Unity PSG Player

This is a library for synthesis PSG (Programmable Sound Generator) sound sources such as retro game consoles on Unity. The performance data is MML (Music Macro Language) text, which describes musical notation in text, allowing easy creation of music data. The library is designed to produce expressions similar to those of NES sound generators (except DPCM).

- The sound generator can produce four types of square waves, a triangle wave, and two types of noise. The triangle wave is a 4-bit waveform.
- Performance expressions include sweep, LFO (vibrato), and volume envelope.

> I'm not aiming for a perfect recreation of the Famicom sound source. (It's too much trouble.)
> My goal is solely to produce sounds using only Unity, without preparing separate sound clips.

The PSG Player produces monophonic (single-note) sound. To play NES 4-note polyphonic sounds (excluding DPCM), you can achieve this by preparing multiple PSG Players and playing them simultaneously.
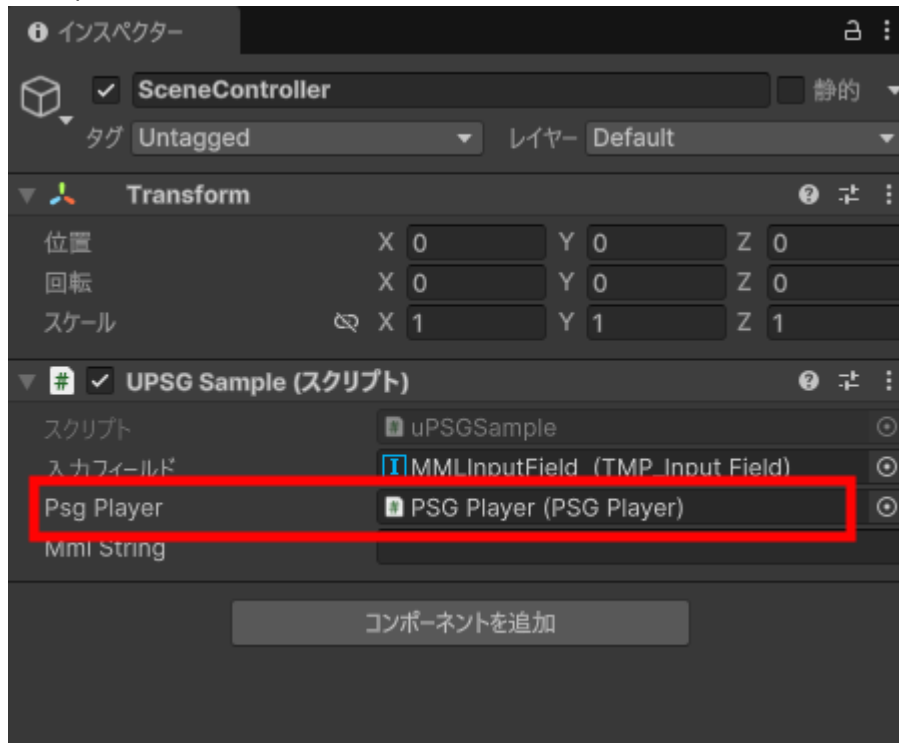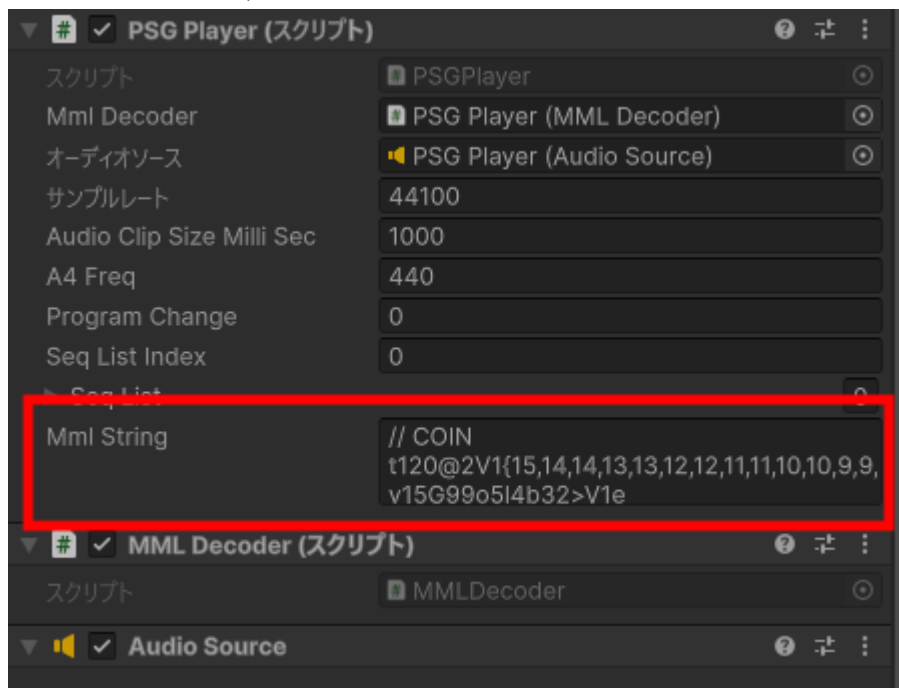
Quick Guide

