config.py

```
"""Configurações"""
LARGURA_JANELA = 960
ALTURA_JANELA = 540
QUADROS_POR_SEGUNDO = 60
COMPRIMENTO_NIVEL = 6000
```

camera.py

```
import pygame
"""Camera"""
class Camera:
    def __init__(self, largura: int, altura: int, largura_janela:int):
        self.deslocamento_x = 0
        self.largura = largura
        self.altura = altura
        self.largura_janela = largura_janela
    def seguir(self,retangulo_alvo: pygame.Rect):
        "Centralizar a Camera com o Personagem"
        centro_alvo_x = retangulo_alvo.centerx
        self.deslocamento_x = max(0, self.largura -
self.largura_janela)
        desloc_maximo = max(0, self.largura - self.largura_janela)
        if self.deslocamento_x > desloc_maximo:
            self.deslocamento_x = desloc_maximo
    def aplicar(self, retangulo: pygame.Rect) -> pygame.Rect:
        return pygame.Rect(retangulo.x - self.deslocamento_x,
retangulo.y,
                           retangulo.width, retangulo.height)
```

util_imagens.py

```
import pygame
"""Carrega as imagens"""
def carregar_imagem(caminho: str, altura_escala: int | None=None) ->
pygame.Surface:
    """Calcula Proporção"""
    imagem = pygame.image.load(caminho).convert_alpha()
    if altura_escala is None:
        return imagem
    largura_atual, altura_atual = imagem.get_size()
    proporção = altura_escala / altura_atual
```

```
    novo_tamanho = (int(largura_atual * proporção), int(altura_atual *
proporção))
    return pygame.transform.smoothcale(imagem, novo_tamanho)
```

cenario.py

```
import os
import pygame
from util_imagem import carregar_imagem
"""Constroi o cenario"""
def construir_cenario(dir_arquivos: str, altura_janela: int,
comprimento_nivel: int) -> tuple[list[pygame.Rect],
list[tuple[pygame.Surface, pygame.Rect]]]
solidos: list[pygame.Rect] = []
decoraçoes = list[tuple[pygame.Surface, pygame.Rect]]= []
chao = pygame.Rect(0, altura_altura - 64 conprimento_nivel, 64)
solidos.append(chao)
caminhos_tijolo = os.path.join(dir_arquivos, "tijolo,png")
img_tijolo = carregar_imagem(caminhos_tijolo, altura_escala = 48)
plataforma_tijolo = [
    (300, altura_janela = 200, 5)
    (900, altura_janela = 260, 4)
    (1400, altura_janela = 180, 6)
]
for inicio_x
```

inimigo.py

```
import os
import pygame
from util_imagem import carregar_imagem
""" Inimigos com patrulha horizontal """
class Inimigo(pygame.sprite.Sprite):
    def __init__(self, posicao_inicial: tuple[int, int], dir_arquivos:
str, limites_patrulha: tuple[int, int]):
        super().__init__()
        caminho_planta = os.path.join(dir_arquivos, "Piranha
Plant.png")
        self.imagem = carregar_imagem(caminho_planta, altura_escala=64)
        self.retangulo = self.imagem.get_rect(topleft=posicao_inicial)
        self.direcao = 1
        self.velocidade = 2
```

```
        self.limites_patrulha = limites_patrulha

    def patrulhar(self):
        esquerdo, direito = self.limites_patrulha
        self.retangulo.x += self.direcao * self.velocidade
        if self.retangulo.left <= esquerdo:
            self.retangulo.left = esquerdo
            self.direcao = 1
        elif self.retangulo.right >= direito:
            self.retangulo.right = direito
            self.direcao = -1

class Inimigo2(Inimigo):
    def __init__(self, posicao_inicial: tuple[int, int], dir_arquivos:
str, limites_patrulha: tuple[int, int]):
        super().__init__(posicao_inicial, dir_arquivos,
limites_patrulha)
        caminho_img = os.path.join(dir_arquivos, "inimigo2.jpeg")
        self.imagem = carregar_imagem(caminho_img, altura_escala=80)
        base = self.retangulo.bottomleft
        self.retangulo = self.imagem.get_rect()
        self.retangulo.bottomleft = base
        self.velocidade = 3
```

