



UNIVERSIDADE FEDERAL DO CARIRI
CENTRO DE CIÊNCIAS E TECNOLOGIA
CIÊNCIA DA COMPUTAÇÃO

CÍCERO IGOR ALVES TORQUATO DOS SANTOS

LABORATÓRIO DE PROGRAMAÇÃO

Juazeiro do Norte - Ceará

2022

ORGANIZAÇÃO:

- MUDANÇAS ESTRUTURAIS
- TESTES
- GITHUB

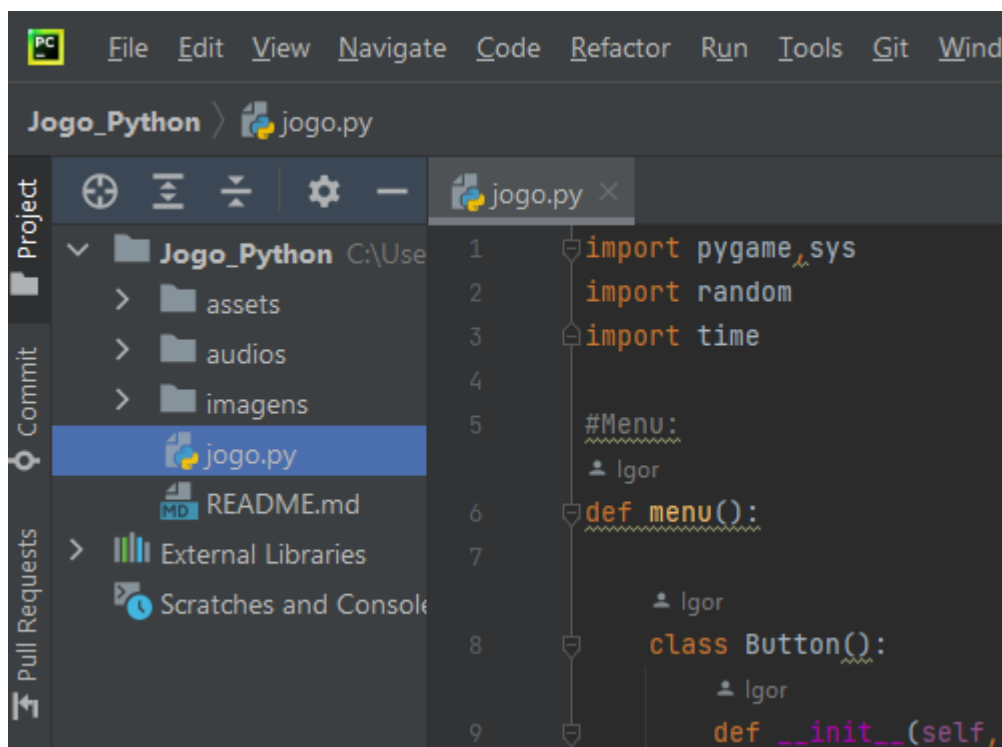
MUDANÇAS ESTRUTURAIS:

INTRODUÇÃO

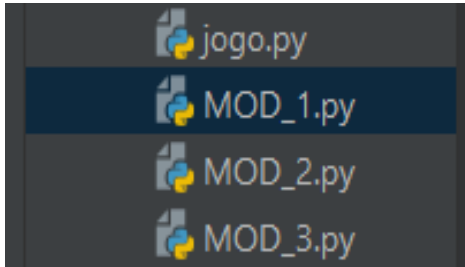
- Na primeira parte do trabalho foi desenvolvido um jogo na linguagem de programação Python e utilizando a biblioteca Pygame. Nesse sentido, agora nosso foco será em melhorar o código desse jogo.
- Executaremos alguns passos que serão importantes para aprimorar cada vez mais o código fonte.

