



# Introdução a Programação de Jogos

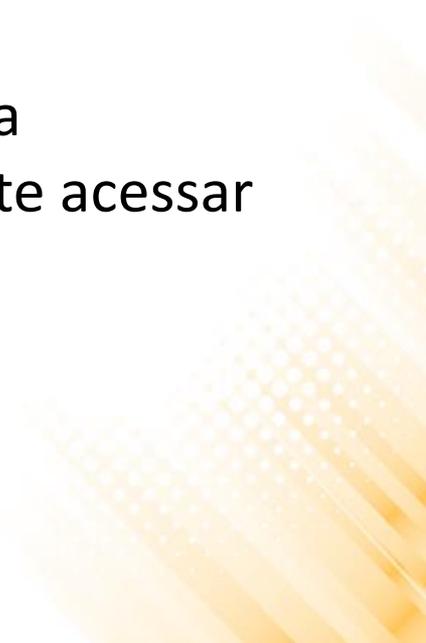
## Aula 08 – Vetores e Matrizes

Edirlei Soares de Lima  
<elima@inf.puc-rio.br>

# Introdução

- Até agora nós temos usado variáveis simples para armazenar valores usados por nossos programas.
- Em várias situações, precisamos armazenar um **conjunto de valores**.
- A partir de agora vamos aprender a usar um mecanismo que nos permite armazenar um conjunto de valores na memória do computador.
  - Posteriormente, estes valores podem ser livremente processados de forma eficiente, pois já estariam na memória do computador.

# Vetores

- Podemos armazenar um conjunto de valores na memória do computador através do uso de **vetores (arrays)**
  - O vetor é a forma mais simples de **organizarmos dados** na memória do computador.
  - Com vetores, os valores são armazenados de forma **sequencial**, um após o outro, e podemos livremente acessar qualquer valor do conjunto.
- 

# Tabelas em Lua

- Em Lua, todas as estruturas de dados, incluindo vetores e matrizes, são representadas através de **tabelas**.
- Lua implementa as tabelas de forma bem eficiente.
- Muitos algoritmos e estruturas de dados são implementados de forma muito mais simples usando tabelas.
- Uma tabela Lua é definida por um conjunto de **pares de chaves e dados**, onde os dados são referenciados pelas chaves.
  - A chave (índice) pode ser de qualquer tipo de dado (exceto nulo).

# Vetores em Lua

- Em Lua, vetores são implementados através da **indexação de tabelas** usando números inteiros.
- Diferente de outras linguagens de programação, em Lua **não precisamos definir o tamanho** máximo de um vetor.
- **Criação de um vetor em Lua:**

```
meu_vetor = {}
```

# Vetores em Lua

- Inicialização de algumas posições do vetor:

```
meu_vetor[1] = 5  
meu_vetor[2] = 11  
meu_vetor[5] = 0  
meu_vetor[10] = 3
```

5	11	?	?	0	?	?	?	?	3
<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>	<i>10</i>

# Vetores em Lua

- É possível **acessar os valores do vetor** através de seu **índice**.

```
meuvetor = { }
```

1	2	3	4	5
5	?	?	8	1

```
meuvetor[1] = 5
```

```
meuvetor[4] = 8
```

```
meuvetor[5] = 1
```

# Vetores em Lua

- **Declaração e inicialização de um vetor:**

```
a = {}      -- novo vetor

for i=1, 1000, 1 do
    a[i] = 0
end
```

- Indiretamente estamos definindo que o tamanho máximo do vetor é 1000.
- Também é possível declarar e inicializar o vetor em uma única expressão:

```
squares = {1, 4, 9, 16, 25, 36, 49, 64}
```

# Exemplo 1: Imprimindo os Valores Armazenados em um Vetor

```
vetor = {2.3, 5.4, 1.0, 7.6, 8.8, 3.9}

for x = 1, 6, 1 do
    io.write(vetor[x], "\n")
end
```

- Em ordem inversa:

```
vetor = {2.3, 5.4, 1.0, 7.6, 8.8, 3.9}

for x = 6, 1, -1 do
    io.write(vetor[x], "\n")
end
```

