

# **Demon's Survivor**

## **PROYECTO W3C - JUEGOS PARA WEB Y REDES SOCIALES**

**TRABAJO REALIZADO POR:**

**Raúl Caño Arévalo.**

**Darío González Fernández.**

**Adrián Soriano Valencia.**

# 1 – Descripción del juego

## 1.1 - Introducción

**Demon's Survivor** es un videojuego web perteneciente al género arcade, con una temática medievo-fantástica. El objetivo principal es conseguir sobrevivir a las continuas hordas de enemigos. Para conseguirlo, el jugador tendrá que tener la habilidad de esquivar cualquier hostilidad que haya a su alrededor.

## 1.2 – Reglas

El videojuego cuenta con las siguientes reglas:

- Cada jugador contará con tres vidas.
- Cada daño recibido por el jugador se penalizará con una vida menos.
- Si se agotan todas las vidas, termina la partida.
- El jugador tendrá que sobrevivir durante un tiempo concreto.
- Existen dos niveles de dificultad.
- A lo largo del nivel aparecen monedas recolectables de forma temporal.
- Objetos de bonificación:
  - Escudo: protege de un golpe al jugador.
  - Botas ligeras: aumenta la velocidad de movimiento del jugador hasta recibir daño.
- En el caso de haber 2 jugadores gana el jugador que mayor puntuación tenga.

## 1.3 – Puntuación



- **Recoger moneda → 20 puntos.**
- **Según el número de vidas con que se termine el nivel:**
  - **3 vidas:** 50 puntos.
  - **2 vidas:** 30 puntos.
  - **1 vida:** no suma puntos.
- **Si se termina nivel sin recibir un solo golpe → 100 puntos.**
- **Finalizar un nivel → 500 puntos.**



## 1.4 – Personajes

### Protagonistas:

Los protagonistas son heroicos caballeros de armadura pesada que buscan completar los desafíos del guardián del inframundo *Hades* para lograr así fama y riqueza.

JUGADOR 1	JUGADOR 2
	

### Enemigos

Nombre	Forma	Descripción
<b>No-Muerto</b>		Es el enemigo más común. Son lentos pero numerosos. Se mueve en línea recta.
<b>Esbirro del infierno</b>		Es el enemigo más rápido. Se mueve en línea recta.
<b>Teniente infernal</b>		Es el enemigo menos predecible. Se mueve haciendo zig-zag.
<b>Calavera Escupefuego</b>		Se mueve en cualquier dirección dejando fuego a su paso.

## 2 – Desglose de páginas

### 2.1 – Main.html

Es la página principal del videojuego. Cuenta con:

- ❖ Título.
- ❖ Menú principal:
  - Jugar:
    - Versión escritorio: lleva a la pantalla de selección de números de jugadores.
    - Versión móvil: lleva a la pantalla del primer nivel.
  - Instrucciones:
    - Reglas del juego.
    - Controles.
  - Puntuaciones: desglose de las 10 mejores puntuaciones.
  - Créditos:
    - Nombres de los creadores.
    - Email de contacto.
  - Opciones.
    - Idioma → español o inglés.
    - Sonido → con o sin sonido.
    - Ambas opciones únicamente se pueden cambiar en este menú y se mantendrán en el resto de páginas.
- ❖ Copyright.

Esta página además cuenta con un código JavaScript. Para que cada opción se aplicara de forma dinámica se ha utilizado jQuery. Las mayores complicaciones han sido el poder cambiar de idioma y apagar o encender el sonido, y que además dichas opciones se respetaran en el resto de páginas.

#### Para las puntuaciones:

Para las puntuaciones se ha utilizado el **localStorage** de html. Se crea un array con 10 puntuaciones que en un principio se inicializan a 0. Por cada partida que se juega, se compara la puntuación del jugador con la puntuación más baja de la lista y si es mayor, se intercambia. Posteriormente, mediante el método de la burbuja, se ordena la lista en orden descendente.

