Young Musicians and Computer Music

Tommy Hoffmann, Willie Payne, and Matt Pennington
University of Colorado Boulder
College of Engineering and Applied Science
001-303-492-7514
Thomas.W.Hoffmann@colorado.edu, William.Payne@colorado.edu,
Matthew.Pennington@colorado.edu

Abstract

The Python Music Environment is a novel approach to teaching computer science concepts via musical composition and transformation of musical ideas. The goal of the Python Music Environment is to extend education in computing to young musicians through an experience that is personally relevant drawing out and expanding upon existing musical knowledge. Our preliminary work with El Sistema Colorado shows promise at using the tool to engage underrepresented minorities.

1. Introduction

We present the Python Music Environment, a Python front end to digital signal processing software Max MSP that utilizes methods with domain-specific musical naming conventions. This tool is designed to introduce programming alongside a self-guided tutorial or a workshop taught in a classroom setting. Our work is informed by the STEAM (STEM + Arts) initiative which is notable for encouraging creativity to improve educational outcomes. With this tool, we aim to address the lack of representation by racial and ethnic minorities in computer science.

1.1. STEAM education

STEAM education builds upon traditional STEM education. Motivated by a desire to design and produce original art, students become more engaged in learning engineering practices and scientific principles to help them accomplish their goals. STEAM education is improving educational outcomes for the general public and for specific demographics. E-textiles, for example, have been effective at improving participation and success by women in understanding circuitry and computer science.[11] Prior work synthesizing music and computer science has been shown to improve skills, knowledge, and general perceptions of both fields.[1] Finally, Jason Freeman's work shows a correlation between an education combining music and computer science and an increase in engagement of underrepresented minorities.[6]

1.2 Underrepresented minorities in computer science

The Bureau of Labor and Statistics reports that people of Hispanic/Latino/a ethnicity account for 6.6% of employees in Mathematical and Computer occupations compared to 16.6% of all occupations. African Americans account for 8.3% of Mathematical and Computer occupations compared to 11.4% of all occupations.[7] As described in *Stuck in the Shallow End*, this discrepancy results from a lack of opportunities in education. In California, African Americans account for 8% of the student population but only 1% of AP CS course participants. Hispanics and Latino/as represent 41% of the student population but only 8% of AP CS course participants.[10] The authors of *Stuck in the Shallow End* identify numerous causes such as a lack of course offerings at ethnically diverse schools, a lack of sufficiently trained CS teachers, and a lack of institutional support. Even when ethnically diverse students are presented with the opportunities of CS relevant course-work and extracurricular activities, social pressures often preclude them from participating or succeeding.

In section 4, we describe a pilot workshop we conducted with El Sistema Colorado, an organization that empowers students through musical training. Because of the diverse backgrounds of many students in this organization, we see a powerful, supportive environment that contrasts the isolating classrooms described in *Stuck in the Shallow End*. The knowledge of music that the students already possess is a great asset and should be utilized as motivation to learn computer science. We hope our tool may combat the stereotype threat and systemic isolation that ethnically diverse students all too frequently encounter in computer science learning.

Beginning in section 2, we evaluate existing tools that cater to cross-domain work in music and computer science and have pedagogical basis or use. Unlike these tools, the Python Music Environment assumes some formal music training. In section 3, we discuss the design and implementation of the Python Music Environment. In section 5, we discuss the results from our workshop, and finally we conclude and consider future directions for our work in section 6.

2. Prior Work

We surveyed three entry level programming tools that combine music and computer science: Sonic Pi, EarSketch, and BlockyTalky. Other popular tools, Supercollider and CSound for example, require a much higher cognitive load to get started and were not considered. Sonic Pi and EarSketch have embedded documentation and tutorials. All three are designed to be used in a curriculum with an educator scaffolding discovery and learning. Sonic Pi has a plethora of documentation detailing how to incorporate it into educational standards in the United Kingdom's school system,[13] and EarSketch has been used in many pedagogical studies in the United States.[9]

2.1. Sonic Pi

Sonic Pi is a live coding programming environment based upon the ruby language. Sonic Pi is open source and scalable and has a large online community of users. A built-in tutorial guides new users

and more experienced users can interface with Minecraft Pi. While initially the tool was developed for the Raspberry Pi, it now runs on Mac OSX and Windows 7+. It is not supported on iOS, Android or Chromebooks, a potential problem since access to desktop computers is low among ethnically diverse and impoverished communities.[8] The tool also lacks methods for programming music with traditional musical representations, as is discussed further in section 3.1.1.

