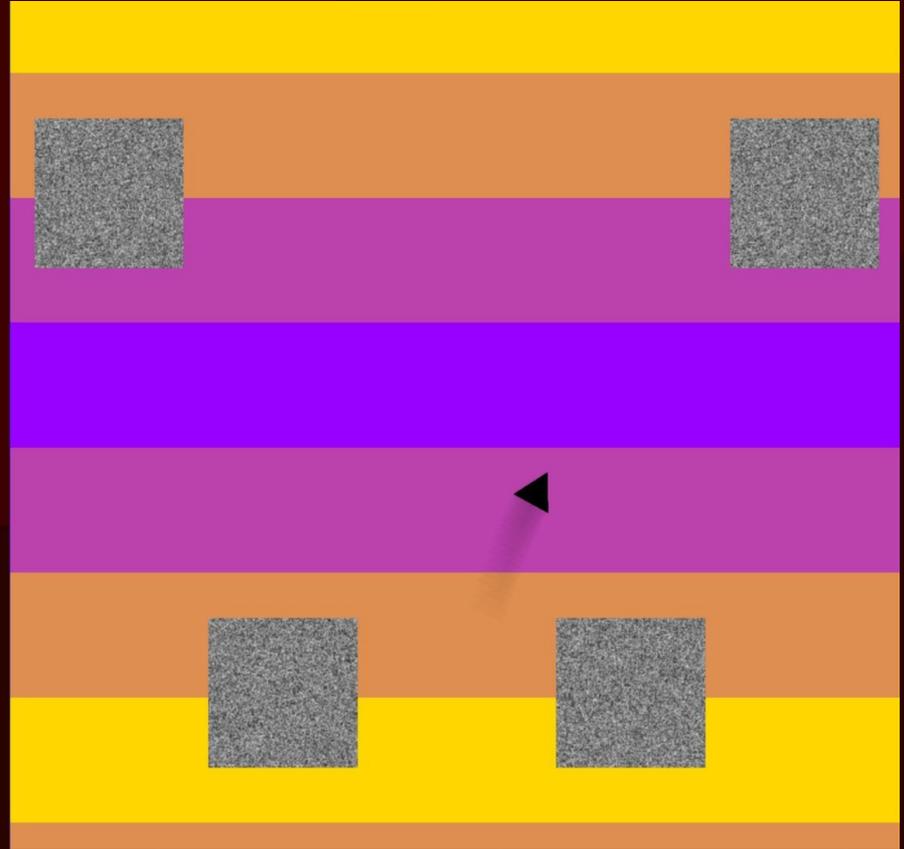


140

Sonic College 2026
Jakob Schmid

Overview

- Introduction
- Music-driven gameplay
- Adaptive music
- Fun audio tricks

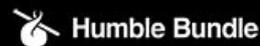


Jeppe Carlsen (design, programming)

Jakob Schmid (audio)

Niels Fyrst, Andreas Peitersen (visual design)

Developed as hobby project over 3 years



IGF award 2013

Excellence in Audio

Honorable mention: Technical Excellence

Spilprisen 2014

Sound of the Year

Nordic Game Award 2014

Artistic Achievement



140 Soundtrack

Vinyl

- iam8bit

Digital

- Steam
- GOG.com
- Spotify
- iTunes
- Amazon

Side A	Side B
140 Title 2:50	140 Part 1 5:07
140 Part 1 0:20	140 Part 2 5:23
140 Part 2 0:20	140 Part 3 5:07
140 Part 3 0:20	140 Part 4 4:48
140 Part 4 0:20	140 Part 5 4:14
140 Part 5 0:20	140 Menu 2:48

Composed and produced by Jakob Schmid
www.iam8bit.dk / www.carlsongames.com
Created with Ableton Live, Reaktor, Kurz X1 software synthesizer
Vinyl produced by [Lambert von Lambbit.com](#) 0821-0871
Mastered for vinyl by David Gardner, Saffron Audio Mastering
The game 140 was created by Jeppe Carlsen, Jakob Schmid, Niels Forst, Andreas Arvid Pedersen
Thanks to Martin Stig Andersen, Peter Ruchardt, Mikkel Svendsen, Mikkel Gjel, SBS Gunner Nyberg, Christian Vogel, Sprog Rive, Jan W. Sillesen, Renee White, Christian Villan
140 is a Carlsen Games. The 140 soundtrack by Jakob Schmid is licensed under a Creative Commons Attribution 4.0 International License. To view a copy of this license, visit www.creativecommons.org/licenses/by/4.0/

Includes:
Digital Soundtrack
Steam Code for Full Game

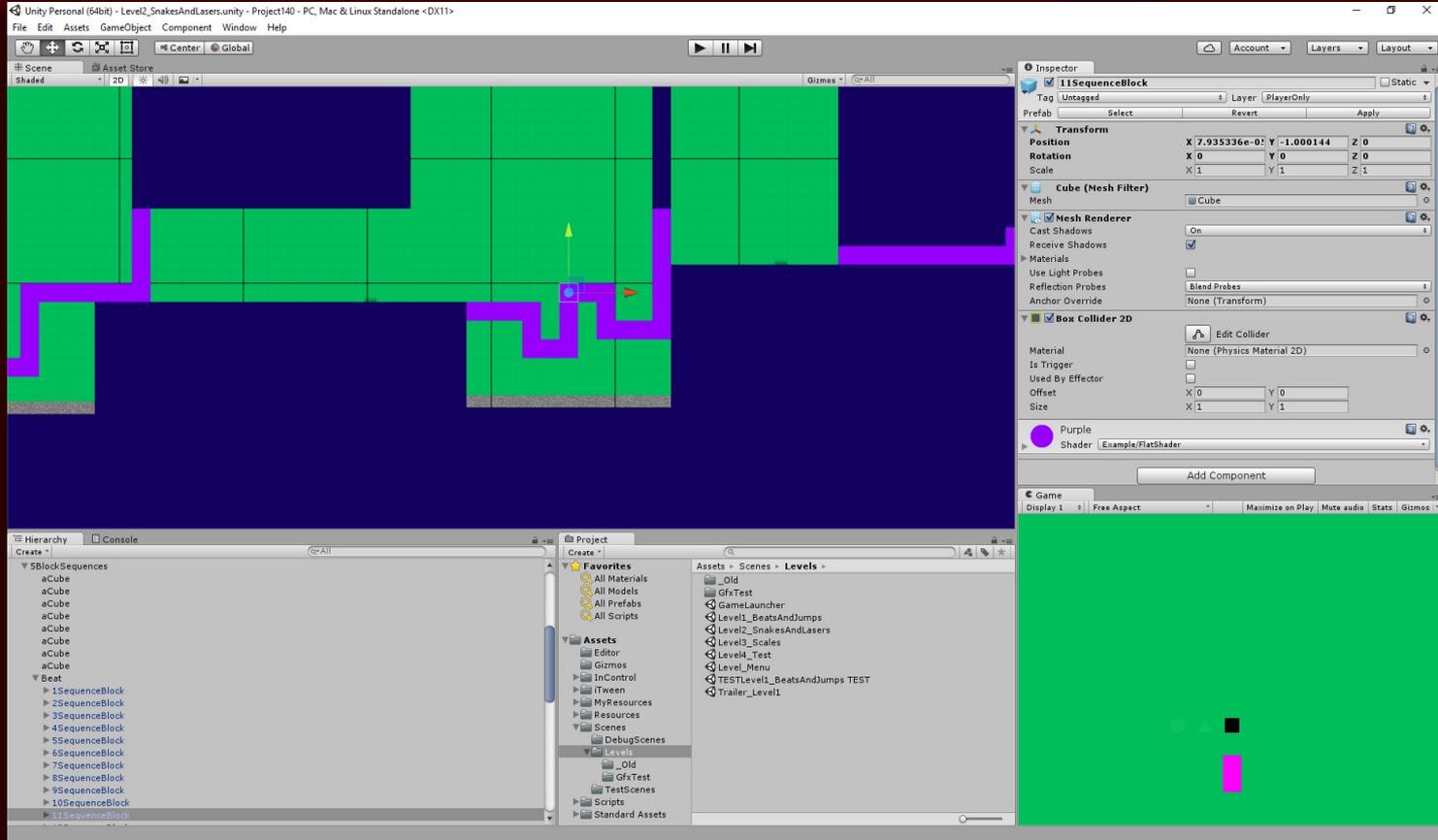
Music by:
Jakob Schmid

Carlsen Games
iam8bit
iam8bit.com

140 Vinyl Soundtrack
Limited Edition of 1400

schmid.dk/games/140/soundtrack/

Developed in Unity 3



Audacity

The screenshot displays the Audacity interface with a spectrogram of an audio file. The top menu bar includes File, Edit, View, Transport, Tracks, Generate, Effect, Analyze, and Help. The transport controls show a play button and a time display of 29.0 to 48.0 seconds. The main window shows a spectrogram with a frequency range from 0.0k to 22.1k Hz. The x-axis represents time in seconds, and the y-axis represents frequency in kHz. The spectrogram shows a complex waveform with various frequency components over time. The bottom status bar indicates the Project Rate is 48000 Hz, Snap To is Off, Selection Start is 002,245,446 samples, End is 000,000,000 samples, and Audio Position is 000,000,000 samples. The status is Stopped.

140.4

File Edit View Transport Tracks Generate Effect Analyze Help

Click to Start Monitoring MME Stereo Mix (Realtek Hi 2 (Stereo) Ret Speakers (Realtek High

29.0 30.0 31.0 32.0 33.0 34.0 35.0 36.0 37.0 38.0 39.0 40.0 41.0 42.0 43.0 44.0 45.0 46.0 47.0 48.0

X: 140.4 22.1k
Stere, 44100Hz
32-bit float
Mute Solo
19.5k
19.0k
18.5k
18.0k
17.5k
17.0k
16.5k
16.0k
15.5k
15.0k
14.5k
14.0k
13.5k
13.0k
12.5k
12.0k
11.5k
11.0k
10.5k
10.0k
9.5k
9.0k
8.5k
8.0k
7.5k
7.0k
6.5k
6.0k
5.5k
5.0k
4.5k
4.0k
3.5k
3.0k
2.5k
2.0k
1.5k
1.0k
0.0k

Project Rate (Hz): 48000 Snap To: Off Selection Start: 002,245,446 samples End: 000,000,000 samples Audio Position: 000,000,000 samples

Stopped. Click and drag to select audio, Ctrl-Click to scrub, Ctrl-Double-Click to scroll-scrub, Ctrl-drag to seek

Music-driven Gameplay

The background is a pixelated, abstract composition. The top half is a solid red color. The bottom half is a solid purple color. A white path starts from the bottom left, moves right, then up, then right again, ending at a small white square in the center. To the right of this path, there is a vertical column of four purple squares, with a white square at the top. The overall style is reminiscent of early computer graphics or video game aesthetics.

Moving a Platform



wait for 16th note #1



start moving



wait for 16th note #8



start moving

Basic Approach

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

Unity built-in audio:

`AudioSource.time` (seconds)

`AudioSource.timeSamples` (samples)

FMOD Unity Integration:

`EventInstance.getTimelinePosition` (milliseconds)

Useful Calculations

For a given tempo, how long is a note in seconds?

