

Programação com a linguagem Céu

Imagens

Anny Caroline

`annycarolinegnr@gmail.com`

Francisco Sant'Anna

`francisco@ime.uerj.br`

Carregando uma imagem

- A imagem deve estar no formato **bmp**
- A imagem deve estar na pasta **res** do seu projeto



Carregando uma imagem

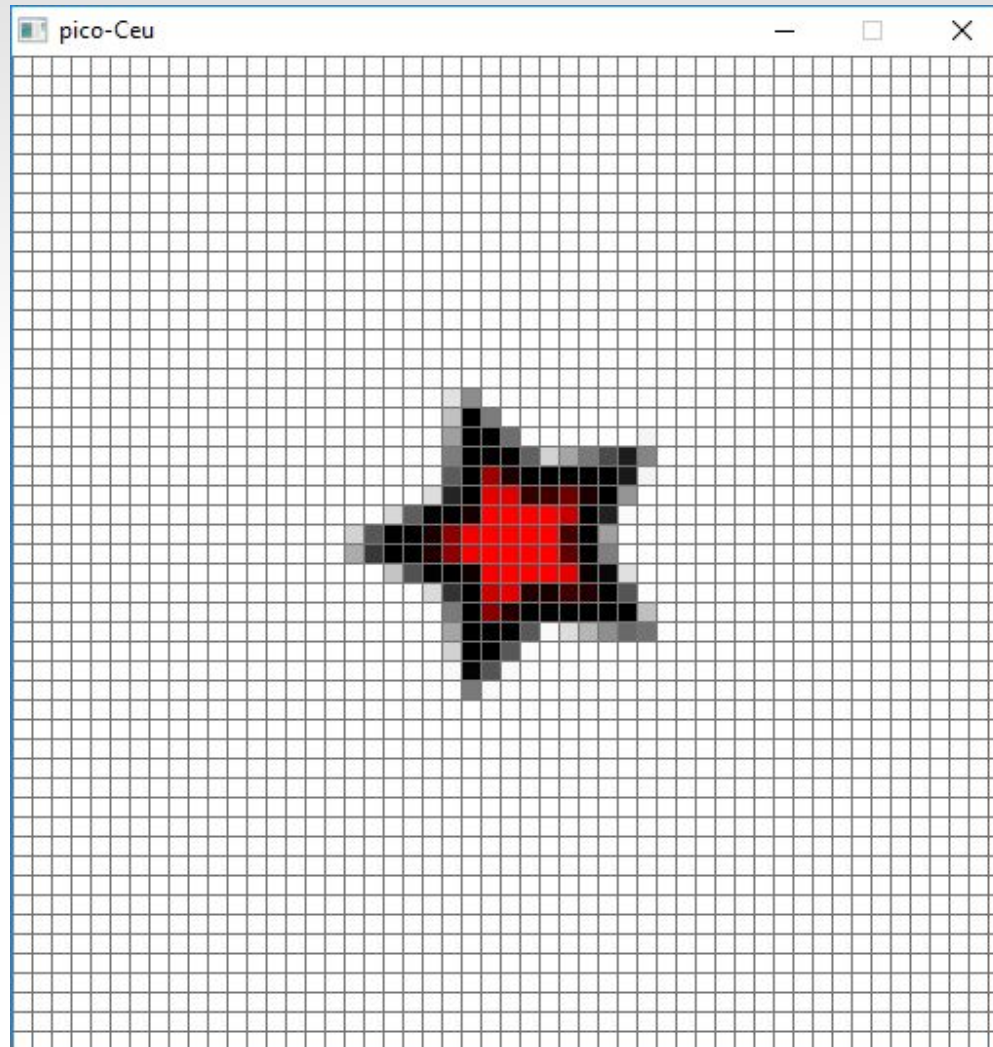
- Para o nosso primeiro exemplo, vamos usar a imagem de uma estrela
- [Baixe aqui as imagens que vamos usar nesse slide](#)



