# Introdução ao PyGame


Criado por

Psycho Mantys / @psycho_mantys

Sky Cube

# O que é o PyGame?

# O que é o Pygame?


- `Python + Game = PyGame`
- Biblioteca de `python` para aplicações multimedia
- Pode ser usado com motor de desenvolvimento de jogos
- Baseado na `SDL`
- Multiplataforma

# Baseado na SDL


- Como o desenvolvimento do **PySDL** parada, o **PyGame** surgiu
- **SDL** é extremamente portável
- Não é uma copia exata da **API**(ainda bem)

# Multiplataforma

- **Linux**, **Windows** e **MacOS**
- **FreeBSD**, **NetBSD**, **OpenBSD** e **BSD/OS**
- Não oficialmente suportado: **AmigaOS**, **Dreamcast**, **Atari**, **AIX**, **OSF/Tru64**, **RISC OS**, **SymbianOS** e **OS/2**
- E ainda mais...

# Filosofia

*Fazer as coisas simples de maneira fácil e as coisas difíceis de maneira direta*

# Instalação

- Você também pode intalar pelo seu gerenciador de pacotes favorito
- Normalmente em todos os sistemas:

```
pip install pygame --user
```

# Aliens!


```
python3 -m pygame.examples.aliens
```

## Debian like:

```
sudo apt-get install python3-pygame
```

## Fedora/Redhat:

```
sudo yum install python3-pygame
```

# On the Road

- Criar uma pasta para o nosso codigo
- Dentro da pasta, vamos criar o **virtualenv**
- Ativar o **virtualenv**
- Cria um arquivo **requirements.txt**
- Instalar as dependencias com o **pip**
- Baixar arquivos de media no repositorio:

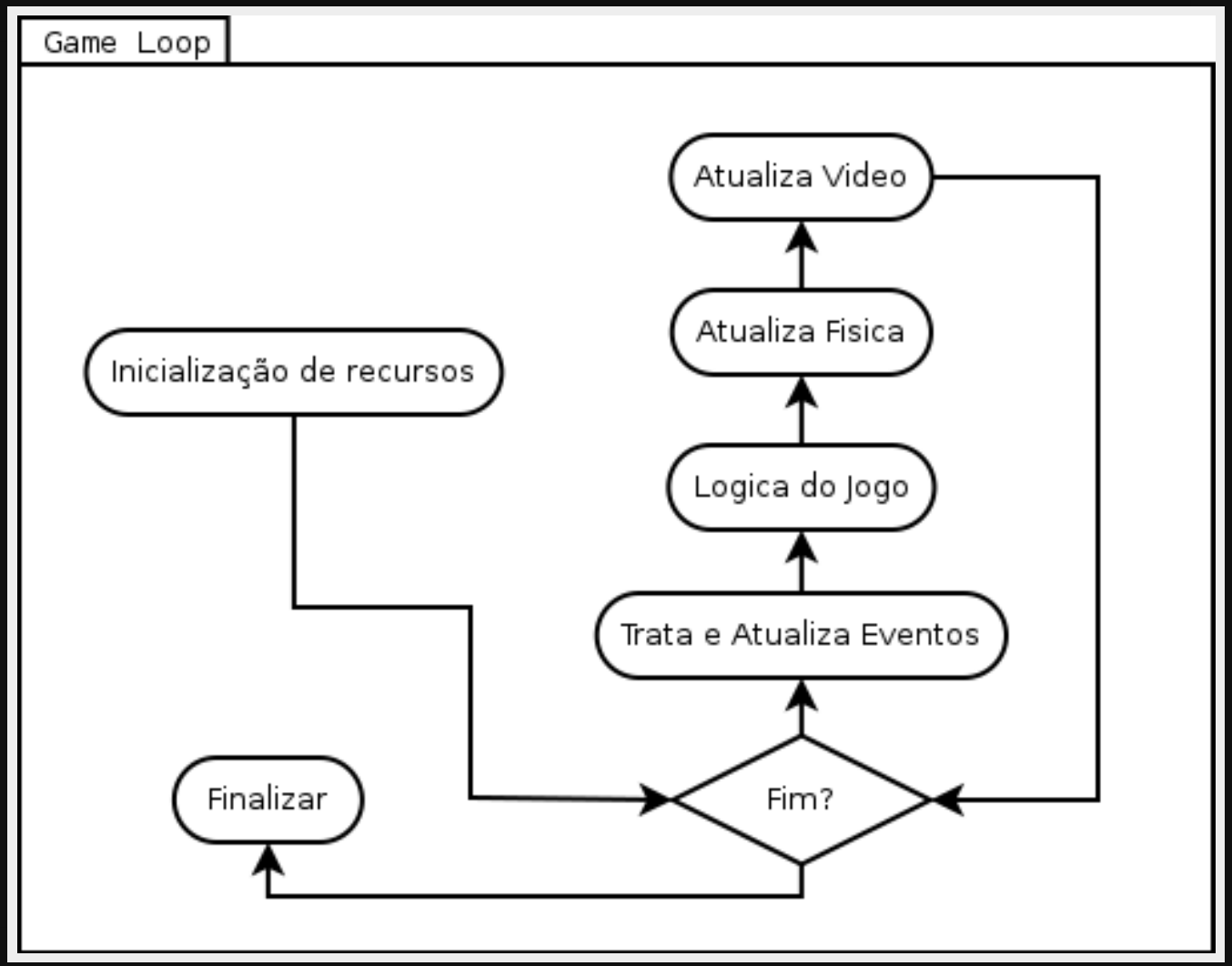
https://github.com/psychomantys/python-day-pygame-tutorial/tree/v1.0/

```
# 1
mkdir frogs
# 2
cd frogs
virtualenv -p python3 user/
# 3
. ./usr/bin/activate
# 4
echo "pygame" > requirements.txt
# 5
pip install -r requirements.txt
# Teste
python -m pygame.examples.aliens
```

