Game Audio

INSIDE and 140

Sonic College 2017

Jakob Schmid

Me

Jakob Schmid

INSIDE, Playdead, audio programmer

140, Carlsen Games, music and sound design

Today’s Plan

- Introduction
- 140 audio
- 140 Q&A
- INSIDE audio
- INSIDE Wwise project demo
- INSIDE Q&A?
- ~ lunch ~
- INSIDE Q&A?
- Wwise and Unity workshop

PDF with all slides available online!

140

Jeppe Carlsen (design, programming)

Niels Fyrst, Andreas Peitersen (visual design)

Jakob Schmid (audio)

Developed as hobby project over 3 years

PSN

STEAM

Wii U

Humble Bundle

XBOX ONE

gog.com

140

IGF award 2013

Excellence in Audio

- Honorable mention: Technical Excellence

Spilprisen 2014

Sound of the Year

Nordic Game Award 2014

Artistic Achievement

140 Technology

Standard Unity 3

Audio in 140

Jakob Schmid

Audio in 140

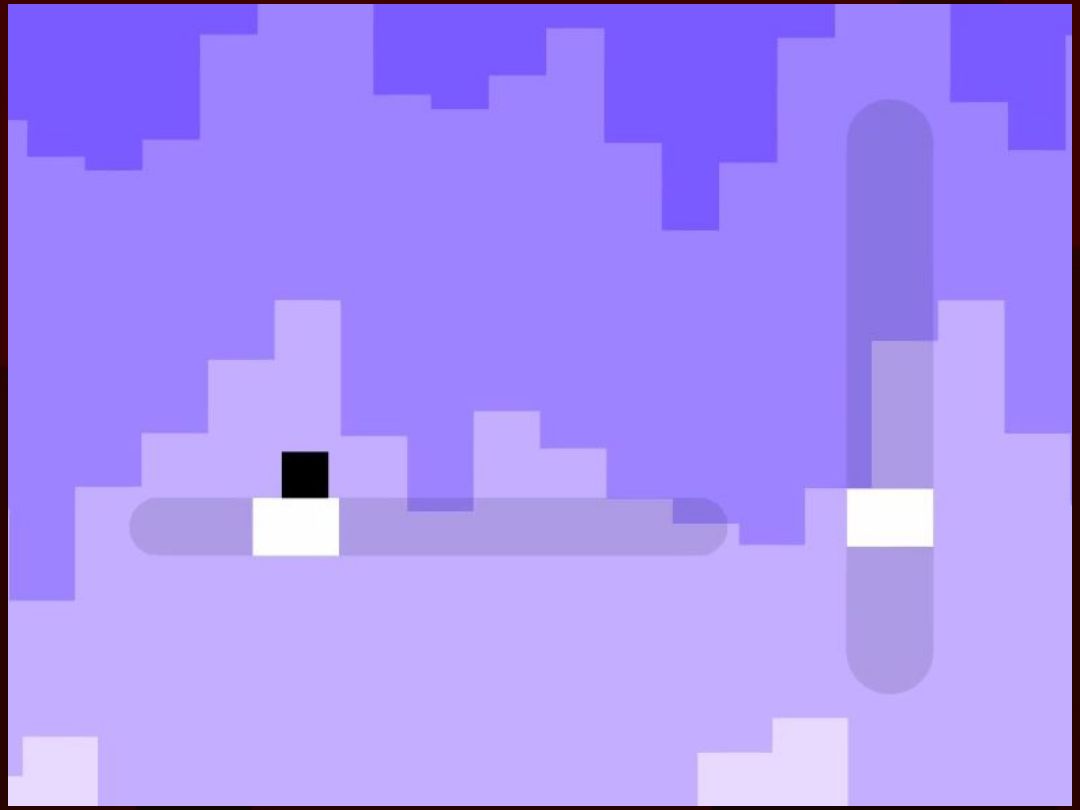
Overview

- Control game from music
- The music timing problem
- Music timing in 140
- Fun audio tricks

140 demo


Control Game from Music

Moving a Platform



Moving a Platform

1. 
wait for 16th note #1

2. 
start moving

3. 
wait for 16th note #8

4. 
start moving

Basic Approach

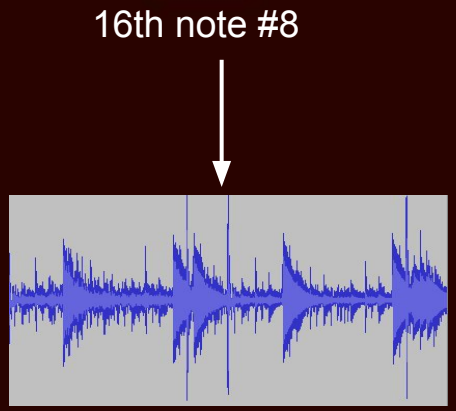
- Play music loop
- Use audio time from loop to control game elements (instead of game time)

```
AudioSource.time          (seconds)
AudioSource.timeSamples   (samples)
```

Music Events

‘Waiting for 16th note #8’ means:

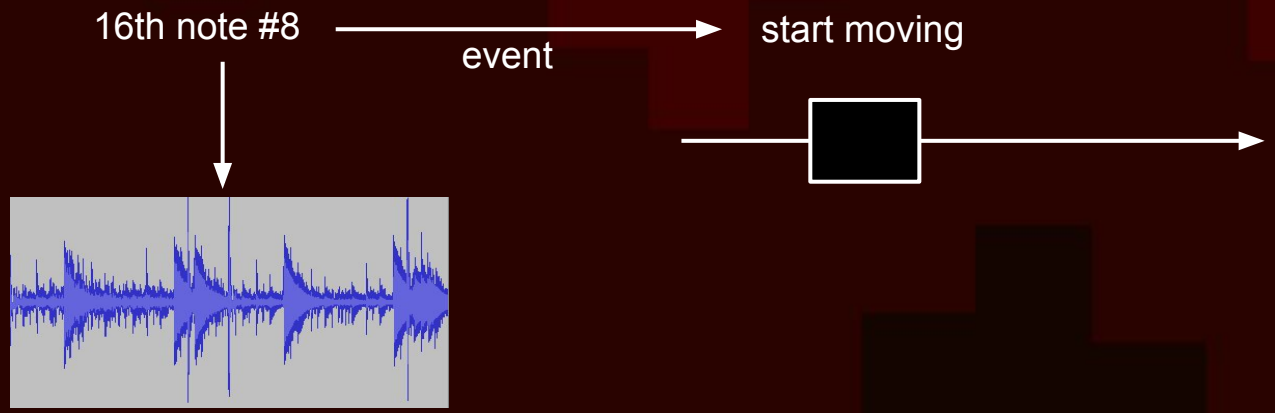
- Get audio time from playing loop
- When next musical beat reached, raise event
- Game elements listen for events and trigger animation on beats



Music Events

‘Waiting for 16th note #8’ means:

- Get audio time from playing loop
- When next musical beat reached, raise event
- Game elements listen for events and trigger animation on beats



Tempo

How long is a 140 BPM 16th note in seconds?

Tempo

How long is a 140 BPM 16th note in seconds?

```
140 beat/m
```

Tempo

How long is a 140 BPM 16th note in seconds?

```
140 beat/m * 4 note/beat = 560 note/m
```

Tempo

How long is a 140 BPM 16th note in seconds?

```
140 beat/m * 4 note/beat = 560 note/m
                         = 560/60 note/s
```

Tempo

How long is a 140 BPM 16th note in seconds?

```
140 beat/m * 4 note/beat = 560 note/m
                         = 560/60 note/s
```

This means that we have:

```
                           60/560 s/note
```

Tempo

How long is a 140 BPM 16th note in seconds?

```
140 beat/m * 4 note/beat = 560 note/m
                         = 560/60 note/s
```

This means that we have:

```
                           60/560 s/note
                        ~= 0.10714 s/note
```

Tempo in Samples

We can use samples instead of seconds for timing.

How long is a 140 BPM 16th note in samples?

```
  60/560 s/note
* AudioSettings.outputSampleRate smp/s
```

If sample rate is 48000 smp/s:

```
5143 smp/note
```

Moving a Platform

1. 
wait for 16th note #1

2. 
start moving

3. 
wait for 16th note #8

4. 
start moving

Moving a Platform

1. 

wait for loop time: 0 s

2. 

start moving

3. 

