Wrock

Julia Cooper, Marilyn Sun, Gabriella Bova

The Project

Create a soundtrack for a story given as input, based on its changing sentimental tone.

Minimum Deliverable

- Program is provided with hardcoded short musical pieces for different moods (Neutral, Happy, Sad, Angry, Excited) and riffs for different characters
- The program analyzes the text by paragraph. It figures out the emotion of each section, and adds character riffs for characters who are present in that section (connected with that section's emotion)
- Then, the program takes the given music and pieces it together to create a continuous soundtrack that can be played in the background of reading the story

Maximum Deliverable

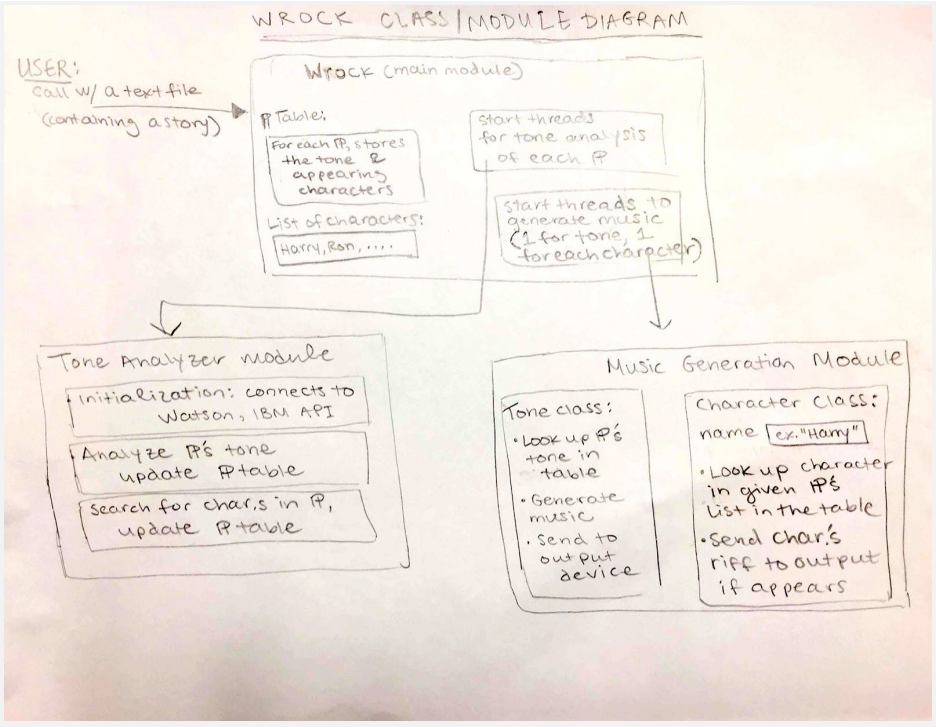
- Program can analyze a long story with many characters, like Harry Potter, and create a song on the fly to play in real time.
- Program is not supplied with predetermined musical pieces for different moods, but generates the music from scratch based on the tone of each section, using multiple instruments.
- Program can save the generated soundtrack in the form of a piece of music file.
- Program can create smooth transitions between moods

