RhythmTool 2.0 Documentation

Table of Contents

Contact & info

If you have any questions, suggestions, feedback or comments, please do one of the following:

- Send me an Email: tim@hellomeow.net
- Make a post in the RhythmTool thread

Overview

RhythmTool is a straightforward scripting package for Unity, with all the basic functionality for creating games that react to music.

RhythmTool analyzes a song without the need of playing the song at the same time. It can analyze an entire song before playing it, or while it's being played.

There are a number of types of data it provides:

- Beats
- Onsets
- Changes in overall intensity
- volume

This data can be used in various ways and is provided through lists and UnityEvents.

RhythmTool is designed to analyze and sync songs with a known length. Unfortunately it is not possible to analyze a continuous stream of data, like a web stream or mic input.

Change log

Version 2.0:

- RhythmTool, RhythmEventProvider and the documentation have been rewritten from scratch
- Removed RhythmTool.IsBeat and RhythmTool.IsChange. ContainsKey or TryGetValue for both the beats and changes collections can be used instead
- Replaced RhythmTool.NewSong() with RhythmTool.audioClip
- Renamed RhythmTool.calculateTempo to RhythmTool.trackBeat
- Renamed RhythmTool.preCalculate to RhythmTool.preAnalyze
- Renamed RhythmTool.storeAnalyses to RhythmTool.cacheAnalysis
- Added RhythmTool.Reset and RhythmEventProvider.Reset, an event that occurs when a song has restarted or a new song has been loaded
- RhythmEventProvider now needs to be given a RhythmTool Component instead of using all RhythmTool components in the scene
- RhythmEventProvider no longer uses UnityEvents. Instead it uses c# events, for a more consistent api. In the previous verions c# events and UnityEvents were used in different cases. This means the events are no longer available in the editor

Setting up RhythmTool

To begin using RhythmTool, import the Unity Package. After that, you can start using the RhythmTool Component to analyze songs.

See the included examples for a quick overview of the package.

Frames

RhythmTool chops up a song in small pieces of roughly 0.03 seconds. In other words, there are 30 pieces of data for every second of the song. With RhythmTool, these pieces are referred to as *Frames*.

When playing and analyzing a song, RhythmTool keeps track of the total number of frames, the last analyzed frame and the frame that corresponds to the current time in the song. Every detected beat or onset and every datapoint of the song has a frame index. This lets you synchronize the analysis results to the music and gameplay in different ways.

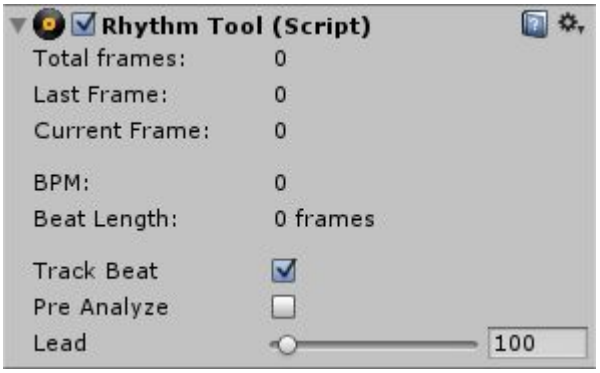
Settings

The RhythmTool Component has some basic settings.

The **Track Beat** option lets you toggle beat tracking. Beat tracking looks at large parts of a song to find the most likely BPM and then synchronizes the BPM as a series of beats. Beat tracking is very useful, but it can increase the time it takes to analyze a song and it might not be needed for some projects.

The **Lead** slider lets you configure how much data will be analyzed ahead of the current time in the song.

When you enable **Pre Analyze**, RhythmTool will analyze the entire song as fast as possible before enabling playback. This also exposes the option **Cache Analysis**. Instead of re-analyzing a song every time it is loaded, it can store and reuse analysis results.

