Graphics & Simulations *(& Games)*, Oh My!

Setting Expectations

50% Show & Tell

50% Tutorial

Medium Code

All Levels of Developer

~180 slides / 35 min = ~5 spm

Ryan Davis

Coding professionally 27 years, 18 years Ruby.

Founder, Seattle.rb; First & Oldest Ruby Brigade.

Author: minitest, flog, flay, debride, ruby_parser, etc.

Pushed over 1000 gems into the Ruby Ecosystem.

Developer's Developer: I ❤️ building tools.

Run Seattle.rb Consulting http://seattlerb.com/

I'm available for hire.

What is Graphics?

gem install graphics --pre

Simple Framework

Simulations

Art

Visualizations

Games

Grade-School Maths

NOT Game Programming Conventions

Demystify

New Tool in Your Toolbox

WARNING:

I Suck at Art

Aesthetically Challenged

Maths = OK

!scientist

!game_dev

Just Like You!

Damn Good Tool Dev

graphics

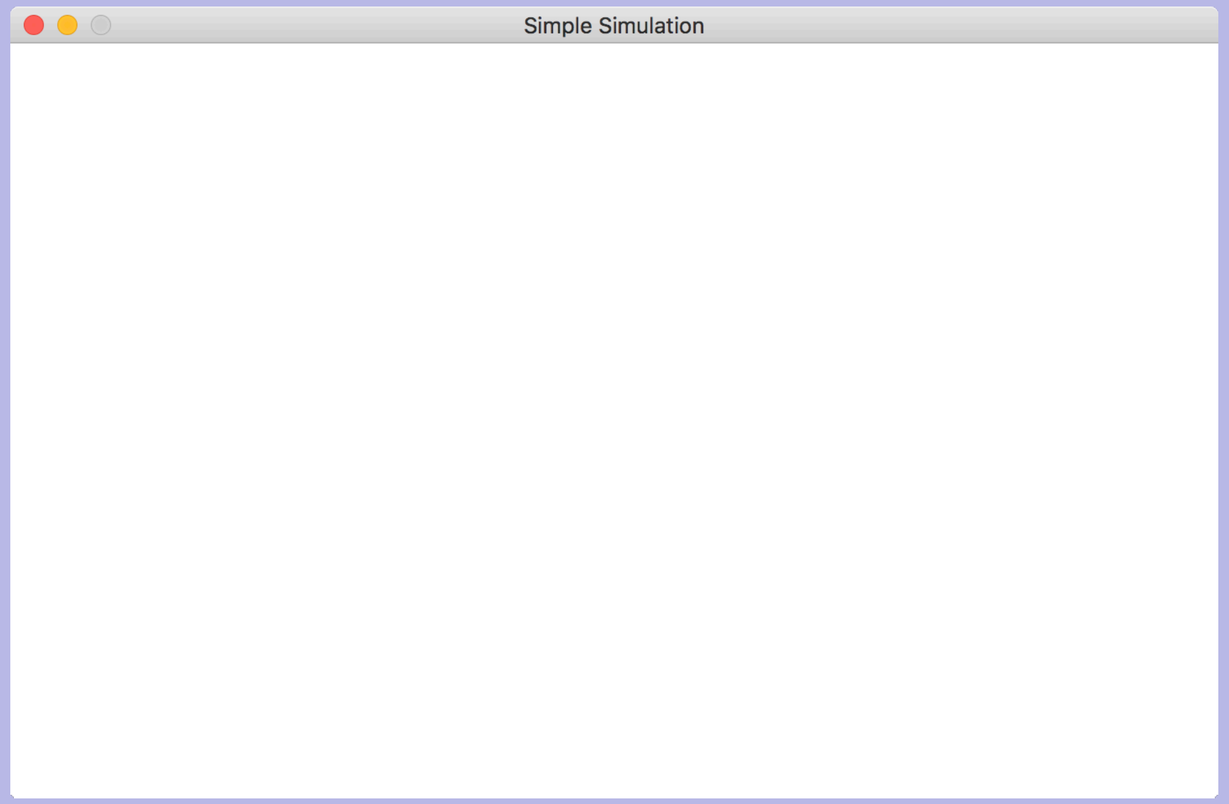
=

simple

Jumping Straight In

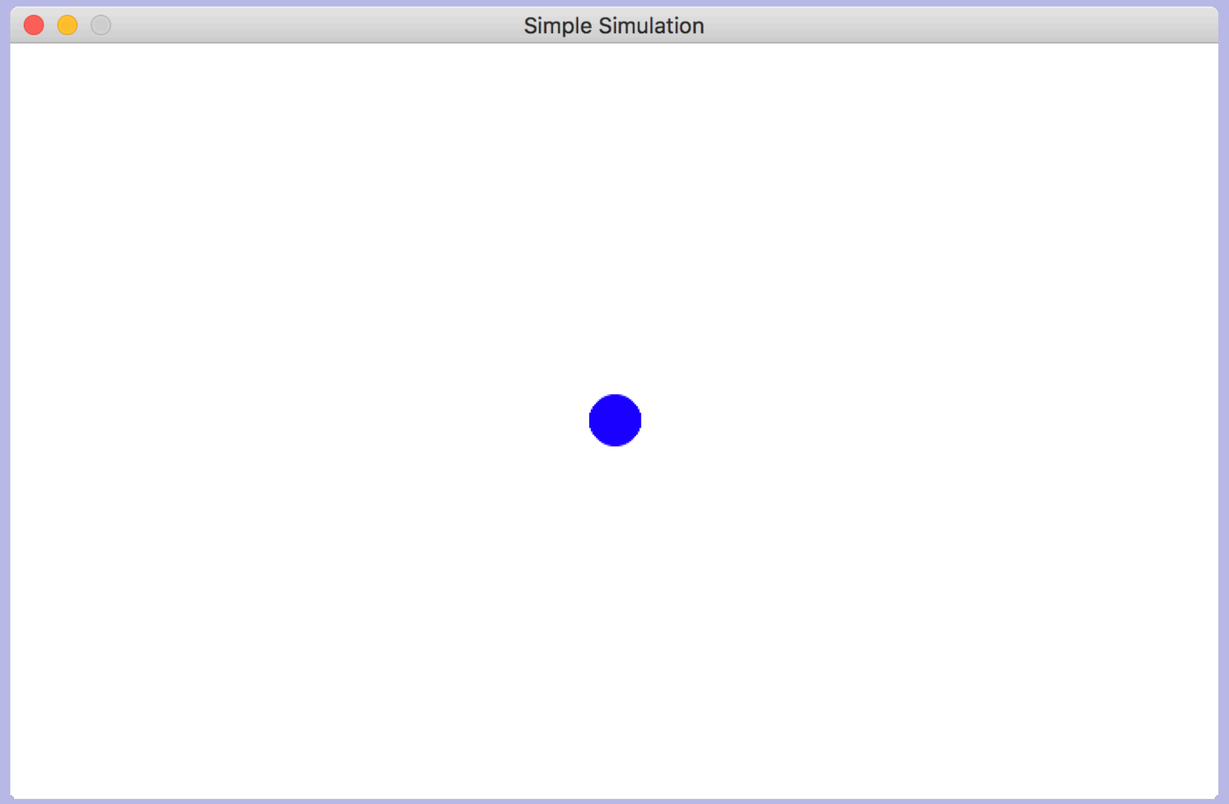
Totally Blank Canvas

```
require "graphics"

class SimpleSimulation < G::S
  include WhiteBackground
end

SimpleSimulation.new.run
```



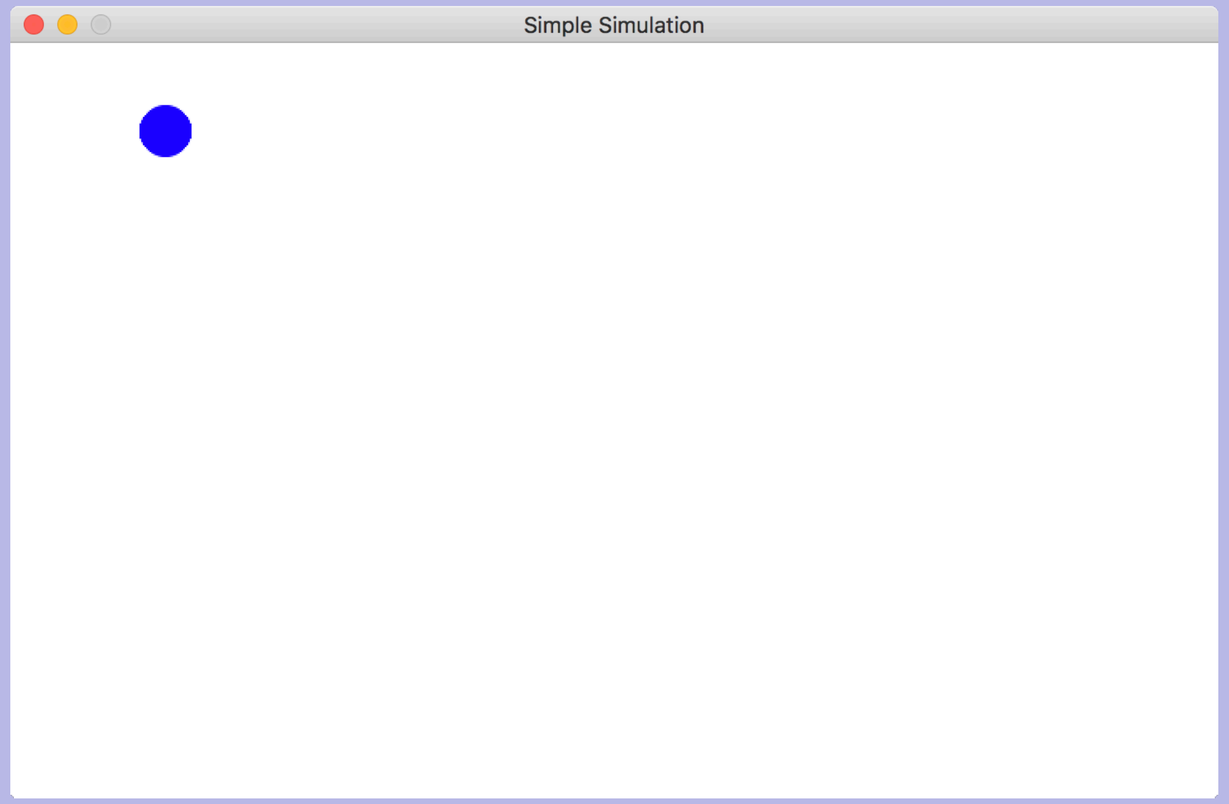
Drawing a Circle

```
# class SimpleSimulation
def draw n
  super
  circle w/2, h/2, 15, :blue, :filled
end
```



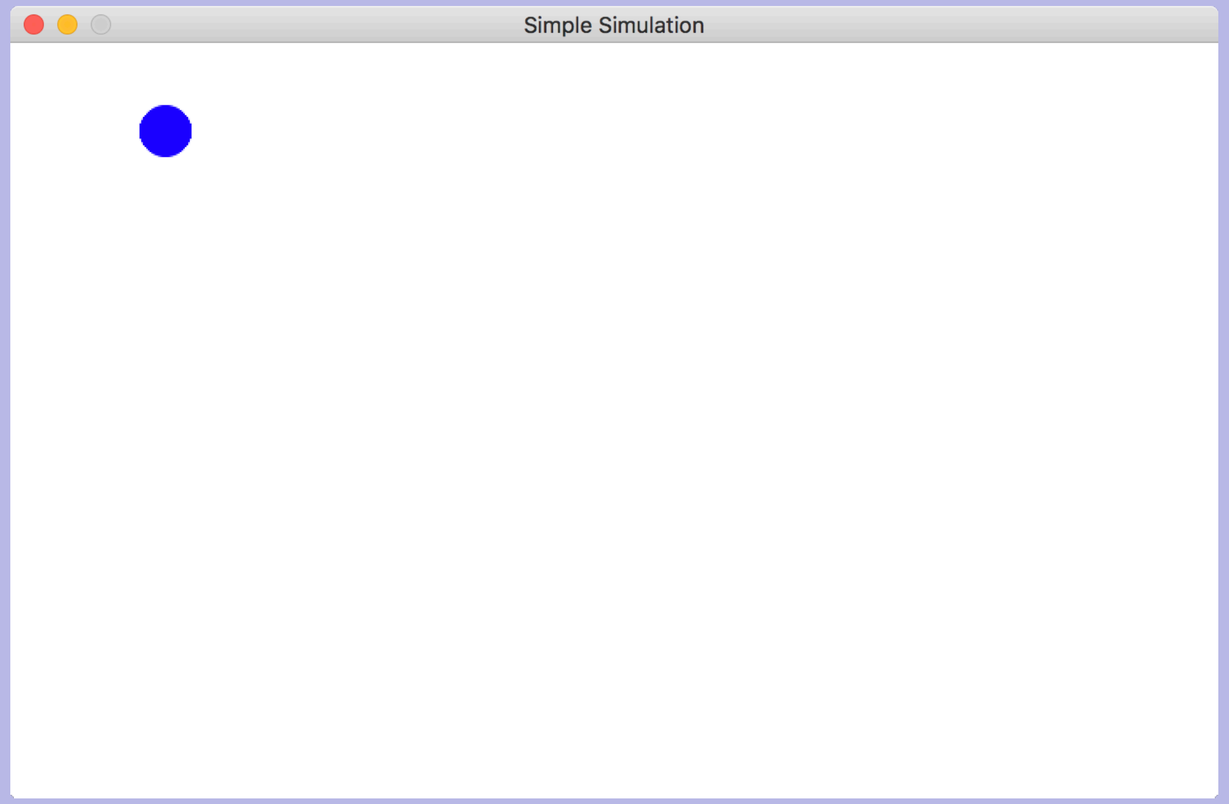
Drawing a Circle, via Body

```
class Ball < Graphics::Body
  COUNT = 1

  class View
    def self.draw w, b
      w.circle b.x, b.y, 15, :blue, :filled
    end
  end
end
```



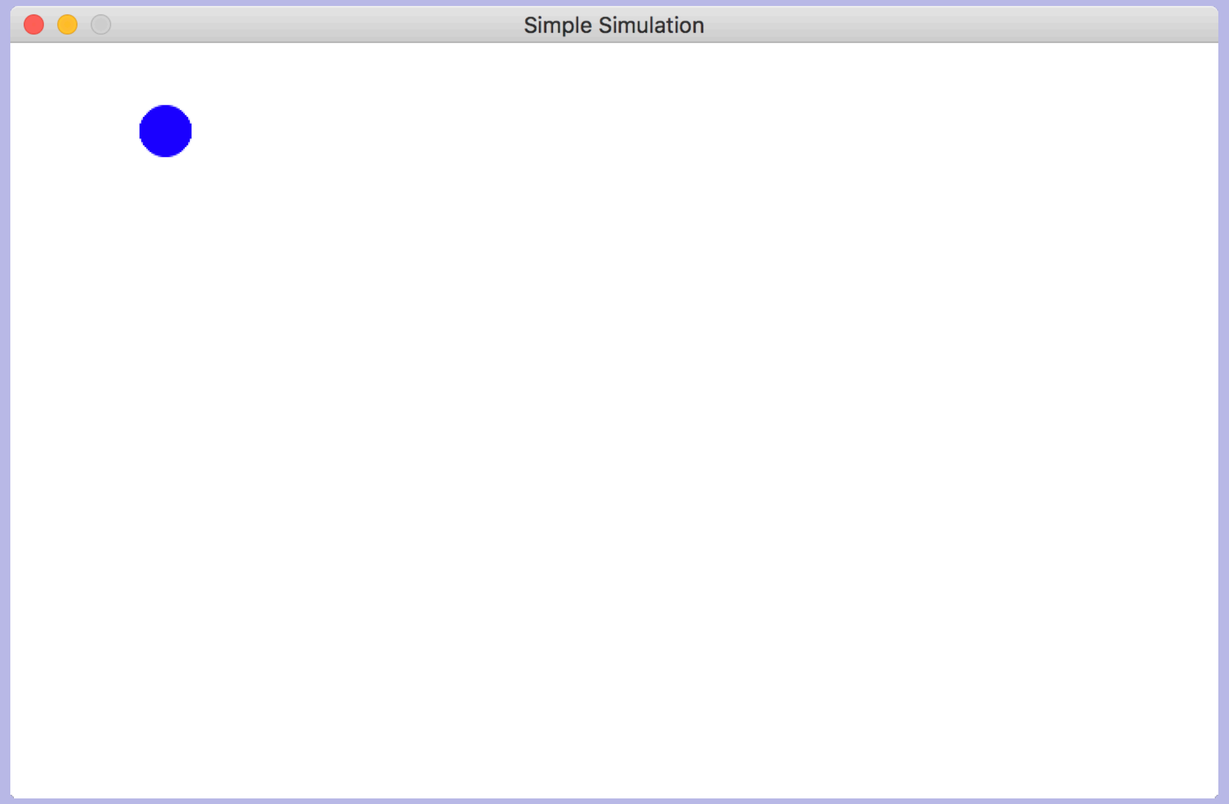
Using a Body

```
# class SimpleSimulation
def initialize
  super

  register_bodies populate Ball
end
```



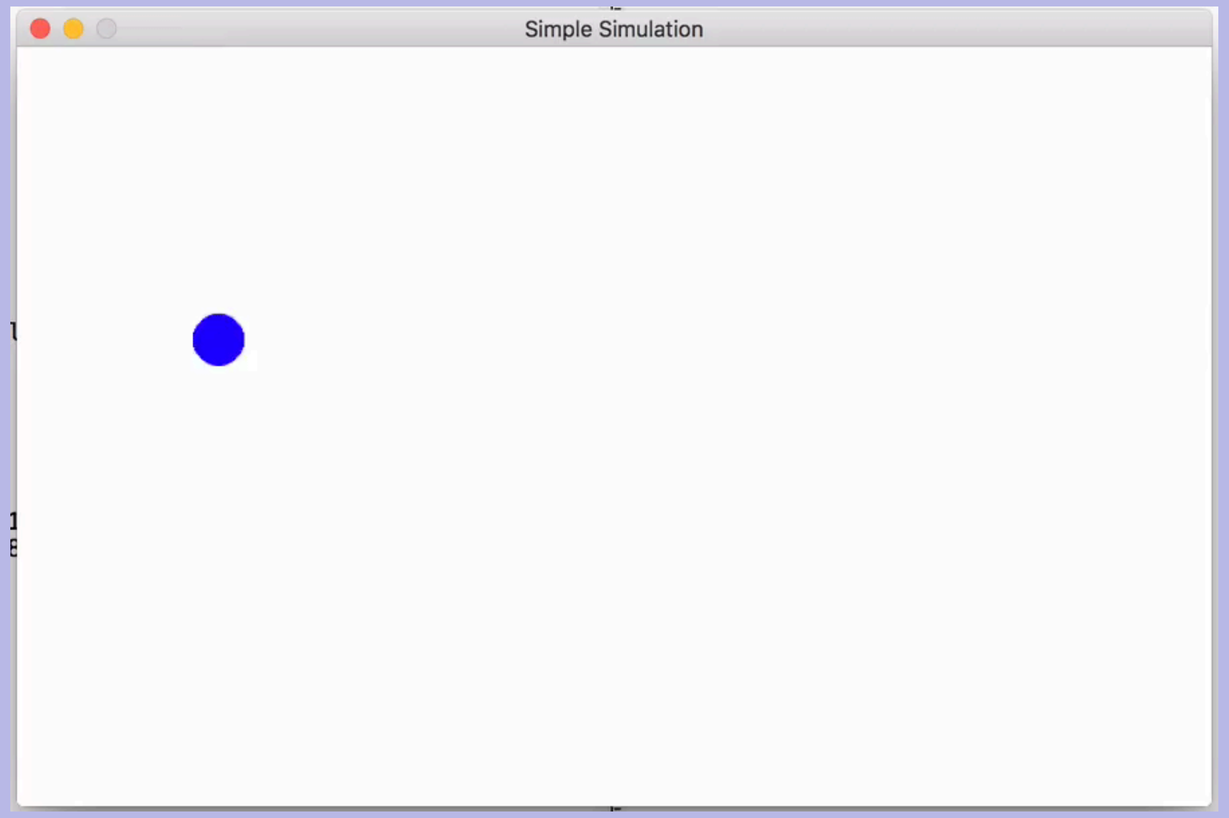
Adding Behavior

```
# class Ball
def initialize w
  super

  self.a = random_angle
  self.m = rand 5..15
end
```



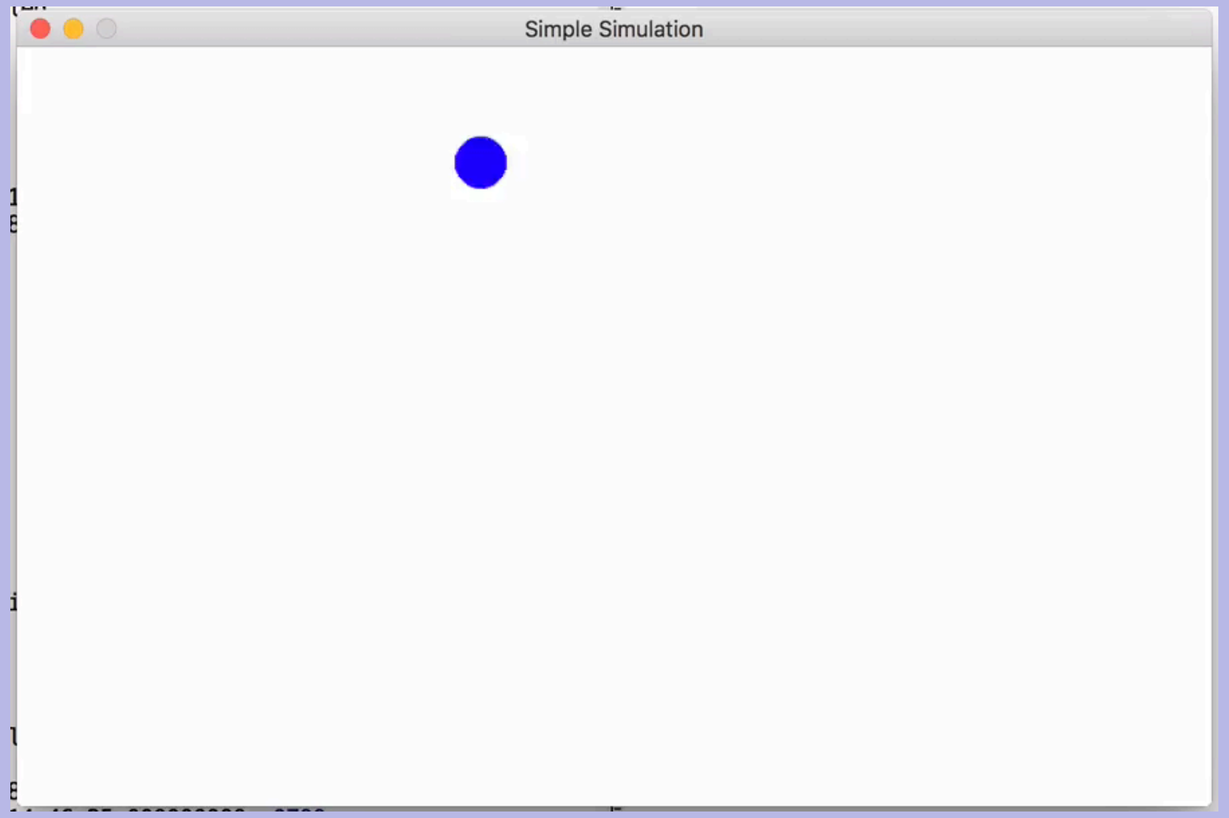
Moving the body—asteroids

```
# class Ball
def update
  move
  wrap
end
```



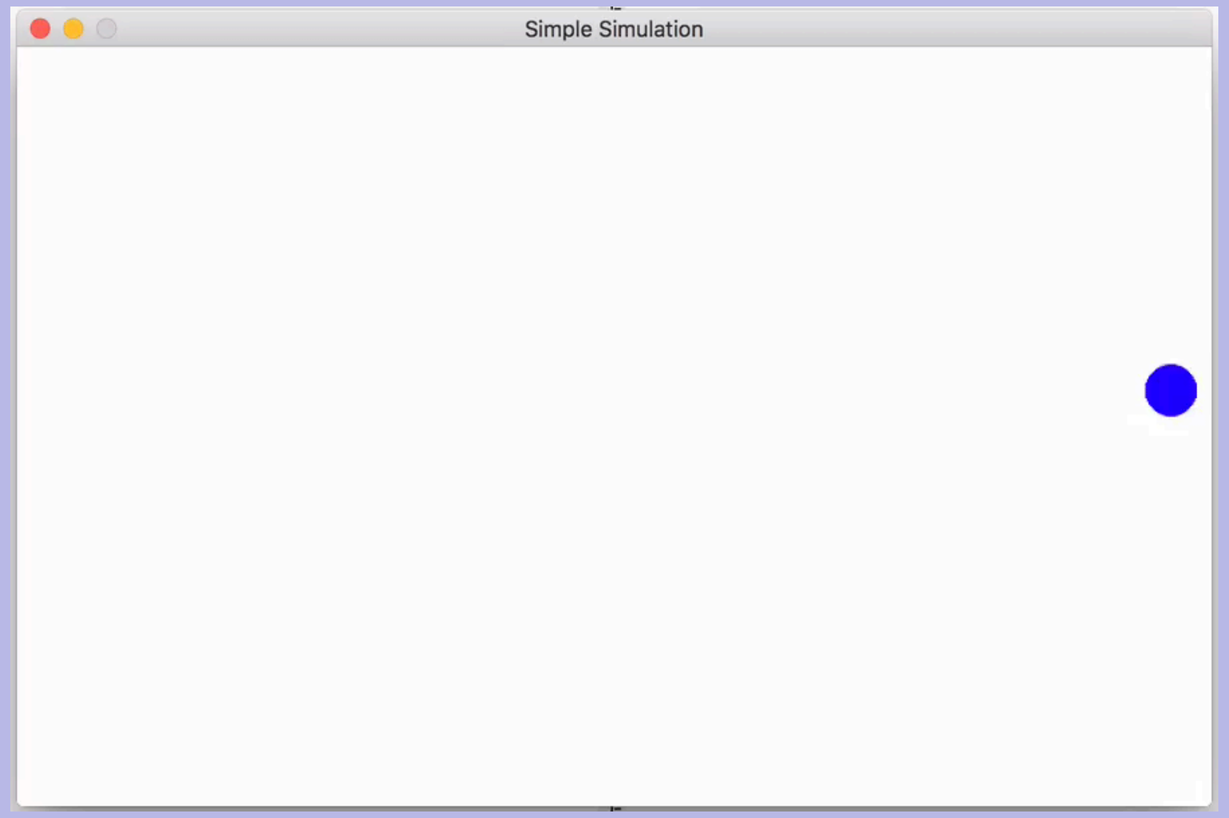
Moving the body—pong

```
# class Ball
def update
  move
  bounce 0 # (0 == no friction)
end
```



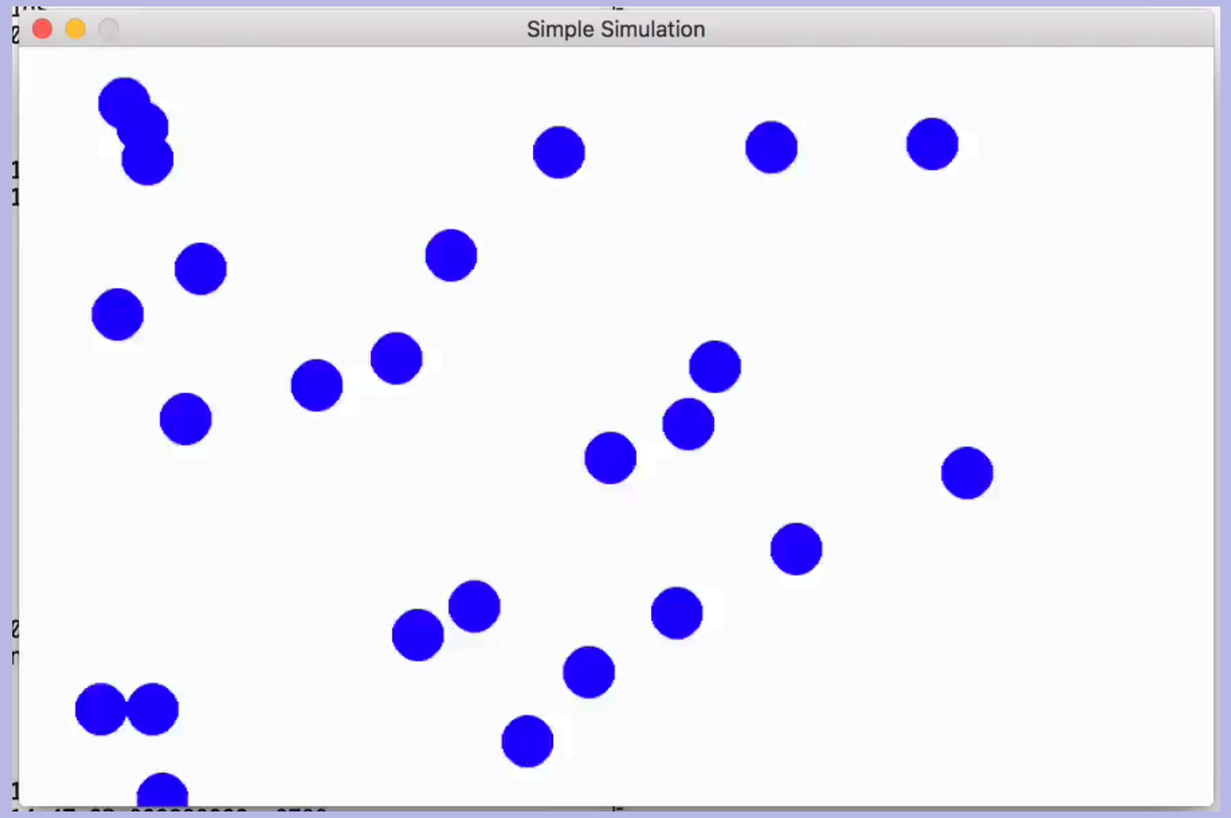
Moving the body—bouncing

```
# class Ball
G = V[0, -0.3]

def update
  self.velocity += G
  move
  bounce
end
```



Many bodies

```
# class Ball
COUNT = 25
```



Final Version

```
require "graphics"

class SimpleSimulation < Graphics::Simulation
  include WhiteBackground

  def initialize
    super

    register_bodies populate Ball
  end
end
# …
SimpleSimulation.new.run
```

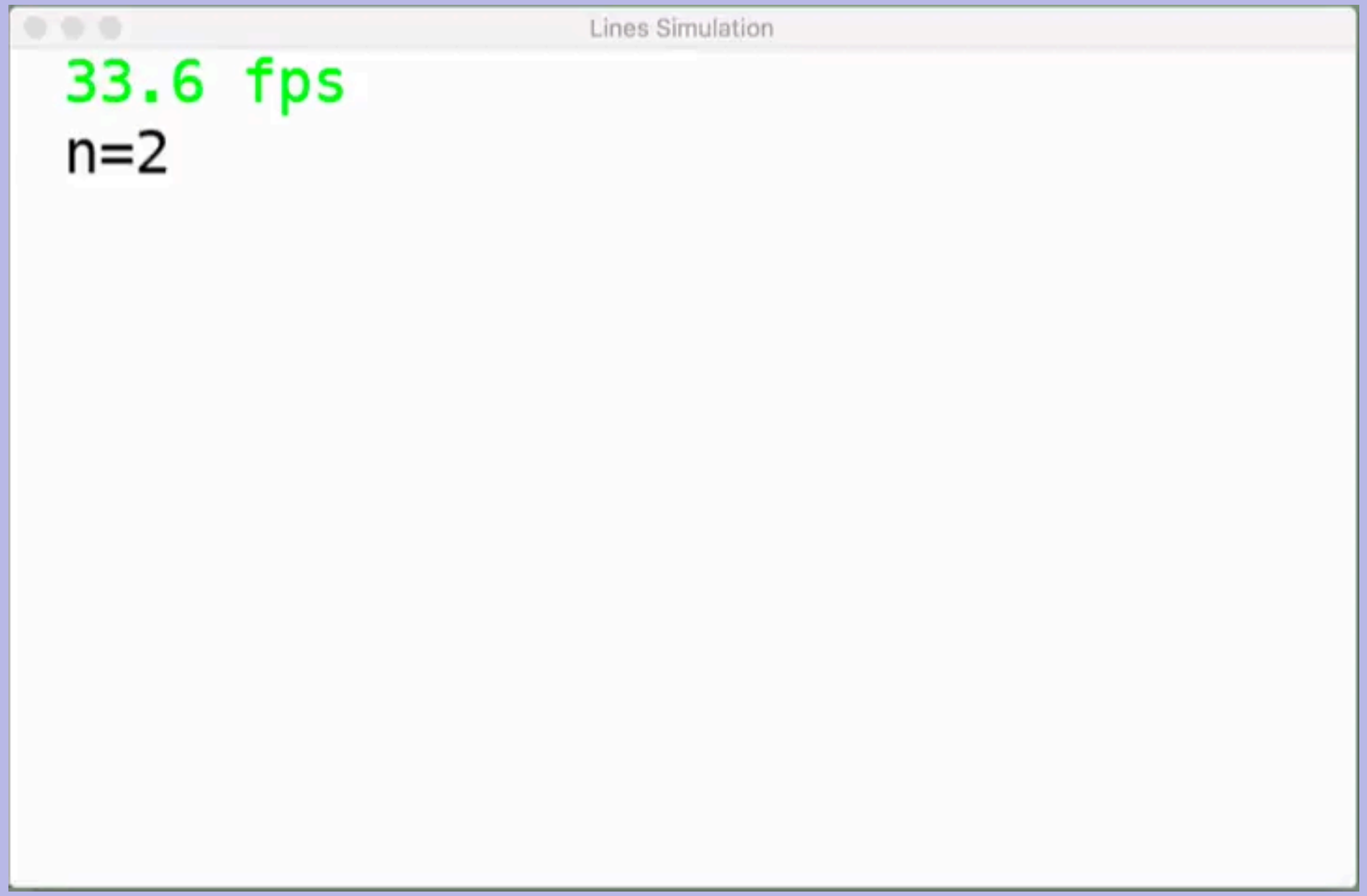
```
class Ball < Graphics::Body
  COUNT = 25
  G = V[0, -18 / 60.0]

  def initialize w
    super

    self.a = random_angle
    self.m = rand 5..15
  end

  def update
    self.velocity += G
    move
    bounce
  end
end
```

```
class Ball
  class View
    def self.draw w, b
      w.circle b.x, b.y, 15, :blue, :filled
    end
  end
end
```

