Top Down Shooter

con GameMaker

Por Héctor Costa

http://escueladevideojuegos.net

Índice

Clase 1: Movimiento Básico

1- room 1536*1200, 60FPS

2- spr_jugador

3- obj_jugador.Create

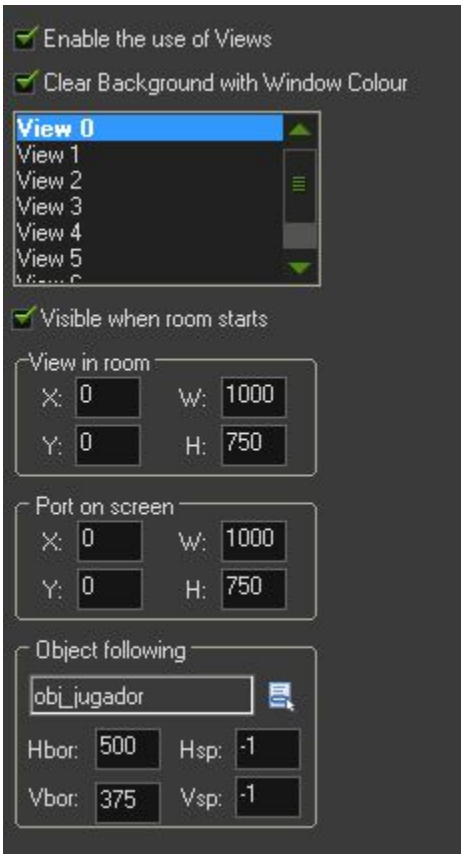
```
/// Velocidades
vv = 0;
vh = 0;
v = 4;
```

4- obj_jugador.Step

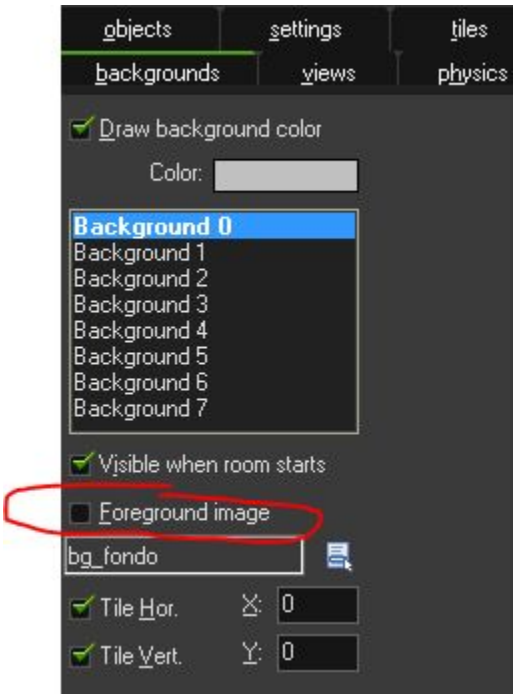
```
/// Dirección, distancia y velocidades
dir = point_direction(x,y,mouse_x,mouse_y);
dis = point_distance(x,y,mouse_x,mouse_y);
vh = 0;
vv = 0;

/// Movimiento
if (keyboard_check(ord('W'))) vv-=v;
if (keyboard_check(ord('S'))) vv+=v;
if (keyboard_check(ord('A'))) vh-=v;
if (keyboard_check(ord('D'))) vh+=v;

/// Establecemos movimiento final y la dirección del sprite
hspeed = vh;
vspeed = vv;
image_angle = dir;
```

Clase 2: Fondo, vista y cámara

1- Creamos la vista:



2- Añadimos el fondo BG fondo:


objects
settings
tiles
backgrounds
views
physics
Draw background color
Color:
Background 0
Background 1
Background 2
Background 3
Background 4
Background 5
Background 6
Background 7
Visible when room starts
Foreground image
bg_fondo
Tile Hor.
X:
0
Tile Vert.
Y:
0

Clase 3: Colisiones y máscaras básicas

1- Creamos obj_solido con sprite 64*64 transparencia al 125

2- Añadimos al jugador, antes del el movimiento final con nuestro sistema de colisiones perfectas:

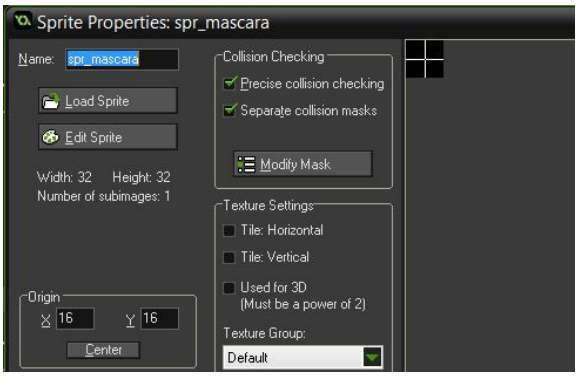
```
/// Control de colisiones contra solidos
if (place_meeting(x+vh,y,obj_solido)) {
    // Comprobando colisión horizonal
    for(var i=0;i<abs(vh);i++){
        if (place_meeting(x+sign(vh),y,obj_solido)) then break;
        x += sign(vh);
    }
    vh = 0;
}
if (place_meeting(x,y+vv,obj_solido)) {
    // Comprobando colisión vertical
    for(var i=0;i<abs(vv);i++){
        if (place_meeting(x,y+sign(vv),obj_solido)) then break;
        y += sign(vv);
    }
    vv = 0;
}
```

3- Ponemos algunas paredes y probamos como nos quedamos parados al chocar y girar:

4- Vamos a debugear que ocurre, dibujando la máscara de colisión del objeto:

/// Debugueamos la máscara de colisión
draw_rectangle(bbox_left, bbox_top, bbox_right, bbox_bottom, true);
draw_self();

5- Solucionamos el problema con una máscara de colisión más pequeña centrada sobre el jugador, centrada. Puede ser de 48*48:



6- Añadimos esta máscara al jugador y probamos de nuevo, todavía choca al girar ¿Por qué?:

7- Choca porque estamos rotando la máscara junto al sprite al hacer el image_angle. Si logramos no girar la máscara de colisión tendremos una colisión perfecta. Para conseguirlo tenemos que evitar cambiar el image_angle, pero podemos dibujar el sprite directamente rotado en el draw gracias a nuestra variable dir, así que comenzamos borrando la línea:

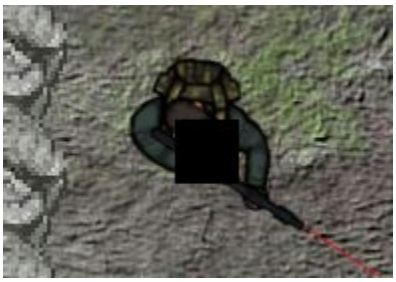
Step
image_angle = dir; // BORRAR

Draw

```
draw_sprite_ext(sprite_index,image_index,x,y,image_xscale,image_yscale,dir,image_blend,image_alpha);
draw_sprite(object_get_mask(object_index),0,x,y);
```

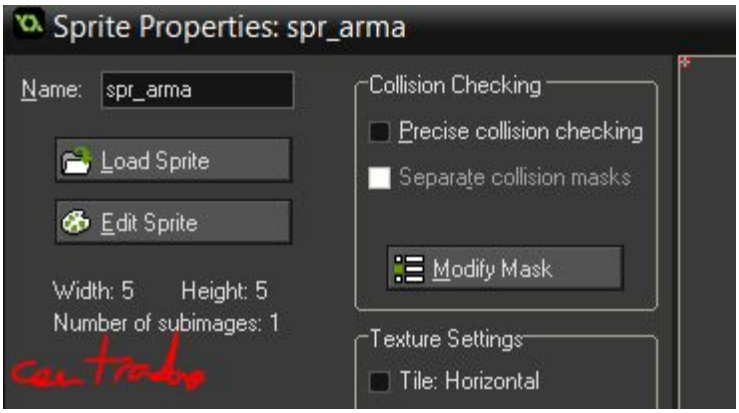
---

8. Debugeamos y veremos que es perfecto, quitamos el debug del draw:

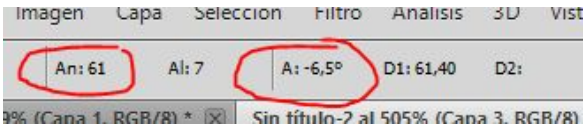


Clase 4: Disparar balas desde el arma

1- Como tenemos un sprite en el que el arma no está centrada vamos a crear un objeto distinto a posicionar automáticamente sobre el arma de nuestro jugador. Le damos profundidad -99 para que esté encima de todo:



2- obj_arma. Posicionaremos en cada fotograma el arma sobre el jugador. Hay que determinar una rotación y una distancia para proyectar un punto y poner el arma:



Step: **/// Posicionamos el arma rotada en base al jugador y su velocidad actual**

```
if (object_exists(obj_jugador)){
    x = obj_jugador.x + obj_jugador.vh + lengthdir_x(61, obj_jugador.dir - 4);
    y = obj_jugador.y + obj_jugador.vv + lengthdir_y(61, obj_jugador.dir - 4);
}
```

3- Creamos el objeto bala, obj_bala 3*3 negro, centrado.