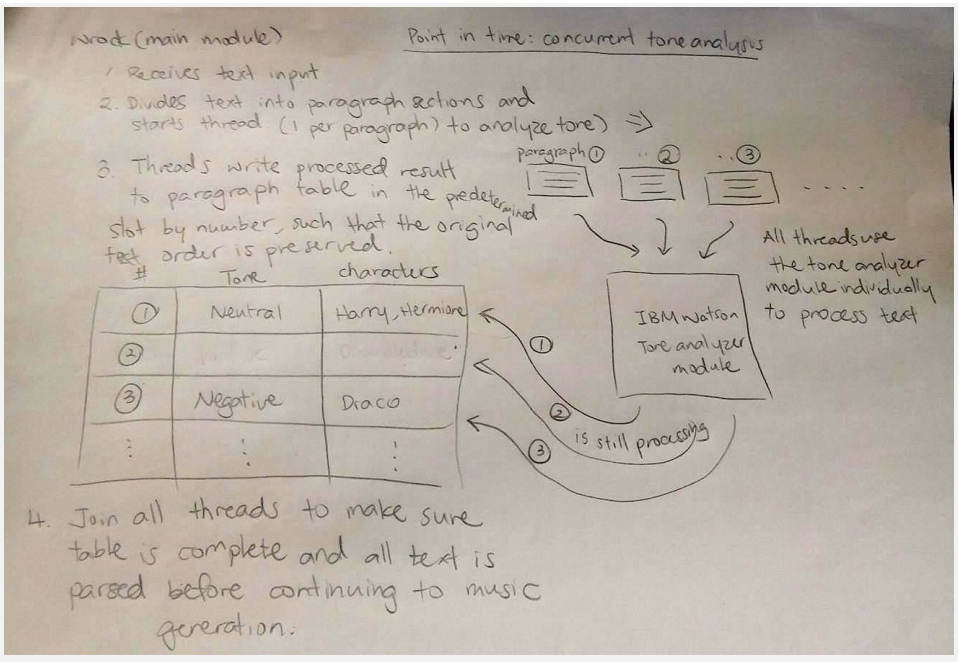
Significant Changes Made In Our Design

1. We decided to change python audio libraries halfway through our projected timeline. Originally we were using pyo, but the we had difficulty understanding the documentation and getting it to work well enough to play multiple musical samples in an order. We switched to Python's pyglet library, which had clearer tutorials and was more effective for the purpose of our project.
2. We experimented with different ways to get multiple pyglet threads to play music. At first we had one media player that each thread could grab the lock for and queue music to. But this way was sequential, only one piece of music could play at a time.
    a. To get multiple music pieces playing simultaneously, we first tried a method of using barriers. Each thread had its own media player and played music in real-time, and a barrier made sure that the threads were in sync with which paragraph they were looking at, so no one thread jumped ahead. However, this method produced errors with the audio driver on our computers.

b. Finally, we gave each thread its own media player. Each thread queues music to its player as it reads through the paragraph table, but does not play in real time. All of the music pieces are approximately the same length (10 seconds) and if a player is not supposed to play during a specific time, it queues 10 seconds of silence, so all music players stay in sync. Once all threads finish, a pyglet PlayerGroup structure plays all of the media players simultaneously.

Diagrams:





Analysis of Project Outcome

We did reach our minimum deliverable because the program is provided with short musical pieces for different moods and characters, it analyzes every paragraph, and chooses what riffs to play for each tone. We achieved one of the goals of our maximum deliverable: the ability to analyze a long story with many characters (*Harry Potter*).

Reflection on Project Design

What was the best decision that you made?

- Switching from pyo to pyglet was a good decision because pyglet was easier to use and offered the ability to create and play music in the way that we wanted.
- We divided the program into three parts: tone analysis, music generation, and a coordinator. This made it easier to complete the project, debug, and improve.

What would you do differently next time?

- Find a way to keep the login credentials for IBM Watson hidden from the user. Currently they are hard-coded into our program, and we would like to find a more secure way to connect to the tone analyzer.
- Experiment with a lot of music libraries early on, and switch libraries sooner if it was too difficult to understand and achieve our goals with the one we were working with
- Hopefully there is a better way to construct the base units of a piece of music. In our project, we used pre-recorded wav files of instrumental music or object sounds such as an owl hooting. Then, we layered multiple wav files to generate music. This was tedious because it required us to listen to different samples from online and manually choose which samples would sound good together. If the base units were individual notes, however, we might've been able to more efficiently and accurately determine which notes sound well together using basic music theory. Of course, a minor tradeoff would be potentially generating less complex and rich sounding music.

Division of Labor

We were able to split our team into two groups: one working on the tone analyzer and the other working on the music generation module. We worked individually and then built the coordinator together and integrated our work.

