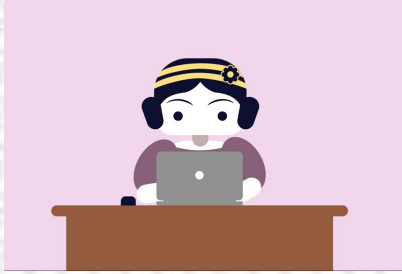
STEM²D

Go girls!


SMS
CODE
like a
girl

INVENCIBILIDADE



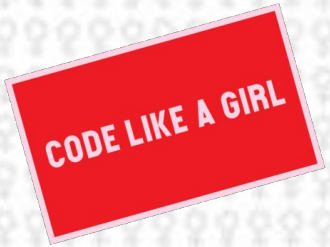
- Após a colisão da jogadora com o impulsionador é interessante que o *status* de invencibilidade adquirido pela jogadora não dure para sempre, mas apenas por um tempo determinado.

- Para garantir isto, implementaremos o método **computarTempoDeInvencibilidade** na classe **TelaDeJogo** (arquivo **TelaDeJogo.py**).

INVENCIBILIDADE

- Este código deve ser transcrito para o seu jogo!

```
def computarTempoDeInvencibilidade(self, game):
    if game.ehInvencivel:
        if game.tempoDeInvencibilidade > 0:
            game.tempoDeInvencibilidade -=1
        else:
            game.ehInvencivel = False
            game.tempoDeInvencibilidade = 15
```



INVENCIBILIDADE

- Para que o tempo de invencibilidade seja computado efetivamente, é preciso chamar o método **computarTempoDeInvencibilidade**, de tempos em tempos, no método **run** da classe **TelaDeJogo**.

- Este código deve ser transcrito para o seu jogo!

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

        while game.telaAtual == self.name and not game.usuarioSaiu:

           ...

            if game.ehInvencivel and self.time % 60 == 0:
                self.computarTempoDeInvencibilidade(game)
            ...
```

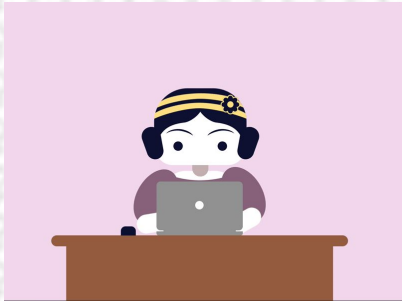
EXERCÍCIO

- Personalize o método **computarTempoDeInvencibilidade** para que ele compute o tempo de invencibilidade que você deseja. **Dicas:**

```
def computarTempoDeInvencibilidade(self, game):
    if game.ehInvencivel:
        if game.tempoDeInvencibilidade > 0:
            game.tempoDeInvencibilidade -=1
        else:
            game.ehInvencivel = False
            game.tempoDeInvencibilidade = 15
```

INVENCIBILIDADE

- Note que o método **computarTempoDeInvencibilidade** e a sua chamada no método **run** apenas contam o tempo de invencibilidade, não mostrando-o na tela. Para viabilizar a visualização da tempo restante de invencibilidade na tela de jogo, implementaremos o método **imprimirTempoDeInvencibilidade**.



INVENCIBILIDADE

- Neste método, carregaremos duas variáveis de texto, uma com a palavra "*INVENCIBILIDADE:*", e outra com o valor do tempo restante, a qual, como já vimos, está armazenada na variável **self.tempoDeInvencibilidade**.

- Posteriormente, basta desenhar estes textos na tela, com o auxílio do método **blit**.

INVENCIBILIDADE

- Este código deve ser transcrito para o seu jogo!

```
def imprimirTempoDeInvencibilidade(self, game):
    if game.ehInvencivel:
        self.invencibilidade = self.fonte1.render("INVENCIBILIDADE: ", True, AZULBB)
        self.invencibilidadeNum = self.fonte1.render(str(game.tempoDeInvencibilidade), True, AZULBB)
        game.janela.blit(self.invencibilidade, (LARGURA_DA_TELA - 335, ALTURA_DA_TELA - 50))
        game.janela.blit(self.invencibilidadeNum, (LARGURA_DA_TELA - 80, ALTURA_DA_TELA - 50))
```

PONTUAÇÃO

- É bastante interessante imprimir a pontuação que o usuário está obtendo ao longo da execução do jogo, para que ele possua um *feedback* da sua performance até então.

- Por causa disso, implementaremos os métodos **computarPontuacao** e **imprimirPontuacao** da **TelaDeJogo** (arquivo **TelaDeJogo.py**).

PONTUAÇÃO

- Para computar a pontuação do usuário, basta que alteremos a variável **game.pontuacao**, incrementando-a ao longo do tempo.

- Este código deve ser transcrito no seu jogo!

