A MIDI Controller Mapper for the Built-in Audio Mixer in the Unity Game Engine

Pedro Lucas
Independent Researcher
Guayaquil, Ecuador
pedro.lucas.bravo@gmail.com

ABSTRACT

Unity is one of the most widely used engines in the game industry, several extensions have been implemented to increase its features in order to create multimedia products in a more effective and efficient way. From the audio development point of view, Unity has included in later versions an Audio Mixer, which facilitates the organization of sounds, effects and the mixing process in general; however, this module can be manipulated only through its graphical interface (GUI). This work describes the design and implementation of an extension tool to map parameters from the Audio Mixer to MIDI external devices, like controllers with sliders and knobs, to allow the developer to easily mix a game with the feeling of a physical interface.

Author Keywords

Audio Mixer, MIDI Controller, Unity3D, Game Development

CCS Concepts

•Applied computing → Sound and music computing; Computer games;

1. INTRODUCTION

In recent years the complexity of audio environments in video games has increased, mainly because there are more elements to be considered inside a virtual world, as well as the dynamic behaviours that demand sounds to improve feedback [6]. In this scenario, mixing all this audio becomes an important challenge for audio developers in the involved areas [7].

With the arrival of fast-prototyping game engines, there have been a significant increase in the production of games, leading developers to constantly look for ways to obtain the best results in terms of productivity and quality [4] [7].

Particularly, game audio development focuses on dynamic mixing to enrich the sound landscape according to the situations a player faces in a virtual environment [7]. In some cases, audio is intended to be used as the main resource for specific experiences as in [5], which focuses on an augmented reality approach based on sound for advance mixing in virtual acoustic environments.

NIME'19, June 3-6, 2019, Federal University of Rio Grande do Sul, Porto Alegre, Brazil.

Sound designers, who have none o little experience in programming, use tools, such as *FMOD* and *Wwise*, for building the dynamic sound environment [7]. In this process, they require to mix the sound elements according to the situations or performing pre-mixing operations that are facilitated by external devices, like MIDI controllers, such as keyboards o sets of knobs, sliders, and pads [8]. These tools are middlewares that have been used to implement audio features in game engines; however, all their capabilities are not fully integrated [8].

Thus, the improvement of game audio development is taken into account in this work by providing the details of the design and implementation of a tool for accelerating the mixing process directly into *Unity*, one of the most commonly used game engines in the game industry [13], and more suitable for beginners, who do not require a high-performance hardware to achieve the mixing results, and offers a straightforward process for developing mobile applications [1].

The tool proposed in this paper is a *MIDI mapper* for the *Unity Audio Mixer*, which helps link audio parameters from sounds and effects to knobs, sliders, or any other control from MIDI devices.

MIDI have been used in games as a medium to control objects and characters in real-time as in [3]. However, this proposal considers a tool for controlling the mixing process efficiently from the development point of view; that is, the inclusion of this tool as part of the *Unity Editor*. The mapper takes the advantages of mixing by *bus structures* (groups) in the Unity Audio Mixer; these structures can be composed of abstract designs as described in [14].

A discussion regarding this MIDI mapper is presented, together with future directions regarding the improvement of the tool and its impact in the game development community.

2. BACKGROUND

2.1 Unity Audio Engine

Unity is a real-time platform intended mainly for video game development. Around half of the world's games has been created through this platform, making it a leader in the game industry [13].

This work is focused in the *Unity Audio System* which supports most standard audio file formats, plays sounds in 3D space, applies filter effects to those sounds, records audio from a microphone, and more specific features for creating an immersive experience [12].

The audio system works as follows: Sound files are represented by Unity as *Audio Sources*, that can be placed in the game scene to reproduce audio. These sounds are received by an *Audio Listener*, which is a reference point of hearing; that is, audio will be played in 3D according to the position of *Audio Sources* in regard to the *Audio Listener*. *Audio*

Filters; such as echo, reverb, distortion, and others, can be used along with the sources, or a set of them, to modify the environment.

2.1.1 Built-in Audio Mixer

Unity implemented an *Audio Mixer* for mixing, applying effects, and perform mastering [11].

