FUNDAMENTOS DE PROGRAMAÇÃO

SUMÁRIO

GERENCIANDO PACOTES

Após instalar o PIP em nosso SO, geralmente junto com o Python, podemos utilizá-lo para diferentes tarefas, como instalar, remover, listar e atualizar pacotes. Veremos agora como realizar cada uma dessas tarefas.

Para a instalação de novos pacotes utilizando o PIP, temos o seguinte comando:

```
pip install nome_do_pacote
```

Este comando irá baixar o pacote desejado e instalar em nosso SO.

PYGAME

Para instalar o pacote Pygame, responsável por criar ambientes de jogos, executamos:

```
pip install pygame
```

Para verificar a versão, execute:

```
pip show pygame
```

E para desinstalar o pacote:

```
pip uninstall pygame
```

PIP

É comum que no desenvolvimento de projetos Python, precisarmos instalar diversas bibliotecas para diferentes necessidades, como a comunicação com algum banco de dados ou até a utilização de testes unitários. Porém, não é viável que a instalação dessas bibliotecas seja feita de forma manual, já que o processo de cada uma delas podem ser, no mínimo, complicadas. Para isso, o Python possui uma ferramenta para gerenciamento de pacotes chamado PIP e é ele que vamos usar neste conteúdo.

O que é o PIP?

O PIP é um gerenciador de pacotes para projetos Python. É com ele que instalamos, removemos e atualizamos pacotes em nossos projetos. É similar aos conhecidos npm e composer (php), por exemplo.

O PIP possui uma página onde nós conseguimos buscar os pacotes disponíveis para a utilização:

https://pypi.org/

Pygame é uma biblioteca de jogos multiplataforma (independente de sistema operacional) feita para ser utilizada em conjunto com a linguagem de programação Python. O seu nome tem origem da junção de Py, proveniente de Python e Game, que significa Jogo, ou seja, Jogos em Python.

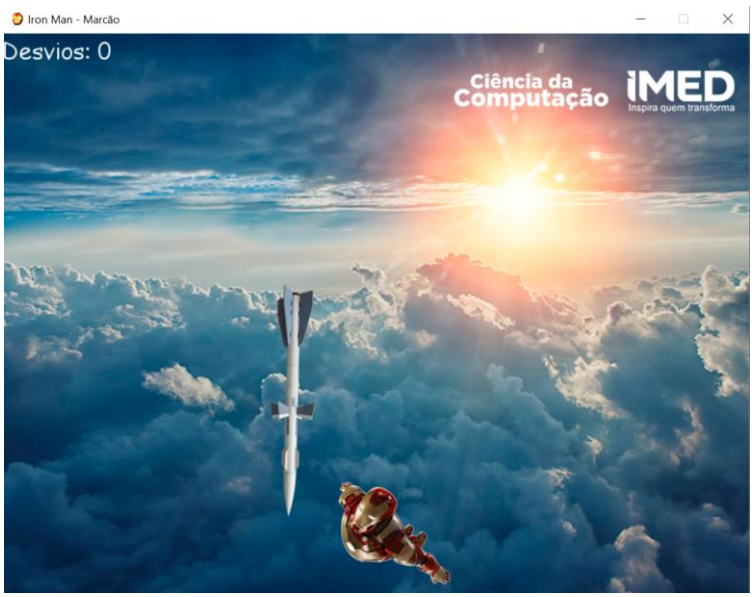
Pygame se baseia na ideia de que as tarefas mais intensivas a nível computacional em um jogo podem ser abstraídas separadamente da lógica principal, ou seja, o uso de memória e CPU (úteis para processar imagens e sons) são tratados pelo próprio código do Pygame e não pelo código do seu jogo. Assim, torna-se possível utilizar uma linguagem de alto nível, como Perl ou mesmo Python para organizar a estrutura do jogo em si.

Pygame é uma biblioteca de jogos multiplataforma (independente de sistema operacional).