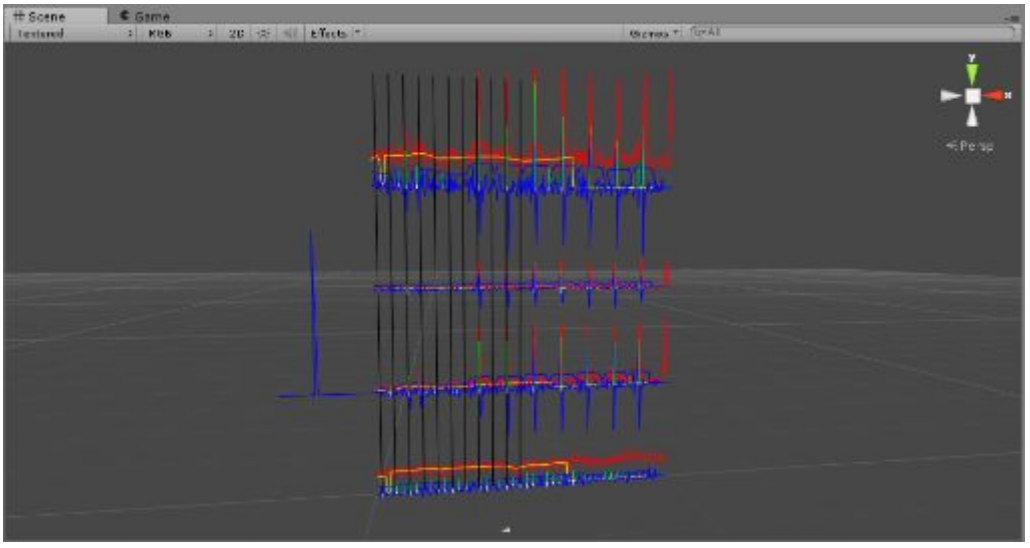


Analyzing a song

To start analyzing a song, assign an AudioClip to RhythmTool.audioClip. This will load the AudioClip and start analyzing it.

When the song has finished loading, RhythmTool.SongLoaded will be called. A song has finished loading when the initial analysis is complete. When preAnalyze is enabled, this will be after the entire song has been analyzed. Otherwise this will be when a number of frames (based on lead) has been analyzed.

With RhythmTool.DrawDebugLines() you can draw some basic graphs that show the analysis results.



Playing a song

After a song has been loaded, use RhythmTool.Play() to start playing the song.

The RhythmTool Component has some methods and properties to control the audio playback. These are mostly the same as the AudioSource Component. These include RhythmTool.Play(), RhythmTool.Stop(), RhythmTool.Pause(), RhythmTool.UnPause(), RhythmTool.pitch and RhythmTool.volume

RhythmTool uses an AudioSource to play a song. It is possible to use this AudioSource to control playback, but this might cause unintended behaviour.

```
using UnityEngine;

public class AnalyzeExample : MonoBehaviour
{
    public RhythmTool rhythmTool;

    public AudioClip audioClip;

    void Start ()
    {
        rhythmTool.SongLoaded += OnSongLoaded;

        rhythmTool.audioClip = audioClip;
    }

    private void OnSongLoaded()
    {
        rhythmTool.Play ();
    }

    void Update ()
    {
        rhythmTool.DrawDebugLines ();
    }
}
```

Using analysis results

RhythmTool has three main types of data:

- Beats
  These represent the rhythm of the song.
- Changes
  These represent changes in the overall intensity of the song and are useful for telling segments of the song apart.
- AnalysisData
  These contain analysis results and detected peaks (onsets) for different frequency ranges.

There are 4 default Analyses. low, mid, high and all. Each of these looks at a specific frequency range and stores the following data:

- Magnitude
  The combined magnitude of all frequencies in the specified frequency range. Essentially the loudness.
- magnitudeSmooth
  A smoothed variant of the magnitude.
- magnitudeAvg
  A version of magnitudeSmooth that interpolates from trough to trough and peak to peak. This is like a smooth version of magnitude, but without big variations.
- Flux
  The difference between a frame's and the previous frame's magnitudes.
- Onsets
  Detected peaks that represent the beginning of a note, sound or beat.

Synchronizing results

All of the analysis results are stored in collections where the index or key is the frame index. The main method of synchronizing analysis results is by using RhythmTool.currentFrame. This is the index of the frame that belongs to the current time in the song.

For example, finding out if an onset has been detected at the current time in the song can be done like this:

```
AnalysisData low = rhythmTool.low;

...

Onset onset;
if(low.onsets.TryGetValue(rhythmTool.currentFrame, out onset))
{
    Debug.Log("An onset occured with a strength of " + onset.strength);
}
```

And finding out if there is a beat:

```
...

Beat beat;
if(rhythmTool.beats.TryGetValue(rhythmTool.currentFrame, out beat))
{
    Debug.Log("A beat occured at " + rhythmTool.currentFrame);
}
```

Or using some other data from the "all" Analysis:

```
AnalysisData all = rhythmTool.all;

...

float volume = all.magnitude[rhythmTool.currentFrame];
```

Synchronization issues

In most cases, the game's frame rate does not perfectly match RhythmTool's sample rate of 30 frames per second of audio data. Because of this, RhythmTool.currentFrame can be the same for multiple frames in a row, or skip some values. This can be an issue when you want to handle every beat, onset or other analysis results exactly once.