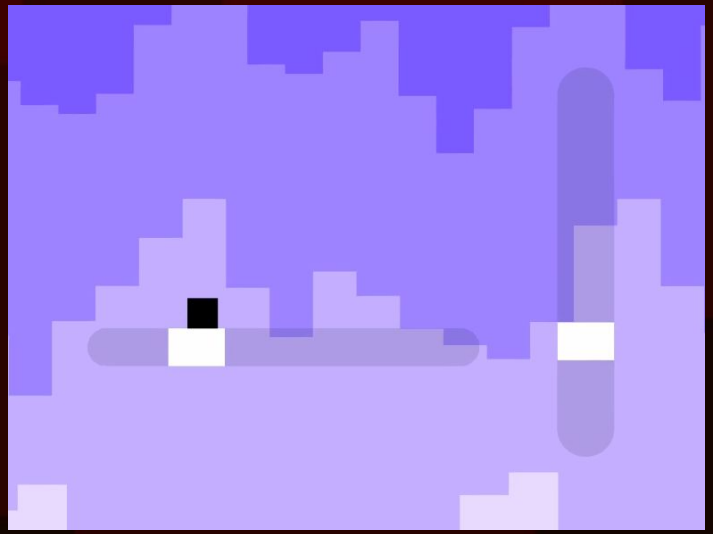
wait for loop time: 0.857 s
or timeSamples: 41143 smp

4. 

start moving

Summary

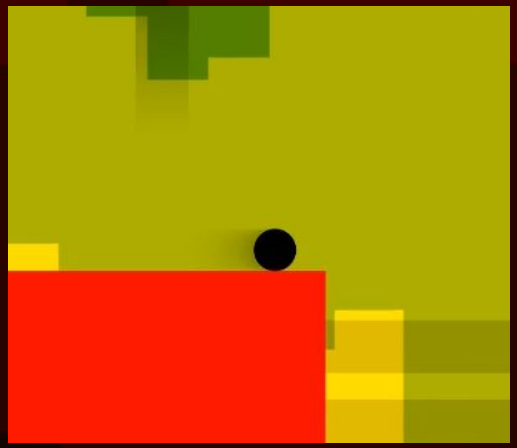
- Game elements wait for music events to control animation
- Music system observes `AudioSource.time` or `AudioSource.timeSamples`
- Tempo can be converted to seconds or samples
- Music events are triggered when a given time has been reached

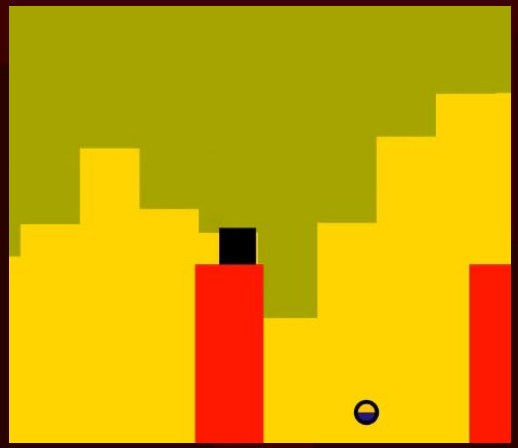


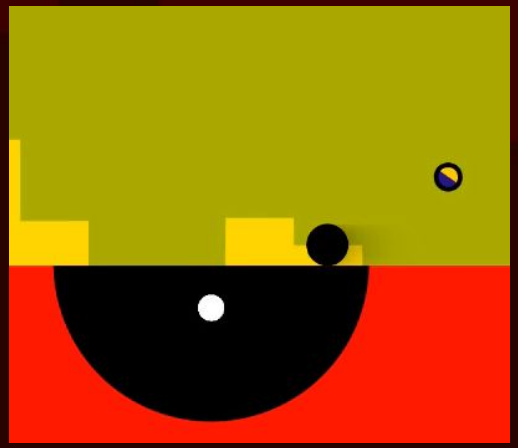

The Music Timing Problem

Interactive Music Mixing

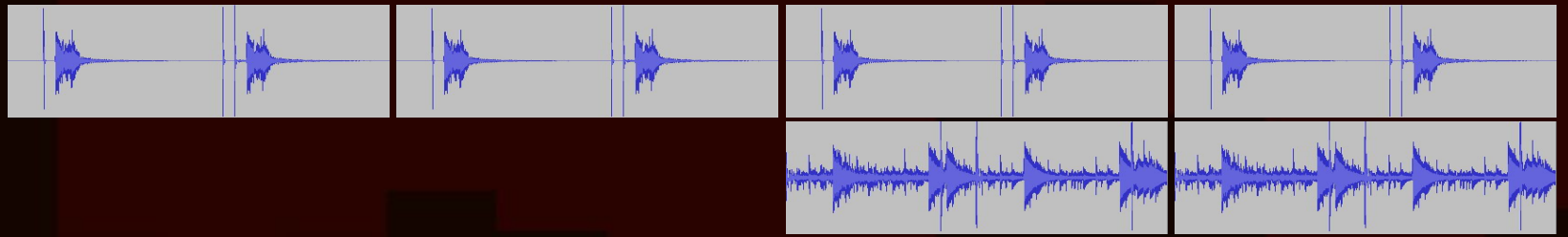
We would like to mix music interactively in Unity.







time



The Music Timing Problem

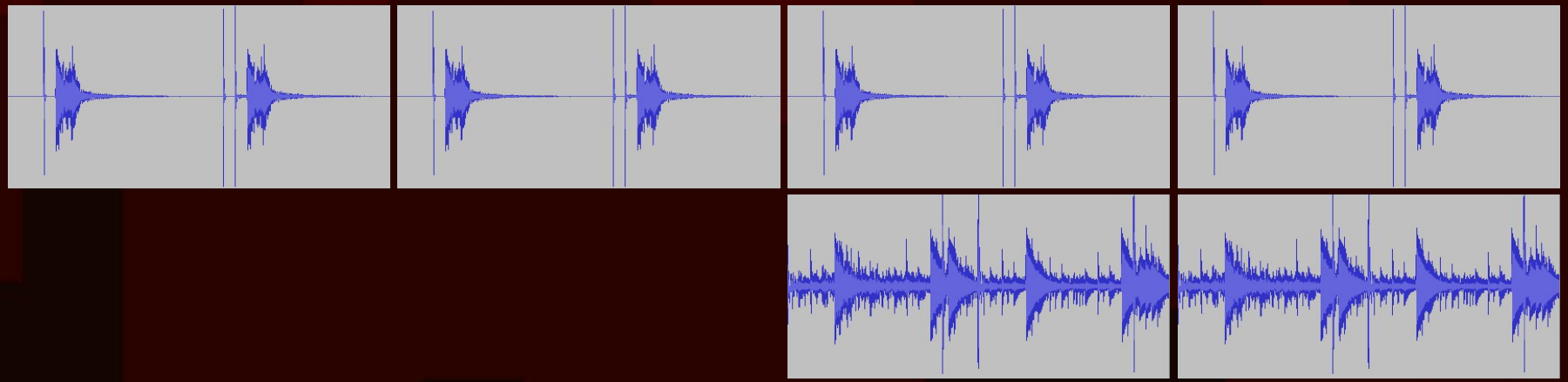
For beat-oriented music, loops should be synchronized with sample accuracy.

- That means a precision of 0.00002 s

Loop Transition

Goal:

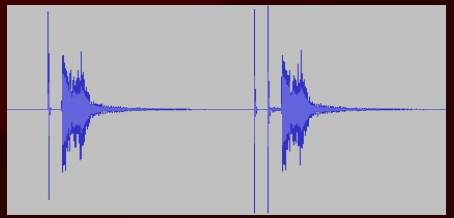
- Start loop A and let it run for a while.
- Then start loop B.
- B should be sample-accurately synchronized with loop A.



