

PBRLRPG

Presentación del Proyecto

Introducción al Proyecto

PBRLRPG

- **Género:** RPG por turnos simple.
- **Plataforma:** Navegador web.
- **Características:**
 - Creación de personaje con estadísticas básicas.
 - Sistema de combate por turnos contra enemigos aleatorios.
 - Progresión mediante niveles y experiencia.
 - Sistema de inventario y equipamiento de armas.
 - Tienda para comprar armas.

- **Características:**

- Persistencia de datos usando `localStorage` .
- **Soporte para control con mando (Gamepad API).**
- Música de fondo dinámica.

Estructura del Código

El proyecto está organizado modularmente en carpetas:

- `/JAVASCRIPT/clases` : Define las entidades principales del juego.
 - `Luchador.mjs` (Base)
 - `Personaje.mjs` (Jugador)
 - `Enemigo.mjs`
 - `Arma.mjs`
 - `Inventario.mjs`
 - `Combate.mjs` (Lógica de batalla)

- `/JAVASCRIPT/ui` : Maneja la interfaz de usuario para cada pantalla.
 - `Inicio.mjs` , `Creador.mjs` , `Lobby.mjs` , `Tienda.mjs` , `Inventario.mjs` .
- `/JAVASCRIPT` (Raíz): Archivos centrales y utilidades.
 - `Main.mjs` : Punto de entrada, inicialización, listeners globales.
 - `Status.mjs` : Gestión del estado global (personaje, combate, pantalla actual, **estado del mando**).
 - `Mando.mjs` : **(Foco Principal)** Lógica para el control por mando.
 - `UtilesExtras.mjs` : Funciones de ayuda (mostrar pantalla, guardar/cargar, mensajes).
 - `GestorAudio.mjs` : Control de la música.
 - `DatoBasico.mjs` : Datos iniciales (armas, enemigos).
 - `Constant.mjs` : Constantes del juego.

- `/Audio` : Archivos de música.
- **Raíz:** `Main.html` , `Main.css` , `README.md` .

Flujo Básico del Juego

1. **Pantalla Inicio (Inicio.mjs)**: Opción de "Continuar" (si hay datos) o "Nueva Partida".
2. **Pantalla Creador (Creador.mjs)**: Si es "Nueva Partida", se crea el personaje asignando nombre y estadísticas.
3. **Pantalla Lobby (Lobby.mjs)**: Centro principal. Muestra estado del personaje y opciones:
 - Ir a Tienda.
 - Gestionar Inventario.
 - Entrar en Combate.
 - Guardar y Salir.
4. **Pantalla Tienda (Tienda.mjs)**: Comprar armas con el dinero obtenido.
5. **Pantalla Inventario (Inventario.mjs)**: Ver armas y equipar la deseada.

6. Pantalla Batalla (`Combate.mjs` , `accionIniciarCombate` en `Lobby.mjs`):

- Se elige un enemigo aleatorio.
- Combate por turnos (Atacar, Defender, Huir).
- Resolución: Victoria (XP, dinero), Derrota (pérdida dinero), Huida (penalización).
- Regreso al Lobby.

Foco Principal: Control por Mando (Mando.mjs)

Este módulo integra el soporte para gamepads usando la **Gamepad API** del navegador.

Objetivo: Permitir la navegación por los menús y la interacción en combate sin necesidad de ratón o teclado.

Mando.mjs : Funcionamiento General

1. Inicialización (`iniciarGamepad` en `Mando.mjs` , llamado desde `Main.mjs`):

- Se añaden listeners para `gamepadconnected` y `gamepaddisconnected` .
- Se comprueba si ya hay mandos conectados al inicio.

2. Detección de Conexión/Desconexión:

- `handleGamepadConnect` : Guarda la referencia al mando en `Status.mjs` , actualiza la UI, inicia el bucle de polling.
- `handleGamepadDisconnect` : Limpia la referencia si el mando desconectado era el activo.

```
// Mando.mjs
function handleGamepadConnect(evento) {
  setMandoConectado(evento.gamepad); // Guarda en Status.mjs
  actualizarEstadoMandoVisual(); //
  if (!intervaloPollingMando) {
    intervaloPollingMando = requestAnimationFrame(pollGamepad); // Inicia lectura
  }
  actualizarSeleccionMandoPantalla(); // Prepara navegación
}
```

Mando.mjs : Bucle de Lectura (Polling)

