# Memoria de la práctica final:


Grupo 84:
Mario Arias Espinosa
Jorge Rodríguez Fraile

# Índice

# 1. Introducción

En esta memoria se recopilan los principales métodos y algoritmos desarrollados a partir de los conocimientos adquiridos durante el curso con la finalidad de replicar el juego de arcade “Galaga” hasta una fase más o menos jugable.
En nuestro caso, hemos conseguido replicar la formación de enemigos del primer nivel, esta es un enjambre que consta de 40 enemigos (4 capitanes Galaga, 16 Goeis y 20 Zakos) dispuestos en 5 filas de 4, 8, 8, 10 y 10 enemigos respectivamente.
Este enjambre se mueve conjuntamente en horizontal hasta los límites del tablero y al golpear con estos cambia de dirección y a su vez baja una casilla, estilo “Space invaders”.
También hemos creado un jugador que puede moverse horizontalmente en la última fila del tablero y no puede salirse de este, el jugador, al igual que algunos enemigos (Goeis), puede disparar al presionar la tecla “Space”.
Tanto los enemigos como el jugador pueden ser golpeados por los proyectiles contrarios y morir (el jugador también puede morir al ser golpeado directamente por un enemigo), el jugador tiene 3 vidas mientras que los enemigos golpeados por un proyectil del jugador son eliminados para siempre. Al morir cualquier enemigo, los sprites originales desaparecen y se inicia una animación de explosión.

# 2. Clases

Nuestro programa se divide en 6 clases:

- **Principal**

  En esta clase se rellena el tablero, se dan los valores iniciales a las variables de control usadas en el programa, se crea un jugador, se crean los enemigos, se controla su movimiento, rebote, animación de “aleteo” y disparos, se controla el impacto de proyectiles contra jugador y enemigos y su desaparición en los márgenes inferior y superior del tablero, se realizan las animaciones de explosión tanto de los enemigos como del jugador, se actualizan los valores de la interfaz y por último se controla la finalización del juego al perder el jugador todas las vidas o al eliminar todos los enemigos.

- **Elemento**

  En esta clase se almacenan las diferentes características (ID, coordenadas X e Y, sprite, dirección, etc.) que definen a los elementos del tablero (enemigos, jugador y proyectiles). Los elementos de esta clase son privados y se accede a ellos mediante funciones “set” y “get”.

- **Constantes**

  En esta clase se almacenan las diferentes direcciones (16) que pueden tomar los sprites de los enemigos (usados en los giros, los cuales no han sido implementados), también se establecen los límites superiores e inferiores de las dos dimensiones del tablero y la vida máxima del jugador. Todos estos elementos son públicos y final.

- **Jugador**

  Esta clase hereda de la clase “Elemento” y contiene un array de tipo “Bala” con 10 elementos, un objeto de tipo “GameBoardGUI” (usado para actualizar al jugador en el tablero de la clase principal), una variable de tipo “int” para contar el número de balas disparado, un constructor complejo para asignar al jugador una ID, unas coordenadas iniciales y una imagen, una función para asignar al jugador la imagen “player.png”, una función para mover al jugador respetando los bordes del tablero y una función llamada “actualizar” que lee las pulsaciones del teclado y realiza la acción correspondiente a la tecla pulsada, en este caso las únicas teclas útiles son “left”, “right” y “space”, sirviendo las dos primeras para desplazarse a izquierda y derecha respectivamente y la restante para disparar.

- **Enemigo**

  Esta clase hereda de “Elemento” y contiene un elemento de tipo “Bala”, un constructor complejo que asigna a los enemigos un ID, unas coordenadas, una dirección y una vida, tres funciones que asignan la imagen a los tres tipos de enemigos teniendo en cuenta si está girado y por último una función que mueve a los enemigos en una dirección.

- **Bala**

  Esta clase hereda de “Elemento” y contiene un constructor complejo que asigna a la bala una ID, unas coordenadas, una imagen y una dirección, dos funciones que asignan a los objetos de tipo “Bala” las imágenes de los torpedos enemigos y del jugador según corresponda y una función para mover las balas verticalmente.