2.2. EarSketch

Earsketch is a programming environment that enables users to remix existing musical samples with one of two popular languages, Python or Javascript. EarSketch is available online and is written in HTML5 making it accessible on nearly all devices including iPad, Chromebooks, and Android tablets (with attached keyboards). The tool has been used in a handful of pedagogical assessments and has been shown to improve education outcomes across demographic groups including women and underrepresented minorities.[6] We found limiting a lack of methods for formal description of music and EarSketch is currently closed source preventing us from adapting it for our own needs.

2.3. BlockyTalky

We briefly reviewed the BlockyTalky environment. The strengths of BlockyTalky are its support for synchronized performance across separate devices and its traditional representation of musical values.[2] Ultimately, we decided against using the tool for three reasons: First, we hope to investigate our tool's accessibility with blind and low-vision students, and a blocks-based GUI may be a barrier to this population. Second, we hypothesized modifying BlockyTalky in a significant way would pose too large a challenge under the time constraint. Finally, we felt the dependency on circuit boards and synthesizers or robotics equipment may present a financial barrier to continued use by impoverished students.

3. Design and Implementation

In this section, we first list the requirements for the design of the language, then we discuss how the current implementation addresses each requirement, and finally we look at the technical implementation of the language.

3.1. Requirements

Before we began development, we identified three primary requirements:

1. The tool must utilize the existing knowledge of the target population, middle school and high school age musicians.
2. The tool must provide a low barrier of entry and enable users to make sounds instantly after opening the environment. Then, the tool must provide a clear direction for growing program complexity.
3. The tool must promote user creativity and discovery, and adept students should demonstrate skills universal to the field of computer science.

3.1.1. Draws upon existing knowledge

We carefully considered classical music notation when establishing the syntax of our language beginning with the base unit in classical music: the note, an object whose position determines its pitch and shape determines its duration. In the Python Music Environment, notes are represented by Python tuples with the first element describing the note's pitch in scientific pitch notation, and the second element describing the note's rhythm in traditional terms. Our implementation is more intuitive to musicians than that of Sonic Pi which introduces pitch in MIDI values, and separates rhythm from pitch.

Sonic Pi	**Python Music Environment**

`play 60`
`sleep 1`
`play 62`
`sleep 0.5`
`play 64`
`sleep 0.5`
	`setTempo(60)`
	`playNote((C5, quarter))`
	`playNote((D5, eighth))`
	`playNote((E5, eighth))`

Figure 1: Identical Programs written in Sonic Pi and Python Music Environment

Other terms derived from traditional music notation include common symbols for describing volume or amplitude (*pp, mf, ff,* etc.) and predefined instrument names (*trumpet, glockenspiel, violin,* etc.). The names of functions in the Python Music Environment refer to techniques composers use when manipulating or transforming musical motifs. For example, the *transpose* function takes in a list of musical notes (referred to as a *noteLine*) and an positive or negative integer determining the number of steps to transpose the entire line by. The *augment* function maintains the same melodic contour of the included *noteLine,* but doubles all of the rhythmic durations making it play back at half speed. Finally, the Python Music Environment displays a graphical user interface (GUI) consisting of a piano keyboard, a musical staff, and number boxes which hold other information about the currently held note's velocity, duration, pitch, and instrument. The goal of the interface is to provide visual feedback and help the user connect his/her source code with its musical output. Additionally, users may play the piano manually by clicking on the keys.

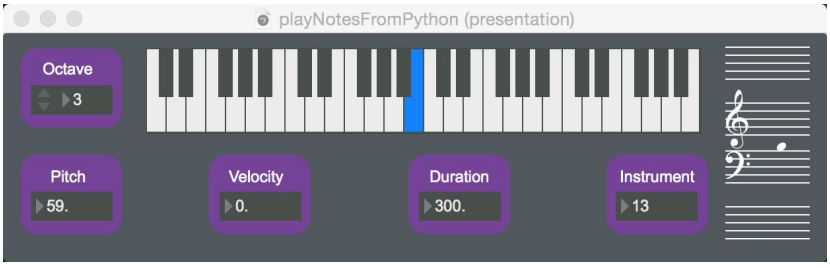

Figure 2: Python Music Environment GUI

3.1.2. Low barrier for entry and scalability

Popular music creation languages, including Max MSP[5] and SuperCollider,[14] often require substantial knowledge of the language's syntax and sound synthesis just to produce the first sound. In the Python Music Environment, the simplest working program requires only two lines of code. In the following example, the user has included our library granting him/her access to all of functions and variables we have defined and calls the *playNote* function with argument *C4* which will play middle C on a piano. (If a duration value is not included, the note will be held the length of a quarter note.)

```
from playMaxNotes include *
playNote(C4)
```

