Rhythm Genesis

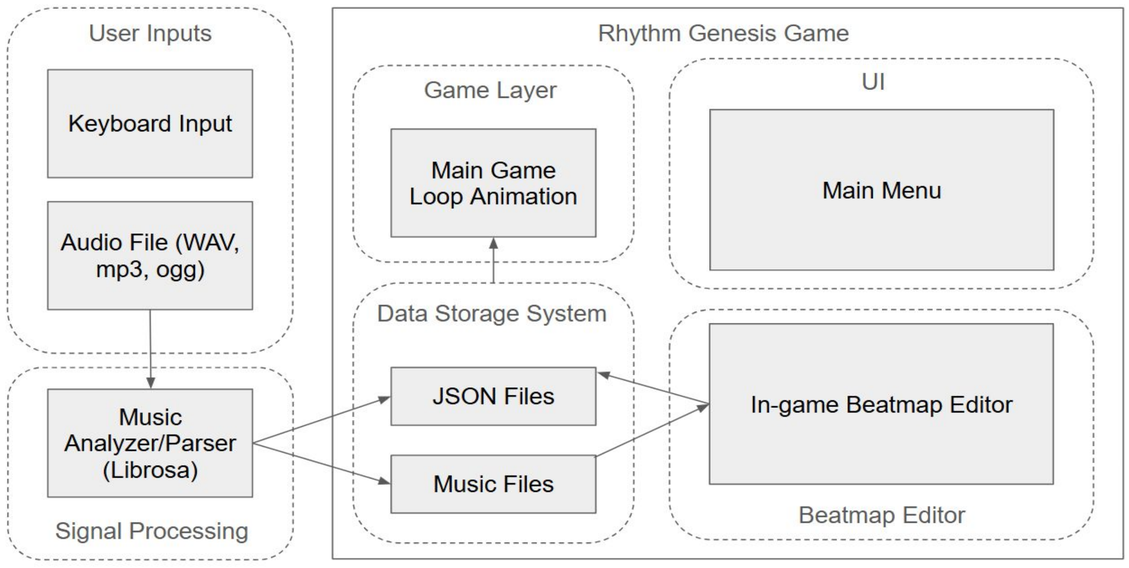
D6: Yuhe Ma, Lucas Storm, Michelle Bryson
18-500 Capstone Design, Spring 2025
Electrical and Computer Engineering Department
Carnegie Mellon University