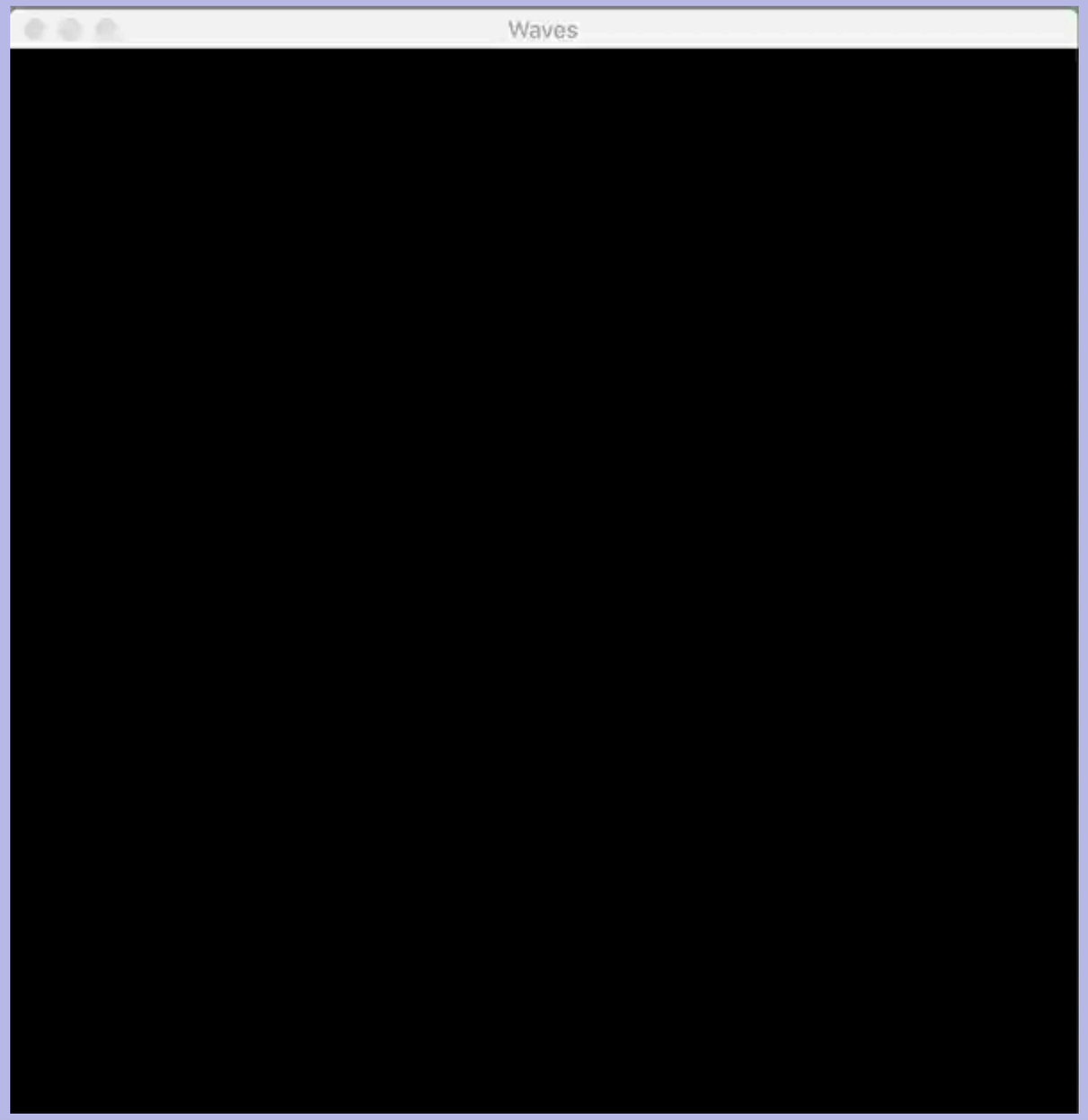
```
require "graphics"

class SimpleSimulation < G::S
  include WhiteBackground

  def initialize
    super
    register_bodies populate Ball
  end

  class Ball < Graphics::Body
    COUNT = 25
    G = V[0, -18 / 60.0]

    def initialize w
      super
      self.a = random_angle
      self.m = rand 5..15
    end

    def update
      self.velocity += G
      move
      bounce
    end

    class View
      def self.draw w, b
        w.circle b.x, b.y,
          15, :blue, :filled
      end
    end
  end
end

SimpleSimulation.new.run
```

