PROYECTO DE ADV

Recreación de un proyecto de juego estilo "Bullet Hell"

Curso 2k22/2k23

Apellidos, nombre	**Cabrera Miñano, Christian** (chcabmia@inf.upv.es)
Titulación	Grado de Ingeniería informática
Fecha	Mayo de 2023

ÍNDICE

Índice de figuras

Índice de pseudocódigo

1 Resumen de las ideas clave

En este documento se describirá detalladamente el proceso de realización de este proyecto, con el fin de mostrar el progreso que se ha tenido de éste a lo largo de la asignatura, presentar problemas y posibles soluciones, e intentar que sirva como ayuda y/o fuente de ejemplos para la realización de cualquier otro trabajo o proyecto de asignatura que realicen grupos de años posteriores.

2 Introducción

Este proyecto es llevado a cabo teniendo como inspiración el **género** de videojuegos conocido popularmente [1] como: **"Bullet Hell", "Matamarcianos", "Shot 'em up", etc..**

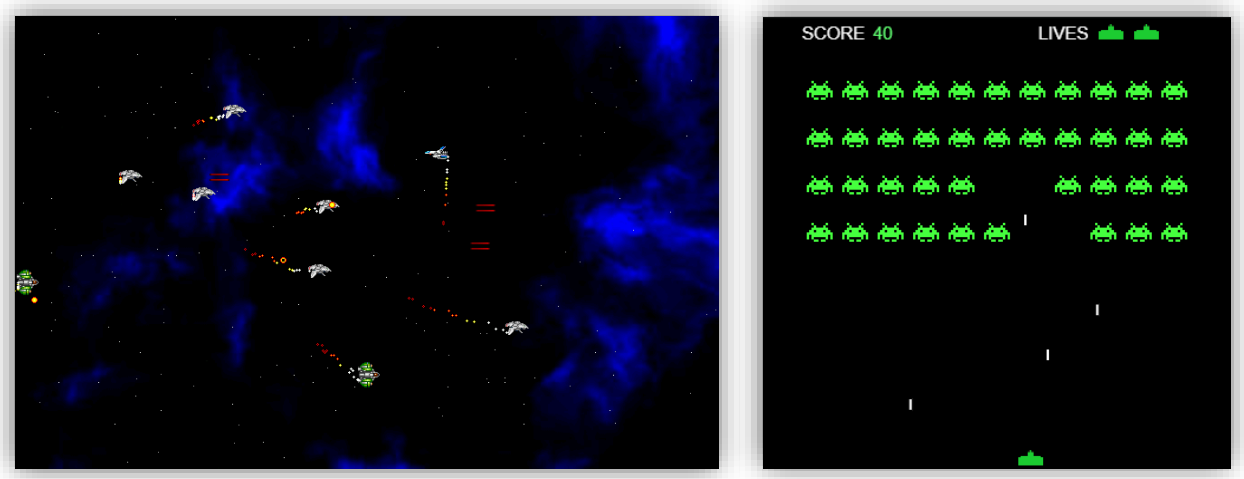


***Figura 1 y 2**: A la izquierda, una captura de pantalla del videojuego "Project Starfighter" y a la derecha, otra captura de pantalla, pero del videojuego "Space Invaders".*

En resumen, un estilo de juegos que se basan en el control de un personaje (normalmente en un escenario 2D) y cuyo objetivo es generalmente derribar a los enemigos a la vez que se esquivan los patrones de proyectiles que ellos lanzan sobre ti. Además, para hacer más entretenida la experiencia de juego, se suelen añadir diversas **mecánicas**, como, por ejemplo: la inserción de habilidades especiales que tu personaje podrá usar en un determinado momento para ayudarse en la pelea, objetos que podrán tener efectos como: curar al personaje, aumentar puntos de algún atributo, etc…

2.1 Inspiración del Proyecto

El videojuego que se ha ido desarrollando a lo largo de la asignatura está **inspirado en** un proyecto ya existente llamado "**Touhou Project**". [2]

Esta saga de videojuegos, creada por la desarrolladora japonesa: "**Team Shangai Alice**" [3] *(que está compuesta por una sola persona, que realiza las labores de: desarrollador, compositor, guionista, etc…)* que lanzó el primer videojuego de la saga en el año 1997 y continúa lanzando títulos en la actualidad, con un total de aprox. 31 *(de los cuales unos 18 pertenecen a la categoría anteriormente mencionada)*.

Dicho esto, cabe aclarar que el proyecto que se va a llevar a cabo en la asignatura constará de diversas similitudes respecto a los títulos de esta saga que corresponden con el género previamente mencionado, pero sin ser exactamente igual, ya que se sustituirán diversas mecánicas con el objetivo de crear un proyecto algo original, dentro de la similitud que tendrá con estos títulos.

***Figura 3**: Duelos entre las protagonistas principales de "Touhou Project"*

(abajo se encuentra el personaje jugable) – Touhou 9 'Imperishable Night'

2.2 Motivación

Como motivación para escoger la saga anteriormente mencionada como ejemplo a seguir para el proyecto, ha sido la **sinergia** entre el estilo de jugabilidad que tiene "Touhou", y la doble pantalla que ofrece la Nintendo 3DS (que simula una pantalla de 400 píxeles de ancho de la pantalla superior, que es la más ancha, y 480 píxeles de alto, que sería la suma de las alturas de las dos pantallas sin contar el espacio que las separa, es decir, quedaría en **400 x 480**).

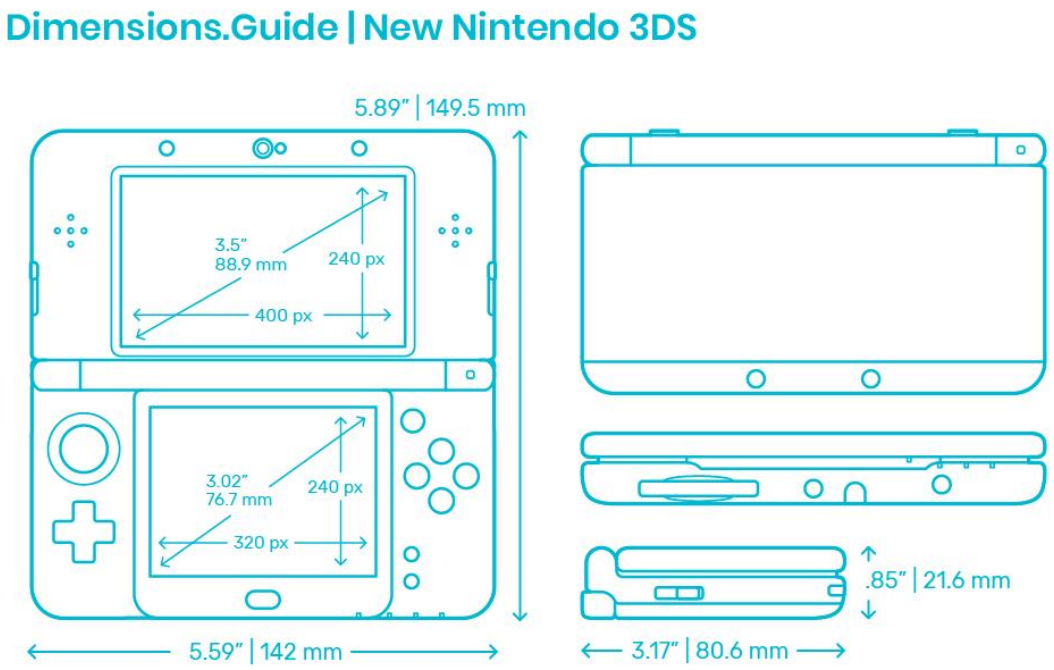


***Figura 4**: Plano de las dimensiones de una "New Nintendo 3DS".*

Dicho esto, se pretende adaptar este tipo de jugabilidad a la Nintendo 3DS utilizando sus 2 pantallas al mismo tiempo, y limitando la movilidad del personaje jugable a únicamente a la zona que correspondería con la pantalla inferior de la consola.

