|  |
| --- |
| AutoCharter |
| A Music Processing Program for Rhythm Game Implements |
|  |
| by Ginoel Ivan Ng Teng |
| for ICSI433 at the State University of New York at Albany |
| December 13, 2016 |

|  |
| --- |
| [Type the abstract of the document here. The abstract is typically a short summary of the contents of the document. Type the abstract of the document here. The abstract is typically a short summary of the contents of the document.] |

Introduction

Rhythm games account for a longstanding, albeit relatively niche genre within the greater scope of video gaming. Such games typically have the player(s) perform certain actions in time with music being played in the background. More immersive simulations allow the player to modify the music in some manner, sometimes simulating the feeling of actually playing the given songs. Having ostensibly started with Konami’s DJ simulation game *Beatmania* (typically stylized as *beatmania*) in 1997, the genre expanded drastically upon the release of *Dance Dance Revolution* in 1998, taking the world by storm with its active and innovative gameplay at the time.



Fig. 1: (left) A picture of a Dance Dance Revolution arcade cabinet

(right) A gameplay screenshot of the latest version, Dance Dance Revolution Ace

In the last five years, rhythm games have enjoyed a significant boom within the mobile market. *jubeat plus* (2010, the mobile port of Konami’s arcade game *jubeat,* 2008), *REFLEC BEAT plus* (2011, the mobile port of Konami’s arcade game *REFLEC BEAT,* 2010), *Cytus* (2012, Rayark), *Deemo* (2013, Rayark), *LoveLive! School Idol Festival* (2013, KLab), and other such rhythm games have all seen immense popularity worldwide, relative to the perceived size of the playerbase that has seemingly shrunk since the slow decline of *Dance Dance Revolution*’s popularity.

A feature once typical of rhythm games, as mentioned, is *keysounding.* In games where players are expected to “perform” the music in the sense that they are given some sort of instrument to complete a background track with their own contributions, the sound bites that were originally part of the song but separated to form the play medium (also known as *note charts*) are called keysounds. The concept has appeared as early on as *beatmania*, and has since been implemented in other games such as *pop’n music* (1998), *GuitarFreaks* and *DrumMania* (1999), O2Jam (2003), and DJMAX (2004). Keysounds are also implemented to a lesser degree in the stateside games *Guitar Hero* (2005), *Rock Band* (2007), as well as Konami’s updated DJ simulation *Sound Voltex (2012)*.

Note that over the past two decades, keysounds have been slowly phased out. There are three explanations as to why. One is that early smartphones and tablets may not have been able to store or efficiently load several high-quality keysounds at a time. Another is that music licensed from indie circles has become very popular as of late, and some of the tracks may be old enough that the original stems are no longer available.

The final and perhaps most compelling reason is the significant investment required to produce keysounds. For fully or mostly keysounded games, in-house sound engineers painstakingly pick apart individual keysounds from multi-track stems. These keysounds, often numbering well into the hundreds, are then laid out onto several notecharts of varying difficulties. Between the two processes, preparing even a single song for a game can be arduous at best.

Such was the motivation for the revival of my AutoCharter project. Initially developed in 2014 as my freshman independent research project, AutoCharter was written in one sizeable Java source file, naturally with procedural behavior. It was configured to be able to read percussion only (and just barely), and did not even have any groundwork for keysound extraction. The most novel feature therein was the fact that it could also read .mp3 files by crudely converting them to .wav. As such, in order to make this truly viable as a novel digital signal processing project, the following improvements were proposed:

1. Allowing the program to read in intervals of triplets to improve flexibility

2. Allowing the program to process harmonic instruments using modified attack/release thresholds, pitch change detection, among other techniques

3. Augmenting detection by utilizing algorithms found in other areas of sound signal processing, such as in speech recognition

4. Utilizing drum source separation algorithms to minimize the number of sound source files required to fully represent a song

5. Crude keysound extraction in combination with the above features

Of the features listed, only 2, 3, and 5 could be fully implemented due to time constraints. (At the risk of undue bluntness, it really has been a rough semester.) There is groundwork already laid out in the code for 1, although unexpected roadblocks in the development of other three features (to be further explained below) necessitated that all resources and attention be directed to finishing each one properly. Furthermore, should any external library used have failed to generate adequate results, it would have been beyond my knowledge to repair. Nonetheless, keysound extraction, which I believe to be the most novel feature in this project, has been accomplished to a passable degree, pending further improvement.