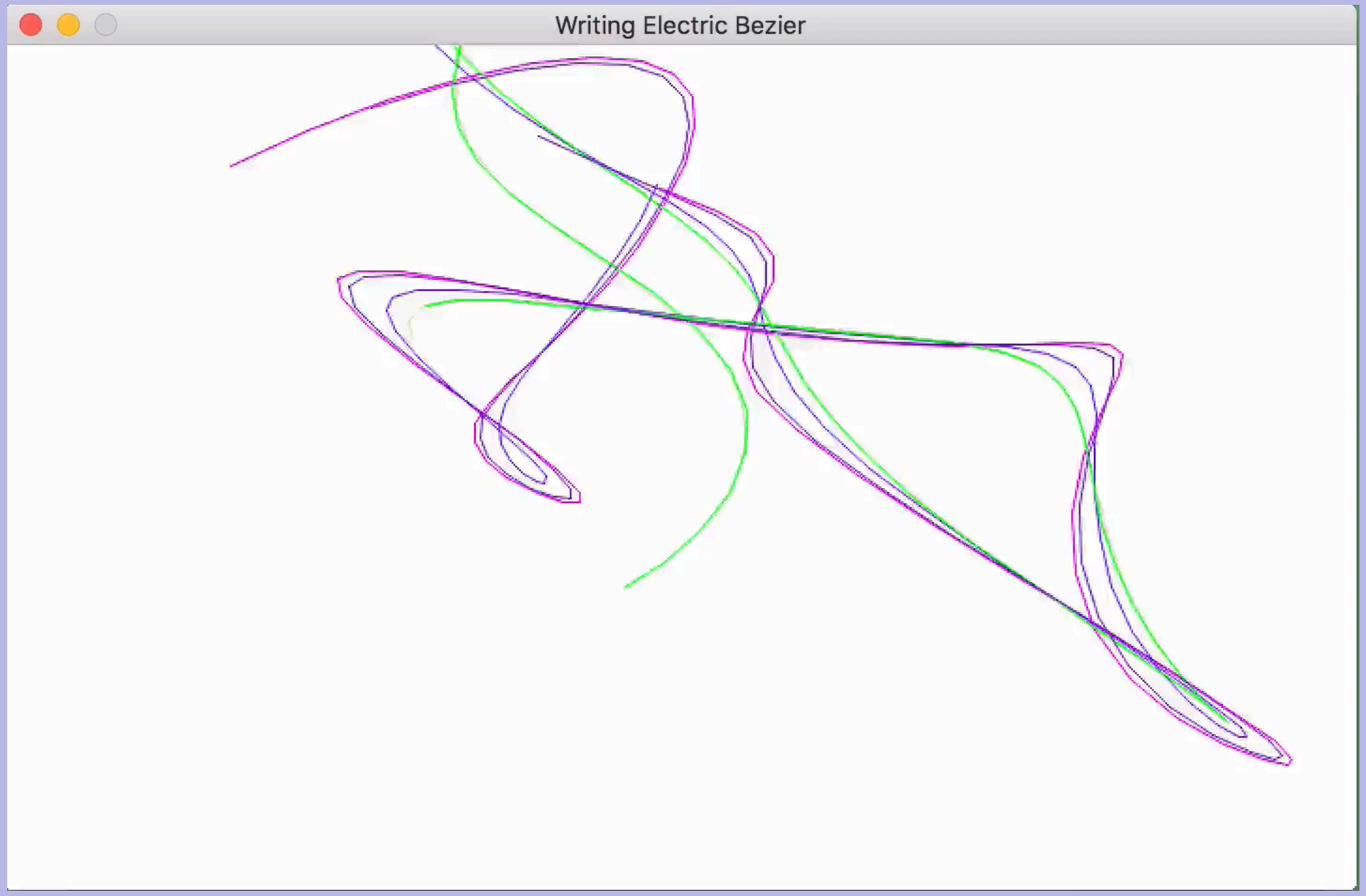
Art Gallery

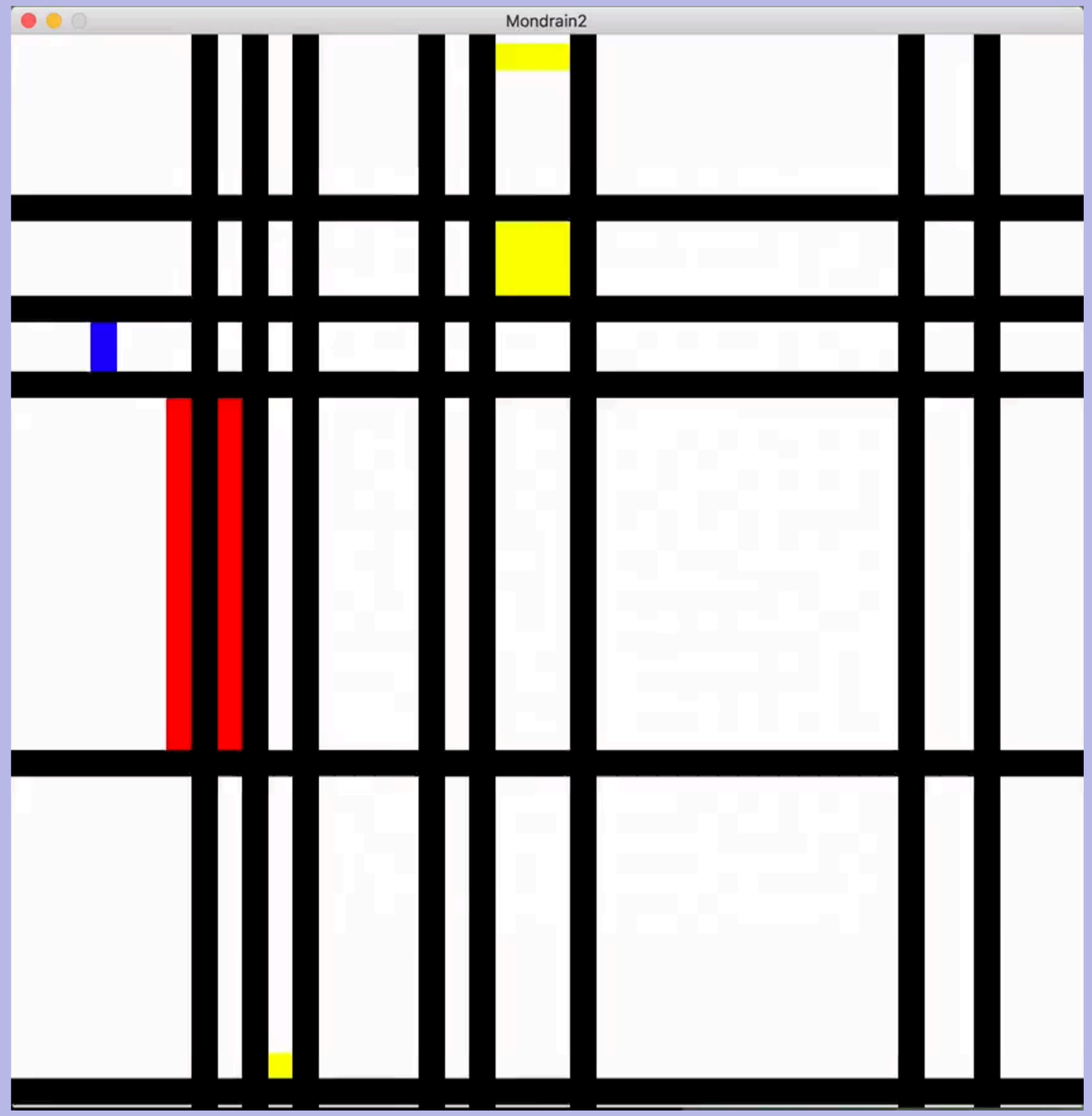
Generative Art

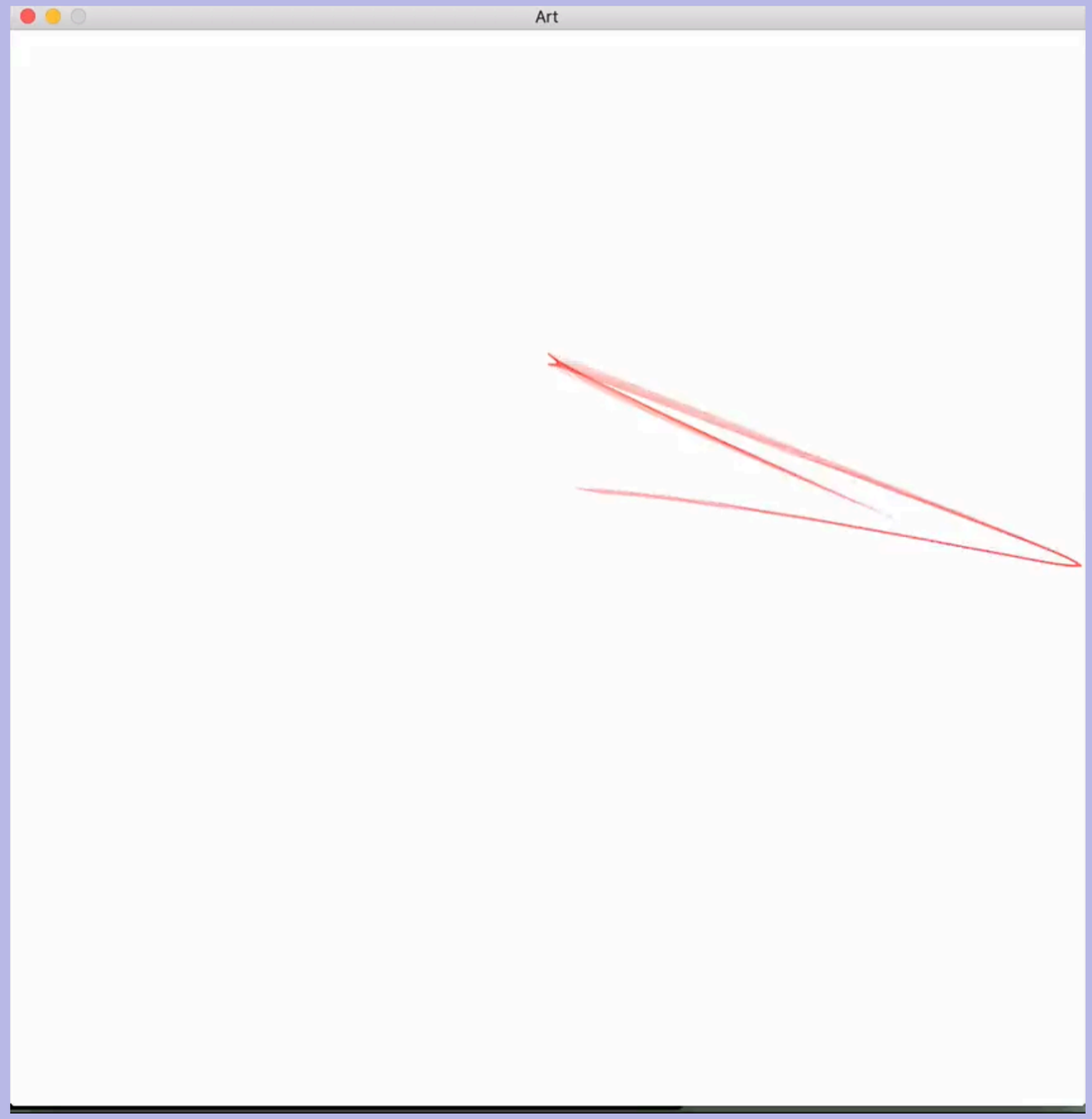

Lines Simulation
33.6 fps
n=2


Lines Simulation
33.4 fps
n=2

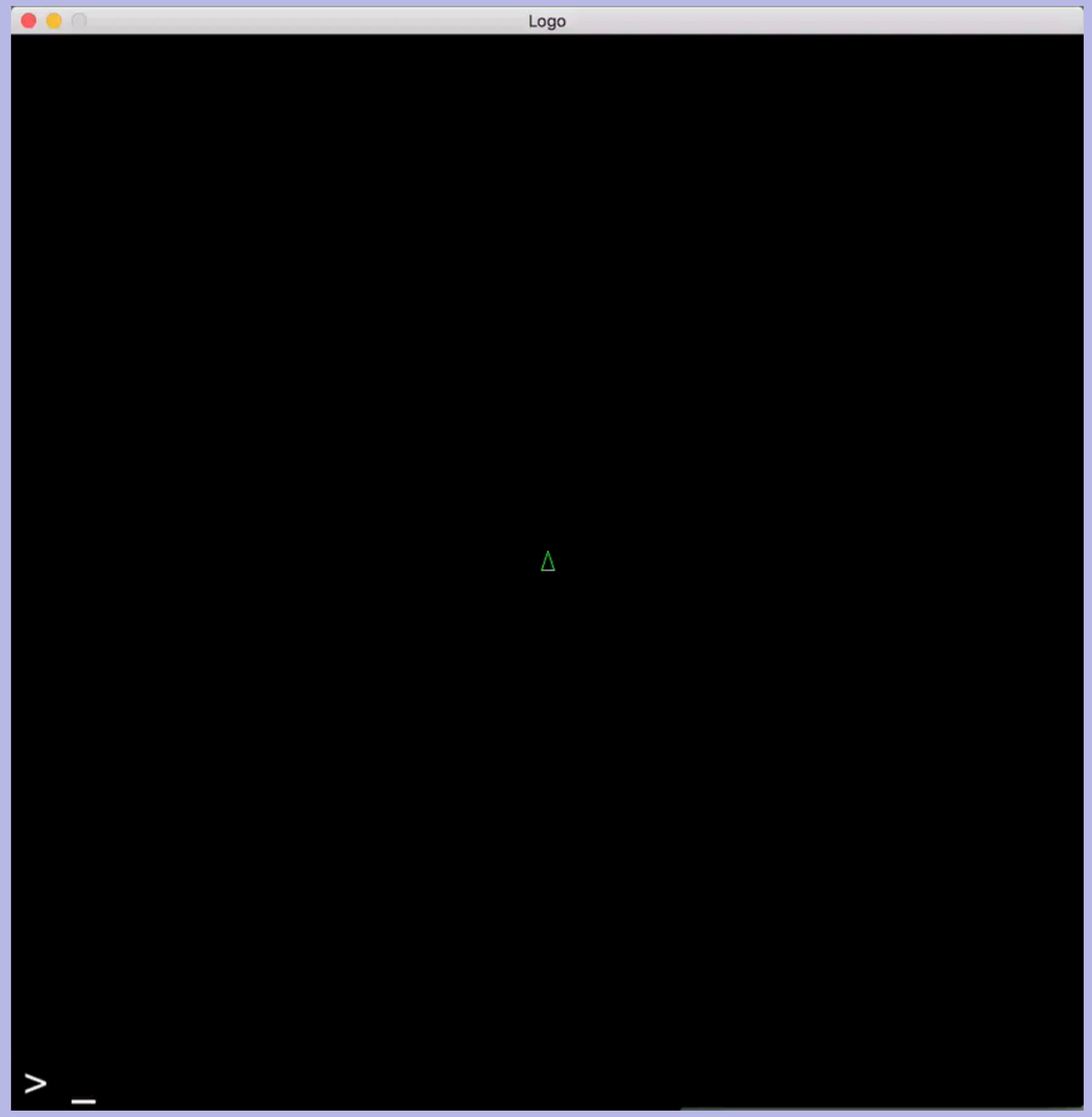

Waves

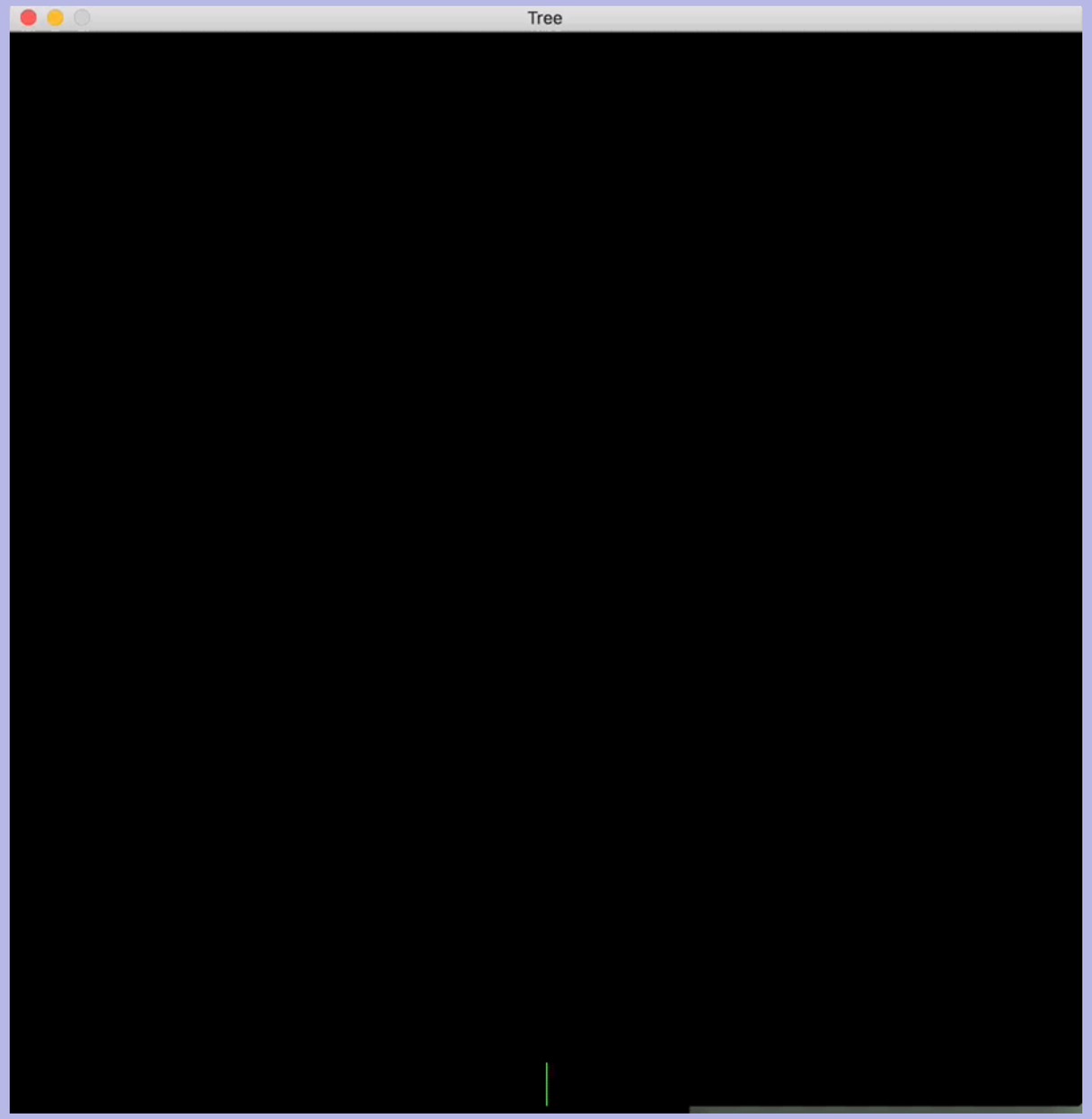

Writing Electric Bezier

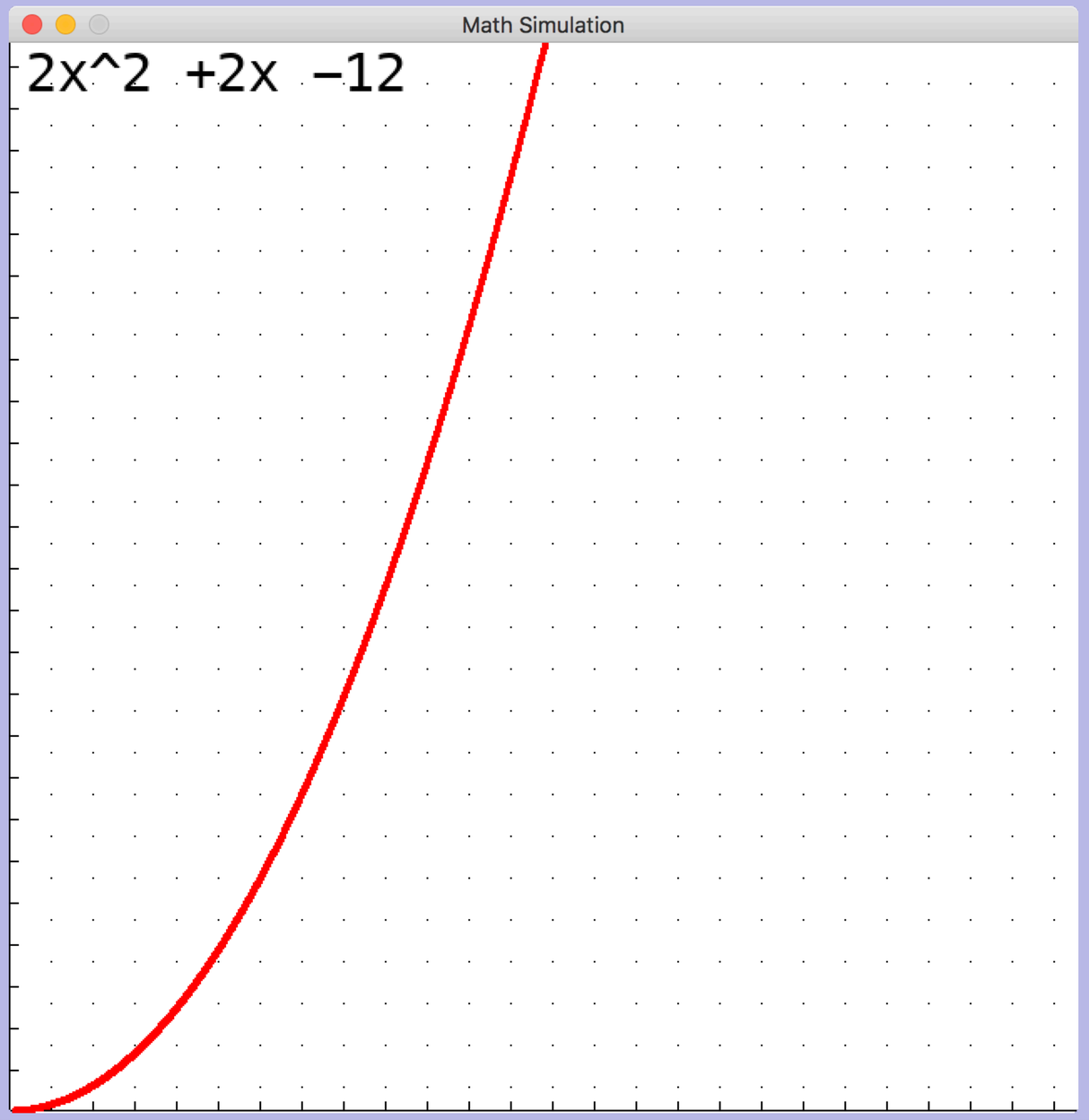

Mondrain2

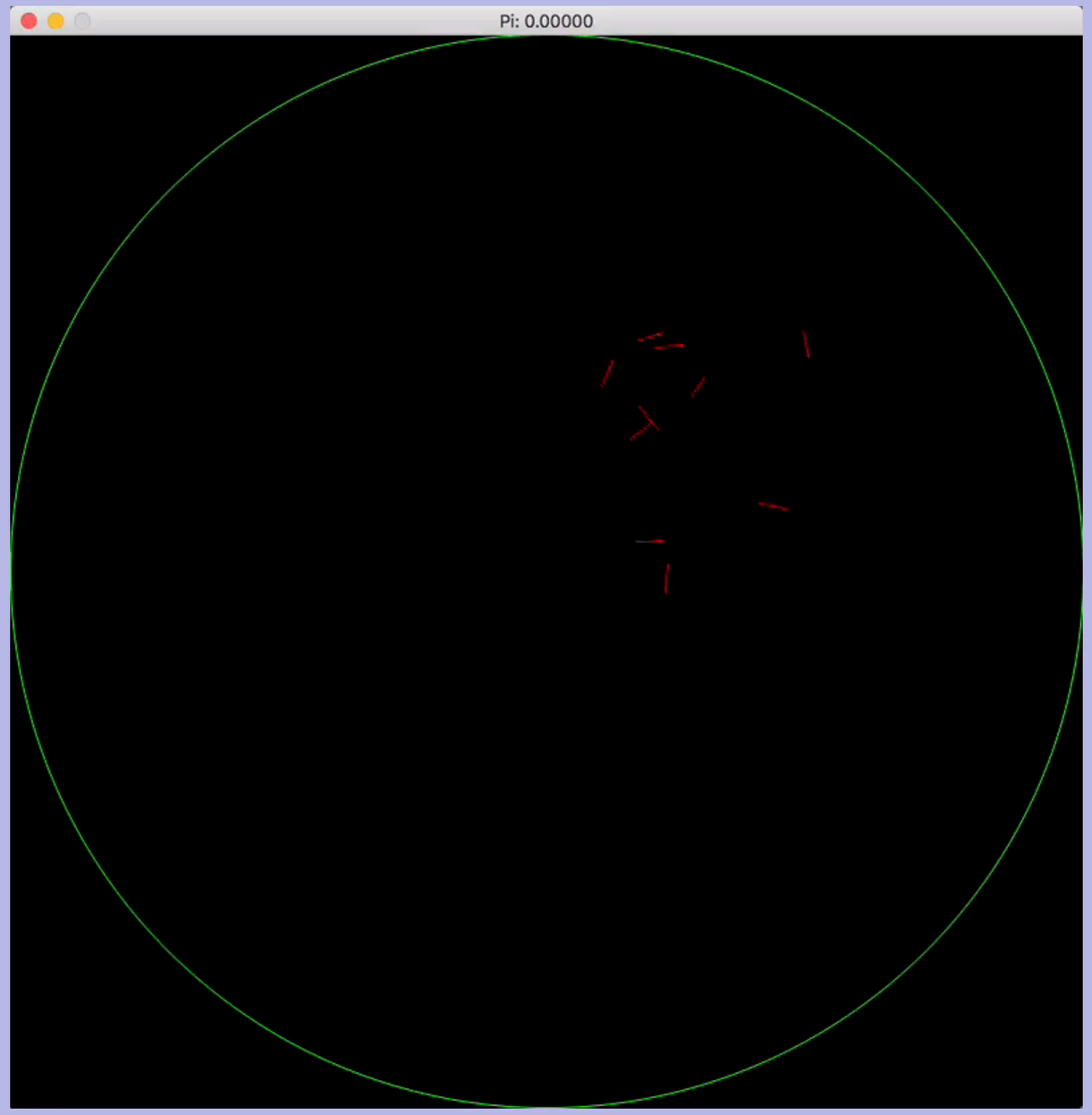

Art

Mathy / Nerdy Things

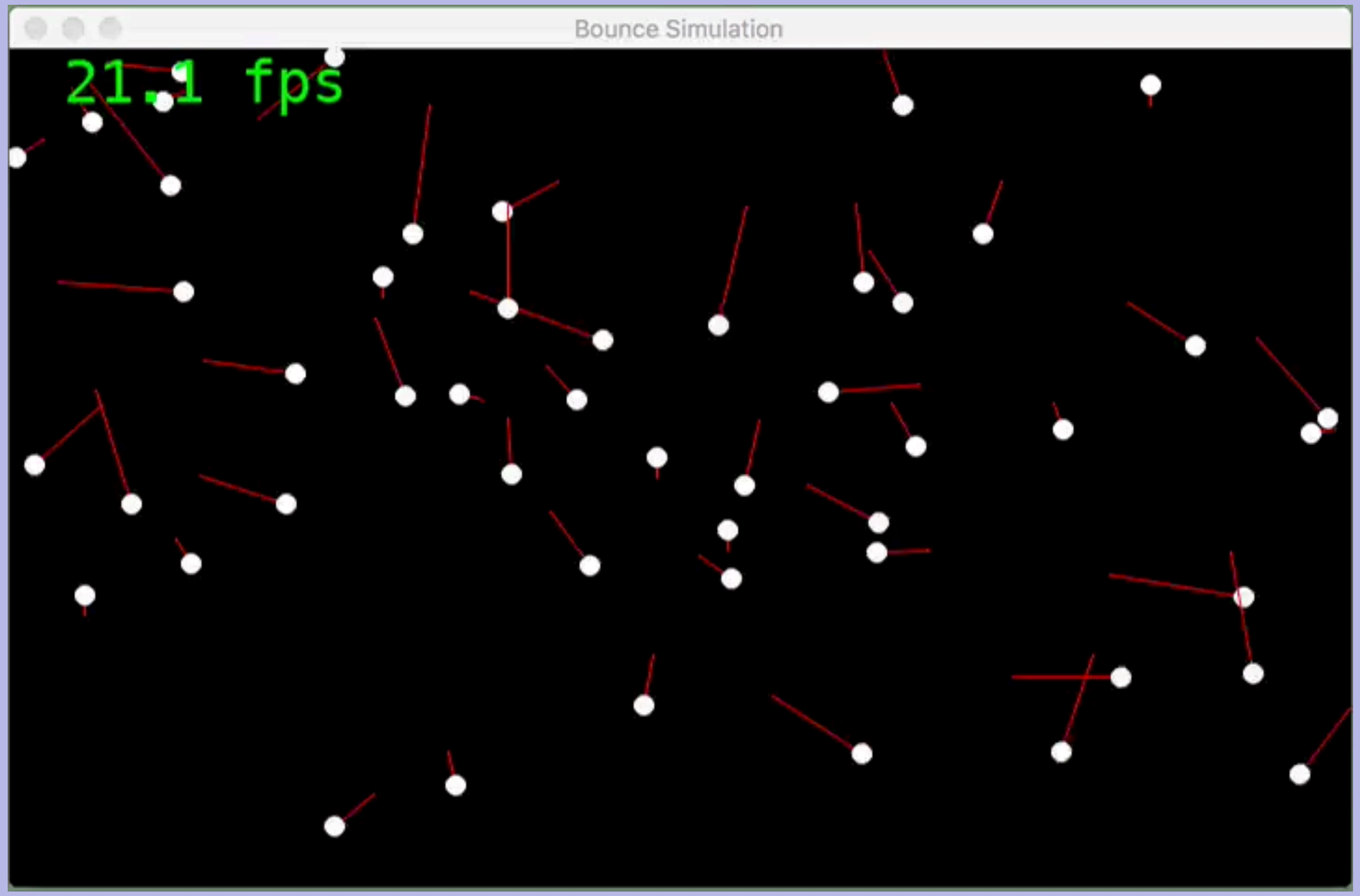

Logo
> _

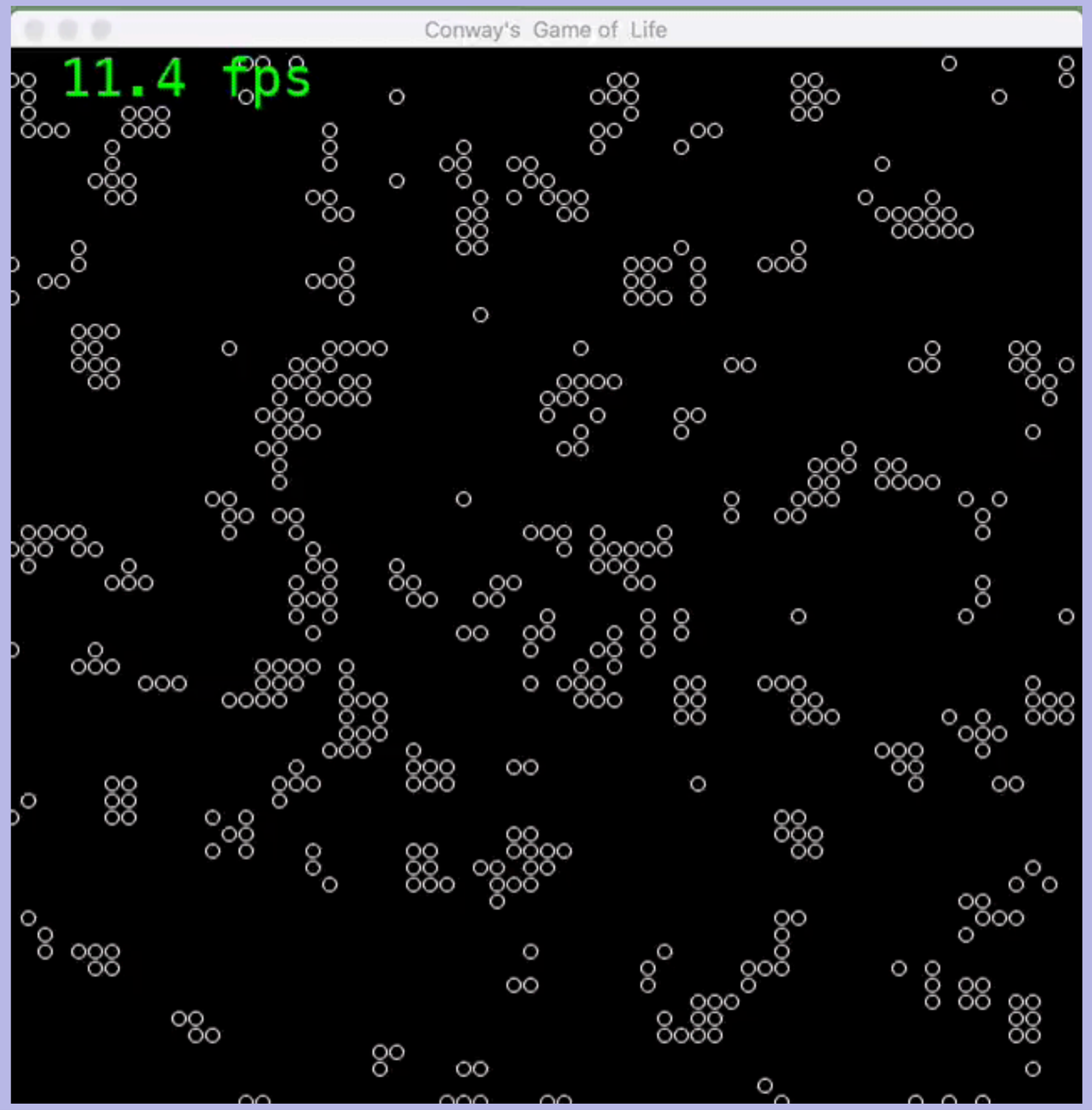

Tree

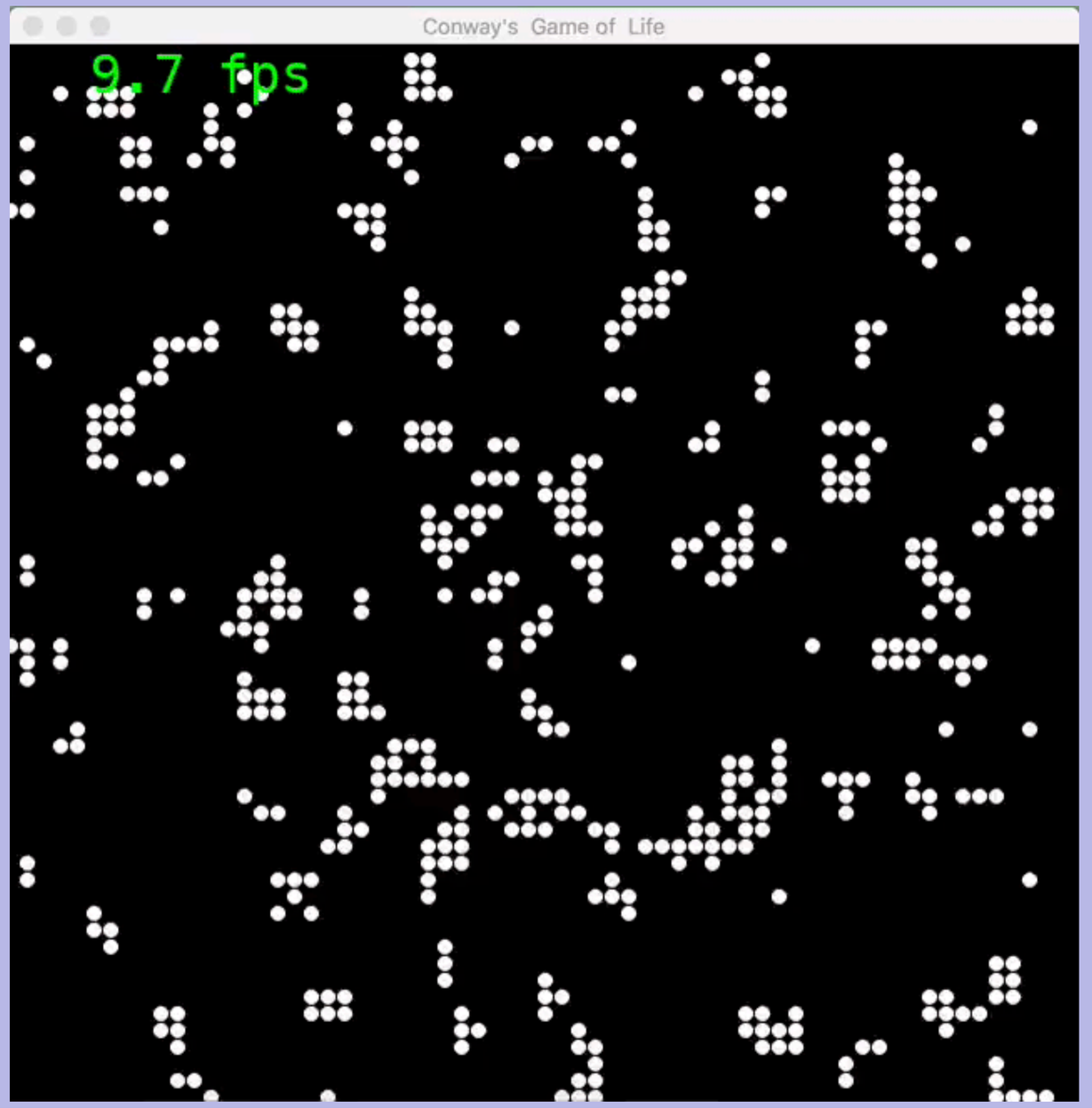

Math Simulation
2x^2 +2x −12


Pi: 0.00000

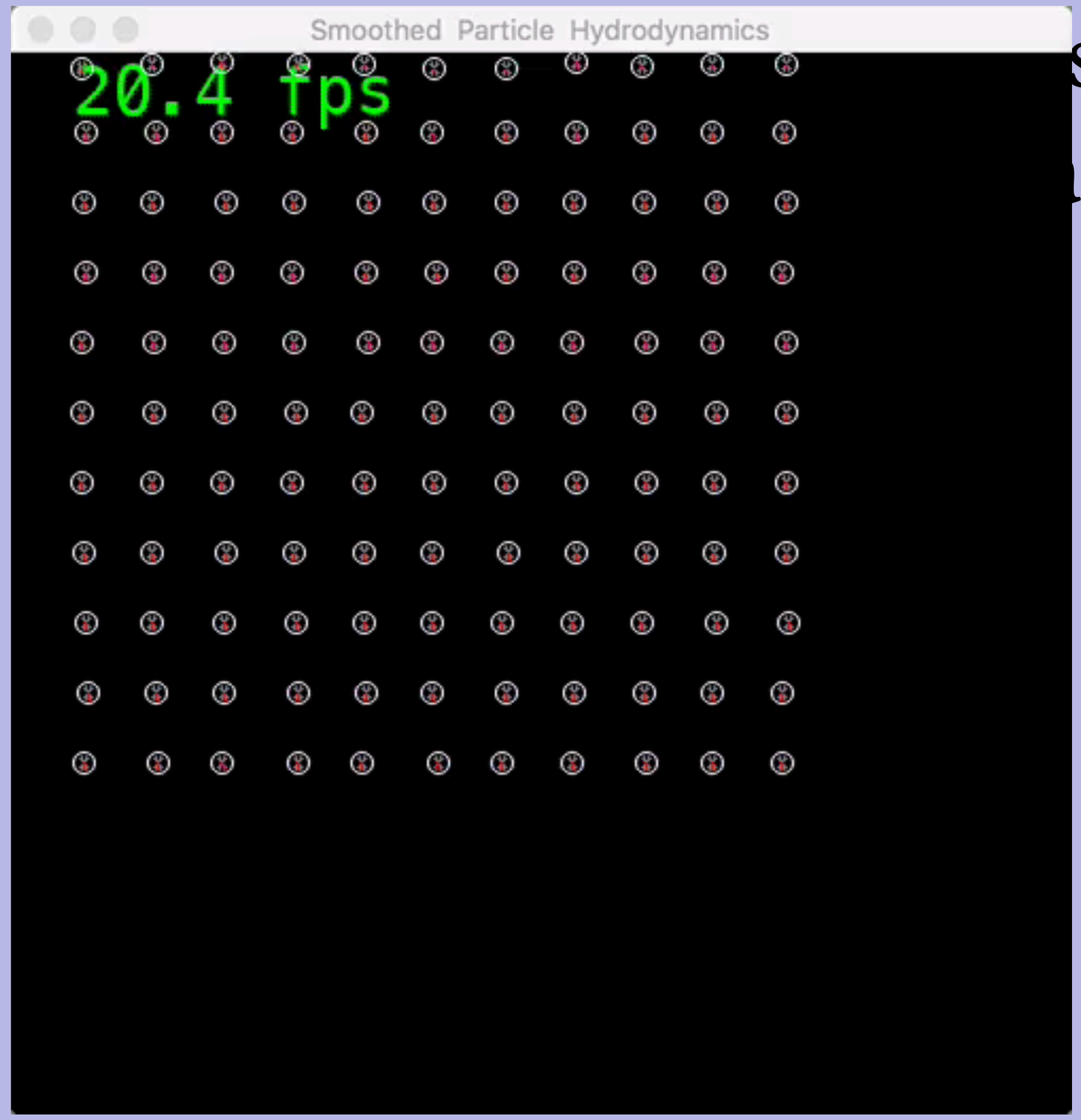
Simulations

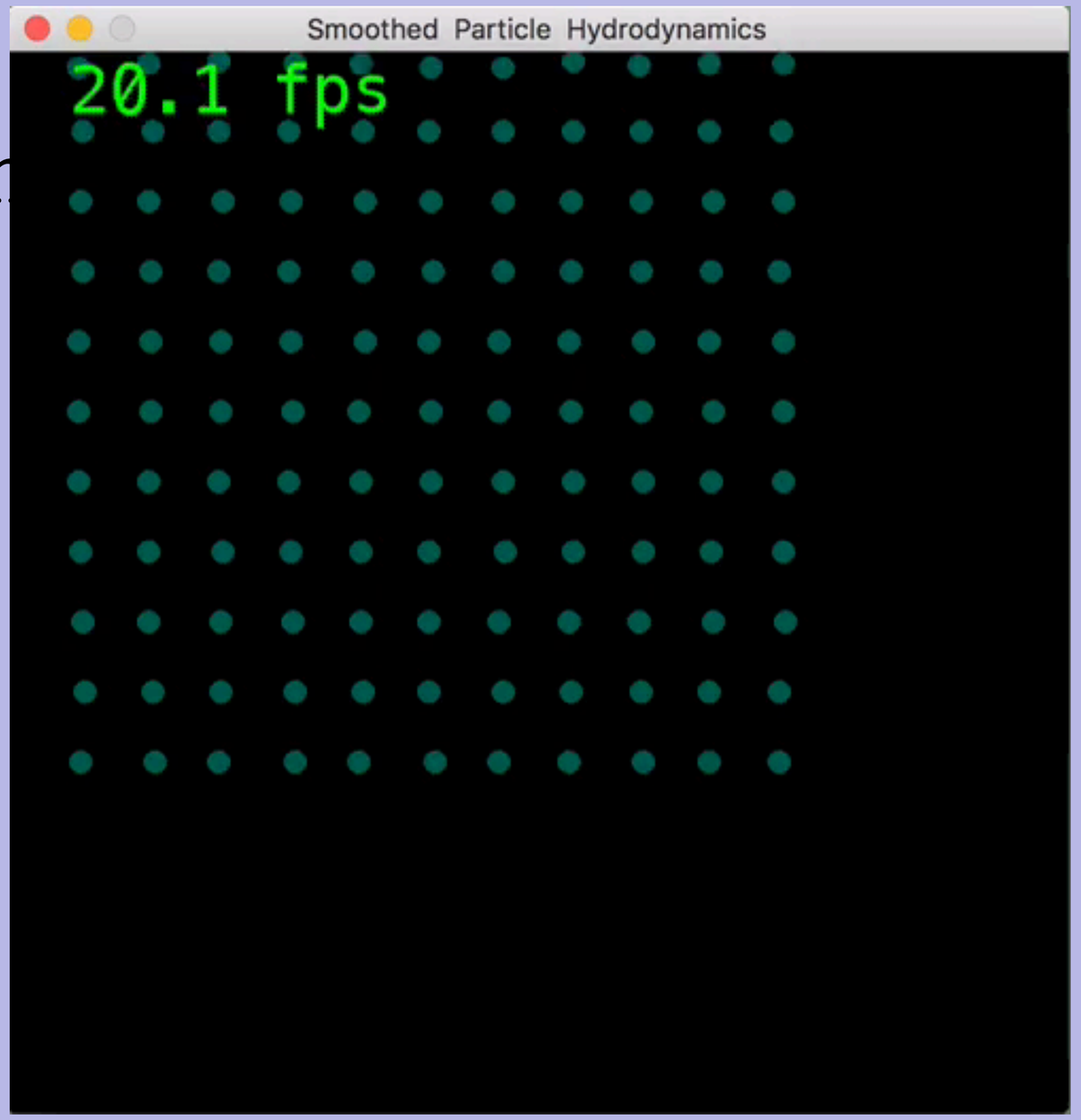

Bounce Simulation
21.1 fps

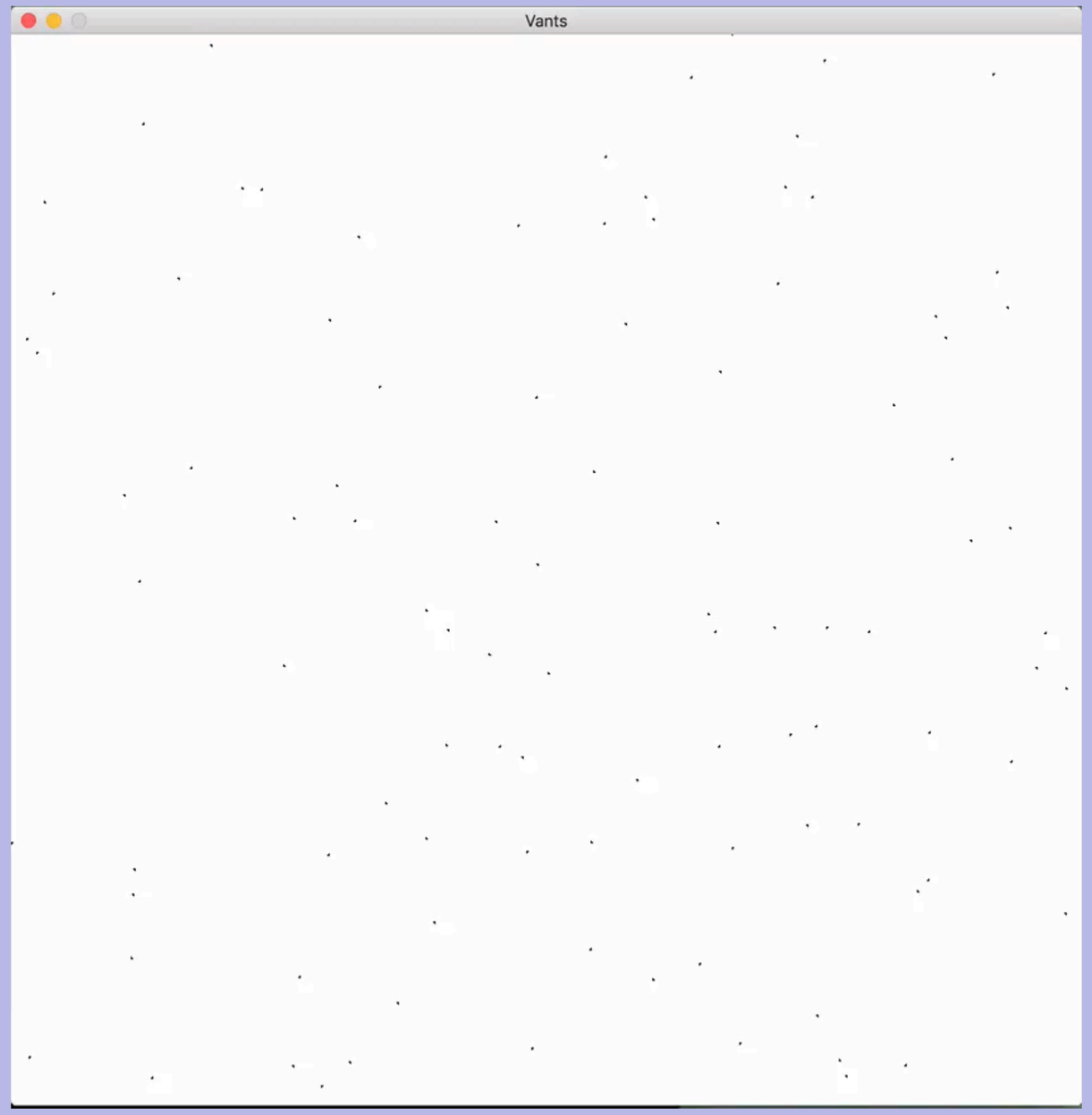

Conway's Game of Life
11.4 fps

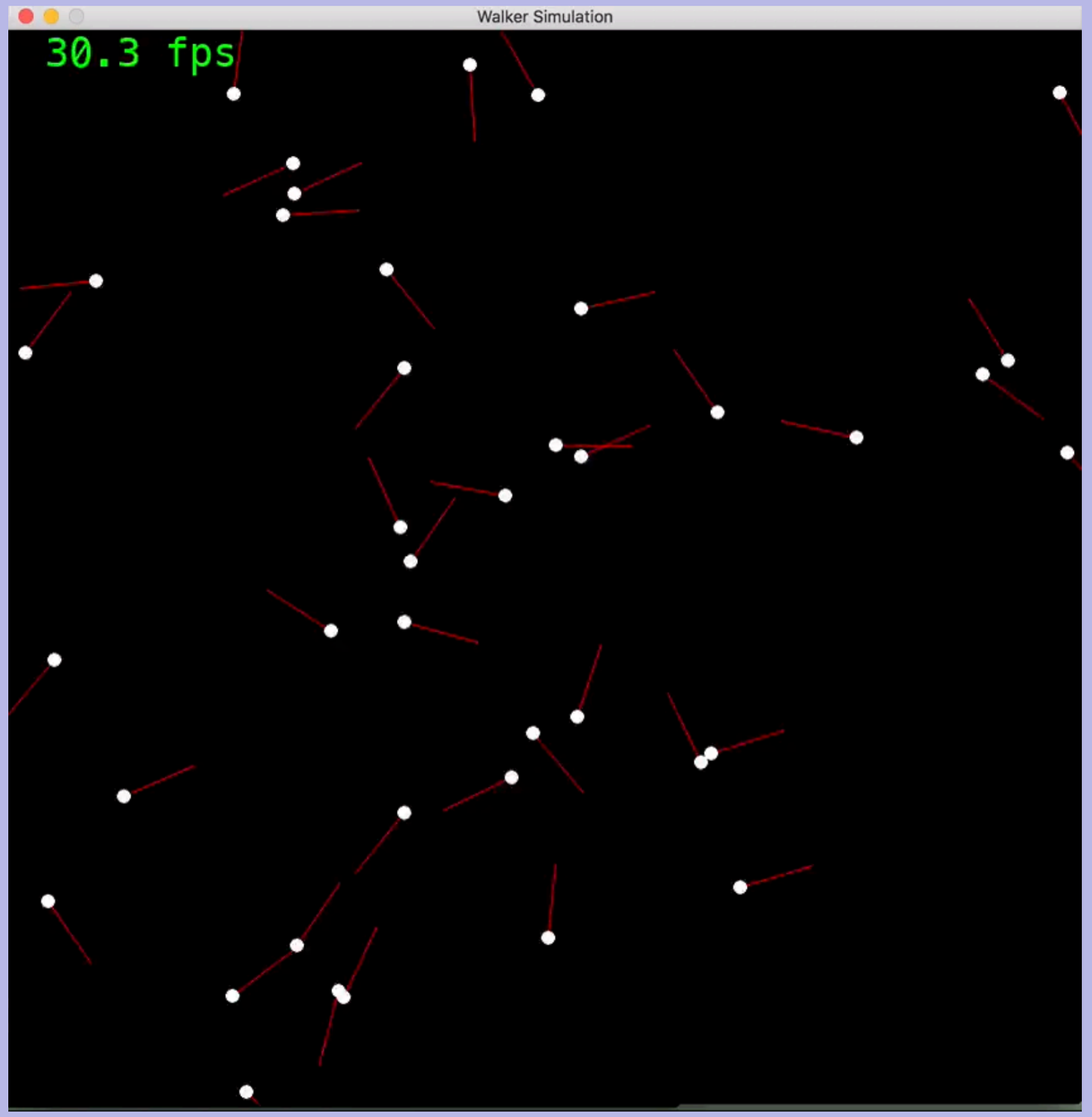

Conway's Game of Life
9.7 fps

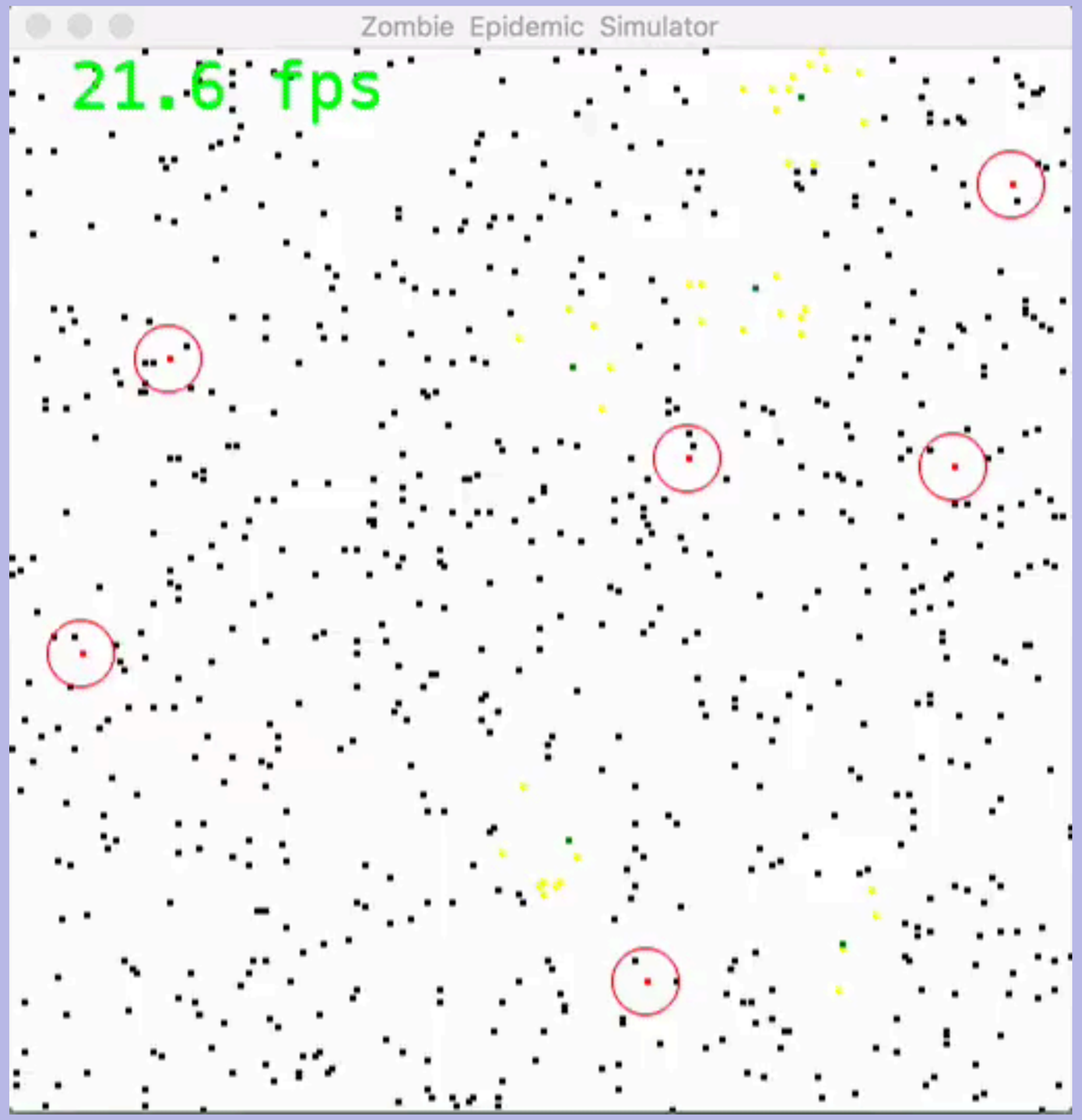

Boids
22.3 fps


Smoothed Particle Hydrodynamics
20.4 fps

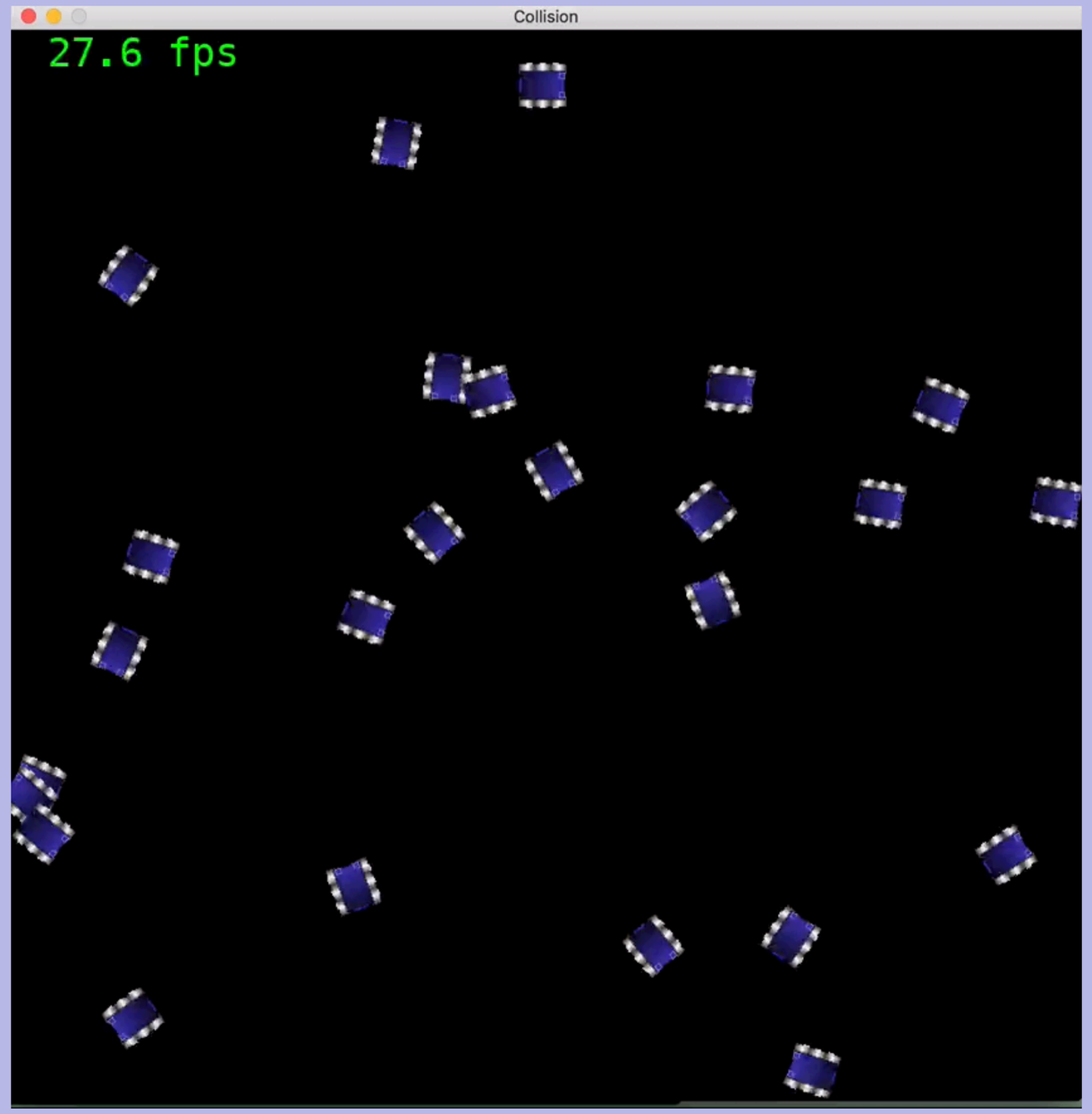

Smoothed Particle Hydrodynamics
20.1 fps


