MODELING MUSIC AND CODE KNOWLEDGE TO SUPPORT A CO-CREATIVE AI AGENT FOR EDUCATION

Jason Smith[1] **Erin J.K. Truesdell**[2] **Jason Freeman**[1]
Brian Magerko[2] **Kristy Elizabeth Boyer**[3] **Tom McKlin**[4]
[1] Center for Music Technology, Georgia Institute of Technology, Atlanta, GA, USA
[2] Expressive Machinery Lab, Georgia Institute of Technology, Atlanta, GA, USA
[3] Computer & Information Science & Engineering, University of Florida, Gainesville, FL, USA
[4] The Findings Group, Decatur, Georgia, USA
{jsmith775, erinjktruesdell}@gatech.edu

ABSTRACT

EarSketch is an online environment for learning introductory computing concepts through code-driven, sample-based music production. This paper details the design and implementation of a module to perform code and music analyses on projects on the EarSketch platform. This analysis module combines inputs in the form of symbolic metadata, audio feature analysis, and user code to produce comprehensive models of user projects. The module performs a detailed analysis of the abstract syntax tree of a user's code to model use of computational concepts. It uses music information retrieval (MIR) and symbolic metadata to analyze users' musical design choices. These analyses produce a model containing users' coding and musical decisions, as well as qualities of the algorithmic music created by those decisions. The models produced by this module will support future development of CAI, a Co-creative Artificial Intelligence. CAI is designed to collaborate with learners and promote increased competency and engagement with topics in the EarSketch curriculum. Our module combines code analysis and MIR to further the educational goals of CAI and EarSketch and to explore the application of multimodal analysis tools to education.

1. INTRODUCTION

Digital music creation environments often use a combination of raw audio data, symbolic information, code, and metadata to represent user-generated music. For example, a digital audio workstation might represent a song through a combination of audio files, genre and artist labels, MIDI events, and a data source indicating the placement of audio and MIDI segments on a multi-track timeline, along with device and mixer settings and automation.

In the context of music information retrieval (MIR), access to these multiple types of music representation can vastly simplify common retrieval tasks that are too complex to perform on audio alone. Examples of this include music preference modeling using a combined model of audio and metadata [1] and genre recognition using feature analysis alongside symbolic representation of the same audio through statistical descriptors of melody, harmony, and rhythm [2, 3]. A multimodal analytical approach can therefore help develop powerful MIR-driven applications within these digital music creation environments, such as creativity-support tools that generate ideas and feedback for users as they create music with the software.

Digital environments that include algorithmic composition elements offer yet another mode of analysis: the code that generates the music. A creativity support tool that incorporates code analysis in addition to these other modalities can potentially generate recommendations not only about the music users create but also about the code that they write and and the conceptual overlap thereof between the code and music.

This paper describes an analysis system we have created that uses audio features, audio metadata, symbolic multi-track music data, and code from user-created projects to understand the structure of algorithmic music and to model users' knowledge of coding and musical techniques. We have created the system in the context of EarSketch [4], an expressive and collaborative learning environment for high school students that is used by roughly 120,000 students per year. In EarSketch, users write Python or JavaScript code to algorithmically create multi-track compositions remixing a library of audio loops. Figure 1 depicts the EarSketch user interface. EarSketch aims to increase student engagement in computing across diverse student populations, promoting student perceptions of authenticity through its use of a professionally-produced audio loop library and design influences from industry-standard digital audio workstations [5].

The EarSketch curriculum contains musical concepts integral to algorithmic music and relevant to coding, including repetition, form, and effect usage. The EarSketch sound library contains over 3,500 sounds from professional

