Verificação de Colisões Utilizando a Biblioteca 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 visa auxiliar os alunos da matéria de Algoritmos e Estruturas de Dados I a compreender a detecção de colisões dentro da biblioteca Pyxel, a fim de facilitar o desenvolvimento de seus projetos.*

1. Introdução

A detecção de colisões é um conceito fundamental no desenvolvimento de jogos, permitindo identificar quando dois ou mais objetos no ambiente do jogo ocupam a mesma área ou interagem de alguma forma. Essa técnica é essencial para criar mecânicas interativas, como movimentos de personagens, interações com obstáculos, disparos e eventos, como coleta de itens. Além de conferir lógica e realismo às interações do jogo, a detecção de colisões contribui para a dinâmica e a experiência do usuário.

2. Definição dos Limites dos Retângulos

A representação dos objetos no jogo, como personagens e obstáculos, é feita por meio de retângulos definidos pelas coordenadas do canto superior esquerdo (x, y) e suas dimensões (largura e altura).

Personagem

- **EsquerdaPersonagem**: x do canto esquerdo do personagem.
- **DireitaPersonagem**: x do canto direito do personagem (self.x + self.largura).
- **SuperiorPersonagem**: y do topo do personagem.
- **InferiorPersonagem**: y da base do personagem (self.y + self.altura).

Obstáculo

- **EsquerdaObstaculo**: x do canto esquerdo do obstáculo.
- **DireitaObstaculo**: x do canto direito do obstáculo (obstaculo.x + obstaculo.largura).
- **SuperiorObstaculo**: y do topo do obstáculo.
- **InferiorObstaculo**: y da base do obstáculo (obstaculo.y + obstaculo.altura).

```
def verificar_colisao(self, obstaculo):
    # Verifica colisão entre o personagem (Retangulo) e o obstáculo (retângulo)
    #personagem
    EsquerdaPersonagem = self.x
    DireitaPersonagem = self.x + self.largura
    SuperiorPersonagem = self.y
    InferiorPersonagem = self.y + self.altura

    #obstaculo
    EsquerdaObstaculo = obstaculo.x
    DireitaObstaculo = obstaculo.x + obstaculo.largura
    SuperiorObstaculo = obstaculo.y
    InferiorObstaculo = obstaculo.y + obstaculo.altura
```

Figura 1. Definindo Limites dos Objetos

3. Condição de Colisão

Uma colisão ocorre quando os retângulos de dois objetos se sobrepõem em ambos os eixos, obedecendo às seguintes condições:

- O lado direito do personagem ultrapassa o lado esquerdo do obstáculo: **DireitaPersonagem > EsquerdaObstaculo**
- O lado esquerdo do personagem está antes do lado direito do obstáculo: **EsquerdaPersonagem < DireitaObstaculo**
- A base do personagem está abaixo do topo do obstáculo: **InferiorPersonagem > SuperiorObstaculo**
- O topo do personagem está acima da base do obstáculo: **SuperiorPersonagem < InferiorObstaculo**

Se todas essas condições forem verdadeiras, há uma colisão, e o método retorna **True**. Caso contrário, retorna **False**.

```
if (DireitaPersonagem > EsquerdaObstaculo and
    EsquerdaPersonagem < DireitaObstaculo and
    InferiorPersonagem > SuperiorObstaculo and
    SuperiorPersonagem < InferiorObstaculo):
    return True
else:
    False
```

Figura 2. Condição de Colisão

4. Colisão Aplicada à Obstáculos

Em jogos, é comum que obstáculos, como plataformas e paredes, limitem a movimentação dos personagens. Para implementar essa funcionalidade, é necessário calcular a colisão considerando o ponto futuro da posição do personagem. Por exemplo, ao mover-se para a esquerda, verifica-se o limite esquerdo do personagem subtraindo sua velocidade (passo).

É importante lembrar que, na função de movimentação, os retornos de **True** e **False** precisam ser invertidos. Isso ocorre porque uma colisão indica que o personagem não pode se mover naquela direção.

```
def verifica_pode_andar(self, obstaculo, direcao):
    # Ajusta os limites do personagem com base na direção do movimento
    if direcao == "esquerda":
        EsquerdaPersonagem = self.x - self.velocidade
        DireitaPersonagem = self.x
        SuperiorPersonagem = self.y
        InferiorPersonagem = self.y + self.altura
```

Figura 3. Exemplo Movimentação Esquerda

```
# Verifica se há colisão
if (DireitaPersonagem > EsquerdaObstaculo and
    EsquerdaPersonagem < DireitaObstaculo and
    InferiorPersonagem > SuperiorObstaculo and
    SuperiorPersonagem < InferiorObstaculo):
    return False
else:
    return True
```