# 3. Algoritmos

**Creación del enjambre de Enemigos** en “Principal”, líneas 51 a 85

Hemos creado un array de 40 enemigos, como la formación consta de 5 filas de enemigos y cada una tiene un tipo de enemigo diferente, usamos 5 bucles tipo “for” en los que asignamos a cada elemento del array un ID, una dirección, un sprite (dependiendo de la fila cambia entre los tres tipos de enemigos, siendo en la primera fila el enemigo verde, en las dos siguientes el rojo y en las dos últimas el azul) y unas coordenadas y a continuación lo hacemos visible y lo colocamos en la posición del tablero correspondiente a las coordenadas asignadas.

```
50        //Creacion del enjambre de Enemigos
51        Enemigo[] enemigos=new Enemigo[40];
52        for (int i=0; i<4; i++) {
53            Enemigo enemigo1=new Enemigo(100+i, 70+i*10, 5, Constantes.DIR_S);
54            enemigos[i]=enemigo1;
55            gui.gb_addSprite(enemigo1.getId(),enemigo1.getImagenV(), true);
56            gui.gb_setSpriteVisible(enemigo1.getId(), true);
57            gui.gb_moveSpriteCoord(enemigo1.getId(), enemigo1.getCoordx(), enemigo1.getCoordy());
58        }
59        for (int i=4; i<12; i++) {
60            Enemigo enemigo1=new Enemigo(100+i, 50+(i-4)*10, 15, Constantes.DIR_S);
61            enemigos[i]=enemigo1;
62            gui.gb_addSprite(enemigo1.getId(),enemigo1.getImagenR(), true);
63            gui.gb_setSpriteVisible(enemigo1.getId(), true);
64            gui.gb_moveSpriteCoord(enemigo1.getId(), enemigo1.getCoordx(), enemigo1.getCoordy());
65        }for (int i=12; i<20; i++) {
66            Enemigo enemigo1=new Enemigo(100+i, 50+(i-12)*10, 25, Constantes.DIR_S);
67            enemigos[i]=enemigo1;
68            gui.gb_addSprite(enemigo1.getId(),enemigo1.getImagenR(), true);
69            gui.gb_setSpriteVisible(enemigo1.getId(), true);
70            gui.gb_moveSpriteCoord(enemigo1.getId(), enemigo1.getCoordx(), enemigo1.getCoordy());
71        }
72        for (int i=20; i<30; i++) {
73            Enemigo enemigo1=new Enemigo(100+i, 40+(i-20)*10, 35, Constantes.DIR_S);
74            enemigos[i]=enemigo1;
75            gui.gb_addSprite(enemigo1.getId(),enemigo1.getImagenA(), true);
76            gui.gb_setSpriteVisible(enemigo1.getId(), true);
77            gui.gb_moveSpriteCoord(enemigo1.getId(), enemigo1.getCoordx(), enemigo1.getCoordy());
78        }
79        for (int i=30; i<40; i++) {
80            Enemigo enemigo1=new Enemigo(100+i, 40+(i-30)*10, 45, Constantes.DIR_S);
81            enemigos[i]=enemigo1;
82            gui.gb_addSprite(enemigo1.getId(),enemigo1.getImagenA(), true);
83            gui.gb_setSpriteVisible(enemigo1.getId(), true);
84            gui.gb_moveSpriteCoord(enemigo1.getId(), enemigo1.getCoordx(), enemigo1.getCoordy());
85        }
```

**Acciones del jugador recogidas por teclado** en “Jugador” líneas 25 a 66

A esta función se accede a través de la línea de código 93 de la clase “Principal”, la función recibe la última acción introducida por teclado (la cual equivale a una cadena de caracteres e.g. “left”) y accede mediante un switch a las acciones de movimiento lateral con “left” y ”right” y a la acción de disparo con “space”.

