Programação de Jogos em Python

Gustavo Sverzut Barbieri

GPSL - UNICAMP

28 de abril de 2005

Python é fácil!

Linguagem de alto nível bem parecida com o modo que pensamos, ideal para fazer a inteligência de um jogo

```
for elemento in elementos:
    if elemento.tipo in tipos_inimigos:
        atire_no_elemento( elemento )
```

Outras vantagens do Python

- Multi-plataforma
- Orientada a Objetos: encapsulamento
- Dinâmica
- Vasta biblioteca padrão e bibliotecas de terceiros
- Grande comunidade de desenvolvedores
- Fácil de interagir com outras linguagens, em especial C
- Geral o bastante para ajudar em áreas diversas, como:
  - Automatização de processos de desenvolvimento
  - Ajudar que roteiristas e artistas experimentem com diversos efeitos, sem que necessitem de um vasto conhecimento de programação

Problemas e Soluções

Nem tudo são maravilhas, o principal problema é:

Python é lento: Python não é ideal para fazer processamento de imagens, iterações sobre grandes matrizes etc.
Solução: Quando a velocidade começar a ser um problema, analise o seu código em um *profiler* e otimze tal região, talvez implemente-a em C/Assembly.

Na verdade grande parte disso já está feito, o PyGame utiliza a biblioteca SDL, que é feita em C. Existem também implementações de *engines* bem otimzados que são utilizáveis no Python.

PyGame

- Como o Python, é multiplataforma
- Fácil de utilizar
- Possui bastante recursos
- Rápida o suficiente

```
import pygame
pygame.init()
screen = pygame.display.set_mode( ( 800, 600 ) )
image  = pygame.image.load( "minha_imagem.png" )
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT: break
    screen.fill( ( 0, 0, 0 ) )
    screen.blit( image, image.get_rect() )
    pygame.display.flip()
```

Surface — o que é?

`Surface` são superfícies nas quais se desenha. Podem ser 2D ou 3D, residir em memória do sistema ou da placa de vídeo e também ter várias profundidades de cores.

```
tamanho     = ( 640, 480 )
cor         = ( 255, 0, 0 ) # vermelho
superficie = Surface( tamanho )
superficie.set_at( ( 10, 20 ), cor )
superficie.fill( cor, ( 11, 21, 50, 50 ) )

tela = pygame.display.set_mode( tamanho )
```

Surface — operações essenciais

fill(cor, area) Preenche uma área com a cor

get_at(posicao) Pega a cor que está na posição

set_at(posicao, cor) Muda a cor da superfície na posição

blit(origem, posicao, area_origem) Copia a parte limitada por `area_origem` da superfície `origem` para a posição desejada na superfície atual.

Surface — profundidade de cor

Em computação representamos uma superfície de desenho como uma matriz de pontos de cor. Dependendo do número de cores possíveis temos uma profundidade de cor e representamos de maneiras diferentes, algumas delas:

256 cores Este modo representa 256 cores possíveis e é indexado, isto é, temos uma palheta de cores com índices de 0 a 255, cada um com uma cor.

24bpp RGB Também conhecido com 16 milhões de cores. Este modo representa cada cor com 3 bytes (daí o nome *24 bits per pixel*) e no formato RGB utiliza-se o primeiro byte para 256 tons de vermelho, o segundo para verde e o terceiro para o azul.

Surface — transparências

As superfícies suportam 3 tipos de transparência:

Colorkey neste modo uma cor é designada para representar a transparência, isto é, ao fazer o *blit* os pontos com esta cor não serão copiados.

Image Alpha neste modo a superfície como um todo tem um valor de translucidez entre 0 (transparente) e 255 (opaco), ao fazer o *blit* desta superfície em outra cada ponto resultante é uma média da origem com o destino proporcional a este valor. Este modo pode ser utilizado em conjunto com o *colorkey*

Per-Pixel Alpha neste modo cada ponto da superfície tem um componente de Alpha associado que é levado em conta na hora do *blit*. Estas superfícies são chamadas de 32bpp RGBA. Este modo não pode ser utilizado com os dois primeiros.

Surface — convertendo profundidade de cor

Toda vez que se faz um `blit()` convertemos a profundidade de cor da superfície de origem para ficar equivalente à superfície de destino, isto é uma operação muito custosa e nem sempre precisamos fazê-la, pois podemos deixar as superfícies já convertidas.
Eliminando este passo podemos deixar o programa até 6 vezes mais rápido!

```
sup = sup.convert() # converte para a tela
img = pygame.image.load("img.png").convert(sup)
```

Quando não usar: quando se quer utilizar *per-pixel alpha* não converta para a profundidade de cor da tela, pois ele perderá tal recurso, neste caso utilize `convert_alpha()`.

