Pygame

Diogo Fernando Trevisan

Abril de 2021

1 Introdução

PyGame é uma biblioteca (módulo) para jogos 2D com um *binding* para Python. Ela é baseada na biblioteca C SDL, que permite criar janelas, desenhar imagens, capturar entrada de dispositivos (mouse, teclado, joystick) e reproduzir sons. Um código base para utilizar o pygame é definir um tamanho para uma janela, um título para ela e entrar no *loop* principal, que, processa eventos, atualiza o estado do game e desenha.

O código abaixo mostra isso. O tamanho e título da janela são definidos com *pygame.display*, e o *loop* é a repetição *while* que apresenta o desenho (pygame.display.flip) e processa os eventos verificando se é para sair.

```
1  import pygame
2  pygame.init()
3
4  tamanhoJanela = [ 800, 600]
5  janela = pygame.display.set_mode(tamanhoJanela)
6  pygame.display.set_caption("Titulo da Janela")
7
8  sair = False
9  while not sair:
10     for event in pygame.event.get():
11         if event.type == pygame.QUIT:
```

```
            sair = True
    pygame.display.flip()

pygame.quit()
```

code/sec1/code1.py

A janela permite definir uma cor de "limpeza" e pintar todo seu fundo. As cores são definidas como tuplas com as três componentes RGB. O trecho abaixo mostra como preencher a cor de fundo da janela com a cor vermelha.

```
import pygame
pygame.init()

tamanhoJanela = [ 800, 600]
janela = pygame.display.set_mode(tamanhoJanela)
pygame.display.set_caption("Titulo da Janela")

sair = False
#definindo a cor
VERMELHO = (255,0,0)
while not sair:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sair = True

    #limpeza
    janela.fill(VERMELHO)
    #atualização dos estados...

    #desenho

    pygame.display.flip()

pygame.quit()
```

code/sec1/code2.py

1.1 Desenhando Primitivas

O pygame permite desenhar algumas primitivas facilmente. O submódulo *pygame.draw* já contém código para desenhar retângulos, círculo, elipse, arco, linha e polígonos definidos por vértices.

Para desenhar um retângulo, é utilizada a função *rect*. Esta pode receber vários parâmetros, sendo obrigatórios a surface (destino, onde será desenhado), a cor e o rect; rect é uma lista com 4 posições, informando a posição X,Y, largura, altura do retângulo.

```
rect(surface, color, rect, width=0)
```

O código abaixo mostra um exemplo que desenha um retângulo (quadrado).

```
1  import pygame
2  pygame.init()
3
4  tamanhoJanela = [ 800, 600]
5  janela = pygame.display.set_mode(tamanhoJanela)
6  pygame.display.set_caption("Titulo da Janela")
7
8  sair = False
9  #definindo a cor
10 VERMELHO = (255,0,0)
11 while not sair:
12     for event in pygame.event.get():
13         if event.type == pygame.QUIT:
14             sair = True
15
16     #limpeza
17     janela.fill(VERMELHO)
18     #atualização dos estados...
19
20     #desenho
21     rect = [10, 10, 30, 30]
22     r_cor = (255,255,255)
```

```
23      pygame.draw.rect(janela, r_cor, rect)
24
25      #com borda
26      #pygame.draw.rect(janela, r_cor, rect,1)
27
28      pygame.display.flip()
29
30  pygame.quit()
```

code/sec1/code3.py

Para desenhar um círculo, deve ser fornecida uma cor, a posição central do círculo e o tamanho do raio. Ainda, um último parâmetro opcional informa a largura da borda, que, se for 0 desenha um círculo preenchido, e, caso seja outro valor, desenha um círculo não preenchido com borda. O código abaixo mostra um exemplo de desenho de círculo usando pygame.

```
1  import pygame
2  pygame.init()
3  tamanhoJanela = [ 800, 600]
4  janela = pygame.display.set_mode(tamanhoJanela)
5  pygame.display.set_caption("Titulo da Janela")
6
7  sair = False
8  #definindo a cor
9  BRANCO = (255,255,255)
10 while not sair:
11     for event in pygame.event.get():
12         if event.type == pygame.QUIT:
13             sair = True
14     #limpeza
15     janela.fill(BRANCO)
16     #atualização dos estados...
17     #desenho
18     c_cor = (255,0,0)
19     pygame.draw.circle(janela, c_cor, (400,300), 200, 0)
20     pygame.display.flip()
```

```
pygame.quit()
```

code/sec1/code4.py

O código abaixo mostra mais 3 primitivas que podemos desenhar: elipse, linha e arco.

```
import pygame
pygame.init()
import math
tamanhoJanela = [ 800, 600]
janela = pygame.display.set_mode(tamanhoJanela)
pygame.display.set_caption("Titulo da Janela")

sair = False
#definindo a cor
BRANCO = (255,255,255)
while not sair:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sair = True
    #limpeza
    janela.fill(BRANCO)
    #atualização dos estados...
    #desenho

    #desenha uma elipse
    e_cor = (255,0,0)
    #surface, cor, RECT
    pygame.draw.ellipse(janela, e_cor,[350,300,100,50])

    #desenhar uma linha
    l_cor =(127,127,0)
    #suface, cor, pos inicial, pos final, largura
    pygame.draw.line(janela, l_cor, [10,10], [750, 550], 10)

    #desenhar um arco. math.pi = 180
```

```
31      a_cor = (255,0,255)
32      #surface, cor, RECT, angulo inicial, angulo final
33      pygame.draw.arc(janela, a_cor, [20,200,100,100], math.pi, math.
        pi *2, 4)
34      pygame.display.flip()
35  pygame.quit()
```

code/sec1/code5.py

1.2 Escrevendo Texto

O pygame permite desenhar textos em uma tela. O texto é renderizado em uma imagem temporária (como se fosse lida de um arquivo), e, essa imagem pode ser desenhada em outra, como a tela. Para o pygame, a tela é uma imagem, assim como texto e outras imagens lidas de arquivo (PNG, Bitmap). O código abaixo mostra como configurar uma fonte (padrão), escrever um texto em uma imagem temporária, e, desenhar essa imagem na janela.

```
1  import pygame
2  pygame.init()
3  import math
4  tamanhoJanela = [ 800, 600]
5  janela = pygame.display.set_mode(tamanhoJanela)
6  pygame.display.set_caption("Titulo da Janela")
7
8  sair = False
9  #definindo a cor
10 BRANCO = (255,255,255)
11
12 #cria uma fonte
13 fonte = pygame.font.Font(None, 24)
14
15 while not sair:
16     for event in pygame.event.get():
17         if event.type == pygame.QUIT:
18             sair = True
```

```
19      #limpeza
20      janela.fill(BRANCO)
21      #atualização dos estados...
22      #desenho
23
24      t_cor = (0,127,50)
25      texto = fonte.render("Meu Texto", True, t_cor)
26      #blit desenha uma imagem em outra.
27      #A janela é uma imagem para o pygame. o texto também.
28      janela.blit(texto, [50,50])
29      pygame.display.flip()
30  pygame.quit()
```

code/sec1/code6.py

1.3 Desenhando Imagens

Pygame permite carregar arquivos de imagem e desenha-los na tela. Para isso é utilizada a função *pygame.image.load* que retorna a referência para um objeto *Surface*. Este objeto, uma imagem, pode ser desenhado na tela com o método blit. Também é possível desenhar partes da imagem, "recortando-a' com o método *subsurface()*. O código abaixo mostra como carregar e desenhar uma imagem PNG carregada de um arquivo.

```
1  import pygame
2  pygame.init()
3  import math
4  tamanhoJanela = [ 800, 600]
5  janela = pygame.display.set_mode(tamanhoJanela)
6  pygame.display.set_caption("Titulo da Janela")
7
8  sair = False
9  #definindo a cor
10 BRANCO = (255,255,255)
11
12 #carrega uma imagem de um arquivo
```

```
imagem = pygame.image.load("raptor.png")
while not sair:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sair = True
    #limpeza
    janela.fill(BRANCO)
    #atualização dos estados...
    #desenho

    #desenha imagem
    janela.blit(imagem,[20,20])

    #desenhar uma parte da imagem: subsurface(inicioX, inicioY,
    largura, altura)
    janela.blit(imagem.subsurface(10,10,50,50),[100,100])
    pygame.display.flip()
pygame.quit()
```

