UNIVERSIDADE FEDERAL DE MATO GROSSO
CAMPUS UNIVERSITÁRIO DO ARAGUAIA
Instituto de Ciências Exatas e da Terra
Curso de Bacharelado em Ciência da Computação

Disciplina: Programação I
Professor: Ivairton M. Santos

Trabalho

Trabalho baseado no tutorial: https://realpython.com/pygame-a-primer/

Neste trabalho vamos utilizar a biblioteca PyGame (https://www.pygame.org/), que tem como princípio proporcionar um conjunto de funcionalidades para desenvolvimento de jogos utilizando a linguagem Python, equivalentes à biblioteca SDL (https://www.libsdl.org/) destinada ao desenvolvimento utilizando a linguagem C/C++.

O principal objetivo deste trabalho é proporcionar uma experiência de desenvolvimento de sistema utilizando uma biblioteca com diversas funcionalidades disponíveis, desenvolver uma lógica de um jogo simples e detectar/manipular eventos do jogo (por exemplo, a colisão entre objetos).

O desenvolvimento será incremental, conforme orientações apresentadas neste documento, ao final do projeto você deverá resolver (por conta própria) alguns desafios.

O contexto do jogo proposto consiste do controle pelo jogador de um bloco (por meio das teclas direcionais do teclado) que deve desviar de inimigos, evitando a colisão. Um exemplo de abstração do jogo seria uma nave espacial que deve desviar de asteroides, ou um avião que deve desviar de misseis.

Configuração inicial:

Baixe e instale a biblioteca (http://www.pygame.org/download.shtml) no seu sistema operacional. Será necessário utilizar o Python em sua versão 3 (mais atual). Caso o pacote PIP3 esteja instalado, basta executar o comando em seu terminal:

```
# pip3 install pygame
```

Para verificar se a instalação está correta, crie um novo arquivo com código (*.py*) e utilize o seguinte código de teste:

```
import pygame
from pygame.locals import *

pygame.init()
```

O exemplo acima importa a biblioteca *pygame* e o pacote *locals*. Se estiver tudo certo, ao executar o programa irá aparecer no terminal a versão instalada do PyGame.

Construção do Jogo:

Para configurar a resolução da tela do jogo, utilize o comando:

```
screen = pygame.display.set_mode( (800, 600) )
```

Então, vamos definir a lógica do loop do jogo. Nesta etapa precisaremos capturar um evento do jogo. Um evento é qualquer ação, desde uma tecla qualquer pressionada, ou uma interação entre componentes/elementos do jogo.

Para controlar a tela principal do jogo vamos "pegar" qualquer evento e então verificar se ele corresponde a uma tecla pressionada e finalmente verificar se a tecla é a tecla ESC (para sair do jogo):

```
# Flag para manter o loop do jogo
running = True

# Loop principal
while running:
    # Laco que captura e verifica todos os eventos no jogo
    for event in pygame.event.get():
        # Verifica se ocorreu o evento KEYDOWN; KEYDOWN eh uma constante
        # definida em pygame.locals
        if event.type == KEYDOWN:
            # Se a tecla ESC foi pressionada, modifica a flag para então
            # terminar o loop do jogo
            if event.key == K_ESCAPE:
                running = False
        # Verifica se ocorreu o evento de SAIDA (fechamento da janela)
        elif event.type == QUIT:
            running = False
```

Definindo uma superfície:

Uma superfície, ou bloco, é qualquer objeto do jogo. Pode se um personagem, ou uma barreira, por exemplo. Vamos definir um simples retângulo e coloca-lo na tela do jogo:

```
# Cria uma superfície com tamanho 50px x 25px
surf = pygame.Surface( (50, 25) )

# Definie o preenchimento com a cor branca (RGB)
surf.fill( (255, 255, 255) )
rect = surf.get_rect()

# Plota na tela a superfície criada, na posição x=400 e y=300
    screen.blit(surf, (400, 300)) # O comando Blit tem o efeito equivalente a
                                  # "Desenhe em Screen"
    pygame.display.flip() # Este comando tem a função de atualizar a posição
                          # dos objetos. É uma atualização do jogo.
```

Sprites:

Sprites é uma representação 2D de algo (objeto) na tela. Tipicamente é uma figura/imagem, que representa um personagem. Vamos usar o modelo de Sprite mais elementar do PyGame. Aqui vamos criar uma Classe (não se preocupe ainda com o conceito de Classe) chamada Player, que representa nosso personagem principal, que o jogador vai controlar. Por enquanto é apenas um

retângulo, que poderá se mover por toda a tela.

```
#Cria uma classe para representar o objeto que o jogador vai controlar
class Player(pygame.sprite.Sprite):
    def __init__(self):
        super(Player, self).__init__()
        self.surf = pygame.Surface((75, 25)) #Tamanho do retângulo
        self.surf.fill((255, 255, 255)) #Define a cor branca
        self.rect = self.surf.get_rect()
```

Entrada de comandos:

Vamos usar as teclas de setas para controlar o retângulo. Para isso vamos usar um comando que captura um evento de uma tecla pressionada, então verifica se a tecla é uma das setas, e então atualiza a posição do retângulo. Para isso usaremos os códigos:

```
# captura uma tecla pressionada
pressed_keys = pygame.key.get_pressed()

(...)

#esta função deve ser implementada dentro da classe Player e irá atualizar a
#posição do retângulo, deslocando 5px de acordo com a tecla pressionada
def update(self, pressed_keys):
    if pressed_keys[K_UP]:
        self.rect.move_ip(0, -5)
    if pressed_keys[K_DOWN]:
        self.rect.move_ip(0, 5)
    if pressed_keys[K_LEFT]:
        self.rect.move_ip(-5, 0)
    if pressed_keys[K_RIGHT]:
        self.rect.move_ip(5, 0)
```

Para garantir que o retângulo não ultrapasse os limites da tela do jogo, vamos adicionar as seguintes regras na função update:

```
    #Mantém o jogador na tela
    if self.rect.left < 0:
        self.rect.left = 0
    elif self.rect.right > 800:
        self.rect.right = 800
    if self.rect.top <= 0:
        self.rect.top = 0
    elif self.rect.bottom >= 600:
        self.rect.bottom = 600
```

Até o momento temos o seguinte código, que permite que o retângulo seja movimentado na tela:

```
import pygame
import random #Biblioteca para geração de números aleatórios
from pygame.locals import *

class Player(pygame.sprite.Sprite):
    def __init__(self):
        super(Player, self).__init__()
        self.surf = pygame.Surface((75, 25))
```

```
        self.surf.fill((255, 255, 255))
        self.rect = self.surf.get_rect()

    def update(self, pressed_keys):
        if pressed_keys[K_UP]:
            self.rect.move_ip(0, -5)
        if pressed_keys[K_DOWN]:
            self.rect.move_ip(0, 5)
        if pressed_keys[K_LEFT]:
            self.rect.move_ip(-5, 0)
        if pressed_keys[K_RIGHT]:
            self.rect.move_ip(5, 0)

        if self.rect.left < 0:
            self.rect.left = 0
        elif self.rect.right > 800:
            self.rect.right = 800
        if self.rect.top <= 0:
            self.rect.top = 0
        elif self.rect.bottom >= 600:
            self.rect.bottom = 600

pygame.init()
screen = pygame.display.set_mode((800, 600))
player = Player()

running = True
while running:
    for event in pygame.event.get():
        if event.type == KEYDOWN:
            if event.key == K_ESCAPE:
                running = False
        elif event.type == QUIT:
            running = False

    pressed_keys = pygame.key.get_pressed()
    player.update(pressed_keys)

    screen.blit(player.surf, player.rect)
    pygame.display.flip()
```

Criando os inimigos:

Vamos criar os inimigos, usando o mesmo princípio de criação do Player, ou seja, definindo um novo Sprite. Para isso também vamos definir uma nova classe e utilizar uma estratégia de sortear a posição inicial (vertical) do inimigo e definir sua velocidade de deslocamento. O código da classe deve ficar como:

```
class Enemy(pygame.sprite.Sprite):
    def __init__(self):
        super(Enemy, self).__init__()
        self.surf = pygame.Surface((20, 10)) #Define tamanho do retangulo
        self.surf.fill((255, 255, 255)) #Define a cor branca
        #Cria o objeto e sorteia sua posição
        self.rect = self.surf.get_rect(center=(820, random.randint(0, 600)))
        self.speed = random.randint(5, 20) #Define sua velocidade num
                                           #intervalo entre 5 e 20

    def update(self):
        self.rect.move_ip(-self.speed, 0) #Move o objeto apenas no eixo x
```

```
                                        #(observe o "-speed")
        if self.rect.right < 0: #Verifica se o objeto alcançou a coordenada x
                                #< 0, então elimina-o
            self.kill()
```

