

# Programação com a linguagem Céu

Colisão entre **Players** e **Pixels**

**Anny Caroline**

`annycarolinegnr@gmail.com`

**Francisco Sant'Anna**

`francisco@ime.uerj.br`

# Quando há uma colisão?

- Quando um **Player** está na mesma posição de um **Pixel**

# Em que parte do código?

■ No **P**layer

■ No **P**ixel

■ “Programa principal”

# Quando verificar?

- Vamos verificar a cada frame
- Poderia ser somente quando um **Player** se movimenta, ou quando um **Pixel** surge

# Colisão

```
pool[5] Pixel pixels;  
pool[2] Player players;  
  
spawn Player(KEY_UP, KEY_RIGHT, KEY_DOWN, KEY_LEFT, COLOR_BLUE) in players;  
spawn Player(KEY_w , KEY_d      , KEY_s      , KEY_a      , COLOR_RED) in players;
```

# Colisão

```
par do
  every 1s do
    spawn Pixel() in pixels;
  end
with
  // Aqui vamos tratar a colisão
end
```

# Colisão

```
every FRAMES_UPDATE do

    var& Player player;
    loop player in players do

        var&? Pixel pixel;
        loop pixel in pixels do
            // verifico a colisão e tomo uma ação
        end
    end

end

end
```

# Acessando as coordenadas

- Para verificar se um **Player** e um **Pixel** colidiram, preciso ter acesso às suas coordenadas



# Acessando as coordenadas

- Para ter acesso às coordenadas do **Player**:

```
code/await Player(var int up, var int right,  
                  var int down, var int left,  
                  var Color color)  
-> (var Point pt) -> NEVER do
```

```
var Point pt = val Point(0,0);
```



```
pt = val Point(0,0);
```

```
player.pt.x
```

# Obs

- Também posso “expor” mais de uma variável

```
code/await Player(var int up, var int right,  
                  var int down, var int left,  
                  var Color color)  
-> (var int x, var int y) -> NEVER do
```

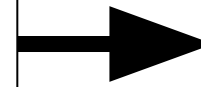
player.x

# Acessando as coordenadas

- Para o **Pixel**, é ligeiramente diferente

```
code/await Pixel (none) -> (var Point pt) -> none do
```

```
var Point pt = call Random_Point();
```



```
pt = call Random_Point();
```

para acessar (no loop)

```
pixel!.pt.x
```

# Exercício

- Como já sabemos como acessar as coordenadas dos **Pixels** e dos **Players**, implemente a verificação da colisão entre eles
- Se houver colisão, execute alguma ação (qualquer uma)

# Solução

```
every FRAMES_UPDATE do
  var& Player player;
  loop player in players do
    var&? Pixel pixel;
    loop pixel in pixels do
      // verifico a colisão e tomo uma ação
      if (player.pt.x == pixel!.pt.x) and
        (player.pt.y == pixel!.pt.y) then
        _printf("Colidiu\n");
      end
    end
  end
end
end
```

# Eliminar o Pixel com a colisão

- Substitua a ação executada quando ocorre colisão pela eliminação do **Pixel** que colidiu com o **Player**

# Eliminar o Pixel com a colisão

```
every FRAMES_UPDATE do
  var& Player player;
  loop player in players do
    var&? Pixel pixel;
    loop pixel in pixels do
      // verifico a colisão e tomo uma ação
      if (player.pt.x == pixel!.pt.x) and
        (player.pt.y == pixel!.pt.y) then
        kill pixel;
      end
    end
  end
end
end
```

# Exercício

Adicione uma pontuação para cada jogador

- Cada jogador inicia com 0 pontos
- O jogador soma um ponto a cada colisão
- A cada colisão, a pontuação do jogador deve ser printada