Vants


Walker Simulation
30.3 fps

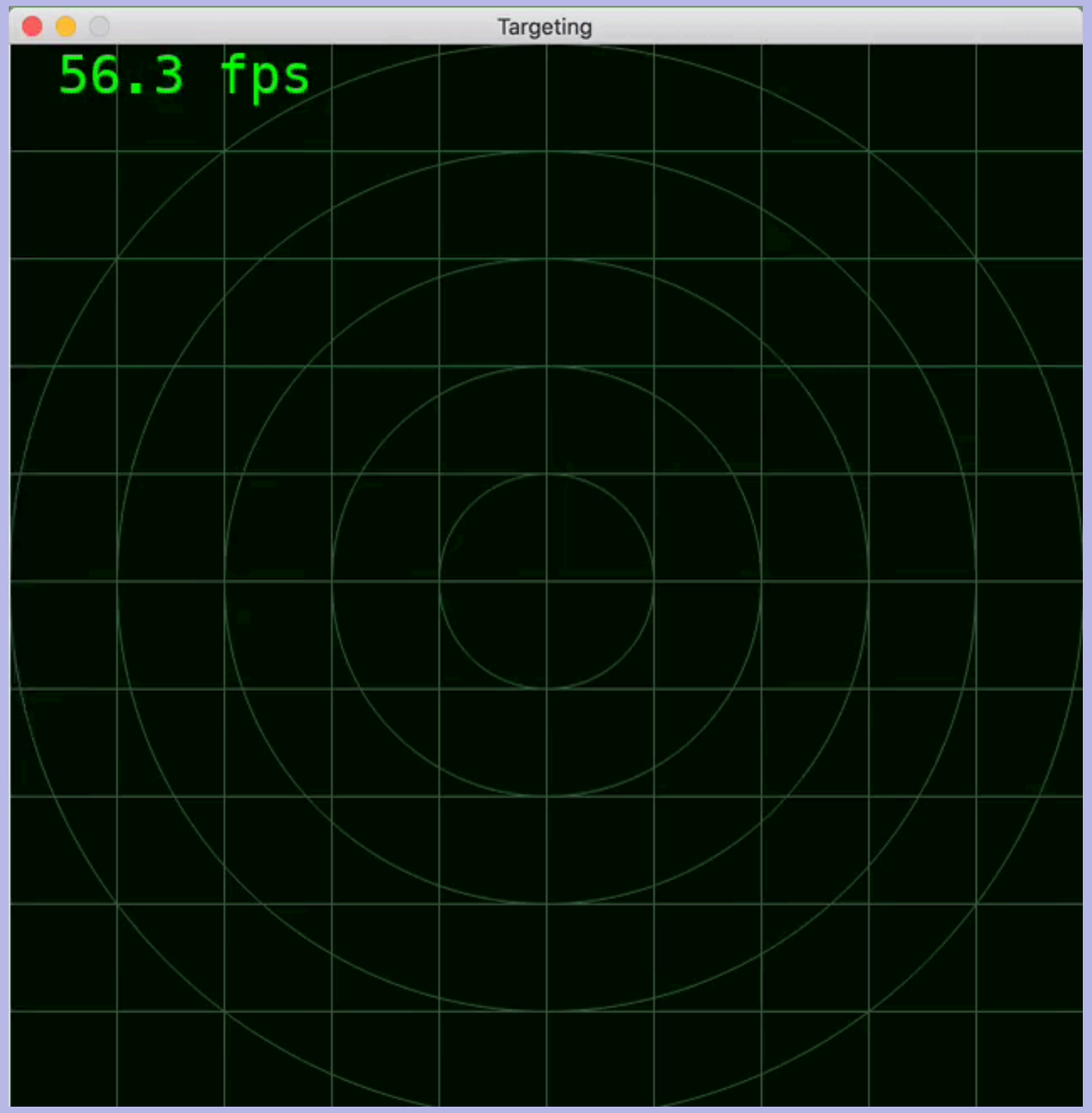

Zombie Epidemic Simulator
21.6 fps

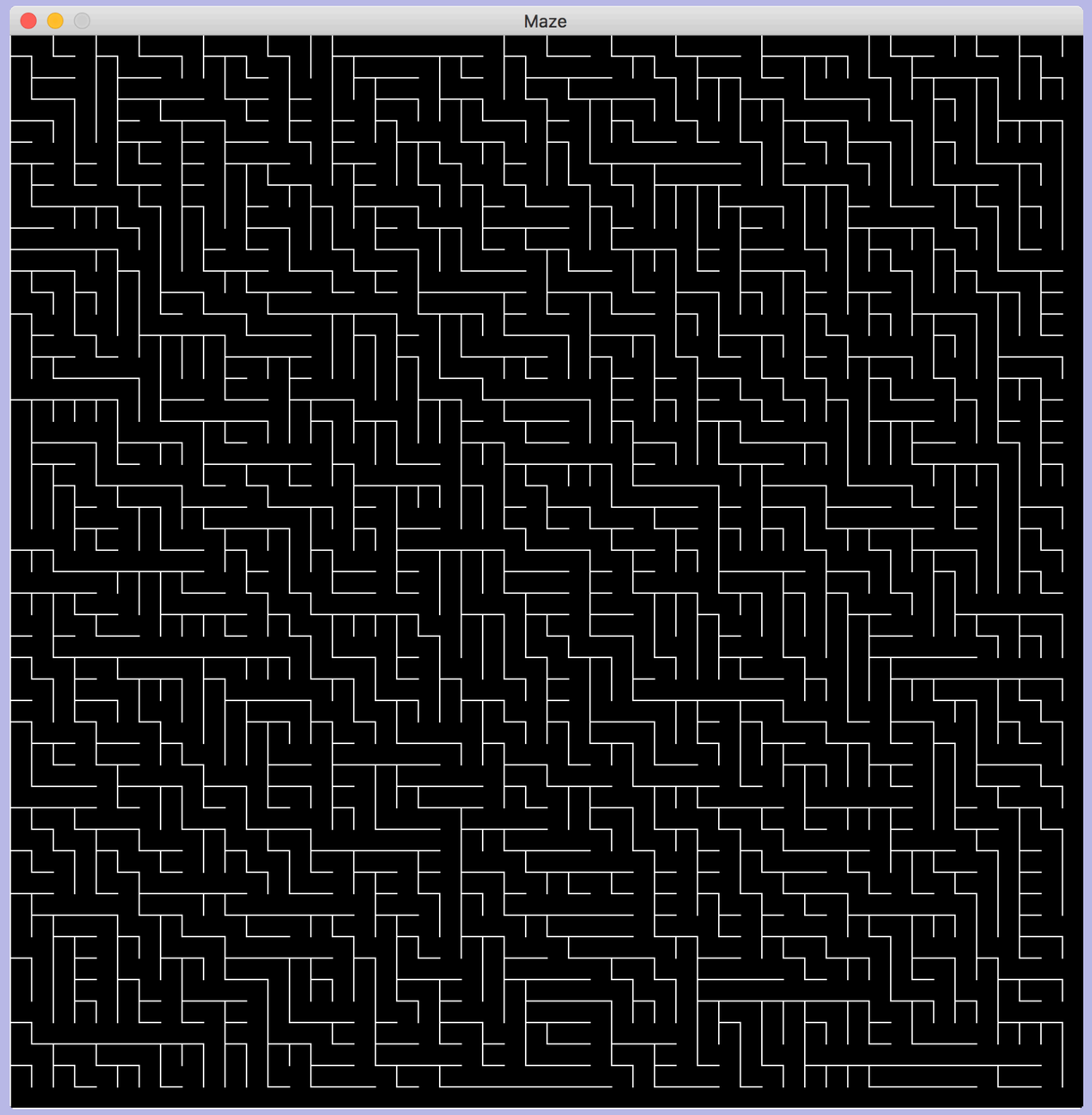
Games


Collision
27.6 fps


Tank Sprites
58.3 fps
0 @ 0.00 @ 100


Tanks
57.7 fps


Targeting
56.3 fps


Maze

Graphics

How is it Different?

Approachable

Undemanding

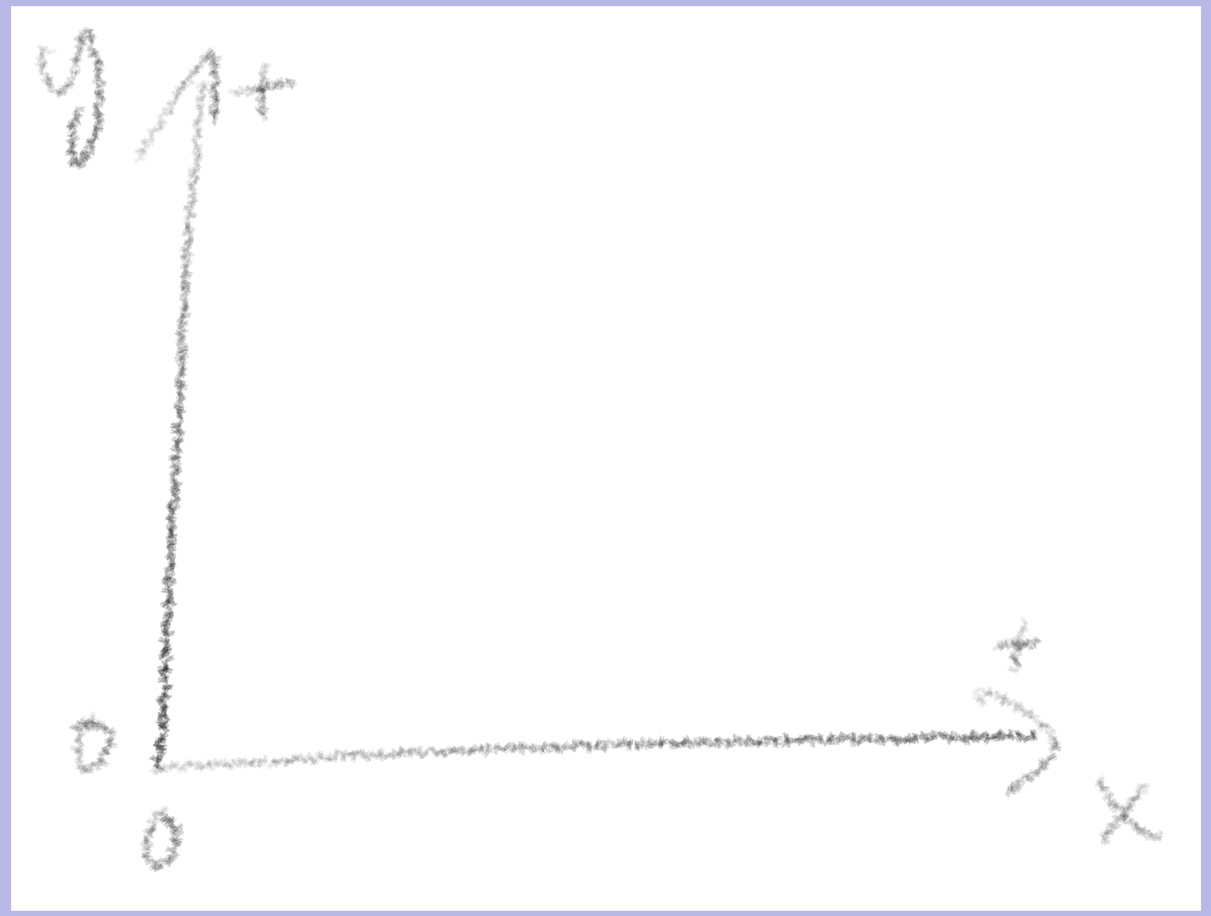
Understandable

Empowering

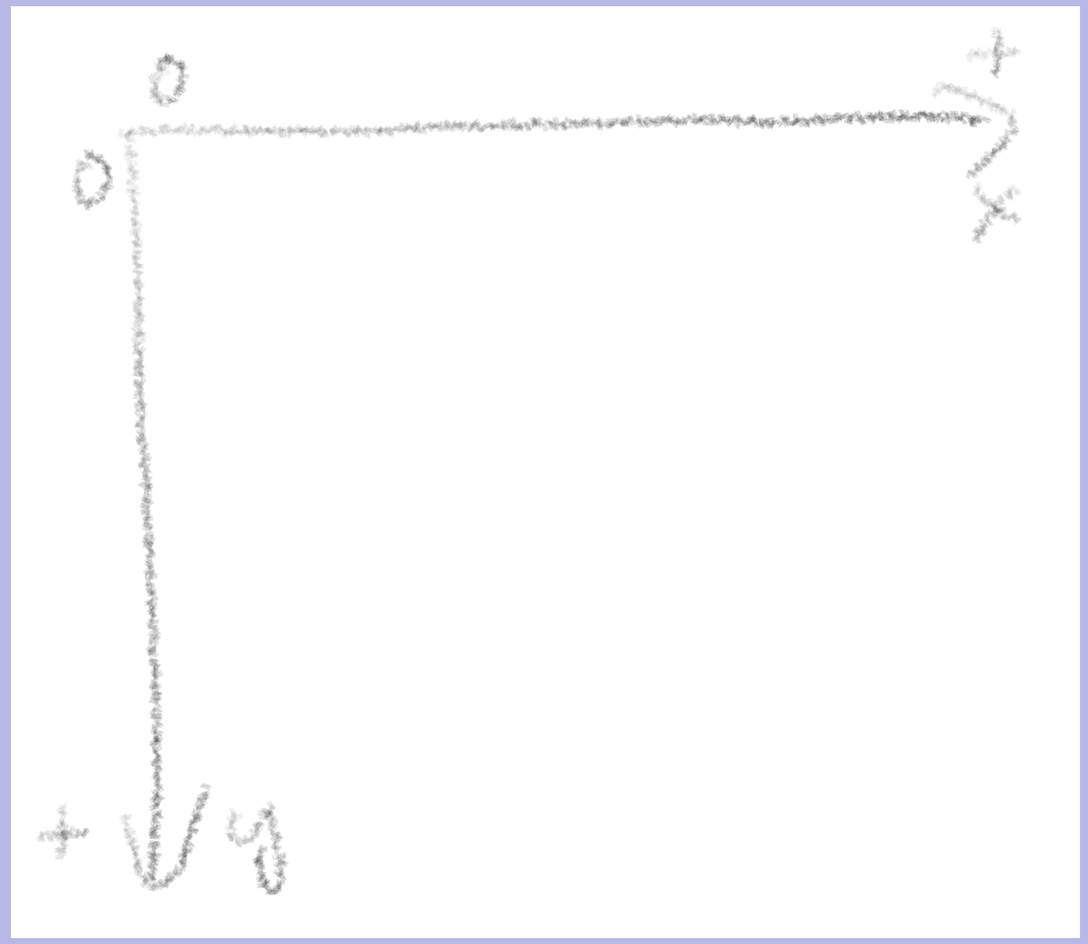
Experimentable

Real Maths!

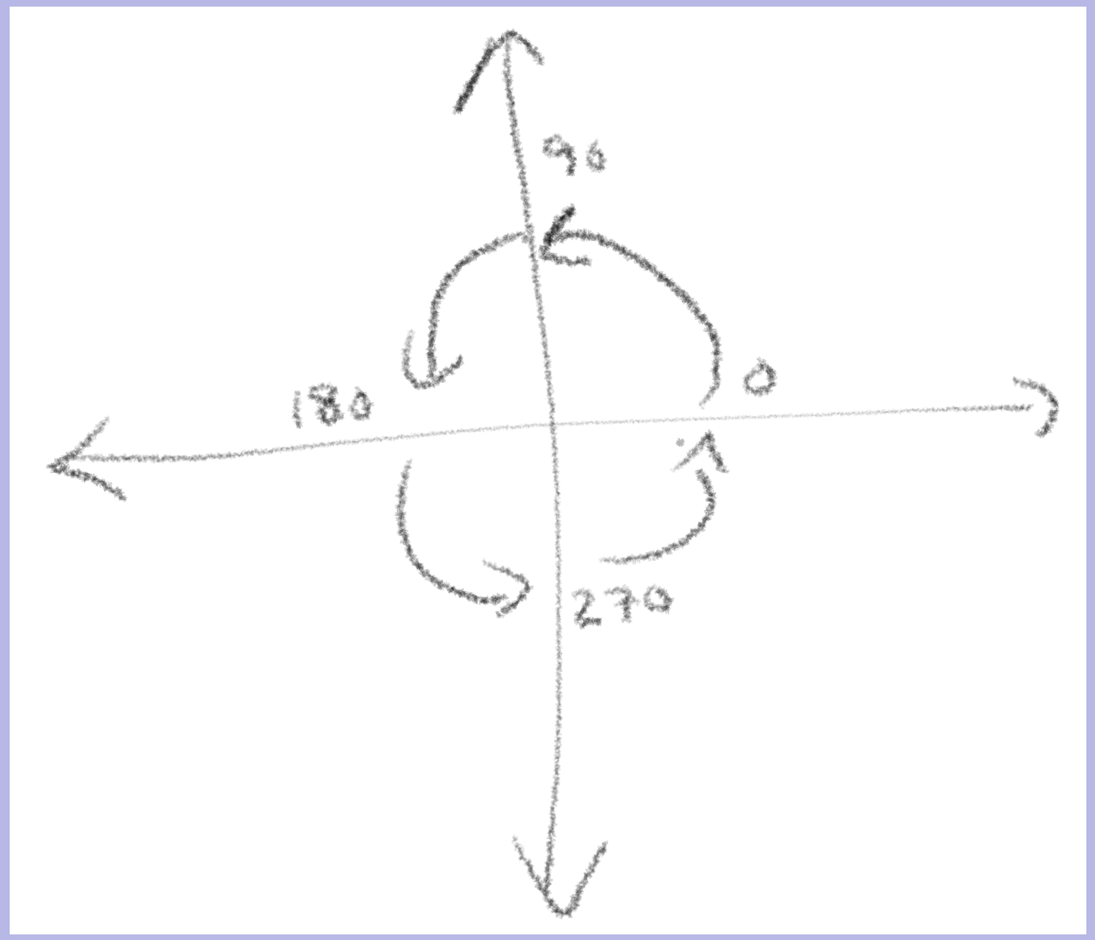
Uses degrees.



Draws in quadrant 1 (0-90 degrees).



Game Math Coordinates


90
180
0
270

Comprehensible

Opinionatedly *Pretty*

20+ Drawing Primitives

Pretty drawing!

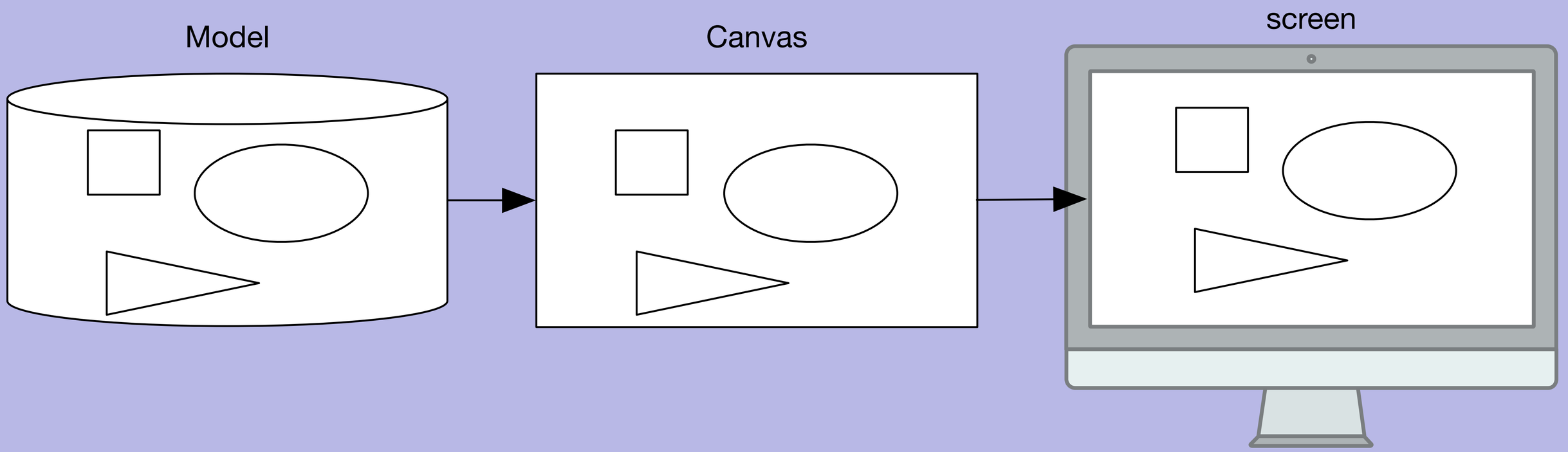
Decorators

```
class MySimulation < Graphics::Simulation
  include WhiteBackground
  include ShowFPS
  include DrawGrid; GRID_WIDTH = 100
end
```

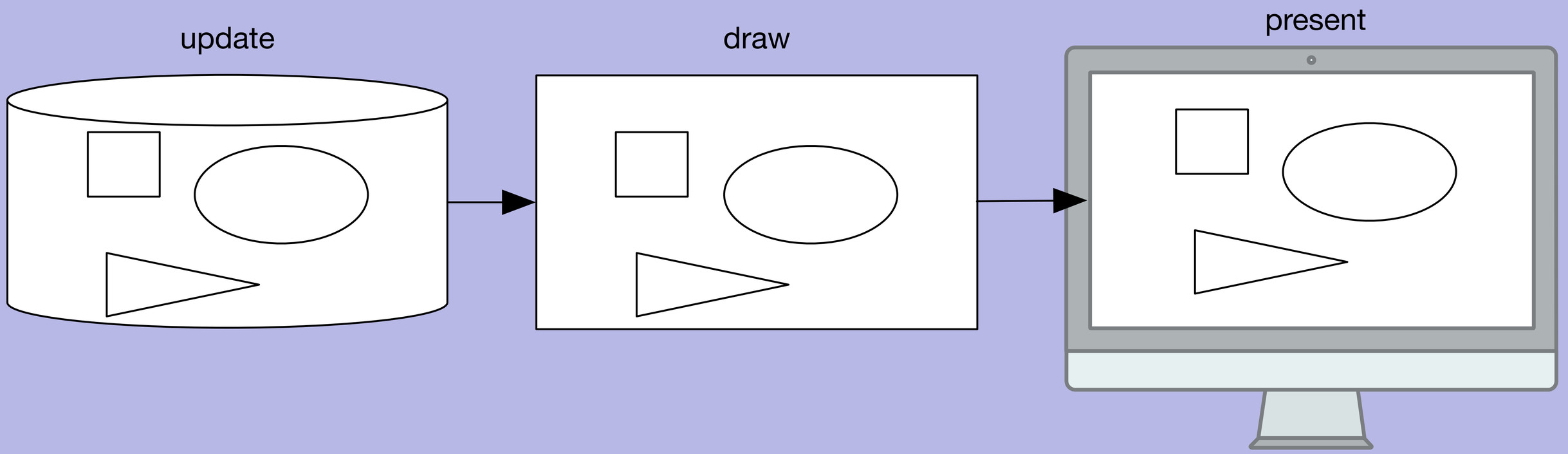
Architecture

Update & Draw Loop

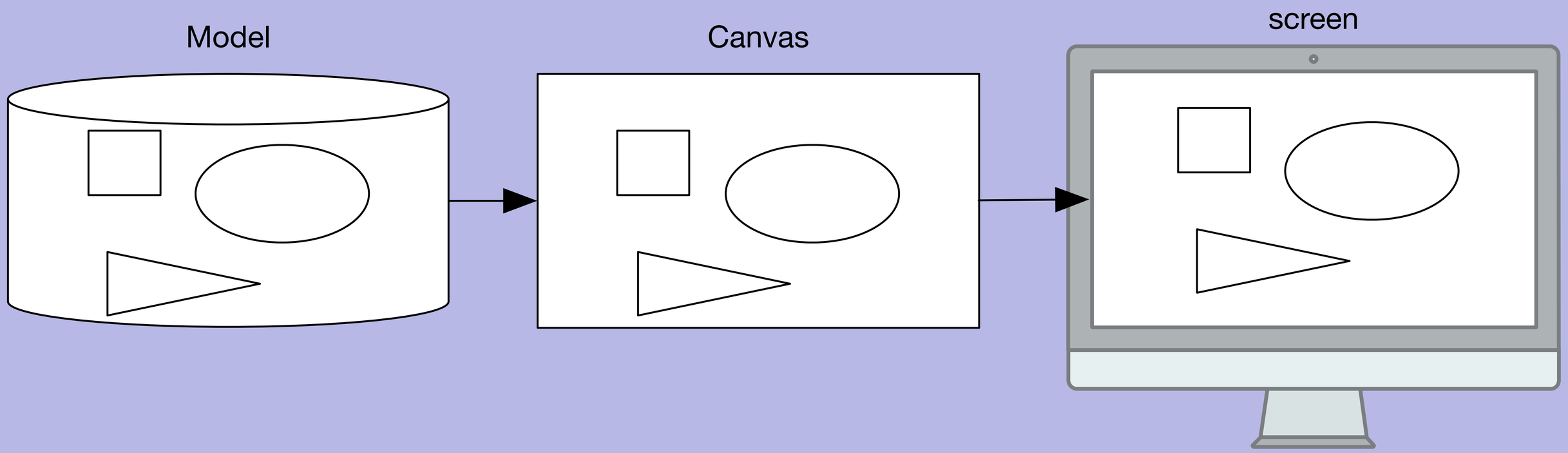
30,000 Foot View



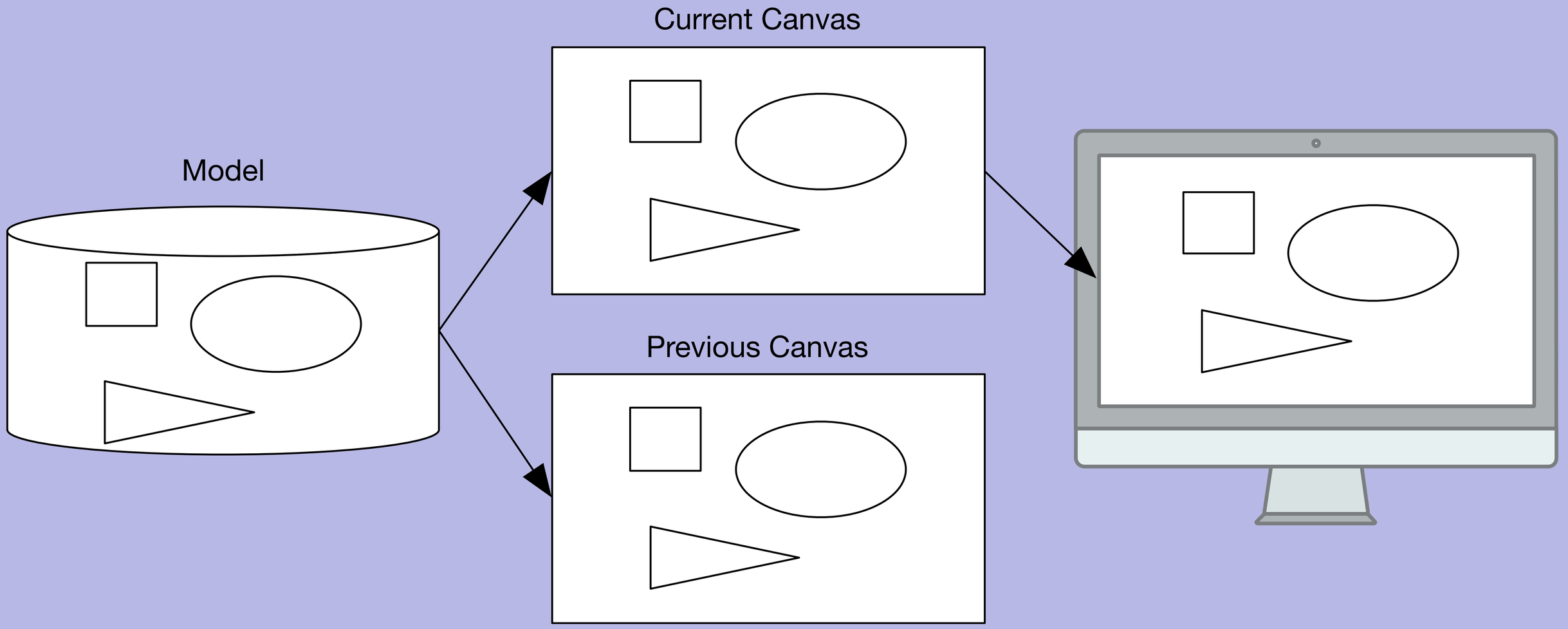
Translates to:



Abstraction of Drawing



"Reality"



