Imagens, animações e programas interativos

Programação Funcional

Marco A L Barbosa
malbarbo.pro.br

Departamento de Informática
Universidade Estadual de Maringá

Introdução

O sgleam permite a criação de imagens.

O sgleam web permite a criação e exibição de imagens, animações e programas interativos.

Uma **imagem** é um dado visual e retangular. Pode ser uma foto, um desenho, uma figura geométrica ou um texto.

Uma **animação** é a exibição contínua de imagens estáticas que variam de forma sequencial criando a ilusão de movimento.

Um **programa interativo** é um programa que reage a ações do usuário modificando seu comportamento e saída de acordo com as entradas recebidas.

Aviso

Esta parte do sgleam é experimental, então pode conter erros. Se você identificar alguma falha, comunique ao professor.

Imagens

O sgleam fornece várias funções para criação de imagens geométricas básicas, como retângulos, elipses, polígonos e textos. Também fornece funções para transformar e combinar imagens, como rotacionar, espelhar, colocar uma ao lado da outra, entre outras.

As funções para criar, combinar e transformar imagens estão no módulo `sgleam/image`.

Uma forma geométrica tem um estilo, que incluí um contorno (*stroke*) e um preenchimento (*fill*). O contorno tem uma cor, uma espessura e outros atributos. O preenchimento pode ou não ter uma cor.

As constantes e funções para especificar o contorno estão no módulo `sgleam/stroke` e para o preenchimento em `sgleam/fill`. Funções mais gerais para o estilo estão no módulo `sgleam/style` e para cores no módulo `sgleam/color`.

Quadrado e retângulo

```
> image.square(60, stroke.red) // lado
```

```
> image.rectangle(20, 60, fill.blue) // largura e altura
```

```
> image.rectangle(100, 30, style.join([stroke.black, fill.gray]))
```

Dimensões

```
> let r = image.rectangle(20, 60, fill.blue)
> image.width(r)
20
> image.widthf(r)
20.0
> image.height(r)
60
> image.heightf(r)
60.0
```

Circulo e elipse

```
> image.circle(30, stroke.red) // raio
```

```
> image.ellipse(30, 50, stroke.purple) // largura e altura
```

```
> image.circlef(20.0, fill.slateblue) // funções que terminam com f
                                      // os argumentos são Float
```

```
> image.line(20, 50, stroke.black) // (0, 0) para (10, 30)
                                   // y aumenta para baixo
```

```
> image.line(20, -50, style.join([stroke.olive, stroke.width(6)]))
```

```
> image.triangle(40, stroke.tan) // lado
```

```
> image.right_triangle(30, 60, fill.yellowgreen) // base e altura
```

```
> image.isosceles_triangle(60, 150, fill.seagreen) // lados iguais e
                                                   // ângulo entre eles
```

```
> image.rhombus(50, 30, stroke.magenta) // lado e ângulo (superior
                                        //                 e inferior)
```

```
> image.rhombus(60, 120, fill.cornflowerblue) // lado e ângulo (superior
                                              //                 e inferior)
```

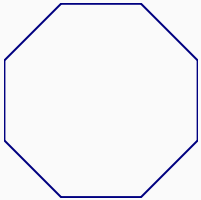
Polígonos

```
> image.polygon(
    [Point(0, 0), Point(-10, 20), Point(60, 0), Point(-10, -20)],
    fill.burlywood,
  )
```

Polígonos

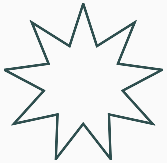
```
> image.regular_polygon(60, 5, fill.chocolate) // lado e número de lados
```

```
> image.regular_polygon(40, 8, stroke.navy) // lado e número de lados
```



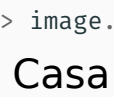
```
> image.star(40, fill.firebrick) // estrela circunscrita em um polígono
                                 // de lado 40
```

```
> image.radial_star(9, 20, 40, stroke.darkslategray) // número de pontas,
                                                      // raio interno e
                                                      // raio externo
```



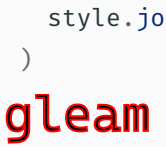
```
> image.star_polygon(40, 7, 3, stroke.seagreen) // estrela circunscrita em
                                                // um polígono de 7 lados
                                                // vertíces ligados de
                                                // 3 em 3
```

```
> image.text("Casa", 20, fill.black) // texto e tamanho da fonte
```



