Musical Trees - Engaging a Different Kind of Student

Jule Schatz
drschatz@illinois.edu
University of Illinois Urbana-Champaign
Urbana, USA

Course CS2
Programming Language C++
Knowledge Unit Trees
CS Topics Trees, memory management, recursion

Synopsis

Money doesn't grow on trees, but music does! Musical Trees is an original project for students in a data structures class that covers trees, recursion, and memory management. In the project, students build a general tree where each node holds a musical motif (5 notes). Students then write code to evolve and prune the tree across multiple generations to elaborate and extend the original motif. Finally, students traverse the tree to produce a unique melody.

Musical Trees is an engaging project that allows students to hear the results of their code. It provides an opportunity to include students with musical backgrounds, while requiring no prior musical knowledge. The project is fully autogradable, yet still allows meaningful design freedom. It was tested and loved by a course of 80 students, many of whom extended the project to incorporate their own music theory ideas. Musical Trees combines the challenges of programming with the creativity of music generation.

Keywords

trees, memory management, recursion, music

ACM Reference Format:
Jule Schatz. July 2026. Musical Trees - Engaging a Different Kind of Student. In *ACM EngageCSEdu.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3819592

1 Engagement Highlights

One of the biggest strengths of the Musical Trees project is its interdisciplinary connection to music. The project lets students to listen to the results of their code and encourages creative application of music theory. Making interdisciplinary connections has been shown to help students see computing in varied contexts and connect it to their own interests and experiences, which can increase motivation and persistence in computing courses [1]. This approach engages a population of students who may feel less interested in more traditional CS projects, while still remaining accessible to all learners.

The project provides opportunities to incorporate student choice. The relevant music theory related code is provided in the starter files, but students are encouraged to expand, adjust, and experiment with it to change the final musical outcome. Incorporating student choice is a recommended practice for engaging learners by connecting coursework to their goals and passions [1]. I also provided an online discussion post where students can share the melodies their code produces and describe any additional logic they implemented. Sharing work with peers supports social learning and helps students build confidence, agency, and a sense of community in computing [1].

Additionally, the project introduces ethical and societal implications for students to reflect on. The code they write will produce unique musical melodies, but what role should technology play in music generation? An article from Carnegie Mellon University addresses these ideas [3] and, when provided to students, can scaffold rich conversations and discussions.

To ensure the project is inclusive, no music theory knowledge is required. Additionally, students do not need to listen to the audio output in order to successfully complete the project. The starter files include support for running the project either with music as the output or with a vector of note pitch and duration values. This inclusive design reflects principles from culturally responsive and sustaining computing education, which emphasize valuing students' identities, experiences, and cultural assets while removing unnecessary barriers to participation [2].

2 Project Overview

At a high level, the students write code that runs a genetic algorithm on a general tree where each node is a motif (5 music notes). The tree starts with 1 node and then grows

ACM EngageCSEdu, July 2026.

ACM ISBN 979-8-4007-2917-1
https://doi.org/10.1145/3819592

and shrinks across many generations. A generation consists of:

- Selection - Nodes in the tree are selected based on random chance and how they rate in musical fitness (how melodic the notes sound). This fitness score is calculated for each node with a provided function that takes into account simple music theory. Students are free to adjust this function.
- Reproduction - The selected nodes produce children nodes with a slightly mutated motif.
- Pruning - If the tree is too big, the nodes with the lowest fitness score are removed.

After a set number of generations run, the tree is pruned down to its top 5 nodes. These nodes are then traversed recursively to produce a melody. This traversal algorithm relies heavily on the current structure of the tree and children-parent relationships which creates a unique and interesting melody.

The nodes of the tree are defined by the MotifNode class. Each node has a motif (series of 5 notes), fitness score (how "good" the notes sound together), a vector of children (MotifNode pointers), and a MotifNode pointer to its parent (if it is not the root). The MusicalTree class manages and keeps track of the MotifNodes on the heap. The MusicalTree class will also have functions that select nodes, reproduce from selected nodes, prune the tree, and more to generate the final melody.

3 Recommendations

Musical Trees was the last project of the semester I gave students in their second computer science course. They had already learned the basics of C++, pointers, memory management, and linked lists. To prepare them for the project, I gave two lectures on trees. The first on general trees, and the other on binary and binary search trees. These lectures included active learning and were followed with short homework assignments.

The students had two weeks for the full project, with part 1 due after week one. The first part involved implementing the MotifNode class, and the MusicalTree constructor, copy constructor, assignment operator, destructor, and the function MusicalTree::PruneNodes(). The second part involved completing the rest of the MusicalTree class to run the full genetic algorithm and produce the final output. The hardest and most rewarding part for the students was the MusicalTree::PruneNodes() function. This required an in depth understanding of pointers and how to manipulate them. Writing the rest of the MusicalTree class also involved challenges but this time around designing their code as I provided almost no starting structure. To help students think about the MusicalTree::PruneNodes() function I recommend giving them the included worksheet called "prune-activity.pdf". This worksheet provides an activity to help student think about the MusicalTree::PruneNodes()'s algorithm before diving into the code.

The project's difficulty is adjustable by changing the amount of starter code. I provided a lot of structure for the MotifNode class to set them up for success for the open ended MusicalTrees class. For example, I declared all the member functions needed for the MotifNode class and described what each function should do. I also encouraged them to do memory checks within each of these function to alert them when they had an invalid pointer. This was specifically helpful for when they wrote MusicalTrees::PruneNodes() by helping them avoid hard to debug pitfalls. I provided little to no structure for the MusicalTrees class. For my students, this was a good level of structure and freedom across two weeks.