Rect — o que é?

`Rect` representa retângulos, com posição e dimensões. Parece irelevante, mas é uma das classes mais úteis de todo PyGame pois ela fornece operações que são muito utilizadas, facilitando o desenvolvimento do jogo

```
r = Rect( ( 10, 10, 50, 100 ) )
print r.top, r.bottom # 10 110
print r.left, r.right # 10 60
print r.midtop, r.midleft # (35, 10) (10, 60)
print r.center # (35, 60)
c1 = r.collidepoint( 30, 40 )
c2 = r.colliderect( ( 0, 0, 100, 200 ) )
r2 = r.inflate( 10, 10 )
r2.move_ip( 5, 5 )
```

Rect — operações essenciais

clamp(area) retorna um novo retângulo que foi movido para ficar dentro de `area`. Útil para limitar movimento dentro de um espaço.

clip(area) retorna um novo retângulo com a `area` cortada para caber no retângulo atual.

collidepoint(x, y) verifica se o ponto está dentro do retângulo atual.

colliderect(area) verifica se a `area` intercepta o retângulo atual.

contains(area) verifica se a `area` está dentro do retângulo atual.

inflate(x, y) retorna um novo retângulo com as dimensões do retângulo atual aumentadas pelos valores passados.

move(x, y) retorna um novo retângulo com as posições do retângulo atual movida pelos valores passados.

display — o que é?

O módulo *display* é utilizado para manipular a tela.

```
modos = pygame.display.list_modes()
tela  = pygame.display.set_mode( modos[ 0 ] )
rect  = pygame.Rect( 0, 0, 10, 10 )
pygame.display.set_caption( "Teste do PyGame" )
while tela.get_rect().contains( rect ):
    tela.fill( ( 0, 0, 0 ) )
    tela.fill( ( 255, 0, 0 ), rect )
    rect.move_ip( 10, 10 )
    pygame.display.flip()
```

display — operações essenciais

flip() atualiza o conteúdo da tela toda.

get_surface() retorna a superfície que representa a tela.

list_modes() lista possíveis dimensões.

set_caption(titulo) muda o título da janela.

set_mode(tamanho) configura a tela para o dado tamanho e retorna a superfície.

toggle_fullscreen() coloca a janela em tela cheia.

update(lista_retangulos) atualiza as áreas da tela determinadas por `lista_retangulos`.

display — cuidados especiais

- Toda vez que a janela de um aplicativo PyGame for sobreposta por outra janela, a área ficará negra (suja) até que a tela seja atualizada. É comum ao fazer testes no *prompt* cobrir a janela e depois não se vê o resultado da operação, neste caso descubra a janela e faça `pygame.display.flip()`.
- **Atualize somente as áreas necessárias!** Se quiser aumentar o desempenho do seu jogo, atualize a tela somente nas áreas que você modificou. Por exemplo, se você tem um componente em movimento, apague a imagem da posição antiga (provavelmente desenhando o pedaço da imagem de fundo no local), desenha a imagem na posição nova e atualize somente estas duas áreas. Esta técnica é chamada de *Dirty Rectangles*.

draw — o que é?

O PyGame tem vários recursos para desenhar nas superfícies.

```
from pygame.draw import line, circle, polygon
tela = pygame.display.get_surface()
line( tela, ( 255, 0, 0 ), ( 0, 0 ), ( 10, 10 ) )
circle( tela, ( 0, 255, 0 ), ( 20, 20 ), 20, 2 )
rect( tela, ( 0, 0, 255 ), ( 10, 10, 50, 50 ) )
polygon( tela, ( 255, 255, 0 ),
          [ ( 200, 200 ), ( 210, 190 ),
            ( 220, 200 ), ( 210, 210 ),
            ( 220, 220 ), ( 200, 220 ) ] )
pygame.display.flip()
```

draw — operações essenciais

line(superficie, cor, inicio, fim, espessura) desenha uma linha.

circle(superficie, cor, posicao, raio, espessura) desenha um círculo.

rect(superficie, cor, retangulo, espessura) desenha um retângulo.

polygon(superficie, cor, lista_pontos, espessura) desenha um polígono.

image — o que é?