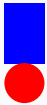

```
> image.text("aTa", 30, stroke.darkgreen) // texto e tamanho da fonte
```


```
> image.text_font(
    "gleam",
    font.Font(..font.default(), size: 25.0, family: "monospace"),
    style.join([stroke.red, fill.black]),
  )
```



```
> image.beside( // ao lado
    image.rectangle(20, 30, fill.blue),
    image.circle(10, fill.red),
)
```

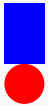
```
> image.above( // acima
    image.rectangle(20, 30, fill.blue),
    image.circle(10, fill.red),
)
```



De forma equivalente, usando *pipeline*

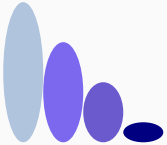
```
> image.rectangle(20, 30, fill.blue)
  |> image.beside(image.circle(10, fill.red))
```

```
> image.rectangle(20, 30, fill.blue)
  |> image.above(image.circle(10, fill.red))
```



Ao lado com posicionamento no eixo y.

```
image.ellipse(20, 70, fill.lightsteelblue)
|> image.beside_align(yplace.Bottom, image.ellipse(20, 50, fill.mediumslateblue))
|> image.beside_align(yplace.Bottom, image.ellipse(20, 30, fill.slateblue))
|> image.beside_align(yplace.Bottom, image.ellipse(20, 10, fill.navy))
```



`yplace` é um módulo que define o tipo `YPlace` com as variantes `Top`, `Bottom` e `Middle`.

Acima com posicionamento no eixo x.

```
> image.ellipse(70, 20, fill.yellowgreen)
  |> image.above_align(xplace.Right, image.ellipse(50, 20, fill.olivedrab))
  |> image.above_align(xplace.Right, image.ellipse(30, 20, fill.darkolivegreen))
  |> image.above_align(xplace.Right, image.ellipse(10, 20, fill.darkgreen))
```

`xplace` é um módulo que define o tipo `XPlace` com as variantes `Right`, `Left` e `Center`.

Sobreposição.

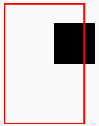
```
> image.rectangle(30, 60, fill.orange)
  |> image.overlay(image.ellipse(60, 30, fill.purple))
```

Sobreposição com posicionamento no eixo x e y.

```
> image.rectangle(30, 60, style.join([fill.orange, fill.opacityf(0.75)]))
  |> image.overlay_align(
    xplace.Left,
    yplace.Middle,
    image.ellipse(60, 30, fill.purple),
  )
```

Os cantos superiores esquerdos começam no mesmo ponto, a segunda imagem é transladada.

```
> image.rectangle(40, 60, stroke.red)
  |> image.overlay_xy(25, 10, image.rectangle(20, 20, fill.black))
```



Coloca o centro da segunda imagem na posição especificada.

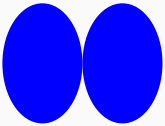
```
> image.square(48, fill.lightgray)
  |> image.place_image(24, 24, image.triangle(32, fill.red))
```

Coloca o ponto dado por `xplace` e `yplace` da segunda imagem na posição especificada.

```
> image. rectangle(64, 64, fill.palegoldenrod)
  |> image.place_image_align(
    64,
    64,
    xplace.Right,
    yplace.Bottom,
    image.triangle(48, fill.yellowgreen),
  )
```

Escala.

```
> image.ellipse(20, 30, fill.blue)
  |> image.scale(2)
  |> image.beside(image.ellipse(40, 60, fill.blue))
```



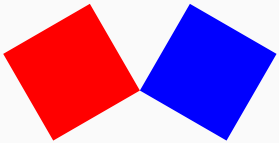
```
> image.text("teste", 16, fill.black) |> image.scale_xyf(2.5, 1.2)
```

Rotaciona no sentido anti-horário.

```
> image.rectangle(20, 60, fill.blue) |> image.rotate(30)
```

Espelha na horizontal.

```
> image.beside(
    image.square(50, fill.red) |> image.rotate(30),
    image.square(50, fill.blue) |> image.rotate(30) |> image.flip_horizontal,
)
```



Espelha na vertical.

```
> image.above(
    image.star(40, fill.firebrick),
    image.star(40, fill.gray) |> image.flip_vertical |> image.scale_xyf(1.0, 0.5),
)
```

Coloca um quadro ao redor da imagem, útil para depuração.

```
> image.frame(image.star(30, stroke.firebrick))
```

