Dynamic and Reactive Audio Synthesis in Video Game Engines by OSC Communication with MaxMSP

Landon P. Viator
Louisiana State University
340 E. Parker Blvd
Baton Rouge, Louisiana
landonviator@gmail.com

ABSTRACT

During the development of video games, whether it be 2D, 3D, or virtual reality, there are many audio components that need to be created and implemented. However, game engines like Unity or Unreal are limited in their power to process audio. These game engines are all but limited to simply triggering audio files to play back; they do not have to ability to synthesize audio. This research has focused on utilizing the powerful object-oriented audio programming environment MaxMSP along side a game engine by communicating via OSC messages. This communication between a game engine and MaxMSP would allow events in the game to facilitate dynamic and reactive synthesis in real time without the limited constraints of previously recorded audio or MIDI events.

1. METHODS AND MATERIALS

The focus of this research was to find a way to facilitate dynamic and reactive procedural audio synthesis inside of a video game engine. For this research I used Unity as the game engine to develop a small scale 3D video game as a proof of concept test. Unity contains a work environment called the *scene* which acts as a visual interface for the developer to drag and drop game assets such as objects, 3D models, scripts, and any other game components needed to build the environment and mechanics of a video game. Any Integrated Development Environment can be used to write scripts for Unity; I used Visual Studio. For the audio aspects I chose to use MaxMSP.

1.1 Unity

During the point in development when audio events are added to a game in Unity, there are a few tools available, but the most important is a component called an *audiosource* which is a component that a developer can drop in a sound file and edit a few attributes that affect playback; these attributes are shown in figure one. These attributes are all public variables that can be manipulated by scripting. While this functionality is very important, the developer is limited to only these attributes and cannot facilitate audio synthesis in any way; they are limited to triggering sound files to playback. This is very useful for triggering sound effects such as gunshots or footsteps, but cannot be used to dynamically synthesize audio based on game events. One example of dynamic synthesis would be a granular synthesizer emulating wind sound live based on previously programmed audio synthesis that can change timbre, speed, or pitch randomly over time or based on weather patterns in the game or even change the wind sound based on the size/type of environment that the player is in. The way I have done this is by connecting Unity to MaxMSP allowing MaxMSP to handle the audio synthesis/digital signal processing (DSP) based on information from Unity.

1.2 My Game

In the game that I created for this project, I have a first-person character for the player to control. The player has a small area to explore that contains a shoreline, a grass-covered dirt plot, some tress, a bonfire, and rising smoke in the back part of the area. The player has a health bar and can be hurt; losing health for walking too close to the fire or smoke. The player's XYZ position is sent out as OSC data that I receive inside of MaxMSP and use to change sound presets in the *nodes* object. When the game is started, it triggers a bool (true or false variable) that switches the MaxMSP program on or off according to whether the game is started or not. Finally, the player's current health is also sent to MaxMSP to use for changing the audio bit rate; the lower the health, the lower the bit rate. Lowering the bit rate of the audio distorts and destroys the sound quality of the audio; when the health goes down, the music sounds worse and when the health goes back up over time, the music starts to sound better. The game is shown in figure two.

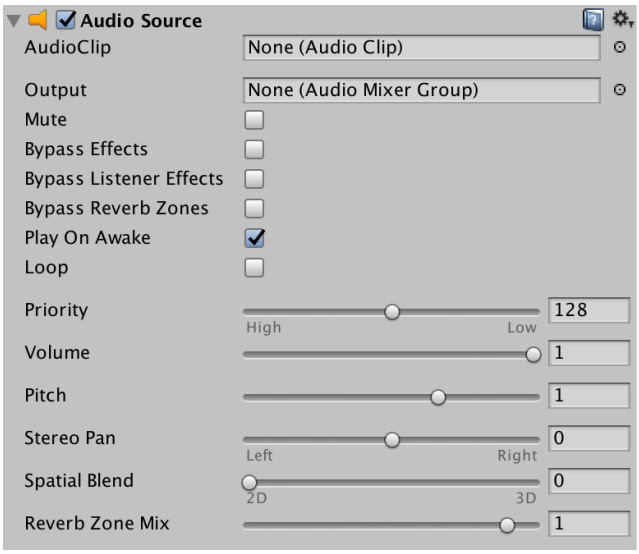

Figure 1: AudioSource component in Unity.

NIME'18, June 3-6, 2018, Blacksburg, Virginia, USA.