***Figura 5**: Prototipo de lo que se pretende lograr en este proyecto y que sirve como motivación de este.*

3 Objetivos

Los objetivos que se proponen a la hora de realizar el proyecto son los siguientes:

- Ser capaz de utilizar correctamente la funcionalidad anteriormente descrita de utilizar las 2 pantallas.
- Incluir diversas mecánicas que hagan de la experiencia de juego más entretenida.
- Añadir elementos gráficos y sonoros para lograr generar un buen ambiente.

4 Planificación

La planificación se ha llevado a cabo de la siguiente manera:

1. Preparativos 07/02/2023 - 14/02/2023 *7 días*

1.1. Pensar en la idea

1.2. Preparación de recursos iniciales

2. Preparativos 15/02/2023 - 23/03/2023 *36 días*

2.1. Personaje Manejable (Sprite, movimiento, disparo, etc…)

2.2. Fondo de escenario

2.3. Lógica de Proyecto

2.3.1. Disparos

2.3.2. Enemigos

2.3.3. Vidas, potenciadores, objetos, etc…

2.4 Pantalla de Inicio, derrota, créditos, etc…

2.5 Asignación completa de sprites/imágenes

2.6. Música y Efectos de Sonido

3. Trabajo Extra Fundamental 24/03/2023 - 09/04/2023 *16 días*

3.1. Partículas

3.2. Acabado del Nivel

3.3. IGU

4. Trabajo Extra Opcional 10/04/2023 - 03/05/2023 *23 días*

4.1. + Enemigos

4.2. + Niveles

4.3. + Patrones de Disparo

TOTAL - **82 Días**

5 Desarrollo

5.1 Herramientas de Desarrollo

Antes de nada, describir las herramientas con las que trabajaremos a lo largo del proyecto:

***Figura 6**: Logos de izquierda a derecha; Citra, DevKitPro y C / C++.*

En primer lugar, utilizaremos el **lenguaje de programación C**, o lo que será más recomendable, **C++**. Esto se debe a que ciertas librerías utilizadas para el desarrollo del proyecto (cwav [4], por ejemplo) requieren el uso de funciones que únicamente son posibles de ejecutar si se está usando C++.

Lo siguiente será instalar las librerías que nos ofrece **DevKitPro**[5], un grupo que se dedica a desarrollar y proveer de ciertas herramientas para desarrollo de consolas (Wii, GBA, Gamecube, Switch, etc...). Para la instalación de estas herramientas, al inicio del curso se proporcionaran los recursos necesarios; aun así, podéis encontrar algunos en los siguientes enlaces en las referencias ([99]).

Por último, instalaremos **Citra** [6], (en mi caso, "Nightly" v. 1850) que se trata de un emulador de Nintendo 3DS, y que podréis descargar desde el enlace de referencia. En él podremos arrancar el ejecutable que compilemos utilizando las herramientas anteriores.

*** *En mi caso, a la hora de ejecutar mi proyecto en Citra, éste se ejecutaba muy lentamente (a pocos FPS y mucho tiempo de respuesta) debido a lo que yo creo que es la inserción de sprites o imágenes de gran tamaño (solo ocurría cuando añadía, p. ej. Un fondo de pantalla [img. de 400 x 240])*.

Logré solucionarlo accediendo a Emulación -> Configurar -> Ok (botón).

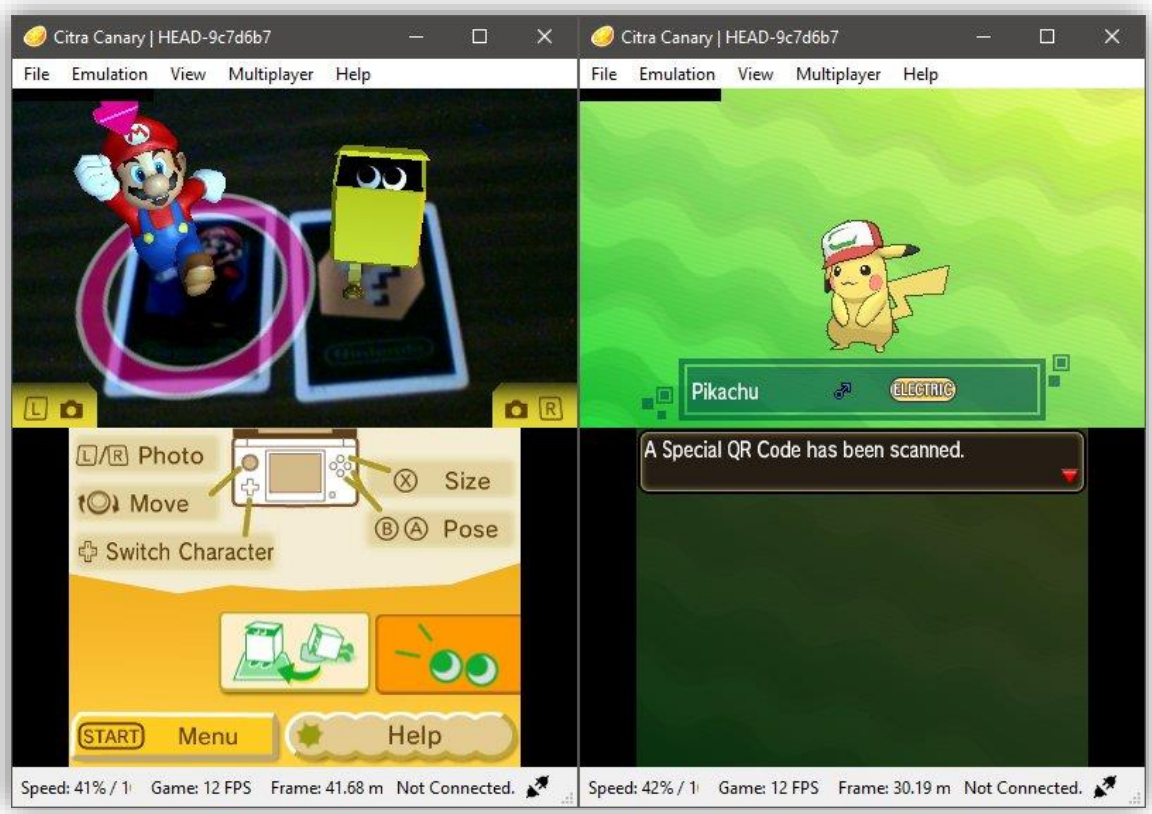


***Figura 7**: Ejemplo de la interfaz que ofrece Citra.*

5.2 Descripción de Retos y Soluciones (si constan)

5.2.1 Retos Sencillos

Entender cómo funciona la ejecución de código que se utilizará para que nuestro proyecto funcione. Lograr esto nada más empezar puede ser algo complicado o que lleve algo de tiempo, ya que tendrás que acostumbrarte a como se manejan las funciones graficas y a donde colocar tus funciones y manejo de datos para que la ejecución del programa funcione correctamente.