Example: 16th note in 140 BPM

Tempo

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

Duration of 16th note:

??? s/note



Tempo

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

140 beat/m

Duration of 16th note:

??? s/note



Tempo

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

$$140 \text{ beat/m} * 4 \text{ note/beat} = 560 \text{ note/m}$$

Duration of 16th note:

??? s/note



Tempo

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

$$\begin{aligned} 140 \text{ beat/m} * 4 \text{ note/beat} &= 560 \text{ note/m} \\ &= 560/60 \text{ note/s} \end{aligned}$$

Duration of 16th note:

??? s/note



Tempo

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

$$\begin{aligned} 140 \text{ beat/m} * 4 \text{ note/beat} &= 560 \text{ note/m} \\ &= 560/60 \text{ note/s} \end{aligned}$$

Duration of 16th note:

$$60/560 \text{ s/note}$$



Tempo

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

$$\begin{aligned} 140 \text{ beat/m} * 4 \text{ note/beat} &= 560 \text{ note/m} \\ &= 560/60 \text{ note/s} \end{aligned}$$

Duration of 16th note:

$$\begin{aligned} &60/560 \text{ s/note} \\ &\approx 0.10714 \text{ s/note} \end{aligned}$$

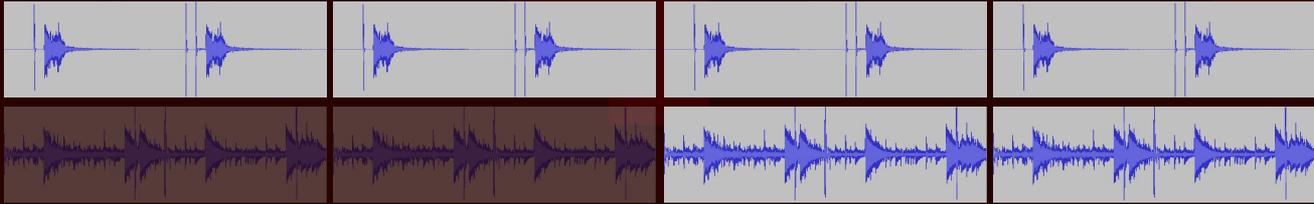


Adaptive Music

The background features a complex, abstract composition of overlapping shapes. A large, bright red area occupies the upper right and central portions. A vibrant purple shape is prominent in the top left and extends into the bottom left. Dark green, jagged shapes are scattered in the lower right and bottom center. The overall effect is a dynamic, multi-colored field.

Solution A: Synchronized Loops

- Vertical remixing
- All loops should be exactly same length, or integer multiples
- All loops should be started in the same frame, possibly muted
- New loops cannot be started
- Never change pitch



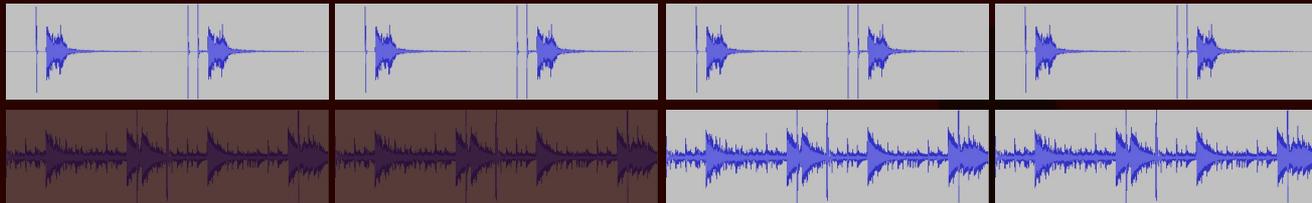
Music Requirements

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.



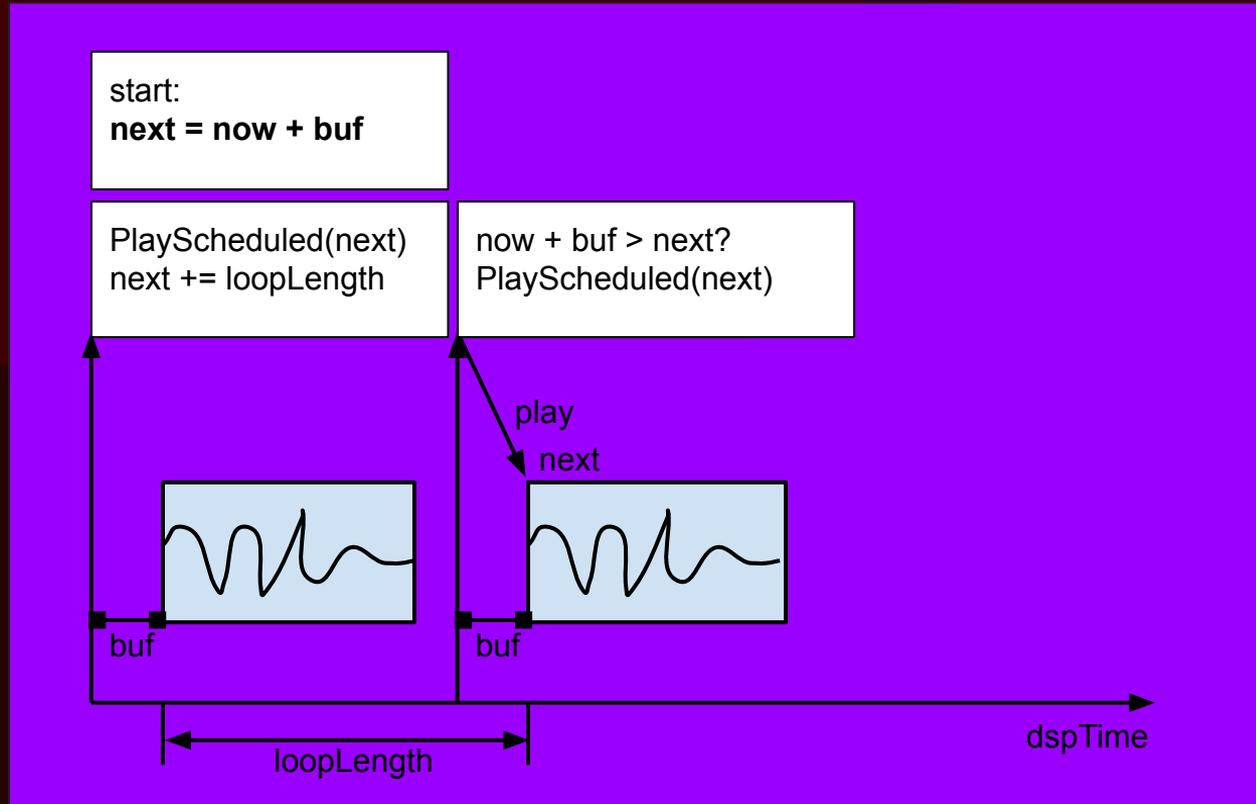
Masking Loops

Music loops can be masked:

- Fade using filters
- Delay
- Unmuting at specific beats



Solution B: PlayScheduled



Solution B: PlayScheduled

Start:

```
buf = 0.1 // as low as possible
next = AudioSettings.dspTime + buf
```

Update:

```
now = AudioSettings.dspTime
if(now + buf > next)
    audio.PlayScheduled( next )
    next += loopLength
```

- see schmid.dk/gallery/play_scheduled/ for C# code



Solution Comparison

Solution A: Synchronized Loops

- Very simple to implement
- Requires a loop for every single independent musical element
- Loops must be same length or integer multiple
- Pitch cannot be changed

Solution Comparison

Solution A: **Synchronized Loops**

- Very simple to implement
- Requires a loop for every single independent musical element
- Loops must be same length or integer multiple
- Pitch cannot be changed

Solution B: **PlayScheduled**

- Non-trivial implementation
- Flexible: Individual notes can be sequenced
- No length requirement for music sounds
- Pitch can be changed

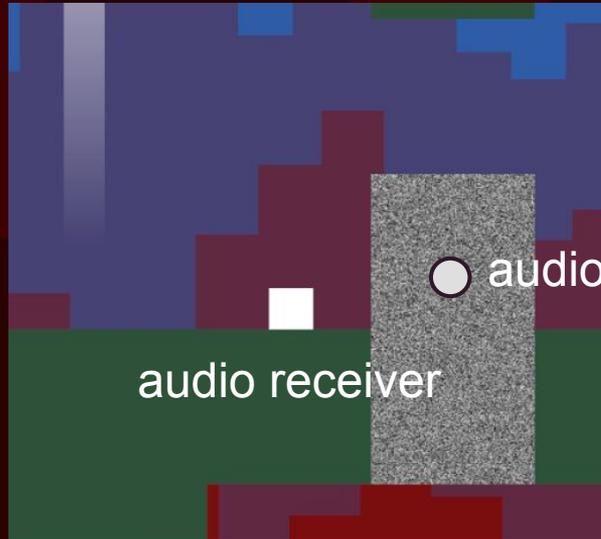


Localized Music Loops

- Music loops are “physically” placed in level geometry
- Dynamic mixing occurs as player moves around
- Certain areas can gain unique atmosphere based on music