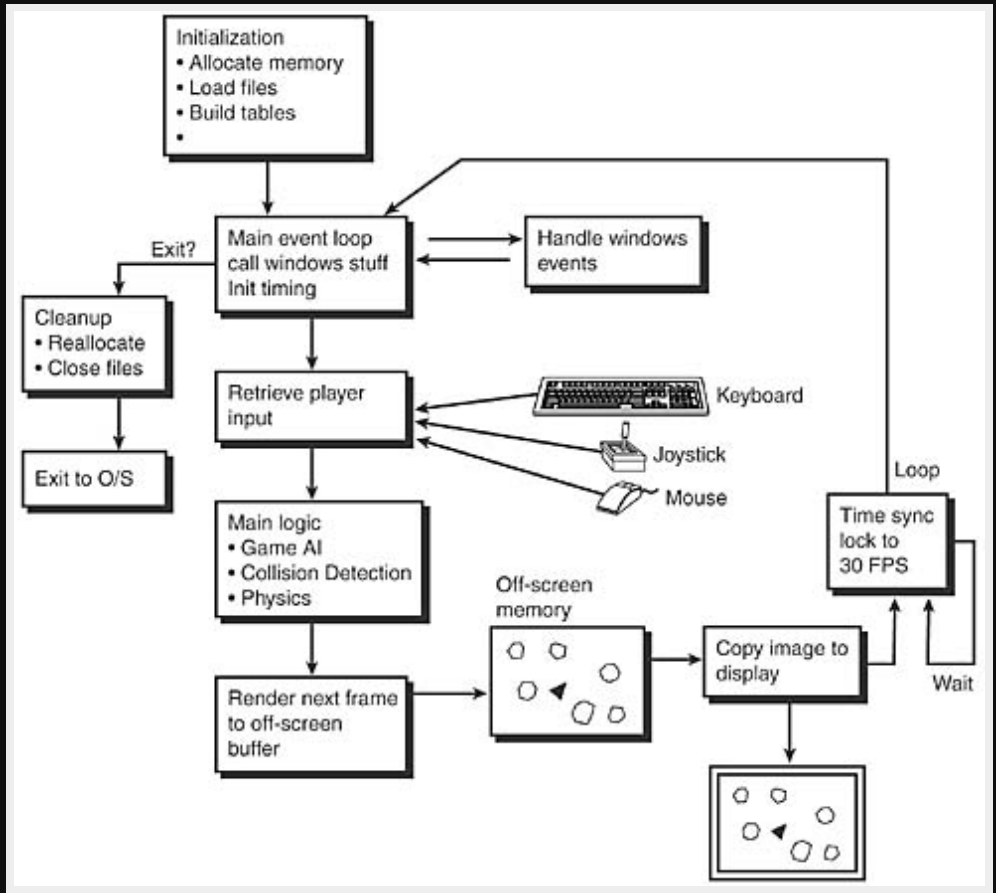
# Game Loop

# Game Loop

- Padrão de projeto usado na essência da construção de jogos
- Uma estrutura de repetição infinita
- Controla o **FPS**
- Incorpora loop de eventos


Game Loop
Atualiza Video
Atualiza Fisica
Inicialização de recursos
Logica do Jogo
Trata e Atualiza Eventos
Finalizar
Fim?

