Header
Conteúdo abordado nesta aula:

1. Introdução

Para dar início a esta terceira aula do nosso módulo avançado, iremos introduzir a Pygame, esta que é uma biblioteca open source usada na criação e gerenciamento de aplicações multimídia como jogos. Dessa forma, a Pygame pode ser intitulada como uma “game engine”, ou seja, um motor de jogo.

É também compreendida como uma camada de abstração construída sobre a biblioteca SDL e, portanto, é altamente portável, ou seja, possui fácil adaptação a todas as plataformas e sistemas operacionais.

1.1 História

A biblioteca teve seu desenvolvimento iniciado nos anos 2000 por Pete Shinners que, ao se familiarizar melhor com a linguagem Python e a biblioteca SDL (simple directmedia library), decidiu unir essas duas coisas com o intuito de “fazer as coisas simples de maneira fácil e as coisas difíceis de maneira direta.”

A biblioteca SDL é uma biblioteca, escrita em linguagem c, responsável, de maneira geral, pelos recursos de multimídia utilizados tanto em aplicações open sources, como em aplicações comerciais e empresariais. A SDL representa uma interface simples para gráficos, imagens, sons e dispositivos de entrada de várias plataformas.

A descontinuidade da implementação de tal biblioteca para python, criada por Mark Baker e conhecida como pySDL, foi umas das principais inspirações para Pete iniciar um projeto mais forte e robusto, sob a SDL, que mais tarde viria a ser chamado de Pygame.

1.2 Vantagens e desvantagens

As vantagens da biblioteca se estendem das vantagens oferecidas pela própria linguagem, vantagens estas que já foram abordadas no primeiro módulo deste curso e que podem ser resumidas na palavra “simplicidade”. Além disso a biblioteca possui uma boa documentação e conta com um grande apoio da comunidade, o que mantém o conteúdo sempre atualizado e com o mínimo de erros e “bugs”.

Já as desvantagens da biblioteca são, de certa forma, um reflexo dessa simplicidade, uma vez que o Python possui uma performance relativamente baixa, resolvendo em sua maioria, apenas gráficos 2D, não havendo um suporte a aceleração de hardware.

Apesar disso, a biblioteca se mostra como uma excelente ferramenta de aprendizado na área de desenvolvimento de jogos e aplicações multimídia, cujo o conhecimento adquirido pode ser levado adiante, servindo como uma boa base para se trabalhar e aprender mais dentro de tal área.

1.3 Objetivos e direcionamento da aula

Nesta aula, teremos como objetivo principal, entender os conceitos básicos para se implementar um jogo utilizando as funcionalidades da biblioteca Pygame. Para isso devemos primeiramente realizar a instalação da biblioteca, esta que pode ser feita de maneira bem simples através de um único comando no terminal:

```
In [ ]: pip install --user pygame
```

Caso o pip não esteja instalado, execute a priori, o comando:

```
In [ ]: apt install python-pip
```

Uma vez instalada a biblioteca, nos deparamos com o seguinte questionamento: “Por onde começar o desenvolvimento de um jogo?”

Um jogo em pygame é composto de maneira geral pelos seguintes tópicos:

- Loop principal (game loop)
- Eventos e tratamento destes

Para complementar tais aplicações, usamos também conhecimentos como:

- Formas e polígonos
- Textos e fontes
- Tabela de cores
- Tratamento de imagens
- Sprites
- Sons

Conhecendo e entendendo bem esses tópicos, já nos tornamos aptos a desenvolver nosso primeiro jogo em pygame, então mãos à obra!

2. Implementação

2.1 Primeiros passos

Para começar nosso jogo utilizando pygame, primeiramente devemos importar a biblioteca através do comando

In [ ]:
```
import pygame
```

Em seguida, utilizamos:

In [ ]:
```
pygame.init()
```

Isso inicia funções básicas da biblioteca, e sem ele, o programa não funciona :(

Após isso, a próxima coisa a se fazer é criar uma janela para o jogo. Para fazer isso, podemos criar a classe “Janela”, que tem como atributos: a altura e a largura, além da cor de fundo da tela.