# Exemplo 2: Somatório dos Valores Armazenados em um Vetor

```
vetor = {2.3, 5.4, 1.0, 7.6, 8.8, 3.9}
```

```
total = 0
```

```
for x = 1, 6, 1 do  
    total = total + vetor[x]  
end
```

```
io.write(total)
```

# Exemplo 3: Encontrar o Maior Valor

```
vetor = {2.3, 5.4, 1.0, 7.6, 8.8, 3.9}

maior = vetor[1]

for x = 2, 6, 1 do
    if vetor[x] > maior then
        maior = vetor[x]
    end
end

io.write(maior)
```

# De Volta ao “Hello World”

- Na ultima implementação do “Hello World”, fizemos 20 “Hello World’s” se moverem na tela.
  - **Problema:** os 20 “Hello World’s” se moviam juntos



- Como podemos fazer cada “Hello World” se mover de forma independente dos outros?

```
local px      -- posição x do texto

function love.load()
    love.graphics.setColor(0, 0, 0)
    love.graphics.setBackgroundColor(255, 255, 255)
    px = 0
end

function love.update(dt)
    px = px + (100 * dt)
    if px > love.window.getWidth() then
        px = 0
    end
end

function love.draw()
    for y = 0, 20, 1 do
        love.graphics.print("Hello World", px, y * 30)
    end
end
```

```
local vetor_px = {}      -- vetor de posições x do texto
```

```
function love.load()
```

```
    love.graphics.setColor(0, 0, 0)
```

```
    love.graphics.setBackgroundColor(255, 255, 255)
```

```
    for y = 1, 20, 1 do
```

```
        vetor_px[y] = love.math.random(0, 800)
```

```
    end
```

```
end
```

```
function love.update(dt)
```

```
    for y = 1, 20, 1 do
```

```
        vetor_px[y] = vetor_px[y] + (100 * dt)
```

```
        if vetor_px[y] > love.window.getWidth() then
```

```
            vetor_px[y] = 0
```

```
        end
```

```
    end
```

```
end
```

```
function love.draw()
```

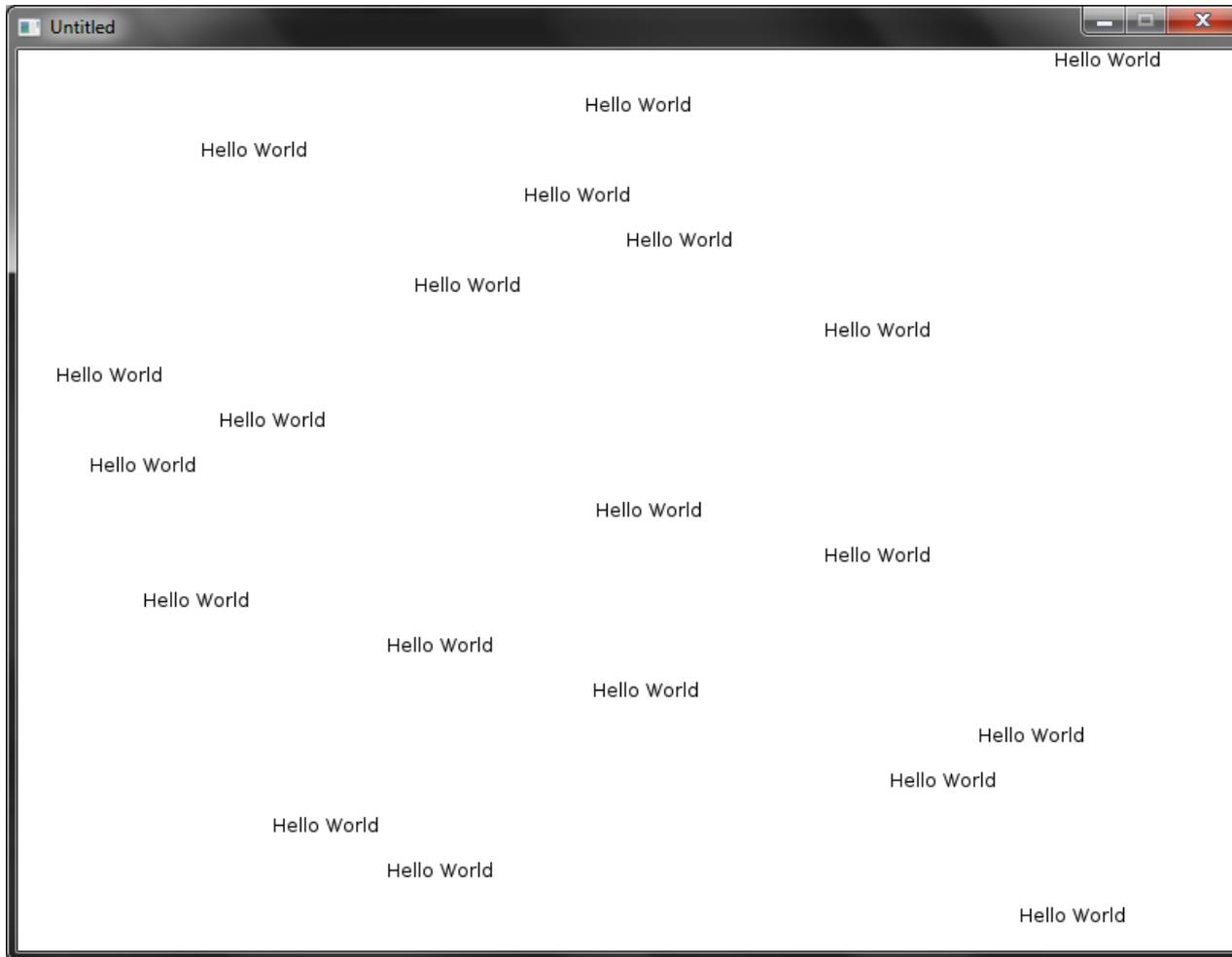
```
    for y = 1, 20, 1 do
```

```
        love.graphics.print("Hello World", vetor_px[y], y * 30)
```

```
    end
```

```
end
```

