RhythmosEngine

Manual

Version 1.1

tobiasbu – 2015-2017

Summary

1. Introduction

Rhythmos Engine is an asset for Unity to create rhythms for musical games or not, based on musical notation. It has two segments: rhythms editor and application.

What is „*Rhythmos*“? From greek *rhythmos*, "any regular recurring motion, symmetry".

The rhythm editor – **Rhythmos Editor** – allows developers to easily create their rhythms in an extended interface in Unity. All rhythms is saved on database based in XML file.
The application – **Rhythmos Engine** – has a range of classes in C# to use during the game, such as the comparison of rhythms, play rhythms at run time, check precision between two rhythms and others features.

Rhythmos Engine brings together two DLLs:

RhythmosEngine.dll: contains a collection of classes and structures to assist the using of rhythms during the game. Location on Unity: Assets/Plugins/RhythmosEngine

RhythmosEditor.dll: It is an extension of Unity to create your own rhythms and notes database. This file must be in Assets/Editor/RhythmosEngine directory. To open the Rhythmos Editor, in Unity Toolbar go to Tools>Rhythmos Editor.

Important Notes:
- All notes (AudioClips) need to be located in the Assets/Resources directory to play sounds during the game.
- The rhythms database is on XML format. It brings together the entire contents of rhythms and notes that you have created. If you want to load this file by code with TextAsset, the database must be in the Assets/Resources directory too.
- RhythmosEditor.dll save a configuration file on Editor directory only to load the last edited database.

1.1. Support

If you have question, suggestions or issues please contact the e-mail tobiasbulrich@gmail.com or access the website: http://tobiasbu.github.io/website/.

1.2. Change-Log

Current Versions:

RhythmosEditor.dll: 1.1.
RhythmosEngine.dll: 1.1.

Complete log:

Date	Changes
23/06/2015	First release of RhythmosEngine.
29/06/2017	- Updated to Unity 4.6.0f version. - Few changes for compatibility in Unity 5 and higher versions. - Change the .NET framework to 3.5 version. - BPM now is float type.

2. Music Notation

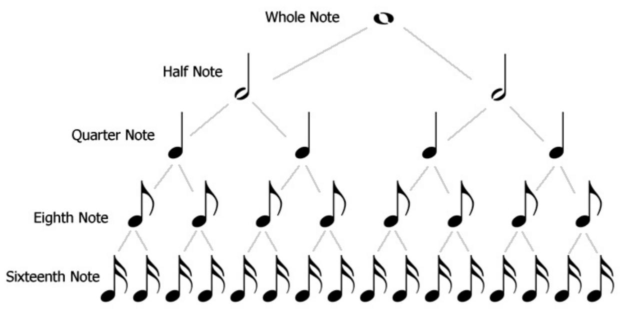
Before we start, let's see a little bit of Music Notation:

Rhythm it is a sequence of sounds at regular intervals. Rhythms can have repetion and alternation of dynamic with strong and weak beat.

The tempo of a Rhythm is the speed, a measure of how quickly the beat flows. This is often measured in 'beats per minute' (BPM). For example, in a song the tempo is 80 BPM, a pulse of this music equals 0.75 seconds.

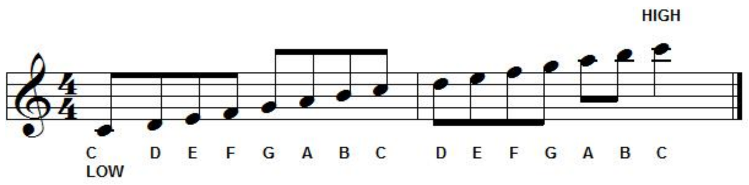
Calculate pulse of BPM:
60 / *bpm = xseconds*   60 / 80 = 0.75s

Musical notes also have duration. To understand the duration of notes and their corresponding values must be understand how this aspect works in the music notation. According to the image below that describes the hierarchy of notes, following a brief description.



Notes can also be assigned as a rest (no sound). At the top of the hierarchy have a note with longer duration, called the whole note. The whole note lasts four pulses of speed of a song. Just below the whole note, we have the half note, which is half of the whole note, lasting two pulses of progress and so on. For example, the tempo of a song is 40 BPM, the length of the quarter note (below of half note) which equals a pulse would be 0.66 seconds.

In musical notation, the notes have so-called "high", which differentiates between high and low tones. Another element are the types and instrument sounds. The image below demonstrates how more low and high notes are differentiated into a musical pentagram on the scale of G major. The positioned above notes are more acute as the most serious are positioned below:

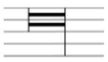
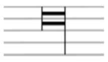
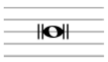
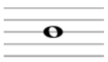
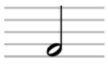
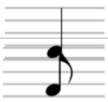
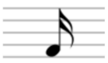
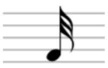
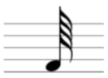
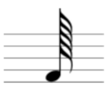


The RhythmosEngine have a problem that it don't enables to simulate the sound of instrument in different heights. In order to cover these aspects, the tool adopted an attribute called **Note Layout.** The Note Layout is the way the RhythmosEngine can differentiate one note from other note. Note Layout is done by a name, an ID number, color for RhythmosEditor and an AudioClip to play the desired sound.

2.1. Notes and BPM Values

In this topic is organized corresponding values for notes and bpm.

Note Values:

Rhythmos Value:	Float format:	Music Notation
8	8	
4	4f	
2	2f	
1	1f	
1/2	0.5f	
1/4	0.25f	
1/8	0.125f	
1/16	0.0625f	
1/32	0.03125f	
1/64	0.015625f	
1/128	0.0078125f	
1/256	0,00390625f	Custom.