Run Loop

```
loop do
  iter_per_tick.times { update n; n += 1 }
  draw n
  renderer.present
end
```

update

```
def update n
  _bodies.each do |ary|
    ary.each(&:update)
  end
end
```

draw

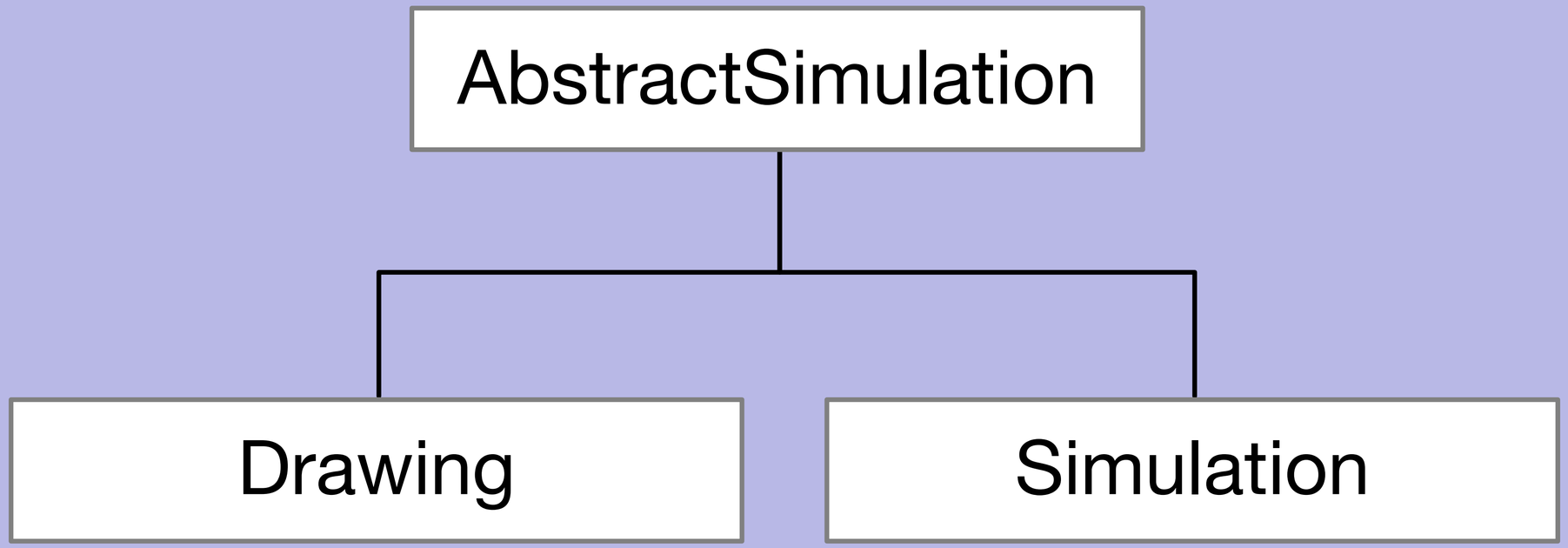
```
def draw n
 clear

 _bodies.each do |ary|
    draw_collection ary
  end
end
```

```
def draw_collection ary
  return if ary.empty?

  cls = get_view_class ary

  ary.each do |obj|
    cls.draw self, obj
  end
end
```

render

```
renderer.present
```

Class Heirarchy



Simulation

“Normal” Functionality

Animation

Clears the window

Smooth Drawing

Drawing

Never Clears

Single Canvas

Single Buffered

Static

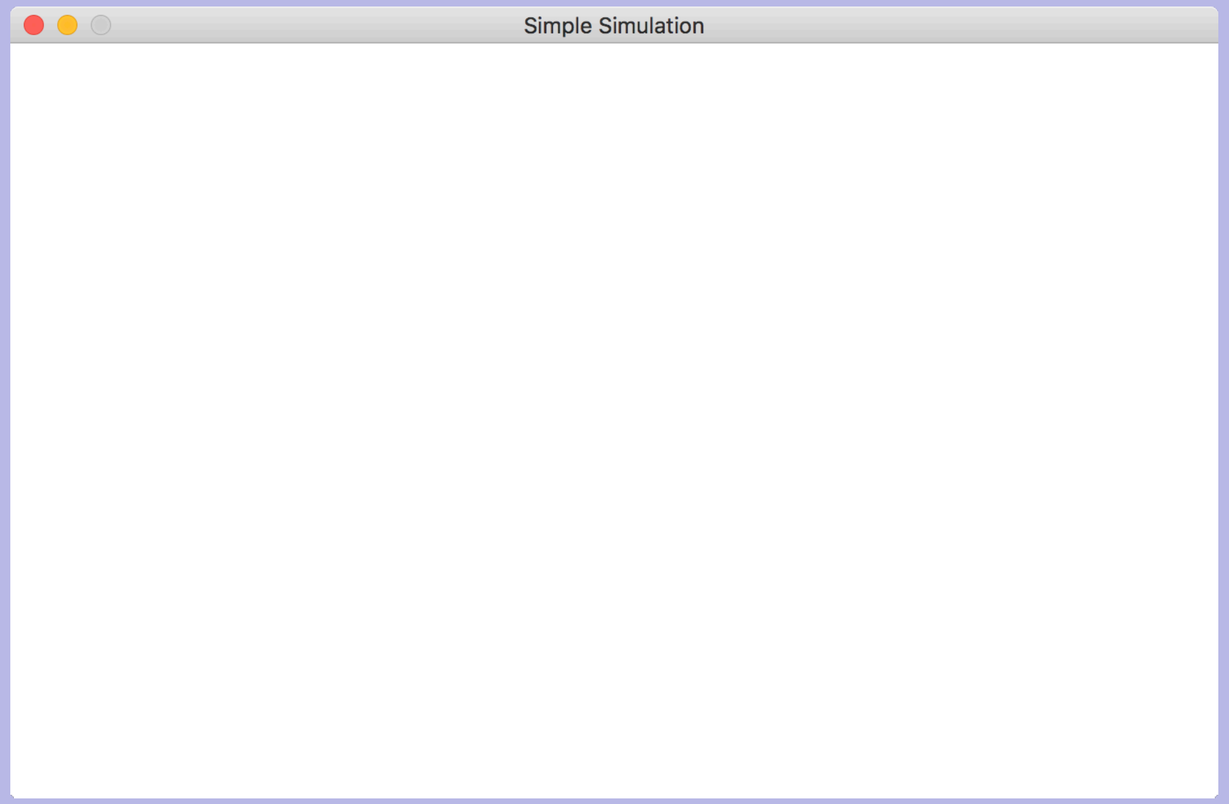
Primitives

Shortcuts

```
m  = 50                          # margin
rx = rand w                      # random x
ry = rand h                      # random y
rc = color.keys.sample           # random color
rb = [true, false].sample # random bool

def r50                          # random value
  rand 50
end
```

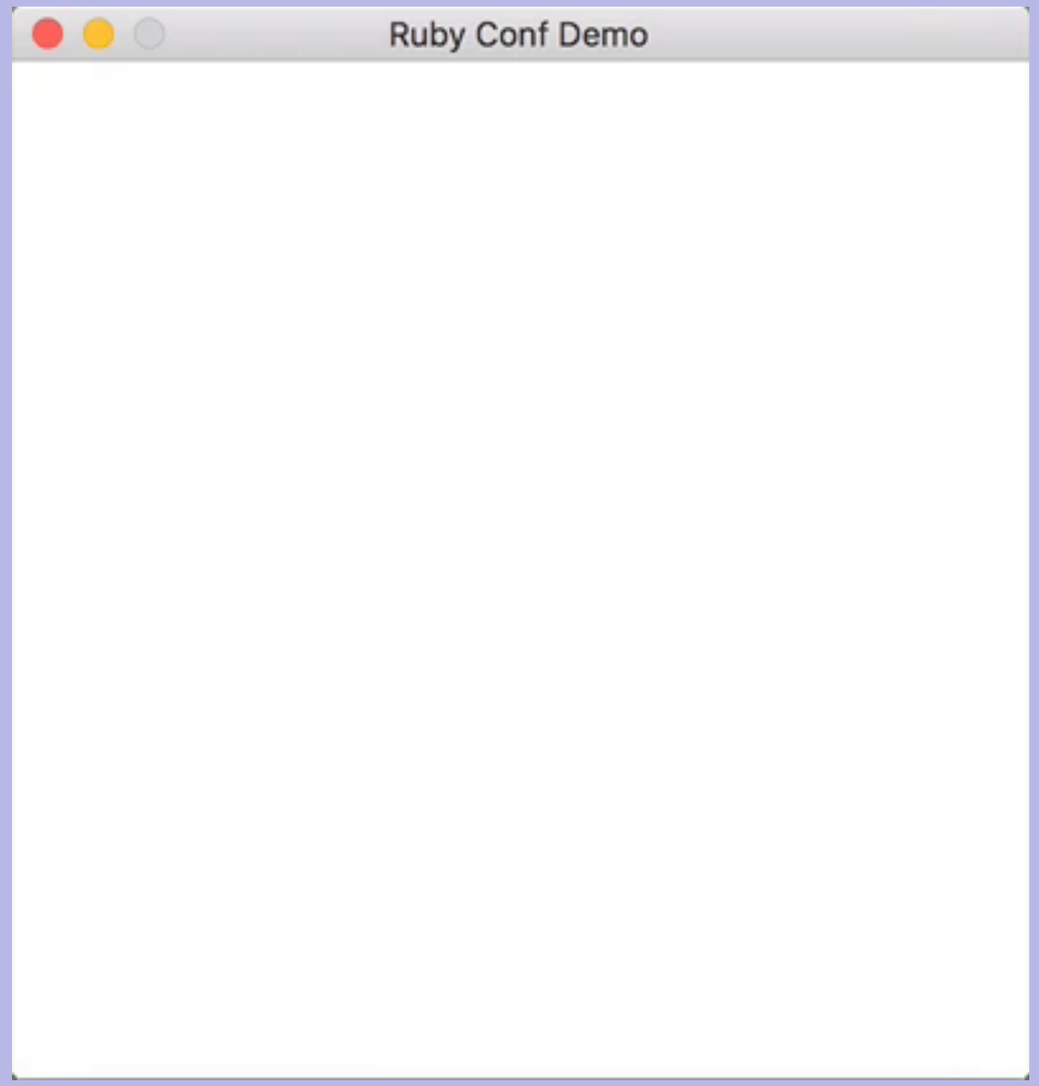
clear

```
clear # defaults to CLEAR_COLOR
clear :red
```



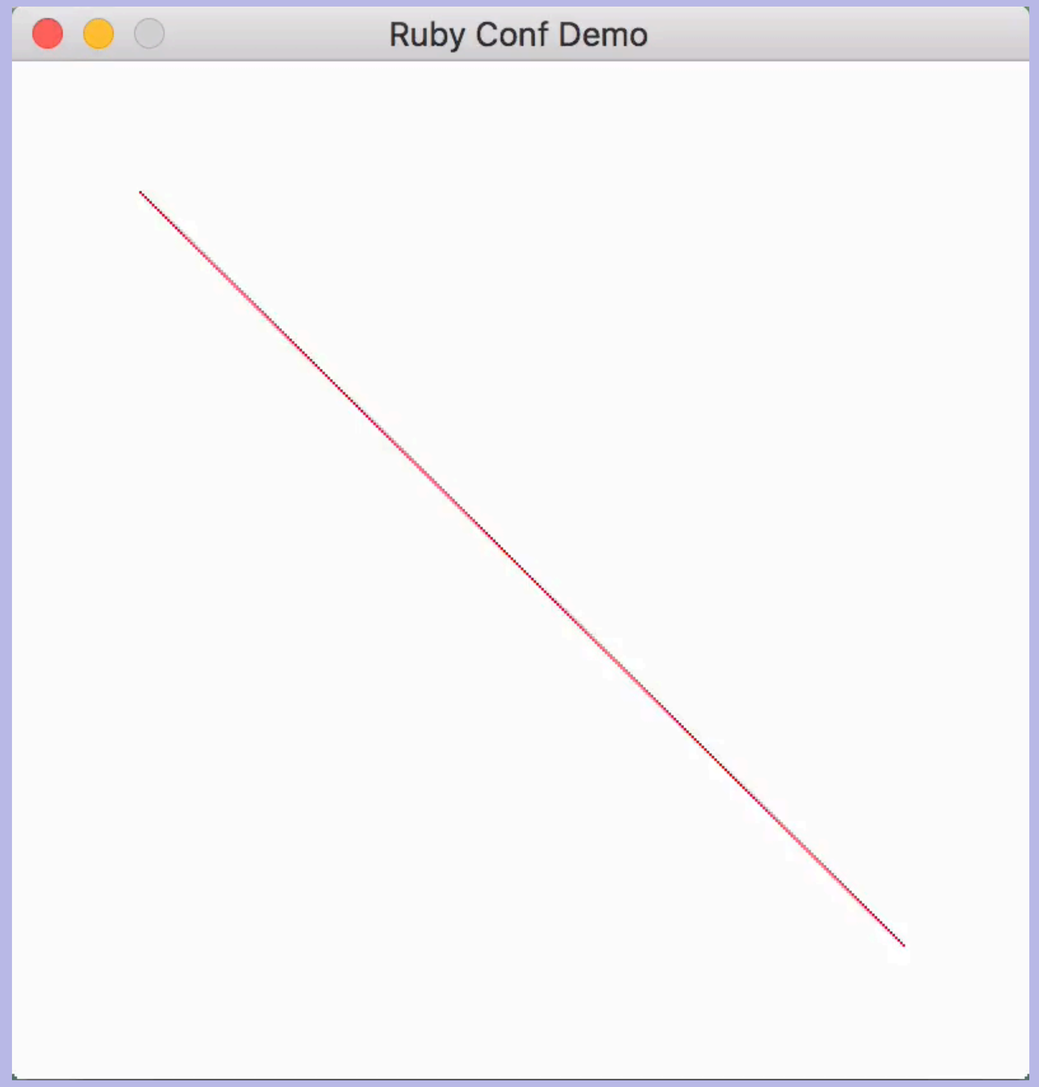
```
1000.times do
  point rx, ry, rc
end
```

point



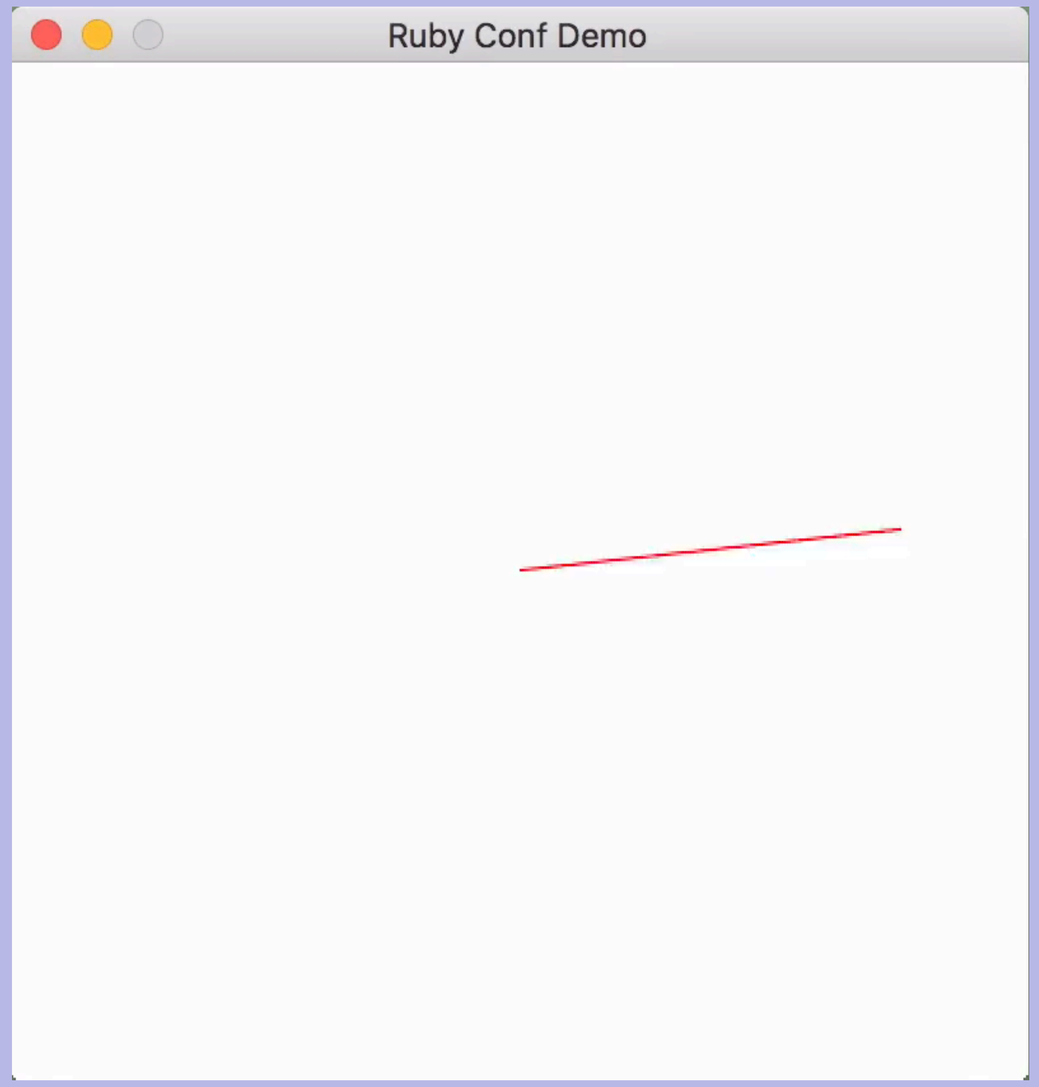
line

```
line m, h-n-m, w-m, n+m, :red
```



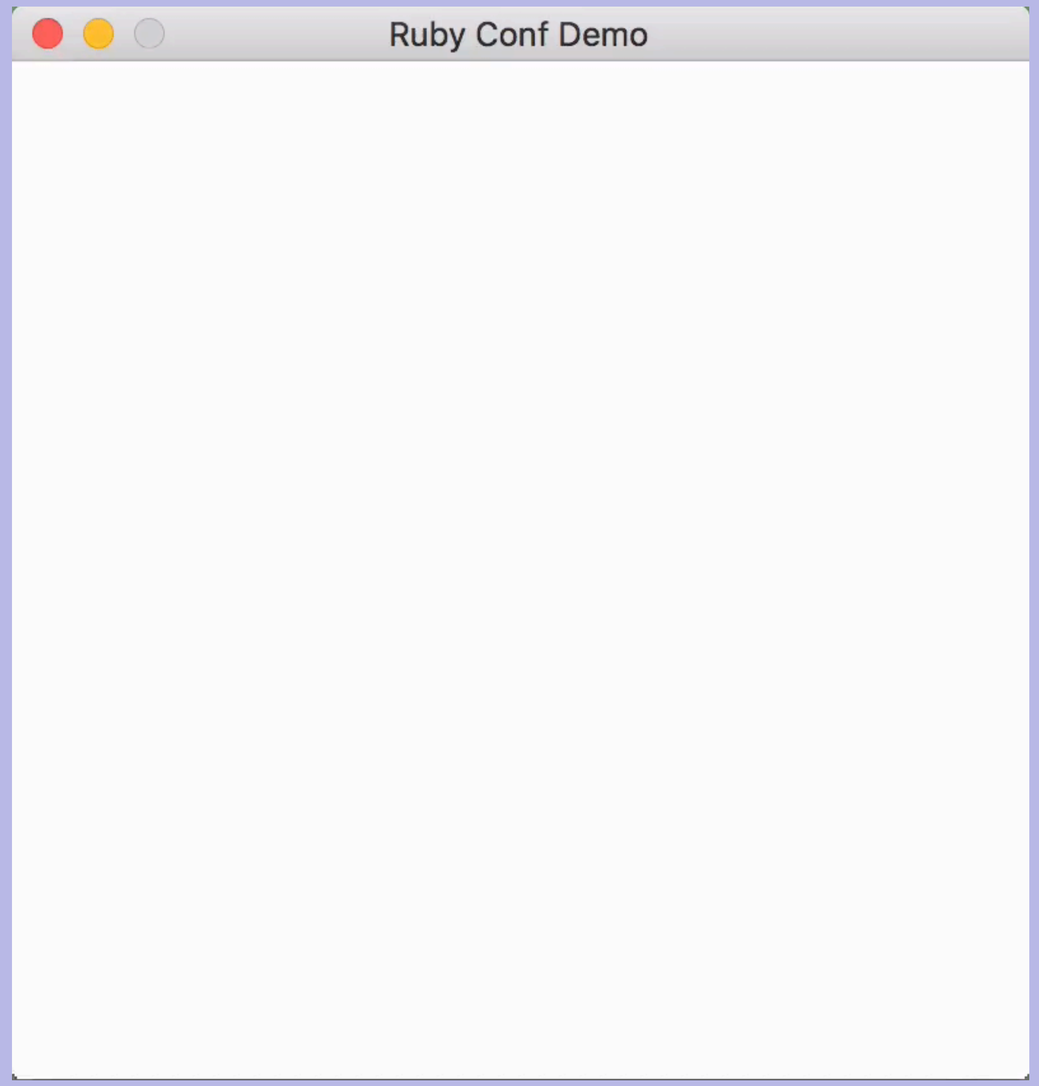
angle

```
angle w/2, h/2, 3*n, h/2-m, :red
```



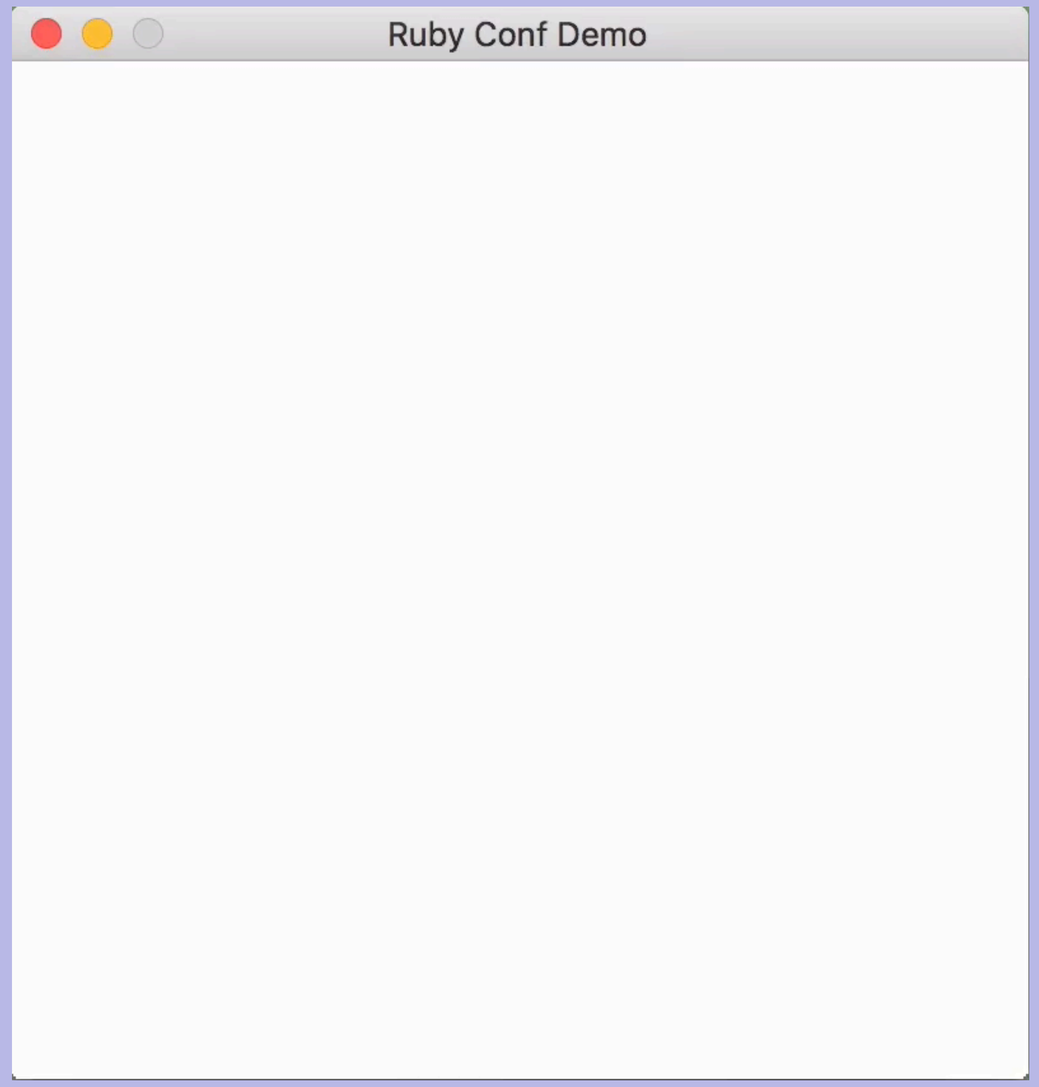
hline/vline

```
hline (3*n)%h, :red
vline (3*n)%w, :blue
```



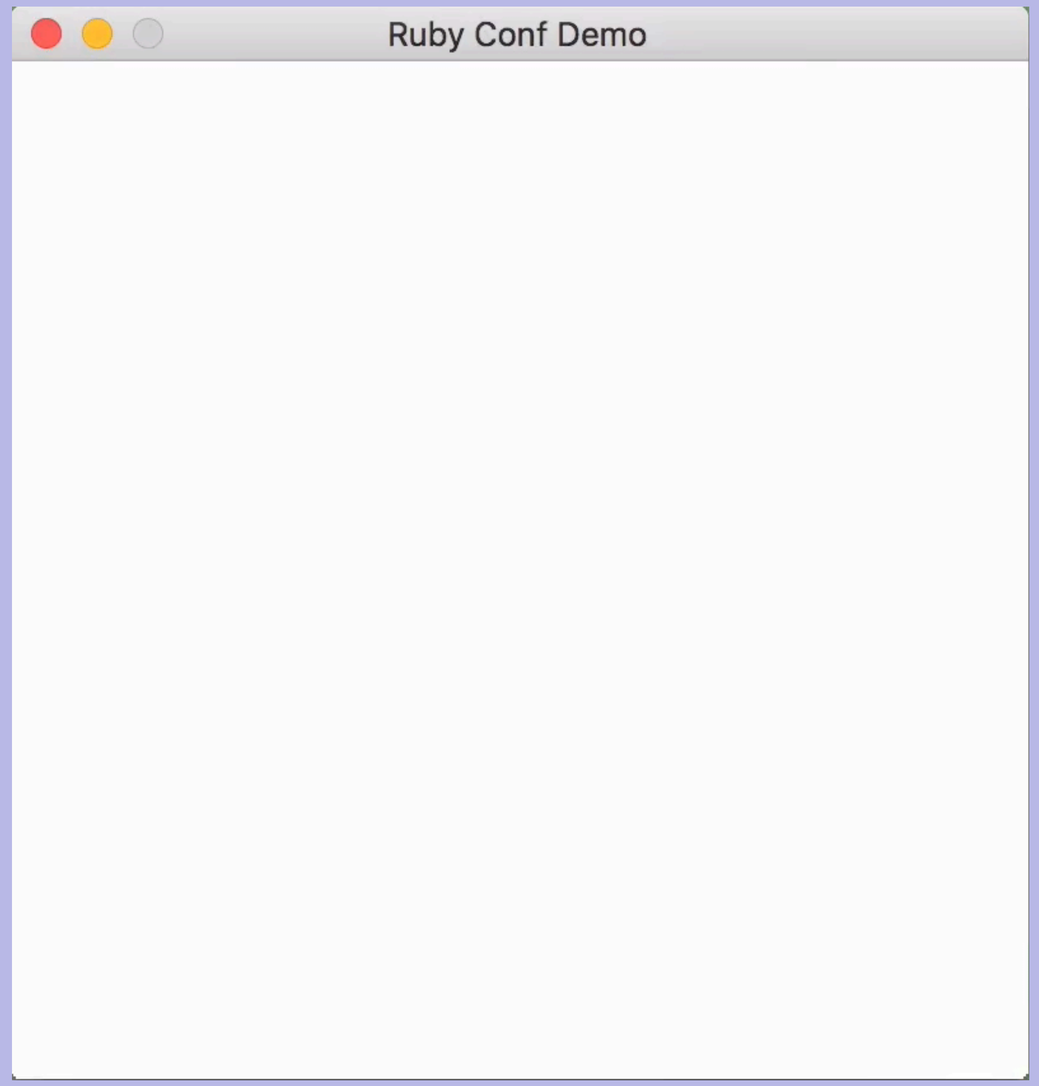
circle

```
circle rx, ry, r50, rc, rb
```



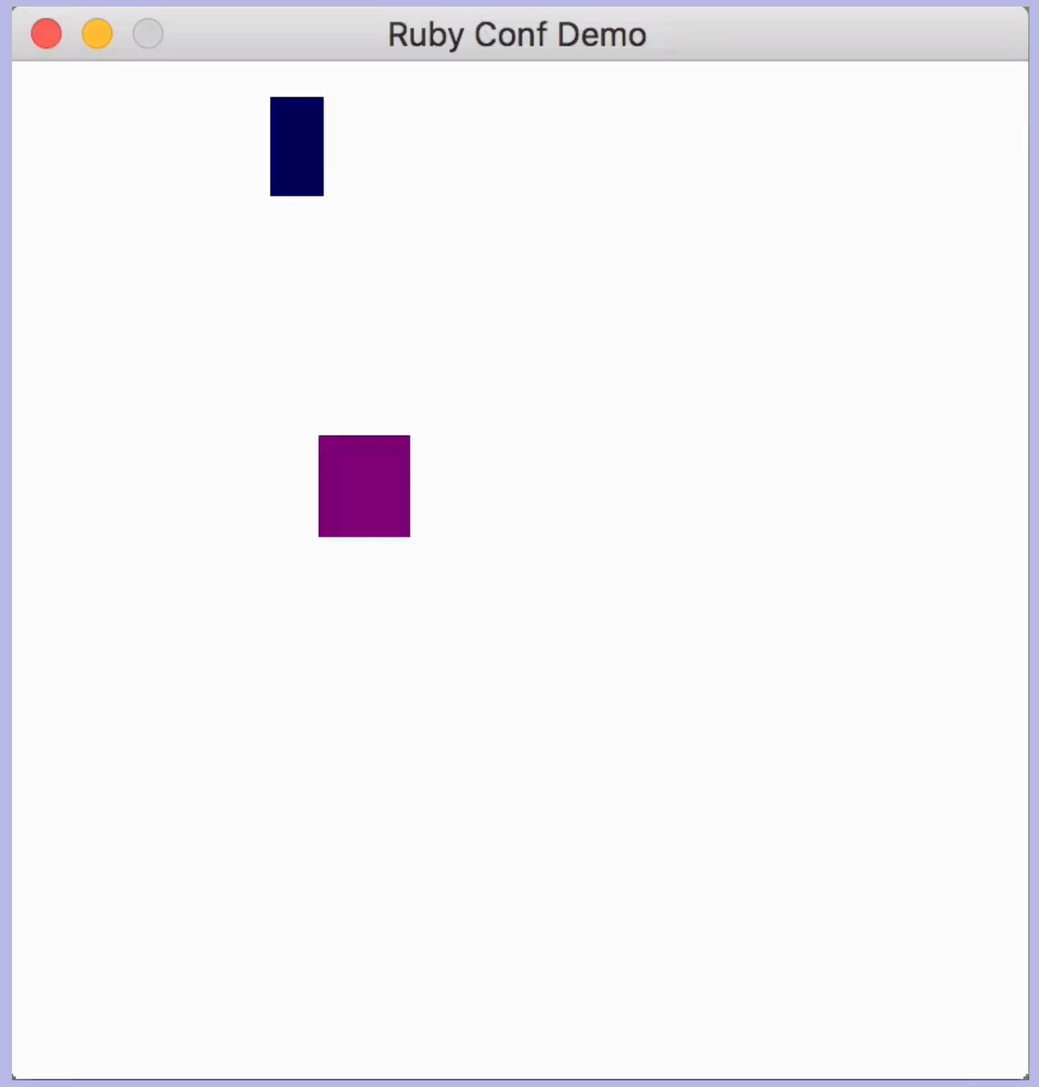
ellipse

```
ellipse rx, ry, r50, r50, rc, rb
```



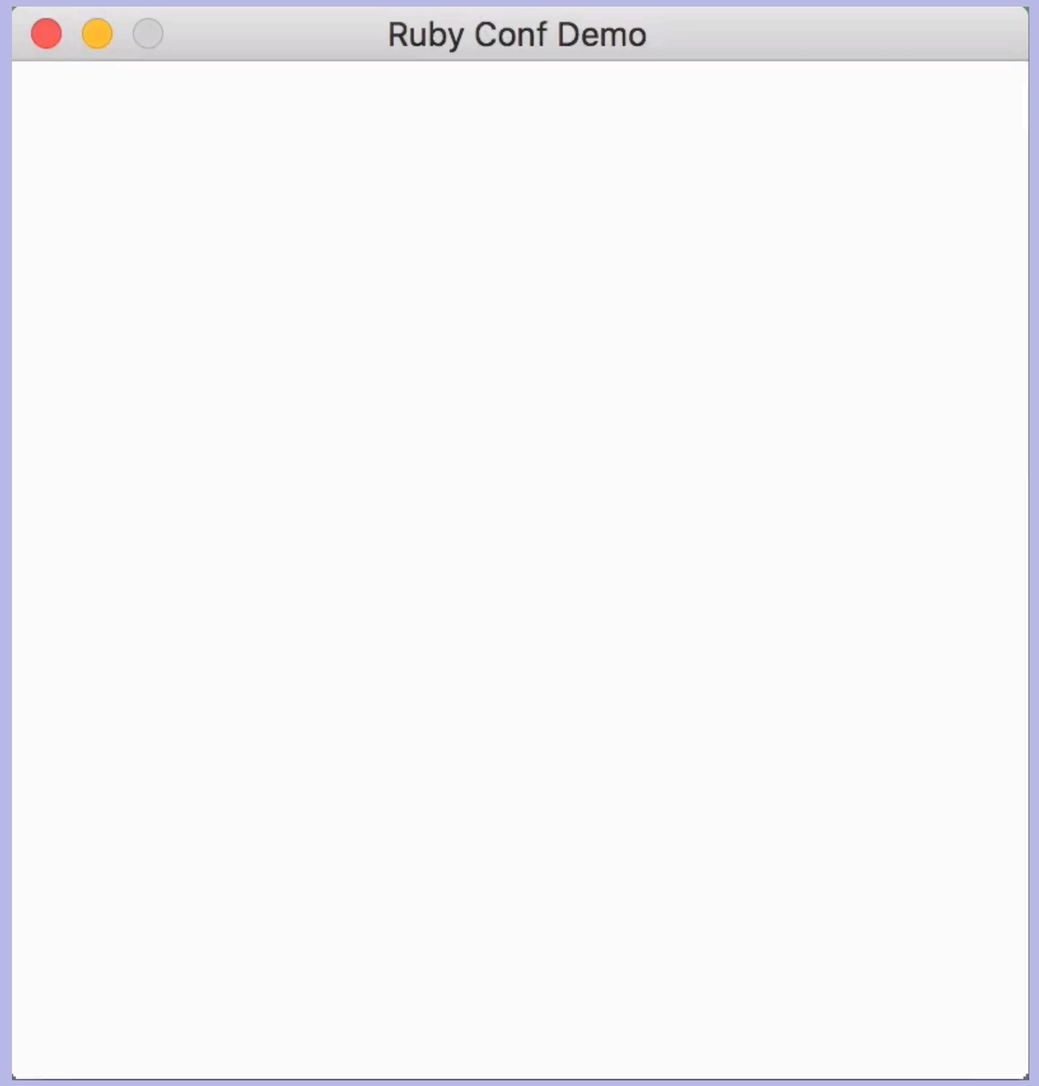
fast_rect

```
fast_rect rx, ry, r50, r50, rc
```