The layout for this tool is shown in (Figure 1). The mixer consists of *Groups*, *Snapshots*, and *Views*. A *Group* is a mix of sounds that controls *Audio Sources* as long as they are linked to that group. Volume attenuation, pitch correction, and effects insertion can be manage from a group. A *Snapshot* saves the state of the values of groups and it is useful when a whole mixing changes according to a location. A *View* is a graphical arrangement of the groups; that is, one or more groups can be hidden and that presentation can be saved as a view, which is useful for avoiding being overwhelmed when there are lots of groups.

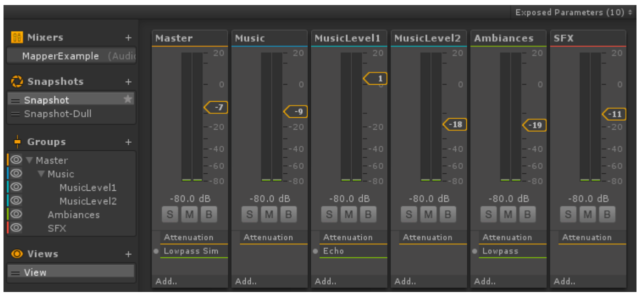


Figure 1: The Unity Audio Mixer.

In the top-right corner in (Figure 1) there is a drop-down list called *Exposed Parameters*. They are mixer parameters like volume, pitch, echo delay, and any other parameter attached to one or more groups. Unity is able to control these parameters by exposing them to script control for in-game-mode [10]. Thus, the value of the exposed parameters can be changed in real-time inside the application; also, they can be manipulated in the audio mixer directly using the editor GUI, and then be saved as an snapshot in editor-mode; however, an editor-script control is not possible, unless it is in the in-game-mode.

3. SYSTEM DESIGN

3.1 Mapper Overview

The proposed main goal is to map the parameters of the Unity Audio Mixer to a physical MIDI interface in order to control them in an easier way.

Unity does not support some features that are necessary to implement the mixer mapper, such as *MIDI communication* and *modification of exposed parameters through scripting in editor-mode.*

To include MIDI communication, the library *NAudio* [2] was integrated to Unity as a plugin in order to support MIDI features in Windows. *NAudio* is an open source .NET audio library, which is compatible with the Unity scripting language *C#*.

To modify mixer parameters through scripting, in editor-mode for real-time mixing, it was necessary to expose them; however, the modification of values for exposed parameters only works in the in-game-mode and are not saved when the application stops. To solve this problem, the mixer mapper modifies the values directly in the audio mixer file, hence the modifications are not lost. This strategy needs to parse the audio mixer file whose format is *YAML*, a data serialization language that is used by all Unity files.

The system overview is shown in (Figure 2). The developer uses the *Editor Mapper GUI* to assign MIDI controls to exposed parameters in the audio mixer; then, the *MIDI controller* can be manipulated to change the values of the parameters. The core of this process is the *Audio Mixer Mapper* in which all the logic resides. The mapper allows the user to match the parameters with the physical controls and save them permanently, by modifying the audio mixer YAML file through a *YAML Parser* module. In this way, the exposed parameters have a direct influence in the *Audio Sources* from the application according to the audio mixer management.

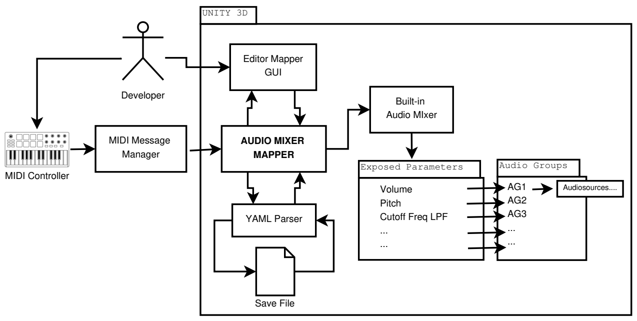


Figure 2: System overview.

3.2 Editor GUI Design

(Figure 3) shows the graphical user interface (GUI) for the mixer mapper. This custom interface was implemented following the standard layout of Unity editor windows. This image presents the main sections for the mapper: the *target mixer*, which is the audio mixer that contains the exposed parameters to be mapped; *MIDI devices* that shows the controllers connected to the computer where Unity is running; the *start snapshot*, which is the configuration that is going to be mixed; and, the *exposed parameters*, represented by sections with relevant values for the parameters to be mapped.

