Engaging CS1 Students with Audio Themed Assignments

Matthew McQuaigue Mack Larson Philip Smith
Sydney Melech Kalpathi Subramanian Erik Saule
Department of Computer Science
University of North Carolina at Charlotte
Charlotte, NC 28223
{mmcquaig,clarson9,psmit145,smelech,krs,esaule}@uncc.edu

Abstract

Early computer science courses (CS1, CS2) are the cornerstone of student understanding of computer science. These courses introduce the foundational knowledge of computer science needed to understand more complex topics and to be successful in follow-on courses. It is thus important to introduce CS concepts in an engaging and easy-to-understand manner to increase student interest and retention. This paper presents a new approach to teaching the Computer Science 1 (CS1) course through our *BRIDGES* system. This approach aims to increase student engagement and improve learning outcomes by using *audio-based assignments* that they can manipulate and process audio signal information, as well as visualize and play them. We explain how to design and implement audio-based assignments and connect them to fundamental programming constructs such as variables, control flow, and simple data structures, such as arrays. These assignments encourage and engage students by using audio data they are interested in to write code, promoting problem-solving and improvements in their critical thinking skills.

1 Introduction

Early computer science courses are pivotal in teaching the foundations of computer science in introducing topics that shape the future of students' academic journeys and careers. On top of foundational knowledge of computer science, students learn computational literacy, problem-solving, and critical thinking skills. As beginning courses in the field contain a lot of information, these courses can be complex and frustrating for students, and for some students,

it could be the first time they are introduced to these topics. Even though progress is being made in retention rates in computer science using a variety of mechanisms and pedagogies, additional motivation and engagement is critical in early CS courses to continue to increase retention, and equip students with a strong foundation for future success. From establishing a strong base for advanced coursework to preparing students for the demands of the workforce, we believe CS1 should emerge as a transformative experience that extends far beyond the confines of the classroom to ensure that they understand the potential of computing to solve real-world problems.

Traditional programming courses often rely on text-based exercises, which can make it difficult for students to stay interested and may not cater to their diverse learning styles. In response to this challenge, our proposed curriculum includes a series of programming tasks that use audio-based stimuli to create a dynamic and multisensory learning experience. Using the *BRIDGES* system, [10, 9, 5] we have created a database of assignments for introductory computer science courses [4] and also demonstrate how they could be used in CS1/CS2. The *BRIDGES* API is available in Java, C++, and Python, and provides students with objects to load and manipulate real-world audio files in early CS courses. It also creates visualizations and plays corresponding audio data to demonstrate student-generated work while maintaining course rigor.

This paper contributes to the ongoing discourse on innovative teaching methodologies in computer science education and provides practical insights and tools for instructors seeking to enhance their students' introductory programming experience. By embracing audio-based assignments, educators can create a more inclusive and interactive learning environment, ultimately preparing better novice programmers for the challenges of the digital era. These assignments rely on the manipulation of wavefront audio information and the generation of signals to create and edit audio data at a level that is appropriate for novice programmers. The proposed assignments provide two advantages, (1) they continue to provide highly engaging assignments with a visual and audio output for CS1 students, and, (2) students' output can be customized based on different songs or around choices of their own interest. This gives more flexibility to assignments and allows students to manipulate real-world audio data that can be shared with other students outside the class.

2 Related Works

2.1 What Makes Students Engaged

In previous years, work has been done to better engage and motivate students in early CS1 courses. According to Hendlesman et al. [21] engagement can include the involvement of skills, participation, emotion, and/or performance.

The methods of implying engagement in the classroom are either *content-based* or pedagogical activities. Content-based engagement focuses on making course content or activities (assignments, lectures, videos, etc) meaningful and relevant to student interests. A good resource for this type of learning engagement can be assignment repositories such as Nifty assignments [29], EngageCSEdu [26], and game-themed assignments [14]. These types of learning can tackle real-world coding challenges in collaborative teams, fostering teamwork and practical problem-solving skills. This allows for the inclusion of code reviews and critiques to encourage a culture of constructive feedback, allowing students to refine their coding practices through peer interaction. Engagement based on activities includes active learning, lab-based instruction, flipped classrooms, gamification, peer learning/coding, and multimedia content [30, 19, 22]. Gamifying by incorporating elements like coding competitions or level progression adds a competitive yet motivating dimension while using methods such as peer learning models to provide hands-on, immediate experiences that solidify theoretical concepts.