code/sec1/code7.py

1.4 Transformações

Para redimensionar e outras operações com as imagens (*surfaces*), pygame possui um pacote especial: pygame.transform. O código abaixo mostra como rotacionar uma imagem.

```
import pygame
pygame.init()
import math
tamanhoJanela = [ 800, 600]
janela = pygame.display.set_mode(tamanhoJanela)
pygame.display.set_caption("Titulo da Janela")

sair = False
#definindo a cor
```

```
10 BRANCO = (255,255,255)
11
12 #carrega uma imagem de um arquivo
13 imagem = pygame.image.load("raptor.png")
14 while not sair:
15     for event in pygame.event.get():
16         if event.type == pygame.QUIT:
17             sair = True
18     #limpeza
19     janela.fill(BRANCO)
20     #atualização dos estados...
21     #desenho
22
23     #desenha imagem
24     janela.blit(imagem,[20,20])
25
26     #new recebe a 'imagem' rotacionada em 45deg
27     new = pygame.transform.rotate(imagem, 45)
28     #desenha a imagem rotacionada
29     janela.blit(new,[100,100])
30     pygame.display.flip()
31 pygame.quit()
```

code/sec1/code8.py

1.5 Exercício

Desenhe, somente com as primitivas, a Figura 1.

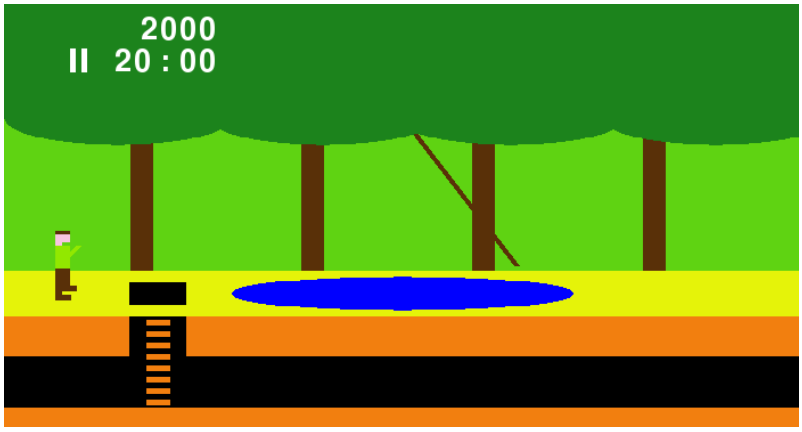

Figura 1: Captura de tela do jogo Piftall do Atari 2600.

1.6 Referências

Para mais informações e exemplos de desenho de primitivas com pygame, visite o endereço http://www.pygame.org/docs/ref/draw.html. Para ver mais métodos sobre fontes, visite https://www.pygame.org/docs/ref/font.html. Para outras transformações, visite https://www.pygame.org/docs/ref/transform.html.

2 Input - Mouse e Teclado

As entradas de mouse, teclado e Joystick são capturadas pelo PyGame e armazenadas. Podemos acessar estes eventos com *pygame.event.get()*, que retorna todos eventos. Então, iteramos pelos eventos e tratamos cada um.

Até agora vimos um evento: QUIT. Este evento é ativado quando fechamos a janela da pygame (clicamos no x). Cada evento é um objeto que é criado automaticamente pela PyGame e temos que tratá-lo. Também podemos criar nossos próprios eventos. Abaixo, o código que toda aplicação que usa PyGame acaba tendo, que varre os eventos (for event...) e verifica se o type é igual a QUIT.

```
for event in pygame.event.get():
    if event.type == pygame.QUIT:
```

Além do evento QUIT, existem outros tipos de eventos, que podem ser vistos em https://www.pygame.org/docs/ref/event.html. Para nós, o que importa são os eventos de teclado e mouse.

2.1 Eventos de Teclado

Os eventos de teclado ocorrem em dois momentos: ao pressionar uma tecla e ao soltar uma tecla. O código mais simples é mostrado abaixo. Nele, é verificado se o evento é do tipo KEYDOWN (tecla pressionada) e o evento é impresso. Nos dados mostrados, é possível verificar o caractere e também o código da tecla.

```
1 import pygame
2 pygame.init()
3 import math
4 tamanhoJanela = [ 800, 600]
5 janela = pygame.display.set_mode(tamanhoJanela)
6 pygame.display.set_caption("Eventos de Teclado")
7
8 sair = False
9 #definindo a cor
```

```
BRANCO = (255,255,255)
while not sair:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sair = True
        elif event.type == pygame.KEYDOWN:
            print(event)

    #limpeza
    janela.fill(BRANCO)
    #atualização dos estados...
    #desenho
    pygame.display.flip()
pygame.quit()
```

code/sec2/code1.py

Como python tem uma tipagem fraca, o evento pode ser de qualquer tipo, tendo atributos variados. Para eventos de tecla, um atributo é *key*, que indica qual tecla foi pressionada. O pygame possui algumas constantes para cada tecla, que podem ser comparadas com o atributo do evento, para verificar se uma tecla específica foi pressionada. O código abaixo verifica se a tecla "ESC" foi pressionada, e, finaliza a aplicação. Para verificar as constantes para outras teclas, veja https://www.pygame.org/docs/ref/key.html.

```
import pygame
pygame.init()
import math
tamanhoJanela = [ 800, 600]
janela = pygame.display.set_mode(tamanhoJanela)
pygame.display.set_caption("Eventos de Teclado")

sair = False
#definindo a cor
BRANCO = (255,255,255)

```

```
raptor = {
    "imagem": pygame.image.load('raptor.png'),
    "posicao": [10,10]
}

while not sair:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sair = True
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_ESCAPE:
                sair = True

    #limpeza
    janela.fill(BRANCO)
    #atualização dos estados...
    #desenho
    pygame.display.flip()
pygame.quit()
```

code/sec2/code2.py

O código abaixo verifica se a tecla pressionada foi a seta direita (K_RIGHT). Caso tenha sido, o raptor tem a posição incrementada em 2 pixels.

```
import pygame
pygame.init()
import math
tamanhoJanela = [ 800, 600]
janela = pygame.display.set_mode(tamanhoJanela)
pygame.display.set_caption("Eventos de Teclado")

sair = False
#definindo a cor
BRANCO = (255,255,255)

raptor = {
```

```
13      "imagem": pygame.image.load('raptor.png'),
14      "posicao": [10,10]
15  }
16
17  while not sair:
18      for event in pygame.event.get():
19          if event.type == pygame.QUIT:
20              sair = True
21          elif event.type == pygame.KEYDOWN:
22              if event.key == pygame.K_ESCAPE:
23                  sair = True
24              elif event.key == pygame.K_RIGHT:
25                  raptor['posicao'][0]+= 2
26
27      #limpeza
28      janela.fill(BRANCO)
29      #atualização dos estados...
30      #desenho
31      janela.blit(raptor['imagem'],raptor['posicao'])
32      pygame.display.flip()
33  pygame.quit()
```

code/sec2/code3.py

Percebe-se, com o código anterior, que, a cada vez que se pressiona a a tecla, o raptor se movimenta um pouco para a direita. Mas isto não é desejado; o ideal é que enquanto a tecla estiver pressionada, o raptor se mova para a direita. Para isso, usamos uma *flag* que fica ativa quando a tecla é pressionada, e, desativada quando a tecla é solta. Um evento do tipo *KEYUP* é lançado quando uma tecla é solta. O código abaixo verifica quando a seta direita é pressionada e solta e seta a *flag* que faz com que o raptor se mova.

```
1  import pygame
2  pygame.init()
3  import math
4  tamanhoJanela = [ 800, 600]
```

```
janela = pygame.display.set_mode(tamanhoJanela)
pygame.display.set_caption("Eventos de Teclado")

sair = False
#definindo a cor
BRANCO = (255,255,255)

raptor = {
    "imagem": pygame.image.load('raptor.png'),
    "posicao": [10,10],
    "andaDireita": False
}

while not sair:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sair = True
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_ESCAPE:
                sair = True
            elif event.key == pygame.K_RIGHT:
                raptor['andaDireita'] = True
        elif event.type == pygame.KEYUP:
            if event.key == pygame.K_RIGHT:
                raptor['andaDireita'] = False
    #limpeza
    janela.fill(BRANCO)
    #atualização dos estados...
    if raptor['andaDireita']:
        raptor['posicao'][0]+= 2
    #desenho
    janela.blit(raptor['imagem'],raptor['posicao'])
    pygame.display.flip()
pygame.quit()
```

