New Utility Classes and Sketches for Developers and Sound Designers in CsoundUnity

Mateo Larrea Ferro[1] and Caio M. Jiacomini[2] ⋆

Berklee College of Music
mlarreaf99@gmail.com caiojmini@gmail.com

Abstract. The CsoundUnity wrapper brings the real-time synthesis and signal processing power of Csound to applications created with Unity. Despite the various benefits that it provides, the lack of an extended set of examples and/or tutorials presents a challenge for those who are new to the workflow. In the following paper, we present a series of C# utility classes that will facilitate the creation of new models and applications by showing what is possible. These include scripts/classes that assist in the formatting of Channel and ScoreEvent data, getting Transform and RigidBody data, establishing trajectories for the spatialization of Audio Sources, processing spectrum data for audio reactivity, and processing real-time input from a microphone or AudioClip. Essentially, this collection aims to exhibit an accessible interface for those who are not Unity or Csound experts - abstracting repetitive methods and providing a layout for the further development of 'sketches' and fully-implemented applications.

Keywords: CsoundUnity, C# Classes, Csound, Unity

1 Context: The Power of CsoundUnity

In late 2015, Cabbage creator Rory Walsh and a group of developers[1] built a wrapper to bridge Csound and Unity[2]. The connection of these two enables a simple way of doing real-time synthesis and digital signal processing inside the Unity engine. What would normally involve hundreds of C# lines of code (using the Unity Scripting API) is replaced with a single CsoundUnity instance that gives Unity access to over 1500 Csound opcodes. With the correct settings, this tool allows building to a variety of platforms, including (and not limited to): Android, Mac, PC, iOS, etc. Despite the significant advantage of utilizing the wrapper to handle the audio-related aspects of a project, the number of applications that use it is limited. Three main factors have been identified to explain this: 1) The lack of examples/sketches makes it hard to contemplate what the framework permits. 2) Not all Unity developers are familiar with the

⋆ Dr. Richard Boulanger, Noah Leong, and Pedro Sodre

[1] 5 additional contributors: Giovanni Bedetti, Bernt Isak Wærstad, Charles Berman, Hector Centeno, and NPatch

[2] Inspired by Richard Henninger's Csound6Net .NET wrapper

Csound syntax, and not all Csound developers are familiar with Unity's C# syntax. 3) The onboarding process is not simple enough for beginners.

2 Context: The Current State of Game Audio Workflow

As of the time of this writing, the most prevalent way of creating audio for games is by authoring and playing back pre-rendered audio files. Sound designers and composers typically create content in their digital audio workstations, export it as an audio file, and import those to the game engine as an asset to be played back at the appropriate places.[3]

About the Unity game engine itself, levels and scenes are created by populating them with *Game Objects*, which are essentially vessels that can represent visuals (like a 3D model or a particle effect) and contain behaviors (like playing audio or interacting with gravity). Behaviors can be assigned to *Game Objects* in one of two ways: either by using Unity's native *components* (like the *Rigidbody*[4] component) or by writing custom scripts with the C# language. Both native components and custom scripts can then be assigned to *Game Objects* through the *Inspector*[5] window.

3 Utility Classes

In this paper, the first two issues are tackled by asking: 1) How to encourage certain design patterns in developers? And 2) How to facilitate the power of CsoundUnity for those who are more concerned with sound than with the code? We have created a series of C# classes that provide a clear layout of what is possible, abstract repetitive methods, and present an accessible interface (through the Unity Inspector) for those who are not Unity or Csound experts. These include the following scripts/classes: *CsoundChannelRange (Scriptable Object)*, *CsoundChannelValue (Scriptable Object)*, *CsoundScoreEvent (Scriptable Object)*, *CsoundSender*, *CsoundTransformAndPhysicsSender*, *CsoundMap*, *Spatializer*, *Real-Time Input Processing, and Audio Reactive.*

4 Scriptable Object Classes For Holding Data

Scriptable object classes provide a way to create custom assets[6] in Unity to hold data. These classes were made to allow developers a degree of modularity when

[3] Creating audio procedurally is still a niche and games that do it usually come up with their own bespoke solutions.

[4] "Rigidbodies allow GameObjects to act under the control of the physics engine." - Unity Manual[1]

[5] The Inspector window allows the user to "view and edit properties and settings for almost everything in the Unity Editor" - Unity Manual

[6] An asset is any file that's used in a Unity project. They can represent visual and audio elements or more abstract concepts such as numeric data, which is what our scriptable objects represent.

defining how data is communicated from Unity to Csound. We created three classes here:

CsoundChannelValueSO: allows users to define an array of Csound channels with fixed values that can be assigned during runtime.

CsoundChannelRangeSO: allows users to define an array of Csound channels, with a minimum and a maximum value range that can be interpolated or randomized during runtime.