To make the project more challenging, you could remove the structure and guidance for the MotifNode class. To make it easier and shorter, you could add more structure to the MusicalTrees class by providing the function declarations and descriptions. Another way to make the project easier would be to change the requirements for MusicalTrees::PruneNodes(). Currently, it requires a very specific algorithm which involves rotating nodes. I could imagine a simpler implementation would also suffice. However, my students, with no background in trees rotations, did well and enjoyed the challenge.

As an optional extension, I also gave students the option of adjusting the function that calculates the fitness score for each node. What sounds better or worse, aka which node has a higher or lower fitness score, is subjective! I provide an implementation based on some simple music theory ideas but there is lots of room for students to play and explore.

For a more open-ended and reflective assessment I have included a worksheet, "reflection.pdf." It focuses on the process students followed to create their program and their reflections on its output. The second page of the worksheet asks students to read an article from Carnegie Mellon University, "As AI-Generated Music Advances, Humans Still Lead in Creativity, CMU Research Finds" [3], which accessibly discusses the considerations surrounding AI-generated music. The worksheet then poses three questions about the article, each requiring a progressively deeper level of engagement.

4 Technical Considerations

In the starter code, I provide two driver files that can be used to run and test the code. Both run the algorithm the students write that generates a melody (a vector of notes). One, "driver.cc", then calls the provided LAME library code

to create an audio file (wav). In contrast, the driver file "driver_no_music.cc" prints out the vector of notes to the standard output. The autograder (provided) doesn't require the audio file. It checks a log file that I require student's code to produce along with the final vector of notes. This is all to ensure everyone has equal access to running and testing the project regardless of their ability to hear music or run the lame library. Grading from a log file also allows students to have design freedom both with the music theory and the code structure.

This project depends on C++20 and the LAME library. Additionally, tools such as AddressSanitizer are useful for helping students identify memory errors during development. At our institution, we support students working across multiple operating systems (Windows, macOS, etc.) by using Docker to provide a consistent Linux-based development environment. Students write code locally using VS Code, while Docker ensures that all required dependencies and tools are pre-configured and identical across machines.

We dedicate time early in the semester to help students set up Docker and VS Code, troubleshoot issues, and ensure everyone can successfully run the development environment. In our experience, this approach significantly reduces environment-related issues later in the course and provides a reliable, consistent setup for all students. I have included the setup tutorial (docker-setup.pdf), the Dockerfile, and starter code with pre-configured files to integrate with the Docker setup. If this is not your preferred method I recommend the following alternative options:

- **Institution-provided virtual machines:** Students can SSH into a centrally managed environment with all dependencies preinstalled.
- **Local setup:** Students install C++20 and the LAME library directly on their own machines. While feasible, this can introduce significant variability and setup challenges.
- **Music creation website:** You could host a website and server that takes in text input of a vector of notes and returns the audio file. This avoids requiring students to install the LAME library, but it does require maintaining a server that runs LAME or a similar tool.
- **Python-based version:** The project could be adapted to Python, simplifying setup but removing opportunities for students to engage with memory management concepts, which are valuable learning experiences in C++. Python is generally easier to get working across multiple machines.

5 Materials

Provided files students don't need to edit. All file paths are in relation to the student-facing/mt-starter directory.

- **includes/catch.hpp** : A file for testing using the catch2 framework.
- **tests/tests.cc** : Tests for part 1 of the project.
- **includes/dr_wav.h** : A file for music generation.
- **includes/utilities.hpp and src/utitlities.cc** : Files for music generating.
- **src/driver.cc** : A main function that runs the musical tree algorithm and produces an audio file.
- **src/driver_no_music.cc** : A main function that runs the musical tree algorithm and produces text output.
- **Makefile** : A file to help students run and test their code.
- **output_example.txt** : An example of the expected output log file their code should produce.
- **print_statmenet_reference.txt** : C++ cout statements to help with spacing/spelling/formatting of the the log file output.
- **output_samples_classic/** : A directory of piano audio files at different durations and pitches to make the output audio produce a series of piano sounds.
- **../student-spec.html** : The instructions given to students.
- **../docker-setup.pdf** : The instructions given to students for setting up Docker.
- **.devcontainer** : An example setup file for VS code with a dev container.
- **.vscode** : An example setup file used for VS code.

Files that students will edit. All file paths are in relation to the student-facing/mt-starter directory.

- **includes/motif_node.hpp and src/motif_node.cc** : Files that Define and declare the MotifNode class which is used as the nodes in the musical tree.
- **includes/musical_tree.hpp and src/musical_tree.cc** : Files that define and declare the MusicalTree class which runs the genetic algorithm and maintains a general tree of MotifNodes.

Optional worksheets that can be provided to students.

- **student-facing/worksheets/prune-activity.pdf** : A worksheet that provides an activity for students to help them prepare to code MusicalTree::PruneNodes()
- **student-facing/reflection.pdf** : A worksheet for after the coding portion to help students reflect on the project and the societal implications.

Files not given to students.

- **instructor-only/autograder-files/musical-trees-part1-catch.cc** : catch2 framework tests for the first part of the project.
- **instructor-only/autograder-files/musical-trees-part2.py** : A python script that grades and provides feedback

from the log output the students code generates for the second part of the project.

- **instructor-only/Dockerfile** : The docker file that can be used to create an image for students to use and run the project.

6 Citations and References

References

[1] [n. d.]. *National Center for Women Information Technology.* Retrieved January, 2026 from https://ncwit.org/engagement-practices-framework/

[2] Kapor Center. 2021. Culturally responsive-sustaining computer science education: A framework. *online document* (2021).

[3] Stacey Federoff. 2026. As AI-Generated Music Advances, Humans Still Lead in Creativity, CMU Research Finds. https://www.cmu.edu/news/stories/archives/2026/january/as-ai-generated-music-advances-humans-still-lead-in-creativity-cmu-research-finds