code/sec2/code4.py

2.2 Mouse

Os eventos de mouse ocorrem, principalmente, quando um botão é pressionado e quanto um botão é solto. O evento é armazenado e pode ser tratado, verificando qual botão foi pressionado e qual a posição do mouse quando o botão foi pressionado. O código abaixo verifica quando o botão foi pressionado, e, adiciona uma posição do mouse a uma lista. A lista é desenhada como vários pontos, onde o mouse foi clicado.

```
import pygame
pygame.init()
import math
tamanhoJanela = [ 800, 600]
janela = pygame.display.set_mode(tamanhoJanela)
pygame.display.set_caption("Eventos de Teclado")

sair = False
#definindo a cor
BRANCO = (255,255,255)

pontos = []
while not sair:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sair = True
        elif event.type == pygame.MOUSEBUTTONDOWN:
            if event.button == 1: #esquerdo
                pontos.append(event.pos)

    #limpeza
    janela.fill(BRANCO)
    #atualização dos estados...
    #desenho
    c_cor = (255,0,0)
    for p in pontos:
        pygame.draw.circle(janela, c_cor, p, 10)
```

```
28
29      pygame.display.flip()
30  pygame.quit()
```

code/sec2/code5.py

Outros eventos como o scroll do mouse e o movimento do mouse também são lançados e podem ser tratados. O código abaixo possui uma *flag* que tem valor verdadeiro quando o mouse é clicado e valor falso quando o mouse é solto (botão esquerdo). Quando o mouse é movido, um evento do tipo *MOUSEMOTION* é lançado. Com este evento é possível obter a posição do mouse e o estado dos botões (não seria necessário utilizar MOUSEBUTTONDOW/MOUSEBUTTONUP, mas para simplificar, foram utilizados). Se o botão esquerdo estiver clicado e o mouse for movimentado, o raptor é desenhado na posição do mouse, semelhante a um "arrastar". O código abaixo mostra como isso pode ser implementado.

```
1  import pygame
2  pygame.init()
3  import math
4  tamanhoJanela = [ 800, 600]
5  janela = pygame.display.set_mode(tamanhoJanela)
6  pygame.display.set_caption("Eventos de Teclado")
7
8  sair = False
9  #definindo a cor
10 BRANCO = (255,255,255)
11
12 clicado = False
13 raptor = {
14     "imagem": pygame.image.load('raptor.png'),
15     "posicao": [10,10]
16 }
17
18 while not sair:
19     for event in pygame.event.get():
20         if event.type == pygame.QUIT:
```

```
21              sair = True
22          elif event.type == pygame.MOUSEBUTTONDOWN:
23              if event.button == 1: #esquerdo
24                  clicado = True
25          elif event.type == pygame.MOUSEBUTTONUP:
26              if event.button == 1: #esquerdo
27                  clicado = False
28          elif event.type == pygame.MOUSEMOTION:
29              if(clicado):
30                  raptor['posicao'] = event.pos
31
32      #limpeza
33      janela.fill(BRANCO)
34      #atualização dos estados...
35      #desenho
36      janela.blit(raptor['imagem'],raptor['posicao'])
37
38      pygame.display.flip()
39  pygame.quit()
```

code/sec2/code6.py

2.3 Exercício

Escreva um código para controlar uma nave na tela, movendo-a para todos lados (esquerda, direita, cima e baixo). Use as setas do teclado para isso.

3 Temporização e Rendering

O Tempo é muito importante em jogos. Seja para computar quanto tempo o Mario tem para terminar uma fase, ou, quanto tempo o Sonic levou em um nível até para a renderização e atualização do estado do jogo. O jogo é desenhado (*rendering*) algumas vezes por segundo. Essa é a tal taxa de quadros por segundo, ou *frames per second - FPS*. Antes de cada quadro desenhado, o status do jogo - personagem, cenário, inimigos - é atualizado um pouco. Assim, controlar o tempo entre os *frames* e quantos *frames* por segundo é crucial.

Alguns jogos se baseiam em deixar a velocidade do jogo controlada pelos *frames*, assim, a taxa de FPS tem que ser mantida igual, ou, o jogo pode rodar muito rápido ou muito devagar, a ponto do personagem ficar "incontrolável". O pygame possui o submódulo *time* para controlar o tempo e também o número de FPS automaticamente. Criando um objeto Clock, podemos definir a taxa de frames e também o tempo decorrido entre o último frame até o atual. O código abaixo mostra como controlar o objeto utilizando a taxa de frames.

```
1  import pygame
2  pygame.init()
3  import math
4  tamanhoJanela = [ 800, 600]
5  janela = pygame.display.set_mode(tamanhoJanela)
6  pygame.display.set_caption("Eventos de Teclado")
7
8  sair = False
9  #definindo a cor
10 BRANCO = (255,255,255)
11
12 raptor = {
13     "imagem": pygame.image.load('raptor.png'),
14     "posicao": [10,10],
15     "andaDireita": False
16 }
17 clock = pygame.time.Clock()
```

```
18 while not sair:
19     for event in pygame.event.get():
20         if event.type == pygame.QUIT:
21             sair = True
22         elif event.type == pygame.KEYDOWN:
23             if event.key == pygame.K_ESCAPE:
24                 sair = True
25             elif event.key == pygame.K_RIGHT:
26                 raptor['andaDireita'] = True
27         elif event.type == pygame.KEYUP:
28             if event.key == pygame.K_RIGHT:
29                 raptor['andaDireita'] = False
30     #limpeza
31     janela.fill(BRANCO)
32     #atualização dos estados...
33     if raptor['andaDireita']:
34         raptor['posicao'][0]+= 2
35     #desenho
36     janela.blit(raptor['imagem'],raptor['posicao'])
37
38
39     clock.tick(30)
40     pygame.display.flip()
41
42 pygame.quit()
```

code/sec3/code1.py

A cada frame o raptor irá se mover 2 pixels para a direita. Caso queiramos que ele se mova mais rápido, podemos aumentar o tamanho do deslocamento ou o número dos frames. Isso pode ser um problema, pois, não podemos garantir que o computador alvo (onde o jogo irá rodar) conseguirá manter a taxa de frames necessária. Ainda, se o deslocamento do objeto for muito alto, ele não se moverá de maneira suave.

