15-112 Project Proposal

Marco Cardenes
Andrew ID : mcardene

November 2023

1 Description

Synth Jumper (wt) is a platformer game based on the principals of Modular Synthesis which allows a playable character to patch together modules and explore the fundamentals of digital music. Your character can explore a virtual rack, turn knobs, carry connectors, and interact with the modules around it.

2 Similar Projects

When searching for other projects such as mine, the most prominent one is "The Signal State". It is an interactive modular synthesizer game which is a post-apocalyptic game where for some reason you need to link modules together to solve missions. It is not entirely clear how the game works without buying it, however it is clear that it is a level-based game, and does not feature a character on a series of platforms, as mine would.

There exist virtual modular synthesizers, some very are advanced, open source, and well maintained, and others are independent projects which look pretty but have fewer modules and capabilities. My project will also be limited in capability due to time, however it will be wholly interactive, more game-like, and will ultimately be more simple– less daunting to someone new to the wonders of modular synthesis.

3 Structural Plan

By the time this project is well underway, a different, more efficient structure is sure to reveal itself, but to begin, the project will be organized as follows. The main MVC functions and global constants will live in *main.py*, which will import the necessary objects from the other files local to the folder, as well as manage the threading with audio. These objects will be split into two sections, *Modules.py* and *GameObjects.py*. Within modules will be stored a parent Module object, which all of the respective modules (oscillator, filter, and others given time) will inherit. The parent will contain the framework required to make it compatible with an MVC game type – including separate drawing, stepping, and controlling methods. All subsequent children will have this capability and will modify these methods as necessary. In the game objects file, there will be a Player, floor, and module connector object definition. Given time, there will also be a graphics sub-folder which holds the sprites, backgrounds, and other essential UI images.

4 Algorithmic Plan

There are two trickiest parts of this project. The first being handling how audio is both delivered and calculated. Due to the fact that any given frequency will have a non-zero endpoint unless deliberately altered, allocating a fixed amount of samples and looping over them to play will not work. The correct method of approach is to make every modular output a generator method, such that it can be called infinitely, resulting in both mathematically pure samples and a maximum space required being number of (samples * number of modules) as opposed to number of (samples * sample rate * seconds * number of modules), where sample rate is about 44100. By allocating so little on the heap, we can quickly process our audio samples without having to resize our arrays for each sample. Each output of our modules will request the output of what came before, and since each module output will eventually return to an oscillator, we end our chain of calls there.

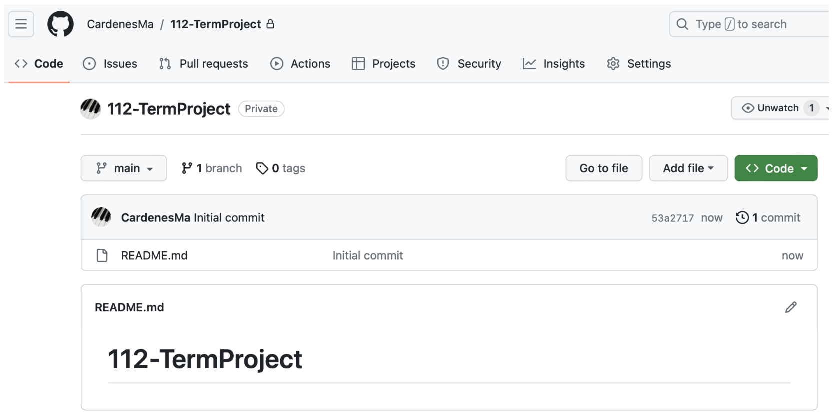
The second tricky part is understanding the player's movements and collisions, and efficiently manipulate the modules based on character movement. This is achieved by collecting all the collisions in the game by brute force, and then storing those collisions in a set. To make this more efficient, we will only calculate collisions if the player is moving (a key is pressed), or if the player is jumping. Otherwise, there is no way for new collisions to be made. Since all the objects which are collided will be hashed, it is easy to check containment.

5 Timeline Plan

The most important thing to build is the Modules and their chain to an ultimate mixer. Thus, I intend to have that built by T1. Once I am sure that the audio component works, I can begin working on the player mechanics and displaying the modules on the screen. The graphics will be rudimentary, and the audio will be running concurrently but without much grace in the audio processing when a frequency is changed. When T3 rolls around, I will have nicer graphics including a sprite, background, nicer floors, and there will also be an automatic limiter on the mixer so that we don't ruin our computers speakers by accidentally over blowing our 2**15-1 amplitude maximum.

