GET BONUS ENTERTAINMENT SYSTEM
2015.09.27

GET BONUS ENTERTAINMENT SYSTEM
2015.09.27

CHECKING MEMORY.....

CODE AREA...0000000f42400000

The Get Bonus

infinite entertainment system

The

Get Bonus

infinite entertainment system

Level 1

Ricoh RP2A03

The

Get Bonus

infinite entertainment system

Level 1

Ricoh RP2A03

The

Get Bonus

infinite entertainment system

Level 1

Ricoh RP2A03

Level 2

NES Chamber Orchestra

The
Get Bonus

infinite entertainment system

Level 1

Ricoh RP2A03

Level 2

NES Chamber Orchestra

The

Get Bonus

infinite entertainment system

Level 1

Ricoh RP2A03

Level 2

NES Chamber Orchestra

Level 3

Bithoven

The

Get Bonus

infinite entertainment system

Level 1

Ricoh RP2A03

Level 2

NES Chamber Orchestra

Level 3

Bithoven

(17034937447589238875292491048880986423348676677269874515734788835411208296694341070947965714463046928135370824185326249995122150239327473127456027462485537556537573207)

Ricoh RP2A03

Ricoh RP2A03

1.789773 MHz

Ricoh RP2A03

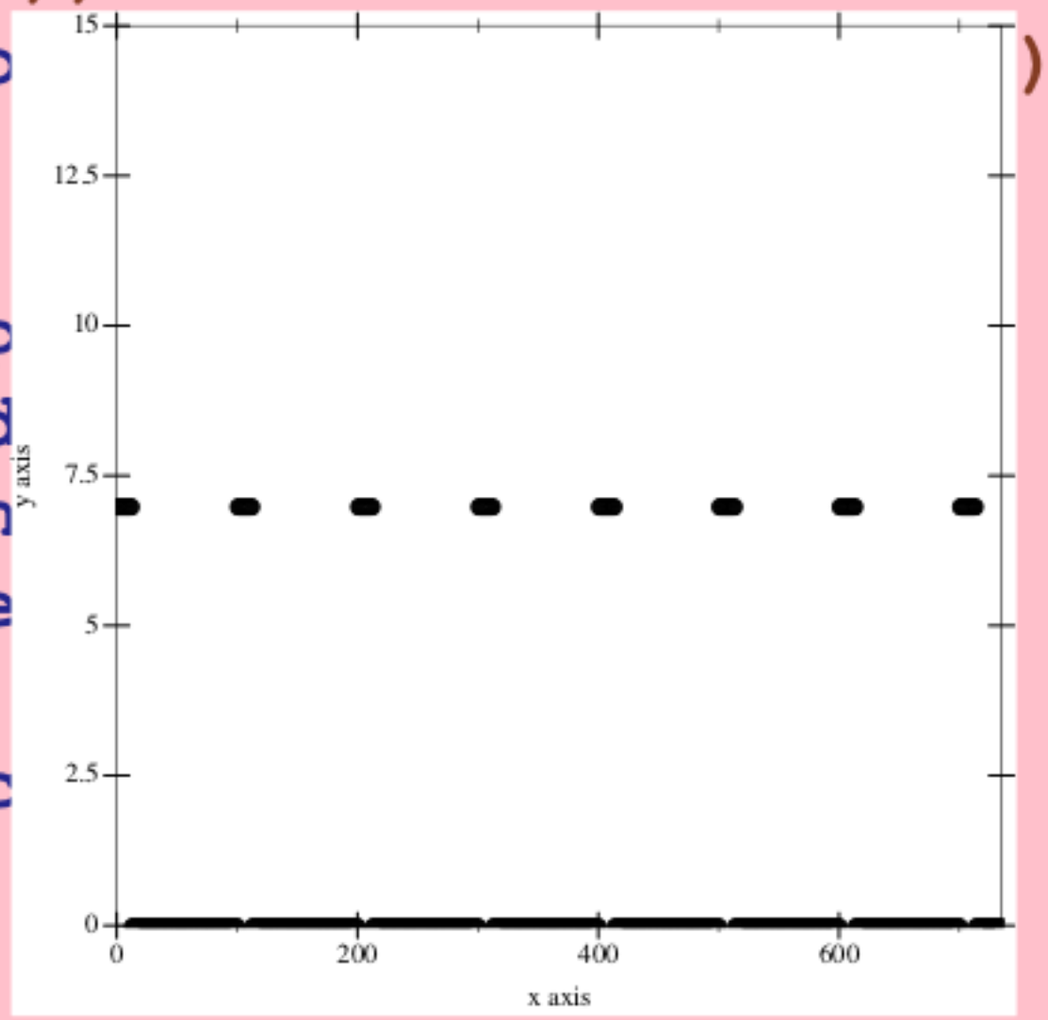
2 Pulse Waves
1 Triangle Wave
1 Noise Channel
7-bit Samples

Ricoh RP2A03

```
(define (pulse-period->freq period)
  (fl/ CPU-FREQ-Hz (fl* 16.0 (fl+ 1.0 (fx->fl period)))))
(define (cycle%-step % freq)
  (define %step (fl/ freq sample-rate.0))
  (define next% (fl+ % %step))
  (fl- next% (flfloor next%)))
(define DUTY-CYCLES (flvector 0.125 0.25 0.5 0.75))
(define (duty-n->cycle n)
  (flvector-ref DUTY-CYCLES n))
(define (pulse-wave duty-n period volume %)
  (define freq (pulse-period->freq period))
  (define duty-cycle (duty-n->cycle duty-n))
  (define next-% (cycle%-step % freq))
  (define out
    (if (fl< next-% duty-cycle)
        volume
        0))
  (values out next-%))
```

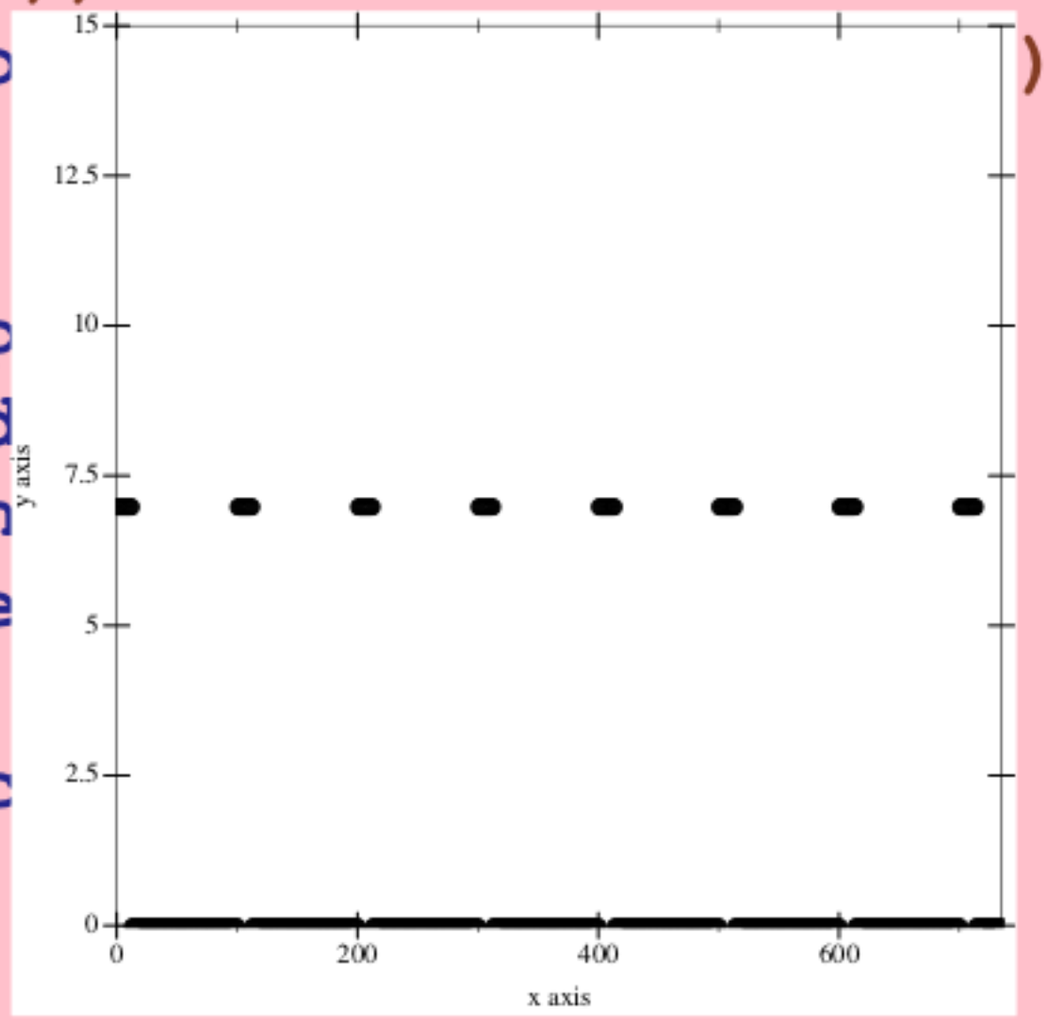
Ricoh RP2A03

```
(define (pulse-period->freq period)
  (fl/ CPU-FREQ-Hz (fl* 16.0 (fl+ 1.0 (fx->fl period)))))
(define (cycle%-step % freq)
  (define %step (fl/ freq sample-rate.0))
  (define next% (fl+ % %step))
  (fl- next% (flfloor next%)))
(define DUTY-CYCLES (flvecto
(define (duty-n->cycle n)
  (flvector-ref DUTY-CYCLES
(define (pulse-wave duty-n p
  (define freq (pulse-period
  (define duty-cycle (duty-n
  (define next-% (cycle%-ste
  (define out
    (if (fl< next-% duty-cyc
        volume
        0))
  (values out next-%))
```



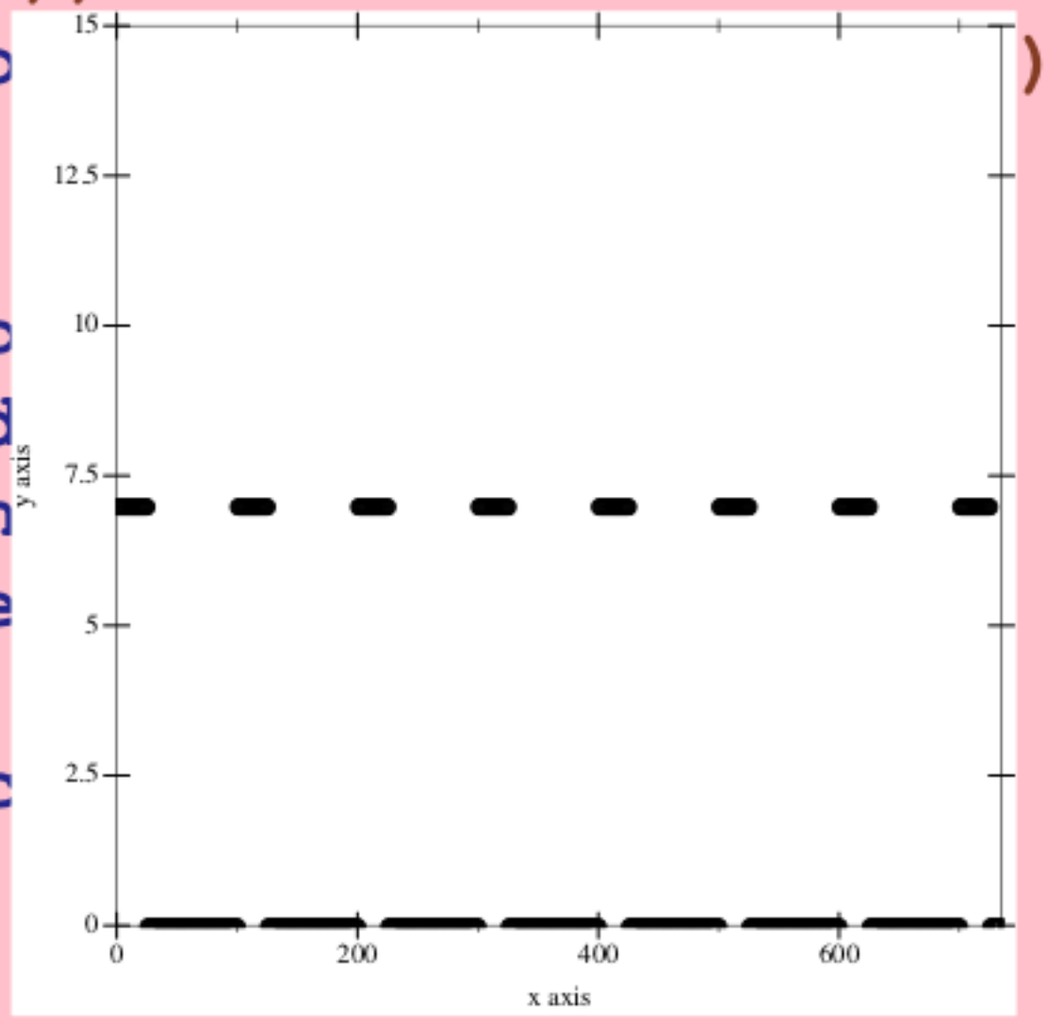
Ricoh RP2A03

```
(define (pulse-period->freq period)
  (fl/ CPU-FREQ-Hz (fl* 16.0 (fl+ 1.0 (fx->fl period)))))
(define (cycle%-step % freq)
  (define %step (fl/ freq sample-rate.0))
  (define next% (fl+ % %step))
  (fl- next% (flfloor next%)))
(define DUTY-CYCLES (flvecto
(define (duty-n->cycle n)
  (flvector-ref DUTY-CYCLES
(define (pulse-wave duty-n p
  (define freq (pulse-period
  (define duty-cycle (duty-n
  (define next-% (cycle%-ste
  (define out
    (if (fl< next-% duty-cyc
        volume
        0))
  (values out next-%))
```



