INFORME AMPLIACIONES

Juego de naves

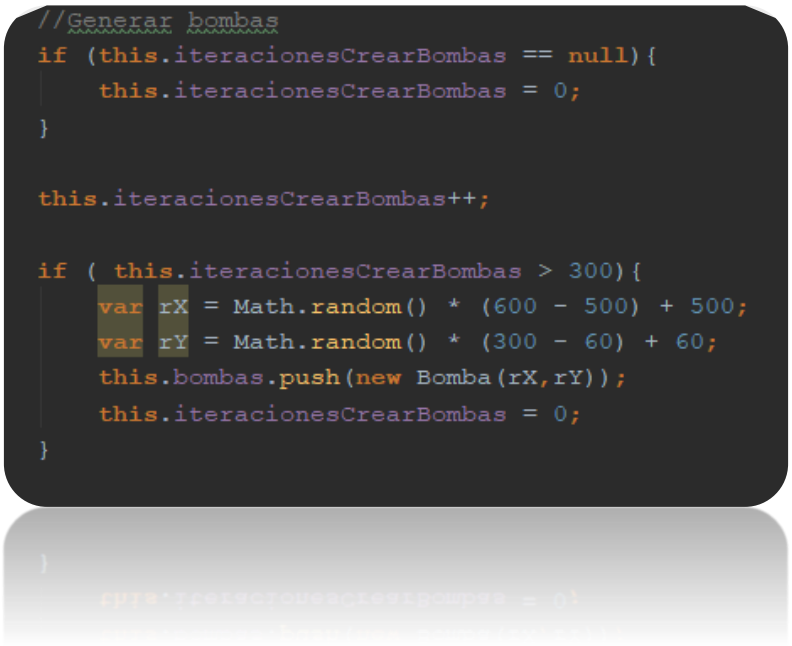
Universidad de Oviedo

Pablo Menéndez Suárez
UO252406

Informe ampliaciones naves

1. **Bombas**

   Para esta ampliación he creado una nueva clase llamada “Bomba” que extiende de la clase “Modelo”. Esta clase tiene el método actualizar que simplemente suma la velocidad de la bomba en negativo para que ésta vaya hacía la izquierda. Para la generación de bombas he creado en la capa “GameLayer” un array de bombas que se va llenando cada 300 iteraciones.

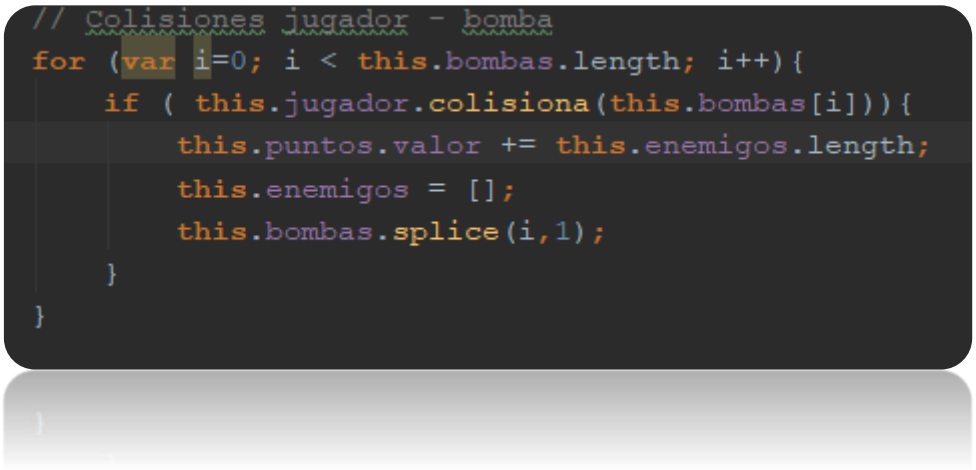


```
//Generar bombas
if (this.iteracionesCrearBombas == null){
    this.iteracionesCrearBombas = 0;
}

this.iteracionesCrearBombas++;

if ( this.iteracionesCrearBombas > 300){
    var rX = Math.random() * (600 - 500) + 500;
    var rY = Math.random() * (300 - 60) + 60;
    this.bombas.push(new Bomba(rX,rY));
    this.iteracionesCrearBombas = 0;
}
```

