Programação de Jogos em Python

Exemplo Prático - Resolvedor de Labirintos

Gustavo Sverzut Barbieri

GPSL - UNICAMP

28 de abril de 2005

Esta apresentação é uma explicação mais detalhada sobre o código que pode ser encontrado em:
`http://www.gustavobarbieri.com.br/labirinto_grafico.py`

Nem todas as partes do código original estão incluídas aqui.

Importando módulos necessários

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

- Incluímos todos os símbolos de pygame.locals para o espaço de nomes (*namespace*) atual.
- Também importamos o Sprite e RenderUpdates para economizar digitação.

Função para desenhar retângulos

```
def desenha_quadrado( tamanho, cor ):
    img = pygame.Surface( tamanho )

    cinza = Color( "#808080" )
    cor1 = pygame.color.add( cor, cinza )
    cor2 = pygame.color.subtract( cor, cinza )

    r = Rect( ( 0, 0 ), tamanho )
    r.inflate_ip( -2, -2 )
    r.topleft = ( 1, 1 )

    img.fill( cor )

    line = pygame.draw.line
    line( img, cor1, r.topleft, r.topright )
    line( img, cor1, r.topleft, r.bottomleft )
    line( img, cor2, r.bottomleft, r.bottomright )
    line( img, cor2, r.bottomright, r.topright )

    return img
# desenha_quadrado()
```

Objeto do jogo "Quadrado"

```
class Quadrado( Sprite ):
    grupos = None # Grupos ao qual este sprite pertence
                  # depois esta variavel de classe sera atribuida
                  # para lembrar dos objetos nao desenhados
    tamanho = ( 50, 50 )
    def __init__( self, pos=( 0, 0 ) ):
        Sprite.__init__( self, self.grupos )
        self.rect = Rect( pos, self.tamanho )
    # __init__()
# Quadrado
```

- **Nunca se esqueça de chamar** `Sprite.__init__`**!**
- `Sprite.__init__` pode receber uma lista de grupos a qual este Sprite pertence. Nós usaremos o atributo de **classe**[1], que será atribuído mais tarde, assim todos os Sprites se adicionam automaticamente ao grupo correto.

[1] Apesar de usarmos `self.grupos`, isso foi feito devido ao método de resolução de atributos do Python, primeiro procura-se na instância, depois nas classes que compõe a instância.

Objetos do jogo “Parede”, “Vazio”, ...

```
class Parede( Quadrado ):            pass
class Vazio( Quadrado ):             pass
class Caminho( Quadrado ):           pass
class Entrada( Quadrado ):           pass
class Saida( Quadrado ):             pass
class CaminhoErrado( Caminho ):      pass
class CaminhoCerto( Caminho ):       pass
class CaminhoPercorrido( Caminho ): pass
```

- Usa-se uma hierarquia de classes para facilitar mais tarde: Um caminho errado ainda é um caminho, um caminho certo também.

Lendo imagens para o labirinto

```
    def __le_imagens( self ):
        """Lê as imagens para cada tipo de peça.

        Usa-se variavel de classe para evitar que cada objeto tenha uma copia
        da mesma imagem, economizando memoria.
        """
        t = self.tam_peca

        if t is None:
            raise Exception( "Você deve usar __arruma_posicoes() primeiro!" )

        Quadrado.tamanho = t

        # Lê imagens:
        Parede.image             = desenha_quadrado( t, Color( "gray35" ) )
        Caminho.image            = desenha_quadrado( t, Color( "wheat" ) )
        Entrada.image            = desenha_quadrado( t, Color( "magenta" ) )
        Saida.image              = desenha_quadrado( t, Color( "green" ) )
        CaminhoCerto.image       = desenha_quadrado( t, Color( "cyan" ) )
        CaminhoErrado.image      = desenha_quadrado( t, Color( "red" ) )
        CaminhoPercorrido.image = desenha_quadrado( t, Color( "yellow" ) )
        Vazio.image              = pygame.Surface( t )
        Vazio.image.set_colorkey( Color( "black" ) )
        Vazio.image.fill( Color( "black" ) )
    # __le_imagens()
```

As atribuições de imagens são feitas para a **classe** pois...

Lendo imagens para o labirinto

- Usamos variáveis de classe para evitar que cada instância carrege uma cópia, economizando memória
- Se precisarmos mudar de todas as instâncias, é só mudar esta variável de classe.
- Se quisermos que uma certa instância tenha uma imagem diferente, é só atribuir à instância e este valor se sobrepõe ao da classe na resolução dos atributos. (Maior Flexibilidade)

Desenhando o Labirinto

```
    def desenhe( self, tudo=False ):
        tela = self.tela
        nao_desenhados = self.nao_desenhados
        if tudo:
            for l in self.mapa:
                for p in l:
                    tela.blit( p.image, p )
        else:
            nao_desenhados.draw( self.tela )

        nao_desenhados.empty()
    # desenhe()
```

- O uso de sprites e grupos facilita operações como desenhar apenas o que mudou.
- Uma instância de Quadrado, quando criada, vai automaticame para o grupo de não desenhados, pois atribuímos este grupo a `Quadrado.grupos`.
- Como sempre desenhamos todos os quadrados não desenhados, após a operação esvaziamos o grupo.

O Jogo

```
class Jogo( object ):
    FPS = 24
    RESOLVE_PASSOS_POR_SEG = 0

    def __init__( self, mapa, tamanho=( 800, 600 ) ):
        self.clock = pygame.time.Clock()
        self.tela  = pygame.display.set_mode( tamanho )
        self.mapa  = mapa
        self.le_labirinto()
    # __init__()


    def le_labirinto( self ):
        self.labirinto = Labirinto( self.tela, self.mapa )
    # le_labirinto()
```

O Jogo: Parando e Atualizando a tela

```
    def termine( self ):
        raise StopIteration
    # termine()


    def atualize( self, tudo=False ):
        if tudo:
            pygame.display.flip()
        else:
            pygame.display.update()
    # atualize()
```

- Note que devido à falta de argumentos para `pygame.display.update()`, este tem o mesmo efeito que `pygame.display.flip()`: atualiza a tela inteira.
- Em uma implementação futura poderíamos lembrar as áreas que mudaram, resultantes do `nao_desenhados.draw()` e passar como argumento para o `pygame.display.update()`, economizando ciclos do processador.

Tratando Eventos

```
    def trata_eventos( self ):
        for e in pygame.event.get( [ KEYDOWN, QUIT, ACTIVEEVENT ] ):
            if   e.type == QUIT:
                self.termine()

            elif e.type == KEYDOWN:
                if   e.key == K_F2:
                    self.le_labirinto()

                elif e.key == K_F3:
                    def callback( estagio ):
                        pass # removido para caber no slide
                    # callback()
                    self.labirinto.resolve( callback )
                    self.labirinto.desenhe()
                    self.atualize()

                elif e.key == K_ESCAPE:
                    self.termine()

            elif e.type == ACTIVEEVENT:
                self.labirinto.desenhe( True )
                self.atualize( True )
    # trata_eventos()
```

O Jogo: Laço Principal

```
    def rode( self ):
        try:
            # ActiveEvent faz desenhar a tela de novo
            pygame.event.post( pygame.event.Event( ACTIVEEVENT ) )
            while True:
                self.clock.tick( self.FPS )
                self.trata_eventos()
                #self.labirinto.desenhe()
                self.atualize()
        except StopIteration:
            return
    # rode()
# Jogo
```

E assim se faz um jogo...

Dúvidas???

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/`