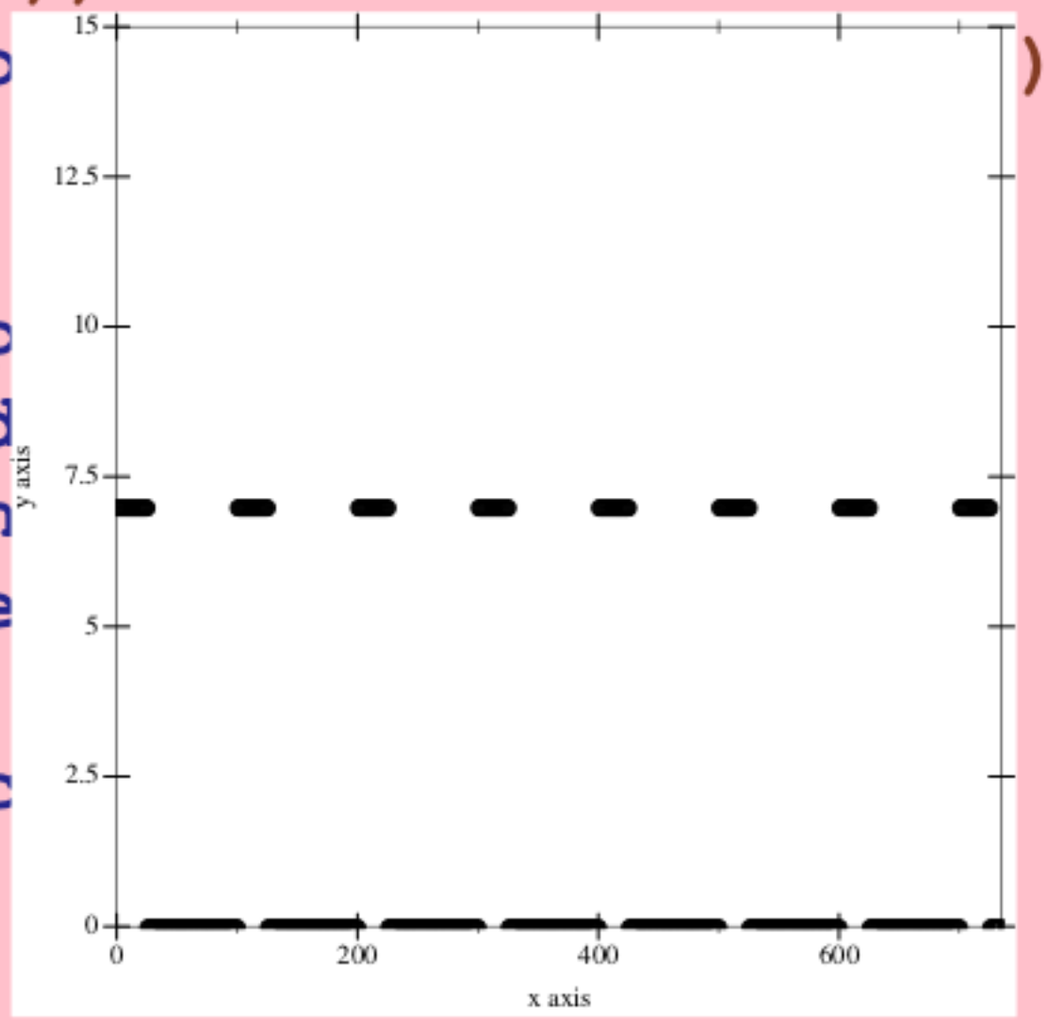
Ricoh RP2A03

```
(define (pulse-period->freq period)
  (fl/ CPU-FREQ-Hz (fl* 16.0 (fl+ 1.0 (fx->fl period)))))
(define (cycle%-step % freq)
  (define %step (fl/ freq sample-rate.0))
  (define next% (fl+ % %step))
  (fl- next% (flfloor next%)))
(define DUTY-CYCLES (flvecto
(define (duty-n->cycle n)
  (flvector-ref DUTY-CYCLES
(define (pulse-wave duty-n p
  (define freq (pulse-period
  (define duty-cycle (duty-n
  (define next-% (cycle%-ste
  (define out
    (if (fl< next-% duty-cyc
        volume
        0))
  (values out next-%))
```



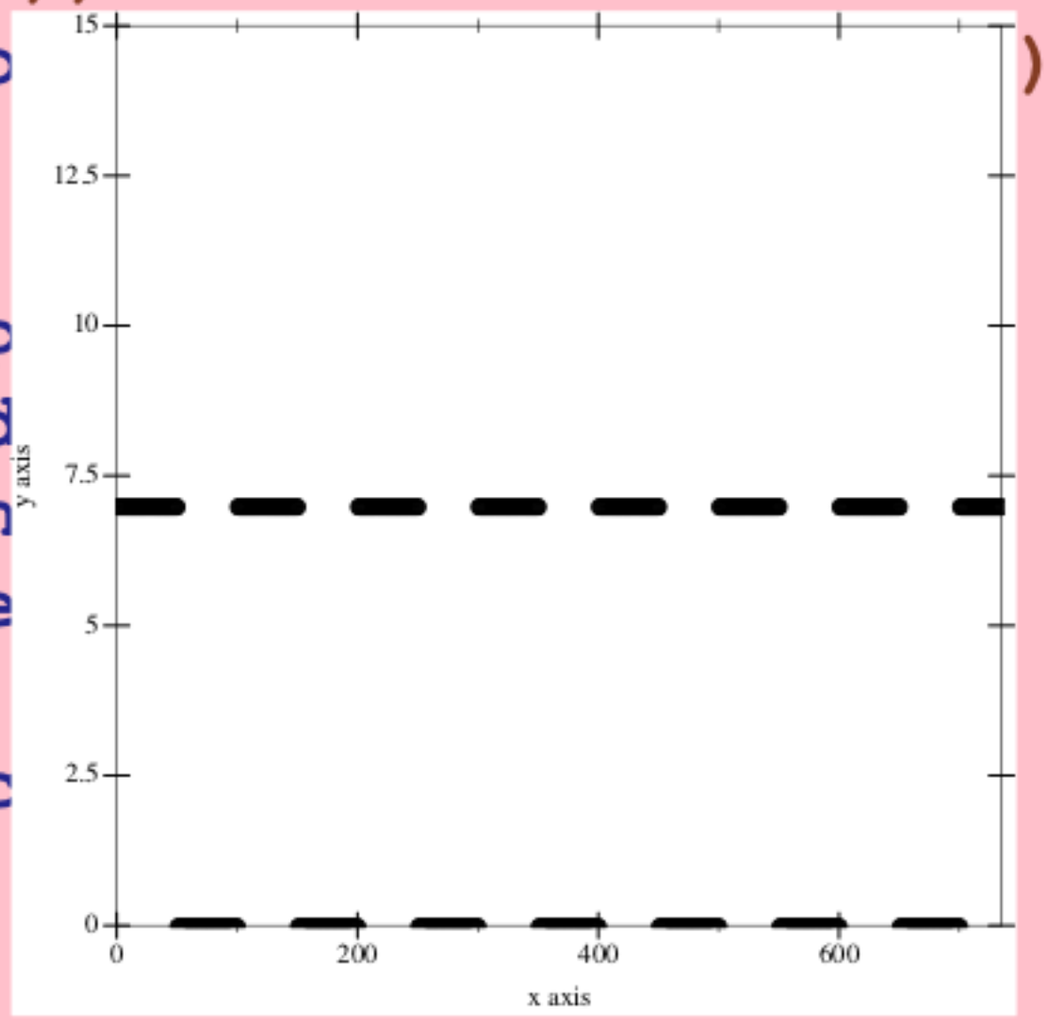
Ricoh RP2A03

```
(define (pulse-period->freq period)
  (fl/ CPU-FREQ-Hz (fl* 16.0 (fl+ 1.0 (fx->fl period)))))
(define (cycle%-step % freq)
  (define %step (fl/ freq sample-rate.0))
  (define next% (fl+ % %step))
  (fl- next% (flfloor next%)))
(define DUTY-CYCLES (flvecto
(define (duty-n->cycle n)
  (flvector-ref DUTY-CYCLES
(define (pulse-wave duty-n p
  (define freq (pulse-period
  (define duty-cycle (duty-n
  (define next-% (cycle%-ste
  (define out
    (if (fl< next-% duty-cyc
        volume
        0))
  (values out next-%))
```



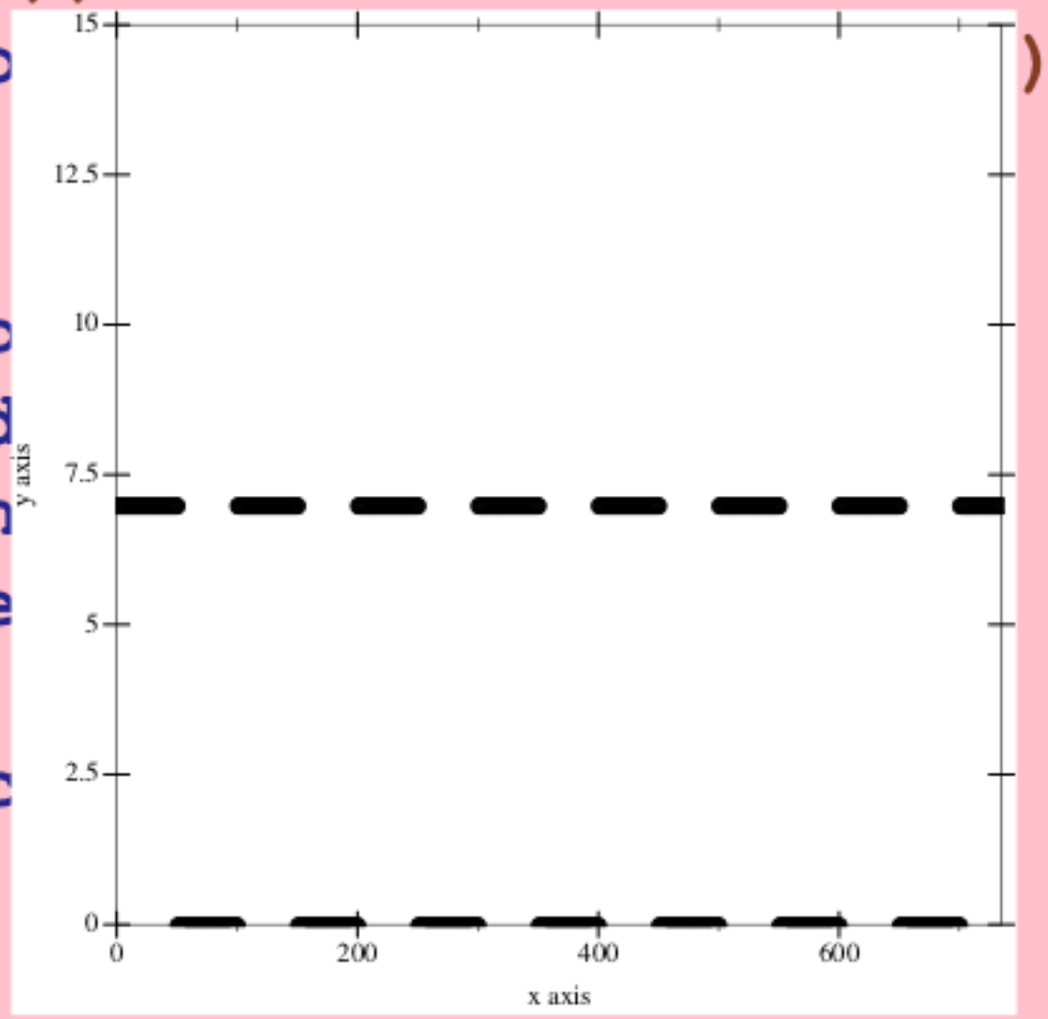
Ricoh RP2A03

```
(define (pulse-period->freq period)
  (fl/ CPU-FREQ-Hz (fl* 16.0 (fl+ 1.0 (fx->fl period)))))
(define (cycle%-step % freq)
  (define %step (fl/ freq sample-rate.0))
  (define next% (fl+ % %step))
  (fl- next% (flfloor next%)))
(define DUTY-CYCLES (flvecto
(define (duty-n->cycle n)
  (flvector-ref DUTY-CYCLES
(define (pulse-wave duty-n p
  (define freq (pulse-period
  (define duty-cycle (duty-n
  (define next-% (cycle%-ste
  (define out
    (if (fl< next-% duty-cyc
        volume
        0))
  (values out next-%))
```



