Guía de Proyecto: Juego con Pygame para Programación I

1. Objetivo Pedagógico Principal

El objetivo ~~no~~ es crear el próximo éxito comercial de videojuegos, sino que los ustedes demuestren la comprensión y aplicación de los conceptos fundamentales de programación en un proyecto integrador. Los pilares son:

- **Estructuras de Control:** Uso de bucles (**while**), condicionales (**if/elif/else**).
- **Funciones:** Modularización del código para evitar repetición y mejorar la legibilidad.
- **Estructuras de Datos:** Manejo de listas (para grupos de enemigos, balas, etc.) y diccionarios (para propiedades de objetos, configuraciones).
- **Programación Orientada a Objetos (POO):** Concepto de clases y objetos para representar entidades del juego (Jugador, Enemigo, Bala). Este es el salto más importante y el que más les va a servir.
- **Resolución de Problemas:** Diseñar una solución lógica a un problema complejo y depurar errores.
- **Manejo de Librerías Externas:** Aprender a leer documentación y usar APIs como las de Pygame(base), Ursina u otros.

---

2. Niveles de Entrega (Mínimos, Intermedio, Avanzado)

Puedes usar estos niveles para establecer la calificación base, la nota esperada y la nota máxima.

Nivel 1: Mínimo Viable (La Entrega para Existir)

Un juego en su forma más simple. Debe tener un principio, un desarrollo y un final claros.

Requisitos Indispensables:

1. **Ventana de Juego:** El programa debe abrir una ventana de Pygame con un título y un tamaño definido, menú, y opciones de configuración.
2. **Bucle de Juego (Game Loop):** Un bucle **while** que se ejecute constantemente para mantener la ventana abierta y procesar eventos.
3. **Control del Jugador mínimo:** Debe existir un "jugador" (no debe ser simple cuadrado) que se pueda mover por la pantalla usando el teclado (ej. flechas o WASD mínimo).
4. **Mas de un Objeto Interactivo:** Debe haber otro elemento en pantalla (un enemigo, un punto coleccionable, un obstáculo).
5. **Condición de Victoria o Derrota:** El juego debe tener un final claro.
    - **Ejemplo de victoria:** "Recoge 10 monedas para ganar".
    - **Ejemplo de derrota:** "Si tocas al enemigo, pierdes".

6. **Retroalimentación en Pantalla:** Mostrar un mensaje simple de "¡Ganaste!" o "Game Over" cuando se cumpla la condición.
7. **Código Estructurado:** El código no puede ser un solo "script" gigante. Debe tener al menos funciones separadas para manejar eventos, actualizar la lógica y dibujar en pantalla.

Ejemplo de Juego: Un cuadrado (jugador) se mueve para evitar otro cuadrado (enemigo) que se mueve en línea recta. Si el jugador dura 30 segundos, gana.

Nivel 2: Intermedio (La Entrega Esperada para una Nota)

Sobre la base del mínimo, se añaden mecánicas y pulido que lo sienten como un "juego de verdad".

Requisitos Adicionales:

1. **Estados del Juego (Game States):** El juego debe tener al menos 3 estados:
    - **Menú Principal:** Una pantalla de bienvenida con instrucciones y un botón para "Jugar".
    - **Jugando:** El bucle de juego principal.
    - **Game Over/Victoria:** Una pantalla que muestre el resultado final y la opción de "Reintentar" o "Volver al Menú".
2. **Uso de Clases (POO):** El jugador, los enemigos y cualquier otro elemento activo deben ser instancias de una clase que herede de **pygame.sprite.Sprite**. Esto es clave para un código limpio.
3. **Gráficos y Sonido:**
    - Usar imágenes (sprites) en lugar de formas geométricas.
    - Incluir al menos un efecto de sonido (ej. al recoger un objeto, al disparar) y música de fondo.
4. **Mecánicas más Complejas:**
    - **Sistema de Puntuación:** Un marcador que aumenta durante el juego.
    - **Sistema de Vidas/Salud:** El jugador tiene más de una oportunidad.
    - **Múltiples Objetos:** Varios enemigos a la vez, o balas que el jugador puede disparar.
5. **Colisiones Precisas:** Uso de **pygame.sprite.groupcollide()** o **pygame.sprite.collide_rect()** para detectar colisiones entre sprites.

Ejemplo de Juego: Un *Space Invaders* simple. El jugador dispara a una fila de enemigos que se mueven. Tiene un marcador y 3 vidas.