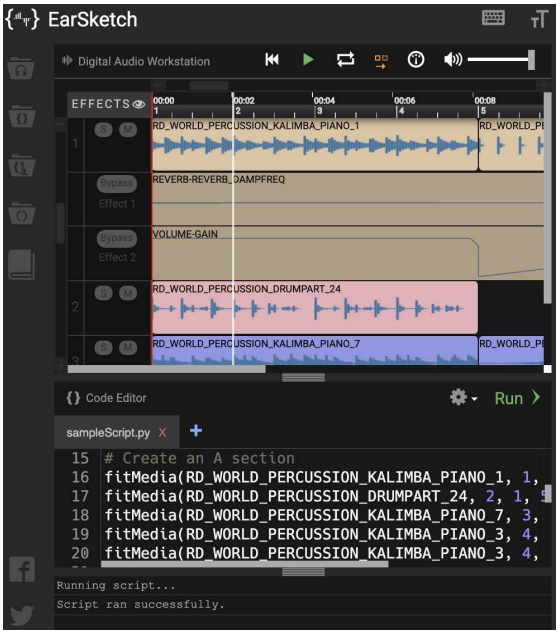

Figure 1. A Screenshot of the EarSketch web-based application, containing the Digital Audio Workstation (top), and Code Editor (bottom).

artists, split into folders and labeled with artist, genre, and instrument type. Users write code with Python and JavaScript APIs to manipulate sounds and create music.

EarSketch is designed for students without formal music training. It does not focus on traditional music theory concepts such as music notation, melody, and harmony. Instead, it uses audio loops, effects, automation, and step-sequenced rhythms to facilitate music production through code without prerequisite knowledge. In addition to modeling understanding of code, our module includes music analysis to compare a user project's traits to the information conveyed by the EarSketch curriculum.

Our analysis system was designed to support a new creativity-support tool within EarSketch called CAI (Co-creative Artificial Intelligence). CAI, which is still in the early stages of design, will assist EarSketch users in learning and practicing pedagogical concepts in both computing and music. CAI will use our multimodal analysis system in combination with user interaction to suggest additions and changes to student music and code, scaffolding student learning and producing co-creative musical output.

In the following sections, we position this work in the context of recent music information retrieval research, describe each component of the analytical system in turn, explain how we coalesce this data into a user model to inform CAI, and outline future areas of work in the design and implementation of the larger CAI system that will leverage this analysis tool.

The development of this analysis module for CAI marks a unique application of MIR and code analysis to educational systems. Further development of the CAI system will continue to illuminate the role of mixed-input models for the goal of supporting learners in expressive computing environments.

2. RELATED WORK

Multimodal music information retrieval relies on representations of information to perform analysis tasks beyond what is possible using raw audio signals [6]. Digital production environments and notation software use symbolic information to represent structural and artistic elements of user-generated music. Due to its prominent use in music software, symbolic music is a significant presence in the domain of music information retrieval, with well-known tools such as Music21 [7] performing operations on symbolic data. MIDI has also been used to train style transfer models [8] and detect meter in live performance [9].

MIDI note messages are direct symbolic representations of musical notes. Other combinations of symbolic and audio analysis performed by labeling audio with statistical descriptors have been used in applications of genre recommendation [2] and pattern-based style identification [3]. Another application [10] uses symbolic music representation to vectorize multiple aspects of music for the purpose of performing song recommendations. The LFM-1b dataset [11] contains recorded listening events tagged with metadata, and has been used in conjunction with audio feature analysis to model user music preference [1]. Our system differs from these applications by using audio and symbolic data to model a user's proficiency with the techniques required to produce musical output, using EarSketch curriculum topics as evaluation criteria.

Our system extends traditional music information retrieval applications in its use of source code analysis as an input. Code analysis tools often rely on the analysis of abstract syntax trees (ASTs) generated from student code to propose edits. Contemporary AST-based systems [12, 13] rely on step calculations between a student's current code-state and a previously seen code-state or family of code-states that fulfills a set of conditions. These systems use AST analysis to simplify a wide range of projects to their most basic structure and provide suggestions in almost all situations. This application has largely focused on moving students from an "incorrect" answer to a "correct" one. In comparison, the application of AST analysis to creative problems without a defined "solution" is relatively unexplored. Our code analysis module brings the advantages of AST analysis into the expressive domain, reorienting the process towards open-ended learning and development.

Examples of software analysis applied to music information retrieval include a tool that uses Source Code Analysis and Manipulation (SCAM) to represent the structure of algorithmic music compositions [14, 15]. Music code analysis has also been performed in the context of live coding, a performance format in which programmers write algorithmic music in real time. A survey of live coding practitioners on creativity [16] discusses the link between analyzing live coding techniques and the development of a creative software agent. EarSketch has previously been examined for its ability to function as an educational live coding platform [17], and a fully developed CAI system will be able to collaborate with learners in a live coding environment.

