Lordran Knight – Documentación - Julián Méndez

Índice:

Clases

Para el desarrollo de este proyecto hago uso de 4 clases. Estas son: Sprite, Jugador, Mago y Plataforma. Jugador y Mago heredan ambas de Sprite pero la diferencia reside en cómo están animados los sprites.

En la clase Jugador los sprites están animados de tal forma que cada frame lo compone una imagen individual. Sin embargo en el caso de Mago los sprites se animan todos a partir de una misma imagen.

(Distribución de las imágenes que componen la animación de atacar. Este sería un ejemplo de un Sprite que use la clase Jugador)


(Ejemplo de cómo se anima un Sprite dentro de una sola imagen. Este sería un ejemplo de un Sprite que use la clase Mago)

La forma que tiene la clase Jugador de recorrer los frames es más simple que la de Mago ya que no hay que ir seleccionando que fragmento de la imagen mostrar si no que simplemente muestra la imagen completa. Entonces lo único que tenemos que hacer es un bucle que va mostrando las diferentes imágenes.

```
actualizarFrames() {
    this.framesTranscurridos++

    if (this.framesTranscurridos % this.framesEspera === 0) {

        if (this.frameActual < this.framesMaxAnimacion - 1) {
            this.imagen.src = this.ruta + this.frameActual + ".png"

            //la ejecucion solo entra aqui cuando ataca. esta condicion esta para saber cuando es que el jugador hace daño
            if (this.estaAtacando && (this.frameActual == 6 || this.frameActual == 7 || this.frameActual == 10 || this.frameActual == 11)) {
                this.haceDano = true
            } else this.haceDano = false

            this.frameActual++

        } else {
            //si el jugador ha muerto no se resetea el contador si no que muestra el ultimo frame de la animacion de muerte
            if (this.animacionMuerte) this.imagen.src = this.ruta + 11 + ".png"
            else this.frameActual = 1

        }
    }
}
```

Para cambiar el tipo de animación del jugador hacemos uso de la función setSprite. Esta función recibe por parámetro una ruta para la nueva animación, frames máximos de la animación y los frames de espera que queremos para esa animación (cuanto mayor es el número de los frames de espera más lenta es la animación y viceversa). Entonces la función se va a encargar de cambiar la ruta de la imagen y de actualizar los campos de frames máximos y frames de espera.

```
setSprite(sprite) {

    if (sprite == sprites.atacar) this.estaAtacando = true
    else this.estaAtacando = false

    if (sprite == sprite.muerte) this.animacionMuerte = true

    this.ruta = sprite.src
    this.framesMaxAnimacion = sprite.framesMaxAnimacion
    this.framesEspera = sprite.framesEspera
}
```

```
const sprites = {
    quieto: {
        src: "/imagenes/sprites/jugador/idle/Warrior_Idle_",
        framesMaxAnimacion: 6,
        framesEspera: 10
    },
    correrDerecha: {
        src: "/imagenes/sprites/jugador/runRight/Warrior_Run_",
        framesMaxAnimacion: 8,
        framesEspera: 5
    },
    correrIzquierda: {
        src: "/imagenes/sprites/jugador/runLeft/Warrior_Run_",
        framesMaxAnimacion: 8,
        framesEspera: 5
    },
    agachar: {
        src: "/imagenes/sprites/jugador/agachar/Warrior_Crouch_",
        framesMaxAnimacion: 4,
        framesEspera: 15
    },
    saltarDerecha: {
        src: "/imagenes/sprites/jugador/saltarRight/Warrior_Jump_",
        framesMaxAnimacion: 3,
        framesEspera: 15
    },
    saltarIzquierda: {
        src: "/imagenes/sprites/jugador/saltarLeft/Warrior_Jump_",
        framesMaxAnimacion: 3,
        framesEspera: 15
    },
    atacar: {
        src: "/imagenes/sprites/jugador/atacar/Warrior_Attack_",
        framesMaxAnimacion: 12,
        framesEspera: 4
    },
```

(Constante la cual contiene los valores que le vamos a pasar a la función setSprite)

La clase Mago usa la misma forma de animar que su clase padre Sprite. Esta forma consiste en dividir la imagen dependiendo del numero frames que compongan la animación. De esta forma lo que hacemos es un bucle que va renderizando fragmentos de la misma imagen. En el caso de la imagen anterior, esta se divide en 8 fragmentos (uno para cada imagen) y cada x tiempo se va mostrando una porción. Al llegar al final esta vuelve al principio.