2.2 What CS1 Courses Typically Look Like

CS1 courses usually serve as the introductory course in computer science. This course can teach the fundamentals of programming such as variables, data types, control flow, functions, and algorithmic thinking. Algorithmic thinking can help break down complex problems into smaller, more manageable components. Some CS1 courses also introduce simple data structures in the form of lists and strings. CS1 often uses programming languages such as Java, C++, or Python. The curriculum heavily involves hands-on coding assignments that range from simple exercises and projects, foster problem-solving skills, language syntax and semantics. Exemplars of CS1 courses can be found in the ACM 2013 Curriculum guidelines [23]. Due to the wide variations in learning environments, tools, student population, and demographics, there is no one-size-fits-all approach to teaching CS1. Additionally, there are strong opinions regarding when and how object-oriented programming should be taught [6]. This is also true for discussion about IDEs [31] and languages [25] to use. Many introductory courses now incorporate graphics, GUIs, and visualizations [17, 10]. These are sometimes used as creative output in student projects, or to illustrate key aspects of underlying objects or algorithms [13, 24, 7, 18, 32].

Furthermore, some institutions are experimenting with introductory computer science courses to now incorporate other advanced topics at a beginner level increasing multidisciplinary value. This includes games [2], robotics [12], image processing and generations [3], and statistics and data science [20, 11].

2.3 Existing Educational Efforts incorporating Audio Related Content in CS Courses

Past audio-based learning tools have explored the CS foundations supporting the integration of auditory learning in computer science education through manipulating audio to reinforce concepts in CS courses. Burg et al. [8] created a book and online learning supplements to make CS concepts more engaging and relevant by using sound/music applications. The authors map their curricular material to core CS courses, giving examples of programming assignments using sound/music applications that teach fundamental CS skills. Adams et al. [1] created a tool called Thread Safe Audio Library (TSAL) to explore the use of algorithm sonification - representing algorithm behavior using sound - as a pedagogical tool for computer science education. An assessment found that sonification improved students' long-term recall of the relative speeds of sorting algorithms, providing evidence that sonification aids learning. Students also rated sessions with visualizations as more engaging than non-visual sessions. TuneScope [27] enabled using sound-based activities into Snap!.

EarSketch [15] developed by Freeman et al. is a free, web-based learning environment that teaches introductory computer science through music. It provides an in-depth introduction to computer science and programming through composing, producing, and remixing music with Python and JavaScript code. Freeman et al. also found that using EarSketch showed statistically significant gains in student attitudes across 7 constructs related to computing engagement and motivation in underrepresented groups in high school [16].

3 A Set of Engaging and Scalable Audio Assignments

This section presents a set of assignments, all suitable for CS1/CS2 classes and leveraging the *BRIDGES* toolkit. Table 1 presents the assignments, related topics that they map to, and their engagement characteristics.

The assignments cover most of the introductory computer science class topics, including function calls, control flow, conditionals, lists, and OOP. The engagement strategies are all content-based. The most common engagement characteristic is that the assignment produces a visual and audio output that can be interpreted and played. Most of the assignments use real audio wave data and perform a real analysis or solve a real-life problem. For all assignments, instructors have the flexibility to scaffold what information and functions to expose to the student.

The assignments are all downloadable online[1] and are scaffolded. Solutions are shared with instructors on request. All assignments can be done in any

[1] https://bridgesuncc.github.io/

Assignment	Topics	Engagement
Subsampling Bitdepth	Variables, Data Repr.	Visual/Audio Output, Audio Choice
Thresholding	Conditionals	Visual/Audio Output, Real Data, Audio Choice
Scaffolded Player	Strings, IO, Loops	Visual/Audio Output, Real Data Analysis, Audio Choice
Effects (Mixing, Fade,Volume Change, Clamping)	1D arrays	Visual/Audio Output, Real Data, Real Problem, Multiple Sounds
Visualize Volume Levels	2D arrays	Visual/Audio Output, Real Data, Real Problem, Audio Choice, New Visualization
Full Frequency Player	Functions	Visual/Audio Output, Real Data Analysis, Audio Choice, Tool Building, Fun
Instrument Types	Basic Objects	Visual/Audio Output, Simulation, Fun, Create Unique Beats
Basic Compression	Project	Visual/Audio Output, Uses all topics, Fun outputs, and Competitions for best compression.