PRIMEIRA ALTERAÇÃO

- Tendo em vista todos os conhecimentos adquiridos nas aulas sobre “Modularização” e “Como tornar seu código mais modular?”, podemos aplicar a primeira alteração no nosso projeto.
- Lembrando que o jogo foi desenvolvido em um só arquivo:



- No código, foram utilizadas classes para os funcionamentos dos botões do jogo, da animação e da criação e movimentação do jogador durante a partida.
- Portanto, nossa primeira alteração consistirá em modularizar o nosso projeto. Dessa forma, reduziremos os números de linhas de código do arquivo principal “jogo.py” e deixaremos o projeto muito mais organizado.
- Módulos:



- O MOD_1 lidará apenas com a classe dos botões.

```

1  #Botões
2
3  class Button():
4      def __init__(self, image, pos, text_input, font, base_color, hovering_color):
5          self.image = image
6          self.x_pos = pos[0]
7          self.y_pos = pos[1]
8          self.font = font
9          self.base_color, self.hovering_color = base_color, hovering_color
10         self.text_input = text_input
11         self.text = self.font.render(self.text_input, True, self.base_color)
12         if self.image is None:
13             self.image = self.text
14         self.rect = self.image.get_rect(center=(self.x_pos, self.y_pos))
15         self.text_rect = self.text.get_rect(center=(self.x_pos, self.y_pos))
16
17     def update(self, screen):
18         if self.image is not None:
19             screen.blit(self.image, self.rect)
20         screen.blit(self.text, self.text_rect)
21
22     def checkForInput(self, position):
23         if position[0] in range(self.rect.left, self.rect.right) and position[1] in range(self.rect.top,
24                                                                                             self.rect.bottom):
25             return True
26         return False
27

```

- O MOD_2 lidará com a classe das superfícies (Retângulos) que o jogador deve desviar no jogo.

```
jogo.py × MOD_1.py × MOD_2.py × MOD_3.py ×
1  #Retângulos
2  import random
3  import pygame
4
5  class Recs(object):
6      def __init__(self, numeroinicial):
7          self.lista = []
8          for x in range(numeroinicial):
9              leftrandom = random.randrange(2, 1100)
10             toprandom = random.randrange(-1124, -10)
11             width = random.randrange(10, 30)
12             height = random.randrange(15, 30)
13             self.lista.append(pygame.Rect(leftrandom, toprandom, width, height))
14
15     def mover(self):
16         for retangulo in self.lista:
17             retangulo.move_ip(0, 2)
18
19     def cor(self, superficie):
20         for retangulo in self.lista:
21             pygame.draw.rect(superficie, (165, 214, 254), retangulo)
22
23     def recriar(self):
24         for x in range(len(self.lista)):
25             if self.lista[x].top > 481:
26                 leftrandom = random.randrange(2, 1100)
```

- O MOD_3 lidará com a classe do player (Jogador) e será importante tanto no funcionamento do jogo quanto na animação na tela de menu.

```
o.py x MOD_1.py x MOD_2.py x MOD_3.py x
#Jogador/Player
import pygame

class Player(pygame.sprite.Sprite):

    def __init__(self, imagem):
        self.imagem = imagem
        self.rect = self.imagem.get_rect()
        self.rect.top, self.rect.left = (315, 500)

    def mover(self, vx, vy):
        self.rect.move_ip(vx, vy)

    def update(self, superficie):
        superficie.blit(self.imagem, self.rect)
```

- Finalmente, usaremos todos os módulos no arquivo principal, para isso importaremos cada um deles. Lembrando que, para utilizar o módulo adequadamente devemos digitar seu nome antes de chamar uma função ou classe presente nele.

```
#Módulos Python:
import pygame, sys
import time

#Módulos Pessoal:
import MOD_1
import MOD_2
import MOD_3

OPTIONS_BACK = MOD_1.Button
```

SEGUNDA ALTERAÇÃO

- Além da primeira alteração, o estado final do código do projeto contava com uma série de “números mágicos”, que em programação é o nome dado aos números que aparecem no código sem explicação. Assim, além desses números, outros eventos de cores, textos e carregamento de arquivos estavam no código e isso gera um problema.
- Posteriormente, para fazer alterações nesse código, ou simplesmente para entendê-lo, seria uma tarefa muito complicada se o programador não soubesse o

que cada elemento significa. Existe uma maneira de resolver esse problema sem precisar comentar em cada linha e deixar o código mais extenso, que é com a criação de “constantes”.

