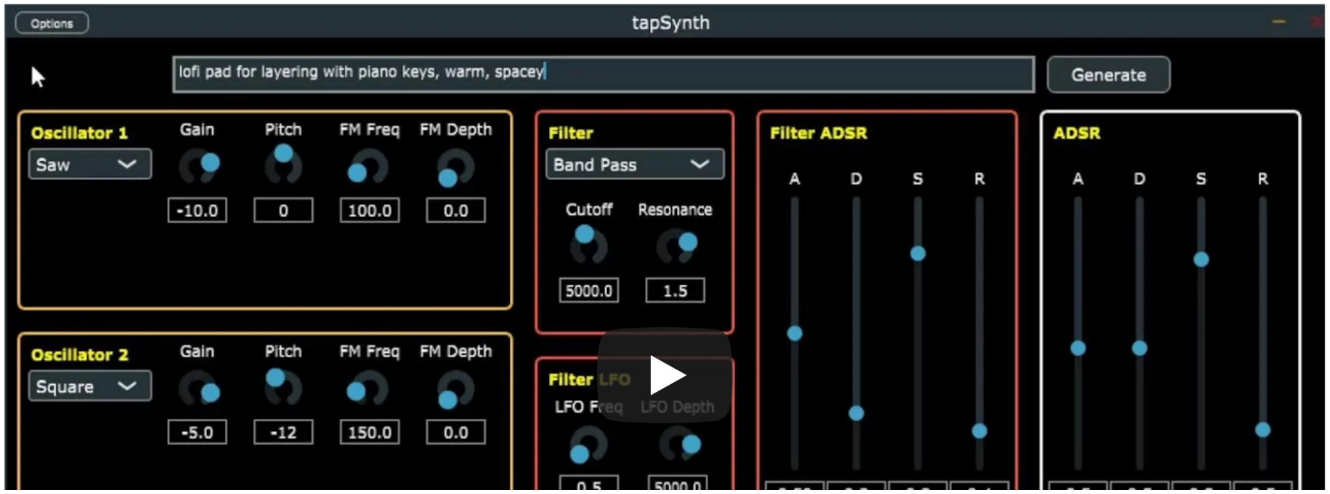
AI Powered Virtual Synthesizer

Built a virtual synthesizer with built in text-to-preset generation software. Wrote a Python script to generate JSON presets using the OpenAI API and integrated it into a JUCE C++ synth plugin.

About

The intentions and applications of this project are best explained in this brief video:



lofi pad for layering with piano keys, warm, spacey

Oscillator 1: Saw — Gain -10.0, Pitch 0, FM Freq 100.0, FM Depth 0.0

Oscillator 2: Square — Gain -5.0, Pitch -12, FM Freq 150.0, FM Depth 0.0

Filter: Band Pass — Cutoff 5000.0, Resonance 1.5

Filter LFO: LFO Freq 0.5, LFO Depth 5000.0

Filter ADSR

ADSR

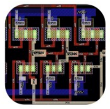
Technical Details

Prior to this project, I had no experience with plugin design or even C++, but I had done a few small projects using the OpenAI API. I found a video series from The Audio Programmer explaining how to build a simple synthesizer plugin. I used the videos to familiarize myself with C++ and the JUCE framework (a popular open-source framework for audio application development). I decided to just use The Audio Programmer's synth that he builds through the tutorial videos instead of following along to build my own, since my focus was on the AI component I wanted to add. In fact, that was the first thing I coded. I made a simple Python script with the OpenAI API to generate a JSON file with values for each parameter in the synth. I worked on this project in late 2023, so LLMs were not nearly as powerful as they are today, and the OpenAI API boasted far fewer features, so I had to brute force my way to getting proper JSON by simply stating it in the prompt (repeatedly, in all caps).

I then added the ability to load JSON files into the synth and set all the parameters based on the JSON — in other words loading presets. With this, I was able to perform manual unit tests on my preset generation script. The results were promising enough for a proof of concept, so I continued on to the final portion of the project: integrating my Python script into the C++ plugin.

This was by far the most difficult portion of the project, since I did not know a lick of C++ (at first, I thought the header files were in a different language since they ended with ".h"). I got around this with very crude vibe coding using GPT 3.5. At the time, vibe coding in the sense that we know it today was not at all viable, so my vibe coding in the sense that we know it today was not at all viable, so my development cycle was as follows: I had GPT generate code, used a separate instance of GPT as a tutor to teach me how the code worked, pasted the code into my project, and tried to compile. I'd run into an error, try to fix it myself, typically fail, then have GPT explain the bug and its fix, recompile, and repeat. Despite my heavy reliance on AI, I think that my process on this project was an extremely effective method of learning. In a short time frame, I was able to grasp the basics of C++, apply my existing knowledge into a new domain, and practice testing and presenting a product all without touching any boring lectures or courses. My goal with this project was not to learn C++, it was to build a tool that I believed was innovative, interesting, and useful to me as a music producer. In retrospect, I cannot think of a better way for me to have accomplished that with the resources that I had available.

PROJECTS



Ring VCO Design + Layout in Cadence Virtuoso

ALL

ALL PROJECTS