Table 1: A set of assignments using audio data to teach CS1.

of the three programming languages supported by *BRIDGES*, namely, C++, Java, and Python.

3.1 How Do Computers Encode Sound?

All our activities rely on the simple processing of sounds in a computer. Audio signals can be represented in several ways in a computer. However, we are mostly interested in manipulating the raw audio signal, similar to what would be stored in an uncompressed wave file.

An audio clip is represented as several channels: 1 for mono, 2 for stereo, 6 for 5.1 systems. Each channel represents the movement that a membrane has to make to play the signal back. The position of the membrane is sampled at regular intervals and that information is encoded as a sampling rate. For instance, a musical CD encodes the music sampled at 44.1KHz, while telephony is often encoded at 8KHz. So a channel can be seen as a given number of samples that encode a particular clip that lasts $\frac{number\_of\_sample}{sampling\_rate}$ seconds.

Finally, each sample represents a displacement of the membrane compared to its resting position which takes its most positive value when the membrane should be the farthest outward from the resting position, and the most negative value when the membrane should be the most inward to the resting position. The samples can be represented in different formats. Usually, they are represented by signed integers, though the number of bits differs per quality level. For instance, music CDs were encoded at 16-bit, while telephony is often en-

coded in 8-bit, and the master used in music production are often encoded in 24-bit or 32-bit.

In summary, an audio signal is essentially one or multiple array(s) of integers that explicitly encode a sound wave. This representation is the one used by *BRIDGES* which lets you load a WAV file in that format and manipulate it with simple API calls. This is the format that students will use. Even though advanced audio processing will require complex mathematics (such as Fourier Transform to shift to the frequency domain), all our activities use the simpler representation which is suitable for novice programmers while enabling multiple assignments to span the content of a CS1 course.

3.2 How *BRIDGES* Works

The *BRIDGES* toolkit provides a simple API to create and manipulate an AudioClip object as a collection of samples, which are numbers, each one representing a single part of the sound wave in the clip. Currently, you can generate an AudioClip by setting the samples individually or by importing a WAV file.

The AudioClip object will also have a sample count, a channel count, a bit depth, and a sample rate. These attributes can be custom-set and retrieved through function calls from the object. The channel count is the number of "channels", or continuous streams of samples. With multiple channels, you can play different sound waves simultaneously, which is used for things like stereo audio (a different wave for each ear).

The WAV file can be passed in as a parameter to the AudioClip object using a string that points to a WAV file. The AudioClip object will read and parse the WAV file in a form for processing for the user. The *BRIDGES* API provides a visualization function that can have an AudioClip attached to and sent to the server for visualization. Students can then visit the website through their account and playback the audio clip with visuals provided, as can be seen in Figure 1. Their visual clips can also be shared with other students.

The *BRIDGES* API has multiple forms of documentation and tutorials on how to manipulate and use the objects *BRIDGES* provides[2]. Due to the simplicity of the API and the level of scaffolding provided in the assignments, minimal effort is needed to understand and use the API even for new programmers.

[2]https://bridgesuncc.github.io/tutorials/AudioClip.html

4 Audio Based Assignments

4.1 Subsampling/Bitdepth

In this assignment, students will conduct audio processing by manipulating the sample rate, and bit depth of the signal through attributes of the AudioClip object to analyze the differences between the original and modified audio signal. Additionally, they are asked to analyze how these changes affect the file size and quality. The assignment also focuses on the impact of changing variable types, such as using 16-bit vs 32-bit integers, or floating point values on precision, and compression.

Since this assignment requires the manipulation of arrays that contain samples and 1D arrays have not been a topic yet, scaffolded code is used that is complete and takes the variables and uses them as needed. In later assignments, more and more of this code can be exposed to the student for implementation and follow-on assignments. The flexibility of what code and when it is exposed to the student is up to the instructor.

Throughout the assignment, students will focus on the understanding of audio processing while encouraging critical thinking about performance-based trade-offs of selecting certain sample rates, bit depths, and variable types. Instructors could grade for the correctness of implementation and clarity in the explanation of parameter effects. Students could also choose different sound files based on their interests to help boost engagement.

