



STEM<sup>2</sup>

— Go girls! —

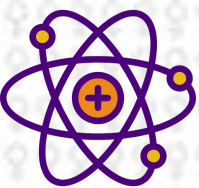


# CODE

*like a*

*girl*





#girlsgonna\_

# Continuando o cenário...



Go girls!

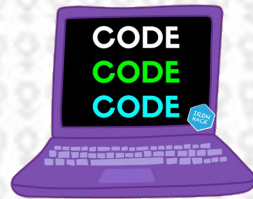


# APARECIMENTO DO CENÁRIO

- Relembrando a última aula...

```
# cria itens do cenario na tela
def criarCenario(self, game):
    # geracao de numero aleatorio
    # game.aparecimentoElementos eh uma constante que vale, inicialmente, 50
    r = random.randrange(0, game.aparecimentoElementos)
    # se o numero gerado for menor que 8
    if r < 8:
        if len(game.obstaculos) == 0:
            # adiciona um obstaculo no jogo, na posicao (LARGURA_DA_TELA, 562) com
            velocidade 10
            game.obstaculos.append(Obstaculo(LARGURA_DA_TELA, 562,
            pygame.image.load(os.path.join('Imagens', 'obstaculo_1_1.png')), 10))
        elif game.obstaculos[-1].x + game.obstaculos[-1].largura + self.tolerancia <
        LARGURA_DA_TELA:
            # adiciona um obstaculo no jogo, na posicao (LARGURA_DA_TELA, 562) com
            velocidade 5
            game.obstaculos.append(Obstaculo(LARGURA_DA_TELA, 562,
            pygame.image.load(os.path.join('Imagens', 'obstaculo_1_1.png')), 5))
```





# EXERCÍCIO

- Implemente o aparecimento do segundo tipo de obstáculo (**obstaculo\_X\_2.png**) no método **criarCenario** da classe **TelaDeJogo**.

**Dica:** Implementação do slide anterior (4)

# EXERCÍCIO

- Implemente o aparecimento de vidas (**vida.png**) no método **criarCenario** da classe **TelaDeJogo**. Lembre-se de deixar este aparecimento mais raro que o de obstáculos e restrinja o aparecimento a uma única vida ao longo da tela.

**Dica:** Implementação no slide 4

## EXERCÍCIO

- Implemente o aparecimento de impulsadores (**impulsador\_X.png**) no método **criarCenario** da classe **TelaDeJogo**. Lembre-se de deixar este aparecimento mais raro que o de obstáculos e restrinja o aparecimento a um único impulsador ao longo da tela.

**Dica:** Implementação no slide 4

# SOLUÇÃO

```
def criarCenario(self, game):
```

```
[...]
```

```
    elif r < 16:
```

```
        if len(game.obstaculos) == 0:
```

```
            game.obstaculos.append(Obstaculo(LARGURA_DA_TELA, 585 - 5*(r%4),  
pygame.image.load(os.path.join('Imagens', 'obstaculo_1_2.png')), 10))
```

```
            elif game.obstaculos[-1].x + game.obstaculos[-1].largura + self.tolerancia <  
LARGURA_DA_TELA:
```

```
                game.obstaculos.append(Obstaculo(LARGURA_DA_TELA, 585 - 5*(r%4),  
pygame.image.load(os.path.join('Imagens', 'obstaculo_1_2.png')), 5 ))
```



## SOLUÇÃO

```
# cria itens do cenario na tela
def criarCenario(self, game):
    if not self.batalha:
        r = random.randrange(0, game.aparecimentoElementos)
        if r < 8:

[...]
```

```
        elif r < 18 and game.vidasExtras < 3 and game.pontuacao > 30 and game.pontuacao%5 ==
0 and len(game.vidas) == 0 and len(game.impulsionadores) == 0:
            game.vidas.append(Vida(LARGURA_DA_TELA, 200 + 5*r,
pygame.image.load(os.path.join('Imagens', 'vida.png')), 5 + game.dvel))

        elif r < 22 and not game.ehInvencivel and game.pontuacao > 30 and game.pontuacao % 5
== 0 and len(game.impulsionadores) == 0 and len(game.vidas) == 0:
            game.impulsionadores.append(Impulsionador(LARGURA_DA_TELA, 150 + 5*r,
pygame.image.load(os.path.join('Imagens', 'impulsionador_1.png')), 5 + game.dvel))
```

