TuneScope: Synthesizing Music and Computational Thinking

Harsh Padhye
University of Virginia
Computer Science and Mathematics, B.A.
Distinguished Majors Thesis in Computer Science

April 20, 2023

Contents

1 Abstract

The Make-To-Learn Lab at the University of Virginia developed TuneScope, a tool to facilitate computational thinking (CT) through music, to help democratize computer science education. Built upon the existing framework of Snap! (University of California, Berkeley), TuneScope leverages sound analysis, design, and music composition to engage novices with CT fundamentals. Existing research shows the benefits of using familiar contexts to teach CT, and TuneScope builds upon this with musical contexts. We have designed a course centered around the use of TuneScope to teach fundamentals of CT through music. In this paper, we investigate how students use TuneScope to develop sequential melodies, build chords, and sample recorded sounds while simultaneously learning fundamental programming principles such as algorithm design and abstraction. We assess and analyze how well introducing computing principles via music composition improves the retention of CT skills.

2 Background

Before discussing the development and implementation of TuneScope, I will explain the requisite terminology used in the project. TuneScope is a low-code library based on the block based programming language Snap! by University of California, Berkeley. Students do not have to write with traditional programming syntax; instead they drag, drop, and connect code blocks on their workspace in order to create scripts and code snippets. Their code directly manipulates a sprite on its interactive stage, allowing the user to visualize and audialize their code snippets in real time. The library accompanies a curriculum designed at the University of Virginia by the Make to Learn Lab. Students learn computer science fundamentals by following art and music modules developed alongside TuneScope. Throughout the course, students are evaluated on their code with four main areas of focus: abstraction, algorithms, data representation, and documentation. Students are encouraged to publish their work on the official Snap! forum, where they can receive feedback from the developers of the Snap! programming language.

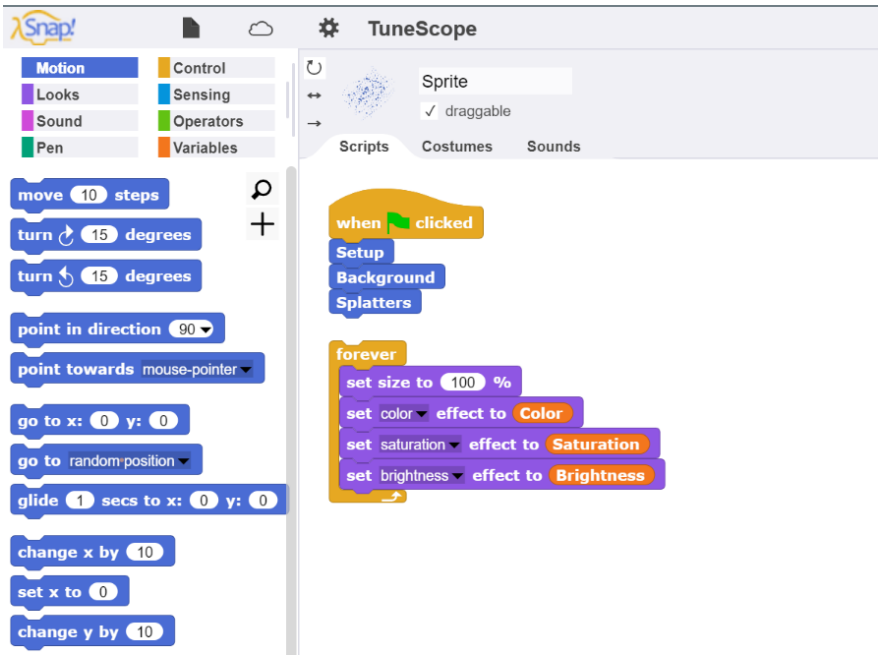

Figure 1: Abstracted code which creates a Jackson Pollock style painting.

The figure above 1 illustrates the user interface of the Snap! workspace. The palette of blocks to the left are separated by method type and function; for instance, blue **Motion** blocks categorize all blocks which move the user's sprite on the stage, while yellow **Control** blocks categorize control structures (if-statements, for-loops, etc.). Users can create their own custom blocks to abstract their code, as seen with the **Setup**, **Background**, and **Splatters** blocks stacked sequentially on the workspace. This code moves the sprite randomly across the stage and paints colorful paint splatters to resemble artwork by Jackson Pollock. Running the code above produces the figure below.