Localized Music Loops

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



audio source

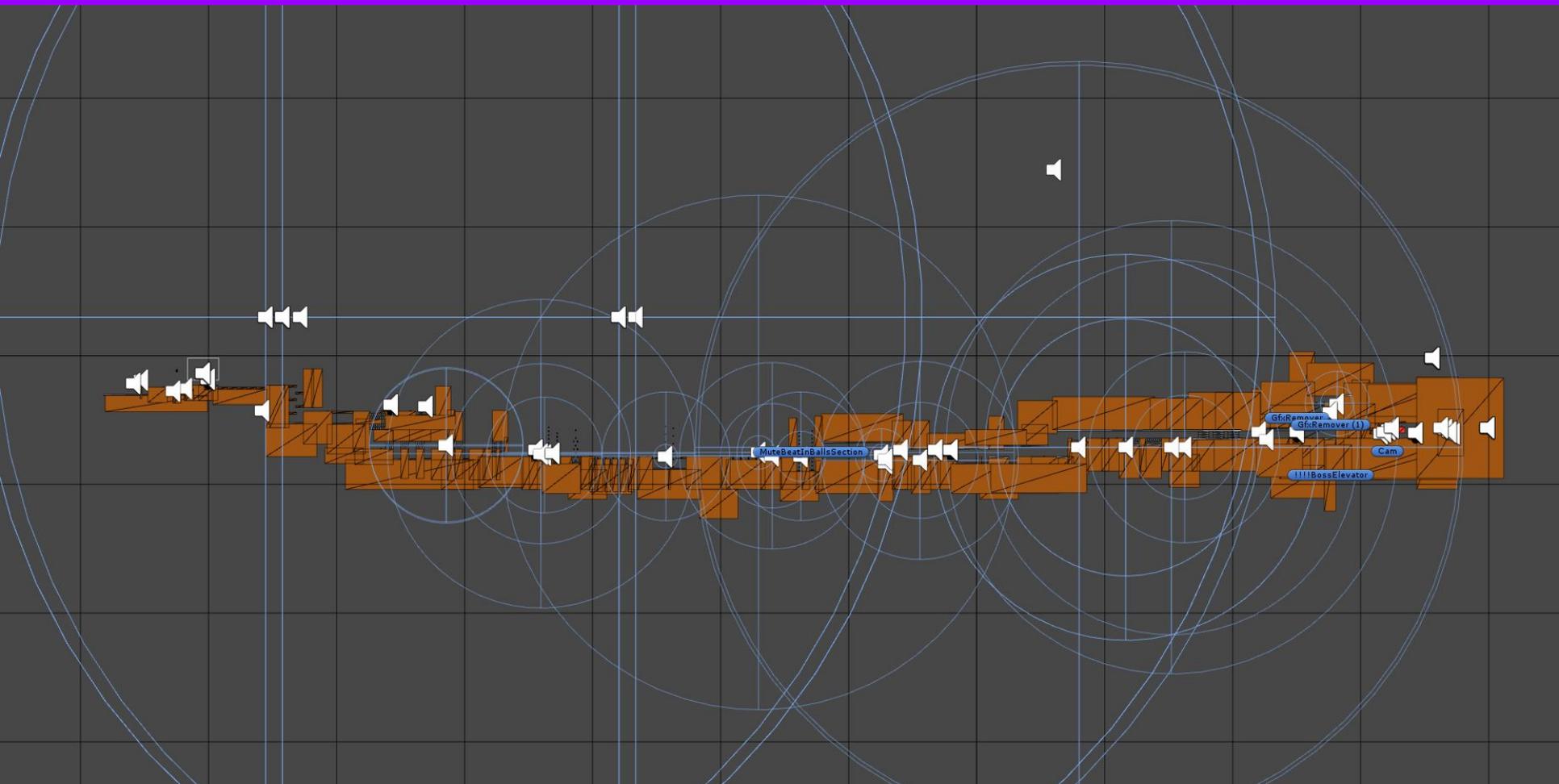
audio receiver



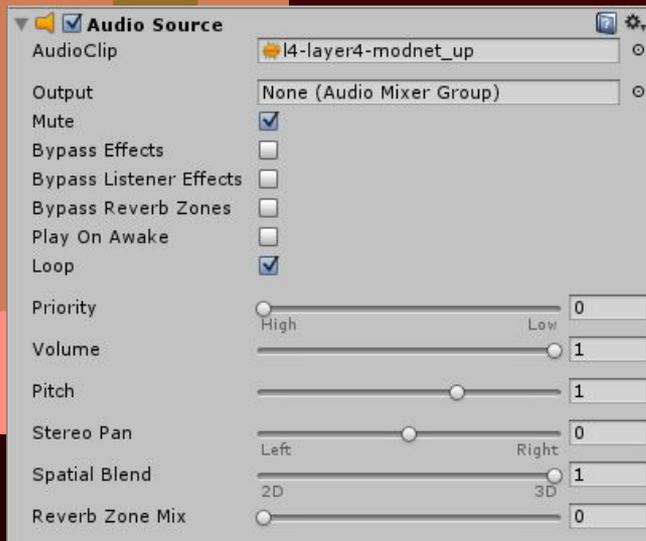
audio source

audio receiver

Localized Music Loops - Level 4

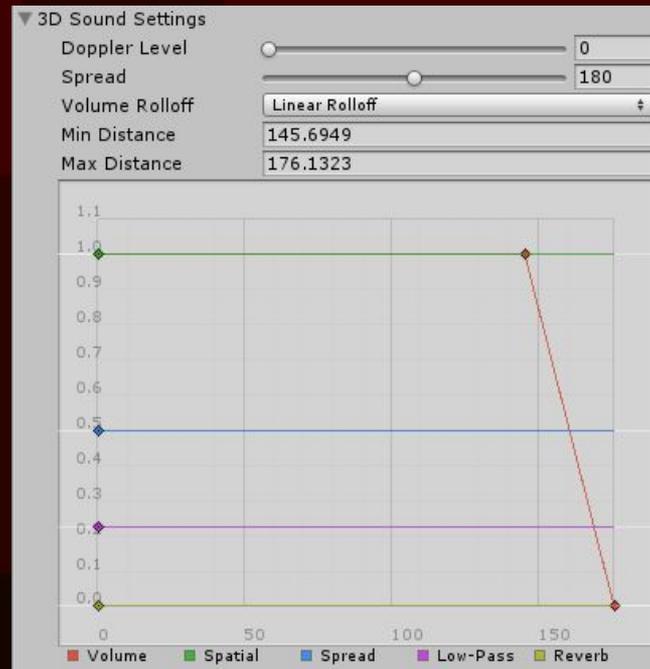


Example Audio Loop



Audio Source

- AudioClip: i4-layer4-modnet_up
- Output: None (Audio Mixer Group)
- Mute:
- Bypass Effects:
- Bypass Listener Effects:
- Bypass Reverb Zones:
- Play On Awake:
- Loop:
- Priority: 0
- Volume: 1
- Pitch: 1
- Stereo Pan: 0
- Spatial Blend: 1
- Reverb Zone Mix: 0



3D Sound Settings

- Doppler Level: 0
- Spread: 180
- Volume Rolloff: Linear Rolloff
- Min Distance: 145.6949
- Max Distance: 176.1323

Graph showing parameter values over distance (0 to 150+):

Distance	Volume	Spatial	Spread	Low-Pass	Reverb
0	1.0	1.0	0.5	0.5	0.0
145.6949	1.0	1.0	0.5	0.5	0.0
176.1323	0.0	0.0	0.0	0.0	0.0

Example Audio Loop Components

AudioSync component: handles all loops

- Handles fade in/out
- Controls filters
- Adds loop to 'layer' (group)

The screenshot displays the Unity Inspector for four audio components. The first component, **Audio Sync (Script)**, has the following settings: Script: AudioSync; Fade In Time: 2; Fade Out Time: 1; Fade On Mutes: checked; Override Start Pitch: -1; Add To Audio Layer: -1; Fade In On Play: unchecked; Dont Reset Volume On Umute: unchecked; Dont Stop On Mirror Death: unchecked; Fade With Filter: checked; Low Pass Filter: airborne (Audio Low Pass Filter); High Pass Filter: None (Audio High Pass Filter); Freq Filter Start: 10; Freq Filter End: 5000; Filter Curve Power: 2. The second component, **Audio Death Down Sample (Script)**, has: Script: AudioDeathDownSample; Beat Layer: 2; Death Volume Scale: 1; Unmute On Death: unchecked; Down Sample Scale: 1; Downsample: 1. The third component, **Audio Low Pass Filter**, has: Cutoff Frequency: 5000; Lowpass Resonance Q: 1. The fourth component, **Audio Echo Filter**, has: Delay: 643; Decay Ratio: 0.4; Wet Mix: 0.4; Dry Mix: 0.6.

Component	Property	Value
Audio Sync (Script)	Script	AudioSync
	Fade In Time	2
	Fade Out Time	1
	Fade On Mutes	<input checked="" type="checkbox"/>
	Override Start Pitch	-1
	Add To Audio Layer	-1
	Fade In On Play	<input type="checkbox"/>
	Dont Reset Volume On Umute	<input type="checkbox"/>
	Dont Stop On Mirror Death	<input type="checkbox"/>
	Fade With Filter	<input checked="" type="checkbox"/>
	Low Pass Filter	airborne (Audio Low Pass Filter)
	High Pass Filter	None (Audio High Pass Filter)
	Freq Filter Start	10
	Freq Filter End	5000
Filter Curve Power	2	
Audio Death Down Sample (Script)	Script	AudioDeathDownSample
	Beat Layer	2
	Death Volume Scale	1
	Unmute On Death	<input type="checkbox"/>
	Down Sample Scale	1
	Downsample	1
Audio Low Pass Filter	Cutoff Frequency	5000
	Lowpass Resonance Q	1
Audio Echo Filter	Delay	643
	Decay Ratio	0.4
	Wet Mix	0.4
	Dry Mix	0.6

Example Audio Loop Components

AudioDeathDownSample:
downsample filter (more on this later)

The screenshot displays the Unity Inspector for four audio components. The first component is 'Audio Sync (Script)', which includes settings for fade times, mute behavior, and filters. The second is 'Audio Death Down Sample (Script)', which controls volume and downsampling on death. The third is 'Audio Low Pass Filter', which allows adjusting the cutoff frequency and resonance. The fourth is 'Audio Echo Filter', which controls delay, decay, and mix levels.

Component	Property	Value
Audio Sync (Script)	Script	AudioSync
	Fade In Time	2
	Fade Out Time	1
	Fade On Mutes	<input checked="" type="checkbox"/>
	Override Start Pitch	-1
	Add To Audio Layer	-1
	Fade In On Play	<input type="checkbox"/>
	Dont Reset Volume On Umute	<input type="checkbox"/>
	Dont Stop On Mirror Death	<input type="checkbox"/>
	Fade With Filter	<input checked="" type="checkbox"/>
	Low Pass Filter	# airborne (Audio Low Pass Filter)
	High Pass Filter	None (Audio High Pass Filter)
	Freq Filter Start	10
Freq Filter End	5000	
Filter Curve Power	2	
Audio Death Down Sample (Script)	Script	AudioDeathDownSample
	Beat Layer	2
	Death Volume Scale	1
	Unmute On Death	<input type="checkbox"/>
	Down Sample Scale	1
Audio Low Pass Filter	Cutoff Frequency	5000
	Lowpass Resonance Q	1
Audio Echo Filter	Delay	643
	Decay Ratio	0.4
	Wet Mix	0.4
	Dry Mix	0.6

Example Audio Loop Components

AudioLowPassFilter: built-in Unity LPF

The screenshot displays the Unity Inspector for an audio loop component, showing four sub-panels:

- Audio Sync (Script)**
 - Script: AudioSync
 - Fade In Time: 2
 - Fade Out Time: 1
 - Fade On Mutes:
 - Override Start Pitch: -1
 - Add To Audio Layer: -1
 - Fade In On Play:
 - Dont Reset Volume On Umute:
 - Dont Stop On Mirror Death:
 - Fade With Filter:
 - Low Pass Filter: #airborne (Audio Low Pass Filter)
 - High Pass Filter: None (Audio High Pass Filter)
 - Freq Filter Start: 10
 - Freq Filter End: 5000
 - Filter Curve Power: 2
- Audio Death Down Sample (Script)**
 - Script: AudioDeathDownSample
 - Beat Layer: 2
 - Death Volume Scale: 1
 - Unmute On Death:
 - Down Sample Scale: 1
 - Downsample: 1
- Audio Low Pass Filter**
 - Cutoff Frequency: 5000
 - Lowpass Resonance Q: 1
- Audio Echo Filter**
 - Delay: 643
 - Decay Ratio: 0.4
 - Wet Mix: 0.4
 - Dry Mix: 0.6