# Game Loop mais detalhado



# Game Loop em python

```
# python3 code/gameloop.py
try:
    game_init()
    while not done:
        event_handler()
        game_logic()
        step_physics()
        render()
finally:
    game_end()
```

# Game Loop OO

```
# python3 code/gameloop_obj.py
class App(object):
   done=False
   def __init__(self):
      pass
   def main_loop(self):
      while not self.done:
         self.event_handler()
         self.game_logic()
         self.step_physics()
         self.render()
   def __del__(self):
      pass
if __name__ == "__main__":
   App().main_loop()
```

# On the Road 2

- Criar o `__init__.py`
- Dentro do arquivo, criar a classe `Frog` com os metodos sem nada:
  - `event_handler()`
  - `game_logic()`
  - `step_physics()`
  - `render()`
- Criar um metodo `main_loop()` e implementar
- Instanciar e rodar `Frog`

# Modulos

| | | |
|---|---|---|
| Som | Teclado | Joystick |
| CDROM | Mouse | Camera |
| Color | Cursor | Display |
| Draw | Event | Font |
| Image | Mask | Mixer |
| Music | Rect | Surface |
| Sprite | Time | Transform |
| BufferProxy | FreeType | GFXDraw |

Muitos modulos a mais disponíveis, alguns desses ainda experimentais

# Inicializando

```
import pygame
# Opcional
from pygame.locals import *

pygame.init()
# ou "pygame.module.init()" Ex:
pygame.display.init()
```

# Terminando

```
pygame.quit()
# ou "pygame.module.quit()" Ex:
pygame.font.quit()
```

# pygame.surface

Modulo que representa a superficies que você usa para desenhar.

É uma tela onde você pode desenhar, formando uma imagem que depois você pode colocar na tela se quiser.

## Metodos uteis

| | |
|---|---|
| `Surface(tamanho)` | Cria uma superficie |
| `s.set_at(point,cor)` | Muda a cor de um ponto |
| `s.fill(cor[, rect])` | Preenche com uma cor um retangulo da superficie |
| `s.blit(surface,posicao)` | Copia a **surface** para **s** em uma **posição** |
| `s=s.convert()` | Converte a profundidade de cor |

# pygame.display

Modulo que ajuda na manipulação da tela.

Pode criar a tela, controlar e configurar os paramentros dela.

## Metodos uteis

| | |
|---|---|
| `flip()` | Renderiza toda a tela |
| `set_caption(caption)` | Muda o titulo da janela |
| `set_mode(mode)` | Configura o tamanho da tela |

# On the same Road 3

## No __init__(self):

```
pygame.init()
self.screen=pygame.display.set_mode((800,400))
pygame.display.set_caption("Python Day")
```

## No __del__(self):

```
pygame.quit()
```

## No render(self):

```
self.screen.fill((0,0,0))
pygame.display.flip()
```

# pygame.time

Modulo para lidar com o tempo, limitar e obter os **FPS**.

## Metodos uteis

| | |
|---|---|
| `tick(fps)` | Atrasa o loop para o programa ter **fps** quadros |
| `get_fps()` | Retorna o numero de fps |

# On the Road 4: The mission

Em `__init__()`:

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

Em `render()`:

```
self.dt=clock.tick(self.fps)
```

# pygame.event

Modulo para tratar eventos do PyGame.

Você pode usar diretamente os submodulos de dispositivos especificos.

## Metodos uteis

| | |
|---|---|
| `get(tipos_evento)` | Retira os eventos que ocorreram de um tipo |
| `poll()` | Retorna apenas um evento da fila |
| `wait()` | Espera ate ter um evento na fila e o retira para tratar |
| `clear(tipo)` | Remove todos os eventos de um tipo da fila |

# Funcionamento basico

```
# python3 code/gameloop.py
def event_handler():
   for event in pygame.event.get():
      # Trata evento QUIT
      if event.type == pygame.QUIT:
         done=True
      elif event.type == pygame.KEYDOWN:
         if event.key==K_ESCAPE:
            done=True
         print(event.key)
```

# On the Road 5: The Origins

## Em `event_handler(self):`

```
for event in pygame.event.get():
   # Trata evento QUIT
   if event.type==pygame.QUIT:
      self.done=True
   elif event.type==pygame.KEYDOWN:
      print(pygame.key.name(event.key))
      print(event.key)
      if event.key==pygame.K_ESCAPE:
         self.done=True
      if event.key==32 or event.key==pygame.K_RIGHT:
         pass
      if event.key==pygame.K_DOWN:
         pass
      if event.key==pygame.K_UP:
         pass
```

# pygame.Rect

Modulo para armazenar e manipular retangulos.