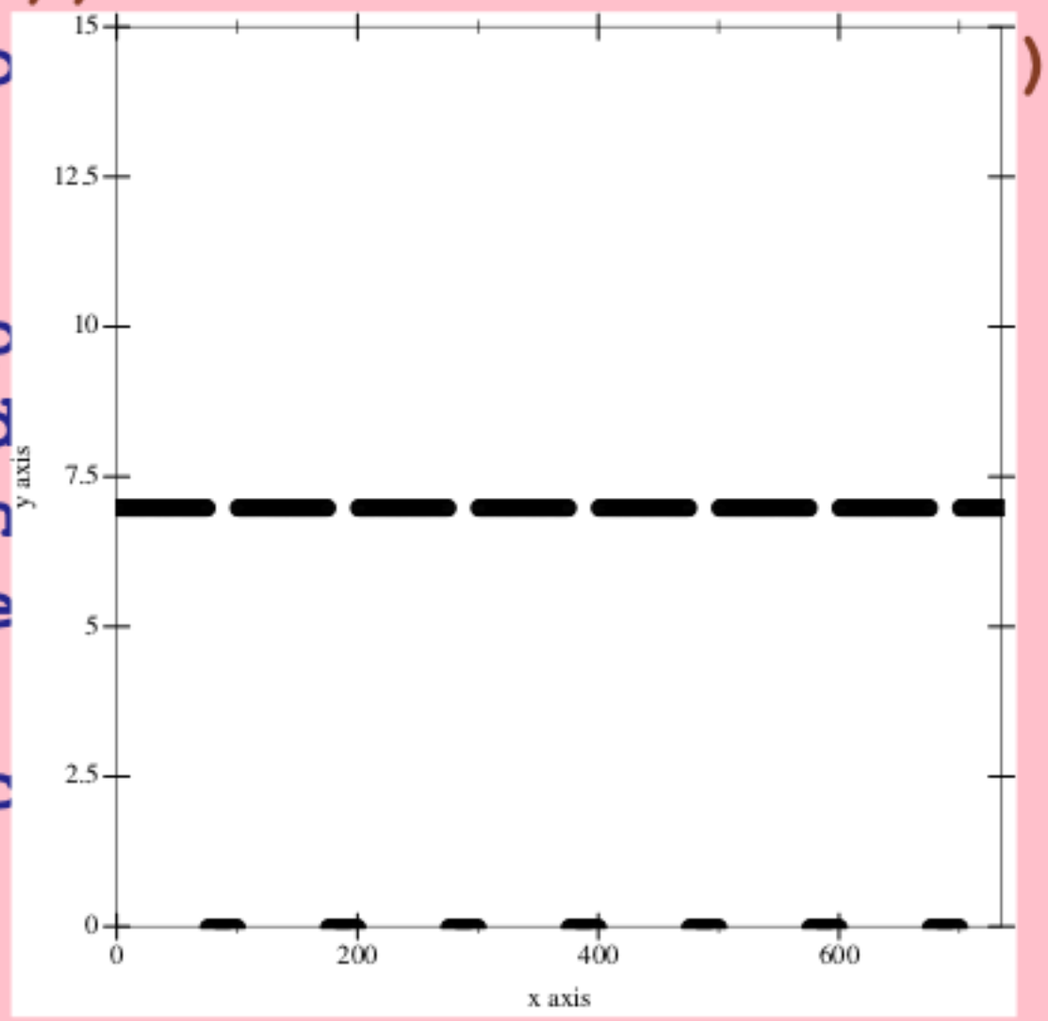
Ricoh RP2A03

```
(define (pulse-period->freq period)
  (fl/ CPU-FREQ-Hz (fl* 16.0 (fl+ 1.0 (fx->fl period)))))
(define (cycle%-step % freq)
  (define %step (fl/ freq sample-rate.0))
  (define next% (fl+ % %step))
  (fl- next% (flfloor next%)))
(define DUTY-CYCLES (flvecto
(define (duty-n->cycle n)
  (flvector-ref DUTY-CYCLES
(define (pulse-wave duty-n p
  (define freq (pulse-period
  (define duty-cycle (duty-n
  (define next-% (cycle%-ste
  (define out
    (if (fl< next-% duty-cyc
        volume
        0))
  (values out next-%))
```



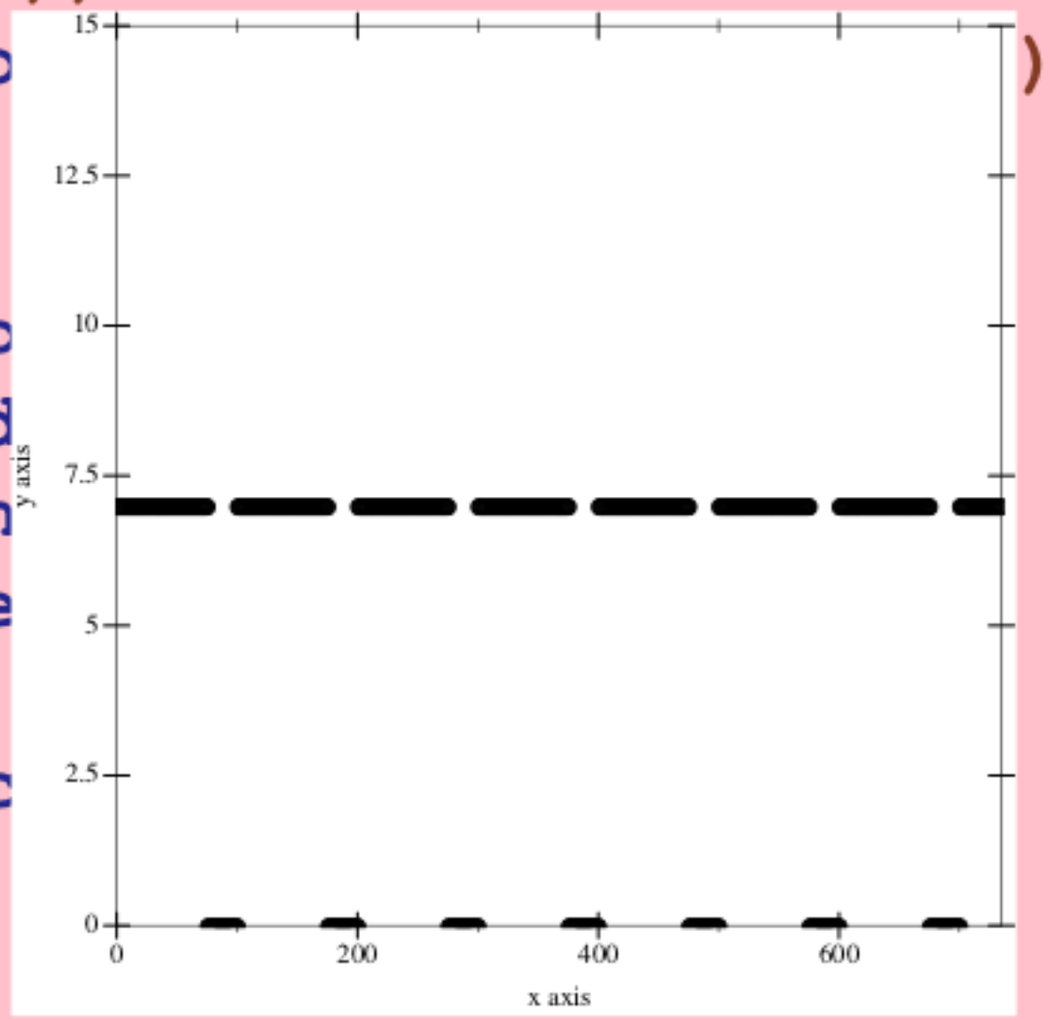
Ricoh RP2A03

```
(define (pulse-period->freq period)
  (fl/ CPU-FREQ-Hz (fl* 16.0 (fl+ 1.0 (fx->fl period)))))
(define (cycle%-step % freq)
  (define %step (fl/ freq sample-rate.0))
  (define next% (fl+ % %step))
  (fl- next% (flfloor next%)))
(define DUTY-CYCLES (flvecto
(define (duty-n->cycle n)
  (flvector-ref DUTY-CYCLES
(define (pulse-wave duty-n p
  (define freq (pulse-period
  (define duty-cycle (duty-n
  (define next-% (cycle%-ste
  (define out
    (if (fl< next-% duty-cyc
        volume
        0))
  (values out next-%))
```



Ricoh RP2A03

```
(define (pulse-period->freq period)
  (fl/ CPU-FREQ-Hz (fl* 16.0 (fl+ 1.0 (fx->fl period)))))
(define (cycle%-step % freq)
  (define %step (fl/ freq sample-rate.0))
  (define next% (fl+ % %step))
  (fl- next% (flfloor next%)))
(define DUTY-CYCLES (flvecto                    )
(define (duty-n->cycle n)
  (flvector-ref DUTY-CYCLES
(define (pulse-wave duty-n p
  (define freq (pulse-period
  (define duty-cycle (duty-n
  (define next-% (cycle%-ste
  (define out
    (if (fl< next-% duty-cyc
        volume
        0))
  (values out next-%))
```

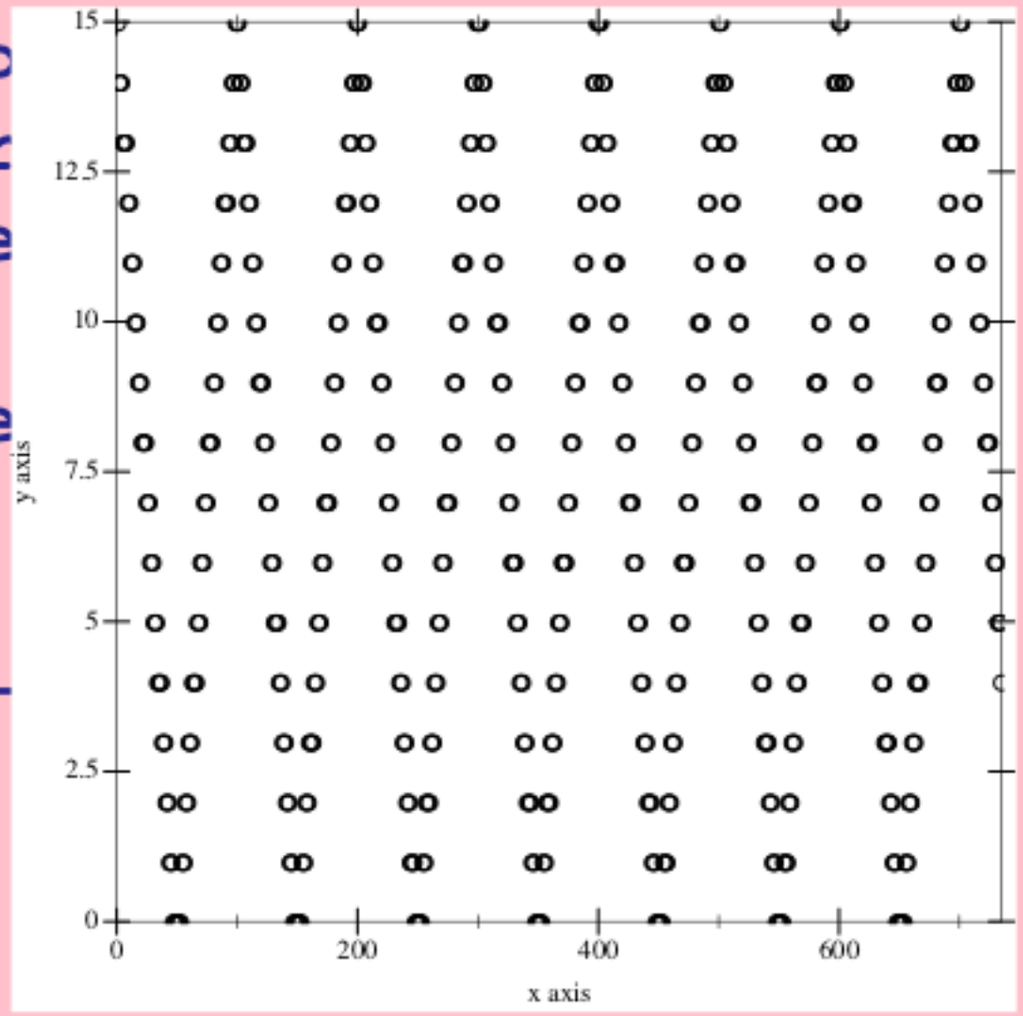


Ricoh RP2A03

```
(define (triangle-period->freq period)
  (fl/ (pulse-period->freq period) 2.0))
(define TRIANGLE-PATTERN
  (bytes
   15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15))
(define (triangle-wave on? period %)
  (define freq (triangle-period->freq period))
  (define next-% (cycle%-step % freq))
  (define %-as-step
    (fl->fx (flround (fl* next-% 31.0))))
  (define out
    (if on?
        (bytes-ref TRIANGLE-PATTERN %-as-step)
        0))
  (values out next-%))
```