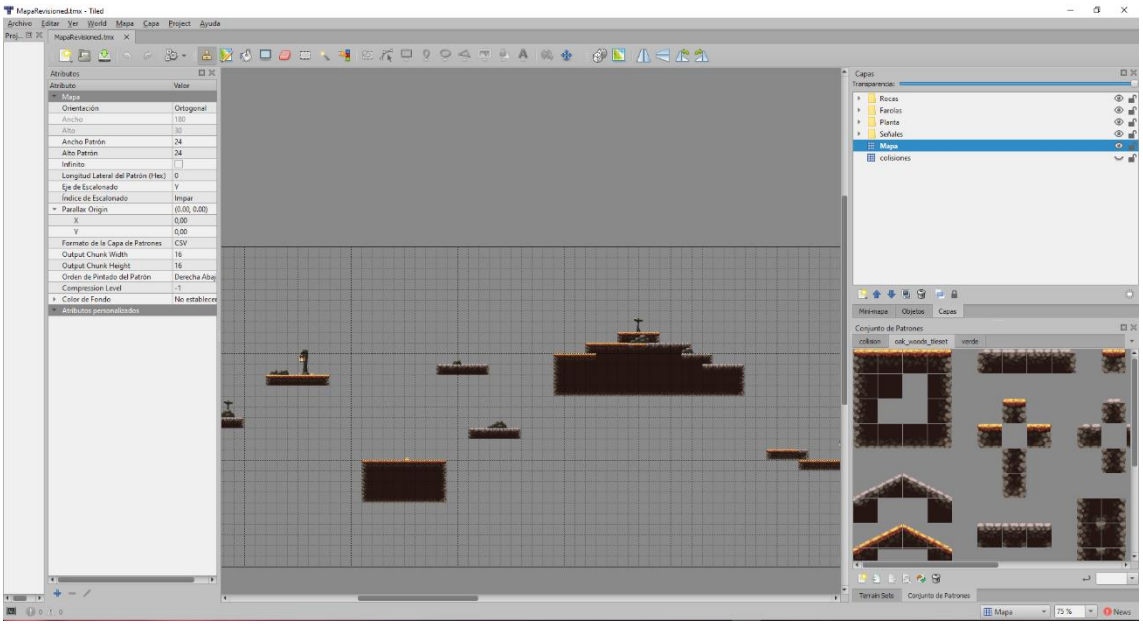
```
render() {
    c.drawImage(this.imagen,
        sx: this.frameActual * (this.imagen.width / this.framesHorizontales),//posicion X inicial
        sy: 0,//posicion Y inicial
        sw: this.imagen.width / this.framesHorizontales,//tamañano del recorte, en este caso es la longitud entre
                                                        // el numero de frames de animacion dentro de la imagen
       this.imagen.height ,//alto del recorte
        dx: this.posicion.x - this.margenSprite.x,//posicion del canvas en la que se va a colocar
        dy: this.posicion.y - this.margenSprite.y,
        dw: (this.imagen.width / this.framesHorizontales) * this.escala,//tamaño que se va a mostrar
        dh: (this.imagen.height ) * this.escala)
}
```

```
actualizarFrames() {
    this.framesTranscurridos++

    if (this.framesTranscurridos % this.framesEspera === 0) {

        if (this.frameActual < this.framesHorizontales - 1) {
            this.frameActual++
        } else {
            this.frameActual = 0
        }
    }
}
```

(Función que se encarga de hacer un bucle a través de los frames de la animación)

Mapa

Para la creación del mapa emplee el software de código abierto y gratuito Tiled, el cual nos permite crear niveles que estén basados en tilesets.

