PLATFORMER GAME - GDD

Documento de Design de Jogo (GDD)

Campo	Detalhes
Autor(a)	Jane Nogueira dos Santos
Projeto	Processo Seletivo Kodland

-----1. Visão Geral do Jogo

Este documento detalha a estrutura, mecânicas, lógica e implementação de código do **"Platformer Game"** desenvolvido em Pygame Zero.1.1. Resumo do Projeto

Característica	Detalhes
Título	Platformer Game
Gênero	Plataforma 2D
Plataforma	Pygame Zero (Python)
Objetivo	O jogador deve navegar pelas plataformas, evitar os inimigos móveis e alcançar a última plataforma no topo do nível para vencer.
Estado	Protótipo Funcional (Inclui Menu, Jogando, Game Over e Vitória)

-----2. Configurações e Variáveis Globais

Estas constantes definem o ambiente do jogo e as regras físicas.2.1. Configurações da Janela

As constantes definem o tamanho da tela e o título do jogo.

- `WIDTH = 512`
- `HEIGHT = 512`
- `TITLE = "Platformer Game"`

2.2. Estado do Jogo

Variáveis de controle do estado e áudio.

- `game_state`: Controla a tela atual. Valores possíveis: `"MENU"`, `"PLAYING"`, `"GAME_OVER"`, `"WIN"`.

- `music_on`: Booleano que controla a reprodução de música.

2.3. Constantes Físicas

Essas variáveis definem as regras de movimento e física do jogo.

Variável	Valor	Função no Jogo
GRAVITY	0.5	Força que puxa o herói para baixo a cada *frame*.
JUMP_VELOCITY	-11	Velocidade vertical inicial para o pulo (valor negativo = movimento para cima).
MOVE_SPEED	10	Velocidade de movimento horizontal do herói.
ANIMATION_SPEED	10	Define a frequência de troca de *frames* para animações.

-----3. Classes de Entidades

As classes definem o comportamento dos elementos interativos do jogo.3.1. Classe **Hero** (Jogador)

O herói é a entidade central, responsável pelo movimento, física e colisão.A. Atributos Principais

- `actor`: O objeto *sprite* (`Actor`) que representa o herói na tela.
- `vy`: Velocidade vertical (usada para gravidade e pulo).
- `on_ground`: Booleano que indica se o herói está tocando uma plataforma (permite pular).
- `invincible_timer`: Contador que impede colisões imediatas com inimigos (usado no início e, futuramente, após tomar dano).

B. Funções (Métodos)

- **`apply_gravity()`**
  - **Lógica:** Adiciona `GRAVITY` a `self.vy` e move `actor.y`. Limita a velocidade máxima de queda.
- **`handle_platforms()`**
  - **Lógica:** Itera sobre as plataformas e verifica a colisão vertical. Se o herói estiver caindo e tocar o **topo** de uma plataforma, zera `self.vy` e define `self.on_ground = True`.

- **`update_animation(running)`**
    - **Lógica:** Alterna entre as listas de imagens (`images_idle`, `images_run`) a cada `ANIMATION_SPEED` *frames*, dependendo se o herói está no chão e se está se movendo.
- **`jump()`**
    - **Lógica:** Só é executado se `self.on_ground` for `True`. Aplica `JUMP_VELOCITY` a `self.vy`.

3.2. Classe **Enemy** (Inimigo)

O inimigo é uma ameaça móvel que patrulha uma área horizontalmente.A. Atributos Principais

- `actor`: O objeto *sprite* que representa o inimigo.
- `direction`: Valor `1` ou `-1` para controlar a direção do movimento.
- `left_limit`, `right_limit`: Coordenadas que definem o ponto de patrulha do inimigo.

B. Funções (Métodos)

- **`update()`**
    - **Lógica de Animação:** Troca o *sprite* do inimigo para dar a sensação de ataque/movimento.
    - **Lógica de Movimento:** Move o inimigo usando `self.direction`. Quando atinge um limite (`left_limit` ou `right_limit`), inverte a `self.direction` e espelha o *sprite* (`self.actor.flip_x`).

-----4. Estrutura do Nível4.1. Plataformas (`platforms`)

As plataformas são definidas usando objetos `Rect` do Pygame, que contêm as coordenadas e dimensões.

Índice	Coordenadas (Posição, Dimensão)	Função
`platforms[0]`	(10, 490), (200, 20)	Plataforma Inferior (Ponto de início do herói).
`platforms[1]`	(300, 360), (200, 20)	Plataforma do meio 1.
`platforms[2]`	(10, 230), (200, 20)	Plataforma do meio 2.
`platforms[3]`	(300, 110), (200, 20)	Plataforma de Chegada/Vitória.

-----5. Lógica de Jogo (Funções Globais)

As funções globais orquestram o fluxo principal do jogo.5.1. `update()`

Chamada a cada *frame* para processar a lógica do jogo.

- **Entrada do Jogador:** Processa `keyboard.left` e `keyboard.right` para movimento horizontal.
- **Física:** Chama `hero.update()` para física e colisão.
- **Invencibilidade:** Decrementa `hero.invincible_timer`.
- **Vitória:** Verifica se o herói está no chão e no topo da `platforms[-1]`. Se sim, define `game_state = "WIN"`.
- **Colisão com Inimigos:**
    - Atualiza a posição de cada inimigo.
    - Usa **distância euclidiana** (`math.hypot`) para verificar se o herói e o inimigo estão a menos de 40 unidades de distância.
    - Se estiverem muito próximos e o herói não estiver invencível, define `game_state = "GAME_OVER"`.

5.2. `draw()`

Chamada a cada *frame* para renderizar os gráficos.

- **Desenho Condicional:** Usa `game_state` para decidir o que desenhar:
    - **"MENU"**: Título e botões **START**, **MUSIC**, **EXIT**.
    - **"PLAYING"**: Plataformas, `hero.draw()`, e `enemy.draw()`.
    - **"GAME_OVER"**/**"WIN"**: Mensagem de estado final e botão **RESTART**.

5.3. `on_mouse_down(pos)`

Gerencia a interação com o mouse (cliques nos botões).

- **Botões:** Utiliza o método `collidepoint(pos)` dos objetos `Rect` definidos para os botões para detectar o clique.
- **Ações de Reset:** Nos estados `GAME_OVER` e `WIN`, o clique no **RESTART** reinicia o `game_state`, a posição do herói e recria as instâncias dos inimigos.
- **Pulo em Jogo:** No estado `"PLAYING"`, qualquer clique na tela aciona `hero.jump()`.