#### Para el idioma:

Para cambiar el idioma hemos creado una variable que guarda la cadena de texto del final de la URL. En este caso puede ser EN (en el caso del idioma inglés) o ES (en el caso del español). Dependiendo del botón que esté activo en el momento (bandera inglesa o española) se cargará el contenido en uno u otro idioma y se actualizará la URL automáticamente a EN o a ES sin necesidad de volver a cargar la página. En el caso de pasar a la siguiente página, este final de URL se mantiene para cargar el contenido con el idioma correcto. Esto funciona tanto hacia adelante como hacia atrás.

### En cuanto al sonido:

Automáticamente, al cargar la página se reproduce una música que se repite al terminar. En el caso de “mutear” el sonido de la web a través del botón de silencio, se cambia el valor de una booleana y dicho valor se guarda en la memoria local. Este método funciona bien hacia adelante, y cuando se cargan posteriores páginas se comprueba la booleana de sonido y se reproduce o no la música.

En el caso de volver hacia atrás, puesto que en el script se limpia la memoria local para evitar errores, es imposible mantener la opción correcta de sonido. Para solventarlo, se sigue un procedimiento similar al idioma. Mediante la URL se añade un ESOFF (español con sonido muteado) o un ENOFF (inglés con sonido muteado) y dependiendo de esto, se reproduce o no el sonido.

## 2.2 – ElegirNumJugadores.html

Esta página es exclusiva de la versión de escritorio y cuenta con:

- ❖ Título.
- ❖ Elegir número de jugadores:
  - Un jugador ➔ lleva a la pantalla del primer nivel con un solo jugador.
  - Dos jugadores ➔ lleva a la pantalla del primer nivel con dos jugadores.
- ❖ Atrás ➔ vuelve al menú anterior (main.html).

Al igual que el main.html cuenta con código integrado JavaScript en el cual, de forma dinámica con jQuery, el contenido se carga en uno u otro idioma. Para saber el idioma elegido por el jugador basta con comprobar el final de la URL.

Para respetar la opción de sonido se guarda en una variable el contenido de una de las variables guardadas en la memoria local. Dependiendo del valor de dicha variable, se carga o no el sonido y se limpia la memoria para evitar errores.

## 2.3 – UnJugador.html

Este html contiene el archivo .js que contiene el juego en sí. Consta de:

# Inicializaciones

## 1. Sprites.

```
var pjImg = new Image();
pjImg.src = "Sprites/Prota.png";

var pjShieldImg = new Image();
pjShieldImg.src = "Sprites/ProtaShield.png";

var skeletonImg = new Image();
skeletonImg.src = "Sprites/Skeleton.png";

var vampireImg = new Image();
vampireImg.src = "Sprites/Vampire.png";

var demonImg = new Image();
demonImg.src = "Sprites/Demon.png";

var fireMobImg = new Image();
fireMobImg.src = "Sprites/Firemob.png";

var fireImg = new Image();
fireImg.src = "Sprites/Fire.png";

var coinImg = new Image();
coinImg.src = "Sprites/Coin.png";

var bootImg = new Image();
bootImg.src = "Sprites/Boots.png";

var shieldImg = new Image();
shieldImg.src = "Sprites/Shield.png";

var fondoImg = new Image();
fondoImg.src = "Interfaz/Mapa1.png";
```

## 2. Arrays de enemigos y objetos.

```
var skeletons = [];  
var vampires = [];  
var demons = [];  
var fireMobs = [];  
var fires = [];  
var coins = [];  
var boots = [];  
var shields = [];
```

## 3. Variables públicas necesarias.

```
var id = 0;  
var speed = 3;  
var shieldTimer = 5;  
var shieldCollision = false;  
var shieldActive = false;  
var firstShield = false;  
  
var pause = false;  
var lock = 0;  
  
var RestartByGover = false;  
var nextLevel = false;  
var salir = false;  
var quiereSalir = false;  
var contMonedas = 0;  
var contVidas = 0;
```