To set the value of the note in seconds, just multiply the float format by corresponding pulse of BPM in seconds.
Example: a note that have ¼ value meaning 0.25 and the rhythm has 80 bpm.
Discover the pulse factor of BPM in seconds: 60/80 = 0.75 seconds.
The duration of this note on the rhythm: 0.25*0.75 = 0.1875 seconds.

Some BPM Values (most used):

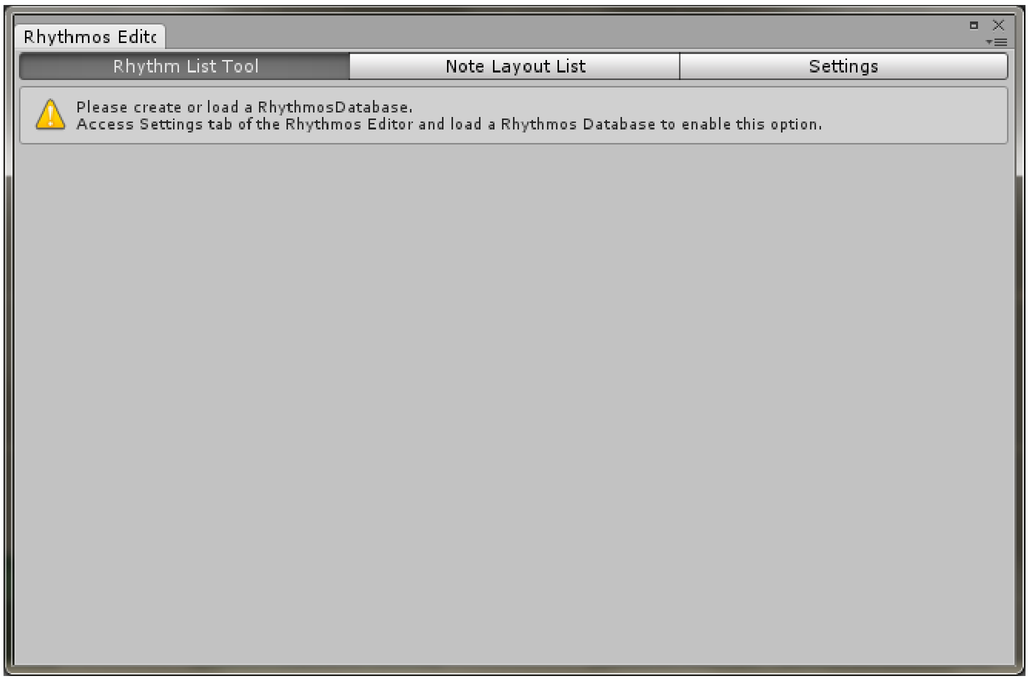
BPM	Seconds:
240	0.25s
180	0.33s
120	0.5s
80	0.75s
60	1s
40	1.5s
20	2s

3. Rhythmos Editor – Tutorial

The Rhythmos Editor is an extension to facilitate creation of rhythms. To open it go to the Tools > RhythmosEditor at the top bar of the Unity.

1. Starting:

When you open for the first time the tool, the following screen appears.



Please create or load a RhythmosDatabase.
Access Settings tab of the Rhythmos Editor and load a Rhythmos Database to enable this option.

You will see a message saying „*Please create or load a RhythmosDatabase. Access Settings tab of the Rhythmos Editor and load a Rhythmos Database to enable this option*“. That's means that RhythmosEditor don't have loaded any RhythmosDatabase file.

Go to Settings tab.

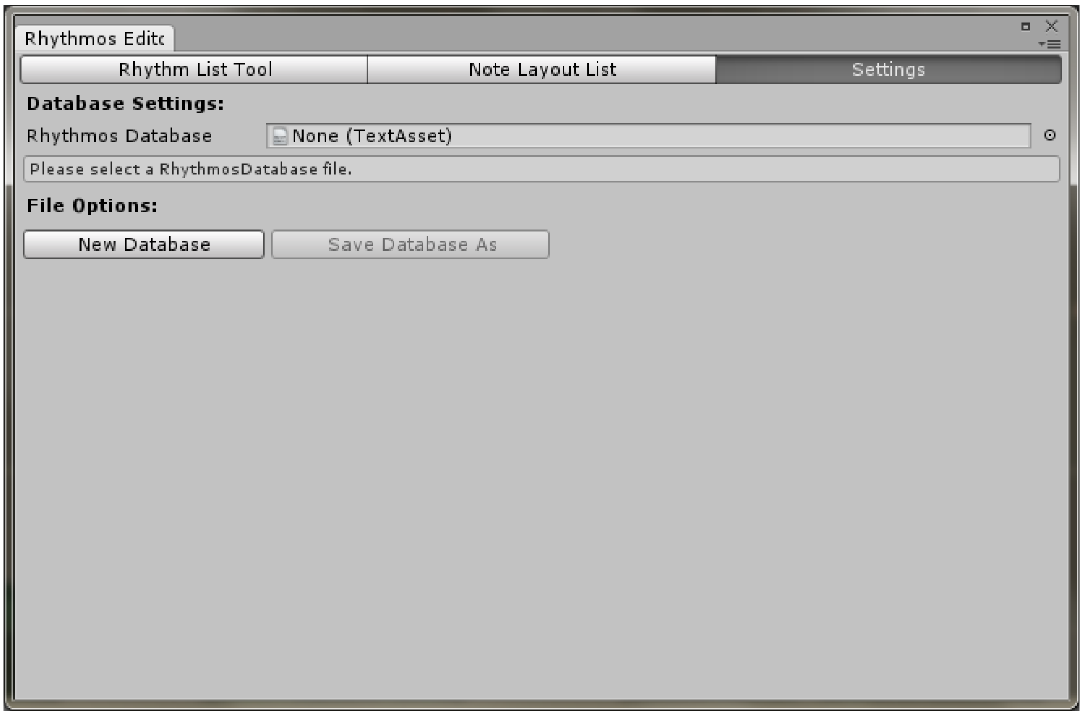
2. Settings Tab:

In the Settings tab you will see a *TextAsset* field just below a status message and file options. In the TextAsset if you have a RhythmosDatabase file you can select it there and all will work.