3.1 Controlando a Velocidade em FPS Variável

Uma maneira de poder deixar o FPS variável é definir uma velocidade por segundo do objeto, e, atualizar seu estado de acordo com o tempo decorrido desde a última atualização. o método *clock.tick(fps)* retorna quantos milissegundos se passaram desde a última atualização. Assim, podemos atualizar a posição do personagem como $p.x = px + velocidade * dt$, onde dt é o tempo decorrido (em segundos). O código abaixo

```
1  import pygame
2  pygame.init()
3  import math
4  tamanhoJanela = [ 800, 600]
5  janela = pygame.display.set_mode(tamanhoJanela)
6  pygame.display.set_caption("Eventos de Teclado")
7
8  sair = False
9  #definindo a cor
10 BRANCO = (255,255,255)
11
12 class Raptor:
13     def __init__(self):
14         self.imagem = pygame.image.load('raptor.png')
15         self.posicao = [10.0,10.0]
16         self.andaDireita = False
17         self.velocidade = 60 #por segundo
18     def atualiza(self, dt):
19         if(self.andaDireita):
20             self.posicao[0] += self.velocidade * dt
21     def desenha(self, destino):
22         pos = [int(self.posicao[0]), int(self.posicao[1])]
23         destino.blit(self.imagem, pos)
24
25 raptor = Raptor()
26 clock = pygame.time.Clock()
27 while not sair:
```

```
28      for event in pygame.event.get():
29          if event.type == pygame.QUIT:
30              sair = True
31          elif event.type == pygame.KEYDOWN:
32              if event.key == pygame.K_ESCAPE:
33                  sair = True
34              elif event.key == pygame.K_RIGHT:
35                  raptor.andaDireita = True
36          elif event.type == pygame.KEYUP:
37              if event.key == pygame.K_RIGHT:
38                  raptor.andaDireita = False
39      #limpeza
40      janela.fill(BRANCO)
41      dt = clock.tick(180)
42      print(dt)
43      #atualização dos estados...
44      raptor.atualiza(dt/1000.0)
45      #desenho
46      raptor.desenha(janela)
47
48
49
50      pygame.display.flip()
51
52  pygame.quit()
```

code/sec3/code2.py

3.2 Exercício

Continue utilizando o código que movimenta a nave. Crie um array de inimigos que *spawnam* com um tempo aleatório, descem de cima para baixo, e, quando saírem da tela, voltam a *spawnar* novamente.

4 Personagens - Sprites e Animação

De acordo com a Wikipedia é "*Em computação gráfica, um sprite (do latim spiritus, significando 'duende', 'fada') é um objeto gráfico bi ou tridimensional que se move numa tela sem deixar traços de sua passagem (como se fosse um 'espírito')*".

4.1 Sprites e Animação de Sprites

Em jogos 2D, os personagens geralmente são desenhados e transformados em bitmaps (imagens 2D). Essas imagens geralmente possuem o mesmo tamanho (jogos de plataforma, por exemplo) e são quadros de uma animação, com uma pequena alteração de movimento entre eles. O conjunto de sprites de um personagem é chamado de *spriteset*.

Para termos o efeito de animação, cada sprite é apresentado por uma fração de segundos, em ciclos.

A primeira parte é definir um Sprite através de uma classe. O sprite possui uma posição que será utilizada para desenhar exatamente na posição do personagem. Isto ajuda caso alguns sprites tenham tamanho diferente, assim, todos serão desenhados a partir de um ponto específico.

```
1  import pygame
2  class Sprite:
3      def __init__(self, imageFile, displayTime, owner = None,
       animation = None):
4          self.image = pygame.image.load(imageFile)
5          self.displayTime = displayTime
6          self.animation = animation
7          #posicao que será usada para desenhar o sprite.
8          #por padrão, o meio do sprite
9          self.alignPosition = [int(self.image.get_width()/2), int(
       self.image.get_height())]
10
11     def getAlign(self):
12         return self.alignPosition
```

```
13      def setAnimation(self, animation):
14          self.animation = animation
15      def getWidth(self):
16          return self.image.get_width()
17      def getHeight(self):
18          return self.image.get_height()
19
20      #desenha o sprite na posicao (na surface) informada (
        onscreenPosition)
21      def render(self, dest, onScreenPosition):
22          dest.blit(self.image, onScreenPosition)
```

code/sec4/Sprite.py

A segunda classe é Animation, para representar uma animação. A animação é a exibição de uma sequência de sprites em determinada ordem. Esta recebe como parâmetro, no método render, o *onScreenPosition*, que é a posição (x,y) para desenhar na tela. As coordenadas do mundo devem ser convertidas para coordenadas de tela.

```
1  import pygame
2  class Animation:
3      def __init__(self, name):
4          self.sprites = []
5          self.loop = True
6          self.timer = 0
7          self.currentSprite = 0
8          self.name = name
9          self.owner = None
10         self.finished = False
11
12     def getCurrentSprite(self):
13         return self.sprites[self.currentSprite]
14     def setOwner(self, owner):
15         self.owner = owner
16
17     def addSprite(self, sprite):
```

```
18          sprite.setAnimation(self)
19          self.sprites.append(sprite)
20
21      def update(self, dt):
22          #atualiza o timer. Verifica se já passou o tempo do sprite
    atual.
23          #caso tenha passado, vai para o próximo e verifica se é o
24          #último sprite.
25          self.timer+=dt
26          if(self.timer > self.sprites[self.currentSprite].
    displayTime):
27              self.currentSprite = self.currentSprite + 1
28              self.timer = 0
29              if(self.currentSprite >= len(self.sprites)):
30                  if(self.loop):
31                      self.currentSprite = 0
32                  else:
33                      self.finished = True
34
35      def render(self, dest, onScreenPosition):
36          if(self.currentSprite < len(self.sprites)):
37              self.sprites[self.currentSprite].render(dest,
    onScreenPosition)
```

code/sec4/Animation.py

A última classe é uma que represente um objeto do game - personagem, inimigo ou powerup. Esta classe possui uma posição (no mundo, que será visto na Seção Cenário), um conjunto de animações e uma animação sendo exibida atualmente. As animações são armazenadas em um dicionário, para ser fácil encontrar cada uma. Como método, a classe possui um para desenho (render) e um para atualizar a lógica do objeto, que deve ser implementado em subclasses. Esta classe apresentará mudanças nos próximos capítulos.

```
1 import pygame
2 class GameObject:
```

```
3     def __init__(self, position = [0,0]):
4         #posição do personagem. Parte inferior, centralizada no
      sprite.
5         #
6         # ---------
7         # |         |
8         # |         |
9         # |____P___|
10        self.position = position#posicao no mundo.
11        self.animations ={} #animações. um dicionário
12        self.currentAnimation = None
13
14
15    #método de atualização, para ser sobrescrito em subclasses.
16    def update(self, dt):
17        pass
18    def render(self, dest):
19        if self.currentAnimation != None:
20            #TODO: 600 vai ser trocado pela altura do mapa.
21            print('-------------------------')
22            print("pos: " + str(self.position))
23            print("algn: " +str(self.currentAnimation.
      getCurrentSprite().getAlign()))
24
25            worldToScreen = [self.position[0] - self.
      currentAnimation.getCurrentSprite().getAlign()[0], 0]
26            worldToScreen[1] = self.position[1]+self.
      currentAnimation.getCurrentSprite().getHeight()
27            worldToScreen[1] = 600 - worldToScreen[1]
28            worldToScreen = [int(worldToScreen[0]), worldToScreen
      [1]]
29            self.currentAnimation.render(dest, worldToScreen)
30
31            print("scr: " +str(worldToScreen))
32            posToScreen = [int(self.position[0]), int(600 - self.
```

```
    position[1])]
33              pygame.draw.circle(dest,(0,0,0),posToScreen,3)
34      def addAnimation(self, animation):
35          animation.setOwner(self)
36          self.animation[animation.name] = animation
```

code/sec4/GameObject.py

4.2 Exemplo - Peixe

Como exemplo, vamos criar um peixe que se movimenta para direita, exibindo uma animação. A Figura 2 mostra os sprites utilizados na animação.