Loop Transition Problem

Start loop A:

```
audioSourceA.Play()
```



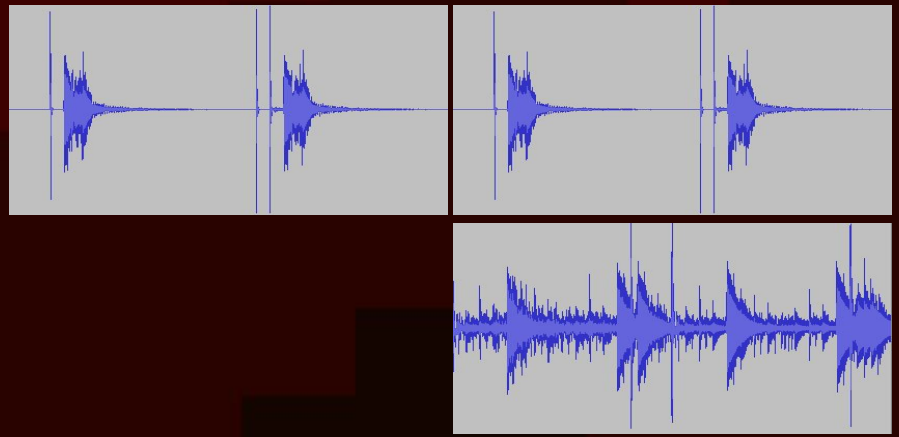
Loop Transition Problem

Start loop A:

```
audioSourceA.Play()
```

Exactly when A loops, start loop B:

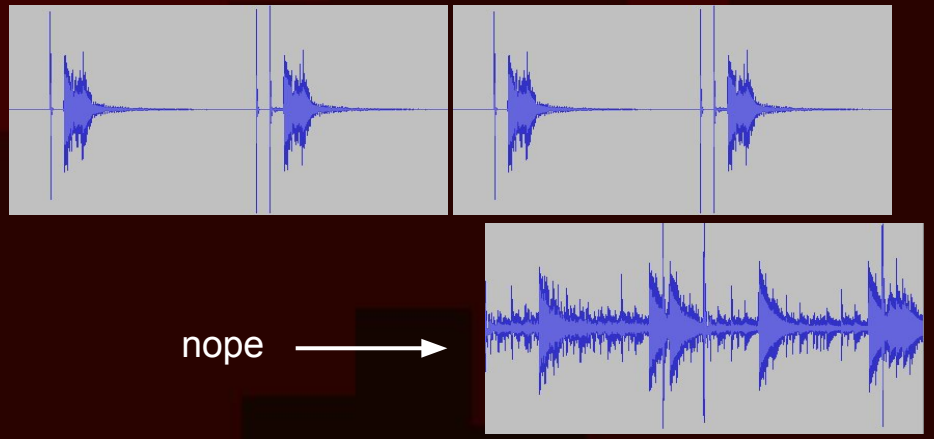
```
Wait for A to loop, then:
  audioSourceB.Play()
```



Loop Transition Problem

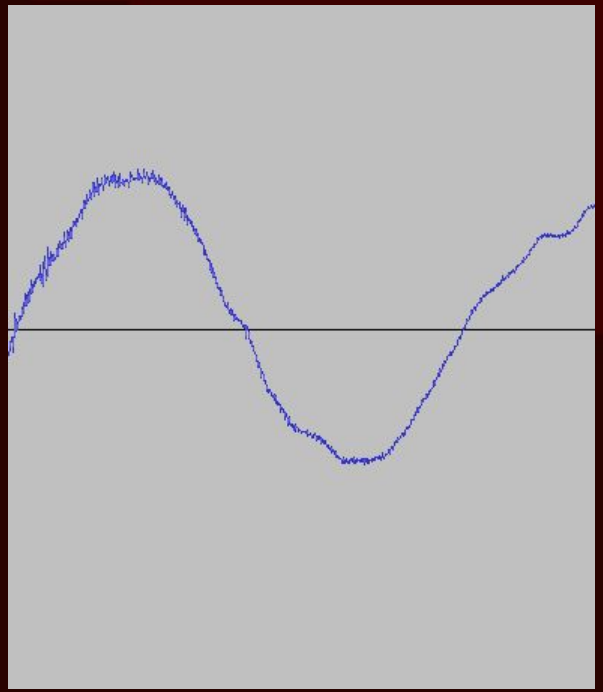
In practice, this approach is not going to work.

Why?



Audio Rendering

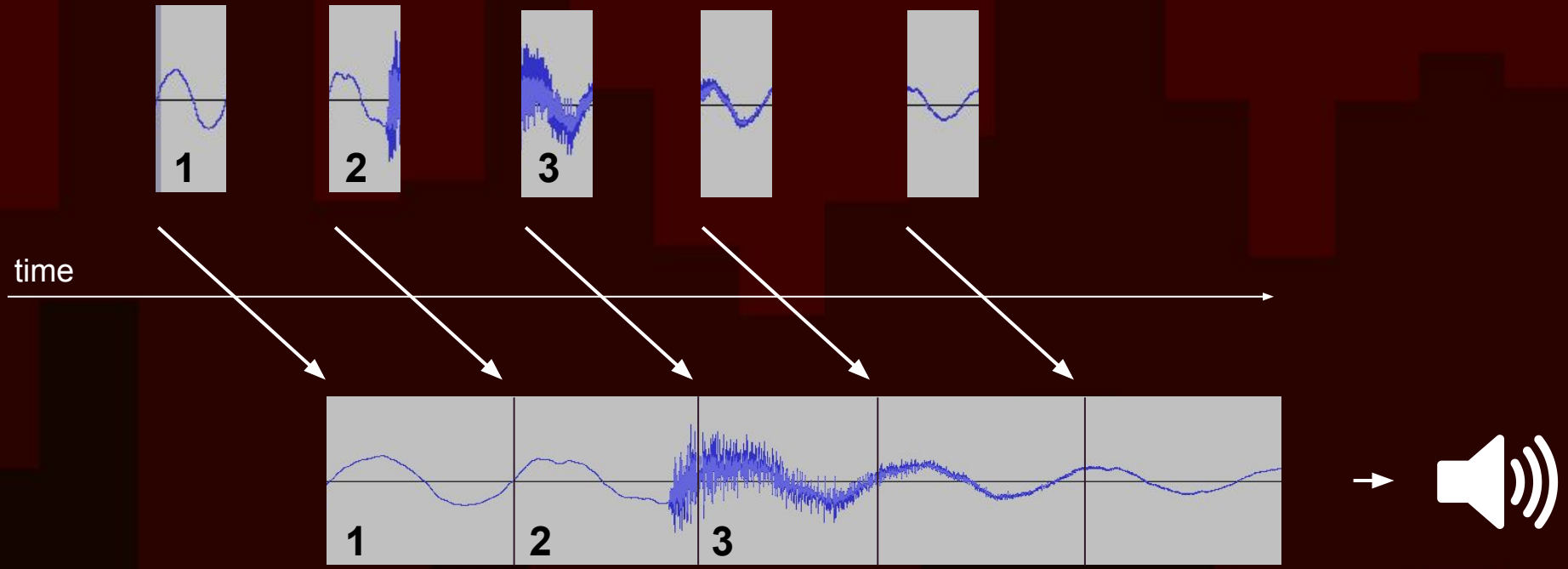
Audio is rendered a fixed number of samples at a time:



1024 sample buffer, 21 ms

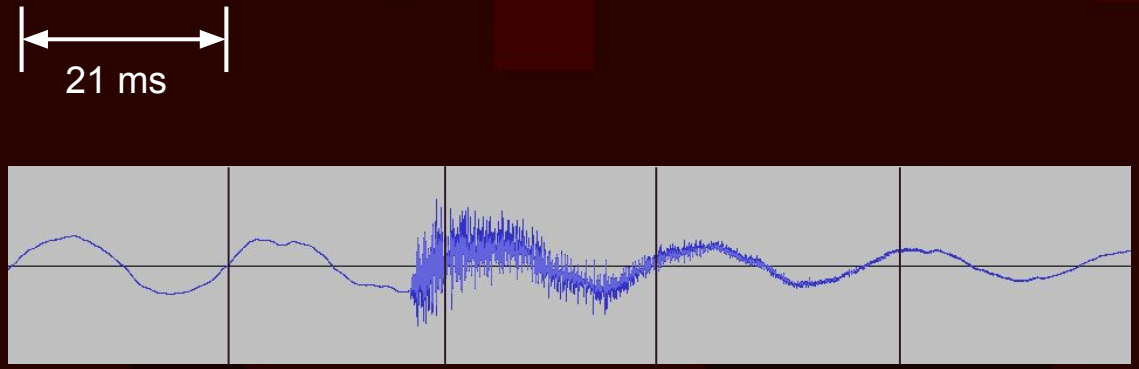
Audio Rendering

The sound card plays a buffer while the next one is being rendered:



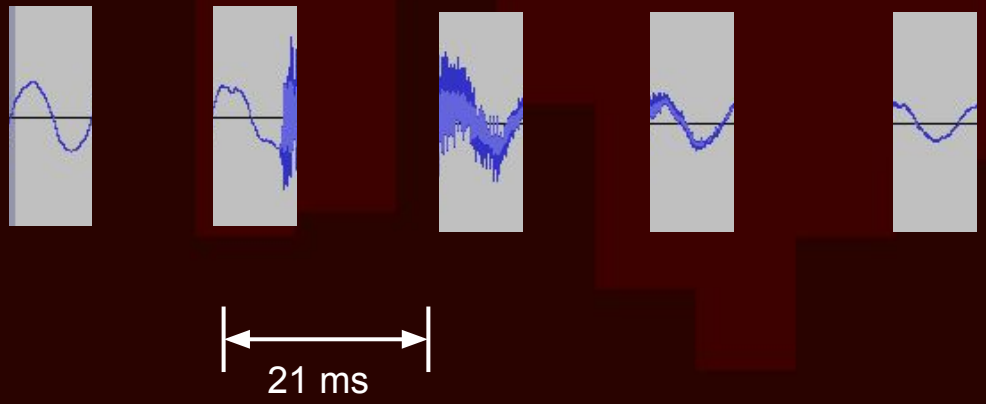
Audio Rendering

If buffers are 1024 samples long, we need a new one every 21 ms.