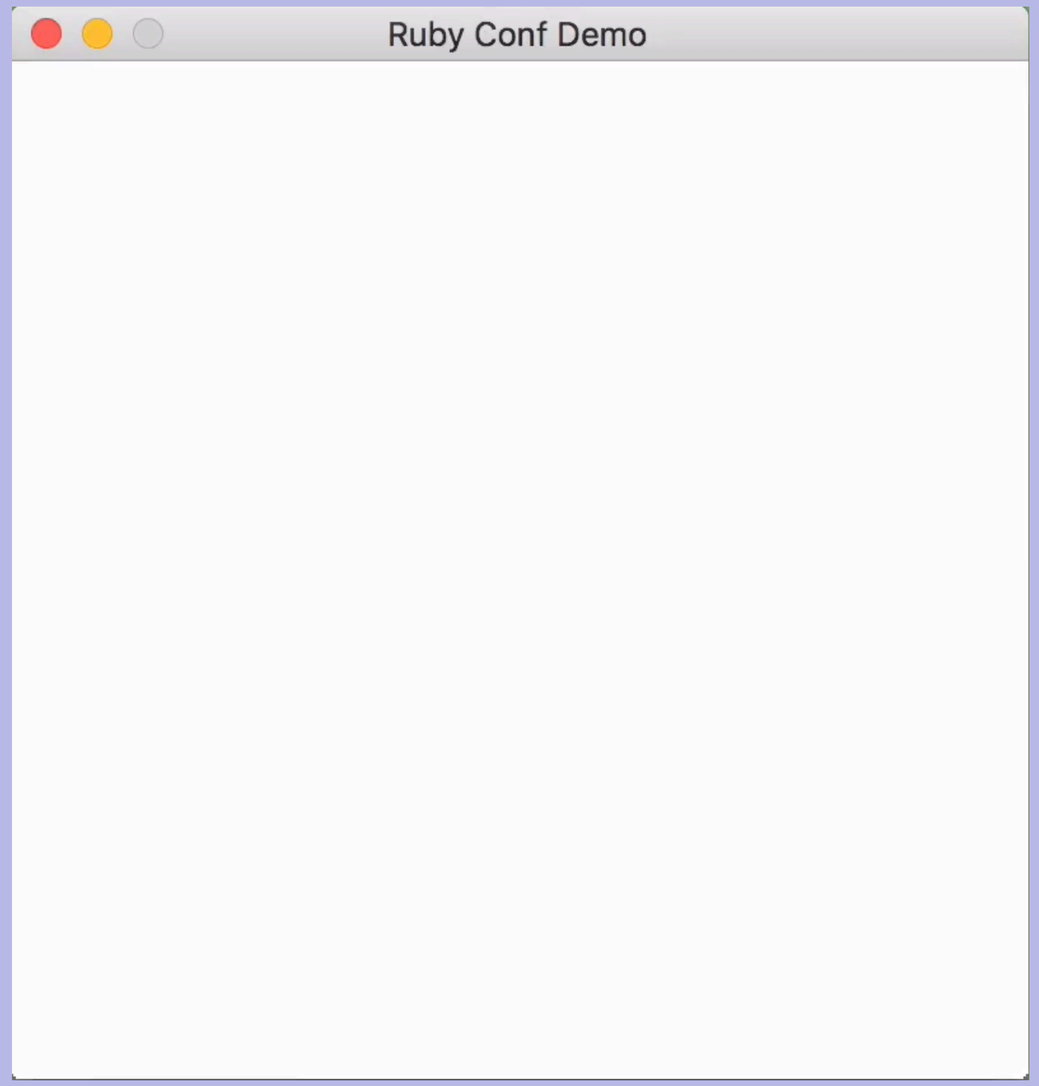
rect

```
rect rx, ry, r50, r50, rc, rb
```



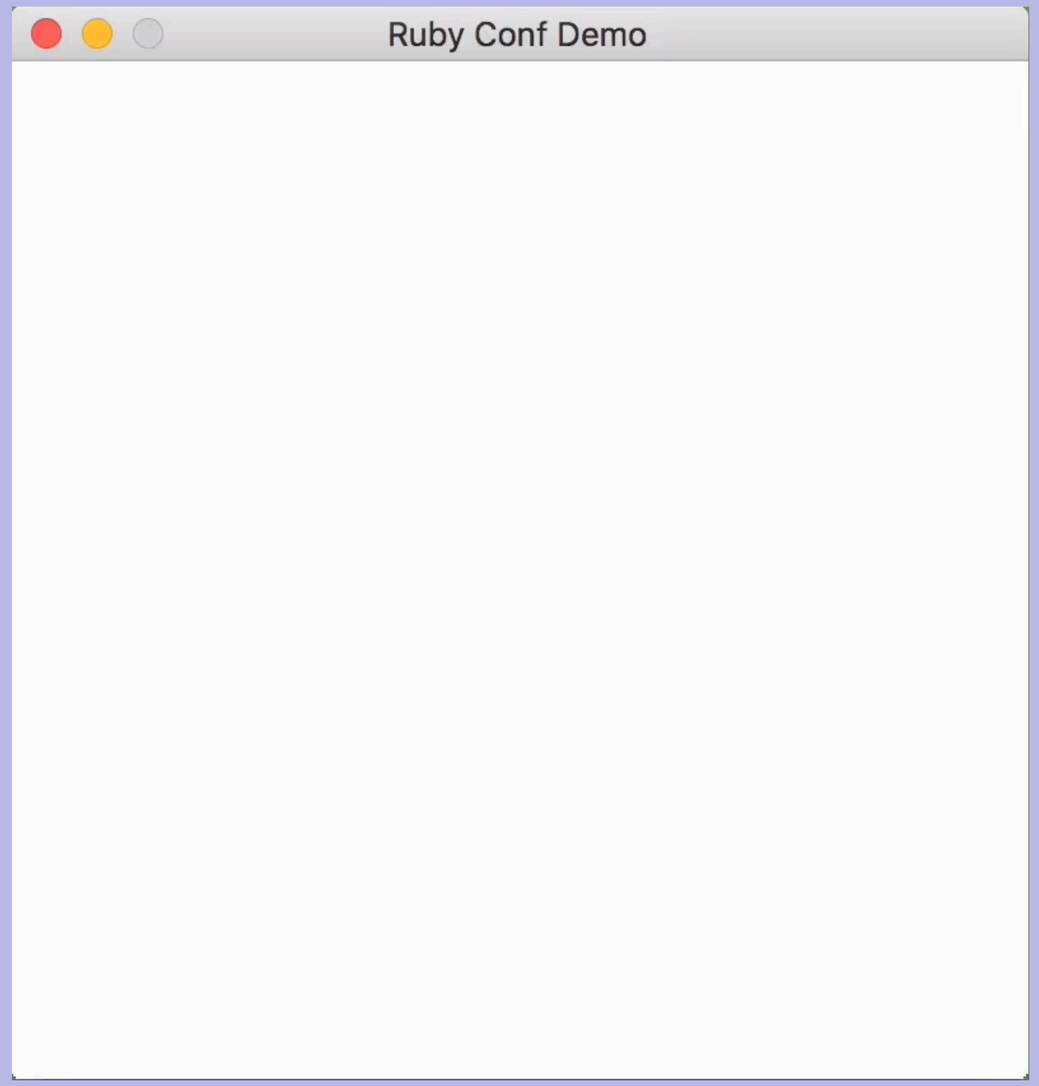
polygon

```
points = rand(3..8).times.map {
  [rx, ry]
}
polygon *points, rc
```



bezier

```
bezier(m,      m+n,    # p1
       2*m,    0,      # c1
       w-2*m,  h,      # c2
       w-m,    h-m-n,  # p2
       rc)
```



```
self.tank = sprite 40, 30 do
  rect 0,  0, 39, 29, :black
  rect 0,  4, 39, 21, :black

  line 0,  2, 39,  2, :black
  line 0, 27, 39, 27, :black
end

self.turret = sprite 41, 16 do
  # … more of the same
end

blit tank,   w/2, h/2, n*2
blit turret, w/2, h/2, n*3
```

sprite & blit

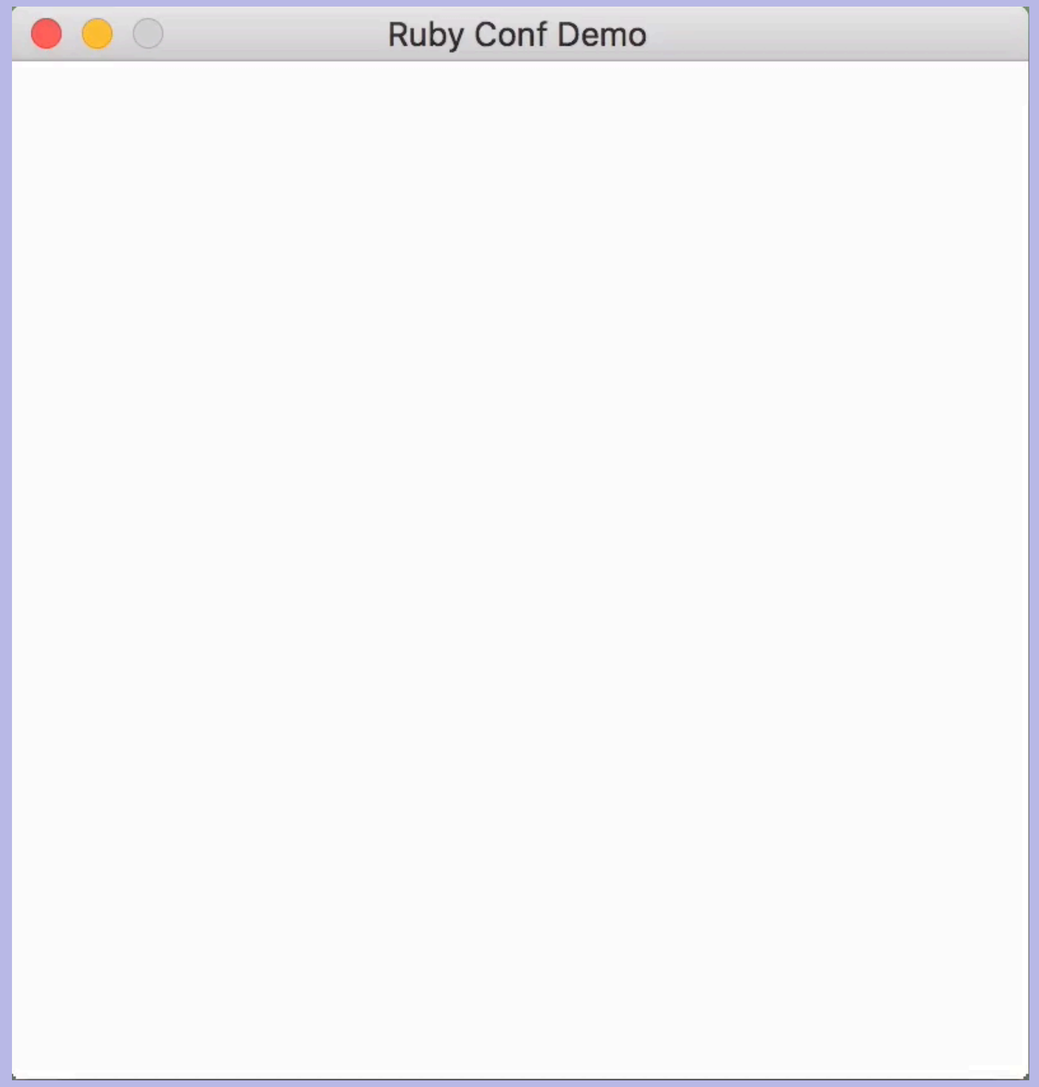
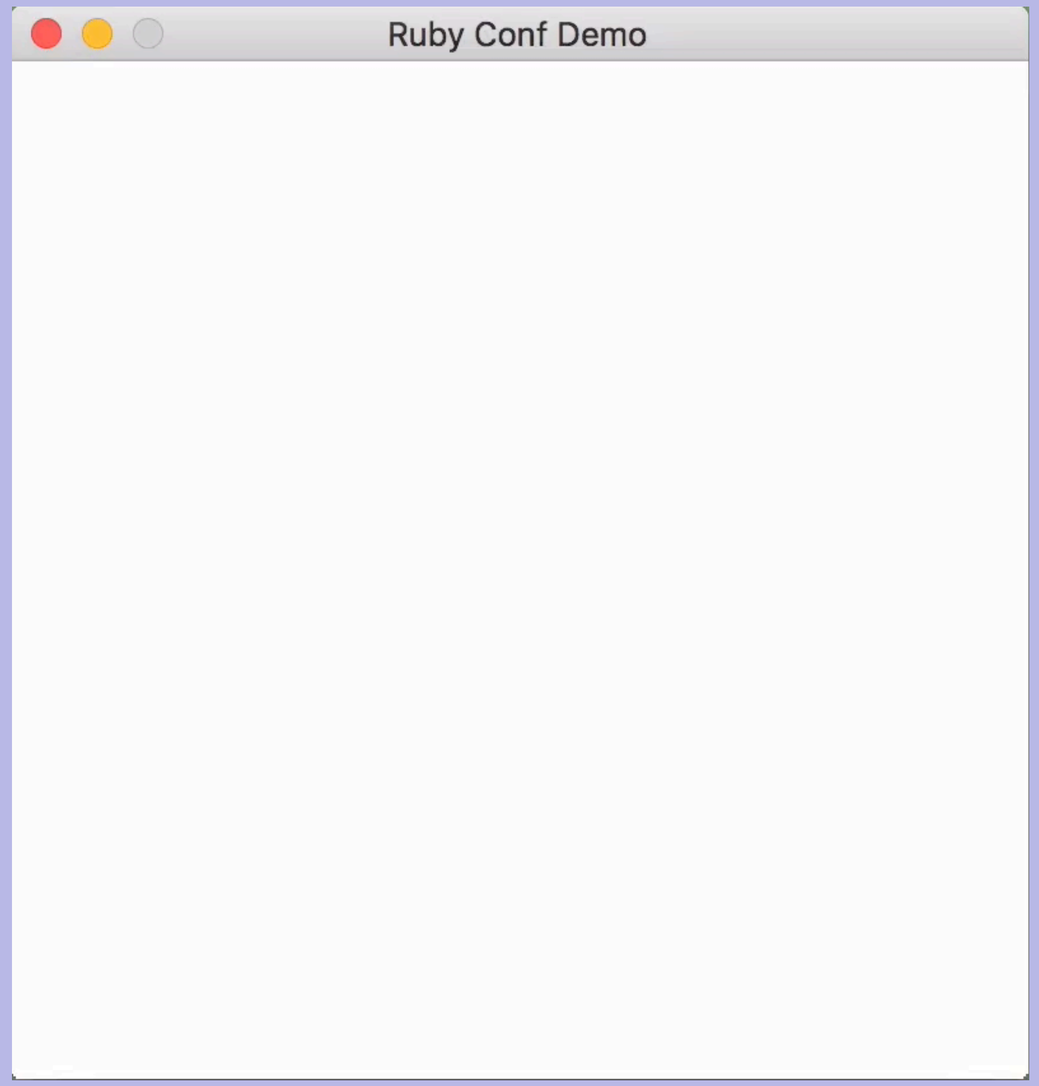


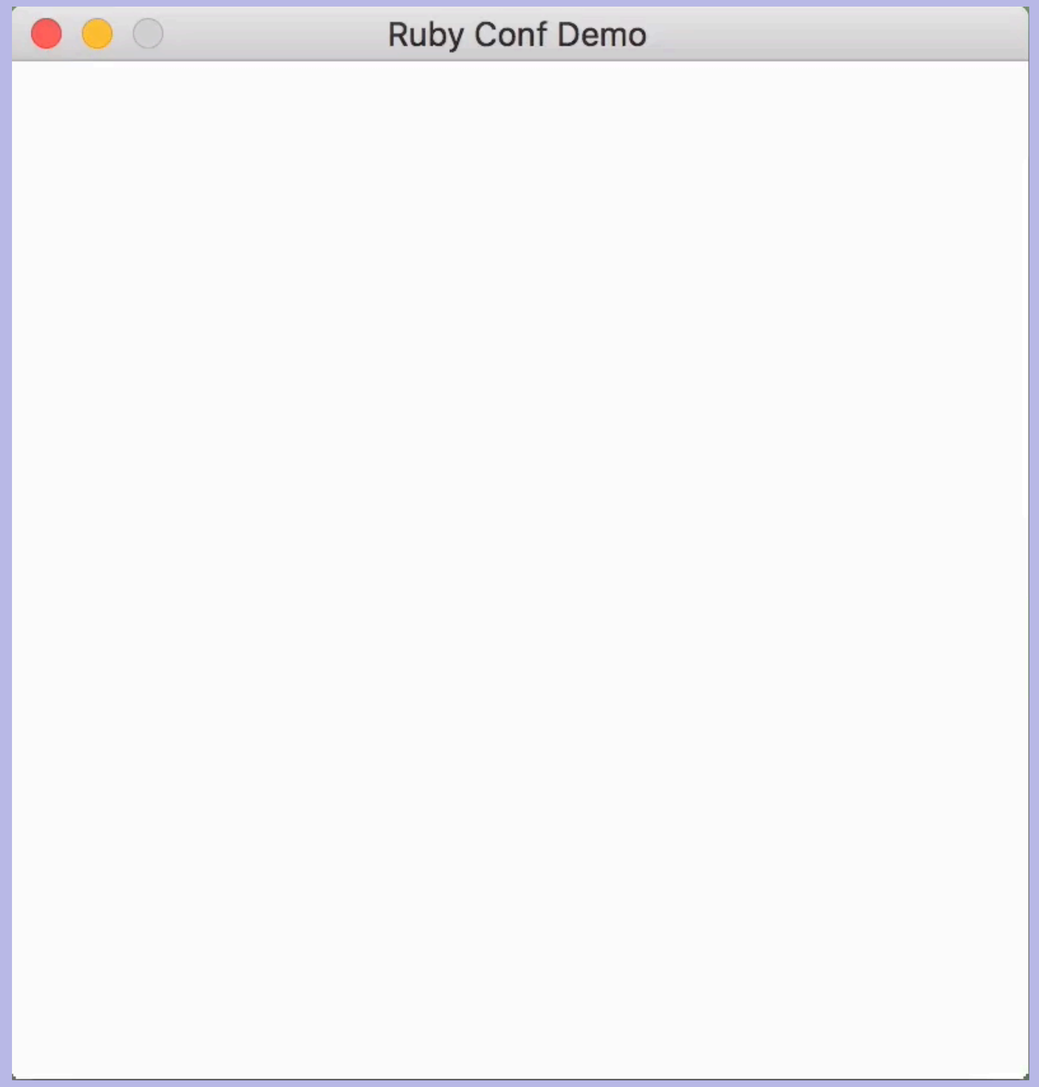
image & blit

```
self.tank   = image "body.png"
self.turret = image "turret.png"

blit tank,   w/2, h/2, n*2
blit turret, w/2, h/2, n*3
```

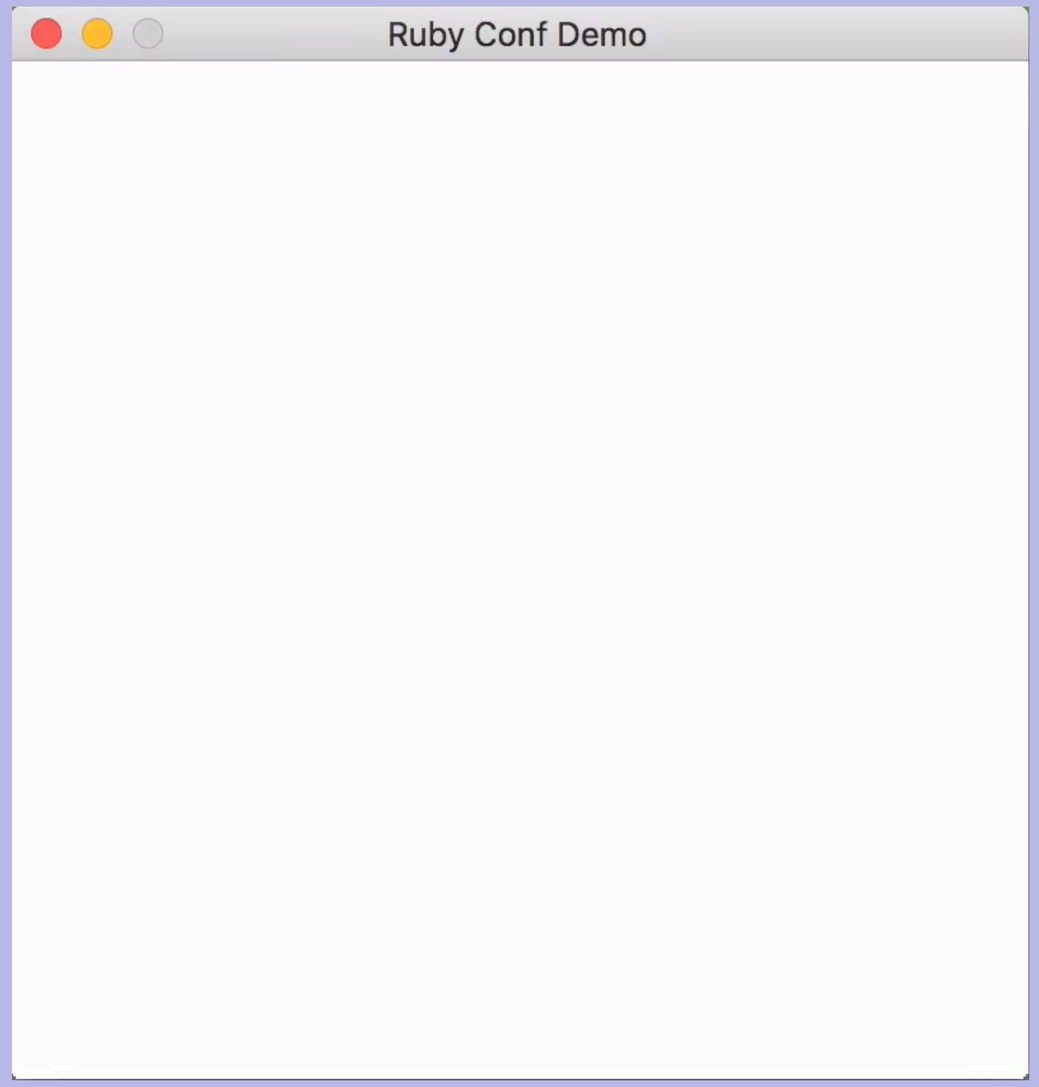


render_text & put

```
font = find_font("Livory", 32)
hello = render_text(
  "Hello RubyConf!”, :red, font)

put hello, w/2, h/2, -n
```

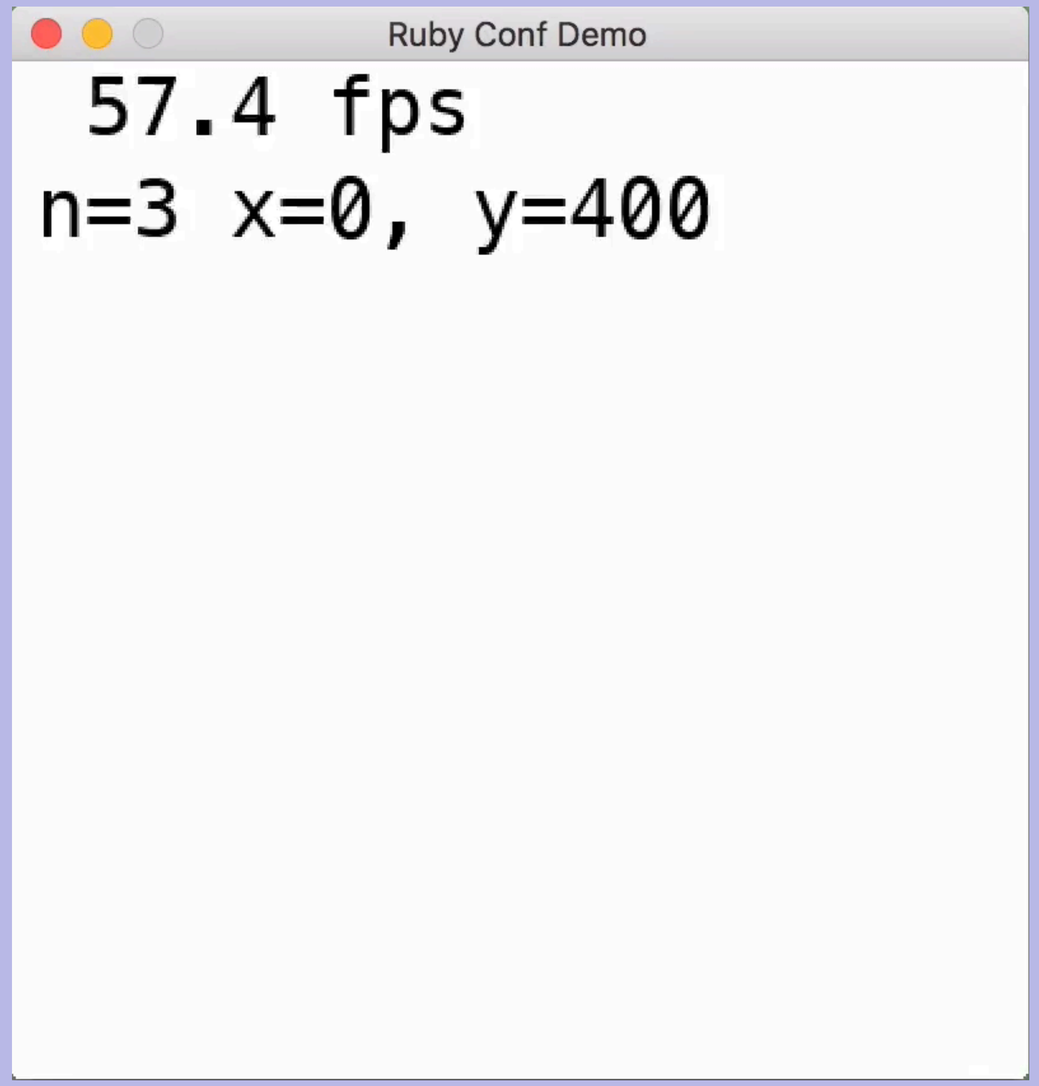


```
hello = "Hello RubyConf!"
text hello, m, n*30, rc
```



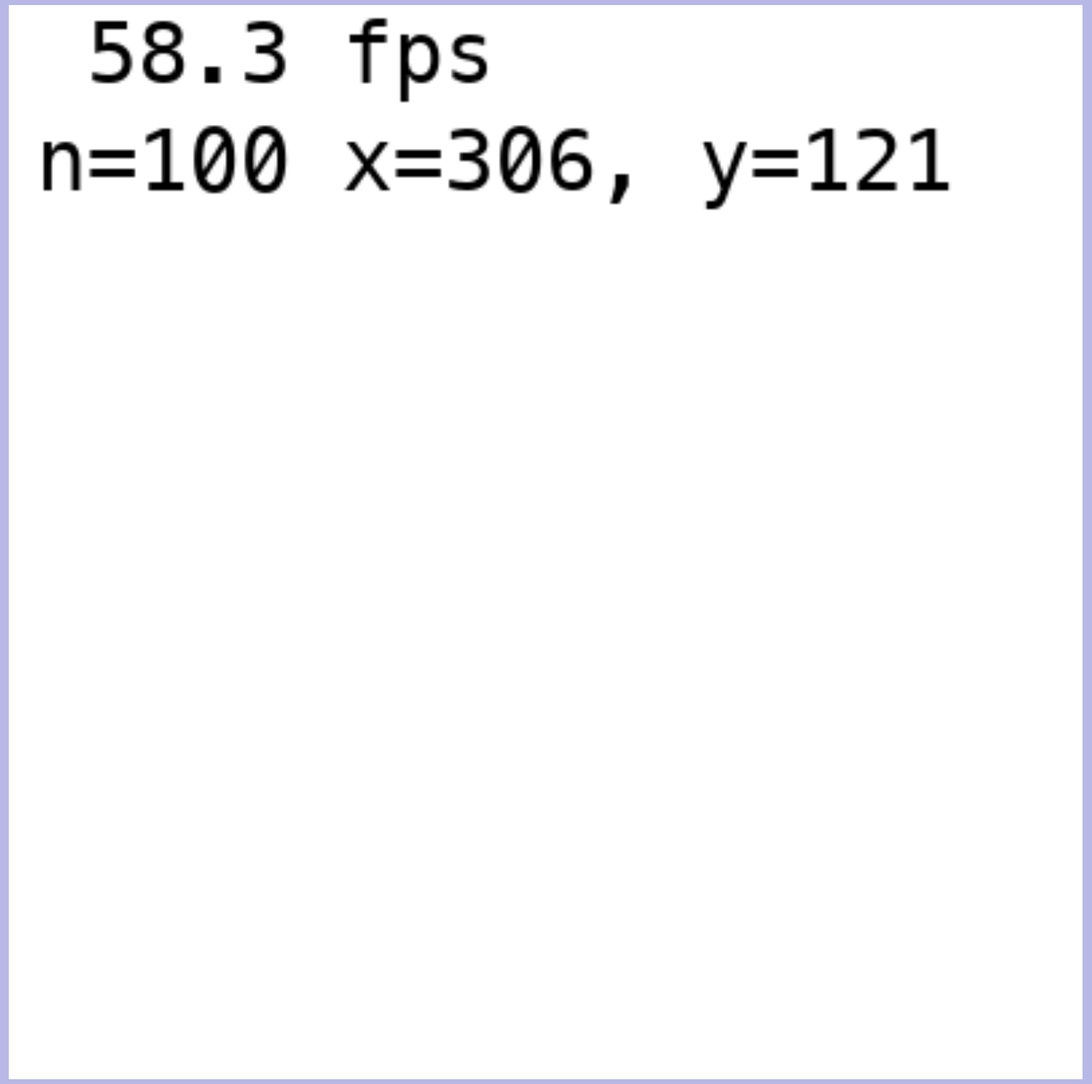
debug, fps, & mouse

```
fps n, :black
x, y, * = mouse
debug "n=%d x=%d, y=%d", n, x, y
```



```
x, y, * = mouse
debug "n=%d x=%d, y=%d", n, x, y
fps n, :black

save "debug.png" if n == 100
```

save

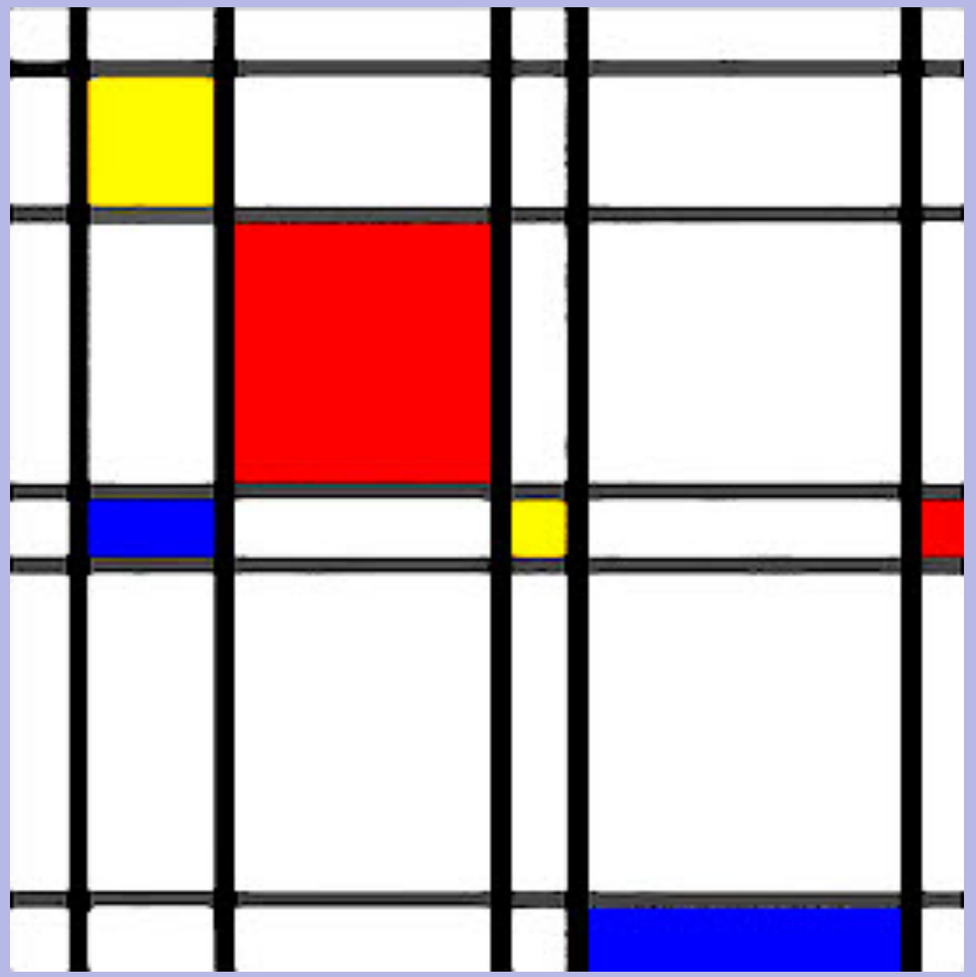


Generative Art

Mondrian-Style Generative Art

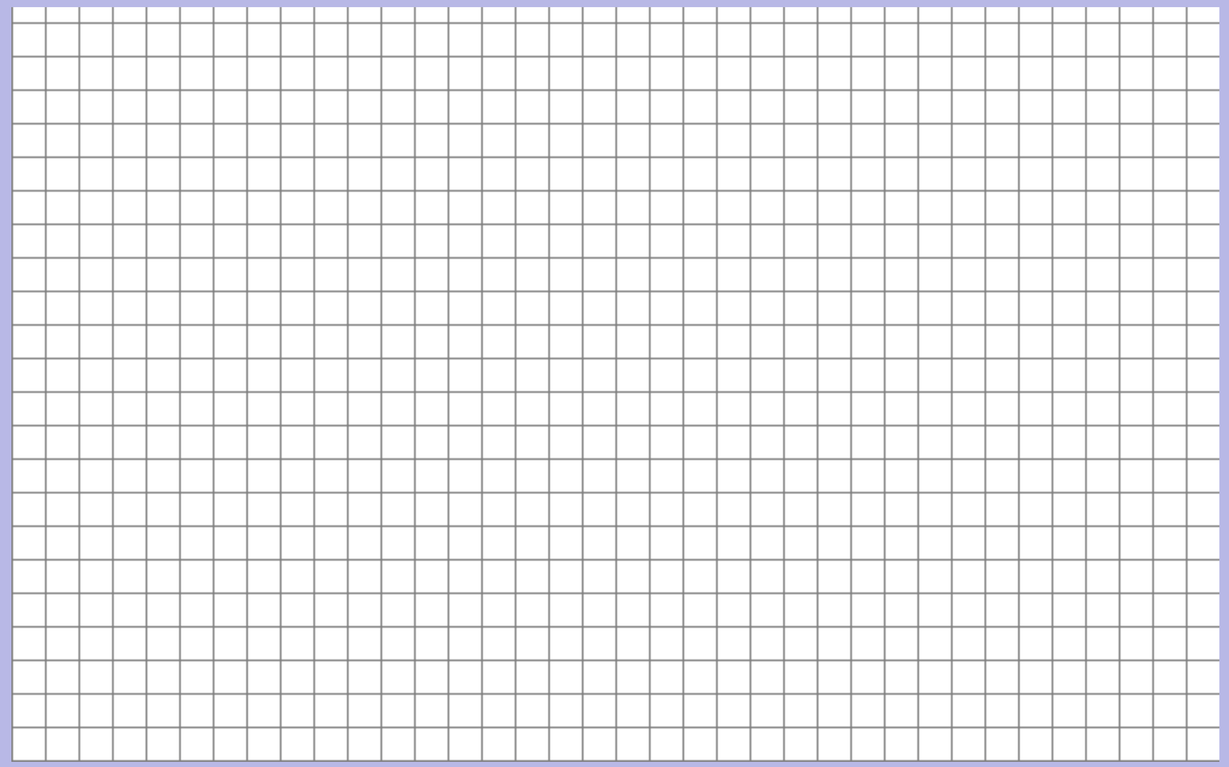
http://www.zenspider.com/ruby/2018/06/interesting-problems-mondrian.html