In [ ]:
```
class Window:
    __width = 0 #largura
    __height = 0 #altura
    __color = (0, 0, 0) #cor

    def __init__(self, largura, altura): #construtor
        self.__width = largura
        self.__height = altura

    def returnWinSize(self): #retorna tamanho da tela
        return (self.__width, self.__height)

    def returnColor(self): #retorna a cor da janela
        return self.__color
```

Então, criamos nossa janela com:

In [ ]:
```
w = Window(1600, 900) #cria objeto janela com dimensões 1600x900

win = pygame.display.set_mode(w.returnWinSize()) #cria a janela do jogo
```

Podemos também mudar o título da janela com o comando

In [ ]:
```
pygame.display.set_caption('Nome_do_Jogo')
```

Por fim, podemos escolher uma cor para o fundo da janela com a função:

In [ ]:
```
win.fill(w.returnColor()) #onde win é o nome da janela que você criou. S
e você chamou de outra coisa, altere este nome!

#entenderemos logo a seguir o porquê de (0, 0, 0)
```

2.2 Cores

Para definirmos uma cor, utilizamos um código de cores muito conhecido, o RGB. Do inglês, RGB significa “Red Green Blue”, ou “Vermelho Verde Azul”, representando as três cores primárias do código de cores. A cor final é a mistura das três cores.

Nosso código de cores serão valores inteiros que variam entre 0 e 255 para cada uma das três cores, formando uma tupla. Por exemplo:

Vermelho é representado como (255, 0, 0);
Verde é representado como (0, 255, 0)
Azul é representado como (0, 0, 255)
Roxo é representado como (255, 0, 255)
Amarelo é representado como (255, 255, 0)

Podemos criar uma cor escrevendo pygame.Color(vermelho, verde, azul)

Você pode encontrar o código RGB da cor que você procura aqui (https://www.w3schools.com/colors/colors_picker.asp)!

2.3 Linhas, retângulos e outros polígonos

A biblioteca pygame facilita bastante o desenho de linhas, retângulos, círculos e outros polígonos, onde cada um possuirá sua determinada função. Todos eles costumam partir de `pygame.draw` . Mas escrever só essa linha não é o suficiente!

Após chamar `pygame.draw.funcao()` , precisamos **atualizar** a tela, para que as alterações sejam mostradas. Para isso, utilizamos o comando `pygame.display.update()` .

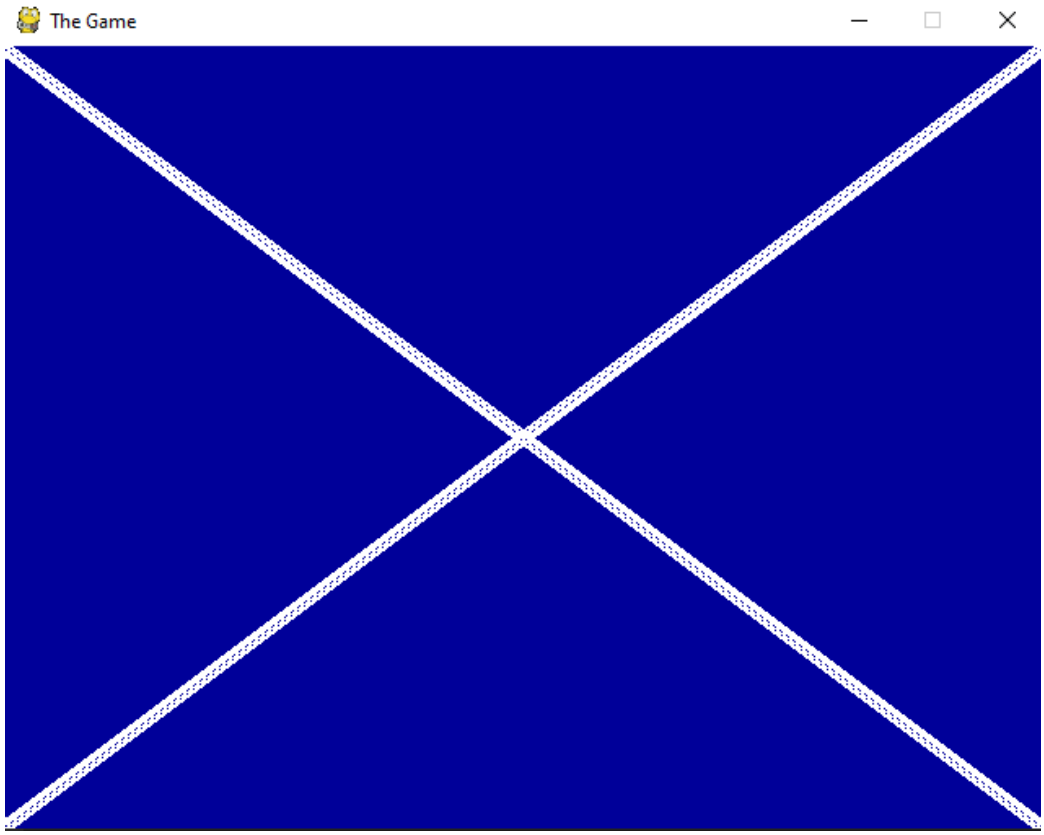
Para criar uma linha, escrevemos:

In [ ]:
```
pygame.draw.line(surf, color, ponto_inicial, ponto_final, largura)
```

No lugar de surf, utilizaremos nossa janela, 'win', para simbolizar que queremos que a linha seja desenhada na janela. No lugar de color, colocamos uma cor (tanto uma tupla de inteiros como um pygame.Color()). Ponto_inicial e ponto_final são referentes aos pontos inicial e final da reta, e por fim, largura é a largura da linha (int).

Exercício

Faça um "X" na tela. A cor e largura da linha ficam a seu critério.



Para fazer um círculo, escrevemos:

```
In [ ]: pygame.draw.circle(surf, color, centro, raio)
```

Surf será o mesmo da linha, assim como color. De forma semelhante a linha, centro é o ponto central do círculo, e raio, o raio do círculo.

Para desenhar um polígono genérico, escrevemos:

```
In [ ]: pygame.draw.polygon(surf, color, ponto1, ponto2, ponto3, etc)
```

Surf e color serão o mesmo serão os mesmo dos acima. Na função Polygon, você pode passar diversos pontos e formar um polígono, que depende do número de pontos passados e a posição deles.

Por fim, para desenhar um retângulo, escrevemos:

```
In [ ]: pygame.draw.rect(surf, color, (ponto1, ponto2))
```

Exercício

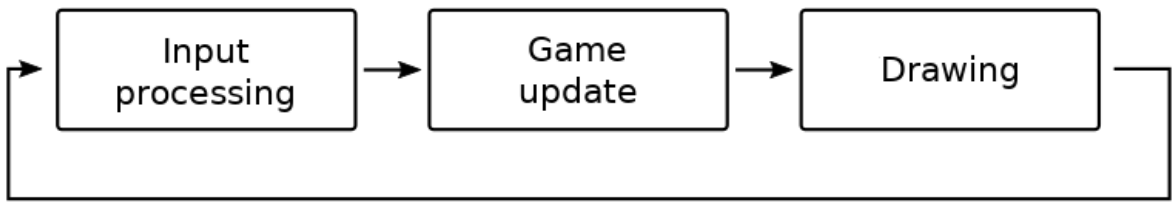
Imagine que você está implementando um jogo de tiro ao alvo. Um bom começo seria desenhando o alvo! O alvo pode ser composto por cinco círculos: Escolha cinco cores e desenhe um alvo com pygame!

Dica: Desenhe os círculos do maior para o menor. Caso você desenhe o maior por ultimo, ele vai ficar "em cima" dos menores, e te dará a impressão de que você só desenhou um círculo.

2.4 Game Loop

O conceito de loop é bem recorrente na programação e uma vez que já estamos familiarizados com ele, fica fácil compreender o funcionamento e a utilidade deste no desenvolvimento de jogos.

O “game loop” é o loop principal de um jogo, sendo uma parte importante da estrutura geral deste. De maneira sucinta, dividimos ele em três subestruturas que se repetem em tempo de execução: Input processing (processamento de entrada) Game update (atualização do jogo) Render (renderização)



A parte de processamento de entrada é também conhecida como processamento de eventos. É a parte responsável por detectar eventos advindos de dispositivo de input externos ao jogo como mouse, teclado etc.