# De Volta ao “Hello World”



# Vetores e Animações

- Animações são criadas através de **sequencias de imagens**.

– **Exemplo:**



- É possível armazenar as animações em vetores de imagens.
  - Facilita a manipulação e execução das animações

Imagens: [http://www.inf.puc-rio.br/~elima/intro-eng/imagens\\_hero.zip](http://www.inf.puc-rio.br/~elima/intro-eng/imagens_hero.zip)

```
local hero_walk = {}  -- vetor de imagens
local hero_anim_frame = 1
local hero_pos_x = 100
local hero_pos_y = 225

function love.load()
  for x = 1, 4, 1 do  -- carrega as imagens da animação
    hero_walk[x] = love.graphics.newImage("Hero_Walk_0" .. x .. ".png")
  end
end

function love.update(dt)
  if love.keyboard.isDown("right") then

    hero_pos_x = hero_pos_x + (100 * dt)  -- movimenta o personagem
    hero_anim_frame = hero_anim_frame + 1  -- incrementa a animação

    if hero_anim_frame > 4 then  -- loop da animação
      hero_anim_frame = 1
    end
  end
end

function love.draw()  -- desenha o personagem usando o indice da animação
  love.graphics.draw(hero_walk[hero_anim_frame], hero_pos_x, hero_pos_y)
end
```

# Exemplo de Animação



- **Problema:** não controlamos a velocidade da animação!
  - Quanto mais rápido o computador, mais rápida será a animação

```
local hero_walk = {}
local hero_anim_frame = 1
local hero_pos_x = 100
local hero_pos_y = 225
local hero_anim_time = 0  -- variavel para controle do tempo da animação

function love.load()
  for x = 1, 4, 1 do
    hero_walk[x] = love.graphics.newImage("Hero_Walk_0" .. x .. ".png")
  end
end

function love.update(dt)
  if love.keyboard.isDown("right") then
    hero_pos_x = hero_pos_x + (100 * dt)
    hero_anim_time = hero_anim_time + dt  -- incrementa o tempo usando dt
    if hero_anim_time > 0.1 then  -- quando acumular mais de 0.1
      hero_anim_frame = hero_anim_frame + 1 -- avança para proximo frame
      if hero_anim_frame > 4 then
        hero_anim_frame = 1
      end
      hero_anim_time = 0  -- reinicializa a contagem do tempo
    end
  end
end

function love.draw()
  love.graphics.draw(hero_walk[hero_anim_frame], hero_pos_x, hero_pos_y)
end
```