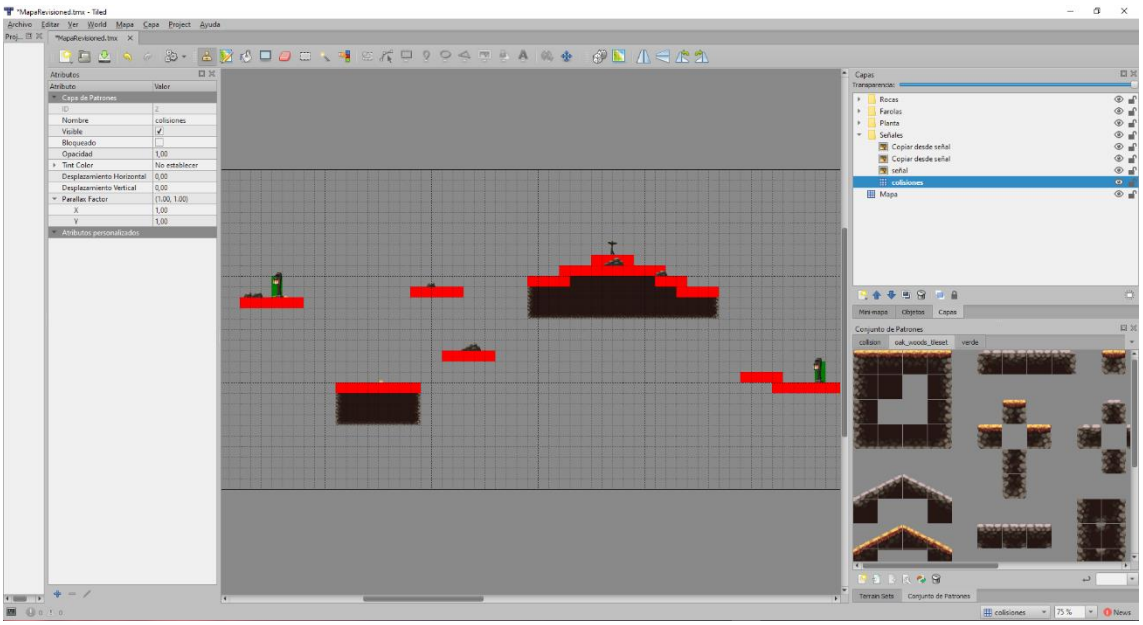


(Proyecto en Tiled del mapa)

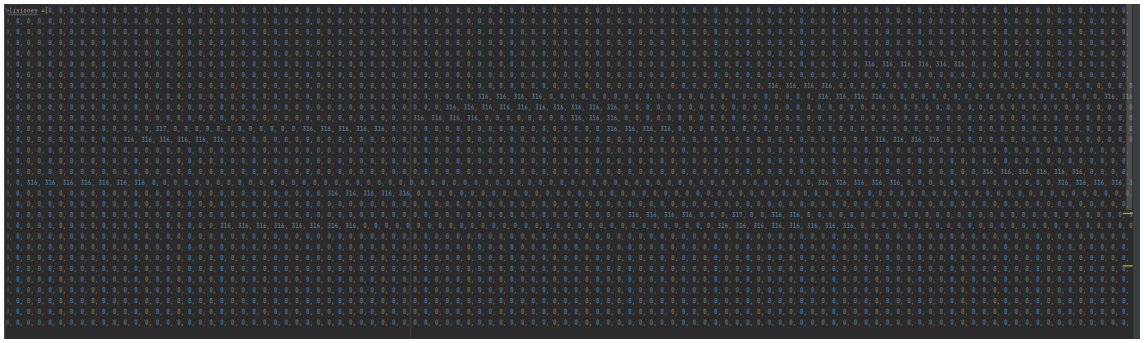
Una vez creado el mapa la propia aplicación nos permite exportar el png el cual agregamos al proyecto del juego cargando la imagen.

```
//cargamos el tilemap
const tilemap = new Sprite( {ruta, posicion, escala, framesHorizontales, framesMaxAnimacion, framesVerticales, posicionVertical, margenSprite}: {
    ruta: "/imagenes/mapa/MapaRevisioned.png",
    posicion: {
        x: 0,
        y: 0
    }
})
```

Una vez cargada la imagen tenemos que generar las colisiones y las hitboxes. La forma de hacerlo es generar otro tilemap dentro de Tiled con un tile que queremos el cual va a hacer las funciones de hitbox



Una vez generado el software nos permite exportar un archivo JSON (JavaScript Object Notation) el cual guarda un array con las coordenadas y un valor numero para cada tipo de tile de nuestro proyecto en Tiled. De esta forma solo tenemos que buscar el array que representa la capa de colisiones y guardarlo dentro de nuestro proyecto de JavaScript.



