Ruby: *seu próximo amor*

Oi, meu nome é Philipe

Sou programador no *venturus*

Conhecendo a linguagem

Vantagens

- Elegância (lindeza)
- Comunidade
- Rails

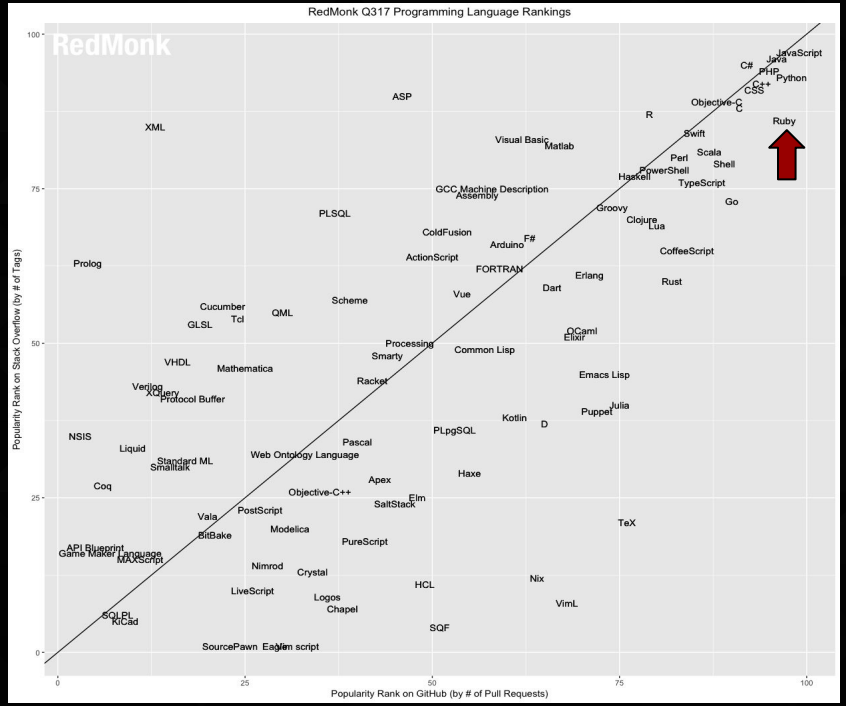
Desvantagens

- Desempenho
- Rails

Conhecendo a linguagem

Comunidade

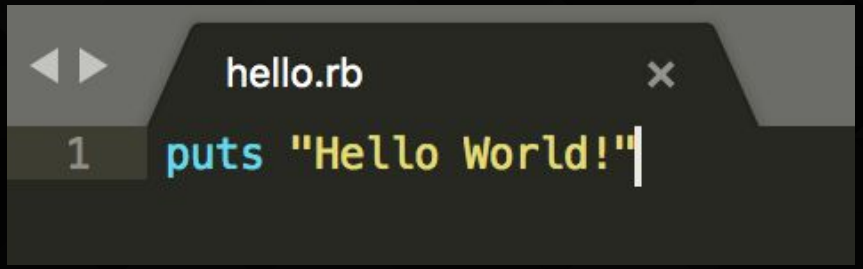
GURU SP



Linguagens comparadas por “Pergunta no Stack overflow” e “Pull requests no github”
Fonte: http://redmonk.com/sogrady/2017/06/08/language-rankings-6-17/

Ruby | Hello World

- Extensão .rb
- método `puts`
- Executando no terminal




```
➜ scripts ruby hello.rb
Hello World!
```

Ruby | Boring parts #1.0 - Variáveis

- Tipagem dinâmica e forte.
- Existem 3 tipos de variáveis:

globais

```
$global = 10
```

Ruby | Boring parts #1.1 - Variáveis

instância

```ruby
@instance_variable = 10
```

locais

```ruby
local_variable = 10
_local_variable = 10
```

Ruby | Boring parts #1.2 - Variáveis

Variáveis Especiais

`self`

```ruby
self.var = 10
```

`nil`

```ruby
new_variable == nil
```

Ruby | Boring parts #2.0 - Condicionais

if

```ruby
i = 10
if i != 10
    puts "i != 10"
elsif i > 10
    puts "i > 10"
else
    puts "i == 10"
end
```

unless

```ruby
i = 10
unless i != 10
    puts "i != 10"
end
```

Ruby | Boring parts #2.1 - Condicionais

case

```ruby
i = 10
case i
    when 1..5
        puts "'i' está entre 1~5"
    when 6..10
        puts "'i' está entre 6~10"
end
```