O PyGame tem recursos para ler e salvar imagens. Por padrão o formato *Bitmap(BMP)* é suportado, porém se a biblioteca SDL_Image estiver instalada vários outros formatos serão suportados, dentre eles PNG, GIF, JPEG.

```
if pygame.image.get_extended():
    nave = pygame.image.load( "nave.png" )
else:
    nave = pygame.image.load( "nave.bmp" )
pygame.image.save( tela, "screenshot.bmp" )
```

image — operações essenciais

load(arquivo) lê a imagem do arquivo.

save(superficie, arquivo) salva a superfície em um arquivo.

get_extended() retorna verdadeiro se a biblioteca SDL_Image estiver instalada.

event — o que é?

Módulo que cuida dos eventos dentro do PyGame. Você pode utilizar duas técnicas para trabalhar com eventos:

- Usar a fila de eventos. Todo evento causado (movimento do mouse, tecla pressionada, ...) geram eventos que vão para a fila de eventos a qual você pode consultar e tomar as ações necessárias. A vantagem é que nunca se perde um evento, a desvantagem é que pode adicionar latência na resposta e também precisa-se manter estados para combinar ações (ié: duas teclas simultaneamente).
- Consultar diretamente os dispositivos. A vantagem é que pode verificar vários estados ao mesmo tempo, porém tem a desvantagem de perder eventos (ié: o usuário solta o botão do mouse justamente quando você verifica o estado).

event — usando fila de eventos

```
from pygame.locals import *
for event in pygame.event.get():
    if event.type == QUIT:
        sys.exit()
    elif event.type == KEYDOWN:
        print event.key
```

event — consultando os dispositivos

Nesta técnica você deve utilizar o módulo que trata de cada dispositivo e periodicamente chamar a função `pump()` do sistema de eventos para que seu programa não trave.

```
from pygame.locals import *
while not ( pygame.mouse.get_pressed()[ 0 ] or \
            pygame.key.get_pressed()[ K_SPACE ] ):
   pygame.event.pump()
```

event — operações essenciais

poll() retorna o próximo evento na fila. Caso não exista um evento, será criado um do tipo `NOEVENT`.

post(evento) coloca um evento na fila.

pump() caso não utilize a a fila de eventos você deve chamar freqüentemente esta função para fazer a manutenção do programa, caso contrário ele pode travar.

get(tipos_evento) retorna todos os eventos de um certo tipo na fila.

clear() limpa a fila de eventos.

wait() retorna o próximo evento na fila ou espera até que um novo evento entre, caso ela estiver vazia.

font — o que é?

Módulo para trabalhar com fontes *True Type*. Ele só está habilitado caso você tenha a biblioteca SDL_ttf instalada.

```
tela = pygame.display.set_mode( ( 640, 480 ) )
if pygame.font:
    font = pygame.font.Font( "fonte.ttf", 12 )
    font.set_underline( True )
    img = font.render( "Olá Mundo", True,
                       ( 255, 255, 0 ) )
    tela.blit( img, ( 0, 0 ) )
    pygame.display.flip()
```

font — operações essenciais

Estas são as operações do sub-módulo:

get_fonts() retorna a lista de fontes disponíveis no sistema.

match_font(nome) retorna o caminho para uma fonte que tenha o nome requerido.

get_default_font() retorna o nome do arquivo da fonte padrão do sistema.

Font(arquivo, tamanho) cria uma nova instância de fonte a partir de um arquivo.

SysFont(nome, tamanho) cria uma nova instância de fonte a partir de uma fonte instalada no sistema.

font.Font — operações essenciais

Estas são as operações da classe Font:

render(texto, antialias, cor_frente, cor_fundo) retorna uma superfície com o texto desenhado.

size(texto) calcula qual será o tamanho da superfície necessária para desenha o texto.

set_italic(opcao) habilita ou desabilita o texto em itálico.

set_bold(opcao) habilita ou desabilita o texto em negrito.

set_underline(opcao) habilita ou desabilita sublinhar o texto.

transform — o que é?

Transform é um módulo para auxiliar modificar superfícies, ele tem operações para rotacionar, espelhar, modificar o tamanho e cortar superfícies.

```
img = pygame.image.load( "minha_imagem.png" )
from pygame.transform import flip, scale, rotate
ponta_cabeca = flip( img, False, True )
inclinada    = rotate( img, 45 )

novo_tam     = img.get_rect().inflate( 5, 5 )
aumentada    = scale( img, novo_tam.size )
```

transform — operações essenciais

flip(superficie, x, y) espelha a imagem horizontal e verticalmente, dependendo se x e y forem verdadeiros ou falsos.

rotate(superficie, angulo) rotaciona a imagem

scale(superficie, tamanho) aumenta ou diminui uma imagem.

mixer — o que é?