Por eso he decidido hacer un mini-resumen de como sería aproximadamente la jerarquía del fichero principal (añadiendo las funciones necesarias posteriormente respecto a las necesidades de cada proyecto):

Librerías

Archivos de cabeceras (.h)

Variables Iniciales

FUNCIONES

Método Main

Código que sólo se ejecuta 1 Vez al arrancar el programa

(carga de sprites, inicialización de posiciones de personajes u objetos, inicialización de variables, funciones que sólo se necesiten ejecutar al principio, comprobaciones, etc . . .)

While

Código que se ejecuta constantemente

(RECONOCIMIENTO DE TECLAS Y BOTONES Y LÓGICA DEL VIDEOJUEGO)

```
// RENDER DE LA PANTALLA INFERIOR, DESPUÉS DE . . .
C3D_FRAMEBEGIN ( C3D_FRAME_SYNCDRAW );
C2D_TARGETCLEAR ( BOTTOM, . . . );
C2D_SCENEBEGIN ( BOTTOM );

C2D_FLUSH( );
// RENDER DE LA PANTALLA SUPERIOR, DESPUÉS DE . . .
C3D_FRAMEBEGIN ( C3D_FRAME_SYNCDRAW );
C2D_TARGETCLEAR ( TOP, . . . );
C2D_SCENEBEGIN ( TOP );

C3D_FRAMEEND( 0 );
```

Otro problema que veo necesario comentar sería el de la **eliminación de objetos** (imágenes, sprites u objetos que contengan imágenes/sprites) **de la pantalla**, para así no sobrecargar la memoria de video.

Para ello, se utilizarán dos formas diferentes en función de lo que se quiera borrar (o simplemente no dibujar en pantalla):

- **Únicamente sprites:** Para ello hay que tener en cuenta que un Sprite es dibujado en pantalla mediante la función "C2D_DrawSprite(&sprite)" y que si no se quiere dibujar el sprite, simplemente insertar esta función dentro de alguna condición que no se cumpla, como por ejemplo:

  If (item.active)
  C2D_DrawSprite(&item);

- **Objetos con sprites:** Este caso depende de como tú definas estos objetos. En mi caso, un objeto tiene un sprite *(C2D_Sprite *sprite)* asociado y una variable booleana que indica si el objeto está activo o no, y en mi proyecto se ha añadido una función *(que se coloca en la zona donde se renderiza la pantalla)* que dibuja los sprites de los objetos si estos están activos.

```
void draw_minions_top(void){

int i = 0;
for (; i < MAX_MINIONS; i++) {
        if (minions[i].state) { // not inactive
                if(minions[i].top == true){
        C2D_SpriteSetPos(minions[i].enemy_spr, minions[i].x, minions[i].y);
        C2D_SpriteSetRotation(minions[i].enemy_spr, deg_to_rad(-minions[i].angle + 90));
        C2D_DrawSprite(minions[i].enemy_spr);
                }
        }
}
}
```

Fragmento sacado del código de Ast3roiDS[7] proyecto de compañeros de años anteriores.

Si el estado alguno de los objetos del array está activo, automáticamente las funciones dentro del condicional se activarán, y dibujarán sólo este objeto.

Sabiendo esto, el último paso será crear una función que compruebe cuando un objeto no está en pantalla, para así borrarlo. Eso lo haremos utilizando la siguiente función:

```
inline int inside_rect(float x, float y, float leftx, float rightx, float downy, float upy)
{
  return x > leftx && x < rightx && y > downy && y < upy;
}
```

Fragmento sacado del proyecto mencionado anteriormente: Ast3roiDS[7]

En la que se introducirán la posición de los objetos existentes y que devolverá un valor de 'false' si alguno de los objetos sale por pantalla *(nótese que se puede modificar el área del rectángulo que se describe),* por lo tanto, **desactivando el objeto**.

```
if( ! inside_rect(bullets[i].x, bullets[i].y, -20, 420, -50, 500)  // x_izq, x_der, y_arriba, y_bajo
    && bullets[i].bot_screen == false) // TOP
    {
        bullets[i].state = BULLET_STATE_INACTIVE;
    }
```

PD: Al tratarse de los proyectiles que aparecen en la pantalla superior, introduzco las dimensiones de esta pero extendiéndola, para que así los proyectiles más grandes no desaparezcan cuando lleguen a ese punto.

Por último dentro de la categoría de retos sencillos, quiero destacar el problema de las colisiones entre los proyectiles y los enemigos y el jugador.

Estas colisiones se llevarán a cabo mediante una función que comprobará si el radio de colisión de la entidad (enemigo, jugador, etc…) ha colisionado con el radio de colisión de un proyectil; pasándose como argumentos los dos radios, y las posiciones (x, y) de los objetos.

```
CIRCLE_INSIDE_CIRCLE ( X_BULLET, Y_BULLET, X_OBJ, Y_OBJ, BULLET_R, OBJ_R )
{
RETURN SQRT( (X_OBJ – X_BULLET)^2  + (Y_OBJ – Y_BULLET)^2) < BULLET_R + OBJ_R
}
```

Esta sería la formula que se usaría, que es una pequeña modificación que realicé de la siguiente formula:

```
INSIDE_CIRCLE ( X_BULLET, Y_BULLET, X_OBJ, Y_OBJ, BULLET_R)
{
RETURN SQRT( (X_OBJ – X_BULLET)^2  + (Y_OBJ – Y_BULLET)^2) < BULLET_R
}
```

// **Fragmento sacado del proyecto mencionado anteriormente: Ast3roiDS**[7]

5.2.2 Retos Intermedios

En este apartado estarán los retos que en mi caso han costado algo más, ya sea por la **dificultad de plantearlos**, lo **complicado** que es **encontrar la información**, o por la **complejidad del código** (o todas).

En primer lugar, estaría la introducción de **audio**.

Aunque al principio intenté incorporar las instrucciones para utilizar "libopusfile", al final me decanté por la librería "**cwav**" [4] , y por lo tanto, a cambiar el lenguaje de programación a C++, por lo que **si estás pensando en utilizar esta librería, haz el cambio lo más pronto posible** para paliar en la medida de lo posible los problemas que conllevan cambiar de lenguaje.

Dicho esto, si quieres instalar las librerías necesarias, dirígete al enlace que te he proporcionado, que te llevará al repositorio de **GitHub** de **PabloMK7** y sigue las instrucciones que allí se encuentran (en las cuales se encuentra la descarga de otra librería llamada "**libncsd**" [8] y una herramienta llamada "**cwavtool**" [9] para convertir algunos tipos de archivos de audio al formato que se usará en la librería "cwav" *(en el repositorio de cwav también podrás encontrar estos enlaces junto a las instrucciones*).

En segundo lugar, estaría la **animación de los sprites**.

Ya que este aspecto es clave en la mayoría de aspectos de desarrollo de videojuegos en 2D, me planteé resolver este caso lo más pronto que pude. Para esto, y después de pedir consejo al profesor de la asignatura, se nos presentó la solución que podemos encontrar en este repositorio de **GitHub** de **NyankoTear**, llamado **reimu_idle** [10]. En este proyecto se ofrece una buena demostración de cómo realizar fácilmente la animación de un objeto que contenga diversas imágenes, aunque con **un pequeño** (en verdad no tan pequeño) **problema**, y es que en mi caso no supe crear diversas instancias de estos objetos *(se podía, pero de alguna forma era incompatible y un objeto influía en otro, por lo que se descartó hacerlo así)*, por lo que la opción que me quedaba era crear un objeto entero desde cero para cada, por ejemplo pequeño enemigo *(cosa inpensable)*. Debido a esto, acabé utilizando éste método para pocos casos, como por ejemplo, la animación del jugador *(que sólo hay 1)*.