## 4. Audios.

```
var audioSelect = document.createElement('audio');  
audioSelect.setAttribute('src', 'Sonido/select.mp3');  
audioSelect.className = "sound";  
var audioHit = document.createElement('audio');  
audioHit.setAttribute('src', 'Sonido/hit.mp3');  
audioHit.className = "sound";  
var audioCoin = document.createElement('audio');  
audioCoin.setAttribute('src', 'Sonido/coin.mp3');  
audioCoin.className = "sound";  
var audioBoots = document.createElement('audio');  
audioBoots.setAttribute('src', 'Sonido/boots.mp3');  
audioBoots.className = "sound";  
var audioShield = document.createElement('audio');  
audioShield.setAttribute('src', 'Sonido/shield.mp3');  
audioShield.className = "sound";  
var audioStart = document.createElement('audio');  
audioStart.setAttribute('src', 'Sonido/start.mp3');  
audioStart.className = "sound";  
var audioPause = document.createElement('audio');  
audioPause.setAttribute('src', 'Sonido/pausa.mp3');  
audioPause.className = "sound";  
var audioOver = document.createElement('audio');  
audioOver.setAttribute('src', 'Sonido/game over.mp3');  
audioOver.className = "sound";  
var audioVictory = document.createElement('audio');  
audioVictory.setAttribute('src', 'Sonido/victory.mp3');  
audioVictory.className = "sound";
```

## 5. Canvas.

```
var canvas = document.getElementById("pjAnimation");

var body = document.body;
console.log(body.offsetWidth);

/////TAMAÑO DEL CANVAS EN FUNCION DE TAMAÑO DE VENTANA

canvas.width = body.offsetWidth * 0.41666667;
canvas.height = body.offsetHeight * 0.6322444;
```

## 6. Temporizadores.

```
var SkeletonTimer, VampireTimer, DemonTimer, FireMobTimer, CoinTimer, BootTimer, ShieldTimer;
var numSkeleton=500, numVampire=2000, numDemon=4000, numFireMob=4000, numCoin=5000, numBoot=10000, numShield=15000;

toHour=0;
toMinute=0;
toSecond=5;
```

# Funciones

## 1. GameLoop.

```
function gameLoop () {

    window.requestAnimationFrame(gameLoop);

    if(!pause){
        var ctx = canvas.getContext("2d");
        ctx.clearRect(0,0,800,600);
        ctx.drawImage(fondoImg, 0, 0);
        checkPosition(pj);
        for(var i = 0; i < skeletons.length; i++){
            if(skeletons[i] != null){
                checkPosition(skeletons[i]);
            }
        }
        for(var i = 0; i < vampires.length; i++){
            if(vampires[i] != null){
                checkPosition(vampires[i]);
            }
        }
        for(var i = 0; i < demons.length; i++){
            if(demons[i] != null){
                checkPosition(demons[i]);
            }
        }
        for(var i = 0; i < fireMobs.length; i++){
            if(fireMobs[i] != null){
                fireMobs[i].count += 1;
                checkPosition(fireMobs[i]);
            }
        }

        pj.update();
        pj.render();

        checkShields();
        checkBoots();
        checkCoins();
        checkFires();
        checkSkeletons();
        checkVampires();
        checkDemons();
        checkFireMobs();
    }
}
```

La función **GameLoop** se encarga de ejecutarse cientos de veces para poder llamar a todas las funciones necesarias para el renderizado y la actualización de la posición de todos los pjs.

Como podemos observar, cada vez que se ejecuta el **GameLoop** se limpia el canvas, se dibuja el fondo, se comprueba que los pjs pueden seguir avanzando sin salirse de los límites, se actualiza y renderiza el pj principal y finalmente se comprueban las colisiones con cada uno de los enemigos.



