TunePad Playbooks

Designing Computational Notebooks for Creative Music Coding

Michael S. Horn
Northwestern University, Computer Science and Learning Sciences
michael-horn@northwestern.edu

Amartya Banerjee
Northwestern University, Computer Science and Learning Sciences
amartya@northwestern.edu

Matthew P. Brucker
Northwestern University, Computer Science and Learning Sciences
matthewbrucker2025@u.northwestern.edu

ABSTRACT

This paper describes the design of an online learning platform that empowers musical creation and performance with Python code. For this platform we have developed an innovative computational notebook paradigm that we call TunePad *playbooks*. While playbooks borrow ideas from popular computational notebooks like Jupyter, we have designed them from the ground up to support creative musical expression including live performances. After discussing our design principles and features, we share findings from a series of artifact-centered interviews conducted with experienced TunePad users. Our results show how systems like ours might flexibly support a variety of creative workflows, while suggesting opportunities for future work in this area.

CCS CONCEPTS

• **Human-centered computing**; • **Human computer interaction (HCI)**;

KEYWORDS

Computational literacy, computational notebooks, music, design, playbooks

ACM Reference Format:
Michael S. Horn, Amartya Banerjee, and Matthew P. Brucker. 2022. TunePad Playbooks: Designing Computational Notebooks for Creative Music Coding. In *CHI Conference on Human Factors in Computing Systems (CHI '22), April 29–May 05, 2022, New Orleans, LA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3491102.3502021

1 INTRODUCTION

The HCI community has long been interested in computational tools to support and expand human creativity (e.g. [28]). Among the many creative domains of interest, music has been one of the most enduring and most transformed by digital technologies, especially over the past 20-30 years. A growing area of research in this larger domain is the intersection of music and computer programming languages [16, 31]. Much in the way that sheet music is a language for expressing musical ideas, code (with its creative, mathematical, and algorithmic affordances) can also be a powerful medium for composing and performing music. Early explorations in music composition on mainframe computers in the 1950s and 60s have given way to an explosion of languages and systems that elevate coding to the level of performance art [5, 16, 31, 36]. For the most part, research in this area has emphasized programming language design—*what are the affordances of computational languages for thinking about and expressing musical ideas?* However, there has been much less emphasis on the holistic design of these systems from an HCI perspective—*how do we design tools to effectively support creative workflows that integrate multiple tools, representations, and collaborative practices in the context of broader creative ecosystems?*

In this paper we share the design of a novel learning platform called TunePad that supports musical composition and performance using the Python programming language. TunePad has been used by several hundred students in afterschool programs and summer camps[1] over the past four years. For this platform, we have developed an innovative computational notebook paradigm that we call *playbooks*[2] (Figure 1). While playbooks borrow ideas from other computational notebook systems such as Jupyter and Google Colaboratory [15], we have designed them from the ground up to support creative musical expression including live performances. The end result is something quite different in functionality and purpose. After discussing our design principles and features, we share findings from a series of artifact-centered interviews conducted with experienced users. Our results show how systems like TunePad playbooks might support a variety of creative workflows, while suggesting opportunities for future work in this area.

2 BACKGROUND

2.1 Related work in music and coding

There is a rich history of computer programming languages and environments designed to support musical expression [16, 31]. Wang describes three eras of music programming beginning with a family of languages collectively known as MUSIC-N that allowed programmers to both define sounds (how digital instruments generate audio) and scores (what instruments play). Direct descendants of MUSIC-N include languages such as Common Lisp Music [30], Nyquist [29], and Csound [17]. As computer processing power has increased, a number of influential real-time systems have emerged including graphical, dataflow languages such as Max/MSP and Pure Data [20, 23], and text-based environments such as SuperCollider [35] and ChucK [32]. Wang's third era of music programming describes

[1] See [12] for a paper that briefly describes findings from a summer camp program with TunePad.
[2] The term playbook combines the idea of "dropping the beat" or "dropping an album" with computational notebooks.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CHI '22, April 29–May 05, 2022, New Orleans, LA, USA
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9157-3/22/04…$15.00
https://doi.org/10.1145/3491102.3502021

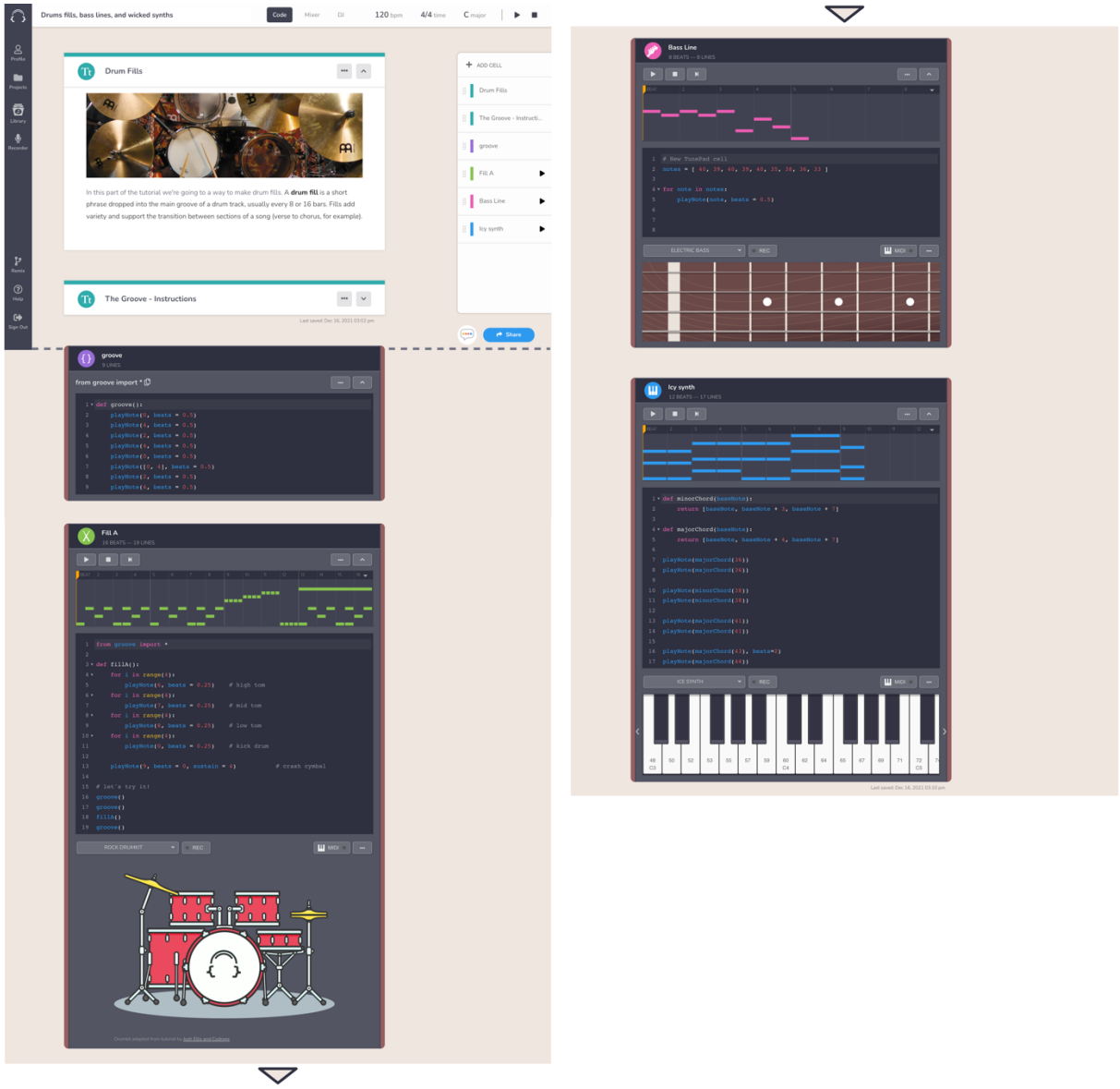

Figure 1: A TunePad playbook with text cells (the second in a compact state), a code cell, and interactive drum, bass and synth cells.

the current proliferation of languages and systems to support live performance [1, 31], reflected in communities such as toplap.org.

