Proceedings of the International Conference IMCS-60, MSU
October 10–13, 2024, Chisinau, Republic of Moldova

Automating Rhythm Game Level Design in Unity Through Beat Detection

Cosmin Irimia, Stefana Biceadă

Abstract

Technology is integral to our lives, impacting everything from daily tasks to critical fields like healthcare. While computers excel at technical tasks, translating intuitive concepts like rhythm into computer logic can be challenging. Detecting rhythm is natural for humans, as seen in activities like tapping or nodding to music. Over time, various beat detection algorithms have been developed for applications such as audio visualization and cardiology, where they monitor heartbeats and rhythms to identify diseases.

Keywords: Game development, sound processing, content generation.

1 Introduction

Rhythm games have gained popularity due to their skill-based gameplay, catchy songs, and engaging visuals. Well-known games in this genre offer large libraries of familiar songs, enhancing their appeal. These games require players to interact with events synchronized to the music's rhythm, with varying difficulty levels. While levels are hand-made, offering polished results, this method limits players to the game's song library and slows updates due to the time-consuming level development process.

Rock songs have simpler rhythms, while classical music presents more complex challenges. The goal of this project is to develop a rhythm game that allows users to upload any audio file to generate levels.

©2024 by Cosmin Irimia, Stefana Biceadă

2 Existing methods

This section aims to provide the general state of the field at the current time, by presenting already existing methods, as well as their strengths and weaknesses.

2.1 Offline versus Online Beat Detection

There are two primary methods for analyzing audio information: offline and online beat tracking. Offline beat tracking uses a pre-existing database to analyze musical events, allowing for detailed examination and fine-tuning, as it can account for patterns and the overall intensity of the music. In contrast, online beat tracking analyzes real-time audio input frame by frame, which can lead to less reliable results because it only has access to past events and must make immediate decisions. While offline tracking offers greater accuracy, online beat tracking is more suited for live performances, where real-time analysis is required. Both methods have distinct advantages and use cases.

2.2 Onset detection

Onset refers to the moment a musical sound becomes perceptible, such as note changes or drum hits. Onset detection algorithms focus on identifying sudden changes in amplitude to detect these events. Early approaches in the mid-1990s focused on monophonic audio, where only one note is played at a time. Over time, these algorithms have evolved to handle polyphonic music, where multiple instruments play simultaneously. Onset detection algorithms typically involve three steps: preprocessing to highlight relevant information, reducing the input with a reduction function, and peak detection to identify onsets. Variations of these algorithms have become more sophisticated over the years, incorporating psychoacoustic [6] principles like frequency resolution and pitch perception to closely mimic human auditory behavior. Klapuri's [2] work, integrating psychoacoustics, led to significant advances in onset detection and won the ISMIR 2004 tempo induction contest.

2.3 Rhythm pattern recognition

Another research topic that is worth discussing about, although it did not receive as much attention as the other approaches that were considered, is rhythm pattern recognition. This refers to the patterns in the rhythm of a song that are recurring in the form of arrangements of beats, and accents. While in theory the concepts of rhythm pattern and tempo are related and could result in similar outputs, the first one received less attention, most probably due to the complexity of creating datasets. Various methods have been developed to recognize rhythm patterns in an audio file. For example, [3] proposed a beat histogram that was derived from an autocorrelation function. Among other notable papers there is [4] in which a harmonic analysis of the rhythmic patterns is proposed.

2.4 Threshold determination methods

Another important aspect when it comes to accurately determining musical events is the determination of the threshold values. Establishing appropriate values can contribute to the accuracy, therefore multiple approaches have been presented.

2.4.1 Fixed Thresholding

This approach involves experimenting with different values and levels and observing the results in order to manually fine-tune the values. A clear disadvantage is its non-adaptiveness to changes in amplitude, resulting in low performance for songs that vary dynamically. Nevertheless, implementing a fixed threshold is the simplest approach being very easy to understand, and it may find its use case in songs with consistent volume and predictable dynamics.

2.4.2 Dynamic Thresholding

Dynamic Threshold: This method involves adaptively adjusting threshold levels based on a previous set of data in order to keep a specific

sensitivity. Although it is more difficult from an implementation point of view, it is more flexible and responsive to changes in the signal, resulting in better detection accuracy. Some things to take into consideration when implementing this method is the parameter tuning such as selecting the appropriate window size and scaling factors. Although dynamic thresholding can adapt to changes in signal amplitude, it may still be sensitive to noise. If the signal contains a lot of noise, the calculated thresholds might be effective, resulting in false detections.

2.4.3 Statistical Thresholding

This method utilizes the statistical aspects of an audio signal in order to determine appropriate values for the threshold. There are multiple techniques used in order to determine the values such as:

- Mean and Standard deviation: The threshold is set based on the computed mean using the amplitude, which reflects the overall intensity or energy of the signal at a given time. By analyzing the mean, we can understand better how a song's signal fluctuates
- Percentile-Based Thresholding: Unlike the later technique, percentile-based algorithms capture the behavior of the song's musical events distribution. This approach focuses on capturing the extreme values within the signal. This means that the lower values of the frequencies are being ignored and they do not contribute to the computation of the threshold itself, making it very effective against noise. One thing to take into consideration is the sensitivity, higher percentiles correspond to a more strict result, while lower percentiles give more lenient values, with a wider range of signal detection.

2.4.4 Machine Learning-Based Thresholding

Machine learning-based thresholding. These methods offer a great approach when it comes to complex musical event detection across a diverse dataset and have very practical implementations in important

fields such as healthcare monitoring. Some of the most popular methods are Supervised Learning Methods, including Support Vector Machines and Convolutional Neural Networks [5] and Unsupervised Learning Methods such as clustering algorithms, as well as Reinforcement Learning Methods such as Q-Learning.