Basic Usage

1. Place the PSG Player prefab in the Hierarchy.
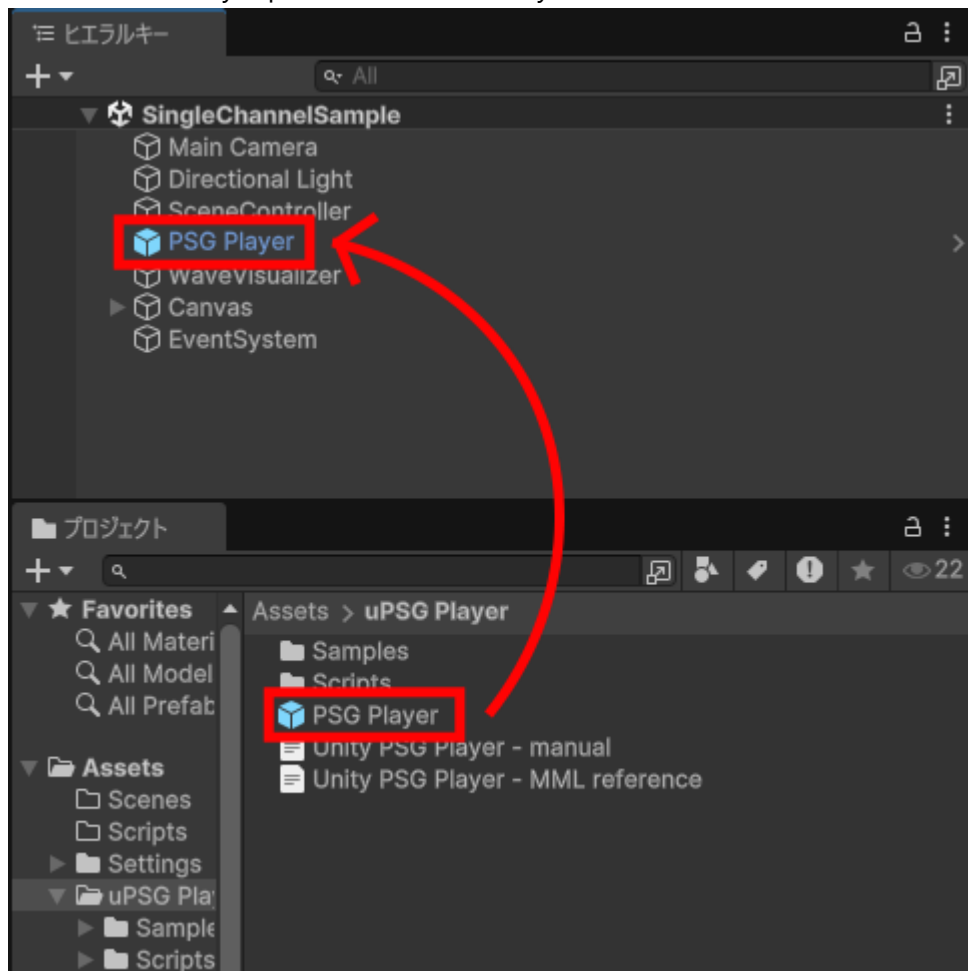
2. Prepare a PSGPlayer class variable in the script you are operating, and attach the PSG Player object you have placed.
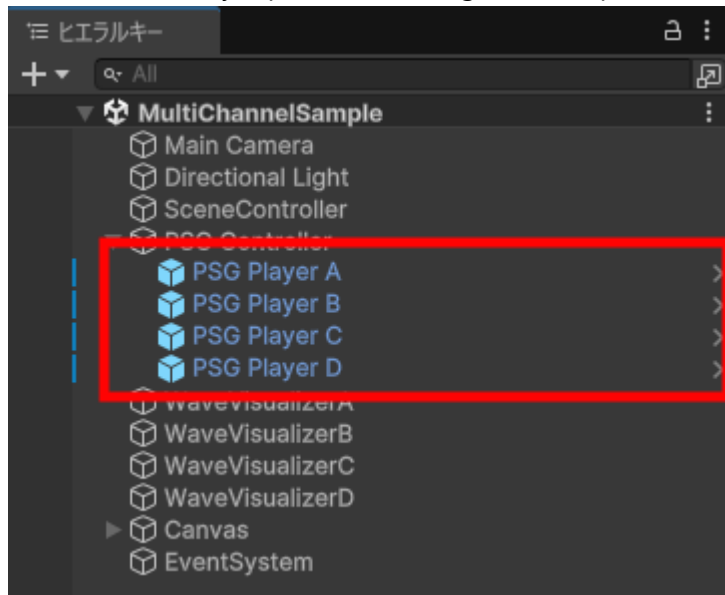