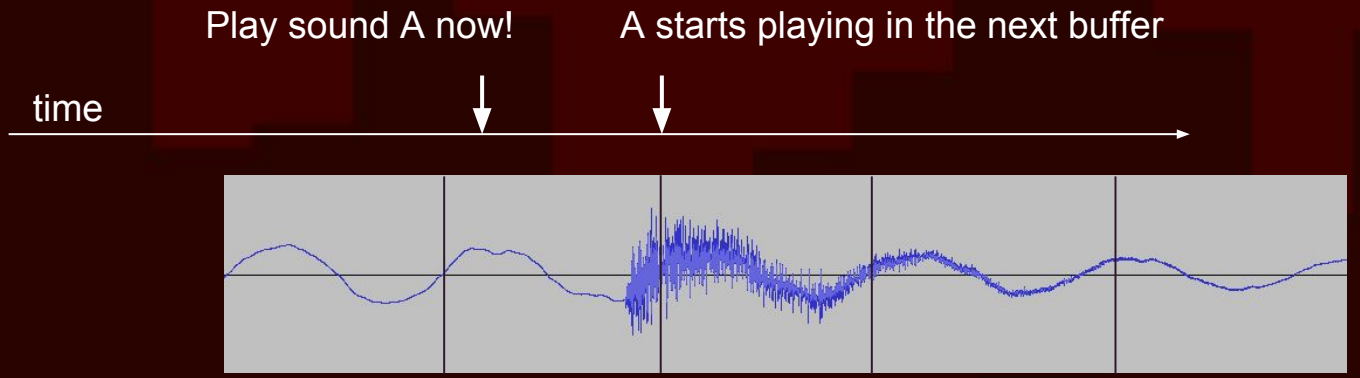
Audio Rendering

So, in this case, a new buffer is rendered every 21 ms:



Audio Rendering

- New sounds won’t start immediately, but earliest in the next audio buffer
- Their start time will also be quantized to buffer start times, e.g. 21 ms



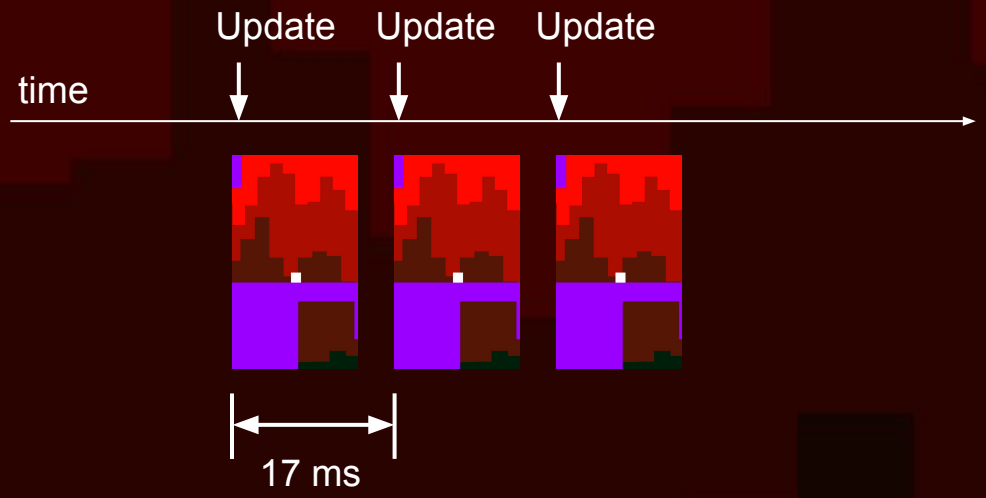
Audio Rendering

In Unity, our audio code will probably be in an Update method

```
using UnityEngine;

public class MyAwesomeScript : MonoBehaviour
{
    public AudioSource mySound;

    // Use this for initialization
    void Start()
    {
    }

    // Update is called once per frame
    void Update()
    {
        mySound.Play();
    }
}
```

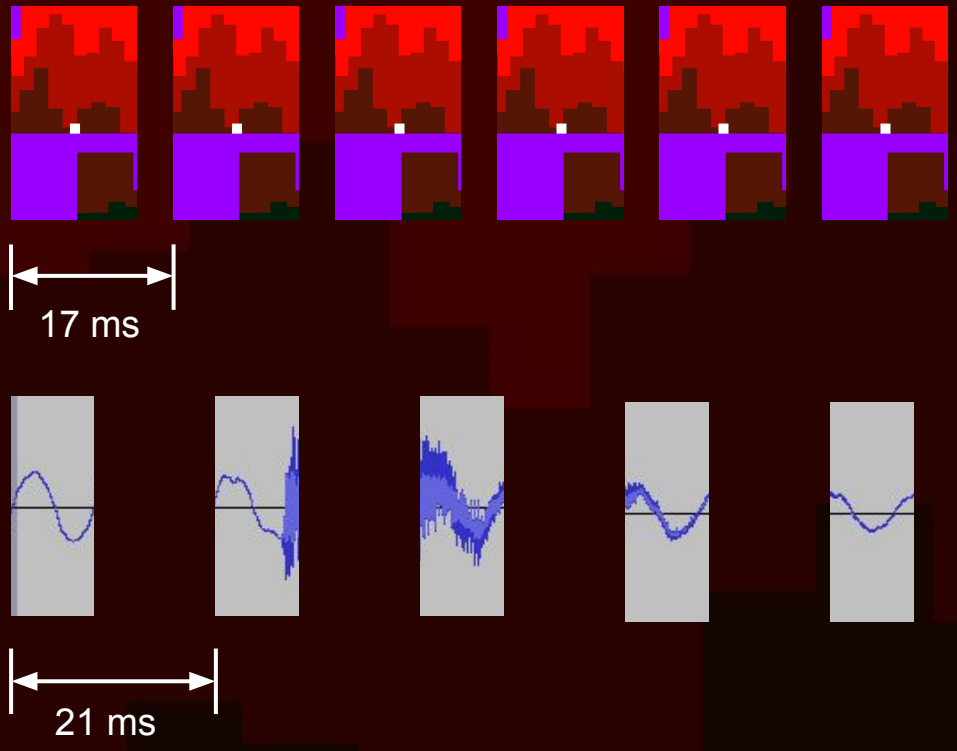
Audio Rendering

Unity Update methods are called for every frame rendered, e.g. 17 ms (at 60 FPS):



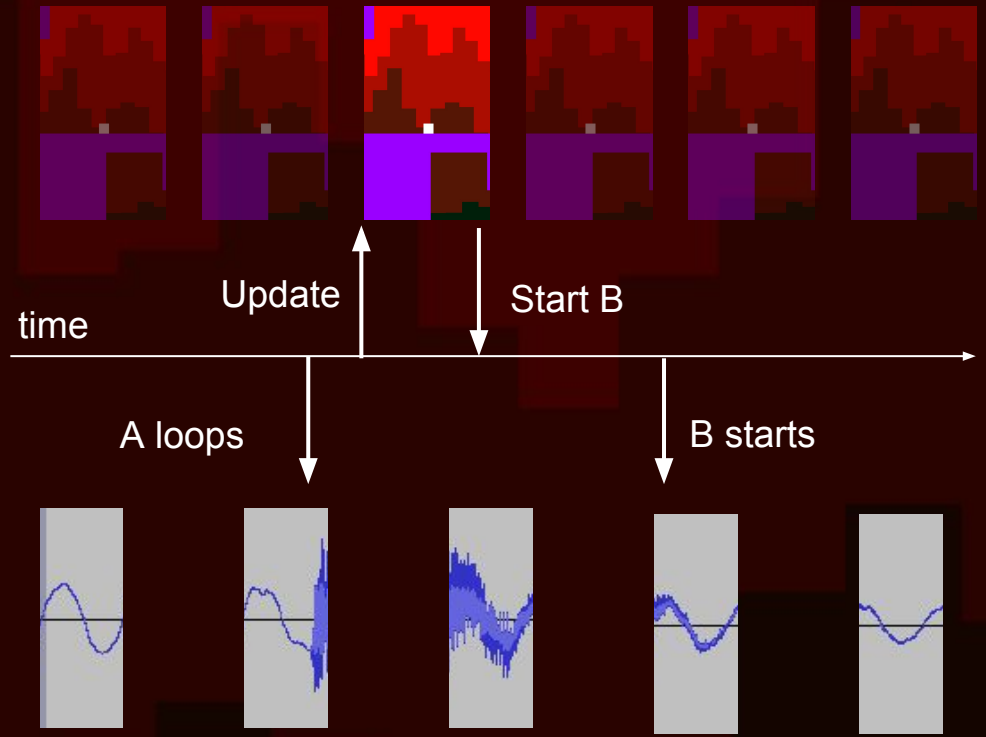
Audio Rendering

Audio buffers and video frames are not synchronized:



Audio Rendering

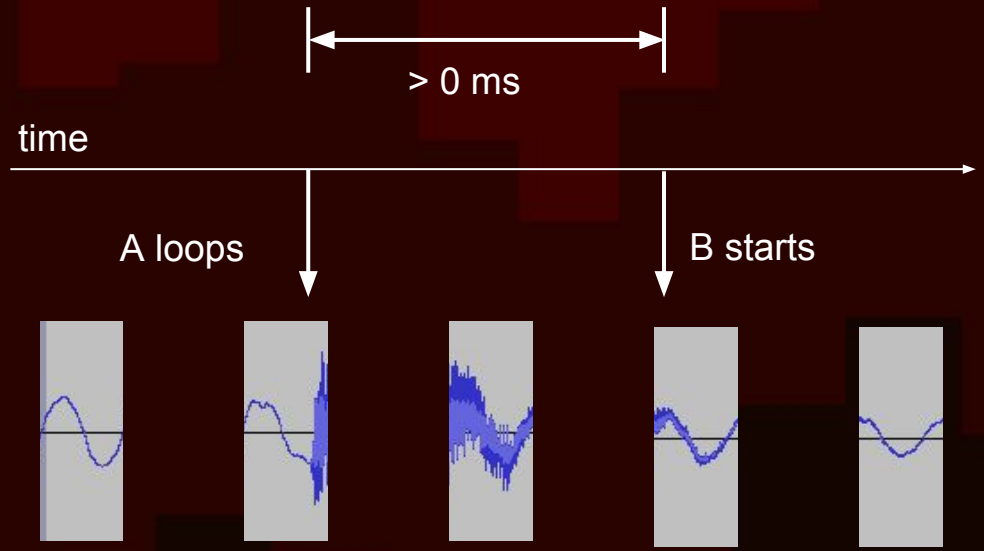
So, if we want to start a sound B exactly when another sound A loops...



Immediately is Too Late

So, if we want to start a sound B exactly when another sound A loops...

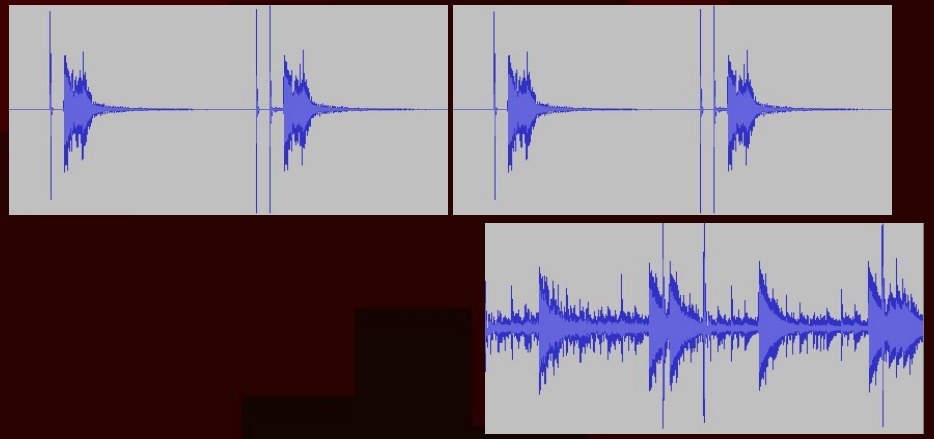
Detecting it in Update and playing B immediately is too late!



Immediately is Too Late

So, if we want to start a sound B exactly when another sound A loops...

Detecting it in Update and playing B immediately is too late!



Alternative Approach

Another approach:

Getting current time of loop

```
nowTime = AudioSource.time
```

and delay the new sound to be synchronized:

```
delay = startTime - nowTime
AudioSource.PlayDelayed(delay)
```

Alternative Approach

Another approach:

Getting current time of loop

```
nowTime = AudioSource.time
```

and delay the new sound to be synchronized:

```
delay = startTime - nowTime
AudioSource.PlayDelayed(delay)
```

← audio frame update could occur here, `nowTime` would be wrong

But when we read the value and start playing the new sound, time will keep running while we're doing it, and we still can't guarantee sample-accuracy.

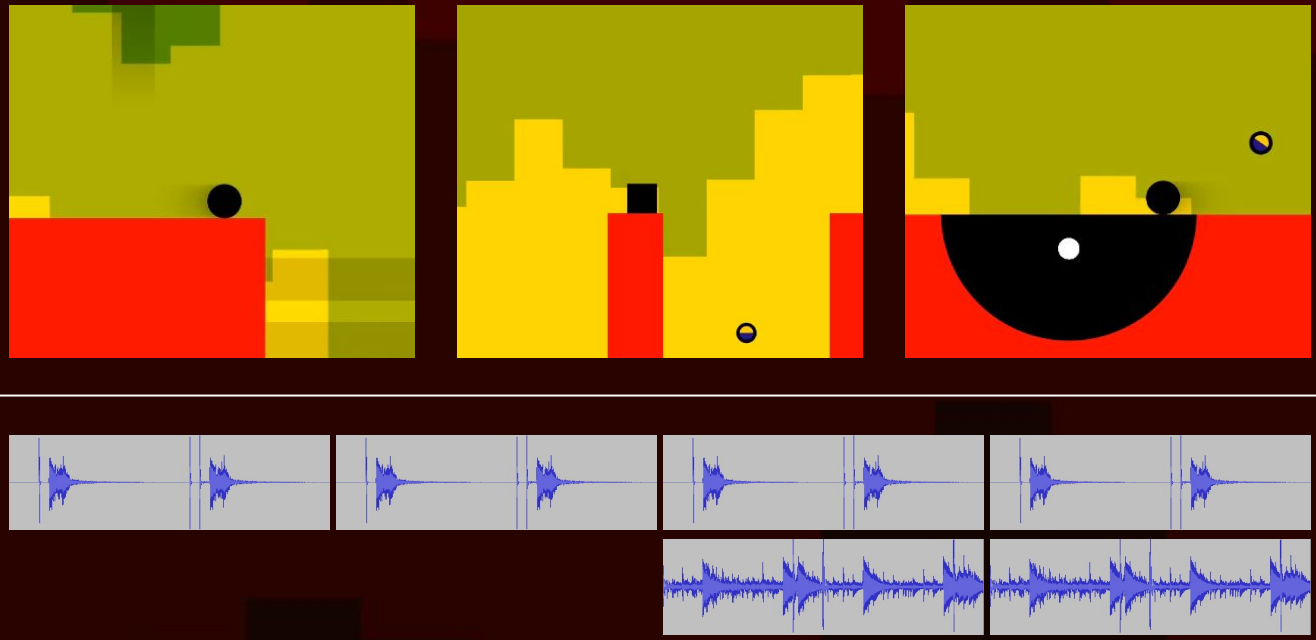
Summary

- Synchronizing loops with sample accuracy is tricky
- Audio is rendered in buffers, delaying and quantizing sounds
- Unity Update calls correspond to video frames, not audio buffers
- Immediately is too late: detecting loop and reacting in Update results in a delay
- Starting sounds in advance with PlayDelayed is also problematic


Music Timing in 140

Music Timing in 140

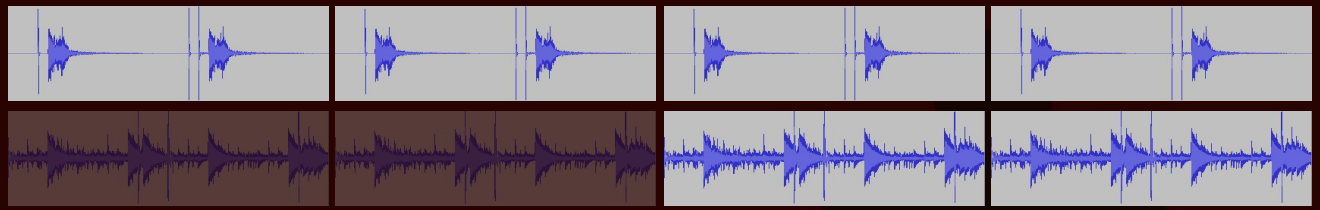
- We wanted the music to be mixed interactively with the gameplay.
- Loops should be sample-accurate.
- We were using Unity 3 at the time, which limited our options.



