Modular Rhythm Package

# Overview

The ‘**Modular Rhythm Engine**’ will be a package designed to streamline the production of rhythm games (E.G. Guitar Hero, Friday Night Funkin, StepMania, etc..).

When used, the package gives the user some user-friendly tools and lists that allow the user to do a range of things, such as adding in specific music tracks, changing the speed the set music track plays at, etc.

The package works using a custom namespace called “**EAudioSystem**”, which contains all of the functions, variables and scripts required for the package to work.

The package works around using **Scriptable Objects** containing data for each level (such as, the level name, the name of the song, the bpm, etc). after the package has been imported, a user can create a **Scriptable Object** by hovering over ‘**ScriptableObjects**’ in the **Create** menu, then selecting ‘**LevelData**’.

# Loading Level Data From Files

The EAudioSystem will be able to load data from JSON files, in a folder called “**Notemaps**” in the systems persistent data path. The information will be stored in two JSON files. The first JSON file containing the timing data for the Notemap, (to tell the a game when to spawn in notes), and the other file containing the levels metadata (e.g. song name, song artist, song BPM, etc..)

Once data has been loaded from a songs notemap and metadata files, the information is stored in an object called “**Notemap**”, storing the information ready to set it up into ‘**LevelData**’ ScriptableObjects.

Loading level information from files is handled using the ‘**NotemapLoader.cs**’ script in the systems ‘**framework**’ scripts folder.

The ‘**NotemapLoader**’ script contains three lists:

* “**notemapFiles**” – A list of the notemap objects created to hold each levels data.
* “**filePaths**” – A list of strings which contain the file path to each of the JSON files. containing the note timings data – NOT INCLUDING THE METADATA.

“**metaFilePaths**” – A list of strings which contain the file path to each of the JSON files containing the metadata for each level

* – NOT INCLUDING THE NOTEMAPS / TIMINGS.

## Functions

### “**SetUpFiles()**”

The “**SetUpFiles()**” function is the initializer for loading the information and data from the JSON files.

It starts by ensuring that the three lists are empty, then calls the “**GetNotemapFiles()**”” function.

Once the “**GetNotemapFiles()**” function has completed, a **FOR** loop that repeats an equal amount of times as the amount of variables contained in the “**filePaths**” list.

As it iterates through the **FOR** loop, it creates a new **Notemap** object, and set its data using the “**fileData()**” function, passing through the current iteration index of the loop. Then, finally, it adds the created **Notemap** object into the “**notemapFiles**” list.

### “**UpdateLevelsArray()**”

### “**GetNotemapFiles()**”

The **“GetNotemapFiles()**” function begins by iterating through all files within the **Notemaps** folder directory, then checks if each file has the “.json” file extension(/ file type), then checks whether or not the file is a metadata file, and if it isn’t, then it’ll add the file path to the “**filePaths**” list. Otherwise, if the file IS a metadata file, then it will instead add it to the “**metaFilePaths**” list.

### “**fileData()**”

The purpose of the “**fileData()**” function is to set up a temporary **Notemap** object, set its data using information read from a JSON file, then return the objects information.

First, in the “**fileData()**” function, a temporary **Notemap** object called “**notemapValues**” that will be used to store the data read from the files.

After creating the temporary **Notemap** object, we search the current JSON files directory for the matching WAV file, and set it as the audioclip for the **Notemap** object.

Once we have collected the **WAV** audio data, we then search the directory again for the matching metadata JSON file, then assign the data from it to the **Notemap** object (e.g. song name, song BPM, etc..).

Then, it utilizes StreamReader, creating a text reader, following the directory path of the current index in the “**filePaths**” list. Then, we utilize **JsonReader** ( from Newtonsoft.Json ) creating a Json Text reader, using the StreamReader we initialized.

Then, while using the newly initialized Json text reader (that we name “reader”), “**SupportMultipleContent**” is set to TRUE to allow multiple pieces of information can be read from the files without erroring.

Then, we iterate through all of the lines within the current notemap JSON file. If the data on the current like is not NULL, then we parse the information from a STRING to a FLOAT, and add the converted data into our “**notemapValues**” object we made before.

This loops until all lines of the file have been read.

Then, finally, all the information is returned.

# ‘LevelData’ ScriptableObjects

A ‘**LevelData**’ ScriptableObject can be added through the create menu, and is made using the ‘**ScriptableObjectHandler**’ script in the packages ‘**Framework**’ scripts folder.

# Required Additional Libraries

* TextMeshPro
* Newtonsoft

# Function List

* **SetUpFiles ( )**
* **GetNotemapFiles ( )**
* **fileData ( int )**
* **CreateLevelObject ( Notemap )**
* **UpdateLevelsArray ( IList<ScriptableObjectHandler> )**
* **CalculateSecPerBeat ( float )**
* **CalculateDSPSongTime ( )**
* **CalculateSongPosition ( float )**
* **CalculateSongPosInBeats ( float, float )**
* **CalculateSongLengthSeconds ( AudioClip )**
* **SetSoundSpeed ( AudioMixerGroup, AudioSource, float )**
* **ListAudioSources ( bool )**
* **TriggerSound ( AudioSource, bool )**
* **checkSizeMatch ( AudioSource[] )**
* **CheckIsInList ( AudioSource )**
* **CopySourcesList ( AudioSource[] )**
* **CopyLevelsList ( ScriptableObjectHandler[] )**
* **SetNotemap ( float[] )**
* **GetCurrentBeat ( )**
* **CheckNoteSpawn ( ScriptableObjectHandler, float, float, float, float, bool )**
* **SetPreSongTimer ( int )**
* **StartLevel ( ScriptableObjectHandler )**
* **AssignLevelDataFromObject ( ScriptableObjectHandler )**
* **GetPlayerScore()**
* **SetPlayerHighscore(int)**
* **GetPlayerHighscore()**
* **SetScoreMultiplier(int)**
* **GetScoreMultiplier()**
* **ResetScoreData()**
* **UpdatePlayerScore(bool, int, int)**