```
def computarPontuacao(self, game):
        game.pontuacao += 1
```

PONTUAÇÃO

- Para que a pontuação seja computada efetivamente, é preciso chamar o método **computarPontuacao**, de tempos em tempos, no método **run** da classe **TelaDeJogo**, conforme fizemos com o método **computarTempoDeInvencibilidade**.

EXERCÍCIO

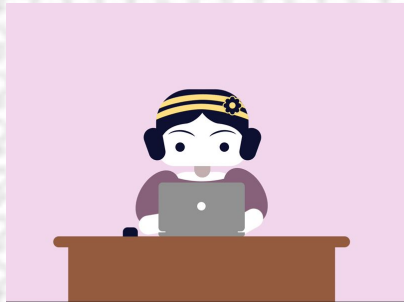
- Personalize o método **computarPontuacao** e a chamada do método **computarPontuacao** dentro do método **run**, de forma que o incremento da pontuação do seu jogo e a taxa com que a pontuação é incrementada sejam realizadas da forma que você deseja. **Dicas:**

```
def computarPontuacao(self, game):
        game.pontuacao += 1
```

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

        while game.telaAtual == self.name and not game.usuarioSaiu:

           ...

            if game.ehInvencivel and self.time % 60 == 0:
                self.computarTempoDeInvencibilidade(game)
            ...
```

PONTUAÇÃO

- Note que o método **computarPontuacao** e a sua chamada no método **run** apenas incrementam a pontuação, não mostrando-a na tela. Para viabilizar a visualização da pontuação na tela de jogo, implementaremos o método **imprimirPontuacao**.



PONTUAÇÃO

- Neste método, carregaremos duas variáveis de texto, uma com a palavra "*SCORE*" (pontuação, em inglês), e outra com o valor da pontuação, a qual, como já vimos, está armazenada na variável **game.pontuacao**.

- Posteriormente, basta desenhar estes textos na tela, com o auxílio do método **blit**.

EXERCÍCIO

- Implemente o método **imprimirPontuacao**. **Dicas:**

```
def imprimirTempoDeInvencibilidade(self, game):
    if game.ehInvencivel:
        self.invencibilidade = self.fonte1.render("INVENCIBILIDADE: ", True, AZULBB)
        self.invencibilidadeNum = self.fonte1.render(str(game.tempoDeInvencibilidade), True, AZULBB)
        game.janela.blit(self.invencibilidade, (LARGURA_DA_TELA - 335, ALTURA_DA_TELA - 50))
        game.janela.blit(self.invencibilidadeNum, (LARGURA_DA_TELA - 80, ALTURA_DA_TELA - 50))
```

#girlsgonna_

Movimentos da jogadora: Pulo

Go girls!