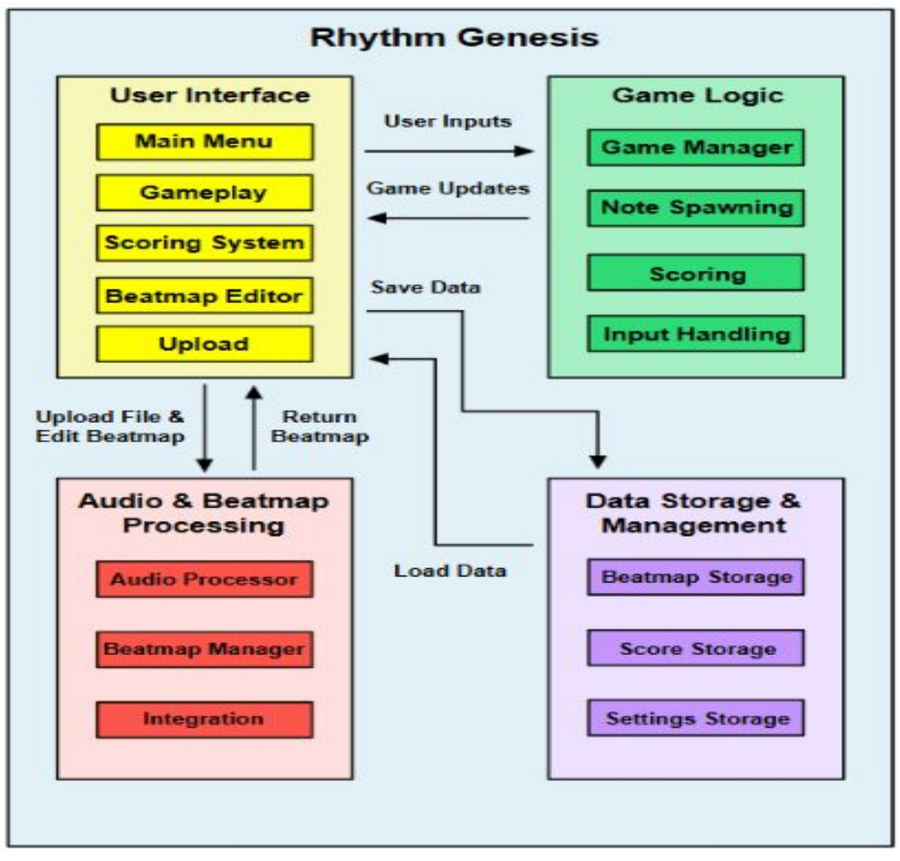
Product Pitch

Rhythm Genesis is a rhythm game for desktop in which players can upload their own songs for a fully customizable experience. The system analyzes the audio to detect the rhythm and generate a corresponding beat map for the game. Our audio processing algorithm, when run on single instrument songs between 50 and 220 BPM, is able to detect the rhythm with over 90% accuracy. The beat map is then used to generate falling tiles that are synchronized with the rhythm of the uploaded song with an animation quality of 30 frames per second. The player presses designated keys in time with when the tiles reach a bar at the bottom of the screen, according to the beat, and receives a score based on how accurately they aligned with the rhythm.

The player can further customize the game using a beat map editor. The target user of the beat map editor is someone who has a programming and/or music background. The editor, which features a waveform viewer and audio playback, allows the player to refine the auto-generated beat map or create a beat map from scratch by adding or removing tuples of the timestamp of the note and the lane number from which the tile should fall. This is especially well-suited for multi-instrumental or vocal tracks, which is out of the scope of the rhythm detection algorithm.

System Architecture





Conclusions & Additional Information

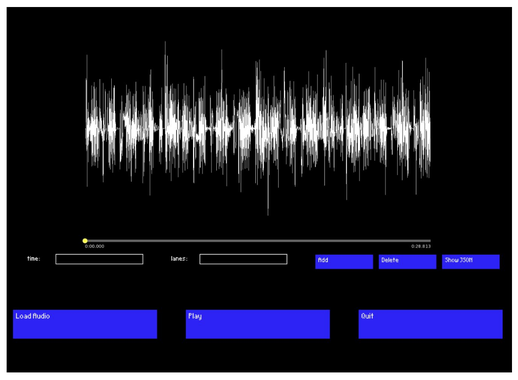
As is the case with most projects, we had some challenges and setbacks - the main one being a shift in engine about halfway through the semester from Unity to our own engine that was far more lightweight and a lot faster, but with fewer capabilities and generally more difficult bugs to work through.

Future work could include varying levels of audio analysis to allow the game to handle a wider range of music. For example, the user could choose to include vocal separation for pop songs, meaning that only the rhythm of the voice part would generate tiles, or use a sliding window algorithm for songs that have wide ranges in dynamics. We could also add different difficulty levels to the same songs by increasing the threshold for note detection to make for an easier version of the game with less tiles. Additionally, we could create better UI and graphics perhaps using a slightly heavier engine.

System Description

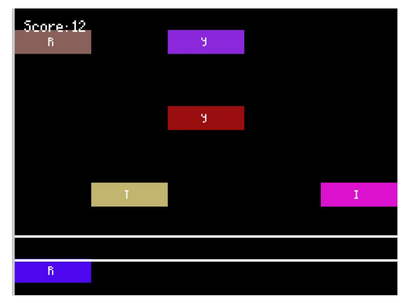
Our system is entirety software based, and as such we don't have any physical devices to show or describe. On the software side, our system is primarily built around our audio processing system, which incorporates Librosa - a Python package that assists in audio analysis - in order to detect beat onsets.

As for the rest of the system, the game itself is built on our own rudimentary engine implemented in C++ that uses SFML, a free open source library that assists in 2D graphics creation as well as handling audio and file navigation.

