



# Making Music through Computer Science Practices

Jared O'Leary  
BootUp PD



# How to reach the resources

- Direct link is in the chat
- [www.JaredOLEary.com](http://www.JaredOLEary.com)
  - Presentations
    - Making Music through Computer Science Practices



# Computer Science

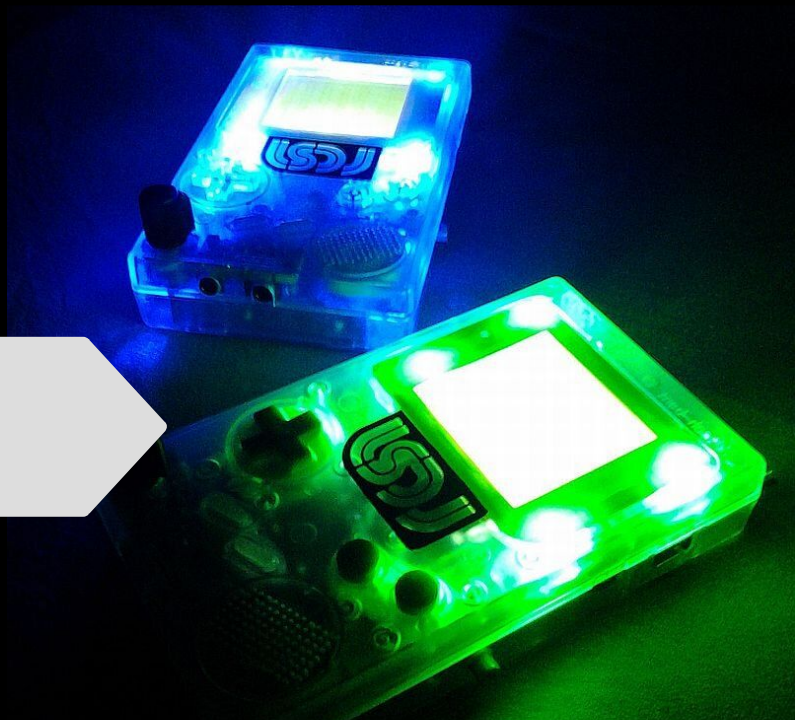
Presentation by Jared O'Leary and uses Creative Commons licensing Attribution-NonCommercial-ShareAlike (BY-NC-SA)

# Performance practices





# Maker practices



# Aesthetic mods

LEDs



# Functionality mods

## Backlighting



# Functionality mods

Prosound

Presentation by Jared O'Leary and uses Creative Commons  
licensing Attribution-NonCommercial-ShareAlike (BY-NC-SA)





# Functionality mods

## Prosound



# Functionality mods

## Clocking

Presentation by Jared O'Leary and uses Creative Commons  
licensing Attribution-NonCommercial-ShareAlike (BY-NC-SA)



# Functionality mods

## Circuit-bending



# Functionality mods

Circuit-bending



# Functionality mods

## Other mods

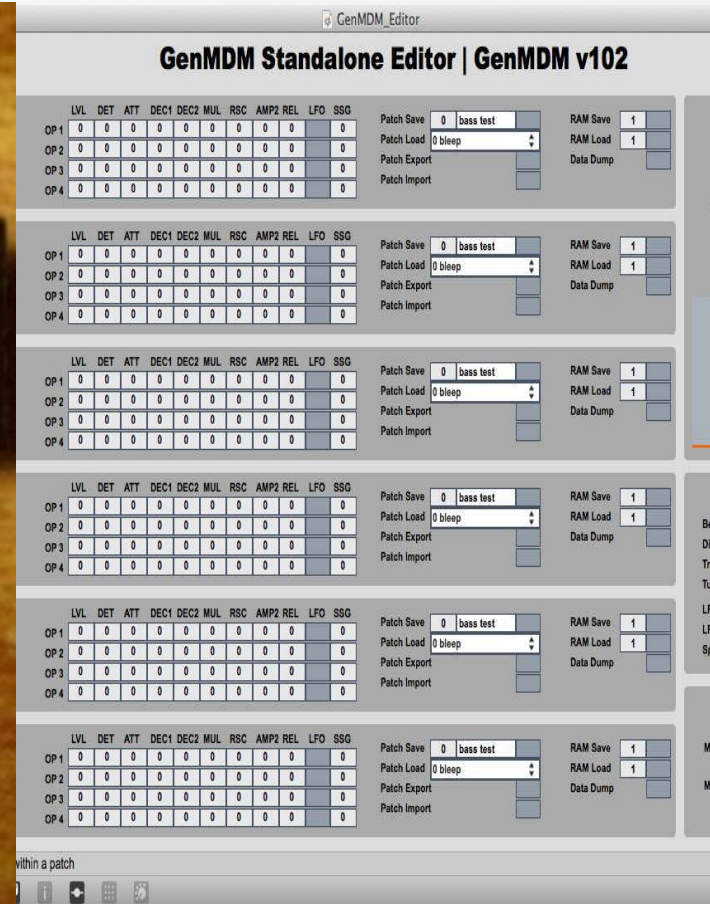


# Functionality mods

## Other mods



# Manufacturing or building new devices



# Coding practices

Here's how I'd do it, starting with the version at <https://github.com/trash80/Arduinoboy>;