Dentro de las acciones de movimiento se utiliza la función “mover”, que además de aumentar o disminuir de 5 en 5 (media cuadrícula) el valor de la coordenada X, controla que no se salgan de los márgenes laterales.

```
19•   public void mover(int x) {
20        if(x>=5 && x<=((Constantes.MAXancho*10)-5) ) {
21        setCoordx(x);
22        }
23    }
```

En la acción de disparo que se activa con el “space”, se controla el número de balas que el jugador puede disparar (un máximo de 10) y en la clase “Principal” las coordenadas de las

balas disparadas se actualizan para que avancen y si se pasan del límite superior desaparecerán.

```
24•    public void actualizar() {
25         accion=guijug.gb_getLastAction().trim();
26         if (accion.length() > 0) {
27             guijug.gb_println(accion);
28         }
29         if(isVivo()) {
30             switch (accion) {
31             //Movimiento
32             case "left":
33                 mover(getCoordx()-5);
34                 guijug.gb_moveSpriteCoord(getId() , getCoordx(), getCoordy());
35                 break;
36             case "right":
37                 mover(getCoordx()+5);
38                 guijug.gb_moveSpriteCoord(getId() , getCoordx(), getCoordy());
39                 break;
40                 //Disparo
41             case "space":
42                 for(int i=0;i<proyectiles.length;i++) {
43                     if(proyectiles[i]==null) {
44                         proyectiles[i]=new Bala(getCoordx(), getCoordy(), (i%10)+10, Constantes.DIR_N);
45                         guijug.gb_addSprite(proyectiles[i].getId(), proyectiles[i].getImagen(),true);
46                         guijug.gb_moveSpriteCoord(proyectiles[i].getId(),proyectiles[i].getCoordx(), proyectiles[i].getCoordy());
47                         guijug.gb_setSpriteVisible(proyectiles[i].getId(), true);
48                         Tshot++;
49                         break;
50                     }
51                 }
52             }
53         }
54         for(int j=0; j<proyectiles.length;j++) {
55             if(proyectiles[j]!=null) {
56                 if (proyectiles[j].getCoordy()<=10) {
57                     guijug.gb_setSpriteVisible(proyectiles[j].getId(), false);
58                     proyectiles[j]=null;
59                 }else {
60                     proyectiles[j].move(Constantes.DIR_N,1);
61                     guijug.gb_setSpriteImage(proyectiles[j].getId(), proyectiles[j].getImagen());
62                     guijug.gb_moveSpriteCoord(proyectiles[j].getId(), proyectiles[j].getCoordx(), proyectiles[j].getCoordy());
63                 }
64             }
65         }
66     }
```

**Velocidad del juego** en “Principal” líneas 95 y 96

Para ralentizar el juego creamos la variable ritmo, a la que se irá sumando uno cada vez que se haga una iteración y cada cuatro se ejecutara parte de nuestro código como son el movimiento de los enemigos.

```
95                  ritmo++;
96                  if(ritmo%4==0) {
```

**Disparo del enemigo** en “Principal” líneas 101 a 123

```
101                 if(i>=4&&i<20&&enemigos[i].bala==null) {
102                     int ran=(int)(Math.random()*150);
103                     if(ran==1) {
104                         enemigos[i].bala= new Bala(enemigos[i].getCoordx(), enemigos[i].getCoordy(), enemigos[i].getId(), Constantes.DIR_S);
105                         gui.gb_addSprite(enemigos[i].getId()+1000, enemigos[i].bala.getImagene(), true);
106                         gui.gb_setSpriteVisible(enemigos[i].getId()+1000, true);
107                     }
108                 }
109                 if(enemigos[i].bala!=null) {
110                     enemigos[i].bala.move(Constantes.DIR_S,1);
111                     gui.gb_setSpriteImage(enemigos[i].getId()+1000, enemigos[i].bala.getImagene());
112                     gui.gb_moveSpriteCoord(enemigos[i].getId()+1000, enemigos[i].bala.getCoordx(), enemigos[i].bala.getCoordy());
113                     //Impacto de bala enemiga en el Jugador
114                     if(Math.abs(enemigos[i].bala.getCoordx()-jugador.getCoordx())<=5 &&
115                             Math.abs(enemigos[i].bala.getCoordy()-jugador.getCoordy())<=5) {
116                         jugador.setVivo(false);
117                         gui.gb_setSpriteVisible(enemigos[i].getId()+1000, false);
118                     }
119                     if(enemigos[i].bala.getCoordy()>=Constantes.MAXLargo*10) {
120                         enemigos[i].bala=null;
121                         gui.gb_setSpriteVisible(enemigos[i].getId()+1000, false);
122                     }
123                 }
```