Ruby | Boring parts #3.0 - Laços

while

```ruby
i = 0
while i < 10  do
    puts("Inside the loop i = #{i}" )
    i +=1
end
```

Ruby | Boring parts #3.1 - Laços

for ... in

```ruby
for number in (1..10)
    puts("Inside the loop number = #{number}" )
end
```

Ruby | Boring parts #3.2 - Laços

until

```ruby
i = 0
until i == 10  do
    puts("Inside the loop i = #{i}" )
    i +=1
end
```

Ruby | Good parts #1.0 - Modificadores

if/unless

```ruby
puts "Hello" if true
puts "World" unless false
```

Ruby | Good parts #1.1 - Modificadores

while/until

```ruby
puts "Something" while true
puts "Other something" until false
```

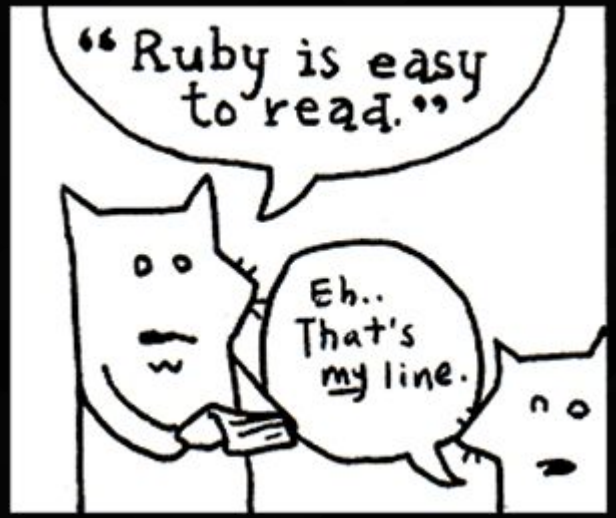
Ruby | Good parts #1.2 - Modificadores

Motivação dos modificadores

(a) Se chover, fique em casa
(b) Fique em casa se chover

```ruby
if chover
    fique_em_casa()
end

fique_em_casa() if chover
```



Ruby | Good parts #2.0 - Iteradores

Sintaxe genérica

```
variable.iterator do
  # Codigo bonito
end
```

Ruby | Good parts #2.1 - Iteradores

times (integer)

```ruby
10.times do
    # código bom
end
```

Ruby | Good parts #2.2 - Iteradores

each

```ruby
var_array = [4, 8, 15, 16, 23, 42]
var_array.each do |element|
    puts element
end
```

Ruby | Good parts #2.3 - Iteradores

select

```ruby
var_array = [4, 8, 15, 16, 23, 42]
bigger_numbers = var_array.select do |element|
    element > 15
end
```

Ruby | Good parts #2.4 - Iteradores

map

```ruby
var_array = [4, 8, 15, 16, 23, 42]
bigger_numbers = var_array.map do |element|
    element+1
end
puts bigger_numbers
```

Ruby | Good parts #2.5 - Iteradores

select

```ruby
var_array = [4, 8, 15, 16, 23, 42]
number_of_even = var_array.count do |element|
    element % 2 == 0
end
puts number_of_even
```

Ruby | Good parts #2.5 - Iteradores

BONUS ROUND

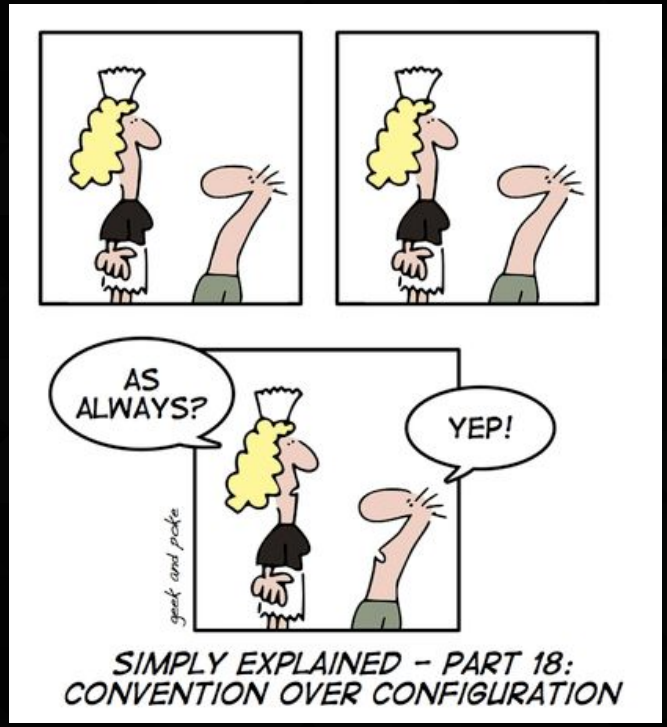
```ruby
var_array = [4, 8, 15, 16, 23, 42]
var_array.each { |element| puts element }
```