- `pollGamepad` : Función ejecutada repetidamente con `requestAnimationFrame` .
- Obtiene el estado actual del mando conectado (`navigator.getGamepads()`).
- **Lectura de Ejes/D-Pad (Navegación):**
 - Lee el eje Y (`axes[1]`) y los botones del D-Pad (`buttons[12]` arriba, `buttons[13]` abajo).
 - Si el movimiento supera un umbral (`UMBRAL_EJE`) o se presiona el D-Pad, y ha pasado un tiempo (`RETRASO_ENTRE_MOVIMIENTOS`), llama a `navegarMando` .
- **Lectura de Botones (Acción):**
 - Compara el estado actual de los botones con el anterior (`botonesAnteriores`).
 - Si un botón pasa de no presionado a presionado, llama a `manejarPulsacionBotonMando` .

JavaScript

```
// Mando.mjs - Dentro de pollGamepad
const ejeY = estadoMandoActual.axes[1] ?? 0; //
const dpadArriba = estadoMandoActual.buttons[12]?.pressed; //
const dpadAbajo = estadoMandoActual.buttons[13]?.pressed; //

if (ahora tiempoUltimoMovimientoEje + RETRASO_ENTRE_MOVIMIENTOS) { //
  if (ejeY < -UMBRAL_EJE || dpadArriba) navegarMando(-1); // Arriba
  else if (ejeY > UMBRAL_EJE || dpadAbajo) navegarMando(1); // Abajo
  tiempoUltimoMovimientoEje = ahora; //
}

estadoMandoActual.buttons.forEach((boton, indice) = { //
  if (boton.pressed && !botonesAnteriores[indice]) { // Flanco de subida
    manejarPulsacionBotonMando(indice); //
  }
});
botonesAnteriores = estadoMandoActual.buttons.map(b => b.pressed); //
```

Mando.mjs : Navegación y Selección

- **actualizarSeleccionMandoPantalla** :
 - Llamada al cambiar de pantalla (`mostrarPantalla` en `UtilesExtras.mjs`) o al conectar el mando.
 - Busca todos los elementos interactivables **visibles** en la pantalla activa (`button:not([disabled]), input, select, a.boton`).
 - Los ordena usando el atributo `data-indice-mando` definido en `Main.html` .
 - Guarda la lista ordenada en `Status.mjs` (`elementosNavegables`).
 - Selecciona el primer elemento (`indiceElementoSeleccionado = 0`).

- **navegarMando(direccion) :**
 - Cambia el `indiceElementoSeleccionado` (-1 o +1).
 - Maneja los límites (si llega al final, va al principio y viceversa).
 - Llama a `seleccionarElementoMando` .
- **seleccionarElementoMando(elemento) :**
 - Añade la clase CSS `.seleccionado-mando` (definida en `Main.css`) para el feedback visual.
 - Llama a `elemento.focus()` para la navegación estándar.
- **quitarSeleccionMando :** Limpia la clase `.seleccionado-mando` del elemento anterior.

HTML

```
<button id="boton-ir-tienda" data-indice-mando="0">Ir a la Tienda</button>  
<button id="boton-gestionar-inventario" data-indice-mando="1">Gestionar Inventario</button>  
<button id="boton-entrar-combate" data-indice-mando="2">¡Entrar en Combate!</button>
```

CSS

```
/* Main.css */  
.seleccionado-mando {  
    outline: 3px solid #63b3ed; /* Resalte visual */ /* */  
    outline-offset: 2px; /* */  
}
```


Mando.mjs : Ejecución de Acciones

- `manejarPulsacionBotonMando(indiceBoton)` :
 - Identifica qué botón se ha presionado (según el estándar, 0 suele ser A/X, 1 suele ser B/O).
 - **Botón 0 (Confirmar):**
 - Obtiene el elemento actualmente seleccionado (`elementosNavegables[indiceElementoSeleccionado]`).
 - Simula un click en ese elemento: `elementoActivo.click()` . Esto dispara los listeners asociados en `Main.mjs` .

- **Botón 1 (Cancelar/Volver):**

- Busca un botón específico de "Volver" en la pantalla actual (ej: `#boton-volver-lobby-desde-tienda`).
- Si lo encuentra, simula un `click` en él.
- Tiene lógica para evitar acciones no deseadas (ej: en medio de una batalla).