3. CODE ANALYSIS

3.1 Code Knowledge Modeling

The code portion of the analysis module combines AST analysis and a computer science learning taxonomy to build a model of user knowledge that will be used by CAI to generate level-appropriate suggestions for EarSketch users. Previous works on computer science assessment [18–20] have adapted general learning taxonomies for computing topics; similarly, we defined a series of knowledge levels for 15 computational concepts from the EarSketch curriculum across 4 knowledge categories (see Table 1) based upon a flattened version of Bloom's Taxonomy [21]. For each concept, we define knowledge levels specific to its usage contexts. Table 2 outlines knowledge levels defined for the "String" and "User-Defined Function" concepts. Level 1 refers to usage of the concept or construct (e.g., a user includes a string in their script). Level 2 is defined as "original" usage of the concept (usage not copied directly from sample code). Our originality measures are described in greater detail in Section 3.2. Subsequent levels focus on increased complexity of use: for example, a user's script could reach level 2 of the "String" concept by merely including an original string in their script, but level 3 requires that the string to be put to use (such as using it as a function argument).

Category	Concepts
Value Types	String, Integer, Float, Boolean
Data Storage	List, Variable
Operations	String Operation, List Operation, Comparison, Boolean Logic, Mathematical Operator
Procedure	For Loop, Conditional Statement, User-defined Function, Console Input

Table 1. Concepts in the analysis module taxonomy.

Level	String	User-Defined Function
0	Does Not Use	Does Not Use
1	Uses	Uses
2	Uses Originally	Uses Originally
3	Uses Originally for Purpose	Uses and Calls Originally
4	Uses Originally and Indexes or Iterates for Purpose	Uses and Calls Originally with Return OR Arguments
5	N/A	Uses and Calls Originally with Return AND Arguments

Table 2. Knowledge levels for two concepts: "String" and "User-Defined Function."

3.2 Code Complexity Analysis

The code analysis module generates knowledge models for student scripts, producing concise information on understanding of each topic. Code knowledge models are generated by analyzing the abstract syntax tree of a user's code, which allows for fast and non-intrusive analysis. The code analysis module searches each AST node for constructs that indicate the user's knowledge level for every concept in our taxonomy.

Prior to analysis, the module performs a series of four passes over the script's AST to gather supporting data. The first pass tests student code for similarity to sample code. We use Andrei Mackenzie's Levenshtein function [1] to calculate the edit distance between each line of a user's code and all lines of EarSketch sample code; if the edit distance is below a manually-defined similarity threshold, the line is marked as "not original." Three subsequent passes gather information about user-defined functions and variable assignments and values. Once this information is collected, each individual node in the hierarchy is checked against discrete rules developed in tandem with the knowledge modeling level table.

Below is an example of a student script progressing through levels of the "user-defined function" concept. In the first code snippet, the user creates and then calls a function to make a section of music, calling `fitMedia()` to place piano and drumpad samples between measures 1 and 16. This corresponds to level 3 of the "user-defined function" item: "Uses and Calls Originally."

```
def sectionA():
  fitMedia(RD_RNB_PIANO_1,1,1,16)
  fitMedia(Y25_DRUMPAD_1,2,1,16)

sectionA()
```

In the second snippet, the function has been modified to take arguments: the function now places the samples between passed "start" and "end" measures. This corresponds to level 4 of the "user-defined function" item: "Uses and Calls Originally with Return OR Arguments."

```
def sectionA(start, end):
  fitMedia(RD_RNB_PIANO_1,1,start,end)
  fitMedia(Y25_DRUMPAD_1,2,start,end)

sectionA(1,16)
```

These node analyses populate an output object with values mapped to knowledge levels, visualized in Table 3. These analyses will be used to support future development of CAI, providing guidance to the system about appropriate code structure for programming and music suggestions when collaborating with a student.

[1] https://gist.github.com/andrei-m/982927

Concept	0	1	2	3	4	5
String	■	■	■	■		
Integer	■	■	■	■		
Float	■	■	■	■		
Boolean	■	■	■	■		
List	■	■	■	■	■	
Variable	■	■	■	■	■	
String Op	■					
List Op	■					
Comparison	■					
Boolean Logic	■					
Math Op	■	■	■	■		
For Loop	■	■	■			
Conditional	■	■				
User Function	■	■	■	■	■	
Console Input	■					

Table 3. Visualization of code analysis output for a sample project.