Esta función primero comprueba que los enemigos estén vivos para poder entrar, si el enemigo es rojo, se genera un número aleatorio entre el 0 y 150, y si el número generado es 1, se crea una bala (con su propia ID, las coordenadas del enemigo y dirección sur) del enemigo correspondiente a la posición del array (que está siendo recorrido por un for).

Debajo se comparan la posiciones del jugador y las de las balas enemigas, si se encuentran dentro de un radio de media cuadrícula, se pone al jugador como muerto para activar su

animación y hacemos desaparecer el proyectil enemigo, el proyectil también desaparecerá si supera el límite inferior del tablero.

**Movimiento del enemigo** en “Principal” líneas 125 a 195

```
125                     if (aleteo%2==0 && enemigos[i].isVivo()) {
126                         if(i<40&&i>=20) {
127                             enemigos[i].setImagen("enemy3G0.png");
128                             gui.gb_setSpriteImage(enemigos[i].getId(), enemigos[i].getImagen());
129                         }
130                         if(i<20&&i>=4) {
131                             enemigos[i].setImagen("enemy2G0.png");
132                             gui.gb_setSpriteImage(enemigos[i].getId(), enemigos[i].getImagen());
133                         }
134                         if(i<4) {
135                             enemigos[i].setImagen("enemy1G0.png");
136                             gui.gb_setSpriteImage(enemigos[i].getId(), enemigos[i].getImagen());
137                         }
138                     }else if (aleteo%2==1 && enemigos[i].isVivo()){
139                         if(i<40&&i>=20) {
140                             enemigos[i].setImagen("enemy3G1.png");
141                             gui.gb_setSpriteImage(enemigos[i].getId(), enemigos[i].getImagen());
142                         }
143                         if(i<20&&i>=4) {
144                             enemigos[i].setImagen("enemy2G1.png");
145                             gui.gb_setSpriteImage(enemigos[i].getId(), enemigos[i].getImagen());
146                         }
147                         if(i<4) {
148                             enemigos[i].setImagen("enemy1G1.png");
149                             gui.gb_setSpriteImage(enemigos[i].getId(), enemigos[i].getImagen());
150                         }
151                     }
152                     //Avance de los enemigos en la horizontal
153                     enemigos[i].move(direne, 1);
154                     gui.gb_moveSpriteCoord(enemigos[i].getId(), enemigos[i].getCoordx(), enemigos[i].getCoordy());
155                 }
156             }
157         }
158         //Movimiento del enjambre de enemigos hacia abajo al tocar el borde
159         for(int o=0;o<enemigos.length;o++) {
160             if(enemigos[o]!=null) {
161                 if(enemigos[o].getCoordx()>=((Constantes.MAXancho*10)-5)) {
162                     for(int k=0;k<enemigos.length;k++){
163                         if(enemigos[k]!=null) {
164                             enemigos[k].move(Constantes.DIR_Abajo, 2);
165                             gui.gb_moveSpriteCoord(enemigos[k].getId(), enemigos[k].getCoordx(), enemigos[k].getCoordy());
166                         }
167                     }
168                 }else if((enemigos[o].getCoordx()+5)<=Constantes.MINancho+10){
169                     for(int k=0;k<enemigos.length;k++){
170                         if(enemigos[k]!=null) {
171                             enemigos[k].move(Constantes.DIR_Abajo, 1);
172                             gui.gb_moveSpriteCoord(enemigos[k].getId(), enemigos[k].getCoordx(), enemigos[k].getCoordy());
173                         }
174                     }
175                 }
176             }
177         }
178         //Aparicion en la parte superior de una fila de enemigos cuando alcanza el limite inferior.
179         for (int o=0;o<enemigos.length;o++) {
180             if(enemigos[o]!=null) {
181                 if(enemigos[o].getCoordy()>=((Constantes.MAXlargo*10)-5)) {
182                     enemigos[o].setCoordy(0);
183                     gui.gb_moveSpriteCoord(enemigos[o].getId(), enemigos[o].getCoordx(), enemigos[o].getCoordy());
184                 }
185             }
186         }
187         for(int i=0;i<enemigos.length;i++){
188             for(int j=0;j<jugador.proyectiles.length;j++){
189                 //Cambio de la direccion de los enemigos en el borde
190                 if(enemigos[i]!=null) {
191                     if(enemigos[i].getCoordx()>=((Constantes.MAXancho*10)-5)) {
192                         direne=Constantes.DIR_W;
193                     }else if((enemigos[i].getCoordx()+5)<=Constantes.MINancho+10){
194                         direne=Constantes.DIR_E;
195                     }
```

