the **Categories and Code Blocks** section of this document.

Sound Menu

Note the Sound Item on the main menu bar is for managing sounds; uploading, deleting, recording.



- ## Browse

  Audibly browse all available sounds. Sounds are categorized to make it easier to locate a specific sound. The user can navigate through the categories and the related sounds using the arrow keys, and space bar to select.

- ## Upload Sound

  Allows the user to upload a sound stored locally on their computer. This option will utilize the standard Windows Open File process. The user may need additional assistance with this.

  The uploaded sound is stored under the sound category ‘My Uploads’ and can then be used in projects.

- ## New Recording

  Allows the user to record their voice or other sound using their computer microphone. The recorded sound is given a system auto-generated filename (e.g. Recording 001, Recording 002 etc.) and stored under the sound category ‘My Recordings’ and can then be used in projects.

- ## Apply Effect

  Allows the user to apply effects to sounds. This creates a new sound. E.g. If you had a sound file called "Cat Meow", and applied the Echo effect to it, a new sound file called "Cat Meow_Echo" would be created.

  Effect include:

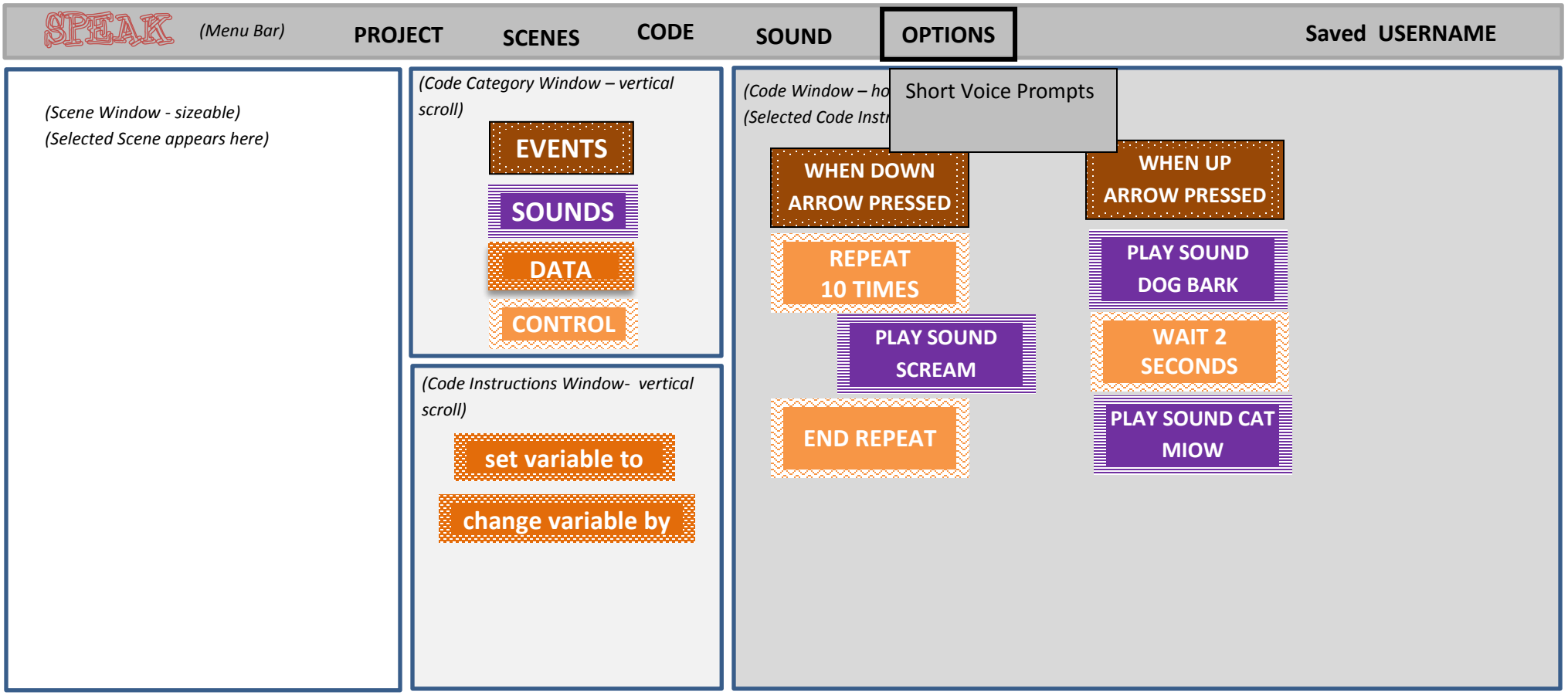
  - Play Backwards
  - Reverb
  - Echo
  - Mouse
  - Robot

- ## Delete Sound

  Allows the user to delete any sounds previously uploaded or recorded. It is not possible to delete any Speech owned sounds. Sounds are audibly listed. Pressing the Space Bar selects it for deletion. The user is asked for confirmation before deletion takes place.

- ### Help
  Provides audible help.

Options



- ## Short/Long Prompts

  Long voice prompts will give the user more detailed audible prompts e.g. "Event Category: always start your code with an event. An event is something that must happens to start your code". A Short Prompt would simply state "Events Category". Default is Long Prompt. Typically, user would set to Short Prompts once they are more experienced.

  Pressing the Space Bar on this option toggles between Short and Long.

