Tvorba zvuku pomocí syntézy pro hru obdobnou stylophonu (vlastní projekt ISS)

Petr Růžanský

7. ledna 2022

1 Úvod

V tomto projektu chci experimentovat s generováním zvuku, který potřebuji pro moji hru.

Hra (aplikace) je inspirována hudebním nástrojem stylophone. Obsahuje 2d dotykovou plochu. Pozice doteku na ose X určuje tón (frekvenci), osa Y určuje modifikaci tónu. Cílem tohoto projektu bylo naimplementovat takovouto aplikaci.

2 Tvorba aplikace

2.1 Engine Unity

Pro tvorbu aplikace jsem se rozhodl použít engine Unity. Ten pro skriptování používá jazyk C#. Ovšem pro potřeby projektu z ISS a pro experimentování se signály jsem se rozhodl použít jazyk Python. Většinu pokusů provádím v přiloženém jupyter notebooku.

2.2 Hrací plocha

Důležitou částí projektu je hrací plocha, která bude zobrazovat noty, překládat souřadnice dotyku na frekvence a stupeň modifikace a volat jednotlivé oscilátory s danými parametry (co dotyk (prst), to jeden oscilátor).

Pomocí jednoduché matematiky a za pomocí funkcí obsažených v Unity, přeložím souřadnici dotyku X, na pozici relativní k hrací ploše v intervalu 0 až 1, kde 0 znamená bod úplně vlevo a 1 úplně vpravo.

Definuji si slovník, kde klíčem je název noty a hodnotou je její frekvence.

Určím minimální a maximální frekvenci a spočítám jejich rozsah:

```
float minFreq = notes["C4"];
float maxFreq = notes["C5"];
float freqRange = maxFreq - minFreq;
```

Pro každou notu spočítám její relativní pozici na hrací ploše:

```
foreach (var item in notes){
    float loc_x_position = (item.Value - minFreq) / freqRange;
...
```

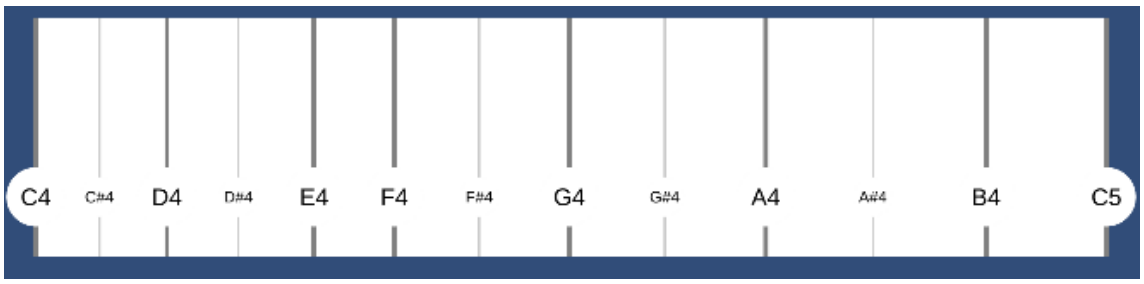
Tu následně vykreslím.

Jak z výsledku na obrázku 1 můžeme vidět, frekvence jednotlivých not neroste lineárně, nýbrž exponenciálně. Musím ji tedy logaritmovat. Jako první logaritmuji limitní frekvence, abych zachoval poměr při výpočtu.

