N_JOG DIG3 - Texto de apoio

Site: EAD Mackenzie
Tema: JOGOS DIGITAIS {TURMA 04B} 2023/2
Livro: N_JOG DIG3 - Texto de apoio

Impresso por: FELIPE BALDIM GUERRA .
Data: segunda, 11 set 2023, 22:17

Descrição

Índice

1. CRIANDO O VISUAL DOS JOGOS

Agora que você já trabalhou com o processo de documentação e conhece os principais aspectos necessários para especificar e construir os jogos digitais (dentro do processo de *Game Design*), podemos começar a explorar alguns aspectos importantes da criação do visual dos jogos.

Os jogos digitais são, por natureza, muito visuais, e toda a equipe dentro do processo de Game Design empreende muito tempo trabalhando em manipulação e tratamento de imagens, tentando oferecer a melhor experiência possível ao usuário.

1.1. Usando e entendendo o pixel

Se você se aproximar da tela do computador, de uma televisão digital, de um celular, ou de algum outro dispositivo de imagem digital, você conseguirá perceber que essas telas são compostas por linhas e colunas preenchidas por pequenos pontos coloridos, semelhante à imagem abaixo.

A uma distância confortável, não conseguimos identificar completamente esses pontos, pois eles formam imagens únicas nos dispositivos de imagem. Cada ponto individual é chamado **Pixel** e pode assumir qualquer cor do padrão RGB (codificação utilizando três cores básicas – *Red, Green, Blue*), no caso do Pygame.

A listagem de código Python a seguir gera uma imagem contendo todas as possíveis cores. Esse script inclusive pode demorar alguns minutos para executar, mas gerará, ao final, um arquivo de imagem.

```
import pygame
pygame.init()

screen = pygame.display.set_mode((640, 480))

all_colors = pygame.Surface((4096,4096), depth=24)

for r in range(256):
    print(r+1, "out of 256")
    x = (r&15)*256
    y = (r>>4)*256
    for g in range(256):
        for b in range(256):
            all_colors.set_at((x+g, y+b), (r, g, b))

pygame.image.save(all_colors, "allcolors.bmp")
pygame.quit()
```

1.2. Trabalhando com cores

Conforme discutido anteriormente, o Pygame trabalha com a padronização para cores RGB, armazenando na forma de uma tupla de três inteiros. Dessa forma, é associado, para cada componente de cor, um valor correspondente. Esses valores devem estar no intervalo entre 0 e 255.

A listagem de código a seguir apresenta um mecanismo que permite a mistura das três cores básicas e viabiliza a apresentação da cor resultante, além de mostrar o valor RGB relacionado a ela.

```
import pygame
from pygame.locals import *
from sys import exit

pygame.init()

screen = pygame.display.set_mode((640, 480), 0, 32)

# Creates images with smooth gradients
def create_scales(height):
    red_scale_surface   = pygame.surface.Surface((640, height))
    green_scale_surface = pygame.surface.Surface((640, height))
    blue_scale_surface  = pygame.surface.Surface((640, height))
    for x in range(640):
        c = int((x/639.)*255.)
        red   = (c, 0, 0)
        green = (0, c, 0)
        blue  = (0, 0, c)
        line_rect = Rect(x, 0, 1, height)
        pygame.draw.rect(red_scale_surface, red, line_rect)
        pygame.draw.rect(green_scale_surface, green, line_rect)
        pygame.draw.rect(blue_scale_surface, blue, line_rect)
    return red_scale_surface, green_scale_surface,
blue_scale_surface

red_scale, green_scale, blue_scale = create_scales(80)

color = [127, 127, 127]
```

```
while True:

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

    screen.fill((0, 0, 0))

    # Draw the scales to the screen
    screen.blit(red_scale, (0, 00))
    screen.blit(green_scale, (0, 80))
    screen.blit(blue_scale, (0, 160))

    x, y = pygame.mouse.get_pos()

    # If the mouse was pressed on one of the sliders, adjust the
color component
    if pygame.mouse.get_pressed()[0]:
        for component in range(3):
            if y > component*80 and y < (component+1)*80:
                color[component] = int((x/639.)*255.)
            pygame.display.set_caption("PyGame Color Test -
"+str(tuple(color)))


    # Draw a circle for each slider to represent the current setting
    for component in range(3):
        pos = ( int((color[component]/255.)*639), component*80+40 )
        pygame.draw.circle(screen, (255, 255, 255), pos, 20)

    pygame.draw.rect(screen, tuple(color), (0, 240, 640, 240))

    pygame.display.update()
```