# Tabelas em Lua

- As tabelas da linguagem Lua permitem que nomes sejam associados aos seus elementos.
- Isso permite a criação de estruturas:

```
player1 = {  
    nome      = "Pedro",  
    image     = love.graphics.newImage("Pedro.png"),  
    pontos    = 1000,  
    vidas     = 3,  
    forca     = 50  
    px        = 300  
    py        = 300  
}
```

# Tabelas em Lua

- É possível acessar os elementos de uma tabela pelo seu nome:

```
io.write(player1.pontos)
```

```
•  
•
```

```
love.graphics.draw(player1.image, player1.px,  
                    player1.py)
```

```
•  
•
```

# Exercício 1

1) Modifique a implementação do exemplo de animação de forma a organizar as variáveis referentes ao personagem dentro de uma tabela estruturada:

```
local hero_walk = {}  
local hero_anim_frame = 1  
local hero_pos_x = 100  
local hero_pos_y = 225  
local hero_anim_time = 0
```

- Faça também as modificações necessárias para que o resto do programa utilize a estrutura de tabela.

# Exercício 2

2) Continue a implementação do exercício anterior adicionando a animação do personagem andando nas outras direções usando as imagens abaixo:



Imagens: [http://www.inf.puc-rio.br/~elima/intro-eng/imagens\\_hero.zip](http://www.inf.puc-rio.br/~elima/intro-eng/imagens_hero.zip)

# Matrizes

- Uma **matriz** representa e armazena um conjunto bidimensional de valores na memória do computador.
- É uma **tabela de variáveis** de mesmo tipo que ocupa uma região contínua de memória.
- Exemplo de matriz numérica:

3	1	8	6	1
7	2	5	4	9
1	9	3	1	2
5	8	6	7	3
6	4	9	2	1

# Matrizes em Lua

- **Declaração e inicialização de uma matriz:**

```
minha_matriz = {}          -- nova matriz

for i=1, 10, 1 do
  minha_matriz[i] = {}    -- nova linha
  for j=1, 10, 1 do
    minha_matriz[i][j] = 0
  end
end
end
```

- Estamos definindo e inicializando uma matriz de 10 colunas e 10 linhas.

# Matrizes

- É possível **acessar os valores da matriz** através de seu **índice bidimensional**.

	0	1	2
0	5	?	1
1	?	?	?
2	?	8	?

```
minha_matriz[0][0] = 5;
```

```
minha_matriz[1][2] = 8;
```

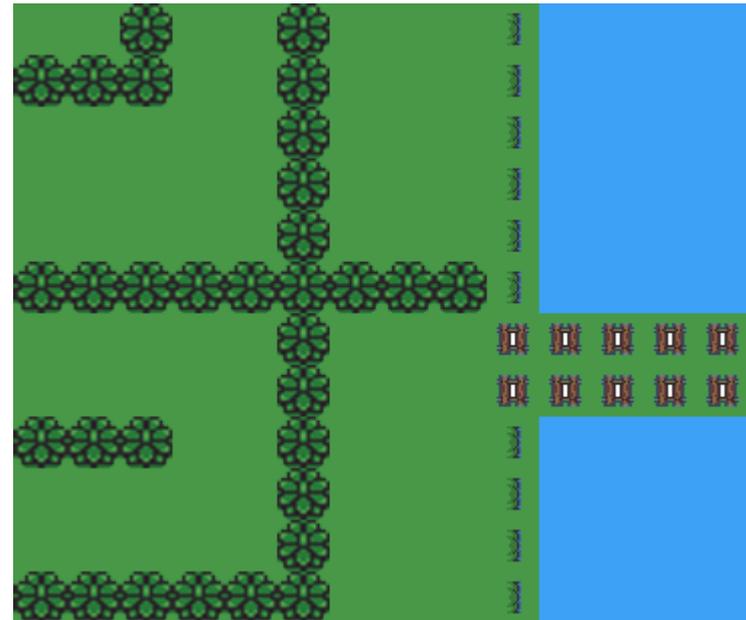
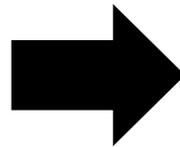
```
minha_matriz[2][0] = 1;
```

# Matrizes e Mapas

- É possível utilizar matrizes para representar cenários e mapas.