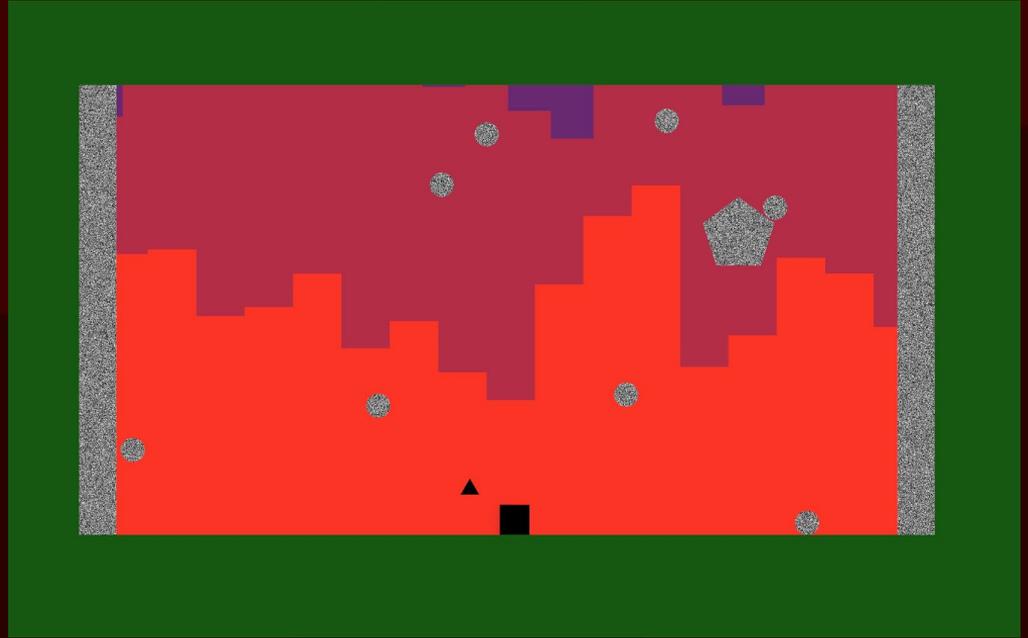
Example Audio Loop Components

AudioEchoFilter: built-in Unity delay

The screenshot displays the Unity Inspector for four audio components. The 'Audio Sync (Script)' component is expanded, showing various settings for audio synchronization. The 'Audio Death Down Sample (Script)' component is also expanded, showing settings for volume and scaling on death. The 'Audio Low Pass Filter' component is expanded, showing a cutoff frequency slider set to 5000 and a lowpass resonance Q of 1. The 'Audio Echo Filter' component is expanded, showing settings for delay, decay ratio, wet mix, and dry mix.

Component	Property	Value
Audio Sync (Script)	Script	AudioSync
	Fade In Time	2
	Fade Out Time	1
	Fade On Mutes	<input checked="" type="checkbox"/>
	Override Start Pitch	-1
	Add To Audio Layer	-1
	Fade In On Play	<input type="checkbox"/>
	Dont Reset Volume On Umute	<input type="checkbox"/>
	Dont Stop On Mirror Death	<input type="checkbox"/>
	Fade With Filter	<input checked="" type="checkbox"/>
	Low Pass Filter	# airborne (Audio Low Pass Filter)
	High Pass Filter	None (Audio High Pass Filter)
	Freq Filter Start	10
	Freq Filter End	5000
Filter Curve Power	2	
Audio Death Down Sample (Script)	Script	AudioDeathDownSample
	Beat Layer	2
	Death Volume Scale	1
	Unmute On Death	<input type="checkbox"/>
	Down Sample Scale	1
	Downsample	1
Audio Low Pass Filter	Cutoff Frequency	5000
	Lowpass Resonance Q	1
Audio Echo Filter	Delay	643
	Decay Ratio	0.4
	Wet Mix	0.4
	Dry Mix	0.6

Level 4 Boss



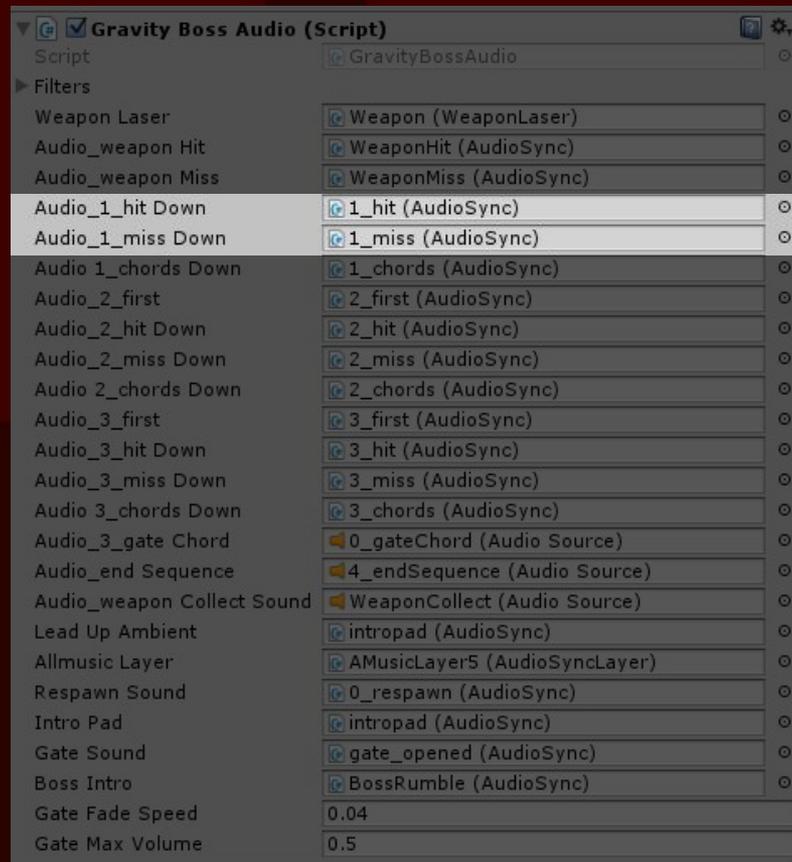
Level 4 Boss

- Over 20 loops running simultaneously



Level 4 Boss

- Separate loops for hit and miss
- Muted / unmuted when hit or miss is determined



Level 4 Boss

- Each stage has its own set of loops



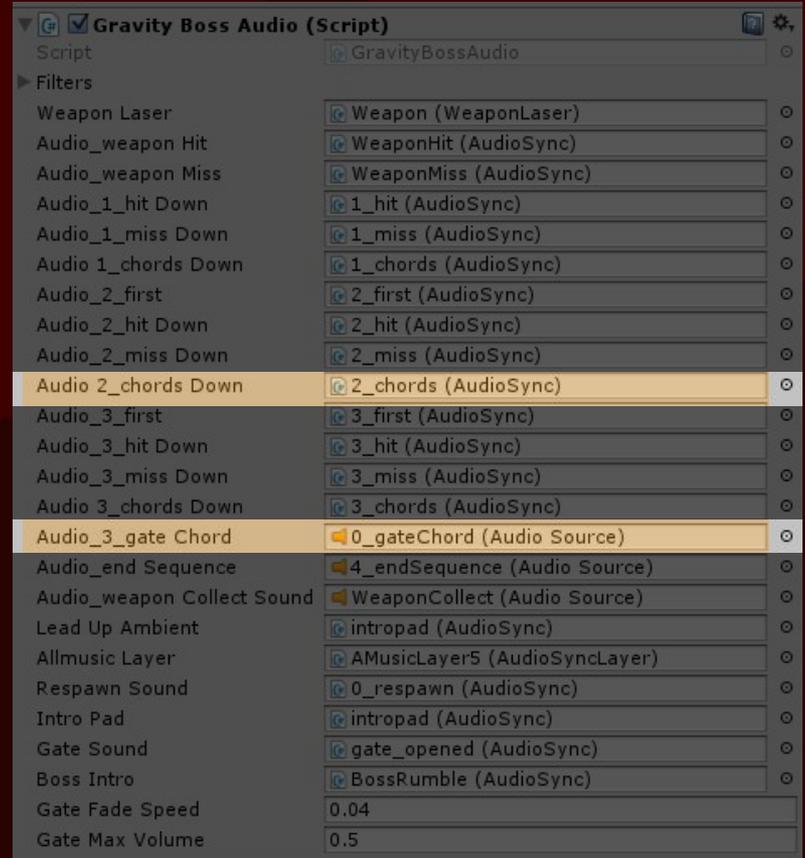
Level 4 Boss

- Each stage is in a different key
- Loops have long Ableton Reverb tails
- Reverb tails and key change requires special transitional first loop



Level 4 Boss

- House chords are faded in using filter between hits to create tension
- Final chord is faded in, anticipating end sequence



140 demo

level 4 boss

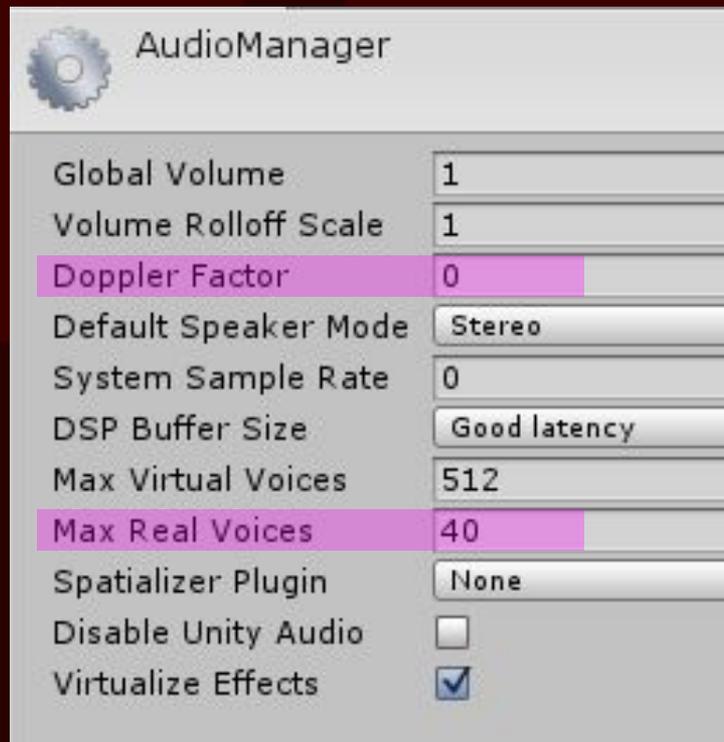


Audio Settings

Disable Doppler effect to avoid drifting loops!

To retain sample accuracy, set Max Real Voices to the max number of loops.

140 uses max 40 voices.