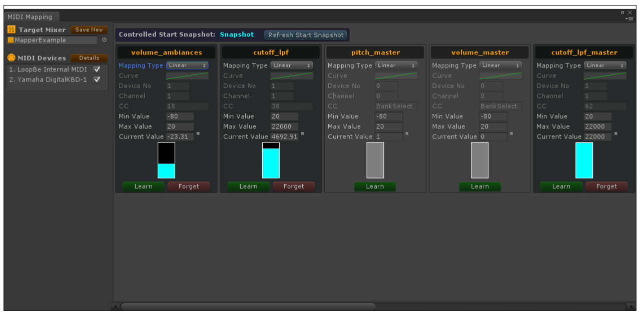


Figure 3: Mixer Mapper with some exposed parameters.

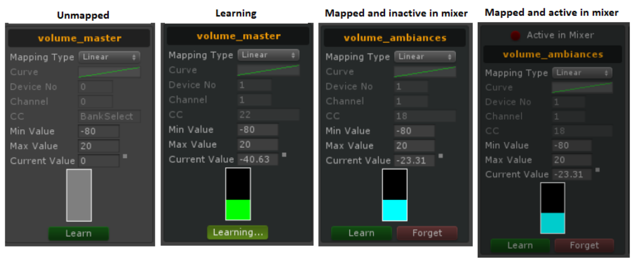
Depending on the state of the mapping process, an *exposed parameter section* can have one of the views presented in (Figure 4) described as follows:

- **Unmapped:** There is no MIDI control associated with the parameter until the user press *"Learn"* to assign a control.
- **Learning:** The tool is waiting for the input from a MIDI device; that is, if any knob, slider or other kind of control is moved in the device, it will be detected and you will see the green bar moving while you move

the control. By pressing the button *"Learning..."* the user exits from this state.

- **Mapped and inactive in mixer:** The parameter is now mapped to a MIDI control and it can be confirmed when the control is being moved and the cyan bar is changing. However, this will not affect the Audio Mixer yet because you need to run the application in the in-game-mode.

- **Mapped and active in mixer:** In this state, the words *"Active in Mixer"* appears on the top of the section, which means that the exposed parameter is reacting to the MIDI control. It can be observed when the control is being moved and both, the exposed parameter and the mapping cyan bar, are changing. This state only happens when the application is playing.



	Unmapped	Learning	Mapped and inactive in mixer	Mapped and active in mixer
				Active in Mixer
	volume_master	volume_master	volume_ambiances	volume_ambiances
Mapping Type	Linear	Linear	Linear	Linear
Device No	0	1	1	1
Channel	0	1	1	1
CC	BankSelect	22	18	18
Min Value	-80	-80	-80	-80
Max Value	20	20	20	20
Current Value	0	-40.63	-23.31	-23.31
	Learn	Learning...	Learn Forget	Learn Forget

Figure 4: States for exposed parameters.

An *exposed parameters section*, as shown in (Figure 4) four parameters are depicted, composed of the following attributes:

- **Name:** The given name to the parameter when it was defined in the Audio Mixer. (volume_master is an example).

- **Mapping Type:** The type of mapping regarding the MIDI control that depends on a curve. This attribute dictates the behavior of the control and how it behaves when it is manipulated. When a type is selected, the curve can be seen in the field below. The types are: *Linear, Logarithmic, Hyperbolic, Quadratic, Exponential, and Custom.*. For the implementation of the curves, the derivative of the functions corresponding to each curve was used in order to create the points and tangent lines that compose the curve.

- **Curve:** The curve that results from the selection of the mapping type. The custom curve enables this *Curve* field where the user is able to provide any curve, as shown in (Figure 5) by adding points and modifying tangent lines manually. This is a very flexible feature that allows to test unusual behaviours for mapping.

- **Device No.:** This attribute is the number of the device that is mapped with the parameter. If there is no mapping, it will be 0 (zero), otherwise it will be the number that you can see in the device panel to the left of the main window in (Figure 3).

- **Channel:** The MIDI channel that was learned in the mapping.

- **CC:** The MIDI control code that was learned in the mapping.

