MANUAL DEL PROGRAMADOR

LA BATALLA DE LOS V PUEBLOS

Realizado por:
Sandra Maria Moñino García
Angel Enrique Reyes Nieto
Cristo Macías Izaguirre

ÍNDICE

INTRODUCCIÓN

Objetivo del proyecto:

Este videojuego estaba basado en una temática parecida al Señor de los Anillos, ya que utiliza las razas de sus personajes. El modo de jugabilidad es combate por turnos, en los cuales primero hará una acción el personaje y luego el enemigo. Encontramos diferentes secciones:

- Index: Inicio del proyecto.
- Creador de personaje: Si el personaje no está creado, se tendrá que crear uno nuevo al pulsar en “Nueva partida”.
- Lobby: Espacio dónde se ve un general del personaje y desde dónde se puede ir a las diferentes secciones.
- Tienda: En la tienda el jugador puede comprar objetos para el personaje y obtener así mejoras.
- Inventario: El jugador puede equipar y desequipar a su personaje los elementos que ha comprado en la tienda, así como tomar pociones.
- Combate: Es dónde sucede el combate por turnos, si el jugador gana la batalla, el personaje obtendrá oro y experiencia.

El propósito general es que el jugador alcance el nivel máximo, con los objetos adecuados dónde se convertirá en invencible.

Resumen del código: El código está estructurado en:

Hay un html por cada escenario, estos contienen también su archivo css y para el funcionamiento, cada html puede estar vinculado con diferentes archivos .js. Más adelante aclararemos la organización y estructuración de este más a detalle.

ESTRUCTURA DE ARCHIVOS Y CARPETAS

Videojuego: (Carpeta principal).

- Arena.html
- CrearPersonaje.html
- Index.html
- Inventario.html
- Lobby.html
- Tienda.html
- Clases(Carpeta de clases)
  - Arma.js
  - Combate.js
  - CreadorPersonaje.js
  - CrearEnemigo.js
  - Cursos.js
  - Enemigos.js
  - FondoAnimado.js
  - Inicio.js
  - Inventario.js
  - InventarioFuncionamiento.js
  - Lobby.js
  - Personaje.js
  - Pocion.js
  - Proteccion.js
  - Sonido.js
  - Tienda.js
  - TiendaFuncionamiento.js
- Estilos(Carpeta donde se guardan los estilos)
  - Arena.css
  - ArenaCombate.css
  - CrearPersonaje.css
  - Index.css
  - Inventario.css
  - Lobby.css
  - Tienda.css
  - Imagenes(Carpeta donde se guardan los recursos, como imágenes, sonidos y documentación).

Esta es la organización del programa.

FUNCIONES PRINCIPALES

Funciones en:

Inicio.js: La función de esta clase está enfocada principalmente en el css. Teniendo en cuenta que si existe un archivo subido en el Local Storage muestre los botones “Continuar partida” o “Eliminar partida”. En caso del Local Storage no contenga ningún personaje, únicamente mostrará los botones principales que serían “Nueva partida”, “Repositorio” e “Instrucciones”.

CreadorPersonaje.js: Esta clase es la encargada de pedir y recoger todos los datos necesarios para el personaje y la elección obligatoria de los objetos principales. Pide en primer lugar la elección de la raza, el nombre del personaje y una vez elegido esto, se abrirán las armas principales que podrá obtener el personaje en consonancia con la raza elegida.

Personaje.js: Haciendo referencia al MVC, esta clase actuaría como Modelo, es decir, es la que permite crear el personaje una vez que la clase anterior ha recogido los datos.

Lobby.js: La clase lobby está enfocada también más en la actuación con el css, lo que hace esta clase es mostrar las características y estadísticas de los personajes, además de ser el portal para ir a cualquier otra acción, como ir a la tienda, al inventario, volver atrás o ir al combate.

Tienda.js: Es el modelo de la tienda, es el que contiene todos los objetos disponibles para las compras del usuario para el personaje.

TiendaFuncionamiento.js: Es la que controla las interacciones con el usuario, es decir, compras, navegar entre las opciones de los objetos a comprar, tener el cuenta el oro suficiente o no para la compra, así como los niveles.

Inventario.js: Es el modelo del inventario, que será un atributo del personaje. Es el lugar donde se guardarán los elementos del personaje.

InventarioFuncionamiento.js: Es el que se encarga de la gestión del inventario con el personaje, es decir, equipa las armas, las desequipa y la

opción de tirarlas. También permite beber pociones. Es el lugar dónde el usuario puede preparar al personaje para el combate.

Pocion.js: Es el modelo para las pociones, es el que establece los puntos de vida o maná que se generarán en el personaje, también establece el precio, el efecto y demás acciones que tienen las pociones.