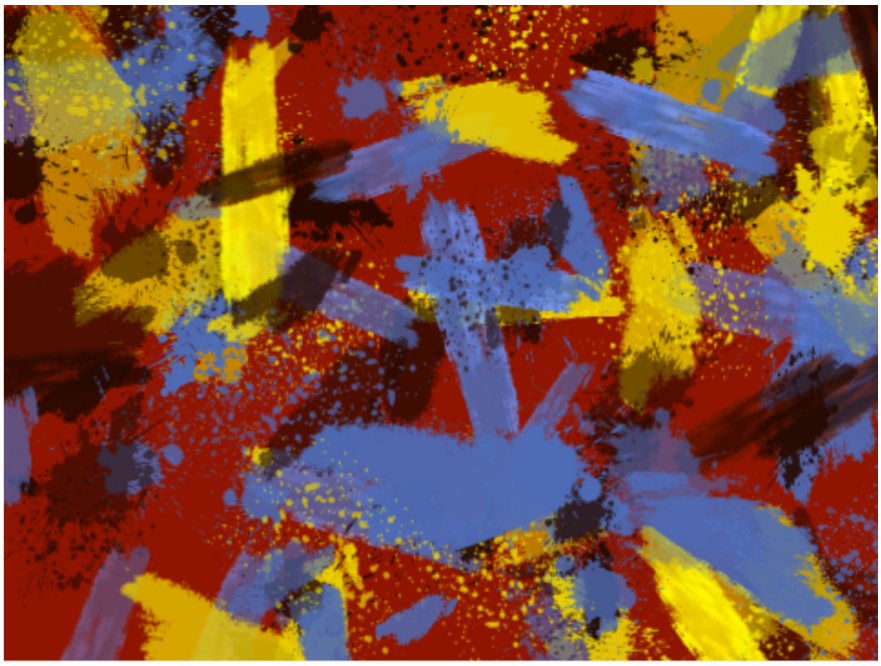

Figure 2: A rendered image which resembles a Jackson Pollock painting.

3 Related Work

3.1 Music as an Educational Tool

My first experience with sound synthesis came in the form of Digital Audio Workstations (DAWs) and synthesizers. DAWs offer a plethora of options to the producer beyond simply playing an instrument. Users can group tracks, automate effects plugins, and even implement boolean logic in their rendering. Creating synthesizers in online simulation software (VCV rack) offered a similar modular experience, but this time with more emphasis on mathematics and signal processing. My prior background in computer science (having already coded for over four years by this point) encouraged me to learn these tools analytically. For a while I was enamored with Euclidean sequencers and conditional boolean gates, eventually submitting multiple assignments with signal flow chains that better modeled backend code than sheet music. While my musical background certainly helped me grasp difficult concepts, it was my computational thinking framework that led to interesting compositions and synthesized sounds.

Could we flip that methodology and encourage computer science education through music? I had always been interested in combining my two passions, but it wasn't until joining the Make to Learn Lab that I realized its potential in democratizing computing fields. In this section, I will discuss previous efforts to increase diversity in computer science education and synthesize music with computational thinking. This will include the use of music as a pedagogical tool in other subjects, the availability of music and computing resources in underprivileged schools, and existing platforms

for code-driven music composition.

Previous studies by music psychologists have shown a link between engagement in musical activities and improved performance in intellect and spatial-temporal reasoning. Spatial-temporal reasoning, as stated by Holmes and Hallam, is "the ability to transform mental images in the absence of a physical model... [allowing] individuals to create mental imagery, develop patterns that can change in space and time, and visualize problems and potential solutions"[16]. In their study, researchers assessed children's spatial-temporal reasoning via a series of pattern and puzzle tests which required the participants to complete a series of puzzles without being given instructions on how to do so. Essentially, the children had to visualize the end product and problem-solve a way to reach the end goal. Results showed a statistically significant increase (favoring the kids in music groups) in scores for the puzzle test at the end of each academic year. A notable facet of this study was the design of the music curriculum used, which was intentionally created to be simple and easy for primary school teachers who are not music specialists to teach and understand the content. Spatial-temporal skills are an important part of computational thinking, as a foundational aspect of computer science is identifying and casing a problem, designing a step-by-step solution, and recognizing patterns to streamline code design [18]. Since the results of Holmes and Hallam's study were most significant with their younger participants, I was conscious of user-friendly and accessible design when devising a method of programmatic music composition. The implementation of accessibility and user design will be discussed later in the paper.

3.2 Inequities in Computer Science Education

A significant concern in computational pedagogy is the lack of resources in offering a computer science education at an early age, with many institutions lacking a program entirely. For instance, while Chicago public school districts mandate computer science classes as a graduation requirement, some sectors of the city do not have the funds to hire qualified teachers; this often results in computing-adjacent teachers (mathematics and other sciences) being forced into the role with little to no programming experience [14].