Bug Report

Our most difficult bugs involved using the music library incorrectly, or in a way that was not supported on the system we were working on. For example:

- The pyglet media players did not support mp3 or compressed music files. We had to figure out which music file formats it did support (uncompressed wav files) from reading the pyglet documentation, and convert all of our files to that format. Solving this bug did not take too long, as we just had to change our file format, but in retrospect, we would

have saved time if we had researched which music file formats pyglet could and could not handle before downloading our music files.

- After coding our music generator to play music on multiple threads using a barrier, we came across errors with playing multiple pyglet players at the same time- the audio driver could not handle this. In order to figure out why our first solution to play music simultaneously was not working, we first tested our mutex barrier without using any pyglet constructs, to make sure the error was not rooted in a race condition or other issue with our concurrency. These tests proved that our barrier itself was working, so we deduced that the error was how we were using the pyglet music players. We could not successfully play these simultaneously in real time, so to solve the problem, we decided to change our method of playing music concurrently, as described above. This bug took a day to notice, test, and figure out that we had to change our solution.
- To find these bugs faster, we could have read the pyglet documentation more in depth, and researched different methods of playing music concurrently before choosing one to code.

Code Overview

wrock.py

- This is the main module. It coordinates the entire program. It opens the story file and ensures there is a character file in the current directory. It collects data about the story's tone and characters, and stores that information in PTable, so the soundtrack can be created.
- PTable: a data structure that stores the tone information of the input by paragraph. Each paragraph has an entry in the table in the form of a tuple (general tone, [list of characters appeared]). The music threads use the information stored in the table to determine what kind of music to generate.

tone_alayzer.py

- This module wraps the Watson API so that the main module can call this set of functions, instead of interacting with the API directly.
- *analyze_tone* takes a paragraph as input and calls the Watson API to analyze the text. *extract_tone* takes the response data from the Watson API call, and returns the best matching tone for the analyzed text. *extract_characters* checks for the appearance of characters.

musicGenerator.py

- This module is a class that can play music on multiple threads
- It is initialized with several pieces of data: the first is a dictionary filled with information acquired from tone analyzing- it maps paragraph numbers to tuples of a tone and a list of characters that appear in that paragraph. This is stored in the class as self.PTable. The second is a number which represents the maximum amount of time the user wants music to play for (ex. 100 seconds). The third and fourth are respectively the names of

files containing a list of characters/objects and paths to their music files, and a list of tones and path to their music files (ex. Harry: objects/harry.wav).
- The *musicGenerator* reads both of these files and creates 2 dictionaries, one mapping character names to pre-loaded music pieces, and one mapping tones to their music.
- The *start* function then starts and coordinates threads to read through the PTable and figure out what music should be played. One thread reads through the PTable and queues a series of tone music pieces to a pyglet media player, and the other threads read through looking for occurrences of a specific character in the table
- Finally, a structure called pgroup (a pyglet PlayerGroup) is used to join all of the players and play the overlapped music.

Instructions on Running the Code

First, download the following folder of our music files:

https://github.com/gbova/wrock/objects.git

To run the program, type the following command into your terminal, in the wrock directory:

> *python wrock.py text_to_analyze.txt*

You will need python's *pyglet* and *request* libraries installed.
You will need these text files in the current directory:

- *text_to_analyze.txt*: This is a command line argument, so you may call it whatever you'd like. This text file should have each paragraph on its own line. (In other words, there should be a new line character *only* at the start of a new paragraph)
    - We have provided a couple of example passages to analyze, called *harrypotter2.txt, quidditch.txt* and *werewolf.txt*
- *characters.txt*: Lists the characters in the story that will have a riff during the piece, and the corresponding .wav file for this riff
- *tones.txt*: Lists the tones that will be encountered during the story, and the corresponding .wav file to play for that tone
    - The possible tones are: analytical, confident, neutral, tentative, joy, sadness, fear, anger

You will also need the following folders containing uncompressed wav files that the music generator samples from:

- *objects*: contains default music samples used in demos
- To customize music: Users can add more music samples to Wrock by downloading uncompressed wav files, and entering the new character/object and the file path of the music sample in characters.txt.

To download all of our code, text files, and music files, clone the following folder:

https://github.com/gbova/wrock.git