

Realizado por

Alejandro López Alonso

# Space Invaders

Proyecto Final de Desarrollo de  
Aplicaciones Multiplataforma

IES Francisco de Goya



## Índice

1 Introducción.....	3
2 Objetivos Iniciales.....	3
3 Tecnologías.....	4
4 Análisis de Requisitos.....	10
5 Entorno de Trabajo.....	11
6 Desarrollo del Proyecto.....	13
7 Pruebas.....	20
8 Conclusión.....	21
9 Bibliografía.....	23

## 1 Introducción










Este documento PDF es la memoria del Proyecto de Fin de Grado del Ciclo Formativo de DAM ( Desarrollo de Aplicaciones Multiplataforma ).

Este videojuego es una referencia al conocido juego retro de Space Invaders desarrollado en el lenguaje de programación C++.

## 2 Objetivos Iniciales

Este proyecto tiene como finalidad, la creación y el desarrollo de un videojuego retro de los 80. De manera que se aprenda a su vez un nuevo lenguaje de programación nuevo, como es el de C++. A parte del nuevo aprendizaje de este, también se verán nuevas tecnologías, como nuevos entornos de desarrollo que posteriormente explicaré de manera más específica.

Los objetivos iniciales, a tener en cuenta para lograr este proyecto son:

-  Aprendizaje y investigación del lenguaje de programación C++.
-  Capturar la consola por pantalla y su desplazamiento en las coordenadas "x"y "y".
-  Delimitar la pantalla y desarrollar un entorno parecido al videojuego.
-  Creación de un objeto Nave, el cuál pueda realizar sus respectivas funciones.
-  Creación de un objeto Ovni, que pueda realizar movimiento y pueda disparar.
-  Creación de un objeto disparo que genere el jugador.
-  Desarrollar una función que permita comprobar si el ovni ha sido tocado por el disparo.
-  Generar un método que detecte la colisión de los Ovnis en la nave.
-  Comprobar si los Ovnis han llegado a tierra o la Nave los ha destruido todos.

### 3 Tecnologías

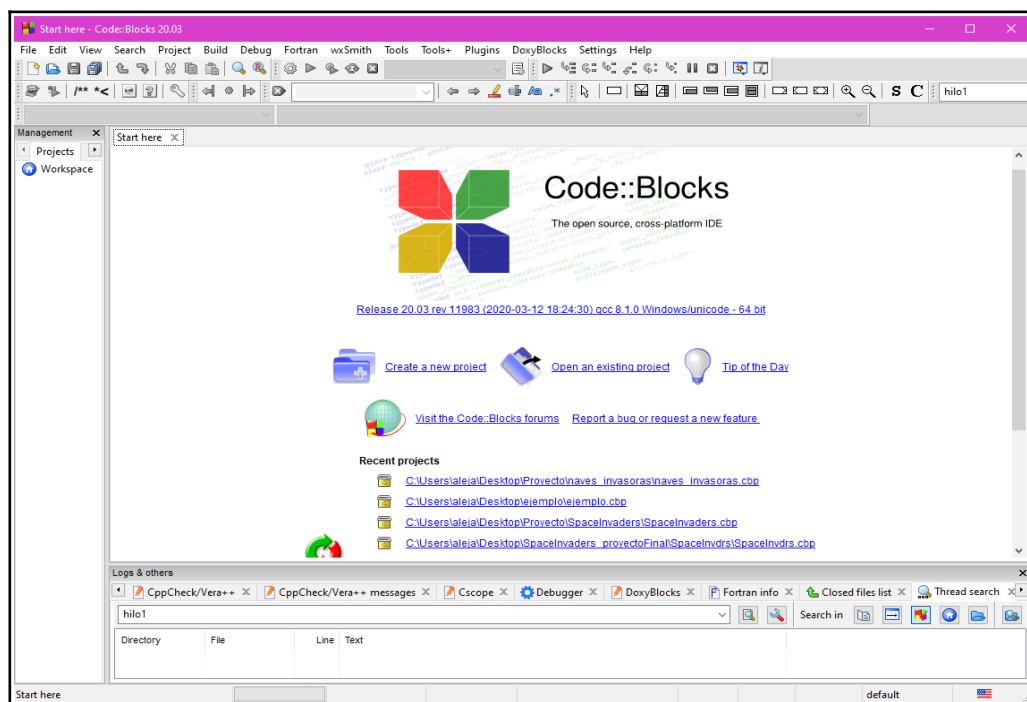
Las tecnologías y entornos de desarrollo usados en este proyecto han sido:



El entorno de desarrollo elegido para el proyecto final ha sido CodeBlock. Este editor es de código abierto y puede soportar múltiples compiladores, como GCC, Visual C++ o Clang.

Contiene un sistema de compilación personalizado y está creado específicamente para C y C++.