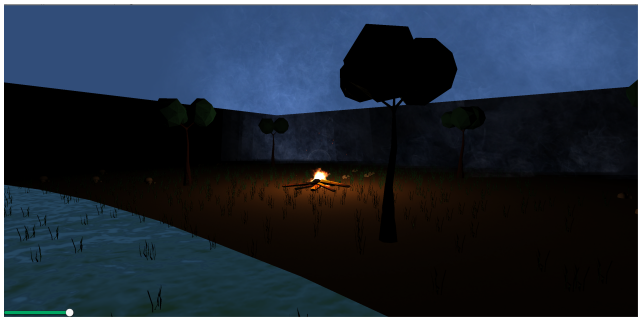

Figure 2: A view of my game.

1.3 OSC

OSC stands for Open Sound Control. It is a protocol with an endless amount of uses but is more commonly used for networking between computers, network systems, or multiple applications on one machine. [5]

1.4 MaxMSP

MaxMSP (Max for short) is an object-oriented programming language/environment for audio DSP (digital signal processing). Max facilitates audio synthesis, signal routing, audio processing, and many other uses. The Max program that I created for this research is shown in figure three. In Max, there is an object called the *udpreceive* object which I used to capture OSC data from Unity. By setting the OSC port on the *udpreceive* object to the same port number as the Unity OSC out port, which is port 6161, all outgoing OSC data from Unity is captured by the *udpreceive* object in Max. After the *udpreceive* object, I needed a way to decipher which set of OSC data I was receiving and route it to the correct place. The *route* object in Max allowed me to do this. All of the outgoing OSC data is divided from Unity is named, like the character's position update named *UpdateXYZ*. By placing a *route* object named *route /UpdateXYZ* after the *udpreceive* object, it allowed me to separately capture this XYZ data and use that data as I wanted in Max. This same *route* object process was used for the triggering of Max on and off by the game starting in Unity and for the player's health amount.

2. CONCLUSIONS

The OSC communication between Unity and Max was successful. When the player walks around the game environment, its' OSC data is sent to Max and the dial of the *nodes* object; the dial moves around the *nodes* object mirroring the player's movement. This changes the audio presets from the *nodes* object; this allowed me to program different areas with different audio presets. These audio presets all use granular synthesis as the sound synthesis and dynamically and procedurally change over time. This means that the audio is always unique unlike playing back sound files which is the standard way to work with audio in Unity.

3. DISCUSSION

To take this research further, I am building a more substantial game in Unity with a much larger environment and more standard game elements and mechanics such as artificially intelligent non-playable characters all with dialogue,

a player attribute system, a player inventory, a larger city, and different sound engines creating audio for different areas, houses, weather, and the time of day. If this project is successful, I believe that it could be the stepping stones to a better work-flow for creating audio for video games.

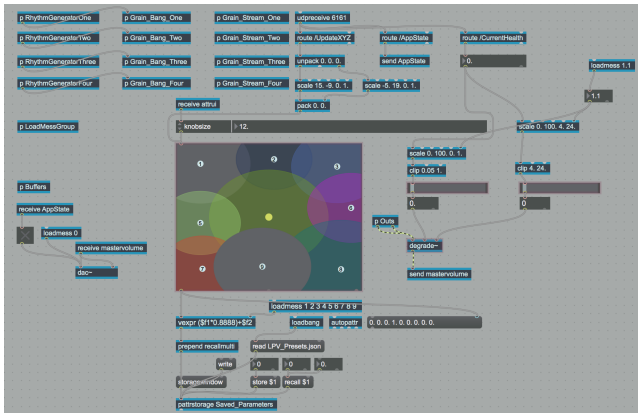

Figure 3: Patch made in MaxMSP for the project.

4. ACKNOWLEDGMENTS

This research was supported/partially supported by the Center for Computation and Technology (CCT) at Louisiana State University. I thank my advisor Jesse Allison who provided insight and expertise that greatly assisted the research.

5. REFERENCES

[1] M. Peltola, T. Lokki, and L. Savioja. Augmented reality audio for location-based games. In *Audio Engineering Society Conference: 35th International Conference: Audio for Games*, Feb 2009.

[2] N. Prior. *Popular Music*, 29(2):319–320, 2010.

[3] F. Rumsey. Audio for games. *J. Audio Eng. Soc*, 59(5):341–345, 2011.

[4] C. Schissler and D. Manocha. Gsound: Interactive sound propagation for games. In *Audio Engineering Society Conference: 41st International Conference: Audio for Games*, Feb 2011.

[5] M. WRIGHT. Open sound control: an enabling technology for musical networking. *Organized Sound*, 10(3):193âĂŞ200, 2005.

[5] [2] [3] [1] [4]