   Por último, compruebo las colisiones entre el jugador y las bombas. En caso de que haya colisión elimino a todos los enemigos, es decir, creo un nuevo array vacío, y elimino la bomba del array de bombas usando el método splice. Además incremento la puntuación en 1 punto por cada enemigo muerto gracias a la bomba.



```
// Colisiones jugador - bomba
for (var i=0; i < this.bombas.length; i++){
    if ( this.jugador.colisiona(this.bombas[i])){
        this.puntos.valor += this.enemigos.length;
        this.enemigos = [];
        this.bombas.splice(i,1);
    }
}
```

2. **Monedas**

Esta ampliación es muy similar a la anterior. Creo una clase “Moneda” que extiende de la clase “Modelo”. Genero monedas con el mismo algoritmo que las bombas, pero cada 50 iteraciones.

```
//Generar Monedas
if (this.iteracionesCrearMonedas == null){
    this.iteracionesCrearMonedas = 0;
}

this.iteracionesCrearMonedas ++;
if ( this.iteracionesCrearMonedas > 50){
    var rX = Math.random() * (600 - 500) + 500;
    var rY = Math.random() * (300 - 60) + 60;
    this.monedas.push(new Moneda(rX,rY));
    this.iteracionesCrearMonedas = 0;
}
```

En el caso de que el jugador colisione con una moneda, se saca ésta del array de monedas y se incrementa la puntuación en 5 puntos.

```
//Colisiones jugador - moneda
for (var i=0; i < this.monedas.length; i++){
    if ( this.jugador.colisiona(this.monedas[i])){
        this.monedas.splice(i, 1);
        this.puntos.valor += 5;
    }
}
```

3. **Enemigos con capacidad de disparo**

Para esta ampliación he creado una clase “DisparoEnemigo” muy similar a la clase “DisparoJugador” que extiende de la clase “Modelo”. Añadimos un método disparar a

la clase enemigo que devuelva un nuevo disparo siempre que respete el tiempo de cadencia.

```
disparar(){
    if ( this.tiempoDisparoEnemigo == 0) {
        // reiniciar Cadencia
        this.tiempoDisparoEnemigo = this.cadenciaDisparoEnemigo;
        return new DisparoEnemigo(this.x, this.y);
    } else {
        return null;
    }
}
```

Por último, generamos disparos enemigos en el método actualizar de la clase "GameLayer" que genera un disparo por enemigo siempre que éste se encuentre dentro de la pantalla y haya recargado ya su cadencia de disparo.

```
// Generamos disparos para los enemigos
for (var i=0; i < this.enemigos.length; i++){

    var nuevoDisparo = this.enemigos[i].disparar();
    if ( nuevoDisparo != null && this.enemigos[i].estaEnPantalla()) {
        this.disparosEnemigo.push(nuevoDisparo);
    }
}
```

4. **Jugador con Vida**

Para esta ampliación he creado una variable vidas dentro de la clase "Jugador" que se va decrementando cada vez que hay una colisión del tipo "Enemigo con jugado" o "disparoEnemigo – jugador".