The need for computing classes is clear. One case study indicates that schools that offer AP Computer Science classes see twice as many female and underrepresented minority students taking AP Computer Science courses and exams from previous years. However, the same cannot be said for less funded schools in the districts. These schools cite barriers such as a lack of updated computer labs, lack of prepared faculty, inadequate student and teacher schedules, and the fact that CS is treated as an elective rather than a part of the core curriculum [17, 9].

Gleasman et al. proposed and implemented a computer science enrichment program called *Girls Creating Opportunities to Develop and Empower Rural Success* (G-CODERS) [8] as a means to reduce the gender disparities in computer science in rural middle schools. They suggest that computational thinking education begins as early as elementary school to combat many young women's culturally imposed biases against computer science. Their research suggests integrating computational thinking, coding structures, and design thinking into one curriculum leads to better retention and engages more female STEM students to pursue computer science as a career.

Mickelson et al. explore reasons why certain ethnic and gender groups are underrepresented in the field, beyond the lack of pedagogical resources. Analyzing demographics at the University of North Carolina at Charlotte, the researchers show that Black, Latino, and Female students are heavily underrepresented in technology fields compared to national and state census numbers. A stark example of this is the fact that "roughly 46% of undergraduates at UNC Charlotte campuses are women, yet only 16.6% of UNC Charlotte computer science majors are women"[12]. Disparities in gender and minority representation are more egregious in computer science than most other STEM fields. Part of what

encourages students to take and continue taking computer science classes is the identity of being a "computer science major". This is directly correlated, as previous literature indicates, with the belief that one can not only "do science", but "be a scientist"[12]. Computer science classes in high school and college too often lack the individually tailored teaching required to encourage increased participation in under-represented groups. Courses that should be inclusive and draw from students' prior experiences are instead taught in "one size fits all" style lectures–only exacerbating the demographic disparity in the field.

What inspired me most about this study was the prevailing idea of identity in computing education. Reflecting on my own experiences as a computer science and mathematics student, I can confidently say I perceive myself as fitting into the "model" of a computer scientist: a person who is methodical, logic-driven, and enjoys finding a singular correct answer. My creative drive in music developed entirely in parallel to my academic desires and only converged after I had determined I would become a computer scientist by profession. This narrow view of what constitutes a computer scientist is partially what Mickelson et al. suggest is preventing female and under-represented minorities from joining the field [12]. What if we broaden the identity of a computer scientist and demonstrate that computational thinking is as much creative and free-form as it is analytical and programmatic? When considering the user experience during the development of TuneScope, part of my mission was to give the user creative agency in music making. While the code blocks themselves had to fit into a computational framework, the user should be able to manipulate them in artistic ways. I wanted the user to assume the role of an artist and a scientist simultaneously, and show firsthand that the delineation between the two is more nebulous than pedagogical stereotypes suggest.

3.3 Other Code-Based Music Composition Platforms

I am far from suggesting that TuneScope is the first platform to synthesize programming and music. Several domain-specific and functional languages exist to facilitate music composition in unique ways. Of these, I will discuss Sonic Pi and TunePad. Sonic Pi runs on the SuperCollider synthesis engine and is written in Ruby. TunePad is written in Python and offers a more DAW-like user interface, in contrast to the IDE-driven development of Sonic Pi.

Previous efforts to integrate music into computer science education using these domain-specific languages have been promising. Sonic Pi was specifically designed with pedagogy in mind, with researchers performing the first experiments on 12-year-old children with little to no coding background. Their goal was to progress from "this is a computer" to the successful completion of an operational program that generated a satisfying piece of music. The initial experiment had children running the SuperCollider synthesis engine on a Raspberry Pi microcontroller, and students wrote their code on a tailored and simplified IDE. The results of the pilot were incredibly promising; all students were successfully able to write a fun "students were extremely enthusiastic about this first encounter with the world of programming"[1].

TunePad prioritizes familiarity for producers and musicians, offering them more freedom in a bottom-up development environment. Users can code segments in Python and interact with the platform as a DAW, manipulating instruments, effects, and sequencers [10].