140 Music Production

The image displays a professional digital audio workstation (DAW) interface, likely Ableton Live, used for music production. The main workspace is a piano roll where various musical elements are arranged across 32 tracks. The tracks are organized into several sections:

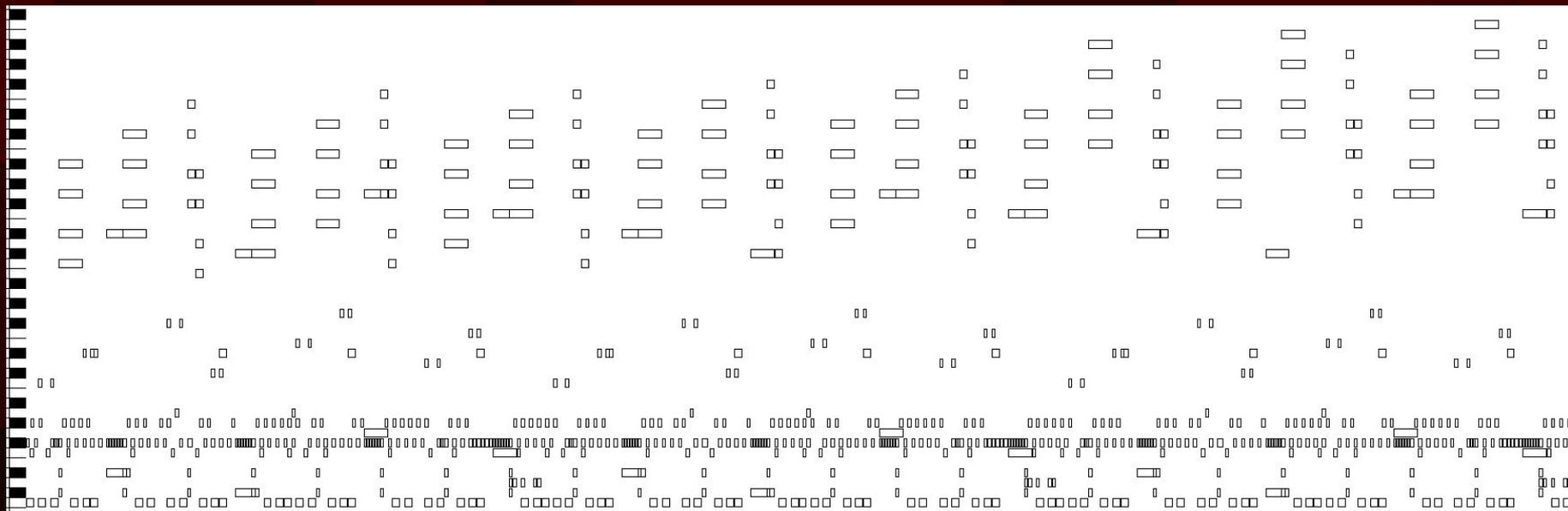
- MUSIC:** Includes tracks for the main mix (MUSIC), individual layers (H-layer0 to H-layer6), stems (H-layer1-key, H-layer1-beat, H-layer2-key, H-layer2-beat, H-layer3-beat, H-layer3-beat2, H-layer4-beat, H-layer4-beat_up, H-layer4-beat_dn, H-layer4-beat_up, H-layer4-beat_dn, H-layer4-beat_up, H-layer4-beat_dn), and variations (H-variation1, H-layer5-key).
- SOUND EFFECTS:** Includes tracks for key_pickup and h_h_c.

The piano roll shows a complex arrangement of notes, including a bass line (BASS) and various rhythmic patterns. The interface includes a transport bar at the top with playback controls and a tempo of 140.00. The bottom of the screen features several audio processing plugins for the selected track (H-layer R):

- Filter:** A low-pass filter with a frequency of 22.4 Hz and a resonance of 64%.
- Waveshaper:** A non-linear processor with a drive of 0.27 and a depth of 0.08.
- Saturator:** A saturation plugin with a drive of 6.66 dB and a depth of 100%.
- Dynamic Tube:** A compressor with a threshold of 0.00 dB and a ratio of 1.00.
- EQ Eight:** A multi-band equalizer with a frequency of 86.6 Hz and a gain of 0.00 dB.

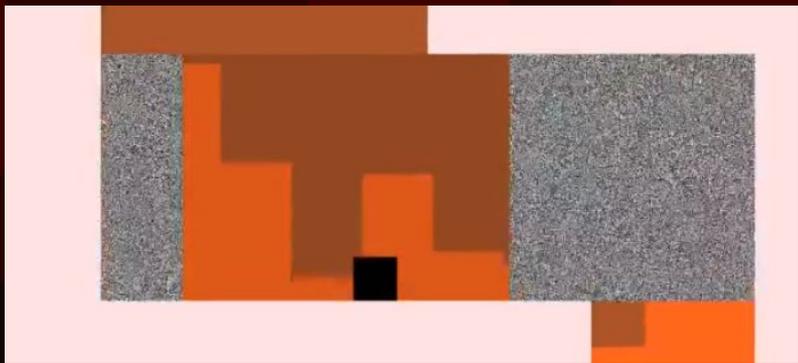
The Puzzle of Music

Making music for a *music game* can be like solving a puzzle



KillBlocks and Toggler

Two example game mechanics

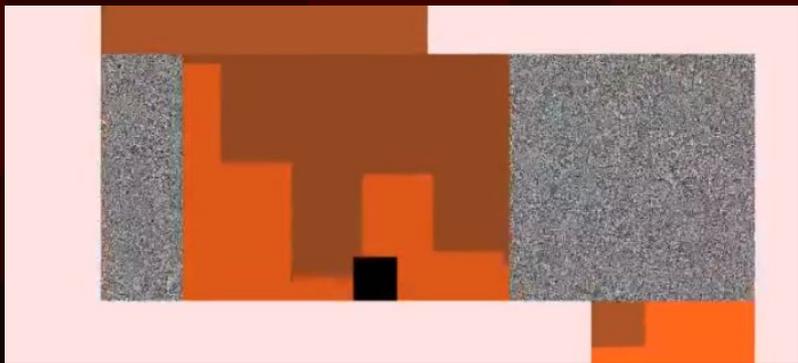


KillBlock

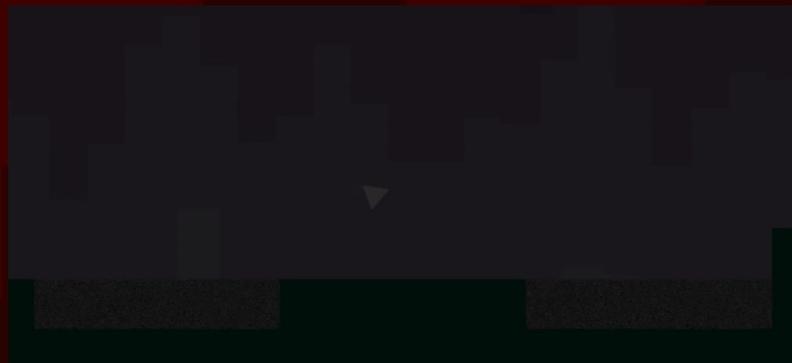


Toggler

KillBlocks

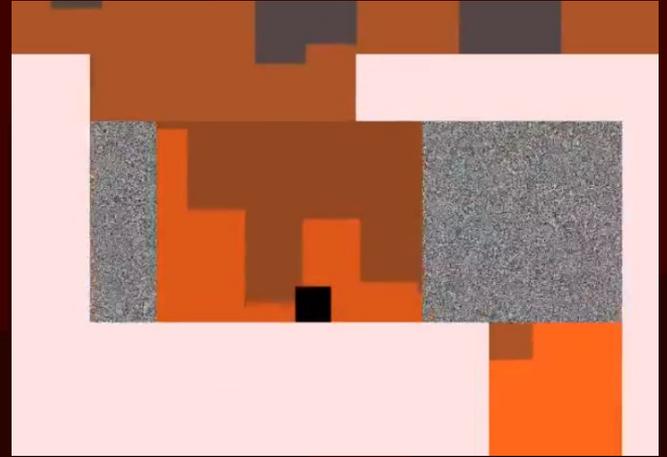


KillBlock



KillBlock Rhythm Pattern

- Communicates to player exactly when certain game areas are either safe or lethal
- Must correspond exactly to game logic timing



time 

> > > > x

> MOVE blocks right

x TOGGLE all blocks

KillBlock Sounds

- > MOVE sound is a 'Landlord stab'
- x TOGGLE sound is a 808 cowbell



KillBlock Rhythm Pattern

- Our KillBlock rhythm is exactly 2 bars



time (bars) 

1 . . . : . . . 2 . . . : . . .

> > > > x

KillBlock Harmony

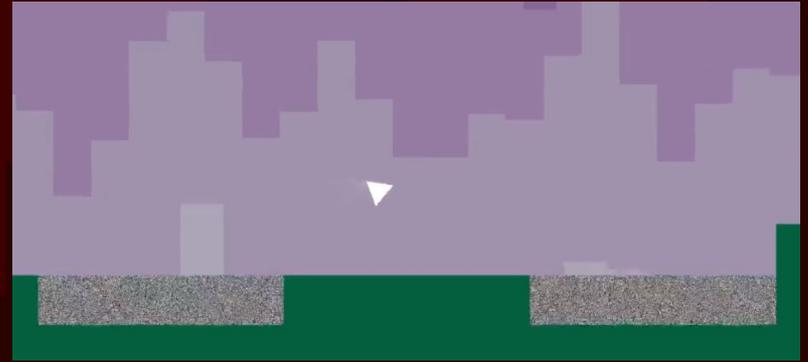
- The result can be heard in the **KillBlock loop** 
- 2-bar rhythmic loop
- 8-bar harmonic loop

bars

1 . : . 2 . : . 3 . : . 4 . : . 5 . : . 6 . : . 7 . : . 8 . : .
Cm D#m F#m C#m Em Gm Dm Fm



KillBlocks and Toggles



Toggler

Toggler Sound

The toggler loop alternates between two different Operator patches

State A Sound

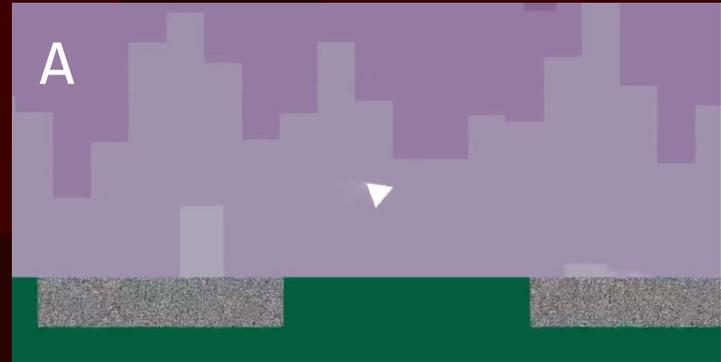
The screenshot shows the State A Operator patch interface. The left panel contains four oscillators with the following settings:

Oscillator	Coarse	Fine	Fixed	Level
1	-3	0	<input type="checkbox"/>	-12 dB
2	2	0	<input type="checkbox"/>	-14 dB
3	1	0	<input type="checkbox"/>	-7.6 dB
4	1	0	<input type="checkbox"/>	-12 dB

The Envelope section shows: Attack 0.00 ms, Decay 1.77 s, Release 1.37 s, Time<Vel 0%, Wave Sin.