Sintaxe genérica

```ruby
variable.iterator { |args| code }
```

Ruby | Good parts #3.0 - C.O.C

Convention
Over
Configuration



Ruby | Good parts #3.1 - C.O.C

Funções que terminam em "?"

```
variable.nil?
variable.empty?
variable.blank?
```

Ruby | Good parts #3.1 - C.O.C

Funções que terminam em "!"

```ruby
var_array = [4, 8, 15, 16, 23, 42]
var_array.map! do |element|
    element+1
end
puts var_array
```

Ruby | Orientação a Objetos

Ruby | Orientação a Objetos #1.0

Sintaxe de uma classe

```ruby
class ClassName

end
```

- Nome em CamelCase

Ruby | Orientação a Objetos #1.1

Inicializando uma instância

```ruby
class ClassName
  def initialize (parms)
    # Esse código é executado
    # quando iniciamos uma instancia
  end
end

class_name = ClassName.new (params)
```

Ruby | Orientação a Objetos #1.2

Chamando um método

```ruby
class ClassName
  def method (params)
    ##
  end
end

class_name = ClassName.new
class_name.method (params)
```

Ruby | Orientação a Objetos #1.3

Atributos dos nossos objetos

```ruby
class ClassName
  attr_reader :attr1
  attr_writer :attr2
  attr_accessor :attr3
end
```

Ruby | Orientação a Objetos #1.4

attr_reader

```ruby
attr_reader :attr1

def attr
  @attr
end
```

Ruby | Orientação a Objetos #1.5

attr_writer

```ruby
attr_writer :attr2

def attr2=(value)
  @attr2 = value
end
```

Ruby | Orientação a Objetos #1.6

attr_acessor

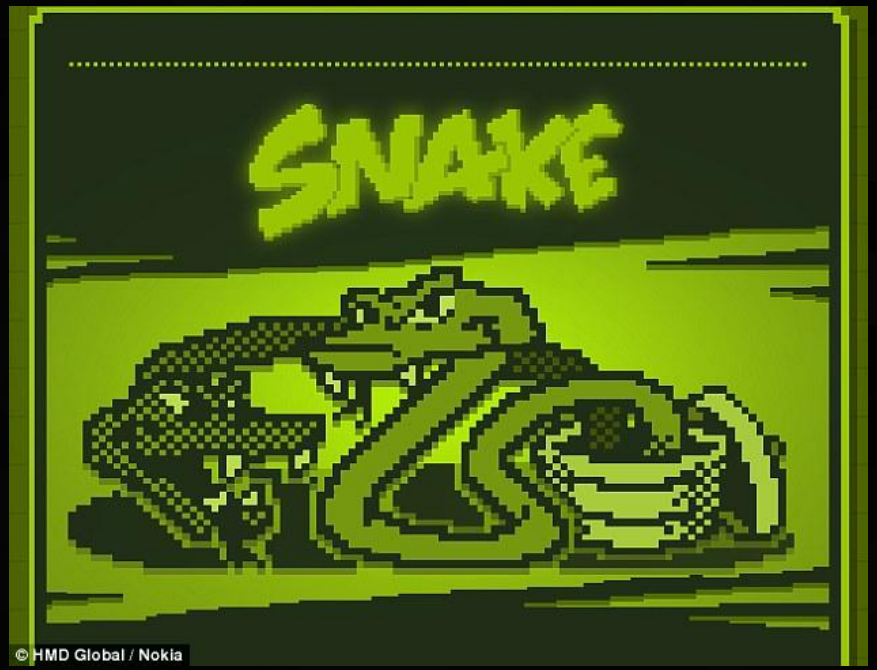
```ruby
attr_accessor :attr3

attr_writer :attr3
attr_reader :attr3
```