4- Añadimos Create:

/// Variables

```
dis_max = 400; // antes de que desaparezca
```

5- Añadimos Step:

/// Distancia máxima borrar disparo

```
dis = point_distance(x,y,obj_jug.x,obj_jug.y);
if (dis>dis_max){
    instance_destroy();
}
```

6- Comprobamos una colisión proyectada en el siguiente fotograma contra una pared y la destruimos:

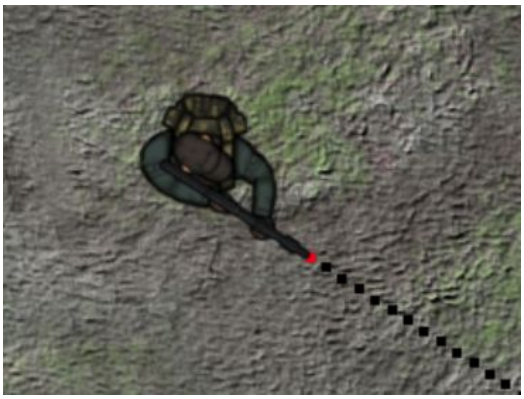
/// Detectar colisión solido

```
if (collision_line(x,y,x+hspeed,y+vspeed,obj_solido,false,false)){
      instance_destroy();
}
```

7- Detectamos cuando el jugador dispara, con el ratón. Detectamos en el step del arma para crear los disparos (1 disparo por step):

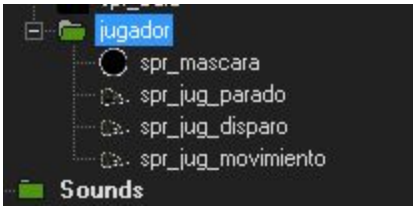
```
/// Detectamos si el jugador dispara
if (mouse_check_button(mb_left)){
    disparo = instance_create(x,y,obj_bala);
    disparo.direction = obj_jugador.dir;
    disparo.speed = 10;
}
```

8- Probamos, disparamos y escondemos el obj_arma: visible = false;



Clase 5: Jugador - Estados y animaciones

1- Importamos los sprites animados para jugador parado, movimiento y disparo, los entramos en el mismo punto:

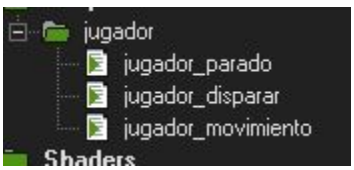


2- Vamos a definir una enumeración de estados para jugador en el Create, por defecto estaremos parados:

/// Definición de estados

```
enum ej {
   parado       =  jugador_parado,
   movimiento  =  jugador_movimiento,
   disparar      =  jugador_disparar
}

estado = ej.parado;
```

3- Ahora crearemos los scripts de estado del jugador para manejar las animaciones:



jugador_parado

```
sprite_index = spr_jug_parado;
image_speed = 0.50;
```

jugador_movimiento

```
sprite_index = spr_jug_movimiento;
image_speed = 0.70;
```

jugador_disparar

```
sprite_index = spr_jug_disparo;
image_speed = 0.40;
```

---

4. Agregamos un control de estados al jugador después de la colisiones. Cambiaremos la variable disparar por el estado disparar:

```
/// Control de estados
if (!ej.disparar){
    if (vh != 0 or vv != 0) estado = ej.movimiento;
    if (vh == 0 and vv== 0) estado = ej.parado;
}
script_execute(estado);
```

5- Ahora añadiremos un evento Animation End para cambiar de estado al acabar la animación de disparo. Así utilizaremos la velocidad de la animación para limitar el número de balas que podemos disparar:

```
/// Detectar final del disparo
if (estado == ej.disparar) estado = ej.parado;
```

6- Finalmente modificamos el momento de crear la bala en obj_arma:

```
/// Detectamos si el jugador dispara
if (mouse_check_button(mb_left) && obj_jugador.estado != ej.disparar){
    disparo = instance_create(x,y,obj_bala);
    disparo.direction = obj_jugador.dir;
    disparo.speed = 10;
    // Establecemos el estado disparando y reiniciamos la animación
    obj_jugador.estado = ej.disparar;
    obj_jugador.image_index = 0;
}
```

Clase 6: Zombies - Estado de movimiento (IA)

1- Creamos el objeto zombie con sprite movimiento de zombie, centrado.

/// Variables iniciales
```
randomize();
dir = random(360);
v = 0;
```

2- Definimos los estados:

/// Definición de estados
```
enum ez {
   movimiento  =  zombie_movimiento
}

estado = ez.movimiento;
```

3- Creamos el script de movimiento

zombie_movimiento
```
sprite_index = spr_zombie_movimiento;
image_speed = 0.30;
speed = 0.50;
```

4- En el step controlamos los estados y el movimiento final:

/// Control de estados
```
script_execute(estado);
```

4.5 Esta vez dibujaremos el zombie directamente en el draw, así nos evitamos utilizar de nuevo el image_angle y el problema de las colisiones. Además el zombie se mueve en la ***direction*** configurada, así que es más sencillo establecer el movimiento final y la dirección:

```
///Establecemos el movimiento final
speed = v;
direction = dir;
```

5- Luego comprobamos las colisiones contra sólidos con nuestro sistema perfecto:

/// Control de colisiones contra sólidos

```
vh = lengthdir_x(v,dir);
vv = lengthdir_y(v,dir);
colision = false;

// Comprobando colisión horizontal
if (place_meeting(x+vh,y,obj_solido)) {
    for(var i=0;i<abs(vh);i++){
        if (place_meeting(x+sign(vh),y,obj_solido)) then break;
        x += sign(vh);
    }
    vh = 0;
    colision = true;
}

// Comprobando colisión vertical
if (place_meeting(x,y+vv,obj_solido)) {
    for(var i=0;i<abs(vv);i++){
        if (place_meeting(x,y+sign(vv),obj_solido)) then break;
        y += sign(vv);
    }
    vv = 0;
    colision = true;
}

// Para evitar que se quede quieto si nos topamos
// contra un solido nos damos la vuelta  180º
if (place_meeting(x+vh,y+vv,obj_solido)) {
    dir = direction + 180;
}
```

---

6. Ahora vamos cambiar la máscara del zombie también a la máscara cuadrada pequeña para optimizar el área de colisión.

---

7 - Y para acabar podemos añadir en la bala una colisión contra el zombie para destruirla:

/// Detectar colisión zombie

```
zombie = collision_line(x,y,x+hspeed,y+vspeed,obj_zombie,false,false);
if (zombie){
    instance_destroy();
}
```

Clase 7: Zombies - Estado de persecución (IA)

1- Creamos un nuevo estado:

/// Definición de estados

```
enum ez {
   movimiento =  zombie_movimiento,
   perseguir  =  zombie_perseguir
}
estado = ez.movimiento;
```

2- Creamos el script:

zombie_perseguir

```
sprite_index = spr_zombie_movimiento;
image_speed = 0.7;
v = 2;
dir = point_direction(x,y,obj_jugador.x,obj_jugador.y);
```

3- Creamos una variable para el radio:

radio_perseguir = 450;

---

4. Implementamos la inteligencia artificial

/// Control de estados
distancia = point_distance(x,y,obj_jugador.x,obj_jugador.y);
es_visible = !collision_line(x,y,obj_jugador.x,obj_jugador.y,obj_solido,false,false);
es_perseguible = es_visible and distancia <= radio_perseguir;

// Inteligencia Artifical
if (es_perseguible) estado = ez.perseguir;
else estado = ez.movimiento;

script_execute(estado);

5- Podemos debugear el radio de persecución y el estado:

/// Dibujo y debug
draw_circle(x,y,radio_perseguir,true);
draw_text(x,y+45,string(estado));
draw_self();

6- El zombie nos sigue pero ocurre un bug, y es que cuando nos persigue y choca contra una pared se da la vuelta, pero como seguimos en su línea de visión nos vuelve a perseguir y se forma un bucle infinito. Para solucionarlo podemos desactivar durante un tiempo su modo perseguir y restablecer luego con una alarma :

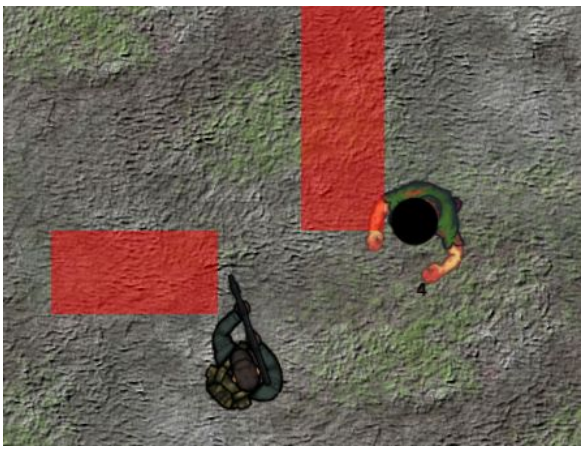
Create
perseguir = true;

Step
// Si hay colisión cambiamos la dirección a un lado aleatorio
if (colision){
    dir = direction + 180;
    // Un segundo para volver a poder perseguir
    if (perseguir) alarm[0] = room_speed*random_range(1,2);
    perseguir = false;
}

Alarma[0]
/// Volver a permitir perseguir
perseguir = true;

Y cambiamos la condición en los estados:

es_perseguible = es_visible and distancia <= radio_perseguir ***and perseguir;***



7- Aún así cuando nos persigue y choca se da la vuelta y al cabo de poco vuelve a por nosotros por el mismo camino. Vamos a hacer que se dé la vuelta pero no perfecta, sino con una dirección algo aleatoria.

/// Control de colisiones contra solidos

…

..

dir = direction + irandom_range(135,270);

8- El tema de la inteligencia artificial daría para mucho.. pero como introducción es suficiente.