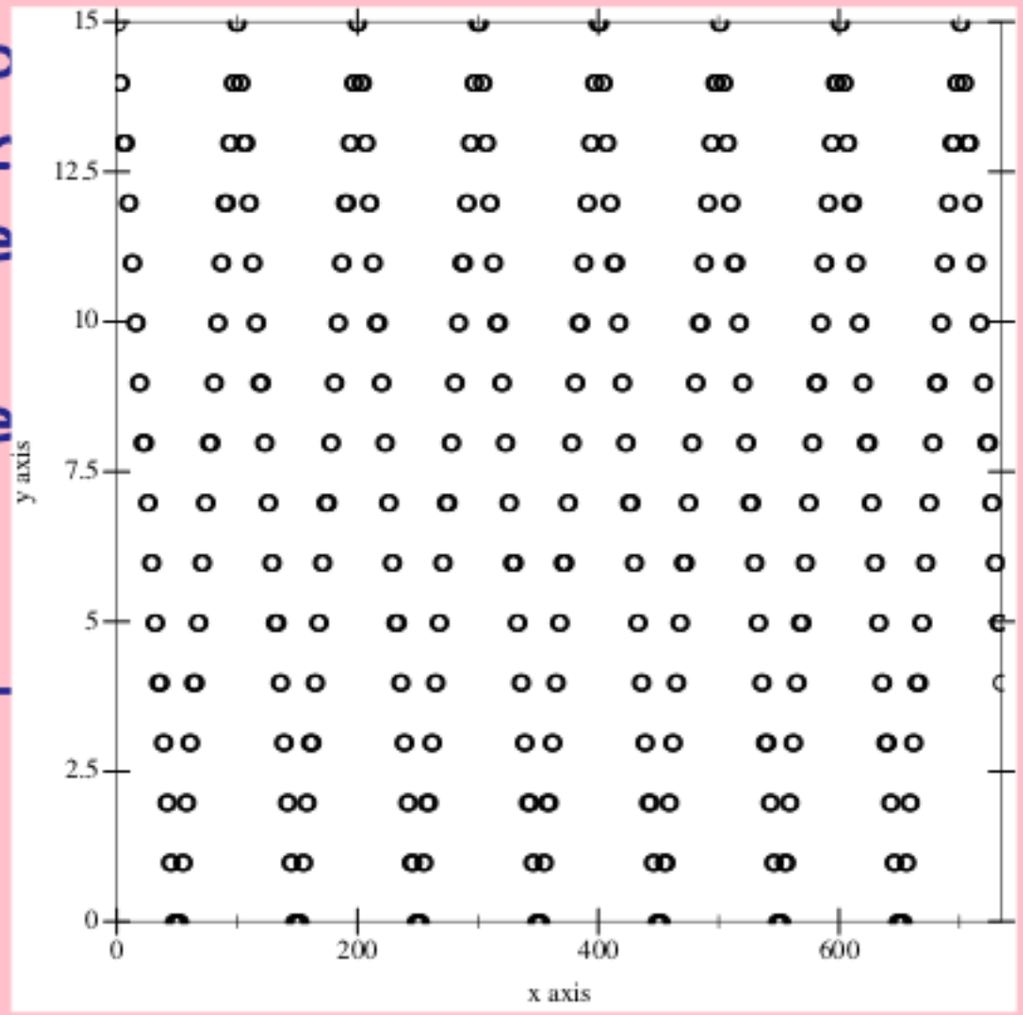
Ricoh RP2A03

```
(define (triangle-period->freq period)
  (fl/ (pulse-period->freq period) 2.0))
(define TRIANGLE-PATTERN
  (bytes
   15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15))
(define (triangle-wave on? p
  (define freq (triangle-per
  (define next-% (cycle%-ste
(define %-as-step
  (fl->fx (flround (fl* ne
(define out
  (if on?
      (bytes-ref TRIANGLE-
      0))
(values out next-%))
```



Ricoh RP2A03

```
(define (triangle-period->freq period)
  (fl/ (pulse-period->freq period) 2.0))
(define TRIANGLE-PATTERN
  (bytes
   15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15))
(define (triangle-wave on? p
  (define freq (triangle-per
  (define next-% (cycle%-ste
(define %-as-step
  (fl->fx (flround (fl* ne
(define out
  (if on?
      (bytes-ref TRIANGLE-
      0))
(values out next-%))
```

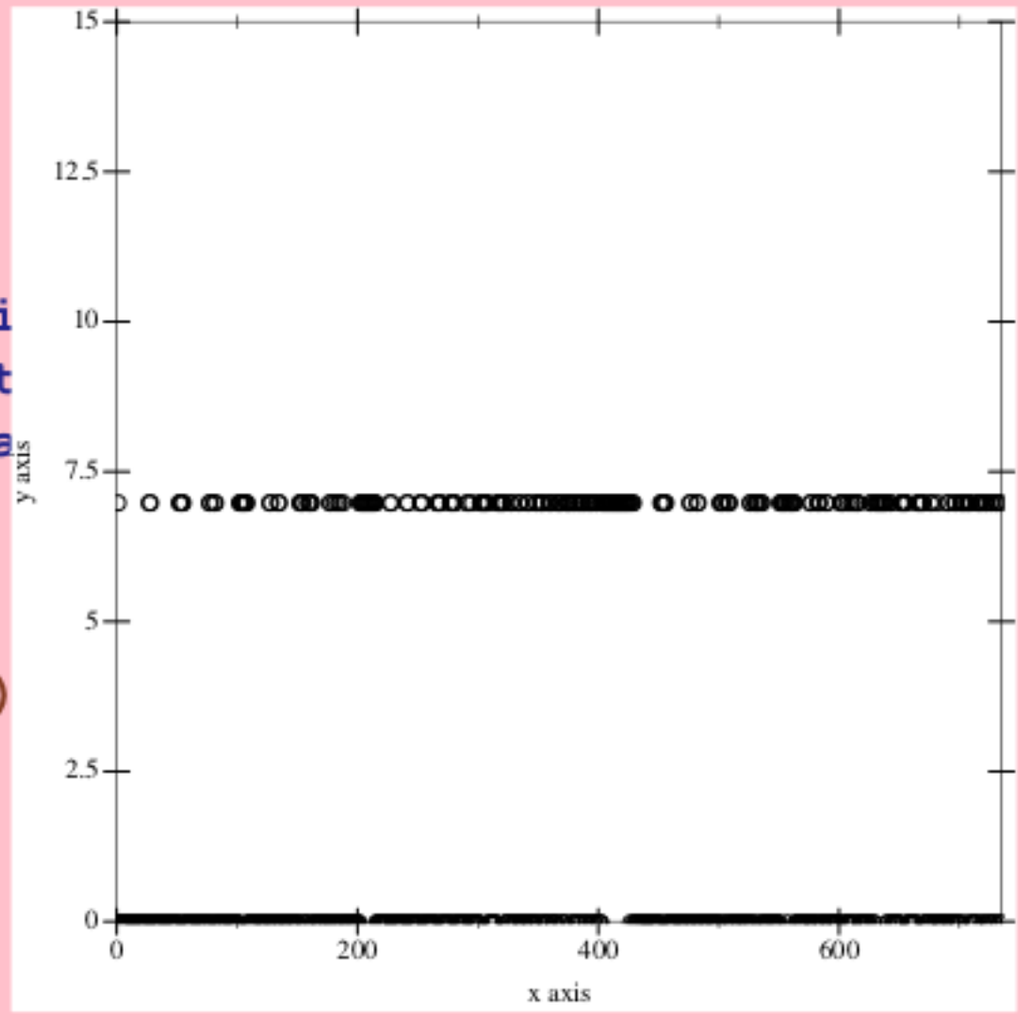


Ricoh RP2A03

```
(define NOISE-PERIODS
  (vector 4 8 16 32 64 96 128 160 202 254 380 508 762 1016 2034 4068))
(define (noise-period->freq period)
  (fl* (pulse-period->freq period) 8.0))
(define (noise short? period volume register %)
  (define freq (noise-period->freq period))
  (define next-% (cycle%-step % freq))
  (define next-register
    (cond
      [(fl< next-% %)
       (define (bit i) (bitwise-bit-field register i (fx+ i 1)))
       (define other-bit (if short? 6 1))
       (define feedback (bitwise-xor (bit 0) (bit other-bit)))
       (define shifted-ref (arithmetic-shift register -1))
       (define feedback-at-bit14 (arithmetic-shift feedback 14))
       (bitwise-ior shifted-ref feedback-at-bit14)]
      [else
       register]))
  (values
   (fx* volume (fxmodulo next-register 2))
   next-register
   next-%))
```

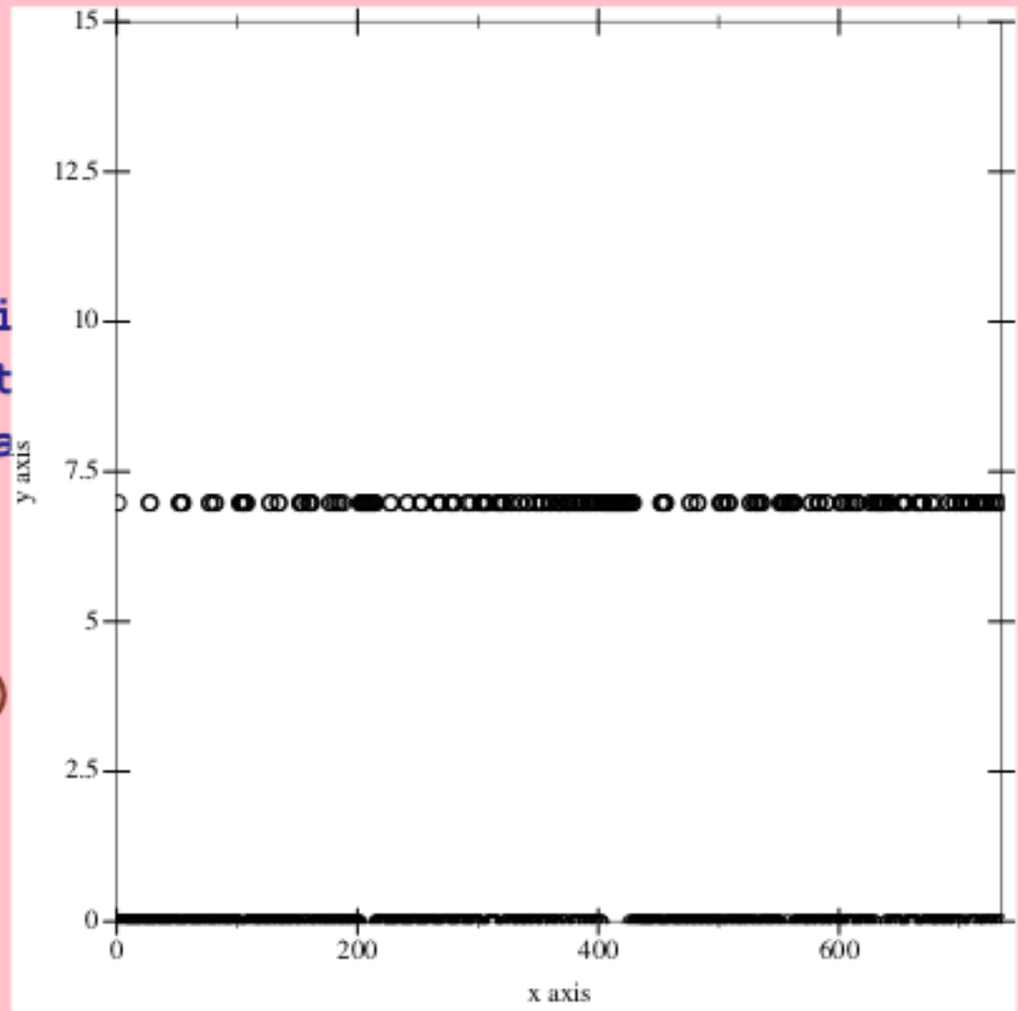
Ricoh RP2A03

```
(define NOISE-PERIODS
  (vector 4 8 16 32 64 96 128 160 202 254 380 508 762 1016 2034 4068))
(define (noise-period->freq period)
  (fl* (pulse-period->freq period) 8.0))
(define (noise short? period volume register %)
  (define freq (noise-period->freq period))
  (define next-% (cycle%-step % freq))
  (define next-register
    (cond
      [(fl< next-% %)
       (define (bit i) (bitwise-bit-field
       (define other-bit (if short? 6 1))
       (define feedback (bitwise-xor (bit
       (define shifted-ref (arithmetic-shi
       (define feedback-at-bit14 (arithmet
       (bitwise-ior shifted-ref feedback-a
      [else
       register]))
(values
 (fx* volume (fxmodulo next-register 2))
 next-register
 next-%))
```