While Sonic Pi and TunePad were designed with novices in mind, they still present some shortcomings. For one, students need access to a Raspberry Pi and the SuperCollider engine to create music in Sonic Pi; this means making music and writing code outside of a classroom is more challenging. In TunePad's case, the authors state that a major limitation of the platform is the focus on "advanced learners" rather than the middle schoolers using it as a part of their summer enrichment curriculum [1, 15]. Second, there is significant room in both cases to make language syntax more beginner-friendly. Current literature indicates that the rise of low-code and no-code development platforms makes

computer science more accessible, especially in educational fields. Lebens and Finnegan concluded that the use of a low-code platform increased students' comfort with the Agile methodology, compared to those who were required to learn programming syntax in their course [11]. Naturally, I wondered how I could incorporate musical pedagogical ideas into a low-code, block-based platform.

Dan Garcia, a professor at the University of California, Berkeley teaches an introductory course titled "The Beauty and Joy Computing". He begins instruction in Snap! and transitions to Python for the latter half of the course. His course approaches computational thinking from a similarly project-oriented manner, even leveraging the native sound functionalities. At the Make to Learn lab, we wanted to build on the successes at Berkeley while teaching computer science with a more artistic focus.

The design of TuneScope and its accompanying curriculum synthesizes these above works to provide a more streamlined, accessible, and equitable method of engaging novices in computational thinking.

4 Design and Iterations

4.1 Inherited Work

Before my introduction to the project in June 2021, the Make to Learn Lab had already finished and cemented the art curriculum and its respective modules for class use. TuneScope at the time could play individual notes and perform small-scale sound synthesis through code but lacked the functionality to compose music and integrate music with the preceding art modules. The goal at the time was to enable students to write music entirely with TuneScope blocks. Naturally, this raised several questions. How can we play several music tracks simultaneously and ensure they are synchronized? How will students be introduced to melodies, chords, and rhythm? Will the module modules seamlessly and effectively integrate with the art modules to enhance computational thinking education? How can we disseminate this to a broader audience?

The following semester of the course offered TuneScope with the ability to create multiple tracks and play them simultaneously. Students were able to select from a plethora of instruments and track types (melody lines and loops, chord sequences and loops, and drum loops). However, the initial design was flawed and raised some confusion among students. One of the primary teaching points of the melody modules is the introduction of nested lists. The initial Play Tracks block accepted melodies with two separate lists: one for all the note values, and one for the note durations. Class feedback indicated that this was not an intuitive way to look at music, though it was intuitive in the backend code to design the musical structure in this manner. I later changed this to better resemble how written sheet music is interpreted: note values and their respective durations paired together, sequentially ordered in a list 3. The students were able to pick the format up within a few modules and create impressive music through TuneScope code. The principles upon which TuneScope was designed mandated simplicity and ease, and this required us to consider a new format for the next iteration of this curriculum.

From here, the team and I prepared TuneScope for integration as an official Snap! library. With feedback from the developers, we changed the Play Tracks module to simplify the creation of chords and melodies and rewrote our custom blocks to be more pedagogically concise. I also implemented MIDI controller capabilities–students were now able to compose a song in TuneScope and play along with a MIDI keyboard in the same browser session. This design iteration sparked several interesting demos, including one by Monty Williams from Virginia Commonwealth University. Monty, an avid xylophonist, played along to a custom chord track with his electric MIDI xylophone to demo TuneScope's capabilities and use it as a tool to teach music and computer science. I was fortunate to present my work at the Thornton Society Dinner in October of 2022, right after I had published TuneScope to the official Snap! library. As a way of

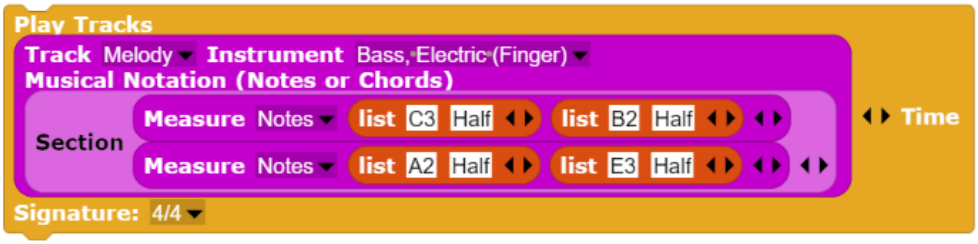

Figure 3: Example melody with the updated measure and section blocks.

showing faculty, donors, and alumni the caliber of the engineering department, my work was presented as part of current undergraduate and graduate research in the department. Here I was able to test the efficacy of TuneScope's design principles by introducing my audience (many of whom lacked programming and music experience) to a simple block-based song composition. The reception from these demos was largely positive, with several people commenting on the ease with which one could develop music with the platform. To further minimize the gap between globally available music and playable music on TuneScope, I expanded its MIDI capabilities to enable file upload and conversion. Students can import MIDI files from their favorite songs, convert them to TuneScope format with one button click, and begin modifying this composition with their ideas.