Por otro lado, hemos definido tres variables nuevas de tipo "Fondo" en el método "Iniciar" de la clase "GameLayer" con una imagen de un corazón que representarán las vidas del jugador.

```
this.fondoPrimeraVida =
    new Fondo(imagenes.vida, 480*0.07,320*0.06);

this.fondoSegundaVida =
    new Fondo(imagenes.vida, 480*0.17,320*0.06);

this.fondoTerceraVida =
    new Fondo(imagenes.vida, 480*0.27,320*0.06);
```

Cuando se produce una de las colisiones anteriormente nombradas, aparte de decrementar la variable vidas, desaparece uno de los corazones. Todo esto se hace llamando al método “decrementarVidas()” de la clase “GameLayer”.

```
decrementaVidas(){

    this.jugador.vidas--;

    switch (this.jugador.vidas){

        case 2:
            this.fondoTerceraVida = null;
            break;

        case 1:
            this.fondoSegundaVida = null;
            break;


    }


}
```

Hay que tener en cuenta que con este método que acabamos de ver cada vez que se llame al método dibujar habrá que comprobar si los fondos son distintos de null.

```
if(this.fondoPrimeraVida != null)
    this.fondoPrimeraVida.dibujar();
if(this.fondoSegundaVida != null)
    this.fondoSegundaVida.dibujar();
if(this.fondoTerceraVida != null)
    this.fondoTerceraVida.dibujar();
```

Por último llamamos comprobamos en el método “actualizar” de la clase “GameLayer” si el número de vidas es <= 0 (debería bastar con == 0) . Si se cumple la condición se reinicia el juego.

```
//Comprobamos si el jugador se ha quedado sin vidas
if(this.jugador.vidas<=0)
    this.iniciar();
```

Para esta ampliación se han limitado el margen superior de la pantalla tanto en la creación de enemigos, bombas y monedas como en el movimiento del jugador para que el “HUD” no interfiera con el juego.

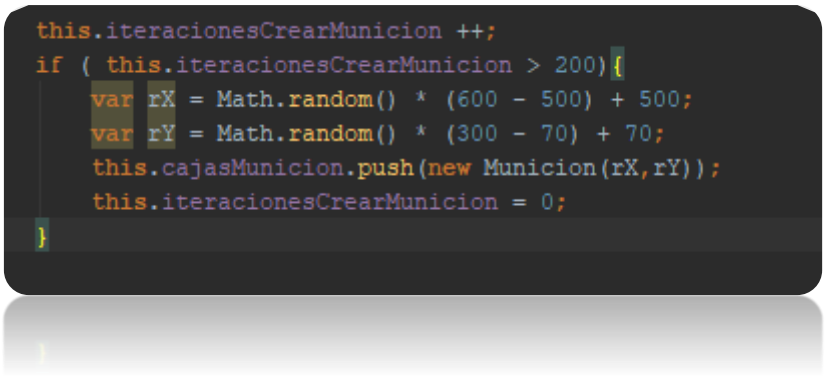
5. **Jugador con disparos finitos**

Lo primero que hacemos es crear una variable dentro de la clase “Jugador” llamada “numDisparos” inicializada a 30, que va a ser el número de disparos iniciales. A continuación, modificamos el método disparar de la clase “Jugador” para que solo dispare si “numDisparos” es mayor que 0, y para que cada vez que dispare decremente “numDisparos”.

```
disparar(){
    if ( this.tiempoDisparo == 0 && this.numDisparos > 0) {
        // reiniciar Cadencia
        this.numDisparos--;
        this.tiempoDisparo = this.cadenciaDisparo;
        return new DisparoJugador(this.x, this.y);
    } else {
        return null;
    }
}
```