Carregando uma imagem

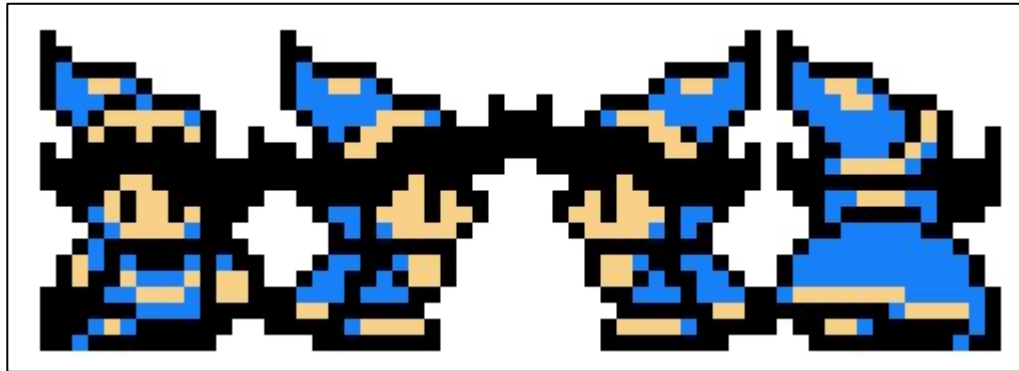
```
emit WINDOW_SET_CLEAR_COLOR_NAME(COLOR_WHITE);  
emit WINDOW_CLEAR;  
  
emit GRAPHICS_DRAW_BMP(0,0, "res/star.bmp");
```

