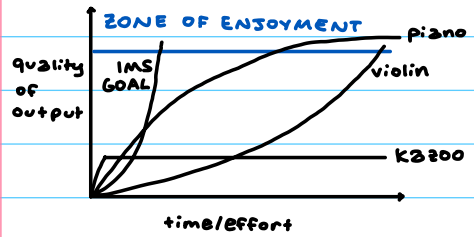
21m.385 - ERAN EGOZY (CTO of Harmonix, Founder of Guitar Hero)

2/2 **INSTRUMENT EFFICIENCY:**



consideration: if instrument is simpler (easier), you can reach zone of Enjoyment faster

- aesthetics

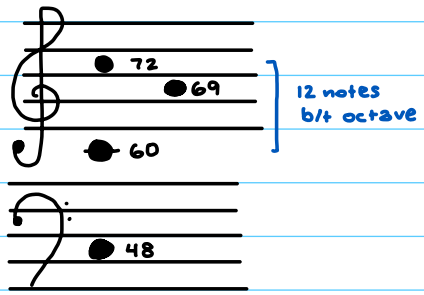
EX: Spotify:

interactive: pressed play xD

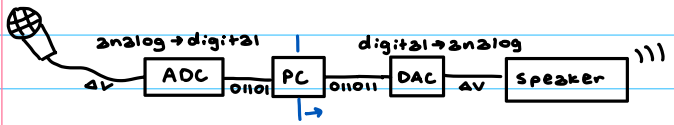
music: came out of phone speakers

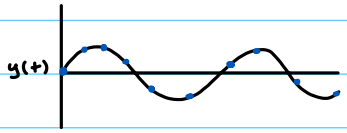
1 button (Spotify)		88 buttons (piano)
low	amount of interaction	high
low	fidelity	high
large	per button response	small/detailed
many things	specificity	high
low	mechanical abstraction	high
low	active agency	high
high	ease	low

MIDI:



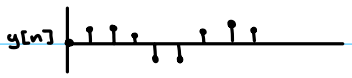
2/4 **MICROPHONE:** transducer; takes a signal from air vibrations



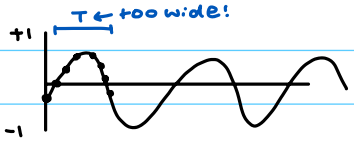


T = sampling period

$F_s = \frac{1}{T}$ Hz sampling frequency

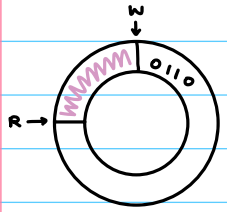


y[n] = y(nT)

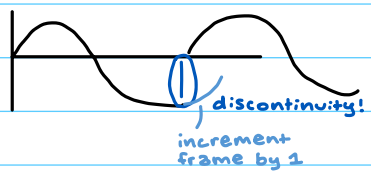


NYQUEST RATE: 2x Fmax

F_s = 44,100 Hz (for purpose of audio)



- cat & mouse (slight lag)
- write ptr writes #s into buffers
- read ptr reads #s



doesn't rly matter

$y(t) = A\sin(\omega t + \phi)$ $\omega = 2\pi f$

$y[n] = A\sin(2\pi f n t) = A\sin(2\pi f n / Fs)$

IN CODE:

freq: # ÷2 → lower octave, *2 → higher octave

sampling rate: 44100

gain: multiplier (sound) ÷2 → 2x softer, *2 → 2x louder

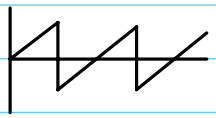
→ numpy faster than loops! use vector manipulation

PITCHES: example

$f_0 = 440$ $f_{12} = 880 = 2 * f_0 = 2^{1/2} \cdot 2^{1/2} f_0$

$f_6 = 2^{1/2} f_0$

$f_1 = 2^{1/12} f_0$



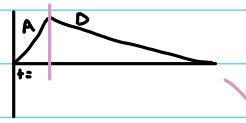
mode 1 f_0 $a_1\sin(2\pi f_0)$ +

mode 2 $2f_0$ $a_2\sin(2\pi 2f_0)$: +

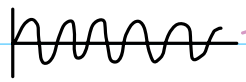
mode 3 $3f_0$ $a_n\sin(2\pi n f_0)$

n

b/c all happening @ same time, these all get added together.