A CLASSE JOGADOR

- Acessem o arquivo Jogador.py:

```
class Jogador():
```

A CLASSE JOGADOR

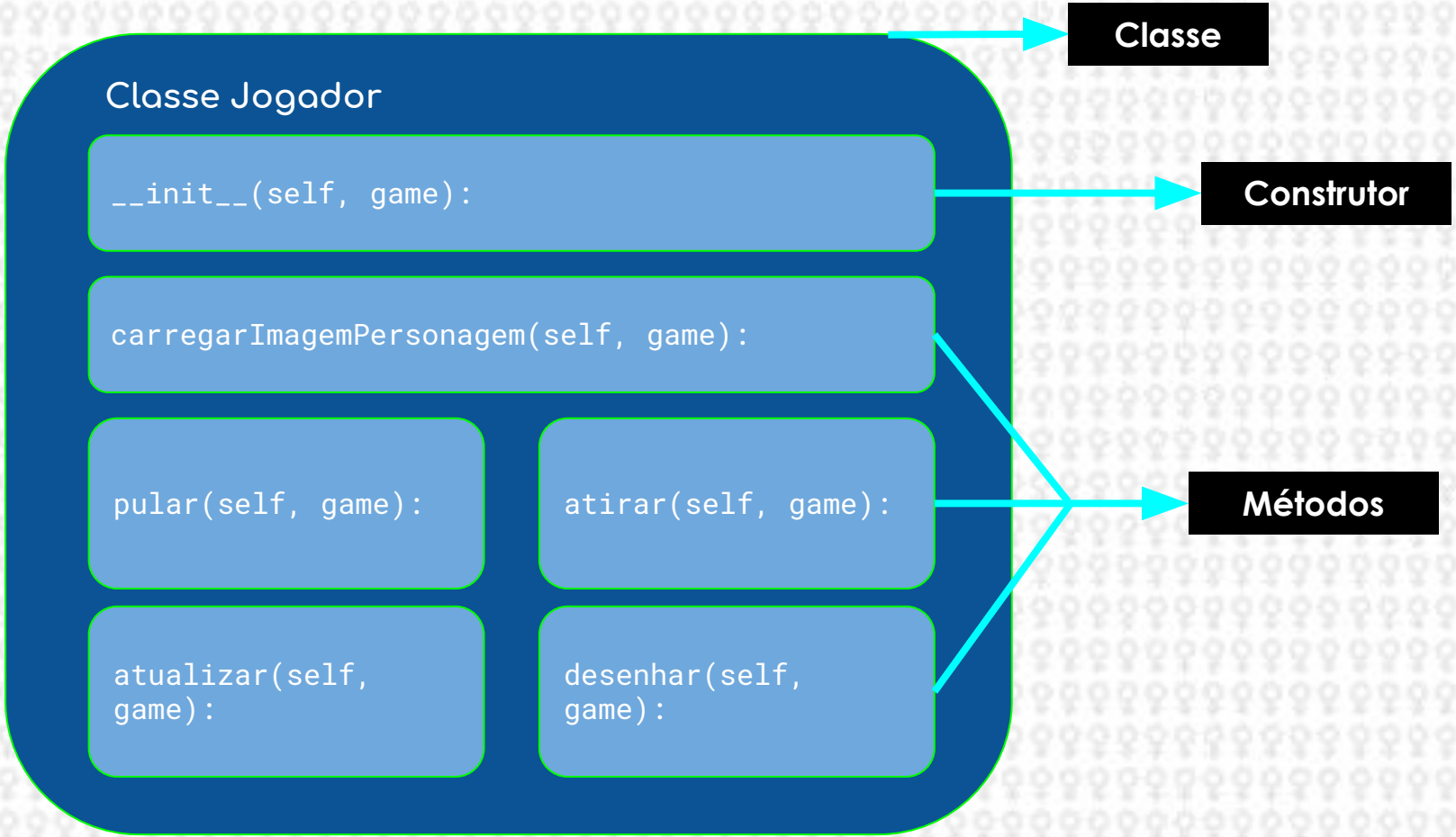
- Acessem o arquivo Jogador.py:

```
class Jogador():
    def __init__(self, game):
        self.x = X_CHAO
        self.y = Y_CHAO
        self.carregarImagemPersonagem(game)
        self.largura = self.image.get_width()
        self.altura = self.image.get_height()
        self.rect = pygame.Rect(self.x, self.y, self.largura, self.altura)#retangulo
de colisoes
        self.rect.center = (self.x + self.largura/2, self.y + self.altura/2)
        self.pos = vec(self.x + self.largura/2, self.y + self.altura/2)
        self.vel = vec(0.0, 0.0)
        self.acc = vec(0.0, 0.0)
```

OS MÉTODOS DA CLASSE JOGADOR

```
# carrega a imagem do personagem de acordo com a escolha do usuario
def carregarImagemPersonagem(self, game):
  [...]

# esse metodo atualiza as posicoes do jogador para que ele pule
def pular(self, game):
   pass

# esse metodo faz carregar o tiro do jogador na tela
def atirar(self, game):
   pass

# esse metodo atualiza as posicoes do jogador
def atualizar(self, game):
   pass

# desenha o jogador na tela com a imagem correspondente ao seu estado atual
def desenhar(self, game, tela):
   pass
   game.janela.blit(self.imagem, (self.rect.left, self.rect.top))
```

A CLASSE JOGADOR



CRIANDO O OBJETO JOGADOR

- Acessem o arquivo **main.py (linha 72):**

```
# esse metodo inicializa as constantes do jogo a cada novo jogo
def novoJogo(self):
[...]
    self.vidasExtras = 3
    self.ehInvencivel = False
[...]
    self.dvel = 0
    self.jogador = Jogador(self)
    self.obstaculos.clear()
    self.inimigos.clear()
    self.vidas.clear()
[...]
    self.tiros.clear()
    self.tirosInimigo.clear()
```