Figure 3: "Hello World" Program in Python Music Environment

All three functions in the Python Music Environment that produce sound only require one argument: a note or list of notes. Other musical structures, velocity or instrument, are abstracted away from the musical notes themselves as in a traditional score, and can be included as optional arguments. Our design of the tool's scalability is reflected in the path that users take in learning to create more involved musical ideas. First, they learn to play a single note via the *playNote* function, then they learn to play a melodic line or list of musical notes via the *playLine* function, and finally, they learn to play multiple musical lines in parallel via the *playCounterpoint* function.

3.1.3. Promotes creativity and understanding of computer science concepts

A novel concept included in the Python Music Environment is the transformation functions, a collection of data processing functions that alter musical lines and are inspired by techniques composers have used throughout history. Beyond the *transpose* and *augment* functions discussed above, transformation functions include *noteLineToMinor* which converts an included melodic line to a minor key by lowering the third and sixth scale degrees, *swing* which alters the rhythms of the melodic line to be swung, and *shufflePitches* which retains the rhythmic contour of the line, but randomizes the order of the pitches. Transformation functions are applied directly to *noteLines* and draw from the Map procedure, a method that applies a function to all elements of a list. While more work and user studies have yet to be undertaken, we hope that transformation functions support users in musical creation, and teach about techniques commonly used in data science and functional programming. The following example shows transformation functions in use altering a *noteLine* called "purcell."

```
playCounterpoint((transpose(purcell, 5), mf, violin),
     (shuffleRhythms(noteLineToMinor(purcell,C0)), f, violin))
```

Figure 4: Transformation Functions

3.2. Technical implementation

The Python Music Environment was developed as a library for Python 3, and the sound synthesis and graphical user interface was built in Max MSP, a visual programming environment designed for musicians, composers, and sound designers. The user composes in a Python document or REPL and when the program is run, messages are sent to Max MSP via Open Sound Control, a flexible and easy-to-use communication protocol. Notes are played from Max MSP via the default MIDI synth which contains over a hundred pre-built instruments and sounds. All source code is open source under the MIT License and available to download online.[12]

4. Pilot Study

We conducted a workshop with El Sistema Colorado during an after-school program at the Bruce-Randolph School. At El Sistema Colorado, 98% of students are on free or reduced lunch, 86% of students are Hispanic, 9% of students are Black/African American, and 61% are actively learning English as a second-language.[3] There are currently no opportunities for computer science or programming coursework at the Bruce Randolph School. We worked with two students who directly expressed interest to band teachers in learning about computer music. David is an 8th grade clarinetist who arrived brandishing a four-sided rubik's cube which he could solve with little effort. He mentioned brief experience with programming referring to workshops in the game development engine Unity given at his library. Michael is a 7th grade bass player who enjoys playing video games. Michael claimed to not have any experience programming. (To protect student confidentiality, we are using pseudonyms.)

We planned in advance the following topics to cover in the workshop:

1. Open ended discussion on the musical elements in the fugue of Benjamin Britten's Young Person's Guide to the Orchestra.
2. Introduction to writing code in Python and the *playNote* function.
3. More complicated melodies and playing the Britten melody via the *playLine* function.
4. Altering Britten's melody via transformation functions.
5. Composing melodies.

Due to a lack of time, we had to cut short the fourth topic and did not reach the fifth topic.

4.1. Challenges Beyond the Curriculum

It is worth noting that during the execution of our planned workshop, we were met with a number of unexpected challenges beyond our control that prevented us from carrying out the complete plan. We initially had three students enrolled; however, one did not show up. From talking to El Sistema teaching staff prior to the workshop, we expected the students would receive instruction over topics including fugues and scientific pitch notation. We also had been informed they would compose their own melodies in advance. However, according to the two students at the workshop,

this did not occur. There were also issues with the timing of the workshop. Due to a scheduling conflict resulting from when the students were released from class and extra time waiting for the third student to show up, we started half an hour later than planned. We we also forced to end early when one participant's mother arrived needing to pick up her son. In total, this left just over 1 hour for instruction, whereas we originally planned for a two-hour period.