Researchers have also been interested in the educational possibilities created through the combination of music and computer code. Here the emphasis is on simplified programming constructs and more accessible and/or popular languages such as Python, Ruby, Java, and blocks-based environments such as Scratch [25] and Microsoft's MakeCode. Influential systems in the educational space include Music Logo and Impromptu developed by Bamberger and colleagues [4]. Music Logo was a minimal addition to the Logo programming language that provided built-in functions for playing musical notes. Though the additions were relatively simple, Bamberger's research was able to demonstrate a rich conceptual space that these new capabilities opened for learners [3, 4]. Other educational environments such as Scratch and Microsoft's MakeCode also provide basic building blocks for music and sound effects. Because these platforms are accessible and widely used, they open music making capabilities to a broad audience. However, in a recent critique, Payne and Ruthmann [22] identify several limitations in the use of Scratch for teaching music for novice learners. They point out that for learners who aren't familiar with music theory, it can be difficult to compose at the level of notes and sounds. Bamberger critiqued her own Music Logo language for much the same reason [4]. These factors, coupled with technical issues requiring unintuitive

workarounds to get multiple tracks to play synchronously, result in serious hurdles for novice learners to make music. Although we do not consider TunePad to be a response to all of the issues raised by Payne and Ruthmann, we do see it as a more serious music making environment that is still accessible to learners.

Other systems such as Bamberger's Impromptu [4] and the popular EarSketch platform created by researchers at the Georgia Institute of Technology [10], have emphasized engaging students in musical creation "without the added barrier of entry of learning music theory about harmony, melody, chord progressions" [18]. With EarSketch learners use either Python or JavaScript to remix pre-produced musical samples into full-length songs in a digital audio workspace (DAW). Quantitative studies with EarSketch have included over 500 students and have shown statistically significant increases in intention to persist and attitudes toward computing, typically with medium or large effect sizes [10, 18, 33]. TunePad was developed in collaboration with the EarSketch team to be a companion to environment, but TunePad's focus is more on music composition from notes and beats rather than from pre-recorded samples.

Perhaps the most closely related system to TunePad is the popular Sonic Pi platform [1], created by Sam Aaron and colleagues and first released in 2012. Sonic Pi ships with the Raspberry Pi platform and can also be downloaded for free for Windows and Mac OS. Sonic Pi is a domain specific IDE (Integrated Development Environment) featuring a text editor with language support, music visualization capabilities, and built-in documentation. It uses a SuperCollider [35] server on the backend to generate high-quality, real-time audio with live coding capabilities. The Sonic Pi language is related to Ruby and supports multi-threaded constructs to layer audio together to create complex compositions.

While TunePad shares some of the features of Sonic Pi, it differs along several dimensions. On a superficial level, TunePad is a browser-based app instead of installed software. This offers advantages for schools and other educational programs that use Chromebooks or that face restrictions installing software. The more substantive differences, however, have to do with how code is organized and musically synchronized. Sonic Pi uses a parallel processing model where different musical parts are structured as threads that can be synchronized with message passing constructs. TunePad, instead uses a model of multiple musical instruments organized as cells in a notebook (see the Design Overview section for details). The instruments can be played manually (with a mouse, keyboard, or MIDI device) or they can be programmed with short Python scripts coded in the cell. Another major difference from a user-experience perspective is that music in TunePad is visualized at the level of an individual cell in the form of an interactive piano roll, waveform, or music notation (see Figure 2). This is in contrast to Sonic Pi where the real-time audio is visualized at the level of the entire set of running threads.

2.2 Computational Notebooks

Computational notebooks such as Jupyter [15] and Google Collaboratory have become pervasive in data science and research communities over the past decade [13]. The standard notebook architecture is based on a read-eval-print loop (REPL) workflow common with scripting languages such as Python. From the user's perspective, notebooks take the form of vertically arranged cells in a browser window that can contain code, rich text, multimedia, data, and interactive plots and visualizations. Code cells are processed by the notebook's kernel, which maintains the shared runtime state. Code cells have an implied top-to-bottom ordering, meaning that code near the top of the page is typically intended to be run before code below it. In practice, however, code cells can be run in any order leading to "enormous" problems for novice users in educational settings who may not fully understand the hidden runtime state of the notebook [8, 13]. Johnson [13] summarizes several pitfalls and benefits of computational notebooks for use in post-secondary classrooms. Among the pitfalls: notebooks are difficult for students to install due to the server/kernel architecture; the hidden runtime state may be unpredictable if cells are executed in an arbitrary order; notebooks offer limited debugging features; and there are security issues inherent in running and sharing notebooks. For Johnson these concerns are outweighed by the pedagogical benefits that come with a narrative structure that facilitates explanations and worked examples.

While TunePad playbooks are superficially similar to Jupyter-style notebooks, they are architecturally different (Table 1). We describe playbooks in more detail below, but the main difference is that playbook cells share a lexical namespace instead of a runtime space. The implication of this architecture is that there is no need for a shared kernel or locally-installed client-server software. It also means that dependencies between cells become more explicit and there is no hidden runtime state. Of course, the playbook architecture comes with tradeoffs. The lack of a shared runtime environment and data flow between cells makes playbooks impractical for data science applications. However, we will argue that this tradeoff optimizes music creation.

There are other examples of music creation using computational notebooks[3]. The Jupylet project, in particular, uses a Python port of Sonic Pi in combination with Jupyter-style notebooks. It is not surprising that people have explored music with notebooks. However, to our knowledge, TunePad is the first platform to create a serious design combining computational notebooks with music that addresses the limitations of standard notebook paradigms.