Otherwise in *File Options* select the option *New Database*. It will open a new window to save the New Database in some directory of your project. After that, you can start create rhythms.

Other options available:

If you have loaded a database you can:
- make a copy using the option *Save Database As.*
- set a Metronome *AudioClip* when you are editing rhythms.


Rhythmos Editc

Rhythm List Tool | Note Layout List | Settings

Database Settings:

Rhythmos Database: None (TextAsset)

Please select a RhythmosDatabase file.

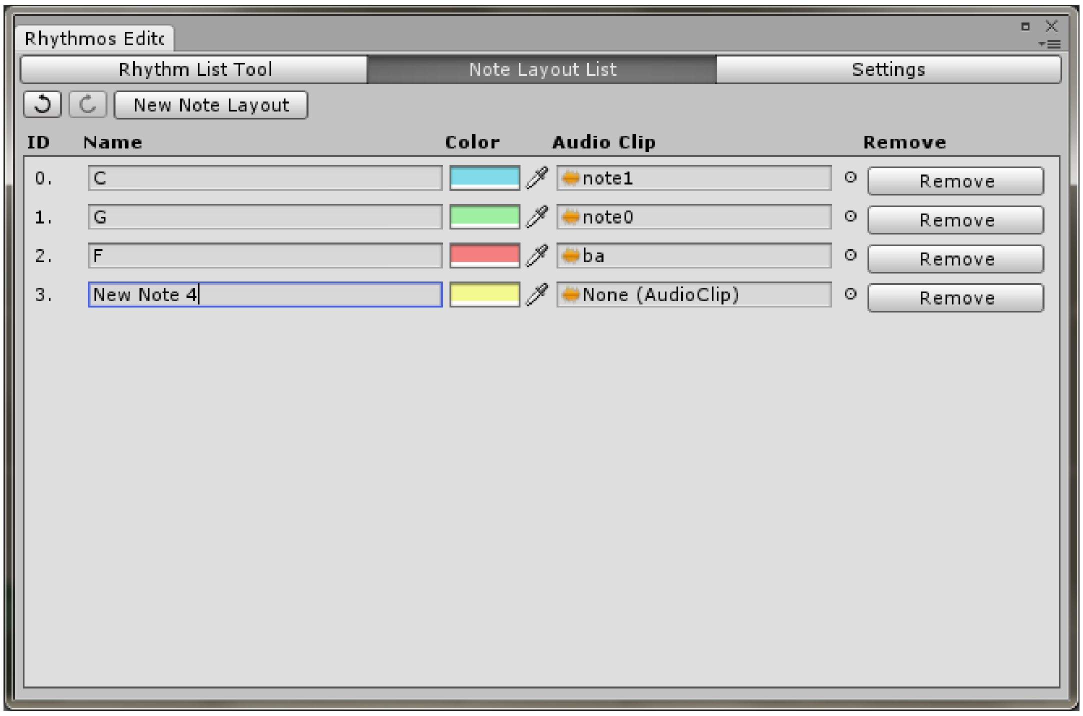
File Options:

New Database | Save Database As

3. Note Layout List tab:

Note Layout List tab lists all NoteLayout you've created. Remember that Note Layout is the way the RhythmosEngine can differentiate one note from other note. Note Layout is done by a name, an ID number, color for RhythmosEditor and an AudioClip to play the desired sound. In the top of window you will see the following options: Undo last action, Redo last action and New Note Layout to creating a new note layout. With Note Layout you can create different types of rhythms.

To create a Note Layout see the *4. Creating NoteLayout* topic.


Rhythmos Editc

Rhythm List Tool | Note Layout List | Settings

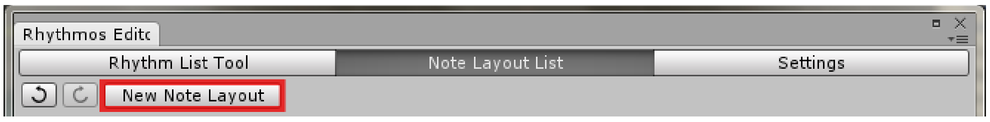
New Note Layout

ID	Name	Color	Audio Clip	Remove
0.	C	🟦	note1	Remove
1.	G	🟩	note0	Remove
2.	F	🟥	ba	Remove
3.	New Note 4	🟨	None (AudioClip)	Remove

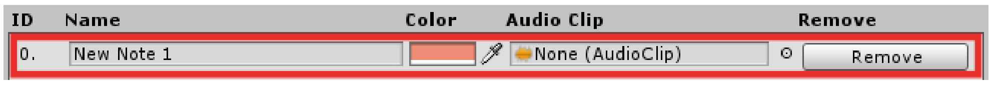
4. Creating NoteLayout:

Step-by-step: creating Note Layouts a necessary structure to play and create rhythms.