The Oscillator section shows: Wave Sin, Feedback 0%, Repeat Off, Loop None, Key 0%, Phase 0%, Osc<Vel 0%.

The right panel shows LFO settings: Rate 111.88, Amount 32%, Filter 12 24 Clean, Freq 18.5 kHz, Res 20%, Pitch Env 0.0%, Spread 0%, Transpose 0 st, Time 29%, Tone 70%, Volume -2.3 dB.



State B Sound

The screenshot shows the State B Operator patch interface. The left panel contains four oscillators with the following settings:

Oscillator	Coarse	Fine	Fixed	Level
1	186 Hz	0.01	<input type="checkbox"/>	-58 dB
2	5	0	<input type="checkbox"/>	-7.0 dB
3	1	0	<input type="checkbox"/>	-8.1 dB
4	1	0	<input type="checkbox"/>	0.0 dB

The Envelope section shows: Attack 0.00 ms, Decay 600 ms, Release 6.42 s, Time<Vel 0%, Wave Sin.

The Oscillator section shows: Wave Sin, Feedback 0%, Repeat Off, Loop None, Key 0%, Phase 0%, Osc<Vel 0%.

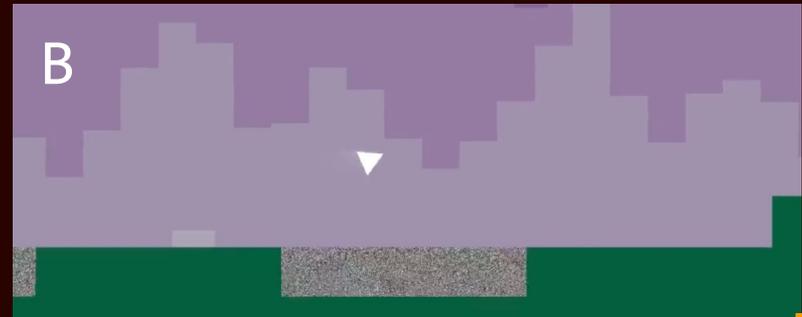
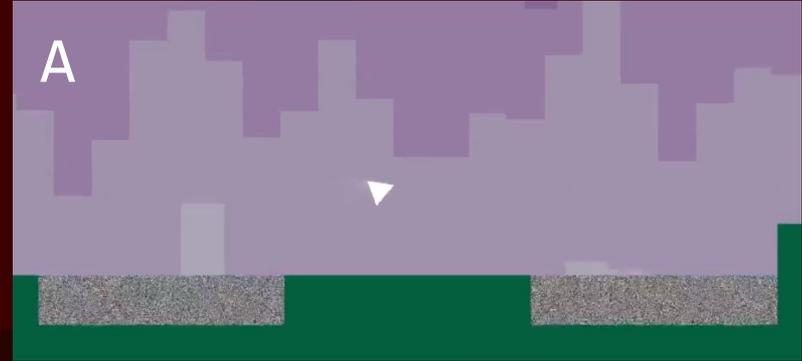
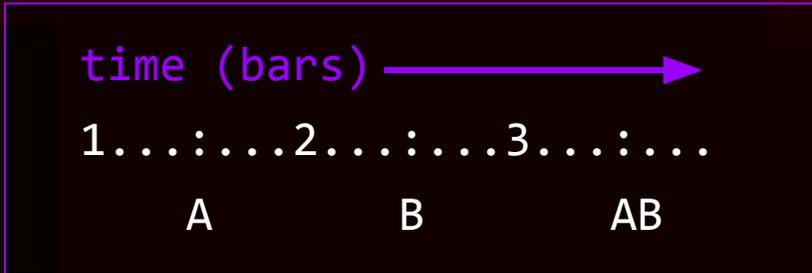
The right panel shows LFO settings: Rate 96.76, Amount 52%, Filter 12 24 OSR, Freq 55.3 Hz, Res 37%, Pitch Env 0.0%, Spread 0%, Transpose 0 st, Time 0%, Tone 70%, Volume -13 dB.



Toggler Rhythm Pattern

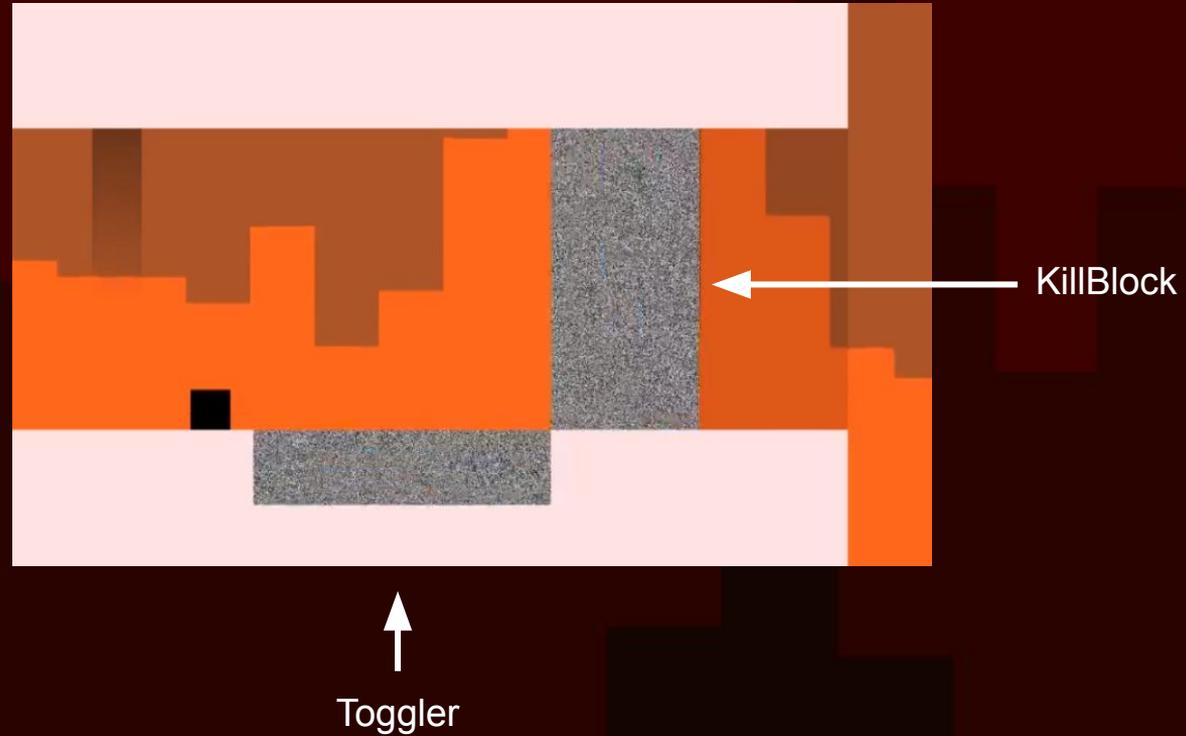
The toggler loop: 

- 3-bar rhythmic loop
- Game logic toggle floors between lethal and non-lethal
- Two states: A and B



Toggler and KillBlock

Both play at once in this jump puzzle!



Toggler and KillBlock Rhythms

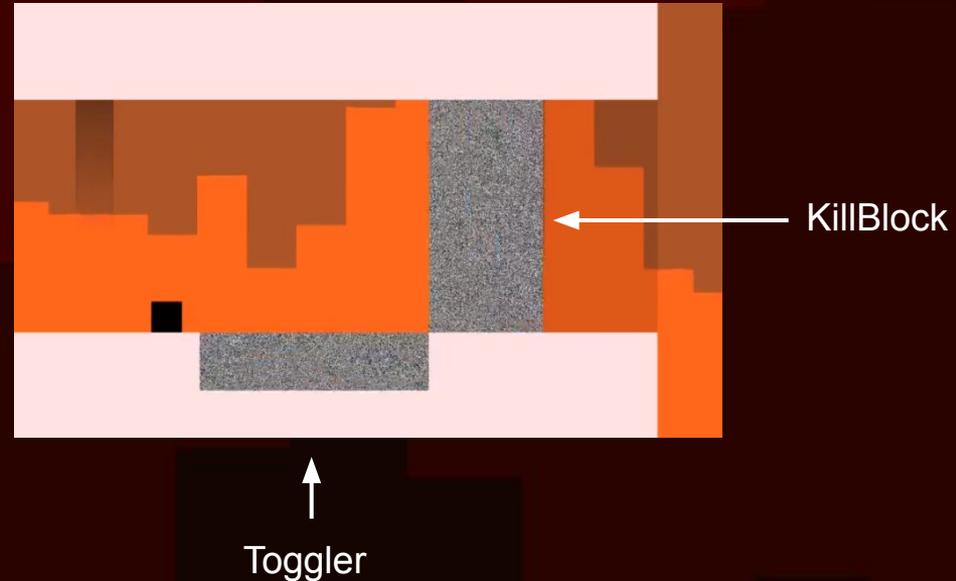
- KillBlock loop is 2 bars
- Toggler loop is 3 bars

KillBlock

```
1 . . . : . . . 2 . . . : . . .  
  > >      > > x
```

Toggler

```
1 . . . : . . . 2 . . . : . . . 3 . . . : . . .  
          A          B          A B
```



Toggler and KillBlock Looped

Loop simultaneously after $2 \times 3 = 6$ bars

KillBlock x 3

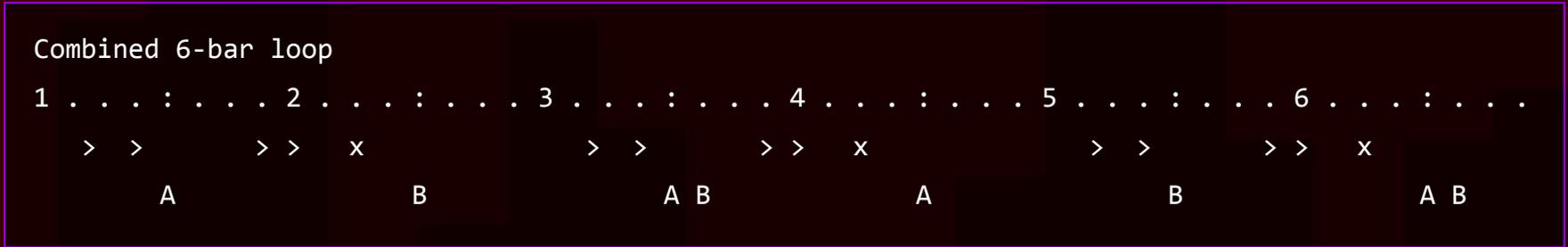
```
1 . . . : . . . 2 . . . : . . . 3 . . . : . . . 4 . . . : . . . 5 . . . : . . . 6 . . . : . . .  
  > >      > > x          | > >      > > x          | > >      > > x  
                    loop                    loop
```

