Music Generation - Functional Spec

Background

The problem we're trying to solve. Why it's important. How users will benefit.

Fundamentally, we want to leverage machine learning (ML) to provide a way for users to **generate custom music for their specific use cases**. See the section below, Users and Use Cases, for examples of these.

Users, of course, have options for obtaining music besides using ML to generate their own. However, upon further exploration, these options aren't as workable as they might at first appear.

- Using **copyrighted music** without permission is both unethical and illegal. Getting permission to use copyrighted music can be onerous and expensive, and in fact, might dead end with the copyright holder simply refusing to grant permission.

- Negotiating with **composers and musicians** to create custom music has the potential to be a costly--and perhaps more importantly--time-consuming process. Also, there is no guarantee that once the music is created it won't be encumbered by licenses that make it difficult to use beyond the initially specified use case.

- With the advent of **open licenses**, such as Creative Commons, a lot of unencumbered music is now available on the Internet. However, the main problem here is *discovery*: How does the user find music that is a good fit for their use case. Personal experience indicates that this is surprisingly difficult.

In contrast, what we offer is an experience where the **user provides a sample** of music, which is similar to what they seek. Our package uses this sample to generate a stream of music--of arbitrary length--that is **similar to the sample but free of licensing restrictions**. The user can then edit this music stream with standard audio editing software[1] to customize it for their use case.

[1] Audacity, Adobe Audition, Apple GarageBand

Users and Use Cases

Who will use your system? What level of computer experience do they require? What domain knowledge must they have?

The users of our package will be anyone who is interested in generating copyright-free music but don’t have the necessary knowledge in music theory or the technical skills required to build/train models to compose music. The following are some examples of our target users:

Podcast Producers
Podcast producers include musical elements in their podcasts to make them more engaging to their listeners. Using intro/outro and background music to move the story forward contributes to a professional sounding podcast. But obtaining license to music can be expensive and might not sound original, so producers can use our package to generate their own unique music. They will need to know basic Python knowledge to install and use our package.

Video Creators
Music is an extremely important element in video production. Video creators can generate music for their workout, cooking or traveling videos using our package and editing it to match the pace or mood of their videos. These users need to know the basics of Python to install and use our package.

Music Enthusiasts
Music enthusiasts who dream of composing their own music but not knowing how to use instruments can generate music similar to their taste using our package. The computer experience they should have is basic Python to install and use our package.

User Experience (UX)

How users will interact with the system and how the system will respond.

The **easy_music_gen** package exposes the following constructor and method.

```
import easy_music_gen as emg
```

```
music_generator = emg.EMG( music_files )
```

music_files: list

A list of MIDI and Music XML files to use as samples. The `EMG.generate()` method generates music based on these samples. The list can include any number**(???)** of MIDI files, MusicXML files, or combination of these. The filetype is inferred from the extension: MID or MIDI for MIDI files, MXL for MusicXML files. If no argument is provided, the EMG object attempts to process all MIDI and MusicXML files in the current directory.

```
music_generator.generate( tempo, length, output_file,
output_format)
```

tempo: int

The tempo of the output music in beats per minute.

length: int

The length of the output music in seconds.

output_file: str

The path to the file where the output music will be written.

output_format: str

The format of the music output: “MID” for MIDI, “MXL” for MusicXML, and “WAV” for WAVE (PCM). If this parameter is omitted, the output format is inferred from the file extension specified in the **output_file** parameter.

Notes

The following functional specification from the Ax/Wx project looks as though it could provide inspiration for our functional specification.

https://github.com/rexthompson/axwx/blob/master/doc/FunctionalSpecification.md

Other projects that might be useful to review:

- https://github.com/khyatiparekh/Searching-for-Success/tree/master/Doc
- https://github.com/sliwhu/UWHousingTeam/tree/master/doc