View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Scholarly Materials And Research @ Georgia Tech

An interactive, graphical coding environment for EarSketch online using Blockly and Web Audio API

Anand Mahadevan
Dolby Laboratories
San Francisco, CA 94103
anand.mahadevan@dolby.com

Jason Freeman
School of Music
Georgia Institute of Technology
Atlanta, GA 30332
jason.freeman@gatech.edu

Brian Magerko
Digital Media Program
Georgia Institute of Technology
Atlanta, GA 30332
magerko@gatech.edu

ABSTRACT

This paper presents an interactive graphical programming environment for EarSketch, using Blockly and Web Audio API. This visual programming element sidesteps syntactical challenges common to learning text-based languages, thereby targeting a wider range of users in both informal and academic settings. The implementation allows seamless integration with the existing EarSketch web environment, saving block-based code to the cloud as well as exporting it to Python and JavaScript.

1. INTRODUCTION

1.1 A brief overview of EarSketch

[1] EarSketch [12] is a novel approach to teaching computer science concepts via algorithmic music composition and remixing in the context of a digital audio workstation paradigm. It is aimed at satisfying both artistic and pedagogical goals of introductory courses in computer science and computer music. EarSketch seeks to increase and broaden participation in computing by creating an engaging and culturally relevant learning experience using a STEAM (science, technology, engineering, arts and mathematics) approach [19]. Students write code (in Python or JavaScript) to creatively and algorithmically manipulate audio samples from a loop library. EarSketch offers a tightly integrated creative and pedagogical environment that captivates students by making abstract computing concepts relevant in a context that borrows from the paradigm of digital audio workstations and music production while remaining readily accessible to those without any prior experience with music or music technology [18].

Since development of EarSketch began in 2011, it has been accessed by over 40,000 users in all 50 states in the US and over 100 countries through summer camps and academic courses. It has also been incorporated into a music technology MOOC [11] that has reached over 35,000 students.

[1]Most of the content for sections 1.1 and 1.2 was borrowed from [16]

Licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0). **Attribution**: owner/author(s).

Web Audio Conference WAC-2016, April 4–6, 2016, Atlanta, USA

© 2016 Copyright held by the owner/author(s).

1.2 Related work

Other recent projects have also tried to engage students in computing by connecting coding to artistic and creative contexts. Alice, for example, is a 3D programming environment that allows students to create animated stories through code while learning computer science principles such as object-oriented programming [10]. Blockly [1] and Scratch [22], on the other hand, teach programming through the creation and sharing of games, animations and simulations. From the standpoint of coding to create music, Wavepot offers a multitude of APIs for novice and experienced musicians to explore and compose music online [5]. Other visual programming based educational projects such as PencilCode [8], use a block editor called Droplet [7]. Droplet allows bidirectional switching between text and blocks-based authoring.

1.3 Why visual programming?

The role of visual programming in computer science education has been prominent in recent years. As mentioned by Deepak Kumar in [14], visual languages help introduce basic computational thinking skills required to learn programming in a fun, interactive digital multimedia setting. One of the primary motivations of Scratch was to "add programmability to media-manipulation activities that are popular in youth culture" [17]. There are many tools currently available that make this possible for a variety of applications at varying levels of scalability. Max/MSP [3] and Pure Data [20], for example, are the most commonly used visual programming languages by artists, composers and performers.

Good visual programming tools transcend human language barriers and are easily understood by people all over the globe [23]. K.N. Whitley, in detail, examines empirical data from studies that show how visual representation can increase comprehension compared to text [24]. The current version of EarSketch has been very successful with students who are teenagers. However, anything from complex syntax to verbose error messages on the console can lead to distraction and loss of interest. Cyndi Rader, in [21], talks in depth about the degrees of comprehension with regard to childrens' understanding of visual programming environments. In order to make sure the learning is engaging and productive, one needs to address the method of delivery. A graphical environment has the ability to keep users captivated and thereby enables them to focus on the computational learning without worrying about low level intricacies related to code syntax and semantics. It can also be thought of as a link between e-learning and playing [6].