4.2 Thresholding

For conditionals, students can learn about thresholding to create a basic noise gate for noise reduction. To start, the students will be given the task of loading an audio file and visualizing its waveform using *BRIDGES*. After that, they will be taught to go through the audio samples and apply a threshold to identify portions of the signal that exceed a certain amplitude level. If arrays haven't been taught yet, a function can already be provided to loop through the array.

After identifying portions of the audio that fall within a certain threshold, students would use a conditional statement to modify or remove those samples. To learn more about thresholding and how it affects output quality, students should be encouraged to experiment with different threshold levels and observe how changing the conditional statements affects the audio output. They would also explore how their approach handles various types of audio signals, such as speech, music, or environmental sounds. Students can see how this form of noise reduction affects different types of audio. Different sound clips or songs will have different levels of noise and thus be affected more.

From this assignment, students get an understanding of how conditionals

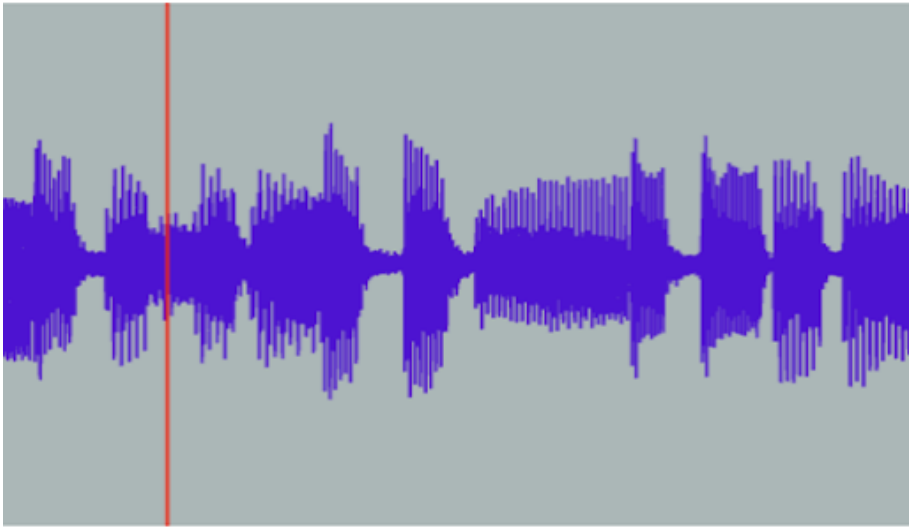

Figure 1: This is a sample audio signal visualized using *BRIDGES*. This visualization has playback buttons along with an interactive window for analyzing the signal.

can be applied to real-world audio signals. This assignment also focuses on the importance of proper parameter selection and the impact choices have on audio quality.

4.3 Scaffolded Player

This assignment tasks the student to use a scaffolded song player program, where a given set of notes and their corresponding durations are read from a file and played sequentially. This assignment is an inspiration from Nifty's Melody assignment [28]. The assignment scaffold provides a pre-defined function responsible for playing a specific note for a specified duration of time. Students are tasked with implementing the functionality to parse the input file, and extract note-frequency pairs along with their durations.

The input file contains entries representing musical notes along with their respective durations, structured as follows: each line contains a note (e.g., F#3) followed by a duration (e.g., 200ms). The program reads this file, extracts note-duration pairs, and plays each note using the provided function for the specified duration.

Unlike a full-frequency player, which is a later assignment in section 4.6, that encompasses the entire process of generating audio signals based on math-

ematical formulas (such as sine waves) and managing playback, this assignment focuses solely on reading note-duration pairs from a file and using a scaffolded function to handle adding the note to the AudioClip object. By completing this task, students reinforce their understanding of file IO operations and string manipulation.

4.4 Sound Effects (Mixing, Fade, Volume Change, Clamping)

In this assignment, students implement basic audio effects such as mixing, fade, volume change, and clamping. For mixing, students would be tasked with combining two or more audio files. Arrays are used as AudioClip objects containing a list of channels and each channel contains a list of samples. For this assignment, the audio to be created will use one channel. To mix two samples you take the average (or a weighted sum) of the two values.