Definindo grupos:

Uma funcionalidade útil da PyGame é a possibilidade de determinar agrupamentos de Sprites. Isso é mais interessante do que utilizar uma lista (vetor/matriz) porque há funcionalidades específicas para controle destes grupos de Sprites, por exemplo a função Kill, que remove o Sprite do jogo. No código a seguir são definidos dois grupos, um com todos os inimigos e outro com todos os Sprites, incluindo os inimigos mais o jogador:

```
enemies = pygame.sprite.Group()
all_sprites = pygame.sprite.Group()
all_sprites.add(player)
```

Isso irá facilitar bastante a atualização de cada Sprites, como por exemplo, usando o código:

```
for entity in all_sprites:
    screen.blit(entity.surf, entity.rect)
```

Bom, agora com a possibilidade de agrupar os inimigos precisamos criar os inimigos. Para isso vamos definir um novo evento, que consiste nesta adição. Basicamente vamos definir um novo evento, que chamaremos de ADDENEMY, que consiste simplesmente de uma contagem. Assim, a cada "contagem" criaremos um novo inimigo. Podemos criar o novo evento com o comando:

```
ADDENEMY = pygame.USEREVENT + 1
```

Este novo evento (ADDENEMY) deverá ser tratado no loop geral do jogo.

Para que o evento ocorra de tempo em tempo, vamos fazer uso de um "Relógio", criando um timer no jogo. Neste exemplo vamos acionar o evento a cada 250 milisegundos:

```
pygame.time.set_timer(ADDENEMY, 250)
```

Então vamos tratar o evento no loop principal do jogo, criando um novo inimigo (*new_enemy*), adicionando ao grupo inimigos (*enemis.add(new_enemy)*) e então adicionando ao grupo de todos os Sprites (*all_sprites.add(new_enemy)*):

```
while running:
    for event in pygame.event.get():
        if event.type == KEYDOWN:
            if event.key == K_ESCAPE:
                running = False
        elif event.type == QUIT:
            running = False
        elif(event.type == ADDENEMY):
            new_enemy = Enemy()
            enemies.add(new_enemy)
            all_sprites.add(new_enemy)
```

Tratando a colisão entre o jogador e os inimigos:

Geralmente, no desenvolvimento de jogos, uma das tarefas mais elaboradas/complicadas é a detecção de colisão entre componentes e a gestão de sua física (simulação do comportamento esperado). Uma das vantagens mais interessantes na biblioteca PyGame é a disponibilidade de várias funções para detecção de colisão entre objetos do jogo.

Um exemplo de função disponível que detecta qualquer tipo de colisão entre objetos é a "SpriteCollideAny" que recebe como parâmetros um Sprite e um grupo de Sprites, de modo que verifica se o Sprite específico está em contato (colidindo) com qualquer outro Sprite do grupo. Portanto, para verificar se nosso player está em colisão com um inimigo, iremos implementar:

```
if pygame.sprite.spritecollideany(player, enemies):
    player.kill()
```

A função "kill", faz com que o jogo acabe/pare, em razão de marcar o player como derrotado.