(Se puede ver como los numero más o menos componen en el mapa que cargamos)

Una vez tenemos el array con las posiciones hay que renderizarlo en la pantalla. Para ello hacemos uso del objeto Plataforma. Pero hay paso previo, este consiste en dividir el array por alturas. Nuestro proyecto consiste en una red de 180 de ancho por 30 de alto. Para dividirlo hago uso de un bucle for que va a dividir el array de las colisiones en otros arrays de 180 elementos. De esta forma podemos renderizar el mapa además de generar un tipo de plataforma dependiendo del numero que tenga.

```
const mapaColisiones = []
for (let i = 0; i < colisiones.length; i += 180) { //180 es el ancho del tilemap
    mapaColisiones.push(colisiones.slice(i, i + 180))
}
mapaColisiones.forEach((fila, i : number ) => {
    fila.forEach((valor, j) => {

        //el valor 316 representa la hitbox de las plataformas
        if (valor == 316) {
            plataformas.push(new Plataforma( {posicion, encendida, numeroDeHoguera}: {
                posicion: {
                    x: j * Plataforma.ancho,
                    y: i * Plataforma.alto
                }
            }))
        }

        //el valor 317 representa las posiciones de las hogueras
        if (valor == 317){
            checkpoints.push(new Plataforma( {posicion, encendida, numeroDeHoguera}: {
                posicion: {
                    x: j * Plataforma.ancho,
                    y: i * Plataforma.alto
                },
                encendida: false,
                numeroDeHoguera: contador
            }))
            posicionHogueras.push({
                x: j * Plataforma.ancho,
                y: i * Plataforma.alto -10
            })
            contador++
        }

        //es la posicion del final del juego
        if (valor == 318) {
            metas.push(new Plataforma( {posicion, encendida, numeroDeHoguera}: {
                posicion: {
                    x: j * Plataforma.ancho,
                    y: i * Plataforma.alto
                }
            }))
        }
    })
})
```

La forma que tenemos de movernos por el mapa es mediante las teclas “AWSD”. Siempre que el jugador este entre x >= 400 y x<=800 lo que se va a mover es el propio Sprite del jugador.

```
if (teclas.izquierda.pulsada && jugador.posicion.x > 400 && !jugador.haColisionado) {

    if (teclas.arriba.pulsada) jugador.setSprite(sprites.saltarIzquierda)
    if (jugador.sobrePlataforma) jugador.setSprite(sprites.correrIzquierda)
    jugador.moverIzquierda()

} else if (teclas.derecha.pulsada && jugador.posicion.x < 800 && !jugador.haColisionado) {

    if (teclas.arriba.pulsada) jugador.setSprite(sprites.saltarDerecha)
    if (jugador.sobrePlataforma) jugador.setSprite(sprites.correrDerecha)
    jugador.moverDerecha()
```

Una vez lleguemos a estos limites lo que se va a mover es el escenario dando la falsa sensación de movimiento.

```
if (teclas.arriba.pulsada) jugador.setSprite(sprites.saltarDerecha)
if (jugador.sobrePlataforma) jugador.setSprite(sprites.quieto)
jugador.parar()

if (teclas.derecha.pulsada && !jugador.haColisionado) {

    if (teclas.arriba.pulsada) jugador.setSprite(sprites.saltarDerecha)
    if (jugador.sobrePlataforma) jugador.setSprite(sprites.correrDerecha)
    tilemap.posicion.x -= velocidadPlataformas
    tienda.posicion.x -= velocidadPlataformas
    plataformas.forEach(plataforma => {
        plataforma.posicion.x -= velocidadPlataformas
    })
    metas.forEach(meta => {
        meta.posicion.x -= velocidadPlataformas
    })
    checkpoints.forEach(cpoint => {
        cpoint.posicion.x -= velocidadPlataformas
    })
    magos.forEach(mago => {
        mago.posicion.x -= velocidadPlataformas
    })
}
if (teclas.izquierda.pulsada && !jugador.haColisionado) {

    if (teclas.arriba.pulsada) jugador.setSprite(sprites.saltarIzquierda)
    if (jugador.sobrePlataforma) jugador.setSprite(sprites.correrIzquierda)
    tilemap.posicion.x += velocidadPlataformas
    tienda.posicion.x += velocidadPlataformas
    plataformas.forEach(plataforma => {
        plataforma.posicion.x += velocidadPlataformas
    })
    checkpoints.forEach(cpoint => {
        cpoint.posicion.x += velocidadPlataformas
    })
    metas.forEach(meta => {
        meta.posicion.x += velocidadPlataformas
    })
    magos.forEach(mago => {
        mago.posicion.x += velocidadPlataformas
    })
```