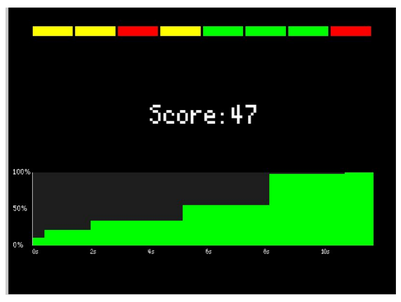


In-Game Beat Map Editor

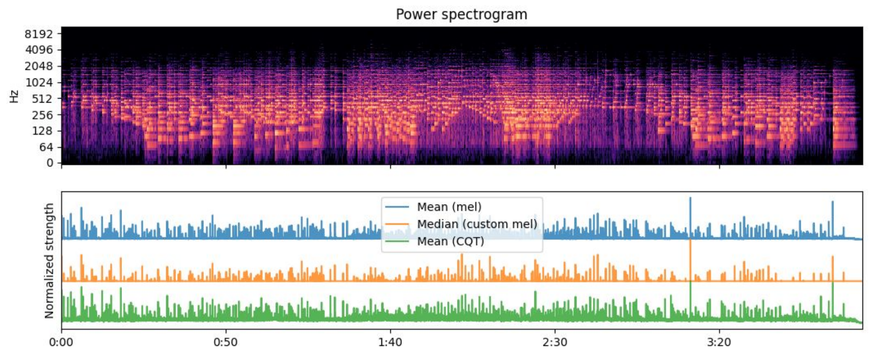
Main Menu



Core Game Loop



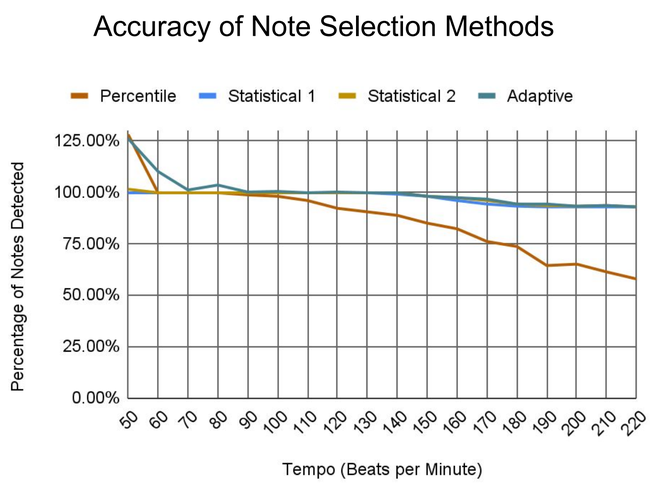
Scoring Page



Onset Strength Envelope of Debussy's Clair de Lune

Rhythm detection is implemented by computing a spectral flux onset strength envelope and selecting the timestamps with the highest peaks, as they correspond to the onset of notes. Various selection techniques were explored as shown below in System Evaluation.

System Evaluation



To test the rhythm detection, we composed short pieces as unit tests and manipulated variables such as the instrument, dynamics, tempo, and articulation. We compared the generated beat map to the ground truth timestamps. We also tested with real songs and aurally verified the rhythm detection accuracy. We used logging to measure to latency of UI interactions. For the beatmap editor, we confirmed that there was synchronization between the waveform viewer, progress bar, and audio playback.

Use-Case Requirements:

Metric	Target	Actual
UI Responsiveness of Input Events and Actions	< 50 ms	5ms
Save/Load Performance for ≤ 7 Minute Songs	< 5 s	4 s
Animation Performance	>= 30 FPS	30 FPS
Beat Map Generation Latency for ≤ 7 Minute Songs	< 10 s	4.5 s
Rhythm Detection Error Rate: False Negatives	< 10%	2.88%
Rhythm Detection Error Rate: False Positives	< 1%	0.05%

https://course.ece.cmu.edu/~ece500/projects/s25-teamd6/

Electrical & Computer ENGINEERING
Carnegie Mellon