1. Click in the top button *New Note Layout*:

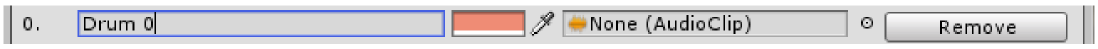


2. It will be appear new empty note on the list:



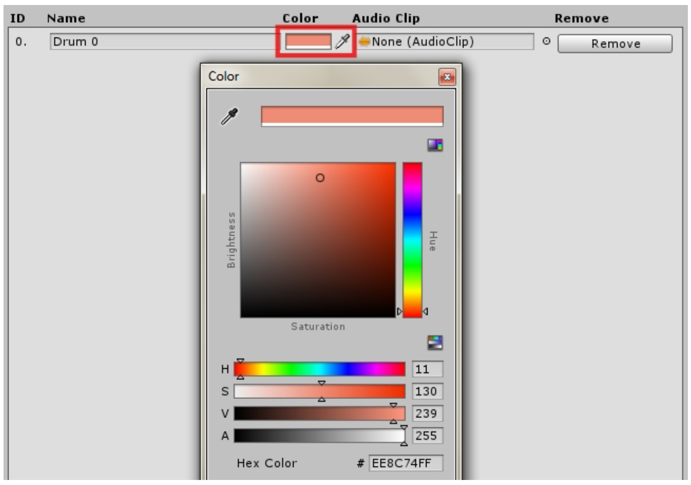
ID	Name	Color	Audio Clip	Remove
0.	New Note 1	🟥	None (AudioClip)	Remove

3. Write a *Name* for your Note Layout.



0.	Drum 0	🟥	None (AudioClip)	Remove

4. If you want, select a *Color* - Colors serve to distinguish notes when you are editing rhythms.

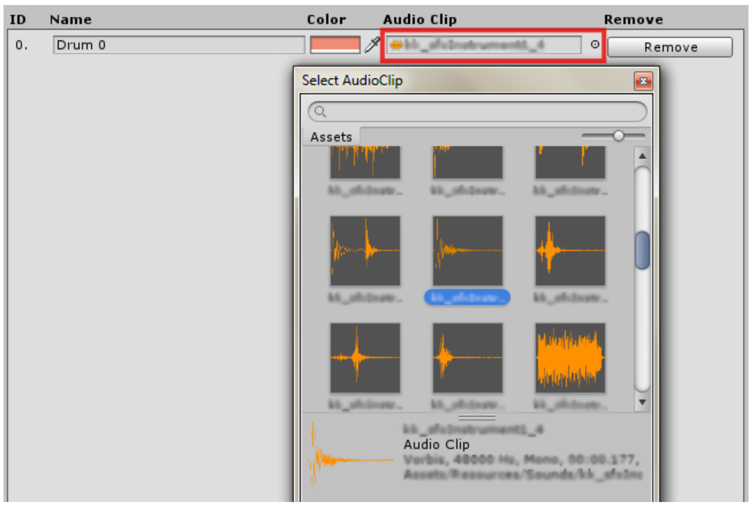


ID	Name	Color	Audio Clip	Remove
0.	Drum 0	🟥	None (AudioClip)	Remove

Color

H 11
S 130
V 239
A 255
Hex Color # EE8C74FF

5. Select an *AudioClip* – if the AudioClip setted to null then the note will be a rest. Is very recommended to choose short sounds.



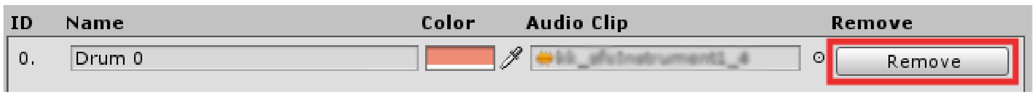
ID	Name	Color	Audio Clip	Remove
0.	Drum 0	🟥		Remove