Cria o objeto um **objeto** chamado **jogador** a partir da **classe Jogador()**

Esse objeto está dentro do objeto **game**:
game.jogador

Identificando Comandos

- Acessem o arquivo **TelaDeJogo.py:**

```
def checarComportamentoJogador(self, game, evento):
   # verificar se o usuario pediu para o jogador fazer algum comando (atirar ou
pular)
   if evento != [] and evento.type == pygame.KEYDOWN: #verificar se há algo na fila
de eventos e se há teclas precionadas
       if evento.key == pygame.K_UP:
           game.jogador.pular(game)
       elif evento.key == pygame.K_SPACE:
           game.jogador.atirar(game)
```

Seta para cima: chama **pular** do **jogador**

Barra de espaço: chama **atirar** do **jogador**

LET'S CODE TOGETHER!

- Implemente o pulo do jogador no método **pular** da classe **Jogador**.

{LET'S CODE TOGETHER}

Ideias para o pulo?

Ideias para o pulo?

- Só pode mudar a posição em x e y.
- Lembre-se da gravidade.
- **Movimento Uniformemente Variado!**

$$v_{atual} = v_{anterior} + a.\Delta t_{intervalo\ de\ tempo}$$

$$s_{atual} = s_{anterior} + v_{atual}.\Delta t_{intervalo\ de\ tempo}$$

MÉTODO PULAR

```
def pular(self, game):
    #verificar se a jogadora está no chão (impossível pular sem estar no chão!)
    if self.pos.y == Y_CHAO:
        #atualizar a velocidade em y
        self.vel.y = VELOC_INICIAL_PULO
        #atualiza a aceleração em y (efeito da gravidade)
        self.acc.y = ACE_GRAV
```

MÉTODO ATUALIZAR

```
def pular(self, game):
    #verificar se a jogadora está no chão (impossível pular sem estar no chão!)
    if self.pos.y == Y_CHAO:
        #atualizar a velocidade em y
        self.vel.y = VELOC_INICIAL_PULO
        #atualiza a aceleração em y (efeito da gravidade)
        self.acc.y = ACE_GRAV
[...]
# esse metodo atualiza as posicoes do jogador
def atualizar(self, game):
    # Equações de Movimento
    dt = 1

    # Atualizar velocidade e posição do jogador
    self.vel.y = self.vel.y + self.acc.y * dt # Atualiza a velocidade
    self.pos += self.vel * dt # Atualiza a posição
    self.rect.midbottom = self.pos # Atualiza o retângulo de colisão
```

Rodem o jogo

O que aconteceu?

O que aconteceu?

Verificar quando a **gravidade deve “parar”**. Ou seja, quando **chegamos ao chão**.

EXERCÍCIO

- Implemente a verificação de se a personagem chegou ao chão e as correções da posição e da velocidade quando ela atingir o chão no método **atualizar()** da classe **Jogador()**.

MÉTODO ATUALIZAR

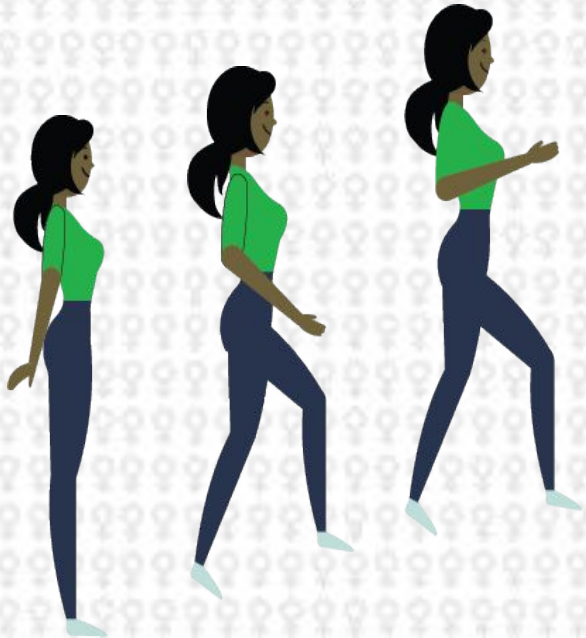
```python
# esse metodo atualiza as posicoes do jogador
def atualizar(self, game):
    # Equações de Movimento
    dt = 1

    # Atualizar velocidade e posição do jogador
    self.vel.y += self.acc.y * dt # Atualiza a velocidade
    self.pos += self.vel * dt # Atualiza a posição
    self.rect.midbottom = self.pos # Atualiza o retângulo de colisão

    #Verificar se a personagem chegou ao chão
    if [...]
        #corrigir a posição para parar exatamente no chão

        #Zerar a velocidade para a personagem parar de cair
```

