⚔️

Aventura en el Reino de JS

↗ Mp	🌐 Cliente

1. 📄 Introducción del proyecto

`"Aventura en el Reino de JS"` es un videojuego basado en navegador. El jugador lucha contra oleadas de enemigos en una arena mística para alcanzar la máxima puntuación. La clave de la supervivencia y el éxito radica en la selección de objetos que el usuario escoja al inicio de cada aventura.

2. ✅ Criterios de evaluación y resultados de aprendizaje

- RA 2 Escribe sentencias simples, aplicando la sintaxis del lenguaje y verificando su ejecución sobre navegadores web.
  - b) Se han utilizado los distintos tipos de variables y operadores disponibles en el lenguaje.
  - c) Se han identificado los ámbitos de utilización de las variables.
  - e) Se han utilizado mecanismos de decisión en la creación de bloques de sentencias.
  - f) Se han utilizado bucles y se ha verificado su funcionamiento.
  - g) Se han añadido comentarios al código.
- RA 3 Escribe código, identificando y aplicando las funcionalidades aportadas por los objetos predefinidos del lenguaje.
  - d) Se han generado textos y etiquetas como resultado de la ejecución de código en el navegador.

    - e) Se han escrito sentencias que utilicen los objetos predefinidos del lenguaje para interactuar con el usuario.
- RA 4 Programa código para clientes web analizando y utilizando estructuras definidas por el usuario.
    - a) Se han clasificado y utilizado las funciones predefinidas del lenguaje.
    - b) Se han creado y utilizado funciones definidas por el usuario.
    - d) Se han creado y utilizado arrays.
    - f) Se ha creado código para definir la estructura de objetos.
    - g) Se han creado métodos y propiedades.
    - h) Se ha creado código que haga uso de objetos definidos por el usuario.
    - i) Se han creado y utilizado módulos.
- RA 5 Desarrolla aplicaciones web interactivas integrando mecanismos de manejo de eventos.
    - d) Se ha creado un código que capture y utilice eventos.
- RA 6 Desarrolla aplicaciones web analizando y aplicando las características del modelo de objetos del documento.
    - c) Se ha creado y verificado un código que acceda a la estructura del documento.
    - d) Se han creado nuevos elementos de la estructura y modificado elementos ya existentes.
    - e) Se han asociado acciones a los eventos del modelo.

3. 🎯 Objetivo principal

El **objetivo principal** es desarrollar una aplicación web interactiva que simula un videojuego RPG simple. En este juego, el jugador comprará objetos en un mercado para mejorar sus estadísticas y luego los usará para luchar en batallas por turnos contra enemigos, con el fin de alcanzar la mayor puntuación posible.

El **objetivo académico** (y requisito fundamental) es aplicar de manera práctica los conceptos avanzados de JavaScript vistos en clase. Esto incluye:

- **Programación Orientada a Objetos:** Usar **clases** (`Jugador`, `Enemigo`, `Producto`), **herencia**, **objetos** y **clonación** (al añadir productos al inventario).
- **Modularidad:** Estructurar el proyecto en **módulos** que separen las diferentes lógicas (Batalla, Mercado).
- **Organización del Código:** Utilizar archivos específicos de utilidades (`utils`) y constantes (`constants`) para mantener un código limpio.
- **Interactividad del DOM:** Crear una interfaz de usuario controlada por **escenas** (mostrando y ocultando elementos) y gestionar todas las acciones del jugador mediante **eventos**, sin mostrar ninguna información en la consola.
- **Buenas Prácticas:** Documentar todo el código usando **JSDoc** y utilizar **Git** para el control de versiones, siguiendo la metodología de ramas y commits aprendida.

4. 🛠️ Requisitos Técnicos y de Código

Estos requisitos definen *cómo* debe estar construido el proyecto:

- **Programación Orientada a Objetos (POO):** Uso intensivo de **Clases**, **Herencia** (`Jefe` hereda de `Enemigo`), **Objetos** y **Clonación** (al comprar productos).
- **Conceptos Clave:** Aplicar desestructuración, agrupación de datos y otros conceptos teóricos vistos.
- **Modularidad:** El proyecto debe dividirse en **módulos** que separen las diferentes lógicas (p.ej., `batalla.js`, `mercado.js`, `ranking.js`).
- **Organización:**
  - Crear un archivo `utils.js` para funciones genéricas e independientes.
  - Crear un archivo `constants.js` para valores fijos (como la puntuación base, rarezas, etc.).

- **Documentación: Todo** el código debe estar documentado utilizando JSDoc.
- **Control de Versiones (Git):**
  - Usar Git para registrar los cambios.
  - Realizar *commits* descriptivos para los cambios importantes.

