Ampliaciones Plataformas

Items Recolectables (ampliación 6)

Creamos la clase Tile. Localizamos los recolectables del mapa como una “R” y los metemos en un array de recolectables.

```
class Tile extends Modelo {

    constructor(rutaImagen, x, y,animacion, velocidadRefresco, framesTotales) {
        super(rutaImagen, x, y)

        this.animacion = new Animacion(animacion,
            this.ancho, this.alto,  velocidadRefresco, framesTotales);
    }


    actualizar () {
        // Actualizar animación
        this.animacion.actualizar();
    }

    dibujar (scrollX){
        scrollX = scrollX || 0;
        this.animacion.dibujar(this.x - scrollX, this.y);
    }
}
```

```
case "R":
    var rec = new Tile(imagenes.icono_recolectable, x,y,imagenes.recolectable,1,8);
    rec.y = rec.y - rec.alto/2;
    // modificación para empezar a contar desde el suelo
    this.recolectables.push(rec);
    this.espacio.agregarCuerpoDinamico(rec);
    break;
```

Actualizamos y dibujamos todos los recolectables y detectamos las colisiones.

```
// colision , Jugador - Recolectable
for (var i=0; i < this.recolectables.length; i++) {
    if (this.jugador.colisiona(this.recolectables[i])) {
        this.recolectables.splice(i, 1);
        this.puntosRecolectables.valor++;
    }
}
```

Si se colisiona con uno de ellos aumentamos la puntuación en el HUD.

```
this.fondoRecolectables =
    new Fondo(imagenes.icono_recolectable, 480*0.73,320*0.07);

this.puntosRecolectables = new Texto(0,480*0.79,320*0.07 );
```

Punto de salvado (ampliación 14)

Localizamos el punto de salvado(checkPoint) en el mapa como una “A”.

```
case "A":
    this.checkPoint = new Bloque(imagenes.check, x,y);
    this.checkPoint.y = this.checkPoint.y - this.checkPoint.alto/2;
    this.espacio.agregarCuerpoDinamico(this.checkPoint);
    break;
```

Actualizamos el checkPoint y lo dibujamos como siempre. Detectamos la colisión entre el jugador y el checkPoint. En caso de que el jugador pase por el checkPoint cambiamos su imagen y ponemos una variable a true que en caso de que el jugador muera nos indicará que debe empezar donde el checkPoint.

```
//  Jugador - CheckPoint
if ( this.checkPoint.colisiona(this.jugador)){
    this.checkPoint.imagen.src = imagenes.checkPassed;
    this.salvar = true;
}
```

En el método iniciar:

```
this.checkPoint.imagen.src = imagenes.check;

if(this.salvar) {
    this.jugador.x = this.checkPoint.x;
    this.checkPoint.imagen.src = imagenes.checkPassed;
}
```

Si pasamos de nivel volvemos a poner la variable a false.

```
if ( this.copa.colisiona(this.jugador)){
    nivelActual++;
    if (nivelActual > nivelMaximo){
        nivelActual = 0;
    }
    this.pausa = true;
    this.mensaje =
        new Boton(imagenes.mensaje_ganar, 480/2, 320/2);
    this.salvar = false;
    this.iniciar();
}
```

Nuevos tipos de enemigos (ampliación 1) y enemigos que mueren al saltar sobre ellos (ampliación 2)

Creo una clase Enemigo a modo de interfaz:

```
class Enemigo extends Modelo{

    finAnimacionMorir(){}

    finAnimacionDisparar(){}

    impactado(){}

    disparar(){}
}
```

Creo una nueva clase Arquero. Localizo en el mapa a estos arqueros con la letra “S” y los meto en una lista de enemigos especiales.

```
case "S":
    var enemigo = new Arquero(x,y);
    enemigo.y = enemigo.y - enemigo.alto/2;
    // modificación para empezar a contar desde el suelo
    this.enemigosEspeciales.push(enemigo);
    this.espacio.agregarCuerpoDinamico(enemigo);
    break;
```

Actualizo y dibujo como siempre a estos enemigos.