6 Version Control

For version control I will be storing the project in a GitHub repo which I will make commits to from an extension in VScode. If needed I will use the command line to make changes but that should be unnecessary unless there is a strange conflict which needs to be resolved.



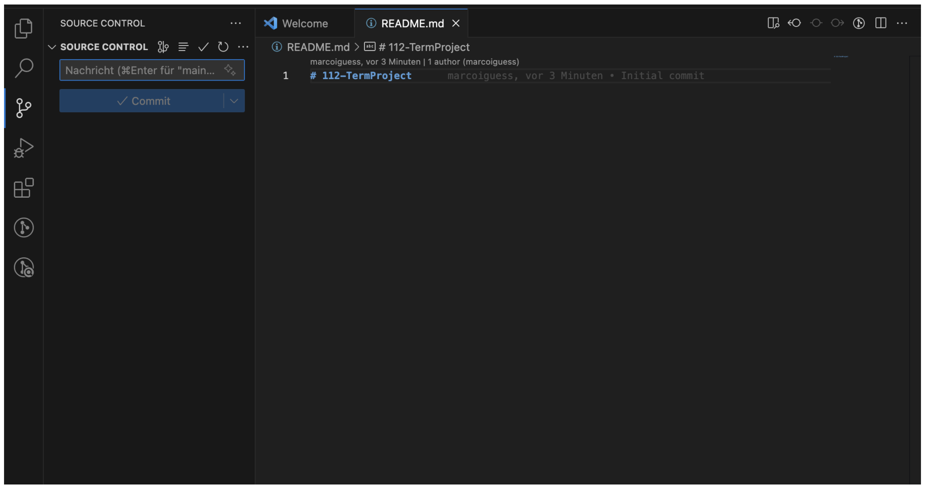
CardenesMa Initial commit 53a2717 now 1 commit

README.md	Initial commit	now

README.md

112-TermProject

Figure 1: GitHub Repository



marcoiguess, vor 3 Minuten | 1 author (marcoiguess)

```
# 112-TermProject      marcoiguess, vor 3 Minuten • Initial commit
```

Figure 2: VSCode Interface with Version Control Extension

7 Module List

I will require two external modules (besides cmu graphics) to make this project work, and one internal module. The external modules are 'numpy', which I will use to efficiently manage the audio sample data, and 'PyAudio', which I will use to crate the audio output stream. The choice of PyAudio over other modules is easy, for PyAudio has a low-level implementation which allows one to play directly to an audio port from a continuous stream, whereas other audio modules only allow for playing predetermined full audio samples which opens and closes the port, locking the stream from real-time manipulation. The internal module I will use is python's built-in Threading. Since the audio and graphics need to run concurrently and the refresh rate for audio and visuals are different, the audio output method needs to run in it's own thread. Since the user will be changing the output of the audio by interactions in the graphics window, it is important that just the audio output method is in a new thread, and the stream can be manipulated on the same thread as the graphics. A tech demo is supplemented.

8 TP1 Update

Since PyAudio and threading aren't allowed until MVP has been reached, the player will be manipulating the visuals of the data outputted by the mixer. There will also be different levels of goals to achieve, increasing in complexity as the player determines the correct controls to create the wave given. This means also that the modules have to be saved, drawn, and created for each level. This allows the entirety of the screen events to live in one wrapper which controls the game. Ideally this runs in a new window, but we'll see about how cmu graphics handles that as the project progresses.

9 TP2 Update

For the first few days of TP1-TP2 week, I was working toward making the project a visualizer of a modular synthesizer, which was graphics intensive and expensive for the program. On Wednesday, I was given approval to use the audio library. This changed my project architecture a great deal. It now uses an audio-cache based approach sending 256 (variable) in each packet. Lots of modifications were made to the UX and UI as well. A player can now use the connectors properly and they will update when disconnected correctly. Player can now manipulate the modules on the screen by adding them and cycling through which to add. The conenctors now have the capability of changing the inputs of other modules. The player also is a stick figure guy that changes based on the direction he's headed. The modules are also rounded on the corners.

10 TP3 Update

The major changes since TP2 is mostly in the documentation, integration of new modules, and everything being more robust. New modules include a Low pass filter, randomizer, sequencer, and now the filter algorithm works much better. The two main issues I'm facing are the speed of cmu-graphics drawing and abrupt changes to the audio when changing values which creates a popping sound. It is subtle enough that it is not worth overhauling the whole project this late in the game. A player can now also delete modules on the screen, and the connectors associated with the module will disappear and stop carrying data. File structure was slightly altered as well, distributing code to more appropriate file locations, as well as adding a requirements.txt file for installing the appropriate packages.