Muitas vezes, em um jogo, precisamos diminuir a intensidade de uma cor, tornando-a mais escura, por exemplo uma nave que dispara um laser, e este laser, à medida que se afasta da nave, perde seu poder de dano; para representar essa alteração visualmente, podemos escurecer a cor. Também quando nosso personagem entra em uma área de sobra (com menos luminosidade) ou perde algum poder que tinha antes, podemos representar essa mudança com o escurecimento de suas cores. Para isso, podemos multiplicar cada um dos componentes da cor por um valor entre 0 e 1. Se usarmos o laranja (221, 99, 20) como exemplo e multiplicar por 0.5 (reduzir pela metade), teremos (110, 49, 10). Lembrando que os valores de cada componente devem ser inteiros, então desprezamos a parte fracionária do resultado.

1.3. Usando imagens

As imagens são parte importante para a grande maioria dos jogos. Em jogos 2D, e até em jogos 3D, as imagens podem ser utilizadas em backgrounds, texto, personagens que simbolizam o avatar dos jogadores ou dos personagens autônomos (NPS, *Non-player Characters*), utilizando alguma forma de Inteligência Artificial (IA), e, também, para serem usadas como texturas de objetos 3D.

Os computadores armazenam imagens com grandes quantidades de cores e diferentes definição. Além dos componentes vermelho, verde e azul (RGB), pode haver um componente para alpha. Esse valor de alpha de uma cor é utilizado para representar a transparência, de modo que – quando desenhamos uma imagem sobre outra – seja possível visualizar a imagem colocada por baixo.

Para armazenar imagens no computador, há uma grande variedade de formatos a serem utilizados. Entretanto, um pequeno grupo de formato tem se destacado como os mais simples e úteis; os mais populares são JPG e PNG, ambos bem suportados pelos mais diversos softwares de edição de imagens e, igualmente, pelo Pygame. Além desses dois formatos, o Pygame também suporta GIF (não animado), BMP, PCX, TGA (sem compressão), TIF, entre outros.

As imagens são carregadas no Pygame por meio de uma única linha simples; pygame.image.load aceita o nome do arquivo a ser carregado e retorna um objeto de superfície, que funciona como um contêiner para uma imagem. Podemos também criar superfícies preenchidas com uma cor base, por exemplo branco (normalmente realizado assim para identificar a superfície principal – a primeira camada de nossas composições). Isso pode ser feito desta forma:

```
Blank_surface = pygame.Surface((256,256))
```

Com frequência, nos jogos que desenvolvemos no Pygame, é necessário fornecer um retângulo para definir a parte da tela que será afetada por alguma chamada de função. Isso pode ser necessário quando, por exemplo, necessitamos restringir o Pygame para que ele desenhe em uma área retangular, definindo um retângulo de clipping (recorte). Para esse tipo de definição, podemos usar uma tupla que contenha quatro valores: as coordenadas x e y do canto superior esquerdo, assim como largura e altura do retângulo; por exemplo:

```
Meu_rect = (100,100,50,30)

Meu_rect2 = ((20,10),(100,100))
```

O Pygame também disponibiliza o módulo rect com vários métodos convenientes para a construção de retângulos. Basta importá-lo como o exemplo abaixo:

```
from pygame import rect

meu_rect3 = Rect(100,100,20,150)

meu_rect4 = Rect(50,100,120,250)
```

1.4. Blitting

Talvez o método dos objetos fornecidos pelo Pygame mais utilizado seja o blit, acrônimo para *bit block transfer*. *Blitting* (ou blitar, a partir de uma tradução livre e sem nenhum compromisso com a gramática dos idiomas português ou inglês, mas uma que é comumente utilizada) simplesmente significa copiar os dados de uma imagem de uma superfície para outra superfície. Usaremos esse método para desenhar backgrounds, fontes, personagens e praticamente qualquer coisa que quisermos na tela para o usuário. A seguir, duas maneiras de usar o método blit.

```
screen.blit(background,(0,0))

screen.blit(ogre, (300,200), (100*frame_no,0,100,100))
```

A primeira instrução faz um blit da superfície referenciada pela variável background no canto superior esquerdo da tela (0,0); vale ressaltar que, se a superfície em questão não tiver o mesmo tamanho de screen, será necessário preencher a superfície screen com alguma cor de fundo.

