Sound generator for video games

Jovan Prodanov
Faculty of Computer and Information Science
University of Ljubljana
Slovenia, Ljbuljana 1000
Email: jp6957@student.uni-lj.si

***Abstract*—One of the main problems of game development industry is getting and making assets. Nowadays, the acquirement of sound is solved by sound generators. The goal of this paper is to make a basic sound generator for video games, which the user can change parameters to generate the sound effect he needs. In the meantime, hopefully the user can get more more knowledge about digital sound.**

I. Introduction

Game developers have always struggled with the acquirement of assets in any form. They would even go to lengths to learn a new software so they can create the assets which they need. The said assets can be in any form: graphics (characters, environments, logos), background music, special effects and sounds effects. In this paper we will focus on the sound effects.

Rather than to acquire sound producers, or even learn the means to produce sounds - game developers found another solution to the problem of sound effects. That is to generate sound - It works so well because in video games, because there is a limited number of sounds. For example: walking, running, shooting, jumping, explosion, etc, these sounds can be generated with just some parameters in the input and results can be obtained to the game developer instantly.

II. Related Work

This idea, of course, isn't a new one, there exists a number of working sound generators for video games: sfxr, jsfxr, chiptone, bfxr...(see figure 1, 2, 3).

So, from my research the tool sfxr [1] is the original tool created on the 10th competition on Ludum Dare, which is a contest of best software made in 48 hours. Then, the elaborations bfxr [2] and jsfxr [3] of sfxr were created. They are just an elaboration of the original sfxr sound generator made with JavaScript. On the other hand ChipTone [4] is an interesting take on the sound generation, it is made to work on HTML.

These solutions are amazing and working perfectly, but my goal would be to create something that the user can get more feedback.

III. Methods

My choice of technology is the Unity Editor and the Unity Engine [5]. The core of it is the `AudioSource` component which I can use to generate, save and play any given sound. And also for 2D visualization, I can use the `Canvas` component.

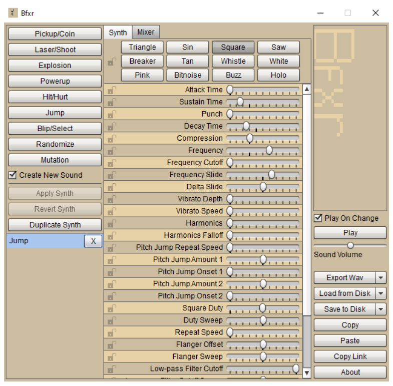
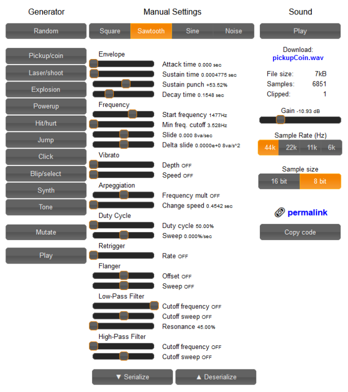


Fig. 1. Software bfxr [2]



Envelope
Attack time 0.000 sec
Sustain time 0.0004775 sec
Sustain punch +53.52%
Decay time 0.1548 sec
Frequency
Start frequency 1477Hz
Min freq. cutoff 3.528Hz
Slide 0.000 8va/sec
Delta slide 0.0000e+0 8va/s^2
Vibrato
Depth OFF
Speed OFF
Arpeggiation
Frequency mult OFF
Change speed 0.4542 sec
Duty Cycle
Duty cycle 50.00%
Sweep 0.000%/sec
Retrigger
Rate OFF
Flanger
Offset OFF
Sweep OFF
Low-Pass Filter
Cutoff frequency OFF
Cutoff sweep OFF
Resonance 45.00%
High-Pass Filter
Cutoff frequency OFF
Cutoff sweep OFF

Download:
pickupCoin.wav

File size:	7kB
Samples:	6851
Clipped:	1

Gain -10.93 dB
Sample Rate (Hz): [✓ 44k] 22k 11k 6k
Sample size: 16 bit [✓ 8 bit]

Fig. 2. Software sfxr [3]

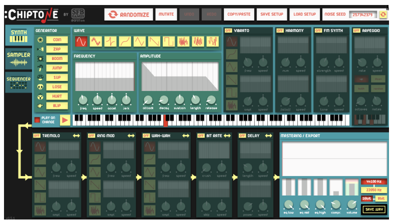


Fig. 3. Software ChipTone [4]

The formula for generating sound will be nearly similar in every implementation. There will always be type of wave, frequency, amplitude, vibrato, arpeggio, harmony, etc. With these parameters, sound effect for video games can be generated.

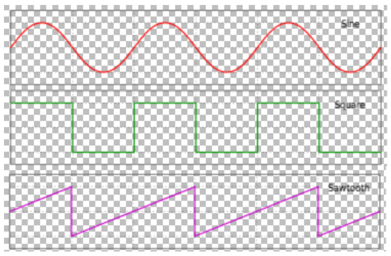


Fig. 4. Sound waves

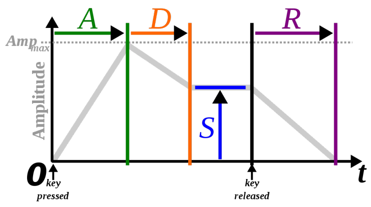


Fig. 5. Envelope: attack, decay, sustain, release

Example: jumping, shooting, explosions...

With such a simple and powerful base with the Unity Engine, I can create a sound generator and change it to my liking however I want. Also the software Audacity [6] was used to double check generated waves.

A. Implementation

First off, I got to know what are the requirements for the generation of video game sounds.

