Manual del Programador: Las Flipantes e Increíbles aventuras

Este documento describe la estructura del código, las clases y funciones principales del juego "Las Flipantes e Increíbles aventuras", y ofrece una guía sobre cómo realizar modificaciones o ampliaciones.

1. Introducción

El juego está desarrollado utilizando tecnologías web estándar:

- **HTML (Main.html):** Define la estructura básica de la página y los contenedores para las diferentes pantallas del juego.
- **CSS (Main.css):** Proporciona los estilos visuales básicos. Utiliza Tailwind CSS para facilitar el estilizado mediante clases de utilidad.
- **JavaScript (Módulos ES6 - .mjs):** Implementa toda la lógica del juego, la interfaz de usuario y las interacciones. El código está organizado en módulos para una mejor separación de responsabilidades.

2. Estructura del Proyecto

El código fuente se organiza de la siguiente manera:

- **Main.html**: El archivo HTML principal que carga el CSS y el script de entrada de JavaScript (Main.mjs). Contiene elementos section para cada pantalla del juego (inicio, creador, lobby, tienda, inventario, batalla).
- **Main.css**: Archivo CSS principal para estilos personalizados adicionales a Tailwind.
- **README.md**: Información básica del proyecto (como el enlace a la versión desplegada).
- **JAVASCRIPT/**: Carpeta que contiene todo el código JavaScript.
    - **Main.mjs**: Punto de entrada de la aplicación. Inicializa el juego, carga los audios, asigna los *event listeners* principales a los botones de navegación entre pantallas y llama a las funciones de configuración iniciales.
    - **Constant.mjs**: Define constantes globales del juego (ej: MAX_PUNTOS_ESTADISTICAS, XP_POR_NIVEL_BASE, CAPACIDAD_INVENTARIO).
    - **DatoBasico.mjs**: Contiene los datos iniciales del juego, como la lista de armas (listaGlobalArmas) y enemigos (listaGlobalEnemigos) disponibles.
    - **GestorAudio.mjs**: Gestiona la carga y reproducción de la música de fondo y efectos de sonido (aunque los efectos no parecen estar implementados actualmente). Requiere interacción del usuario para iniciar la reproducción.
    - **Mando.mjs**: Implementa la lógica para detectar y manejar la entrada de un

gamepad/mando, permitiendo la navegación por los menús.

- **Status.mjs**: Mantiene el estado global del juego, principalmente la instancia del Personaje actual y la instancia del Combate activo. Proporciona funciones get y set para acceder a estos estados.
- **UtilesExtras.mjs**: Contiene funciones de utilidad reutilizables:
  - mostrarPantalla(): Cambia entre las diferentes secciones (<section>) del HTML.
  - guardarProgreso(), cargarProgreso(), eliminarProgreso(): Gestionan la persistencia de datos usando localStorage.
  - mostrarMensaje(): Muestra mensajes temporales al usuario en las diferentes pantallas.
- **ui/**: Carpeta con módulos específicos para cada pantalla de la interfaz de usuario.
  - Inicio.mjs, Creador.mjs, Lobby.mjs, Tienda.mjs, Inventario.mjs: Cada módulo exporta funciones para configurar su pantalla respectiva (configurarPantalla*) y manejar las acciones del usuario dentro de esa pantalla (accion*). Interactúan con el DOM para mostrar datos y responder a eventos.
- **clases/**: Carpeta que contiene las clases principales que definen la lógica del juego.
  - Luchador.mjs: Clase base abstracta para cualquier entidad que participa en combate. Define propiedades comunes (vida, ataque, defensa, etc.) y métodos básicos (recibirDanio, estaVivo).
  - Personaje.mjs: Representa al jugador. Hereda de Luchador. Añade lógica de experiencia, niveles, dinero, gestión de inventario (delegando a la clase Inventario), cálculo de ataque con arma, y métodos de serialización/deserialización (toJSON, fromJSON) para guardar/cargar.
  - Enemigo.mjs: Representa a un enemigo. Hereda de Luchador. Añade recompensas (XP, dinero), un ID, una URL para su avatar y un patrón de IA (decidirAccion).
  - Arma.mjs: Define la estructura de un objeto Arma (id, nombre, ataque, precio, descripción).
  - Inventario.mjs: Gestiona la colección de objetos Arma del jugador. Controla la capacidad, el equipamiento y proporciona métodos para añadir, eliminar y buscar armas.
  - Combate.mjs: Orquesta la lógica de una batalla entre un Personaje y un Enemigo. Gestiona los turnos, calcula el daño, procesa las acciones (atacar, defender, huir), actualiza la UI de batalla (actualizarUIBatalla), y maneja el final del combate (victoria, derrota, huida) llamando a un