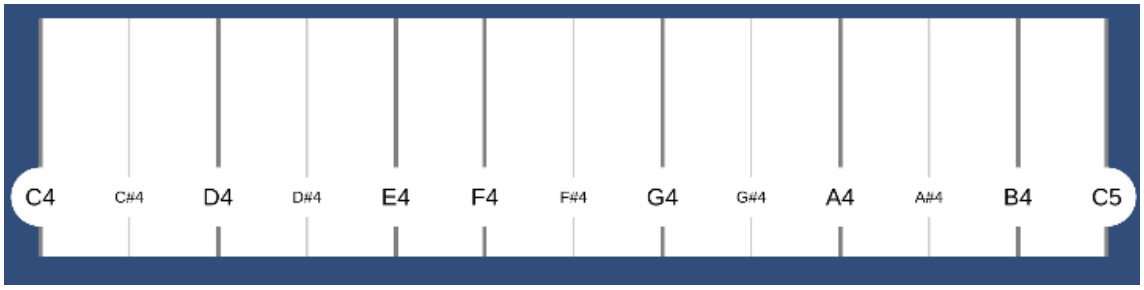
```
minFreq = Mathf.Log10(minFreq);
maxFreq = Mathf.Log10(maxFreq);
freqRange = maxFreq - minFreq;
```

A následně onu frekvenci tónu.

```
foreach (var item in notes){
    float loc_x_position = (Mathf.Log10(item.Value) - minFreq)/freqRange;
```



Obrázek 1: Hrací plocha před logaritmováním.



Obrázek 2: Hrací plocha po logaritmováním.

Jelikož se jedná o poměr v rozsahu 0 až 1, základ logaritmu na výsledek nemá vliv. Já jsem použil logaritmus o základu 10.

Výsledek můžeme sledovat na obrázku 2

Podobně můžeme realizovat převod pozice prstu v rámci hrací plochy na frekvenci. Z předchozího vzorce na získání *loc_x_position*, která určuje relativní pozici na ploše v rozsahu 0 až 1, odvodíme frekvenci (*item.Value*).

```
float freq = Mathf.Pow(10, x_position_ratio * freqRange + minFreq);
```

2.3 Oscilátory

Hrací plocha nám generuje frekvence podle pozice prstu. Nyní musíme vygenerovat nějaký vlnový průběh (waweform), který následně přehrajeme. Generátorům základního waveformu se říká oscilátor. Mezi základní tvary patří sine wave (sinus), square wave (čtverec), saw wave (pila) a triangle wave (trojúhelník). Nejlepší je začít vlnou sinusového tvaru. Ta se dá následně modifikovat na ostatní tvary.

V enginu Unity si vytvořím herní objekt `Oscillator` a k němu přiřadím komponentu `Audio Source`, která umožňuje přehrávat zvuky, ať už ze souboru (`.mp3, .wav...`) tak i hudbu generovanou za běhu, což bude můj případ. Vytvořím si C# skript jménem `Oscillator.cs`, který objektu přiřadím. Skript bude volán hrací plochou, která nám bude předávat frekvenci a stupeň modifikace a náš skript danou frekvenci vygeneruje a nechá přehrát.

V skriptu přidám mimo jiné funkci `void OnAudioFilterRead(float[] data, int channels)`. Funkce je volána jednou za čas komponentou `Audio Source`, kterou jsem přidal v minulých krocích. Funkce mi předává pole `data`, která přehraje. Já je následně můžu modifikovat a tím za běhu určovat, jaký zvuk bude hrát.

2.3.1 Sinusová vlna

Začnu jednoduchým sinem.

```
private void OnAudioFilterRead(float[] data, int channels){

    increment = frequency * 2 * Mathf.PI / f_sampling;

    for (int i = 0; i < data.Length; i += channels)
    {
        phase += increment;
        data[i] = volume * Mathf.Sin((float)phase);
```

```
                // Pokud ma Audio Player vice kanalu, rozkopiruj zvuk na vsechny z nich
                for (int ch = 1; ch < channels; ch++)
                {
                    data[i + ch] = data[i];
                }

                // Zabraneni preteceni
                if (phase > (Mathf.PI * 2))
                {
                    phase -= Mathf.PI * 2;
                }
        }
    }
```

Zdroj většiny kódu: [2]

Na začátku si zvolím velikost inkrementu (skoku), kde `frequency` je frekvence, kterou oscilátoru nastaví hrací plocha. Proměnná `phase` uchovává aktuální stav a na konci, pokud je proměnná větší jak 2π, tak od ní 2π odčítám, abych zabránil přetečení.