4. SYMBOLIC MUSIC ANALYSIS

4.1 EarSketch Music Representation

The music component of the analysis module uses a simplified symbolic representation generated by EarSketch to apply sounds and effects to the Digital Audio Workstation view and generate audio playback. When a learner runs a script, the parsed abstract syntax tree of the code is represented internally as a dictionary of tracks containing sound and effect usage for each measure in a piece of music.

```
#Setup
from earsketch import *
init()
setTempo(120)

# Create an A section
fitMedia(RD_WORLD_PERCUSSION_KALIMBA_PIANO_1,1,1,5) # main
fitMedia(RD_WORLD_PERCUSSION_DRUMPART_24,2,1,5) # drums
fitMedia(RD_WORLD_PERCUSSION_KALIMBA_PIANO_7,3,1,5) # bassline
fitMedia(RD_WORLD_PERCUSSION_KALIMBA_PIANO_3,4,1,2) # backing
fitMedia(RD_WORLD_PERCUSSION_KALIMBA_PIANO_3,4,3,4) # backing

# Create a 4 measure B section between measures 5 and 9
fitMedia(RD_WORLD_PERCUSSION_DRUMPART_3,1,5,9) # sparse drums
fitMedia(RD_WORLD_PERCUSSION_SEEDSRATTLE_1,3,5,9) # rattling
fitMedia(RD_WORLD_PERCUSSION_KALIMBA_PIANO_3,4,5,6) # backing

# Then back to section A at measure 9
fitMedia(RD_WORLD_PERCUSSION_KALIMBA_PIANO_1,1,9,13) # main
fitMedia(RD_WORLD_PERCUSSION_DRUMPART_24,2,9,13) # drums
fitMedia(RD_WORLD_PERCUSSION_KALIMBA_PIANO_7,3,9,13) # bassline
fitMedia(RD_WORLD_PERCUSSION_KALIMBA_PIANO_3,4,9,10)  # backing
fitMedia(RD_WORLD_PERCUSSION_KALIMBA_PIANO_3,4,11,12) # backing

# Effects
setEffect(1,REVERB)
setEffect(1,VOLUME,GAIN,0,4.9,-12,5)
setEffect(1,VOLUME,GAIN,-60,5,0,9)

#Finish
finish()
```

Figure 2. Code in the Code Editor of an example project created in EarSketch.

The track listing for the example project (Figure 2) is an array of tracks, each containing a series of *clip* and *effect* objects reflecting those used in the code. Each clip object contains the name of the sound file used (*RD_WORLD_PERCUSSION_KALIMBA_PIANO_1*), as well as its start measure (*1*), and end measure (*5*). Each effect object contains each instance of a specific effect (*VOLUME-GAIN*), its starting value and measure (*-60, 5*) and ending value/measure (*0, 9*).

The music analysis tool begins by converting this track representation to a timeline representation, in order to ascertain temporal patterns in the song. The timeline is created by converting the track dictionary to a dictionary of measures, as measure numbers are used in the EarSketch API for audio and effect sequencing. Figure 3 shows this timeline for four measures of the example, which has sounds used throughout the song such as *RD_WORLD_PERCUSSION_KALIMBA_PIANO_1* and sounds used in a single section such as *RD_WORLD_PERCUSSION_SEEDSRATTLE_1*, as well as the value of the gain adjustment at each measure.

```
0: ["RD_WORLD_PERCUSSION_KALIMBA_PIANO_1",
    "VOLUME GAIN 1",
    "REVERB REVERB_DAMPFREQ 8000",
    "RD_WORLD_PERCUSSION_DRUMPART_24",
    "RD_WORLD_PERCUSSION_KALIMBA_PIANO_7",
    "RD_WORLD_PERCUSSION_KALIMBA_PIANO_3"]

4: ["RD_WORLD_PERCUSSION_DRUMPART_3",
    "VOLUME GAIN 0",
    "REVERB REVERB_DAMPFREQ 8000",
    "RD_WORLD_PERCUSSION_SEEDSRATTLE_1",
    "RD_WORLD_PERCUSSION_KALIMBA_PIANO_3"]

6: ["RD_WORLD_PERCUSSION_DRUMPART_3",
    "VOLUME GAIN 0.4995",
    "REVERB REVERB_DAMPFREQ 8000",
    "RD_WORLD_PERCUSSION_SEEDSRATTLE_1"]

8: ["RD_WORLD_PERCUSSION_KALIMBA_PIANO_1",
    "VOLUME GAIN 0.999",
    "REVERB REVERB_DAMPFREQ 8000",
    "RD_WORLD_PERCUSSION_DRUMPART_24",
    "RD_WORLD_PERCUSSION_KALIMBA_PIANO_7",
    "RD_WORLD_PERCUSSION_KALIMBA_PIANO_3"]
```