3 Project Implementation

Building upon the knowledge presented in the previous chapter, this chapter explores the methodologies and overall development of this project, including the audio analysis, the feature extraction and the generation of a rhythm game level.

3.1 Workflow overview

The main goal of the project is for each user to input an audio file of their choice in order for that to be analyzed and become a playable rhythm game level. Therefore the project begins with inputting a song/audio file provided by the user. This file can be of any compatible format (e.g., MP3) which will later be processed by the algorithm.

3.2 Song analysis

The song analysis algorithm, implemented in C# using Unity, processes audio files provided by users to generate game levels. Users can select any song in .mp3 or .wav format from their computer, and the file is attached to a Game Object that handles audio playback through Unity's AudioSource component. This component offers various features, including the 'GetSpectrumData' method, which retrieves the frequency spectrum data from the audio file. The method takes three parameters and plays a key role in analyzing the audio to create the corresponding game levels.

- samples: an array of floats where the spectrum data will be stored

- channel: the audio channel that is being used to extract the spectrum data, either 0 (for the left channel) or 1 (for the right channel)

- window: the type of window to use in order for the FFT to use. Taking into consideration the points made in the 6th Chapter of this paper, for this project, the windowing function that was chosen was BlackmanHarris because of its great reduction when it comes to spectral leaking. In order to make it as general as possible, the algorithm takes into consideration both channels.

The "Update" method in Unity is used to call 'MakeFrequencyBands' once per frame during the song, processing audio data by dividing it into 8 frequency bands. A logarithmic scale is applied, allowing lower frequencies to be captured in finer detail while still including higher frequencies. The frequency bands are normalized using the highest recorded value to ensure all values range between 0 and 1. Additional methods calculate amplitude and create buffers for smoother results. The processed data, along with timestamps and amplitude, is serialized into a JSON string and saved locally in the project's assets for future access and use.

3.3 JSON analysis and calculating the threshold

The data from each JSON is deserialized and processed to calculate threshold values, with frequency bands combined for clarity. Higher frequency bands are merged to improve algorithm efficiency. Several threshold computation methods were tested, with percentile-based thresholding chosen due to its effectiveness in handling noise and focusing on amplitude spikes. This method also allows for difficulty scaling by selecting predefined percentiles. To optimize computation, minibatching is used, dividing data into smaller parts for more accurate results. Tailoring thresholds for each frequency band improves detection accuracy, enhancing system performance.

3.4 Generating obstacles

The rhythm game level itself now has all the components it needs in order to be generated. The same approach that was used when initially analyzing the songs serves as the base on the level generation as well. The specific audio Source is played and observed per frame, thanks to the Update method from Unity. The data from the JSON is again deserialized from the file. Within this method the script is continuously checking the current timestamp of the audio that is playing in the background and checks it with the timestamps from the saved file. As the audio reaches the timestamps found in the file, the scripts check for the band values and the threshold values and determine if game objects that serve as the obstacles need to be activated. Upon being activated, they are given a certain force in order to simulate the movement.

The mini batching is again employed in this part of the project's code, the script incorporating logic that determines exactly which of the threshold values to use for each of the determined batches.

4 Conclusion

Our work set out with the purpose of developing a rhythm game that diminishes the amount of manual work developers need to put in to create a level. This thesis included a developing part as well a theoretical research part covering techniques such as Fourier transforms, filtering and thresholding. By understanding the basics of signal processing and researching various algorithms, the project presents an algorithm with a strong foundation in the realm of beat detection [1]. By combining the knowledge acquired from this theoretical part with game design elements, this project accomplished its goal of creating a game that can take as input any audio file and generate a playable rhythm game level by analyzing the frequencies of the inputted audio file. In conclusion, while the current state of the project indeed accomplishes its primary goal of automating a manual process, it also allows for future improvement.

References

[1] J. Balázs. *Beat detection and correction for DJing applications*, 2013. http://dsp.mit.bme.hu/userfiles/diploma/balazsdiploma13.pdf.

[2] A. Klapuri. *Procedural content generation of rhythm games using deep learning methods*. Hal-Inria, April 26, 2022. Retrieved April 9, 2024, from https://inria.hal.science/hal-03652042/document.

[3] G. Zanetakis and P. Cook. *Musical genre classification of audio signals*, July, 2002. https://ieeexplore.ieee.org/abstract/document/1021072/authors#authors.

[4] G. Peeters. *Spectral and temporal periodicity representations of rhythm for the automatic classification of music audio signal*, October 21, 2010. https://ieeexplore.ieee.org/document/5607297/authors#authors.

[5] D. Fiocchi. *Beat tracking using recurrent neural network: a transfer learning approach*, 2016-2017. https://www.politesi.polimi.it/retrieve/a81cb05c-b4bf-616b-e053-1605fe0a889a/tesi.pdf.

[6] A. Klapuri, *Sound onset detection by applying psychoacoustic knowledge*, March, 1999.

Cosmin Irimia[1], Stefana Biceadă[2]

[1]Faculty of Computer Science, "Alexandru Ioan-Cuza" University, Iași, Romania
E–mail: cosmin.irimia@info.uaic.ro
ORCID: https://orcid.org/0009-0002-7536-1161

[2]Faculty of Computer Science, "Alexandru Ioan-Cuza" University, Iași, Romania
E–mail: alexandra.biceada@students.info.uaic.ro
ORCID: https://orcid.org/0009-0008-0385-1595