Dentro de este apartado hemos considerado oportuno meter también la animación de “aleteo” de los enemigos, que se desarrolla de las líneas 125 a la 151, esta animación es muy simple ya que con una sola variable de tipo “int” y el módulo de esta controlamos que en cada iteración los enemigos tengan un sprite diferente a la anterior, alternando los sprites con la terminación “G1” y “G0”.

En las líneas 153 y 154 se produce el movimiento en horizontal del enjambre de enemigos, este se hace con la función “move” la cual requiere una dirección y un número de unidades (4 décimos de casilla, las direcciones este (4,0) y oeste (-4,0)). En las líneas 190 a 195 se realiza la comparación de la posición de los enemigos vivos con los límites laterales del tablero, si algún enemigo los supera, la dirección del enjambre cambia.

En las líneas 159 a la 177 se desarrolla el algoritmo de desplazamiento vertical descendente del enjambre al sobrepasar los enemigos los límites laterales del tablero, de igual forma que en el cambio de dirección, al cumplirse esta comparación el enjambre baja una casilla, en este caso la función “move” recibe como dirección “DIR_Abajo” (0,1) y 2 como unidades de desplazamiento. Si una fila de enemigos llega al límite inferior del tablero esta aparecerá por la parte superior (líneas 179 a 186).