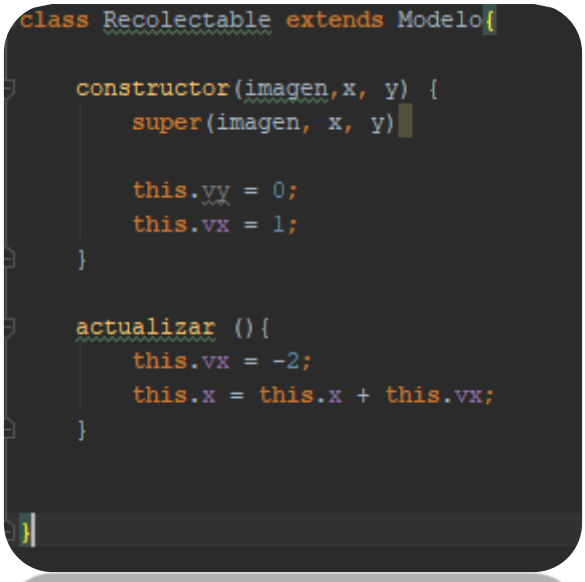
Luego creamos la clase “Munición” que extiende de modelo y es muy similar a la clase “Moneda”. Simplemente hace que la munición vaya hacia la izquierda. Declaramos la clase “Municion.js” dentro del “index.html” y creamos un array en la clase “GameLayer” llamado “cajasMunicion”. La creación de las cajas de munición sigue el

algoritmo anteriormente visto para las monedas o las bombas. Además pintamos y actualizamos todos los elementos dentro de “cajasMunicion”.



```
this.iteracionesCrearMunicion ++;
if ( this.iteracionesCrearMunicion > 200){
    var rX = Math.random() * (600 - 500) + 500;
    var rY = Math.random() * (300 - 70) + 70;
    this.cajasMunicion.push(new Municion(rX,rY));
    this.iteracionesCrearMunicion = 0;
}
```

En este punto me he dado cuenta que como tenemos tres clases con el mismo comportamiento que son “Bomba”, “Moneda” y “Munición”, he creado una nueva clase llamada “Recolectable” que extiende de modelo y de la que extenderán las clases anteriormente nombradas.



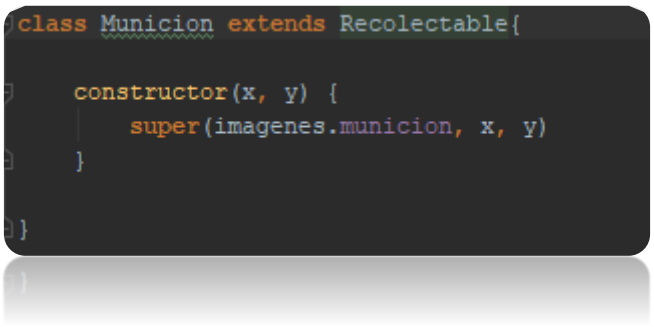
```
class Recolectable extends Modelo{

    constructor(imagen,x, y) {
        super(imagen, x, y)

        this.yy = 0;
        this.vx = 1;
    }

    actualizar (){
        this.vx = -2;
        this.x = this.x + this.vx;
    }


}
```

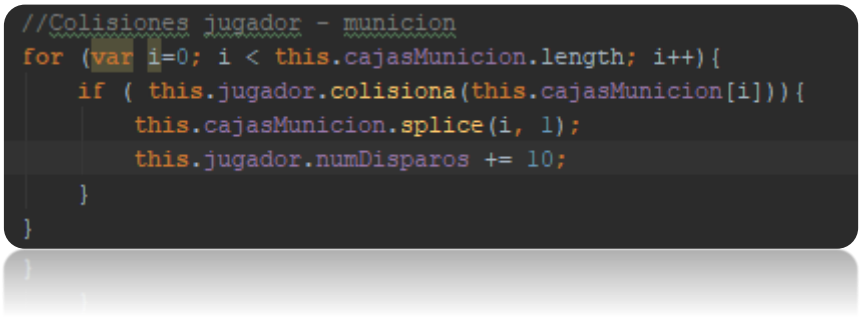


```
class Municion extends Recolectable{

    constructor(x, y) {
        super(imagenes.municion, x, y)
    }

}
```

Por último, miramos las colisiones “Jugador-Munición” y en caso de que se produzca una de estas, aumentamos la variable “numDisparos” del jugador en 10 unidades.



Como complemento he creado un contador de balas en el “HUD” que sigue la misma implementación que el contador de puntos. Cuando la nave dispara se decrementa y cuando se colisiona con una caja de munición se incrementa en 10.