MÉTODO ATUALIZAR: RESPOSTA

```python
# esse metodo atualiza as posicoes do jogador
def atualizar(self, game):
    # Equações de Movimento
    dt = 1

    # Atualizar velocidade e posição do jogador
    self.vel.y += self.acc.y * dt # Atualiza a velocidade
    self.pos += self.vel * dt # Atualiza a posição
    self.rect.midbottom = self.pos # Atualiza o retângulo de colisão

    #Verificar quando chegamos ao chão
    if self.pos.y >= (Y_CHAO):
        #corrigir a posição para parar exatamente no chão
        self.pos.y = (Y_CHAO)
        #Zerar a velocidade para a personagem parar de cair
        self.vel.y = 0
```

TORNAR O MOVIMENTO MAIS NATURAL

- Para tornar o pulo mais natural, vamos usar as imagens **personagem_principal_DIR**, **personagem_principal_ESQ** e **personagem_principal_FEC**, gerando um efeito gradativo.



EXERCÍCIO

- Implemente a transição de imagens no método **desenhar()** da classe **Jogador()**, para tornar o pulo mais natural.

MÉTODO DESENHAR

```
# vetor com as imagens
if game.ehInvencivel == False:
    self.imagens = [self.imagemF, self.imagemD, self.imagemE]
[...]
# desenha o jogador na tela com a imagem correspondente ao seu estado atual
def desenhar(self, game, tela):
    # gerar efeito gradual no pulo
    #Se a posição estiver muito baixa, manter a imagem fechada
   if self.pos.y == (Y_CHAO):
       self.imagem = self.imagens[0]
   #Para uma posição intermediária, usar a posição intermediária

   #Para os demais casos, usar a imagem final (braços abertos)
```

MÉTODO DESENHAR: Resposta

```
# vetor com as imagens
if game.ehInvencivel == False:
    self.imagens = [self.imagemF, self.imagemD, self.imagemE]
[...]
# desenha o jogador na tela com a imagem correspondente ao seu estado atual
def desenhar(self, game, tela):
    # gerar efeito gradual no pulo
    #Se a posição estiver muito baixa, manter a imagem fechada
   if self.pos.y == (Y_CHAO):
       self.imagem = self.imagens[0]
   #Para uma posição intermediária, usar a posição intermediária
   elif self.pos.y < (Y_CHAO) and self.pos.y > (0.95 * Y_CHAO):
       self.imagem = self.imagens[1]
   #Para os demais casos, usar a imagem final (braços abertos)
   else:
       self.imagem = self.imagens[2]
```

HORA DA CARTEAÇÃO

- Para permitir que nossa personagem pule todos os obstáculos, precisamos atualizar sua velocidade e aceleração no método **pular()** conforme o aumento da velocidade dos objetos e inimigos.
- Não explicarei a lógica por de trás das relações, mas **o cálculo completo está na apostila**.
- Desses cálculos chegamos que a **aceleração** deve aumentar **proporcionalmente ao quadrado** da **velocidade do objeto** e a velocidade deve **aumentar proporcionalmente**.

MÉTODO PULAR - EXPLICAÇÃO

```
# esse metodo atualiza as posicoes do jogador para que ele pule
def pular(self, game):
[...]
            #Atualizar a aceleração para uma constante vezes o quadrado da
velocidade do obstaculo
            self.acc.y = 0.014 * game.obstaculos[0].vel * game.obstaculos[0].vel
            # Atualizar a velocidade para uma constante vezes a velocidade do
obstaculo
            self.vel.y = -2.5 * game.obstaculos[0].vel
        if len(game.inimigos):
            # Atualizar a aceleração para uma constante vezes o quadrado da
velocidade do inimigo
            self.acc.y = 0.014 * game.inimigos[0].ve
            # Atualizar a velocidade para uma consta
inimigo
            self.vel.y = -2.5 * game.inimigos[0].vel
```

Altere as constantes multiplicativas (0.014 e 2.5), para ajustar o pulo.


STEM2D
Go girls!
Muito obrigada!