Sketch



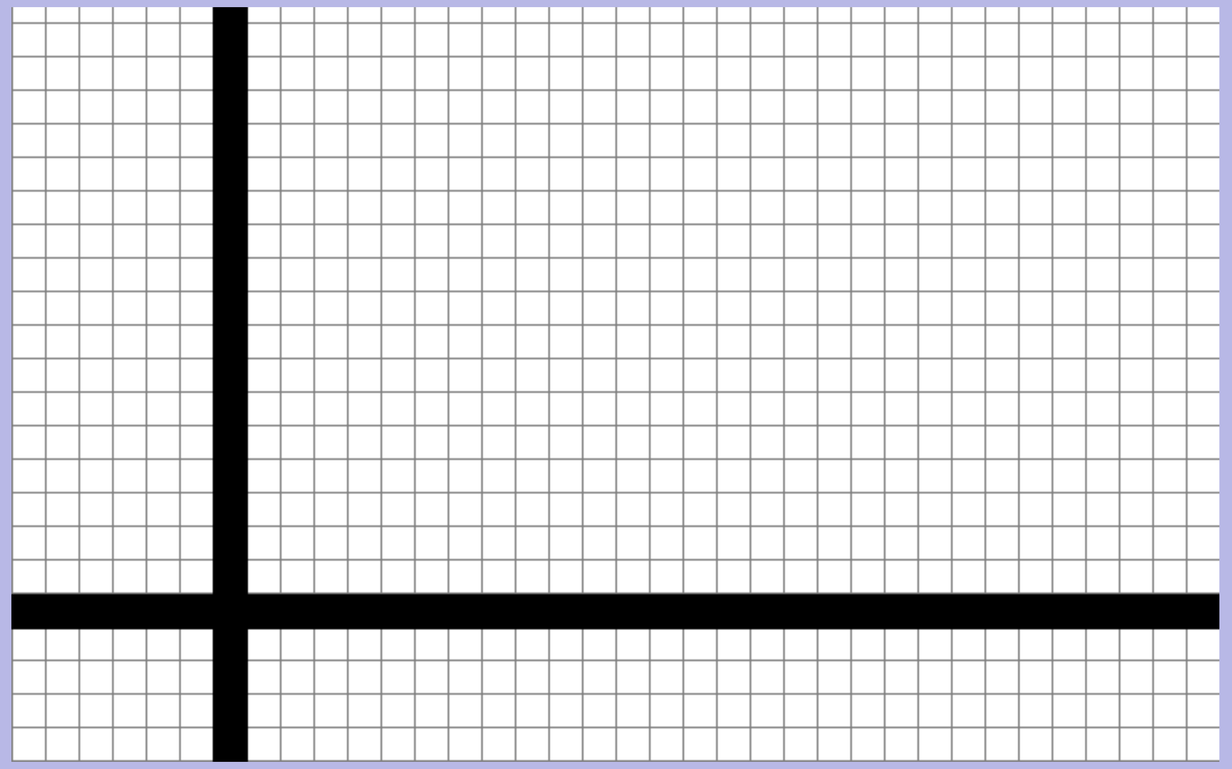
```
class Mondrian < G::D
  W = 20
  include WhiteBackground
  include DrawGrid; GRID_WIDTH=W

  def initialize
    super do
      art
    end
  end

  def art
    # where the work goes
  end

  alias r fast_rect
  # other helpers: rx, ry, etc
end
```

Blank



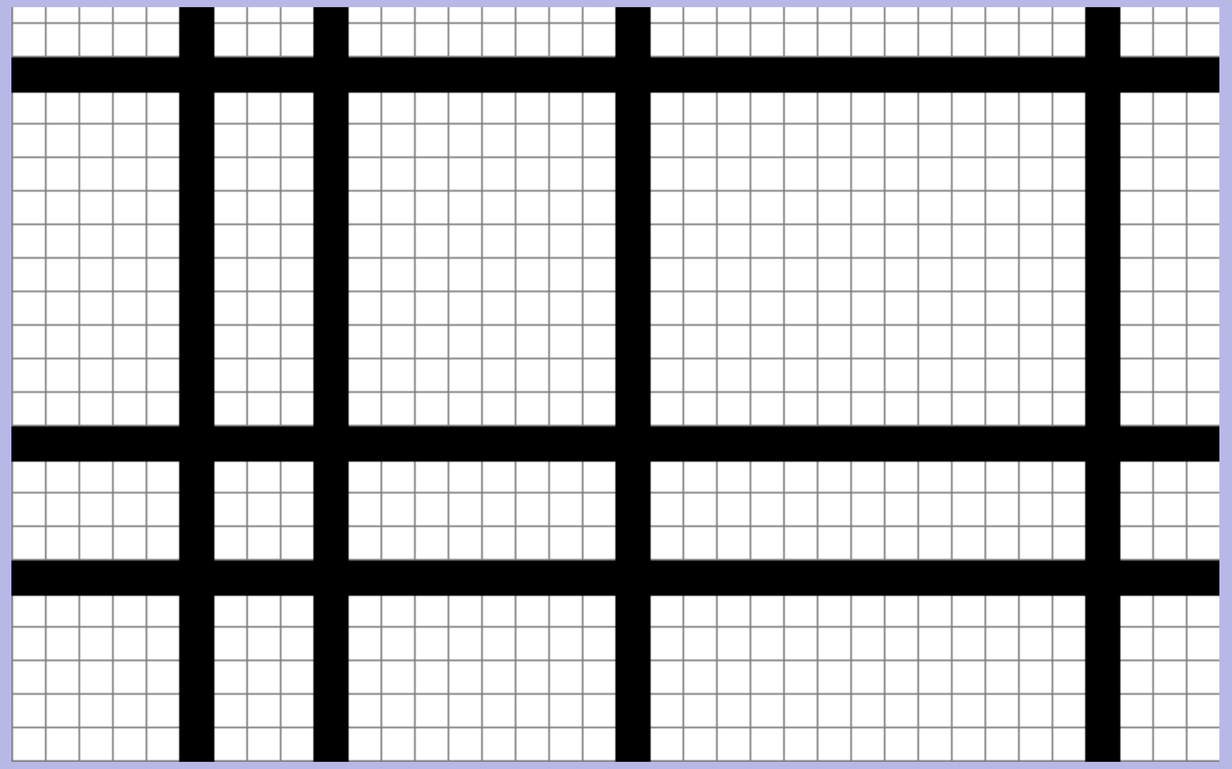
Black Rectangle

```
def art
  r 120,  0, W, h, :black
  r   0, 80, w, W, :black
end
```



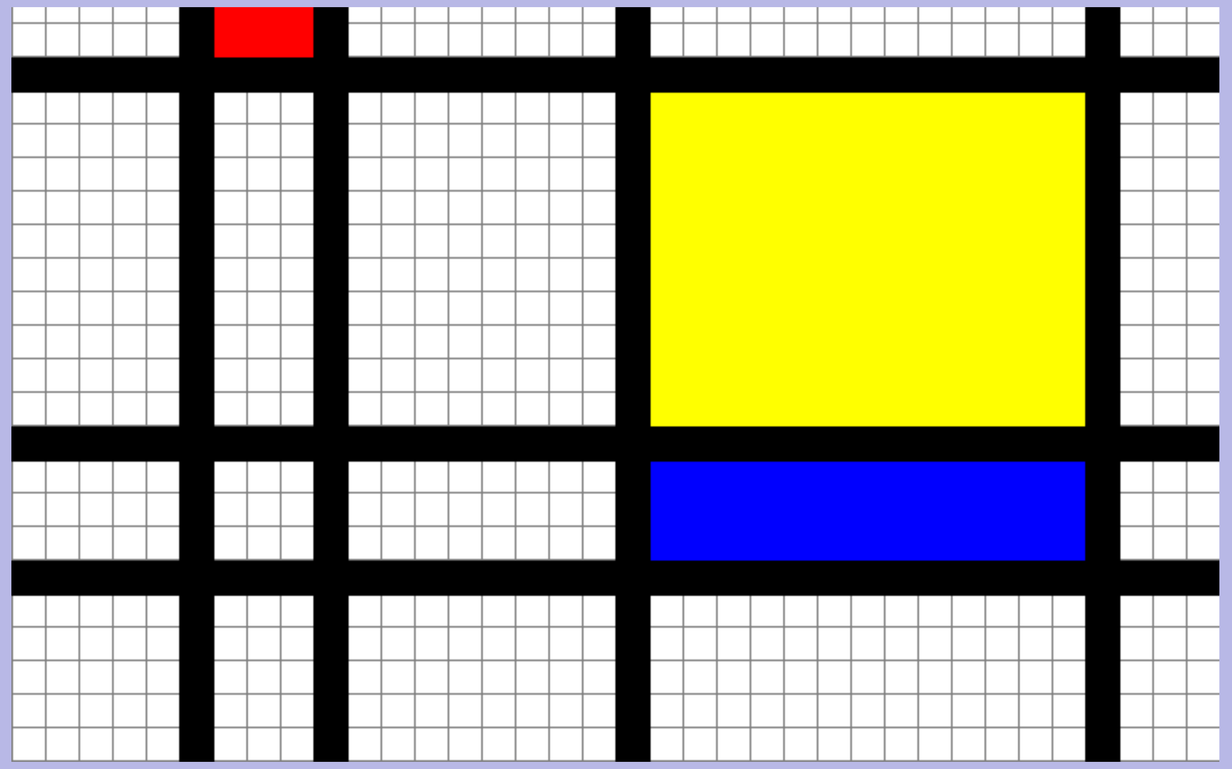
Stripes Spanning Window

```
def art
  xs = rx rand 2..5
  ys = ry rand 2..5

  xs.each do |x|
    r x, 0, W, h, :black
  end

  ys.each do |y|
    r 0, y, w, W, :black
  end
end
```

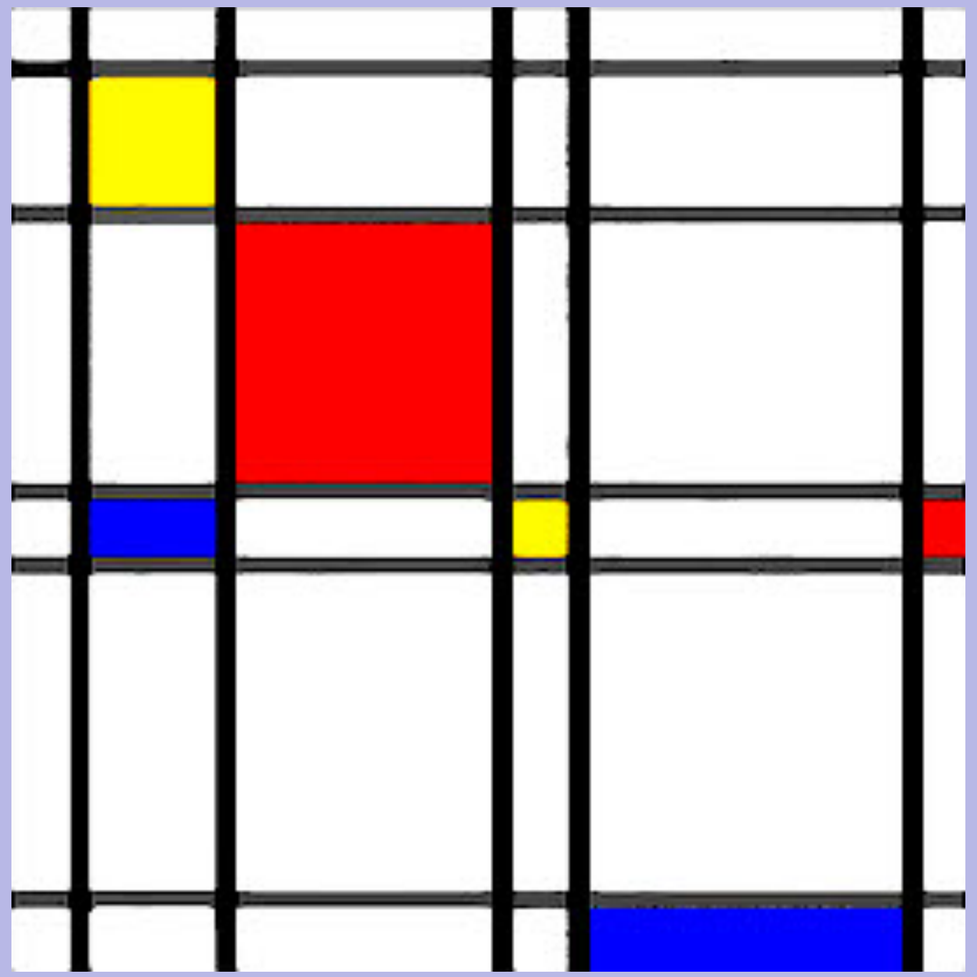


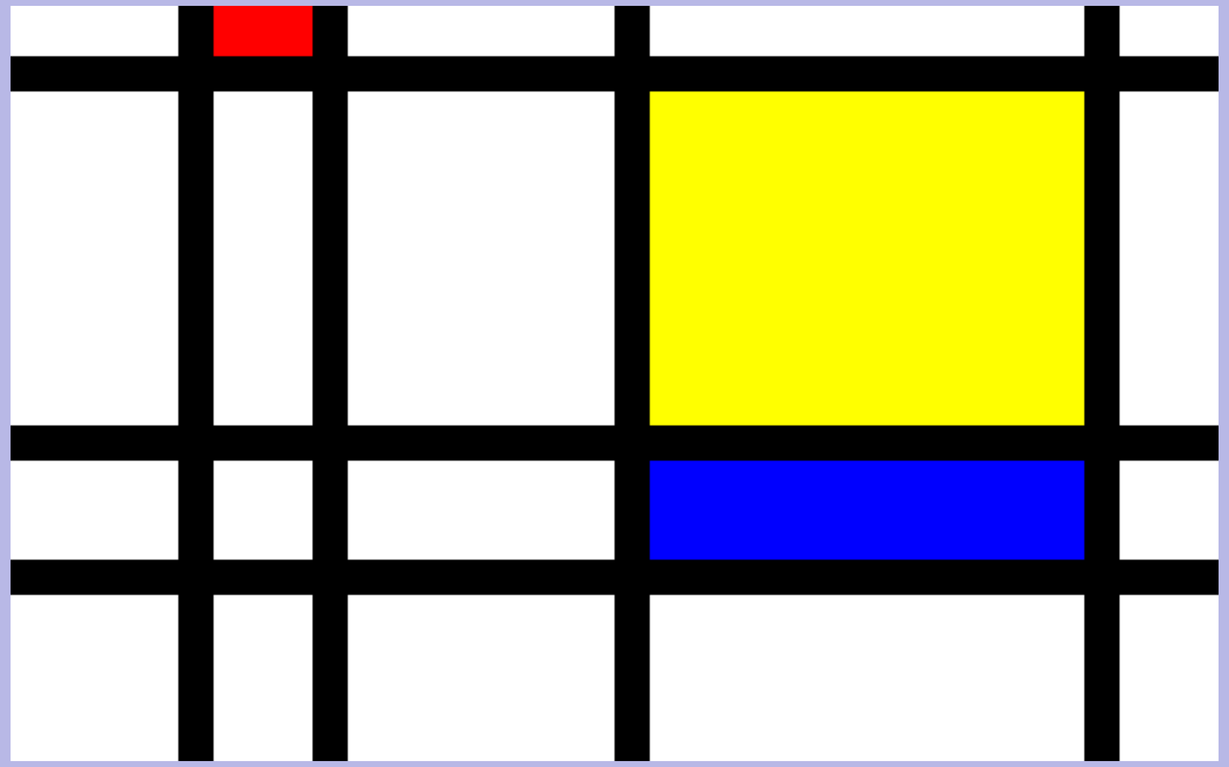
Filled Regions

```
COLOR = [:red, :yellow, :blue]

def regions xs, ys
  rand(1..5).times.each do
    xi      = rand(xs.size-1)
    yi      = rand(ys.size-1)
    x0, x1 = xs[xi..xi+1]
    y0, y1 = ys[yi..yi+1]
    c       = COLOR.sample

    r x0, y0, x1-x0, y1-y0, c
  end
end
```



Comparing





Visualization

Plotting

Too Much Data

Message Size / Time

Gathering Data is Easy

Just Looking Doesn't Help

```
>> d = Marshal.load File.read "plot.cache"; nil
=> nil
>> d.size
=> 954165
>> d.first 5
=> [[100, 0.5], [30, 0.1], [46, 0.6], [439, 0.6], [215, 0.1]]
>> d.transpose.map(&:min)
=> [30, 0.0]
>> d.transpose.map(&:avg)
=> [151, 0.240460507354598]
>> d.transpose.map(&:max)
=> [1740, 88.8]
>> dsr = d.map { |a| a.map { |f| f.round 2 } }; nil
=> nil
>> h = d.count_by(&:itself); nil
>> pp h.sort_by { |(k,v), c| [-v, -c, k] }.first 5; nil
[[[20.4, 500.0], 1],
 [[13.51, 479.17], 1],
 [[20.11, 462.84], 1],
 [[20.4, 458.9], 1]]
=> nil
```

But Plotting It?

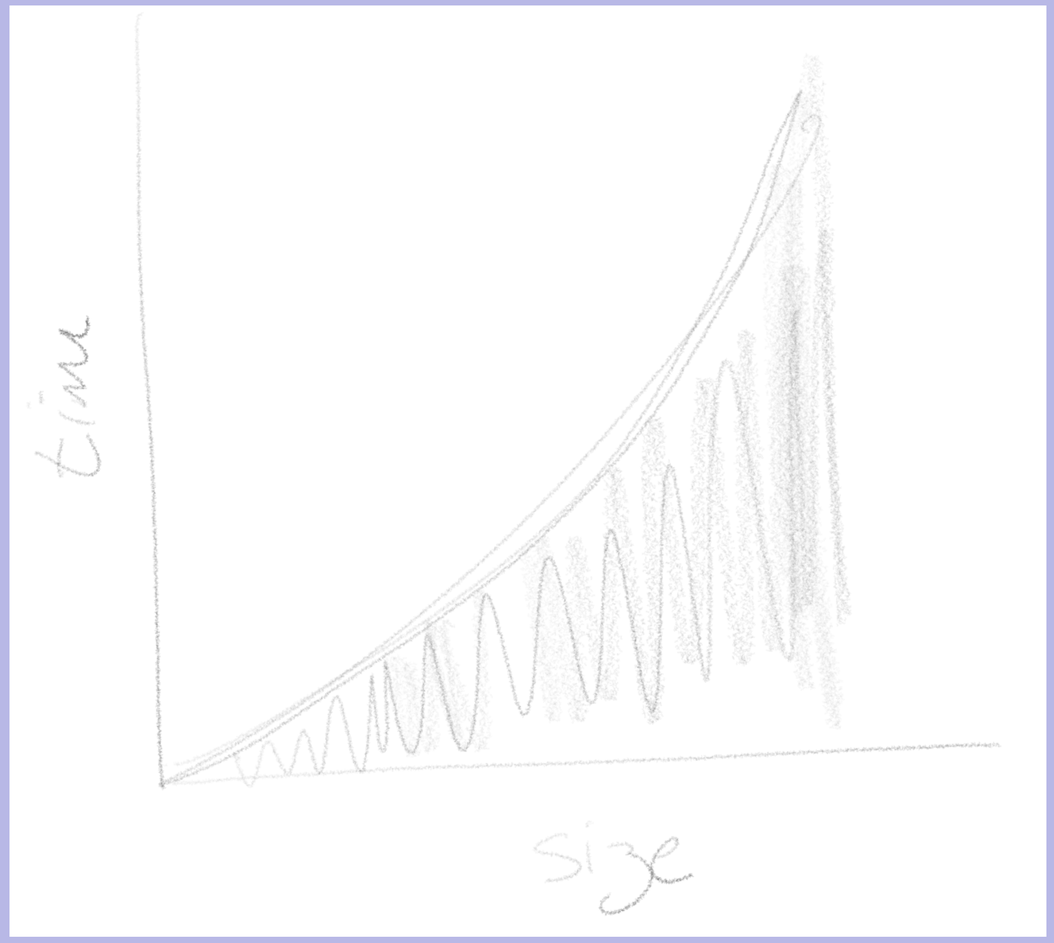
R?

Excel?

Or...?

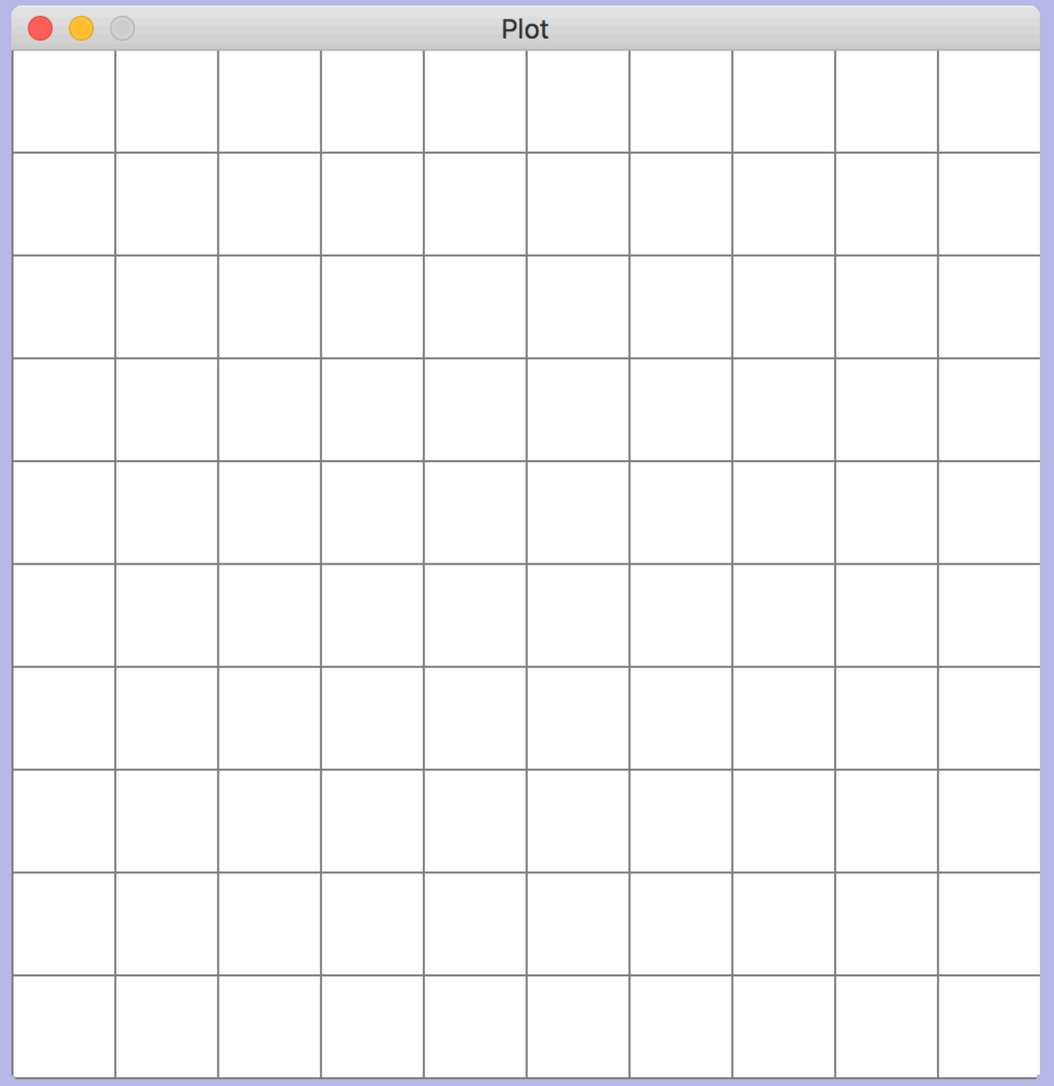
Just Use Ruby™!

Sketch



```
class Plot < G::S
  MAX = 500

  include WhiteBackground
  include DrawGrid
  GRID_WIDTH = 50

  def initialize data
    super MAX, MAX
  end

  def draw n
    super
  end
end
```

Blank Canvas

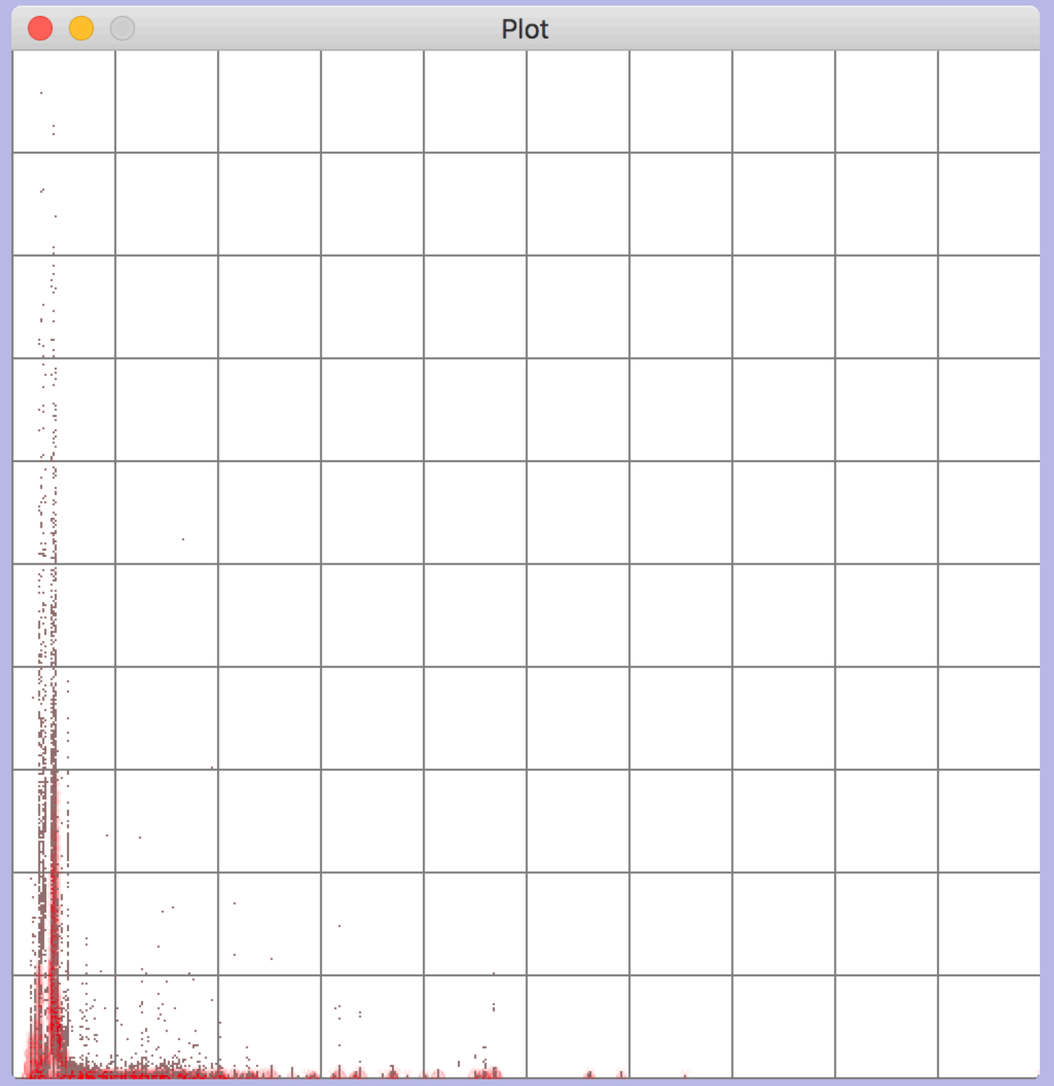


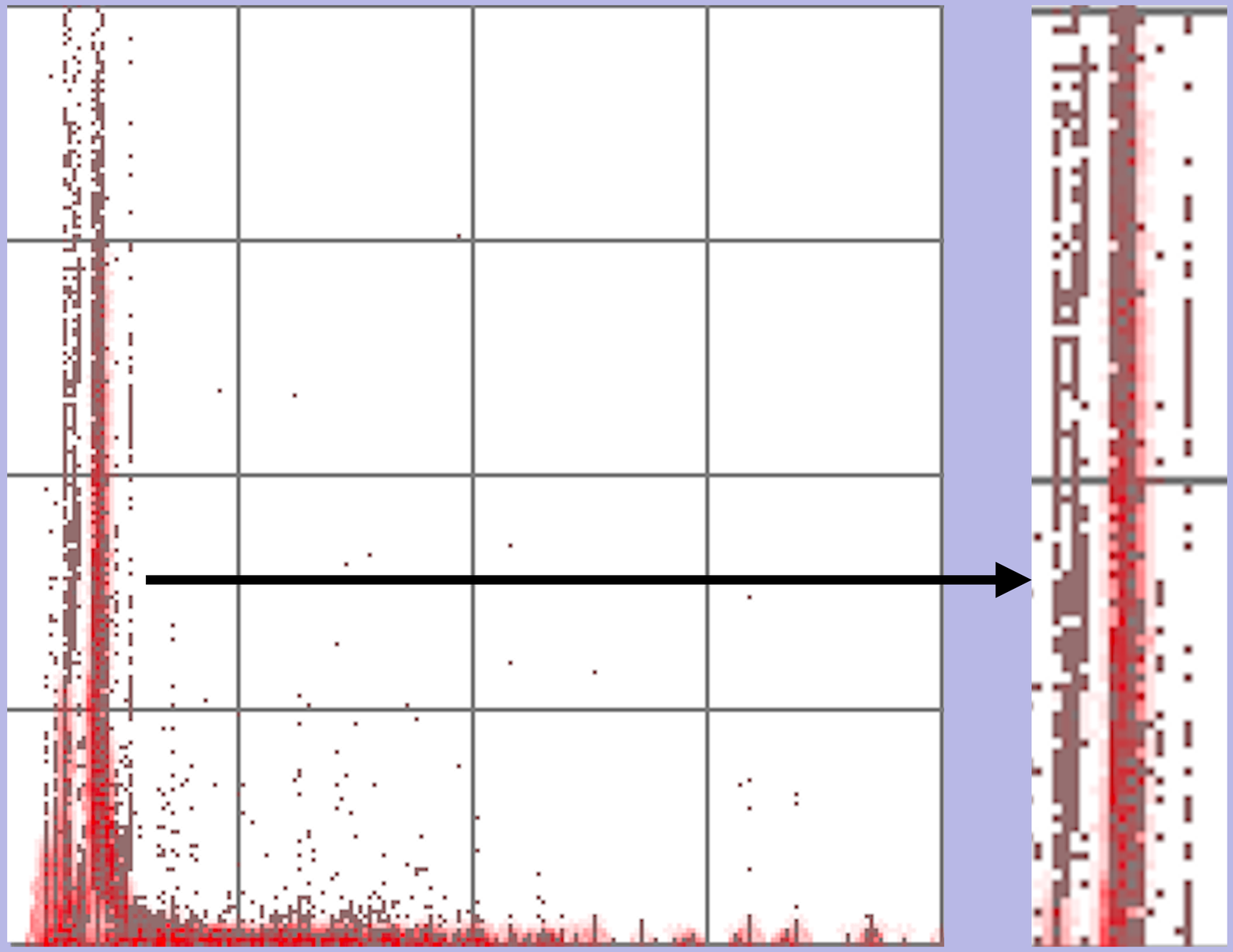
Prepare Data

```
attr_accessor :data

def initialize d
  super MAX, MAX

  register_color :point, 255, 0, 0, 16 # very translucent red

  d = scale_to d                       # scale to window
  self.data = d.group_by(&:itself) # group by values

  data.each do |k, v|
    data[k] = Math.log(v.size)     # count groups, log scale
  end
end
```

Visualize

```
def draw n
  super

  data.each do |(x, y), c|
    point  x, y,    :gray
    circle x, y, c, :point, :fill
  end
end
```





Extend

Experiment!