Armazena coordenadas e faz operações com esse retangulo.

A maior parte dos metodos tem uma versão ***in place***.

## Metodos uteis

| | |
|---|---|
| `r=pygame.Rect()` | Cria um retangulo, aceita pares de coordenadas ou as coordenadas diretamente |
| `r.x` | Coordenada *X* do retangulo |
| `r.y` | Coordenada *Y* do retangulo |
| `r.center` | Par de coordenadas do centro do retangulo |
| `r.size` | Tamanho do retangulo |

## Metodos uteis parte 2

| | |
|---|---|
| `r.move(x,y)` | Retorna um retangulo movido pelo ponto ***(x,y)*** |
| `r.move_ip(x,y)` | Mesma coisa do `move()`, mas **in place** |
| `r.clamp_ip(Rect)` | Move o retangulo para ficar dentro de `Rect` |
| `r.collidepoint(x,y)` | Verifica se ***(x,y)*** esta dentro do retangulo |
| `r.colliderect(Rect)` | Verifica se `Rect` esta dentro do retangulo |
| `r.contains(Rect)` | Verifica se `Rect` esta dentro do retangulo |

# Exemplo Rect()

```
# python3 code/gameloop.py
if self.rect.left<0:
    self.speed_x=-self.speed_x
    self.turn()
if self.rect.right>800:
    self.speed_x=-self.speed_x
    self.turn()
if self.rect.top<0:
    self.speed_y=-self.speed_y
if self.rect.bottom>600:
    self.speed_y=-self.speed_y

rect1.right=10
rect2.center=(20,30)
rect.move_ip(x,y)
```

# pygame.image

Modulo para carregar e salvar imagens.

## pygame.image

| | |
|---|---|
| `load(filename)` | Lê uma imagem de um arquivo |
| `save(surface,filename)` | Salva uma surface para um arquivo |

# On the Road 6: Last Chapter

Vamos criar uma função para carregar as imagens mais facil

- Aceitar dois parametros
  - nome do arquivo da imagem
  - Cor de transparencia
- Caso transparencia for `-1`
  - Usa Primeiro pixel da imagem como transparencia
- Caso seja `None`, não usa transparencia

```
def load_image(filename, colorkey=None):
   image=None
   try:
      image=pygame.image.load(filename)
   except pygame.error as err:
      raise SystemExit(str(err))
   image = image.convert()
   if colorkey is not None:
      if colorkey is -1:
         colorkey=image.get_at( (0,0) )
      image.set_colorkey(colorkey)
   return image
```

Para testar:

No `__init__()`:

```
self.car=load_image("media/img/car_01.png",-1)
```

No `render()`:

```
self.screen.blit(self.car, self.car.get_rect())
```

# pygame.sprite

Modulo para representar e facilitar elementos basicos e visuais de um jogo.

Existem varias classes e utilitarios nesse modulo, mas vamos falar de dois basicamente.

## pygame.sprite.Sprite

| | |
|---|---|
| pygame.sprite.Sprite | Classe para representar objetos visiveis do jogo |
| Sprite.rect | Retangulo que representa onde a imagem vai se desenhada |
| Sprite.image | Superficie que vai ser desenhada |
| Sprite.update() | A ser sobrecarregado. Controla comportamento do sprite. |
| Sprite.kill() | Remove o sprite de todos os grupos |

# On the Road 7: The New Blood

- Criar uma classe `tile`
- Herdar de `pygame.sprite.Sprite`
- Implementar o construtor
- Implementar um metodo para desenhar na tela

```
class Tile(pygame.spirte.Sprite)
   def __init__(self, img, x=0, y=0, velocity=(0,0)):
      pygame.sprite.Sprite.__init__(self)
      self.image=load_image(filename,-1)
      self.rect=self.image.get_rect()
      self.rect.x=x
      self.rect.y=y
      self.x=x
      self.y=y
      self.velocity=velocity
   def draw(self, screen):
      return screen.blit(self.image, self.rect)
   def update(self,dt):
      self.rect.x=self.x
      self.rect.y=self.y
```

# Para testar

## No `__init__(self)`:

```
self.player=Tile("media/img/frog.png", 200, 0, (10,0))
```

## No `render(self)`:

```
self.player.draw(self.screen)
```

## No `step_physics(self)`:

```
self.player.update(self.dt)
```

# On the Road 8: New Nightmare

- Vamos extender Tile para dar movimento
- Criar um metodo `tile.move(x,y)`
- Implementar o movimento no `update(self)`

```
    def move(self, x=0, y=0):
        self.rect.move_ip(x, y)
    def dt_x_move(self, dt):
        dx=dt*(self.velocity[0]/1000)
        self.x+=dx
        self.rect.x=self.x
    def dt_y_move(self, dt):
        dy=dt*(self.velocity[1]/1000)
        self.y+=dy
        self.rect.y=self.y
    def dt_move(self, dt):
        self.dt_x_move(dt)
        self.dt_y_move(dt)
    def update(self, dt):
        self.dt_move(dt)
```