4.2. Summary of Workshop

The workshop began with a brief informal interview with the students about their interests and background with computing. Once it was decided to commence, the Benjamin Britten piece was played while students sat and listened. Afterwards we lead a discussion in which we wrote down suggestions provided by the students answering what changes took place throughout the piece. David engaged and responded to the majority of our questions, while Michael provided one word answers when prompted and eventually put his head down on the desk. Students provided suggestions including the instruments coming in and out, loudness, and identified certain sections as being happy or sad. We led the discussion in the direction of changes between major and minor, though the students themselves never identified these specifically.

We transitioned to the next phase of the workshop, where students were provided with laptops with a text editor, terminal, and our graphical interface. From there, we lead a basic demonstration of how to write the function call to play a single note and briefly explained how to use the command line to run a python script. After successfully playing individual notes, the students were allowed to explore the different options of the *playNote* function including changing the instruments and dynamics of the notes. When students began copying and pasting the function call repeatedly, we introduced the *playLine* function and explained how to call it on the pre-defined *britten* melody.

When students began to feel comfortable writing variations of of the *playLine* call for *britten*, the setTempo was highlighted in more detail. Students began experimenting with what could be input into the function. Michael attempted playing 1000, 0.2, -1000, 0.0001, and 1, noting which of these inputs the function did not allow. Once students exhausted this, we began discussing how to transpose the piece, prompting students to find out how many steps would be required to transpose the melody to a different range. In the middle of this discussion, David was required to leave. Michael stayed a bit longer and attempted to successfully write the function calls to play a transposed version of the Britten piece. We wrapped up and asked a few closing questions about Michael's impressions of the experience.

5. Results

This initial study was conducted to test the viability of the Python Music Environment as an effective tool for engagement and learning in the combined discipline of Computer Music. In this

section, we highlight our impressions of what was most successful and what were the largest problems encountered.

5.1. Workshop Successes

Our first impression was the overwhelming excitement with which the students engaged in writing code in the environment. Each new function we introduced resulted in exploration from the students as they tested the outer-bounds of possibility provided by the functionality of each. When working on the laptops, the students remained focused on working with the tool, and had no instances of becoming distracted or losing engagement. At the end of the session, it was clear that the students did not want to leave. David kept requesting to his parent to be allowed to stay longer. Both asked when we would return for another lesson.

Additionally, the students were able to begin producing sound within the first 5 to 10 minutes of opening their laptops with some guidance from instructors. Students could then immediately begin experimenting with the basic examples, in which they encountered aspects of composing computer music. Both played around changing the function inputs to produce different results.

5.2 Workshop Problems Encountered

Some issues were encountered in our lesson plan as we introduced the environment. Where we assumed the students would be able to identify changes between major and minor and hint at other types of transformations, the students did not talk about these in our discussion. This potentially made it more difficult to motivate function transformations later on in the lesson. The students also tended to keep playing with the last function they had been given and did not show expressed interest in moving on to learning new functions.

Another major problem we encountered was a clear lack of understanding of functions and variables. Because we gave examples and had students modify them, we completely sidestepped teaching what a variable was in a technical sense, along with other details of Python syntax. Two observations could be made. The first was that the documentation provided did not provide clear ways to understand what was a function and what was a pre-defined variable. Additionally, in the few minutes at the end of the workshop, it was clear the students would not be able to understand composing functions nor assigning a new variable to get the output of a function without heavy guidance from instructors. If these more complicated tasks are to be required in the future, more frontloading of how a programming language works would be required to give students the understanding to complete the task.

6 Conclusions and Future Work

We left the first workshop feeling confident the Python Music Environment shows potential for being an engaging learning too. It is clear from witnessing the students struggle with function calls

and function composition, work remedying this will need to be undertaken in the future. A possibility is building in a more formal introduction to programming syntax and semantics so that defining variables and understanding the exact syntax of function calling is more precise. Additionally, the documentation can be pushed to better visibly distinguish the difference between our functions and predefined variables. What this also suggests is that the Python Music Environment might be less suited for a first introduction to programming, but more suited for someone with very basic knowledge of programming syntax. This would also suggest the environment is well suited for students who are transitioning from a block-based introductory language to more traditional text-based languages.