Really Good Parts

Vamos programar

Really Good Parts



Ruby | Snake v0.1 - Engine

Engine.rb

```ruby
class Engine
  def run
    while true
      puts "Running"
    end
  end
end

engine = Engine.new
engine.run
```

```
Running
Running
Running
Running
Running
Running
Running
```

Ruby | Snake v0.2 - Input

`input.rb`

- `STDIN` é "igual" o `stdio.h`
- operador `<<`
- copie do endereço:

http://josenberg.com.br/input.rb

```ruby
require 'io/console'

def read_movement
  STDIN.echo = false
  STDIN.raw!

  input = STDIN.getc.chr
  if input == "\e" then
    input << STDIN.read_nonblock(3) rescue nil
    input << STDIN.read_nonblock(2) rescue nil
  end

  STDIN.echo = true
  STDIN.cooked!

  case input
  when "\e[A"
    return "up"
  when "\e[B"
    return "down"
  when "\e[C"
    return "right"
  when "\e[D"
    return "left"
  else
    return "quit"
  end
end
```

Ruby | Snake v0.3 - Screen

screen.rb

```ruby
class Screen
  def draw
    10.times do
        10.times do
            print "[ ]"
        end
        print "\n"
    end
    print "_____"
  end
end
```

engine.rb

```ruby
def run
  screen = Screen.new
  while true
    # Le a entrada do usuario
    movement = read_movement
    # para de executar o programa
    # caso algo alem das setinhas seja recebido
    break if movement == "quit"
    # desenha um novo frame
    screen.draw
  end
end
```

Ruby | Snake v0.4 - Snake

snake.rb

```ruby
class Snake
    attr_accessor :position
    def initialize
        self.position = [{:x => 5, :y => 5}]
    end
end
```

Ruby | Snake v0.4 - Snake

Acessando a posição em outra classe

engine.rb

```ruby
def run
  screen = Screen.new
  snake = Snake.new
  while true
    puts "Snake position: x:#{snake.position[0][:x]} y:#{snake.position[0][:y]}"
    # Le a entrada do usuario
    movement = read_movement
    # para de executar o programa caso algo alem das setinhas seja recebido
    break if movement == "quit"
    # desenha um novo frame
    screen.draw snake.position
  end
end
```

Ruby | Snake v0.4 - Snake

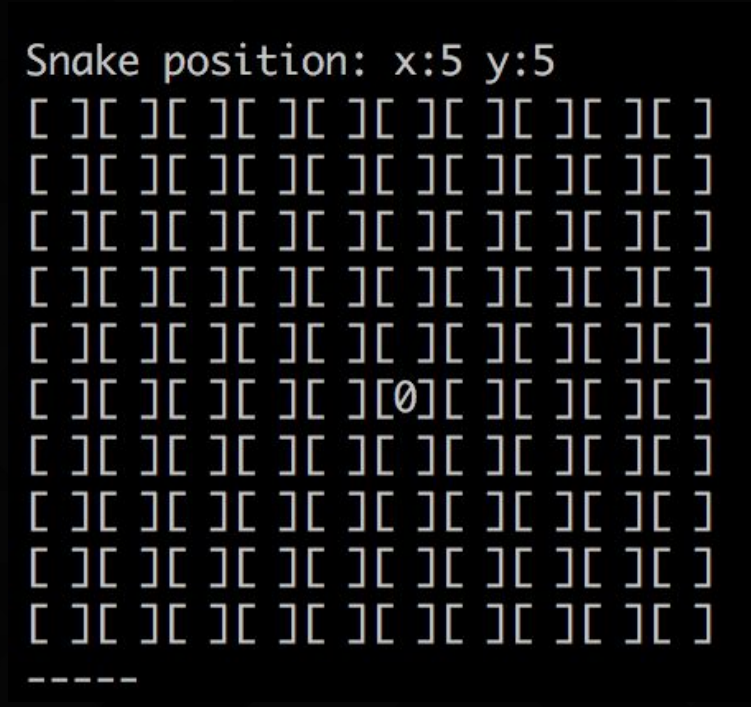
screen.rb

```ruby
def is_snake? snake_positions, x, y
  snake_positions.any? do |position|
      position[:x] == x && position[:y] == y
  end
end
def draw (snake_positions)
  10.times do |i|
      10.times do  |j|
          if self.is_snake? snake_positions, i, j
              print "[0]"
          else
              print "[ ]"
          end
      end
      print "\n"
  end
  puts "_____"
end
```