Music Requirements

Simple solution with sample-accurate timing:

- All music must be loops of a fixed length, or multiples of that length.
- Start all loops in same frame, possibly muted.



Music Requirements

Simple solution with sample-accurate timing:

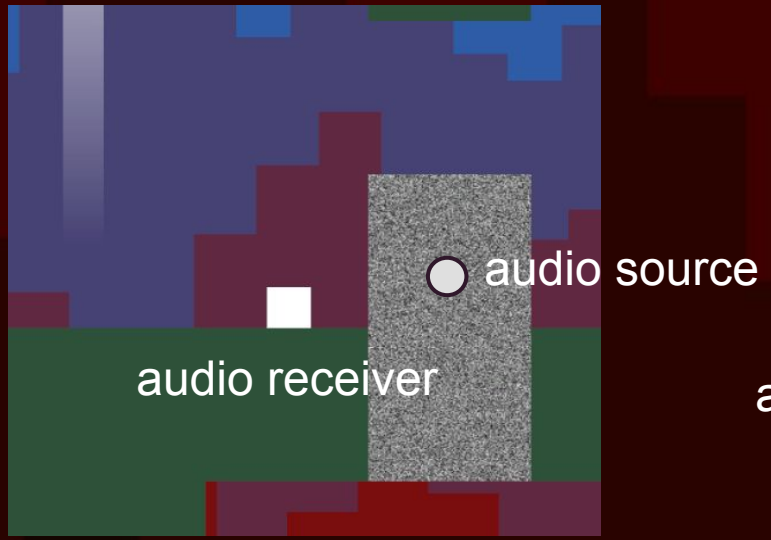
- All music must be loops of a fixed length, or multiples of that length.
- Start all loops in same frame, possibly muted.

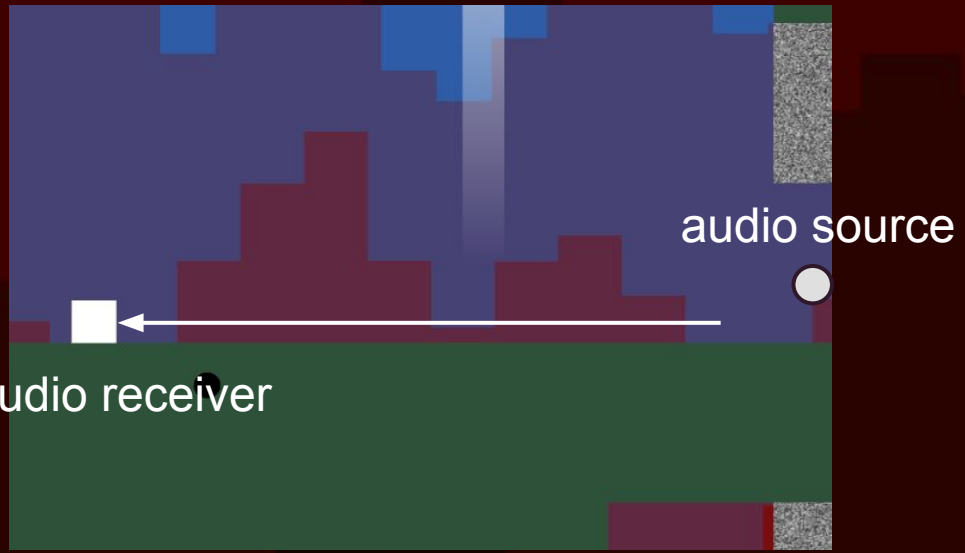
During game progression:

- Control volume/muting and pan.
- Never change pitch unless just before stopping a loop.

Attenuation and Panning

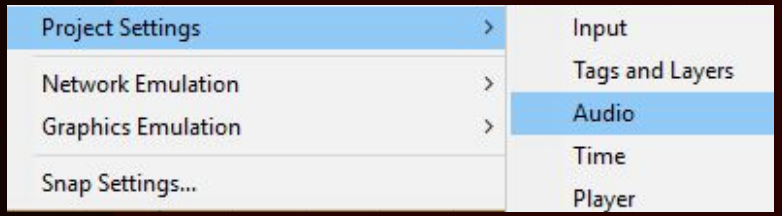
Simple attenuation and panning for music loops using the built-in audio system:

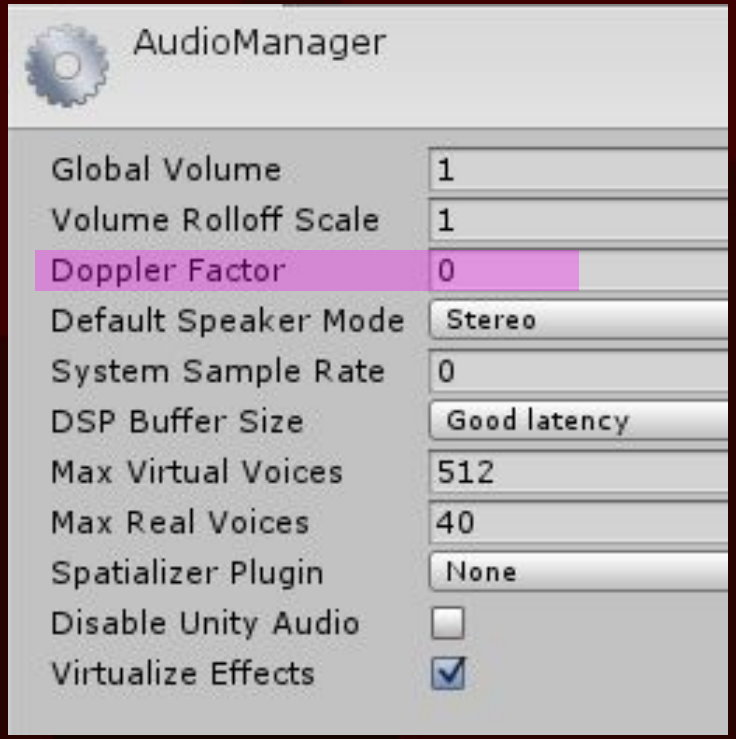




Doppler Effect

- Disable Doppler effect!



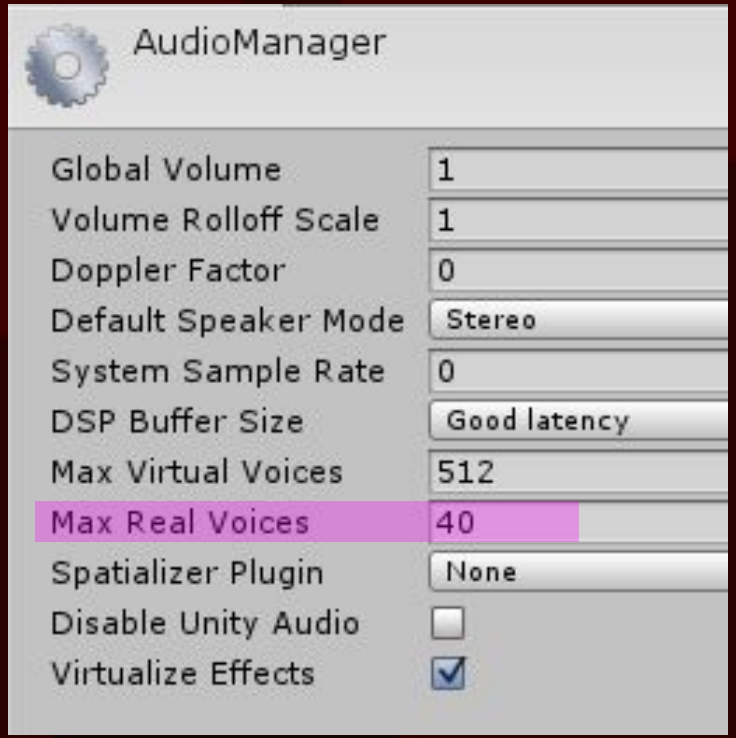


Max Real Voices

In Unity 3, if more than 32 sounds are playing at once, we lose sample accuracy!

Same limitation in Unity 5, but the number can be increased.

140 currently uses 40 voices.



Summary

- In 140, we start all loops at once and control their volume.
- Volume and pan using built-in audio system.
- Disable Doppler effect.
- More than ‘Max Real Voices’ sounds at once: no sample accuracy.