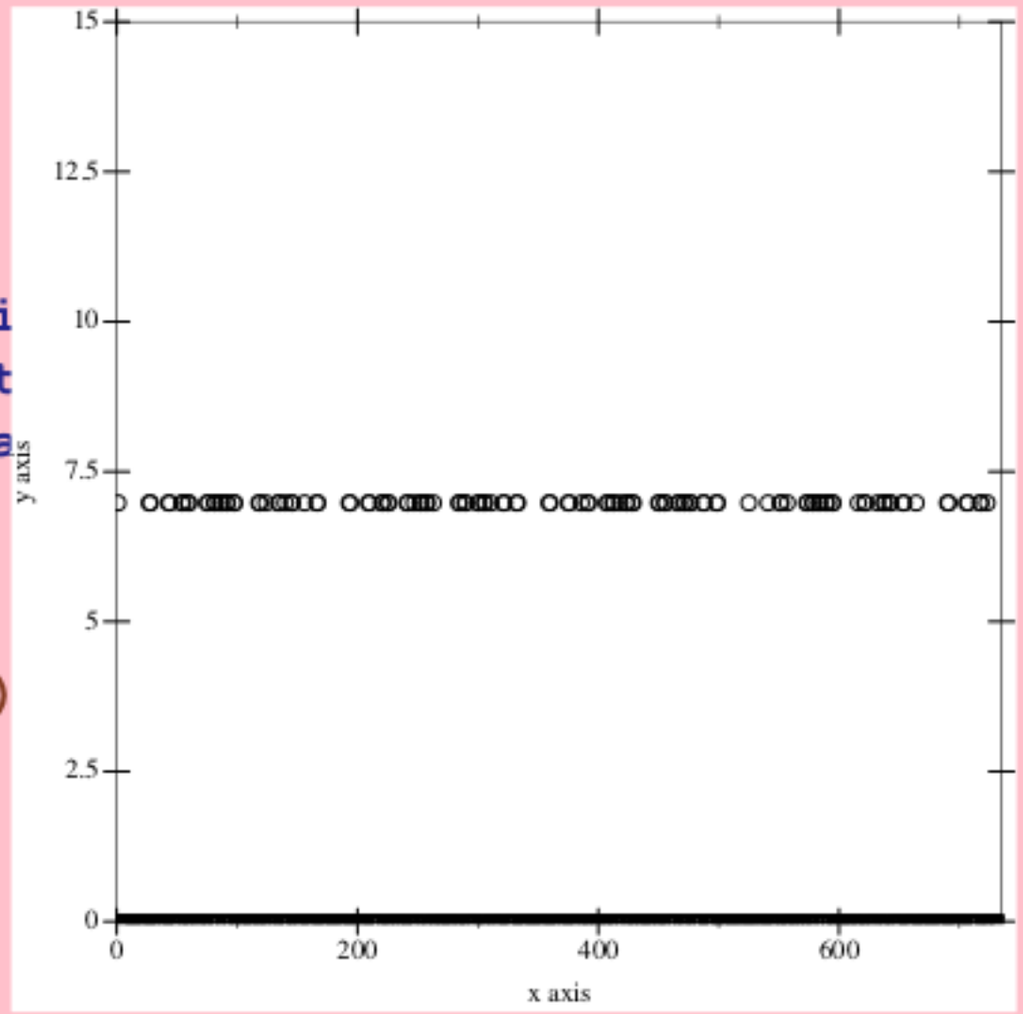
Ricoh RP2A03

```
(define NOISE-PERIODS
  (vector 4 8 16 32 64 96 128 160 202 254 380 508 762 1016 2034 4068))
(define (noise-period->freq period)
  (fl* (pulse-period->freq period) 8.0))
(define (noise short? period volume register %)
  (define freq (noise-period->freq period))
  (define next-% (cycle%-step % freq))
  (define next-register
    (cond
      [(fl< next-% %)
       (define (bit i) (bitwise-bit-field
       (define other-bit (if short? 6 1))
       (define feedback (bitwise-xor (bit
       (define shifted-ref (arithmetic-shi
       (define feedback-at-bit14 (arithmet
       (bitwise-ior shifted-ref feedback-a
      [else
       register]))
  (values
   (fx* volume (fxmodulo next-register 2))
   next-register
   next-%))
```



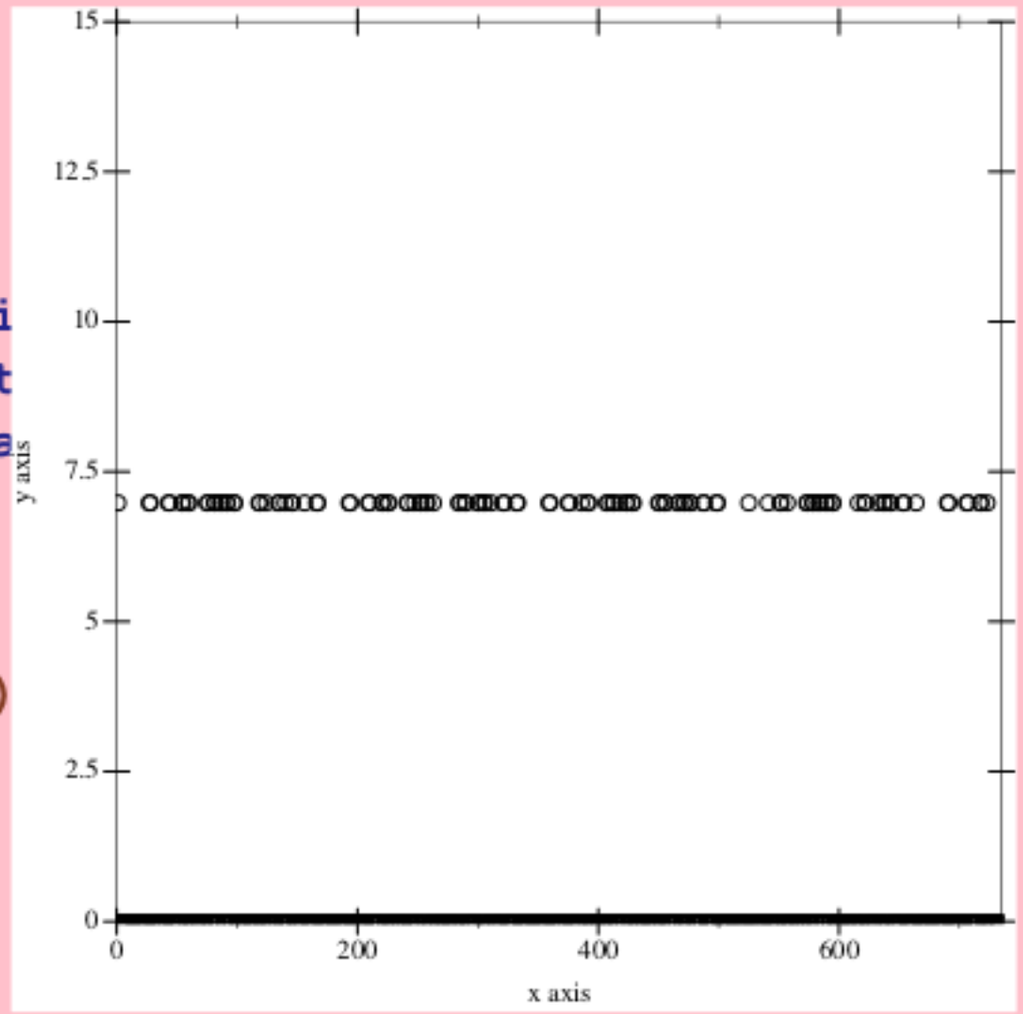
Ricoh RP2A03

```
(define NOISE-PERIODS
  (vector 4 8 16 32 64 96 128 160 202 254 380 508 762 1016 2034 4068))
(define (noise-period->freq period)
  (fl* (pulse-period->freq period) 8.0))
(define (noise short? period volume register %)
  (define freq (noise-period->freq period))
  (define next-% (cycle%-step % freq))
  (define next-register
    (cond
      [(fl< next-% %)
       (define (bit i) (bitwise-bit-field
       (define other-bit (if short? 6 1))
       (define feedback (bitwise-xor (bit
       (define shifted-ref (arithmetic-shi
       (define feedback-at-bit14 (arithmet
       (bitwise-ior shifted-ref feedback-a
      [else
       register]))
  (values
   (fx* volume (fxmodulo next-register 2))
   next-register
   next-%))
```



Ricoh RP2A03

```
(define NOISE-PERIODS
  (vector 4 8 16 32 64 96 128 160 202 254 380 508 762 1016 2034 4068))
(define (noise-period->freq period)
  (fl* (pulse-period->freq period) 8.0))
(define (noise short? period volume register %)
  (define freq (noise-period->freq period))
  (define next-% (cycle%-step % freq))
  (define next-register
    (cond
      [(fl< next-% %)
       (define (bit i) (bitwise-bit-field
       (define other-bit (if short? 6 1))
       (define feedback (bitwise-xor (bit
       (define shifted-ref (arithmetic-shi
       (define feedback-at-bit14 (arithmet
       (bitwise-ior shifted-ref feedback-a
      [else
       register]))
  (values
   (fx* volume (fxmodulo next-register 2))
   next-register
   next-%))
```

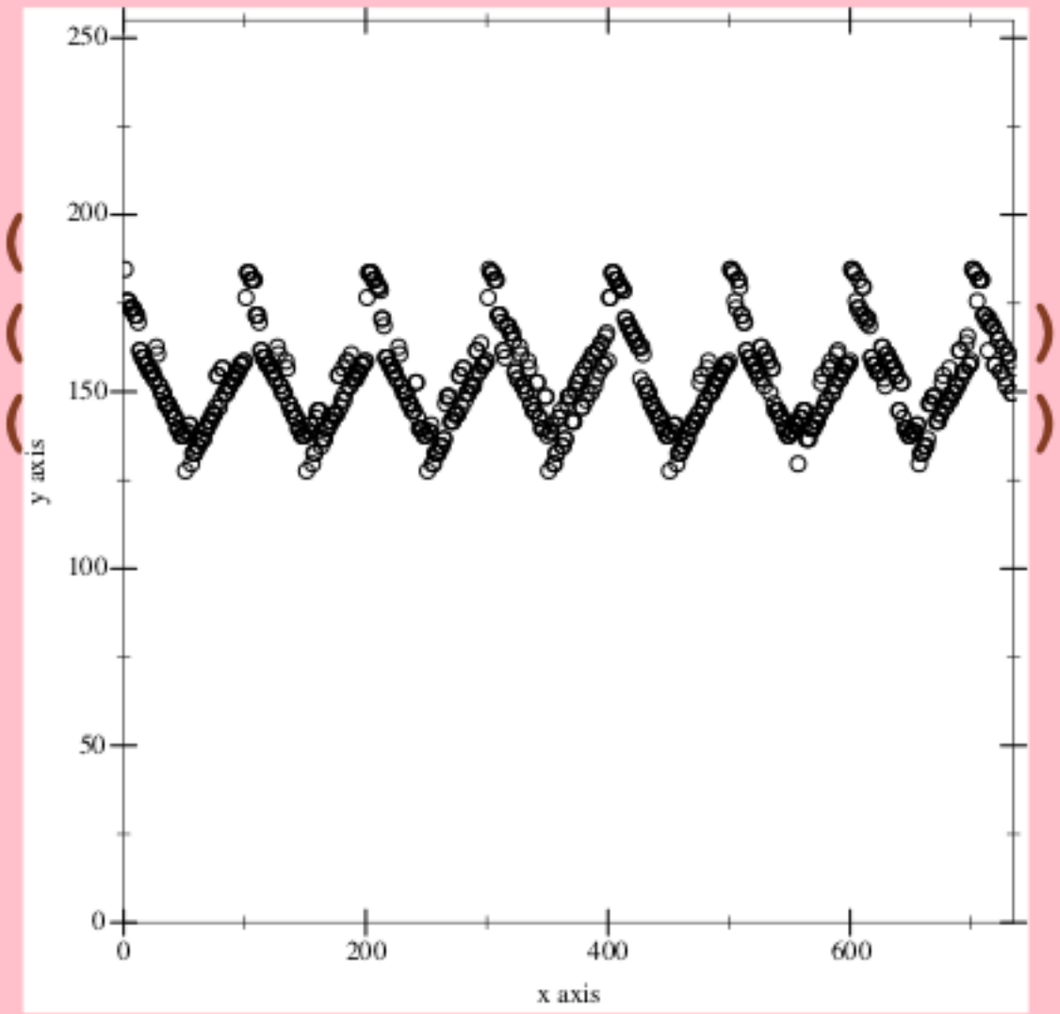


Ricoh RP2A03

```
(define (raw-p-mix p1 p2)
  (fl/ 95.88
       (fl+ (fl/ 8128.0
                 (fx->fl (fx+ p1 p2)))
            100.0)))
(define (raw-tnd-mix t n d)
  (fl/ 159.79
       (fl+ (fl/ 1.0
                 (fl+ (fl/ (fx->fl t) 8227.0)
                      (fl+ (fl/ (fx->fl n) 12241.0)
                           (fl/ (fx->fl d) 22638.0)))))
            100.0)))
(define (mix p tnd)
  (fx+ 128 (fx+ p tnd)))
```