Figura 4. Retornos Colisão de Movimentação

A implementação da verificação de colisão dentro da função de movimentação do personagem está disponível no arquivo **"ColisaoObstaculo.py"**.

```
def mover(self, obstaculo):
    if pyxel.btn(pyxel.KEY_LEFT) and self.verifica_pode_andar(obstaculo, "esquerda"):
        self.x -= self.velocidade
    if pyxel.btn(pyxel.KEY_RIGHT) and self.verifica_pode_andar(obstaculo, "direita"):
        self.x += self.velocidade
    if pyxel.btn(pyxel.KEY_UP) and self.verifica_pode_andar(obstaculo, "cima"):
        self.y -= self.velocidade
    if pyxel.btn(pyxel.KEY_DOWN) and self.verifica_pode_andar(obstaculo, "baixo"):
        self.y += self.velocidade
```

Figura 5. Utilização da Função de Colisão Aplicado à Função de Movimentação

5. Escalabilidade

Em jogos com múltiplos obstáculos, a verificação de colisões deve ser eficiente para garantir a viabilidade do processamento. A programação orientada a objetos é essencial nesse contexto, permitindo que cada obstáculo seja representado como um objeto. Assim, é possível armazenar todos os obstáculos em uma lista, realizando a checagem com todos os elementos presentes na cena.

```
class Jogo:
    # Classe principal do jogo
    def __init__(self):
        self.personagem = Personagem(80, 60, 4, 4, 1, 4)  # Inicializa o jogador
        self.obstaculo1 = Obstaculo(50, 50, 20, 5, 1)
        self.obstaculo2 = Obstaculo(110, 10, 30, 10, 1)
        self.obstaculo3 = Obstaculo(100, 50, 20, 30, 1)
        self.listaObstaculos = [self.obstaculo1, self.obstaculo2, self.obstaculo3]
        pyxel.init(160, 120, title="Jogo com Obstáculos")
        pyxel.run(self.update, self.draw)

    def update(self):
        # Atualiza o estado do jogo
        self.personagem.mover(self.listaObstaculos)

    def draw(self):
        # Desenha os elementos na tela
        pyxel.cls(12)
        for obstaculo in self.listaObstaculos:
            obstaculo.desenhar()
        self.personagem.desenhar()
```

Figura 6. Utilização de Listas de Obstáculos

Perceba que para que isso seja possível, devemos realizar algumas alterações nos métodos *mover* e *verifica_pode_andar*. A função *verifica_pode_andar* é apenas um loop de que chama a função de verificação já implementado anteriormente nas figuras 3 e 4.

```
def verifica_pode_andar(self, lista_obstaculos, direcao):
    # Verifica se o personagem pode andar sem colidir com nenhum obstáculo
    for obstaculo in lista_obstaculos:
        if not self._verifica_pode_andar_obstaculo(obstaculo, direcao):
            return False  # Colidiu com pelo menos um obstáculo
    return True

def mover(self, lista_obstaculos):
    # Movimenta o personagem apenas se não houver colisão em nenhuma direção
    if pyxel.btn(pyxel.KEY_LEFT) and self.verifica_pode_andar(lista_obstaculos, "esquerda"
        self.x -= self.velocidade
    if pyxel.btn(pyxel.KEY_RIGHT) and self.verifica_pode_andar(lista_obstaculos, "direita"
        self.x += self.velocidade
    if pyxel.btn(pyxel.KEY_UP) and self.verifica_pode_andar(lista_obstaculos, "cima"):
        self.y -= self.velocidade
    if pyxel.btn(pyxel.KEY_DOWN) and self.verifica_pode_andar(lista_obstaculos, "baixo"):
        self.y += self.velocidade

def _verifica_pode_andar_obstaculo(self, obstaculo, direcao):
    # Ajusta os limites do personagem com base na direção do movimento
    if direcao == "esquerda":
        EsquerdaPersonagem = self.x - self.velocidade
        DireitaPersonagem = self.x
        SuperiorPersonagem = self.y
        InferiorPersonagem = self.y + self.altura
    elif direcao == "direita":
        EsquerdaPersonagem = self.x + self.largura
        DireitaPersonagem = self.x + self.largura + self.velocidade
        SuperiorPersonagem = self.y
        InferiorPersonagem = self.y + self.altura
    elif direcao == "cima":
        EsquerdaPersonagem = self.x
        DireitaPersonagem = self.x + self.largura
        SuperiorPersonagem = self.y - self.velocidade
        InferiorPersonagem = self.y
    elif direcao == "baixo":
        EsquerdaPersonagem = self.x
```

Figura 7. Adaptações para Múltiplos Obstáculos

6. Considerações Finais

A detecção de colisões é uma ferramenta indispensável no desenvolvimento de jogos, possibilitando interações coerentes e envolventes entre os objetos do ambiente. Este documento apresentou uma abordagem prática para implementar essa funcionalidade utilizando a biblioteca Pyxel, desde a definição dos limites dos objetos até a escalabilidade para múltiplos obstáculos.

Ao aplicar os conceitos discutidos, os alunos poderão estruturar seus projetos de maneira eficiente e modular, facilitando a manutenção e a expansão de suas aplicações. A integração da detecção de colisões com outros elementos do jogo, como física e inteligência artificial, pode ser explorada em projetos futuros, incentivando a criatividade e o aprendizado contínuo.

Todos os exemplos utilizados neste documento estarão disponíveis na ordem: Colisao.py, ColisaoObstaculo.py e ColisaoObstaculosVariados.py.