RHYTHMTOOL DOCUMENTATION

TABLE OF CONTENTS

CONTACT & INFO

If you have any questions, 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 with all the basic functionality for creating rhythm/music games based on the player's own music collection.

RhythmTool gives you the ability to import and analyze mp3 files (mainly music) at run time. This means that you can let the player select a song from his/her computer. Subsequently there will be data generated based on that song. You get to have the song's data for the song more than 10 seconds in advance.

HOW TO USE

IMPORTING RHYTHMTOOL

To begin using RhythmTool, just import the Unity Package. From there, you can use it in any script you like.

RhythmTool is made to be able to be used in any script in unity. In any game made with Unity, I've found it most practical to have one or more controller-objects and/or scripts that govern all systems in the game.

RhythmTool should be added to such a script where it's most practical. That is probably going to be the script where you need to have the actual data.

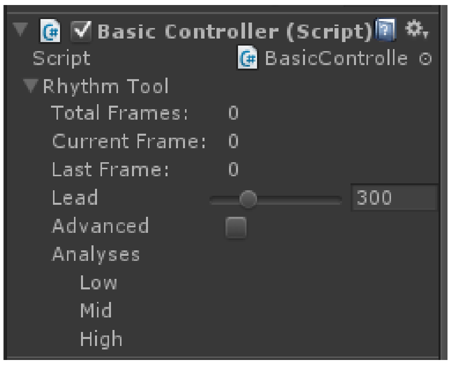
Important: The examples use windows.forms, which require the api compatibility level to be set to .Net 2.0 in the player settings. Set the api compatibility level to .NET 2.0 and restart Unity if this is not the case.

TUTORIAL 1: BASIC SETUP

In this tutorial, we're going to make a simple script that will import and play a song and show the raw data. Start out with a new script and Add RhythmTool. To add RhythmTool to such a script, declare it like this:

```
//This is a basic controller. It imports a song, passes it on to the analyzer and starts playing it.
//See ExampleController and the example scene for an example on how to configure the analyzer and get the data in a game.
public class BasicController : MonoBehaviour
{
    /// <summary>
    /// RhythmTool. Configure it in the inspector.
    /// </summary>
    public RhythmTool rhythmTool;
```

After that, it is configurable in the inspector. It's not required to configure anything if there is no need for specific parameters for the analysis. It's not necessary to instantiate a new RhythmTool object. Unity does that automatically. It should look like this:



```
Basic Controller (Script)
Script              BasicControlle
Rhythm Tool
  Total Frames:     0
  Current Frame:    0
  Last Frame:       0
  Lead              300
  Advanced
  Analyses
    Low
    Mid
    High
```

To begin using RhythmTool for loading and analyzing a song, it needs to be initialized first.

```
// Use this for initialization
void Start ()
{
    //Initialize RythmTool.
    rhythmTool.Init(this,false);
```

Init(Monobehaviour script, bool preCalculate) has a parameter for the script from which it's initialized and if you want it to precalculate the results. When preCalculate is set to true, the song will be analyzed in just a few frames. This allows for some more analysis, since more information is known about the entire song.

After it's initialized, a song can imported from the user's library at runtime, or a song can be selected as an AudioClip from the existing assets. The latter is meant to give the option to analyze music that is imported as an asset into unity.