- Em Python a regra de nomeação das constantes segue um padrão parecido com as de variáveis, com a diferença de que todas as letras são maiúsculas e separadas por underline “_”.
- Porém, o Python possui tipagem dinâmica e fraca, o que permite que uma variável possa armazenar dados de diferentes tipos em fases diferentes do script. Assim uma constante, em Python, não é bem uma constante, porque pode ser alterada. Mas para resolver o problema no nosso código esse recurso será importante e útil.
- Exemplos das constantes numéricas:

```
#Constantes Numéricas:
TELA_MENU_LARGURA = 1000
TELA_MENU_ALTURA = 630
TELA_JOGO_LARGURA = 1024
TELA_JOGO_ALTURA = 500
FONTE_TITULO = 30
FONTE_TEXTO = 10
FONTE_MAIN_MENU = 100
FONTE_BOTOES = 75
FONTE_BACK = 50
FONTE_SCORE = 15
VELOCIDADE_INICIAL = 0
VELOCIDADE_FINAL_MENU = 20
VELOCIDADE_JOGADOR = 2
CENTRALIZACAO_X = 0
CENTRALIZACAO_Y = 0
INCREMENTO_VELOCIDADE_MENU = 1
INCREMENTO_DIFICULDADE = 10
```

- Exemplos das constantes de carregamento:

```
#Constantes Load:
BACKGROUND = pygame.image.load("imagens/fundo - Copia.png")
IMAGEM_NAVE = pygame.image.load("imagens/nave.png")
EXPLOSAO = pygame.image.load("imagens/explosao.png")
MUSICA_MENU = "audios/Star Wars The Force Awakens _Force Theme_"
MUSICA_JOGO = "audios/musica.ogg"
SOM_EXPLOSAO = "audios/explosao2.ogg"
SOM_MOVIMENTACAO = "audios/som2.ogg"
FONTE_PADRAO = "assets/font.ttf"
FONTE_ARIAL = "Arial"
RETANGULO_PLAY = "assets/Play Rect.png"
RETANGULO_OPTIONS = "assets/Options Rect.png"
RETANGULO_QUIT = "assets/Quit Rect.png"
```

- Exemplos das constantes de cores:

```
#Cores:
BRANCO = "White"
PRETO = "Black"
VERDE = "Green"
AMARELO = "#b68f40"
VERDE_CLARO = "#d7fcd4"
LARANJA_RGB = (255, 140, 0)
```

- Diferença do código antes das mudanças e depois das mudanças:

1- Antes:

```
pygame.init()

WIDTH = 1000
HEIGHT = 630
SCREEN = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Escape from Earth")

B6 = pygame.image.load("imagens/fundo - Copia.png")

img_photo = pygame.image.load("imagens/nave.png").convert_alpha()

pygame.mixer.music.load("audios/Star Wars The Force Awakens _Force Theme_ [8 Bit Version North Pole Twin].ogg")
pygame.mixer.music.play(50)

Igor
def get_font(size):
    return pygame.font.Font("assets/font.ttf", size)
```

2- Depois:

```

pygame.init()

SCREEN = pygame.display.set_mode((TELA_MENU_LARGURA, TELA_MENU_ALTURA))
pygame.display.set_caption(NOME_JOGO)

pygame.mixer.music.load(MUSICA_MENU)
pygame.mixer.music.play(LOOP_MUSICA)

def get_font(size):
    return pygame.font.Font(FONTE_PADRAO, size)

```

- Enfim, é perceptível que houve uma redução significativa no código após as alterações e para fazer uma mudança em vários elementos o programador só precisa alterar uma variável que está no início do arquivo principal “jogo.py”. Além disso, com a nomeação de cada constante, o entendimento do código melhorou bastante e o código ficou mais simples para possíveis mudanças posteriores.