## 2. Restart.

```
function Restart(numVidasActual, RestartByGover){
  skeletons = [];
  vampires = [];
  demons = [];
  fireMobs = [];
  fires = [];
  coins = [];
  boots = [];
  shields = [];

  id = 0;
  speed = 3;
  shieldTimer = 5;
  shieldCollision = false;
  shieldActive = false;
  firstShield = false;

  lock = 0;

  pj.lifes = 3;
  pj.score = 0;

  toSecond=20;
  $('#Ctimer').empty();
  $('#Ctimer').append(toMinute+" ":"+ toSecond);

  console.log("Entra en la de restart");
  console.log(RestartByGover);
  console.log(numVidasActual + "HASKDHASKDWWQE");
  if(RestartByGover && !quiereSalir){
    $("#divVidasContenedor").append('');
    $("#divVidasContenedor").append('');
    $("#divVidasContenedor").append('');
  }
  switch(numVidasActual) {
    case 3:
      break;
    case 2:
      $("#divVidasContenedor").append('');
      break;
    case 1:
      $("#divVidasContenedor").append('');
      $("#divVidasContenedor").append('');
      break;
    default:
      break;
  }

  //countDown();
}
```

La función **restart** se encarga de reiniciar los arrays de enemigos, temporizadores, la vida del personaje y la puntuación. Se llama al pasar de nivel, reiniciar o perder.

### 3. Sprite.

```
function sprite (options) {
  var that = {},
      tickCount = 0,
      numberOfFramesX = options.numberOfFramesX || 9,
      numberOfFramesY = options.numberOfFramesY || 4;
  that.frameIndex = 1;
  that.ticksPerFrame = options.ticksPerFrame;
  that.context = options.context;
  that.width = options.width;
  that.height = options.height;
  that.imgwidth = options.imgwidth;
  that.imgHeight = options.imgHeight;
  that.xPos = options.xPos;
  that.yPos = options.yPos;
  that.xSpeed = options.xSpeed;
  that.ySpeed = options.ySpeed;
  that.dir = options.dir;
  that.stop = options.stop;
  that.image = options.image;
  that.count = options.count;
  that.lives = 3;
  that.score = 0;
}
```

Inicialización de la función que va a servir de prototipo para todos los personajes que requieran moverse y renderizarse.

```
that.render = function (){
  if(numberOfFramesY > 1){
    that.context.drawImage(
      that.image,
      that.frameIndex * that.width / numberOfFramesX,    //xPos en la imagen
      that.dir * that.height / numberOfFramesY,    //yPos en la imagen
      that.width / numberOfFramesX,    //width en la imagen
      that.height / numberOfFramesY,    //height en la imagen
      that.xPos,    //xPos en el canvas
      that.yPos,    //yPos en el canvas
      50,    //Animation width
      50);    //Animation height
  }else{
    that.context.drawImage(
      that.image,
      that.frameIndex * that.width / numberOfFramesX,    //xPos en la imagen
      that.dir * that.height / numberOfFramesY,    //yPos en la imagen
      that.width / numberOfFramesX,    //width en la imagen
      that.height / numberOfFramesY,    //height en la imagen
      that.xPos,    //xPos en el canvas
      that.yPos,    //yPos en el canvas
      35,    //Animation width
      35);    //Animation height
  }
};
```

Renderiza los personajes a 50x50 y los objetos a 35x35.

```

that.update = function () {
    tickCount += 1;

    if(tickCount > that.ticksPerFrame){
        tickCount = 0;

        if(that.frameIndex < numberOfFramesX - 1 && that.stop == false){
            that.frameIndex += 1;
        }else{
            that.frameIndex = 1;
        }
    }
}

};