JavaScript

```
// Mando.mjs
function manejarPulsacionBotonMando(indiceBoton) {
  const elementoActivo = getElementosNavegables()[getIndiceElementoSeleccionado()];

  if (indiceBoton === 0 && elementoActivo) { // Botón A/X
    <span class="hljs-built_in">console.log(`Mando: Activando <span class="hljs-subst">${elementoActivo.id || elementoActivo.tagName}`); //
    elementoActivo.click(); // &lt;-- Simula el click
  }
  else if (indiceBoton === 1) { // Botón B/O
    const botonVolver = pantallaActiva.querySelector(/* Selectores de botones volver */);
    if (botonVolver) {
      botonVolver.click();
    }
    // ... más lógica ...
  }
}
```

Integración con Otros Módulos

- `Status.mjs` : Clave para compartir el estado del mando (referencia al gamepad, elementos navegables, índice seleccionado) entre `Mando.mjs` y el resto de la aplicación si fuera necesario.
- `UtilesExtras.mjs` (`mostrarPantalla`): Llama a `actualizarSeleccionMandoPantalla` cada vez que se cambia de vista, asegurando que la navegación por mando se adapte a los elementos de la nueva pantalla.
- `Main.html` : Define los elementos interactuables y les asigna `data-indice-mando` para controlar el orden de navegación.
- `Main.css` : Provee el estilo `.seleccionado-mando` para el feedback visual.
- `Main.mjs` : Inicializa el sistema de mando (`iniciarGamepad()`). Los listeners de click estándar funcionan tanto para ratón/teclado como para los clicks simulados por el mando.

Investigación Personal Extra: HTML y CSS

Detalles adicionales de tecnologías usadas en `Main.html` y `Main.css` que complementan la base del curso:

HTML (`Main.html`):

- `<script src="https://cdn.tailwindcss.com"></script>` : Importa **Tailwind CSS** a través de una CDN (Content Delivery Network). Tailwind es un framework CSS "utility-first" que permite estilizar elementos HTML directamente con clases predefinidas (ej. `text-3xl` , `font-bold` , `mb-6`) en lugar de escribir CSS personalizado. Agiliza el diseño y mantiene la consistencia.
- `<link rel="preconnect" ...>` y `<link href="...family=Inter..." ...>` : Cargan la fuente "Inter" desde **Google Fonts**. `preconnect` establece una conexión temprana con el servidor de fuentes para acelerar la carga. Usar fuentes externas mejora la tipografía y el aspecto visual.

- **<label>** : Asocia texto descriptivo a un elemento de formulario (`<input>` , `<select>`). Mejora la **accesibilidad**, permitiendo a los usuarios hacer clic en la etiqueta para enfocar/activar el control asociado, y es crucial para lectores de pantalla.
- **Placeholders (Uso Implícito)**: Aunque no se usa el atributo `placeholder` directamente en el HTML estático, el concepto se aplica en la UI. Por ejemplo, las imágenes de avatar (`#img-avatar-jugador` , `#img-avatar-enemigo`) muestran una imagen genérica (`https://placeholder.co/...`) como *placeholder* hasta que el JavaScript carga la imagen real del personaje/enemigo. Sirven como contenido temporal o predeterminado.

CSS (`Main.css`):

- `@keyframes aparecer` : Define una animación CSS llamada "aparecer". Describe cómo cambian los estilos (opacidad y transformación Y) a lo largo del tiempo, creando un efecto de *fade-in* y ligero deslizamiento hacia arriba. Se aplica a las `.pantalla` para suavizar las transiciones.
- Reglas para `input[type="number"]` :
 - `input[type="number"] { -moz-appearance: textfield; }` : Oculta los controles de incremento/decremento (spinners) en Firefox.
 - `input[type="number"]::-webkit-outer-spin-button, input[type="number"]::-webkit-inner-spin-button { -webkit-appearance: none; margin: 0; }` : Oculta los spinners en navegadores basados en WebKit (Chrome, Safari, Edge). Se usan para obtener un campo de número limpio, sin los controles nativos del navegador.

Conclusión

- El proyecto PBRLRPG implementa un juego de rol básico pero funcional en el navegador.
- La estructura modular facilita la comprensión y extensión del código.
- El módulo `Mando.mjs` añade una capa de accesibilidad y comodidad importante, integrándose limpiamente con el flujo existente.
- El uso de **Tailwind CSS** y **Google Fonts** (investigación extra) mejora significativamente la estética y la eficiencia del desarrollo de la interfaz.
- Técnicas CSS como `@keyframes` y la personalización de controles de input aportan pulido a la experiencia de usuario.

Posibles Mejoras:

- Soporte para más botones (ej: inventario rápido, menú).
- Navegación por ejes X.
- Feedback háptico (vibración).
- Configuración de botones.

FIN