6.1. El Sistema

We plan on moving forward with our collaboration with El Sistema. There is a potential for expanding our curriculum as a daily extra-curricular activity at El Sistema's week long summer camps in June and July. In preparation for this, there is much work that can be done to improve our curriculum. In addition to the goal of improving perceptions of computer science, we should select a small set of specific concepts that we'd like to teach and brainstorm activities to support these goals. We plan to investigate methods to scaffold discussions about the problems we encountered, including confusion about the differences between functions and arguments, lack of understanding about variables and data types, and lack of time to discuss composition of functions.

6.2. Python Music Environment

More work will need to be done improving and expanding upon the capabilities of the Python Music Environment. First, we intend to implement an web-based version of the application. Drawing from the findings of EarSketch developers, the current version is feasible for pilot studies, but will be challenging to install and setup in schools at a larger scale. If the application is contained within a website, it will be much more easily accessed and scalable. Second, we aim to research methods of making the language usable in live coding, a performance practice where the programmer improvises music and, in some cases, projects his/her code to audience members as it is written. Such an improvement will first require adding a loop feature which continually plays music and allows it to be updated in realtime. In order to be truly musical, our interface should support Level 4 Liveness, the highest level of feedback a programming environment provides to its user.[4] Since our target population are musicians, the Python Music Environment should respond like an instrument. Finally, we have brainstormed many other features including the ability to save musical output as both an audio file and a piece of sheet music, the inclusion of higher quality sounds, and the ability to interface with hardware inputs such as MIDI instruments and outputs such as musical robots.

References

1. Burg J, Romney J, Schwartz E. Computer science "big ideas" play well in digital sound and music. Proceeding of the 44th ACM technical symposium on computer science education. ACM; 2013:663-668. http://0-dl.acm.org.libraries.colorado.edu/citation.cfm?id=2445391
2. BlockyTalky. Retrieved April 27, 2016, from https://github.com/LaboratoryForPlayfulComputation/blockytalky
3. El Sistema Colorado: Accomplishments. (2013) Accessed April 27, 2016 from http://www.elsistemacolorado.org/our-program/accomplishments/
4. Church, L., Nash, C., & Blackwell, A. F. (2010). Liveness in notation use: From music to programming. Proceedings of PPIG 2010, 2-11.
5. Cycling '74 Max 7. (n.d.). Retrieved April 24, 2016, from https://cycling74.com/
6. Freeman, J., Magerko, B., McKlin, T., Reilly, M., Permar, J., Summers, C., & Fruchter, E. (2014, March). 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* (pp. 85-90). ACM.
7. Labor Force Characteristics by Race and Ethnicity, 2014. In *BLS Reports.* (2015, November). U.S. Bureau of Labor Statistics, Report 1057. Accessed April 27, 2016, from http://www.bls.gov/opub/reports/race-and-ethnicity/archive/labor-force-characteristics-by-race-and-ethnicity-2014.pdf
8. Lopez, M.H., Gonzalez-Barrera, A., Patten, E. (2013, March) Closing the Digital Divide: Latinos and Technology Adoption. In *PewResearchCenter: Hispanic Trends.* Accessed April 27, 2016, from http://www.pewhispanic.org/2013/03/07/closing-the-digital-divide-latinos-and-technology-adoption/
9. Mahadevan, A., Freeman, J., Magerko, B., & Martinez, J. C. (2015). EarSketch: Teaching computational music remixing in an online Web Audio based learning environment. In *Web Audio Conference.*
10. Margolis, Jane. *Stuck in the Shallow End.* Cambridge, Mass. MIT Press, 2008.
11. Peppler, K. STEAM-powered computing education: Using e-textiles to integrate the arts and STEM. IEEE Computer, September 2013, pp. 38-43. Accessed April 24, 2016 from http://kpeppler.com/Docs/2013_Peppler_STEAM-Powered.pdf
12. Python Music Environment. (n.d.). Retrieved April 24, 2016, from https://github.com/Huriphoonado/python-midi-env
13. SonicPi. Retrieved April 27, 2016, from http://sonic-pi.net/
14. SuperCollider. (n.d.). Retrieved April 24, 2016, from http://supercollider.github.io/