In the file Mode.ino edit the function switchMode() as follows;

```
void switchMode()
{
  switch(memory[MEM_MODE])
  {
    case 0:
      modeLSDJSlaveSyncSetup();
      break;
    case 1:
      modeMidiGbSetup();
      break;
  }
}
```

And in the file Arduinoboy.ino, change the line;

```
#define NUMBER_OF_MODES 7 //Right now there are 7 modes, Might be more in the fut
```

```
#define NUMBER_OF_MODES 2
```

That should pretty much do what you want.



# Software development

Presentation by Jared O'Leary and uses Creative Commons  
licensing Attribution-NonCommercial-ShareAlike (BY-NC-SA)



# Software development

SONG	PU1	PU2	WAV	NOI	PU1
00	05	7F	04	03	
01	06	7F	05	03	
02	01	02	00	03	188
03	01	02	00	03	
04	01	02	00	03	1
05	01	02	00	03	2
06	07	08	09	0A	WC 6
07	20	08	09	0A	NC 5
08	21	29	09	0A	
09	0C	00	0E	13	
0A	11	10	12	0F	
0B	15	14	17	16	
0C	01	02	00	03	
0D	01	02	00	03	P
0E	22	08	09	0A	SCPIT
0F	23	08	09	0A	G

# Creating music with code

The image shows a code editor on the left with C++ code for a musical synthesizer. The code includes several loops for playing notes, each with a specific duration and pitch. The code is as follows:

```
live_loop :tick, :beat_duration => 0.5 do
  sample :mix, :kick, :velocity => 0.5
end

live_loop :drum, :beat_duration => 0.5 do
  sample :mix, :snare, :velocity => 0.5
end

live_loop :bass, :beat_duration => 0.5 do
  sample :mix, :bass, :velocity => 0.5
end

live_loop :melody, :beat_duration => 0.5 do
  sample :mix, :melody, :velocity => 0.5
end

live_loop :percussion, :beat_duration => 0.5 do
  sample :mix, :percussion, :velocity => 0.5
end

live_loop :fx, :beat_duration => 0.5 do
  sample :mix, :fx, :velocity => 0.5
end
```

The terminal window on the right shows the output of the code, displaying the name of the sample being played and the thread ID. The output is as follows:

```
sample "/usr/local/share/sounds/.../.../..." :kick :velocity => 0.5
sample "/usr/local/share/sounds/.../.../..." :snare :velocity => 0.5
sample "/usr/local/share/sounds/.../.../..." :bass :velocity => 0.5
sample "/usr/local/share/sounds/.../.../..." :melody :velocity => 0.5
sample "/usr/local/share/sounds/.../.../..." :percussion :velocity => 0.5
sample "/usr/local/share/sounds/.../.../..." :fx :velocity => 0.5
```

# Live coding

```
1 Report: Sound.Tidal.MIDI.Output
2 Report: Sound.Tidal.VoiceKeys
3
4 keyStreams ← keyProxy 1 6 keys [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
5
6 [bd,k2,k3,k4,k5,k6,k7,k8,k9,k10,k11,k12,k13,k14,k15,k16] ← sequence keyStreams
7
8 bpm (120/150)
9
10 k16 6 note "[44 66 64]M16"
11 |> kenvtff "0.5"
12 |> vofegint "0"
13 |> ifrate "0.01"
14 |> ifcuttoffint "0.5"
15 |> actum "0"
16 |> portamento "0.1"
17 |> sustain "1"
18 |> dur "0.15"
19 |> attack "0"
20 |> voice "0.5"
21
22 dl 0
23 stack [
24 sound "MIDI2P16",
25 sound "s:2 [- k:3] - k:2 k:3 - k:1 -" |> steps "0.5",
26 sound "~ = [- k]" ,
27 sound "~ = [- 1001 - -]" |> speed "0.5",
28 foldEvery [0,4] (0.25 0) 0
29 almspread (density) [1, 1.5, 1, 1, 4] 0 alms + 0 (strata 10 0 sound "[frs]**4") |> gain "0.5" |> speed "[1 0.5, [1.5 2]/2]" |> delay "0.4"
30 |> delaytime "0.5" |> delayfeedback "0.5"
```