```

Función utilizada para decidir el número de pasadas que se tiene que pasar de fotograma.

```

that.checkSkeleton = function(){
    var die = false;

    if(that.xSpeed > 0){
        if (that.xPos > body.offsetWidth * 0.41666667 - 50){
            die = true;
        }
    }else{
        if(that.xPos < 3){
            die = true;
        }
    }
    if(that.ySpeed > 0){
        if (that.yPos > body.offsetHeight * 0.63224447 - 50){
            die = true;
        }
    }else{
        if(that.yPos < 3){
            die = true;
        }
    }

    return die;
};

```

Estas funciones (**checkDemon**, **checkFiremob**, etc) comprueban si el personaje sigue dentro de la zona del canvas, en caso contrario los elimina. En el caso de los demonios solo se eliminan cuando llegan abajo, si chocan con las paredes de los lados cambian de sentido. Los *Firemobs* van haciendo una escalera dejando un rastro de fuego que se elimina a los ciertos segundos.

```

var pj = sprite ({
    context: canvas.getContext("2d"),
    ticksPerFrame: 5,
    frameIndex: 1,
    width: 274,
    height: 365,
    xPos: 350,
    yPos: 350,
    xSpeed: 0,
    ySpeed: 0,
    dir: 0,
    stop: true,
    numberOfFramesX: 3,
    numberOfFramesY: 4,
    image: pjImg
});

```

Inicialización del pj principal usando el prototipo sprite.

#### 4. CheckPosition.

```
function checkPosition(pjs){
  if(pjs.xSpeed > 0 && pjs.xPos <= body.offsetWidth * 0.41666667 - 50)
    pjs.xPos += pjs.xSpeed;
  if(pjs.xSpeed < 0 && pjs.xPos >= 3)
    pjs.xPos += pjs.xSpeed;
  if(pjs.ySpeed > 0 && pjs.yPos <= body.offsetHeight * 0.63224447 - 50)
    pjs.yPos += pjs.ySpeed;
  if(pjs.ySpeed < 0 && pjs.yPos >= 3)
    pjs.yPos += pjs.ySpeed;
}
```

Actualiza la posición del pj pasado por parámetro si se encuentra dentro del canvas.

#### 5. Checks.

```
function checkSkeletons(){
  for(var i = 0; i < skeletons.length; i++){
    if(skeletons[i] != null){
      if(intersectRect(
        pj.yPos + 5,
        pj.yPos + 45,
        pj.xPos + 12,
        pj.xPos + 38,
        skeletons[i].yPos,
        skeletons[i].yPos + 50,
        skeletons[i].xPos + 10,
        skeletons[i].xPos + 40 || skeletons[i] != null))
      {
        pj.image = pjImg;
        pj.lives -= 1;
        quitarVida(pj.lives);

        if(pj.lives == 0){
          audioOver.play();
          endGame();
        }
        shieldCollision = true;
        console.log("Lives: "+pj.lives);
        skeletons.splice(i, 1);
      }else if(skeletons[i] != null){
        if(skeletons[i].checkSkeleton() == true){
          skeletons.splice(i, 1);
        }else{
          skeletons[i].update();
          skeletons[i].render();
        }
      }
    }
  }
}
```

Cada enemigo y cada objeto posee una función similar a esta en las que se comprueba si el pj ha colisionado con el enemigo o el objeto. En caso de colisionar con un enemigo, se resta una vida y se elimina el enemigo. Si es un objeto, se añadirán sus propiedades al pj, por ejemplo, una moneda suma 20 puntos. También se comprueba si el personaje ha perdido todas sus vidas, caso en el que perdería la partida.

#### 6. IntersectRect.

```
function intersectRect(topA, bottomA, leftA, rightA, topB, bottomB, leftB, rightB) {
  return !(leftB > rightA ||
    rightB < leftA ||
    topB > bottomA ||
    bottomB < topA);
}
```

Comprueba la colisión de dos rectángulos.