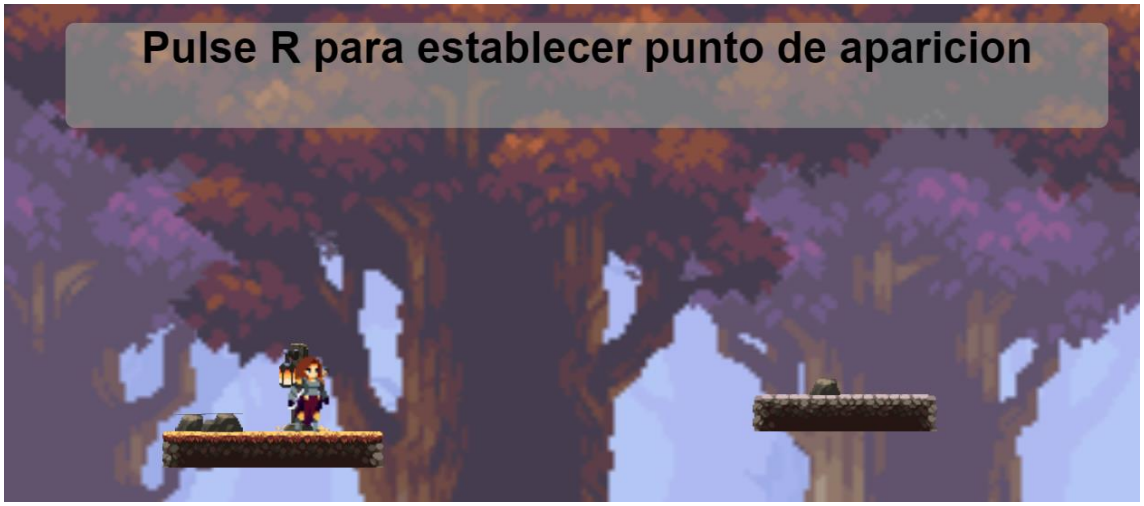
Aquí podemos observar como una vez estamos en los limites de 400 y 800 pixeles dependiendo de que botón pulsamos se mueven las plataformas en una dirección u otra. Con este movimiento relativo del escenario no solo se mueve la imagen, también se mueven las hiboxes, los enemigos, las lámparas y la tienda del final, así como el arco del final del juego.

Lampara

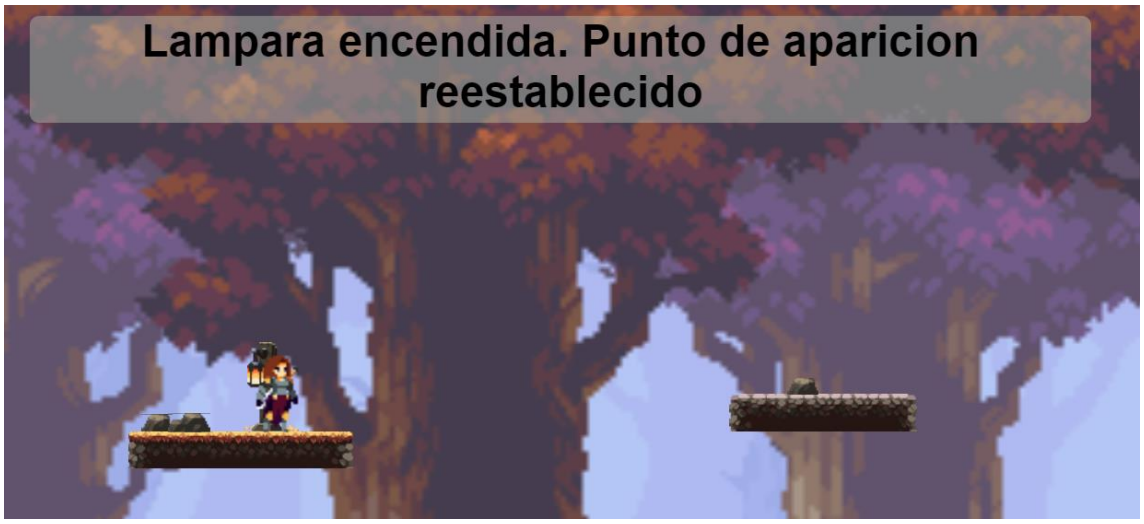
El funcionamiento de las lámparas es el siguiente. Cuando creamos las lámparas se guardan sus coordenadas dentro de un array el cual nos va a permitir después colocar al jugador ahí si decide reiniciar el nivel o si muere.

```
checkpoints.push(new Plataforma( {posicion, encendida, numeroDeHoguera}: {
    posicion: {
        x: j * Plataforma.ancho,
        y: i * Plataforma.alto
    },
    encendida: false,
    numeroDeHoguera: contador
}))
posicionHogueras.push({
    x: j * Plataforma.ancho,
    y: i * Plataforma.alto -10
})
contador++
```

