APPLICATION OVERVIEW

OBJECTIVE

The Beat Detektor application is designed to function as interactive visualizer for user chosen audio files. Using various signal processing techniques the program analyzes the desired audio file and creates a set of data to generate visuals and gameplay elements in synch with audio playback. The programming environment of Processing was selected for its strong support for graphics programming and integrated audio codec libraries.

APPLICATION DETAILS

Audio Analysis

The first step in creating an interactive visualization experience for audio files is of course, to analyze the audio. While real time processing would be ultimately desirable, due to design goals and limits on the available codec libraries, our application first loads and analyzes the chosen file as a whole, and then plays the file afterwards, making references to the processed data via temporal indexing. The audio analysis itself has a variety of steps, listed in detail below:

- **Chunking and Fast Fourier Transforms -** The initial step in the process, audio data is separated into amplitude sample 'chunks', with a size of 1024 samples per chunk. Fast Fourier Transforms, given the source audio sample rate, are run on each of these separate chunks to provide a frequency mapping for each chunk and this data is stored. From this data we can, at any given time, access the intensity (amplitude) of individual frequency ranges in the signal's spectrum.
- **Calculate Spectral Flux -** Next, the application uses these spectrum values and compares each chunk of time's spectrum to the following spectrum element. By calculating the difference between each frequency in the spectrum pair, we determine the overall amount of spectral flux from time index to time index. This provides us with a useful aspect of the audio for determining the onset points of various notes and instrumental elements in the audio. By filtering our flux calculation to only worry about positive flux, we refine the data to ignore changes in signal irrelevant to our goal.
- **Determine Appropriate Threshold Levels -** Given pre-determined values for number of time windows and multiplier based on best average results in testing, the flux values are averaged against their neighbouring flux values within the windowed range and averaged together. When multiplied by the decided value, this allows us to set a threshold value for every time index in order to aid in identifying flux values as onsets. These thresholds are then stored along with the other values.
- **Create Limited Flux Map -** Iterating through the time windows of the signal, the stored spectral flux values are compared against the associated threshold values, and replaced with zeros if they fall under the threshold. Without this thresholding, echoing signals in the flux would all be detected as onsets. Using averaging around large onset spikes allows such noise to be later filtered out.

- **Finalize Peak Values -** The final major processing step iterates over the limited flux set and compares values to their adjacent values. If an item is larger than its previous and following flux, then it is a peak flux, and will remain in the set. Otherwise, the element is set to 0. This isolates peaks and removes increasing and decreasing slopes that survived thresholding. What remains is the closest approximation to major instrumental onsets and beats without much deeper analysis such as starting with different frequency subsets and comparing the results, applying pattern recognition, and the like.
- **Relative Amplitude -** The final step in our program's analysis creates a secondary set to use for determining visual elements. The source spectrum values are analyzed and an array of relative amplitude values scaled from 0 to the highest amplitude point is created. Thus, at any given point we can look up a decent guess of how loud/intense a part of the audio may be in relation to the whole track.

Visual Generation

Given the analyzed data in the preceding step, the next element of the application uses this data to generate unique visuals and gameplay elements in synch with playback of the audio file.