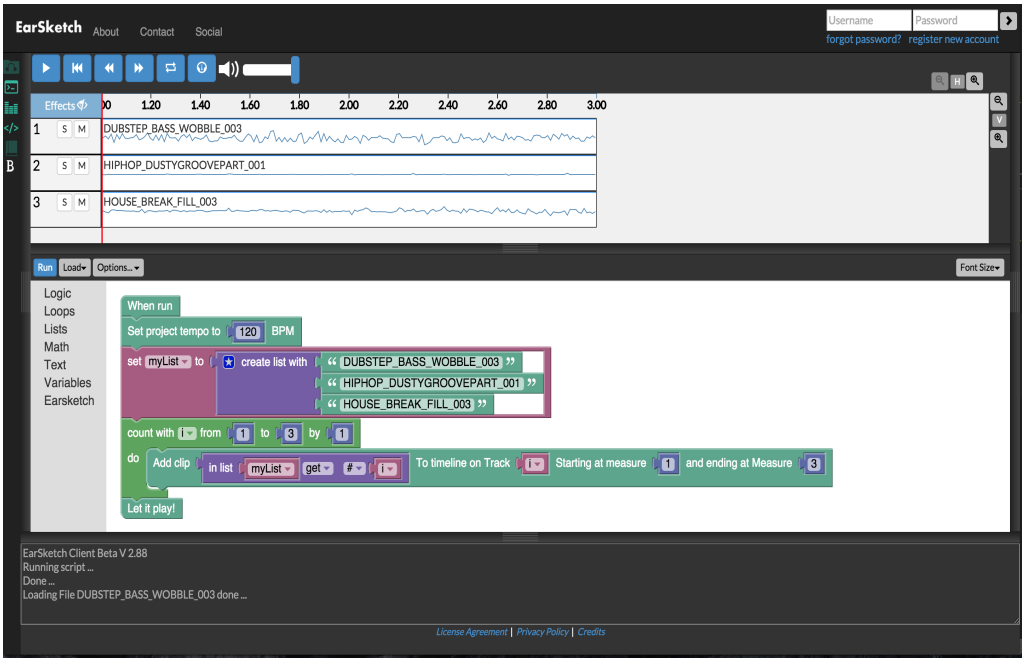

Figure 1: The EarSketch Blockly Interface

1.4 Relevant frameworks

EarSketch resides completely on the web and uses the Web Audio API for all of its audio scheduling and processing. EarSketch previously supported coding in either JavaScript or Python through a text editor integrated into the browser-based interface. To add support for visual programming to EarSketch, we needed a simple and robust framework to author and execute visual code directly in the browser without any external plugins and dependencies. The approach to visual programming also needed to convey key concepts of computing like conditionals and loops which are essential to the EarSketch curriculum as it is taught in schools.

Scratch [4] and Blockly [1] were shortlisted to be the plausible options. Both Scratch and Blockly are open source, customizable, client based, and completely cross browser.

Customization of blocks is imperative for EarSketch as it provides functions that closely mimic popular operations in a digital audio workstation (DAW) workflow such as placing audio clips on a multi-track timeline, adding effects and step sequencing rhythms. Ultimately, Blockly was chosen, as it allowed exporting from blocks to JavaScript, Python and Dart, had a stable code base that was conducive to expansion and customization and could readily be injected into the EarSketch UI.

2. DESIGN AND WORK FLOW

The following section goes in depth into the design and implementation of EarSketch APIs with Blockly. All the EarSketch APIs were redesigned and implemented using the Block Factory utility provided by Blockly. A UI button with the icon "B" located on the top left corner, allows the user to switch between the Blockly editor and the code editor.

2.1 EarSketch API abstraction

Figure 1 shows a sample script executed on EarSketch with Blockly. Before proceeding further, it is worth reviewing some of the core EarSketch APIs. The focus of this discussion will be restricted to the following API functions - `setTempo()`, `fitMedia()`, `setEffect()` and `makeBeat()`. The project tempo is set using the `setTempo()` function.

An audio file is placed on the multi-track timeline using the `fitMedia()` function. Audio files are specified by constants. EarSketch comes bundled with over 4000 audio loops in a variety of genres like hip-hop, soul, rock, techno and house that we commissioned from Richard Devine, an experimental electronic musician and sound designer, and Young Guru, an audio engineer and DJ best known for his long-running collaboration with Jay Z. The remaining arguments to fitMedia() specify a track number, start measure, and end measure. Figure 2 shows the `fitMedia()` API in context, juxtaposed with the python equivalent . All the blocks have tool tips built in to guide the user filling the arguments of the block. The blocks also take care of type checking. For example, a text field cannot be added to a field that expects a number.

The `setEffect()` function adds an effect to a particular track. For example, the following statement

```
setEffect(1, VOLUME, GAIN, -60, 1, 0, 5)
```

adds a volume effect on track 1 whose gain is being automated between -60db and 0db between measures 1 and 5. Figure 3, shows a subset of the supported effects in a drop down fashion.