Precisamos colocar um símbolo diferente na posição em que a snake estiver

Ruby | Snake v0.4 - Snake

Resultado:



Ruby | Snake v0.5 - Movimentos

snake.rb

```ruby
def move direction
    new_position = {:x => self.position[0][:x], :y => self.position[0][:y]}
    case direction
    when "up"
        new_position[:y] -= 1
    when "down"
        new_position[:y] += 1
    when "right"
        new_position[:x] += 1
    when "left"
        new_position[:x] -= 1
    end

    self.position.unshift (new_position)
    self.position.pop
end
```

Ruby | Snake v0.5 - Movimentos

engine.rb

```ruby
puts "Snake position: x:#{snake.position[0][:x]} y:#{snake.position[0][:y]}"
# Le a entrada do usuario
movement = read_movement
snake.move movement
# para de executar o programa caso algo alem das setinhas seja recebido
break if movement == "quit"
# desenha um novo frame
screen.draw snake.position
```

Ruby | Snake v0.6 - Maçã

Ruby | Snake v0.6 - Maçã

apple.rb

```ruby
class Apple
  attr_accessor :position
  def generate_apple
    self.position = {:x => rand(10), :y => rand(10)}
  end
  def initialize
    self.generate_apple
  end
end
```

Ruby | Snake v0.6 - Maçã

screen.rb

```ruby
def is_apple? apple_position, x, y
  apple_position[:x] == x && apple_position[:y] == y
end
def draw (snake_positions, apple_position)
  10.times do |i|
      10.times do  |j|
          if (self.is_snake? snake_positions, j, i)
              print "[0]"
          elsif (self.is_apple? apple_position, j, i)
              print "[X]"
          else
              print "[ ]"
          end
      end
      print "\n"
  end
  puts "_____"
end
```

Ruby | Snake v0.6 - Maçã

engine.rb

```ruby
snake = Snake.new
apple = Apple.new
while true
  puts "Snake position: x:#{snake.position[0][:x]} y:#{snake.position[0][:y]}"
  puts "Apple position: x:#{apple.position[:x]} y:#{apple.position[:y]}"
  # Le a entrada do usuario
  movement = read_movement
  snake.move movement
  # para de executar o programa caso algo alem das setinhas seja recebido
  break if movement == "quit"
  # desenha um novo frame
  screen.draw snake.position, apple.position
end
```

Ruby | Snake v0.7 - Eventos

Eventos do nosso jogo

Bons

- Cobra na mesma posição que uma maçã

Ruins

- Cobra na mesma posição dela mesma
- Cobra em uma posição fora da dela

Ruby | Snake v0.7a - Eventos

Vamos tratar de comer a maçã

Ruby | Snake v0.7a - Eventos

event.rb

```ruby
class Event
    def snakeAteApple? snake_positions, apple_position
        snake_positions.any? do |position|
            position[:x] == apple_position[:x] && position[:y] == apple_position[:y]
        end
    end
end
```

Ruby | Snake v0.7a - Eventos

engine.rb

```
if event.snakeAteApple? snake.position, apple.position
  score += 1
  apple.generate_apple
end
```

Ruby | Snake v0.7b - Eventos

Colisões com *the wall*

Ruby | Snake v0.7b - Eventos

event.rb

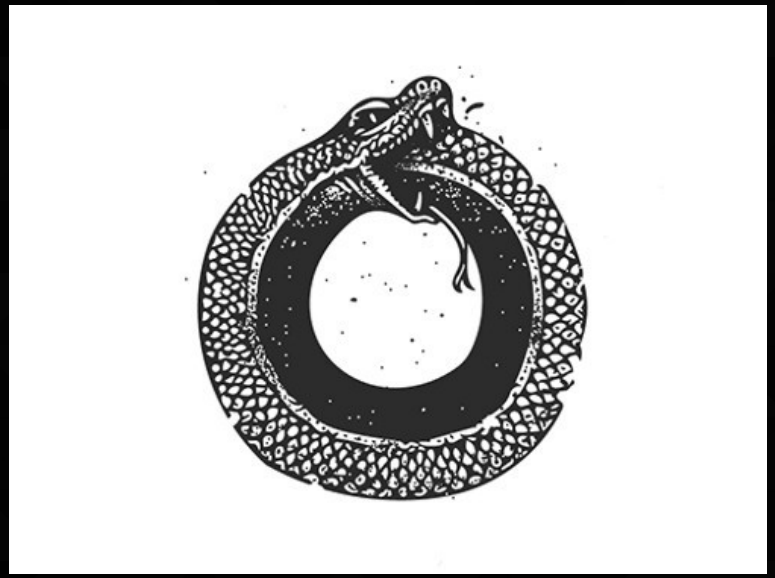
```ruby
def snakeHitWall? snake_positions
    snake_positions.any? do |position|
        (position[:x] > 9 || position[:x] < 0) || (position[:y] > 9 || position[:y] < 0)
    end
end
```