Review of Related Literature

A similar premise of onset detection for rhythm game use has been employed by Chen et al. in 2015 for their conference paper and game, AutoRhythm. It deals heavily with one of the features listed above (“Utilizing drum source separation algorithms to minimize the number of sound source files required to fully represent a song”). However, that feature was not included in this iteration of AutoCharter. Furthermore, AutoRhythm’s main draw is real-time input detection of varied and likely nonstandard percussion sources, whereas AutoCharter’s main purpose is to create notecharts using distinct, discrete, pre-rendered, and isolated musical stems, mostly in order to extract keysounds.

Other work on rhythm game charting has been attempted by Gold and Olivier in 2010. Using *Guitar Hero* and *Frets on Fire* (a simulator of the former) as the bases, the main purpose of the project is to be able to work backwards from a high difficulty notechart to expedite creation of easier notecharts. However, unlike this project, it does not take into account Guitar Hero’s keysounding system, granted that its keysounding was minimal in the first place.

As such, there appears to have been no publicized attempt to date to extract keysounds in the manner to be described in this paper. The given model can receive any sort of musical waveform and attempt to extract keysounds and place notes based on metrics and patterns specific to percussive or harmonic categories.

Methods

Using the techniques described below, up to 36 files may be used as input, and subsequently processed by the program to accomplish two tasks:

* Extract keysounds and match them where appropriate to minimize output file count, and
* Map the respective keysounds to their charts

The following techniques are used in this approximate order.

Threshold Detection

Each file is read in 16th note chunks, whereupon basic characteristic profiles, distinguished between those used for percussive and harmonic modes, are used to gauge whether or not a potential keysound has been detected. The first potential keysound in each lane is taken at face value, as long as it meets a certain energy threshold. For every chunk after the first detected, the attack value of the first half of the chunk is gauged by comparing its energy against that of the second half of the previous chunk. All energy equations heretofore are given by:

**Algorithm 1** The total energy computation for a given input waveform

for(all samples in waveform){

if( current sample is negative and future samples appear to trend upward, or current sample is positive and future samples appear to trend downward)

E += abs(current sample)

}

Some distorted percussion may have tails with high amplitude but somewhat lower frequency.



A normalized value to compare against, henceforth referred to as density, helps with this. This density is given by an equation similar to that of energy’s:

**Algorithm 2** The total energy computation for a given input waveform

for(all samples in waveform){

if( current sample is negative and future samples appear to trend upward, or current sample is positive and future samples appear to trend downward)

D += 1

}

Percussion Profile

For sound files that the user has registered as percussive, the following characteristics are tested.

**Threshold set 1** For percussion

1. EPCSH is not 0. ENCFH/EPCSH > 1.55.

2. EPCSH is 0. ENCFH > 100.

3. EPCSH and ENCFH > some low value. DPCSH is not 0. DNCFH/DPCSH >= 2.0

\* PCSH = previous chunk’s second half

\* NCFH = new chunk’s first half

Instrument Signatures – Percussion

Once a keysound has been confirmed with at least one of the above characteristics, its signature will be compared against unique chunks that have been read prior. This is accomplished with linear predictive coding. Using Gimpel and Wimba’s open-source implementation of LPC, for some set of float values x and a lag window of 16, a set of 16 autocorrelation values are obtained like so:

**Algorithm 3** The autocorrelation algorithm

while (lag-- > 0) {

for (i=lag, d=0; i<n; i++)

d += x[i] \* x[i-lag];

ac[lag] = d;

}

References

Chen, P.-P., Yeh, T.-C., Jang, J.-S. R. & Liou, W. (2015). AutoRhythm: A music game with automatic hit-time generation and percussion identification.. ICME (p./pp. 1-6), : IEEE Computer Society. ISBN: 978-1-4799-7082-7

Preview (early draft): http://www.terasoft.com.tw/conf/ismir2014/LBD%5CLBD26.pdf

Full text (paywall): http://ieeexplore.ieee.org/document/7177487/

Gold, K.; Olivier, A. (2010). Using machine translation to convert between difficulties in rhythm games. In Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE'10), G. Michael Youngblood and Vadim Bulitko (Eds.). AAAI Press 27-32.

Full text: http://www.aaai.org/ocs/index.php/AIIDE/AIIDE10/paper/viewFile/2130/2540