## pygame.sprite.Group()

| pygame.sprite.Group() | Gerencia e contem varios sprites |
|---|---|
| g.update(args) | Chama o metodo update() de todos os sprites do grupo |
| g.draw(surface) | Desenha as imagens de todos os sprites do grupo |

# On the Road 9: The Last Final Chapter

- Criar um grupo para representa a camada de cima
- Criar um grupo para representa a camada de baixo
- Na parte da **fisica**, os grupos devem executar update
- No **render**, desenhar o conteudo dos grupos
- No **construtor**, colocar o chão na camada de baixo

## Em __init__(self):

```
    self.layer_front=pygame.sprite.Group()
    self.layer_background=pygame.sprite.Group()
    self.layer_front.add(self.player)
    flip=True
    for x in range(0, self.screen_mode[0], self.tile_size):
        if flip:
            t=Tile("media/img/grass_field.png", x)
        else:
            t=Tile("media/img/road_field.png", x)
        flip=not flip
        self.layer_background.add(t)
```

## Em render(self):

```
    layer_background.draw(self.screen)
    layer_front.draw(self.screen)
```

## Em step_physics(self):

```
    layer_background.update(self.dt)
    layer_front.update(self.dt)
```

# Intermission

# "Vamos dar movimento ao sapo"

```
if event.key==32 or event.key==pygame.K_RIGHT:
   self.player.move(self.tile_size/4)
if event.key==pygame.K_DOWN:
   self.player.move(0,self.tile_size/4)
if event.key==pygame.K_UP:
   self.player.move(0,-self.tile_size/4)
```

# Colisões

Não é bem um modulo, e sim uma parte de **sprite**.

Como é uma parte muito importante e usa muitas coisas do PyGame, fica por ultimo.

## Colisões

Detecta quem colidiu entre um **sprite** com um **grupo**

```
pygame.sprite.spritecollide(sprite,grupo,mate)
```

Detecta quem colidiu entre um **grupo1** com um **grupo2**

```
pygame.sprite.groupcollide(grupo1,grupo2,mate1,mate2)
```

Caso **mate** seja `True`, vai ser executado o metodo `kill()` dos sprites correnspondetes que colidiram.

# On the Road X

- Criar um contador de tempo para colocar carros
- Aleatoriamente, criar carros no topo da pista
- Detectar se já existe um carro onde estamos criando

```
self.car_tick+=self.dt
if self.car_spawn_time<self.car_tick:
    self.car_tick=0
    img="media/img/car_0"+str(randint(1,3))+".png"
    x=(1+2*randint(0,(self.screen_mode[0]/self.tile_size)/2))*(self.tile_size)
    car=tile.Tile(img, x, 0, self.car_base_velocity)
    y=-car.image.get_rect().height
    car.y=y
    while pygame.sprite.spritecollide(car, self.layer_front, False):
        x=(1+2*randint(0,(self.screen_mode[0]/self.tile_size)/2))*(self.tile_size)
        car=tile.Tile(img, x, y, self.car_base_velocity)
    self.layer_front.add(car)
```

# Toques Finais

# Colidir carro com sapo

```
# Verifica se o jogador bateu em algo da layer_front
collisions=pygame.sprite.spritecollide(self.player, self.layer_front, False)
for collision in collisions:
   # Se ele bateu em alquem que não é um sapo, acaba o jogo
   if collision.name!='Frog':
      self.done=True
```

# Remover carros que sairam da tela

```
# Se o carro saiu da tela, remove ele de todos os grupos
for car in self.layer_front:
   if self.screen_mode[1]<car.rect.y:
      car.kill()
```

# Verifica se o sapo passou dos limites

```
if self.screen_mode[0]<self.player.x:
    self.done=True
if self.screen_mode[1]<self.player.y:
    self.done=True
```

# Links Úteis

- Site oficial do PyGame
- Documentação no site oficial
- Aprendendo a programar em python+PyGame
- Passo a passo de um jogo de tabuleiro
- Como fazer sua Fisica

# Game Loop

- Game loop
- Fix Your Timestep!
- deWiTTERS Game Loop

# THE END

Por

Psycho Mantys

Follow Follow @psychomantys