Carregando uma imagem



Carregando outra imagem

- Agora, vamos usar uma outra imagem

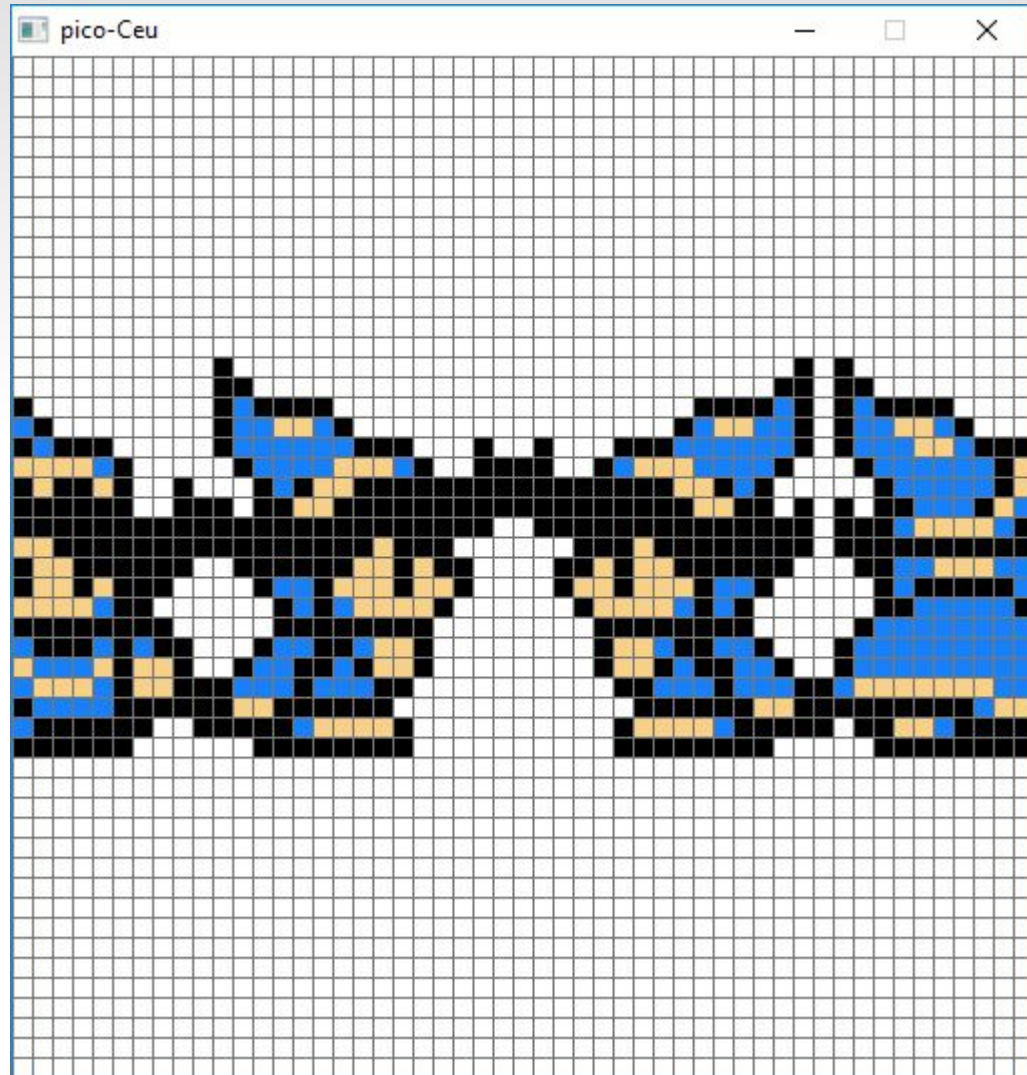


Carregando outra imagem

```
emit WINDOW_SET_CLEAR_COLOR_NAME(COLOR_WHITE);  
emit WINDOW_CLEAR;  
  
emit GRAPHICS_DRAW_BMP(0,0, "res/player1.bmp");
```

2-player.ceu

Carregando outra imagem



Frame

- Primeiro, vamos definir qual frame vamos utilizar

```
emit WINDOW_SET_CLEAR_COLOR_NAME(COLOR_WHITE);
```

```
emit WINDOW_CLEAR;
```

```
emit GRAPHICS_SET_BMP_FRAME(0, 4);
```

```
emit GRAPHICS_DRAW_BMP(0, 0, "res/player1.bmp");
```

3-playerFrame.ceu

Escala de desenho

- Para diminuir o tamanho do nosso jogador, vamos modificar a escala de desenho

```
emit GRAPHICS_SET_SCALE(0.5, 0.5);
```

- “Altera a escala de desenho de todas as operações de desenho subsequentes **GRAPHICS_DRAW_BMP**, **GRAPHICS_DRAW_RECT** e **GRAPHICS_DRAW_TEXT**.”

Escala de desenho

```
emit WINDOW_SET_CLEAR_COLOR_NAME(COLOR_WHITE);
```

```
emit WINDOW_CLEAR;
```

```
emit GRAPHICS_SET_BMP_FRAME(0, 4);
```

```
emit GRAPHICS_SET_SCALE(0.3, 0.3);
```

```
emit GRAPHICS_DRAW_BMP(0,0, "res/player1.bmp");
```

Exercício

- Modifique o nosso jogo para adicionar o desenho do jogador

Exercício

- Adicione a imagem **player2.bmp** ao repositório **res**
- Modifique a abstração de código **Pixel** para exibir a imagem **player1.bmp** ou **player2.bmp** dependendo do id do jogador

Exercício

- Renomeie a abstração **Pixel** para **Coin**
- Adicione a imagem **coin.bmp** ao diretório **res**
- Desenha a imagem baixada na abstração de código **Coin** (antiga **Pixel**), assim como fizemos com os jogadores