Ruby | Snake v0.7b - Eventos

engine.rb

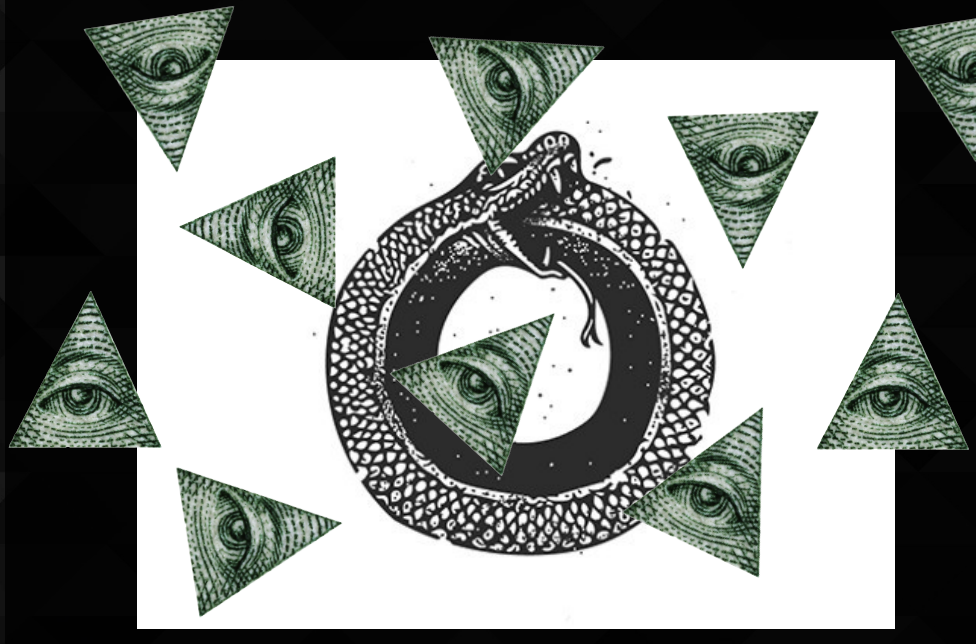
```
if event.snakeHitWall? snake.position
  puts "GAME OVER"
  puts "Final Score: #{score}"
  break
end
```

Ruby | Snake v0.7c - Eventos



Quando a cobra se colide com ela mesma

Ruby | Snake v0.7c - Eventos



Quando a cobra se colide com ela mesma

Ruby | Snake v0.7c - Eventos

Mas para a cobra colidir com si mesma é preciso que ela cresça, então vamos precisar voltar e implementar isso.

Ruby | Snake v0.7c - Eventos

snake.rb

```ruby
attr_accessor :size_to_increase

def grow
    self.size_to_increase += 1
end
```

Função responsável pelo crescimento

Ruby | Snake v0.7c - Eventos

snake.rb

```
    if self.size_to_increase == 0
        self.position.pop
    else
        self.size_to_increase -= 1
    end
end
```

Como fazer ela não diminuir mais;

Ruby | Snake v0.7c - Eventos

```
current score: 7
Snake position: x:4 y:8
Apple position: x:9 y:4
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][0][0][ ][ ][ ][X]
[ ][ ][ ][ ][0][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][0][0][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][0][ ][ ][ ][ ]
[ ][ ][ ][ ][0][0][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
-----
```

Agora que é possível ela se colidir em si mesma nós implementamos o evento

Ruby | Snake v0.7c - Eventos

event.rb

```ruby
def snakeAteItsefl? snake_positions
    snake_positions.uniq{ |p| p.values_at(:x, :y) }.size < snake_positions.size
end
```

Como você faria esse algoritmo em C?