Se tivermos uma imagem com vários frames de um ogro andando, seria possível usar a segunda instrução para fazer seu blit na tela.

O Pygame também disponibiliza o módulo pygame.draw, fornecendo diversas funções específicas para desenhar formas geométricas básicas. O clássico jogo Asteroids ou o jogo Pong, por exemplo, podem ser implementados completamente utilizando essas funções de desenhos de formas geométricas básicas. A seguir, alguns exemplos de utilização dessas funções básicas.

1.5. Desenhando retângulos:

```python
import pygame
from pygame.locals import *
from sys import exit

from random import *

pygame.init()
screen = pygame.display.set_mode((640, 480), 0, 32)

screen.lock()
for count in range(10):
    random_color = (randint(0,255), randint(0,255), randint(0,255))
    random_pos = (randint(0,639), randint(0,479))
    random_size = (639-randint(random_pos[0],639), 479-randint(random_pos[1],479))
    pygame.draw.rect(screen, random_color, Rect(random_pos, random_size))

screen.unlock()

pygame.display.update()

while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            exit()
```

1.6. Desenhando polígonos de vários lados:

```python
import pygame
from pygame.locals import *
from sys import exit

pygame.init()
screen = pygame.display.set_mode((640, 480), 0, 32)

points = []

while True:

    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            exit()
        if event.type == MOUSEBUTTONDOWN:
            points.append(event.pos)

    screen.fill((255,255,255))

    if len(points) >= 3:
        pygame.draw.polygon(screen, (0,255,0), points)
```

```
23      for point in points:
24          pygame.draw.circle(screen, (0,0,255), point, 5)
25
26      pygame.display.update()
```

1.7. Desenhando um círculo:

```python
import pygame
from pygame.locals import *
from sys import exit

pygame.init()
screen = pygame.display.set_mode((640, 480), 0, 32)

points = []

while True:

    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            exit()
        if event.type == MOUSEMOTION:
            points.append(event.pos)
            if len(points)>100:
                del points[0]

    screen.fill((255, 255, 255))

    if len(points)>1:
        pygame.draw.lines(screen, (0,255,0), False, points, 2)
```

```
25
26      pygame.display.update()
27
```

1.8. Desenhando uma elipse:

```python
import pygame
from pygame.locals import *
from sys import exit

from random import *

pygame.init()
screen = pygame.display.set_mode((640, 480), 0, 32)

while True:

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

    x, y = pygame.mouse.get_pos()
    screen.fill((255,255,255))
    pygame.draw.ellipse(screen, (0,255,0), (0,0,x,y))

    pygame.display.update()

```

1.9. Desenhando um arco, ou seja, uma seção de uma elipse:

```python
import pygame
from pygame.locals import *
from sys import exit

from random import *
from math import pi

pygame.init()
screen = pygame.display.set_mode((640, 480), 0, 32)

while True:

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

    x, y  = pygame.mouse.get_pos()
    angle = (x/639.)*pi*2.
    screen.fill((255,255,255))
    pygame.draw.arc(screen, (0,0,0), (0,0,639,479), 0, angle)

    pygame.display.update()

```

1.10. Desenhando linhas entre dois pontos:

```python
import pygame
from pygame.locals import *
from sys import exit

pygame.init()
screen = pygame.display.set_mode((640, 480), 0, 32)

points = []

while True:

    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            exit()
        if event.type == MOUSEMOTION:
            points.append(event.pos)
            if len(points)>100:
                del points[0]

    screen.fill((255, 255, 255))

    if len(points)>1:
        pygame.draw.lines(screen, (0,255,0), False, points, 2)

    pygame.display.update()
```


pygame.display.update()

1.11. Desenhando linhas em sequência que podem, de acordo com os parâmetros

informados, serem fechadas, formando um polígono:

```python
import pygame
from pygame.locals import *
from sys import exit

pygame.init()
screen = pygame.display.set_mode((640, 480), 0, 32)

points = []

while True:

    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            exit()
        if event.type == MOUSEMOTION:
            points.append(event.pos)
            if len(points)>100:
                del points[0]

    screen.fill((255, 255, 255))

    if len(points)>1:
        pygame.draw.lines(screen, (0,255,0), False, points, 2)

```

```
26      pygame.display.update()
27
```

2. Referências

- MCGUGAN, Will. *Beginning game development with Python and Pygame:* from novice to professional. New York: Apress, 2007.