Loops de Animação e Pulos Utilizando a Pyxel

Prof. Alessandro de Lima Bicho, Prof. Cleo Zanella Billa, Thiago Novinski Machado

[1]Centro de Ciências Computacionais – Universidade Federal do Rio Grande (FURG)

Resumo. *Este documento apresenta o processo de implementação de animações para movimentos de um personagem em um jogo utilizando a biblioteca Pyxel. São abordados conceitos de animação quadro a quadro, controle de direção, e integração de física para simular o pulo, com o objetivo de auxiliar os alunos na criação de jogos interativos.*

1. Introdução

Animações e mecânicas de movimento são componentes fundamentais no desenvolvimento de jogos, adicionando realismo e interatividade à experiência do jogador. Este documento ilustra a criação de um personagem que anda e pula em um ambiente bidimensional utilizando Pyxel.

2. Estrutura do Personagem

O personagem é representado como uma classe contendo atributos de posição, tamanho, velocidade e estado atual. A implementação é baseada em três estados principais:

- **Parado:** O personagem está imóvel.
- **Andando:** O personagem se move horizontalmente.
- **Pulando:** O personagem está em movimento vertical devido a um pulo.

Atributos do Personagem

- **x, y:** Coordenadas que definem a posição do personagem na tela.
- **velocidade:** Determina a rapidez dos movimentos horizontais.
- **velocidade_vertical:** Controla o movimento vertical durante o pulo.
- **estado:** Indica o comportamento atual ("parado", "andando" ou "pulando").
- **direcao:** Define para qual lado o personagem está virado ("esquerda" ou "direita").
- **frame:** Controla o quadro atual da animação.

3. Animação de Movimentação

A animação é realizada alternando quadros em um *spritesheet*, onde cada quadro corresponde a uma imagem representando uma etapa do movimento. É recomendado que todos os *frames* de animação tenham o mesmo tamanho para evitar problemas, como os observados no arquivo *Animacao(GatinhoBugado)*, onde diferenças de tamanhos causaram bugs inesperados.

Configuração dos Quadros

Cada estado do personagem é representado por uma linha específica do *spritesheet*, percorrida conforme a animação:

Parado: Representa a primeira linha do spritesheet (gatinho balança a cauda).

Andando: Representa a segunda linha do spritesheet (gatinho caminha).

Pulando: Representa a terceira linha do spritesheet (gatinho sobe ou desce conforme a velocidade vertical)

Figura1. Spritesheet dos gatinhos

Desenho do Personagem

A função *desenhar* utiliza o método *pyxel.blt()* para renderizar os quadros apropriados. Mas como exatamente funciona o método *BLT*?

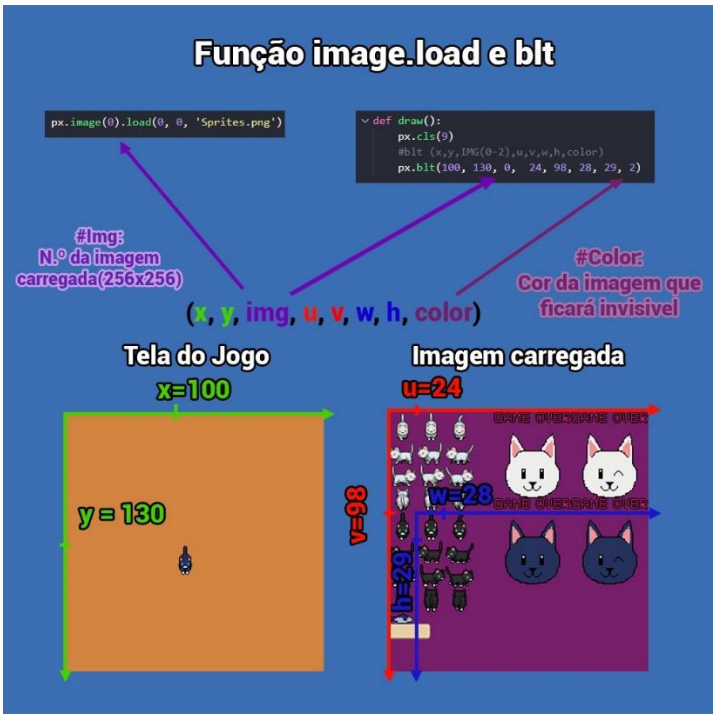

Figura 2. Ilustração função BLT

Antes de utilizar blt, é necessário carregar um *spritesheet* em um dos três bancos de imagem disponíveis na biblioteca Pyxel (0, 1 ou 2).

```
def __init__(self):
    pyxel.init(160, 120, fps=10, title="Animações")
    pyxel.image(0).load(0, 0, "spritesGatinhosExpandido.png")
```

Figura 3. Uso das funções Image e Load

Parâmetros da Função BLT

A função blt possui oito parâmetros, cuja complexidade pode causar confusão aos usuários. A seguir, descrevemos cada um deles:

- **X:** Posição horizontal inicial onde o sprite será desenhado na tela do jogo.
- **Y:** Posição vertical inicial onde o sprite será desenhado na tela do jogo.
- **Img:** Banco de imagem onde o *spritesheet* foi carregado.
- **U:** Coordenada horizontal inicial da área que será extraída do spritesheet.
- **V:** Coordenada vertical inicial da área que será extraída do spritesheet.
- **W:** Largura do trecho extraído do spritesheet a partir de U.
- **H:** Altura do trecho extraído do spritesheet a partir de V.
- **Color:** Cor que será ignorada no sprite, deixando-a transparente.