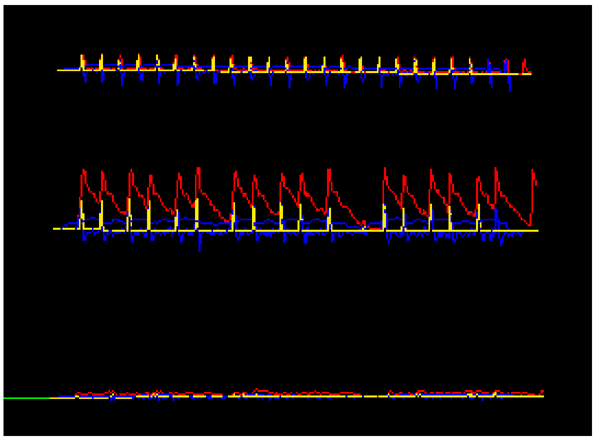
To import a song from the user's library, you need to have the file path to that song. Unity does not have a file browser to let the user select a file. RhythmTool currently has a utility for this, which only works on the Windows platform.

```
// Use this for initialization
void Start ()
{
    //Initialize RythmTool.
    rhythmTool.Init(this,false);

    //Load a song.
    string songPath=Util.OpenFileDialog();
    rhythmTool.NewSong (songPath);

    //Start playing the song
    rhythmTool.Play ();
}
```

With Play(), the song will begin playing. Now the song is playing, it's doing just that. Unless we update the analysis, it won't analyze anything. To do this, simply call Update().

```
// Update is called once per frame
void Update ()
{
    //Update RhythmTool.
    rhythmTool.Update ();
    rhythmTool.DrawDebugLines();
}
```

DrawDebugLines() will draw a couple of graphs, containing the results from the analysis.



What do those graphs mean? Without doing any “advanced” configuration in the inspector, there are three different analyses going on and there are three separate graphs.

That’s not a coincidence. Each graph represents an analysis of a certain frequency range. One for low frequencies, one for mid-range frequencies and one for high frequencies. This way, there is a relatively easy way of differentiating between certain kinds of occurrences in a song.

What this data actually means and how to use it, will become apparent when we’re going to use this data in something that could resemble a game.

TUTORIAL 2: HOW TO USE THE DATA

The data comes in the form of frames, for a lack of better words. The Analysis takes a bunch of samples at a certain rate and stores the results in an array of frames, which is easily synchronized with the song.

One frame contains the following information:

- magnitude
  The combined magnitude of all frequencies in the specified frequency range.
- flux
  The difference between this and the previous frame’s magnitudes.
- threshold
  The threshold that is used for peak picking.
- onset
  The beginning of a note or sound. Every value higher than 0 is an onset of some kind. Lower values might be false-positives.
- onsetRank
  A number indicating how this onset relates to neighboring onsets. Onsets are ranked from 5 to 0.
  5 is a peak, 4 is a peak when a neighboring, higher peak is ignored and so on. Useful for when only the most prominent onsets are needed.

How do we get these frames? GetResults(string name) returns the array of frames from one of the analyses. The basic controller from the previous tutorial can easily be expanded, to get access to the results from the default low-frequency analysis.

```
/// RythmTool. Configure it in the inspector...
public RhythmTool rhythmTool;

/// Frames from "Low" analysis...
private Frame[] low;

// Use this for initialization
void Start ()
{
    //Initialize RythmTool.
    rhythmTool.Init(this,false);

    //Load a song.
    string songPath=Util.OpenFileDialog();
    rhythmTool.NewSong (songPath);

    //Start playing the song
    rhythmTool.Play ();


    //get the data from the analyzer.
    //"Low", is the name for one of the default analyses.
    low=rhythmTool.GetResults("Low");
}
```

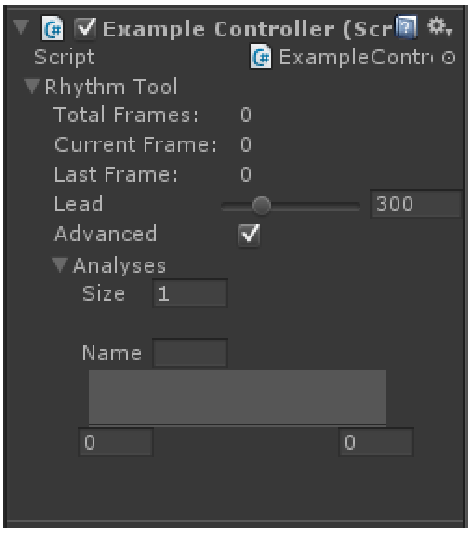
Now we have the data, how do we use it? As an example in this tutorial, we're going to make some lines to represent the data. After updating RhythmTool, we're going to draw the lines.