callback.

3. Clases y Funciones Principales

- **Main.mjs** -> **inicializarJuego()**: Función principal que se ejecuta al cargar la página. Configura los listeners globales y muestra la pantalla inicial.
- **UtilesExtras.mjs**:
  - mostrarPantalla(idPantalla): Oculta todas las pantallas y muestra la indicada por idPantalla. Llama a la función configurarPantalla* correspondiente si existe.
  - guardarProgreso(): Serializa el objeto Personaje (usando Personaje.toJSON()) y lo guarda en localStorage.
  - cargarProgreso(): Carga los datos de localStorage, los deserializa (usando Personaje.fromJSON()) y actualiza el estado global (setPersonaje).
  - eliminarProgreso(): Borra los datos guardados de localStorage.
  - mostrarMensaje(texto, tipo, duracion): Muestra un mensaje feedback en la pantalla activa.
- **Status.mjs**:
  - getPersonaje(), setPersonaje(p): Acceden/modifican la instancia global del personaje.
  - getCombate(), setCombate(c): Acceden/modifican la instancia global del combate activo.
- **clases/Personaje.mjs**:
  - constructor(...): Inicializa un nuevo personaje.
  - ganarExperiencia(cantidad): Añade XP y comprueba si se sube de nivel.
  - subirNivel(): Incrementa el nivel, mejora estadísticas aleatoriamente, recalcula XP necesaria.
  - agregarArmaInventario(arma), equiparArmaInventario(idArma): Delegan en el objeto Inventario.
  - toJSON(): Convierte el estado del personaje a un objeto JSON simple para guardar.
  - fromJSON(jsonData, listaGlobalArmas): Reconstruye una instancia de Personaje a partir de datos JSON y la lista global de armas.
- **clases/Inventario.mjs**:
  - agregarArma(arma): Añade un arma si hay espacio y no está duplicada.
  - equiparArma(idArma): Marca un arma como equipada por su ID.
  - obtenerArmaEquipada(): Devuelve la instancia del arma equipada.
- **clases/Combate.mjs**:
  - constructor(personaje, enemigo, UIElements, onCombateTerminado): Prepara el combate.

    - iniciar(): Determina el primer turno y comienza el flujo del combate.
    - ejecutarAccionJugador(accion): Procesa la acción elegida por el jugador.
    - ejecutarTurnoEnemigo(): Obtiene la acción de la IA y la ejecuta.
    - realizarAtaque(atacante, defensor): Calcula y aplica el daño de un ataque.
    - terminarCombate(jugadorGano, huyo): Gestiona el final del combate, muestra resultados y otorga recompensas/penalizaciones.
    - actualizarUIBatalla(): Refresca los elementos visuales del combate (barras de vida, nombres, avatares).
- **ui/*.mjs**:
    - configurarPantalla*(): Se llaman al mostrar una pantalla. Actualizan el contenido dinámico (ej: lista de armas en la tienda, estadísticas en el lobby).
    - accion*(): Funciones que se ejecutan como respuesta a eventos del usuario (clics en botones, envíos de formulario). Contienen la lógica específica de esa interacción.