O **código completo** do que temos até então é:

```
# Biblioteca PyGame
import pygame
# Biblioteca para geração de números pseudoaleatórios
import random
# Módulo da biblioteca PyGame que permite o acesso às teclas utilizadas
from pygame.locals import *

# Classe que representar o jogador
class Player(pygame.sprite.Sprite):
    def __init__(self):
        super(Player, self).__init__()
        self.surf = pygame.Surface((75, 25)) #Define o retangulo que representa o
                                             #player
        self.surf.fill((255, 255, 255)) #Preenche o retangulo com branco (RGB)
        self.rect = self.surf.get_rect()

    # Determina ação de movimento conforme teclas pressionadas
    def update(self, pressed_keys):
        if pressed_keys[K_UP]:
            self.rect.move_ip(0, -5)
        if pressed_keys[K_DOWN]:
            self.rect.move_ip(0, 5)
        if pressed_keys[K_LEFT]:
            self.rect.move_ip(-5, 0)
        if pressed_keys[K_RIGHT]:
            self.rect.move_ip(5, 0)

        # Mantém o jogador nos limites da tela do jogo
        if self.rect.left < 0:
            self.rect.left = 0
        elif self.rect.right > 800:
            self.rect.right = 800
        if self.rect.top <= 0:
            self.rect.top = 0
        elif self.rect.bottom >= 600:
            self.rect.bottom = 600

# Classe que representa os inimigos
class Enemy(pygame.sprite.Sprite):
    def __init__(self):
        super(Enemy, self).__init__()
        self.surf = pygame.Surface((20, 10)) #Definição do retangulo
        self.surf.fill((255,255,255)) #Preenche com cor branca (RGB)
```

```
        self.rect = self.surf.get_rect( #Coloca na extrema direita (entre 820 e 900) e
                                        #sorteia sua posição em relacao à coordenada y
                                        #(entre 0 e 600)
            center=(random.randint(820, 900), random.randint(0, 600))
        )
        self.speed = random.uniform(1, 15) #Sorteia sua velocidade, entre 1 e 15

    #Função que atualiza a posiçao do inimigo em função da sua velocidade e termina
    #com ele quando ele atinge o limite esquerdo da tela (x < 0)
    def update(self):
        self.rect.move_ip(-self.speed, 0)
        if self.rect.right < 0:
            self.kill()


# Inicializa pygame
pygame.init()

# Cria a tela com resolução 800x600px
screen = pygame.display.set_mode((800, 600))

# Cria um evento para adição de inimigos
ADDENEMY = pygame.USEREVENT + 1
pygame.time.set_timer(ADDENEMY, 250) #Define um intervelao para a criação de cada
                                     #inimigo (milisegundos)

# Cria o jogador (nosso retangulo)
player = Player()

# Define o plano de fundo, com a cor preta (RGB)
background = pygame.Surface(screen.get_size())
background.fill((0, 0, 0)) #Cor preta

enemies = pygame.sprite.Group() #Cria o grupo de inimigos
all_sprites = pygame.sprite.Group() #Cria o grupo de todos os Sprites
all_sprites.add(player) #Adicionar o player no grupo de todos os Sprites

running = True #Flag para controle do jogo

while running:
    #Laco para verificação do evento que ocorreu
    for event in pygame.event.get():
        if event.type == KEYDOWN:
            if event.key == K_ESCAPE: #Verifica se a tecla ESC foi pressionada
                running = False
        elif event.type == QUIT: #Verifica se a janela foi fechada
            running = False
        elif(event.type == ADDENEMY): #Verifica se é o evento de criar um inimigo
            new_enemy = Enemy() #Cria um novo inimigo
            enemies.add(new_enemy) #Adiciona o inimigo no grupo de inimigos
            all_sprites.add(new_enemy) #Adiciona o inimigo no grupo de todos os
                                       #Sprites
    screen.blit(background, (0, 0)) #Atualiza a exibição do plano de fundo do jogo
                                    #(neste caso não surte efeito)
    pressed_keys = pygame.key.get_pressed() #Captura as teclas pressionadas
    player.update(pressed_keys) #Atualiza a posição do player conforme teclas usadas
    enemies.update() #Atualiza posição dos inimigos
    for entity in all_sprites:
        screen.blit(entity.surf, entity.rect) #Atualiza a exibição de todos os Sprites

    if pygame.sprite.spritecollideany(player, enemies): #Verifica se ocorreu a colisão
                                                        #do player com um dos inimigos
        player.kill() #Se ocorrer a colisão, encerra o player

    pygame.display.flip() #Atualiza a projeção do jogo
```

Desafios:

Agora implemente uma versão aprimorada do jogo, que atenda aos seguintes requisitos:

Requisito 1:

Use imagens/figuras para representar o bloco do jogador e os inimigos. Além de atribuir uma cor ou textura ao plano de fundo. Para isso, estude as funções "pygame.image.load" e "image.set_colorkey", que deverão substituir as funções "pygame.Surface" e "surf.fill".

Requisito 2:

Implemente um placar para o jogador, que pode considerar a pontuação em função do tempo decorrido do jogo, ou do número de inimigos que foram evitados.

Requisito 3:

Implemente uma mensagem que avise ao jogador quando o jogo terminar em razão de uma colisão. A mensagem deve conter também a pontuação (placar) obtida.

Requisito 4:

Implemente uma lógica que faça com que o jogo fique gradativamente mais difícil. Isso pode ser implementado a partir de 2 estratégias (escolha uma delas, ou ambas): 1) à medida que o tempo de jogo avança, os inimigos devem ter mais velocidade; ou 2) estabeleça fases, onde cada fase tem um tempo limitado de jogo e à medida que o usuário avança nas fases, elas correspondem a inimigos com velocidades maiores.