TESTES:

CAIXA PRETA (5) :

- Iniciando a testagem do nosso código, o *pytest* é um recurso que nos ajudará a fazer os testes iniciais no nosso projeto.
- No projeto, criamos um arquivo chamado “Testes.py”. É nele que criaremos funções que irão checar alguns pontos cruciais para o funcionamento do jogo.
- Lembrando que, para o jogo funcionar corretamente são necessárias uma série de elementos definidas de forma coesa e correta. Então os testes serão utilizados para verificar a integridade de pontos específicos do código que precisam, além de estarem certos, seguir um certo padrão.
- Para os cinco primeiros testes decidi explorar esses pontos importantes: A tela, a velocidade da nave, a música, os botões e as superfícies (Retângulos).
- Todos esses cinco elementos não podem ter nenhum defeito para que o jogo entregue algo mínimo ao usuário. As dimensões da tela devem ser diferentes de zero; A velocidade inicial da nave deve ser nula inicialmente para que o usuário possa alterá-la e para que a animação funcione; A música deve ser reproduzida; Os botões devem estar em pleno funcionamento para que o usuário possa clicar sobre ele.

- Arquivo:

```
import jogo
import MOD_1
import MOD_2

def test_tela():
    assert jogo.TELA_MENU_ALTURA != 0
    assert jogo.TELA_MENU_LARGURA != 0
    assert jogo.TELA_JOGO_ALTURA != 0
    assert jogo.TELA_JOGO_LARGURA != 0

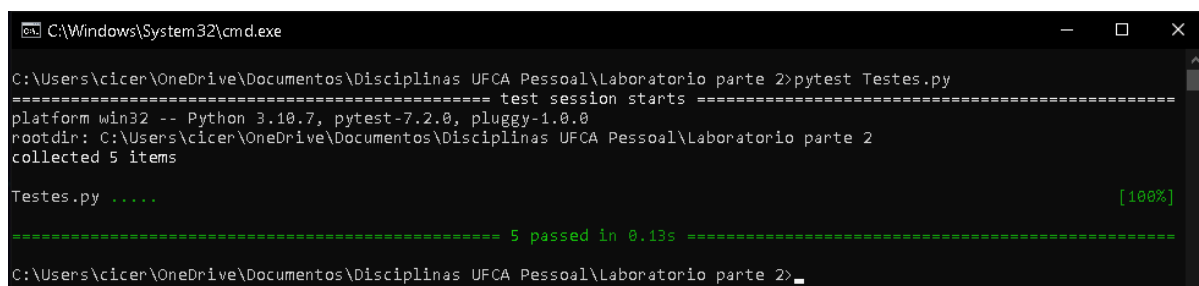
def test_velocidade_inicial():
    assert jogo.VELOCIDADE_INICIAL == 0

def test_musica():
    assert jogo.MUSICA_JOGO == "audios/musica.ogg"

def test_button():
    assert MOD_1.Button != None

def test_recs():
    assert MOD_2.WIDTH != 0
    assert MOD_2.HEIGHT != 0
```

- Executando os testes:



```
C:\Windows\System32\cmd.exe

C:\Users\cicer\OneDrive\Documentos\Disciplinas UFCA Pessoal\Laboratorio parte 2>pytest Testes.py
===== test session starts =====
platform win32 -- Python 3.10.7, pytest-7.2.0, pluggy-1.0.0
rootdir: C:\Users\cicer\OneDrive\Documentos\Disciplinas UFCA Pessoal\Laboratorio parte 2
collected 5 items

Testes.py ..... [100%]

===== 5 passed in 0.13s =====

C:\Users\cicer\OneDrive\Documentos\Disciplinas UFCA Pessoal\Laboratorio parte 2>_
```

CAIXA BRANCA (8):

- O teste da caixa branca, se preocupa com a estrutura interna do software, assim precisaremos analisar o código fonte. Com o código, é importante fazer a testagem de todos os caminhos possíveis que o programa pode rodar. Tendo em vista essas

observações, dividi o código em alguns blocos para que os testes possam ser realizados.