# APARECIMENTO DO CENÁRIO

- Para tornar o aparecimento de elementos no jogo ainda mais interessante, podemos **aumentar a velocidade com a qual os objetos são inicializados** ao longo do tempo, bem como **aumentar a taxa de aparecimento de obstáculos**.



# APARECIMENTO DO CENÁRIO



- Para aumentar a taxa de aparecimento de obstáculos, decrementaremos a variável **game.aparecimentoElementos** no método **run**, de tempos em tempos.
- Essa variável é o limite superior do intervalo de números possíveis de serem gerados pela função random que existe dentro do método **criarCenario**.

# APARECIMENTO DO CENÁRIO

- Desta forma, quanto menor for o intervalo, maior serão as probabilidades de serem gerados números que atendem às condições de aparecimento de elementos do cenário.

# APARECIMENTO DO CENÁRIO

- Este código deve ser transcrito para o seu jogo!

```
def run(self, game):  
    self.tempo = 1  
  
    while game.telaAtual == self.name and not game.usuarioSaiu:  
        # aumentar a taxa de aparecimento de elementos do cenario  
        if game.aparecimentoElementos > 25 and self.tempo % 300 == 0:  
            game.aparecimentoElementos -= 1  
  
        if self.tempo % 30 == 0:  
            self.criarCenario(game)  
  
        self.interpretarEventos(game)  
        self.checarColisoas(game)  
        self.atualizar(game)  
        self.desenhar(game)  
  
        self.tempo += 1
```



# APARECIMENTO DO CENÁRIO

- Já para aumentar a velocidade com a qual os objetos são inicializados ao longo do tempo, pode-se utilizar uma variável, por exemplo, **game.dvel** que é somada à velocidade inicial dos objetos criados no jogo. Então, no método **run**, incrementa-se o valor desta variável de tempos em tempos.



# APARECIMENTO DO CENÁRIO

- Adicione o game.dvel no aparecimento dos elementos do jogo:

```
# cria itens do cenario na tela
def criarCenario(self, game):
    # geracao de numero aleatorio
    # game.aparecimentoElementos eh uma constante que vale, inicialmente, 50
    r = random.randrange(0, game.aparecimentoElementos)
    # se o numero gerado for menor que 8
    if r < 8:
        if len(game.obstaculos) == 0:
            # adiciona um obstaculo no jogo, na posicao (LARGURA_DA_TELA, 562) com velocidade 10
            game.obstaculos.append(Obstaculo(LARGURA_DA_TELA, 562,
pygame.image.load(os.path.join('Imagens', 'obstaculo_1_1.png')), 10 + game.dvel))
        elif game.obstaculos[-1].x + game.obstaculos[-1].largura + self.tolerancia <
LARGURA_DA_TELA:
            # adiciona um obstaculo no jogo, na posicao (LARGURA_DA_TELA, 562) com velocidade 5
            game.obstaculos.append(Obstaculo(LARGURA_DA_TELA, 562,
pygame.image.load(os.path.join('Imagens', 'obstaculo_1_1.png')), 5 + game.dvel))
```

# APARECIMENTO DO CENÁRIO

- Este código deve ser transcrito para o seu jogo!

```
def run(self, game):
    self.tempo = 1

    while game.telaAtual == self.name and not game.usuarioSaiu:
        # aumentar a taxa de aparecimento de elementos do cenario
        if game.aparecimentoElementos > 25 and self.tempo % 300 == 0:
            game.aparecimentoElementos -= 1
        # aumentar a velocidade com a qual os elementos do cenario sao inicializados
        if self.tempo % 1200 == 0:
            game.dvel += 1

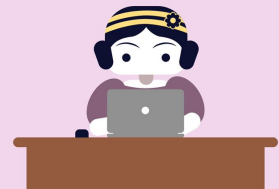
        if self.tempo % 30 == 0:
            self.criarCenario(game)

        self.interpretarEventos(game)
        self.checarColisoas(game)
        self.atualizar(game)
        self.desenhar(game)

    self.tempo += 1
```