## 7. Creates.

```
function createSkeleton(){
  var skeleton = sprite({
    context: canvas.getContext("2d"),
    ticksPerFrame: 8,
    width: 274,
    height: 365,
    xPos: 0,
    yPos: 0,
    xSpeed: 0,
    ySpeed: 0,
    dir: Math.floor((Math.random() * 4)),
    stop: false,
    numberOfFramesX: 3,
    numberOfFramesY: 4,
    image: skeletonImg
  });

  switch(skeleton.dir){
    case 0:
      skeleton.xPos = Math.floor((Math.random() * body.offsetWidth * 0.41666667 - 50));
      skeleton.ySpeed = 1;
      break;
    case 1:
      skeleton.xPos = body.offsetWidth * 0.41666667 - 50;
      skeleton.yPos = Math.floor((Math.random() * body.offsetHeight * 0.63224447 - 50));
      skeleton.xSpeed = -1;
      break;
    case 2:
      skeleton.yPos = Math.floor((Math.random() * body.offsetHeight * 0.63224447 - 50));
      skeleton.xSpeed = 1;
      break;
    case 3:
      skeleton.yPos = body.offsetHeight * 0.63224447 - 50;
      skeleton.xPos = Math.floor((Math.random() * body.offsetWidth * 0.41666667 - 50));
      skeleton.ySpeed = -1;
      break;
  }
  skeletons.push(skeleton);
}
```

Cada enemigo y cada objeto posee una función similar a esta donde se crea cada enemigo eligiendo la dirección y posición en la que aparece aleatoriamente.



## 8. Move & Stop.

```
function move(e){  
    if(e.keyCode==39 && pj.xPos<body.offsetWidth * 0.41666667 - 50){  
        pj.xSpeed = speed;  
        pj.dir = 2;  
    }  
    if(e.keyCode==37 && pj.xPos>0){  
        pj.xSpeed = -speed;  
        pj.dir = 1;  
    }  
    if(e.keyCode==40 && pj.yPos<body.offsetHeight * 0.63224447 - 50){  
        pj.ySpeed = speed;  
        pj.dir = 0;  
    }  
    if(e.keyCode==38 && pj.yPos>0){  
        pj.ySpeed = -speed;  
        pj.dir = 3;  
    }  
    if(!pause){  
        if(e.keyCode==80 || e.keyCode==32){  
            pausa();  
        }  
    }else if(e.keyCode == 80 || e.keyCode == 32){  
        vuelta();  
    }  
    pj.stop = false;  
}  
  
function stop(e){  
    if(e.keyCode == 39 || e.keyCode == 37){  
        pj.xSpeed = 0;  
        if(pj.ySpeed > 0){  
            pj.dir = 0;  
        }else if(pj.ySpeed < 0){  
            pj.dir = 3;  
        }  
    }  
  
    if(e.keyCode == 40 || e.keyCode == 38){  
        pj.ySpeed = 0;  
        if(pj.xSpeed > 0){  
            pj.dir = 2;  
        }else if(pj.xSpeed < 0){  
            pj.dir = 1;  
        }  
    }  
  
    if(pj.xSpeed == 0 && pj.ySpeed == 0){  
        pj.stop = true;  
    }  
}
```

Estas funciones son llamadas respectivamente cuando pulsamos o levantamos una tecla.

En la función **move** cuando pulsamos una tecla se establece una velocidad en una dirección dependiendo del botón pulsado.

En la función stop cuando se levanta el dedo de una tecla se elimina la velocidad y se pone el pj en posición estática.

## 9. endGame.

```
function endGame(){
  pause = true;
  pauseSpawn();
  parameters = location.search.substring(1).split("&").toString();
  if(parameters === "ES"){
    $("#interfaz").append('<div id="contGameOver"/>');
    $("#contGameOver").append('<div id="divTituloGameOver"><img id=');
    $("#contGameOver").append('<div id="botonesGameOver"><div id="d');
  }
  else{
    $("#interfaz").append('<div id="contGameOver"/>');
    $("#contGameOver").append('<div id="divTituloGameOver"><img id=');
    $("#contGameOver").append('<div id="botonesGameOver"><div id="d');
  }
  obtener_localStorage(pj.lifes);
}
```

Función llamada cuando perdemos. Pausa la generación de enemigos y despliega un menú con varios botones para poder reiniciar o salir.

