Ampliaciones

**1. Nuevos tipos de enemigos**

He aprovechado la clase Enemigo.js y he creado dos nuevas clases: EnemigoSencillo.js y EnemigoMejorado.js

EnemigoSencillo.js es el enemigo básico que ya existía

EnemigoMejorado.js es el enemigo nuevo que tiene distinta velocidad, tamaño, imagen y, además, es capaz de disparar.

```
class EnemigoMejorado extends Enemigo {

    constructor(x, y){
        super(imagenes.enemigoMejorado, imagenes.enemigoMejorado_movimiento,
imagenes.enemigoMejorado_muriendo, x, y, -1.5);
        this.tiempoDisparo = 120;
        this.orientacion = orientaciones.derecha;
    }

    actualizar() {
        super.actualizar();

        // Establecer orientación
        if ( this.vx > 0 ){
            this.orientacion = orientaciones.izquierda;
        }
        if ( this.vx < 0 ){
            this.orientacion = orientaciones.derecha;
        }

        this.tiempoDisparo--;
    }

    disparar(){

        if ( this.tiempoDisparo == 0 && this.estado != estados.muriendo) {
            // reiniciar Cadencia
            this.tiempoDisparo = 120;
            var disparo = new DisparoEnemigo(this.x, this.y, this.vx);
            if ( this.orientacion == orientaciones.izquierda ){
                disparo.vx = disparo.vx*-1; //invertir
            }
            return disparo;
        } else {
            return null;
        }

    }
}
```

En GameLayer.js he incluido al método que carga el mapa un caso con el nuevo enemigo y he cambiado el anterior caso de enemigo para que añada enemigos sencillos.

Todo lo que antes referenciaba a Enemigo.js ahora referencia a EnemigoSencillo.js y a EnemigoMejorado.js

Se actualiza todo para enemigos sencillos y también para enemigos mejorados (por separado) y lo mismo a la hora de dibujar.

Además he añadido una nueva lista con los enemigos mejorados.

**2. Enemigos que mueren al saltar sobre ellos**

He incluido esta opción solo a los enemigos sencillos. Además cuando el jugador salta sobre el enemigo y matarlo, salta otra vez.

```
for (var i=0; i < this.enemigosSencillos.length; i++){
    if ( this.jugador.colisiona(this.enemigosSencillos[i])){
        if(this.jugador.y < this.enemigosSencillos[i].y){
            this.espacio.eliminarCuerpoDinamico(this.enemigosSencillos[i]);
            this.enemigosSencillos[i].impactado();
            this.enemigosSencillos.splice(i, 1);
            i = i - 1;
            this.puntos.valor++;
            this.jugador.enElAire = false;
            this.jugador.saltar();
        } else {
            this.iniciar();
        }
    }
}
```

**5. Scroll en el eje Y**

Para realizar esta ampliación simplemente he tenido que añadir una nueva variable en el método dibujar de modelo, jugador y enemigo. Esta variable es tratada en el método de las clases mencionadas de igual manera que scrollX pero aplicado al eje Y. Además he tenido que añadir dos nuevos casos en calcular scroll dentro del game layer para que este se modifique también en función del eje Y.

**10. Tiles destruibles 2**

He incluido un nuevo tipo de bloque (BloqueDestruible.js) que hereda los métodos de la clase Bloque.js

Cuando se carga el mapa ahora se tiene en cuenta un nuevo caso, cuando un tile es destruible y lo pinta con ladrillos.

En Bloque.js he incluido un nuevo atributo que indica si se puede destruir el bloque. En la clase BloqueDestruible.js, al crear el objeto, se pone dicho atributo a true.

Y en el método actualizar si hay una colisión del disparo del jugador con el tile destruible, este se borra por completo.

```
for(var i=0; i < this.disparosJugador.length; i++){
    for(var j=0; j < this.bloques.length; j++){
        if(this.bloques[j].destruible){
            if(this.disparosJugador[i].colisiona(this.bloques[j])){
                this.disparosJugador.splice(i, 1);
                i = i - 1;
                this.espacio.eliminarCuerpoEstatico(this.bloques[j]);
                this.bloques.splice(j, 1);
                j = j - 1;
```

```
            }
        }
    }
}
```

**12. Plataformas de salto**

Al igual que con los bloques destruibles, he creado otra clase (BloqueSalto.js) y he añadido otra variable en Bloque.js que indica si es un bloque de salto.

Cuando se crea el bloque, dicha variable se inicializa a true.

En la clase BloqueSalto.js he incluido un método que recibe como parámetro un modelo y, si es posible, lo impulsa hacia arriba.

```
impulsar(objeto){
    if(objeto.vy != null){
        if(objeto.y < this.y){
            objeto.vy = -20;
        }
    }
}
```

En el método actualizar se recorre la lista de bloques y si estos son de salto, comprueba si algún modelo está colisionando, y en caso de colisionar, los impulsa.

```
for(var i=0; i < this.bloques.length; i++){

    if(this.bloques[i].salto){
        if(this.jugador.colisiona(this.bloques[i])){
            this.bloques[i].impulsar(this.jugador);
        }

        for(var j=0; j < this.enemigosSencillos.length; j++){
            if(this.enemigosSencillos[j].colisiona(this.bloques[i])){
                this.bloques[i].impulsar(this.enemigosSencillos[j]);
            }
        }

        for(var j=0; j < this.enemigosMejorados.length; j++){
            if(this.enemigosMejorados[j].colisiona(this.bloques[i])){
                this.bloques[i].impulsar(this.enemigosMejorados[j]);
            }
        }
    }
}
```