3 DESIGN PRINCIPLES

TunePad is influenced by a long history of Constructionist learning environments that feature programming as a medium for creation (e.g. [1, 9, 21, 25, 34]). A broad goal of this project is to provide *authentic* learning experiences. However, the term authentic is neither binary nor uni-dimensional. Learning experiences can have degrees of authenticity along dimensions such as fidelity to professional tools and practice, and the extent to which activities are personally meaningful to learners [19, 27]. There are also cultural dimensions to authenticity, meaning the degree to which activities are culturally resonant and true to learners' cultural identities. Inauthentic experiences can feel fake, culturally irrelevant, or worse. Following [27] we strive to be authentic in the tools of practice for both coding and music, leading to six broad design principles:

[3]See https://blog.changshe.io/making-music-in-a-jupyter-notebook-19c57791e636 and the Jupylet project https://jupylet.readthedocs.io

Table 1: Key differences between Jupyter-style computational notebooks and TunePad playbooks

Jupyter-Style Notebooks		TunePad playbooks
Optimized for data science	→	Optimized for music and art
Optimized for professionals	→	Optimized for learners
Designed for REPL workflows	→	Designed for creative workflows
Shared runtime state (kernel)	→	Shared lexical state
Can require local installation	→	No installation needed
Static output	→	Interactive output
Not "performable"	→	Performable artifact

3.1 Playful Interaction First

Provide a playful, exploratory environment that encourages learners to experiment with sounds and music without feeling overwhelmed by intimidating technical details. In other words, we hope that learners will first see TunePad as a platform for creative expression and later recognize the role of code in empowering music.

3.2 From Playful Exploration to Deep Dive

As with Scratch [25] and Logo [21] before it, we are committed to the ideal of "low floors, wide walls, and high ceilings" [24]. We aim to make it easy for learners to get started but also support them as they progress towards expertise when they are motivated and ready. Because the underlying language is Python, there's a high ceiling when it comes to the complexity of programs that can be created.

3.3 Music that Sounds Good

Provide a satisfyingly rich world of music and code to explore. Here we are fortunate to have access to the Web Audio API built into all modern web browsers. Web Audio aims to support professional grade audio production in the browser, giving us the ability to design sophisticated synthesizer patches, filters, and effects. We have also focused on creating high quality music with tools that closely mirror real-world audio production software.

3.4 Performability (Live Coding)

Existing live coding environments such as Sonic Pi [1] make it possible for users to develop musical compositions in real time as part of a live performance. In this sense, performance artists are creating compositions that showcase not only their music ability but also their technical prowess. We designed TunePad to provide similar live coding features described below.

3.5 Shareability and Collaboration

A cornerstone of Constructionist learning environments is the ability to share personally meaningful artifacts with others. This has motivated us to make sharing and remixing as easy as possible. TunePad playbooks can be exported to MP3, WAV and MIDI, and the ability to add text and multimedia cells to projects (described below) is intended to support creative expression with artwork, lyrics, poetry, and personal reflections. Similar to the Scratch community [25], our aim is for users to create their own tutorials to promote a culture of teaching from one another in the spirit of computational participation [11]. We also support real-time collaboration with other creators in the same project.

3.6 Coding Amplifies Creativity

It is essential for us that coding is seen as an empowering language of musical expression that allows learners to go above and beyond what is possible with typical music production tools. There is a real risk (especially for beginners) that coding ends up hindering musical expression. For example, step sequencers make it trivial to play around with different beat patterns without coding anything. Further, if users are accustomed to reading sheet music or other standard musical notations, the transition to code can feel daunting and frustrating.

4 DESIGN OVERVIEW

TunePad is a free, online platform for creating music with Python code. The platform includes learning resources such as step-by-step tutorials, covers of popular songs, and a library of pre-coded samples such as drum loops and synth pads. Projects in TunePad are organized in the form of computational notebooks called playbooks (Figure 1) that consist of an arbitrary number of cells containing Python code, playable musical instruments, rich text, and multimedia elements such as images and video. Instruments (e.g., Figure 2, bottom) can be played with the mouse, keyboard, or attached MIDI device.

They can also be programmed with short Python scripts (Figure 2, middle). The resulting audio will automatically loop rounding up to the nearest measure length. Playbooks synchronize the audio of multiple instruments at the beat and measure level according to the project's time signature.

The script in Figure 2 plays a simple run of hi-hats punctuated with random stutter steps (a common pattern in popular music). This code includes a for-loop that sets the basic repeated pattern, and a conditional statement that decides whether to play a single 8th note or two 16th notes based on a random number. These 10 lines of code convey underlying patterns that can be more difficult to see with more traditional musical notation systems. Specifically, there is a foundational pattern that should be accented with seemingly random stutter steps that occur roughly two times out of ten.

Playbooks automatically evaluate code changes on the client side in the background whenever there are edits and the cursor is moved to a different line. When a change is made, the piano roll immediately refreshes to provide a visual cue for the updates, and

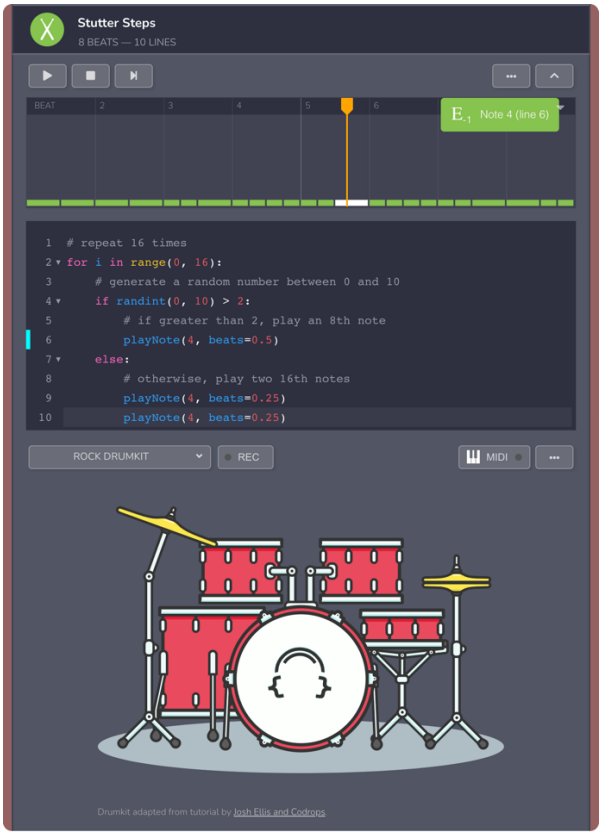

Figure 2: In TunePad, each cell may contain Python code (middle) that is tightly integrated with interactive piano rolls (top) and playable instruments (bottom).

the resulting audio will be cued to start at the beginning of the next loop.

The combination of multimedia elements and rich text in computational notebooks makes it easy to build interactive tutorials where learners can run code in context and then tinker with the scripts in individual cells or remix a project in its entirety [13]. With TunePad this makes it possible to add lyrics, personally meaningful images, or embedded videos. For example, when creating a cover, users will often embed a YouTube video to show what the original song sounds like.

4.1 Independent evaluation and cell scope

Playbook cells share a lexical namespace (through Python *import* statements) instead of a runtime space. For example, a learner might have one cell called *groove* that defines variables and functions.

```
kick = 0
hat = 4
snare = 2
def groove():
    playNote(kick, beats = 0.5)
    playNote(hat, beats = 0.5)
    playNote(snare, beats = 0.5)
    playNote(hat, beats = 0.5)
    playNote(kick, beats = 0.5)
    playNote([kick, hat], beats = 0.5)
    playNote(snare, beats = 0.5)
    playNote(hat, beats = 0.5)
```

The other cells in the playbook can then import and use this code:

```
from groove import *
groove()
```

The syntax is the same as standard Python, with the name of each cell becoming the module name used for imports. If a cell redefines variables or functions, those changes are local to that cell. The implication of this architecture is that there is no need for a shared kernel or locally installed client-server software. It also means that dependencies between cells become explicit through import statements in code and that there is no hidden runtime state that can change in unpredictable ways depending on the order in which cells are run. This helps simplify some of the more confusing aspects of Jupyter-style notebooks for novices.