Toggler x 2

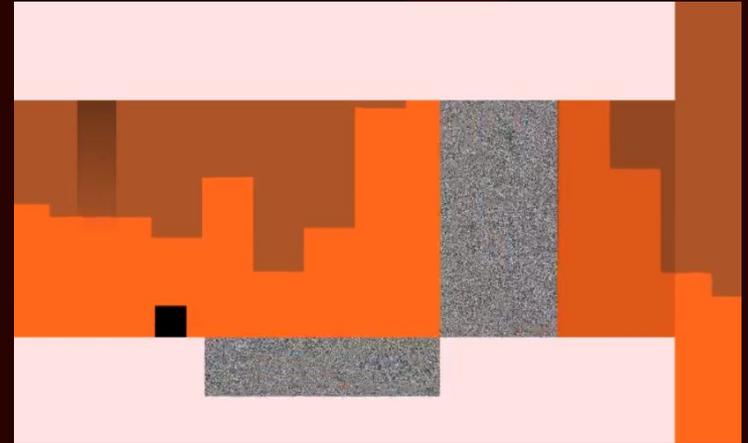
```
1 . . . : . . . 2 . . . : . . . 3 . . . : . . . 4 . . . : . . . 5 . . . : . . . 6 . . . : . . .  
          A          B          A B          |          A          B          A B  
                    loop
```

Toggler and KillBlock Combined

The combined 6-bar loop of **Toggler and KillBlock**: 



This pattern is what the player must grasp
to pass the jump puzzle



Toggler Harmony

- Toggler loop must be in harmony with KillBlock loop
- 8-bar harmonic loop

KillBlock:	Cm	D#m	F#m	C#m	Em	Gm	Dm	Fm
Toggler:	Cm7	D#m7	Bm11	C#m7	Em7	Cm11	Dm7	Fm7

Toggle Full Loop

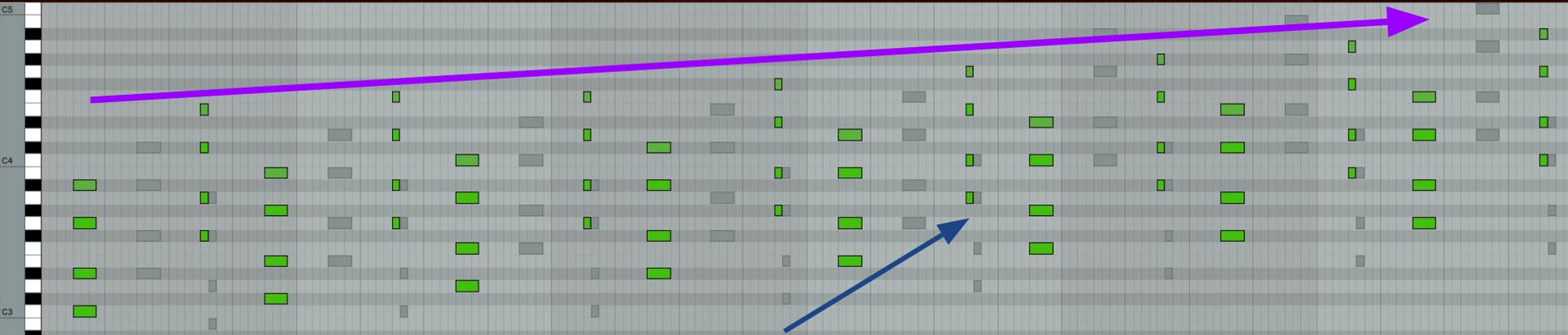
1	2	3	4	5	6	7	8	9	10	11	12			
A	B	AB	A	B	AB	A	B	AB	A	B	AB			
Cm7	D#m7	Bm11	C#m7	Em7	Cm11	Dm7	Fm7	Cm11	D#m7	F#m7	C#m11			
13	14	15	16	17	18	19	20	21	22	23	24			
A	B	AB	A	B	AB	A	B	AB	A	B	AB			
Em7	Gm7	Dm11	Fm7	Cm7	D#m11	F#m7	C#m7	Em11	Gm7	Dm7	Fm11			



Rising Pattern

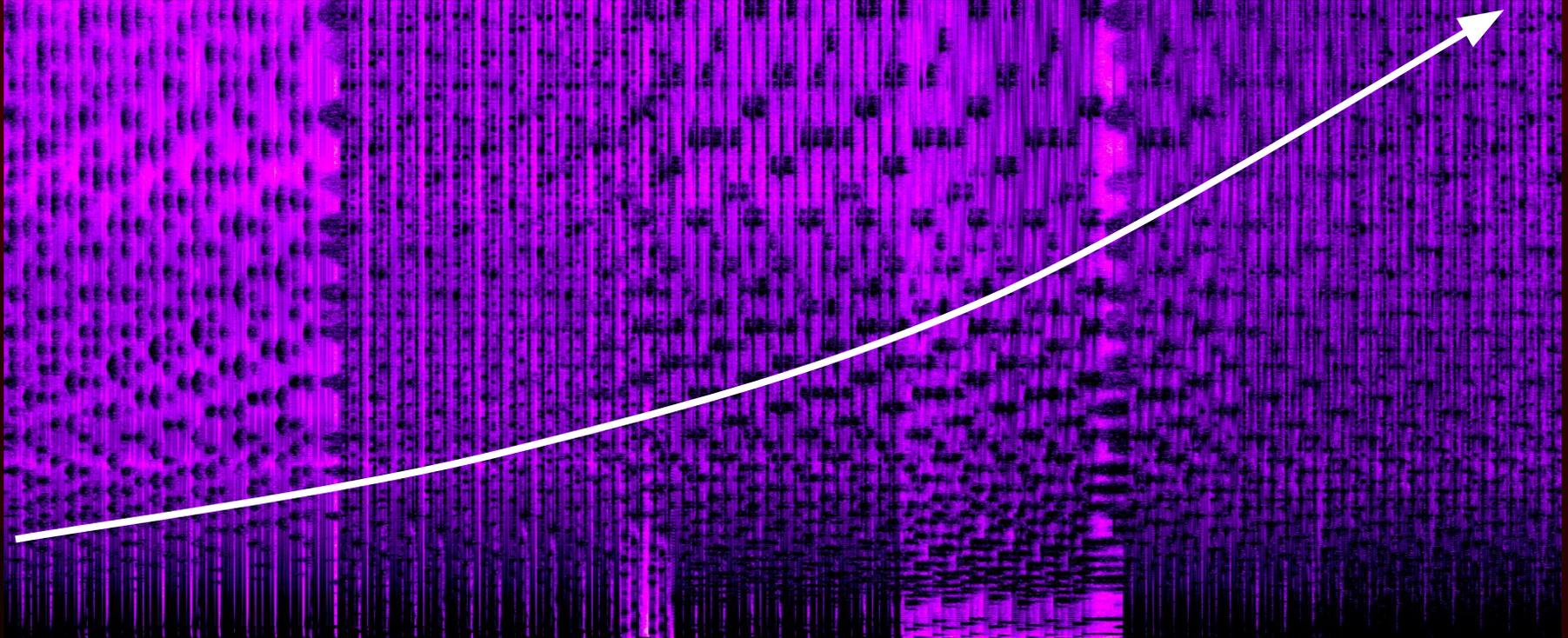
Level 4 is composed to emulate frequency continuously rising:

- Uses chord inversions to create 4-chord rising sequences
- Chord notes generally ascend over full 24-bar loop



Rising Pattern

Spectral analysis of soundtrack version shows rising frequency pattern



Fun Audio Tricks

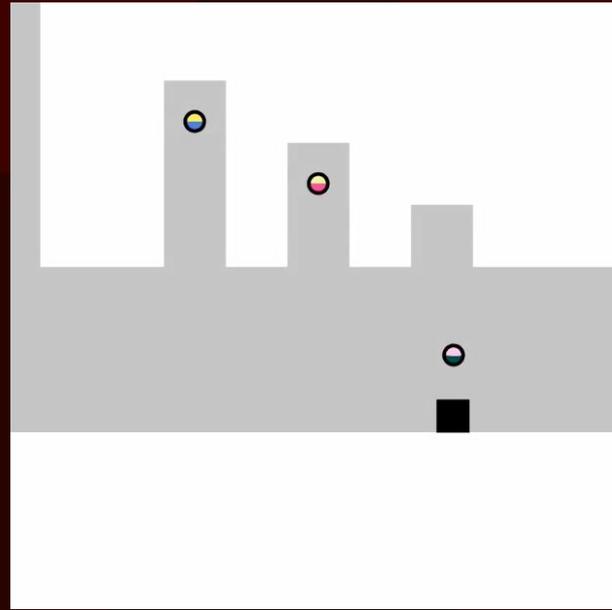
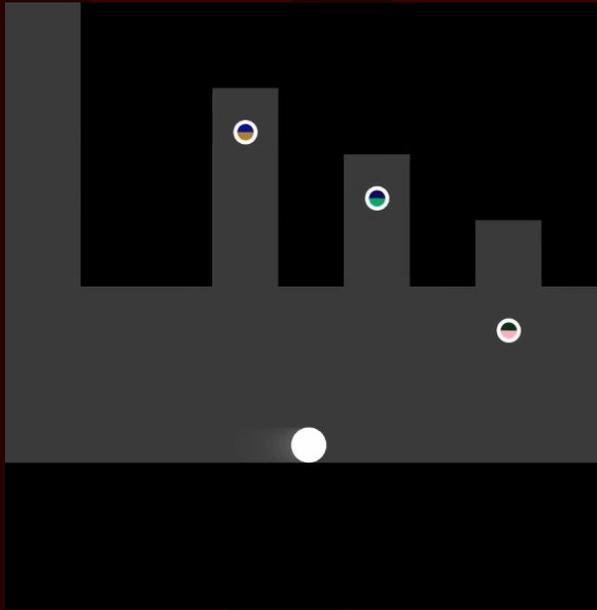
The background is a pixelated, low-resolution scene. The top half is a bright red sky. The bottom half is a brown ground. A purple path starts from the bottom left, goes right, then up, then right again, leading to a white door. There are some white and light purple pixels scattered around the door area.

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):

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

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.

