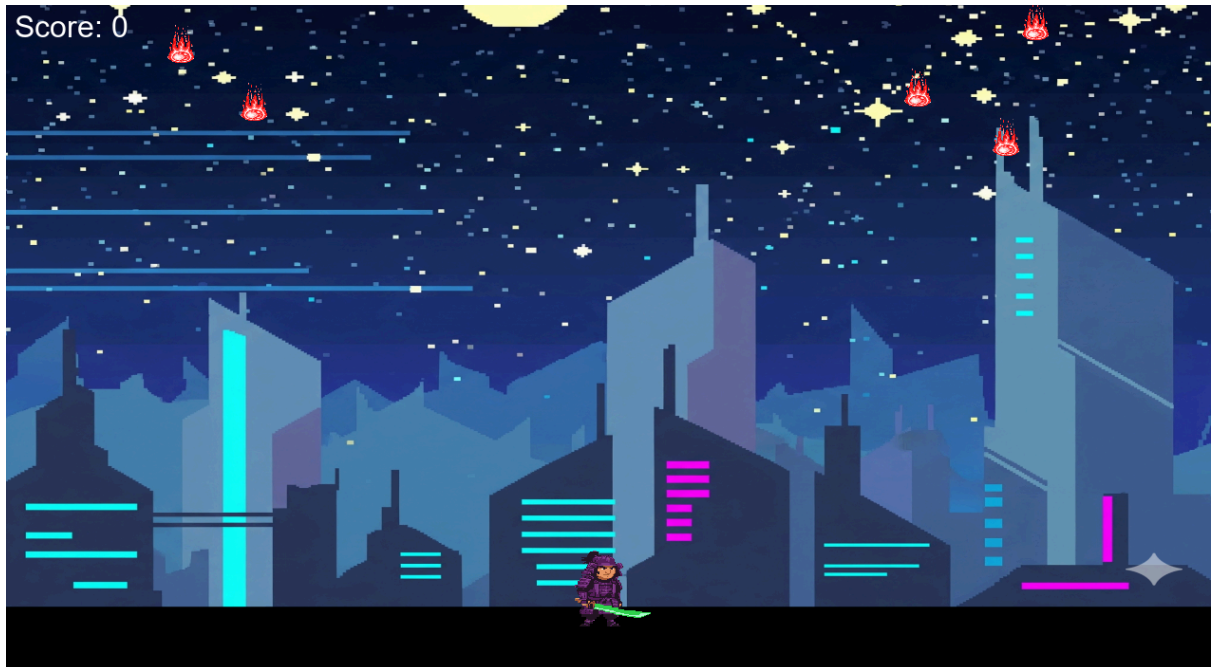


Explicação do Código com Conceitos de Computação Gráfica



1. Inicialização e criação da janela

```
pygame.init()
screen = pygame.display.set_mode((800, 600))
pygame.display.set_caption("Protótipo Gráfico com Música")
```

- **Surface/Framebuffer:** a `screen` é a **Surface principal** do jogo, que representa um **framebuffer** – onde os pixels serão desenhados antes de serem exibidos.
- **Raster (ou Bitmap):** internamente, essa `Surface` é uma **matriz de pixels (bitmap)** onde cada pixel é representado no **modelo de cor RGB (0–255 por canal)**.

2. Sistema de coordenadas e jogador

```
player = pygame.Rect(400, 500, 50, 50)
```

- **Sistema de coordenadas 2D:** em Pygame, a origem `(0, 0)` está no canto superior esquerdo, com **x aumentando para a direita** e **y aumentando para baixo**.
 - **Modelagem 2D:** usamos formas geométricas simples (um retângulo) para representar objetos.
 - **Translação:** quando alteramos `player.x += 5`, estamos **transladando** o objeto no eixo X.
-

3. Entrada de teclado (input)

```
keys = pygame.key.get_pressed()
if keys[pygame.K_LEFT]:
    player.x -= 5
if keys[pygame.K_RIGHT]:
    player.x += 5
```

- **Loop de renderização:** `input` → `update` → `render` → `display`: aqui temos a parte do `input`.
 - Alteramos a posição do jogador (translação no espaço 2D).
-

4. Objetos que caem (estímulo sonoro)

```
objects = []
objects.append(pygame.Rect(random.randint(0, 750), 0, 50, 50))
```

- **Rasterização:** ao desenharmos o retângulo, ele será convertido em pixels na `Surface`.
 - Esses objetos são **entidades modeladas em 2D**, com posição `(x, y)` e dimensões `(w, h)`.
-

5. Colisão

```
if player.colliderect(obj):
    objects.remove(obj)
```

- **Retângulos/colisão (AABB):** usamos `pygame.Rect` com **Axis-Aligned Bounding Box (AABB)**, que detecta se dois retângulos se sobrepõem.
 - Muito usado em jogos por ser eficiente.
-

6. Renderização (desenho na tela)

```
screen.fill((0, 0, 0))
pygame.draw.rect(screen, (0, 255, 0), player)
pygame.draw.rect(screen, (255, 0, 0), obj)
```

- **Modelo de cor RGB (0–255 por canal):** `(0, 255, 0)` = verde (player), `(255, 0, 0)` = vermelho (objetos).
 - **Contraste/legibilidade:** o fundo preto `(0, 0, 0)` aumenta a legibilidade dos objetos coloridos.
 - **Rasterização + Blitting:** `draw.rect` escreve diretamente os pixels na `Surface`.
-

7. Texto

```
font = pygame.font.SysFont(None, 36)
text = font.render("Score: 10", True, (255, 255, 255))
screen.blit(text, (10, 10))
```

- **Texto e antialiasing:** o `True` aplica **antialiasing**, suavizando as bordas das letras.
 - **Blitting:** `screen.blit(text, (10, 10))` copia o **bitmap do texto** para a superfície principal.
-

8. Ordem de desenho

```
screen.fill((0, 0, 0))
pygame.draw.rect(screen, (0, 255, 0), player)
pygame.draw.rect(screen, (255, 0, 0), obj)
```

- **Painter's algorithm:** os objetos são desenhados **na ordem em que os comandos aparecem**. Se invertêssemos, o player poderia ficar escondido atrás dos objetos.
-

9. Loop principal

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

    # input
    # update
    # render

    pygame.display.flip()
    clock.tick(60)
```

- **Loop de renderização:**
 - **Input:** capturamos teclas e eventos.
 - **Update:** atualizamos posições dos objetos.
 - **Render:** desenhamos tudo na **Surface**.
 - **Display.flip():** troca o **back buffer** pelo **front buffer** → isso é o **double buffering**, evitando flickering (tremulação).
- **Controle de FPS:** `clock.tick(60)` limita a taxa de quadros para **60 FPS**.