There is an added advantage that syntax errors in one instrument will not stop playback of the rest of the project. Unless code is imported from other cells, mistakes in one cell will not bog down other parts of the composition. We think this helps creators tinker with code fluidly, without undue concern about ripple effects. Independent evaluation is especially helpful when multiple people are collaborating on a shared project; a syntax error in one person's cell won't break the music output for the others.

4.2 Support for live coding

TunePad adopts concepts from computational notebooks but also adds the ability to perform and improvise music in real time. Live Coding refers to the real-time editing of computer code to generate music as part of a live performance [1, 5, 9, 36]. In this way, the performance is as much about the resulting music as it is the performers' technical virtuosity. Performance is no longer just about the end product (the music); it is as much about exposing the process of making music – the algorithms and computation behind it – for everyone to see [9, 36]. Live coding in TunePad is made possible because: a) cells can be run independently to generate music in layers; b) multiple instruments are automatically synchronized by beat and measure; c) changes in code are automatically recompiled in the background without having to pause and restart the resulting music; d) and, changes to the audio output are cued to begin at the start of the next loop (similar to Sonic Pi's *live_loop* construct [1]), which means that musical changes are incorporated seamlessly without stopping the beat. The video figure shows examples of live coding with TunePad.

4.3 Collaborative editing & pair programming

Music is inherently collaborative. We have been inspired by community-centered participation structures (drum circles, peace circles, improvisational performance, and jam sessions) to engineer playbooks so that multiple participants can simultaneously contribute to shared online compositions by writing computer code to play virtual musical instruments. Participants can edit code in the same cell, or they can each add their own musical instrument

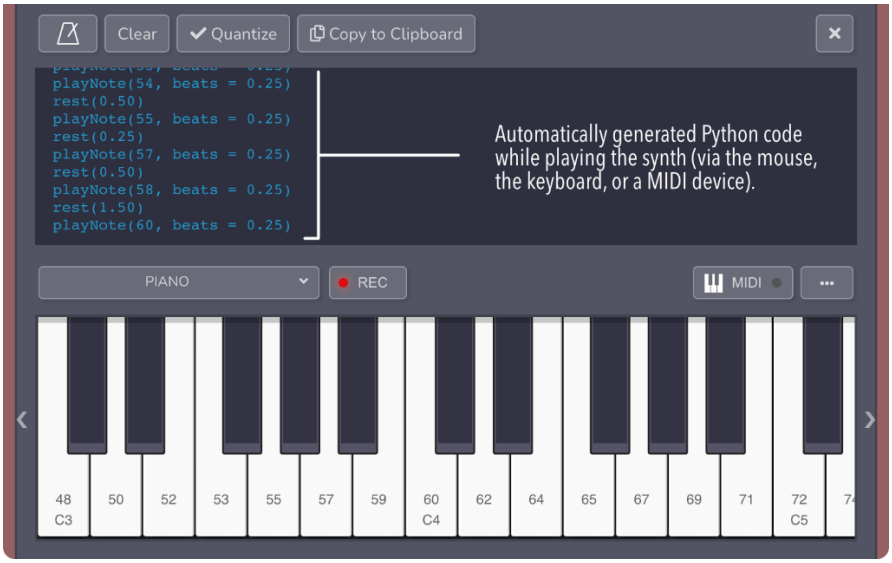

Figure 3: Enabling the "Record" button while playing an instrument (in this case, a synth) generates Python code that can serve as a helpful and quick starting point.

to the same project. Imagine one participant adds a drum cell, another adds a bass, and a third adds a piano keyboard. Compositions emerge in real time through the collective efforts of participants who learn to listen to one another and work together as a team. The distributed, collaborative features of playbooks have also proven valuable during online learning in the pandemic where students in a virtual session can all create cells in the same project, and the instructor can see everyone's work and provide help in real time without having to change whose screen is being shared.

The workflow for collaborative music making on TunePad looks something like this: a learner creates a new playbook and marks it "Public" to generate a unique and shareable link. Using that shared link, other learners can join that playbook and can see (and hear) the work of their peers emerge in real-time. In terms of the collaborative infrastructure, this works similar to Google Docs. By synchronizing code changes from multiple creators in a shared project, we hope to achieve the feel of a live jam session that retains a large degree of creativity and playfulness. We see similar ideas in platforms like Scratch remixes [26] where students share their code with one another. However, we think that collaboration on shared projects creates a dramatically different form of participation and learning. Another notable aspect of this collaborative workflow is that it affords pair programming and spectating. Learners can choose to simply watch their peers (or an instructor) while they are composing music.

4.4 Multiple representations of music

For instrument cells, the Python code (Figure 2, middle) is closely coupled with the piano roll (top) and the playable musical instrument (bottom). The playable instrument has a "record" feature that can generate Python code corresponding to the notes currently being played (Figure 3). The piano roll can be swapped out for a waveform visualization or standard sheet music notation (Figure 4). As a learner writes or edits a line of code, the changes are reflected in the piano roll instantaneously. This closely coupled visual representation not only provides immediate feedback, but also aids debugging—hovering over the piano roll reveals the note being played as well as the line of code responsible for that note—and encourages experimentation. In general, we made a conscious effort to bring in forms and conventions of digital audio workstation (DAW) software such as Ableton, Logic, and GarageBand.

4.5 Debugging features

Playbook instruments provide access to a console for debug logs as well as the ability to trace through lines of code, one at a time. Each time a note or rest is added to the code, we create an automatic breakpoint for code tracing. The Step button built into instrument cells will run code to the next breakpoint, playing notes along the way. In addition to code debugging, the interactive piano roll helps with 'musical' debugging. Creators can move the playhead on the piano roll to highlight the line of code that is responsible for a particular note (or notes), making it easier to edit or debug the line in question. Using online programming resources such as Stack Overflow is a useful skill for learners to pick up. Whenever there is an error or exception in a cell, the error message/window includes a direct link to Stack Overflow results from both Google and DuckDuckGo.

4.6 Coded sound library

TunePad includes a library of coded musical samples that can be added to a playbook. The mechanism is evocative of most audio production tools that have pre-recorded sound-loops or samples with one crucial difference: these samples are created with Python code that gets added as a new cell in a project. The coded samples support learning by example and can be easily 'remixed' by editing or adding new lines of code.

Figure 4: Multiple representations: piano roll (left), waveform (middle), and sheet music.

Figure 5: Recording Studio: creators can record audio samples and manipulate them before using them in a cell or programmable instrument. (right) DJ Tools: virtual turntable to mix the tracks in a playbook.