Cria um quadro que pode ser usado como fundo para colocar outras imagens.

```
> image.empty_scene(70, 50)
```

Construindo uma casa

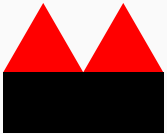
Uma casa simples.

```
> image.triangle(40, fill.red)
  |> image.above(image.rectangle(40, 30, fill.black))
```

Construindo uma casa

Uma casa com dois telhados.

```
> image.triangle(40, fill.red)
  |> image.beside(image.triangle(40, fill.red))
  |> image.above(image.rectangle(80, 30, fill.black))
```

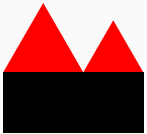


Se os telhados não forem do mesmo tamanho, eles não ficam alinhados.

```
> image.triangle(40, fill.red)
  |> image.beside(image.triangle(30, fill.red))
  |> image.above(image.rectangle(70, 30, fill.black))
```

Se os telhados não forem do mesmo tamanho, eles não ficam alinhados.

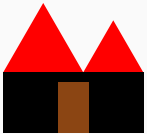
```
> let casa = image.triangle(40, fill.red)
  |> image.beside_align(yplace.Bottom, image.triangle(30, fill.red))
  |> image.above(image.rectangle(70, 30, fill.black))
```



Colocando uma porta.

```
> let porta = image.rectangle(15, 25, fill.saddlebrown)
```

```
> image.overlay_align(porta, xplace.Center, yplace.Bottom, casa)
```

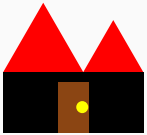


Construindo uma casa

Colocando uma porta, agora com maçaneta.

```
> let porta_macaneta = image.overlay_align(
    image.circle(3, fill.yellow),
    xplace.Right,
    yplace.Middle,
    porta,
  )

> image.overlay_align(porta_macaneta, xplace.Center, yplace.Bottom, casa)
```



Salvando imagens

No sgleam (linha de comando) é possível salvar as imagens no formato svg.

Por exemplo, dado o arquivo `estrela.gleam` abaixo

```
import sgleam/image
import sgleam/fill

pub fn smain() {
  image.star(40, fill.firebrick)
  |> image.to_svg()
}
```

Execute

```
sgleam estrela.gleam > estrela.svg
```

Cores

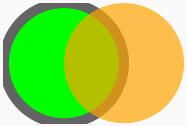
white	darkgray	darkred	gold	lawngreen	cyan	darkblue
snow	darkgrey	firebrick	yellow	lime	lightcyan	navy
ghostwhite	gray	indianred	lightyellow	limegreen	paleturquoise	midnightblue
whitesmoke	grey	brown	lemonchiffon	seagreen	aquamarine	purple
floralwhite	dimgray	sienna	lightgoldenrodyellow	forestgreen	turquoise	indigo
oldlace	dimgrey	rosybrown	papayawhip	darkgreen	mediumturquoise	blueviolet
ivory	slategray	salmon	seashell	green	darkturquoise	darkviolet
honeydew	slategrey	lightsalmon	moccasin	mediumseagreen	darkcyan	darkslateblue
mintcream	lightslategray	darksalmon	peachpuff	mediumspringgreen	mediumaquamarine	mediumslateblue
azure	lightslategrey	coral	bisque	springgreen	steelblue	plum
beige	darkslategray	lightcoral	blanchedalmond	palegreen	lightsteelblue	slateblue
aliceblue	darkslategrey	tomato	cornsilk	lightgreen	powderblue	mediumpurple
antiquewhite	black	orangered	navajowhite	darkolivegreen	lightblue	thistle
linen	red	saddlebrown	wheat	olivedrab	skyblue	violet
lavender	crimson	chocolate	burlywood	yellowgreen	lightskyblue	orchid
lavenderblush	deeppink	sandybrown	tan	olive	deepskyblue	mediumorchid
mistyrose	hotpink	peru	darkkhaki	darkseagreen	dodgerblue	darkorchid
gainsboro	pink	orange	khaki	lightseagreen	cornflowerblue	magenta
lightgray	lightpink	darkorange	palegoldenrod	cadetblue	royalblue	fuchsia
lightgrey	palevioletred	goldenrod	greenyellow	teal	blue	darkmagenta
silver	maroon	darkgoldenrod	chartreuse	aqua	mediumblue	mediumvioletred

Cores

Cores também podem ser criadas usando valores rgb ou rbga.

```
> let verde = color.rgb(0, 255, 0) // valores entre 0 e 255
> let laranja = color.rgba(255, 165, 0, 0.7) // opacidade entre 0 a 1
> let estilo = style.join([
    fill.with(verde), // com uma cor existente
    stroke.rgb(100, 100, 100), // diretamente com rgb
    stroke.width(5),
  ])

> image.circle(30, estilo)
  |> image.underlay_xy(30, 0, image.circle(30, fill.with(laranja)))
```



Animações

Uma **animação** é a exibição contínua de imagens estáticas que variam de forma sequencial criando a ilusão de movimento.