With a loop, we are going to look at the frame corresponding with the current time in the song, and the 100 frames after that. If there is an onset at a frame, we are going to draw a line.

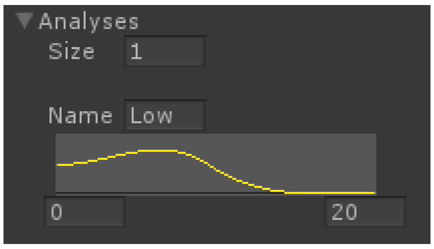
The horizontal position of this line is determined by the index of this frame and the current time in the song, so the lines will appear to move towards a point, and meet that point when their time in the song arrives.

```
// Update is called once per frame
void Update ()
{
    //Update RhythmTool and draw debug lines.
    rhythmTool.Update ();
    rhythmTool.DrawDebugLines();

    //Game logic
    //Look at 100 frames and draw a line if there is an onset.
    for(int i = 0; i<100; i++)
    {
        //the index of the frame we are going to check and possibly draw a line for
        int frameIndex = Mathf.Min(i+rhythmTool.CurrentFrame, rhythmTool.TotalFrames);

        //the onset value for this frame.
        float onset = low[frameIndex].onset;

        if(onset>0)
        {
            //horizontal position of the line.
            float x = i-rhythmTool.Interpolation;

            //create two vectors for drawing a line.
            //The magnitude of the onset is used as the length of the line.
            Vector3 l1 = new Vector3(x,-20,0);
            Vector3 l2 = new Vector3(x,onset-20,0);
            Debug.DrawLine(l1,l2,Color.red);
        }
    }

    //Draw one line at the start, so we can see where it is.
    Debug.DrawLine(new Vector3(-1,-20,0),new Vector3(-1,-10,0),Color.red);
}
```

Now we can setup RhythmTool, and use the data, there still is one thing we can learn about it. What does the "Advanced" tick box do?


Example Controller (Scr
Script ExampleContr
Rhythm Tool
Total Frames: 0
Current Frame: 0
Last Frame: 0
Lead 300
Advanced ✓
Analyses
Size 1
Name
0 0

In short, it allows you to configure your own analyses. You can set a name and a frequency range and it allows you to configure a curve that represents how much each frequency is weighed.


Analyses
Size 1
Name Low
0
20

This example will analyze frequencies from the index of 0-20, with 5-10 being weighed more and with a slow drop-off to zero, from 15-20. Configuring an analysis like this can be useful if you're planning on fine-tuning an analysis for certain music, or gameplay.

EXAMPLES

Currently, the RhythmTool package contains all finished tutorials and one more fleshed-out version of tutorial #2.

RHYTHMTOOL CLASS

FUNCTIONS

AudioSource Init (MonoBehaviour script, bool preCalculate)

Initialize RythmTool.

bool NewSong (string songPath)

Imports a (new) song. (re) initializes analyses and variables.

bool NewSong (AudioClip audioClip)

Use an AudioClip as the new song. (re) initializes analyses and variables.

void Play ()

void Stop ()

void Update ()

Update RhythmTool. If the analysis is complete, Update() needs to be called in order to keep the time.

Frame[] GetResults (string name)

Gets the results.

void DrawDebugLines ()

Draws the debug lines.

PROPERTIES

int TotalFrames [get]

The total number of frames the analysis has

int CurrentFrame [get]

The index of the frame corresponding with the current time in the song.

float Interpolation [get]

Use to interpolate between the current frame and the next.

int LastFrame [get]

The last analyzed frame.

int Lead [get, set]

How much of a lead (in frames) the analysis has on the song.

bool IsDone [get]

If true, the analysis is done.

bool Initialized [get]

If true, RhythmTool is initialized.