4.7 Authentic music production tools

TunePad positions music and coding as *equal peers* – in particular we not only consider how music production can be a context for developing computational literacy, but we also explore how coding can be seen in terms of practices normally associated with making music. Some examples of the music production tools built into TunePad are:

4.7.1 Recording Studio. TunePad has a built-in recording studio. Using a microphone as an input, creators can record short samples, trim or edit the sample by changing its volume and pitch, add a reverb, or tinker with the low, mid or high frequencies (Figure 5, left). Once this sample is edited, it can be used within a cell programmatically with a playSound function call.

4.7.2 Mixer and DJ Tools. TunePad includes mixing tools and a virtual turntable (Figure 5, right) that lets creators mix the tracks they have programmed. They can create flowing transitions, fade-in and fade-out tracks, change the tempo on the fly, and change the gain and equalizer settings in real time. Anyone unfamiliar with music mixing tools can learn the basics, while those already familiar with virtual turntables would feel right at home. In live coding scenarios, the DJ tools afford yet another avenue for creative expression.

4.8 Multiple *voices* for each instrument

Most music production tools have a rich set of instruments and voices to choose from. In TunePad's case, we made it easy for a creator to change not only the sound from a programmable instrument, but also change the instrument itself without having to change the code. For example, in Figure 6, changing the snare sound from a Rock Drumkit to an 808 Drumkit is just a couple of clicks away. As we discuss later, this design feature is commonly used by creators trying to compose covers of popular songs and get it to sound as close to the original as possible.

5 SOFTWARE ARCHITECTURE AND IMPLEMENTATION

Figure 7 is a high-level software system diagram with two connected users working on a single Playbook; one tinkering with a drum, and another working on a synth. On the client side, TunePad uses the Dart programming language (dartlang.org) along with standard web technologies (HTML, CSS, JavaScript, and the DOM). We use the Skulpt library (skulpt.org) to compile Python scripts on the client-side in real time as the user makes edits. Our music functions (e.g. playNote and rest) generate print statements with stringified JSON objects. The output from a Python script then allows

us to produce a musical trace that we iterate through to generate audio events for the Web Audio API. We send other Python print statement output to a Python console visible to the users. We intercept print statements as a Python script is being compiled as a means to prevent accidental infinite loops that would hang the browser. In other words, we can halt a program that generates too many output lines (as determined by a configurable threshold value). Audio playback is based on the Web Audio framework, which provides very precise timing. Web Audio also allows for sophisticated effects, filters, timing envelopes (e.g. ADSR envelopes), and modular synthesis. We expose some of these features through language-level constructs. Our future work plan involves exposing more of these features to users through additional language functions. To support collaborative editing, we use a WebSocket server that sends messages corresponding to the current state of the playbook to each connected client (i.e. user).

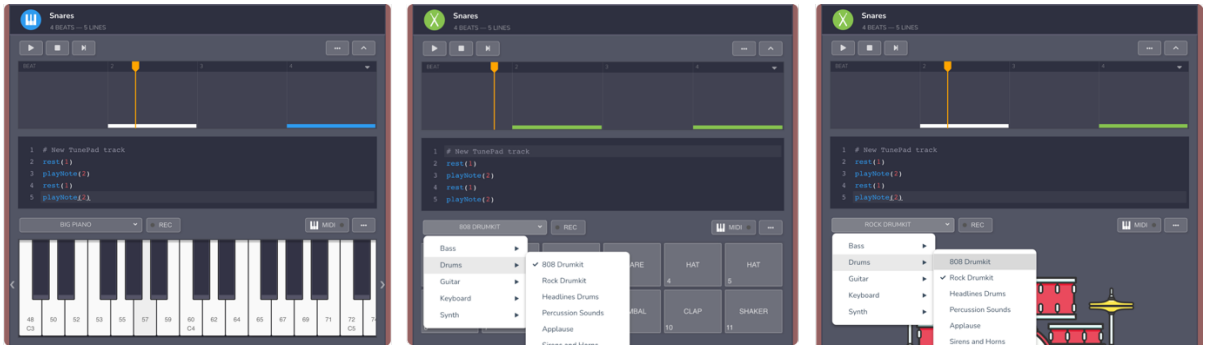

Figure 6: A drop-down menu lets creators change the type of instrument or the sound/voice itself without changing the code. In this case, a synth is replaced by an 808 drumkit, which in turn is swapped out for a rock drumkit.

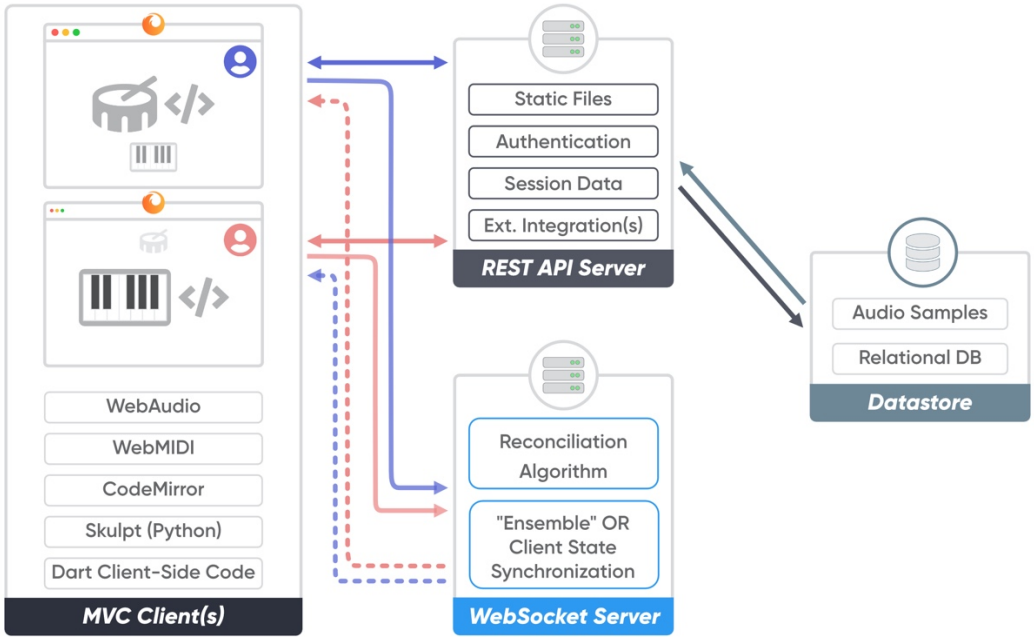

Figure 7: TunePad's software architecture and core technologies.

6 EXAMPLES OF CREATIVE PRODUCTION WITH TUNEPAD

TunePad has been used by several hundred students in afterschool programs and summer camps over the past four years and has undergone several major design revisions. To enable new forms of creative production that blends computational literacy and music, TunePad's design had to go beyond providing a "low floor" for novices and afford rich forms of production for learners with more coding or music production experience (i.e. provide a "high ceiling"). For the purposes of this paper, we have chosen to report on how TunePad's design works with more advanced learners and their creative practices.

6.1 Study population

New learners are introduced to TunePad in a variety of settings, including schools, libraries, community and youth centers. Over the project's lifetime, we have hired dozens of facilitators to help new learners. We refer to these facilitators as "coaches" in order to emphasize their role as mentors and guides for program participants. Most of our coaches are undergraduate Computer Science students, and a significant proportion of them are also enrolled in music composition or performance programs, or they play a musical instrument. While most had prior experience with Python programming, all were new to the TunePad platform.