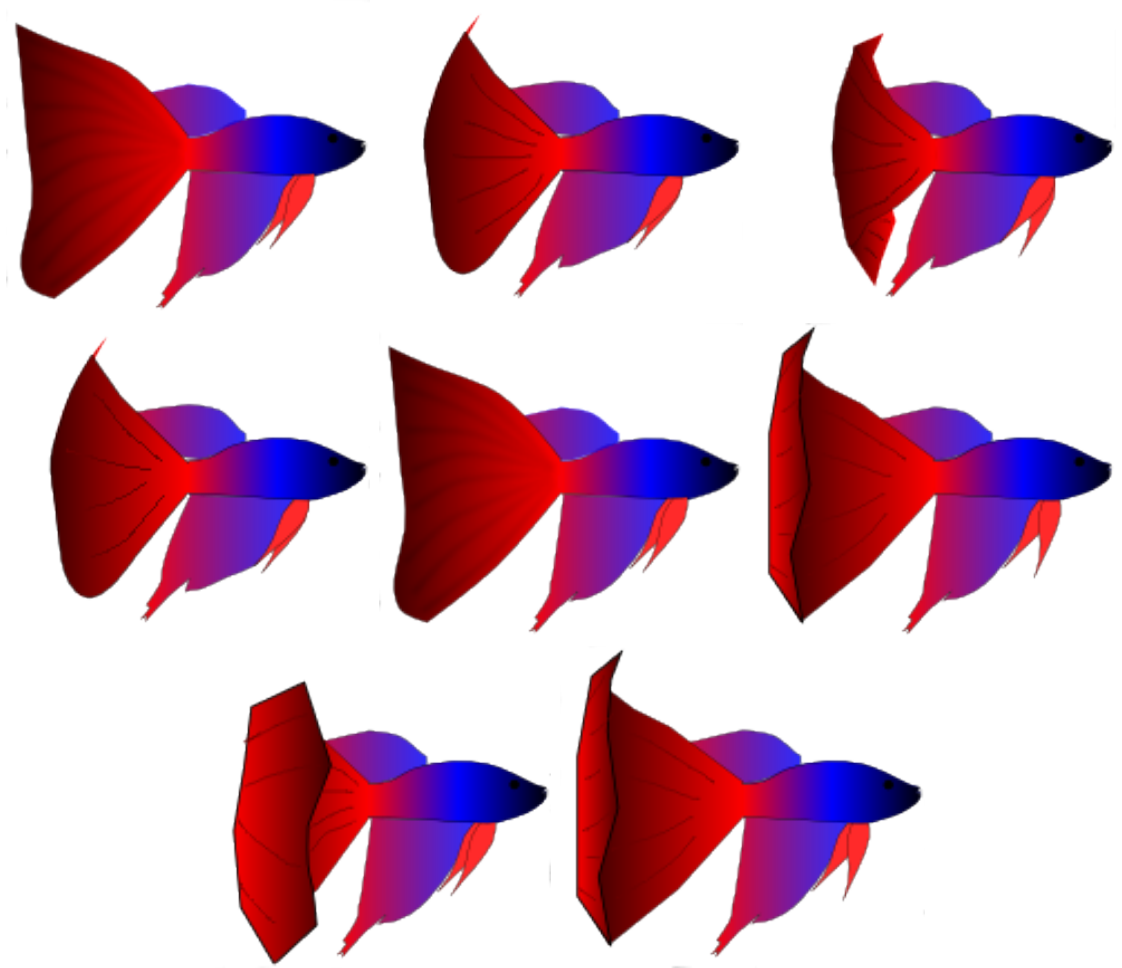

Figura 2: Sprites de um peixe nadando para direita.

O código abaixo começa a definir o comportamento do peixe. No construtor, é criara uma animação (Animation) e são inseridos os sprites da animação. A ani-

mação nadaDireita é adicionada ao dicionário (self.animations) e também é definida como a animação atual (currentAnimation). o atributo *velocidade* é específico para o peixe. O método *update* para o peixe atualiza a animação atual e atualiza a posição em x, considerando o tempo decorrido dt e a velocidade do peixe.

```
from GameObject import *
from Animation import *
from Sprite import *
class Peixe(GameObject):
    def __init__(self):
        GameObject.__init__(self)

        #Cria animacao...
        nadaDireita = Animation("nadaDireita")
        nadaDireita.setOwner(self)
        nadaDireita.addSprite(Sprite('betaFish/1.png',200/1000.0,
    self, nadaDireita))
        nadaDireita.addSprite(Sprite('betaFish/2.png',200/1000.0,
    self, nadaDireita))
        nadaDireita.addSprite(Sprite('betaFish/3.png',200/1000.0,
    self, nadaDireita))
        nadaDireita.addSprite(Sprite('betaFish/4.png',200/1000.0,
    self, nadaDireita))
        nadaDireita.addSprite(Sprite('betaFish/5.png',200/1000.0,
    self, nadaDireita))
        nadaDireita.addSprite(Sprite('betaFish/6.png',200/1000.0,
    self, nadaDireita))
        nadaDireita.addSprite(Sprite('betaFish/7.png',200/1000.0,
    self, nadaDireita))
        nadaDireita.addSprite(Sprite('betaFish/8.png',200/1000.0,
    self, nadaDireita))

        #adiciona às animações do Objeto
        self.animations[nadaDireita.name] = nadaDireita
        self.currentAnimation = nadaDireita
        self.velocidade = 100 #100px/s
```

```
24
25
26      def update(self, dt):
27          self.currentAnimation.update(dt)
28          self.position[0] += self.velocidade * dt
```

code/sec4/Peixe.py

4.3 Exercício

Finalize o objeto peixe. Permita ao jogador controlá-lo e movimentá-lo para cima, baixo, esquerda ou direita.

5 Cenário

O que seria um game sem um ambiente para ele acontecer, não?! Dependendo do estilo do jogo, o cenário pode ser construído de maneira diferente. Em jogos de nave (shoot em up) o cenário é longo com elementos repetitivos (de acordo com o *game*) e podem ser utilizados objetos que ficam passando ao fundo de maneira aleatória, como um fundo que represente um oceano e algumas ilhas, ou nuvens (similar ao jogo Biometal do SNES).

Outro estilo de visão dos cenários é uma vista superior. Este tipo de visão é comum principalmente entre os jogos de RPG. O personagem e elementos são mostrados de uma visão superior e os caminhos são bloqueados por blocos que não podem ser atravessados.

Uma terceira maneira de criar cenários que foi bastante utilizada na era 16 bits é a isométrica. O cenário é mostrado de um ângulo superior, porém, com certo grau de inclinação.

Por fim, o estilo de cenários mais utilizado, especialmente por jogos de plataformas, são os side scrollers, que, possuem rolagem lateral (podem ter para cima também). Este estilo é bastante utilizado em jogos de plataforma como o Super Mario (desde o Bross 1). A Figura 3 mostra exemplos de cenários em jogos 2D.

Figura 3: Cenários em jogos 2D: "vazio" com alguns elementos de fundo ou frente; visão superior; visão isométrica, e; visão lateral.

Para economizar memória, os cenários são criados com reuso de elementos. Um tileset é um conjunto de tijolos que são utilizados para criar um cenário. Cada tileset possui diferentes tipos de "tijolos", partes de uma imagem, que são desenhados para formar um cenário maior.

5.1 Tiles e Tileset

Para economizar memória, muitos games das eras 8, 16 e também da 32 bits utilizavam imagens reutilizáveis para representar cenários. Desta maneira, uma única imagem poderia servir para gerar um cenário completo e muito maior. Mais especificamente, o hardware dos consoles 8 bits e 16 bits (como o Nintendo, Sega Master System, Super Nintendo e Mega Drive) tinham funções específicas para desenhar tiles (tijolos, pequenas imagens) e jogo era desenvolvido assim. O NES (Nintendo Entertainment System) permitia desenhar tiles e tinha uma pequena memória para

arranjar alguns tiles que seriam desenhados nas TV's de tubo dos jogadores.

Atualmente, os games 2D são desenhados em alta definição, os cenários não precisam ser criados utilizando técnicas que visem economizar tanta memória e vários elementos podem ser apresentados na tela. Porém, uma maneira de representarmos os cenários em games 2D são os tilesets, os quais serão utilizados neste "motor de jogos".

Um tile é uma pequena subimagem de uma imagem maior, aqui chamada de tileset ou conjunto de tiles. Neste motor, os tiles terão tamanho quadrado e múltiplos de 2[1]. O tileset é então dividido em vários quadrados menores; o mapa ou cenário do jogo é formado por várias referências a estes quadradinhos, que, no processo de *rendering* são desenhados na tela para o jogador.