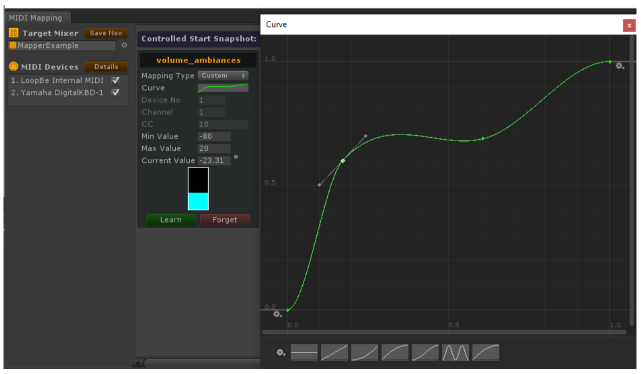
- **Min Value and Max Value:** These values can be modified and must have the limits of the parameters that are exposed. By default, the values are -80 and 20; it assumes that a *volume* parameter is going to be controlled.

- **Current Value:** The parameter that will be reflected in the Audio Mixer. It can be changed manually in order to give more precision if required. To its right side, there is a small red led that twinkles when the parameter is being manipulated by a MIDI device.

- **Parameter Bar:** This bar is a graphical feedback exclusive for the MIDI device; that is, it reacts to the movement of the MIDI control that is mapped to the parameter. Its color is green when it is learning and cyan when is already mapped. Also, it can be manipulated with the mouse pointer.

- **Learn Button:** When this button is pressed, the parameter enters to a learning state to catch a MIDI control and hides the rest of parameters.

- **Forget Button:** This button only appears when the parameter is mapped, and allows the mapping to be released and the controlling stopped.



MIDI Mapping
Target Mixer: MapperExample
MIDI Devices
1. LoopBe Internal MIDI ✓
2. Yamaha DigitalKBD-1 ✓

Controlled Start Snapshot:
volume_ambiances
Mapping Type: Custom
Device No: 1
Channel: 1
CC: 18
Min Value: -80
Max Value: 20
Current Value: -23.31

Figure 5: Custom curve for exposed parameter.

The following steps describe the use of the mapper:

1. Create the exposed parameters to be controlled via MIDI devices.

2. Open the tool using the menu MidiControllerMapping -> Open, which is a custom menu.

3. Assign your Audio Mixer to the *Target Mixer* field.

4. Connect a MIDI controller to the computer. The *MIDI Devices panel* will show information about the controller.

5. Map any parameter by pressing the *Learn* button and moving a control in the MIDI device.

6. The Audio Mixer only reacts to the mapping if the application is playing; also, the *Edit in Playmode* feature of Unity must not be used since the tool is responsible to save the Audio Mixer when the application is stopped, or by pressing the *Save Now* button.

A simple demo was developed as an example for the user. An screenshot of the demo is depicted in (Figure 6), which contains three main categories: *music*, *ambiances*, and *sound effects*, where audio sources are linked to the corresponding category, and the mixer mapper is set with exposed parameters ready to be used.

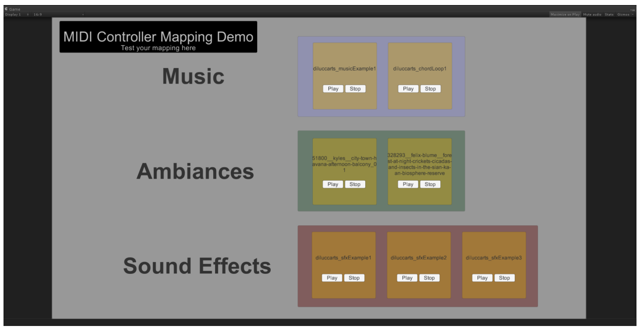


Figure 6: Demo of sounds to be mixed.

4. DISCUSSION

This tool is a useful add-on feature for the Unity editor, mainly for audio developers that are not familiar with programming. However, it requires previous knowledge about the built-in audio components from Unity since the audio mixer have to be assigned to the audio sources in the Unity game scene; that is, the developer needs to assign audio groups to each source in order to be included into the mixing.

Currently, the tool supports only the Windows operating system since *NAudio* library was built for this OS; mainly because of the dependency of MIDI features regarding the OS specific architecture.