Una vez estemos dentro de su hitbox nos aparecerá en pantalla un mensaje para poder establecer el punto de aparición



Una vez pulsamos R el mensaje cambia y ahora una vez muramos volveremos a aparecer en esta lampara.



Este es el fragmento que se encarga de controlar cuando aparece el mensaje, cuando se establece un nuevo punto de aparición y además cambia el DIV del HTML que muestra el mensaje. Adicionalmente he añadido la mecánica que hace que solo puedas tener una lampara activa a la vez. Es decir, si tu activas la 2da lampara la 1ra y la 3ra se apagarán.

```
//funciones relacionadas con las lamparas
checkpoints.forEach(cpoint => {

    cpoint.render()

    //muestra el texto para establecer el punto de aparicion
    if ((jugador.derecha >= cpoint.posicion.x
        && jugador.posicion.x <= cpoint.posicion.x + cpoint.anchoPlataforma)) {

        if (cpoint.encendida) {
            hogueraEncendida.classList.add("display")
        } else hogueraApagada.classList.add("display")


        addEventListener( type: "keydown",  listener: ({key : string }) => {
            if (key === "r") {

                console.log(jugador.derecha + " || " + cpoint.posicion.x)

                if ((jugador.derecha >= cpoint.posicion.x
                    && jugador.posicion.x <= cpoint.posicion.x + cpoint.anchoPlataforma)) {
                    hogueraEncendida.classList.add("display")
                    hogueraApagada.classList.remove( tokens: "display")
                    cpoint.encendida = true
                    checkpoints.forEach(points => {
                        if (points != cpoint) {
                            points.encendida = false
                        }
                    })
                }

            }

        })

    } else if (jugador.derecha >= cpoint.posicion.x + cpoint.anchoPlataforma
        && jugador.posicion.x >= cpoint.posicion.x) {
        hogueraApagada.classList.remove( tokens: "display")
        hogueraEncendida.classList.remove( tokens: "display")
    }
```

Jugador

El jugador es una instancia de la clase Jugador. Este se puede mover por el mapa mediante las teclas “WASD”. El jugador cuenta con la particularidad de que tiene gravedad y siempre y cuando no se encuentre encima de una plataforma este va a caer.

```
if ((this.base + this.velocidad.y <= canvas.height)) {
    this.velocidad.y += gravedad
} else this.velocidad.y = 0
```

(Fragmento dentro de la clase Jugador que hace que este caiga)

Una vez está encima de la plataforma su velocidad en el eje de las Y se pone a 0. Esto se controla de forma individual para cada plataforma.

```
if (jugador.base <= plataforma.top
    && jugador.base + jugador.velocidad.y >= plataforma.top
    && jugador.derecha >= plataforma.posicion.x
    && jugador.posicion.x <= plataforma.posicion.x + plataforma.anchoPlataforma
) {
    jugador.sobrePlataforma = true
    jugador.haColisionado = false
    jugador.velocidad.y = 0
}
```

(Fragmento que se encarga de hacer que no se caiga una vez está encima de la plataforma)

El jugador también cuenta con una animación de ataque la cual se activa con la “Q” una vez pulsamos esta tecla el jugador no podrá moverse y además infligirá daño a los enemigos que se encuentren dentro de su hitbox de ataque.

```
} else if (teclas.q.pulsada) {
    jugador.parar()
    jugador.setSprite(sprites.atacar)

} else {
```