No motor, um tile é representado pela classe Tile. Um tile tem uma imagem (subimagem do tileset) e também um nome (apelido para referenciá-lo). A classe Tileset possui uma imagem, um prefixo (que será utilizado para nomear os seus tiles) e também um dicionário como todos os tiles. Os códigos abaixo mostram os códigos de ambas classes.

```
1  import pygame
2  class Tile:
3      def __init__(self, name, image):
4          self.image = image
5          self.name = name
6          #o tile poderá ter mais propriedades...
7
8      def getName(self):
9          return self.name
10     def getImage(self):
11         return self.image
```

code/sec5/Tile.py

```
1  import pygame
2  from Tile import *
```

[1]Em testes empíricos pessoais, tiles com tamanhos múltiplos de 2 apresentaram melhor performance

```
3  class Tileset:
4      def __init__(self, image_file, size, prefix="t_"):
5          self.image = pygame.image.load(image_file)
6          if self.image.get_width()%size != 0 or self.image.
       get_height()%size != 0:
7              print("ERROR. Size is not compatible with image. ")
8              exit()
9          if size%2 !=0:
10             print("ERROR. Size must be multiple of 2 ")
11             exit()
12         self.namePrefix = prefix
13         self.size = size
14
15         self.tiles = {}
16
17
18         cont = 0
19         #para altura...
20         for i in range(0, int(self.image.get_height()/size)):
21             #para largura
22             for j in range(0, int(self.image.get_width()/size)):
23                 #retângulo de 'corte'
24                 rect =[j * self.size, i *self.size, self.size, self
       .size]
25                 #cria uma subsurface, "filha" da imagem original
26                 subS = self.image.subsurface(rect)
27                 #novo tile
28                 t = Tile(prefix+str(cont), subS)
29                 #dicionario para acesso mais fácil
30                 self.tiles[t.getName()] = t
31                 cont+= 1
32     def getSize(self):
33         return self.size
34     def getTile(self, name):
35         print("name: " +str(name))
```

```
36          if name in self.tiles:
37              return self.tiles[name]
38          return None
```

code/sec5/Tileset.py

A Figura 4 mostra um exemplo de tileset e também um mapa criado utilizando-o.

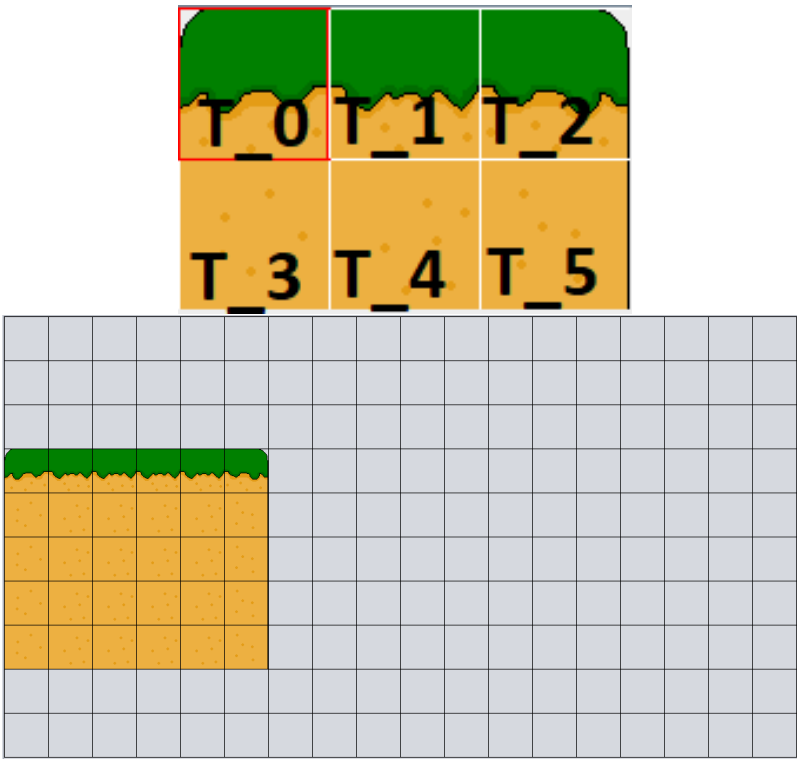

Figura 4: Tileset (imagine que o prefixo fornecido tenha sido "t") e um mapa.

5.2 Mapa

Não adianta possuir um tileset todo bonitinho se não fizermos coisa alguma com ele. O ambiente do jogo (cenário) que utiliza tiles pode ser visto como uma matriz com referências para tiles de um tileset específico. Assim, o mapa é uma matriz (lista de listas em python) com referências a tiles de um tileset, ou, None (nulo) caso seja um espaço vazio. Os tiles são colidíveis com os personagens (NPC ou PC) e isso será visto em uma seção mais adiante.

O mapa não é renderizado completamente. Uma janela móvel representa qual

porção do mapa é desenhado. A partir dessa janela são calculados os tiles iniciais e finais do mapa. Para facilitar, o mapa é desenhado de cabeça para baixo, para que as coordenadas da matriz coincidam com as coordenadas de tela - lembre-se que as coordenadas da tela são de cima para baixo e esquerda para direita; assim, pensando em uma matriz, o tile que está na posição (0,0) está na verdade no ponto mais alto do mapa desenhado, assim como as alturas dos personagens também seriam invertidas. Não se preocupe, o calculo está todo feito e *no meu funcionou*. A Figura 5 mostra um resumo como estão as coordenadas do mapa.

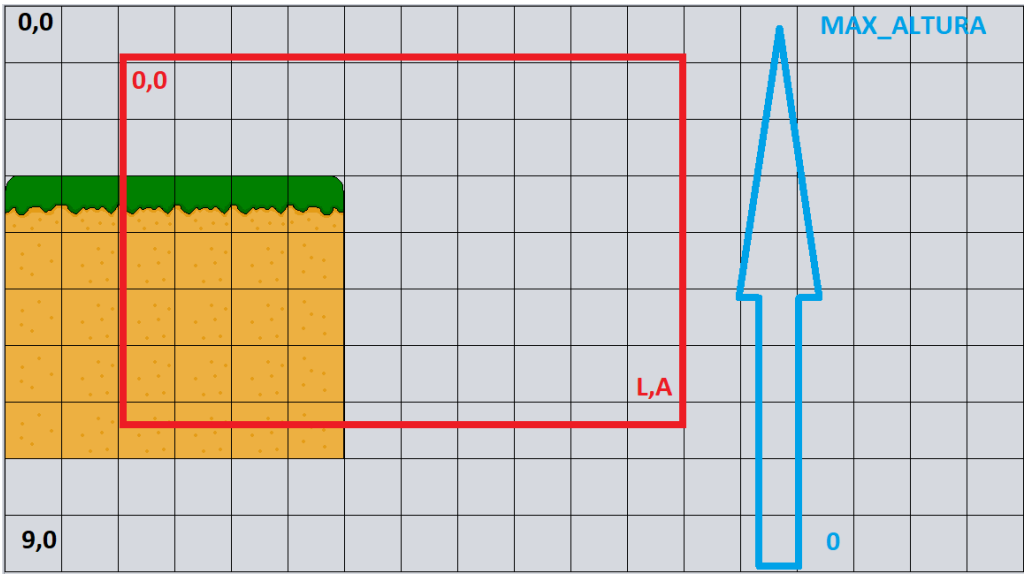

Figura 5: O mapa é feito de uma matriz, a janela (em vermelho) representa a porção que é mostrada na tela e a altura do mapa começa debaixo, ou seja, da maior posição da matriz. Tudo isso é convertido e a posição dos personagens e também da janela começa embaixo e cresce para cima na tela, não na matriz (seta azul).

Passos para criar um mapa utilizando o motor:

- Criar ou encontrar uma imagem de tileset, na qual cada tile tenha tamanho múltiplo de 2 (16px, 32, 64, 128, ...);
- Criar o tileset;
- Definir a altura e largura (em tiles) do mapa;

- Pensar no tamanho da janela: não pode ser maior que a largura ou altura do mapa!

- Definir os tiles do mapa.

Uma versão preliminar do código do mapa pode ser visto em "code/sec5/GameMap.py". Veja que ele já possui um método para carregar o mapa de um arquivo e também para mover a janela de visualização, que representa qual porção do mapa será desenhada na tela.