Já a parte de atualização é responsável por mudar o estado dos componentes do jogo de acordo com o processamento de entrada. Através de linhas de código, é decidido o que cada tipo de entrada encadeia dentro do nosso jogo, como por exemplo, apertar a barra de espaço e executar um pulo.

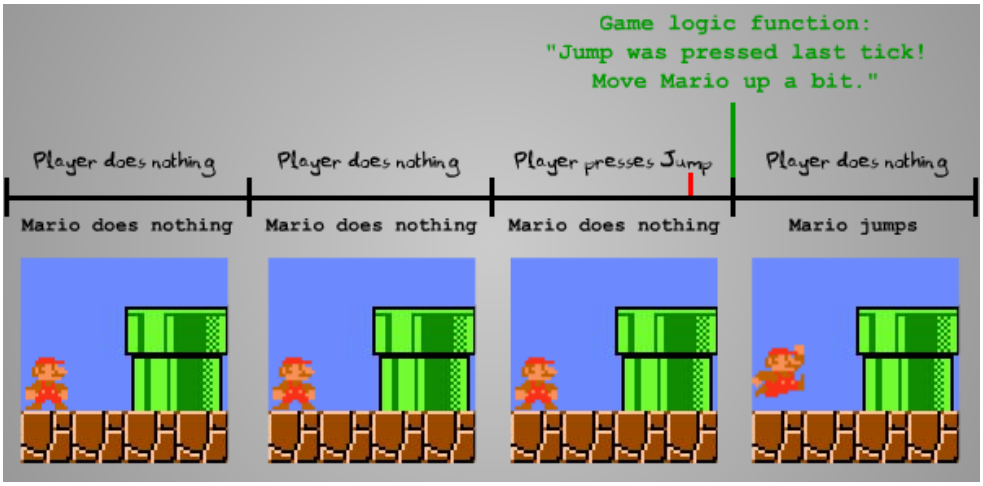
Por fim, temos a renderização. Ao final do game loop, com todas alterações dentro do jogo feitas, ocorre a renderização do nosso jogo, ou seja, sua construção visual.

É válido pontuar que estes acontecimentos ocorrem em tempo de execução, portanto, é interessante nos atentar ao quão rápido essas coisas acontecem.

Dentro de um gameloop, também precisamos dizer um tempo mínimo para o programa rodar cada iteração: se você roda o mesmo jogo em um computador antigo e um computador de ponta, o que acontece é que a velocidade do jogo ficará muito mais baixa no computador antigo, e por mais que ambos consigam processar os dados em um tempo razoavel, isso gera inconsistências. Existe mais de um meio de consertar esse problema. O mais utilizado é o `pygame.time.clock()`. Porém, sua abordagem é um pouco mais complexa, e por isso, utilizaremos `pygame.time.delay()`.

2.5 Eventos

Como citado na parte de processamento de entrada, temos a introdução do conceito de eventos. Estes podem ser definidos como tudo aquilo que acontece externamente ao jogo. Dessa forma, podemos listar como exemplo de eventos, entradas provenientes do teclado e mouse.



Os eventos são essenciais aos jogos, uma vez que se faz necessário uma leitura dos padrões executados pelo jogador para que as coisas aconteçam dentro do jogo.

Todas as teclas do teclado que forem apertadas entrarão para uma fila de teclas a serem processadas pelo computador. A função `pygame.event.get()` nos da uma lista com essas teclas, e então só nos resta iterar sobre essa lista e checar quais teclas foram apertadas:

In [ ]:
```
run = True

while run:
    for event in pygame.event.get():
            if event.type == pygame.QUIT:
                run = False
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_ESCAPE:
                    run = False
```

Exemplo

Fazer 4 if / elif que consigam identificar as setas do teclado.