Animação de moeda girando

- Vamos substituir a animação de pixel “piscando”, pela de uma moeda girando



- A ideia é ir modificando qual frame deve ser exibido de tempos em tempos

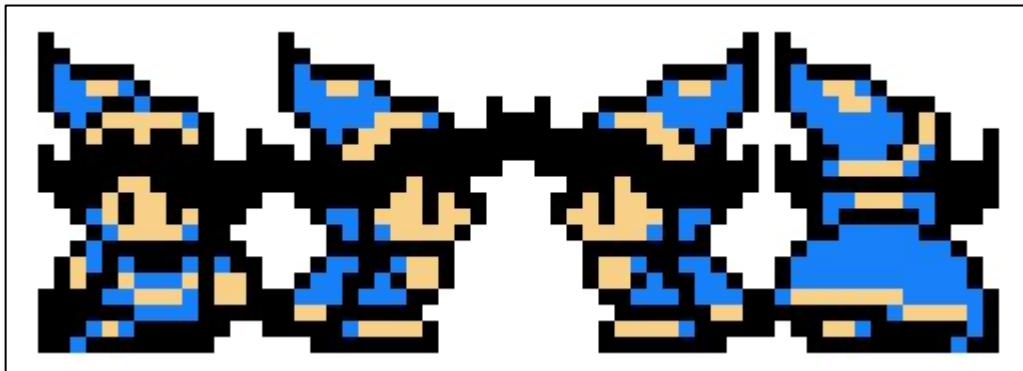
```

code/await Coin (none) -> (var Point pt) -> none do
  pt = call Random_Point_In_Square(-25,-25,25,17);
  var int coinFrame = 0;
  par/or do
    every 50ms do
      coinFrame = (coinFrame + 1) % 9;
    end
  with
    every FRAMES_REDRAW do
      //desenha bmp
      emit GRAPHICS_SET_BMP_FRAME(coinFrame, 9);
      emit GRAPHICS_SET_SCALE(0.2,0.2);
      emit GRAPHICS_DRAW_BMP(pt.x, pt.y, "res/coin.bmp");
      //desenha pixel
      emit GRAPHICS_SET_SCALE(1.0, 1.0);
      emit GRAPHICS_SET_COLOR_NAME(COLOR_RED);
      emit GRAPHICS_DRAW_PIXEL(pt.x, pt.y);
    end
  with // await MOUSE_CLICK;
  with // await outer.game_over;
end
end

```


Exercício

- Modifique nosso jogo para que o jogador se virar para onde está se movendo

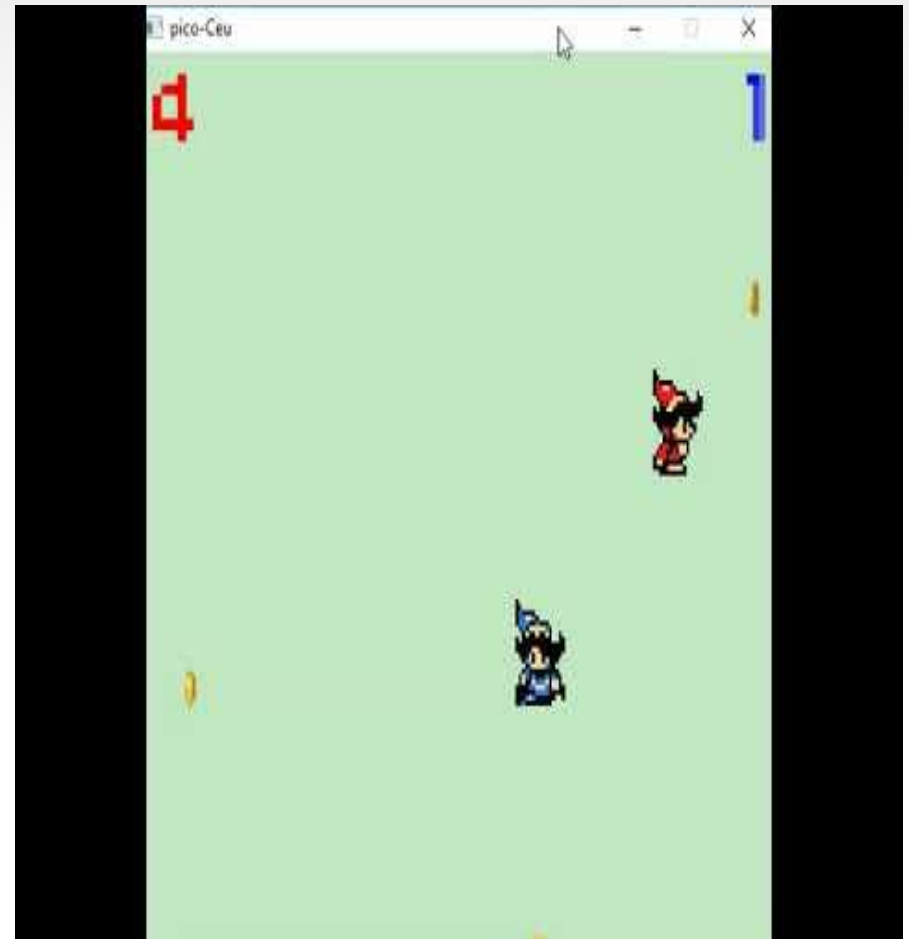


0

1

2

3



Corrigindo a colisão

- Do jeito que implementamos, a colisão não está sendo feita entre as imagens, e sim entre os pixels