3. The MML written in the mmlString variable of the PSGPlayer will be played by Play() function.
   For details on MML, refer to the MML Reference.

Multi-channel Usage

1. Place the PSG Player prefab according to the required number of channels.

2. Attach the MMLSplitter script to the appropriate game object.

3. Assign the placed PSG Player to the psgPlayers of the MMLSplitter from the Inspector.
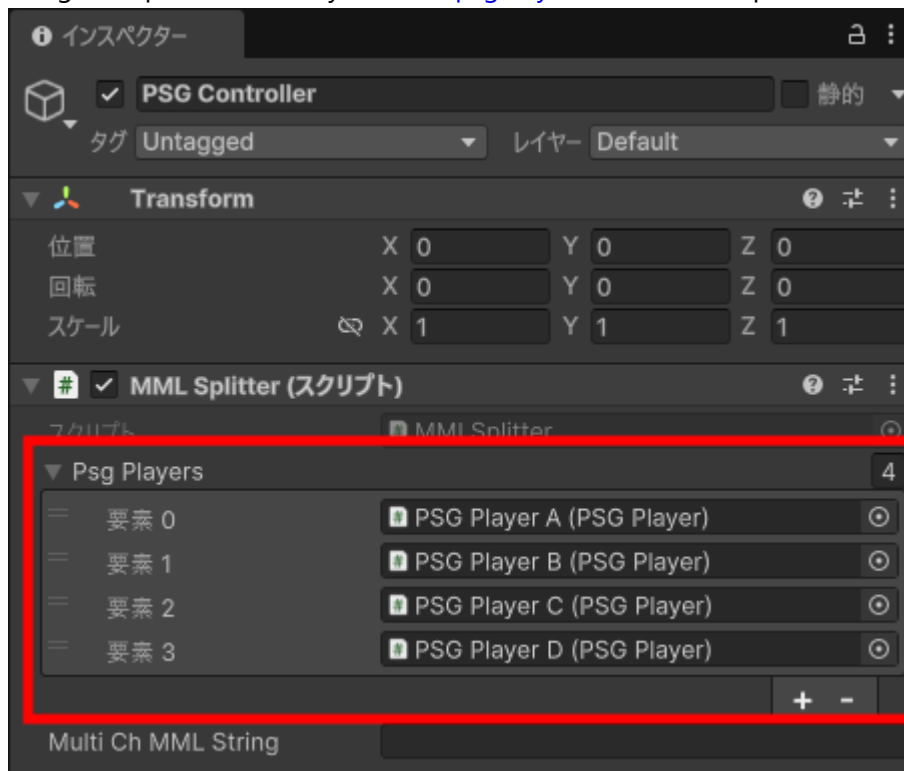
4. Place the MML into the multiChMMLString variable of MMLSplitter, distribute the MML to each channel using the SplitMML() function, and play it using the PlayAllChannels() function.
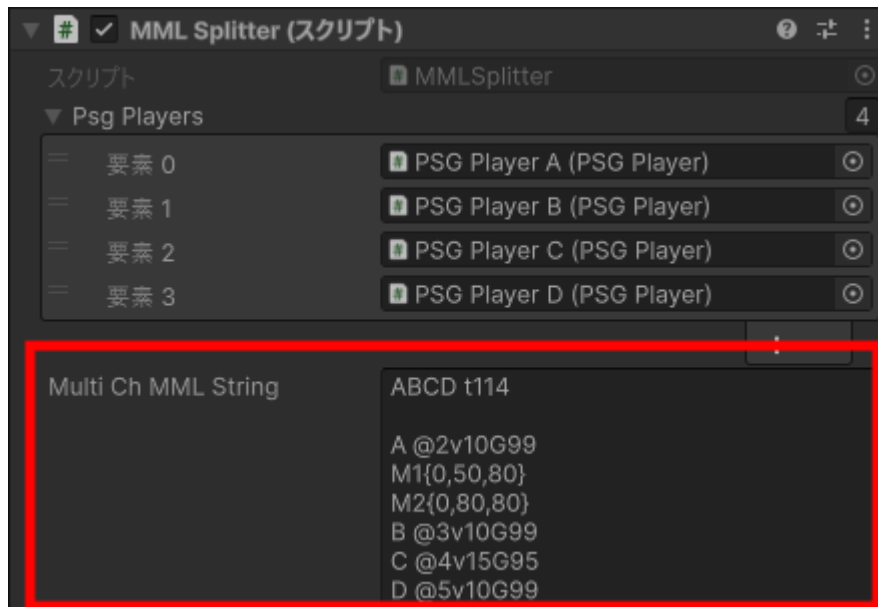
Composition