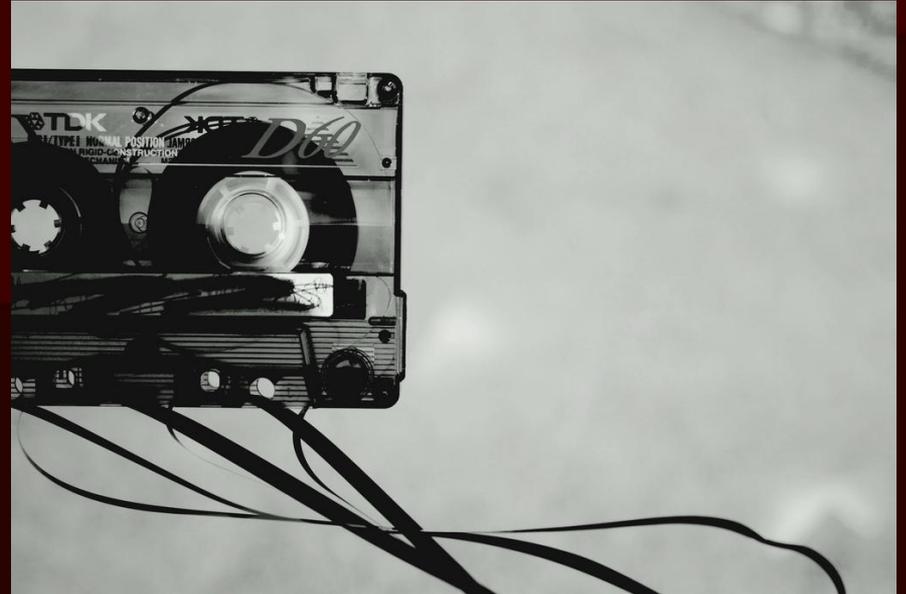


Image credit: Kristi Bogel

Cassette Tape Jam Code

The tape jam effect is achieved with:

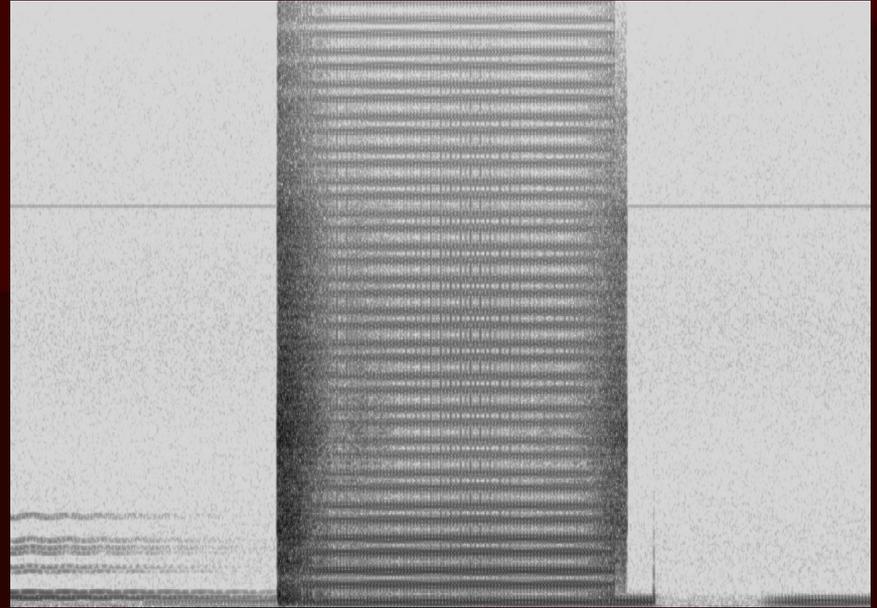
- applying strong vibrato to all music tracks,
- enveloping pitch towards 0 and volume towards silence.

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
```

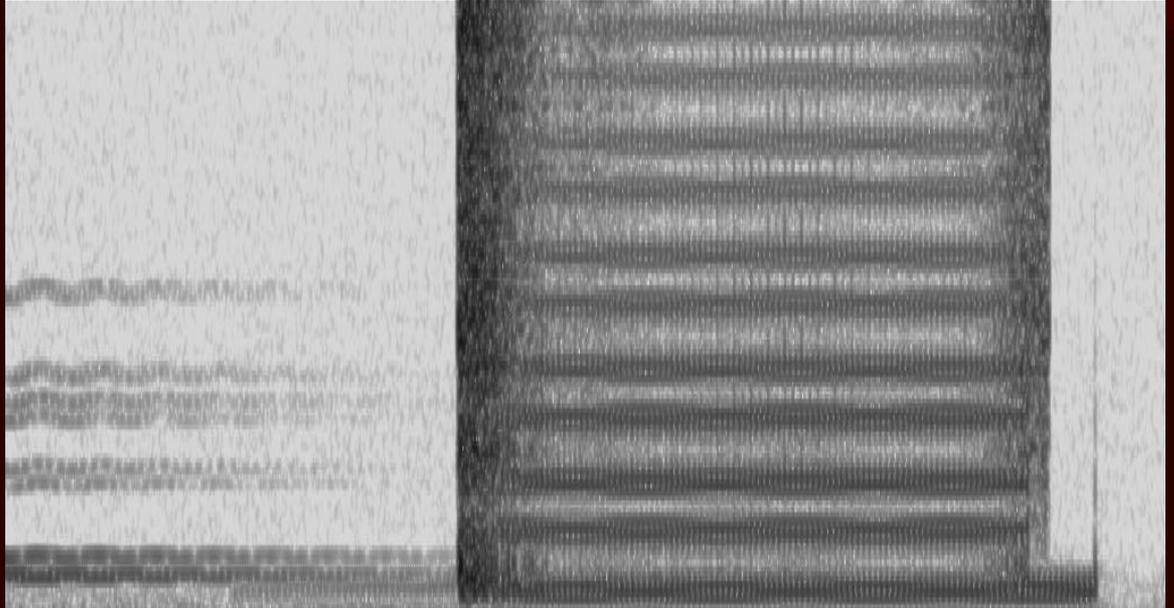
Downsampling

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.



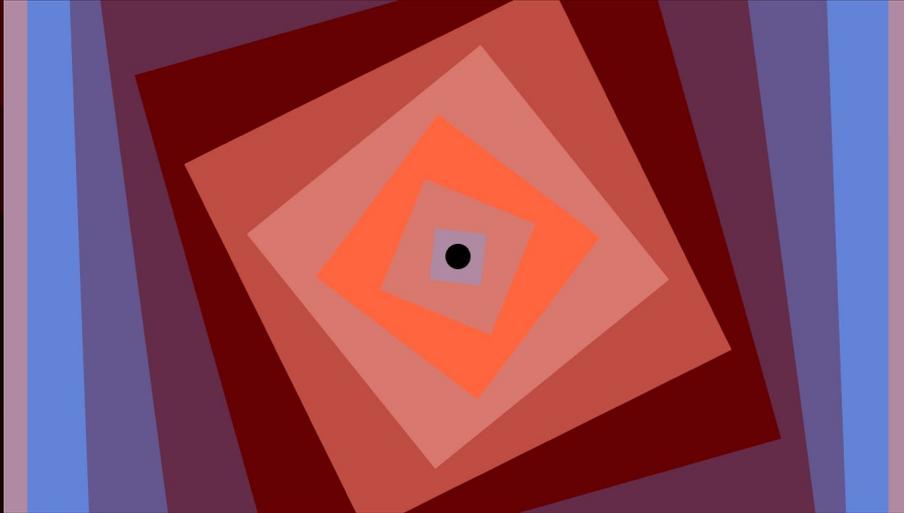
Downsampling Filter Code

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  
        }  
    }  
}
```

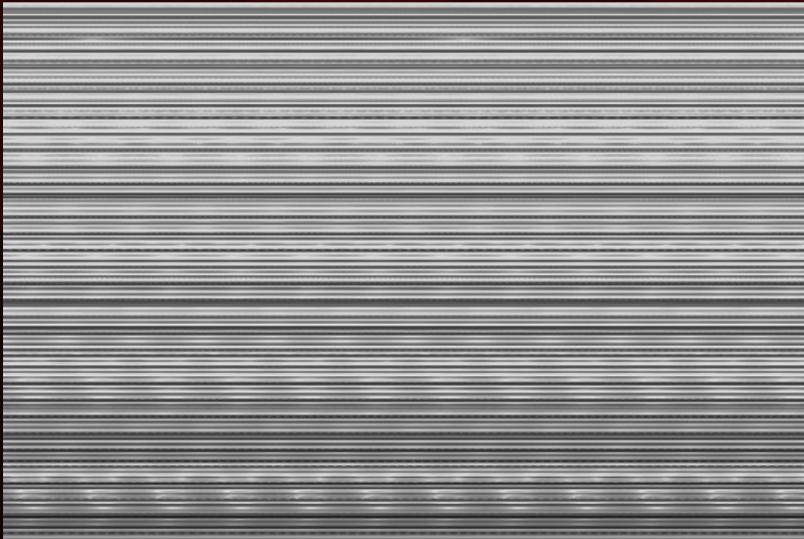
Fake crash

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.



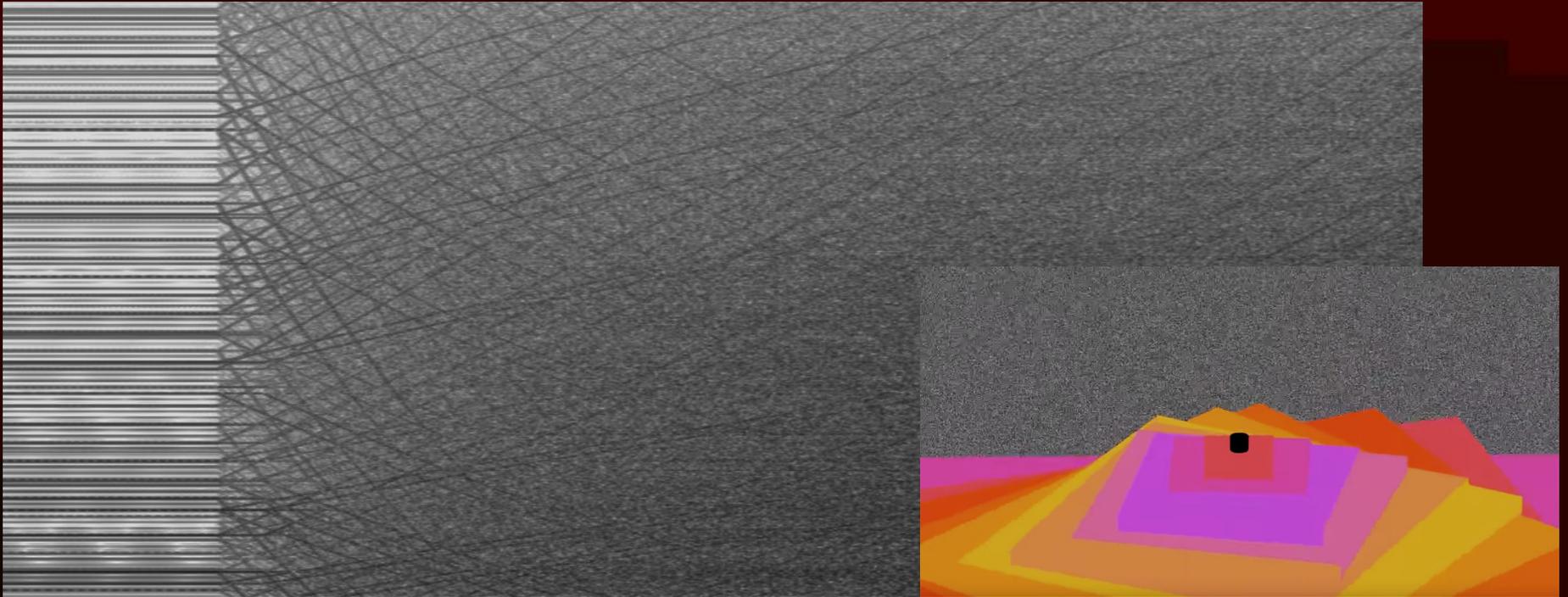
Fake crash

- 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.



Questions?