- Na função *main()* do jogo , podemos realizar uma divisão em 9 blocos:
 1. Inicialização do *pygame* e variáveis.
 2. Condicional *while sair != True*.
 3. Evento do *pygame* usando o laço de repetição *for*.
 4. Condicional *pygame.event.get()* que checa se o usuário fechou o programa.
 5. Condicional *colidiu == False* que checa os eventos de teclado.
 6. Condicional *colisão* que checa se houve colisão.
 7. Condicional *colidiu == False* que realiza a movimentação do jogador, dos retângulos e apresenta a pontuação.
 8. Comandos que realizam a recriação dos retângulos, aumento da dificuldade do jogo, apresentação do jogador e atualização de frames do display.
 9. Evento de saída *pygame.quit()*.
 10. Evento do *pygame* usando o laço de repetição *for*.

```
if colidiu == False:
    ret.mover()
    jogador.mover(vx, vy)

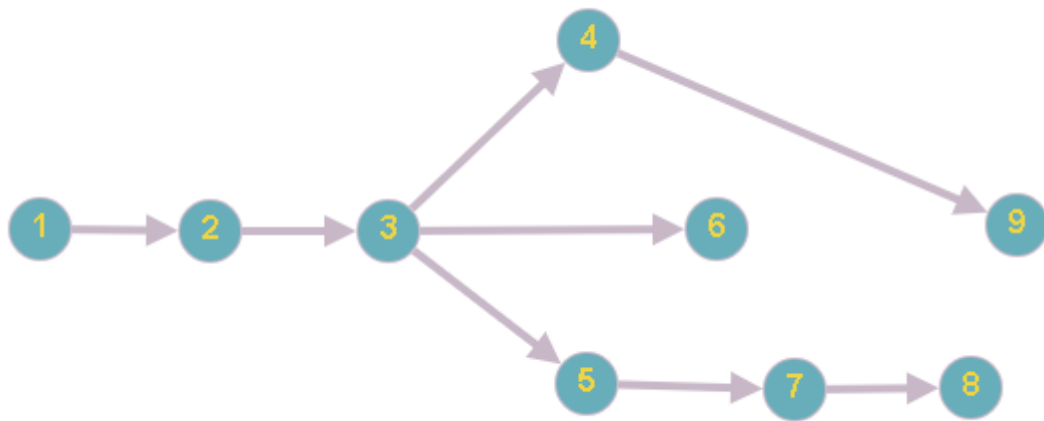
    tela.blit(imagem_fundo,(CENTRALIZACAO_X,CENTRALIZACAO_Y))
    segundos = (pygame.time.get_ticks()/CORRECAO_SEGUNDOS) - tempo_menu/CORRECAO_SEGUNDOS
    segundos = round(segundos,ARREDONDAMENTO)
    segundos = str(segundos)
    contador = texto.render("Pontuação:{}".format(segundos), 0, LARANJA_RGB)
    tela.blit(contador, (SCORE_X, SCORE_Y))

relogio.tick(dificuldade)
ret.cor(tela)
ret.recriar()
jogador.update(tela)
pygame.display.update()

pygame.quit()
```

Exemplo dos Blocos (6,7,8)

- Com os blocos devidamente enumerados, podemos montar um grafo que representará o fluxo que o algoritmo segue. Dessa forma, saberemos a quantidade de testes que será preciso ser executado para que o jogo funcione corretamente. Utilizarei o método de cobertura de instruções para saber se todas elas foram executadas.



- Testes:

Análise	Nós Percorridos
sair == True (No bloco 3)	{1,2,3,4,9}
sair != True e colidiu == False (Condição normal de jogo)	{1,2,3,6}
sair != True e colidiu == True (Condição de derrota)	{1,2,3,5,7,8}

- Podemos concluir que todos os fluxos de caminho de funcionamento do jogo estão corretos e funcionando à medida que o jogador desvia dos obstáculos ou opta por clicar em *quit*.
- Agora vamos fazer o mesmo procedimento para o menu. Na função *main_menu()* vamos dividir o código em blocos e enumerá-los para posteriormente criarmos um grafo com os caminhos possíveis e testar cada um deles.
 1. Definição das variáveis que serão utilizadas na animação.
 2. Condicional *while True*.
 3. Incremento das variáveis de velocidade da animação e apresentação do background.
 4. Condicional *YVEL == VELOCIDADE_FINAL_MENU* que limita a velocidade das naves da animação.
 5. Definição das variáveis dos botões e apresentação da animação.
 6. Evento de mudança de cor do botão ao ser sobreposto pelo mouse com laço *for*.
 7. Evento do *pygame* com laço *for*.
 8. Condicional de *quit* ao clicar no “x” do canto superior da janela.
 9. Condicional de mouse do *pygame* que checa se o usuário clicou em algum botão.