2.3.2 Hranatá vlna

Jelikož zvuk sinusové vlny je málo výrazný, dočasně ji transformuji na vlnu hranatou, než najdu jiný způsob, jak zvuk ještě více zvýraznit. Toho dosáhnu jednoduchým způsobem: Poté, co pomocí sinu vygeneruji novou hodnotu, tak ji změním na maximum nebo minimum v závislosti na tom, zda je větší nebo menší rovna nule.

```
        for (int i = 0; i < data.Length; i += channels){
            phase += increment;
            data[i] = Mathf.Sin((float)phase);

            // prevod na ctverec
            if (data[i] >= 0)
            {
                data[i] = volume * 0.6f;
            }
            else
            {
                data[i] = -volume * 0.6f;
            }
```

Tím bych měl oscilátor hotový. Celkově do hry vložím 4 tyto oscilátory a hrací desku nastavím tak, aby uměla volat všechny tyto oscilátory, a já byl schopen hrát až čtyrmi prsty na jednou.

2.4 Vyzkoušení aplikace

Tím je základ aplikace hotový. Ukázku si můžete prohlédnout zde: https://youtu.be/hj1RVJNv7Og

2.5 Modifikace zvuku

Změnu frekvence už mám. Nyní bych ale potřeboval měnit výraznost tónu. Ten se bude měnit v závislosti pozici prstu na ose Y. Dole bych chtěl mít málo výrazný zvuk, čím by se šlo ale výše, tím by byl zvuk mnohem více výrazný. Nabízí se několik řešení. Téměř všechna z uvedených řešení, která se v této sekci vyskytují, jsem experimentoval a demonstroval v jupyter notebooku, který můžete najít zde: https://mybinder.org/v2/gh/Ruuza/sound_analysis_ISS/bd292518f866057a8ea45f027b6a0f572272aad4?urlpath=lab%2Ftree%2Fsound_analysis.ipynb

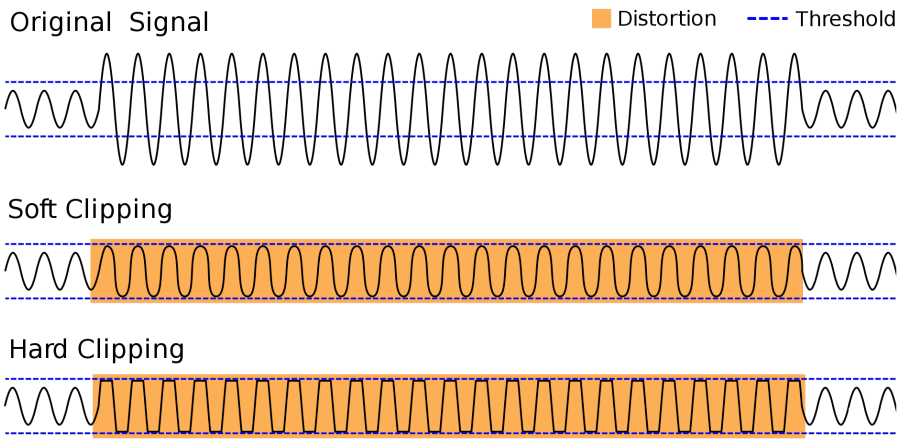
2.5.1 Přechod mezi sinusovou a hranatou vlnou

První z možností, která mě napadla, je přechod mezi sinusovou a čtvercovou vlnou. Jak ve zmíněné ukázce v jupyter notebooku výše můžeme slyšet, sinusová vlna je o dost méně výraznější, jak hranatá.

Přechod mezi vlnami je velmi jednoduchý a dá se například implementovat váženým průměrem.