- Type of wave - to generate all the needed sound for video games, only need waves are: Sine, Square, Sawtooth and Noise, see figure 4.
- Volume - relative strength of the sound waves.
- Frequencies - start frequency and cut off frequency.
- Vibrato - the vibration and the pulsating of the sound, it has depth and frequency.
- Envelope - describes how the sound changes over time (see figure 5). The most common way of envelope generation is with four stages: attack, decay, sustain and release. Attack time is the time from the start taken for the sound to have its highest point, or to peak. Decay time is the time that the sound runs down from the attack level to the designated sustain level. Sustain time is during the main sequence of the sound's duration. And, for last Release time is the time taken from the sustain level to zero.
- Arpeggio - the Italian word is *a type of broken chord*, in my example it gives the user the option play the chords in ascending or descending order.
- Noise frequency - input the frequency of the noise wave, this is the only parameter that works on the noise wave.

With this knowledge, the script/component for the sound generation can be programmed in one component/script.

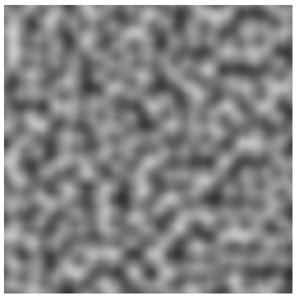


Fig. 6. Example of Perlin noise

B. Architecture

Good architecture in Unity has influence on many things. To start off, it is much more performant, as we can divide each functionality of our program to separate things and can be easily improve or worked upon. Another thing is upgradability down the line, in some time if someone wants to work on a part of the code, there isn't a need to understand the whole project as a whole. So, in my project I have the following architecture and functionality of the parts.

1) Generator controller: The core of the program, it has the `AudioSource` component linked and the generation to the sound clip happens here. It has methods for generation, randomization and returning of float array data ready for display and visualization.

The main parts this controller is for every sample calculating the **envelope** and **phase**.

```
var t = i / (float)SampleRate;
float envelope;
if (t < attackTime)
    envelope = t / attackTime;
else if (t < attackTime + decayTime)
    envelope = 1 - (t - attackTime) / decayTime *
else
    envelope = sustainLevel * (1 - (t - attackTime
```

Next up is calculating the phase with:

$$phase = (frequency + arpeggio + vibrato) * time * 2 * \pi$$

With it we can calculate with the formulas at any time in respect of what type of wave it is.

- Square - $sing(\sin(phase)) * envelope * volume$ where sign() is the sign of the value.
- Sawtooth - $2 * (phase\%(2 * \pi))/(2 * \pi) - 1) * envelope * volume$
- Sine - $\sin(phase) * envelope * volume$
- Noise - noise is calculated with the Unity's PerlinNoise (see figure 6) function $(PerlinNoise(t * noiseFrequency) * 2 - 1) * envelope * volume$

2) Sound generator presenter: The presenter handles the initialization and updating the UI components of the `Canvas` (sliders and buttons) as we can see in figure 7.

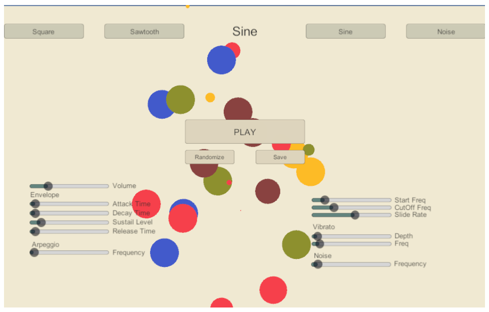


Fig. 7. My sound generator application

3) Sound visualization controller: In order to make my application more unique, I made some visualizations. I created a `ParticleSystem` of different colored balls that changes size over their lifetime, the move up and down based on the frequency of the sound generated. Also if the amplitude crosses a threshold the particle system emits more particles.

When the sound is generated the sound visualization controller generates an image with its sound wave in time domain, this is the part I used to compare the generated image to the one in Audacity [6]. Also it should be noted that this controller is only liked to the audio source, particle system and the image it generates to.

Note: the visualizations can be only seen in the executable build of the sound generator, because of Unity's optimizations and WebGL's limitations.

IV. Results

The results I had were acceptable, I had created a working and useful sound generator [7] that can generate most of the sounds of simple (retro) video game. The tool I created can also be helpful at learning more about digital sounds, as it has visualizations and room for users to play with.

Some pros are that with the technologies used at developing the tool, it can be built for more platforms, it has optimizations for what kind of hardware it is used and great upgradability in the future.

V. Conclusion

To conclude, there were some uncomplete stuff like browser downloading files through Unity was not possible, but it works in the executable downloadable build and it saves them in the folder of the tool. And of course, the WebGL limitations.

Although my take on the subject, the above aforementioned examples are more extensive than my example, but I am certain because of the technologies used and the architecture, with more effort, my program can progress like that and even better.

References

[1] D. Petter, "Sfxr," 2007. [Online]. Available: https://www.drpetter.se/project$_sfxr.html$

[2] S. Lavelle, "Bfxr," 2021. [Online]. Available: https://www.bfxr.net/

[3] DrPetter, "Jsfxr," 2011. [Online]. Available: https://sfxr.me/

[4] T. Vian, "Chiptone," 2021. [Online]. Available: https://sfbgames.itch.io/chiptone

[5] U. Technologies, "Unity," 2017. [Online]. Available: https://unity.com/

[6] A. Team, "Audacity," 2021. [Online]. Available: https://www.audacityteam.org/

[7] J. Prodanov, "Sound generator for video games," 2023. [Online]. Available: https://wrathchild14.itch.io/sound-generator