(y algunas cosas más que se quedaron en el código pero no utilicé, como la barrera de los jefes, el jefe en cuestión y algo más que probablemente se implementará en un futuro después de la realización de la asignatura)

5.2.2 Retos Complicados

En este apartado contaré detalladamente en qué consisten los **problemas más duros** a los que me he enfrentado a la hora de desarrollar el proyecto.

En primer lugar, estaría la **creación de los objetos** (que no sería lo mismo que dibujarlos, explicado anteriormente).

Hay que recordar que **estamos trabajando en un proyecto que utiliza las dos pantallas al mismo tiempo**, por lo que implementar una solución para este problema era fundamental. Al principio pensé en crear una función que dibujase los sprites dependiendo de la distancia a la pantalla a la que se dirigían, pero resultó ser algo ineficiente.

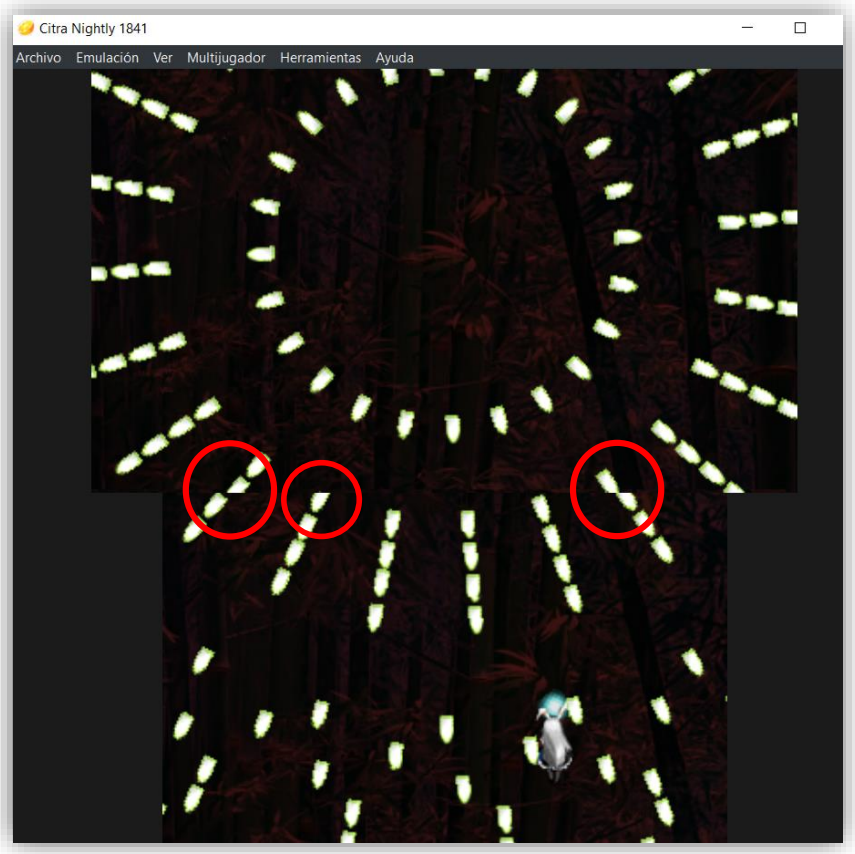


***Figura 8**: Prototipo temprano de un nivel de "Blue Invaders".*

Una vez descartada la idea anterior de como crear los objetos, se me ocurrió otra distinta que dio un **mejor resultado**, y que consistiría en crear un bucle que compruebe si el objeto está activo; si lo está, continuar buscando en el array de objetos hasta encontrar uno que no lo esté, y una vez encontrado, crearlo y romper el bucle tal que así:

```
FUNCIÓN CREA SPRITES ( X, Y, Vx, Vy, HP, Sprite, etc... ){

        FOR ( i = 0  |  i < MAX_SPRITES / 2  | i++ )
                IF ( OBJETO NO ESTÁ ACTIVADO )
                        CREAR ( . . . , BOT )

        BREAK

        FOR ( i = MAX_SPRITES / 2  |  i < MAX_SPRITES  | i++ )
                IF ( OBJETO NO ESTÁ ACTIVADO )
                        CREAR ( . . . , TOP )

        BREAK
}
```

Además, se puede observar que **se han creado dos bucles distintos**, que serían uno para trabajar con los sprites de la pantalla superior, y otro con los de la pantalla inferior.

Una vez que se detecta un objeto que no está activado, este se crea *(por lo que se activa)* y sale del bucle para continuar con el siguiente, con la peculiaridad de que el siguiente empieza por la mitad de la capacidad del array y termina al final, lo que significa que para cada objeto que se cree, se creará otro pero con el identificador sumado por la mitad de la capacidad del bucle. **Explicado de manera fácil**: para MAX_SPRITES = 100 , **si** un **objeto no existe** *(objeto[5])*, **se crea** ese objeto para dibujar en TOP, y **posteriormente se crea otro** *(objeto[55])* para ser dibujado en la pantalla BOT; además, habrá **otra función que borre esos objetos**, por lo que **cuando objeto[5] sea borrado**, **también lo será objeto[55],** lo que permitirá el reciclaje de objetos y la consistencia de la creación de estos gracias a esta función.

Dicho código **produjo buenos resultados**, exceptuando varios casos aislados donde entran en pantalla sprites que no han pasado por la pantalla que deberían de haber pasado antes *(pocos casos pero ha ocurrido)*, probablemente debido a alguna otra función. Aun así, **los resultados de aplicar esta función aportan mejoras notables a la anterior**, a pesar de que tal vez sea una manera **algo ineficiente** para resolver este problema, ya que, al fin y al cabo, estamos creando sprites que realmente no estamos viendo.

***Figura 9**: Prototipo final de un nivel de "Blue Invaders".*

Y ahora sí, después de aplicar lo anteriormente explicado, podemos ver como el problema de los sprites que se entrecortaban se ha resuelto de manera exitosa, excluyendo los pequeños casos en los que aparecen sprites *(siempre proyectiles, por lo que puede ser causa de otra función relacionada con estos, pero lo desconozco)*.

El siguiente problema complicado del proyecto ha sido la **gestión del tiempo en el que ocurren los eventos**.

Durante el transcurso del nivel, van ocurriendo diversos eventos (aparición de enemigos, disparo de proyectiles por parte de estos y su posterior salida por pantalla). Estos eventos transcurren después de pasado un tiempo, pero: **¿Cómo hacemos esto?**

Yo opté por utilizar la función **OsGetTime( )**, que literalmente **devuelve el tiempo transcurrido desde un suceso** bastante lejano en **ms** *(cosa que no nos importa, ya que nosotros calcularemos la diferencia)*.

```
t  -  TIEMPO T = OsGetTime( )

t2 - TIEMPO TRANSCURRIDO ( x ms ) = OsGetTime( ) - TIEMPO T ( t hace x ms )


FUNCIÓN-EVENTO-DESPUÉS-DE ( TIEMPO T, T ( ms ) )
{
        IF ( OsGetTime( ) - TIEMPO T  >= T )
                True
        ELSE
                False
}
```

Como se puede ver en el pseudocódigo proporcionado, **crearemos una función que devuelva un valor booleano dependiendo de si el tiempo que queremos que transcurra ha pasado o no**. Después de que el tiempo haya pasado y la función devuelva el valor 'true', podremos entrar dentro del condicional en el cual se encuentre *(o en su defecto utilizar [! EVENTO-DESPUES-DE] para que una vez pasado x tiempo que no ocurra lo que el condicional contenga)*. **Eso sí**, teniendo en cuenta que si queremos que el evento ocurra de nuevo pasado x tiempo, actualizar dentro del condicional la variable TIEMPO T, para que así vuelva a tener que esperar x tiempo para ejecutarse y que no se ejecute constantemente *(ya que como el tiempo ya ha transcurrido, se entrará en el condicional permanentemente)*.