Figure 3. Timeline representation for measures 1, 5, 7, and 9 of the sample EarSketch project (see Figure 2).

Recognition of patterns in sound and effect usage over time can be used in determining form as described in section 5. It allows CAI to propose changes to sound choices, effects, and parameters at specific points in time, or to suggest optimized code structure to realize those patterns. For example, if a user places the same sound at regular intervals and the code analysis does not observe any loops containing the variables related to that sound in their code, CAI can suggest the use of a loop.

Audio usage requires students only to select sounds from the library; conversely, effects can be manipulated through envelopes and are introduced in the EarSketch curriculum in multiple levels of complexity. The analysis module conducts a hierarchical score analysis for effect usage, while simply recording the selection of audio at each measure to inform its audio analysis tools.

4.2 Effect Usage Scores

Effect envelopes in EarSketch are applied using a start value, end value, start time, and end time. The following code example (found in Figure 2) shows a use of the volume effect on track 1 having its gain parameter changed from -60 dB to 0 dB between measures 5 and 9:

```
setEffect(1, VOLUME, GAIN, -60, 5, 0, 9)
```

The internal track representation stores a single recording of these four values for each effect parameter. Consequently, we use linear interpolation to store the value of each effect parameter at each measure when converting effect envelopes to the timeline.

This reorganization allows the analysis module to classify the level of usage the user demonstrates for each audio effect in the library, such as gain, filters, delay, and reverb. The EarSketch curriculum teaches basic effect usage, followed by use of effect parameters, and then the use of envelopes to change parameter values over time. Similarly, the levels of effect usage are *0: Does not use*, *1: Uses standard parameters*, *2: Uses non-standard parameters*, and *3: Changes parameters over time*. EarSketch's goal is to teach coding technique through music production, so students are encouraged to use increasingly complex effect parameter manipulation through increasingly complex algorithmic structure. These scores, along with the scores from the code analysis described in section 3, form a composite score to represent user knowledge.

In addition to the timeline representation and modeling of effects usage, the analysis module records the length of the piece and whether or not the user sets a tempo different than the default 120 bpm. The combination of these music analysis outputs can be used by CAI to characterize a user project and to identify which sounds or effects to include and where to include them, or areas for further improvement in a script.

5. AUDIO FEATURE ANALYSIS

In addition to storing user knowledge modeling, the analysis module is designed to analyze the sound usage, form, and genre of a user project. These are not represented in ordered levels to model conceptual understanding like effects are, but are recorded to display a detailed overview of the piece of music and to better target suggestions made by the CAI system. For example, if a user is primarily choosing sounds of a certain genre in a specific musical section, CAI might suggest other sounds in that genre for that section, while suggesting contrasting sounds for a different section.

5.1 Audio Recommendations

The analysis module includes an expansion of the existing EarSketch recommendation system [22], which uses the code of an active user project to make real-time recommendations. The recommendation system uses audio features Short-time Fourier Transform (STFT) and Mel-Frequency Cepstral Coefficients (MFCC) [23] to represent temporal and non-temporal information, respectively, of each EarSketch library sound whose name is found in the code [24]. Feature vectors for each sound are generated in offline scripts, and feature distances and relative co-usage by EarSketch users between each pair of sound vectors are uploaded to the EarSketch server. This avoids the need to calculate the features, feature distances, and co-usage data in real time. The advantage of this form of analysis in a digital production system such as EarSketch is that audio feature distances can be measured against metadata tags for artist, genre, and instrument type. The system is able to provide recommendations that either specifically conform to the user project's genre and instrumentation or allow the user to consciously explore other creative options.