IRON MAN

A proposta a partir deste momento é criar um jogo com a temática do Homem de Ferro. A proposta baseia-se em o usuário controlar o movimento do Homem de Ferro, na linha horizontal, desviando de mísseis que virão do topo da janela, aumentando progressivamente a velocidade dos mesmos, sendo que a cada desvio, é contabilizado um ponto a mais, e em caso de possibilidade de choque, a perda do jogo. Com recursos como som, o game deverá ser disponibilizado no formato executável.

Abaixo telas demonstrativas do game:



LOOPING DE TELA

Um dos pontos mais importantes do desenvolvimento de telas, é entender que ela deve estar ativa até que alguma ação determine o seu fechamento. Logo devemos utilizar um while:

```
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            quit()
    pygame.display.update()
    clock.tick(60)
```

É possível visualizar que a função event.get() devolve os eventos da janela do Pygame. Caso seja igual a QUIT o looping deverá ser encerrado (pygame.QUIT é quando clicamos no X já janela).

A função update atualiza a tela e o clock define a taxa de atualização FPS (frame por segundo).

ESTRUTURA BASE

A criação de uma estrutura base, se dá através da configuração da biblioteca e inicialização de algumas funções com parâmetros.

O primeiro passo é importar a Lib Pygame:

```
import pygame
```

Após devemos iniciar a Lib através do comando abaixo:

```
pygame.init()
```

Em seguida precisamos definir o tamanho da janela em largura e altura (width e height).

```
display_width= 800
display_height= 600
gameDisplay=pygame.display.set_mode(
(display_width,display_height))
```

Como pretendemos controlar o fluxo de quadros (frames) por segundo, vamos precisar iniciar o relógio do pygame, através do comando:

```
clock = pygame.time.Clock()
```

PASSO 1 – ESTRUTURA BASE


```
import pygame

#Inicializando o Pygame

pygame.init()

#Setando a variável de tela, com tamanho de 800 por 600 pixels.

display_width= 800

display_height= 600

gameDisplay=pygame.display.set_mode((displayWidth,displayHeight))

#Criado uma variável para controle de tempo ()

clock = pygame.time.Clock()

# Looping Infinito até um break ou quit

while True:

    # comando pygame.event.get() retorna uma lista de eventos que estão acontecendo

    for event in pygame.event.get():

        # se o tipo de evento for igual a fechar Janela, interrompe o looping do while

        if event.type == pygame.QUIT:

            pygame.quit()

            quit()

    #atualiza a tela do pygame

    pygame.display.update()

    #Indica para o controlador de tempo o FPF ou Quadros por Segundo.

    clock.tick(60)
```

CRIANDO VARIÁVEIS

Primeiro vamos ajudar a indicação de tamanho, com valores armazenados em variáveis, assim podemos utilizá-los.

```
display_width = 800

display_height = 600

gameDisplay = pygame.display.set_mode((display_width,display_height))
```

Após vamos criar variáveis de cores, baseadas na escala RGB:

```
black = (0,0,0)

white = (255,255,255)
```

Também podemos criar variável que irá armazenar a figura:

```
ironManImg = pygame.image.load('assets/ironLarge.png')
```

CRIANDO FUNÇÕES

O próximo passo é criar uma função que irá inserir determinada imagem na tela. A função blit, do display, insere a imagem na posição X e Y, enviada por parâmetros para a DEF através do plano cartesiano.

```
def mostraIron(x,y):

    gameDisplay.blit(ironManImg, (x,y))
```

Também podemos definir variáveis de posicionamento inicial do componente, para ser utilizado após.

```
ironPosicaoX = (display_width * 0.45)

ironPosicaoY = (display_height * 0.8)
```

Com as variáveis e funções criadas, podemos atualizar o looping principal do jogo:

```
while True:

    for event in pygame.event.get():

        if event.type == pygame.QUIT:

            pygame.quit()

            quit()

    gameDisplay.fill(white)

    mostraIron(ironPosicaoX,ironPosicaoY)

    pygame.display.update()

    clock.tick(60)
```