5.2.1 Ferramenta para Edição do Mapa

Criar um mapa todo na mão seria muito difícil. Pensando nisso, uma ferramenta (muito sensível a erros!) fornecida permite configurar um mapa. O editor é um código simples HTML e Javascript, que acompanha o motor. A ferramenta visual (Figura 6) gera então um arquivo JSON que pode ser utilizado para carregar o mapa, através do método "loadFromFile" – lembre-se que o arquivo de imagem deve estar na raiz do game, pois ele é procurado lá (no mesmo que a "main").

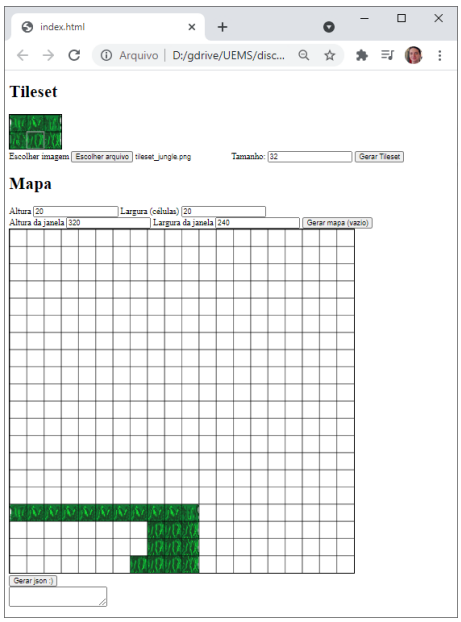

Figura 6: Ferramenta HTML/JS para editar um mapa e gerar um JSON para ele.

5.3 Exercício

Utilize o editor de mapas e crie um nível de super mario. Utilize o código exemplo "main_loadMap.py" como exemplo. Ele carrega o JSON do mapa e também permite navegar no mapa utilizando as setas do teclado.

6 Plano de fundo

Um artifício utilizado para deixar o cenário mais bonito é o plano de fundo. Ao invés de uma cor sólida, alguns detalhes deixam tudo melhor. Para criar o plano de fundo, pode-se utilizar uma imagem que o preencha completamente ou tornar essa imagem móvel. A imagem pode se mover verticalmente ou horizontalmente ou em ambas direções. Diferentes camadas de plano de fundo podem ser adicionadas a um cenário, criando novos comportamentos para o plano de fundo.

No motor em desenvolvimento cada mapa possui um plano de fundo. O plano de fundo é composto de diferentes camadas, cada uma possuindo uma imagem e um comportamento para essa imagem. A classe BackgroundLayer representa uma camada do plano de fundo e possui como atributos uma imagem e um tipo que identifica o comportamento que a camada irá realizar. O código abaixo mostra o código da classe BackgroundLayer.

```
1  import pygame
2  import math
3  class BackgroundLayer:
4      def __init__(self, imageFile,type, moveSpeed=None):
5          self.image = pygame.image.load(imageFile)
6          self.moveSpeed = moveSpeed
7          self.startPoint = [0,0]
8          self.background = None
9          self.width = self.image.get_width()
10         self.height = self.image.get_height()
11         self.type = type # fill, repeat-x, repeat-y, move
12         self.deltaMove = [0,0]
13
14
15
16     def setBackground(self, bkg):
17         self.background = bkg
18
19     def update(self, dt):
```

```
20          w = self.background.getMapWindow()
21          self.startPoint = [w[0],w[1]]
22
23          w = self.background.getMapWindow()
24          if self.startPoint[0] < 0:
25              if (self.startPoint[0] + self.width) < 0:
26                  self.startPoint[0] = w[2] -1
27          if self.startPoint[0] > w[2]-1:
28              self.startPoint[0] = 0
29
30          if self.type == "MOVE":
31              self.deltaMove[0] += dt * self.moveSpeed[0]
32              self.deltaMove[1] += dt * self.moveSpeed[1]
33              if self.deltaMove[0] > self.width-1:
34                  self.deltaMove[0] = 0
35              if (self.deltaMove[0] + self.width) < 0:
36                  self.deltaMove[0] = self.width-1
37
38      def render(self, dest):
39          #desenha quantas couberem na tela
40          w = self.background.getMapWindow()
41
42          screenW = w[2]
43          screenH = w[3]
44
45          if(self.type =='FILL'):
46              redim = pygame.transform.scale(self.image, (w[2],w[3]))
47              dest.blit(redim,[0,0])
48          elif self.type == "REPEAT-X": #repete em x...
49              offsetX = self.startPoint[0]%self.width
50              startTileX = math.floor(self.startPoint[0]/self.width)
51              endTileX   = math.floor((self.startPoint[0]+screenW)/
    self.width)
52
53              numTilesX = endTileX - startTileX
```

```
54
55              #reescala em y....
56              redim = pygame.transform.scale(self.image, (self.width,
    w[3]))
57              print("endx: " +str(endTileX) + " start: " +str(
    startTileX)+" num Tiles x: " + str(numTilesX))
58              #se o bkg for menor que a tela so desenha 1...
59              for i in range(0, numTilesX+1):
60                  dest.blit(redim, [i*self.width -offsetX,0])
61
62          elif self.type == "REPEAT-Y": #repete em y..
63              offsetY = self.startPoint[1]%self.height
64              numTilesY = math.ceil(self.height/screenH)
65              print("offset: " +str(offsetY) + " num tiles y: " +str(
    numTilesY))
66              redim = pygame.transform.scale(self.image, (screenW,
    self.height))
67
68              #se o bkg for do tamanho da janela, falta 1. Por isso,
    + 1.
69              if numTilesY == 1:
70                  numTilesY+= 1
71              for i in range(0, numTilesY):
72                  # -self.heigh, pois o ponto de desenho é o superior
     esquerdo da imagem,...
73                  dest.blit(redim, [0,-self.height + screenH - (i*
    self.height)+offsetY])
74
75          elif self.type == "REPEAT": #repete em ambos...
76              #para x, mesmo que acima..
77              offsetX = self.startPoint[0]%self.width
78              startTileX = math.floor(self.startPoint[0]/self.width)
79              endTileX   = math.floor((self.startPoint[0]+screenW)/
    self.width)
80              numTilesX = endTileX - startTileX
```

```
81
82              #para y, mesmo que acima...
83              offsetY = self.startPoint[1]%self.height
84              numTilesY = math.ceil(self.height/screenH)
85
86
87
88              #se o bkg for do tamanho da janela, falta 1. Por isso,
    + 1.
89              if numTilesY == 1:
90                  numTilesY+= 1
91
92              for i in range(0, numTilesY):
93                  for j in range(0, numTilesX+1):
94                      # -self.heigh, pois o ponto de desenho é o
    superior esquerdo da imagem,...
95                      dest.blit(self.image, [j*self.width -offsetX,-
    self.height + screenH - (i*self.height)+offsetY])
96
97          elif self.type =='MOVE': #cenario se move automaticamente.
98              #a imagem deve ser maior na direção que vai se mover...
99              print(self.deltaMove)
100             dest.blit(self.image, [self.deltaMove[0],self.deltaMove
    [1]])
101             if self.deltaMove[0] > 0:
102                 dest.blit(self.image, [-self.width+self.deltaMove
    [0],self.deltaMove[1]])
103             elif self.deltaMove[0] < 0:
104                 dest.blit(self.image, [self.deltaMove[0]+self.width
    ,self.deltaMove[1]])
```

code/sec6/BackgroundLayer.py

O tipo da camada pode ser:

- *"FILL"*: preenche toda a área do fundo da janela, escalando a imagem;
- *"REPEAT-Y"*: preenche a área do fundo da janela sem redimensionar a ima-

gem, repetindo verticalmente caso seja necessário;

- *"REPEAT-X"*: preenche a área do fundo da janela sem redimensionar a imagem, repetindo horizontalmente caso seja necessário;
- *"REPEAT"*: preenche a área do fundo da janela sem redimensionar a imagem, repetindo horizontalmente e verticalmente caso seja necessário;
- *"MOVE"*: o plano de fundo se move em uma direção indicada por um vetor e a imagem é repetida horizontalmente e verticalmente.