## 10. Timer.

```
function Timer(callback, delay) {
  var timerId;

  this.pause = function() {
    window.clearTimeout(timerId);
  };

  this.resume = function() {
    window.clearTimeout(timerId);
    timerId = window.setTimeout(callback, delay);
  };

  this.levelup = function(){
    window.clearTimeout(timerId);
    console.log("Level up");
    delay = delay/1.5;
    console.log(delay);
    timerId = window.setTimeout(callback, delay);
  }

  this.leveldown = function(){
    window.clearTimeout(timerId);
    console.log("Level down");
    delay = delay * 1.5;
    console.log(delay);
    timerId = window.setTimeout(callback, delay);
  }

  this.resume();
}
```

Función prototipo de un temporizador, el cual se llama repetidamente cada cierto tiempo. Ese tiempo se puede reducir con el método **levelup** o aumentar con **leveldown**. También se puede pausar el temporizador.

## 11. BeginGame.

```
function beginGame(){
    SkeletonTimer = new Timer(function() {
        createSkeleton();
        SkeletonTimer.resume();
    }, numSkeleton);

    VampireTimer = new Timer(function() {
        createVampire();
        VampireTimer.resume();
    }, numVampire);

    DemonTimer = new Timer(function() {
        createDemon();
        DemonTimer.resume();
    }, numDemon);

    FireMobTimer = new Timer(function() {
        createFireMob();
        FireMobTimer.resume();
    }, numFireMob);

    CoinTimer = new Timer(function() {
        createCoin();
        CoinTimer.resume();
    }, numCoin);

    BootTimer = new Timer(function() {
        createBoot();
        BootTimer.resume();
    }, numBoot);

    ShieldTimer = new Timer(function() {
        createShield();
        ShieldTimer.resume();
    }, numShield);
}
```

Se crean los distintos temporizadores a partir del prototipo Timer, cada uno con sus tiempos de generación. Dentro de cada temporizador se llama a las funciones que crean cada tipo de enemigo o de objeto.

## 12. Control de los temporizadores.

```
function resumeSpawn(){
    pause = false;
    SkeletonTimer.resume();
    VampireTimer.resume();
    DemonTimer.resume();
    FireMobTimer.resume();
    CoinTimer.resume();
    BootTimer.resume();
    ShieldTimer.resume();
}

function pauseSpawn(){
    SkeletonTimer.pause();
    VampireTimer.pause();
    DemonTimer.pause();
    FireMobTimer.pause();
    CoinTimer.pause();
    BootTimer.pause();
    ShieldTimer.pause();
}

function levelup(){
    SkeletonTimer.levelup();
    VampireTimer.levelup();
    DemonTimer.levelup();
    FireMobTimer.levelup();
    CoinTimer.levelup();
    BootTimer.levelup();
    ShieldTimer.levelup();
}

function leveldown(){
    SkeletonTimer.leveldown();
    VampireTimer.leveldown();
    DemonTimer.leveldown();
    FireMobTimer.leveldown();
    CoinTimer.leveldown();
    BootTimer.leveldown();
    ShieldTimer.leveldown();
}
```

Todas estas funciones se usan para pausar, continuar, acelerar o decelerar los temporizadores creados anteriormente.