MOVIMENTANDO UM OBJETO

Para movimentar um objeto, vamos precisar de algumas variáveis no projeto:

```
movimentoX = 0
```

Precisamos ajustar o looping do jogo, inserindo a leitura de teclas pressionadas:

```
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            quit()
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                movimentoX = -5
            elif event.key == pygame.K_RIGHT:
                movimentoX = 5
        if event.type == pygame.KEYUP:
            if event.key == pygame.K_LEFT or event.key == pygame.K_RIGHT:
                movimentoX = 0
    ironPosicaoX += movimentoX
    gameDisplay.fill(white)
    mostraIron(xInicial,yInicial)
    pygame.display.update()
    clock.tick(60)
```

DEFININDO TAMANHO DO OBJETO

Para controlar as colisões com as bordas da tela, precisamos primeiro armazenar o tamanho do objeto:

```
iron_width = 120
```

Para controlar as colisões do objeto, vamos atualizar o looping o jogo com a seguinte função:

```
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            quit()
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                movimentoX = -5
            elif event.key == pygame.K_RIGHT:
                movimentoX = 5
        if event.type == pygame.KEYUP:
            if event.key == pygame.K_LEFT or event.key == pygame.K_RIGHT:
                movimentoX = 0
    ironPosicaoX += movimentoX
    gameDisplay.fill(white)
    mostraIron(xInicial,yInicial)
    if ironPosicaoX > display_width - iron_width or ironPosicaoX < 0:
        quit()
    pygame.display.update()
    clock.tick(60)
```

REINICIANDO O JOGO

Para que possamos ter a possibilidade de reiniciar o jogo, vamos colocar toda a estrutura de variáveis do jogo e do while em uma função denomiada game_loop:

```
def game_loop():
   .....
game_loop()
```

ESCREVENDO NA TELA

Para escrever na tela, vamos criar algumas funções uteis, como:

```
def text_objects(text, font):

    textSurface = font.render(text, True, black)

    return textSurface, textSurface.get_rect()

def message_display(text):

    largeText = pygame.font.Font('freesansbold.ttf',115)

    TextSurf, TextRect = text_objects(text, largeText)

    TextRect.center = ((display_width/2),(display_height/2))

    gameDisplay.blit(TextSurf, TextRect)

    pygame.display.update()

    time.sleep(2)

    game_loop()

def dead():

    message_display('Você Morreu')
```

Quando sair da tela, mudar para dead() e não terminar o jogo:

```
        if ironPosicaoX > display_width - iron_width or ironPosicaoX < 0:
            dead()
```

ARMAS

Para inserir armas que podem matar o Homem de Ferro, vamos importar a figura, nas variáveis:

```
missile = pygame.image.load('assets/missile.png')
```

Também podemos criar uma função para mostrar as armas:

```
def mostraArmas(posicaoX, posicaoY):

    gameDisplay.blit(missile,(posicaoX, posicaoY))
```

Podemos atualizar a função game_loop() inserindo as armas:

```
    missileX = random.randrange(0, display_width)

    missileY = -600

    missileHeight = 250

    missileWidht = 50

    missileXSpped = 7
```

No while, podemos inserir o seguinte trecho de código:

```
.....
       mostraArmas(missileX, missileY)
       missileY += missileXSpped


       if missileY > display_height:
           missileY = 0 - missileHeight
           missileX = random.randrange(0, display_width)
           missileXSpped+=1
......
```

COLISÃO COM OS OBJETOS

Para verificar se houve colisão podemos utilizar a seguinte lógica. No While, primeiro analisamos se a posição de Y do Homem de Ferro é maior que posição Y do Missel mais o tamanho de altura dele. Se sim, podemos analisar se houve ou não colisão:

```
.....
         if ironPosicaoY < missileY + missileHeight:
            #print("Analisando Colisão")
            #print(ironPosicaoX, ironPosicaoX+iron_width)
            #print(missileX, missileX+missileWidht)
            if ironPosicaoX < missileX and ironPosicaoX+iron_width > missileX or missileX+missileWidht
> ironPosicaoX and missileX+missileWidht < ironPosicaoX+iron_width:
                dead()
.....
```