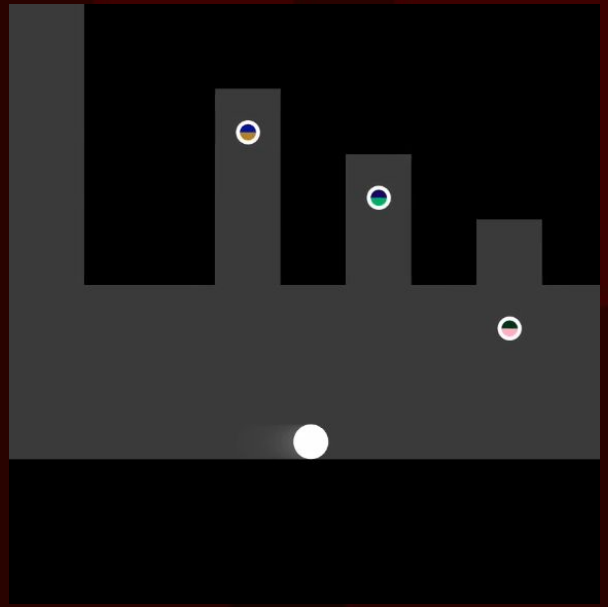
Fun Audio Tricks

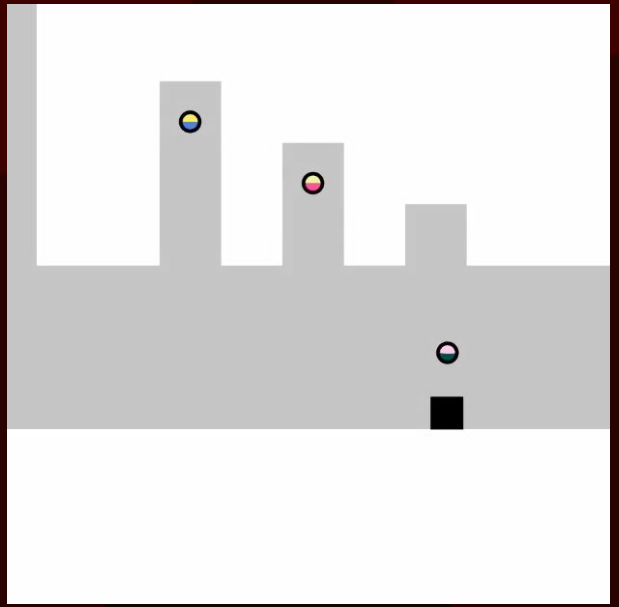
Fun Audio Tricks

- Modulation
- Cassette tape jam
- Downsampling
- Fake crash

Menu Modulation

- When picking up a mirror mode key, modulate ambient track from Cm to Dm.
- Track contains no rhythmic elements, so loop synchronization is not an issue.





From Semitones to Frequency

Modulate ambient track from Cm to Dm:

Frequency of D relative to C (+2 semitones, well-tempered):

$$2^{2/12} \sim 1.12246204830937$$

From Semitones to Frequency

Modulate ambient track from Cm to Dm:

Frequency of D relative to C (+2 semitones, well-tempered):

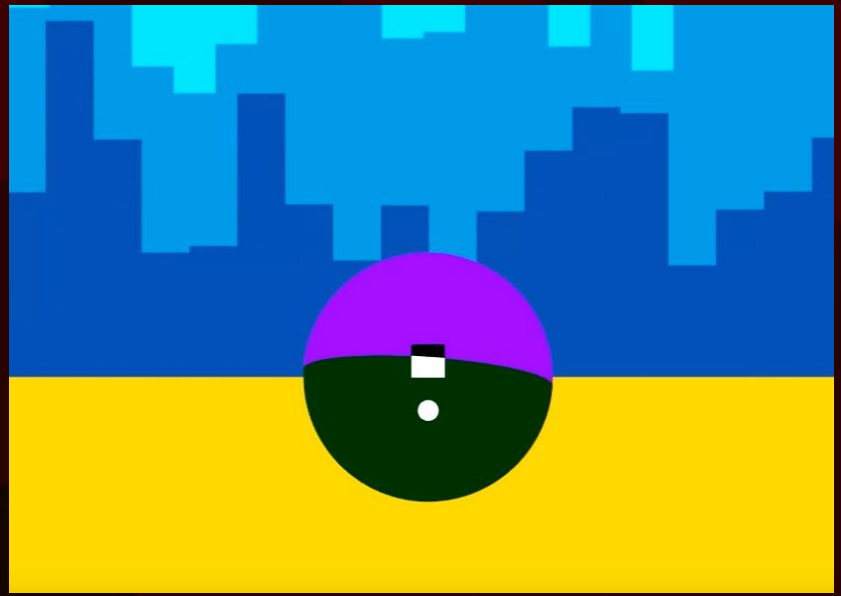
$$2^{2/12} \sim 1.12246204830937$$

Gradual pitch change code, as f goes from 0 to 1:

```
relativePitch = pow(2.0, 2.0 / 12.0)
source.pitch = lerp(1.0, relativePitch, f)
```

Cassette Tape Jam

When a key is delivered, the playing music is stopped with a cassette tape jam-inspired effect.



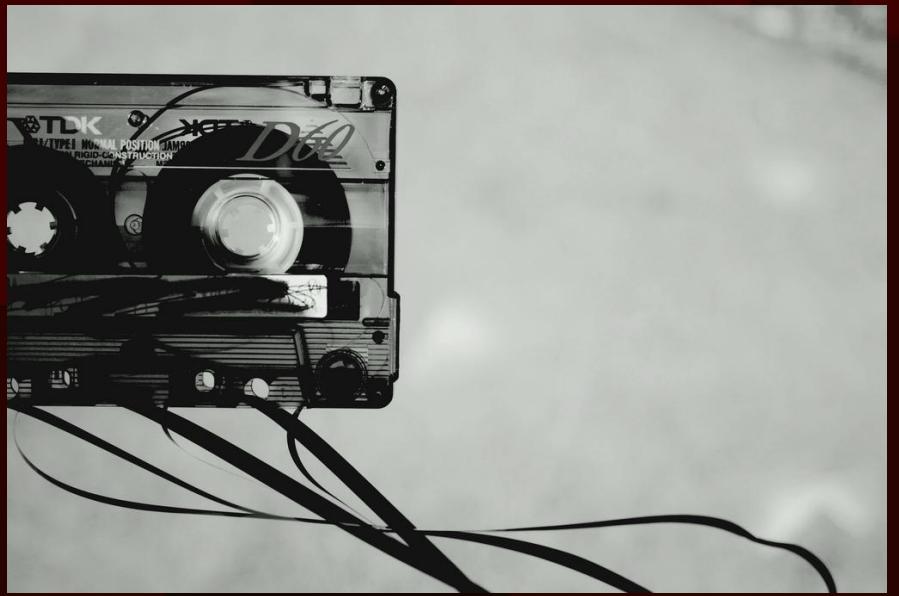


Image credit: Kristi Bogel

Cassette Tape Jam

The tape jam effect is achieved with:

- applying strong vibrato to all music tracks,
- enveloping pitch towards 0 and volume towards silence.

Cassette Tape Jam Code

The tape jam effect is achieved with:

- applying strong vibrato to all music tracks,
- enveloping pitch towards 0 and volume towards silence.

As f goes from 0 to 1:

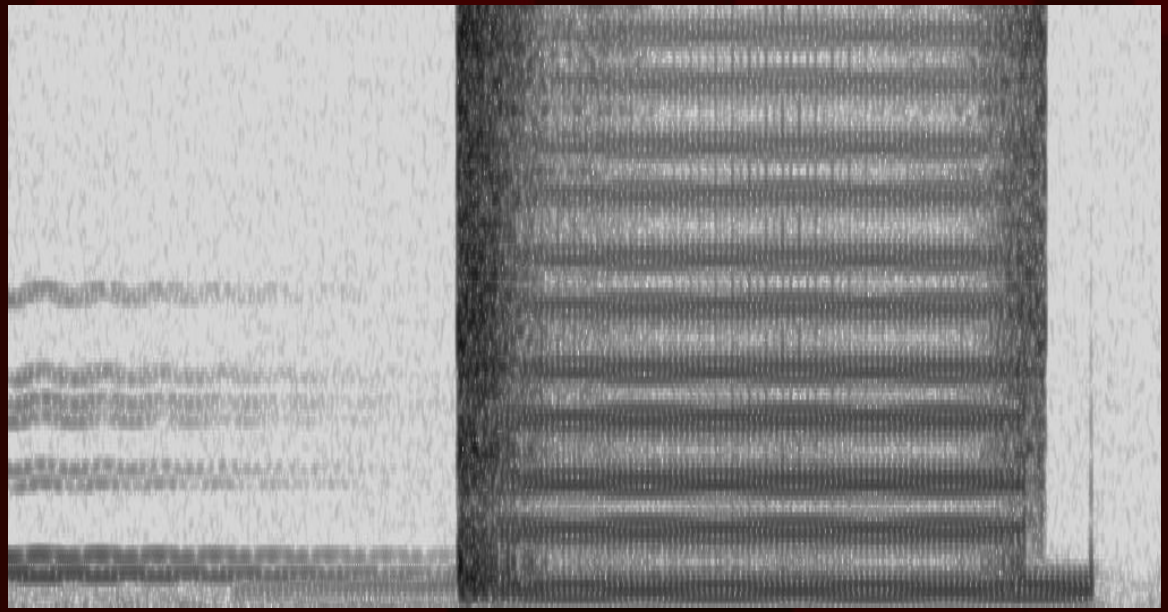
```
source.pitch = (1-f)                          // pitch envelope
             + sin(time * FREQ) * STRENGTH    // vibrato
source.volume = lerp(maxVolume, 0, f)         // amp envelope
```

Downsampling

When the player dies, the visuals turn black and white, and the audio is brutally distorted.

