

Programação com a linguagem Céu

Eventos internos

Anny Caroline

`annycarolinegnr@gmail.com`

Francisco Sant'Anna

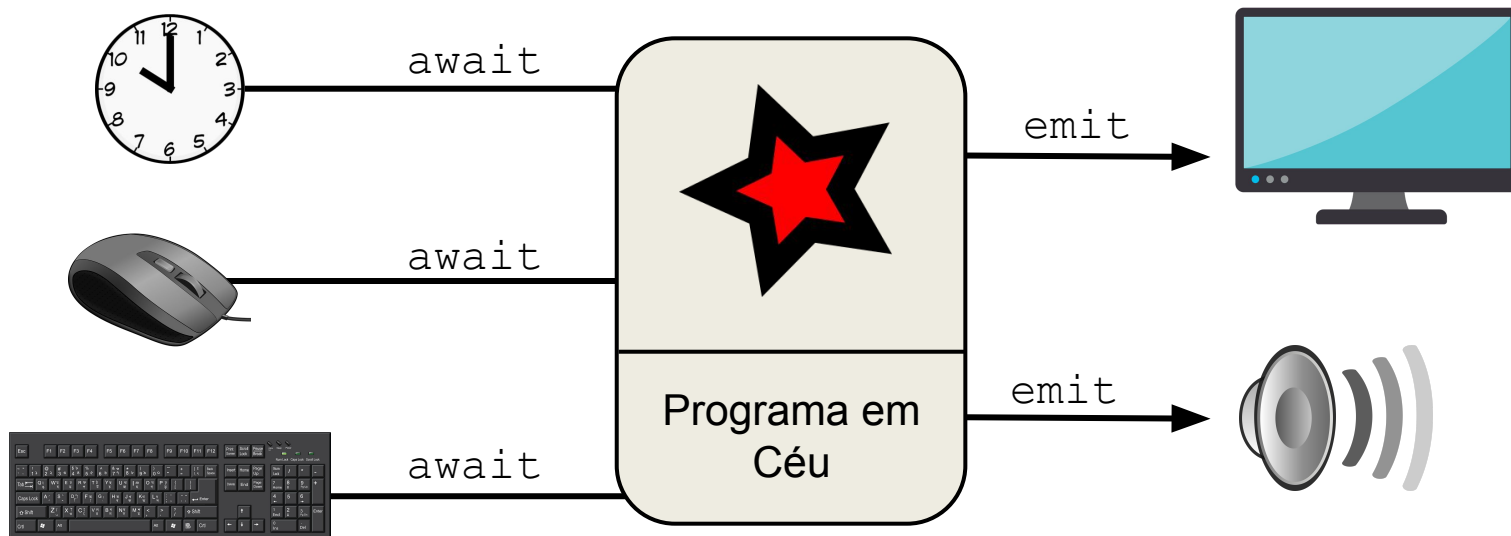
`francisco@ime.uerj.br`

Continuando nosso exemplo

- Discutimos no slide anterior o `code/await` **Player** deve ser responsável por atualizar e desenhar sua pontuação
- Já conseguimos desenhar, mas a atualização ainda está no “programa principal”
- O ideal seria que o programa principal pudesse indicar quando uma colisão ocorresse, e que todos os interessados reagissem a essa colisão
 - para isso usaremos um **evento interno**

Mas o que é um evento interno?

- Até agora usamos somente eventos externos
- Os eventos externos permitem que nosso programa se comunique com o mundo externo, por exemplo, emitindo um evento de desenho ou de áudio, ou aguardando um evento de clique de tecla ou passagem de tempo



Mas o que é um evento interno?

- Já os eventos internos são usados na comunicação interna do nosso programa, permitindo que parte do código se comuniquem com outras

```
event none draw;  
  
par do  
    await draw;  
    emit GRAPHICS_DRAW_PIXEL(0,0);  
with  
    await MOUSE_CLICK;  
    emit draw;  
end
```

Continuando nosso exemplo

- Vamos começar declarando um evento interno logo no começo do programa

```
#include "random.ceu"  
emit FRAMES_SET(yes);
```

```
event none collision;
```

```
//...
```

1-event.ceu

// 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;

emit **collision**;

end

end

end

end

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

```
loop do  
await outer.collission;  
score = score + 1;  
_printf("%d - score %d\n", score);  
end
```

```
end
```

```
end
```

Ainda não está correto

- Como os dois **Players** aguardam o evento **collision**
- Cada vez que ele é emitido, isto é, a cada colisão, os dois **Players** são notificados
- Console, após 3 colisões

```
1 - score 1  
2 - score 1  
1 - score 2  
2 - score 2  
1 - score 3  
2 - score 3
```


Como resolver?

- Eventos podem carregar valores

```
event int evento1;  
event bool evento2;  
event (int, int) evento3;
```

- Vamos fazer o evento **collision** carregar o id do **Player**

```
event int collision;
```

2-event.ceu

Como resolver?

- Enviar o id pelo evento quando houver uma colisão

```
// 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;
        emit collision(player.id);
      end
    end
  end
end
end
```

Como resolver?

- Por último, vamos aguardar o evento no `code/await` **Player** comparando o id recebido com o já presente no **Player**.

```
loop do
```

```
var int r_id = await outer.collision until r_id == id;
```

```
score = score+1;
```

```
_printf("%d - score %d\n", id, score);
```

```
end
```

Exercício

- Implemente o término do jogo
- O jogo termina quando um jogador completa 3 pontos
- Ao terminar, o jogo deve limpar a tela e exibir o jogador vencedor
- Dica:
 - Utilize um evento interno para indicar quando o jogo termina