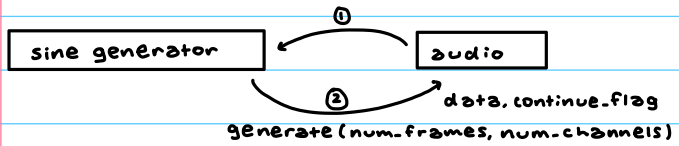


Envelope

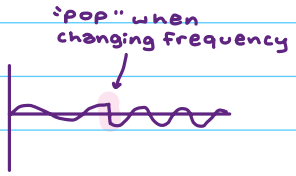


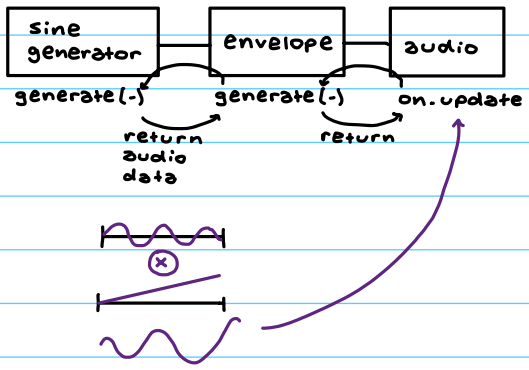
MULTIPLY pt by pt to get a note

ALIASING: b/c trying to push frequencies above Nyquist, frequencies will actually go lower

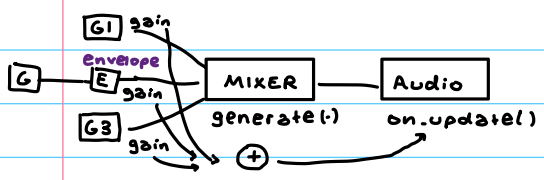


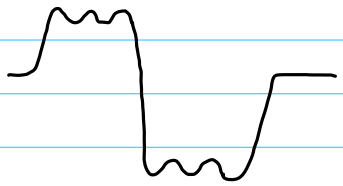
- if audio is > 16 ms load, it'll crackle (not long enough)
- AudioWriter writes —# samples into .wav file
  - can zoom in on waves in Audacity & examine →

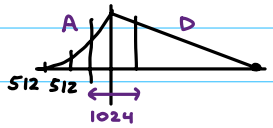




MIXER / ENVELOPE:







questions?

P : 2222

Office hours

Buddy

2/9/2026

class MainWidget(BaseWidget):

def __init__(self):

Extension (everyone gets 2 free extensions, 6 Psets total)

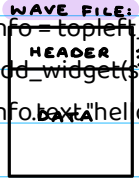
super(MainWidget, self).__init__()

2/17 WAVE FILE:

self.info = topleft_label()

self.add_widget(self.info)

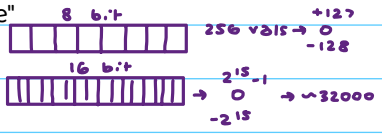
self.info.text "hello there everyone"

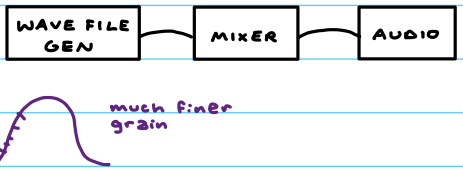


sample_width < 8 bits (256 vals) / 16 bits

frame_rate - typically 44100

channels - mono vs. stereo (1 or 2)



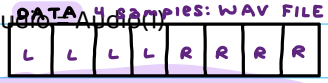


self.cnt = 0

self.audio = Audio(1)

self.sine_gen.set_freq(0

DATA: 4 samples: WAV FILE



← this one may be easier to extract

← this one is correct! (stream w/o knowing set length)

one sample, but part of the same frame

def on_key_down(self, keycode, modifiers):

if keycode[1] == 'q':

print("q was pressed")

BROADCASTING:

self.info.text = "q was pressed" # will appear on console

a[3:7:2] = ['a', 'b'] → a[3] becomes 'a' & a[5] = 'b' ; modifies a

can set frequencies depending on key binds

STEREO: if num.channels == 2:

def on_update(self):

self.

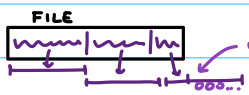
stereo = np.empty(num.frames * num.channels)

stereo[0::2] = output ← evens

stereo[1::2] = output ← odds

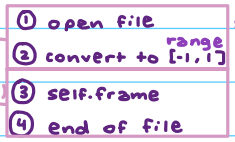
output = stereo

run(MainWidget())



won't fill whole buffer (request larger than amt) pad w/ zeros

self.wave = wave.open(filepath) ← handy python library



output.astype(np.float32)

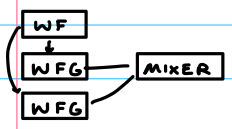
output /= 1/32768.0

WAVEFILE ← get frame (start, # of frames) ← WAVE GEN

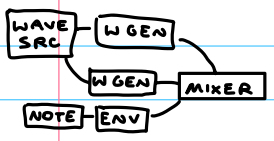
bad! added twice
& will play @ same time
& will play 2x as fast