Select AudioClip

Assets

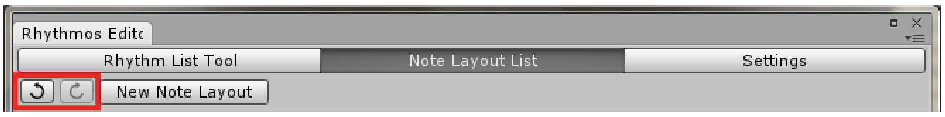
Audio Clip

6. You can Remove NoteLayouts. But be careful! This could damage the rhythms that use these note layouts.



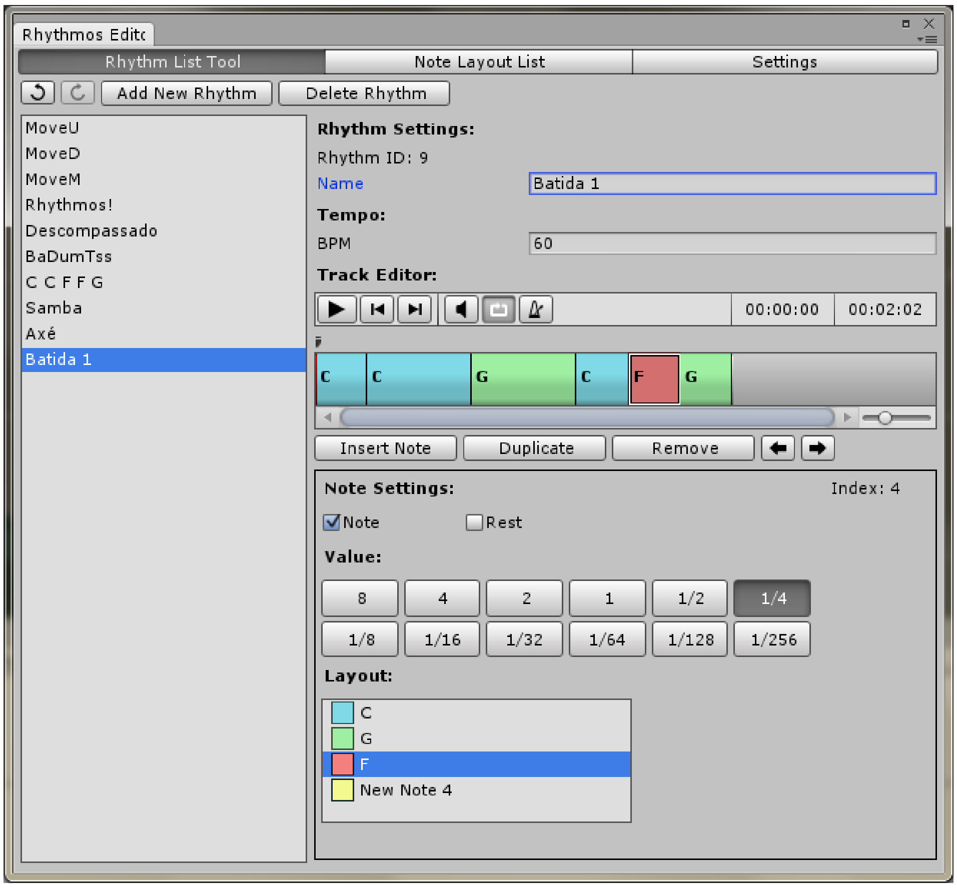
ID	Name	Color	Audio Clip	Remove
0.	Drum 0	🟥	[illegible]	Remove

7. You can use Undo or Redo buttons in the top window to re/set the last edition.



5. Rhythm List Tool:

This tab is the main feature of RhythmosEditor. This tab has a lists of all rhythms that you have created and it also has the rhythm track editor:
At the top there are the following options Undo, Redo, Add New Rhythm and Delete the selected rhythm. On the left is listed all the rhythms created according to the name of each.



Rhythm Settings:

Rhythm ID: 9

Name: Batida 1

Tempo:

BPM: 60

Track Editor:

00:00:00 00:02:02

Note Settings: Index: 4

[✓ Note] Rest

Value: 1/4

Layout:

- 🟦 C
- 🟩 G
- 🟥 F
- 🟨 New Note 4

When a rhythm is selected, the edit section is opened. This section has three divisions:

- Rhythm Settings: place to set name and tempo of rhythm.

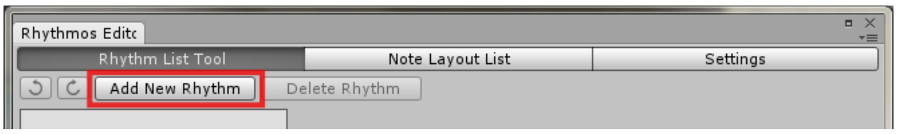
- Track Editor: displays the rhythm track time-line and options for Play/Stop rhythm, Select the first note, Select the last note, Switch to mute playback, Switch loop playback and Switch metronome mode.

Below the rhythm time-line have the option to Add Note.
When a note is selected new options will be show: Insert Note at selected place, Duplicate selected note, Remove selected note, Move selected note to left and Move selected note to right.

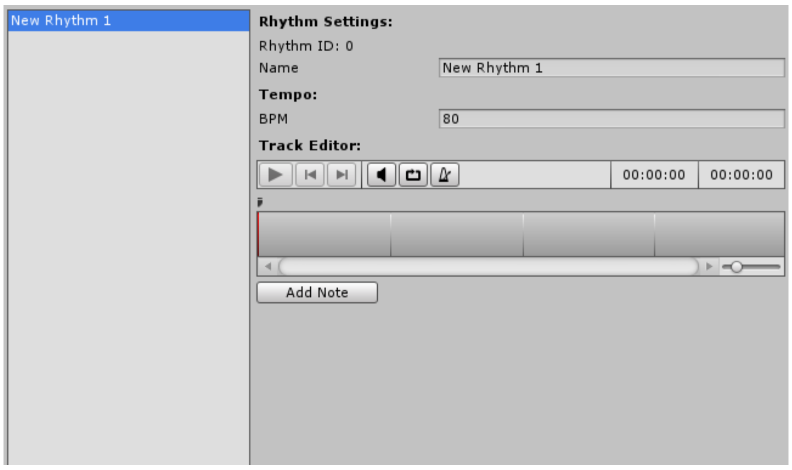
- Note Settings: When a note is selected you can set whether the Note is a rest or a common note. In both options the Note has a duration value and you can set the value in the Value grid. If want more information about note values, check the next session *4.1 Note and BPM Values.* Finally, if a note is not a pause, it can have a Note Layout.

6. Creating Rhythms

1. Click in the top button *Add New Rhythm*:



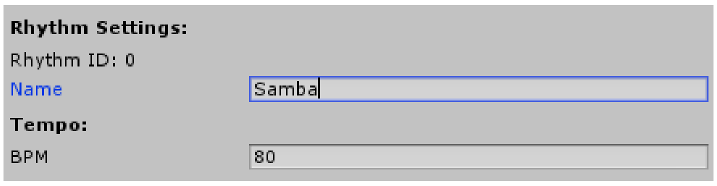
2. It will appear a new empty rhythm on the list and will automatically opens the rhythms editing session.