- **Exemplo:**

```
XXGXXGXXXEAAAA  
GGGXXGXXXEAAAA  
XXXXXGXXXEAAAA  
XXXXXGXXXEAAAA  
XXXXXGXXXEAAAA  
GGGGGGGGGEAAAA  
XXXXXGXXXPPPPP  
XXXXXGXXXPPPPP  
GGGXXGXXXEAAAA  
XXXXXGXXXEAAAA  
XXXXXGXXXEAAAA  
GGGGGGXXXEAAAA
```



# Matrizes e Mapas – Exemplo 1

- Gerar aleatoriamente uma matriz desenhá-la na tela usando cores.

```
1312103010112210
2113021120130132
1203201321012021
2113013023120231
3120312031031201
2010311103013130
0310312130310321
1010101203102031
3210321013210130
3203132031201110
```

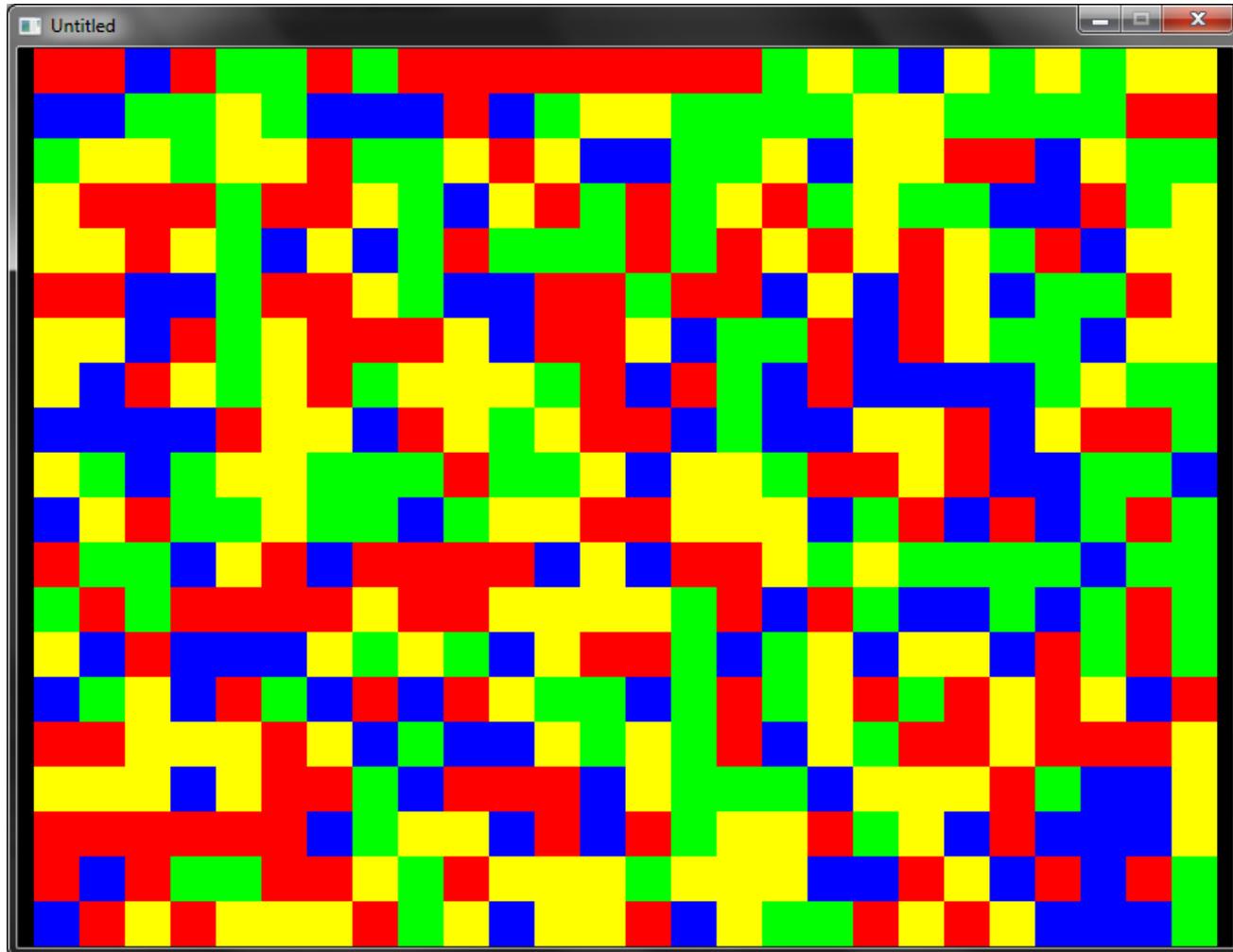


```
local mapa = {}

function love.load()
  for i=1, 26, 1 do    --Gera aleatoriamente uma matriz 26 x 20
    mapa[i] = {}
    for j=1, 20, 1 do
      mapa[i][j] = love.math.random(0, 3)
    end
  end
end

function love.draw()
  for i=1, 26, 1 do  --Percorre a matriz e desenha quadrados coloridos
    for j=1, 20, 1 do
      if (mapa[i][j] == 0) then
        love.graphics.setColor(255,0,0)
      elseif (mapa[i][j] == 1) then
        love.graphics.setColor(0,255,0)
      elseif (mapa[i][j] == 2) then
        love.graphics.setColor(0,0,255)
      elseif (mapa[i][j] == 3) then
        love.graphics.setColor(255,255,0)
      end
      love.graphics.rectangle("fill", (i * 30)-20, (j * 30)-30, 30, 30)
    end
  end
end
```