5. 🎮 Requisitos Funcionales (Lógica del Juego)

5.1 Clase `Jugador`

- **Propiedades:** `nombre`, `avatar` (imagen), `puntos`, `inventario` (lista/array), `vida` (y opcionalmente `vidaMaxima`).
- **Métodos:**
  - `Añadir objeto al inventario`: Debe **clonar** el producto de la tienda antes de añadirlo.
  - `Sumar puntos`: Actualiza la puntuación al ganar batallas.
  - `Obtener ataque total`: Calcula el ataque sumando los bonus de las armas en el inventario.
  - `Obtener defensa total`: Calcula la defensa sumando los bonus de las armaduras.
  - `Obtener vida total`: Calcula la vida sumando los bonus de los consumibles.

5.2 Clases `Enemigo` y `Jefe`

- **Clase `Enemigo`:**
  - **Propiedades:** `nombre`, `avatar` (imagen), `nivel de ataque`, `puntos de vida`.
- **Clase `Jefe` (hereda de `Enemigo`):**
  - **Propiedades Adicionales:** `multiplicador de daño` (con un valor por defecto, ej: 1.2).

5.3 Clase `Producto`

- **Propiedades:** `nombre`, `imagen`, `precio`, `rareza` (Común, rara, etc.), `tipo` (solo "Arma", "armadura" o "consumible"), `bonus` (numérico).
- **Lógica de Bonus:**
  - **Arma:** El bonus se suma al ataque.
  - **Armadura:** El bonus se suma a la defensa.
  - **Consumible:** El bonus se suma a la vida.
- **Métodos:**
  - `Formatear atributos`: Ej. convertir precio `950` a `9,50€`.
  - `Aplicar un descuento`: Recibe un valor y devuelve una **nueva copia** (clon) del producto con el precio modificado.

5.4 Módulo `Mercado`

- **Contenido:** Debe tener un listado predefinido de productos.
- **Funciones:**
  - `Filtrar productos`: Por rareza.
  - `Aplicar un descuento`: A productos de un tipo o rareza específica.
  - `Buscar un producto`: Por nombre.

5.5 Módulo `Batalla`

- **Función `combate(enemigo, jugador)`:**
  - **Retorno:** El ganador y los puntos obtenidos por el jugador (0 si pierde).
  - **Lógica de Turnos:** Se repite hasta que la vida de uno llegue a 0.
  - **Cálculo por Turno:**
    - `Vida del jugador = (vida actual + defensa) - ataque enemigo`
    - `Vida del enemigo = vida actual - ataque jugador`
  - **Cálculo de Puntos (al ganar):**
    - `Puntos = 100` (base) + `ataque del enemigo`.
    - Si es un `Jefe`: `Puntos = Puntos * multiplicador de daño`.

5.6 Módulo `Ranking`

- **Función `distinguirJugador(puntuacion, umbral)`:**
  - **Lógica:** Debe tener un argumento que defina un `umbral` por defecto, que puede ser modificado.
  - **Retorno:** "Veterano" si la puntuación supera el umbral, "Novato" si no.

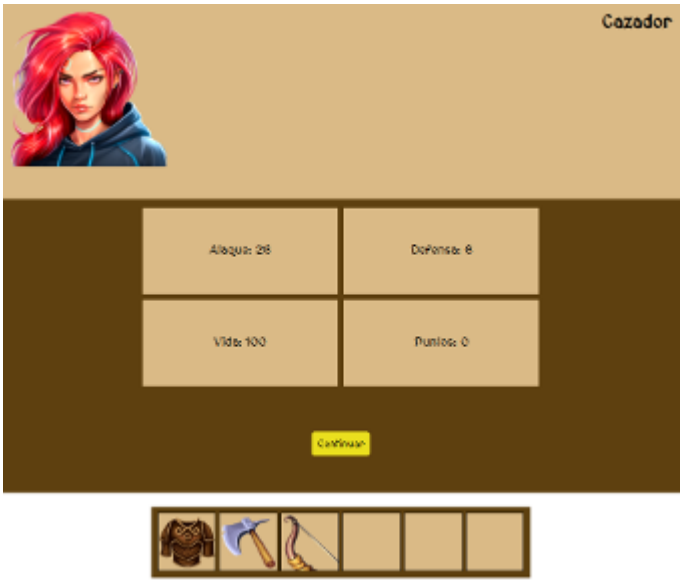
6. 🖥️ Requisitos de Interfaz de Usuario (UI) y Eventos

Esto define *cómo* el usuario interactúa con el juego:

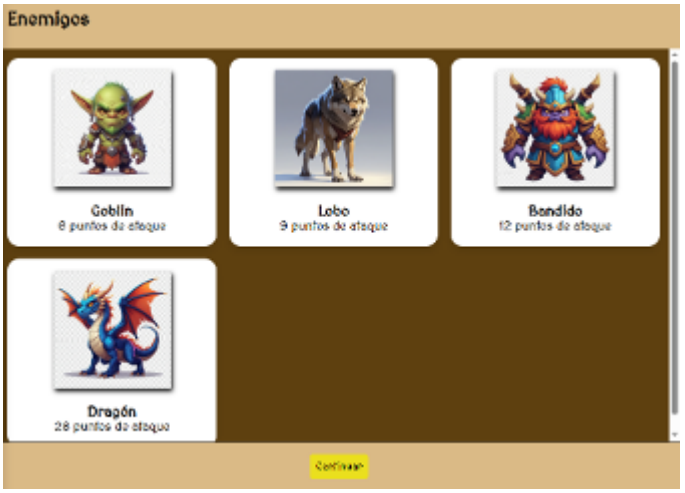
- **Sin Consola:** Toda la información y actividad (batallas, compras, estadísticas) debe mostrarse en la pantalla (HTML), **nunca en la consola del navegador**.
- **Eventos:** Toda la interacción del usuario (comprar, continuar, reiniciar) debe manejarse mediante **eventos** (clics en botones, etc.).
- **Sistema de Escenas:**
  - La navegación se basa en escenas.
  - Todas las escenas deben estar precargadas en el HTML, pero solo una estará visible a la vez (usando `display: none` o similar).
  - **Las imágenes que se mostrarán son de ejemplo, en la tarea de diseño se especificarán los requisitos.**
- **Flujo de Escenas:**
  1. **Escena 1 (Inicio):** Muestra la "tarjeta" del jugador (estado inicial).

2. **Escena 3 (Estado Actualizado):** Muestra al jugador con sus nuevas estadísticas (ataque, defensa, vida) calculadas en base a los productos comprados.

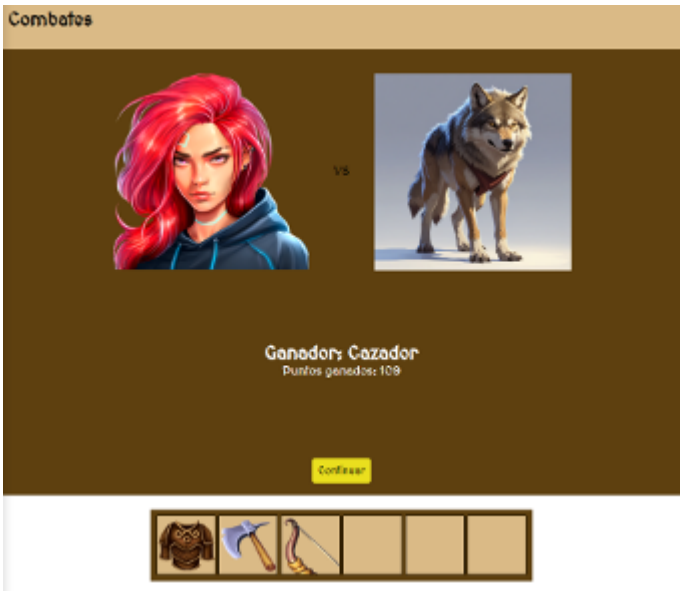
    1. **Escena 2 (Mercado):**
        - Muestra los productos (imagen, bonus, precio, botón "añadir").
        - **Descuento Aleatorio:** Al cargar esta escena, se debe aplicar un descuento a todos los productos de una **rareza elegida aleatoriamente**.
        - **Interacción de Compra:**
            - Clic en "Añadir": El fondo del producto cambia de color y el botón cambia a "Retirar".
            - Clic en "Retirar": Revierte el estado anterior.
        - **Cesta:** Los productos comprados se muestran en una cuadrícula en pantalla.



3. **Escena 4 (Enemigos):** Muestra los enemigos y sus estadísticas.



4. **Escena 5 (Batallas):** Muestra el resultado de las batallas. Debe haber un botón "Continuar" entre cada batalla. Se debe de mostrar un div con el contenido del combate (Quien ha atacado, daño realizado, vida restante...)



5. **Escena 6 (Final):** Muestra los puntos finales y el rango ("Veterano" o "Novato").

6. **Reinicio:** La última escena debe tener un botón para reiniciar el juego desde cero.

7. 📫 Requisitos de Entrega y Portfolio

- **Fecha Límite:** 1 de Diciembre.
- **Entrega Principal:** El proyecto completo comprimido, subido a Moodle.
- **Portfolio:** Una URL a tu portfolio personal que debe contener:
  1. **Enlace de Ejecución:** Un enlace al proyecto funcionando (live demo).
  2. **Enlace de Código:** Un enlace al repositorio en GitHub.
  3. **Enlace de Documentación:** Un enlace a la documentación generada con JSDoc.