K_LEFT = esquerda (←)
K_RIGHT = direita (→)
K_UP = cima (↑)
K_DOWN = baixo (↓)

In [ ]:
```
for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                print("←")
            elif event.key == pygame.K_RIGHT:
                print("→")
            elif event.key == pygame.K_UP:
                print("↑")
            elif event.key == pygame.K_DOWN:
                player.moveDown("↓")
            elif event.key == pygame.K_ESCAPE:
                run = False
```

Além disso, podemos receber inputs do mouse também. Para saber se algum botão do teclado foi pressionado, usaremos uma estratégia similar a do teclado:

In [ ]:
```
for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False

        if event.type == pygame.MOUSEBUTTONDOWN:
            if event.button == 1:
                print("O botão esquerdo do mouse foi clicado!")
                mouseX, mouseY = pygame.mouse.get_pos()


            # event.button == 1 => botão esquerdo
            # event.button == 2 => botão do scroll
            # event.button == 3 => botão direito
            # event.button == 4 => qualquer botão
```

2.6 Textos e fontes

A biblioteca pygame traz consigo módulos para carregar e renderizar *.TTF (Truetype fonts) em forma de imagem dentro dos jogos e assim gerar textos.

Para iniciar o módulo, utilizamos o comando:

In [ ]:
```
pygame.font.init()
```

E para desativá-lo

In [ ]:
```
pygame.font.quit()
```

Às vezes, pode ser útil a verificação da inicialização deste módulo. Para isso, utilizamos:

In [ ]:
```
pygame.font.get_init()
```

Essa função retorna um booleano True caso o módulo tenha sido inicializado e False caso contrário.

Podemos importar uma determinada fonte a nossa escolha, uma vez que já tenhamos o arquivo .ttf desta em mãos. Para isso utilizamos a função:

In [ ]:
```
pygame.font.Font("nome_do_arquivo", size)
```

Tal função é responsável por criar um objeto do tipo fonte a partir de um arquivo, este que deve estar no mesmo diretório do código fonte, caso contrário, seu diretório de origem deve ser passado para a função concatenado com seu nome. Além disso passamos o tamanho que desejamos para nosso texto.

Uma vez criado este objeto, podemos manipulá-lo de diversas formas diferentes, a nosso critério. Algumas funcionalidades que a biblioteca já nos fornece são:

In [ ]:
```
objeto.set_bold(bool) #negrito

objeto.set_italic(bool) #itálico

objeto.set_underline(bool) #underline
```

Para ambas as funções, é passado um booleano e, caso este seja True, a operação é feita sobre nosso objeto.

Por fim, chegamos na parte de renderização de nossos textos. Para isso, contamos com a função:

In [ ]:
```
superf = objeto.render(text, antialias, color)
```

Tal função cria uma superfície e desenha nosso texto previamente configurado nela. Passamos para essa função, nosso texto, um booleano responsável por aplicar ou não o antialias (anti serrilhado) , e a cor que desejamos aplicar ao nosso texto (utilizando código RGB).

Agora precisamos desenhar essa superficie sobre nossa janela. É importante mencionar que ao fazer `objeto.render(...)` além da superficie, criamos também um retângulo que representa a área ocupada pelo texto. Para imprimir o texto, usaremos a função `.blit()` , colocando antes do ponto a janela criada.

Para conseguir o retângulo que envolve o texto, basta escrever `superf.get_rect()` . Além disso, podemos escolher onde o texto será colocado na tela movendo a posição deste retângulo. Para acessar o centro do retângulo, escrevemos:

In [ ]:
```
superfRect = superf.get_rect()
superfRect.center = ((800), (450))

win.blit(superf, superfRect) #lembrando sempre de chamar pygame.display.
update(), caso contrario, a tela não será atualizada
```

Exemplo

Um "Hello world" diferenciado:

In [ ]:
```
pygame.font.init()

minha_fonte = pygame.font.Font('minha_fonte',50) #cria um objeto "fonte"
com uma determinada fonte .ttf e um determinado tamanho

superf = minha_fonte.render("Hello World", True, (255,255,255)) #renderi
za um texto com antialias na cor branca

superfRect = superf.get_rect()
superfRect.center = ((800), (450))

win.blit(superf, superfRect)
```

3. Conteúdo extra

A biblioteca Pygame tem muitas outras funções não abordadas aqui. A documentação completa pode ser encontrada clicando aqui (https://www.pygame.org/docs/).