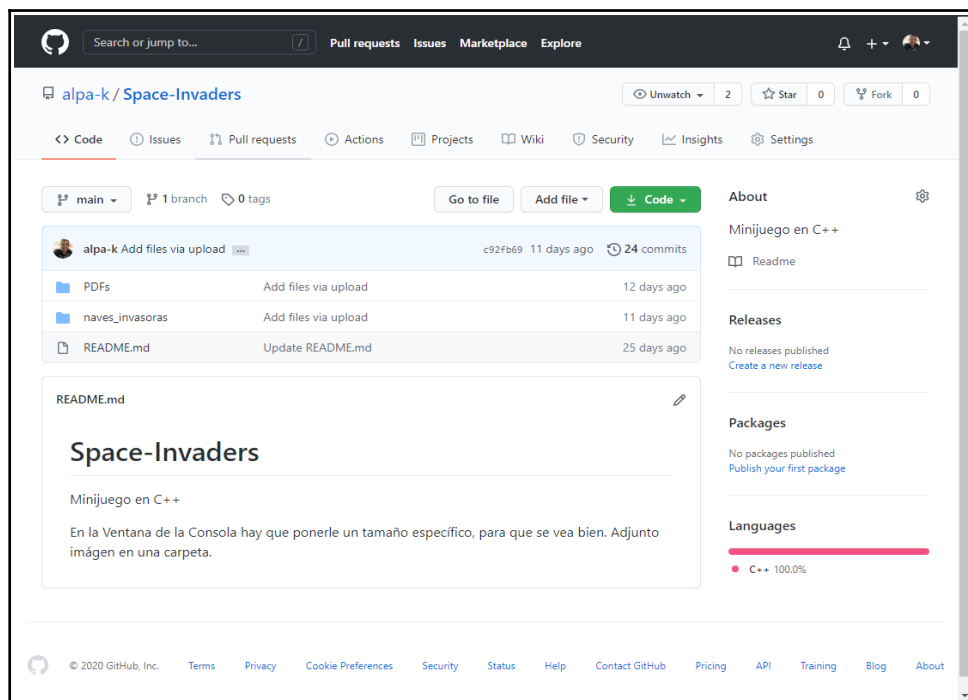
En cuanto a su entorno, es bastante orientativo, contando con un panel arriba en el cuál está la barra de herramientas para poder configurar todo. A la Izquierda los proyectos abiertos junto a una ramificación de sus carpetas y ficheros. Y seguido a la derecha de un esquema de implementado en el proyecto, como sus diferentes clases, funciones, etc. Por último, cuenta con una ventana de herramientas con diferentes opciones de depuración, consola, debugger y muchas más opciones.





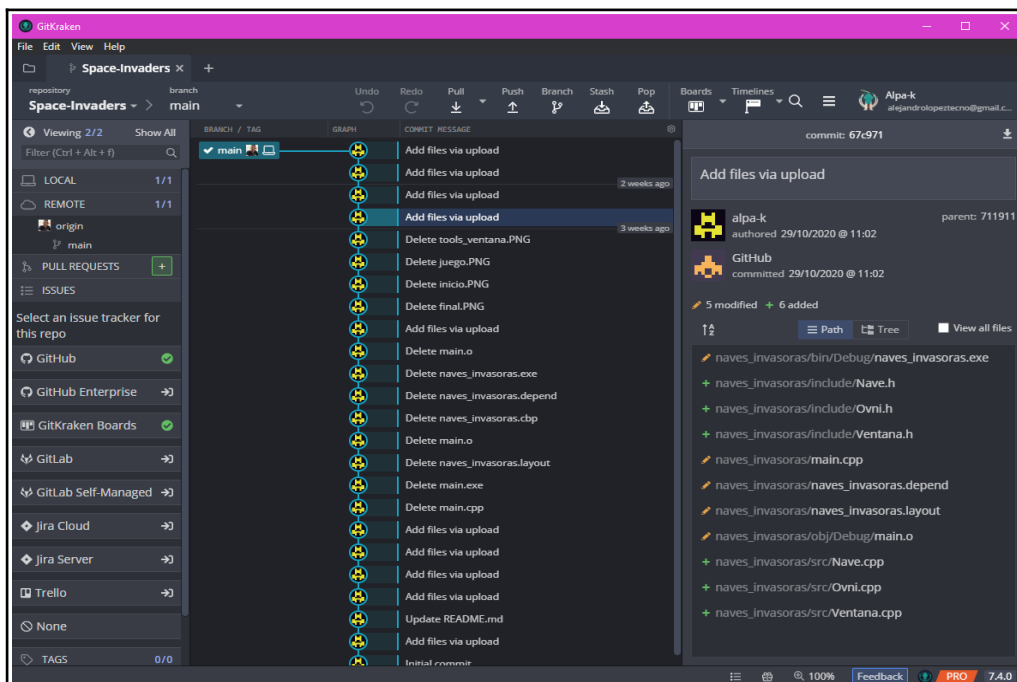
Es una plataforma basada en el desarrollo de proyectos de forma colaborativa, basada para el uso en proyectos utilizando un control de versiones Git. Y que puedan guardarse los cambios en la nube permitiendo así un funcionamiento cooperativo entre varias personas.