Substituindo o Pixel por um Retângulo



No code/await Player

```
code/await Player(var int id, var int up, var int  
right, var int down, var int left, var Color  
color) -> (var Rect rect) -> none do  
    rect = val Rect(0,0,4,5); //x,y,h,w  
    //...  
end
```

No code/await Player

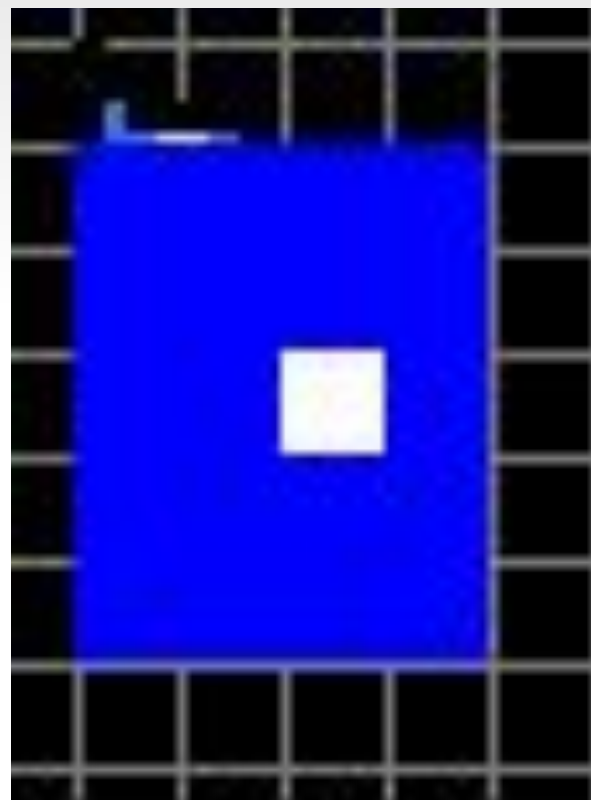
```
every FRAMES_UPDATE do
  //atualiza pt com base em x_axis e y_axis
  if (x_axis == 1) and (rect.x < 25) then
    rect.x = rect.x + 1;
    playerFrame = 1;
  else/if (x_axis == -1) and (rect.x > -25) then
    rect.x = rect.x - 1;
    playerFrame = 2;
  end

  if (y_axis == 1) and (rect.y < 25) then
    rect.y = rect.y + 1;
    playerFrame = 3;
  else/if (y_axis == -1) and (rect.y > -25) then
    rect.y = rect.y - 1;
    playerFrame = 0;
  end
end
end
```

No code/await Player

```
every FRAMES_REDRAW do
  // desenha bmp ...
  //desenha score
  emit GRAPHICS_SET_SCALE(1.0,1.0);
  emit GRAPHICS_SET_COLOR_NAME(color);
  if (id == 1) then
    emit GRAPHICS_DRAW_INT(-21,21,score);
  else
    emit GRAPHICS_DRAW_INT(23,21,score);
  end
  // desenha rect
  emit GRAPHICS_DRAW_RECT(rect.x, rect.y, rect.w, rect.h); //x,y,w,h
  emit GRAPHICS_SET_COLOR_NAME(COLOR_WHITE);
  emit GRAPHICS_DRAW_PIXEL(rect.x, rect.y);
end
```

No code/await Player



No code/await Pixel

```
code/await Coin (none) -> (var Rect rect) -> none do
```

```
var Point pt = call Random_Point_In_Square(-25,-25,25,17);  
rect = val Rect(pt.x, pt.y, 2, 2);
```

```
//...
```

```
end
```