It is important to note that this tool depends on the audio mixer file from Unity. By default, files in Unity are binary, but it can be changed in order to be more readable by humans. In this case, the files can be saved with the YAML format; however, it makes the Unity project to consume more disk space, which can be a considerable problem for big projects. The tool can avoid the dependency of this file when, in future releases, Unity enables the script control for exposed parameter in editor-mode.

In terms of functionality, the tool can be seen as a *meta-interface* which allows the development of audio and musical interfaces with specific purposes, such as sound synthesizers, interactive instruments based on touch-screens, even in instruments that rely on MIDI devices. All of them can take the advantage of the features provided by a game engine that supports graphics, physics, audio capabilities and more.

The mapper has been uploaded to the *Unity Asset Store* [9] and it is available to Unity developers.

5. CONCLUSIONS

This work presents a Unity audio extension for its audio mixer to map parameters to a MIDI controller. The proposed design considers the inclusion of a MIDI message manager to connect external devices with the computer that runs the Unity editor, which allows the use of physical controls through a graphical user interface (GUI) implemented under the Unity standards.

The tool enables audio developers to mix a video game in real-time with the feeling of physical controls, which is an easier and productive way to enhance the audio environment in a virtual world. This MIDI extension is intended to be part of the development work-flow of a video game or any other application implemented in Unity, enabling composers and sound designers to work with familiar hardware interfaces.

The Unity engine needs to update some features regarding its audio mixer in order to improve the mapper for large projects; however, the tool can be used for small and medium projects, offering an optimal work for audio development.

For future work, the mapper will be extended to Mac and Linux operating systems supported by Unity. Also, productivity tests will be performed to explore the improvements in the video game development process regarding dynamic audio scenes, as well as the conduction of tests to assess whether a developer prefers the use of the classical GUI or the physical control provided by the mapper.

6. ACKNOWLEDGMENTS

The author would like to thank Dr. Enrique Peláez, professor at ESPOL University, for his constructive criticism of the manuscript.

7. REFERENCES

[1] E. Christopoulou and S. Xinogalos. Overview and Comparative Analysis of Game Engines for Desktop and Mobile Devices. *International Journal of Serious Games*, 4(4):21–36, 2018.

[2] M. Heath. NAudio Library. Retrieved from: https://github.com/naudio/NAudio, 2014.

[3] J. Holm, J. Arrasvuori, and K. Havukainen. Using MIDI to Modify Video Game Content. *Proceedings of the 2006 International Conference on New Interfaces for Musical Expression (NIME-06)*, pages 65–70, 2006.

[4] H. A. Mitre-Hernández, C. Lara-Alvarez, M. González-Salazar, and D. Martín. Decreasing Rework in Video Games Development from a Software Engineering Perspective. In *Advances in Intelligent Systems and Computing*, volume 405, pages 295–304. Springer International Publishing, 2016.

[5] N. Moustakas, A. Floros, E. Rovithis, and K. Vogklis. Augmented Audio-Only Games: A New Generation of Immersive Acoustic Environments through Advanced Mixing. In *Audio Engineering Society Convention 146*, mar 2019.

[6] B. Schmidt. Interactive Mixing of Game Audio. In *Audio Engineering Society Convention 115*, oct 2003.

[7] A. E. Society. Audio for Games - Conference Report. *AES 49th International Conference*, 61(6), 2013.

[8] G. Somberg. *Game Audio Programming.* A K Peters/CRC Press, sep 2018.

[9] Unity. Unity - MIDI Controller Mapping for Audio Mixer. Retrieved from: https://assetstore.unity.com/packages/tools/audio/midi-controller-mapping-for-audio-mixer-69554, 2015.

[10] Unity. Unity - Exposed AudioMixer Parameters. Retrieved from: https://unity3d.com/es/learn/tutorials/topics/audio/exposed-audiomixer-parameters, 2018.

[11] Unity. Unity - Manual: Audio Mixer. Retrieved from: https://docs.unity3d.com/Manual/AudioMixer.html, 2018.

[12] Unity. Unity - Manual: Audio Overview. Retrieved from: https://unity3d.com/unity, 2018.

[13] Unity. Unity Game Engine - Features. Retrieved from: https://unity3d.com/unity, 2018.

[14] D. Zlobin. *Audio Design in Mid-Core Mobile Games.* PhD thesis, Aalto University, 2018.