PROTECTION:

```
if gen not in self.generators:
    self.generators.append(gen)
```



better! can now play & layer
w/o changing tempo



MAPPING
keys to src of data

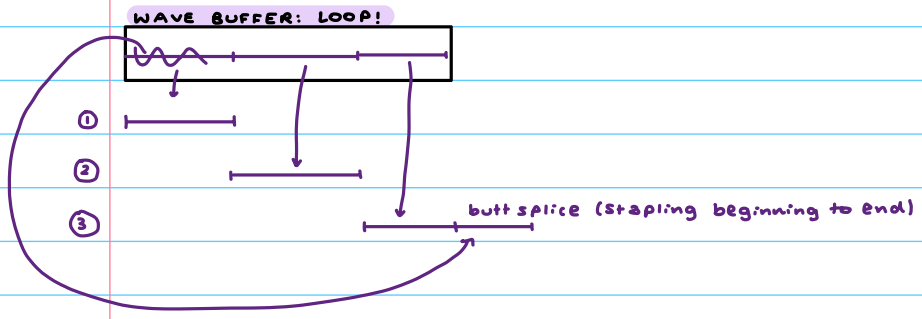
lookup(key, ["p", "a", "s"], [, ,]) ← snippets of audio
src1, 2, 3

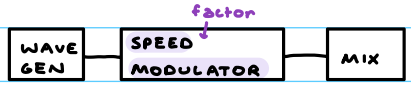
WAVE BUFFER

SONIC VISUALIZER: application on mac/windows

- layer → new regions layer (has start & end) → can get sections/snippets of audio
- can save sections as .txt & put into .py file

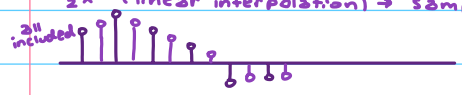
WAVE BUFFER: LOOP!





2x (toss every other) → compressed $\frac{\text{samples}}{2}$

$\frac{1}{2}$x (linear interpolation) → samples × 2



1.5x faster:

np.interp ← done for you

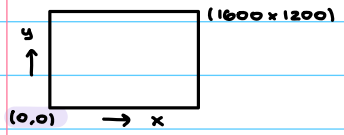
VOICE (can also take mic input)

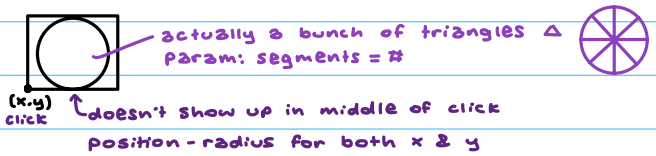
SEMITONES convert to speed: $\text{SPEED} = 2^{\left(\frac{\text{semitone\_delta}}{12}\right)}$

2/23 KIVY:

- init
- loop
  - input ← events / polling ← on key down
  - update model ← state ← prev ← passage of time
  - draw
- 60 fps → screen rate
- 44100 Hz → audio rate

DEMOS:
- tracking mouse position
- time





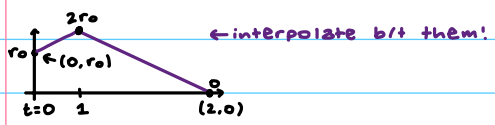
instructions list

CANVAS:
- Color ← self.color
- ellipse() ← need to make object & hook up & connect
- ellipse() → Color

INSTRUCTION GROUP:
- BUBBLE

- Color()
- Ellipse() } → BUBBLE
- Color()
- Ellipse

KEY FRAMES:

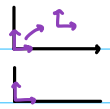


NUMERICAL INTEGRATION:

$x(t+\Delta t) = x(t) + v \cdot \Delta t$ ← for position from speed

$v(t+\Delta t) = v(t) + a \cdot \Delta t$

CANVAS:
- Translate(x,y)
- Rotate(θ)



2/25 MUSIC:
- pitch
- loudness
- duration
- start time
- melodic contour
- rhythm
- tempo (rit, accel.)
- chords / polyphony
- articulation (staccato / legato)
- harmonic
- chord quality
- consonance / dissonance
- tonal center
- instrument / timbre

GRAPHICS:
- color / vibrance / opacity
- motion / direction / speed / duration
- shape
- size
- text
- position
- texture
- object composition / density

3/1 **MIDI** — **Fluid Synth** (midi package on Python)

- 16 channels
- Supports polyphony

noteon(channel, pitch, vel) — 0-127 0-127

noteoff(channel, pitch) ← 1 to 1 matching

program(channel, bank, preset) — bank preset; 000-000 = Yamaha Grand Piano

cc(channel, ctrl, volume) — ctrl=0 → vibrato; ctrl=7 → volume; ctrl=64 → sustain pedal