EarSketch's `makebeat()` function allows users to create rhythmic beats and phrases by using strings to piece together contents of different audio files at a 16th note resolution. Borrowing from Thor Magnusson's ixi lang [15] and Freeman and Van Troyer's LOLC [13], our API uses a string representation to sequence individual sixteenth notes over a full measure. The notation is fairly straightforward; a num-


```
from earsketch import *
myList = None
i = None

init()
setTempo(120)
myList = [DUBSTEP_BASS_WOBBLE_003, HIPHOP_DUSTYGROOVEPART_001, HOUSE_BREAK_FILL_003]
for i in range(1, 4):
  fitMedia(myList[int(i - 1)],i,1,3)
finish()
```

Figure 2: Blockly abstraction for fitMedia and equivalent Python code

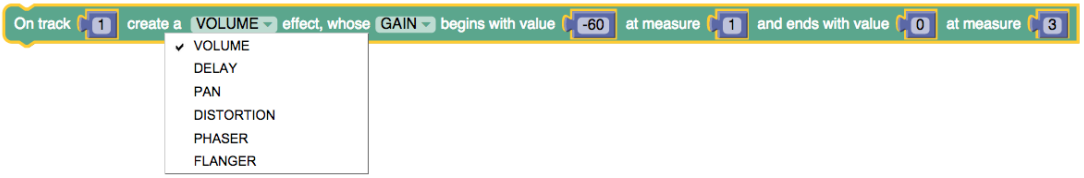

Figure 3: Dropdown list of effects supported by setEffect

ber represents a single audio file or an index in a list of audio files, a "+" sign extends the duration of the preceding sound by a sixteenth note, and a "-" sign indicates a rest.

Bolstered by the default libraries provided by Blockly (such as logic, loops, text and variables), the implemented framework is now capable of effectively leveraging the EarSketch approach to teaching computer science in a visual programming context.

2.2 Saving and exporting

Blockly provides low level routines to switch between DOM and XML. The XML for these groups of blocks in the main workspace is the same as Blockly's XML save format. The equivalent XML is stored just like any other JavaScript or Python script on the EarSketch server. When the user attempts to load his or her Blockly XML, the Blockly editor is automatically brought up and populated with the corresponding DOM. With a single click, the user can also export the Blockly code to Python or JavaScript code in a new tab in the code editor. This can be particularly useful to illustrate how abstractions translate to high level code. It also enables enthusiastic individuals to adopt different methodologies for composition, for example to design a skeletal framework via blocks and perform some advanced editing through code. It should be noted however, that unlike PencilCode [8] and Code.org's App Lab [2], bidirectional conversion between blocks and text views of code is not yet possible: users can only convert from blocks to text code but not the other way.

3. CONCLUSION AND FUTURE WORK

In order to make this project appealing to a broader audience, further refinement and evaluation is required. The criteria for evaluation must be chosen carefully. From the point of view of cognitive factors involved in visual programming, Blackwell et al.[9] lists out various intelligible 'dimensions' such as Abstraction gradient, Closeness of mapping and hidden dependencies. Their research highlights the importance of the underlying paradigms of visual programming languages (control flow versus data flow) and that it must be considered before performing empirical evaluations.

There are several constraints with this environment that need to be addressed. It is difficult to write complex scripts using Blockly due to limited available real estate on the canvas. Although Blockly supports collapsing of blocks, it is not a scalable solution for larger projects. Unlike the code editor which supports multiple tabs, Blockly follows a singleton paradigm. Because of this, the user is limited to having one script open at a time. It is also challenging to address certain basic computing concepts such as variable scope within Blockly.

Despite the aforementioned constraints, there is opportunity for improvement with regard to further abstraction and improved user experience. The `setEffect()` API, for instance, can be broken down into separate entities for creating the effect and applying automation envelopes on them. Intelligently capturing logical errors like infinite loops and unreachable code sections leaves less room for confusion while debugging. Also, the notion of bidirectional conversion be-

tween code and blocks opens a new avenue for EarSketch to explore. Animations between conversions and re-rendering unformatted text into clean blocks are some of the things users can benefit from.

The project also stands as an example of the flexibility of authoring web applications with Web Audio API. Before the advent of the Web Audio API, audio-intensive applications such as EarSketch could only be developed as standalone desktop application software. The Web Audio API enables EarSketch to be deployed in the browser, where it can easily integrate a variety of third-party APIs, frameworks, and web services to add new functionality to the platform much more quickly than would be possible with desktop software. The rapid integration of Blockly into EarSketch to enable a visual programming paradigm is one such example. Because web development tools and APIs make it relatively trivial to integrate new coding paradigms into the EarSketch environment, our team has been able to concentrate its resources on exploring which programming paradigms map best onto digital audio workstation workflows and onto computer science learning objectives, and on API and curricular design that supports these approaches.

4. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant Nos. CNS #1138469 and DRL #1417835. Many thanks to the entire EarSketch project team (http://earsketch.gatech.edu/personnel).

5. REFERENCES

[1] Blockly - https://blockly-games.appspot.com/ accessed: 04/21/2015.

[2] Code.org's app lab - https://code.org/educate/applab accessed: 02/17/2016.

[3] Max / msp - https://cycling74.com/ accessed: 04/21/2015.

[4] Scratch - https://scratch.mit.edu/ accessed: 04/21/2015.

[5] Wavepot - http://wavepot.com/ accessed: 04/21/2015.

[6] A. Baratè, M. G. Bergomi, and L. A. Ludovico. Development of Serious Games for Music Education. *Journal of e-Learning and Knowledge Society*, 9(2), May 2013.

[7] D. Bau. Droplet, a Blocks-based Editor for Text Code. *J. Comput. Sci. Coll.*, 30(6):138–144, June 2015.

[8] D. Bau and D. A. Bau. A Preview of Pencil Code: A Tool for Developing Mastery of Programming. In *Proceedings of the 2nd Workshop on Programming for Mobile Touch*, PROMOTO '14, pages 21–24, New York, NY, USA, 2014. ACM.

[9] A. F. Blackwell, K. N. Whitley, J. Good, and M. Petre. Cognitive factors in programming with diagrams. *Artificial Intelligence Review*, 15(1-2):95–114, 2001.

[10] S. Cooper, W. Dann, and R. Pausch. Teaching objects-first in introductory computer science. In *ACM SIGCSE Bulletin*, volume 35, pages 191–195. ACM, 2003.

[11] J. Freeman. Survey of music technology https://www.coursera.org/course/musictech, October 2014.

[12] J. Freeman, B. Magerko, T. McKlin, M. Reilly, J. Permar, C. Summers, and E. Fruchter. Engaging underrepresented groups in high school introductory computing through computational remixing with earsketch. In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 85–90. ACM, 2014.

[13] J. Freeman and A. Van Troyer. Collaborative textual improvisation in a laptop ensemble. *Computer Music Journal*, (2):8, 2011.

[14] D. Kumar. Digital Playgrounds for Early Computing Education. *ACM Inroads*, 5(1):20–21, Mar. 2014.

[15] T. Magnusson. The IXI lang: A SuperCollider parasite for live coding. International Computer Music Conference, pages 503–506. San Francisco, Huddersfield, International Computer Music Association, Centre for Research in New Music University of Huddersfield, 2011.

[16] A. Mahadevan, J. Freeman, B. Magerko, and J. C. Martinez. Earsketch: Teaching computational music remixing in an online web audio based learning environment. In *Proceedings of the 2015 Web Audio Conference, Paris, France*, 2015.

[17] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, 10(4):16, 2010.

[18] S. McCoid, J. Freeman, B. Magerko, C. Michaud, T. Jenkins, T. Mcklin, and H. Kan. EarSketch: An integrated approach to teaching introductory computer music. *Organised Sound*, 18(2):146–160, Aug. 2013.

[19] N. Park and Y. Ko. Computer Education's Teaching-Learning Methods Using Educational Programming Language Based on STEAM Education. In J. J. Park, A. Zomaya, S.-S. Yeo, and S. Sahni, editors, *Network and Parallel Computing*, number 7513 in Lecture Notes in Computer Science, pages 320–327. Springer Berlin Heidelberg, Jan. 2012.

[20] M. Puckette et al. Pure data: another integrated computer music environment. *Proceedings of the Second Intercollege Computer Music Concerts*, pages 37–41, 1996.

[21] C. Rader, C. Brand, and C. Lewis. Degrees of comprehension: children's understanding of a visual programming environment. In *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems*, pages 351–358. ACM, 1997.

[22] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai. Scratch: Programming for All. *Commun. ACM*, 52(11):60–67, Nov. 2009.

[23] N. C. Shu. *Visual programming*. Van Nostrand Reinhold New York, 1988.

[24] K. N. Whitley. Visual programming languages and the empirical evidence for and against. *Journal of Visual Languages & Computing*, 8(1):109–142, 1997.