Si la animación es de ataque se pondrán a “true” los valores esta atacando el cual hace que si se encuentra en los frames 6, 7, 10 y 11 de la animación de ataque este cuente como que hace daño.

```
actualizarFrames() {
    this.framesTranscurridos++

    if (this.framesTranscurridos % this.framesEspera === 0) {

        if (this.frameActual < this.framesMaxAnimacion - 1) {
            this.imagen.src = this.ruta + this.frameActual + ".png"

            //la ejecucion solo entra aqui cuando ataca. esta condicion esta para saber cuando es que el jugador hace daño
            if (this.estaAtacando && (this.frameActual == 6 || this.frameActual == 7 || this.frameActual == 10 || this.frameActual == 11)) {
                this.haceDano = true
            } else this.haceDano = false

            this.frameActual++

        } else {
            //si el jugador ha muerto no se resetea el contador si no que muestra el ultimo frame de la animacion de muerte
            if (this.animacionMuerte) this.imagen.src = this.ruta + 11 + ".png"
            else this.frameActual = 1

        }
    }
}
```

(Fragmento que se encarga de controlar si está en los frames de ataque y de decir si estos hacen daño o no)

Magos

Los magos son los enemigos del juego. Lo mas remarcable del mago es que es capaz de recibir daño. Una vez han recibido un daño mayor que sus puntos de vida desaparece.

Cuando pulsamos la “Q” y la hitbox del arma del jugador esta dentro de la del mago es cuando se le quita vida al mago.

```
case "q":
    teclas.q.pulsada = true
    magos.forEach(mago => {

        if (
            jugador.haceDano &&
            jugador.hitboxAtaque.posicion.x + jugador.hitboxAtaque.ancho >= mago.posicion.x + 160
            && jugador.hitboxAtaque.posicion.x <= mago.posicion.x + mago.ancho
            && jugador.hitboxAtaque.posicion.y >= mago.posicion.y
            && jugador.hitboxAtaque.posicion.y >= mago.base
        ) {
            mago.vida = mago.vida - 1
        }

    })

    break
```

Pantalla principal y menú.

La pantalla principal es lo primero que vemos una vez ejecutamos el juego. En ella podemos ver que tenemos 2 opciones. Jugar ahora o ver los controles.

Cuando hacemos click a Jugar ahora lo que hace el programa es ocultar el DIV que se encarga de mostrar el menú principal y además muestra el canvas que compone el juego.

```
//evento que nos permite jugar al hacer click en jugar ahora en el menu principal
jugar.addEventListener( type: "click",  listener: () => {
    canvas.style.display = "block"
    intro.style.display = "none"
    menuDisponible = true
})
```

Al hacer click en controles se despliega un pop up con las imágenes y los controles.

```
//evento que muestra el popup con los controles
controles.addEventListener( type: "click",  listener: ()=>{

    menuControles.style.display = "block"
    menuControles.style.width = "30%"
    menuControles.style.height = "40vh"

})
```

Una vez estamos dentro del juego, al pulsar “Esc” se despliega otro menú popup con los controles y con la posibilidad de reiniciar nivel.


```
//muestra el menu al darle al escape
addEventListener( type: "keydown",  listener: ({key : string }) => {

    if (key == "Escape") {
        if (menuDisponible) {
            menuEsc.style.display = "block"
            menuEsc.style.width = "20%"
            menuEsc.style.height = "50vh"
            menuDisponible = false
        } else {
            menuEsc.style.display = "none"
            menuDisponible = true
        }
    }
```

Es ahora cuando empezamos una de las partes más enrevesadas del proyecto.

Hay dos tipos de reinicio en el juego, el reinicio inmediato el cual hace que no pases por la pantalla principal y reapareces en la ultima lampara y el reinicio completo.

El reinicio inmediato o parcial se emplea cuando hacemos click en “Reiniciar nivel” en el menú o al morir. El reinicio completo se emplea cuando hacemos F5 o cuando llegamos al final del juego.

Al hacer click en reiniciar nivel llamamos a la función reiniciarMapa el cual comprueba si hay puntos de reaparición disponibles. Si es así, se almacena en una variable local llamada “respawn” el número asignado a la lampara. Esto significa que cuando carguemos otra vez el juego este va a saber en qué lampara ponernos. Una vez asignada la lampara de reaparición hacemos una llamada a la función temporizador.