Este problema aunque parezca fácil de resolver, ha sido todo un rompecabezas el tratar de plantear una solución consistente a esto, ya que se debía de pensar en una forma para gestionar múltiples eventos, ya que no podemos confiar en una sola variable de tiempo para todo el proyecto porque los sucesos ocurren simultáneamente uno tras otro.

Y dicho esto, procederé a explicar el mayor problema que ha tenido a la hora de realizar este proyecto sin duda alguna, y que viene relacionado con el punto que acabamos de hablar: **el tiempo**.

Este proyecto tiene ciertas mecánicas que describiré más adelante, en concreto, en el punto Descripción y Guía del juego, y para este problema entra una de ellas: la mecánica de "**Parada de Tiempo**" *(TS a partir de ahora "Time Stop")* que resulta ni más ni menos que la habilidad del personaje.

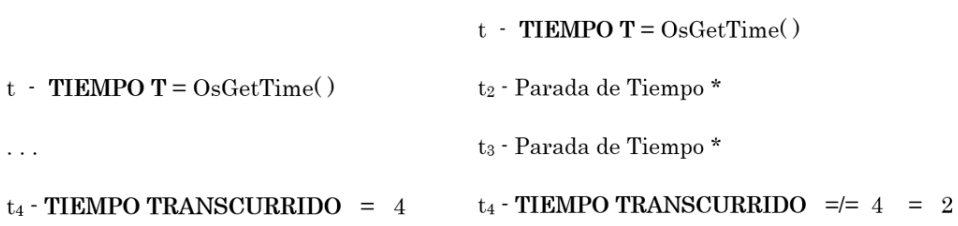
Y como tal vez puedas imaginar, ese es precisamente el problema principal: **tratar de que los eventos sigan ocurriendo correctamente** a pesar de que en un determinado momento el tiempo "se pare".

Lo que se pretendía al principio es que, por ejemplo:

1. Enemigo se mueve durante 2 seg. hacia el punto medio de la pantalla superior.
2. El enemigo se para y comienza a disparar durante 2 seg.
3. El enemigo se dirige hacia fuera de la pantalla para desaparecer durante 2 seg.

Pero si se utilizaba el **TS** *(que siempre dura alrededor de 1 seg.)* **al inicio de este evento**, lo que de verdad ocurría era que:

1. Enemigo se mueve durante **1 seg**. hacia **un punto más alto al punto medio** de la pantalla superior.
2. El enemigo se para y comienza a disparar durante 1 seg.
3. Se utiliza **TS**.
4. El enemigo no dispara más.
5. El enemigo se dirige hacia fuera de la pantalla para desaparecer durante 1 seg.
6. Se utiliza **TS**.
7. El enemigo queda al borde de la pantalla permanentemente.



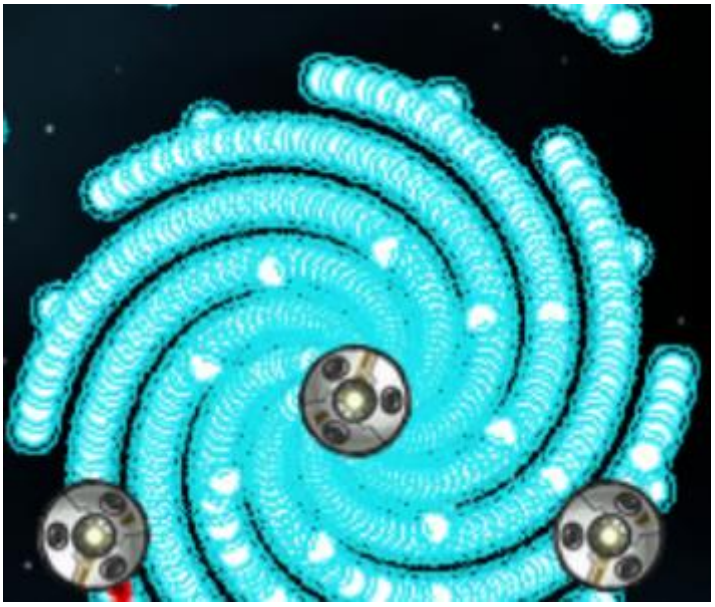


***Figura 10**: Fallo en un nivel de "Blue Invaders" debido al problema del TS.*

Este tipo de comportamiento es intolerable si se pretende que haya consistencia en los eventos que ocurren, por lo que necesitaba plantear una solución a todo costo.

Para ello, se propuso la siguiente solución: crear una **variable que almacene el tiempo transcurrido durante la ejecución del TS** y cuando este acabe, **aplicarla al calculo total del tiempo** que se le pasa a la función para calcular la diferencia de tiempo anteriormente mencionada.

t - **TIEMPO T** = OsGetTime( )

t_2 - Parada de Tiempo * **TIEMPO PARADO ++**

t_3 - Parada de Tiempo * **TIEMPO PARADO ++**

t_4 - **TIEMPO TRANSCURRIDO** - **TIEMPO PARADO** = $4 - 2 = 2$

APAÑO

IMPORTANTE !

Acumular los valores del Tiempo Parado únicamente cuando sigamos utilizando la variable Tiempo T, si ésta se reinicia, se debe reiniciar también Tiempo Parado, ya que si no se hace afectará al Tiempo Transcurrido adelantándolo.

Una vez explicado el concepto, la forma de la función resultante **quedaría** de tal y como se muestra en la siguiente imagen del **pseudocódigo**, y se **comportaría** como muestra la **Figura 11**:

```
FUNCIÓN-EVENTO-DESPUÉS-DE ( TIEMPO T, T ( ms ) )
{
        IF ( OsGetTime( ) - TIEMPO T  >= T )
                True
        ELSE
                False
}
```

APAÑO FINAL

```
FUNCIÓN-EVENTO-DESPUÉS-DE ( TIEMPO T + TIEMPO PARADO, T ( ms ) )
{
        IF ( OsGetTime( ) - ( TIEMPO T + TIEMPO PARADO )  >= T )
                True
        ELSE
                False
}
```

***Figura 11**: Fallo resuelto en un nivel de "Blue Invaders" debido al problema del TS.*

Y haciendo esto **lograremos solucionar** este problema **en cierta medida**, y digo esto debido a que hay ciertos casos en los que la ejecución de esta función no es la que se espera por razones que desconozco; no de forma tan grave como lo que muestra la **Figura 10**, pero lo suficiente para que no se adecue a lo que realmente se pretende que ocurra. Y es por esto por lo que, a día de hoy, este sigue siendo el reto a resolver completamente más difícil de todo el proyecto *(para mi)*.

6 Descripción y Guía del juego

En este apartado describiremos el nivel del tutorial del videojuego, para así aprender cuales serán las mecánicas principales y los puntos clave a tener en cuenta, a la vez que se muestran fragmentos de como ha quedado el proyecto.