# Matrizes e Mapas – Exemplo 1



# Matrizes e Mapas – Exemplo 2

- Ler uma matriz de um arquivo de texto desenhá-la na tela usando cores.

Mapa.txt

```
GGGGGGAGGGGGGGG
GGGGGGAGGGGGGGG
GGGGGGAGGGGGGGG
GGGGGGAGGGGGGGG
GGGGGGAAAGGGGGG
GGGGGGGGAGGGGGG
GGGGGGGGAGGGGGG
PPPPPPPPAPPPPP
AAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAA
```

# Matrizes e Mapas – Exemplo 2

```
local mapa = {}

function LoadMap(filename)      --Le o conteúdo do arquivo para a matriz
  local file = io.open(filename)
  local i = 1
  for line in file:lines() do
    mapa[i] = {}
    for j=1, #line, 1 do
      mapa[i][j] = line:sub(j,j)
    end
    i = i + 1
  end
  file:close()
end

function love.load()
  LoadMap("Mapa.txt")
end

.
.
.
```

# Matrizes e Mapas – Exemplo 2

.  
. .  
. .

```
function love.draw()
  for i=1, 10, 1 do    --Percorre a matriz e desenha quadrados coloridos
    for j=1, 14, 1 do
      if (mapa[i][j] == "P") then
        love.graphics.setColor(230,235,134)
      elseif (mapa[i][j] == "G") then
        love.graphics.setColor(39,153,0)
      elseif (mapa[i][j] == "A") then
        love.graphics.setColor(63,125,232)
      end
      love.graphics.rectangle("fill", (j * 50), (i * 50), 50, 50)
    end
  end
end
```



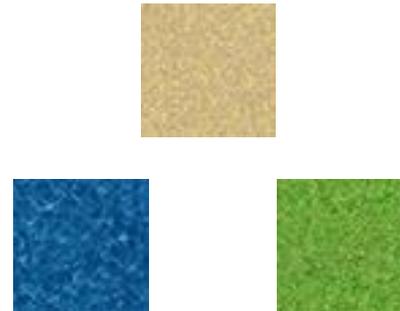
# Matrizes e Mapas – Exemplo 3

- Ler uma matriz de um arquivo de texto desenhá-la na tela usando imagens.