O código do plano de fundo é bem simples, e possui uma referência para o mapa e também uma lista com as camadas. O código pode ser visto abaixo.

```
1  import pygame
2  class Background:
3      def __init__(self):
4          self.layers = []
5          self.map = None
6      def getMapWindow(self):
7          return self.map.getWindow()
8      def addLayer(self, layer):
9          layer.setBackground(self)
10         self.layers.append(layer)
11     def setMap(self, map):
12         self.map = map
13     def update(self, dt):
14         for l in self.layers:
15             l.update(dt)
16     def render(self, dest):
17
18         for l in self.layers:
19             l.render(dest)
```

code/sec6/Background.py

Agora, a classe GameMap também foi atualizada e possui um atributo novo "background", além do método "setBackground()". O método de desenho também

foi atualizado; a primeira coisa é verificar se o mapa possui um plano de fundo, e, caso possua, desenhá-lo como mostra o trecho de código abaixo.

```
def render(self, dest):
  if self.background !=None:
    self.background.render(dest)
```

A classe Background possui referência ao mapa e pode obter a janela (window). Através dessa janela é calculada a movimentação do plano de fundo.

7 Utilizando um Plano de Fundo

A ferramenta para editar o cenário não permite configurar o plano de fundo, tarefa que deve ser feita através de código. Para criar um plano de fundo, as camadas são criadas separadamente, adicionadas ao plano de fundo que é "setado" para o mapa. O trecho abaixo mostra como configurar um plano de fundo que se move para a esquerda a 90 pixels por segundo. Este código pode ser testado em *sec6/main_loadMap.py*

```
mapa = GameMap(None)
mapa.loadFromFile('map.json')
bkg = Background()
#cria a camada
bkgl = BackgroundLayer('jpbkg.png','MOVE',[-90,0])
#adiciona a camada ao plano de fundo
bkg.addLayer(bkgl)
#'seta' o plano de fundo
mapa.setBackground(bkg)
```

7.1 Exercício

Adicione uma névoa que fica na frente do cenário e que se move lentamente.

8 Áudio

Em *games*, podemos pensar em dois tipos de áudio que são executados: músicas (de fundo, trilha sonora) e efeitos especiais. As músicas geralmente são arquivos comprimidos e reproduzidos por *streaming* (parte do arquivo é lido, "descomprimido" e executado). Os efeitos de áudio (por exemplo pulo, tiro, etc.) geralmente são arquivos de áudio mais curtos, não comprimidos e que estão carregados na memória.

As músicas de fundo podem ser por meio de arquivos do tipo MIDI – um formato que não armazena amostras, mas um conjunto de comandos que são executados pelo *hardware*; pense no speaker do seu computador e que podemos controlar a velocidade e o tom que ele vai tocar. As músicas também podem ser arquivos em formatos de áudio não comprimido, como WAV, que utilizam **muito** espaço de armazenamento, ou, arquivos comprimidos como MP3 ou OGG.

8.1 SFX - Efeitos Especiais

A pygame permite carregar e reproduzir efeitos especiais, de maneira bem simples. Para facilitar o gerenciamento e acesso dos efeitos especiais, um dicionário comportará a todos, sendo acessados por uma chave especial. A classe SFXManager gerenciará todos efeitos sonoros que serão utilizados no game - esta classe será inserida no motor de jogos.

```
1  import pygame
2  class SFXManager:
3      def __init__(self):
4          self.sounds = {}
5
6      def play(self, key):
7          if key in self.sounds:
8              self.sounds[key].play()
9
10     def add(self, key, sound):
11         self.sounds[key] = sound
```

code/sec7/SFXManager.py

8.2 Música

A pygame permite carregar uma música, reproduzi-la, pausar e parar.

8.3 Exercício

Termine de adicionar os componentes da bateria (bumbo, caixa) e uma música de fundo para acompanhar.

9 Colisões: Sprite - Sprite

Para haver interação entre os objetos na tela, geralmente verificamos se houve alguma colisão, ou seja, se algum objeto se chocou/tocou outro. Por exemplo, o Mario foi atingido por um inimigo? Pegou um *powerup*?

Para verificar a colisão, pode-se checar pixel a pixel dos objetos, se algum intersecta outro (o que é um pouco pesado), ou, adicionar um polígono mais simples que comporte o objeto e que permita verificar uma colisão aproximada. Geralmente, são utilizados retângulos (ou caixas de colisão) ou círculos.

Neste motor, a colisão é gerenciada por caixas. Uma caixa é posicionada nas coordenadas do sprite (dentro da imagem) e o motor permite calcular ela transladada (movida) para a posição atual do jogador, ou seja, em coordenadas do mundo. Com essas caixas transladadas é então possível verificar se houve colisão entre duas caixas. A Figura 7 mostra o posicionamento de uma caixa de colisão dentro de um sprite. Por padrão, o construtor do sprite cria uma caixa de colisão que tem o mesmo tamanho do sprite, ou seja, todo o sprite é "colidível".

Figura 7: Uma caixa de colisão (em vermelho) posicionada dentro do sprite.

No motor, o código responsável pela caixa de colisão está na classe CollisionBox (CollisionBox.py). Esta classe possui o método *collides()* que verifica se a caixa

colide com outra.

Existem outros métodos que podem ser úteis:

- *isPointInside(self, p):* verifica se um ponto (em coordenadas de mundo) está dentro da caixa;
- *checkColType(self, corners):* verifica o tipo de colisão que uma caixa sofre (esquerda, direita, baixo, cima);
- *collisionType(self, anotherBox):* verifica as colisões que duas caixas sofrem; é retornada uma lista com os tipos de colisão sofridos por cada caixa.
- *computeOnScreen*: calcula o retângulo (uma nova CollisionBox) em coordenadas de tela.

9.1 Exercício

Utilize o peixe que se move, feito na Seção 4. Faça com que a minhoca, ao colidir com o peixe, desapareça e *spawne* em outro local.

10 Colisões: Sprite - Cenário

Para poder posicionar o personagem no cenário, de maneira que ele não caia ou que tenha limites que possa visitar, são calculadas colisão entre os sprites do personagem (as caixas de colisão) e o cenário. A cada instante ocorrem colisões e o personagem deve ser reposicionado no cenário. Se estiver caindo (pela gravidade, por exemplo) a caixa de colisão irá invadir os tiles abaixo e o personagem terá que ser empurrado para cima, para fora dos tiles, como mostra a Figura 8.

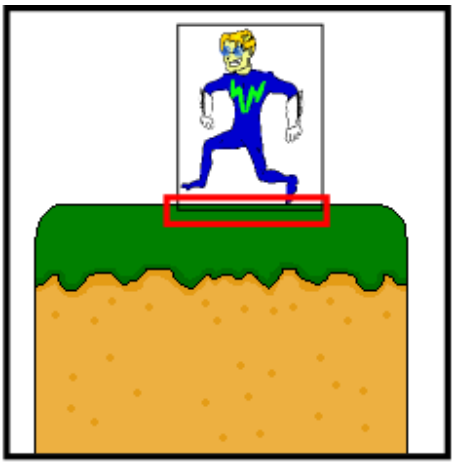

Figura 8: O personagem tem o sprite movido para baixo devido à gravidade, porém, o tratamento de colisões com o cenário o empurra de volta para fora do tile.

O tratamento de colisão com o cenário foi inserido na classe "GameObjetct" no método "checkMapCollision". Este método verifica se algum dos lados da caixa de colisão entrou em algum tile e o põe para fora. Para isso, é verificado qual foi o último movimento do personagem (cima, baixo, esquerda, direita) e depois o tratamento de colisão é feito, pois desta maneira sabe-se em qual local a colisão provavelmente teria ocorrido.

Agora também a movimentação do personagem é feita utilizando o método "moveTo(pos)" da classe "GameObject" pois este guarda em um atributo da classe (*lastMove*) qual foi o último movimento realizado pelo personagem. Os códigos são extensos, porém, o exemplo em "sec9" apresenta um exemplo que permite movimentar um peixe com colisão com o cenário.

11 Exercício

Crie uma classe para um inimigo que se move de um lado para o outro, sempre que bater em um tile.