# Solução

```
code/await Player(var int up, var int right,  
                  var int down, var int left,  
                  var Color color) ->  
  (var Point pt, var int score) -> NEVER do  
  
    score = 0;  
    //...  
  
end
```

```
every FRAMES_UPDATE do
  var& Player player;
  loop player in players do
    var&? Pixel pixel;
    loop pixel in pixels do
      // verifico a colisão e tomo uma ação
      if (player.pt.x == pixel!.pt.x) and
        (player.pt.y == pixel!.pt.y) then
        kill pixel;
        player.score = player.score + 1;
        _printf("%d\n", player.score);
      end
    end
  end
end
end
```

# Exibindo a pontuação na tela

Ao invés de printar no console, vamos desenhar na tela a pontuação de cada jogador

- Vamos utilizar o evento **GRAPHICS\_DRAW\_INT**

```
emit GRAPHICS_DRAW_INT(0,0,10);
```

- O **code/call Random\_Point\_In\_Square** também pode ser importante

```
var Point pt = call Random_Point_In_Square(bl_x,bl_y,tr_x,tr_y);
```

# Onde emitir o evento **GRAPHICS\_DRAW\_INT ?**

- No “programa principal”
  - Junto com a verificação da colisão
  - Em um bloco separado, que responde ao evento **FRAMES\_REDRAW**
- Na abstração **Player**

**Vamos ver caso a caso**

# No “programa principal”, junto com a verificação da colisão

- (o código está no próximo slide)
- Por que não funciona?
  - Lembre-se que a cada chegada do evento **FRAMES\_REDRAW** a tela é apagada, e que ele é emitido logo após o **FRAMES\_UPDATE**.

Nesse caso, como a pontuação é desenhada na resposta do evento **FRAMES\_UPDATE**, ela é logo apagada quando o **FRAMES\_REDRAW** chega.

```
// Aqui vamos tratar a colisão
```

```
var int playerId = 1;
```

```
every FRAMES_UPDATE do
```

```
  var& Player player;
```

```
  loop player in players do
```

```
    var&? Pixel pixel;
```

```
    loop pixel in pixels do
```

```
      // verifico a colisão e tomo uma ação
```

```
      if (player.pt.x == pixel!.pt.x) and (player.pt.y == pixel!.pt.y) then
```

```
        kill pixel;
```

```
        player.score = player.score + 1;
```

```
        if (playerId == 1) then
```

```
          emit GRAPHICS_DRAW_INT(-21,21,player.score);
```

```
        else
```

```
          emit GRAPHICS_DRAW_INT(23,21,player.score);
```

```
        end
```

```
      end
```

```
    end
```

```
    playerId = 2;
```

```
  end
```

```
  playerId = 1;
```

```
end
```

# No “programa principal”, em um bloco separado, que responde ao evento **FRAMES\_REDRAW**

- **Uma** ideia para solução (mostrada no próximo slide) seria copiar o trecho de código responsável pelo desenho da pontuação para um novo bloco paralelo, que responde ao evento **FRAMES\_REDRAW**

```

var int playerId = 1;
var int p1Score = 0; var int p2Score = 0;
par do
    // spawn Pixel every 1s
with
    every FRAMES_UPDATE do
        var& Player player;
        loop player in players do

            var&? Pixel pixel;
            loop pixel in pixels do
                if (player.pt.x == pixel!.pt.x) and (player.pt.y == pixel!.pt.y) then
                    kill pixel;
                    player.score = player.score + 1;
                    if (playerId == 1) then
                        p1Score = player.score;
                    else
                        p2Score = player.score;
                    end
                end
            end
            playerId = 2;
        end
        playerId = 1;
    end
with
    // every FRAMES_REDRAW do
end

```



```
var int playerId = 1;
```

```
var int p1Score = 0; var int p2Score = 0;
```

```
par do
```

```
    // spawn Pixel every 1s
```

```
with
```

```
    // every FRAMES_UPDATE do
```

```
with
```

```
    every FRAMES_REDRAW do
```

```
        emit GRAPHICS_SET_COLOR_NAME(COLOR_BLUE);
```

```
        emit GRAPHICS_DRAW_INT(-21,21,p1Score);
```