Módulo para trabalhar com sons e canais de reprodução. Por padrão existem 8 canais para reprodução simultânea.

```
from pygame.locals import *
musica = pygame.mixer.Sound( "musica_fundo.wav" )
efeito = pygame.mixer.Sound( "efeito.wav" )
aviso  = pygame.mixer.Sound( "aviso.wav" )
musica.play( -1 )
while True:
    for e in pygame.event.get( [ KEYDOWN ] ):
        if    e.key == K_SPACE:
            efeito.play()
        elif e.key == K_ESCAPE:
            musica.fadeout( 1500 )
            aviso.play()
```

mixer — operações essenciais

Sound(arquivo) cria uma nova instância de objeto de som.

fadeout(tempo) demora o tempo requerido para deixar todos os canais mudos, o volume vai abaixando gradualmente.

pause() pára temporariamente a reprodução de todos os canais.

stop() pára a reprodução de todos os canais.

unpause() restaura a reprodução dos canais parados.

mixer.Sound — operações essenciais

Estas são as operações essenciais da classe Sound:

fadeout(tempo) demora o tempo requerido para deixar o canal que reproduz este som mudo, o volume vai abaixando gradualmente.

get_length() retorna quantos segundos tem este som.

get_volume() retorna o valor da altura do som.

play(repeticoes) inicia a reprodução do som.

set_volume(valor) configura a altura do som, de 0,0 a 1,0.

stop() pára a reprodução do som.

Clock — o que é?

Classe para trabalhar com tempo, atrasar a execução (e consequentemente limitar os quadros por segundo) e também obter tempo entre os quadros.

```
clock = pygame.time.Clock()
FPS   = 60
while True:
   clock.tick( FPS )
   print "Quadros por segundo:", clock.get_fps()
```

A operação mais usada é `tick( atraso )` que serve para manter uma taxa de quadros por segundo.

sprite — O que é?

Sprite é uma imagem bi-dimensional que faz parte de uma cena maior, isto é, os componentes que aparecem no jogo.
O PyGame traz um módulo com vários utilitários para trabalhar com *Sprites* e tornar o desenvolvimento muito mais fácil. As classes básicas são:

Sprite deve ser herdada pelos componentes do seu jogo.

Group serve para agrupar *Sprites*, existem especializações desta classe para ajudar com tarefas rotineiras.

sprite.Sprite — o que é?

`Sprite` é a classe básica que deve ser especializada pelos componentes do seu jogo. Ela implementa métodos necessários pelas classes de grupo e mantém a lista de grupos a qual pertence. Um `Sprite` só é considerado "vivo" se está dentro de um grupo.

sprite.Sprite — operações essenciais

add(lista_grupos) adiciona o Sprite à lista de grupos

alive() retorna verdadeiro se está vivo, isto é, dentro de algum grupo

groups() retorna a lista de grupos em que este Sprite está.

kill() remove o Sprite de todos os grupos

remove(lista_grupos) remove o Sprite da lista de grupos.

update() função que não faz nada, deve ser modificada pelas classes especializadas para atualizar o *sprite* (movimentar, modificar a imagem, ...).

sprite.Group — o que é?

`Group` é a classe básica que contém *sprites*, existem várias especializações:

GroupSingle mantém apenas o último *sprite* adicionado.

RenderUpdates implementa o método `draw( superficie )` que desenha todos os *sprites* do grupo na superfície e retorna a lista de áreas modificadas, então podemos utilizar a técnica de *Dirty Rectangles* apenas fazendo: `pygame.display.update( meugrupo.draw( tela ) )`.

OrderedUpdates especialização de RenderUpdates, mas faz as atualizações na ordem em que os *sprites* foram adicionados.

sprite.Group — operações essenciais

add(lista_sprites) adiciona a lista de *Sprites* ao grupo.

clear(superfice, fundo) limpa a superfície, para isto copia os pedaços que foram "sujos" anteriormente do fundo.

empty() remove todos os *Sprites* do grupo.

has(sprite) verifica se o *Sprite* está no grupo.

remove(lista_sprites) remove a lista de *Sprites* do grupo.

sprites() retorna os *Sprites* contidos neste grupo.

update(*args) chama `update( *args )` em todos os *Sprites* do grupo.