The original EarSketch recommendation system [22], which used the source code of a script to generate recommendations for an entire user project. The analysis module expands this system by presenting a series of sound recommendations per measure. These recommendations can be used to increase or decrease musical contrast at specific points throughout a composition, or use stored feature distances to ascertain the best measure to include a sound.

5.2 Form and Structure Analysis

The use of symbolic music representation greatly simplifies the task of determining sections in a song through changes in instrumentation and effects. Sound and effect usage are represented for each measure, and instrument type and genre are included in metadata tags.

Self-similarity can be used in retrieving [25] and visualizing [26] musical structure. An array of self-similarity values for instrumentation at every measure in the timeline view allows the analysis module to infer musical structure from sounds used by marking the first measure in a sequence to pass above or below a threshold of similarity.

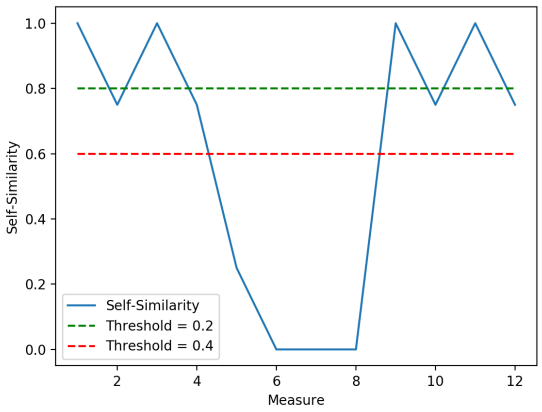

Figure 4. Self-similarity of instrumentation for the example EarSketch project (see Figure 2), indicating an A-B-A form. Dotted lines represent similarity thresholds of 0.4 and 0.2 that, when crossed, mark section beginnings.

The threshold of similarity can also be hierarchically defined to determine different granularities of sections and subsections, from a whole piece to the measure level. The example in Figure 4 shows ternary form, a musical structure represented in the EarSketch curriculum. If the similarity threshold is above 0.25, then the analysis module will cross the threshold three times and predict section beginnings at measures 1, 5, and 9. If the threshold is lower, then it will be crossed five times and predict section beginnings at measures 3 and 11 as well. When analyzing a script, the analysis module generates a list of section pre-

dictions for thresholds of every ten percent between 0.9 and 0.1. Any list of a unique length (or with unique values) is recorded. In the example of Figure 4, lists would be generated to form a section prediction of *[1, 5, 9]* with a subsection prediction of *[1, 3, 5, 9, 11]*.

5.3 Genre Analysis

Each sound in the EarSketch sound library is labeled with a tag for one of 21 genres such as Rock, Funk, Hip Hop, and EDM. These tags were manually added to the sounds by the artists and EarSketch developers who uploaded them.

Our music analyzer combines these genre tags with audio features (as described in section 5.1) to predict a genre for each measure in a user project. It applies the k-means genre clustering algorithm [27] to each measure, approximating the closest distance in audio fingerprints between the average for the measure's sounds and the average for a genre found in the EarSketch library. The genre tag for each sound used in a measure is also added to increase the confidence value for that genre. This combination affords our system the knowledge of the original genre label for each sound as presented to the user, but allows it to recognize a user who has creatively used sounds in genre applications separate from their label. This genre analysis is used to determine the likelihood of a user project belonging to a specific genre at the script, section, subsection, or measure level - depending on the granularity of the self-similarity measurement used to determine form.

This genre analysis can be used for CAI to target its audio suggestions to a song's identified genre. If a user is writing a song mainly using sounds tagged with a single genre, such as in the example code (Figure 2), CAI can suggest sounds in that genre or present options for different genres to generate contrast.

6. PRELIMINARY RESULTS, USAGE, AND INSIGHTS

Two informal testing methods have been used to aid in the iterative development of the code analysis module: a review of output correctness, and an ongoing large-scale process to identify bugs in the module.