Enemigo.js: El enemigo, sería el modelo. Es decir, recibirá las características de creador de personaje y creará este.

CrearEnemigo.js: Esta clase genera de manera automática y aleatoria las características del personaje. Teniendo unos datos establecidos y los elegirá de manera aleatoria para luego pasarlo al modelo de Enemigo y que lo crease.

Sonido.js: Se encargará de la gestión de los sonidos.

Proteccion.js: Es el modelo para las armaduras, escudos y amuletos. Que serán lo que incrementará la durabilidad del personaje. Estasblecerá qué efecto tienen y el aumento de las estadísticas del personaje relacionadas con su defensa.

COMO REALIZAR MODIFICACIONES Y AMPLIACIONES

Para realizar modificaciones, es necesario tener en cuenta los elementos que se involucran directamente en la acción que vayamos a realizar. Las clases están definidas de manera intuitiva. Modificaciones posibles serían en cuanto a estilos, que cada vez que se ataque haga un sonido y movimiento, como vibración y colores diferentes cuando se elimine a los jugadores. Para ello haría falta crear las clases del css necesarias para que se modificaran. Para tener en cuenta lo necesario que afecta directamente a las clases podemos observarlos en los .html, en los contenedores <script> asociados.

FondoDinaminco.js:

Este código crea un efecto de partículas animadas en un lienzo HTML (canvas). Al cargarse, genera 75 partículas con posiciones, velocidades, tamaños y niveles de opacidad aleatorios. Cada partícula se mueve suavemente en distintas direcciones y, si llega a un borde del lienzo, reaparece en el lado opuesto, creando un movimiento continuo. La animación se actualiza en tiempo real usando requestAnimationFrame, y el lienzo se ajusta automáticamente si el tamaño de la ventana cambia. El resultado es un fondo dinámico y visualmente atractivo con pequeñas partículas flotantes.

```
// Crear partículas
for (let i = 0; i < numParticulas; i++) {
    particulas.push(new Particle());
}

// Animar partículas
function animacion() {
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    particulas.forEach(particle => {
        particle.update();
        particle.draw();
    });
    requestAnimationFrame(animacion);
}

// Ajustar tamaño si la ventana cambia
window.addEventListener("resize", () => {
    canvas.width = window.innerWidth;
    canvas.height = window.innerHeight;
});

// Iniciar animación
animacion();
```

Sonido.js:

El script Sonido.js gestiona todos los efectos de sonido y música que hay en el videojuego. Se reproduce un sonido común a los botones generales al pasar el ratón por encima de los enlaces <a>.

En todas las páginas nos encontraremos con dos botones: #boton-musica para mutear o reproducir la musica de fondo y #silenciar-botones para mutear o activar los efectos de sonido de los botones.

La siguiente función es la función genérica que se encarga de apagar o encender ambos botones al oir el evento de ‘click’ y cambiar la imagen de ambos botones para mostrar play(cuando están activados) y pause(cuando están muteados). Además añade al sessionStorage el estado para que en el resto de páginas, mantenga el estado y no tener que estar interactuando con ellos en cada página.

El código de primeras puede asustar pero está explicado y ha sido la manera de crear esta función única para todas las páginas, además para los dos botones, y evitar la repetición de código. Con más tiempo se podría refactorizar aún más.

```
function botonesSonido(idBoton,imagen,sonidosEfectos,imagenPlay,imagenPause){
    const boton=document.querySelector(idBoton);//Recibimos el id del boton
    let sonidos=[sonidosEfectos];//Creamos array
    if(Array.isArray(sonidosEfectos)){//Si lo que nos viene ya e sun array
        sonidos=sonidosEfectos;//Lo igualamos para que no sea un array de arrays
    }
    if(boton && imagen && sonidos && sonidos.length){ //Comprobamos que existen todos en el DOM
        boton.addEventListener('click',()=>{//Para el evento de click en el boton
            const muteados=sonidos.every(sonido=> sonido.muted); //Recorremos todo el array de sonidos para capturar el muted
            if(muteados){//Si están muteado en true
                sonidos.forEach(sonido=>{ //Recorremos sonidos
                    if(sonido){//Si sonido existe en el DOM (esto es para controlar lo sonidos en otras páginas ya que esto funciona para todos los html)
                        sonido.muted=false;//Quitamos el muteo
                        if(sonido===audioMusica){ //Controlamos si en concreto es el audio de musica de fondo
                            sonido.play();//Reproducimos la musica
                            sessionStorage.setItem('musicaHabilitada','false');// Añadimos a sesion storage el estado de la musica para que persista entre paginas
                        }else{
                            sessionStorage.setItem('sonidosHabilitados','false');//Aañadimos a sesion storage el estado de los sonidos para que persista entre paginas
                        }

                    }
                });
            }else{//Si no están muteados, los apaga
                sonidos.forEach(sonido=>{//Recorremos el array de sonidos
                    if(sonido){//Si sonido existe en el DOM
                        sonido.muted=true; //Mutea a true para silenciar
                        if(sonido!==audioMusica){//Si sonido es distinto a la musica de fondo
                            sessionStorage.setItem('sonidosHabilitados','true'); //Añadimos al session storage los sonidos a true muteados
                        }else{
                            sessionStorage.setItem('musicaHabilitada','true');//Añadimos al session stotage la musica a true muteado
                        }
                    }
                })
            }
            imagen.src=muteados ? imagenPlay : imagenPause;//Cambiamos la imagen según esté reproduciendo o no
        });
    }
}
```