Simulations

Langston's Ants

https://en.wikipedia.org/wiki/Langton's_ant

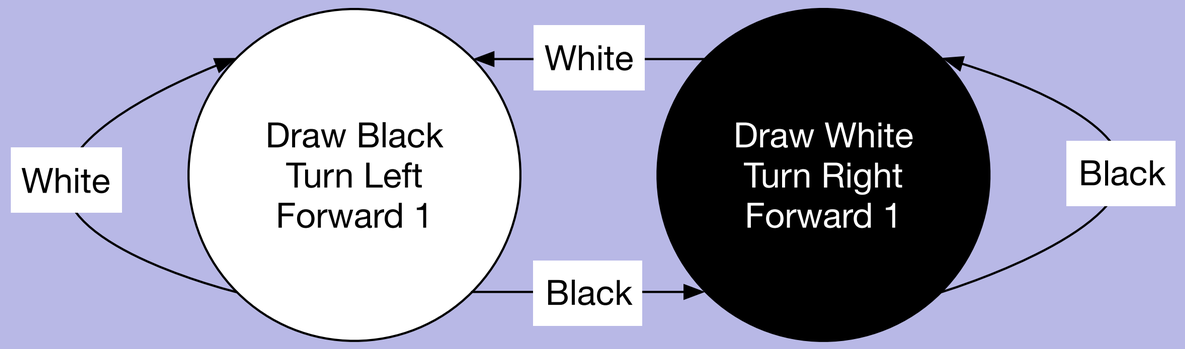
Simple Rules

if ⚪: draw ⚫, turn ↰

if ⚫: draw ⚪, turn ↱

move one

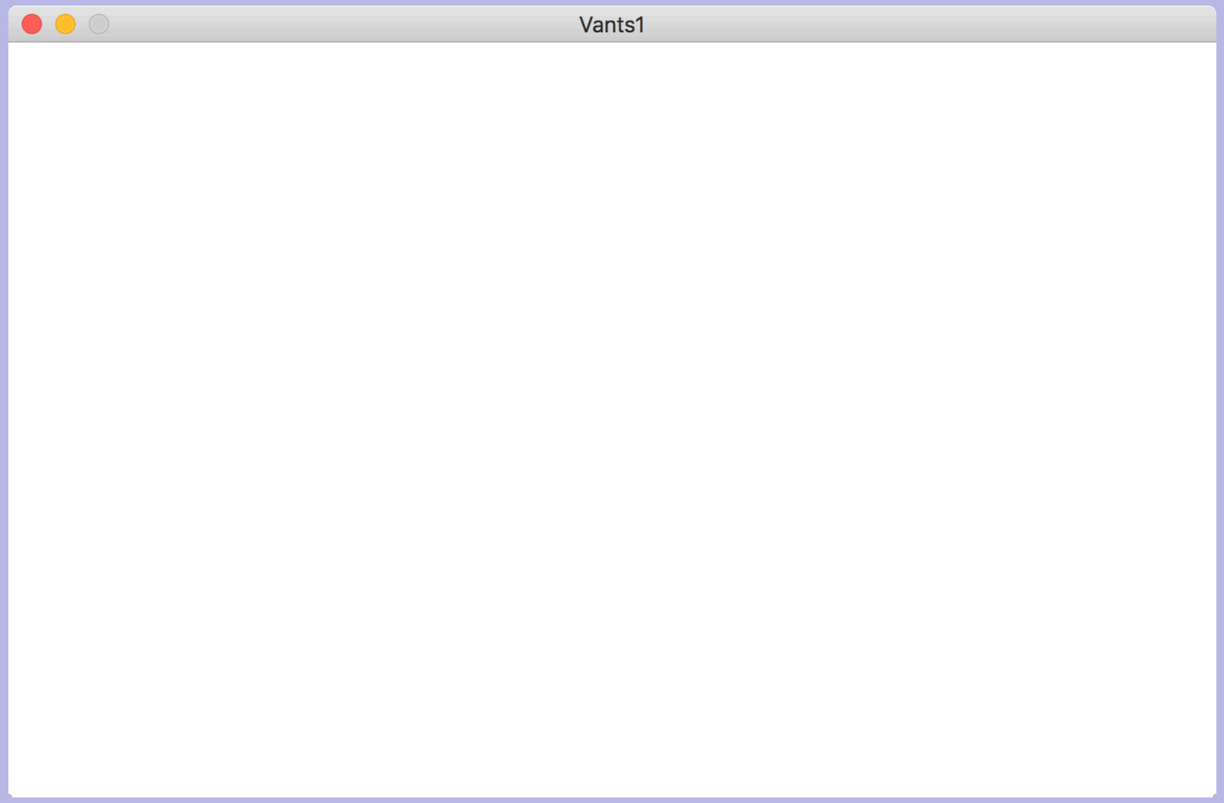
repeat

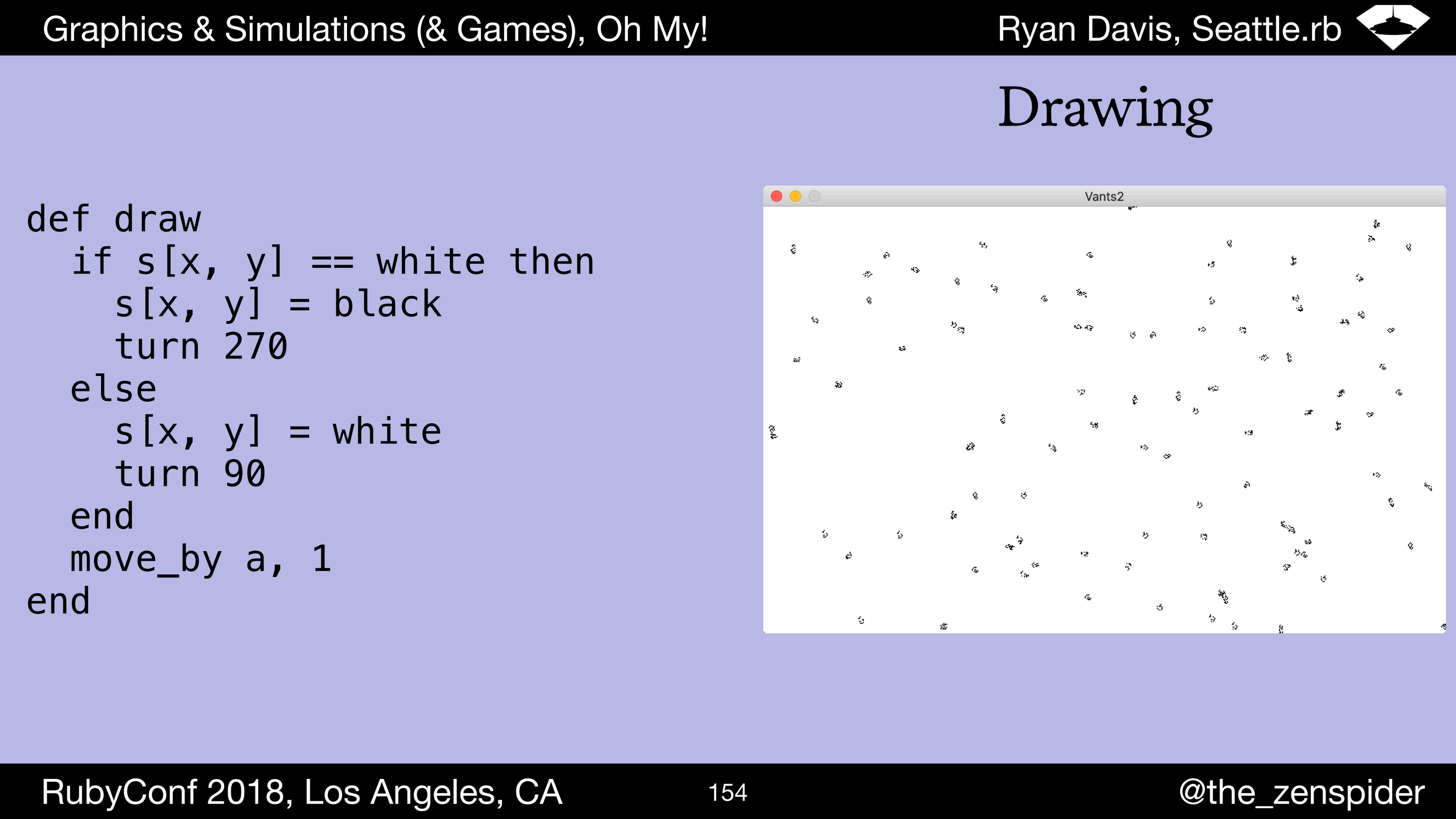


Sketch

```
class Vants0 < Graphics::Drawing
  include WhiteBackground

  def initialize
    super

    register_bodies populate Vant
  end

  # …other stuff…

  class Vant < Graphics::Body
    COUNT = 100
  end
end
```

Blank canvas

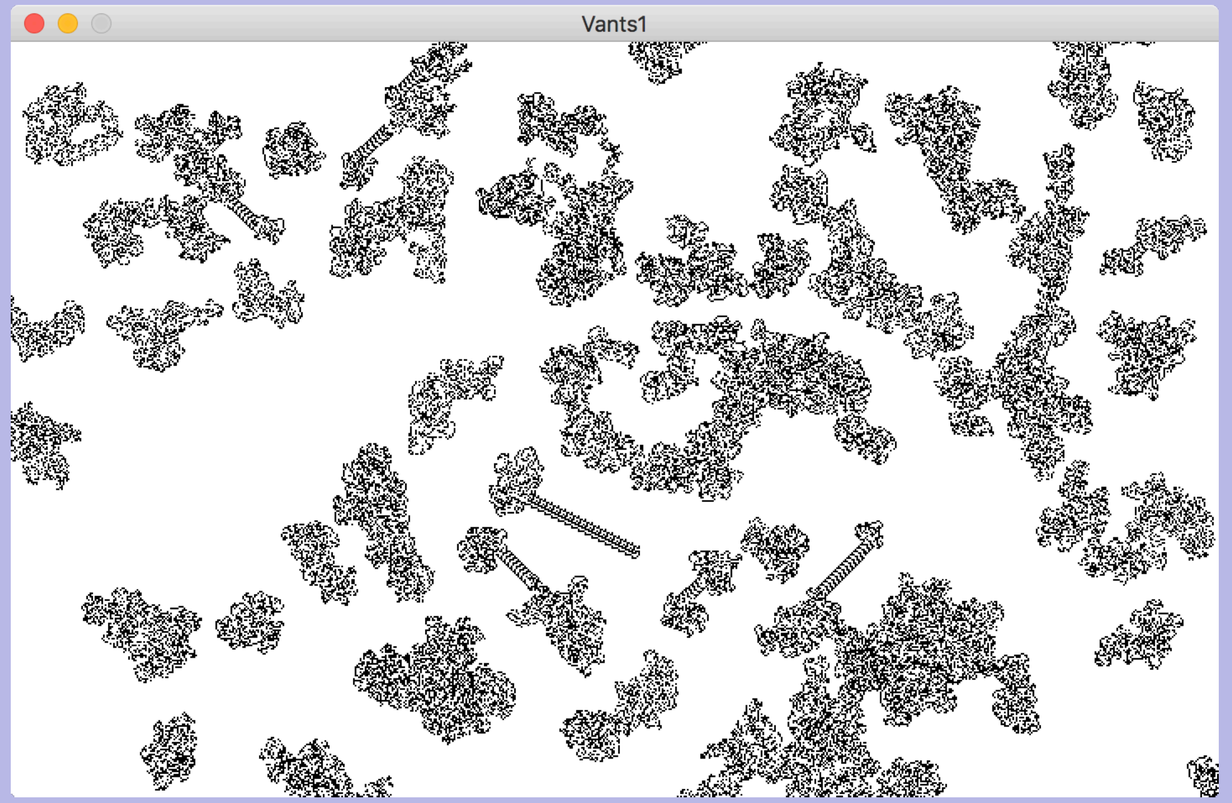




Drawing

```
def draw
  if s[x, y] == white then
    s[x, y] = black
    turn 270
  else
    s[x, y] = white
    turn 90
  end
  move_by a, 1
end
```

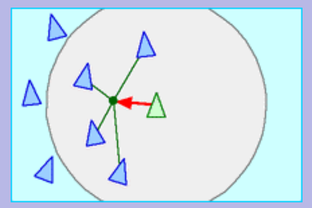
Ant Farm!



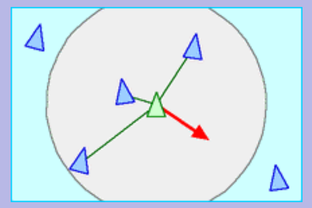
Boids

https://en.wikipedia.org/wiki/Boids

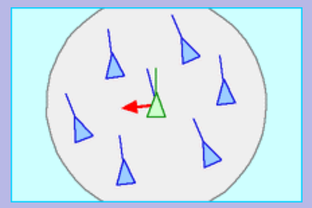
Rules



cohesion: steer to move toward the average position (center of mass) of local flockmates.



separation: steer to avoid crowding local flockmates.



alignment: steer towards the average heading of local flockmates.

The Usual Start

```
class Boids < G::S
  include WhiteBackground
  include ShowFPS

  attr_accessor :boids

  def initialize
    super

    self.boids = populate Boid
    register_bodies boids
  end
end
```

```
class Boid < G::B
  COUNT        = 25
  MAX_DISTANCE = 100
  MAX_VELOCITY = 5

  def initialize w
    super

    self.m = rand * MAX_VELOCITY
    self.a = random_angle
  end

  class View
    def self.draw w, b
      x, y, a, m = b.x, b.y, b.a, b.m

      w.circle x, y, MAX_DISTANCE, :gray90
      w.angle  x, y, a, 3 * m,     :red
      w.circle x, y, 5,            :black, :filled
    end
  end
end
```

Rule 1: Cohesion

```
def rule1
  (center_mass - position) * PCT_DAMPENER
end

def center_mass
  nearby = self.nearby

  return position if nearby.empty?

  nearby.avg(V::ZERO, &:position)
end
```

Rule 2: Separation

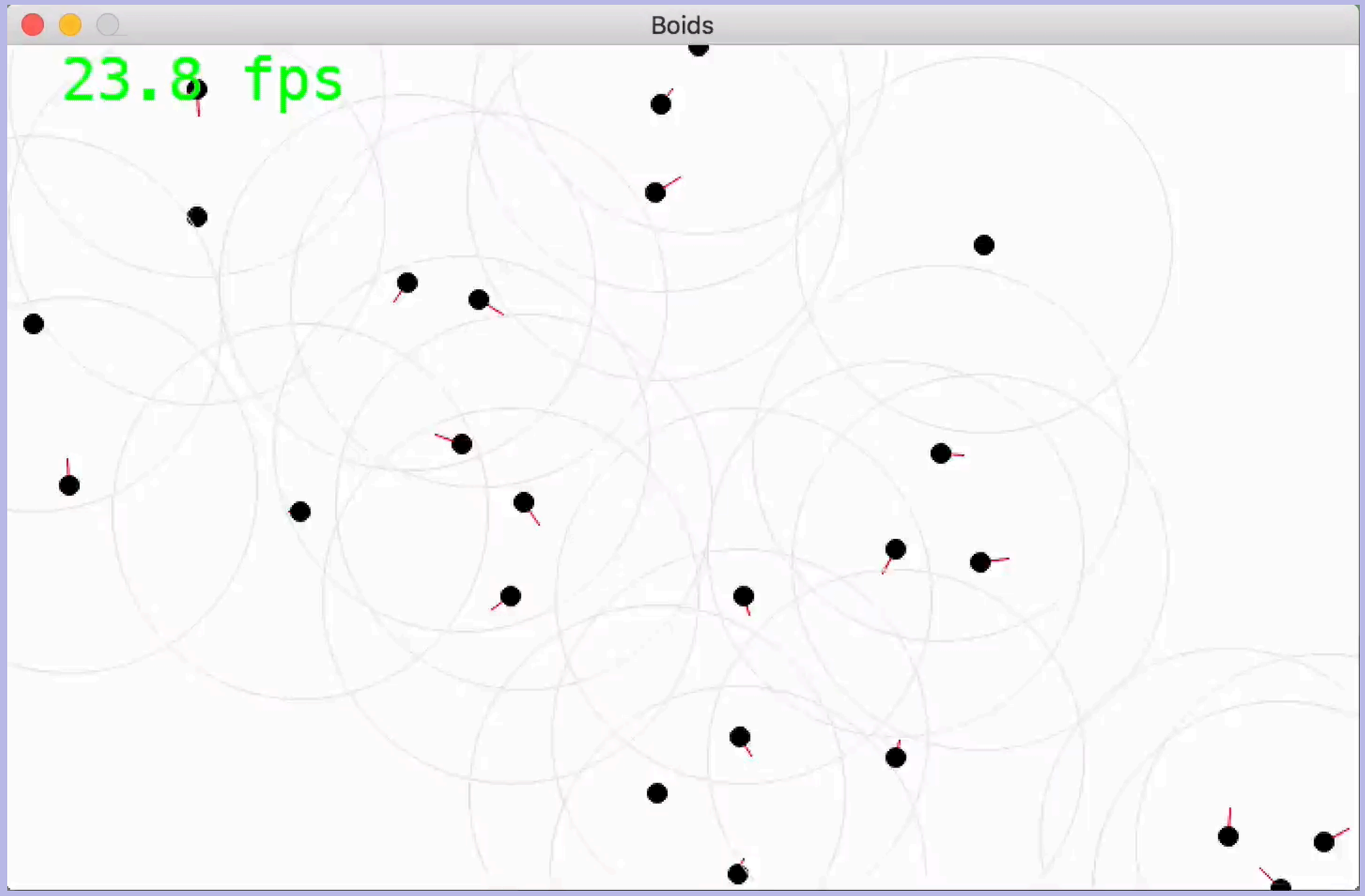
```
def rule2
  too_close = nearby.filter_map { |b|
    diff = b.position - self.position
    next if diff.magnitude > TOO_CLOSE
    diff
  }

  if too_close.empty? then
    V::ZERO
  else
    -too_close.avg(V::ZERO) / 8
  end
end
```

Rule 3: Alignment

```
def rule3
  nearby = self.nearby

  return velocity if nearby.empty?

  v = nearby.avg(V::ZERO, &:velocity)

  (v - velocity) / 4
end
```

Sum and Move

```
def update
  v1 = rule1
  v2 = rule2
  v3 = rule3

  self.velocity += v1 + v2 + v3
  limit_velocity

  move
  wrap
end
```

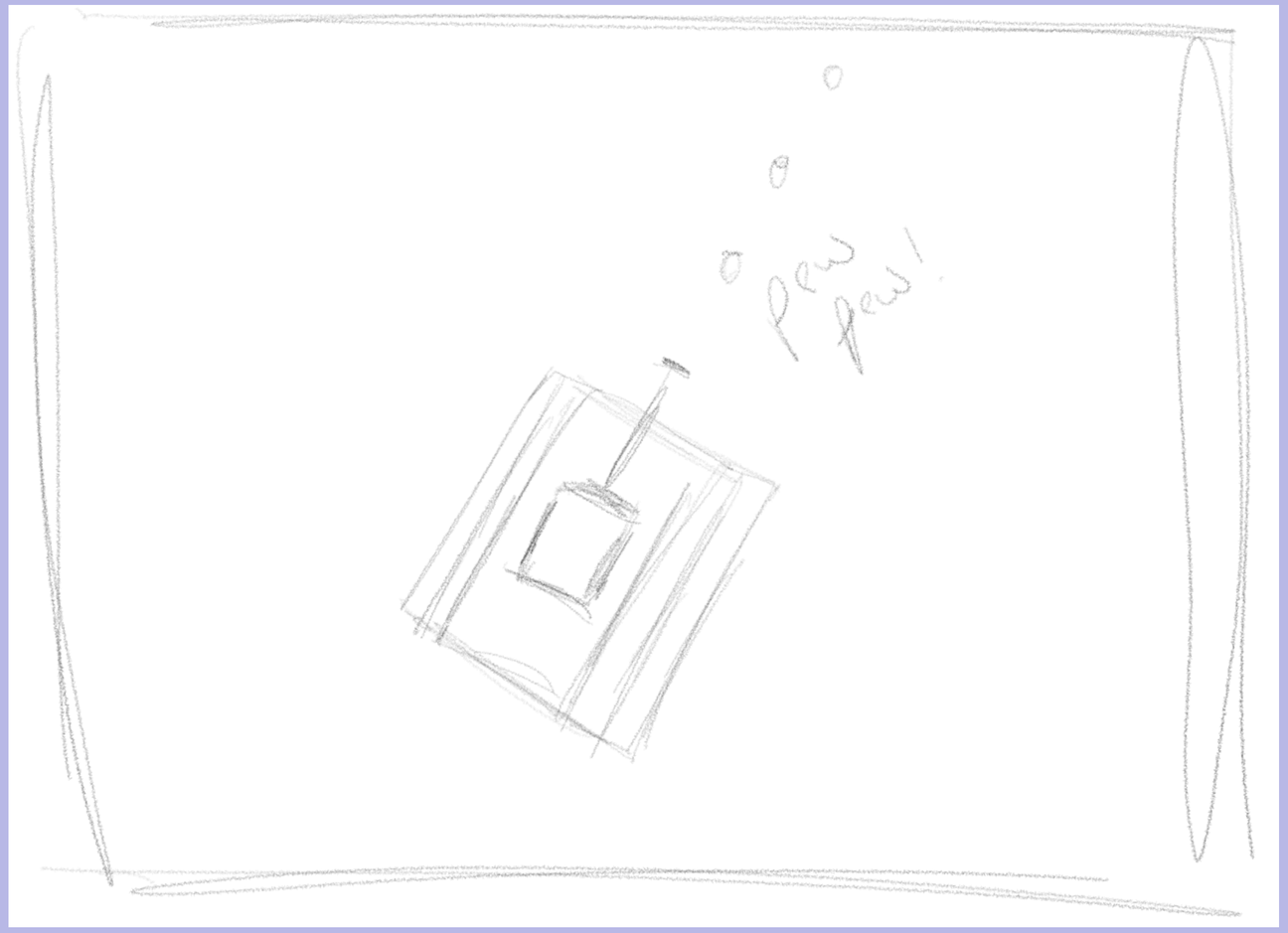

Boids
23.8 fps

Game

Tanks

(pew pew)

Sketch



Rules

tank is in arena

left/right: turn tank

forward/back: accelerate/decelerate

‘a’ / ’s’: turn the turret

Bullets and firing left as an exercise.

Blank canvas

```
require "graphics"

class TankGame < Graphics::Simulation
  def initialize
    super 640, 640

    register_body Tank.new self
  end
end

TankSprites.new.run
```

Bare Tank Canvas

```
class Tank < Graphics::Body
  attr_accessor :turret

  def initialize w
    super

    self.x      = w.w/2
    self.y      = w.h/2
    self.turret = 0
  end
end
```

Sprite

```
# class TankGame
  def initialize
    # …
    a, b = 40, 30
    self.body_img = sprite a, b do
      rect 0,  0, a-1, 29, :black
      rect 0,  4, a-1, 21, :black

      line 0,   2, a-2,   2, :black
      line 0, b-4, a-2, b-4, :black
    end

    a, b = 41, 16
    self.turret_img = sprite a, b do
      rect  a/2-8,  b/2-8,     15,    15, :black
      angle a/2-1,  b/2-1,      0,    28, :black
      line  a/2+20, b/2-3, a/2+20, b/2+1, :black
    end
  end
```

Blit

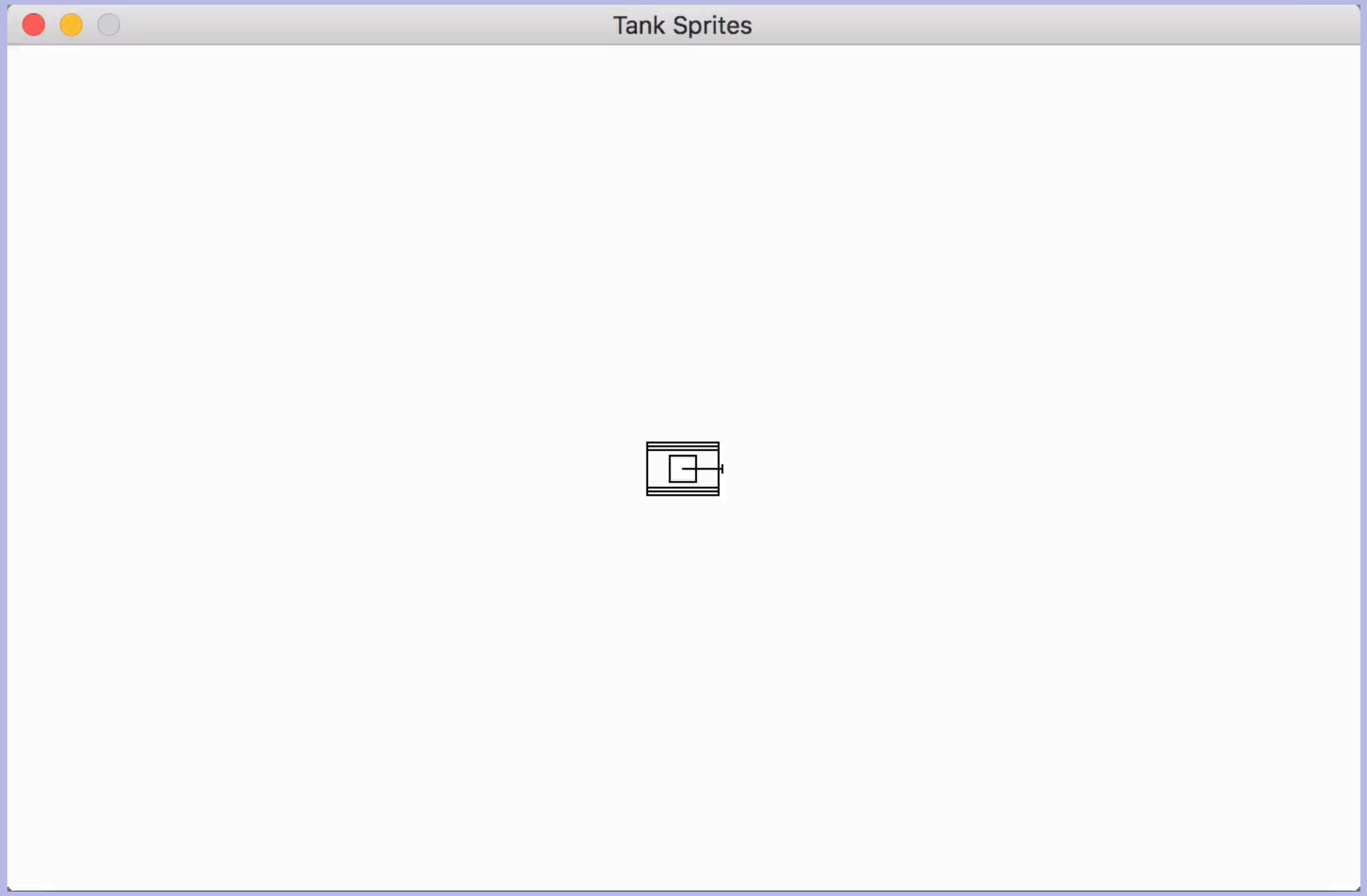
```
# class Tank
  class View
    def self.draw w, b
      x, y, a, t = b.x, b.y, b.a, b.turret

      w.blit w.body_img,   x, y, a
      w.blit w.turret_img, x, y, t
    end
  end
```

Behavior

```
# class Tank
  def update
    self.m = m.clamp 0, MAX_SPEED
    move
    clip
  end

  def turn_right; self.a -= ROTATION; aim_right; end
  def turn_left;  self.a += ROTATION; aim_left;  end
  def aim_right;  self.turret -= ROTATION;       end
  def aim_left;   self.turret += ROTATION;       end
  def accelerate; self.m += ACCELERATE;          end
  def decelerate; self.m -= DECELERATE;          end
```

Event Handling

```
# class TankGame
  def initialize_keys
    super

    add_key_handler(:RIGHT) { tank.turn_right }
    add_key_handler(:LEFT)  { tank.turn_left  }
    add_key_handler(:UP)    { tank.accelerate }
    add_key_handler(:DOWN)  { tank.decelerate }
    add_key_handler(:A)     { tank.aim_left   }
    add_key_handler(:S)     { tank.aim_right  }
  end
```


Tank Sprites

Conventions

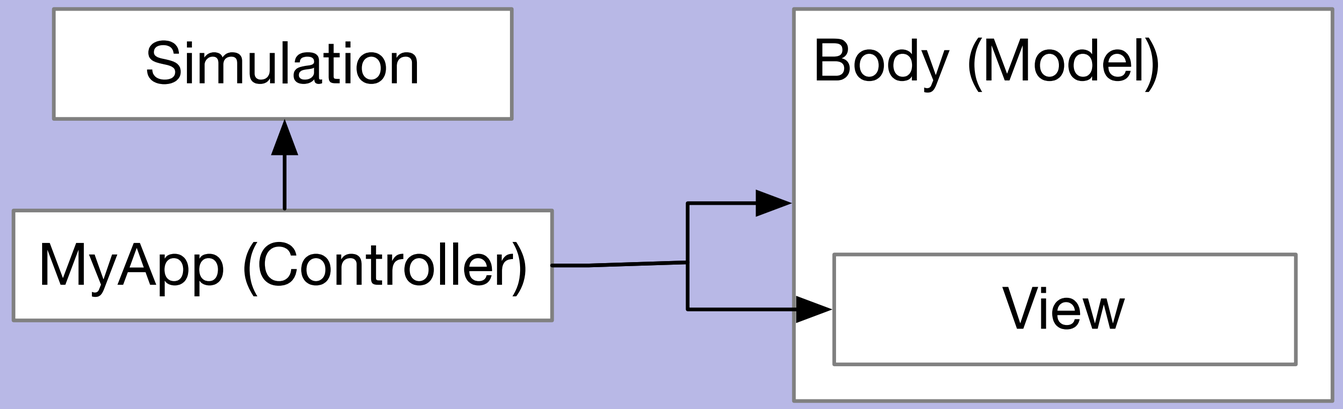
Sketch & Rules

Blank Canvas

Build Up

Drawing ➡ Simulation

MVC



Decorators

DisplayFPS

WhiteBackground

DrawGrid

etc

Thank You!

If you want to sponsor my work on Open Source:

https://www.patreon.com/zenspider

If you want to hire me:

http://www.seattlerb.com/