# Hot Cross Buns (with Sonic Pi)

Presentation by Jared O'Leary and uses Creative Commons licensing Attribution-NonCommercial-ShareAlike (BY-NC-SA)



# Setting our tempo

1. `use_bpm 144`



# Adding our notes

1. use\_bpm 144
- 2.
3. play :e
4. play :d
5. play :c



# Separating our notes

1. use\_bpm 144
- 2.
3. play :e
4. sleep 2
5. play :d
6. sleep 2
7. play :c
8. sleep 4





# Defining a function

1. use\_bpm 144
- 2.
3. **define :buns do**
4.     play :e
5.     sleep 2
6.     play :d
7.     sleep 2
8.     play :c
9.     sleep 4
10. **end**




Congratulations,  
you recreated Cage's 4' 33"!



# Calling our function

```
3.   define :buns do
4.     play :e
5.     sleep 2
6.     play :d
7.     sleep 2
8.     play :c
9.     sleep 4
10.  end
11.
12.  buns()
13.  buns()
```



# Starting our next phrase

- 12. buns()
- 13. buns()
- 14.
- 15. play :c
- 16. sleep 1



# Using repeats

```
12. buns()  
13. buns()  
14. 4.times do  
15.   play :c  
16.   sleep 1  
17. end
```



# Using repeats

```
12. buns()  
13. buns()  
14. 4.times do  
15.   play :c  
16.   sleep 1  
17. end  
18. 4.times do  
19.   play :d  
20.   sleep 1  
21. end
```



# Completing our song

```
12.  buns()  
13.  buns()  
14.  4.times do  
15.    play :c  
16.    sleep 1  
17.  end  
18.  4.times do  
19.    play :d  
20.    sleep 1  
21.  end  
22.  buns()
```



# Changing our synth

1. `use_bpm 144`
2. `use_synth :tri`
- 3.
4. `define :buns do`
5. `play :e`
6. `sleep 2`
7. `play :d`
8. `sleep 2`
9. `play :c`
10. `sleep 4`
11. `end`





# Shaping our notes

1. use\_bpm 144
2. use\_synth :tri
- 3.
4. define :buns do
5.   play :e, release: 2
6.   sleep 2
7.   play :d, release: 2
8.   sleep 2
9.   play :c, release: 4
10.   sleep 4
11. end

# Adding effects

13.

14. `with_fx :echo do`

15. `buns()`

16. `buns()`

.....

24. `buns()`

25. `end`



# In a different buffer

1. `use_bpm 144`



# Creating our loop

1. use\_bpm 144
- 2.
3. live\_loop :perc do
4. end



# Metal

1. use\_bpm 144
- 2.
3. live\_loop :perc do
4.     sample :bd\_haus
5.     sleep 0.25
6. end



# EDM

1. use\_bpm 144
- 2.
3. live\_loop :perc do
4. sample :bd\_haus if (spread 1, 4).tick
5. sleep 0.25
6. end



# Adding in another rhythm

1. use\_bpm 144
- 2.
3. live\_loop :perc do
4.   sample :bd\_haus if (spread 1, 4).tick
5.   sample :elec\_bong if (spread 3, 8).look
6.   sleep 0.25
7. end



## ...and another

1. use\_bpm 144
- 2.
3. live\_loop :perc do
4.   sample :bd\_haus if (spread 1, 4).tick
5.   sample :elec\_bong if (spread 3, 8).look
6.   sample :perc\_snap if (spread 3, 4).look
7.   sleep 0.25
8. end





# Adjusting our amplitude

1. `use_bpm 144`
- 2.
3. `live_loop :perc do`
4. `sample :bd_haus if (spread 1, 4).tick`
5. `sample :elec_bong if (spread 3, 8).look`
6. `sample :perc_snap, amp: 0.3 if (spread 3, 4).look`
7. `sleep 0.25`
8. `end`

# Back in our original buffer

```
12.  
13.  define :song do  
14.    with_fx :echo do  
15.      buns()  
16.      buns()  
    .....  
25.      buns()  
26.    end  
27.  end
```



# Press Run for Cage's encore

(there is a purpose for this)



# Hip cross buns

1. use\_bpm 144
- 2.
3. live\_loop :perc do
4.   sample :bd\_haus if (spread 1, 4).tick
5.   sample :elec\_bong if (spread 3, 8).look
6.   sample :perc\_snap, amp: 0.3 if (spread 3, 4).look
7.   sleep 0.25
8. end
- 9.
10. **song()**

# Exploring Sonic Pi

- Sonic Pi's built-in help
  - Tutorials
  - Examples
  - Synths
  - Fx
  - Samples
  - Lang(uage)
- [www.JaredOLeary.com/sonic-pi](http://www.JaredOLeary.com/sonic-pi)

Let's Share What  
We Created!