Técnicas De Programación Avanzada

Marcianitos

Grupo épsilon: Arturo Márquez, Andrés Moreno, Asier Muñoz, Adrián Oliva

Descripción general

Nuestro grupo ha decidido desarrollar un videojuego en 2D, el objetivo principal es crear una experiencia interactiva sencilla pero entretenida, con un estilo visual atractivo y jugabilidad dinámica. Todo ello con las técnicas aprendidas en clase e implementas por todos los miembros del grupo.

Temática

El juego está inspirado en el clásico de los “marcianitos”, donde los jugadores deben enfrentarse a oleadas de enemigos espaciales que descienden desde la parte superior de la pantalla, con una nave situada en la parte inferior, el reto consiste en esquivar disparos enemigos y eliminar a los invasores antes de que logren invadir la zona de juego. La temática combina acción, reflejos rápidos y una ambientación retro que busca recrear la esencia de los primeros videojuegos arcade.

Supuesto prototipo

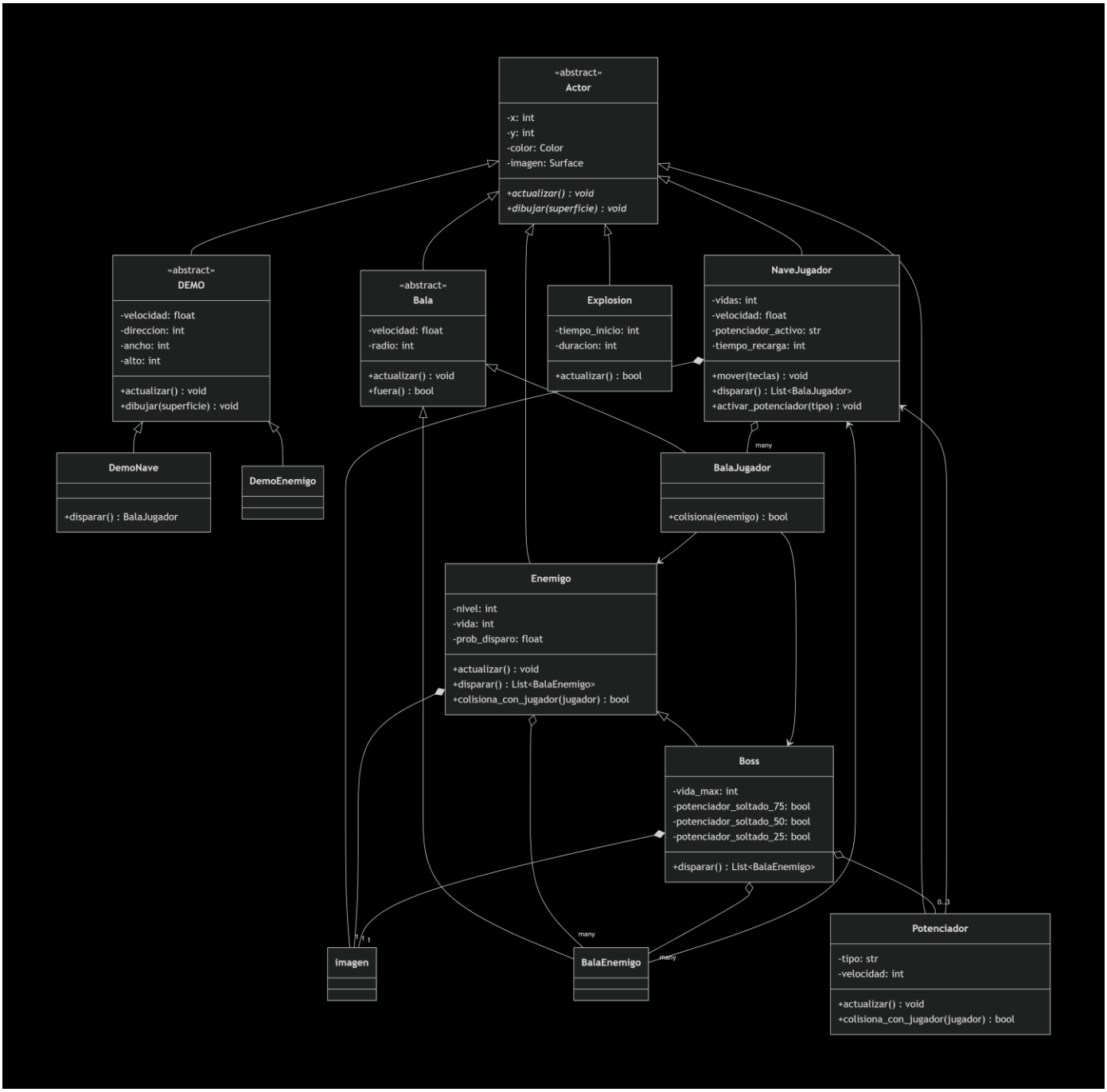
El prototipo inicial del juego consistirá en un escenario sencillo con un fondo espacial y una nave controlada por el jugador en la parte inferior de la pantalla. La nave podrá moverse de izquierda a derecha y disparar proyectiles hacia arriba. En la parte superior aparecerán filas de enemigos que descenderán poco a poco, imitando la mecánica del clásico juego de marcianitos.