2.5.2 Zkreslení sinusové vlny

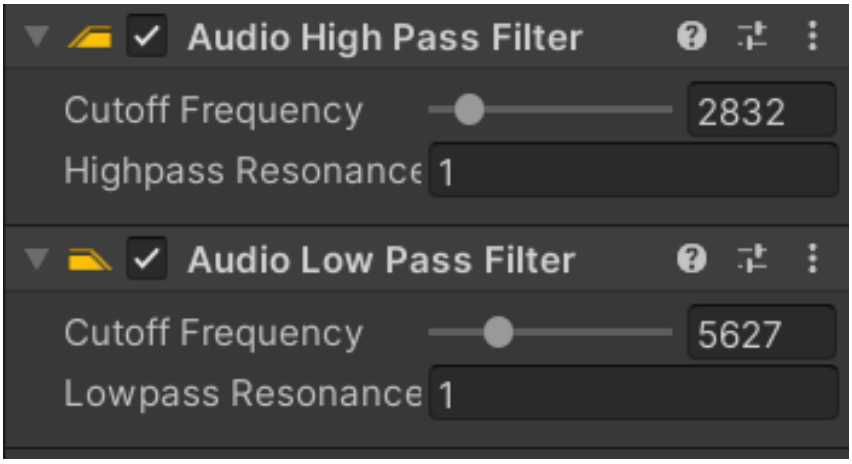
Přebuzením (zvolením vysoké amplitudy sinusové vlny) a následným omezením maxima vlny na specifickou hodnotu, by jsme mohli dosáhnout efektu, kterému se anglicky říká *clipping*. Výsledek bude velmi podobný, jako u přechodu mezi sinusovou a hranatou vlnou zmíněným výše. Máme sinusovou vlnu, kterou roztahujeme do výšky (zvyšujeme amplitudu) a následně ořežeme na nějaké maximum. Po ořezaní se vlna blíží podobě hranaté vlny.



Obrázek 3: Ukázka z [1]. Sinusová vlna je zkreslena na vlnu podobající se vlně hranaté, pomocí metody *hard clipping*.

2.5.3 Filtrování signálu

Další možností, kterou jsem řešil, bylo filtrování signálu pomocí horní, dolní nebo pásmové propusti. Engine Unity obsahuje komponenty `Audio Highpass Filter` (horní propusť) a `Audio Lowpass Filter` (dolní propusť) pro filtrování signálu, které práci ještě více ulehčily. Viz obrázek 4. Pokud použiji oba dva filtry zároveň, můžu i simulovat pásmovou propusť.



Obrázek 4: Filtry horní a dolní propusť v enginu unity.

Bohužel, filtry nebyly to, co jsem hledal. Jejich efekt se mi moc nezamlouval.

2.5.4 Součet více sinusových komponent o n-násobné frekvenci

Posledním a asi nejvíce, pro moje potřeby, vyhovujícím řešením je součet více sinů dohromady. Ale ne jen tak ledajakých. Aby si zvuk zachoval svůj tón, mělo by se převážně jednat o n-násobky původní frekvence. Původní frekvence by měla mít největší amplitudu. Více o této možnosti ve výše zmíněném jupyter notebooku.

2.6 Závěr

Výsledek projektu hodnotím kladně. Mnohé jsem se naučil. Pochopil jsem, jak vzniká zvuk a jak se dá se zvukem různě pracovat a generovat ho. Můj osobní cíl úkolu se mi povedlo splnit, a to jest najít rozumnou možnost, jak měnit zvuk tónu pro potřeby mé nadcházející bakalářské práce.

Reference

[1] File:clipping waveform.svg. URL: https://commons.wikimedia.org/wiki/File:Clipping_waveform.svg#/media/File:Clipping_waveform.svg.

[2] MCV Staff. Procedural audio with unity, Jul 2012. URL: https://www.mcvuk.com/development-news/procedural-audio-with-unity/.