Clase 8: Zombies - Estado de ataque IA

1- Importamos el sprite de ataque y creamos una variable para el radio:

radio_atacar = 55;

2- Debugeamos este radio:

draw_circle(x,y,radio_atacar,true);

3- Añadimos un nuevo estado:

/// Definición de estados
enum ez {
    movimiento = zombie_movimiento,
    perseguir = zombie_perseguir,
    atacar   = zombie_atacar
}
estado = ez.movimiento;

4- Y el script

sprite_index = spr_zombie_atacar;
image_speed = 0.25;
v= 0;
dir = point_direction(x,y,obj_jugador.x,obj_jugador.y);

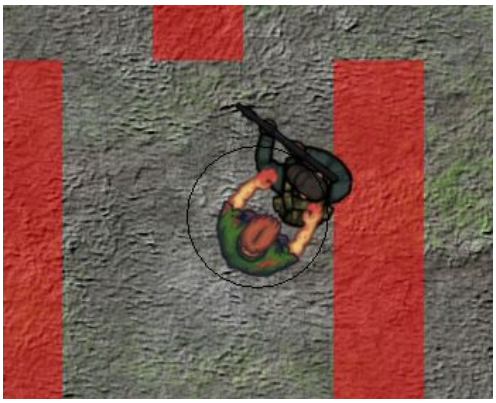
5- Manejamos el nuevo estado:

/// Control de estados
distancia   = point_distance(x,y,obj_jugador.x,obj_jugador.y);
es_visible   = !collision_line(x,y,obj_jugador.x,obj_jugador.y,obj_solido,false,false);
es_perseguible = es_visible and distancia <= radio_perseguir and perseguir;
es_atacable   = es_visible and distancia <= radio_atacar;

// Inteligencia Artifical
if (es_atacable) estado = ez.atacar;
else if **(es_perseguible) estado = ez.perseguir;**
else estado = ez.movimiento;

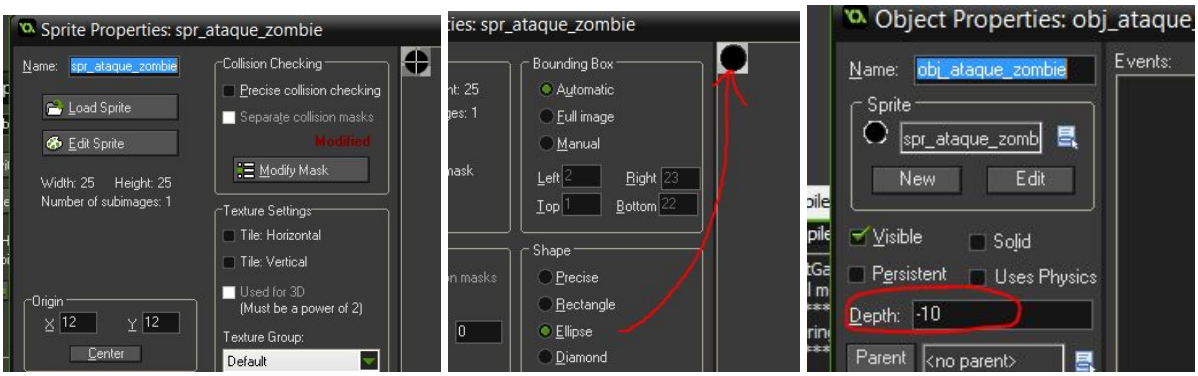
script_execute(estado);

6- Ponemos la profundidad del zombie a -1 para estar encima del jugador y desactivamos el debug:



Clase 9: Zombies - Colisión de ataque

1- Creamos un obj_ataque_zombie, con un sprite que sea la mitad o así de la máscara pequeña ellipse, centrado con una profundidad -10, para ponerlo encima de todo:



2- En el objeto zombie, script de ataque, detectaremos cuando la subimagen del sprite está en posición de colisionar y crearemos una instancia de objeto colisión:

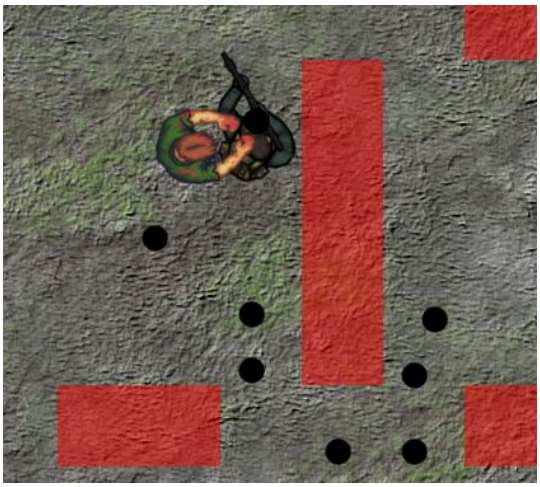
zombie_atacar

```
...
// Detectar subimagen de la colisión
if (image_index == 5){
    instance_create(obj_jugador.x,obj_jugador.y,obj_ataque_zombie);
}
```

3- Si ejecutamos el juego veremos que algo falla y nosiempre se crean las instancias, es porque hay que reiniciar la animación cuando cambiemos de estado:

```
/// Control de estados
estado_inicial  = estado;
…
..
// Reiniciar animación
if (estado != estado_inicial) image_index = 0;

script_execute(estado);
```

Ahora ya funciona bien:



4- Volvemos al jugador, creamos un nuevo bloque Step para detectar colisiones contra ataques después de las colisiones contra sólidos. Mostraremos que hay un ataque y borraremos su instancia :

```
/// Control de colisiones contra ataques de zombie
ataque_zombie = instance_place(x,y,obj_ataque_zombie);
if (ataque_zombie) {
    show_debug_message("Me atacan! id=" + string(ataque_zombie));
    with(ataque_zombie) instance_destroy();
    alarm[0] = room_speed/4 // coloreamos de rojo ¼  de segundo
}
```

5- En este punto hay que comentar que solo estamos comprobando una colisión de ataque por fotograma. No es problema porque difícilmente tendremos más de 60 zombies pegando a la vez. Sin embargo para hacer un juego totalmente perfecto, tendremos que comprobar en cada step si hay más de un ataque, simplemente repitiendo el proceso por el número de ataques creados que haya:

repeat(instance_number(obj_ataque_zombie)){

```
    ataque_zombie = instance_place(x,y,obj_ataque_zombie);
    ….
}
```

6- Podemos hacer un image_blend rojo cuando el zombie nos golpea durante unos instantes con una alarma. La podemos crear vacía **Alarma[0] y comprobarlo justo al final del Step:**

```
/// Colorear de rojo
if (alarm[0] > -1) image_blend = c_red;
else image_blend = c_white; // No colorear
```

7- Ya que ponemos blends podríamos cambiar durante un instante el image_blend del zombie al recibir un disparo de la misma manera, con otra alarma **Alarm[1]**:

En la bala:

```
/// Detectar colisión zombie
zombie = collision_line(x,y,x+hspeed,y+vspeed,obj_zombie,false,false);
if (zombie){
    zombie.alarm[1] = room_speed/15;
    instance_destroy();
}
```

En el zombie al final del step:

```
/// Colorear de rojo oscuro (de rojo se ve demasiado y molesta)
var rojo_oscuro = make_color_rgb(155,017,030);
if (alarm[1] > -1) image_blend = rojo_oscuro;
else image_blend = c_white; // No colorear
```

http://ralcolores.mrket.net/index.php?index=HEX_DOWN

8- Ya lo tenemos, solo nos falta hacer invisible este objeto ataque zombie.

Clase 10: Datos - Gestionar vida de jugador y zombies

1- Añadimos unas variables con la vida del jugador y el poder_zombie:
vida_base = 150;
vida = vida_base;
poder_zombie = 5;

2- Cuando detectemos una colisión de ataque le restamos vida al jugador:
// Restamos algo de vida al jugador
vida -= poder_zombie ;

3- Y en el step detectamos si el jugador tiene 0 vida para reiniciar la room

/// Reiniciar juego si morimos
if (vida <= 0) room_restart();

4- Añadimos unas variables con la vida del zombie:
..
vida_base = 200;
vida = vida_base;

5- Creamos una variable poder en la bala. Restamos vida al zombie cuando choca la bala contra él:

Create
poder = 6;

Step colisión zombie
zombie.vida -= poder ;

6- En el Step del zombie comprobamos si su vida es 0 y lo destruimos:

/// Comprobar si el zombie muere
if (vida <= 0) instance_destroy();

7- En el Draw del jugador y el zombie debugeamos su vida y hacemos pruebas:

Zombie
draw_text(x,y+45,string(vida));

Jugador
draw_text(x,y+40,string(vida));


62
200
140
200

Clase 11: Jugador - Barra vida y avatar (HUD)

1- En el create del jugador redimensionamos el tamaño de la GUI y los componentes de la barra:

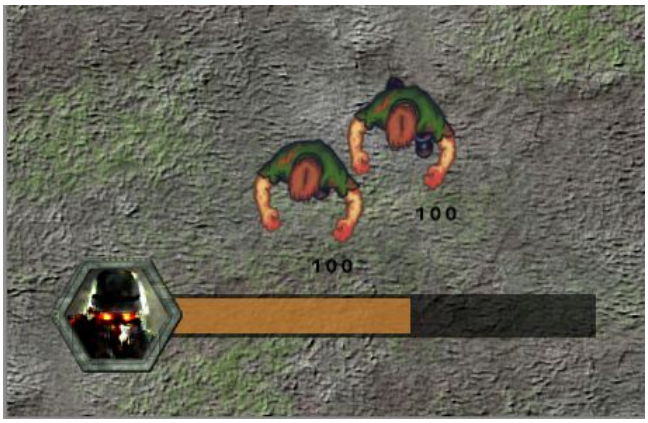
```
/// Determinamos el tamaño de la GUI y componentes
display_set_gui_size(view_wview[0],view_hview[0]);
barra_h = 300;
barra_v = 24;
barra_c1 = c_green;
barra_c2 = c_orange;
barra_c3 = c_red;
barra_cf = c_black;
ofx = 115;
ofy = view_hview[0]-barra_v-60;
```