Exemplo: Movimentando uma Bola

```
#!/usr/bin/env python

from copy import copy
import pygame
from pygame.locals import *
from pygame.sprite import Sprite, RenderUpdates
```

```
class Bola( Sprite ):
  def __init__( self, pos, *grupos ):
    Sprite.__init__( self, *grupos )
    self.rect = Rect( 0, 0, 100, 100 )
    self.rect.center = pos
    # Cria imagem
    r = self.rect
    self.image = pygame.Surface( r.size )
    self.image.set_colorkey( ( 0, 0, 0 ) )
    self.image.fill( ( 0, 0, 0 ) )
    pygame.draw.circle( self.image,
                        ( 255, 255, 255 ),
                        ( r.width / 2,
                          r.height / 2 ),
                          r.width / 2 )
  # __init__()
  def move( self, x, y ):
    self.rect.move_ip( x, y )
  # move()
```

```
# Configurações iniciais
pygame.init()
tela  = pygame.display.set_mode( ( 640, 480 ) )
grupo = RenderUpdates()
bola  = Bola( ( 0, 0 ), grupo )
clock = pygame.time.Clock()

fundo = pygame.Surface( tela.get_size() )
fundo.fill( ( 0, 0, 255 ) )
tela.blit( fundo, ( 0, 0 ) )
pygame.display.flip()

key = { K_UP: False, K_DOWN: False,
        K_LEFT: False, K_RIGHT: False }
```

```
# Laço principal
while True:
  clock.tick( 24 )
  # Trata eventos
  for e in pygame.event.get( [ KEYUP, KEYDOWN ] ):
    valor = ( e.type == KEYDOWN )
    if   e.key == K_ESCAPE:
      raise SystemExit, "Fim."
    elif e.key in key.keys():
      key[ e.key ] = valor
  # Movimenta a bola de acordo com as teclas
  if key[ K_UP ]:    bola.move(   0, -10 )
  if key[ K_DOWN ]:  bola.move(   0,  10 )
  if key[ K_LEFT ]:  bola.move( -10,   0 )
  if key[ K_RIGHT ]: bola.move(  10,   0 )

  grupo.clear( tela, fundo )
  pygame.display.update( grupo.draw( tela ) )
```

Detectando colisões

Um dos recursos mais utilizados no desenvolvimento de jogos é detectar colisões e o PyGame torna esta tarefa fácil com as funções:

spritecollide(sprite, grupo, mate) detecta a colisão do *Sprite* com os elementos do grupo. Caso `mate` for verdadeira executa o método `kill()` dos *sprites* que colidiram. Esta função retorna uma lista com os elementos afetados.

groupcollide(grupo1, grupo2, mate1, mate2) detecta a colisão dos *Sprites* do `grupo1` contra os do `grupo2` e executa o método `kill()` nos elementos baseado nos valores dos parâmetros `mate1` e `mate2`. Esta função retorna um dicionário no qual as chaves são elementos do `grupo1` e os valores são listas de elementos do `grupo2` com os quais eles colidiram.

Tutorial e Exemplos

- Em `http://www.gustavobarbieri.com.br/jogos/` existem alguns códigos de jogos simples porém funcionais que podem ser utilizados como base de novos projetos.
- O tutorial passo a passo de como planejar e construir um jogo pode ser encontrado em: `http://www.gustavobarbieri.com.br/jogos/jogo/doc/`. Ele aborda como separar os elementos do jogo e com isso conseguir uma plataforma fácil de manter e extender.

Referências

- Python: `http://www.python.org/`
- PyGame: `http://www.pygame.org/`
- A Newbie Guide to pygame: `http://www.pygame.org/docs/tut/newbieguide.html`
- Introdução ao Pygame: `http://www.pygame.org/docs/tut/intro/intro.html`
- Dicas de Performance para Python: `http://www.python.org/moin/PythonSpeed/PerformanceTips`
- Introdução ao Módulo Sprite do PyGame: `http://www.pygame.org/docs/tut/SpriteIntro.html`

Contato

Gustavo Sverzut Barbieri

Email:	barbieri@gmail.com
Website:	`http://www.gustavobarbieri.com.br`
ICQ:	17249123
MSN:	barbieri@gmail.com
Jabber:	gsbarbieri@jabber.org

Obtenha esta palestra em:
`http://palestras.gustavobarbieri.com.br/pygame/`