This issue can be solved by looping over every frame that has passed since the last Update(). This way, any skipped frames will be dealt with and no frames will be used twice.

RhythmTool already does this. The RhythmTool.FrameChanged and the RhythmEventProvider.FrameChanged events will occur once for every frame that has passed.

Using events

RhythmTool has a number of events:

- FrameChanged (int currentFrame, int lastFrame)
  Occurs for every frame that has passed.
- TimingUpdate (int currentFrame, float interpolation)
  Occurs every Update and passes currentFrame and interpolation.
- Reset
  Occurs When a song has been loaded, has started playing or has stopped playing.
- SongEnded
  Occurs when the song has ended.
- SongLoaded
  Occurs when a song has been loaded and is ready to start playing.

RhythmEventProvider

The RhythmEventProvider Component provides a number of useful events and features.

It can be given a RhythmTool component that will be used to trigger a number of events.
It can also be given an offset, so events can be triggered ahead of time

In addition to each of RhythmTool's events, the following events are available:

- Beat (Beat beat)
  Occurs every beat.
- SubBeat (Beat beat, int count)
  Occurs every quarter beat.
- Onset (Onset onset)
  Occurs every Onset.
- Change (float change)
  Occurs every time a change happens.

For example, to subscribe to the Beat event:

```
public RhythmEventProvider eventProvider;

void Start()
{
    eventProvider.Beat += OnBeat;
}

private void OnBeat(Beat beat)
{
    Debug.Log("A beat occured at " + beat.index);
}
```

When using events in C# with Unity, make sure to unsubscribe from events for Components that are destroyed. Otherwise the subscribed method will still be called even if the Component is destroyed.

```
void OnDestroy()
{
    eventProvider.Beat -= OnBeat;
}
```

Importing songs at runtime

To import songs at runtime a separate package needs to be imported into the project.

AudioImporter.unitypackage

The package contains 3 different importers:

- MobileImporter, which is for mobile platforms only. It uses Unity's WWW / UnityWebRequest class, that can load mp3 files on mobile platforms only.
- NAudioImporter, which uses NAudio, which is licensed under the Ms-PL. It might require licensing for using MP3 technology.
- BassImporter, which uses Bass.dll and Bass.net, that require separate licenses for commercial projects. This importer can use the OS's mp3 decoder, so no MP3 license is needed.

MP3 technology is patented and might need additional licensing. More information

If you're not planning on using one of the importers, just delete its associated assets.

BassImporter needs two additional steps:

1. Get bass.dll from http://www.un4seen.com/ and place the required libraries (based on the editor platform and the project's build target) in a Plugins folder.
2. Get bass.net.dll from http://bass.radio42.com/ and place it somewhere your project's assets folder. Bass.Net comes with a number of versions. Unity needs the one from the "V2.0" folder. You can register Bass.Net on http://bass.radio42.com/bass_register.html and by editing line 17 of BassImporter.cs.

In order for some of the importers to work properly, the API compatibility level needs to be set to ".NET 2.0" in player settings. Otherwise it will only work in the editor, if at all.
For Android, Write Permission has to be set to External.

Importing a song

The AudioImporter Components have two methods for importing a song. **Import()** and **ImportStreaming()**. Import() imports the whole file at once, while ImportStreaming() imports a file gradually. This makes the song available for playback much faster, because it can be imported while it is being played.

After using Import() or ImportStreaming(), the importer will start loading the song into an AudioClip. This happens in the background as much as possible. Use the **Loaded** event to obtain the audioclip.

In Unity 5.3 or newer, these methods return a CustomYieldInstruction named **ImportOperation**, which can be used in a coroutine.

Browser

The package also includes a simple file browser, in the form of a prefab with an attached Browser Component. The browser uses Unity's UI system, so the prefab has to be parented to a Canvas.

Use the Browser Component's **FileSelected** event to handle selected files.

```
using UnityEngine;

public class ImporterExample : MonoBehaviour
{
    public Browser browser;
    public AudioImporter importer;
    public AudioSource audioSource;

    void Awake()
    {
        browser.FileSelected += OnFileSelected;
        importer.Loaded += OnLoaded;
    }

    private void OnFileSelected(string path)
    {
        importer.ImportStreaming(path);
    }

    private void OnLoaded(AudioClip clip)
    {
        audioSource.clip = clip;
        audioSource.Play();
    }
}
```