4.2 Assessing TuneScope's Effectiveness as a Learning Tool

Throughout the course, students are exposed to structured instructor feedback on their code. I used these feedback reports to examine student growth throughout eight modules from the Fall 2022 iteration of the course. Using sentiment analysis via TextBlob, I computed the average sentiment score across all students per module, and subsequently tracked the change over time. Feedback appeared as a combination of a rubric score, analysis of project design with respect to art and music, analysis of code, and suggestions for code improvements. Focusing solely on strings regarding the computer science fundamentals being assessed (abstraction, algorithms, data representation, and documentation), I filtered the feedback for each module to create a corpus explicitly for code feedback. From here, I ran each corpus through the TextBlob sentiment analysis model to determine whether feedback on student assignments and their grasp of computer science concepts improved despite the introduction of challenging new topics like music. I also split the overall feedback into sub-components to gather more CT-specific data.

5 Results

The results from this analysis 4 showed a positive increase in comprehension of computational thinking concepts after the introduction of TuneScope music modules. However, the study lacked empirical data on student performance after music modules were used in the curriculum. Furthermore, the study could not record how well the students retained computational thinking concepts after the semester had concluded.

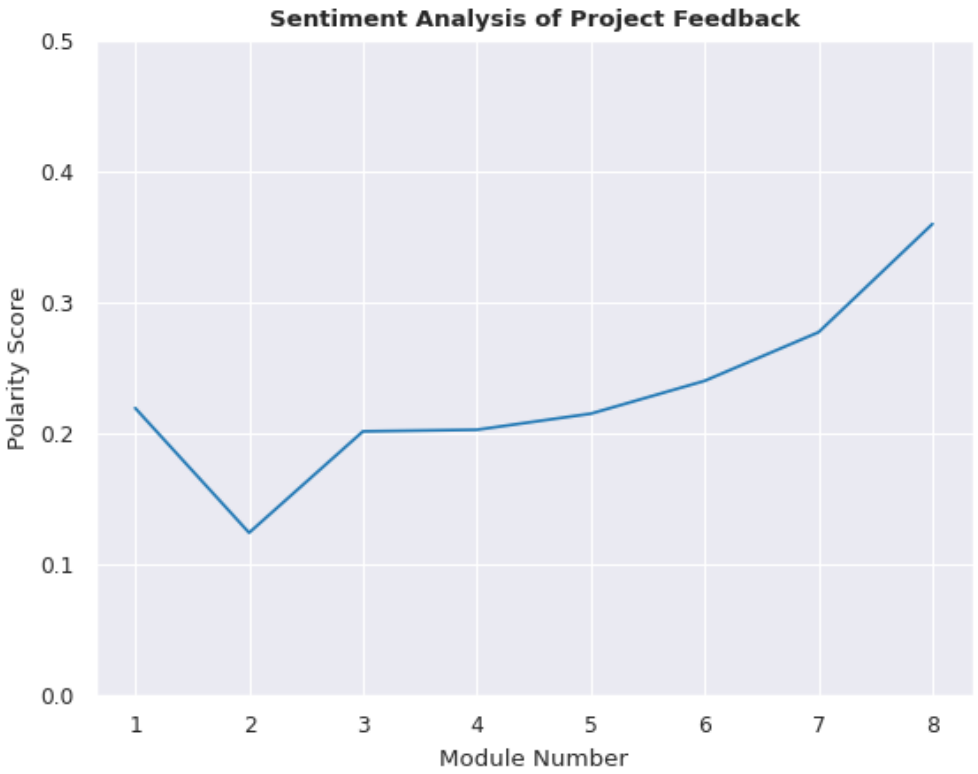

Figure 4: Polarity scores of feedback from the Fall 2022 semester, aggregated by module number. Polarities closer to +1 indicate very positive sentiment in the feedback reports. Each module averages polarities from the following subcategories: algorithms, abstraction, data representation, and abstraction.