Temporizador es una promesa que se encarga de establecer una variable local la cual determina si se muestra el menú principal y cuanto tiempo hay que esperar.

```
//reiniciar el nivel desde el menu
reiniciar.addEventListener( type: "click",  listener: ({key}) => {

    reiniciarMapa()

})

function reiniciarMapa(){

    let hogueraEncendida

    if (finalizado){
        checkpoints.forEach(cpoint =>{
            if (cpoint.encendida){
                hogueraEncendida = cpoint.numeroDeHoguera
            }
        })
    }

    cargar = true
    if (hogueraEncendida != null)localStorage.setItem("respawn", hogueraEncendida)

    if (reiniciando)
    {
        hasMuerto.style.display = "block"
        hasMuerto.classList.add("animacionMuerte")
        temporizador()
    }else temporizador( tiempo: 0)


}
```

```
//esta promesa nos permite simular un Thread sleep de Java
function sleep(ms) {
    return new Promise( executor: resolve => setTimeout(resolve, ms));
}

async function temporizador(tiempo :number = 4) {
    for (let i = 0; i < tiempo; i++) {
        await sleep( ms: i * 1000);
    }

    localStorage.setItem("cargarMenu", false)
    location.reload()
}
```

En caso de que haya una lampara en la que poder reaparecer este fragmento se encarga de desplazar todos los elementos del mapa a la coordenada necesaria.

```
//en caso de haber muerto esto nos permite cargar el mapa desde la ultima lampara con el mapa desplazado
if (localStorage.getItem( key: "respawn")) {

    tilemap.posicion.x -= posicionHogueras[localStorage.getItem( key: "respawn")].x - jugador.posicion.x
    tienda.posicion.x -= posicionHogueras[localStorage.getItem( key: "respawn")].x - jugador.posicion.x
    plataformas.forEach(plat => {
        plat.posicion.x -= posicionHogueras[localStorage.getItem( key: "respawn")].x - jugador.posicion.x
    })
    checkpoints.forEach(cpoint => {
        cpoint.posicion.x -= posicionHogueras[localStorage.getItem( key: "respawn")].x - jugador.posicion.x
    })
    magos.forEach(mago =>{
        mago.posicion.x -= posicionHogueras[localStorage.getItem( key: "respawn")].x - jugador.posicion.x
    })
}
```

La otra forma de poder hacer un reinicio parcial es si el jugador muere. Esto pasa cuando el jugador toca el borde inferior de la pantalla. Cuando esto pasa aparece un mensaje de muerte, ocurre una animación de muerte en el jugador y el juego vuelve al ultimo punto de guardado. En caso de no haber ninguna lampara encendida te devuelve al principio del juego.


```
//reiniciar el mapa y animacion de muerte
if (jugador.base >= suelo - 2 && !reiniciando) {
    teclas.m.pulsada = true
    jugador.parar()
    jugador.animacionMuerte = true
    reiniciando = true
    reiniciarMapa()
}
```

Una vez tocamos el suelo hacemos que ocurra la animación de muerte, que el jugador se pare y ponemos le valor de reiniciando a true y hacemos otra llamada a reiniciarMapa. Lo que hace el valor de “reiniciando” a “true” es que aparezca o no el mensaje de muerte. Por eso cuando reiniciamos el nivel desde el menú no sale.

En resumen el reinicio rápido se consigue gracias a que guardamos en una variable local si queremos que se muestre el menu o no.

```
const localStorage = window.localStorage

let cargarMenu = localStorage.getItem( key: "cargarMenu")

if (cargarMenu) {
    canvas.style.display = "block"
    intro.style.display = "none"
    menuDisponible = true
} else {
    canvas.style.display = "none"
    menuDisponible = false
}
```

En ultimo lugar tenemos el reinicio completo que se da cuando pulsamos F5 o al llegar al final del mapa. Este reinicio nos lleva la pantalla principal.

Para detectar si el jugador empleamos un nuevo tipo de plataforma con otro valor. Cuando el jugador entra dentro de su hitbox cambiamos el mensaje de muerte por uno de victoria y llamamos a la función de victoria

```
//reiniciar el mapa y animacion de victoria
if (dentrofinal) {
    hasMuerto.innerHTML = "<p>Victoria</p>"
    teclas.m.pulsada = true
    jugador.parar()
    jugador.animacionMuerte = true
    victoria()
}
```

```
function victoria(){
    hasMuerto.style.display = "block"
    hasMuerto.classList.add("animacionMuerte")
    temporizador()
}
```