Al igual que GitKraken, te permite tener una sesión remota y otra local, la cuál vas subiendo a la nube a medida que vayas realizando cambios en la local. Al igual que con GitKraken, estos dos Git comparten un gran parecido y están enlazados de forma sincronizada. Para cuando hagas un “push” se actualice en ambas.



Es un software de control de versiones, con la funcionalidad de registrar los cambios realizados por versiones subidas o “commits”, este está enlazado al repositorio de GitHub y se va sincronizando cada vez que se actualice cada versión. Además cuenta con una interfaz gráfica bastante llamativa de forma visual, que muestra de manera jerárquica los cambios realizados por cada persona que esté en el proyecto. En mi caso, únicamente estoy yo, por lo que solo hay una línea de ramificación.

Por la parte de entorno es bastante similar a muchas otras, ya que contiene a la izquierda un panel en el que se muestran los repositorios con los que se están trabajando de manera local, como a la nube (en mi caso será a GitHub). En la parte superior hay una barra de herramientas, bastante simplificada a los comandos que serían de "Push", "Pull", "add ." del Git Bash de consola. Es bastante útil ya que es más intuitivo y no hace falta saberse los comandos. Para concluir, a la derecha de este entorno hay una ventana que te muestra lo que contiene cada carpeta y archivo y los cambios realizados junto a los comentarios editados por cada persona.



Para concluir, quería mostrar también la forma de añadir un comentario (Commit) a una versión nueva del código y subirlo a la raíz del proyectos.



## → **Notion**

Es una aplicación que proporciona una gran paleta de herramientas administrativas, como base de datos, tablas, calendarios, checks de tareas para un seguimiento más meticuloso. Te permite conectarte tanto tu como varios usuarios y ir haciendo cambios a las tareas realizadas de forma simultánea. Posteriormente te permite ir subiendo los cambios y exportarlos a PDF.

Está desarrollada para poder usarla tanto en cualquier dispositivo ya sea ordenador o móvil y también directamente en la web.

Aa TAREAS	ESTADO	Comentarios
<u>Puntos de la Nave</u>	Finalizado	
<u>Delimitar pantalla con bordes</u>	Finalizado	
<u>Crear Nave</u>	Finalizado	
<u>Crear Disparos</u>	Finalizado	
<u>Crear Ovnis</u>	Finalizado	
<u>Aliviar Procesador</u>	Finalizado	
<u>Acabar juego cuando no haya ovnis</u>	Finalizado	
<u>Previsión de Calendario</u>	Finalizado	
<u>Manual del Proyecto</u>	Finalizado	
<u>Crear Hilos</u>	Finalizado	
<u>Investigación del Proyecto</u>	Finalizado	
<u>Aprendizaje del lenguaje</u>	Finalizado	
<u>Untitled</u>		
<u>Tareas en Proceso</u>		-----
<u>Presentación</u>	En Proceso	<a href="https://prezi.com/view/E5Xu4VC8aHqIQj77N2PL/">https://prezi.com/view/E5Xu4VC8aHqIQj77N2PL/</a>
<u>Ejercicios de Programación</u>	En Proceso	
<u>Simplificar Clase Ovní con bucles</u>	En Proceso	
<u>Untitled</u>		
<u>Tareas sin Empezar</u>		-----
<u>Mejorar movimiento ovnis</u>	Sin Empezar	
<u>Memoria del Proyecto</u>	Sin Empezar	
<u>Crear Disparos ovnis</u>	Sin Empezar	
<u>Untitled</u>		

## → C++

Es un lenguaje de programación híbrido, ya que este lenguaje combina características de alto nivel como la manipulación de objetos y las de bajo nivel como el manejo de bits. Esto te permite usar la programación de forma estructurada o de forma que esté orientada a objetos.

Este lenguaje ha contribuido en influenciar muchos otros tipos de lenguajes como Perl, Java, PHP, C#, etc.