**Interacciones enemigo-jugador** en “Principal” líneas 187 a 240

```
187             for(int i=0;i<enemigos.length;i++){
188                 for(int j=0;j<jugador.proyectiles.length;j++){
189                     //Cambio de la direccion de los enemigos en el borde
190                     if(enemigos[i]!=null) {
191                         if(enemigos[i].getCoordx()>=((Constantes.MAXancho*10)-5)) {
192                             direne=Constantes.DIR_W;
193                         }else if((enemigos[i].getCoordx()+5)<=Constantes.MINancho+10){
194                             direne=Constantes.DIR_E;
195                         }
196                         //Hits proyectil del jugador contra enemigo
197                         if(jugador.proyectiles[j]!=null) {
198                             if(Math.abs(enemigos[i].getCoordx()-jugador.proyectiles[j].getCoordx())<=5 &&
199                                     Math.abs(enemigos[i].getCoordy()-jugador.proyectiles[j].getCoordy())<=5) {
200                                 gui.gb_setSpriteVisible(jugador.proyectiles[j].getId(), false);
201                                 jugador.proyectiles[j]=null;
202                                 hit++;
203                                 enemigos[i].setVivo(false);
204                                 }
205                         }
206                         //Hit del jugador contra enemigo
207                         if(Math.abs(enemigos[i].getCoordx()-jugador.getCoordx())<=5 &&
208                                 Math.abs(enemigos[i].getCoordy()-jugador.getCoordy())<=5) {
209                             jugador.setVivo(false);
210                             //Si el enjambre golpea al jugador vuelven a su posicion inicial(los vivos)
211                             for(int k=0;k<enemigos.length;k++) {
212                                 if(enemigos[k]!=null) {
213                                     if(k<4) {
214                                         enemigos[k].setCoordx(70+k*10);
215                                         enemigos[k].setCoordy(5);
216                                         gui.gb_moveSpriteCoord(enemigos[k].getId(), enemigos[k].getCoordx(), enemigos[k].getCoordy());
217                                     }
218                                     if(k<12&&k>=4) {
219                                         enemigos[k].setCoordx(50+(k-4)*10);
220                                         enemigos[k].setCoordy(15);
221                                         gui.gb_moveSpriteCoord(enemigos[k].getId(), enemigos[k].getCoordx(), enemigos[k].getCoordy());
222                                     }
223                                     if(k<20&&k>=12) {
224                                         enemigos[k].setCoordx(50+(k-12)*10);
225                                         enemigos[k].setCoordy(25);
226                                         gui.gb_moveSpriteCoord(enemigos[k].getId(), enemigos[k].getCoordx(), enemigos[k].getCoordy());
227                                     }
228                                     if(k<30&&k>=20) {
229                                         enemigos[k].setCoordx(40+(k-20)*10);
230                                         enemigos[k].setCoordy(35);
231                                         gui.gb_moveSpriteCoord(enemigos[k].getId(), enemigos[k].getCoordx(), enemigos[k].getCoordy());
232                                     }
233                                     if(k<40&&k>=30) {
234                                         enemigos[k].setCoordx(40+(k-30)*10);
235                                         enemigos[k].setCoordy(45);
236                                         gui.gb_moveSpriteCoord(enemigos[k].getId(), enemigos[k].getCoordx(), enemigos[k].getCoordy());
237                                     }
238                                 }
239                             }
240                         }
```

En las líneas 197 a 205 se comprueba si un proyectil del jugador ha impactado contra un enemigo, gracias al método “Math.abs” el cual hace que si las coordenadas del enemigo y las del proyectil coinciden dentro de un radio de 5 unidades (décimas de casilla), inicie las siguientes acciones: el proyectil se hace invisible y “null”, la constante “hit” se actualiza sumando uno (esta variable se usa para calcular la estadística de “accuracy”) y la característica “vivo” del enemigo (tipo “boolean”) se cambia a “false”.

En las líneas 207 a 240 al igual que antes se comprueba la proximidad del jugador con los enemigos, si alguno de estos está a 5 unidades (décimos de casilla) o menos, el jugador pasará a tener el atributo “vivo” (tipo “boolean”) “false”. Cuando un enemigo golpea directamente al jugador, hemos considerado oportuno mover a los enemigos del enjambre que queden con vida a sus posiciones iniciales para no hacer perder al jugador más vidas de las precisas.

**Animación de explosión (Enemigos)** en “Principal” líneas 242 a 263

```
241                     //Animacion de la explosion del enemigo
242                     if(!enemigos[i].isVivo()) {
243                         enemigos[i].setTi(enemigos[i].getTi()+1);
244                         if(enemigos[i].getTi()<=10) {
245                             enemigos[i].setImagen("explosion20.png");
246                             gui.gb_setSpriteImage(enemigos[i].getId(), enemigos[i].getImagen());
247                         }
248                         if(enemigos[i].getTi()<=20&&enemigos[i].getTi()>10) {
249                             enemigos[i].setImagen("explosion21.png");
250                             gui.gb_setSpriteImage(enemigos[i].getId(), enemigos[i].getImagen());
251                         }
252                         if(enemigos[i].getTi()<=30&&enemigos[i].getTi()>20) {
253                             enemigos[i].setImagen("explosion22.png");
254                             gui.gb_setSpriteImage(enemigos[i].getId(), enemigos[i].getImagen());
255                         }
256                         if(enemigos[i].getTi()<=40&&enemigos[i].getTi()>30) {
257                                 enemigos[i].setImagen("explosion23.png");
258                                 gui.gb_setSpriteImage(enemigos[i].getId(), enemigos[i].getImagen());
259                         }
260                         if(enemigos[i].getTi()>40) {
261                             enemigos[i].setImagen("explosion24.png");
262                             gui.gb_setSpriteImage(enemigos[i].getId(), enemigos[i].getImagen());
263                         }
```