Tendo isso em mente, podemos implementar o loop de animação. Considerando que os sprites tenham tamanhos uniformes ou estejam distribuídos regularmente no spritesheet, o loop de animação pode ser controlado por uma variável, como o atributo *frame* da classe *Personagem*. Essa variável é multiplicada pela largura do sprite (*W*) para ajustar o parâmetro *U*. Quando o valor de frame atinge o máximo (último quadro da linha), ele é redefinido para 0.

```
def desenhar(self):
    if self.estado == "parado":
        # Determina o tamanho do sprite com base na direção
        if self.direcao == "direita":
            self.tamanhoSprite = 17
        elif self.direcao == "esquerda":
            self.tamanhoSprite = -17  # Sprite espelhado para esquerda

        # Avança para o próximo quadro da animação
        self.frame += 1
        if self.frame > 7:  # Reseta ao completar a animação
            self.frame = 0

        # Desenha o quadro atual do personagem
        pyxel.blt(self.x, self.y, 0, 17 * self.frame, 0, self.tamanhoSprite, 17, 2)
```

Figura 4. Método Desenhar

É importante observar que, ao utilizar valores negativos para o parâmetro *W* em *BLT*, o sprite será desenhado de forma espelhada. Essa técnica é útil para economizar espaço no spritesheet, eliminando a necessidade de duplicar sprites que diferem apenas pela orientação.

4. Implementação do Pulo

O pulo é modelado utilizando conceitos básicos de física, incluindo uma velocidade inicial e a aplicação gradual de gravidade.

Física do Pulo

Subida: A velocidade vertical é inicializada com um valor negativo (por conta da orientação do plano), movendo o personagem para cima.

Gravidade: Um incremento positivo é adicionado à velocidade vertical em cada quadro.

Queda: O personagem desce até atingir o solo, onde o movimento vertical é interrompido.

Funções Implementadas

pular(): Inicia o pulo, alterando o estado e a velocidade vertical.

atualizar_pulo(): Aplica a gravidade e ajusta a posição vertical do personagem.

```
    # Inicia o pulo se o personagem estiver no chão e a tecla de pulo for pressionada
    if pyxel.btnp(pyxel.KEY_UP) and self.no_chao:
        self.pular()

    # Atualiza a física do pulo (gravidade e movimento vertical)
    self.atualizar_pulo()

def pular(self):
    # Configura os parâmetros para o pulo
    self.estado = "pulando"
    self.velocidade_vertical = -4  # Velocidade inicial do pulo (subida)
    self.frame = -1  # Reinicia o quadro da animação de pulo
    self.no_chao = False  # Personagem não está mais no chão

def atualizar_pulo(self):
    # Gerencia a física do pulo
    if self.estado == "pulando":
        self.y += self.velocidade_vertical  # Atualiza a posição vertical
        self.velocidade_vertical += 0.5  # Aplica gravidade

        # Detecta o contato com o chão
        if self.y >= 100 - self.altura:
            self.y = 100 - self.altura  # Ajusta a posição no chão
            self.no_chao = True  # Permite pular novamente
            self.estado = "parado"  # Retorna ao estado parado
```

Figura 5. Implementação do Pulo

5. Movimentação Horizontal

A movimentação horizontal responde às teclas de direção e é condicionada ao estado atual do personagem. Se o personagem estiver pulando, a direção ainda pode ser alterada, mas o estado permanece o mesmo.

```python
def mover(self):
    # Controla o movimento e atualiza o estado do personagem
    if pyxel.btn(pyxel.KEY_LEFT):
        if self.estado != "pulando":  # Evita alterar o estado durante o pulo
            self.estado = "andando"
        self.direcao = "esquerda"
        self.x -= self.velocidade
    elif pyxel.btn(pyxel.KEY_RIGHT):
        if self.estado != "pulando":
            self.estado = "andando"
        self.direcao = "direita"
        self.x += self.velocidade
    else:
        if self.estado != "pulando":
            self.estado = "parado"
```

Figura 6. Implementação Movimentação

6. Escalabilidade

A lógica apresentada pode ser expandida para incluir obstáculos, interações com plataformas móveis ou elementos coletáveis. Além disso, a modularidade do código permite fácil manutenção e integração com outras mecânicas de jogo.

7. Considerações Finais

Este documento apresentou uma abordagem prática para implementar animações e mecânicas de pulo utilizando a biblioteca Pyxel. Os conceitos discutidos são aplicáveis em jogos simples e podem ser escalados para projetos mais complexos. Com esta base, espera-se que os alunos desenvolvam suas habilidades em programação de jogos e aprendam a estruturar projetos de maneira eficiente e criativa.

8. Arquivos Disponíveis e Referências

Os arquivos de exemplo evoluem em sua construção na sequência: *Animacao(GatinhoBugado).py*, *Animacao(MetodoExpansao).py*, *Animacao(Andar).py*, *Animacao(Pular).py*. Lembrando que para o funcionamento visual desses códigos eles devem estar na mesma pasta que as imagens *spritesGatinhosColados.png* e *spritesGatinhosExpandido.png*.

Recorte das sprites utilizadas retiradas de:

https://bowpixel.itch.io/cat-50-animations?download

Recomendação de repositórios de sprites gratuitas:

https://itch.io/game-assets/free