Categories and Code Blocks

The space bar (or double mouse click) is used to select a specific code instruction block. When a code block is selected, the block auto moves to the code window and snaps into place under the last placed code block.

Events

All code scripts start with an event. Selecting a new Event block starts a new code set of code instructions.

- **When Key Pressed**
  Audibly lists all keys, including (and starting with) the arrow keys. Pressing Space Bar selects the key, and the **When Key Pressed** Event code block (including the key selected) is added to the Code Window.

  Note - the space bar will not be available as this is reserved for use by Speak.

Sounds

- **Play Sound**
  Plays the selected sound. Control immediately moves to the next instruction.

- **Play Sound Until Done**
  Plays the selected sound. Control does not move to the next instruction until the sound has completed playing.

  Selecting the above code blocks, opens the Sound Library for the user to audibly browse and select. Sounds are organized under the following categories:

  - My Recordings
  - My Uploads
  - Phrases
  - Numbers
  - Animals
  - Vocals
  - Instruments
  - Percussion
  - Electronic
  - Effects
  - Human

- **Set Instrument to**
  As per Scratch

- **Set Tempo To**
  As per Scratch

- **Play Note n for n Beats**
  As per Scratch

- **Play Drum n for n Beats**
  As per Scratch

- **Stop All Sounds**
  Immediately stops any sounds currently playing

- **Change Volume By (%)**
  Changes the volume of a sound, by percentage

Data (Variables)

- **Set Variable to**
  Sets the variable to either a numeric value
- **Change Variable by**
  Change the variable by a numeric value (+ or -)

  Assigned values need to be predefined to allow the user to select them from an audible list. The following values will be available:

  0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25, 30, 50, 60, 100.

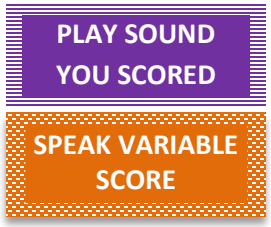
  For users with touch typing skills, it will also be possible to directly enter any value (up to 999), via the keyboard. When entering any value from the keyboard, the value is audibly read back to the user, e.g. if entering '103' the user would hear 'one-zero-three'.

  Variables Names are predefined (declared) and audibly listed when referencing them with any of the code blocks above. The following variables will be available:

  - Points

- Lives
- Score
- Time
- Counter
- Level
- X
- Y

- **Speak Variable Value**
  Audibly plays the variable value. This can be used together with Phrases, for example:



  Will result in “You Score 25” (if Variable Score = 25).

Control (Conditions)

- **Wait n Seconds**
  Wait the selected number of seconds before control continues to the next Code Block (instruction)

- **Repeat n times**
  Repeat the code instructions between this Code Block and the End Repeat Code Block, the selected number of times. Used in conjunction with the **End Repeat** Code Block.

- **Repeat Until**
  Repeat the code instructions between this Code Block and the End Repeat Code Block, until the selected condition is met. Used in conjunction with the **End Repeat** Code Block.

  If this Code Block is selected, the pre-declared Variables will be audibly listed allowing the user to select one:
  - Points
  - Lives
  - Score
  - Time
  - Counter
  - Level
  - X
  - Y

  Once the Variable has been selected, the Operators will be audibly listed allowing the user to select one:
  - Equal to
  - Not Equal to
  - Less Than
  - Greater Than

  Finally, the user must set the Condition to be met from the audible list:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25, 30, 50, 60, 100.

- **Repeat Forever**
  Repeat the code instructions between this Code Block and the End Repeat Code Block continuously, until the user exits Play Mode by pressing the Space Bar. Used in conjunction with the **End Repeat** Code Block.

- **End Repeat**
  Marks the end of the Code Blocks to be repeated using the loop Code Blocks above.

- **If**
  Used to build a conditional statement. If the conditional statement is true, then all Code Blocks listed between this Code Block and the **End If** Code Block are executed.

  If this Code Block is selected, the pre-declared Variables will be audibly listed allowing the user to select one:
  - Points
  - Lives
  - Score
  - Time
  - Counter
  - Level
  - X
  - Y

  Once the Variable has been selected, the Operators will be audibly listed allowing the user to select one:
  - Equal to
  - Not Equal to
  - Less Than
  - Greater Than

Finally, the user must set the Condition to be met from the audible list:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25, 30, 50, 60, 100.

- **Else**
  Used as part of a conditional statement. If the conditional statement is False, then all Code Blocks listed between this Code Block and the **End If** Code Block are executed

- **End If**
  Marks the end of the Code Blocks to be executed using the conditional Code Blocks above.

Notes and Application Build Considerations

- Code Instruction blocks are shown in the design as rectangles; in the finished application, they should be of similar design and shape as Scratch; i.e. 'jigsaw' elements that snap together. When code block is selected (space bar or mouse click), the block should auto move to the code area and snap into place (animation). Selecting a new Event block should start a new code instruction list.

- All images must have detailed, descriptive alternative text (ALT Text) attributed to it as screen readers (if used) will read this to the user.

- Avoid Using Color to Convey Information/errors. If a user is color blind, they won't be able to see this indication, so it becomes difficult to understand which fields have errors. The use of icons (e.g. ticks and crosses) and labels to show which fields are invalid better communicates the information to a color-blind user.

- Apply patterned overlays to solid colors to give users a clearer way to distinguish between elements.