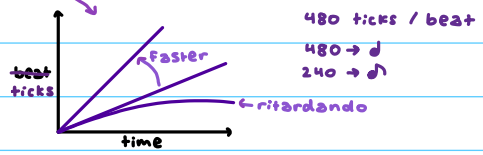
pitchbend(channel, volume) → -8k to +8k; -2^{13} to $+2^{13}-1$

Synth ⇄ Audio

generate()

Clock — get_time()

Tempo Map — tick_to_time(t), time_to_ticks(t)



480 ticks / beat

480 → ♩

240 → ♪

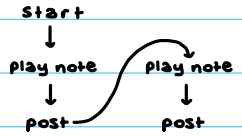
Scheduler

cmd = post_at_tick(func, tick, arg)

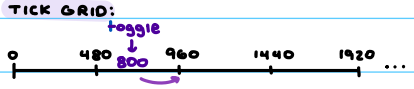
on_update()

cancel(_)

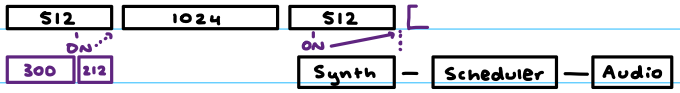
SINGLE LINE NOTE GENERATOR:

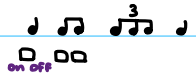


TICK GRID:



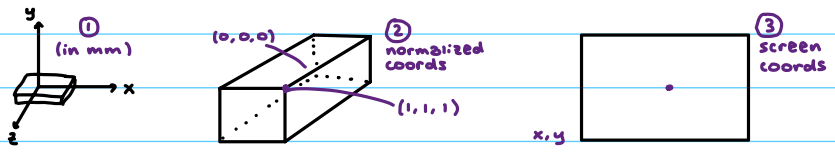
800 → 960



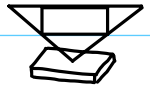


3/9

DEVICE	DISCRETE/CONTINUOUS	SUGGESTIVE	FEEDBACK
light switch	D	Yes	click sound, tactile
game pad	both (buttons D, joyst. C)	a little	tactile, sound vibration
guitar hero guitar	D	Yes	click tactile
launchpad	D	weak	visual, tactile
motion sensor	C	No	None



HYSTERESIS: buffer zone



DIRECT SENSOR FEEDBACK

COMPONENT ACTION/RESULT

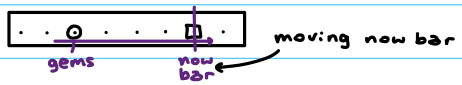
COMPONENT INTERACTION FEEDBACK

- push: can't change mind, will turn on what you hover over
- relative push: has a threshold before button is pushed
- timed push: takes longer to push, but more careful & has threshold

2/16 history of beatmatching games:

PARAPPA:

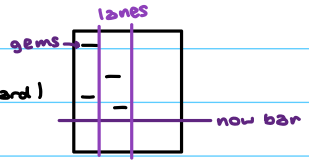
- calls & response



- 5 levels only in game, but lots of interaction

BEATMANIA:

- 2 hands (L = turntable, R = keyboard)
- fixed now bar



- no call & response, no storyline
- song: vivaldi "winter" (wl drums)
- streak indicator (competitive)

SAMBA:

- has characters
- Latin America (culturally not appropriate)

AMPLITUDE (harmonix)

- you are the now bar (fixed now bar, but feels like you're moving through the screen)
- 3 lanes per instrument
- track in 3D space (things further back are smaller)

- character in a box @ the right

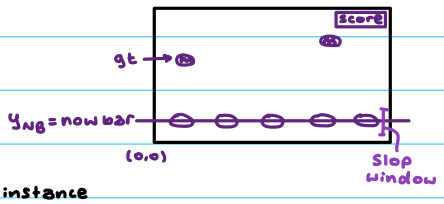
KARAOKE REVOLUTION

- character: singer on screen
- fixed nav bar
- trying to match pitch (gems)

GUITAR HERO:

- fixed nav bar
- 5 lanes & characters
- licensed music
- levels of difficulty introduced
- when you mess up, guitar track stops playing.

- _bg = everything but guitar
- _solo = lead guitar
- in Sonic Visualizer: make a time instance layer
  - pressing ";" gives you a time instance.
  - replaying will give you a click @ each time instance
- need to convert time to pixels



$y_g = \frac{\Delta y}{\Delta t} t_g + y_{NB}$ → if m is bigger, less notes (think of m as density)

$y_g(t) = \frac{\Delta y}{\Delta t}(t_g - t) + y_{NB}$

gem @ a time; time of now bar