New Rhythm 1

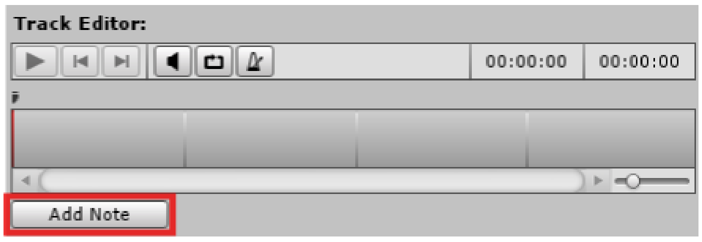
Rhythm Settings:
Rhythm ID: 0
Name: New Rhythm 1
Tempo:
BPM: 80
Track Editor:

3. In the Rhythm Settings, set a *Name* and *Tempo* for the rhythm. The BPM can never be zero but is recommended to not to use very high values.

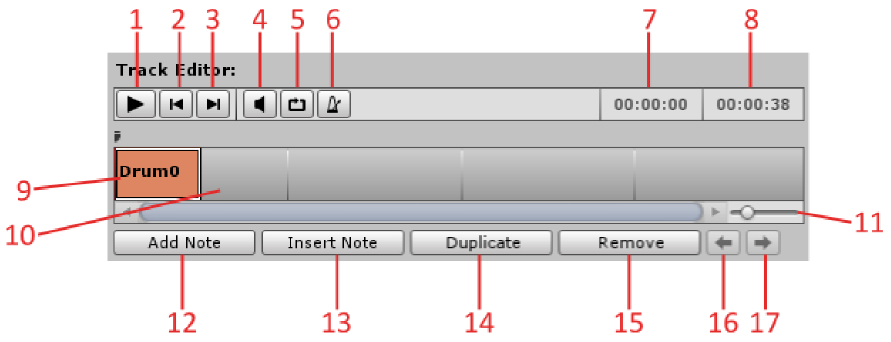


Rhythm Settings:
Rhythm ID: 0
Name: Samba
Tempo:
BPM: 80

4. Initially the options in the Track Editor are unavailable because the rhythm is empty. Click on *Add Note* to append a new note to the rhythm.



5. You will see a new note in the time-line. The general description of the elements of the track editor:

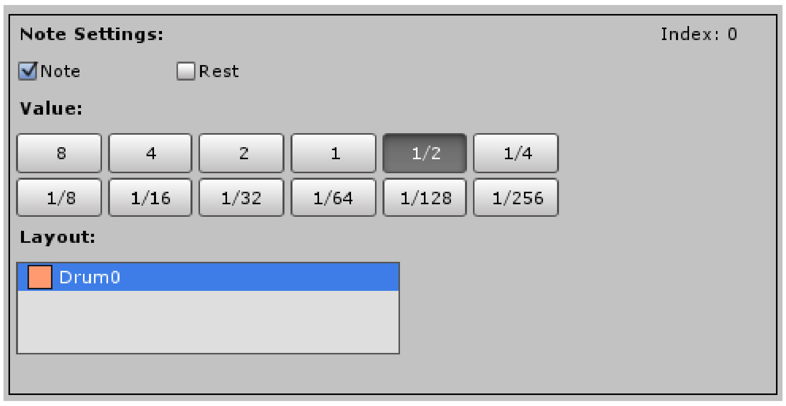


Playback options:
1. Play/Stop Rhythm
2. Select the first note.
3. Select the last note.
4. Mute/Unmute the playback.
5. Enable / Disable loop playback.
6. Enable / Disable metronome.
7. Current playback position. Minutes : Seconds : Miliseconds.
8. Rhythm duration.

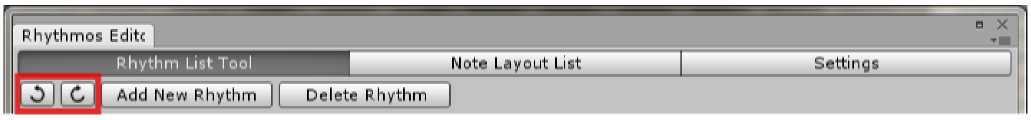
Time-line:
9. Representation of a note on the time-line.
10. Time-line.
11. Zoom slider of the time-line.

Rhythm options: (when you select a note, more options will available).
12. Add new note at the last position of rhythm.
13. Insert new note on the select note.
14. Duplicate the selected note.
15. Remove the selected note.
16. Move the selected note to left.
17. Move the selected note to right.

6. When you select a note, you can set configurations of the selected note in the *Note Settings* section.
- Is the note a *Rest*? When the note is a rest means an interval time in rhythm with no sound.
- *Value* is the duration of the note or rest. See the *2.1 Notes and BPM values* section for more information.
- *Layout* is the Note Layout that you just created. You can create more Layouts and change it. If the note is a rest, the Layout will be ignored.



7. You can use Undo or Redo buttons in the top window to re/set the last edition.



4. Rhythmos Engine – Implementation

This session has code examples as tutorial to implementing the RhythmosEngine in your game.

4.1. Using RhythmosDatabase

The main class RhythmosDatabase loads your rhythms that you have created. The code below is simplest way to implement in RhythmosDatabase as GameObject component to use in your game.