A animação mais simples no sgleam é uma simulação baseada no tempo. O trabalho do programador é providenciar uma função que cria uma imagem a partir de um número natural.

A função de animação está no pacote `sgleam/world`.

```
pub fn animate(create_image: fn(Int) -> image.Image) -> Nil
```

O relógio da animação faz 28 tiques por segundo, a cada tique a função `create_image` é chamada com o número de tiques passados deste o início da animação.

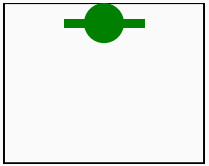
Exemplo óvni

Vamos criar uma animação de um óvni descendo a tela (veja o arquivo `animacao_ovni.gleam`).

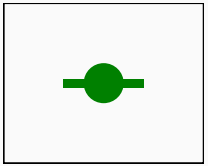
Exemplo óvni

```
fn cena_ovni(tick: Int) -> image.Image {
  let ovni =
    image.circle(10, fill.green)
    |> image.overlay(image.rectangle(40, 4, fill.green))
  image.empty_scene(100, 80) |> image.place_image(50, tick, ovni)
}
```

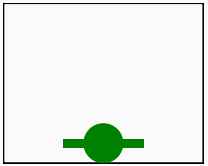
```
> cena_ovni(10)
```



```
> cena_ovni(40)
```



```
> cena_ovni(70)
```



```
pub fn main() {
  world.animate(cena_ovni)
}
```

Exemplo texto com uma letra colorida

Vamos criar uma animação que colore uma letra de um texto por vez (veja o arquivo `animacao_texto.gleam`).

Exemplo texto com uma letra colorida

```
fn cena_texto(tick: Int) -> image.Image {
  let texto = "computação"
  // dividimos por 10 para desacelerar a animação
  let i = tick / 10 % string.length(texto)
  let inicio = string.slice(texto, 0, i)
  let letra = string.slice(texto, i, 1)
  let fim = string.slice(texto, i + 1, string.length(texto))
  let texto =
    image.text(inicio, 30, fill.black)
    |> image.beside(image.text(letra, 30, fill.red))
    |> image.beside(image.text(fim, 30, fill.black))
  image.empty_scene(230, 50) |> image.overlay(texto)
}
```

```
> cena_texto(10)
```


```
> cena_texto(40)
```


```
> cena_texto(76)
```

Programas interativos

Um **programa interativo** é um programa que reage a ações do usuário modificando seu comportamento e saída de acordo com as entradas recebidas.

Para projetar um programa interativo é necessário definir três coisas:

- Definição do estado do programa e constantes;
- Uma função que gera uma imagem a partir do estado do programa;
- Uma função que modifica o estado do programa a partir de um evento de teclado.

Opcionalmente,

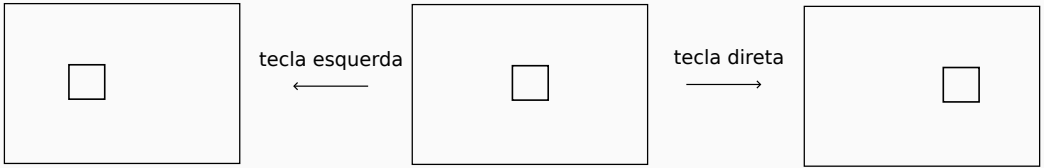
- Uma função que avança o estado do programa com o passar do tempo.

Exemplo movimento horizontal

Vamos criar um programa que exibe um quadrado que pode ser deslocado para a esquerda ou direita usando as teclas apropriadas e que alterna a cor entre verde, amarelo e vermelho a cada segundo (veja o arquivo `move_horizontal.gleam`).

Começamos com um esboço do que queremos.

Exemplo movimento horizontal



O que precisamos para representar o estado do programa? O que muda com as interações ou com o tempo? A coordenada x do centro do quadrado. A cor. Vamos definir uma estrutura.