The fade effect could be introduced to teach students how to change the amplitude of audio samples over time. The goal is to fade one audio source out while another audio source fades in. Think about how song compilations will fade a song out near the end while fading in the next song so that the transition is smooth. As a sound is closer to the end, its amplitude will be dampened, and the next sound amplitude with be increased.

To teach volume change, students can apply a scalar value to the entire audio array, allowing them to understand how changing the amplitude uniformly affects the overall volume of the audio signal.

Lastly, the clamping effect could be implemented to teach students about limiting the amplitude values within a specific range. This is similar to the threshold assignment, but the values are limited to the threshold rather than being removed.

All assignments involving these effects include using conditional statements and array iteration to identify and modify samples.

4.5 Visualize Volume Levels

Students will be tasked with visualizing the volume (amplitude) of an audio signal over time. The goal is to introduce the concept of representing time-varying data using a 2D array, where one axis represents time, and the other represents amplitude. Students could read in the audio file with a 1D representation of samples. Students then could create a *BRIDGES* ColorGrid object, which supports the creation of 2D images and has methods for coloring individual image cells.

For this image, the x-axis or columns correspond to time (time bins or frames), and the y-axis or rows represent the amplitude values at each time point. The idea is to separate the audio signal into frames to capture its change

over time. Divide the audio signal into consecutive time frames, and for each frame, calculate the average amplitude level. Each frame/bin is represented by the current image column with the average calculated amplitude plotted on the y-axis.

Students can analyze the generated visualization to gain an understanding of how the volume of the audio signal changes over time. They can begin to recognize different parts of sounds and music that represent transitions from loud and soft segments. They could also recognize repetition and see which parts are similar to another.

4.6 Full Frequency Player

Students can create a basic frequency player through the implementation of a sine wave generator that controls amplitude, frequency, and time of a note. This is an extension of the scaffolded player in section 4.3. Students are tasked with generating sample notes for specified durations that correspond to a sine wave, utilizing a given mathematical formula. The notes are also read from an external file describing notes and durations in the same way as the scaffolded player.

Students are also tasked to adjust the start and end parameters of the sine wave to correspond to specific time intervals measured in seconds, thus allowing for the generation of sine wave samples over a defined duration. This enables the creation of a sine wave audio clip spanning a specified time range playing multiple notes, effectively serving as the foundation for a basic frequency player. Students are encouraged tom implement smooth transitions between successive samples enhancing the quality of the frequency player's output.

Overall, students creating a basic frequency player develop proficiency in several fundamental programming concepts. Firstly, it aims to increase understanding of modular programming by utilizing methods or functions to compartmentalize code, thereby enhancing code organization and reusability. The task enhances proficiency in data type manipulation and conversion, particularly through the casting between different data types.

4.7 Instrument Types

To teach the use of objects and encapsulation, students can create basic musical instruments that have their attributes. Students create an instrument class, where they encapsulate attributes like pitch, amplitude, and waveform type. Each class is structured to represent a distinct musical instrument, introducing students to the fundamental principles of object-oriented programming. Students could create an Instrument base class with individual instrument classes to teach polymorphism. Within these classes, students implement a play

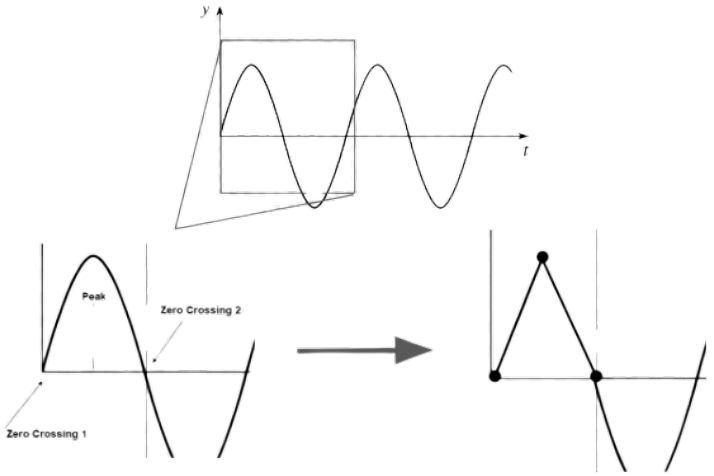

Figure 2: A sample illustration of compression one peak from a sin wave using two zero crossing points and a local peak. This creates a linear signal between the peaks and zero crossings.

method that generates the instrument's sound based on specified attributes, thereby delving into the concept of encapsulation and method definition within objects. Students proceed to instantiate objects from these classes, creating instances of various musical instruments and customizing their attributes. The assignment advances to the concatenation of these objects into a single AudioClip allowing for the playback of multiple instruments to create musical compositions through mixing and adding to different channels. Students would already know how to perform mixing from previous assignments.