Mapa.txt

```
GGGGGGGAGGGGGGGG
GGGGGGGAGGGGGGGG
GGGGGGGAGGGGGGGG
GGGGGGGAGGGGGGGG
GGGGGGGAAAGGGGGG
GGGGGGGGGAGGGGGG
GGGGGGGGGAGGGGGG
GGGGGGGGGAGGGGGG
PPPPPPPPAPPPPP
AAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAA
```

Imagens:



# Matrizes e Mapas – Exemplo 3

```
local mapa = {}
local tile_grass
local tile_water
local tile_sand

function LoadMap(filename)      --Le o conteúdo do arquivo para a matriz
    local file = io.open(filename)
    local i = 1
    for line in file:lines() do
        mapa[i] = {}
        for j=1, #line, 1 do
            mapa[i][j] = line:sub(j,j)
        end
        i = i + 1
    end
    file:close()
end

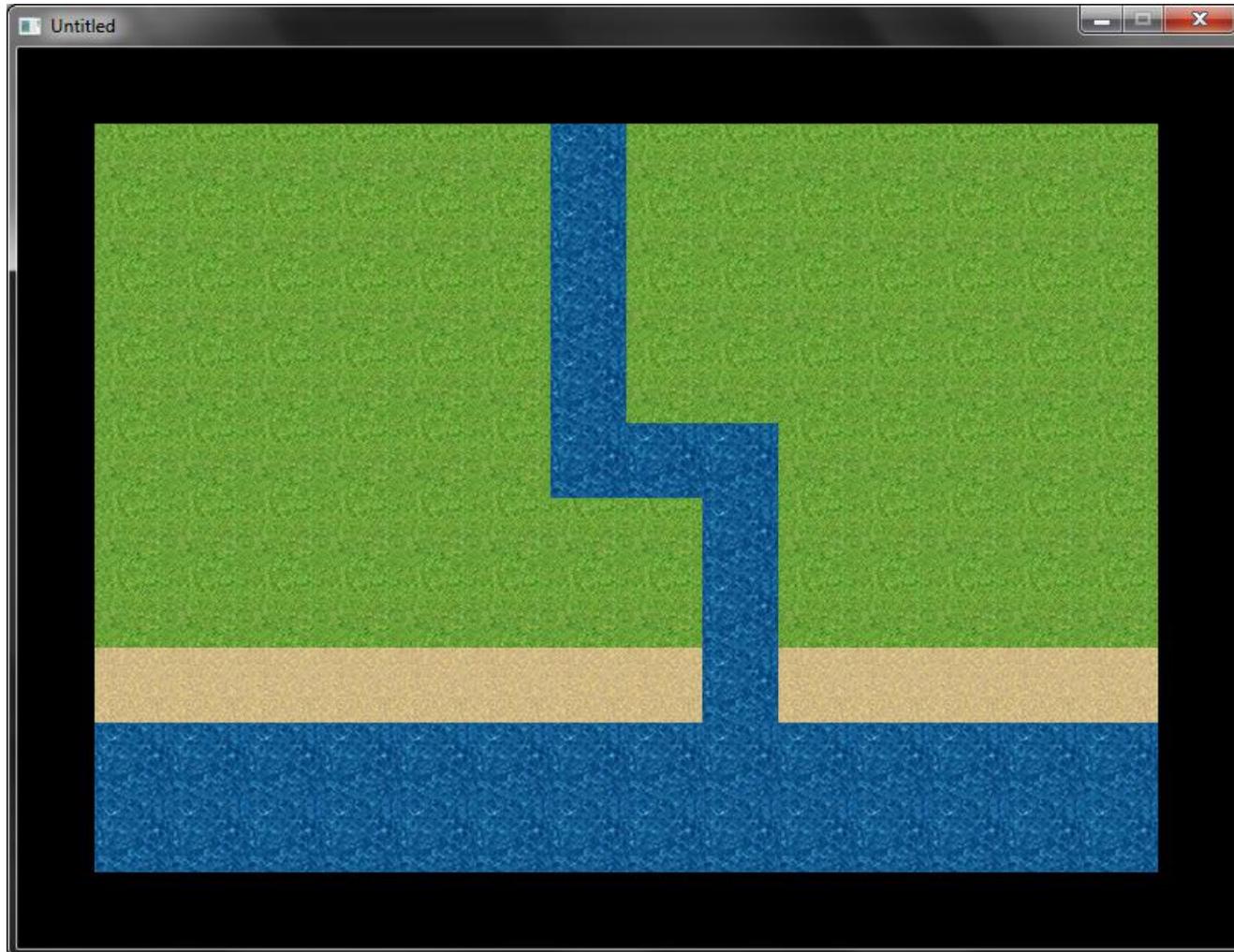
.
.
.
```