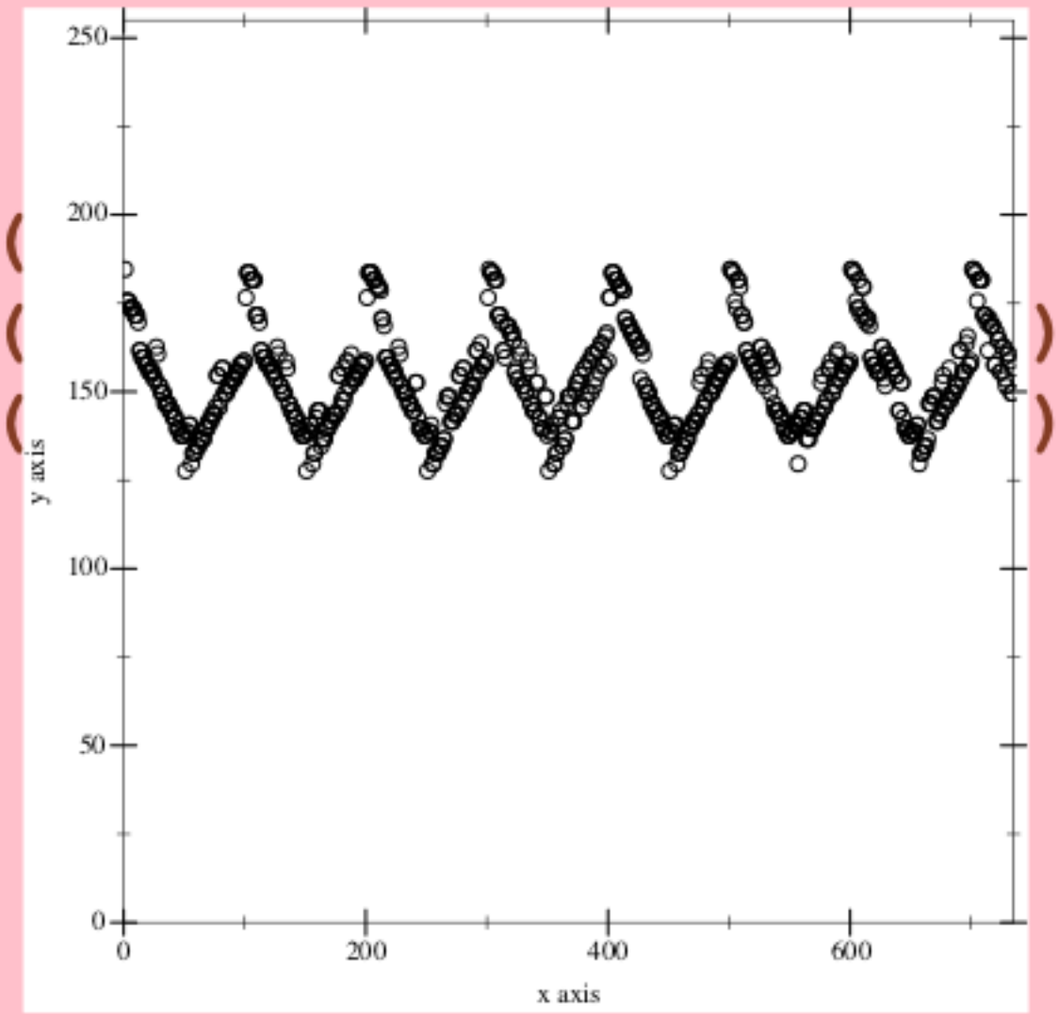
Ricoh RP2A03

```
(define (raw-p-mix p1 p2)
  (fl/ 95.88
       (fl+ (fl/ 8128.0
                 (fx->fl (fx+ p1 p2)))
            100.0)))
(define (raw-tnd-mix t n d)
  (fl/ 159.79
       (fl+ (fl/ 1.0
                 (fl+ (fl/ (
                           (fl+ (
                                 
                                 
                 100.0)))
(define (mix p tnd)
  (fx+ 128 (fx+ p tnd)))
```



Ricoh RP2A03

```
(define (raw-p-mix p1 p2)
  (fl/ 95.88
       (fl+ (fl/ 8128.0
                 (fx->fl (fx+ p1 p2)))
            100.0)))
(define (raw-tnd-mix t n d)
  (fl/ 159.79
       (fl+ (fl/ 1.0
                 (fl+ (fl/ (
                           (fl+ (
                                 
                                 
                 100.0)))
(define (mix p tnd)
  (fx+ 128 (fx+ p tnd)))
```



NES Chamber Orchestra

NES Chamber Orchestra

tones = frequencies

notes = 2^{-n}, 0 <= n <= 4

metronome = note x bpm

```
(define (frames-in-note.0 me note)
  (match-define (cons beat-unit beats-per-minute) me)
  (define beats-per-second
    (fl/ (fx->fl beats-per-minute) 60.0))
  (define beats-per-frame (fl/ beats-per-second 60.0))
  (define frames-per-beat (fl/ 1.0 beats-per-frame))
  (define beats-in-note (fl/ note beat-unit))
  (define frames-in-note
    (fl* beats-in-note frames-per-beat))
  frames-in-note)
```

NES Chamber Orchestra

tones = frequencies

notes = 2^{-n}, 0 <= n <= 4

metronome = note x bpm

```
(define (frames-in-note.0 me note)
  (match-define (cons beat-unit beats-per-minute) me)
  (define beats-per-second
    (fl/ (fx->fl beats-per-minute) 60.0))
  (define beats-per-frame (fl/ beats-per-second 60.0))
  (define frames-per-beat (fl/ 1.0 beats-per-frame))
  (define beats-in-note (fl/ note beat-unit))
  (define frames-in-note
    (fl* beats-in-note frames-per-beat))
  frames-in-note)
```

NES Chamber Orchestra

scale = list tone, octave
scale kind = tone -> scale
mode = scale rotation
chord (triad) = tones 2 2 0

```
(define-scale scale-diatonic-major '(2 2 1 2 2 2 1))
(define-scale scale-natural-minor '(2 1 2 2 1 2 2))
(define-scale scale-melodic-minor '(2 1 2 2 2 2 1))
(define-mode mode-ionian 0)
(define-mode mode-dorian 1)
(define-mode mode-phrygian 2)
```

NES Chamber Orchestra

scale = list tone, octave
scale kind = tone -> scale
mode = scale rotation
chord (triad) = tones 2 2 0

```
(define-scale scale-diatonic-major '(2 2 1 2 2 2 1))
(define-scale scale-natural-minor '(2 1 2 2 1 2 2))
(define-scale scale-melodic-minor '(2 1 2 2 2 2 1))
(define-mode mode-ionian 0)
(define-mode mode-dorian 1)
(define-mode mode-phrygian 2)
```

NES Chamber Orchestra

NES Chamber Orchestra

song = list parts
part = list measures
measure = list pulses
pulse = note, atone x3, drum, em?
atone = scale-idx, d-octave

```
(track
 (scale-diatonic-major 'C)
 '((((0.25 [0 4] [2 3] [4 2] #f)
     (0.25 [1 4] [3 3] [5 2] #t)
     (0.25 [2 4] [4 3] [6 2] #f)
     (0.25 [3 4] [5 3] [0 3] #t)))))
```

NES Chamber Orchestra

instrument = frames atone em?

-> wave

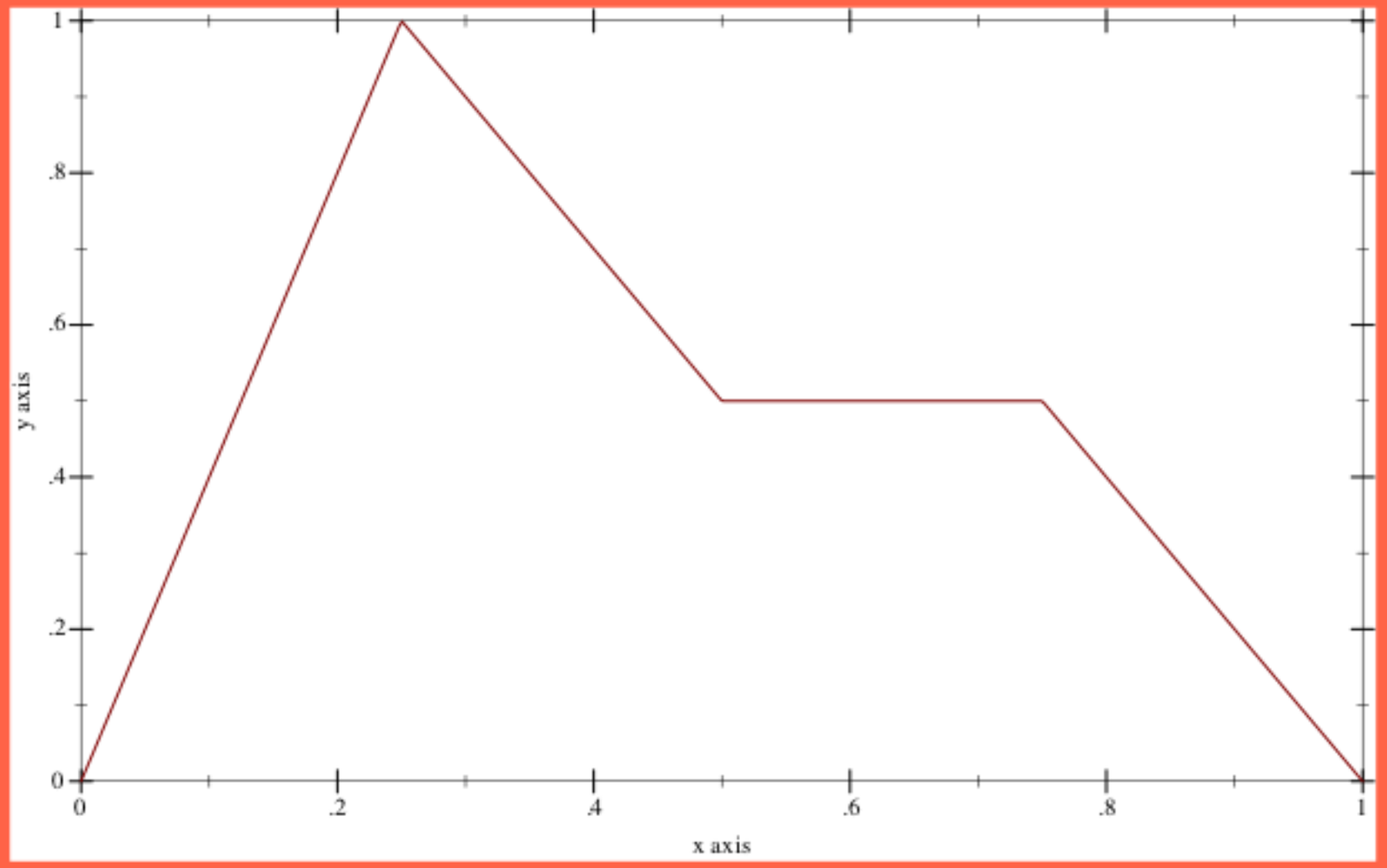
NES Chamber Orchestra

frp-like instrument dsl

NES Chamber Orchestra

frp-like instrument dsl

ADSR
attack
decay
sustain
release



NES Chamber Orchestra

```
(define (i:pulse:basic duty)
  (i:pulse/spec
   #:duty (spec:constant duty)
   #:period (spec:constant 0)
   #:volume (spec:constant 7)))
```

NES Chamber Orchestra

```
(define (i:pulse:basic duty)
  (i:pulse/spec
   #:duty (spec:constant duty)
   #:period (spec:constant 0)
   #:volume (spec:constant 7)))
```

NES Chamber Orchestra

```
(define (i:pulse:plucky duty)
  (i:pulse/spec
   #:duty (spec:constant duty)
   #:period (spec:constant 0)
   #:volume
   (spec:adsr 'release
              4 (spec:constant 14)
              4 (spec:linear 14 7)
              4 (spec:constant 7)
              4 (spec:linear 7 0))))
```

NES Chamber Orchestra

```
(define (i:pulse:plucky duty)
  (i:pulse/spec
   #:duty (spec:constant duty)
   #:period (spec:constant 0)
   #:volume
   (spec:adsr 'release
              4 (spec:constant 14)
              4 (spec:linear 14 7)
              4 (spec:constant 7)
              4 (spec:linear 7 0))))
```

NES Chamber Orchestra

```
(define (i:pulse:natural duty)
  (i:pulse/spec
   #:duty (spec:constant duty)
   #:period (spec:constant 0)
   #:volume
   (spec:adsr 'sustain
              4 (spec:constant 14)
              4 (spec:linear 14 7)
              4 (spec:constant 7)
              4 (spec:linear 7 0))))
```

NES Chamber Orchestra

```
(define (i:pulse:natural duty)
  (i:pulse/spec
   #:duty (spec:constant duty)
   #:period (spec:constant 0)
   #:volume
   (spec:adsr 'sustain
              4 (spec:constant 14)
              4 (spec:linear 14 7)
              4 (spec:constant 7)
              4 (spec:linear 7 0))))
```

NES Chamber Orchestra

```
(define hihat-adsr
  (spec:adsr 'release
             1 (spec:constant 4)
             2 (spec:constant 3)
             4 (spec:constant 2)
             4 (spec:constant 0)))
(define i:drum:hihat
  (i:drum/spec #:mode (spec:constant #f)
               #:period (spec:constant 12)
               #:volume hihat-adsr))
(define i:drum:bass
  (i:drum/spec #:mode (spec:constant #f)
               #:period (spec:constant 9)
               #:volume
               (spec:adsr 'release
                          1 (spec:constant 10)
                          2 (spec:constant 7)
                          4 (spec:linear 4 2)
                          4 (spec:constant 0))))
(define i:drums:basic
  (i:drums (vector i:drum:hihat i:drum:bass i:drum:snare)))
```

```
(define snare-adsr
  (spec:adsr 'release
             1 (spec:constant 11)
             4 (spec:linear 11 6)
             8 (spec:linear 6 2)
             4 (spec:constant 0)))
(define i:drum:snare
  (i:drum/spec #:mode (spec:constant #f)
               #:period (spec:constant 7)
               #:volume snare-adsr))
(define beat:straight-rock
  (list (cons 0.125 1) (cons 0.125 0)
        (cons 0.125 2) (cons 0.125 0)
        (cons 0.125 1) (cons 0.125 0)
        (cons 0.125 2) (cons 0.125 0)))
```