# APARECIMENTO DO CENÁRIO

- É importante setar um valor inicial para as variáveis **game.aparecimentoElementos** e **game.dvel** a cada novo jogo, com o intuito de garantir que elas assumirão os valores esperados. Por causa disso, na classe **AdministradorDoJogo** do arquivo **main.py**, faremos as inicializações no método **novoJogo**.

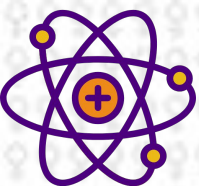


# APARECIMENTO DO CENÁRIO

# esse metodo inicializa as constantes do jogo a cada novo jogo

```
def novoJogo(self):  
    self.pontuacao = 0  
    self.aparecimentoElementos = 50  
    self.vidasExtras = 3  
    self.ehInvencivel = False  
    self.respostaCorreta = 0  
    self.respostaUsuario = 0  
    self.tempoDeInvencibilidade = 15  
    self.dvel = 0  
    self.jogador = Jogador(self)  
    self.obstaculos.clear()  
    self.inimigos.clear()  
    self.vidas.clear()  
    self.impulsionadores.clear()  
    self.tiros.clear()  
    self.tirosInimigo.clear()
```





Scientist Game

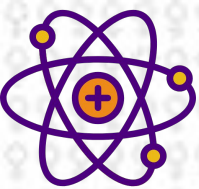


O QUE ESTÁ  
ERRADO?



SCORE:8





#girlsgonna\_

# Colisões



Go girls!





# PARA QUE FAZER COLISÕES?



- ⚛ Detecta a sobreposição entre elementos do jogo
- 🎮 Possibilita a criação de diferentes tomadas de decisão dada a interação de diferentes elementos do jogo



# FUNÇÕES DE COLISÃO

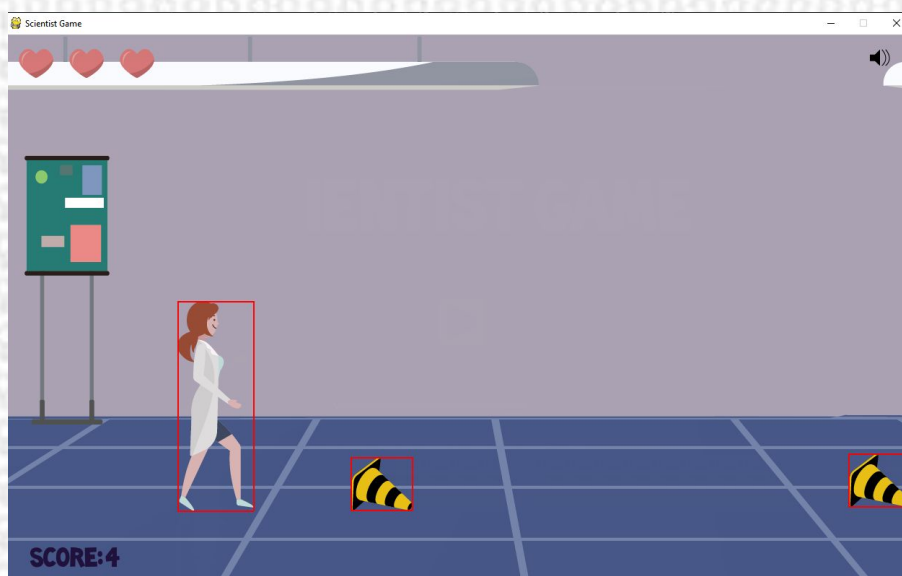


 **rect.colliderect**

.... e o  
obstáculo.

```
if self.rect.colliderect(objeto1):
```

Na imagem mostrada, seria o jogador e assim a função estaria dentro de jogador.py ....