placar.py

```
import os
""" Leitura, escrita e formatação de tempos do placar """
def ler_tempos(caminho: str) -> list[tuple[int, str]]:
    if not os.path.exists(caminho):
        return []
    resultados: list[tuple[int, str]] = []
    try:
        with open(caminho, "r", encoding="utf-8") as f:
            for linha in f:
                partes = linha.strip().split(";")
                if len(partes) == 2:
                    try:
                        ms = int(partes[0])
                        nome = partes[1]
                        resultados.append((ms, nome))
                    except ValueError:
                        pass
```

```
    except OSError:
        return []
    return resultados

def salvar_tempo(caminho: str, ms: int, nome: str) -> None:
    try:
        with open(caminho, "a", encoding="utf-8") as f:
            f.write(f"{ms};{nome}\n")
    except OSError:
        pass

def top5(registros: list[tuple[int, str]]) -> list[tuple[int, str]]:
    return sorted(registros, key=lambda x: x[0])[:5]

def formatar_ms(ms: int) -> str:
    s = ms // 1000
    ms_rem = ms % 1000
    return f"{s}.{ms_rem:03d}s"
```

util imagem

```
import os
import pygame
from util_imagem import carregar_imagem


""" Entidade do jogador e sua física simples """


class Jogador(pygame.sprite.Sprite):
    def __init__(self, posicao_inicial: tuple[int, int], dir_arquivos: str):
        super().__init__()
        caminho_mario = os.path.join(dir_arquivos, "mario.gif")
        self.imagem = carregar_imagem(caminho_mario, altura_escala=64)
        self.retangulo = self.imagem.get_rect(topleft=posicao_inicial)
        self.velocidade_x = 0.0
        self.velocidade_y = 0.0
        self.no_chao = False

        self.aceleracao_chao = 0.8
        self.aceleracao_ar = 0.5
        self.atrito = 0.85
        self.gravidade = 0.9
        self.forca_pulo = -17
        self.velocidade_maxima_x = 8
```

```
        self.velocidade_maxima_queda = 22

    def ler_entrada(self):
        teclas = pygame.key.get_pressed()
        desejado = 0
        if teclas[pygame.K_LEFT] or teclas[pygame.K_a]:
            desejado -= 1
        if teclas[pygame.K_RIGHT] or teclas[pygame.K_d]:
            desejado += 1
        aceleracao = self.aceleracao_chao if self.no_chao else self.aceleracao_ar
        self.velocidade_x += desejado * aceleracao
        if desejado == 0 and self.no_chao:
            self.velocidade_x *= self.atrito
        if self.velocidade_x > self.velocidade_maxima_x:
            self.velocidade_x = self.velocidade_maxima_x
        if self.velocidade_x < -self.velocidade_maxima_x:
            self.velocidade_x = -self.velocidade_maxima_x

    def tentar_pular(self):
        teclas = pygame.key.get_pressed()
        if (teclas[pygame.K_SPACE] or teclas[pygame.K_UP] or teclas[pygame.K_w]) and
self.no_chao:
            self.velocidade_y = self.forca_pulo
            self.no_chao = False

    def aplicar_gravidade(self):
        self.velocidade_y += self.gravidade
        if self.velocidade_y > self.velocidade_maxima_queda:
            self.velocidade_y = self.velocidade_maxima_queda

    def mover_e_colidir(self, solidos: list[pygame.Rect]):
        self.retangulo.x += int(self.velocidade_x)
        for bloco in solidos:
            if self.retangulo.colliderect(bloco):
                # indo para a direita
                if self.velocidade_x > 0:
                    self.retangulo.right = bloco.left
                # indo para a esquerda
                elif self.velocidade_x < 0:
                    self.retangulo.left = bloco.right
                self.velocidade_x = 0
        self.retangulo.y += int(self.velocidade_y)
        self.no_chao = False
        for bloco in solidos:
            if self.retangulo.colliderect(bloco):
                # caindo
                if self.velocidade_y > 0:
                    self.retangulo.bottom = bloco.top
```

```
                self.velocidade_y = 0
                self.no_chao = True
            # subindo
            elif self.velocidade_y < 0:
                self.retangulo.top = bloco.bottom
                self.velocidade_y = 0

    def atualizar(self, solidos: list[pygame.Rect]):
        self.ler_entrada()
        self.tentar_pular()
        self.aplicar_gravidade()
        self.mover_e_colidir(solidos)
```