NES Chamber Orchestra

```
(define hihat-adsr
  (spec:adsr 'release
             1 (spec:constant 4)
             2 (spec:constant 3)
             4 (spec:constant 2)
             4 (spec:constant 0)))
(define i:drum:hihat
  (i:drum/spec #:mode (spec:constant #f)
               #:period (spec:constant 12)
               #:volume hihat-adsr))
(define i:drum:bass
  (i:drum/spec #:mode (spec:constant #f)
               #:period (spec:constant 9)
               #:volume
               (spec:adsr 'release
                          1 (spec:constant 10)
                          2 (spec:constant 7)
                          4 (spec:linear 4 2)
                          4 (spec:constant 0))))
(define i:drums:basic
  (i:drums (vector i:drum:hihat i:drum:bass i:drum:snare)))
```

```
(define snare-adsr
  (spec:adsr 'release
             1 (spec:constant 11)
             4 (spec:linear 11 6)
             8 (spec:linear 6 2)
             4 (spec:constant 0)))
(define i:drum:snare
  (i:drum/spec #:mode (spec:constant #f)
               #:period (spec:constant 7)
               #:volume snare-adsr))
(define beat:straight-rock
  (list (cons 0.125 1) (cons 0.125 0)
        (cons 0.125 2) (cons 0.125 0)
        (cons 0.125 1) (cons 0.125 0)
        (cons 0.125 2) (cons 0.125 0)))
```

Bithoven

Bithoven

```
(require data/enumerate)
```

Bithoven

```
(define time-sig/e
  (fin/e time-sig/ts:4:4 time-sig/ts:3:4))
```

Bithoven

```
(define time-sig->accents
  (hash
   time-sig/ts:4:4
   (list (accent-pattern "standard"  1 '(#t #f #f #f))
         (accent-pattern "on-beats"  2 '(#t #f #t #f))
         (accent-pattern "off-beats" 2 '(#f #t #f #t)))
   time-sig/ts:3:4
   (list (accent-pattern "waltz"  1 '(#t #f #f)))))
```

```
(define (accent-pattern/e ts)
  (apply fin/e (hash-ref time-sig->accents ts)))
```

Bithoven

```
(define form/e
  (fin/e
   (form "strophic"
         '((A . 1))
         '(A))
   (form "medley"
         '((A . 1) (B . 1) (C . 1) (D . 1))
         '(A B C D))
   (form "double medley"
         '((A . 1) (B . 1) (C . 1) (D . 1))
         '(A A B B C C D D))
(form "binary"
         '((A . 1) (B . 1))
         '(A B))
(form "double binary"
         '((A . 1) (B . 1))
         '(A A B B)
                  ....))
```

Bithoven

```
(define chord-progression/e
  (fin/e
   (progression '(0 3 4 4))
   (progression '(0 0 3 4))
   (progression '(0 3 0 4))
   (progression '(0 3 4 3))
   (progression '(0 2 4 4))
   (progression '(0 0 2 4))
   (progression '(0 2 0 4))
   (progression '(0 2 4 3))
   (progression '(0 3 4 2))
   (progression '(0 1 4))
   (progression '(1 4 0))
   (progression '(0 3 4))
   (progression '(0 5 3 4))
   (progression '(5 1 4 0))
   (progression '(0 4 0)) ....))
```

Bithoven

```
(define (chord-pulses/e pulse-count chord-count)
  (list-of-length-n-summing-to-k-with-no-zeros/e
   chord-count
   pulse-count))
```

Bithoven

```
(define (part/e ts ap cp measures len)
  (define cp-s (progression-seq cp))
  (define pulses
    (* len measures
       (accent-pattern-pulses-per-measure ap)))
  (define cp/e
    (chord-pulses/e
     pulses
     (length cp-s)))
  (dep/e
#:one-way? #f
#:flat? #t
#:f-range-finite? #t
cp/e
(λ (cps)
  (traverse/e
   (λ (cp)
     (rhythm/e
      ts
      (* cp (accent-pattern-notes-per-pulse ap))))
   cps))))
```

Bithoven

```
(define bithoven/e
  (vector/e
   (dep/e
    #:one-way? #f
    #:flat? #t
    #:f-range-finite? #t
    time-sig/e
    (λ (ts)
      (dep/e
       #:one-way? #f
       #:flat? #t
       #:f-range-finite? #t
       (accent-pattern/e ts)
       (λ (ap)
         (dep/e
          #:one-way? #f
          #:flat? #t
          #:f-range-finite? #t
          (cons/e form/e chord-progression/e)
          (λ (f*cp)
            (match-define (cons f cp) f*cp)
            (define cp-s (progression-seq cp))
            (define measures-per-part
              (*
               (let ()
                 (ceiling
                  (/ (length cp-s)
                     (accent-pattern-pulses-per-measure ap))))
               (let ()
                 (define pat-length (length (form-pattern f)))
                 (cond
                   [(< pat-length 3) 4]
                   [(< pat-length 5) 2]
                   [else 1])))))
            (define (this-kind-of-part/e len)
              (part/e ts ap cp measures-per-part len))
            (traverse/e
             (λ (p) (this-kind-of-part/e (cdr p)))
             (form-part-lens f)))))))))
  bass-notes/e))
```

Bithoven

1703493744758923887529249104888098642334867667726987451573478883541120829669434107094796571446304692813537082418532624995122150239327473124672517141053289956448792938868933533218648351214565581695124587274339009253612993279292174177459719

Bithoven

```
'#((4 . 0.25)
   (#t #f #f #f)
   (A B C D)
   #hasheq((C
            .
            (((0.25 ((2 . 0) (2 . 0) (0 . 0)) . #t)
              (0.25 ((0 . 0) (4 . 0) (0 . 0)) . #f)
              (0.25 ((4 . 0) (0 . 0) (4 . 0)) . #f)
              (0.25 ((0 . 0) (0 . 0) (0 . 0)) . #f))
             ((0.25 ((0 . 0) (2 . 0) (0 . 0)) . #t)
              (0.25 ((4 . 0) (0 . 0) (0 . 0)) . #f)
              (0.25 ((0 . 0) (0 . 0) (0 . 0)) . #f)
              (0.25 ((0 . 0) (2 . 0) (0 . 0)) . #f))
             ((0.25 ((3 . 0) (5 . 0) (3 . 0)) . #t)
              (0.25 ((1 . 0) (5 . 0) (3 . 0)) . #f)
              (0.25 ((1 . 0) (3 . 0) (3 . 0)) . #f)
              (0.25 ((1 . 0) (3 . 0) (3 . 0)) . #f))
             ((0.25 ((5 . 0) (1 . 0) (3 . 0)) . #t)
              (0.25 ((5 . 0) (1 . 0) (3 . 0)) . #f)
              (0.25 ((3 . 0) (3 . 0) (3 . 0)) . #f)
              (0.25 ((3 . 0) (5 . 0) (3 . 0)) . #f))
             ((0.25 ((3 . 0) (5 . 0) (3 . 0)) . #t)
              (0.25 ((1 . 0) (1 . 0) (3 . 0)) . #f)
              (0.25 ((1 . 0) (5 . 0) (3 . 0)) . #f)
              (0.25 ((3 . 0) (3 . 0) (3 . 0)) . #f))
             ((0.25 ((2 . 0) (6 . 0) (4 . 0)) . #t)
              (0.25 ((2 . 0) (4 . 0) (4 . 0)) . #f)
              (0.25 ((2 . 0) (2 . 0) (4 . 0)) . #f)
              (0.25 ((4 . 0) (6 . 0) (4 . 0)) . #f))
             ((0.25 ((5 . 0) (5 . 0) (3 . 0)) . #t)
              (0.25 ((0 . 1) (3 . 0) (0 . 0)) . #f)
              (0.25 ((0 . 1) (3 . 0) (0 . 0)) . #f)
              (0.25 ((3 . 0) (0 . 1) (0 . 0)) . #f))
             ((0.25 ((0 . 1) (0 . 1) (0 . 0)) . #t)
              (0.25 ((5 . 0) (3 . 0) (0 . 0)) . #f)
              (0.25 ((0 . 1) (3 . 0) (3 . 0)) . #f)
              (0.25 ((5 . 0) (5 . 0) (3 . 0)) . #f))
             ((0.25 ((5 . 0) (3 . 0) (0 . 0)) . #t)
              (0.25 ((5 . 0) (5 . 0) (3 . 0)) . #f)
              (0.25 ((0 . 1) (5 . 0) (0 . 0)) . #f)
              (0.25 ((5 . 0) (5 . 0) (0 . 0)) . #f))
             ((0.25 ((6 . 0) (4 . 0) (4 . 0)) . #t)
              (0.25 ((1 . 1) (6 . 0) (4 . 0)) . #f)
              (0.25 ((4 . 0) (6 . 0) (4 . 0)) . #f)
              (0.25 ((4 . 0) (6 . 0) (4 . 0)) . #f))))
   (A
    .
    (((0.25 ((2 . 0) (2 . 0) (0 . 0)) . #t)
      (0.25 ((2 . 0) (4 . 0) (4 . 0)) . #f)
      (0.25 ((0 . 0) (0 . 0) (4 . 0)) . #f)
      (0.25 ((0 . 0) (2 . 0) (4 . 0)) . #f))
     ((0.25 ((4 . 0) (2 . 0) (0 . 0)) . #t)
      (0.25 ((2 . 0) (2 . 0) (4 . 0)) . #f)
      (0.25 ((0 . 0) (2 . 0) (0 . 0)) . #f)
      (0.25 ((4 . 0) (4 . 0) (0 . 0)) . #f))
     ((0.25 ((5 . 0) (5 . 0) (3 . 0)) . #t)
      (0.25 ((0 . 1) (3 . 0) (0 . 0)) . #f)
      (0.25 ((0 . 1) (3 . 0) (0 . 0)) . #f)
      (0.25 ((3 . 0) (0 . 1) (0 . 0)) . #f))
     ((0.25 ((0 . 1) (0 . 1) (0 . 0)) . #t)
      (0.25 ((5 . 0) (3 . 0) (0 . 0)) . #f)
      (0.25 ((0 . 1) (3 . 0) (3 . 0)) . #f)
      (0.25 ((5 . 0) (5 . 0) (3 . 0)) . #f))
     ((0.25 ((5 . 0) (3 . 0) (0 . 0)) . #t)
      (0.25 ((5 . 0) (5 . 0) (3 . 0)) . #f)
      (0.25 ((0 . 1) (5 . 0) (0 . 0)) . #f)
      (0.25 ((5 . 0) (5 . 0) (0 . 0)) . #f))
     ((0.25 ((6 . 0) (4 . 0) (4 . 0)) . #t)
      (0.25 ((1 . 1) (6 . 0) (4 . 0)) . #f)
      (0.25 ((4 . 0) (6 . 0) (4 . 0)) . #f)
      (0.25 ((4 . 0) (6 . 0) (4 . 0)) . #f))))
   (B
    .
    (((0.25 ((1 . 0) (1 . 0) (3 . 0)) . #f)
      (0.25 ((3 . 0) (5 . 0) (3 . 0)) . #f))
     ((0.25 ((4 . 0) (6 . 0) (4 . 0)) . #t)
      (0.25 ((4 . 0) (2 . 0) (4 . 0)) . #f)
      (0.25 ((2 . 0) (6 . 0) (4 . 0)) . #f)
      (0.25 ((6 . 0) (2 . 0) (4 . 0)) . #f))
     ((0.25 ((3 . 0) (3 . 0) (0 . 0)) . #t)
      (0.25 ((3 . 0) (5 . 0) (0 . 0)) . #f)
      (0.25 ((0 . 1) (3 . 0) (0 . 0)) . #f)
      (0.25 ((5 . 0) (0 . 1) (0 . 0)) . #f))
     ((0.25 ((0 . 1) (5 . 0) (3 . 0)) . #t)
      (0.25 ((5 . 0) (5 . 0) (3 . 0)) . #f)
      (0.25 ((5 . 0) (3 . 0) (0 . 0)) . #f)
      (0.25 ((5 . 0) (3 . 0) (0 . 0)) . #f))
     ((0.25 ((3 . 0) (3 . 0) (3 . 0)) . #t)
      (0.25 ((5 . 0) (5 . 0) (3 . 0)) . #f)
      (0.25 ((3 . 0) (3 . 0) (3 . 0)) . #f)
      (0.25 ((1 . 0) (5 . 0) (3 . 0)) . #f))
     ((0.25 ((5 . 0) (5 . 0) (3 . 0)) . #t)
      (0.25 ((1 . 0) (1 . 0) (3 . 0)) . #f)
      (0.25 ((3 . 0) (1 . 0) (3 . 0)) . #f)
      (0.25 ((5 . 0) (5 . 0) (3 . 0)) . #f))
     ((0.25 ((3 . 0) (3 . 0) (3 . 0)) . #t)
      (0.25 ((3 . 0) (5 . 0) (3 . 0)) . #f)
      (0.25 ((3 . 0) (1 . 0) (3 . 0)) . #f)
      (0.25 ((1 . 0) (5 . 0) (3 . 0)) . #f))
     ((0.25 ((6 . 0) (4 . 0) (4 . 0)) . #t)
      (0.25 ((6 . 0) (6 . 0) (4 . 0)) . #f)
      (0.25 ((6 . 0) (2 . 0) (4 . 0)) . #f)
      (0.25 ((4 . 0) (2 . 0) (4 . 0)) . #f))
     ((0.25 ((4 . 0) (6 . 0) (4 . 0)) . #t)
      (0.25 ((4 . 0) (6 . 0) (4 . 0)) . #f)
      (0.25 ((2 . 0) (2 . 0) (4 . 0)) . #f)
      (0.25 ((6 . 0) (6 . 0) (4 . 0)) . #f))
     ((0.25 ((2 . 0) (4 . 0) (4 . 0)) . #t)
      (0.25 ((6 . 0) (2 . 0) (4 . 0)) . #f)
      (0.25 ((6 . 0) (4 . 0) (4 . 0)) . #f)
      (0.25 ((1 . 1) (4 . 0) (4 . 0)) . #f)
      (0.25 ((4 . 0) (4 . 0) (4 . 0)) . #f))))
   (D
    .
    (((0.25 ((4 . 0) (4 . 0) (4 . 0)) . #t)
      (0.25 ((0 . 0) (4 . 0) (4 . 0)) . #f)
      (0.25 ((2 . 0) (4 . 0) (0 . 0)) . #f)
      (0.25 ((2 . 0) (0 . 0) (0 . 0)) . #f))
     ((0.25 ((3 . 0) (3 . 0) (3 . 0)) . #t)
      (0.25 ((5 . 0) (5 . 0) (3 . 0)) . #f)
      (0.25 ((3 . 0) (3 . 0) (3 . 0)) . #f)
      (0.25 ((1 . 0) (5 . 0) (3 . 0)) . #f))
     ((0.25 ((5 . 0) (5 . 0) (3 . 0)) . #t)
      (0.25 ((1 . 0) (1 . 0) (3 . 0)) . #f)
      (0.25 ((3 . 0) (1 . 0) (3 . 0)) . #f)
      (0.25 ((5 . 0) (5 . 0) (3 . 0)) . #f))
     ((0.25 ((3 . 0) (3 . 0) (3 . 0)) . #t)
      (0.25 ((3 . 0) (5 . 0) (3 . 0)) . #f)
      (0.25 ((3 . 0) (1 . 0) (3 . 0)) . #f)
      (0.25 ((1 . 0) (5 . 0) (3 . 0)) . #f))
     ((0.25 ((6 . 0) (4 . 0) (4 . 0)) . #t)
      (0.25 ((6 . 0) (6 . 0) (4 . 0)) . #f)
      (0.25 ((6 . 0) (2 . 0) (4 . 0)) . #f)
      (0.25 ((4 . 0) (2 . 0) (4 . 0)) . #f))
     ((0.25 ((4 . 0) (6 . 0) (4 . 0)) . #t)
      (0.25 ((4 . 0) (6 . 0) (4 . 0)) . #f)
      (0.25 ((2 . 0) (2 . 0) (4 . 0)) . #f)
      (0.25 ((6 . 0) (6 . 0) (4 . 0)) . #f))
     ((0.25 ((2 . 0) (4 . 0) (4 . 0)) . #t)
      (0.25 ((6 . 0) (2 . 0) (4 . 0)) . #f)
      (0.25 ((6 . 0) (4 . 0) (4 . 0)) . #f)
      (0.25 ((1 . 1) (4 . 0) (4 . 0)) . #f)
      (0.25 ((4 . 0) (4 . 0) (4 . 0)) . #f))))
   (E
    .
    (((0.25 ((6 . 0) (6 . 0) (4 . 0)) . #f)
      (0.25 ((2 . 0) (6 . 0) (4 . 0)) . #f))
     ((0.25 ((5 . 0) (0 . 1) (3 . 0)) . #t)
      (0.25 ((3 . 0) (0 . 1) (0 . 0)) . #f)
      (0.25 ((0 . 1) (5 . 0) (0 . 0)) . #f)
      (0.25 ((0 . 1) (3 . 0) (3 . 0)) . #f))
     ((0.25 ((5 . 0) (3 . 0) (0 . 0)) . #t)
      (0.25 ((5 . 0) (3 . 0) (0 . 0)) . #f)
      (0.25 ((3 . 0) (3 . 0) (3 . 0)) . #f)
      (0.25 ((0 . 1) (5 . 0) (3 . 0)) . #f))
     ((0.25 ((6 . 0) (4 . 0) (4 . 0)) . #t)
      (0.25 ((1 . 1) (4 . 0) (4 . 0)) . #f)
      (0.25 ((4 . 0) (6 . 0) (4 . 0)) . #f)
      (0.25 ((1 . 1) (6 . 0) (4 . 0)) . #f))))
   (B
    .
    (((0.25 ((0 . 0) (2 . 0) (4 . 0)) . #t)
      (0.25 ((2 . 0) (4 . 0) (0 . 0)) . #f)
      (0.25 ((0 . 0) (0 . 0) (4 . 0)) . #f)
      (0.25 ((0 . 0) (2 . 0) (4 . 0)) . #f))
     ((0.25 ((2 . 0) (4 . 0) (0 . 0)) . #t)
      (0.25 ((2 . 0) (0 . 0) (4 . 0)) . #f)
      (0.25 ((0 . 0) (2 . 0) (0 . 0)) . #f)
      (0.25 ((0 . 0) (0 . 0) (0 . 0)) . #f))
     ((0.25 ((2 . 0) (2 . 0) (0 . 0)) . #t)
      (0.25 ((2 . 0) (4 . 0) (0 . 0)) . #f)
      (0.25 ((0 . 0) (4 . 0) (0 . 0)) . #f)
      (0.25 ((4 . 0) (2 . 0) (0 . 0)) . #f))
     ((0.25 ((3 . 0) (1 . 0) (3 . 0)) . #t)
      (0.25 ((1 . 0) (3 . 0) (3 . 0)) . #f)
      (0.25 ((1 . 0) (3 . 0) (3 . 0)) . #f)
      (0.25 ((1 . 0) (5 . 0) (3 . 0)) . #f))
     ((0.25 ((5 . 0) (5 . 0) (3 . 0)) . #t)
      (0.25 ((3 . 0) (1 . 0) (3 . 0)) . #f)
      (0.25 ((1 . 0) (1 . 0) (3 . 0)) . #f)
      (0.25 ((5 . 0) (3 . 0) (3 . 0)) . #f))
     ((0.25 ((2 . 0) (6 . 0) (4 . 0)) . #t)
      (0.25 ((4 . 0) (4 . 0) (4 . 0)) . #f)
      (0.25 ((2 . 0) (6 . 0) (4 . 0)) . #f)
      (0.25 ((2 . 0) (4 . 0) (4 . 0)) . #f))
     ((0.25 ((3 . 0) (5 . 0) (0 . 0)) . #t)
      (0.25 ((3 . 0) (5 . 0) (0 . 0)) . #f)
      (0.25 ((3 . 0) (0 . 1) (0 . 0)) . #f)
      (0.25 ((0 . 1) (0 . 1) (3 . 0)) . #f))
     ((0.25 ((4 . 0) (4 . 0) (4 . 0)) . #t)
      (0.25 ((1 . 1) (4 . 0) (4 . 0)) . #f)
      (0.5 ((1 . 1) (6 . 0) (4 . 0)) . #f))
     ((0.25 ((4 . 0) (4 . 0) (4 . 0)) . #t)
      (0.25 ((6 . 0) (4 . 0) (4 . 0)) . #f)
      (0.25 ((4 . 0) (4 . 0) (4 . 0)) . #f)
      (0.25 ((6 . 0) (1 . 1) (4 . 0)) . #f))))))
```