2- En el evento draw GUI dibujamos la barra de vida:

```
/// Dibujamos la barra
var porcentaje = round(vida*100/vida_base)/100;
var ancho = clamp(barra_h * porcentaje,0,barra_h);

draw_set_alpha(0.6)
draw_set_color(barra_cf);
draw_rectangle(115-3,ofy-3,barra_h+ofx+3,barra_v+ofy+3,false);

if(porcentaje > 0.67) draw_set_color(barra_c1);
else if (porcentaje > 0.33)  draw_set_color(barra_c2);
else draw_set_color(barra_c3);

draw_rectangle(ofx,ofy,ancho+ofx,barra_v+ofy,false);

draw_set_color(c_black);
draw_set_alpha(1)
```

3- También dibujamos el avatar encima después de importar el sprite:

```
/// Dibujamos el avatar
draw_sprite(spr_avatar,0,30,view_hview[0]-120);
```



4- Desactivamos el debug de vida del jugador

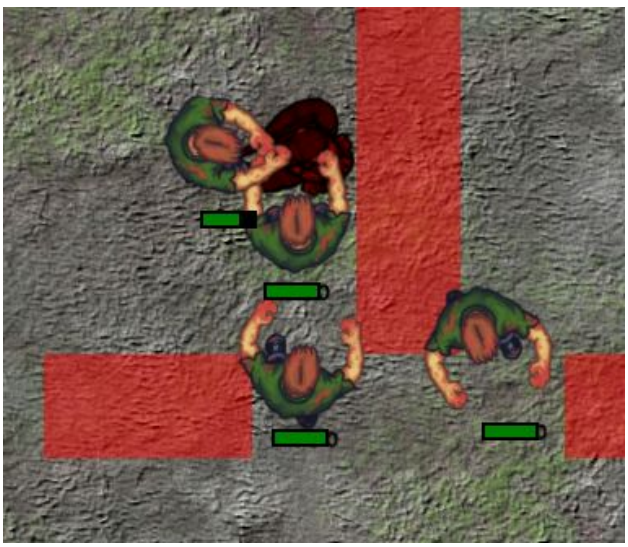
Clase 12: Zombies - Barra de vida que desaparece

1- Determinamos el tamaño de los elementos de los zombies para su interfaz de vida:

```
/// Determinando componentes de la barra
barra_h = 30;
barra_v = 6;
barra_c1 = c_green;
barra_c2 = c_orange;
barra_c3 = c_red;
barra_cf = c_black;
ofx = -8;
ofy = 48;
```

2- Dibujamos en el DRAW normal

```
/// Dibujamos la barra
var porcentaje = round(vida*100/vida_base)/100;
var ancho = clamp(barra_h * porcentaje,0,barra_h);


draw_set_color(barra_cf);
draw_rectangle(x-2+ofx,y-2+ofy,x+barra_h+2+ofx,y+barra_v+2+ofy,false);

if(porcentaje > 0.67) draw_set_color(barra_c1);
else if (porcentaje > 0.33)  draw_set_color(barra_c2);
else draw_set_color(barra_c3);
draw_rectangle(x+ofx,y+ofy,x+ancho+ofx,y+barra_v+ofy,false);

draw_set_color(c_black);
```



3- Para hacer la barra de vida de los enemigos visible únicamente durante un momento cuando les atacamos podemos utilizar una variable con la opacidad por defecto a 0:

barra_alpha = 0;

4- Cuando una bala impacte un zombie, estableceremos la variable:

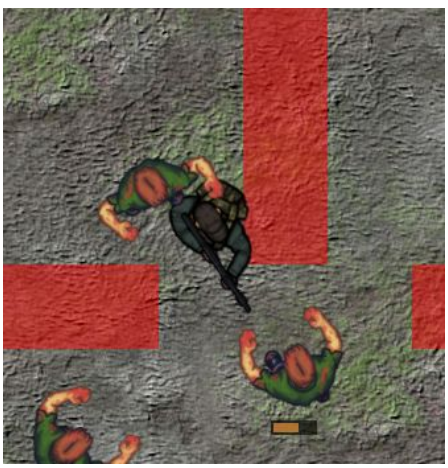
/// Detectar colisión zombie
zombie = collision_line(x,y,x+hspeed,y+vspeed,obj_zombie,false,false);
if (zombie){
    zombie.vida-=damage;
    zombie.barra_alpha = 0.7;

5- Y finalmente restamos un poco de opacidad en cada Draw para hacerla invisible gradualmente:

/// Dibujamos la interfaz
var porcentaje = round(vida*100/vida_base)/100;
var ancho = clamp(barra_h * porcentaje,0,barra_h);

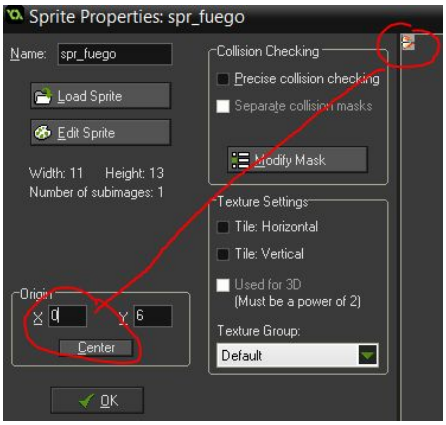
draw_set_alpha(barra_alpha)
..
...
draw_set_alpha(1)
barra_alpha-=0.0025;

6- Desactivamos el debug de vida del zombie



Clase 13: Arma - Efecto destello al disparar

1- Importamos el sprite de fuego del arma, centrado a la izq en medio:



2- Creamos una alarma vacía para gestionar el tiempo a mostrar el sprite en el arma:

arma.Alarm[0]
/// Alarma para el destello

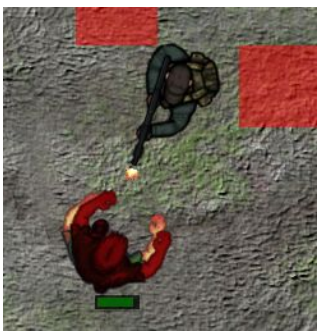
3- En el step detectamos el disparo y establecemos la alarma durante unos instantes para mostrar el destello:

```
…
disparo.speed = 10;
alarm[0] = room_speed/4; /// mostraremos el destello ¼ de segundo
```

4- En el draw del arma añadimos:

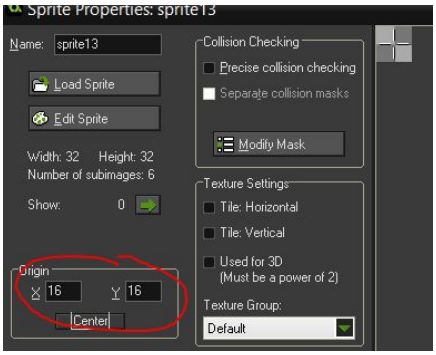
```
/// Dibujar fuego disparo en el mismo sitio que el arma
if(obj_jug.estado = ej.disparar && alarma[0] != -1){
    draw_sprite_ext(spr_fuego,0,x,y,1,1,obj_jugador.image_angle,c_white,1);
}
```

5- Nos aseguramos que el arma esté visible



Clase 14 : Arma - Sustituir cursor por mirilla

1- Empezamos importante los sprites de la mirilla, centrados:



2- Creamos un nuevo objeto mirilla y establecemos la animación:

```
/// Variables
image_speed = 0.15;
image_alpha = 0.85;
```

3- En el step posicionamos la mirilla sobre el mouse

```
/// Posicionamos la mirilla sobre el cursor
x = mouse_x;
y = mouse_y;
```

4- Ahora vamos a quitar el objeto arma de la room y lo crearemos en el Create del jugador, a la vez que la mirilla:

```
/// Crear objetos dinámicos del jugador
instance_create(0,0,obj_arma);
instance_create(0,0,obj_mirilla);
```

5- Ahora **pondremos la profundidad de la mirilla arriba de todo -99 y escondemos el ratón, al final del create de la mirilla:**

```
/// Escondemos el ratón
window_set_cursor(cr_none);
```

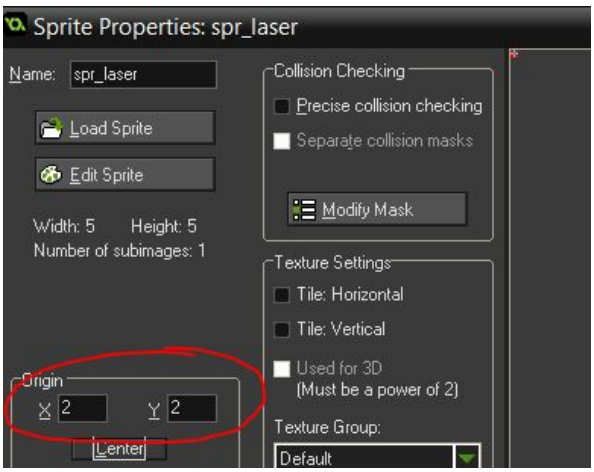
6- Finalmente, algo que queda bien es iniciar el juego con el jugador mirando hacia el medio. Podemos posicionar el ratón en medio de la pantalla fácilmente en el create de la mirilla:

/// Escondemos el ratón y lo ponemos en medio de la room
window_set_cursor(cr_none);
window_mouse_set(view_wview/2,view_hview/2);

Clase 15 Arma: - Láser proyectado en objetivo

1- Crear un láser es muy sencillo gracias a una función de GameMaker que permite mover objetos hasta que choquen. Siguiente este patrón, el láser se basa en un objeto que en cada fotograma parte de la punta del arma y lo movemos en la dirección que mira el jugador hasta que choque contra un objeto. Luego se traza una línea entre arma y el objeto y ahí tenemos el láser.