CONTADOR DE DESVIOS

O primeiro passo é criar uma função para escrever na tela os desvios:

```
def escrevendoPlacar(count):
    font = pygame.font.SysFont(None, 25)
    text = font.render("Desvios: "+str(count), True, black)
    gameDisplay.blit(text,(0,0))
```

Primeiro vamos criar uma função contadora de desvios, no game_loop():

```
desvios =0
```

NO while, precisamos adaptar para ser chamada a função:

```
....
gameDisplay.fill(white)
mostraIron(xInicial, yInicial)
escrevendoPlacar(desvios)
......
```

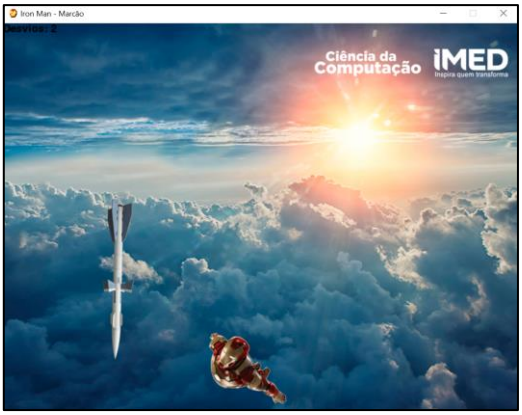
E após enviar mais um missel, contamos os desvios:

```
....
 if missileY > display_height:
      missileY = 0 - missileHeight
      missileX = random.randrange(0, display_width)
      missileXSpped += 1
      desvios += 1
....
```

SETANDO ÍCONES E TÍTULO DA JANELA

```
gameIcon = pygame.image.load('assets/ironIcon.png')
pygame.display.set_icon(gameIcon)
pygame.display.set_caption('Iron Man - Marcão')
```

BACKGROUND



Primeiro precisamos carregar a imagem de fundo do jogo:

```
bg = pygame.image.load("assets/sky.png")
```

NO While insira a seguinte linha de código:

```
....
xInicial += x_altera
gameDisplay.fill(white)
gameDisplay.blit(bg, (0,0))
....
```

INSERINDO SONS

Vamos criar variáveis para sons esporádicos:

```
explosaoSound = pygame.mixer.Sound("assets/explosao.wav")

missileSound = pygame.mixer.Sound("assets/missile.wav")
```

Na função de looping do jogo, podemos chamar a trilha base:

```
def game_loop():

    pygame.mixer.music.load('assets/ironsound.mp3')

    pygame.mixer.music.play(-1)

....
```

Quando enviamos um novo Missel, rodamos o som:

```
if missileY > display_height:

            pygame.mixer.Sound.play(missileSound)

            missileY = 0 - missileHeight

            missileX = random.randrange(0, display_width)

            missileXSpped += 1

            desvios += 1
```

E quando for atingido, chamamos o som também:

```
def dead():

    pygame.mixer.Sound.play(explosaoSound)

    pygame.mixer.music.stop()

    message_display('Você Morreu')
```

CRIANDO EXECUTÁVEL

Primeiramente é necessário instalar a LIB:

```
python -m pip install cx_Freeze –upgrade
```

Após crie um arquivo chamado setup.py

```
import cx_Freeze

executables = [cx_Freeze.Executable(script="jogo.py",icon="assets/ironIcon.ico")]

cx_Freeze.setup(

    name="Iron Man Marcão",

    options={"build_exe": {"packages":["pygame"],

                           "include_files":["assets"]}},

    executables = executables

    )
```

Comando para criar um instalador para windows:

```
python setup.py bdist_msi
```

Se você estiver em um Mac, faça:

```
python setup.py bdist_dmg
```

Ou se quiser somente gerar um Build:

```
python setup.py build
```

A máquina que irá executar, não precisa do Python instalado, mas sim do C++.

https://support.microsoft.com/pt-br/help/2977003/the-latest-supported-visual-c-downloads