- 
- 
- 

```
function love.load()  
  LoadMap("Mapa.txt")  
  tile_grass = love.graphics.newImage("grass.png")  
  tile_water = love.graphics.newImage("water.png")  
  tile_sand = love.graphics.newImage("sand.png")  
end
```

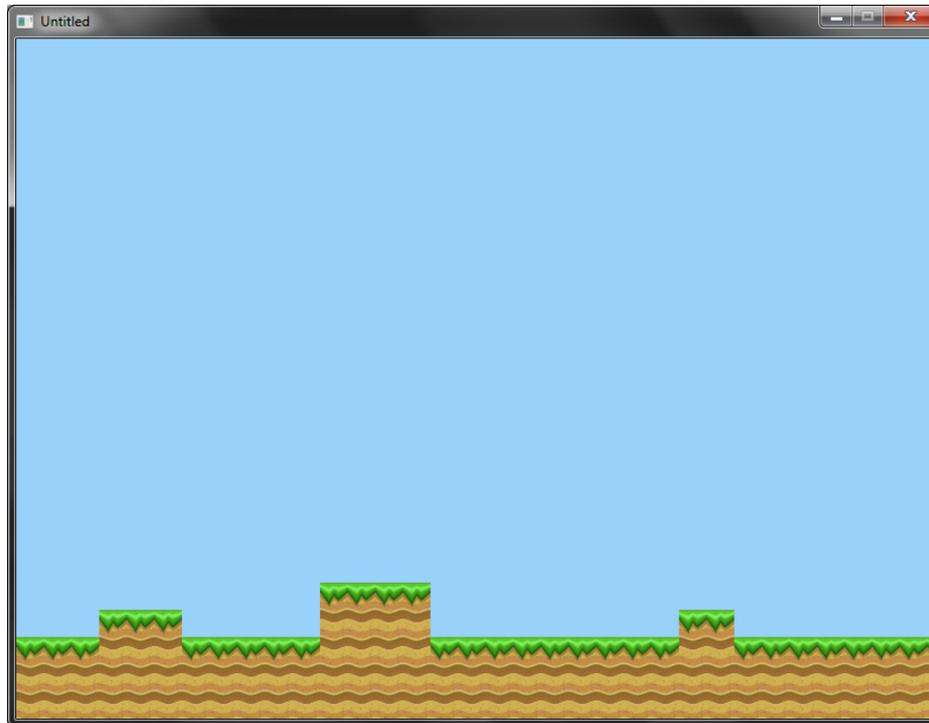
```
function love.draw()  
  for i=1, 10, 1 do      --Percorre a matriz e desenha quadrados imagens  
    for j=1, 14, 1 do  
      if (mapa[i][j] == "P") then  
        love.graphics.draw(tile_sand, (j * 50), (i * 50))  
      elseif (mapa[i][j] == "G") then  
        love.graphics.draw(tile_grass, (j * 50), (i * 50))  
      elseif (mapa[i][j] == "A") then  
        love.graphics.draw(tile_water, (j * 50), (i * 50))  
      end  
    end  
  end  
end  
end
```

# Matrizes e Mapas – Exemplo 3



# Exercício 3

3) Implemente um programa para exibir o mapa de um jogo de plataforma definido em um arquivo de texto.



Mapa: [http://www.inf.puc-rio.br/~elima/intro-eng/mapa\\_exercicio3.zip](http://www.inf.puc-rio.br/~elima/intro-eng/mapa_exercicio3.zip)

# Exercício 4

4) Continue o programa desenvolvido no exercício anterior implementando uma câmera virtual para permitir que o usuário possa movimentar a câmera e visualizar todo o mapa da cenário.