To evaluate the ability of the module to produce a correct analysis of a script, we ran it on a series of 103 student- and researcher-generated scripts (77 Python, 26 JavaScript). Student scripts were selected to ensure the module could accurately respond to programming choices made by EarSketch users; researcher scripts were selected or written to test the module's ability to respond to complex scenarios. For each test script, the analysis module's output was judged against a researcher-generated score. Any discrepancies between the two indicated a missing component in the module. Modifications were made until the analysis module could correctly score the script. This testing identified a number of situations not originally accounted for in the module.

To efficiently locate bugs in the code analysis script, we have included a version of the Code Analysis module on EarSketch that runs each time any user runs a script. Upon the encounter of an exception while analyzing a project, the module sends a report including the exception and stack trace to an analytics engine. These reports are frequently reviewed to identify bugs and improve the ability of the code analysis module to run without error.

The music analysis tool has been integrated into the EarSketch autograder, an automatic grading tool used in evaluating the complexity of code submissions in previous course projects [4]. This autograder is a separate web page that, given any number of EarSketch script sharing IDs, generates a list of code topics, their categorized scores, the list of section markings, measure-by-measure audio recommendations, and genre predictions. This analysis can be performed on large samples of scripts, such as on multiple instances of a script to track improvement over its history, and is used in ongoing evaluation of the analysis module's ability to model form and genre. As the analysis module undergoes future iterative development, observations of emergent patterns in output will further aid the development and evaluation of the CAI recommendation system as well as its component parts.

Though the CAI system that will make use of this analysis module has yet to be completed, the development of this module has generated insights about the applications of MIR work in tandem with other disciplines, particularly in the context of education. Its combined analysis of symbolic music, audio features, and code knowledge highlights the ability of multimodal analysis to provide a comprehensive body of information to support [systems that do two things]. The use of this combination in an educational context will allow CAI to assist learners in simultaneous musical and programming development and further the goals of educational platforms that intersect domains. Additionally, the development of this module for a co-creative AI indicates potential for additional knowledge to be developed as the module is used to inform the agent's outputs.

7. FUTURE WORK

This analysis module has been implemented in the production version of EarSketch. We are still in the early stages of designing CAI, the agent that will leverage this analysis data to interact with students through dialogue and generate suggestions for their music and code. Over the past year, we have performed studies to better understand how students interact through chat with each other in student-to-student chat experiments. We have also conducted chat experiments between students and researchers posing as AI agents to simulate the intelligence that CAI will eventually provide. We are using the findings from these studies to inform the design of CAI. The initial version of CAI will exist as a chat-style interface within EarSketch, where students will choose prompts from a menu-driven system and receive natural language responses based on their interaction with the system and its analysis of their code and music as described in this paper. Through the addition of CAI to EarSketch, we hope to further increase student engagement and creativity with the platform.

8. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation Award No. 1814083. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. EarSketch is available online at *https://earsketch.gatech.edu.*

9. REFERENCES

[1] E. Zangerle and M. Pichl, "The many faces of users: Modeling musical preference." in *Proc. of ISMIR*, 2018, pp. 709–716.

[2] T. Lidy, A. Rauber, A. Pertusa, and J. M. I. Quereda, "Improving genre classification by combination of audio and symbolic descriptors using a transcription systems." in *Proc. of ISMIR*, 2007, pp. 61–66.

[3] P. J. P. De León and J. M. Inesta, "Pattern recognition approach for music style identification using shallow statistical descriptors," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 37, no. 2, pp. 248–257, 2007.

[4] B. Magerko, J. Freeman, T. Mcklin, M. Reilly, E. Livingston, S. Mccoid, and A. Crews-Brown, "Earsketch: A STEAM-based approach for underrepresented populations in high school computer science education," *ACM Transactions on Computing Education (TOCE)*, vol. 16, no. 4, pp. 1–25, 2016.

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

[6] C. C. Liem, M. Müller, D. Eck, G. Tzanetakis, and A. Hanjalic, "The need for music information retrieval with user-centered and multimodal strategies," in *Proc. of the 1st International ACM Workshop on Music information Retrieval With User-Centered and Multimodal Strategies*, 2011, pp. 1–6.

[7] M. S. Cuthbert and C. Ariza, "music21: A toolkit for computer-aided musicology and symbolic music data," in *Proc. of ISMIR*, 2010, pp. 637–642.