4.8 Basic Compression

This audio-processing assignment is focused on wavefront compression through zero crossings with activation thresholds,

Beginning with the loading and representation of audio data as a sequence of samples, students conduct a detailed analysis of the waveform, pinpointing instances of zero crossings where the audio waveform intersects the zero amplitude line, and marking significant signal transitions. The compression algorithm, designed by the students, captures the audio data by storing only the temporal positions of zero crossings and the peak values between zero crossings (see Figure 2).

This approach aims to reduce the number of stored samples in the com-

pressed representation while retaining perceptually relevant features. Students experiment with various activation threshold values, exploring the trade-offs between compression efficiency and the preservation of audio information. Decompressing the signal involves reconstructing the signal using the positions of the zero crossings, the peaks between the zero crossing, and using a linear function (line segment to approximate the original samples) removed during the compression phase. Decompression and playback assess signal preservation after compression.

Instructors can look at compression effectiveness, considering factors like compression ratio, file size reduction, and perceptual quality, when grading this work. Students gain a comprehensive understanding of the impact and trade-offs associated with lossy compression, and see if certain signals are worth the quality loss. Students could compete to see who can make the best audio compression algorithm to help enforce engagement with that assignment and other students. Instructors can also relate audio compression to image/video compression (such as JPEG, MPEG) in their coverage of this topic.

5 Conclusion

We have presented a sequence of audio-themed assignments appropriate for CS1/CS2 level courses supported as part of our *BRIDGES* toolkit, that allows for students to easily use and edit audio Wavefront files for content-based learning that promotes engagement. These assignments are also easy to incorporate in the course for instructors with an already completed repository of assignments along with the solutions and scaffolds needed. The assignments map to the most commonly covered topics and learning objectives within CS1, with flexibility to customize each assignment by student interests, and available for wider use in Java, Python, or C++. Each assignment builds on prior knowledge from the one before it with an end-of-class project incorporating all knowledge.

Some limitations of this work is that it has not been incorporated into a full CS1 course yet. The tool has been used with individual assignments by current and past *BRIDGES* users for multiple semesters, with positive feedback from both instructors and students. Also, currently, our tool only supports the generation of our own signals from scratch or by importing files in WAV formats. Extensions to other audio formats can be envisioned. In the future, we also plan on adding additional assignments with more variations.

References

[1] J. C. Adams, B. D. Allen, B. C. Fowler, M. C. Wissink, and J. J. Wright. The sounds of sorting algorithms: Sonification as a pedagogical tool. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education - Volume 1*, SIGCSE 2022, page 189–195, New York, NY, USA, 2022. Association for Computing Machinery.

[2] J. D. Bayliss and S. Strout. Games as a flavor of cs1. In *ACM SIGCSE Bulletin*, volume 38, pages 500–504. ACM, 2006.

[3] A. Beckman, M. Mcquaigue, A. Goncharow, D. Burlinson, K. Subramanian, E. Saule, and J. Payton. Engaging Early Programming Students with Modern Assignments Using BRIDGES. volume 35, page 74–83, Evansville, IN, USA, apr 2020. Consortium for Computing Sciences in Colleges.

[4] BRIDGES Development Team. BRIDGES Assignment Repository. `http://bridgesuncc.github.io/newassignments.html`, 2022.

[5] BRIDGES Development Team. BRIDGES Website. `http://bridgesuncc.github.io`, 2022.

[6] K. B. Bruce. Controversy on how to teach cs 1: a discussion on the sigcse-members mailing list. In *ACM SIGCSE Bulletin*, volume 36, pages 29–34. ACM, 2004.

[7] S. Buchanan, B. Ochs, and J. J. LaViola Jr. Cstutor: a pen-based tutor for data structure visualization. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 565–570. ACM, 2012.

[8] J. Burg, J. Romney, and E. Schwartz. Computer science "big ideas" play well in digital sound and music. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, page 663–668, New York, NY, USA, 2013. Association for Computing Machinery.

