Mini documentación — Mejoras UI/UX y diseño para el Proyecto Nexus en Tkinter

Contexto general

Actualmente, el proyecto Nexus tiene una ventana sencilla en Tkinter con elementos básicos: título, imagen, botones. El diseño es funcional pero con margen para mejorar la experiencia visual y la interacción del usuario.

Cambios y mejoras planeadas

1. Fondo oscuro como base

- Cambiar el fondo de la ventana principal a un tono oscuro (#1e1e1e) para dar una atmósfera más inmersiva y acorde con un RPG.
- Los textos (títulos, etiquetas, botones) cambiarán a colores claros (blanco o verde claro) para generar contraste y facilitar la lectura.
- Esto mejorará la estética y reducirá la fatiga visual, especialmente en sesiones prolongadas.

2. Integración de imágenes

- Se incluirá una imagen representativa (logo, personaje, etc.) en formato PNG, redimensionada con la librería Pillow para adaptarla al tamaño adecuado.
- La imagen se puede colocar usando distintas técnicas según el efecto deseado:
    - pack(pady=...) para posicionarla de forma sencilla y centrada con espacio vertical.
    - place(x=..., y=...) para control absoluto de la posición en píxeles, útil para diseños más precisos (ej. colocar la imagen en una esquina).
    - grid(row=..., column=...) para layouts basados en filas y columnas, flexible para interfaces más complejas.
- El uso de Pillow (PIL.Image y ImageTk.PhotoImage) permite cargar imágenes en formatos avanzados y redimensionarlas dinámicamente sin perder calidad, algo que tk.PhotoImage no permite con todos los formatos.

3. Tipografía y estilos

- Se utilizarán fuentes legibles y con estilo adecuado para RPG, por ejemplo “Georgia” para títulos y “Arial” para textos descriptivos.
- El tamaño y estilo (negrita, cursiva) serán consistentes para jerarquizar la información (título principal, subtítulos, instrucciones).
- Se aplicarán márgenes y paddings adecuados para evitar que los elementos se vean amontonados.

4. Botones con diseño personalizado

- Los botones tendrán colores personalizados para destacar acciones importantes, por ejemplo:
    - Fondo verde (#4CAF50) con texto blanco para el botón "Comenzar/Iniciar".
    - Efectos visuales simples (cambios de color al pasar el mouse, deshabilitar botones para evitar clics múltiples).
- Esto mejora la usabilidad y guía al usuario en la interacción.

5. Mensajes de estado dinámicos

- Se implementará un label para mostrar mensajes de estado (ej. “El juego está iniciando...”).
- Al iniciar, el botón se deshabilita para evitar interacciones erróneas.
- Se puede expandir para mostrar carga de recursos o mensajes durante la navegación entre pantallas.

6. Estructura modular y escalable

- Aunque por ahora es un prototipo, se plantea organizar el código para futuras ventanas y pantallas del juego (menús, selección de personaje, configuración).
- Separar lógica visual y funcional para facilitar mantenimiento y ampliaciones.

Aspectos técnicos del código (uso de imágenes y layout en Tkinter)

- **Pillow para imágenes:**
  La librería Pillow permite abrir imágenes en formatos comunes (PNG, JPG, etc.) y redimensionarlas con .resize((ancho, alto)). Después se convierte a formato compatible con Tkinter mediante ImageTk.PhotoImage() para poder mostrarla en widgets como Label.
- **Mostrar imágenes en Tkinter:**
    - pack(): Coloca el widget de forma automática en el orden de llamada, con opciones para agregar espacios (paddings) o anclar a los lados. Ideal para layouts simples.
    - place(x=..., y=...): Posiciona el widget en coordenadas absolutas dentro de la ventana, útil para diseños precisos, pero menos flexible ante redimensionamientos.
    - grid(row=..., column=...): Ubica widgets en una grilla, facilitando diseños complejos con filas y columnas, permitiendo mayor control de alineación y expansión.
- **Recomendaciones:**

- Para interfaces simples o prototipos, usar pack() es más rápido y sencillo.
- Para posicionar la imagen en una esquina o lugar exacto, usar place().
- Si el proyecto crece con varias pantallas o elementos, migrar a grid() para mantener orden y escalabilidad.
- Mantener siempre la referencia de la imagen en una variable global o de instancia para evitar que Python la elimine y no se muestre.