Ruby | Snake v0.8 - Melhorias

Podemos adicionar cor:

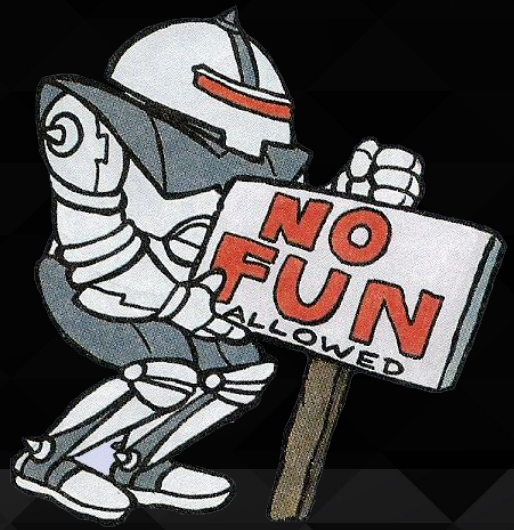
```
if (self.is_snake? snake_positions, j, i)
    print "[\e[32m0\e[0m]"
elsif (self.is_apple? apple_position, j, i)
    print "[\e[31mX\e[0m]"
else
    print "[ ]"
end
```

```
-----
current score: 5
Snake position: x:7 y:4
Apple position: x:2 y:0
[ ][ ][X][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][0][0][0][0][0][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][0][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
```

Ruby | Snake v1.0-release

```
-----
current score: 5
Snake position: x:7 y:4
Apple position: x:2 y:0
[ ][ ][X][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][0][0][0][0][0][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][0][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
```

Foi maneiro fazer o joguinho, mas falando sério, o que dá pra fazer com ruby?



Ruby on Rails

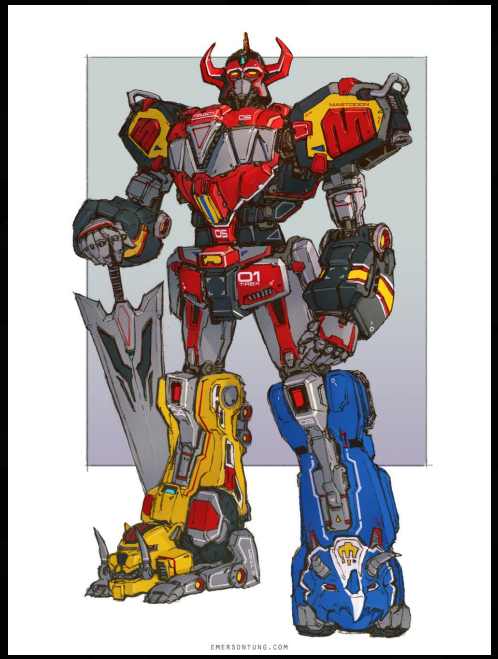
- Framework web
- Facilidade desde o desenvolvimento até o deploy
- muito mais amor <3

Ruby on Rails

==

Ruby on Rails

==



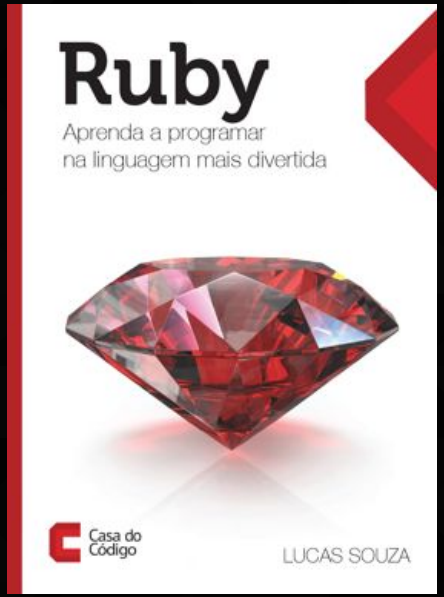
Agradecimentos

venturus

inovação & tecnologia

Referências

- http://geek-and-poke.com
- http://poignant.guide
- https://www.casadocodigo.com.br/products/livro-ruby
- https://code.tutsplus.com/tutorials/ruby-for-newbies-iterators-and-blocks--net-17089
- www.zenruby.info
- ruby-doc.org



SEE YOU SPACE COWBOY...