[9] D. Burlinson, M. Mcquaigue, A. Goncharow, K. Subramanian, E. Saule, J. Payton, and P. Goolkasian. Bridges: Real world data, assignments and visualizations to engage and motivate cs majors. *Education and Information Technologies*, 2023.

[10] D. Burlinson, M. Mehedint, C. Grafer, K. Subramanian, J. Payton, P. Goolkasian, M. Youngblood, and R. Kosara. BRIDGES: A System to Enable Creation of Engaging Data Structures Assignments with Real-World Data and Visualizations. In *Proceedings of ACM SIGCSE 2016*, pages 18–23, 2016.

[11] S. Dahlby Albright, T. H. Klinge, and S. A. Rebelsky. A functional approach to data science in cs1. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 1035–1040. ACM, 2018.

[12] A. Delman, A. Ishak, L. Goetz, M. Kunin, Y. Langsam, and T. Raphan. Development of a system for teaching cs1 in c/c++ with lego nxt robots. In *FECS*, pages 396–400, 2010.

[13] P. Dewan. How a language-based gui generator can influence the teaching of object-oriented programming. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 69–74. ACM, 2012.

[14] P. Drake and K. Sung. Teaching introductory programming with popular board games. In *Proceedings of ACM SIGCSE*, SIGCSE '11, pages 619–624, 2011.

[15] J. Freeman, B. Magerko, D. Edwards, and L. Ikkache. Earsketch, a web-application to teach computer science through music (abstract only). pages 640–640, 03 2017.

[16] J. Freeman, B. Magerko, T. McKlin, M. Reilly, J. Permar, C. Summers, and E. Fruchter. Engaging underrepresented groups in high school introductory computing through computational remixing with earsketch. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, page 85–90, New York, NY, USA, 2014. Association for Computing Machinery.

[17] I. Greenberg, D. Kumar, and D. Xu. Creative coding and visual portfolios for cs1. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 247–252. ACM, 2012.

[18] P. J. Guo. Online python tutor: embeddable web-based program visualization for cs education. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 579–584. ACM, 2013.

[19] M. Guzdial. A media computation course for non-majors. In *Proceedings of the ITICSE 2003*, pages 104–108, 2003.

[20] O. A. Hall-Holt and K. R. Sanft. Statistics-infused introduction to computer science. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pages 138–143. ACM, 2015.

[21] M. Hendelsman, W. Briggs, N. Sullivan, and A. Towler. A measure of college student course engagement. *Journal of Educational Research*, 98(3):166–175, 2005.

[22] D. Horton, M. Craig, J. Campbell, P. Gries, and D. Zingaro. Comparing outcomes in inverted and traditional CS1. In *Proceedings of the ITICSE 2014*, pages 261–266, 2014.

[23] Joint Taskforce on ACM Curricula. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science (Page 513)*. ACM/IEEE Computer Society, 2013.

[24] A. N. Kumar. The effectiveness of visualization for learning expression evaluation. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pages 362–367. ACM, 2015.

[25] W. M. Kunkle and R. B. Allen. The impact of different teaching approaches and languages on student learning of introductory programming concepts. *ACM Transactions on Computing Education (TOCE)*, 16(1):3, 2016.

[26] NCWIT, 2018. `https://www.engage-csedu.org/`.

[27] N. R. Nguyen, H. Padhye, E. Stein, and G. Bull. Tunescope: Engaging novices to computational thinking through music. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 2*, SIGCSE 2022, page 1181, New York, NY, USA, 2022. Association for Computing Machinery.

[28] A. Obourn and M. Stepp. Melody. `http://nifty.stanford.edu/2015/obourn-stepp-melody-maker/`.

[29] N. Parlante. Nifty assignments, 2018.

[30] J. Pirker, M. Riffnaller-Schiefer, and C. Gütl. Motivational active learning: Engaging university students in computer science education. In *Proceedings of ITICSE*, pages 297–302, 2014.

[31] C. Reis and R. Cartwright. Taming a professional ide for the classroom. In *ACM SIGCSE Bulletin*, volume 36, pages 156–160. ACM, 2004.

[32] S. Schaub. Teaching java with graphics in cs1. *ACM SIGCSE Bulletin*, 32(2):71–73, 2000.