- **Flux Map, Timer Bar, Flux Tracker -** The most basic element simply shows a graphed version of the entire file's flux in the background of the screen, along with a vertical timer bar that progresses from left to right across the screen as the audio plays. Beat events should synch to the timer bar hitting high peaks of flux. This also provides a user with the sense of the overall progression of the audio and the current time in file playback. The flux tracker is a thin horizontal red line that raises and lowers to match the current flux strength. At any given time, the intersection of the flux tracker and timer bar should match a point on the flux map.
- **Beat Balls -** The core 'gameplay' element of the application, Beat Balls are generated on the right edge of the screen as playback hits major beat points identified by the finalized peak values. These values themselves are used to determine the size of the ball generated. These elements travel from right to left on the screen once generated and are removed once they go offscreen.
- **Background Colour Phasing -** The background colour of the application changes along with shifts in the source audio file. On beats, the background will gain intensity in its blue channel, much like the beat balls, having a stronger effect the stronger the identified flux. The relative amplitude calculations are used to shift a red/green balance in the background, with 'louder' portions shifting towards green, and 'quieter' portions shifting towards red.
- **Particle Storm -** At the top of the field of play are three particle emitters. While modifying their intensity of production mid file proved a bit too challenging (and demanding on the processor!), the colour of their emitted particles changes much in the same way as the background, with blue value added for beats, but slightly differentiated in that red and green values stay the same, providing a yellow aspect to the colour. This results in the particles being largely shades of black-yellow, with blue-green mixed in on beats.
- **Ship Waveform Jets -** The player ship is given waveforms emitting from its rear (left side) stretching to the left edge of the screen. These waveforms represent the left and right channels of the currently playing audio as a final visualization element.

Gameplay

Gameplay and UI elements of the application are designed to be relatively straightforward, and controlled via the keyboard, as detailed below:

- **File Selection** - From the main screen, hitting O will open a JFileChooser allowing the user to select any audio file they wish to attempt play on.
- **Particle Intensity** - Using the number keys 1-8, users can adjust the number of particles that appear onscreen in order to reduce the load on their machines to enable smoother gameplay.
- **Play** - A default file is already selected without loading to play if no file is chosen, but regardless simply pressing 'S' to start on the main screen will launch analysis followed by gameplay.
- **Player Movement** - The player controls a spaceship on the field of play, moved onscreen by the arrow keys. The ship cannot exit the screen/field of play.
- **Scoring** - Scoring for now is a very simple counter that increments as gameplay goes on. Every time the player ship hits a Beat Ball, 500 points are subtracted from the player's score, and the screen will flash bright red to indicate this.

DEVELOPMENT PROCESS

What Worked

- **GitHub -** As usual Git was extremely useful in facilitating team programming work remotely, as well as providing major version control points to revert to if anything went wrong in coding.
- **Concept -** Our team felt that our concept was a relatively strong foundation to build a project off of. The variable scope in our proposal from core Beat Detection + Visualization to adding gameplay elements given time and ability allowed us to expand on our idea in ways we may not otherwise have given more limiting scope, and provided a chance to make a more interesting feeling project.
- **Multimedia Elements -** Given that our application provides both audio processing + analysis, as well as graphical display components and realtime gameplay aspects, we feel that the project is a good showcase of an understanding of multimedia programming.
- **Development Environment -** Processing provided a simple to comprehend well documented language for both team members to understand and work in, with good library support that helped avoid having to dig *too* deep into messy codec or graphic library details.

What Didn't Work

- **Codecs -** While the application has good support for most mp3 files tested, there remain a variety of codecs our development environment and thus application could not quite handle.
- **Real Time Processing -** As mentioned in an earlier section, we really wanted to do real time analysis, but it just did not prove possible with acceptable lag levels given some of the ways we needed to analyze values. This

results in very long audio files requiring far more work, as the entire track's information needs to be pre-stored in memory.

- **Complex Gameplay -** Given the memory loads and processing already on the system in a Java based environment, without a specialized gameplay engine to manage some more complex elements more efficiently, we were unable to implement some of our more ambitious gameplay ideas.

What To Improve

- **Beat Detection -** The core of our application, while still something we are proud of accomplishing, could still likely be refined to be even better with further understanding of the underlying math and longer analysis time. Using FFTs to isolate specific octaves or notes, comparing averages and flux values, and pattern recognition could all be potentially used to track beats from specific instrument types, (bass guitar, drum hi-hats) separately and produce different actions for each of their charts. Unfortunately, the math behind that and understanding of specific instrument timbres were not something we could achieve in the given time.
- **Visuals -** Given more time, we would adjust values of colours in game to better match various psychological colour theories to provide more interesting/dynamic colour ranges.