Los eliminamos si están muertos:

```
for (var j=0; j < this.enemigosEspeciales.length; j++) {
    if (this.enemigosEspeciales[j] != null &&
        this.enemigosEspeciales[j].estado == estados.muerto) {

        this.espacio
            .eliminarCuerpoDinamico(this.enemigosEspeciales[j]);
        this.enemigosEspeciales.splice(j, 1);

    }
}
```

Detectamos las colisiones de estos enemigos:

```
// colisiones especiales
for (var i=0; i < this.enemigosEspeciales.length; i++){
    if ( this.jugador.colisiona(this.enemigosEspeciales[i])){

        if(this.jugador.colisionaEncima(this.enemigosEspeciales[i])
            && this.jugador.vy > 0 && this.enemigosEspeciales[i].estado != estados.muriendo
            && this.enemigosEspeciales[i].estado != estados.muerto){
            this.enemigosEspeciales[i].impactado();
            this.puntos.valor++;
        }else{

            if(this.enemigosEspeciales[i].estado != estados.muriendo
                && this.enemigosEspeciales[i].estado != estados.muerto)
                this.jugador.golpeado();
                if (this.jugador.vidas <= 0){
                    this.iniciar();
                }
        }
    }

}
```

Creamos una clase DisparoEnemigo. Generamos disparos y detectamos sus colisiones:

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

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

```
// Eliminar disparos enemigos fuera de pantalla
for (var i=0; i < this.disparosEnemigo.length; i++){
    if ( this.disparosEnemigo[i] != null &&
        !this.disparosEnemigo[i].estaEnPantalla() ||
        this.disparosEnemigo[i].vx == 0){
        this.espacio
            .eliminarCuerpoDinamico(this.disparosEnemigo[i]);
        this.disparosEnemigo.splice(i, 1);
    }
}

// Eliminar disparos que chocan
for (var i=0; i < this.disparosEnemigo.length; i++){
    for (var j=0; j < this.bloques.length; j++) {
        if (this.disparosEnemigo[i].colisiona(this.bloques[j])) {
            this.espacio
                .eliminarCuerpoDinamico(this.disparosEnemigo[i]);
            this.disparosEnemigo.splice(i, 1);
        }
    }
}
```

```
// colisiones , disparoEnemigo - Jugador
for (var i=0; i < this.disparosEnemigo.length; i++){
        if (this.disparosEnemigo[i] != null &&
            this.disparosEnemigo[i].colisiona(this.jugador)) {

            this.espacio
                .eliminarCuerpoDinamico(this.disparosEnemigo[i]);
            this.disparosEnemigo.splice(i, 1);
            this.jugador.golpeado();
            if (this.jugador.vidas <= 0){
                this.iniciar();
            }

        }
}
```

No permitimos que el jugador mate a estos enemigos disparándoles:

```
// colisiones , disparoJugador - EnemigoEspecial
for (var i=0; i < this.disparosJugador.length; i++){
    for (var j=0; j < this.enemigosEspeciales.length; j++){

        if (this.disparosJugador[i] != null &&
            this.enemigosEspeciales[j] != null &&
            this.disparosJugador[i].colisiona(this.enemigosEspeciales[j])) {

            this.espacio
                .eliminarCuerpoDinamico(this.disparosJugador[i]);
            this.disparosJugador.splice(i, 1);

        }
    }
}
```

Tiles destruibles (ampliación 9)

Detectamos los tiles con la letra “W” y los metemos en un array llamado boxes.

```
case "W":
    var box = new Tile(imagenes.box_icon, x,y,imagenes.box,1,4);
    box.y = box.y - box.alto/2;
    // modificación para empezar a contar desde el suelo
    this.boxes.push(box);
    this.espacio.agregarCuerpoEstatico(box);
    break;
```

Detectamos las colisiones:

```
// colision , Jugador - Tile
for (var i=0; i < this.boxes.length; i++) {
    if (this.jugador.colisiona(this.boxes[i])) {
        if(this.jugador.colisionaEncima(this.boxes[i]) && this.jugador.vy > 0){
            this.actualTile = i;
            this.contacto = true;
        }
    }
}

if(this.contacto)
    this.contactoTile();
```

Si se produce la colisión guardamos el índice del tile para borrarlo posteriormente y ponemos una variable a true para saber que tenemos que llamar a un método que será el encargado de eliminar dicho tile una vez pasado el tiempo de delay.

```
contactoTile(){
    if(this.delayTile == 30) {
        this.espacio.eliminarCuerpoEstatico(this.boxes[this.actualTile]);
        this.boxes.splice(this.actualTile, 1);
        this.delayTile = 0;
    }else
        this.delayTile++;

}
```