Bithoven

```
(vector/e
 tone-names/e scales/e tempo/e
 pulse1/e pulse2/e triangle/e drums/e
 mhb/e
 (fin/e 2 3) (fin/e 1 2) (fin/e 1 2)
 (hash-traverse/e
  #:get-contract
  (λ (x)
    (listof
     (cons/c real? exact-nonnegative-integer?)))
  (λ (_) (drum-measure/e ts ap))
  parts)
(hash-traverse/e
 #:get-contract
 (λ (x)
   (listof exact-nonnegative-integer?))
 (λ (ms)
   (dep/e
    rest-n/e
    #:f-range-finite? #t
    (λ (rest-n)
      (if rest-n
          (listof-n/e
           (below/e rest-n)
           (add1 (ceiling (/ (length (append* ms)) rest-n))))
          (single/e '())))))
 parts))
```

Bithoven

2767

0481

7745

6027

4624

8553

7556

5375

7320

```
'#(C
   #<procedure:scale-diatonic-major>
   160
   #<procedure:...n/instrument.rkt:155:2>
   #<procedure:...n/instrument.rkt:155:2>
   #<procedure:...n/instrument.rkt:169:2>
   #<procedure:...n/instrument.rkt:191:2>
   (0 1 2)
   3
   1
   1
   #hash((C
          .
          ((0.125 . 1)
           (0.125 . 0)
           (0.125 . 2)
           (0.125 . 0)
           (0.125 . 1)
           (0.125 . 1)
           (0.125 . 0)
           (0.125 . 2)))
         (A
          .
          ((0.0625 . 1)
           (0.0625 . 0)
           (0.0625 . 2)
           (0.0625 . 0)
           (0.0625 . 1)
           (0.0625 . 0)
           (0.0625 . 2)
           (0.0625 . 0)
           (0.0625 . 2)
           (0.0625 . 0)))
         (D
          .
          ((0.125 . 1)
           (0.125 . 0)
           (0.125 . 2)
           (0.125 . 0)
           (0.125 . 1)
           (0.125 . 0)
           (0.125 . 2)
           (0.125 . 0)))
         (B
          .
          ((0.125 . 1)
           (0.125 . 0)
           (0.125 . 2)
           (0.125 . 0)
           (0.125 . 1)
           (0.125 . 0)
           (0.125 . 2)
           (0.125 . 0))))
   #hash((C . (4 3 3 0 3 2 0 3 3 1 1 2))
         (A . (6 5 1 3 5 3 1 5 4))
         (D . (6 5 4 3 5 1 4 3 2))
         (B . (5 0 2 4 0 2 4 3 4 0))))
```

Bithoven

Bithoven

The

Get Bonus

infinite entertainment system

Level 1

Ricoh RP2A03

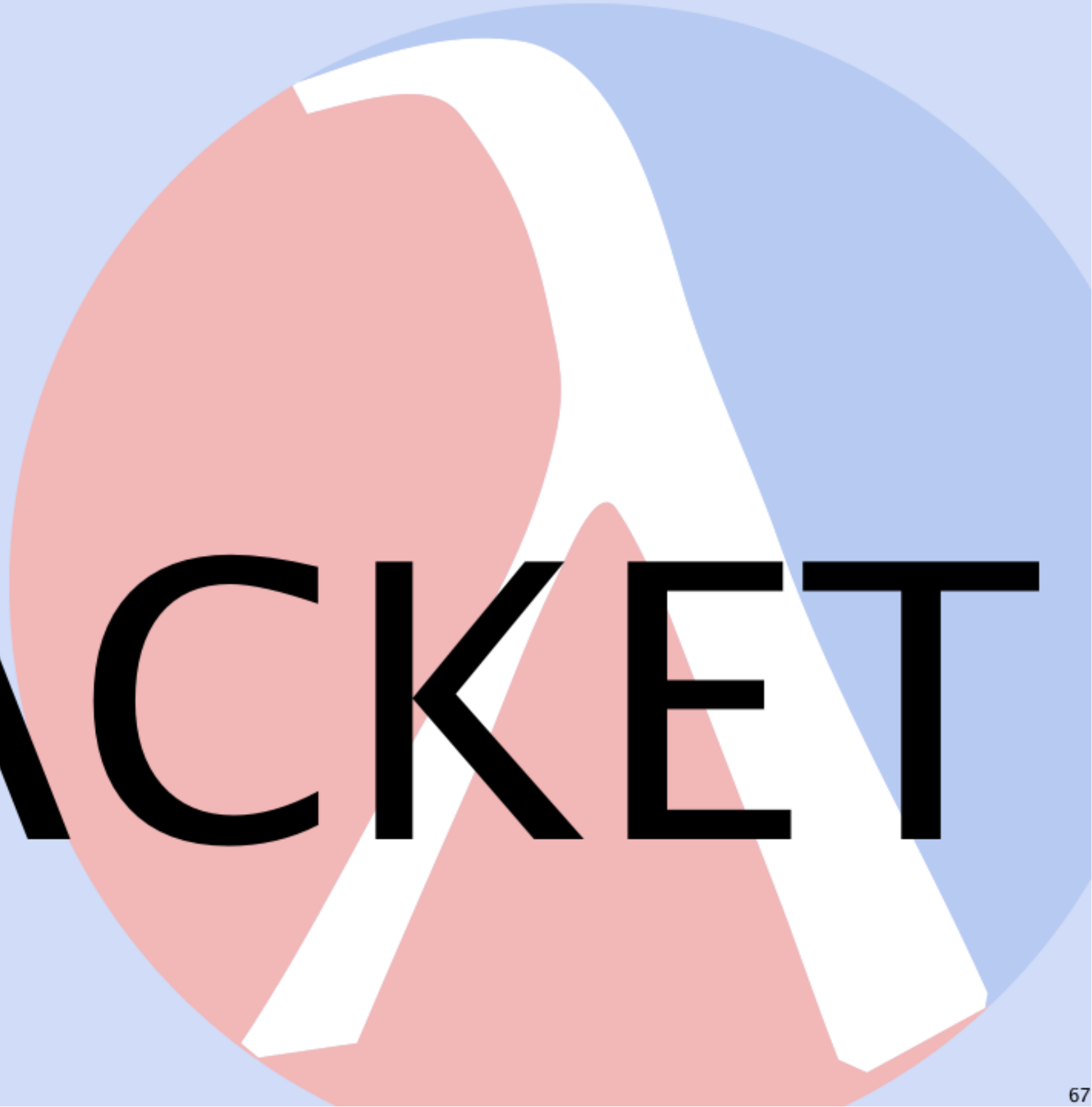
Level 2

NES Chamber Orchestra

Level 3

Bithoven

Now You're Playing With



RACKET