Las funciones para la reproducción del resto de botones con los clicks.
Tenemos asignarSonidoVarios, para botones con mismos sonidos y asignarSonidoUnico para botones únicos. (Posible refactorización para que sea general a ambos.)

```
function asignarSonidoUnico(botonUnico,idSonido){
    const boton=document.querySelector(botonUnico);
    const sonido=document.querySelector(idSonido);
    if(sonido) { efectos.push(sonido)};
    if(boton && sonido){
        boton.addEventListener('click',()=>{sonido.play()});
    }
}
```

```
137     function asignarSonidoVarios(botonesVarios,idSonido){
138         const botones=document.querySelectorAll(botonesVarios);
139         const sonido=document.querySelector(idSonido);
140         if(sonido){efectos.push(sonido)};
141         if(botones.length && sonido){
142             botones.forEach(boton=>{
143                 boton.addEventListener('click',()=>sonido.play());
144             });
145         }
146
147     }
```

Comprueba si existe el sonido en el DOM y lo añade al array de efectos. Si botones.length tiene algo/existe el botón y existe el sonido entonces itera sobre botones para activar los sonidos al hacer click en los botones.

```
45  window.addEventListener('load',()=>{
46      let sonidosHabilitados=sessionStorage.getItem('sonidosHabilitados') === 'true';
47      let musicaHabilitada=sessionStorage.getItem('musicaHabilitada')==='true';
48      actualizarBoton(imagenMusica,musicaHabilitada,imagenPlay,imagenPause);
49      actualizarBoton(imagenSilenciar,sonidosHabilitados,imagenPlay,imagenPause);
50      efectos.forEach(sonido=>{
51          if(sonido)
52          sonido.muted=sonidosHabilitados;
53      });
54      audioMusica.muted=musicaHabilitada;
55      audioMusica.play();
56  })
```

Mediante el evento de load, trabajamos la persisitenca en el sessionSotrage para que se mantenga durante la sesion las opciones de los sonidos.

Cursor.js

Es una función que recibe la raza del personaje, y según cual sea, se modifica el cursor del HTML y del body, para ello se pone el cursor en inherit de base en cada html.

```
const body=document.querySelector("body");

export function cambiarCursor(raza){
    if(raza==="humano"){
        document.documentElement.style.cursor ="url('./Imagenes/cursores/cursorHumano.svg') 16 16, auto";
        body.style.cursor="url('./Imagenes/cursores/cursorHumano.svg') 16 16, auto";
    }else if(raza==="orco"){
        document.documentElement.style.cursor ="url('./Imagenes/cursores/cursorOrco.svg') 16 16, auto";
        body.style.cursor="url('./Imagenes/cursores/cursorOrco.svg') 16 16, auto";
    }else if(raza==="enano"){
        document.documentElement.style.cursor ="url('./Imagenes/cursores/cursorEnano.svg') 16 16, auto";
        body.style.cursor="url('./Imagenes/cursores/cursorEnano.svg') 16 16, auto";
    }else if(raza==="elfo"){
        document.documentElement.style.cursor ="url('./Imagenes/cursores/cursorElfo.svg') 16 16, auto";
        body.style.cursor="url('./Imagenes/cursores/cursorElfo.svg') 16 16, auto";
    }else{
        document.documentElement.style.cursor ="url('./Imagenes/cursores/cursorMago.svg') 16 16, auto";
        body.style.cursor="url('./Imagenes/cursores/cursorMago.svg') 16 16, auto";
    }
}
```

REQUISITOS PREVIOS

Para poder utilizar el videojuego, únicamente se necesita conexión a internet, un navegador y entrar en el link del videojuego. No han sido instaladas dependencias adicionales y nada que pueda provocar errores en la jugabilidad.

CONTACTO Y AUTORÍA

Nombre de desarrolladores:

- Sandra Maria Moñino García.
- Angel Enrique Reyes Nieto.
- Cristo Macías Izaguirre.

GitHub del proyecto:
https://github.com/CristoMacias/Videojuego.git