Menú principal

***Figura 12**: Menú principal del proyecto "Blue Invaders".*

Antes de entrar al tutorial, enseñar cómo se accede a él. Esta es **la primera pantalla que verás al iniciar el juego**; aquí se muestran los modos a los que se puede acceder: el nivel principal (cohete), el tutorial (pizarra) y los créditos (libro).

Cada vez que se cambia de escena, se aplica un efecto de aparición progresiva en la pantalla ("fade in / fade off").

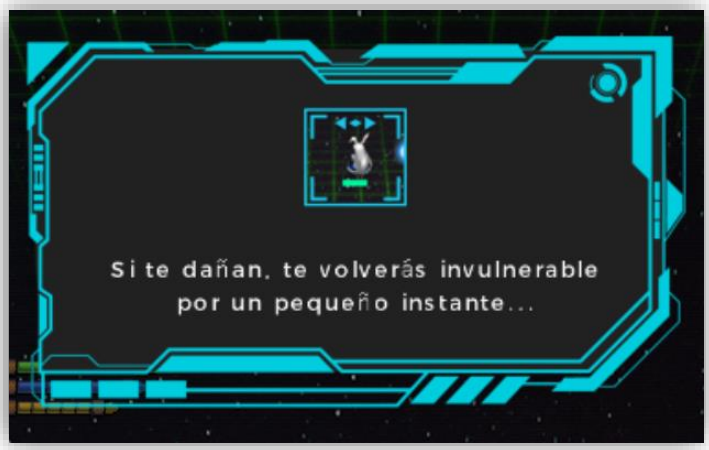
Ahora sí, al acceder al tutorial, lo primero que nos muestran sería:

La **barra de Puntos de Salud (PS),** que determina la vida del personaje.

Una vez llegue a 0, perderás la partida, viendo la pantalla de "Game Over" y regresando al menú principal para así elegir alguna de las opciones que se encuentran en éste *(no puedes perder en el tutorial)*.

Cabe destacar que al inicio del tutorial, muchas de las mecánicas están bloqueadas, y se irán desbloqueando a medida que se vaya avanzando en él, junto con su explicación.

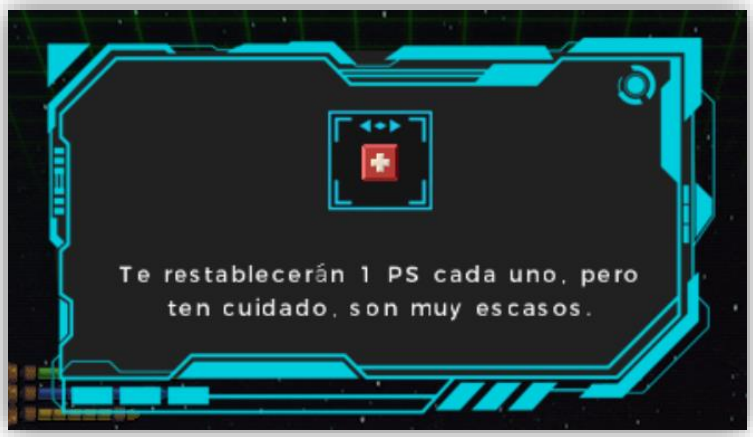
A continuación, se muestra la **barra de invulnerabilidad** *(mecánica no incluida en la saga de juegos en la que se basa el proyecto, pero que he considerado conveniente)*.



Esta barra muestra el tiempo que resta para que los proyectiles vuelvan a poder dañarnos, y que nos permitirá hacernos una idea del tiempo que tenemos para reposicionarnos entre los proyectiles que nos están disparando.

Ahora hablaremos de los **botiquines,** que simplemente te restaurarán 1 PS cada vez que colisiones con uno.

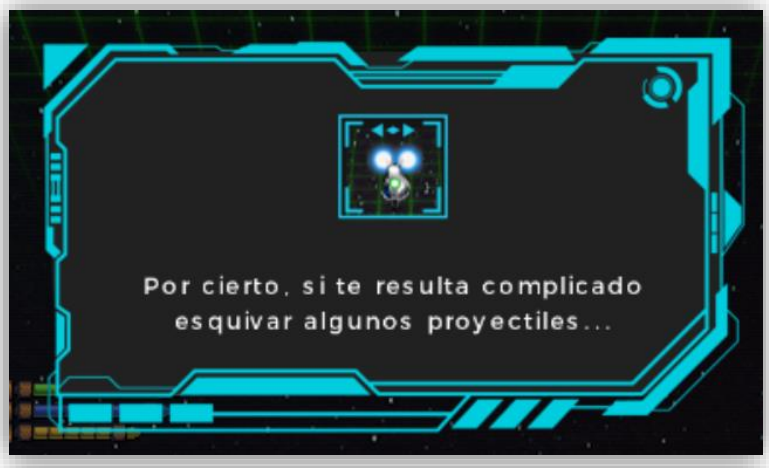
En la implementación del proyecto se pretende que no hayan muchos, ya que éste no constará de la dificultad que tienen los títulos en los que se basa el proyecto.

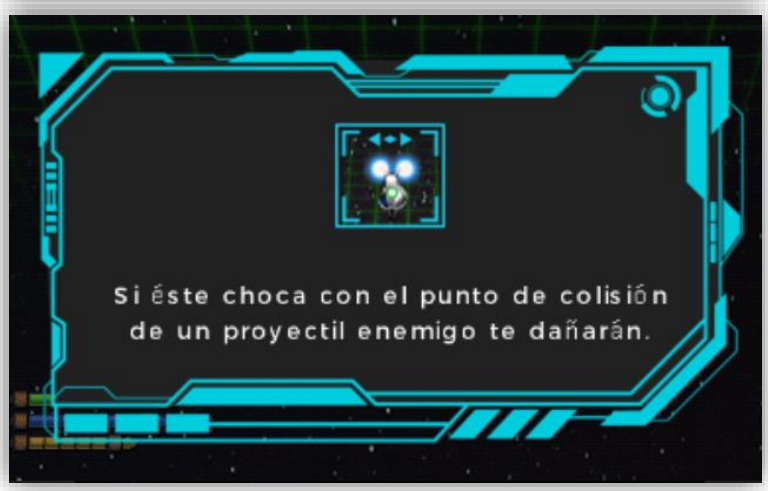


Además, se pretendía agregar una pequeña animación para cuando se recoge uno de estos, pero que al final no pudo realizarse. Aun así, se realizará en un futuro, ya que no es una tarea difícil, pero no era imprescindible.

Lo siguiente sería la mecánica conocida en este tipo de títulos como "**focus**" o **concentración**.

Si mantienes pulsado X (se pretende que sea R en 3DS), **ocurrirán 2 cosas**: te **moverás más lentamente,** lo que te dará más control del personaje a la hora de esquivar ciertos proyectiles *(ya que sin estar concentrado, la velocidad es mayor y podrías chocarte),* y además, aparecerá un pequeño punto en el centro del personaje.