The Unity PSG Player consists of:

- The "**PSG Player**" that synthesizes sound according to sequence data
- The "**MML Decoder**" that converts MML to sequence data
- The "**MML Splitter**" that splits MML
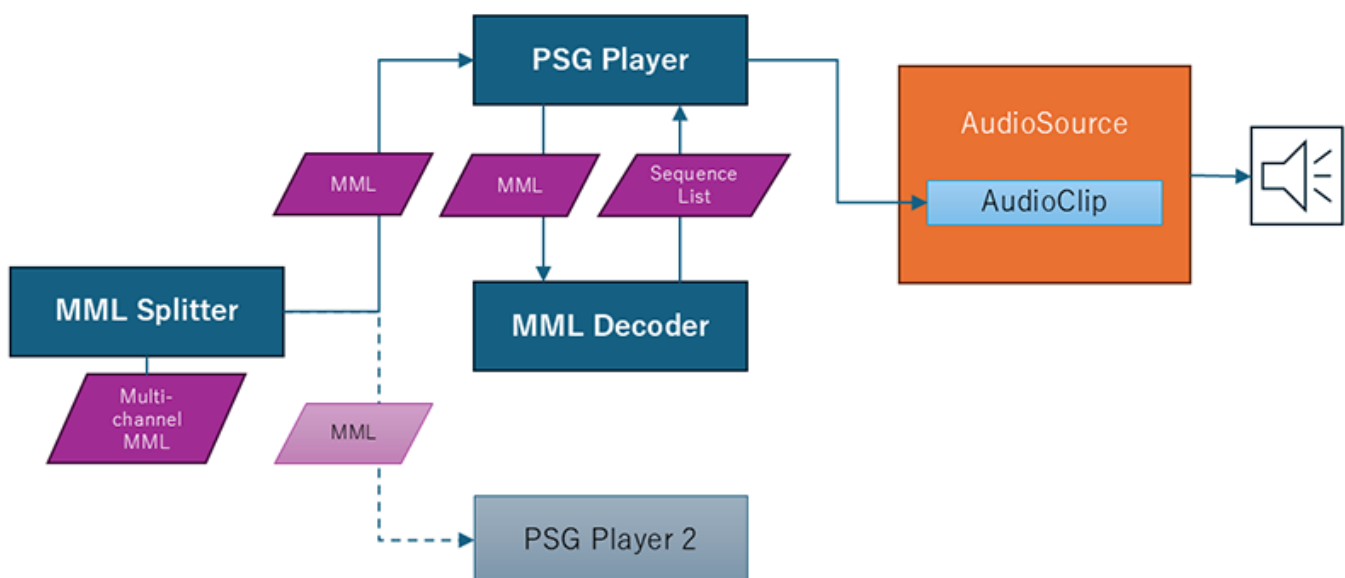
MML Splitter is not required for single sound use.
An AudioSource component is also required to play the sound generated by PSG Player.