Clases y Estructura Final del Código

Clase	Clase Base	Atributos Clave Implementados	Métodos Clave Implementados	Propósito en el Juego
Actor		x, y (posición), imagen, ancho, alto.	actualizar(), dibujar().	Clase base que define la estructura mínima de cualquier objeto en pantalla.
NaveJugador	Actor	vidas, velocidad, tiempo_recarga, potenciador_activo, potenciador_tiempo_fin.	mover(), disparar(), activar_potenciador(), actualizar_potenciador().	Representa la nave controlada por el usuario. Gestiona la lógica de disparo triple y las vidas restantes.
Bala	Actor	velocidad, radio, color.	actualizar(), fuera().	Proyectil base. Define el movimiento rectilíneo.
BalaJugador	Bala	Velocidad: Negativa (sube), Color: Blanco.	colisiona(enemigo).	Proyectil del jugador. Implementa la lógica de colisión específica con los enemigos.
BalaEnemigo	Bala	Velocidad: Positiva (baja), Color: Amarillo.	N/A.	Proyectil del enemigo.
Enemigo	Actor	vida, velocidad, direccion, prob_disparo.	actualizar(), disparar(), colisiona_con_jugador().	Invasor estándar. Su vida y velocidad escalan según el nivel. Desciende 20px al rebotar.
Boss	Actor	vida_max, vida, potenciador_sol	disparar() (doble bala).	Enemigo final de cada 5 niveles. Suelta

		tado_XX (banderas).		potenciadores en umbrales de vida fijos.
Potenciador	Actor	tipo (ej. 'disparo_triple'), velocidad.	colisiona_con_jugador().	Objeto que cae y otorga una ventaja temporal al ser recogido.
Explosion	Actor	duracion.	actualizar() (retorna True al finalizar).	Efecto visual temporal para impactos y destrucción.

Diagrama UML



Riesgos Previsibles

Categoría de Riesgo	Problema Encontrado (Experiencia Real)	Solución Implementada en el Código
Inicialización y Recursos [7]	**Errores de Inicialización:** Fallo al cargar las imágenes debido a rutas no relativas. Se resolvió agrupando recursos en una carpeta específica, como se sugirió en foros.	1. **Centralización:** Uso de **config.py** y cargar_imagenes() para gestionar rutas y escalados. 2. **Rutas Robustas:** Uso de os.path.join para compatibilidad de rutas entre sistemas operativos.
Lógica y Rendimiento [8]	**Ralentización y Bloqueo:** El juego se volvía lento porque los enemigos y balas de las oleadas anteriores no se eliminaban al avanzar de nivel, duplicando los actores activos.	1. **Control de FPS:** Se usa clock.tick(60) para regular la velocidad[9]. 2. **Limpieza Rigurosa de Listas:** Al cambiar de nivel, las listas de actores (enemigos, balas_jugador, balas_enemigas) se limpian completamente (usando .clear() o comprensión de listas) para eliminar la sobrecarga de recursos.
Integridad del Código	**Errores de Compilación/Sintaxis:** Fallos recurrentes con paréntesis, comillas y dos puntos durante la codificación.	El uso de un IDE (Visual Studio Code) fue la mitigación más efectiva, proporcionando corrección de sintaxis en tiempo real, facilitando la ejecución del código.
Jugabilidad y Control [10]	*No se encontraron problemas mayores.*	Uso de pygame.key.get_pressed() para el movimiento de la nave, lo que asegura una **respuesta suave y continua** al mantener presionadas las teclas direccionales.
Estabilidad del Programa [11]	*No se encontraron problemas mayores.*	Manejo del evento pygame.QUIT en el bucle principal y en el menú para asegurar un cierre ordenado del programa.

Repositorio de GitHub:

https://github.com/xndresmoreno/Marcianitos