HANDOUT ASTEROIDS 2 - AULA PYGAME

Os objetivos de aprendizado dessa aula são:

- Continuar no processo de recriar um jogo simples baseado no Asteroids, usando o *framework* Pygame.

Não se esqueça: *commit, pull, push* no repositório ao término de cada fase.

Copie os novos assets para o seu diretório de assets.

CARREGANDO TODOS OS ASSETS APENAS UMA VEZ

Vamos começar com o código em gabarito/asteroids-4.py para garantir que estamos todos na mesma página.

Crie a função vista a seguir logo depois das definições de classe:

```
# Carrega todos os assets uma vez só.
def load_assets(img_dir, snd_dir):
    assets = {}
    assets["player_img"] = pygame.image.load(path.join(img_dir, "playerShip1_orange.png")).convert()
    assets["mob_img"] = pygame.image.load(path.join(img_dir, "meteorBrown_med1.png")).convert()
    assets["bullet_img"] = pygame.image.load(path.join(img_dir, "laserRed16.png")).convert()
    assets["background"] = pygame.image.load(path.join(img_dir, 'starfield.png')).convert()
    assets["boom_sound"] = pygame.mixer.Sound(path.join(snd_dir, 'expl3.wav'))
    assets["destroy_sound"] = pygame.mixer.Sound(path.join(snd_dir, 'expl6.wav'))
    assets["pew_sound"] = pygame.mixer.Sound(path.join(snd_dir, 'pew.wav'))
    return assets
```

Coloque o código a seguir na seção de inicialização do jogo.

```
# Inicialização do Pygame.
pygame.init()
pygame.mixer.init()

# Tamanho da tela.
screen = pygame.display.set_mode((WIDTH, HEIGHT))

# Nome do jogo
pygame.display.set_caption("Navinha")

# Carrega todos os assets uma vez só e guarda em um dicionário
assets = load_assets(img_dir, snd_dir)

# Variável para o ajuste de velocidade
clock = pygame.time.Clock()
```

Agora modifique todas as classes para que estas recebam os seus vários *assets* via construtor, ao invés de carregar o asset toda vez que um objeto for criado. Veja o exemplo da classe Player:

```
# Classe Jogador que representa a nave
class Player(pygame.sprite.Sprite):

    # Construtor da classe.
    def __init__(self, player_img):

        # Construtor da classe pai (Sprite).
        pygame.sprite.Sprite.__init__(self)

        # Carregando a imagem de fundo.
        self.image = player_img
```

E mais abaixo, na hora de criar um Player, passamos a imagem que está guardada no dicionário de assets:

```
# Cria uma nave. O construtor será chamado automaticamente.
player = Player(assets["player_img"])
```

Faça o mesmo para as demais classes.

Agora modifique também o código de carga de imagens que existe no corpo principal do jogo:

```
# Carrega o fundo do jogo
background = assets["background"]
background_rect = background.get_rect()

# Carrega os sons do jogo
pygame.mixer.music.load(path.join(snd_dir, 'tgfcoder-FrozenJam-SeamlessLoop.ogg'))
pygame.mixer.music.set_volume(0.4)
boom_sound = assets["boom_sound"]
destroy_sound = assets["destroy_sound"]
pew_sound = assets["pew_sound"]
```

ADICIONANDO ANIMAÇÕES

Uma animação nada mais é do que uma sequência de imagens. Vamos adicionar código para carregar a lista de imagens da animação de explosão do meteoro:

```
# Carrega todos os assets uma vez só.
def load_assets(img_dir, snd_dir):
    assets = {}
    assets["player_img"] = pygame.image.load(path.join(img_dir, "playerShip1_orange.png")).convert()
    assets["mob_img"] = pygame.image.load(path.join(img_dir, "meteorBrown_med1.png")).convert()
    assets["bullet_img"] = pygame.image.load(path.join(img_dir, "laserRed16.png")).convert()
    assets["background"] = pygame.image.load(path.join(img_dir, 'starfield.png')).convert()
    assets["boom_sound"] = pygame.mixer.Sound(path.join(snd_dir, 'expl3.wav'))
    assets["destroy_sound"] = pygame.mixer.Sound(path.join(snd_dir, 'expl6.wav'))
    assets["pew_sound"] = pygame.mixer.Sound(path.join(snd_dir, 'pew.wav'))
    explosion_anim = []
    for i in range(9):
        filename = 'regularExplosion0{}.png'.format(i)
        img = pygame.image.load(path.join(img_dir, filename)).convert()
        img = pygame.transform.scale(img, (32, 32))
        img.set_colorkey(BLACK)
        explosion_anim.append(img)
    assets["explosion_anim"] = explosion_anim
    return assets
```