We were able to gather subjective feedback from the students after the semester; the course received an average feedback score of 4.8 out of 5.0, and many of the comments indicated that students thoroughly enjoyed the course. The following are selected quotes that highlight TuneScope's effectiveness in engaging students with computing via art and music:

> "I had a great time in this course! The projects were a lot of fun and it was great to see some creative uses for coding. Snap was easy to use and a good platform for teaching this class, that way people with no coding experience could approach it comfortably. I feel like the techniques that I learned will be applicable to how I write code in general."
>
> "I am [matriculated as] a CS minor and I thoroughly enjoyed this course. In my CS courses we only learn the nitty gritty of coding, but in this class, I got to use that knowledge to make some really cool projects. I loved the intersection of art and coding. This course piqued my interest in the back–end of Snap and how it was coded."
>
> "I loved the mix of music, art, and programming that this course offered. I really enjoyed working on every assignment and I would love to see this course be continued."
>
> "An amazing experience that allowed me to apply my computer science knowledge to new areas. It assisted me in learning how to apply abstract concepts to art, animation, and music. I will use this knowledge in the future for sure."

Despite the course having a mix of novice programmers and older computer science majors, most students found the use of art and music a refreshing and engaging method of teaching CS principles. Several students also expressed how this course will positively influence their thinking in future computing courses. Notably in quote 3, the student verbalizes how using the Snap! platform engaged their curiosity in software design and engineering projects. They contextualized

the art and music course in the broader scope of computing courses offered at UVA, noting how they learned both the principles of computing and how to use these directly in hands-on creative projects. Quote 4 emphasizes the success of synthesizing art, music, and computing, and suggests the course has the potential for repeated instruction and support from the student body.

While we were able to garner some empirical and anecdotal data showing positive engagement from students, we also needed to tackle the issue of spreading TuneScope instruction to a wider audience, namely the users of Snap! and other introductory computer science instructors. In January 2023, I submitted a short paper detailing some of the results above to the Special Interest Group on Computer Science Education (SIGCSE), a subdivision of the Association for Computing Machinery (ACM) which allows educators to discuss the "development, implementation, and/or evaluation of computing programs [and] curricula" [13]. I was accepted as a contestant in the Student Research Competition (SRC) at SIGCSE 2023. I attended the conference in May 2023 and had the privilege of presenting my work to computer science instructors, pedagogy-focused corporations, and other research-oriented students from across the globe. Feedback from instructors on our course design and implementation was equally insightful. Many attendees commented on the novelty of teaching introductory computer science using music and were thoroughly interested in how students were taught music theory in tandem with programming. Of these, a few instructors had mentioned their unsuccessful attempts at integrating existing music-as-code tools such as SonicPi and TunePad, and appreciated the simplicity of block-based programming in this endeavor. I discussed with one professor how TuneScope's user interface implies a visual similarity to musical staves and notation, while still adhering to basic data structures such as nested lists to store notes and melodies. He found it particularly interesting, from an avid musician and computer science instructor's perspective, how TuneScope represented musical data and presented it to the user in a programming-focused manner. I also had the privilege of meeting with some of the lead developers of Snap!, namely Dan Garcia and Michael Ball, who expressed their enthusiasm for having the TuneScope music library officially integrated into Snap! With version 8.0.

6 Future Work and Reflections

6.1 Expanding Upon TuneScope

One of the current goals of the Make to Learn lab is to reach a wider audience with TuneScope. A major leap in this mission was the inclusion of TuneScope as an official library in Snap! 8.0. I believe there are two obvious and actionable paths to pursue from here.

For one, we can look at Dan Garcia's course *The Beauty and Joy of Computing*, which is offered as a self-guided online course. Since TuneScope is already segmented into distinct modules and is accompanied by a comprehensive reference manual, it makes logical sense to publish our work at the University of Virginia to an online education platform. A few problems arise with this, namely the need for automated feedback and grading of TuneScope projects (both of which are in development at the Make to Learn lab). However, this will allow us to reach an enormous audience due to its asynchronicity, and could encourage teachers in underprivileged and underfunded school districts to employ the course in their curricula.

Second, we can look to local Charlottesville school districts as primary outreach targets. Building on Gleasman et al.[8], we can introduce children to block based programming as early as middle school. By then, they will have already had some exposure to music education which could incentivize young students to experiment with TuneScope and computational thinking in their own time.