C# Code – Rhythms.cs

```
using UnityEngine;
using System.Collections;
using RhythmosEngine;

public class Rhythms : MonoBehaviour {

        RhythmosDatabase m_rhythmosDatabase;
        public TextAsset m_rhythmosFile;

        // Use this for initialization
        void Awake () {

                m_rhythmosDatabase = new RhythmosDatabase(m_rhythmosFile);

        }

        // Update is called once per frame
        void Update () {

        // Display in the console the amount of rhythms and note layouts.
                Debug.Log("Total of Rhythms: " + m_rhythmosDatabase.RhythmsCount.ToString());
                Debug.Log("Total of NoteLayout: " + m_rhythmosDatabase.NoteLayoutCount.ToString());

        }
}
```

4.2. Single-ton pattern for RhythmosDatabase

If your game use a lot of rhythms, is a good idea to make a global component of rhythms. Use a single GameObject that's in charge of our rhythms storage class.

C# Code – GlobalRhythms.cs

```
using UnityEngine;
using RhythmosEngine;
using System.Collections;

public class GlobalRhythms : MonoBehaviour {

        public RhythmosDatabase m_rhythmosDatabase;

        // reference
        private static GlobalRhythms m_instance;

        // get the instance of GlobalRhythms
        public static GlobalRhythms globalInstance
        {
                get
                {
                        if (m_instance == null)
                        {
                        // In Unity above of 4.2 or above use this line of code:
                        //_instance = GameObject.FindObjectOfType<MusicManager>();

                        m_instance = (GlobalRhythms)GameObject.FindObjectOfType(typeof(GlobalRhythms));

                        // Set at unity this GameObject can't be removed.
                        DontDestroyOnLoad(m_instance.gameObject);
                        }

                        return m_instance;
                }
        }

        void Awake()
        {
                if (m_instance == null)
                {
                        // If don't have a singleton and this component is the first, make this component
                        the singleton
                        m_instance = this;
                        // create the RhythmosDatabase
                        m_rhythmosDatabase = new RhythmosDatabase();

                        // load a RhythmosDatabase file from the Resources directory
                        m_rhythmosDatabase.LoadRhythmosDatabase("Files/RhythmosDatabase");

                        DontDestroyOnLoad(this);
                }
                else
                {
                        //Destroy another GlobalRhythms if already exist this singleton.
                        if (this != m_instance)
                                Destroy(this.gameObject);
                }
        }

        void Update() {

                // Display in the console the amount of rhythms and note layouts.
                Debug.Log(m_rhythmosDatabase.RhythmsCount);

        }
}
```

4.3. Creating rhythms in run-time

The following snippet creates a rhythm. remembering that when played using RhythmosPlayer a must have reference to some RhythmosDatabase. Don't forget to use `using RhythmosEngine;` in your code header.

C# Function: Create Rhythm

```
Rhythm CreateRhythm() {

        Rhythm rhythm = new Rhythm();

        // set a name
        rhythm.Name = "Jazzy Beat";

        // set a bpm
        rhythm.BPM = 120;

        // create notes
        // remember: layout index is a index reference to a RhythmosDatabase.
        Note note1 = new Note(0.5f,false,0);
        Note note2 = new Note(0.25f,false,1);
        Note note3 = new Note(0.5f,false,0);

        // append it
        rhythm.AppendNote(note1);
        rhythm.AppendNote(note2);
        rhythm.AppendNote(note2);
        rhythm.AppendNote(note3);

        // return a new rhythm
        return rhythm;
}
```

4.4. Playing Rhythms

The following small snippet just play a Rhythm from RhythmosDatabase.

C# Function: Play Rhythm

```
void PlayRhythm() {

        // play a rhythm from a loaded RhythmosDatabase
        RhythmosPlayer player = m_rhythmosDatabase.PlayRhythm("MyBeatifulRhythm",1f);
        // set playback loop
        player.loop = true;
        // should the GameObject of RhythmosPlayer destroy on end of playback?
        player.destroyOnEnd = false;

}
```

5. Classes and Functions

This section describes functions and classes of *RhythmosEngine.dll* to use during the game. To use the following structures use the below syntax in the header of your code:

```
using RhythmosEngine;
```

The classes and structs of Rhythmos Engine includes the following list:

Class RhythmosDatabase:

This is storage class for Rhythms and NoteLayouts list.
If your game is using a lot of rhythms look in the *5.4 Single-ton pattern for Rhythmos Engine* section.

Constructors:

```
public RhythmosDatabase();
```

Initializes the RhythmosDatabase instance – create necessary data structures.

```
public RhythmosDatabase(TextAsset textAsset);
```

Initializes the RhythmosDatabase instance and load a RhythmosDatabase (.XML) file.

Properties:

```
public int NoteLayoutCount;
```

get: size of the notes layout list.

```
public int RhythmsCount;
```

get: size of the rhythms list.

Public Functions:

```
public bool LoadRhythmosDatabase(string resourcePath);
```

Load a RhythmosDatabase from a resource path.
Returns true if was successfully load, false otherwise.
Usage example: LoadRhythmosDatabase(„Resources/Files/RhythmosDatabase.“);

```
public bool LoadRhythmosDatabase(TextAsset resourcePath);
```

Load a RhythmosDatabase from a resource path.
Returns true if was successfully load, false otherwise.

```
public void AddRhythm(Rhythm newRhythm);
```

Add a new rhythm to Database.

```
public void RemoveRhythm(int index);
```

Remove rhythm at index.

```
public void AddNoteLayout(NoteLayout newNoteLayout);
```

Add a new note layout to Database.

```
public void RemoveNoteLayout(int index);
```

Remove note layout at index.

```
public void ClearRhythmList();
```
Clear rhythm list.

```
public void ClearNoteLayoutList();
```
Clear note layout list.

```
public List<Rhythm> GetRhythmList();
```
Return the rhythms list.

```
public List<Rhythm> GetNoteLayoutList();
```
Return the note layout list.

```
public Rhythm FindRhythmByName(string name);
```
Finds a rhythm in the rhythm list by name.
Returns the founded Rhythm, null otherwise .

```
public Rhythm FindNoteLayoutByName(string name);
```
Finds a note layout in the note layout list by name.
Returns the founded NoteLayout, null otherwise.

```
public RhythmosPlayer PlayRhythm(string name, float volume = 1f, bool
destroyOnEnd = true);
```
Easy way to play rhythms from database. Finds a rhythm and play it.
If is a valid rhythm, returns RhythmosPlayer, null otherwise.