# FUNÇÕES DE COLISÃO



## 🔬 De volta ao nosso código...

Os retângulos envolventes das imagens são definidos no arquivo cenário.py

```
class Cenario:
    def __init__(self, x, y, imagem, vel):
        self.x = x
        self.y = y
        self.image = imagem
        self.largura = imagem.get_width()
        self.altura = imagem.get_height()
        self.rect = pygame.Rect(self.x, self.y, self.largura, self.altura) # retangulo de colisoes
        self.vel = vel # velocidade de atualizacao horizontal na tela
```

Definição da posição e tamanho do retângulo envolvente

```
# descreve como o item do cenario tem sua posicao horizontal atualizada na tela
def atualizacaoBasica(self):
    pass
```

```
# desenha o item do cenario na tela
def desenhar(self, game):
    game.janela.blit(self.image, (self.x, self.y))
    # pygame.draw.rect(game.janela, (255, 0, 0), self.rect, 2)
```

Remova o # para poder ver o retângulo envolvente





# FUNÇÕES DE COLISÃO



## ⚛️ **rect.colliderect**

- Vantagens: rápida execução
- Desvantagens: detecta colisões em momentos em que essa não ocorreu de fato (retângulos são maiores que a imagem em si)

**Solução**



**Combinação  
com  
máscaras:**



image



mask



# FUNÇÕES DE COLISÃO



⚛ ***Sprite.spritecollide / sprite.collide\_mask /  
sprite.spritecollideany***

Imagem

Vetor de  
imagens

Retorna uma lista contendo  
todas as imagens  
do objeto2 que colidiram  
com a imagem do objeto1

```
If self.rect.colliderect(objeto1):  
    colisoes = pygame.sprite.collide(objeto1, objeto2, False)  
    callback = pygame.sprite.collide_mask  
    colisao = pygame.sprite.spritecollideany(objeto1,colisoes,callback)  
    if colisao:
```

Cria as  
máscaras

Verifica se houve colisão usando o método passado,  
que no caso é o collide\_mask, retornando True ou False.



# FUNÇÕES DE COLISÃO



⚛ No arquivo main encontramos as seguintes declarações de elementos do jogo:

```
class AdministradorDoJogo:  
    def __init__(self):  
        # inicializando pygame  
        pygame.init()
```

```
        [...]
```

```
        self.obstaculos = []  
        self.inimigos = []  
        self.vidas = []  
        self.impulsionadores = []  
        self.tiros = []  
        self.tirosInimigo = []
```

```
        [...]
```

```
        self.novoJogo()
```

**Listas de  
elementos  
preenchida com  
a função append  
em criaCenario  
em Cenario.py**

≠

```
def novoJogo(self):  
    self.pontuacao = 0  
    self.aparecimentoElementos = 50  
    self.vidasExtras = 3  
    self.ehInvencivel = False  
    self.respostaCorreta = 0  
    self.respostaUsuario = 0  
    self.tempoDeInvencibilidade = 15  
    self.dvel = 0  
    self.jogador = Jogador(self)  
    self.obstaculos.clear()  
    self.inimigos.clear()  
    self.vidas.clear()  
    self.impulsionadores.clear()  
    self.tiros.clear()  
    self.tirosInimigo.clear()
```

```
[...]
```

**Um único  
elemento**



# FUNÇÕES DE COLISÃO

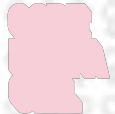


⚛ Desse modo, todos os elementos do jogo, exceto o jogador são na verdade uma lista de objetos de mesma classe.