We iteratively developed a set of curricula for TunePad by working closely with the coaches. Many coaches also developed their

own activities and exercises specific to the groups they taught, and they contributed to a library of compositions (mostly covers of existing songs) featured on the TunePad website for younger students to learn from. This dynamic was evocative of prior research on near-peer mentors in computer science education [6]; mentors' coding projects motivated younger students and it showed them "what could be done" with computer programming. Similarly, Kafai et al.'s [14] examination of the experiences of STEM near-peer mentors presented a fluid picture of mentorship whereby mentors took on dual roles as teachers and learners. We view coaches' experiments in music-making integral to developing a creative community that motivates future learners.

6.2 Data collection and methods

To illustrate the creative affordances of TunePad, we interviewed six participants – 2 women and 4 men – from our most recent cohort of coaches. All of the participants were undergraduate students who had "coached" younger learners in TunePad camps during spring and summer of 2021—examples include a weekly virtual camp for middle school students, virtual summer programming for youth ages 18-24, or a weeklong in-person summer camp. All coaches had taken at least one undergraduate-level computer science class.

In terms of data, we looked at the coaches' TunePad playbooks as well as video and audio recordings of semi-structured interviews that were conducted virtually. These coach interviews were based on Brennan and Resnick's artifact-based interview format [7], modified to reveal their music-making and computational workflow. Participants were asked to recreate a song they had previously composed in TunePad while staying as close to their original composition and arrangement process as possible. Participants were free to use any supporting materials they wished, including listening to the original song (all participants' chosen songs were covers), using sheet music, or referencing their original TunePad playbook. We think that recreating a song helped highlight specific interactions with the platform and let participants demonstrate their range of creative expression without the overhead (or pressure) of having to start from scratch in a time constrained interview. Interviews were transcribed, and video recordings were coded by the research team. A codebook was developed iteratively based on repeated viewings of interviews and broadly classified into two types of codes: specific interactions, i.e., moments where participants make use of a specific design feature of TunePad, and higher-level interactions that did not directly correspond to a specific design feature. The codes were refined by the research team to highlight recurrent themes that we describe below. Methodologically, we started with several rounds of open coding across all participants to develop a codebook consisting of around 15 codes. This first round of coding focused on trying to broadly characterize coaches' creative processes. Comparing similarities and differences among coaches through extensive discussion helped us consolidate codes and move towards themes. Our analytical focus, in turn, narrowed to focus on understanding how coaches brought in and coordinated various tools of production and representations of music. We also attended to easier and harder parts of their process and their stated goals for creating music with the platform.

7 RESULTS AND EMERGENT *THEMES*

The coach interviews highlight the differences in the creative workflow while using TunePad. As a group of individuals with varied musical preferences and levels of expertise, the coaches produced a wide range of final musical pieces. We describe the common interaction patterns with TunePad and also report on some higher-level variations in participants' (the participant / coach names are *pseudonymized*) overall composition processes.

All participants followed a highly iterative composition process, often alternating between periods of coding and composition followed by "musical debugging". That is, playing one or more of their TunePad cells and comparing it to the original song, or playing completed and "in progress" cells simultaneously to verify timing. Since all the songs were covers, changes made between iterations were directed towards refining the composition towards a "correct" version of the song rather than open-ended musical experimentation. However, even within this more constrained context, participants employed considerable ingenuity and creativity.

7.1 Theme 1: Using multiple representations and playable instruments

Bamberger and diSessa [2] argue that different music-related representations foreground different aspects of sound: "each [representation] captures some features while ignoring or minimizing others." Because participants in our sessions were covering existing songs, their compositions were grounded in the original pieces, with the goal of the composition being to replicate the original sound in a manner deemed accurate or suitable.

All the participants played the original song multiple times and listened for melodies/hooks or percussive patterns that stood out. Particularly for participants who were less familiar with conventional sheet music notation, the virtual, playable musical instruments at the bottom of each TunePad cell served as an important tool for translating between different types of musical representations. In keyboard/synth cells (Figure 3 and Figure 6, left), for instance, the virtual playable instrument includes both the name of each note and the corresponding numeric value used by the playNote function. This affords a one- or two-step translation from alternate representations, including online guitar tabs featuring note names. One participant, who had no previous music composition experience, was working on the song *Fake Love* by Drake. He made extensive use of a piano tutorial video he found on YouTube, which featured a top-down video of hands playing the melody on piano, as well as an overlaid piano roll view in which each note scrolls down from the top of the screen in time with its position in the song. He made use of the text cells' capability for embedding videos, placing a text cell directly above the cell he was working in and embedding the tutorial video within it. To compose the song, he repeatedly moved between the embedded tutorial and his keyboard cell, visually comparing the position of the falling notes on the video's piano keys to the virtual piano keys in the TunePad cell, then noting the overlaid note number on the TunePad cell before writing playNote statements. Thus, the playable instruments served as a straightforward translation tool for a variety of visual representations. Additionally, nearly all participants used the playable instruments to quickly test out a section of the song based on what

they heard in the original, allowing for translation based on sound as well.

7.2 Theme 2: Using musical constants

Aside from using the playable instruments to translate between external representations of a song and the numeric representations used by TunePad, several participants created mappings in code between a numeric representation and a note name, either by declaring variables or by using a constants module built into TunePad. For a participant with prior music composition experience, the constants allowed for a more direct process of converting between representations—he was able to read a sheet music representation and construct playNote statements for each chord, without needing to use the playable piano. This underlines the fact that a particular melodic phrase could be constructed by leveraging a creator's composition experience and the availability of sheet music, without having to constantly switch back-and-forth between coding and playing the notes on the keyboard. One nice property of computer code is the freedom to define variables and other abstractions to suit the task at hand and the creator's own workstyle.

7.3 Theme 3: Using the piano roll to verify and debug

The automatically generated "piano roll" view (see Figure 2 for an example) is an output representation that played multiple roles in the participants' composition processes. Participants used the piano roll view to quickly check and verify the output of a cell. They often looked at the piano roll before listening to the output to double-check that a track was the correct number of measures, or to ensure that a particular note or musical phrase started on the correct/intended beat. Because the vertical position of a bar in the piano roll view corresponds to the pitch of a note, the piano roll was often used to visually trace through a song to isolate a note of concern. One of the coaches, *Ian,* made use of this feature in recreating *Everything I Wanted* by Billie Eilish. In this example of extended debugging— after coding a melody, he scrolled through the piano roll view to verify that it was the correct number of beats. He played the cell and counted measure-by-measure to discover that he was one measure short. After listening to the original song, he realized one of his notes was one beat short; he then traced through the cell by moving his mouse cursor over each bar in the piano roll view and humming along. Upon reaching the problematic note, Ian moved to the corresponding line in the code section (hovering over an entry in the piano roll highlights the corresponding line of code) and fixed his mistake. As he summed it up: "*if I just had an IDE with an empty text document that I threw a bunch of Python code into, I'd really have no idea what this was doing without this visual aid [the piano roll] here*".