6.2 Future Studies on the Effectiveness of TuneScope

With the recent publication of our specialized textbook for the course [4], we have reached a point where we can truly begin iterating a standardized version of the course. As prior editions of the course lacked certain key features which

were implemented over my time as lead developer (synchronized multi-tracking, MIDI functionality, integration as an official Snap! library), we could not conduct a longitudinal appraisal of the music modules. Doing such a study would also enable us to observe whether our course is attracting more underrepresented groups to computer science. One of the key criticisms I received at the SIGCSE conference was that the paper didn't have data from multiple semesters of instruction. While this certainly was a shortcoming, I believe we can gather sufficient data by the conclusion of the Fall 2023 semester, especially if we implement a comprehensive numerical rubric for each assignment. Several instructors had suggested this for future semesters, but introducing a numerical grading system brings its own challenges. For one, it is difficult to judge something as abstract and creative as music with an objective, numerical score. This is one of the many ways in which computer science and music differ. While different programmers/musicians will approach their desired end product in unique ways, one can theoretically assign "correct" and "incorrect" to a programming assignment while the standards for doing so for a music composition are less defined. Professors can write test cases and demand certain product specifications for software, but who can objectively assess the merit of two different music pieces? I believe this is part of what makes TuneScope a powerful learning tool in the first place. Students are not unilaterally focused on delivering a "correct" end product and instead are encouraged to take creative liberties within project guidelines. A study by Jennifer Crocker at the University of Michigan showed that more than 80% of students based self worth on academic competence, with 66% basing self worth on "doing better than others" [5]. Students have been found to take on greater academic risks and "seek out more challenging problems"[6] when grades are not involved, compared to when they are graded on the work they produce. Can we at the Make to Learn lab separate artistic expression from computational competency, and assess the latter while encouraging students to take on academic and creative challenges [3]?

Part of what has made the objective assessment of students' computational thinking comprehension difficult is the notion of "equitable grading practices". In a panel regarding this topic at SIGCSE 2023, several practices and their efficacies were discussed which could help Make to Learn analyze the course's success over time. One professor explained her method of grading students on an integer scale from 1 to 4, where 4 denotes mastery in a topic or concept. The key facet to this was the ability for students to receive feedback on their assignments, make changes to their submissions, and submit it after the due date to bump up their grade. Another professor (namely Dan Garcia, key member of the Snap! team) also implemented this "infinite deadline" concept to great success [7]. Perhaps future iterations of our course can implement an empirical rubric and allow students to adjust their grade using the feedback from instructors (the same kind of feedback used in our sentiment analysis).

TuneScope students over the last few semesters have developed immensely creative projects that test the boundaries of the course and beget the development of new features on our end. We want to continue empowering students to self-regulate their learning and explore computational thinking on their own volition.

6.3 TuneScope and Diversity in Computing

The overarching goal of the TuneScope project is to promote diversity in computer science. While we may not have enough longitudinal evidence to track enrollment and retention of underrepresented groups, prior studies underlining the importance of "identity" in computing and current feedback from many of our students (half of whom come from underrepresented groups in computing) suggest we are on the right track. Our students have demonstrated pride and ambition in their work, and have explicitly stated that they will continue to take computer science courses and employ the skills they learned via TuneScope.

As I reflect on my own identity as a computer scientist and musician, I realize how much my career has been influenced by the support and encouragement I received as a novice. My early computer science courses and activities (read: robotics) were given to me as sandbox environments. I was encouraged to explore, take risks, make mistakes, and learn with nuance. Moreover, the projects I developed were uniquely mine, and this motivated me to pursue the field at the

collegiate level. In tandem, music has always been a hobby to me; with no pressure to perform or be judged on my performance, I was again given autonomy to explore my own interests and passions.

I think about how many students are not afforded the same liberties and privileges I was afforded. I was lucky to take a computer science course in high school; some districts don't even have the funds to buy spare computers for their students, let alone hire or train new computer science teachers. There are hundreds upon thousands of kids who could have been inspired to pursue computing, but never had the opportunity to even begin. Working with TuneScope has made me more cognizant of my role in reducing this disparity, and I will carry this understanding throughout my career as a software engineer and computer scientist.

7 Conclusion

TuneScope has shown great promise in engaging novice computer science students with fundamental computational thinking skills. Music has proven to be an effective medium to inspire students to take creative risks and explore these skills through a low cost, low code platform. While the long term effects on computer science retention and diversification of TuneScope at the University of Virginia and beyond are yet to be realized, our initial work at the Make to Learn lab has shown favorable progress toward these goals. Focusing efforts to provide a low-overhead, music-based computing curriculum to underfunded and underrepresented schools will be the next step in democratizing computational thinking.

8 Acknowledgements

I would like to recognize my research advisor N. Rich Nguyen and the rest of the Make to Learn team: Glen Bull, Jo Watts, Rachel Gibson, and Eric Stein. Without them I would not have had the opportunity to work on such a meaningful project, attend incredible conferences, and most importantly realize my desire to combine music and computer science. I would also like to thank my parents, Sapna and Vikrant Padhye, for encouraging me to pursue my passions and supporting my decisions every step of the way. Finally my friends, to whom I owe the utmost gratitude for showing how much of a joy it has been to live and study at the University of Virginia.