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

```
        emit GRAPHICS_DRAW_INT(23,21,p2Score);
```

```
    end
```

```
end
```

# No “programa principal”, em um bloco separado, que responde ao evento `FRAMES_REDRAW`

- Essa solução acaba usando uma maior quantidade de variáveis auxiliares
- Além disso, é preferível que o próprio **Player** seja responsável por manter e desenhar sua pontuação

# Na abstração Player

- Nessa solução, vamos deixar que o `code/await Player` seja responsável desenhar sua pontuação na tela
- Também é interessante que ele seja o responsável por atualizar a sua própria pontuação (veremos isso depois)

```
par do
  every 1s do
    spawn Pixel() in pixels;
  end
with
  // Aqui vamos tratar a colisão
  every FRAMES_UPDATE do
    var& Player player;
    loop player in players do
      var&? Pixel pixel;
      loop pixel in pixels do
        // verifico a colisão e tomo uma ação
        if (player.pt.x == pixel!.pt.x) and (player.pt.y == pixel!.pt.y) then
          kill pixel;
          player.score = player.score + 1;
        end
      end
    end
  end
end
end
end
```

```

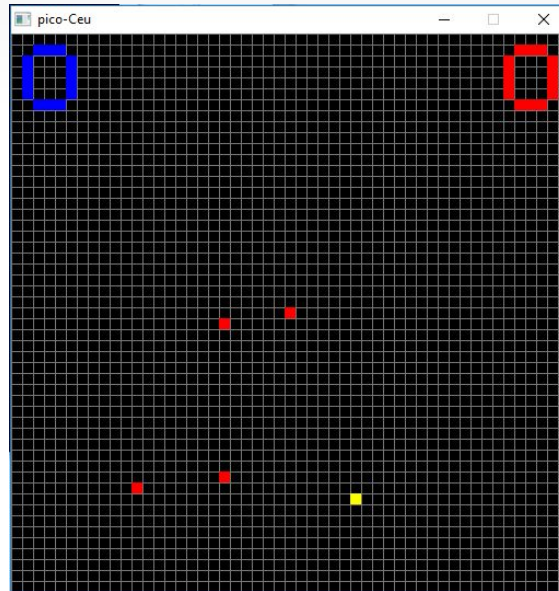
code/await Player(var int up, var int right, var int down, var int left, var
Color color) -> (var Point pt, var int score) -> NEVER do
    score = 0;
    pt = val Point(0,0);
    var int x_axis = 0; var int y_axis = 0;

    par do //every key in KEY_PRESS do
    with // every key in KEY_UNPRESS do
    with // every FRAMES_UPDATE do
    with
        every FRAMES_REDRAW do
            //desenha pixel
            emit GRAPHICS_SET_COLOR_NAME(color);
            emit GRAPHICS_DRAW_PIXEL(pt.x, pt.y);
            //desenha score
            emit GRAPHICS_DRAW_PIXEL(0,0,score);
        end
    end
end
end
end

```

# Exercício

- No exemplo anterior, as duas pontuações estão sendo exibidas na mesma posição ( $x=0$  e  $y=0$ )
- Modifique-o para exibir as pontuações da seguinte forma:



- Pode ser necessário adicionar novo(s) parâmetro(s) ao `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 Point pt, var int score) -> NEVER do
    score = 0;
    pt = val Point(0,0);
    var int x_axis = 0; var int y_axis = 0;

    par do //every key in KEY_PRESS do
    with // every key in KEY_UNPRESS do
    with // every FRAMES_UPDATE do
    with
        every FRAMES_REDRAW do
            //desenha pixel
            emit GRAPHICS_SET_COLOR_NAME(color);
            emit GRAPHICS_DRAW_PIXEL(pt.x, pt.y);
            if (id == 1) then
                emit GRAPHICS_DRAW_INT(-21,21,score);
            else
                emit GRAPHICS_DRAW_INT(23,21,score);
            end
        end
    end
    end
end

```