2- Empezamos creando el objeto laser con un sprite rojo de 3*3 centrado.



3- En el Create del láser establecemos su alcance máximo:

```
// Variables
alcance = 350;
```

4- En el Step gestionamos el láser:

```
/// Gestionamos el laser

// Lo posicionamos en el arma
x = obj_arma.x;
y = obj_arma.y;

// Luego lo movemos hasta el primer objeto sólido en dirección del jugador:
move_contact_solid(obj_jugador.image_angle,longitud_laser);
```

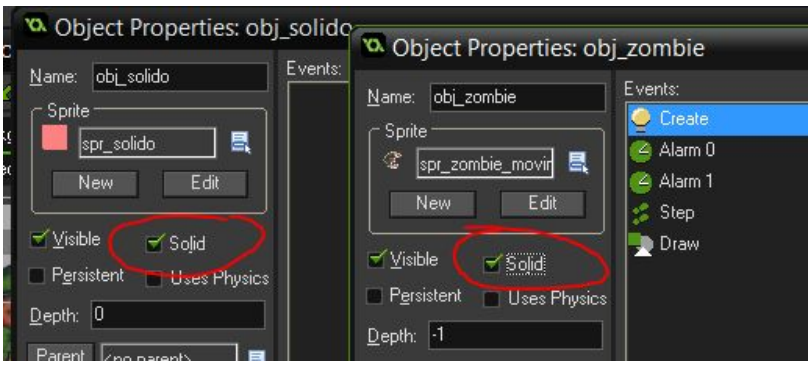
---

5. En el draw trazamos una línea desde el arma hasta el láser, y también dibujamos el propio láser (luego ya lo esconderemos cuando ataquemos cuerpo a cuerpo):

```
/// Dibujar laser
draw_set_color(c_red);
draw_set_alpha(0.75);
```

```
draw_line(obj_arma.x,obj_arma.y,x,y);
draw_set_color(c_black);
draw_set_alpha(1);
draw_self();
```

6- Ahora hacemos que las paredes y los zombies sean objetos sólidos:

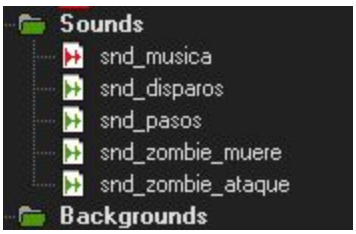


7- Y finalmente creamos el láser junto al jugador:

```
/// Crear objetos dinámicos del jugador
instance_create(0,0,obj_arma);
instance_create(0,0,obj_mirilla);
instance_create(0,0,obj_laser);
```

Clase 16 Audio: Añadir música y sonidos

1- Importamos el audio:



- Música de fondo: snd_musica
- Disparos: snd_disparos
- Pasos: snd_pasos
- Zombie_Ataque: snd_zombie_ataque
- Zombie_Muere: snd_zombie_muere

2- Establecemos una regulación del sonido:

Disparos

Zombie Ataque

Zombie Muere

Pasos

3- Llamamos los sonidos…

Música: En el jugador en el create al final

```
/// Reproducir musica de fondo
audio_stop_all();
audio_play_sound(snd_musica, 100, false);
```

Disparos: En el Arma en el Step que crea los disparos:

```
…
        alarm[0] = room_speed/16;
        // Sonido de disparos
        if (!audio_is_playing(snd_disparos)){
            audio_play_sound(snd_disparos,70,true); // hacemos un loop
        }
}
// Si no estamos apretando el ratón paramos los disparos
if (!mouse_check_button(mb_left)){
```

```
    audio_stop_sound(snd_disparos);
}
```

Zombie Atacar: En el script del estado atacar del zombie

```
if (image_index == 5){
    instance_create(obj_jugador.x,obj_jugador.y,obj_ataque_zombie);
    audio_play_sound(snd_zombie_ataque,70,false);
}
```

Zombie Morir: En el momento que destruimos la instancia:

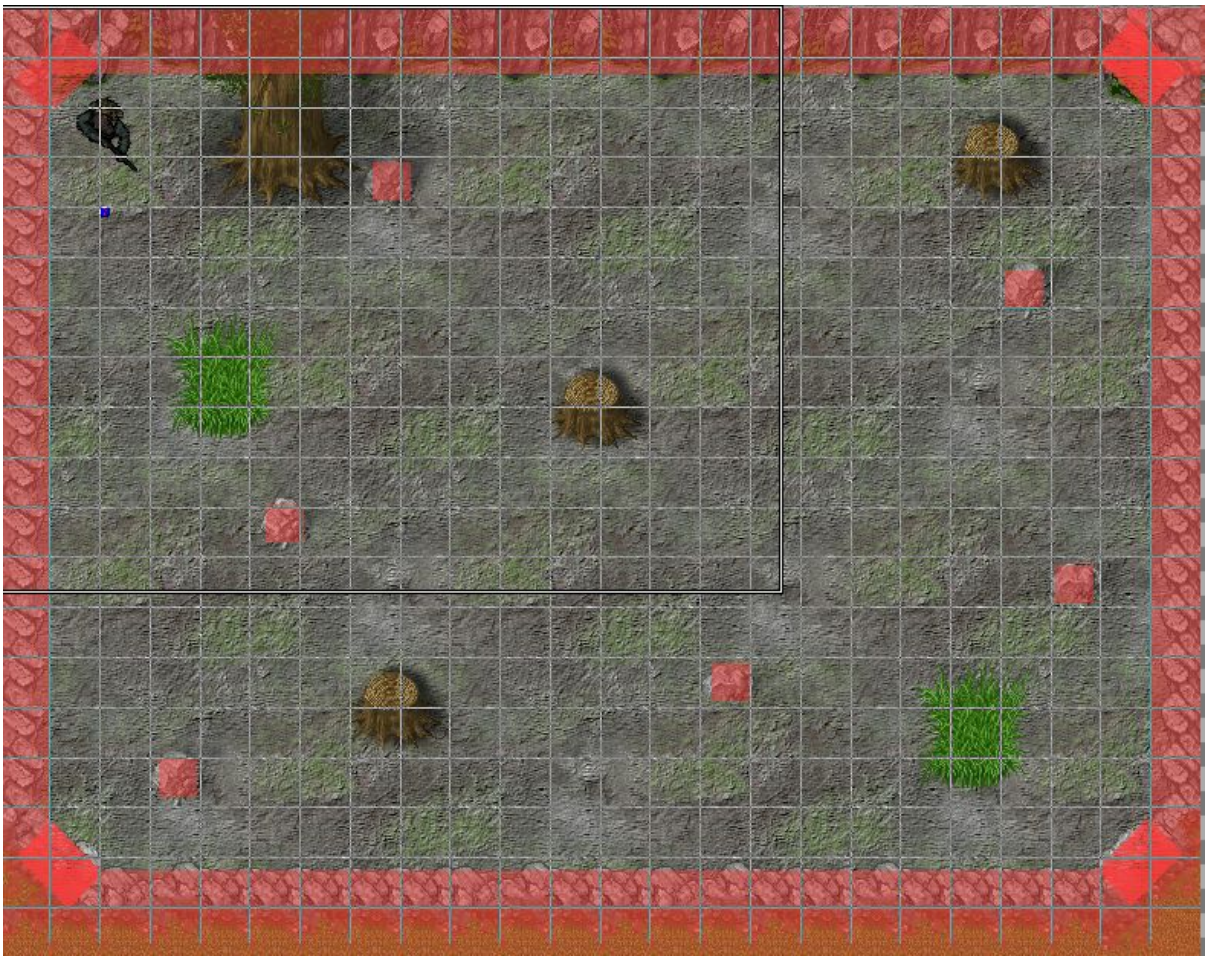
```
/// Comprobar si el zombie muere
if (vida <= 0) {
     audio_play_sound(snd_zombie_muere,70,false);
    instance_destroy();
}
```

Pasos: En el momento justo antes de establecer el movimiento

```
if (vh != 0 or vv != 0) {
    if (! audio_is_playing(snd_pasos)){
        audio_play_sound(snd_pasos,100,false);
    }
}
```

Clase 17 Escenario: Tiles y elementos sólidos

- Improvisación
- Importación de los backgrounds (use as Tileset)
- Creación de los sprites del tronco y el árbol, hjos de sólido
- Crear el mapa con tiles, crear otra capa para piedras y adornos
- Añadir overlay objetos sólidos invisibles



Clase 18: Partículas - Balas que se desintegran o parecen sangre

---

1- Vamos a crear un sistema de partículas para generar ráfagas cuando los disparos se destruyan al chocar. Lo haremos en el láser, que es el punto donde acaban chocando las partículas.