En las líneas 242 a 263 se produce la animación de muerte de los enemigos, esta animación solo se produce cuando la propiedad “vivo” del enemigo es “false”, los enemigos también disponen de la variable “ti” ya que esta está incluida en la clase “Elemento” de la cual heredan tanto los enemigos como el jugador, esta se utiliza para, según el número de iteraciones que hayan ocurrido desde la muerte del enemigo, cambiar los sprites de la animación, y al finalizar esta, hacer al enemigo invisible y “null”.

**Contador de puntos** en “Principal” líneas 265 a 289

```
264                     //Feed de los enemigos muertos y suma de puntos
265                     if(enemigos[i].getTi()==50) {
266                         if(i<40&&i>=20) {
267                             ptos+=100;
268                             gui.gb_setValuePointsUp(100);
269                             gui.gb_println("Zako muerto");
270                         }
271                         if(i<20&&i>=4) {
272                             ptos+=250;
273                             gui.gb_setValuePointsUp(250);
274                             gui.gb_println("Goei muerto");
275                         }
276                         if(i<4) {
277                             ptos+=500;
278                             gui.gb_setValuePointsUp(500);
279                             gui.gb_println("Capitan Galaga muerto");
280                         }
281                         gui.gb_setValuePointsDown(ptos);
282                         enemigos[i].setTi(0);
283                         gui.gb_setSpriteVisible(enemigos[i].getId(), false);
284                         if(enemigos[i].bala!=null) {
285                             gui.gb_setSpriteVisible(enemigos[i].getId()+1000, false);
286                         }
287                         enemigos[i].bala=null;
288                         enemigos[i]=null;
289                     }
```

Al realizarse la animación de muerte de un enemigo se comprueba que tipo de enemigo era para sumar a la variable “ptos” la cantidad correspondiente, se refleja en las líneas de comando el tipo de enemigo, se actualiza en la interfaz la puntuación total y los puntos que ha dado el último enemigo eliminado.

**Animación de explosión (Jugador)** en “Principal” líneas 295 a 323

```
294             //Animacion de la explosion del jugador
295             if(!jugador.isVivo()) {
296                 jugador.setTi(jugador.getTi()+1);
297                 if(jugador.getTi()<=10) {
298                     gui.gb_animateDamage();
299                     jugador.setImagen("explosion11.png");
300                     gui.gb_setSpriteImage(jugador.getId(), jugador.getImagen());
301                 }
302                 if(jugador.getTi()<=20&&jugador.getTi()>10) {
303                     jugador.setImagen("explosion12.png");
304                     gui.gb_setSpriteImage(jugador.getId(), jugador.getImagen());
305                 }
306                 if(jugador.getTi()<=30&&jugador.getTi()>20) {
307                     jugador.setImagen("explosion13.png");
308                     gui.gb_setSpriteImage(jugador.getId(), jugador.getImagen());
309                 }
310                 if(jugador.getTi()>30) {
311                     jugador.setImagen("explosion14.png");
312                     gui.gb_setSpriteImage(jugador.getId(), jugador.getImagen());
313                 }
314                 if(jugador.getTi()==40) {
315                     gui.gb_setSpriteVisible(juvida, false);
316                     juvida--;
317                     gui.gb_setValueHealthCurrent(juvida);
318                     jugador.setTi(0);
319                     gui.gb_setSpriteImage(jugador.getId(), jugador.getImagn());
320                     gui.gb_println("Has perdido una vida, te quedan: "+(juvida));
321                     jugador.setVivo(true);
322                 }
323             }
```

En las líneas 295 a 323 se produce la animación de muerte del jugador, la animación solo se produce cuando la característica “vivo” del jugador es “false”, esta ocurre de manera similar a la de los enemigos, cambiando solo los sprites, y al finalizar, se restaura la imagen original al jugador, se actualiza la propiedad “vivo” del jugador a “true”, se resta una vida y se reinicia la variable “ti” a 0.