Downsampling

- The death audio effect is a simple variable downsampling filter.
- It sounds ugly on purpose.



Downsampling Filter Code

The simplest variable downsampling filter:

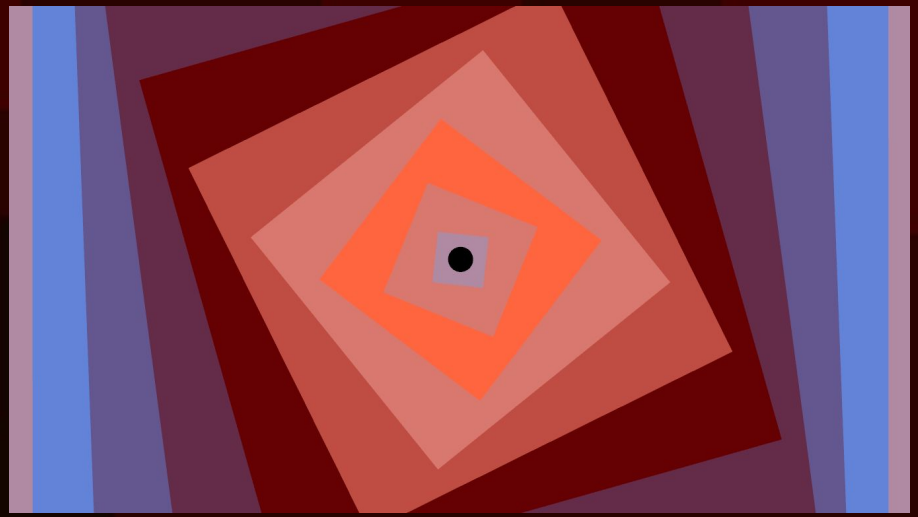
repeat every D’th sample D times.

```
void OnAudioFilterRead(float[] data, int channels) {
  if (D > 1) {                                  // if filter active
    for (int s = 0; s < data.Length; s+=2) {    // for all samples
      data[s]   = data[s / D * D];              // left channel
      data[s+1] = data[s / D * D + 1];          // right channel
    }
}
```

Fake crash

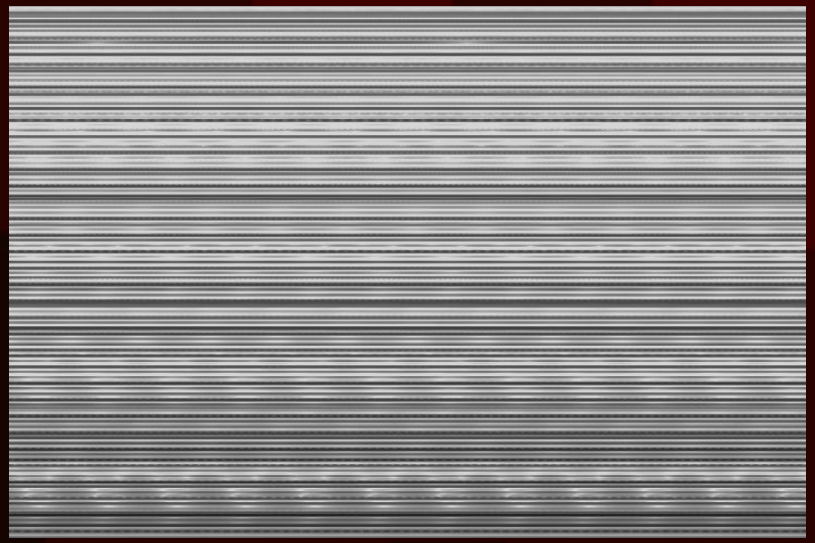
When the final boss is beaten, the game simulates the game crashing.

Or rather, how the game *would* crash if it was running on a SEGA Genesis.



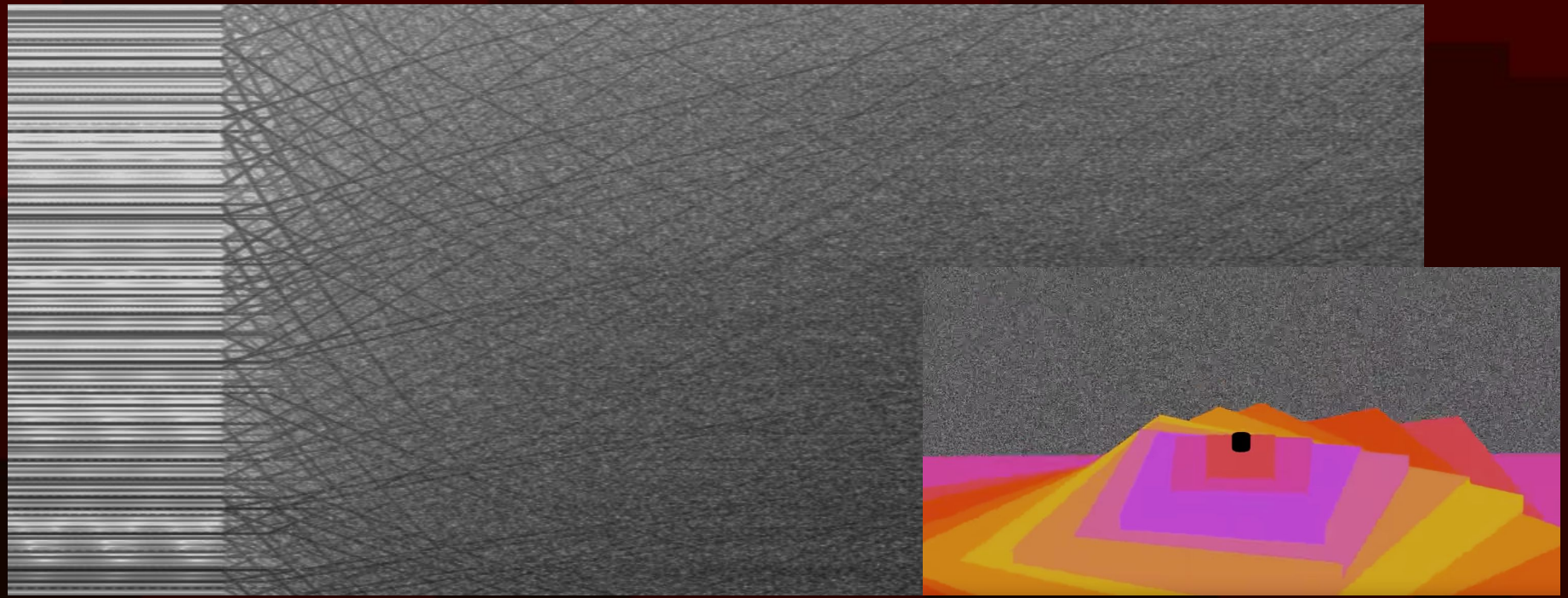
Fake crash

- The final chord of the boss fight and the screen is unchanged for 9 seconds, leaving at least one YouTuber very nervous.
- Crashes on old oscillator-based systems would have similar behaviour.



Glissando and 3D Rotation

- The oscillators slowly starts individually wandering towards a final chord.
- The game rotates the view, for the first time exposing a 3D world.



Summary

- Pitch change on playing track works for ambient music.
- Vibrato and amplitude and pitch envelopes simulate cassette tape jam.
- Downsampling filter is implemented `OnAudioFilterRead` method.
- Game ends with fake crash sound with hanging oscillators.
- Fake crash is resolved with oscillators gliding towards final chord.

Questions?