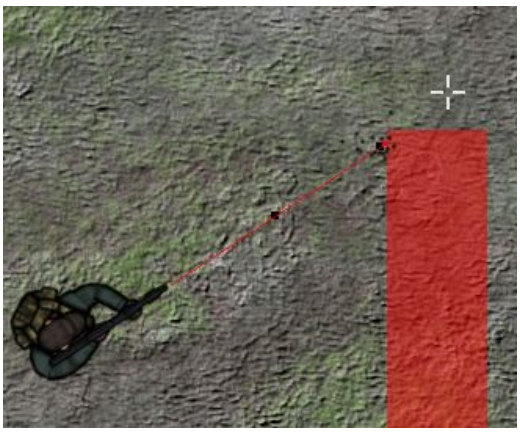
2- En el Create del láser creamos todo el sistema de partículas:

```
/// Sistema de partículas para los disparos
sistema = part_system_create();     // Creamos el sistema
part_system_depth(sistema, -150); // Profundidad sobre todos los elementos de la room

particula = part_type_create(); // Creamos una partícula

// Definimos la partícula
part_type_shape(particula,pt_shape_pixel); // Tipo píxel 1*1
part_type_size(particula,1,1,0,0); // Tamaño
part_type_scale(particula,1,1);   // Escala
part_type_color1(particula,c_black); // Color
part_type_alpha1(particula,1); // Opacidad
part_type_speed(particula,0.50,1,0,0); // Velocidad
part_type_direction(particula,0,100,0,0); // Dirección
part_type_orientation(particula,0,0,0,0,1); // Orientación
part_type_blend(particula,0); // Mezcla de color
part_type_life(particula,10,30); // Fotogramas de vida

// Creamos un emisor
emisor = part_emitter_create(sistema);
```

3- Ahora justo en el momento que la bala choca contra un sólido creamos una ráfaga de partículas:

```
/// Detectar colisión pared
if (collision_line(x,y,x+hspeed,y+vspeed,obj_solido,false,false)){
    with(obj_laser){
        // Dinámicamente cambiaremos la vida de la partícula
        part_type_life(particula,10,30);
        // Su color a negro
        part_type_color1(particula,c_black);
        // Y una dirección al lado contrario a donde está chocando
```

```
        part_type_direction(particula, -180+obj_jugador.image_angle-50,
180+obj_jugador.image_angle+50,0, 0);
        // Definimos la región del emisor,
        part_emitter_region(sistema,emisor,other.x+hspeed,other.x+hspeed,
other.y+vspeed,other.y+vspeed,0,0);
        // Creamos la ráfaga con un número de partículas
        part_emitter_burst(sistema,emisor,particula,8);  // Número de partículas de la ráfaga
    }
    instance_destroy();
}
```



4- Para las balas que impactan en el zombie hacemos lo mismo cambiando el tiempo de vida al doble, el color a rojo y una variación deángulo:

```
/// Detectar colisión zombie
zombie = collision_line(x,y,x+hspeed,y+vspeed,obj_zombie,false,false);
if (zombie){
    zombie.vida-=poder;
    zombie.barra_alpha = 0.7;
    zombie.alarm[1] = room_speed/15;
    with(obj_laser){
        // Dinámicamente cambiaremos la vida de la partícula
        part_type_life(particula,20,60);
        // Su color a negro
        part_type_color1(particula,c_red);
        // Y una dirección al lado contrario a donde está chocando
        part_type_direction(particula,
-180+obj_jugador.image_angle-90,180+obj_jugador.image_angle+90,0,0);
        // Definimos la región del emisor,
        part_emitter_region(sistema,emisor,other.x+hspeed,other.x+hspeed,
other.y+vspeed,other.y+vspeed,0,0);
        // Creamos la ráfaga con un número de partículas
        part_emitter_burst(sistema,emisor,particula,8);  // Número de partículas de la ráfaga
```

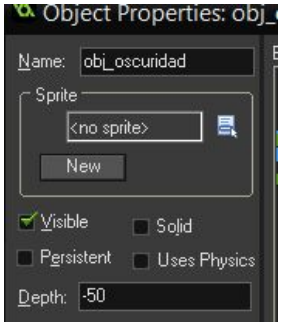
```
    }
    instance_destroy();
}
```

5- Por último limpiamos la memoria en un evento Room End para el láser:

```
/// Limpiamos la memoria
part_emitter_clear(sistema,emisor);
part_type_clear(particula);
part_system_clear(sistema);
```

Clase 19: Superfícies - Efecto oscuridad e iluminación

1- Vamos a crear un **objeto oscuridad** para gestionar la capa de sombra.



Le daremos depth -50 para que se dibuje por encima del jugador y los zombies:

En el evento Create crearemos una superficie del tamaño de la view:

```
activar = true; // Para permitirnos desactivarla
oscuridad = surface_create(view_wview, view_hview);
```

2- Luego en el Step, como las capas son volátiles y a veces desaparecer, tenemos que asegurarnos que existe antes de dibujar encima o volverla a crear:

```
if (surface_exists(oscuridad )){
    surface_set_target(oscuridad  );
    draw_set_color(c_white);
    draw_rectangle(0,0,view_wview, view_hview,false); // dibujamos un rect. blanco
    surface_reset_target();
} else {
    oscuridad = surface_create(view_wview, view_hview);
}
```

3- Y en el draw dibujamos la superficie para cubrir toda la view:

```
if (activar){
    draw_surface(oscuridad ,view_xview, view_yview);
    draw_set_blend_mode(bm_normal);
}
```

4- Podemos crear un botón debugger para desactivar la oscuridad con F1:

```
activar = !activar;
```

5- Ahora creamos dinámicamente la oscuridad con el jugador y si probamos lo veremos todo negro, eso es bueno porque significa que vamos bien.

```
/// Crear objetos dinámicos del jugador
```

…..
instance_create(0,0,obj_oscuridad);

6- Si ponemos el juego en marcha veremos lo siguiente:

Una capa blanca lo cubre todo. Claro, te estarás preguntando, porqué hemos dibujado la capa en blanco y no en negro? Bueno, pues eso es debido a que el blanco representa la máxima ilumación. Dado que ahora tenemos una superficie con el máximo de luz, tenemos en bandeja la posibilidad de sustraer toda la luz de esta capa (el color blanco) cuando la dibujemos sobre la superficie de aplicación, que es la superficie genérica de GameMaker. Si en el Draw cambiamos el modo de dibujo tendremos lo siguiente:

```
if (activar){
   draw_set_blend_mode(bm_subtract);
   draw_surface(oscuridad ,view_xview, view_yview);
   draw_set_blend_mode(bm_normal);
}
```

Como véis, al dibujarla sustrayendo todo el color blanco del juego, ahora nos queda todo negro, esto significa que ya tenemos la oscuridad preparada. Por cierto, la GUI se dibuja en otra capa más tarde, por eso no le afecta lo que hagamos en la superficie de aplicación.

7- En este momento vamos a crear algo de iluminación, pero como lo hacemos? Pues sustrayendo color blanco (la luz máxima) de esta capa tan blanca que teníamos que luego se veía negra. Es decir, hay que sustraer color blanco de alguna forma antes de dibujarla en el Draw.

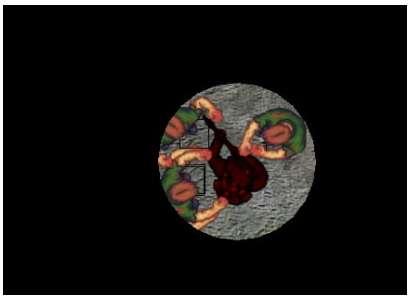
Vamos a dibujar una ellipse encima de la posición del jugador, justo después de dibujar el rectángulo. Empezamos activando el modo de mezcla para sustraer color, dibujamos la ellipse y volvemos a poner el modo de mezcla por defecto:

draw_set_blend_mode(bm_subtract);
draw_ellipse(obj_jugador.x-radio_luz_jugador,obj_jugador.y-radio_luz_jugador,obj_jugador.x+radio_luz_jugador,obj_jugador.y+radio_luz_jugador,false);
draw_set_blend_mode(bm_normal);

Si ponemos el juego en marcha encontraremos un par de problemillas:

8- El primero es que al movernos fuera de la view, la ellipse no nos sigue. Para que nos siga correctamente tenemos que dibujar restando la posición de la view en todo momento:

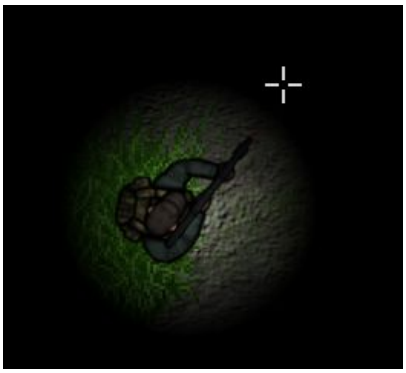
draw_ellipse(obj_jugador.x-radio_luz_jugador**-view_xview**,obj_jugador.y-radio_luz_jugador**-view_yview**,obj_jugador.x+radio_luz_jugador**-view_xview**,obj_jugador.y+radio_luz_jugador**-view_yview**,false);



---

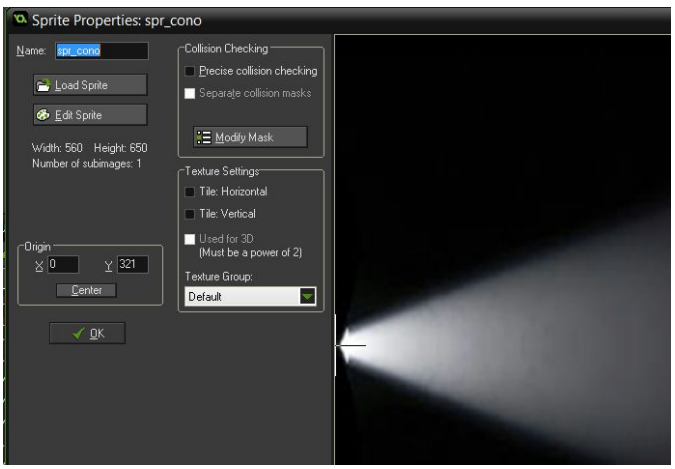
9 Muy bien, sustraemos una ellipse pero de forma uniforme y queremos simular una luz ¿Como lo hacemos? Dado que estamos extrayendo podríamos dibujar la ellipse con una gradiente blanco-negro para simular una luz con color progresivo:

draw_ellipse_colour(obj_jugador.x-radio_luz_jugador-view_xview,obj_jugador.y-radio_luz_jugador-view_yview,obj_jugador.x+radio_luz_jugador-view_xview,obj_jugador.y+radio_luz_jugador-view_yview,**c_white,c_black**,false);



Genial.