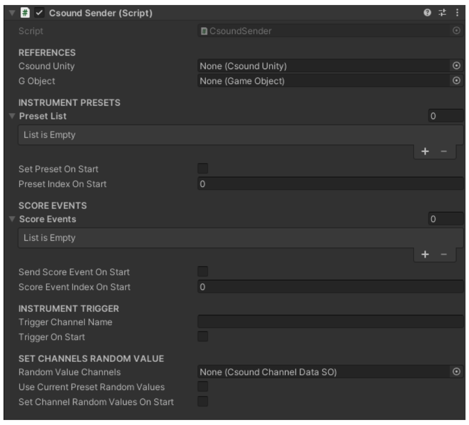
CsoundScoreEventSO: allows users to define Csound score events in a modular way, allowing the possibility to modulate each p-field independently during runtime.

5 Main Classes

5.1 CsoundSender

This class acts as a bridge between the scriptable object classes described above and Csound, using the CsoundUnity component to handle the communication. It contains functions that allow developers to set instrument presets, send score events using the *CsoundScoreEventSO* class, methods to toggle a Csound channel between a value of 1 and 0 as a way to implement instrument triggers, as well as using the *CsoundChannelValueSO* and *CsoundChannelRangeSO* to set channels to both fixed and random values.

Relevance: This class provides an interface to define general data that can be passed to a Csound instrument as well as methods to pass that data, minimizing the amount of extra code that needs to be written to start working with CosundUnity.



Csound Sender (Script)

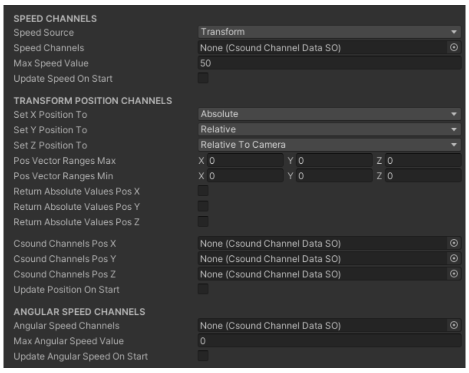
Script	CsoundSender
REFERENCES	
Csound Unity	None (Csound Unity)
G Object	None (Game Object)
INSTRUMENT PRESETS	
Preset List	0
List is Empty	
Set Preset On Start	☐
Preset Index On Start	0
SCORE EVENTS	
Score Events	0
List is Empty	
Send Score Event On Start	☐
Score Event Index On Start	0
INSTRUMENT TRIGGER	
Trigger Channel Name	
Trigger On Start	☐
SET CHANNELS RANDOM VALUE	
Random Value Channels	None (Csound Channel Data SO)
Use Current Preset Random Values	☐
Set Channel Random Values On Start	☐

Fig. 1. Unity's inspector view of the CsoundSender Class.

5.2 CsoundTransformAndPhysicsSender

This class takes values from a GameObject's *Transform*[7] and *Rigidbody* components, scales the data to a value range defined with a Csound Channel Range scriptable object asset, and passes the scaled value to a Csound channel. It can pass values based on the GameObject's position, rotation, and scale values from the Transform, as well as speed and angular speed from the *Rigidbody* component.

Relevance: This class provides a way to translate an object's movement in Unity into data for Csound, streamlining how movement and interactions can be used to shape and morph Csound instruments.



SPEED CHANNELS	
Speed Source	Transform
Speed Channels	None (Csound Channel Data SO)
Max Speed Value	50
Update Speed On Start	☐
TRANSFORM POSITION CHANNELS	
Set X Position To	Absolute
Set Y Position To	Relative
Set Z Position To	Relative To Camera
Pos Vector Ranges Max	X 0 Y 0 Z 0
Pos Vector Ranges Min	X 0 Y 0 Z 0
Return Absolute Values Pos X	☐
Return Absolute Values Pos Y	☐
Return Absolute Values Pos Z	☐
Csound Channels Pos X	None (Csound Channel Data SO)
Csound Channels Pos Y	None (Csound Channel Data SO)
Csound Channels Pos Z	None (Csound Channel Data SO)
Update Position On Start	☐
ANGULAR SPEED CHANNELS	
Angular Speed Channels	None (Csound Channel Data SO)
Max Angular Speed Value	0
Update Angular Speed On Start	☐

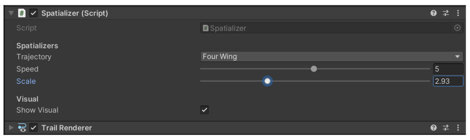
Fig. 2. Unity inspector of CsoundTransformAndPhysicsSender Class (partial view)

5.3 Spatializer

The Spatializer Class enables real-time displacement of a GameObject based on chaotic strange attractors[8], parametric equations, and circular motion. The class allows the user to choose from a series of possible three and two-dimensional trajectories, set the speed, and scale vectorial values at any moment of the simulation. The result is a moving Audio Source that accounts for changes in azimuth and zenith - meaning it relies on the Head Related Transfer Function (HRTF) from either Unity or Csound. Additionally, the class comes with an optional trail renderer that exhibits the path of the moving object through time.