Nivel 3: Avanzado (Para los "Duchos" y Nota Media)

Aquí es donde los alumnos pueden desafiarse a sí mismos y donde entra en juego la opción 3D.

Opciones a Elegir (una o más):

1. **Inteligencia Artificial (IA) Básica:**
    - Enemigos que no solo se mueven en línea recta, sino que persiguen al jugador (pathfinding simple).
    - Diferentes tipos de enemigos con comportamientos distintos.
2. **Generación Procedural:**
    - Un mapa o un nivel que se genere de forma aleatoria cada vez que se juega (ej. un laberinto simple, una mazmorra).
3. **Sistemas de Partículas:**
    - Efectos visuales para explosiones, humo, magia, etc.
4. **Menús Interactivos:**
    - Un menú de opciones para cambiar volumen, dificultad, etc.
    - Botones con efectos de hover (cambian de color al pasar el ratón).
5. **Persistencia de Datos:**
    - Guardar y cargar la puntuación más alta en un archivo de texto (**txt**) o JSON.
    - Guardar el progreso del juego.

Opción Especial: Aceleración 3D con OpenGL

¡ADVERTENCIA IMPORTANTE! Esto es un salto de complejidad **GRANDE**. No es recomendable para la mayoría de los alumnos.

"Desafío de honor" para alumnos muy motivados y con sólidas bases de matemáticas (vectores, matrices).

Guía Mínima para la opción 3D:

- **Tecnología:** Usarán **Pygame** para crear la ventana y el contexto de renderizado, y la librería **PyOpenGL** para dibujar en 3D.
- **Requisitos Mínimos 3D:**
    1. **Crear un Contexto OpenGL:** Configurar Pygame para usar **DOUBLEBUF|OPENGL**.
    2. **Dibujar una Forma 3D Simple:** Un cubo o una pirámide que rote en el centro de la pantalla. Esto implica definir vértices, caras y usar matrices de proyección y modelo/vista.

3. **Control de Cámara:** Permitir al usuario mover la "cámara" con el ratón o el teclado para ver el objeto desde diferentes ángulos.
4. **Iluminación Básica:** Aplicar una fuente de luz simple para que el objeto tenga volumen y no sea solo una silueta plana.

Ejemplo de Juego 3D Realista: Un simulador de laberinto 3D en primera persona, donde las paredes son cubos y el objetivo es encontrar la salida. La física y las colisiones son mucho más complejas que en 2D.

---

3. Guía de Organización y Buenas Prácticas

Indica a tus alumnos cómo deben estructurar su proyecto. Esto es tan importante como el código en sí.

Estructura de Archivos Recomendada:

mi_juego/

├── main.py # Archivo principal. Inicia Pygame y el bucle de juego.

├── player.py # Clase del Jugador.

├── enemy.py # Clase del Enemigo.

├── settings.py # Constantes del juego (tamaño de pantalla, colores, etc.).

├── assets/ # Carpeta para recursos

│ ├── images/

│ │ ├── player.png

│ │ └── enemy.png

│ └── sounds/

│ ├── laser.wav

│ └── music.mp3

└── README.md # Archivo explicando cómo ejecutar el juego y sus controles.

Control de Versiones:

- Usar **Git** y a subir su progreso a un repositorio en **GitHub** o **GitLab**.

---

4. Rúbrica de Evaluación

Criterio	Insuficiente (1-4)	Suficiente (5-6)	Notable (7-8)	Sobresaliente (9-10)
Funcionalidad	El juego no se ejecuta o carece de los mínimos.	Cumple todos los requisitos del **Nivel 2**.	Cumple todos los requisitos del **Nivel 3**.	Cumple los requisitos del **Nivel 2, Nivel 3 y extras**.
Estructura del Código	Código monolítico, difícil de leer.	Usa funciones básicas. El código es algo legible.	Usa clases y funciones de forma efectiva. Código modular y claro.	Código excelentemente estructurado, con comentarios, docstrings y una arquitectura limpia.
Calidad del Juego	Sin gráficos ni sonido, o estos no funcionan.	Usa formas geométricas. La jugabilidad es básica pero funcional.	Usa sprites y sonido. La jugabilidad es fluida y entretenida.	Pulido excepcional. Menús completos, efectos visuales, IA, etc. Experiencia de usuario cuidada.
Creatividad y Complejidad	Copia un tutorial sin cambios.	Idea simple pero implementada por sí mismos.	Idea original con mecánicas bien definidas.	Idea muy original o compleja (ej. 3D, procedural generación). Demuestra iniciativa y aprendizaje autónomo.