No code/await Pixel

```
every FRAMES_REDRAW do
```

```
  //desenha bmp
```

```
  emit GRAPHICS_SET_BMP_FRAME(coinFrame, 9);
```

```
  emit GRAPHICS_SET_SCALE(0.2,0.2);
```

```
  emit GRAPHICS_DRAW_BMP(pt.x, pt.y, "res/coin.bmp");
```

```
  //desenha rect
```

```
    emit GRAPHICS_SET_SCALE(1.0, 1.0);
```

```
    emit GRAPHICS_SET_COLOR_NAME(COLOR_RED);
```

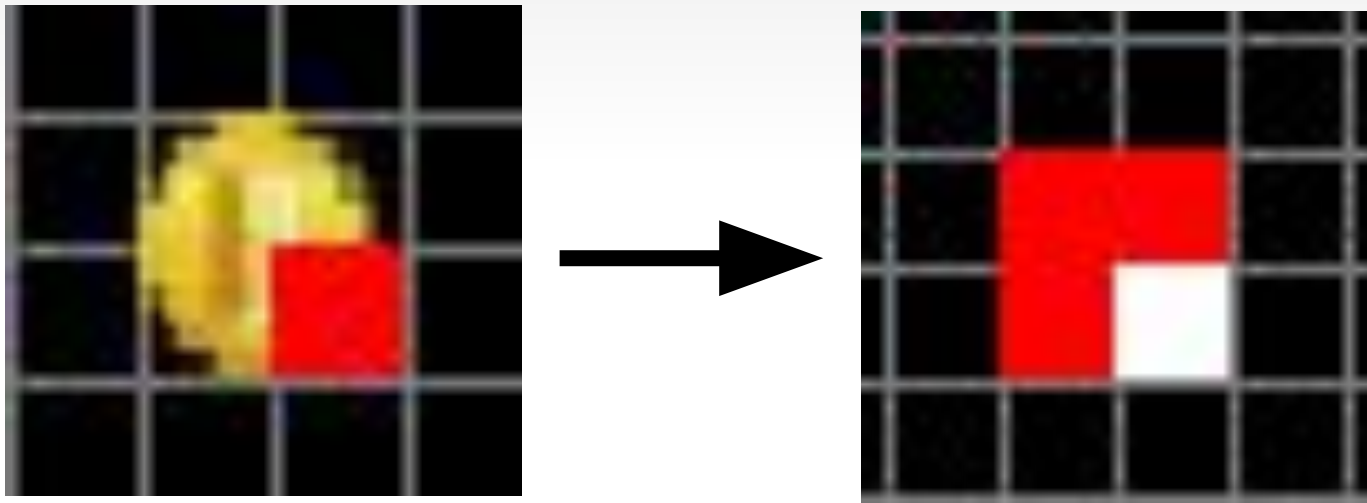
```
    emit GRAPHICS_DRAW_RECT(rect.x, rect.y, rect.w, rect.h);
```

```
    emit GRAPHICS_SET_COLOR_NAME(COLOR_WHITE);
```

```
    emit GRAPHICS_DRAW_PIXEL(rect.x, rect.y);
```

```
end
```

No code/await Pixel



No programa principal

■ Relembrando...

```
// Aqui vamos tratar a colisão
every FRAMES_UPDATE do
  var&? Player player;
  loop player in players do
    var&? Coin coin;
    loop coin in coins do
      if (player!.pt.x == coin!.pt.x) and
        (player!.pt.y == coin!.pt.y) then
        kill coin;
        emit collision(player!.id);
      end
    end
  end
end
end
```

No programa principal

```
// Aqui vamos tratar a colisão
every FRAMES_UPDATE do
  var&? Player player;
  loop player in players do
    var&? Coin coin;
    loop coin in coins do
      if (call Intersects(&player!.rect,&coin!.rect)) then
        kill coin;
        emit collision(player!.id);
      end
    end
  end
end
end
```

code/call Intersects

```
code/call Intersects (var& Rect rect1, var& Rect rect2) -> yes/no do
  escape rect1.x+rect1.w/2 >= rect2.x-rect2.w/2 and
    rect1.x-rect1.w/2 <= rect2.x+rect2.w/2 and
    rect1.y+rect1.h/2 >= rect2.y-rect2.h/2 and
    rect1.y-rect1.h/2 <= rect2.y+rect2.h/2;
end
```

10-collision.ceu