🎮 Para trabalhar com essas listas utilizamos métodos como:

- pop: remove o elemento mais “antigo” da lista
- index: retorna a posição do elemento desejado
- append: adiciona um elemento na lista





# FUNÇÕES DE COLISÃO



Transcreva o código de colisão entre o jogador e os obstáculos no arquivo `Obstaculos.py`

```
def checarColisoos(self, game):  
    if self.rect.colliderect(game.jogador):  
        colisoos = pygame.sprite.spritecollide(game.jogador, game.obstaculos, False)  
        callback = pygame.sprite.collide_mask  
        colisao = pygame.sprite.spritecollideany(game.jogador, colisoos, callback)  
        if colisao:  
            if game.ehInvencivel:  
                game.obstaculos.pop(game.obstaculos.index(self))  
            else:  
                if game.vidasExtras > 0:  
                    game.obstaculos.pop(game.obstaculos.index(self))  
                    game.vidasExtras = game.vidasExtras - 1  
                else:  
                    game.ultimaTela = 'Tela de Jogo'  
                    game.telaAtual = 'Tela de Fim'
```










**QUAIS COLISÕES  
PRECISAMOS  
IMPLEMENTAR??**



**QUAIS  
CONDIÇÕES  
VAMOS QUERER??**



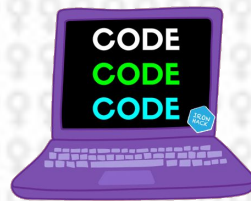
# Colisões e condições

-  **Obstáculo e jogador:** jogador perde uma vida, obstáculo some; se acabar todas as vidas do jogador, irá para a tela de fim de jogo;
-  **Vida e jogador:** vida some e uma vida é aumentada para o jogador;
-  **Impulsionador e jogador:** impulsionador some e o jogador se torna invencível;
-  **Tiros e jogador/inimigo/tiro:** tiro some e uma vida é retirada do jogador ou do inimigo, se a colisão for entre tiros ambos somem.
-  **Inimigo e jogador:** irá para a tela de fim de jogo;

# EXERCÍCIO

- Implemente a colisão do jogador com a vida no arquivo Vida.py usando `rect.colliderect`. As condições são que a vida deve sumir ao entrar em contato com o jogador e a quantidade de vidas extra (`game.vidasExtras`) do jogador deve ser aumentada em 1 unidade.

**Dica:** Implementação da colisão entre jogador e obstáculo (slide 22)



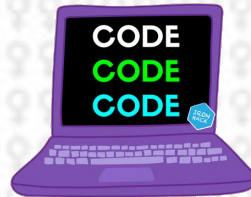


# EXERCÍCIO

- Implemente a colisão do jogador com o impulsor no arquivo `Impulsor.py` usando 'máscara'. As condições são que o impulsor deve sumir ao entrar em contato com o jogador e a variável `invencibilidade` do jogador (`game.ehInvencivel`) deve se tornar verdadeira (`=True`) durante um tempo

**Dica1:** Implementação da colisão entre jogador e obstáculo (slide 22)

**Dica2:** função `pygame.time.wait(-tempo em ms-)` “segura” loop pelo tempo determinado





# SOLUÇÃO

```
def checarColisoes(self, game):  
    # incrementar a variavel inteira game.vidasExtras caso ocorra a colisao do personagem com  
    a vida  
    # fazer a vida desaparecer depois da colisao  
    if self.rect.colliderect(game.jogador):  
        game.vidasExtras = game.vidasExtras + 1  
        game.vidas.pop()
```

# SOLUÇÃO

```
def checarColisoos(self, game):  
    # fazer a variavel game.ehInvencivel ser verdadeira caso ocorra a colisao entre o  
    personagem principal e o impulsionario  
    # fazer o impulsionario desaparecer depois da colisao  
    if self.rect.colliderect(game.jogador):  
        colisoos = pygame.sprite.spritecollide(game.jogador, game.impulsionadores, False)  
        callback = pygame.sprite.collide_mask  
        colisao = pygame.sprite.spritecollideany(game.jogador, colisoos, callback)  
        if colisao:  
            game.impulsionadores.pop(game.impulsionadores.index(self))  
            pygame.time.wait(1500)  
            game.ehInvencivel = True
```



STEM<sup>2</sup>

*Go girls!*

Muito obrigada!