7.4 Theme 4: Different composition choices

Participants took distinct approaches to structure and compose their song in TunePad. Their overall vision(s) for each composition was informed by the intended audience for the piece (personal or for younger learners), their personal aspirations for the finished composition, and their take on TunePad playbooks being software or musical artifacts. We identify different strategies and how TunePad's design affords a wide range of creative approaches.

7.4.1 Composition Workflow. There was no all-encompassing workflow used to arrive at a "completed" piece. In creating covers of existing songs, participants exhibited considerable creativity. Participants made distinct choices in whether and how to iteratively refine individual cells or sections of their composition. One participant, *George*, described his typical process as starting by seeing "what [he] could mess around with," choosing to "tinker" with the timing of repeated bass notes early on rather than modifying the tempo of the entire composition. He was working on a recreation of *Time for Some Action* by N.E.R.D. After creating the percussion and bassline cells, he added an additional percussion cell with a different drumkit, attempting to layer additional drum sounds on top of his original percussion to make it more closely match the original sound. Choices of sound, note volume, instrument, or voice, and combining multiple sounds to match the original song were common across all participants; one participant (*Cass*) also used TunePad's built-in recording studio to record a shaker sound for her composition, since she felt that the sounds in the pre-recorded sound library were not "close enough" to the original song, *Billie Jean* by Michael Jackson. These decisions added variation and depth to even simple compositions, with some participants devoting time to precisely select a 'correct' sound in a cell before starting to compose, and others leaving it for later refinement. In particular, *Ian* categorized certain aspects of TunePad as "post-production," akin to what would be done in professional music production contexts. Thus, TunePad's design afforded workflows where participants could choose which elements of their sound to refine, and in which order to refine them.

7.4.2 Top-down vs. Bottom-up. A majority of participants adopted a "bottom-up" approach— starting by breaking the song into manageable pieces, coding each piece in chronological order, note-by-note, usually starting with the track that appeared first in the song. However, some coaches took a more "top-down" approach, starting by coding higher-level structural pieces and filling in the individual notes later. For instance, in coding one track of *Bad Guy* by Billie Eilish, *Naza* began by first defining variables for each note that would be played, writing the definition for the function that would play the notes, and within that function, writing the beginning of a loop. Then, she filled in the contents of the loop based on what needed to be looped in the song. No participant used solely a top-down or bottom-up approach, but this difference represents a shift in how participants think about their compositions as computational creations. By starting with a loop, *Naza* began with an understanding of the repeated components in her song and how they could be represented in code. While other participants recognized repeated sections as they were composing, and often copy-pasted lines of code in these instances, Naza's use of a for loop made this pattern immediately more prominent.

7.5 Theme 5: Emergent coding (best) practices

The choices of song structure and representation in code relate to participants' understanding of their compositions as software artifacts. As undergraduate Computer Science students, all participants

were presumably somewhat familiar with good coding practices. Participants exhibited this understanding numerous times, for instance by making use of variables and constants, which "make a whole lot more sense reading it than just random numbers everywhere" (*Ian*). In the interview participants also discussed other elements of their coding style; multiple participants mentioned that for most instances of writing code in TunePad, they would include helper functions or refactor their code to make it more readable not just for themselves but also because of their roles as coaches/mentors for younger learners.

7.6 Theme 6: Composition for intended audience and live performance

Participants' approaches to composition were also influenced by their intended audience. For example, *Marshall* was recreating *Mood* by 24kgoldn to present and perform for one of his sessions with middle-school learners. At various points in describing his composition process, Marshall identified how he structured his composition to make the piece easy for novice learners and "performable," such that he could show the piece to his students in a more engaging and illustrative way than by just pressing the play button. He split up the composition into multiple cells, each containing minimal code. The percussion was split into multiple drumkit cells, one for each distinct drum sound. By sequentially pressing the play button on each cell, he could "perform" the piece live by layering each element of the piece sequentially. He pointed out several places where he might have used a while loop or a function to make the code "cleaner," but that it would made the code less readable or confusing for beginners.

In another instance, Marshall noted a difference between how he would approach writing code if it were intended for himself rather than for an audience of novice learners. Realizing that a section "need[ed] to get pitched up a lot," he indented the section and added a line with transpose(12). this put the phrase in a transpose block which pitched it up an octave; after playing the section and verifying correctness, he undid his changes and modified the variables he was using by changing their assignment to their original value, plus 12. He said that the transpose(12): statement would be less immediately understandable to an external audience.

Finally, Marshall also made implicit creative choices in how he translated a song into a shortened version in TunePad. In line with his desire to make it performable for novice learners, Marshall chose a part of the song's chorus that looped seamlessly; that way, he could begin by playing one track, then "layer" each additional track on top by starting it while the other tracks loop continuously. This choice in converting a full song (several minutes in duration) to a short TunePad composition was different from other participants' approaches, which was typically to cover a section of a song which started at the beginning. Short segments designed to be looped are a natural fit for TunePad.

8 DISCUSSION

We hope that these artifact-based interviews make a convincing case that music making with code can be best characterized as part of a rich, creative ecosystem that goes well beyond a programmer interacting with an IDE. Our coaches made use of a variety of representations and resources—including YouTube videos, guitar tabs, MIDI keyboards, TunePad's built-in instruments, and abstractions that they created for themselves in code. They also relied heavily on the visual piano roll to musically debug layers of their composition. The multiple playbook cells offered flexibility in terms of the structural composition of their cover songs and how they would use their projects with their intended audience of middle school learners. Our highest-level takeaway is that when thinking about designs that bring together music and coding, we should consider how creators go about coordinating a wide variety of tools, representations, and workflows. It's important to focus on the overall creative environment, not just the programming language itself. A secondary takeaway is that our computational notebook paradigm, while far from the only solution, can support flexible workflows that bring together a variety of music making resources. As a side note, it was interesting to hear from coaches how tinkering with compositions through code changed the way that they thought about music. One of the coaches (*Naza*) recounted a moment of hearing a song on the radio when in the car with other TunePad coaches, when the group suddenly came to the collective realization that the song they were listening to could be easily broken up into parts that fit together as a TunePad composition: "*I feel like all of us have just kind of been like forced, and just like, I'm just never gonna get out of the habit now, of like separating the different parts of the melody.*" This same coach, a Computer Science student who had never taken an in-person CS course due to the pandemic, told us that she had become more aware of how computer science as a field of study can be creative (or artistic). We take these quotes as anecdotal but also encouraging that the TunePad playbook platform affords meaningful creative experiences with code.

9 LIMITATIONS AND FUTURE WORK

A major limitation of the current study is that we focus more on advanced learners (coaches) and not the middle school students who were using TunePad as part of summer out-of-school programs. It's clear that the experience of complete beginners will look different, and this should be documented in future work. From a technical and design perspective, there are many features and improvements we hope to add to the platform over time. A major area of future work will be to understand the collaborative music-making and live performance features better, including additional designs to support those use cases.