Summary

PSG Player synthesizes monophonic waveforms: square waves (pulse waves), triangle waves, and noise. The generated sound is attached as an AudioClip to an AudioSource resource and streamed. During streaming, the AudioClip sequentially requests the necessary buffer data, and PSG Player calculates the waveform data for each request.

PSG Player sequentially processes sequence data from a List array to calculate pitch and volume. Conversion from MML to sequence data is performed by the MML Decoder.

Additionally, use the MML Splitter to distribute multi-channel MML. It processes each line of the MML and determines the destination channel based on the characters at the beginning of the line. For details, refer to the MML Reference. Besides distributing MML, the MML Splitter can also centralize control over each PSG Player.

When playing sound effects using PSG, retro game consoles would interrupt the playback of one BGM channel to play sound effects in real time. However, the PSG Player buffers pre-loaded sounds, making it impossible to play sound effects with precise timing.
To play sound effects in PSG Player similarly to retro game consoles, prepare a separate PSG Player instance dedicated solely to sound effects and mute one of the BGM channels.

When using PSG Player in multi-channel mode, using the AudioMixer makes it easier to adjust volume levels and other settings.

About Sample rate

PSG Player allows you to set the sample rate for generated AudioClips. The sample rate is the number of samples per second. Since processing occurs for each sample, increasing the sample rate also increases CPU load.

Additionally, the sample rate affects the upper limit of the frequency of the generated sound.
The number of samples required per wavelength varies depending on the tone, as follows:

Tone	Samples
Square wave	2 samples
25%(75%) Pulse wave	4 samples
12.5% Pulse wave	8 samples
Triangle wave	32 samples

The frequency of sound that can be produced without issues is determined by dividing the sample rate by the number of samples mentioned above, so the highest pitch achievable, especially for a triangle wave, will be lower.

- This represents the upper limit for achieving the expected pitch; higher notes will cause the tone to deteriorate.
- Due to unique noise processing, lower sample rates will cause volume to decrease at high frequencies.

PSG Player script refference

List of Variables & Public Functions

- Variables
  - mmlDecoder
  - audioSource

Variables

mmlDecoder

```
[SerializeField] private MMLDecoder mmlDecoder;
```

Register the MML Decoder component that converts MML into sequence data.

audioSource

```
[SerializeField] private AudioSource audioSource;
```

Register the AudioSource that will receive the generated AudioClip.

sampleRate

```
public int sampleRate = 32000;
```

Sets the sample rate for the AudioClip.
The default is 32000Hz (32kHz).

audioClipSizeMilliSec

```
public int audioClipSizeMilliSec = 1000;
```

Sets the length of the AudioClip.
The default is 1000 milliseconds (1 second).

---

a4Freq

```
public float a4Freq = 440f;
```

Set the frequency of the A note in the fourth octave (o4a), which serves as the standard for the musical scale.
The default is 440Hz.
This variable can be changed using MML commands.

---

tickPerNote

```
public int tickPerNote = 960;
```

Set the resolution to one beat (quarter note).
Note duration is converted to the number of ticks based on this resolution, and the actual note length (in seconds) is calculated from the tempo and this resolution.

(Example: An eighth note corresponds to 480 ticks. At a tempo of 120, the note duration is
60[sec] / 120[notePerMin] * 480[tick] / 960[tickPerNote] = 0.25[sec]
which equals 0.25 seconds.)

---

programChange

```
public int programChange;
```

The number of the tone to be generated.

Number	Wave form
0	Pulse wave（12.5%）
1	Pulse wave（25%）
2	Square wave（50%）

Number	Wave form
3	Pulse wave（75%）
4	Triangle wave
5	Noise
6	Short-cycle noise

This variable can be changed using MML commands.

---

seqListIndex

```
[SerializeField] private int seqListIndex = 0;
```

Current position in sequence data processing.
Mainly displayed for debugging purposes.