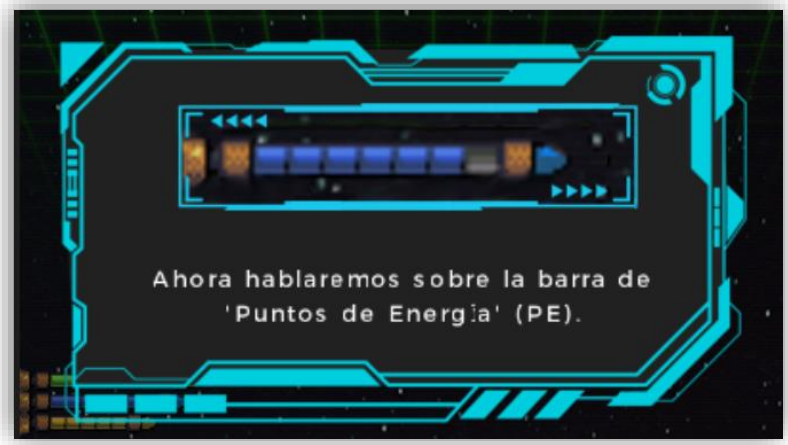




Este punto se trata del **Punto de Colisión,** que coincide con los datos que usamos en la función anteriormente vista (circleInsideCircle) para calcular si el circulo del personaje, coincide con el punto de colisión de un proyectil.

Pd: El punto de colisión de un proyectil suele ser más pequeño que su sprite, dando la sensación de que se esquiva el proyectil por los pelos. Se trata de una mecánica muy usada en los videojuegos que incluyen proyectiles de cualquier tipo.

Lo siguiente sería la barra de **Puntos de Energía**. Esta barra indica la cantidad de energía que nos falta para obtener un "**Gran Orbe**", que es una esfera que girará en torno a nuestro personaje y de la cual saldrán los proyectiles que les dispararemos a nuestros oponentes *(se muestra en la imagen de la siguiente pág.)*.



Una vez se llene la barra, obtendremos permanentemente un Gran Orbe hasta un máximo de 4, y la barra se quedará permanentemente llena.

A continuación hablaremos de los **Orbes Energéticos,** que constan de dos tipos: los azules que proporcionan poca energía, pero se recogerán automáticamente, y los amarillos, que el jugador deberá recoger, pero que proporcionan más energía.

Cabe tener en cuenta que en función de la rapidez con la que se obtengan los orbes energéticos, antes se dispondrá del aumento de daño que esto conlleva, por lo que vale la pena esforzarse en recoger los orbes cuanto antes (al menos los amarillos, ya que los azules se dirigen al jugador).

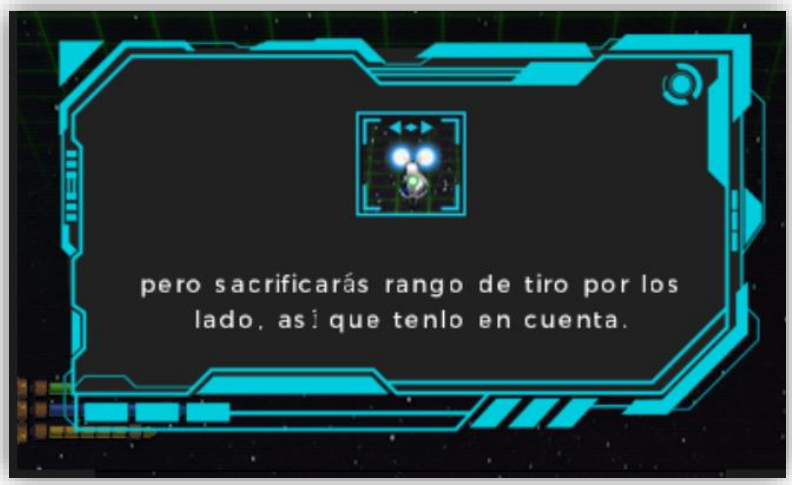
Ahora sí, los **Gran Orbes,** que desempeñan **la fuente de disparo y por ende, de daño**, **del personaje**. Obtendrás uno hasta un máximo de 4 por cada barra de energía que completes.

Al mantener pulsado **Z** *(o lo que se pretende en **3DS**, el **botón A**)*, se podrá disparar una ráfaga de proyectiles en línea recta que se utilizará para derribar a los enemigos.

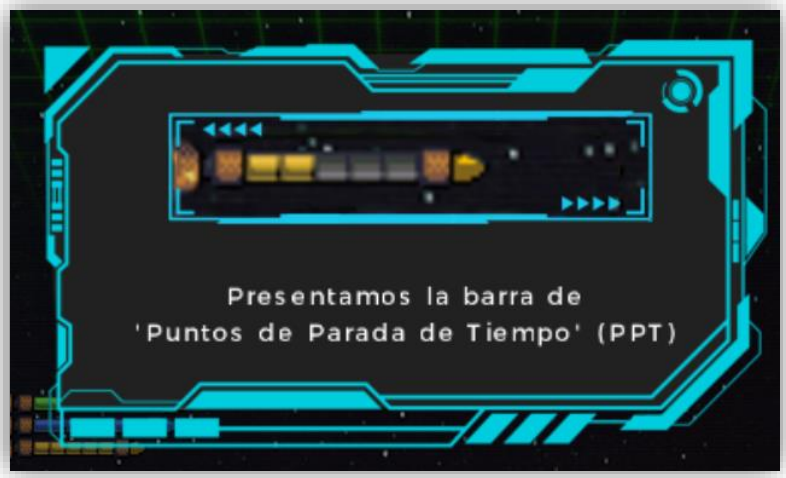
Pd: los proyectiles que dispares no tienen límite.

Por cierto, **si te concentras (R),** reunirás los orbes delante de ti, <u>sacrificando rango a cambio de precisión en el eje vertical al personaje</u>.

Esto viene bien utilizarlo cuando quieres neutralizar a un enemigo rápidamente o para acabar con muchos que se encuentren en un mismo punto.



Por último, tenemos la barra de **Parada de Tiempo (PT),** que indica la cantidad de puntos que disponemos para lanzar nuestra habilidad *(mencionada en el apartado de problemas complicados)*.



Si presionas **S** *(se pretende L en 3DS)*, **podrás lanzar la habilidad**, que consiste en una **parada temporal** *(abreviada **TS** anteriormente "Time Stop")* en la cual nada podrá moverse a excepción del jugador, y la cual **durará** teóricamente **1 segundo** *(1,4 seg. prácticamente, para dar una mayor capacidad de reacción al jugador)* y que servirá para reposicionarse en un mejor lugar.

Para **cargar la barra de TS** deberemos rozar los proyectiles lo mas cerca que podamos sin llegar a colisionar con ellos para cargar el reloj *(lore, idk)* y así poder usar más veces la habilidad *(mecánica existente en el proyecto en el que se basa el videojuego, pero que no realiza la misma función, y he visto conveniente implementar para **recompensar el riesgo** de los jugadores al aproximarse demasiado a los proyectiles)*.

Para esto hay que tener en cuenta que **cada proyectil tendrá un área externa de colisión** *(evidentemente, más grande que la que daña al personaje)* que permitirá al personaje **cargar el medido**r que se encuentra abajo a la izquierda de la pantalla inferior una vez choque con ella.

Cuando el medidor llegue al 100%, se sumara 1 PT a la barra de PT's.

7 Conclusión

Después de completar el proyecto (y a medida que se ha ido desarrollando) me he dado cuenta realmente de lo complicado que es desarrollar un proyecto tal que:

1. **Dispone de poco material de referencia**: La gran mayoría de referencias las encontrareis investigando los trabajos de años anteriores que se os faciliten durante el curso y no mucho más; y es una lástima porque la mayoría del tiempo de realización del proyecto se invierte en investigación y pruebas (al menos en mi caso), y por lo tanto mucho tiempo se "pierde" en la búsqueda de material de referencia que no se encuentra en los recursos del curso.