References

[1] AARON, S., AND BLACKWELL, A. F. 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* (New York, NY, USA, 2013), FARM '13, Association for Computing Machinery, pp. 35–46. event-place: Boston, Massachusetts, USA.

[2] BARR, V., AND STEPHENSON, C. Bringing Computational Thinking to K-12: What is Involved and What is the Role of the Computer Science Education Community? *ACM Inroads 2*, 1 (Feb. 2011), 48–54.

[3] BASHIR, M. M. The Value and Effectiveness of Feedback in Improving Students' Learning and Professionalizing Teaching in Higher Education. *Journal of Education and Practice* (Jan. 2016).

[4] BULL, G., GIBSON, R., WATTS, J., NGUYEN, N. R., AND DAHL, L. *TUNESCOPE Creating Digital Music in Snap!* Association for the Advancement of Computing in Education, University of Virginia, Charlottesville, VA, 2023.

[5] CROCKER, J., AND KNIGHT, K. M. Contingencies of Self-Worth. *Current Directions in Psychological Science 14*, 4 (2005), 200–203.

[6] DECI, E. L., RYAN, R. M., AND WILLIAMS, G. C. Need satisfaction and the self-regulation of learning. *Learning and Individual Differences 8*, 3 (1996), 165–183.

[7] GARCIA, D., CAMARENA, M., LIN, K., AND WESTERLAND, J. Equitable Grading Best Practices. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2* (New York, NY, USA, 2023), SIGCSE 2023, Association for Computing Machinery, pp. 1200–1201. event-place: Toronto ON, Canada.

[8] GLEASMAN, C., FEGELY, A., AND KOESTER, C. Making Computer Science Education Accessible to Rural Middle School Female Students: A Conceptual Framework. In *Proceedings of Society for Information Technology & Teacher Education International Conference 2020* (Online, Apr. 2020), D. Schmidt-Crawford, Ed., Association for the Advancement of Computing in Education (AACE), pp. 24–28.

[9] GOODE, J., SKORODINSKY, M., HUBBARD, J., AND HOOK, J. Computer Science for Equity: Teacher Education, Agency, and Statewide Reform. *Frontiers in Education 4* (2020).

[10] HORN, M., BANERJEE, A., AND BRUCKER, M. TunePad Playbooks: Designing Computational Notebooks for Creative Music Coding. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2022), CHI '22, Association for Computing Machinery. event-place: New Orleans, LA, USA.

[11] LEBENS, M., AND FINNEGAN, R. Using a Low Code Development Environment to Teach the Agile Methodology. In *Agile Processes in Software Engineering and Extreme Programming* (Cham, 2021), P. Gregory, C. Lassenius, X. Wang, and P. Kruchten, Eds., Springer International Publishing, pp. 191–199.

[12] MICKELSON, R. A., MIKKELSEN, I., DORODCHI, M., CUKIC, B., AND HORN, T. Fostering Greater Persistence Among Underserved Computer Science Undergraduates: A Descriptive Study of the I-PASS Project. *Journal of College Student Retention: Research, Theory & Practice 0*, 0, 15210251221086928.

[13] PADHYE, H., GIBSON, R., BULL, G., AND NGUYEN, N. R. Does musical context improve computational thinking skills? In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2* (New York, NY, USA, 2023), SIGCSE 2023, Association for Computing Machinery, p. 1231.

[14] PARKER, M. C. Barriers and supports to offering computer science in high schools: A case study of structures and agents. *ACM Trans. Comput. Educ. 23*, 2 (mar 2023).

[15] PETRIE, C. Programming music with Sonic Pi promotes positive attitudes for beginners. *Computers & Education 179* (2022), 104409.

[16] SYLWIA HOLMES, S. H. The impact of participation in music on learning mathematics. *London Review of Education 15*, 3 (2017), 425–438.

[17] WANG, J., HONG, H., RAVITZ, J., AND HEJAZI MOGHADAM, S. Landscape of K-12 Computer Science Education in the U.S.: Perceptions, Access, and Barriers. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (New York, NY, USA, 2016), SIGCSE '16, Association for Computing Machinery, pp. 645–650. event-place: Memphis, Tennessee, USA.

[18] YADAV, A., HONG, H., AND STEPHENSON, C. Computational Thinking for All: Pedagogical Approaches to Embedding 21st Century Problem Solving in K-12 Classrooms. *TechTrends 60*, 6 (Nov. 2016), 565–568.