[8] G. Brunner, A. Konrad, Y. Wang, and R. Wattenhofer, "Midi-vae: Modeling dynamics and instrumentation of music with applications to style transfer," in *Proc. of ISMIR*, 2018, pp. 747–754.

[9] A. McLeod and M. Steedman, "Meter detection and alignment of midi performance." in *Proc. of ISMIR*, 2018, pp. 113–119.

[10] K. Watanabe and M. Goto, "Query-by-blending: a music exploration system blending latent vector representations of lyric word, song audio, and artist," in *Proc. of ISMIR*, 2019, pp. 144–151.

[11] M. Schedl, "The LFM-1b dataset for music retrieval and recommendation," in *Proc. of the 2016 ACM International Conference on Multimedia Retrieval*, 2016, pp. 103–110.

[12] T. W. Price, Y. Dong, and D. Lipovac, "iSnap: towards intelligent tutoring in novice programming environments," in *Proc. of the 2017 ACM SIGCSE Technical Symp. on Computer Science Education*, 2017, pp. 483–488.

[13] K. Rivers, E. Harpstead, and K. R. Koedinger, "Learning curve analysis for programming: Which concepts do students struggle with?" in *Proc. of the 2016 ACM Conference on International Computing Education Research*, 2016, pp. 143–151.

[14] N. Gold, "Knitting music and programming: Reflections on the frontiers of source code analysis," in *2011 IEEE 11th International Working Conference on Source Code Analysis and Manipulation*, 2011, pp. 10–14.

[15] M. Harman, M. Munro, L. Hu, and X. Zhang, "Source code analysis and manipulation," *Information and Software Technology*, vol. 44, no. 13, pp. 717–720, 2002.

[16] A. McLean and G. A. Wiggins, "Live coding towards computational creativity." in *International Conference on Computational Creativity*, 2010, pp. 175–179.

[17] J. Freeman and B. Magerko, "Iterative composition, coding and pedagogy: A case study in live coding with earsketch," *Journal of Music, Technology & Education*, vol. 9, no. 1, pp. 57–74, 2016.

[18] U. Fuller, C. G. Johnson, T. Ahoniemi, D. Cukierman, I. Hernán-Losada, J. Jackova, E. Lahtinen, T. L. Lewis, D. M. Thompson, C. Riedesel *et al.*, "Developing a computer science-specific learning taxonomy," *ACM SIGCSE Bulletin*, vol. 39, no. 4, pp. 152–170, 2007.

[19] C. W. Starr, B. Manaris, and R. H. Stalvey, "Bloom's taxonomy revisited: specifying assessable learning objectives in computer science," *ACM SIGCSE Bulletin*, vol. 40, no. 1, pp. 261–265, 2008.

[20] E. Thompson, A. Luxton-Reilly, J. L. Whalley, M. Hu, and P. Robbins, "Bloom's taxonomy for cs assessment," in *Proc. of the tenth conference on Australasian computing education-Volume 78*, 2008, pp. 155–161.

[21] D. R. Krathwohl and L. W. Anderson, *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives*. Longman, 2009.

[22] J. Smith, M. Jacob, J. Freeman, B. Magerko, and T. Mcklin, "Combining collaborative and content filtering in a recommendation system for a web-based daw," in *Proc. of the International Web Audio Conference*, 2019.

[23] A. Lerch, *An Introduction to Audio Content Analysis: Applications in Signal Processing and Music Informatics*, 1st ed. Wiley-IEEE Press, 2012.

[24] J. Smith, D. Weeks, M. Jacob, J. Freeman, and B. Magerko, "Towards a hybrid recommendation system for a sound library," in *ACM IUI Workshops*, 9.

[25] B. Martin, M. Robine, and P. Hanna, "Musical structure retrieval by aligning self-similarity matrices." in *Proc. of ISMIR*, 2009, pp. 483–488.

[26] J. Foote, "Visualizing music and audio using self-similarity," in *Proc. of the seventh ACM International Conference on Multimedia (Part 1)*, 1999, pp. 77–80.

[27] D. Turnbull and C. Elkan, "Fast recognition of musical genres using RBF networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 4, pp. 580–584, 2005.