# 4. Funcionalidad

Hemos completado los siguientes objetivos:

- **Sprint 1:** generar el enjambre en su posición inicial, crear al jugador en la parte inferior y que este no supere los límites laterales al moverse y dotarlo de puntos, puntos totales, vida actual y vida máxima.

- **Sprint 2:** mover el enjambre de enemigos lateral y verticalmente limitándolo dentro del tablero, animar los enemigos a medida que se mueven, crear y animar los proyectiles del jugador al pulsar la barra espaciadora y eliminarlos al impactar con un enemigo. **Opcional:** que aparezca en la consola que enemigo ha sido matado.

- **Sprint 4:** mostrar en el tablero el número de vidas del jugador, representadas con tres naves en la esquina inferior izquierda, crear y animar los proyectiles enemigos, que son disparados por probabilidad y pueden impactar con el jugador, si esto pasa,

este pierde una de sus vidas, desapareciendo una de las naves de la esquina. Si el enemigo colisiona con el jugador estos volverán a su posición inicial, esta decisión ha sido propia y tan solo alarga la “vida del juego”, es decir el tiempo que dura, pero para hacerlo como indica el enunciado solo sería necesario añadir:

```
for(int i=0; i<enemigos.length; i++){
        if(Math.abs(enemigos[i].getCoordx()-jugador.getCoordx())<=5 &&
        Math.abs(enemigos[i].getCoordy()-jugador.getCoordy())<=5) {
                salir=false;
        }
}
```

- **Sprint 5:** animar la muerte del jugador y de los enemigos al ser impactados con distintas imágenes.

- **Extra:** Botón salir, posibilidad de ganar y salir (al matar a todos los enemigos), posibilidad de perder y salir (al perder las 3 vidas), “feedback” de las pulsaciones en la consola, estadísticas de porcentaje de precisión y disparos realizados, animaciones de “aleteo” de los enemigos, “portrait” del jugador, animación de “garra” en el “portrait” del jugador al ser golpeado, hemos eliminado la cuadrícula, hemos ajustado la velocidad del juego y las animaciones para que queden fluidas.

# 5. Conclusión

La realización de esta práctica final ha supuesto un desafío y una introducción a la programación con bibliotecas proporcionadas por la universidad, aunque nos hemos encontrado dificultades en no entender correctamente cómo funcionaban algunas funciones, mediante prueba y error nos hemos adaptado de la mejor manera que nos ha sido posible, llegando a hacer este juego, lo que ha sido bastante satisfactorio tras haber invertido bastantes horas en él.

Los **problemas** a los que nos hemos enfrentado han sido varios, pero los más destacables fueron:

- **El array de balas:** nuestro jugador dispone de un array de tipo “Bala” con 10 elementos, para la creación y correcto funcionamiento del mismo, tuvimos que hacer muchas pruebas, al principio no sabíamos cómo movernos por el array para crear nuevas balas y, además, mover las que ya habían sido disparadas, además, al disparar las 10 balas, no podíamos crear nuevas al estar el array lleno, a esto le pusimos solución haciendo que al disparar la bala número 11, usando el módulo de 10 (11%10=1), la bala número 1 se hiciera null, y así reutilizar ese espacio del array, para actualizar la posición de las balas hicimos que se recorriera el array de balas en cada iteración y si la bala era diferente de “null”, siguiera subiendo.

- **Botón “Salir”:** Al pulsar el botón “Salir” intentábamos recibir de la consola el comando que aparecía (”exit game”), pero no conseguíamos que la comparación de la cadena de texto “exit game” y nuestra última acción recogida de la consola se

hiciera cierta nunca. Para solventar este problema lo que hicimos fue coger la última acción recogida por teclado (donde al final vimos que sí aparecía) y compararla con “exit game”, por lo que finalmente la comparación se hace cierta y se cambiaba el boolean que permitía terminar el juego.