10- En cuanto al cono de luz hacia donde estamos mirando cómo lo hacemos? Bueno, este caso es más complejo, pero sigue la misma lógica de restar color. Sin embargo el cono no lo dibujaremos dinámicamente, sino que lo importamos de un sprite que tengo preparado y que creé en Photoshop utilizando pinceles. Centramos el sprite justo en su el inicio del cono en X = 0 y a mitad de altura:



Con este cono, que como véis es más iluminado por la parte inicial, podemos restar color al dibujarlo de la misma forma que antes al con la ellipse. Lo dibujaremos después de la ellipse del jugador, encima del propio jugador y en la dirección que éste mira, teniendo en cuenta igual que antes que debemos restar la posición actual de la view para cada eje:

draw_sprite_ext(spr_cono,0,obj_jugador.x-view_xview,obj_jugador.y-view_yview,1,1,obj_jugador.dir,c_white,1);

Clase 20: Gestor de niveles y generador de zombies

1- Esto va a ser largo…Comenzamos creando un obj_generador **Persistente**, ya que gestionará el juego permanentemente una vez lo creamos al principio. En el create del objeto vamos a añadir unas variables:

```
/// Variables
nivel = 1;
zpn = 5; // zombies por nivel
```

2-Ahora, cuando la room se pone en marcha **Room Start,** crearemos una pila para gestionar los zombies del nivel actual automáticamente:

```
/// Determinar zombies del nivel
zombies_vencidos = 0;
zombies_ronda = zpn*nivel;
zombies_maximos_activos  = zpn+nivel;
zombies = ds_stack_create();

// Creamos una pila de numeros
for(var i=0;i<zombies_ronda;i++){
    // Un numero por zombie que vamos a crear
    ds_stack_push(zombies,i);
}

// Sacamos los primeros numeros de la pila hasta el máximo activo de zombies
for(var i=0;i<zombies_maximos_activos;i++){
    // Si queda algun numero en la pila lo sacamos y creamos un zombie
    if (ds_stack_size(zombies)>0) {
        ds_stack_pop(zombies);
        instance_create(0,0,obj_zombie);
    }
}
```

3- Cuando finaliza la room Room End, liberamos la pila de la memoria:

```
/// Liberamos la memoria
ds_stack_clear(zombies);
```

4- Ahora vamos a crer un evento de usuario que se encargará de una importante tarea. Lo llamaremos cada vez que matemos un zombie, sumará 1 al contador de zombies vencidos

e intentará sacar de la pila un número, si lo consigue creará un zombie, además llamará una alarma que comprobará si tenemos que pasar al siguiente nivel al cabo de 2 segundos:

User Defined 0 /// Evento para crear un zombie y siguiente nivel

```
zombies_vencidos++;

if (ds_stack_size(zombies)>0) {
    ds_stack_pop(zombies);
    instance_create(0,0,obj_zombie);
}
// siguiente nivel
alarm[1] = room_speed * 2;
```

5- Si el numero de zombies vencidos es igual al numero de zombies que tiene en total la ronda, sumaremos 1 al nivel y reiniciaremos la room (recordad que al ser este objeto Persistente no se borra ni reinicia)

Alarm[1]

```
/// Siguiente nivel?
if (zombies_vencidos>=zombies_ronda){
    nivel++;
    room_restart();
}
```

6- Vamos un momento al zombie, justo antes de destruirlo si muere ejecutamos el evento de usuario, lo que disparará el evento, sumará zombie a vencidos y todo éso:

```
if (vida <= 0) {
    audio_play_sound(snd_zombie_muere,70,false);
    // Llamamos el evento de creacion de zombies
    with(obj_generador) {
        event_user(0);
    }
    instance_destroy();
}
```

7- Ahora borramos todos los zombies de la room y ponemos el generador, si prendemos las luces veremos que todos los zombies aparecen arriba a la izquierda… Claro, es el punto 0,0 que hemos marcado. Tenemos que crear un sistema que posicione automáticamente los nuevos zombies. Para hacerlo vamos al código del zombie, y enel Create vamos a determinar un lugar aleatorio, que no esté encima de un sólido, pero que a la vez esté a un mínimo de distancia del jugador para que no aparezcan encima:

```
/// Determinacion automatica del sitio
do{
    // Buscamos un lugar aleatorio en la room
    x = random_range(0,room_width);
    y = random_range(0,room_height);
    // Debe ser a cierta distancia del jugador
    hay_distancia = point_distance(obj_jugador.x,obj_jugador.y,x,y) > 550;
    // Y no sobre cualquier objeto
    no_colision = place_free(x,y);
} until(hay_distancia and no_colision);
```

Clase 21: Dibujar el marcador y el nivel

1- Si ponemos el juego en marcha ya debería funcionar bien. Si matamos algunos zombies nos debería reiniciar la room automáticamente, lo malo es que no sabemos qué está pasando… necesitamos implementar algo de interfaz. Vamos a crear 2 fuentes, una para mostrar el nivel al principio de cada room, y otra para escribir el número de zombies por matar arriba a la derecha. Podéis usar la que queráis, yo he usado la fuente Chiller en negrita, una a 36px y la otra a 72px:

2- Voy a crear una alarma Alarm[0] , para gestionar si muestro el mensaje de Nivel:

Alarm[0]

```
/// Esconder Nivel
mostrar_nivel = false;
```

Que llamaré al inicio de la room en el evento Room Start y se verá durante 3 segundos:

```
/// Mostrar nivel al inicio
mostrar_nivel = true;
alarm[0] = room_speed * 3;
```

Para dibujarlo y que se haga transparente es tan fácil como en el evento Draw GUI añadir un bloque así:

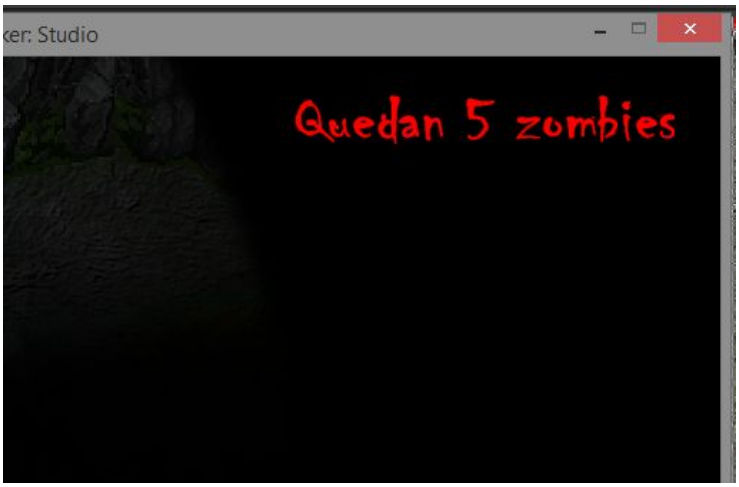
```
/// Dibujar Nivel
if(mostrar_nivel){
    draw_set_font(fnt_nivel);
    draw_set_halign(fa_center);
    draw_set_valign(fa_middle);
    draw_set_color(c_white);
    draw_set_alpha(alarm[0]/100);
    draw_text(view_wview/2,view_hview/2,"Nivel " +string(nivel));
    draw_set_alpha(1);
```

}

La opacidad se gestionará utilizando el valor de la alarma/100, lo cual me dará el decimal necesario (recordar que la transparencia se define de 0 a 1)

3- En cuanto al marcador de zombies, sólo tenemos que añadir otro bloque en la gui que dibuje el marcador en la parte superior derecha:

```
/// Dibujar Marcador
draw_set_font(fnt_marcador);
draw_set_halign(fa_right);
draw_set_valign(fa_middle);
draw_set_color(c_red);
draw_text(view_wview-30,45,"Quedan " + string(zombies_ronda-zombies_vencidos)+" zombies");
```



Clase 22: Superficies - Pausar juego

1- Ya casi estamos, vamos a añadir una opción para pausar el juego. Lo haremos como no, utilizando superficies. Debemos hacerlo así porque es la única forma de pausar todos los elementos del juego con una función especial de GameMaker y a la vez conservar una captura de pantalla de los mismos en una superficie.