2. **Estamos limitados a las librerías que nos ofrecen**: "DevKitPro" nos ofrece una gran cantidad de librerías y ejemplos para trabajar con el desarrollo en las consolas disponibles, pero muchas veces no disponen de recursos de ayuda básicos, como por ejemplo: no disponen de ejemplos en los que se utilicen ficheros de **audio**, únicamente el ejemplo de "opus-decoding", pero ni siquiera dispone de funciones para realizar bucles en las pistas de audio o ni siquiera de gestión de la reproducción de las pistas (no puedes ejecutar 2 sonidos al mismo tiempo), u otro ejemplo sería la incorporación de ejemplos en los que se utilicen **animaciones de sprites**, funcionalidad básica en prácticamente cualquier videojuego en 2D.

3. **El Tiempo**: A pesar de que disponemos de alrededor de casi 3 meses para realizar el proyecto, no hay que confiarse. A lo largo de estos 3 meses *(dependiendo de la carga de trabajo media de cada persona y del número de personas que formen tu equipo*) hay que tener en cuenta los posibles contratiempos que os pueden surgir, tanto en el ámbito académico, como en el personal.
   En mi caso ocurrieron **2 contratiempos importantes**:

   El primero fue precisamente lo que se comentó en el punto anterior: me llevó más tiempo del esperado encontrar material de referencia válido para la realización de funcionalidades clave para el videojuego (sonido, por ejemplo).

El segundo fue que, debido a un problema personal que me ocurrió, y que para mí fue totalmente **imprevisible**, me vi obligado a realizar una pausa en la realización de trabajos académicos, lo que supuso un retraso en las tareas en las que me veo involucrado en el curso (sobre todo las individuales, como la realización de este proyecto).

Dicho esto, el **tiempo** es una variable importante a tener en cuenta y que al final nos ha afectado a casi todos por igual (incluso los compañeros que han realizado las exposiciones de sus proyectos comentaron la falta de tiempo). Por ende, mi consejo es que seas realistas en la medida de lo posible, y que tengáis en cuenta que uno mismo tiene más obligaciones que pueden consumir parte importante del tiempo que tenemos para la realización del proyecto, por lo que es importante tener esto en cuenta.

8 Trabajos Futuros

Como ejemplo para trabajos futuros podría tomar simplemente el **desarrollo completo del videojuego**, añadiendo las **mecánicas** y **funcionalidades** que no han podido incluirse por falta de tiempo, e incluso la incorporación de algunas nuevas, es decir, realizar por completo las tareas que aparecen en el apartado "Trabajo Extra Opcional" del apartado de Planificación.

Ya que se trata de un proyecto que me ha hecho mucha ilusión realizar, pretendo dedicarle algo más de tiempo del que pueda disponer en un futuro para así:

- Seguir aprendiendo sobre lo que nos permiten realizar las librerías que se nos proporcionan.
- Es un proyecto que empecé dedicado a mi hermano menor *(a quien le encanta esta videoconsola)* y que me gustaría terminar por él.

Por eso, **iré actualizando el proyecto cuando pueda en mi repositorio de GitHub**[11]. Así que, si alguien tiene alguna duda respecto al proyecto, puede escribirme al correo agregado en la portada, que contestaré con mucho gusto cuando lea.

9 Créditos y Agradecimientos

9.1 Agradecimientos

Me gustaría agradecer a:

- El **profesor** que me ha instruido en la asignatura, que nos ha enseñado a dar los primeros pasos y el cual nos ha proporcionado la ayuda necesaria para desarrollar el proyecto: **Manuel Agustí Melchor (Manolo)**[12].

- Los **desarrolladores** de "**DevKitPro**" [5] por proporcionar las herramientas necesarias para el trabajo con las videoconsolas.

- Los que no son **desarrolladores** de "DevKitPro" pero han desarrollado cosas la mar de útiles y las han compartido: como **PabloMK7**[4] **y NyankoTear**[10] **entre otros…**

- Los **alumnos** los cuales han realizado la asignatura anteriormente y de los que me he basado para hacer mi proyecto. Y a los demás, también, ¿por qué no…?

- Los **artistas** los cuales ofrecen sus maravillosos recursos gratuitamente y de los que daré crédito en el siguiente apartado.

- Al **creador de la obra** en la que se basa el proyecto: **"ZUN"**.

- A **mi hermano** por animarme a hacer el proyecto.

- **A ti, que estás leyendo esto.**

Pd: (si has leído/vas a leer el código quería pedir disculpas de antemano, ya que en mi opinión está bastante mal estructurado, con fragmentos que de momento no se usan, etc… pero a medida que pasa el tiempo iré apañándolo mejor [eso sí, no creo que sea pronto] a la vez que voy aprendiendo estructurarlo mejor para así en próximas ocasiones hacerlo mejor).

De nuevo, gracias por leer, y muy buenos días/tardes/noches.

9.2 Créditos

Música

IECORBAK

Pieza: 永夜の報い　～ Imperishable Night.

Disponible en: https://www.youtube.com/watch?v=w0-Cv6PsY4c

THE BEAVERHOUSE

Pieza: His World 8 Bit Remix V2.

Disponible en: https://www.youtube.com/watch?v=w0-Cv6PsY4c

TOUHOU ALBUM

Pieza: 封印されし神々

Disponible en: https://www.youtube.com/watch?v=EleElqqaBao

Circle: PHOENIX Project

ARTE GRÁFICO

Stars in Shadow - Sprites de los enemigos.

Disponible en: http://stars-in-shadow.com/artwork_license.html

Página del juego: https://store.steampowered.com/app/464880/Stars_in_Shadow/

Pimen – heal animation (unused) - itch.io

Disponible en: https://pimen.itch.io/cutting-and-healing

Fightswithbears – botiquines – itch.io

Disponible en: https://fightswithbears.itch.io/2d-health-and-ammo-pickups

hIkaMI_sAmA – Sprites de los proyectiles

Disponible en:

https://www.reddit.com/r/touhou/comments/tiqbdm/i_found_out_that_using_sprite_sheets_ripped_from/

10 Bibliografía

[1]
Wikipedia: "Matamarcianos (shot 'em up)"
Disponible en: https://es.wikipedia.org/wiki/Matamarcianos

[2]
TouhouWiki: "Touhou_Project"
Disponible en: https://es.touhouwiki.net/wiki/Touhou_Project

[3]
TouhouWiki: "Team_Shanghai_Alice"
Disponible en: https://es.touhouwiki.net/wiki/Team_Shanghai_Alice

[4]
GitHub PabloMK7: "libcwav"
Disponible en: https://github.com/PabloMK7/libcwav

[5]
DevKitPro Home Screen
Disponible en: https://devkitpro.org/

[6]
Citra Home Screen
Disponible en: https://citra-emu.org/

[7]
M. H. David y M. M. Moises - Ast3roiDS – Proyectos años anteriores 2019/20
Disponible en: Proyectos de la asignatura (enlace privado UPV).

[8]
GitHub PabloMK7: "libncsnd"
Disponible en: https://github.com/PabloMK7/libncsnd

[9]
GitHub PabloMK7: "cwavtool"
Disponible en: https://github.com/PabloMK7/cwavtool

[10]
GitHub NyankoTear: "reimu_idle"
Disponible en: https://github.com/NyankoTear/reimu_idle

[11]
GitHub KhrisGit: "AEVProject"
Disponible en: https://github.com/Khrisgit/AEVProject

[12]
A. M. Manuel "UPV Ficha Personal"
Disponible en: http://www.upv.es/ficha-personal/magustim