Relevance: The Spatializer Class encourages the use of space as a compositional tool. Its purpose is to facilitate the animation of 3D acoustical objects and emulate periodic and aperiodic motions.

[7] "Every object has a Transform component. It's used to store and manipulate the position, rotation, and scale of the object" - Unity Manual
[8] Lorenz, Four Wing, Rossler, and Chen



Spatializer (Script)

Script	Spatializer
Spatializers	
Trajectory	Four Wing
Speed	5
Scale	2.93
Visual	
Show Visual	✓

Trail Renderer

Fig. 3. View of the Unity inspector of the Spatializer Class.

5.4 Real-Time Input Processing

The Real-Time Input Processing Class enables the use of external audio input devices to be used in Csound instruments. This opens up possibilities for real-time digital signal processing that utilizes the vast collection of Csound opcodes. The class also includes an option to sample incoming audio and load it into an f-table.

Relevance: This class is particularly useful for live performance and adding a degree of interactivity to applications.

5.5 Audio Reactive

The Audio Reactive Class allows the animation of any GameObject based on a Fast Fourier Transform analysis. In essence, the spectrum data (of virtually any sound) is obtained through a native Unity method called GetSpectrumData. The benefit, however, lies in two main aspects: 1) frequency bands can be segregated - meaning the user can make the object uniquely react to the desired frequency distribution, and 2) it connects the spectrum data to the scale, mesh, material, or shader of the object without the need of writing any lines of code.

Relevance: This class is particularly relevant for the animation of characters, non-player characters, environmental elements, and the creation of interactive and linear visualizers.

6 Examples created using these classes

Some of the current sketches we have include Trapped in the Garden, Sonic Sketcher, Chaos Spatializers, Collision-Based Sampler, and Blowing Instrument. All of these will become available via GitHub in the near future.

7 Future Plans and Conclusions

Utility Classes: The intention is to create four new classes in the near future. 1) A class that takes care of communication protocols such as Open Sound Control (OSC) or Serial. 2) A class that takes advantage of Normcore's infrastructure[9] to allow the creation of audio-based multiplayer experiences. 3) A class that

[9] Multiplayer room-based service

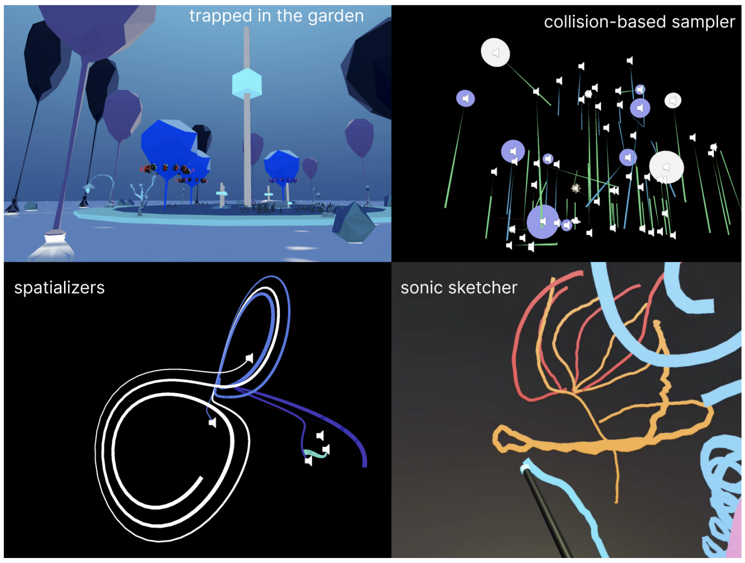


Fig. 4. Applications created using the library

connects Emotiv and Muse EEG devices to CsoundUnity. And 4) An easy-to-use class for physical models and additive synthesis.

A set of Sketches/Examples: One of the main motivations behind creating these utility classes consists in facilitating the development of new sketches/examples and applications. We are focused on a) the development of fun/accessible functional models that developers can take as the starting point for their own inventions and b) the creation of a VR-based tool for sound design and audio implementation inside of Virtual Reality.

Video Series for learners: Once the examples are done, the intention is to create 10-12 video tutorials that show how to code some of the 'sketches' from scratch. Inspired by the work of people like Daniel Shiffman[10], the tutorials will show learners how to recreate the models but also encourage their expansion through challenges.

[10] Host of the Coding Train Youtube Channel

References

1. Unity manual: `https://docs.unity3d.com/Manual/index.html`