Comenzamos creando un obj pausa. Este se encargará de pausar el juego mientras exista y desactivará todas las otras instancias temporalmente, además de pausar todos los sonidos.

```
/// Variables
pausado = false;
captura = -1;
audio_pause_all();
```

2- El truco de hacer una captura con los elementos de la room consiste en clonar la superficie de aplicacion en otra superficie (solo la porción visible) juste después de que todos los elementos se hayan dibujado. Para hacerlo podemos utilizar el event PostDraw para clonar un dibujo de la room. Solo lo haremos una vez, así que utilizamos la variable pausado para saber que hemos tomado la captura:

```
if (pausado == false){
    captura = surface_create(view_wview,view_hview);
    surface_copy_part(captura,0,0,application_surface,0,0, view_wview,view_hview);
    instance_deactivate_all(true);
    pausado = true;
}
```

---

3 - para dibujar esta capa en la superficie de aplicación y encima el texto de pausa, en su evento Draw añadimos:

```
if (surface_exists(captura) and pausado == true){
    draw_rectangle(0,0,room_width,room_height,false);
    // La appsurface mide lo mismo que la view
    draw_surface(captura,view_xview,view_yview);
    draw_set_font(fnt_nivel);
    draw_set_color(c_white);
    draw_set_halign(fa_center);
    draw_set_valign(fa_middle);
    draw_text((view_xview+view_wview/2),(view_yview+view_hview/2)+225,"PAUSA");
    draw_set_color(c_black);
```

}

4- Por último rematamos la faena creando un evento al apretar la tecla espacio que borrará el objeto pausa, pero antes pondrá todo en funcionamiento de nuevo:

```
instance_activate_all();
audio_resume_all();
instance_destroy();
```

Y hacemos que desde el objeto_jugador, al apretar el espacio creamos este objeto_pausa:

```
/// Creamos una pausa
instance_create(0,0,obj_pausa);
```

Clase 23: Jugabilidad - Añadir golpe cuerpo a cuerpo

1- Si he dejado para el final esta opción es para mostraros cómo añadir una nueva funcionalidad sin tener que modificar considerablemente nuestro juego. Como programador y desarrollador de software, os quiero transmitir la idea de la importancia de mantener nuestros proyectos ordenados y bien diseñados, pues la gracia de un programa o videojuego es poderlo mejorar gradualmente sin tener que alterar el comportamiento inicial.

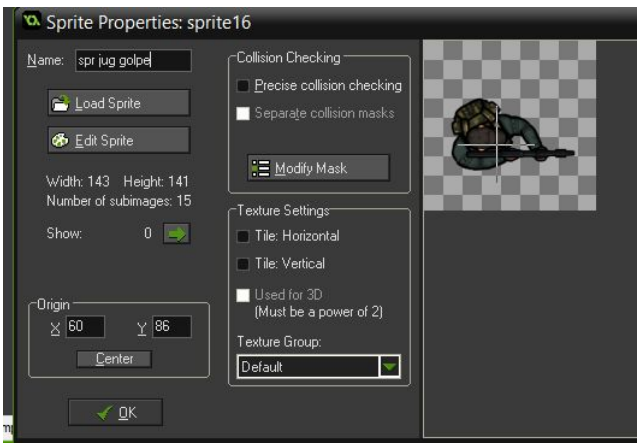
2- Vamos a poner en perspectiva lo que queremos hacer:

- Gestionar el momento del golpe cuerpo a cuerpo
- Gestionar la animación cuerpo a cuerpo
- Gestionar la colisión del golpe cuerpo a cuerpo
- Gestionar los efectos de sonido

¿Cómo añadimos esta funcionalidad al jugador? Pues gracias a nuestra implementación por estados podemos añadir un nuevo estado golpe al jugador. Este gestionará la animación y también se encargará una colisión cuerpo a cuerpo, prácticamente igual que hicimos con el golpe del zombie. Vamos a ponernos a ello.

3- Vamos a duplicar el objeto obj_ataque_zombie a obj_ataque_jugador.

4- Añadimos el nuevo Sprite de golpe:



5- Creamos el nuevo estado:

```
enum ej {
   parado        =  jugador_parado,
   movimiento  =  jugador_movimiento,
   disparar      =  jugador_disparar,
```

```
    golpear      =  jugador_golpear
}
```

6- Y el Script:

```
image_speed = 0.50;
sprite_index = spr_jug_golpe;
```

7- Ahora vamos a capturar el boton derecho para saber si el jugador quiere atacar. Podremos atacar siempre que no estemos ya atacando:

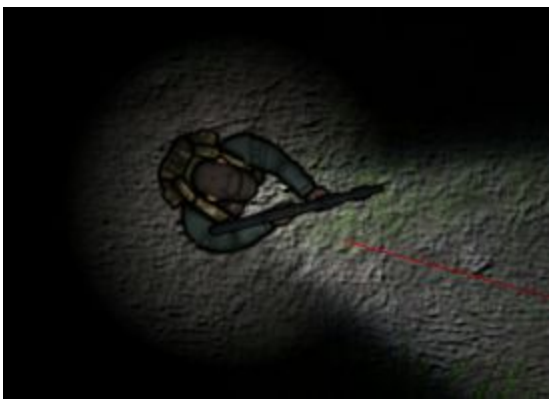
```
if (mouse_check_button(mb_right) && estado != ej.golpear){
    image_index = 0;
    estado = ej.golpear;
}
```

8- Controlamos los estados con la nueva posibilidad de golpear:

```
/// Control de estados
if (!ej.disparar and !ej.golpear){
    if (vh != 0 or vv != 0) estado = ej.movimiento;
    if (vh == 0 and vv== 0) estado = ej.parado;
}
```

9- Y también controlaremos si ha finalizado la animación de golpe, para volver a poner el estado a parado en el Animation End:

```
/// Detectar final del golpe
if (estado == ej.golpear) estado = ej.parado;
```



10- Ya nos funciona el golpe, pero ha ocurrido un bug y es que si apretamos el botón derecho y el izquierdo a la vez la metralleta funciona mal y dispara muchas balas. Para arreglarlo tenemos que comprobar antes de disparar que el estado actual no sea golpear, en el objeto_arma:

```
/// Detectamos si el jugador dispara
if (mouse_check_button(mb_left) && obj_jugador.estado != ej.disparar && 
obj_jugador.estado != ej.golpear){
```

También cancelamos el sonido de la metralleta más abajo cuando estemos apretando el botón derecho de golpear:

```
if (!mouse_check_button(mb_left) or mouse_check_button(mb_right)){
    audio_stop_sound(snd_disparos);
}
```

11- Otro pequeño detalle es que tendríamos que desactivar el láser mientras golpeamos, sólo tendremos que añadir una comprobación en el Draw del láser:

```
/// Dibujar laser
if (obj_jugador.estado != ej.golpear) {
   draw_set_color(c_red);
   draw_set_alpha(0.75);
   draw_line(obj_arma.x,obj_arma.y,x,y);
   draw_set_color(c_black);
   draw_set_alpha(1);
   draw_self();
}
```

12- Ya lo tenemos, ahora nos falta crear la colisión. Lo podemos hacer parecido a como hicimos el golpe del Zombie. En el script de jugador_golpear, en el instante que la subimagen esté totalmente atacando crearemos una instancia de golpe de jugador pero tenemos un problema y es que a diferencia de antes no sabemos a qué zombie estamos atacando, por tanto lo tendremos que encontrar.

Vamos a trazar una línea de colisión en busca de un zombie a un rango máximo de 80px en la dirección que mira el jugador. Si encontramos uno, tomaremos ese como referencia y crearemos la colisión encima suyo:

```
// Detectar subimagen de la colisión
if (image_index == 8){
   zombie =
collision_line(x,y,x+lengthdir_x(80,dir),y+lengthdir_y(80,dir),obj_zombie,false,false);
   if (zombie){
```

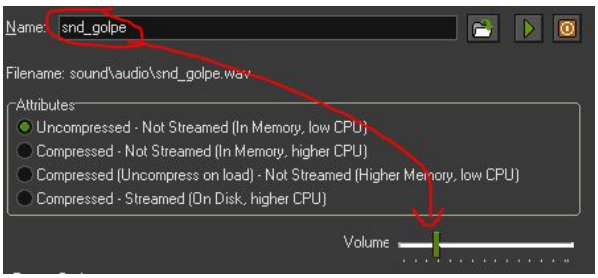
```
        instance_create(zombie.x,zombie.y,obj_ataque_jugador);
        //audio_play_sound(snd_zombie_ataque,70,false);
    }
}
```

Podemos hacer visible el obj_ataque_jugador para comprobar si funciona, con una profundidad bastante baja -150:

13- Ahora crearemos la colisión con este objeto en el zombie y le restamos vida antes de borrar la instancia y también mostramos la barra del zombie y algunas partículas sobre él, igual que hicimos con las balas:

```
ataque_jugador = instance_place(x,y,obj_ataque_jugador);
if (ataque_jugador) {
    with(ataque_jugador) instance_destroy();
    // Restamos algo de vida al zombie
    vida -= 25;
    // Mostramos la barra
    barra_alpha = 0.7;
    alarm[1] = room_speed/15;
    // Y creamos unas partículas
    with(obj_laser){
        part_type_life(particula,20,60);
        part_type_color1(particula,c_red);
        part_type_direction(particula,0,50,0,0);
        part_emitter_region(sistema,emisor,x,x,y,y,0,0);
        part_emitter_burst(sistema,emisor,particula,12);
    }
    instance_destroy();
}
```

14- El resultado es espectacular, sólo nos falta añadir un sonido al golpe y fin:




```
if (zombie){
	instance_create(zombie.x,zombie.y,obj_ataque_jugador);
	audio_play_sound(snd_golpe,70,false);
}
```

Felicidades, ya hemos acabado :)