Justificación de Diseño

El diseño del proyecto "Flappy Rat" se basa en la programación modular y la Orientación a Objetos (OOP), utilizando la librería Pygame para la gestión eficiente de gráficos, eventos y la lógica del juego.

1. Jerarquía de Herencia (Uso de pygame. sprite. Sprite)

Decisión

Se eligió la herencia de la clase base pygame. sprite. Sprite para las clases Rat (Personaje) y Tubo (Obstáculo).

Justificación

La clase pygame. sprite.Sprite es fundamental en Pygame para cualquier elemento gráfico que necesite:

Detección de Colisiones Eficiente (Hitbox): La herencia proporciona automáticamente el atributo rect (un objeto pygame. Rect) que permite realizar verificaciones de colisión de manera trivial y optimizada mediante funciones como pygame.sprite.groupcollide(). Sin esta herencia, la gestión manual de rectángulos y colisiones sería mucho más compleja y propensa a errores.

Manejo de Grupos (Batch Processing): Permite organizar los objetos en instancias de pygame.sprite.Group o pygame. sprite.RenderUpdates (como grupo_rat y grupo_tubo). Esto simplifica el bucle principal, ya que se pueden llamar métodos como grupo_tubo.update() y grupo_tubo.draw(screen) con una sola línea de código, aplicando la lógica a todos los elementos del grupo simultáneamente.

En resumen: Esta jerarquía es el patrón de diseño estándar en Pygame para crear elementos de juego activos, desacoplando la lógica de movimiento y estado (en el método update()) de la lógica principal del juego.

2. Selección de Tipos de Datos Abstractos (TDA)

Las decisiones sobre los TDA se centran en la eficiencia del acceso y la gestión dinámica de los elementos del juego.

a) Lista o Arreglo (list) para Animación

Tarea

Almacenamiento de Frames

TDA Seleccionado

list (en el atributo self.images de Rat)

Justificación

Acceso O(1) por Índice: Los frames de la animación (rat1.png, rat2.png, etc.) son estáticos y se cargan al inicio. Una lista permite un acceso directo y muy rápido (O(1)) a la imagen actual del aleteo a través de su índice (self.index), lo cual es crucial para mantener la velocidad de 60 FPS.

b) Grupos de Sprites (pygame.sprite.Group)

Tarea

Gestión de Obstáculos

TDA Seleccionado

pygame.sprite.Group (para grupo_tubo y grupo_rat)

Justificación

Optimización de Operaciones: Un Group no es solo una lista; está diseñado para optimizar el dibujo y el cálculo de colisiones de sus miembros. Permite la eliminación segura de elementos (sprite.kill()) durante la iteración y soporta funciones de Pygame altamente optimizadas (groupcollide), mejorando el rendimiento general.

Generación y Eliminación Dinámica

pygame.sprite.Group

Los tubos se crean (.add()) y se eliminan (.kill()) constantemente. El Group gestiona estos cambios de tamaño dinámicamente, asegurando que solo los tubos visibles y activos participen en los cálculos de la física.

c) Variables Booleanas y Enteras

Tarea

Control de Estado

TDA Seleccionado

Variables Booleanas (volando, juego_terminado, paso_tubo)

Justificación

Control de Flujo Lógico Sencillo: Las banderas booleanas son el TDA más simple y directo para controlar bifurcaciones en el flujo del juego (ej: if juego_terminado == False: ...). paso_tubo es una bandera de estado de transición que garantiza que la puntuación solo se incremente una vez por cada par de tubos, evitando el flickering o el conteo múltiple en un solo frame.

Control de Tiempo y Posición

TDA Seleccionado

Enteros y Flotantes (scroll_suelo, puntuacion, vel, ultimo tubo)

Justificación

Eficiencia Numérica: Los números enteros son suficientes para manejar posiciones de píxeles y la puntuación. El uso de float en la velocidad (self.vel) en la clase Rat es necesario para modelar la aceleración constante de la gravedad de forma precisa antes de que el movimiento se trunque a un entero para la posición del píxel (int(self.vel)).