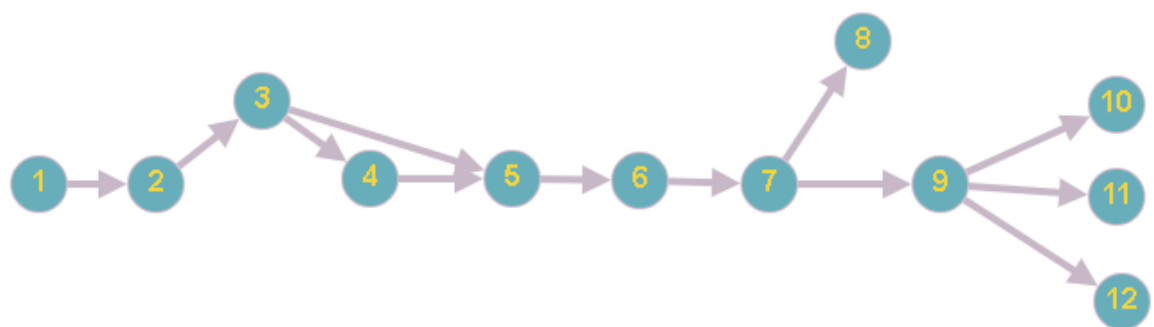
10. Condicional que checa se o usuário clicou no botão que executa *play()*.
11. Condicional que checa se o usuário clicou no botão que executa *options()*.
12. Condicional que checa se o usuário clicou no botão que executa *quit()*.

```

for event in pygame.event.get(): 7
    if event.type == pygame.QUIT:
        pygame.quit()
        sys.exit() 8
    if event.type == pygame.MOUSEBUTTONDOWN: 9
        if PLAY_BUTTON.checkForInput(MENU_MOUSE_POS): 10
            play()
        if OPTIONS_BUTTON.checkForInput(MENU_MOUSE_POS): 11
            options()
        if QUIT_BUTTON.checkForInput(MENU_MOUSE_POS):
            pygame.quit()
            sys.exit() 12

```

Exemplo dos Blocos (7,8,9,10,11,12)



- Testes:

Análise	Nós Percorridos
event.type == pygame.QUIT (Quando for ativo esse evento no bloco 8)	{1,2,3,4,5,6,7,8}
PLAY_BUTTON.checkForInput(MENU_MOUSE_POS) (Ao clicar em jogar)	{1,2,3,4,5,6,7,9,10}
OPTIONS_BUTTON.checkForInput(MENU_MOUSE_POS) (Ao clicar em manual)	{1,2,3,4,5,6,7,9,11}

QUIT_BUTTON.checkForInput(MENU_MOUSE_POS) (Ao clicar em sair)	{1,2,3,4,5,6,7,9,12}
if YVEL != 20 and XVEL != 20 (Teste de clicar rápido nos botões sem esperar a animação concluir)	{1,2,3,5,6,7,8}

- Pode-se concluir que o nó 4 aparece em todos os caminhos mesmo que seu nó anterior, 3, possa ser direcionado ao vértice 5 e isso faz sentido devido a limitação da posição imposta às naves da animação. Nesse sentido, já que elas se movimentam até a condicional imposta pelo bloco 4 essa condição sempre será checada para que ocorra esse controle da animação. Somente em um caso muito específico ela não será checada, que é quando o usuário não espera pela animação do menu e clica em algum dos botões.

GITHUB:

- A nova versão do sistema com as mudanças e as suítes de teste estão no repositório do seguinte link: https://github.com/IgorTorquatto/Jogo_Python