---

seqList

```
[SerializeField] private List<SeqEvent> seqList = new();
```

A List array of sequence data.
Mainly displayed for debugging purposes.

---

mmlString

```
[Multiline] public string mmlString = "";
```

This is the MML string to be played.
Pass this variable to the MML Decoder to convert it into sequence data.

Public Functions

Play()

```
public void Play();
```

- Parameter : None

Converts the MML string in the mmlString variable into sequence data and begins playback.

Play(string _mmlString)

```
public void Play(string _mmlString);
```

- Parameter : **_mmlString** MML string

Pass the parameter arguments to the mmlString variable, convert them to sequence data, and start playback.

DecodeMML()

```
public bool DecodeMML();
```

- Parameter : None
- Return value : True if decoding succeeds

Pass the MML string from the mmlString variable to the MML Decoder to convert it into sequence data.

PlayDecoded()

```
public void PlayDecoded();
```

- Parameter : None

Plays back decoded sequence data.
Since no conversion processing is performed, reduced CPU load is expected.

Stop()

```
public void Stop();
```

- Parameter : None

Stop the currently playing audio.

IsPlaying()

```
public bool IsPlaying();
```

- Parameter : None
- Return value : True if playing

Returns the playback status of AudioSource.

Mute

```
public void Mute(bool isOn);
```

- Parameter: **isOn** Set True to Mute on

When muted, the AudioSource is silenced and the generated sample is set to 0 (silence).
When unmuted with `false`, the AudioSource is immediately unmuted, but the sample remains silent until the next note event occurs.
However, if unmuted before the buffered sample plays out, the already generated sample will be played.

MML Splitter script refference

[MML Splitter] List of Variables & Public Functions

- Variables
  - psgPlayers
  - multiChMMLString
- Public Functions
  - SplitMML()
  - SplitMML(string _multiChMMLString)
  - SetAllChannelsSampleRate(int _rate)
  - SetAllChannelClipSize(int _msec)
  - PlayAllChannels()
  - PlayAllChannelsDecoded()
  - StopAllChannels()
  - IsAnyChannelPlaying()
  - MuteChannel(int channel, bool isMute)

[MML Splitter] Variables

---

psgPlayers

```
[SerializeField] private PSGPlayer[] psgPlayers;
```

Register a PSG Player component for each channel to send the MML in segments.

multiChMMLString

```
public string multiChMMLString;
```

Register the original MML string to be sent in segments.

[MML Splitter] Public Functions

SplitMML()

```
public void SplitMML();
```

- Parameter : None

Splits the MML string from multiChMMLString and sends it to the PSG Player registered in psgPlayers.
For channel assignment details, refer to "MML Reference".

SplitMML(string _multiChMMLString)

```
public void SplitMML(string _multiChMMLString);
```

- Parameter : **_multiChMMLString** Multi-channel MML string

Pass the parameter arguments to the multiChMMLString variable to send the MML to PSG Player in segments.

SetAllChannelsSampleRate(int _rate)

```
public void SetAllChannelsSampleRate(int _rate);
```

- Parameter : **_rate** Sample rate

Set the sample rate for all PSG players (in Hz).

SetAllChannelClipSize(int _msec)

```
public void SetAllChannelClipSize(int _msec);
```

- Parameter : **_msec** AudioClip length

Sets the AudioClip length for all PSG Players (in milliseconds).

PlayAllChannels()

```
public void PlayAllChannels();
```

- Parameter : None

All PSG Players decode MML and play simultaneously.

PlayAllChannelsDecoded()

```
public void PlayAllChannelsDecoded();
```

- Parameter : None

Play back the decoded sequence data simultaneously on all PSG Players.

StopAllChannels()

```
public void StopAllChannels();
```

- Parameter : None

Stop playing all PSG players.

IsAnyChannelPlaying()

```
public bool IsAnyChannelPlaying();
```

- Parameter : None
- Return value : True if any PSG Player is currently playing

Returns true if any of the AudioSources for each PSG Player is currently playing.

MuteChannel(int channel, bool isMute)

```
public void MuteChannel(int channel, bool isMute);
```

- Parameter: **channel** Target channel
             isMute True for mute on

Mutes the specified channel.