4. Flujo del Juego

1. **Inicialización (Main.mjs)**: Se carga Main.html, se ejecuta inicializarJuego() en Main.mjs.
2. **Pantalla de Inicio (ui/Inicio.mjs)**: Se muestra pantalla-inicio. Se comprueba si hay datos guardados para habilitar "Continuar".
3. **Nueva Partida -> Creador (ui/Creador.mjs)**: El usuario introduce datos, valida estadísticas, elige arma. Al crear, se instancia Personaje, se guarda (guardarProgreso) y se va al Lobby.
4. **Continuar Partida -> Lobby (ui/Inicio.mjs, UtilesExtras.mjs)**: Se llama a cargarProgreso(), se deserializa el Personaje y se va al Lobby.
5. **Lobby (ui/Lobby.mjs)**: Muestra estado del personaje. Permite navegar a Tienda, Inventario o iniciar Combate.
6. **Tienda (ui/Tienda.mjs)**: Muestra armas, permite comprar si se cumplen condiciones. Actualiza Personaje y guarda.
7. **Inventario (ui/Inventario.mjs)**: Muestra armas poseídas, permite equipar. Actualiza Personaje y guarda.
8. **Combate (ui/Lobby.mjs, clases/Combate.mjs)**: Se selecciona un Enemigo aleatorio, se instancia Combate, se muestra pantalla-batalla. El objeto Combate gestiona el flujo hasta que termina. Al finalizar, se llama al *callback* onCombateTerminado que guarda el progreso y vuelve al Lobby.
9. **Guardar y Salir (ui/Lobby.mjs)**: Llama a guardarProgreso() y deshabilita botones.

5. Modificaciones y Ampliaciones

- **Añadir Contenido:**
  - **Armas/Enemigos:** Edita las listas listaGlobalArmas y listaGlobalEnemigos en JAVASCRIPT/DatoBasico.mjs. Asegúrate de que los IDs sean únicos. Puedes añadir nuevas propiedades si es necesario (ej: tipo de daño, resistencias) y modificar las clases correspondientes para usarlas.
  - **Avatares:** Reemplaza las URLs de placeholder en Main.html y las URLs por defecto en Personaje.mjs y Enemigo.mjs (o en DatoBasico.mjs para enemigos específicos) con rutas a tus propios archivos de imagen.
- **Nuevas Mecánicas:**
  - **Habilidades/Magia:** Modifica Luchador.mjs y Personaje.mjs para incluir maná o puntos de habilidad. Añade métodos para aprender y usar habilidades en Personaje.mjs. Modifica Combate.mjs para manejar nuevas acciones y sus efectos. Actualiza la UI de combate para mostrar y seleccionar habilidades.
  - **Efectos de Estado (Veneno, Quemadura):** Utiliza el array efectosEstado en Luchador.mjs. Define objetos para cada efecto (nombre, duración, efecto por turno). Modifica Combate.mjs para aplicar efectos al atacar y procesar sus efectos al inicio/final de cada turno. Actualiza la UI para mostrar los efectos activos.
  - **Objetos Consumibles (Pociones):** Crea una nueva clase Consumible.mjs. Modifica Inventario.mjs para manejar diferentes tipos de objetos (o crea un InventarioConsumibles). Añade una opción "Usar Objeto" en combate y/o en el Lobby/Inventario.
  - **Armaduras/Accesorios:** Crea clases Armadura.mjs, Accesorio.mjs. Modifica Personaje.mjs para tener ranuras de equipamiento adicionales. Modifica Inventario.mjs (o crea inventarios separados). Actualiza el cálculo de estadísticas (calcularDefensa, etc.) en Personaje.mjs. Añade UI para gestionar este equipo.
- **Mejoras:**
  - **IA Enemiga:** Expande el método decidirAccion en Enemigo.mjs con patrones más complejos (ej: usar habilidades, reaccionar a la vida del jugador/propia).
  - **UI/UX:** Mejora los estilos en Main.css. Añade animaciones o transiciones. Reestructura el HTML en Main.html para una mejor presentación. Considera usar una librería de componentes si la complejidad aumenta.
  - **Sonido:** Implementa efectos de sonido para acciones (ataque, recibir daño, comprar, etc.) usando GestorAudio.mjs.
  - **Balance:** Ajusta las constantes en Constant.mjs y las estadísticas/recompensas en DatoBasico.mjs para equilibrar la dificultad.