### 13. countdown.

```
function countdown(){
    toSecond=toSecond-1;
    $('#ctimer').empty();
    $('#ctimer').append(toMinute+": "+ toSecond);
    if(toSecond<0 && toMinute!=0)
    {
        toSecond=59;
        toMinute=toMinute-1;
    }
    console.log(toMinute+": "+toSecond);

    if(toSecond == 0 && toMinute == 0){
        pj.score += 500;
        switch (pj.lives){
            case 3:
                contVidas = 50;
                break;
            case 2:
                contVidas = 30;
                break;
            default:
                break;
        }
        pj.score += contVidas;

        levelCompleted();
        Restart(pj.lives, false);

        obtener_localStorage(pj.score);

    }else if(!pause){
        setTimeout("countDown()",1000);
    }
}
```

En esta función se incluye el temporizador general del juego, el cual señala el tiempo que tienes que aguantar en el nivel para pasarlo. Cuando el contador llega a 0, se llama a la función de **Restart** de la que hemos hablado antes, así como a **LevelCompleted** para poder acceder al siguiente nivel. También se suman puntos por acabar el nivel con 3 vidas o 2.

Además de las funcionalidades del juego, el archivo .js incluye unas líneas **jquery** en las que se configura el apartado de la interfaz y la interacción dinámica.



## 2.4 – DosJugadores.html

Este html incluye el .js para la modalidad de 2 jugadores. Incluye exactamente lo mismo que en la modalidad de 1 jugador, con la salvedad de que se introduce la funcionalidad de un segundo jugador y la eliminación de los dos niveles.

En este modo, los dos jugadores compiten por conseguir el mayor número de monedas posibles. No hay límite de tiempo. El juego termina cuando los dos jugadores mueren.

## 2.5 – Versiones móviles

Para las versiones móviles se han añadido dos html extra (**M\_Main.html**, **M\_UnJugador.html**), y un .js y .css específicos para adaptarlo a este formato.

# 3 – Diseño

## 3.1 – Apartado Visual

Se ha seguido una estética similar a la que se puede encontrar en juegos como Diablo (Blizzard). Hemos buscado material con licencia de tipo *creative commons* como requería la práctica.

Hemos escogido una misma temática para los fondos de pantalla. ([Referencia 1](#)).

Para generar las fuentes de los títulos y la fuente de los botones se ha utilizado una web generadora de fuentes ([Referencia 2](#)). Posteriormente, estas fuentes han sido modificadas con Photoshop.

La fuente utilizada en descripciones o similares, ha sido la fuente Carousel ([Referencia 3](#)).

Para los botones se ha partido de una base ([Referencia 4](#)), modificándose en Photoshop y añadiéndose la fuente necesaria.

Para las imágenes de las instrucciones se ha seguido el mismo procedimiento que con los botones.

Los personajes, enemigos, ítems del juego e imágenes de créditos se han conseguido a partir de diversas páginas ([Referencia 5](#)). Se ha intentado seguir un estilo uniforme en la elección de estos elementos. Algunos elementos han sido modificados con Photoshop.

El mapa de juego es creación propia.

Además, hemos incluido una imagen de favicon ([Referencia 6](#)) y hemos cambiado el estilo del ratón ([Referencia 7](#)).

## 3.2 – Sonido

En cuanto al sonido:

- La música que suena en cada html ha sido compuesta y creada por el equipo con el software FL Studio.
- Los efectos de sonido han sido descargados de la web [www.freesound.org](http://www.freesound.org) y han sido modificados con el programa **Adobe Audition**.

## 4 – CSS

En la hoja de estilos se ha trabajado con medidas de porcentajes tanto en la versión de escritorio como en la versión móvil para que la visualización se adapte a los distintos tamaños.

# REFERENCIAS

Referencia 1 - <http://www.xsijys.com/fantasy-lava-wallpapers.html>

Referencia 2 - <https://es.cooltext.com/>

Referencia 3 - <https://www.dafont.com/es/>

Referencia 4 - <https://opengameart.org/content/buttonsss>

Referencia 5 - <http://www.freeflagicons.com/> <https://opengameart.org>

Referencia 6 – <https://openclipart.org>

Referencia 7 - <http://chaosrealm.info/topic/8632598/2/>