O que não muda? O tamanho da cena e do quadrado. A coordenada y do centro do quadrado. A coordenada x inicial do centro do quadrado. O deslocamento em cada interação.

Exemplo movimento horizontal

Estado do programa.

```
type Quadrado {
  Quadrado(x: Int, cor: Cor)
}

type Cor {
  Verde
  Amarelo
  Vermelho
}
```

Constantes.

```
const largura = 100
const altura = 100
const deslocamento = 10
const q_lado = 20
const q_y = 50
const q_x_inicial = 50
```

Exemplo movimento horizontal

Função que gera uma imagem a partir do estado.

```
pub fn desenha(q: Quadrado) -> image.Image {
  let fill = case q.cor {
    Verde -> fill.green
    Vermelho -> fill.red
    Amarelo -> fill.yellow
  }
  image.empty_scene(largura, altura)
  |> image.put_image(q.x, q_y, image.square(q_lado, fill))
}
```

Exemplo movimento horizontal

Função que modifica o estado a partir de um evento de teclado.

```
/// Desloca *q* para a esquerda (substrai *deslocamento* de *x*) se *tecla* for
/// ArrowLeft e para direita (soma *deslocamento* a *x*) se tecla for ArrowRight.
/// Devolve *q* para outras teclas.
pub fn move(q: Quadrado, tecla: world.Key) -> Quadrado {
  case tecla {
    world.ArrowLeft -> Quadrado(..q, x: q.x - deslocamento)
    world.ArrowRight -> Quadrado(..q, x: q.x + deslocamento)
    _ -> q
  }
}

pub fn move_examples() {
  check.eq(move(Quadrado(50, Verde), world.ArrowLeft), Quadrado(50 - deslocamento, Verde))
  check.eq(move(Quadrado(50, Verde), world.ArrowRight), Quadrado(50 + deslocamento, Verde))
}
```

Exemplo movimento horizontal

Função que modifica a cor com a passagem do tempo

```
/// Muda a cor de *q* da seguinte forma:
/// Verde -> Amarelo, Amarelo -> Vermelho, Vermelho -> Verde
pub fn muda_cor(q: Quadrado) -> Quadrado {
  let cor = case q.cor {
    Verde -> Amarelo
    Amarelo -> Vermelho
    Vermelho -> Verde
  }
  Quadrado(..q, cor: cor)
}
pub fn mudar_cor_examples() {
  check.eq(muda_cor(Quadrado(50, Verde)), Quadrado(50, Amarelo))
  check.eq(muda_cor(Quadrado(50, Amarelo)), Quadrado(50, Vermelho))
  check.eq(muda_cor(Quadrado(50, Vermelho)), Quadrado(50, Verde))
}
```

Exemplo movimento horizontal

```
pub fn main() {
  // Cria um mundo com um estado inicial e uma função de desenho
  world.create(Quadrado(q_x_inicial, Verde), desenha)
  |> world.on_key_down(move) // move é atualiza o estado quando uma tecla é pressionada
  |> world.on_tick(muda_cor) // mudar_cor atualiza o estado a cada tique de relógio
  |> world.tick_rate(3)      // quantas vezes o relógio faz tique por segundo
  |> world.run()
}
```

As funções têm as seguintes assinaturas

```
pub fn create(initial: a, draw: fn(a) -> Image) -> World(a)
pub fn on_key_down(world: World(a), handler: fn(a, Key) -> a) -> World(a)
pub fn on_tick(world: World(a), handler: fn(a) -> a) -> World(a)
pub fn tick_rate(world: World(a), rate: Int) -> World(a)
pub fn run(world: World(a)) -> Nil
```

Tipo `word.Key`

```
pub type Key {
  ArrowLeft
  ArrowRight
  ArrowUp
  ArrowDown
  PageUp
  PageDown
  Home
  End
  Backspace
  Tab
  Enter
  Escape
  Delete
  Insert
  F1
  F2
  F3
  F4
  F5
  F6
  F7
  F8
  F9
  F10
  F11
  F12
  CapsLock
  NumLock
  ScrollLock
  PrintScreen
  Pause
  Shift
  Control
  Alt
  Meta
  Char(String)
}
```

Revisão

Vimos o que é uma imagem e como criar, combinar e transformar imagens.

Vimos o que é uma animação e como criar animações coma função `world.animate`.

Vimos o que é um programa interativo e como projetar programas interativos usando o módulo `world`.