Agora vamos criar uma nova classe de Sprite para representar uma explosão:

```
# Classe que representa uma explosão de meteoro
class Explosion(pygame.sprite.Sprite):

    # Construtor da classe.
    def __init__(self, center, explosion_anim):
        # Construtor da classe pai (Sprite).
        pygame.sprite.Sprite.__init__(self)

        # Carrega a animação de explosão
        self.explosion_anim = explosion_anim

        # Inicia o processo de animação colocando a primeira imagem na tela.
        self.frame = 0
        self.image = self.explosion_anim[self.frame]
        self.rect = self.image.get_rect()
        self.rect.center = center

        # Guarda o tick da primeira imagem
        self.last_update = pygame.time.get_ticks()

        # Controle de ticks de animação: troca de imagem a cada self.frame_ticks milissegundos.
        self.frame_ticks = 50
```

```
    def update(self):
        # Verifica o tick atual.
        now = pygame.time.get_ticks()

        # Verifica quantos ticks se passaram desde a ultima mudança de frame.
        elapsed_ticks = now - self.last_update

        # Se já está na hora de mudar de imagem...
        if elapsed_ticks > self.frame_ticks:

            # Marca o tick da nova imagem.
            self.last_update = now

            # Avança um quadro.
            self.frame += 1

            # Verifica se já chegou no final da animação.
            if self.frame == len(self.explosion_anim):
                # Se sim, tchau explosão!
                self.kill()
            else:
                # Se ainda não chegou ao fim da explosão, troca de imagem.
                center = self.rect.center
                self.image = self.explosion_anim[self.frame]
                self.rect = self.image.get_rect()
                self.rect.center = center
```

Estude como funciona essa classe. Em particular, note o mecanismo de atualização de quadros da animação no método *update()*. Por que precisamos guardar o tick do ultimo update em *self.last_update*? Não bastaria uma variável local *last_update*?

Bom, carregamos a animação e criamos uma classe de Sprite para exibir essa animação. Falta usar isso tudo! Adicione o seguinte código no momento de tratamento de colisão entre tiros e meteoros:

```
# Verifica se houve colisão entre tiro e meteoro
hits = pygame.sprite.groupcollide(mobs, bullets, True, True)
for hit in hits: # Pode haver mais de um
    # O meteoro e destruido e precisa ser recriado
    destroy_sound.play()
    m = Mob(assets["mob_img"])
    all_sprites.add(m)
    mobs.add(m)

    # No lugar do meteoro antigo, adicionar uma explosão.
    explosao = Explosion(hit.rect.center, assets["explosion_anim"])
    all_sprites.add(explosao)
```

EXPLODINDO A NAVE

Agora os meteoros estão bem legais, mas e a nave? Vamos agora introduzir um truque super útil para jogos (e vários outros programas de computador): a máquina de estados.

Compare as diferenças entre seu código atual e o código em *asteroids-7.py*. Para comparar diferenças entre códigos, uma ferramenta de *diff* muito útil é o Meld (http://meldmerge.org/) que funciona em Windows e Linux. Usuários de Mac tem uma ferramenta nativa: o FileMerge.

Observe que no novo arquivo mudamos o mecanismo do loop de jogo. Antes usávamos uma variável booleana para indicar se o jogo estava rodando ou não. Agora temos uma variável inteira que indica se estamos rodando o jogo normalmente, se estamos em estado de explosão da nave, ou se o jogo acabou.

Se você quisesse que o jogo nunca acabasse, mas que ao término da explosão da nave tivéssemos o ressurgimento da nave, como você deveria alterar o jogo?

PLACAR

O arquivo asteroids-8.py mostra como carregar uma fonte (estilo de letra) com jeitão de arcade dos anos 80, e como registrar a pontuação do jogador (use sua ferramenta de *diff*).

Atividade: Modifique o código para que a cada 10000 pontos a mensagem “Awesome!” apareça no centro da tela por 1 segundo, sem interromper o fluxo de jogo!

VIDAS

O arquivo asteroids-9.py inclui um mecanismo de vidas. Como você alteraria o programa para ganhar automaticamente uma vida a cada 100000 pontos?

CONSTRUINDO MÚLTIPLAS TELAS

Este código está ficando muito grande! Está na hora de quebrar esse monolito em funções simples. Isso vai facilitar o desenvolvimento de novas telas de jogo.

A atividade de reorganizar o código sem adicionar nenhuma feature, mas melhorando a estrutura do programa para futuros desenvolvimentos é chamada de *refatoração*.

Observe no arquivo asteroids-10.py como o corpo principal do jogo foi movido para a função *game_screen()*.

Agora, no arquivo asteroids-11.py, o corpo principal do jogo foi transferido para um arquivo separado, facilitando o desenvolvimento concorrente do produto pelos membros da equipe.

Por fim, no arquivo asteroids-12.py vemos como essa refatoração ajudou a desenvolver uma nova tela de jogo! Vemos também o truque da máquina de estados para organizar a aplicação principal.