Class RhythmosPlayer:

A manager to play rhythms during run-time. Inherits from MonoBehavior.

Properties:

```
public RhythmosDatabase rhythmosDatabase;
```
get, set: RhythmosDatabase reference.

```
public bool loop;
```
get, set: is the rhythm looping?

```
public float volume;
```
get, set: the volume of the rhythm.

```
public int noteIndex;
```
get: the current index of the played rhythm.

```
public float trackPosition;
```
get: playback position in seconds.

```
public bool isPlaying;
```
get: is the rhythm playing?

```
public bool isPlaused;
```
get: is the rhythm plaused?

Public Functions:

```
public void Play();
```

Play the rhythm.

```
public void Stop();
```

Stop the rhythm.

```
public void Pause();
```

Pause the rhythm.

```
public void UnPause();
```

Unpause the rhythm.

```
public bool HasNotePlayed();
```

Get the if a note was played in the moment.

```
public Note GetCurrentNote();
```

Get the current note.

```
public AudioSource GetLastAudioSource();
```

Get the last AudioSource.

```
public static RhythmosPlayer PlayRhythm(Rhythm rhythm, RhythmosDatabase, float
volume = 1f, bool loop = false, bool destroyOnEnd = true);
```

If is a valid rhythm and valid database, returns RhythmosPlayer, null otherwise.

Class RhythmosUtility:

Utilities to use with rhythms: check if a rhythm matches with another rhythm and check rhythm precision useful for games that compare Player rhythm with Game rhythm.
Static class.

Public Functions:

```
public static bool CheckRhythmsMatch(Rhythm rhythmA, Rhythm rhythmB);
```

Check if a rhythm A is the same as rhythm B.
Returns true if was matched, false otherwise.

```
public static Rhythm CheckDatabaseRhythmsMatch(RhythmosDatabase rhythmDatabase,
Rhythm sourceRhythm);
```

Check if the source rhythm matches with some of rhythms from RhythmosDatabase.
Returns the matched Rhythm if was matched, null otherwise.

```
public static float CheckRhythmsPrecision(Rhythm rhythmSource, Rhythm
rhythmCompare, float failureRate);
```

Check the precision of source rhythm with the compare rhythm by a failure rate.
Returns a value 0.0 to 1.0 as a percentage of success.

<u>Class Rhythm:</u>

A representation of rhythm.

Constructors:

```
public Rhythm();
```
Initializes the Rhythm instance.

```
public Rhythm(string name, float bpm);
```
Initializes the Rhythm instance with a name and BPM.

```
public Rhythm(Rhythm rhythm);
```
Initializes the Rhythm instance cloning another rhythm instance.

Properties:

```
public string Name;
```
get, set: name of rhythm.

```
public float BPM;
```
get, set: tempo of rhythm.

```
public int NoteCount;
```
get: amount of rhythm notes.

Public Functions:

```
public void AppendNote(Note note);
```
Add a new note to the rhythm.

```
public void AppendNote(int layoutIndex, float noteDuration, bool isRest);
```
Add a new note to the rhythm.

```
public void InsertNoteAt(int index, Note note);
```
Insert a note at index.

```
public void RemoveNote(int index);
```
Remove a note by index.

```
public void ReplaceNote(int index, Note note);
```
Replaces a note at specific index.

```
public void SwapNote(int index, int swapIndex);
```
Swaps a note located in index from another not located in swapIndex.

```
public void Clear();
```
Clear the notes of rhythm.

```
public List<Note> NoteList();
```
Returns the list of notes of the rhythm.

```
public float Duration();
```
Returns the duration in seconds of the rhythm.

```
public float GetNoteAt(int index);
```
Returns a note of rhythm at index.

Struct Note:

A representation of note.

Constructors:

```
public Note(float duration, bool isRest, int layoutIndex);
```
Initializes the Note instance.

```
public Note(Note note);
```
Initializes the Note instance cloning another Note.

Variables:

bool isRest – if a note is a rest, the layoutIndex is ignored.
float duration – note duration in seconds.
int layoutIndex – index of RhythmosDatabase of a NoteLayout.

Class NoteLayout:

NoteLayout defines the attributes of a note: sound, name and color for RhythmosEditor.

Constructors:

```
public NoteLayout();
```
Initializes the NoteLayout instance.

```
public NoteLayout(string name);
```
Initializes the NoteLayout instance.

```
public NoteLayout(string name, AudioClip clip);
```
Initializes the NoteLayout instance.

```
public NoteLayout(string name, AudioClip clip, Color color);
```
Initializes the NoteLayout instance.

```
public NoteLayout(NoteLayout noteLayout);
```
Initializes the NoteLayout instance cloning another noteLayout.

Properties:

```
public string Name;
```
get, set: name of note layout.

```
public AudioClip Clip;
```
get, set: AudioClip of note layout.

```
public Color Color;
```
get, set: color of note layout.