ACKNOWLEDGMENTS

The authors would like to thank Melanie West, Cameron Roberts, Izaiah Wallace, Nichole Pinkard, Jason Freeman, and Brian Magerko for their assistance with interface design, curriculum design, data collection, and coding. TunePad's graphic design was done by Tom Knapp (https://99designs.com/profiles/celadon). We thank our undergraduate coaches and many school/community partners. This research was supported by grants DRL-1612619, DRL-1451762, DRL-1837661, and DRL-2119701 from the National Science Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

REFERENCES

[1] Samuel Aaron and Alan F Blackwell. 2013. From Sonic Pi to Overtone: creative musical experiences with domain-specific and functional languages. In *Proceedings of the first ACM SIGPLAN workshop on Functional art, music, modeling & design*. ACM, 35–46.

[2] Jeanne Bamberger and Andrea diSessa. 2003. Music as Embodied Mathematics: A study of mutually informing affinity. *International Journal of Computers for Mathematical Learning* 8 (2003), 123–160.

[3] Jeanne Bamberger. 2013. *Discovering the musical mind: A view of creativity as learning*. Oxford University Press.

[4] Jeanne Bamberger. 2015. A Brief History of Music, Computers and Thinking: 1972–2015. *Digital Experiences in Mathematics Education* 1, 1 (2015), 87–100.

[5] Alan F Blackwell and Sam Aaron. 2015. Craft practices of live coding language design. In *Proc. First International Conference on Live Coding*. Zenodo.

[6] Jody Clarke-Midura, Frederick Poole, Katarina Pantic, Megan Hamilton, Chongning Sun, and Vicki Allan. 2018, February. How near peer mentoring affects middle school mentees. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (pp. 664-669).

[7] Karen Brennan and Mitchel Resnick. 2012. New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American Educational Research Association*, Vancouver, Canada, Vol. 1. 25.

[8] Souti Chattopadhyay, Ishita Prasad, Austin Z. Henley, Anita Sarma, and Titus Barik. 2020. What's Wrong with Computational Notebooks? Pain Points, Needs, and Design Opportunities. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pp. 1-12. 2020.

[9] Jason Freeman and Brian Magerko. 2016. Iterative composition, coding and pedagogy: A case study in live coding with EarSketch. *Journal of Music, Technology & Education* 9.1 (2016), 57-74.

[10] Jason Freeman, Brian Magerko, Doug Edwards, Tom Mcklin, Taneisha Lee, and Roxanne Moore. 2019. EarSketch: engaging broad populations in computing through music. *Commun. ACM* 62, 9 (2019), 78–85.

[11] Philip J Guo. 2013. Online Python Tutor: Embeddable Web-Based Program Visualization for CS Education. In *Proceedings of the 44th ACM Technical Symposium on Computer Science Education*. ACM, 579–584.

[12] Michael Horn, Amartya Banerjee, Melanie West, Nichole Pinkard, Amy Pratt, Jason Freeman, Brian Magerko, Tom McKlin. 2020. TunePad: Engaging learners at the intersection of music and code. Proceedings of the International Conference of the Learning Sciences.

[13] Jeremiah W Johnson. 2020. Benefits and Pitfalls of Jupyter Notebooks in the Classroom. In *Proceedings of the 21st Annual Conference on Information Technology Education*. 32–37.

[14] Yasmin B Kafai, Shiv Desai, Kylie A Peppler, Grace M Chiu, and Jesse Moya. 2008. Mentoring partnerships in a community technology centre: A constructionist approach for fostering equitable service learning. *Mentoring & Tutoring: Partnership in Learning*, 16(2), 191-205.

[15] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. 2016. *Positioning and Power in Academic Publishing: Players, Agents, and Agendas*. IOS Press, Chapter Jupyter Notebooks—a publishing format for reproducible computational workflows, 87–90.

[16] Victor Lazzarini. 2013. The development of computer music programming systems. *Journal of New Music Research* 42, 1 (2013), 97–110.

[17] Victor Lazzarini, Steven Yi, Joachim Heintz, Øyvind Brandtsegg, Iain McCurdy, *et al.* 2016. *Csound: a sound and music computing system*. Springer.

[18] Brian Magerko, Jason Freeman, Tom Mcklin, Mike Reilly, Elise Livingston, Scott Mccoid, and Andrea Crews-Brown. 2016. Earsketch: A STEAM-Based Approach for Underrepresented Populations in High School Computer Science Education. *ACM Transactions on Computing Education* (TOCE) 16, 4 (2016), 14.

[19] Tom McKlin, Brian Magerko, Taneisha Lee, Dana Wanzer, Doug Edwards, and Jason Freeman. 2018. Authenticity and personal creativity: How EarSketch affects student persistence. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. 987–992.

[20] Max MSP. A Playground for Invention. 2021. Retrieved September 2, 2021 from https://cycling74.com/products/max

[21] Seymour Papert. 1980. *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc.

[22] William Payne and S. Alex Ruthmann. 2019. Music Making in Scratch: High Floors, Low Ceilings, and Narrow Walls. *Journal of Interactive Technology and Pedagogy* 15, (2019).

[23] Pure Data. 2021. Retrieved September 2, 2021 from https://puredata.info.

[24] Mitchel Resnick and Brian Silverman. 2005. Some reflections on designing construction kits for kids. In *Proceedings of the 2005 Conference on Interaction Design and Children*. 117–122.

[25] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay S Silver, Brian Silverman, and others. 2009. Scratch: Programming for all. *Commun. ACM* 52, 11, 60–67.

[26] Ricarose Roque, Natalie Rusk, and Mitchel Resnick.2016. Supporting diverse and creative collaboration in the Scratch online community. In *Mass collaboration and education*. Springer, 241–256.

[27] David Williamson Shaffer and Mitchel Resnick. 1999. " Thick" authenticity: New media and authentic learning. *Journal of Interactive Learning Research* 10, 2 (1999), 195–216.

[28] Ben Shneiderman. 2003. *Leonardo's laptop: human needs and the new computing technologies*. MIT Press.

[29] Mary Simoni and Roger B Dannenberg. 2013. *Algorithmic Composition: A Guide to Composing Music with Nyquist*. University of Michigan Press.

[30] Heinrich Taube. 1991. Common Music: A music composition language in Common Lisp and CLOS. *Computer Music Journal* 15, 2 (1991), 21–32.

[31] Ge Wang. 2007. A history of programming and music. In *The Cambridge Companion to Electronic Music*. Cambridge University Press, Cambridge, UK, 55-71.

[32] Ge Wang, Perry R Cook, *et al.* 2003. ChucK: A concurrent, on-the-fly, audio programming language. In *Proceedings of the International Computer Music Conference (ICMC)*.

[33] Dana Wanzer, Tom McKlin, Doug Edwards, Jason Freeman, and Brian Magerko. 2019. Assessing the Attitudes Towards Computing Scale: A Survey Validation Study. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. ACM, 859–865.

[34] Uri Wilensky and Seymour Papert. 2010. Restructurations: Reformulations of knowledge disciplines through new representational forms. *Constructionism* (2010).

[35] Scott Wilson, David Cottle, and Nick Collins. 2011. *The SuperCollider Book*. The MIT Press.

[36] Xambó, Anna, Jason Freeman, Brian Magerko, and Pratik Shah. 2016. Challenges and new directions for collaborative live coding in the classroom. In *International Conference of Live Interfaces* (ICLI 2016).