Cuenta con un gran número de entornos de desarrollo, como Visual Studio, Dev-C, CodeLite y CodeBlock. Este último es el con el que he decidido trabajar. También cuenta con un gran número de Softwares creados en este lenguaje como es Google Chrome, Adobe, Windows phone, Bitcoin o µTorrent .

```
#include <iostream>

using namespace std;

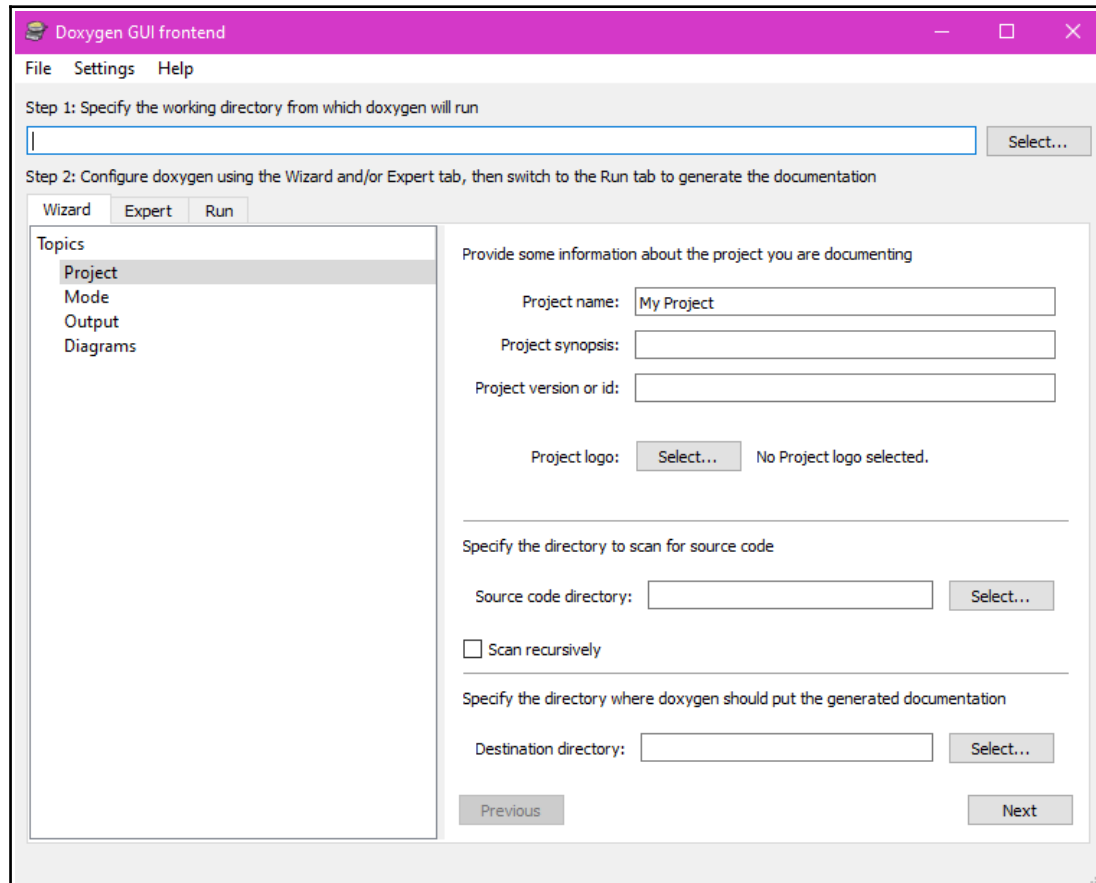
int main()
{
    cout << "Hello world!" << endl;
    return 0;
}
```

## → Doxygen

Es un generador de documentación para C++, C. Sirve tanto para sistemas Unix, Windows y Mac OS.

Para producir el documento html, primero tenemos que copiar la ruta del directorio y una puesta la ruta, iremos configurando sus características, como vayamos requiriendo. También permite estilizar el documento mediante una gama de colores. Es bastante simple pero permite documentar el código de manera eficiente.





## 4 Análisis de Requisitos

Este proyecto fue elegido para el aprendizaje, investigación del lenguaje de programación C++ y puesta en práctica de los conocimientos adquiridos mediante un juego en dicho lenguaje.

Las características base que se han tenido como requisito han sido:

- 🚗 Capturar la salida por pantalla y delimitarla.
- 🚗 Una clase llamada Nave, el cuál sea capturado por la pantalla de consola y se pueda mover por las coordenadas "x" e "y".
- 🚗 Otra clase llamado disparo, que sea producido por el usuario cuando él lo desee. Tendrá unas limitaciones y generará una interacción si este alcanza a otro objeto.
- 🚗 Por último, una clase con el nombre de Ovni, el que se vaya desplazando de forma independiente por la pantalla y llegue a interactuar con la nave y el disparo producido. También tendrá una interacción si esta clase colisiona con la nave, produciendo una disminución de puntos de vida.
- 🚗 El uso del mayor número de conocimientos adquiridos en clase y implementados en este proyecto, como hilos para generar procesos ligeros y no cargar el procesador, excepciones, para depurar errores que puedan ir saliendo, etc.
- 🚗 Y por último, crear un entorno para el jugador interactivo y parecido al juego original.

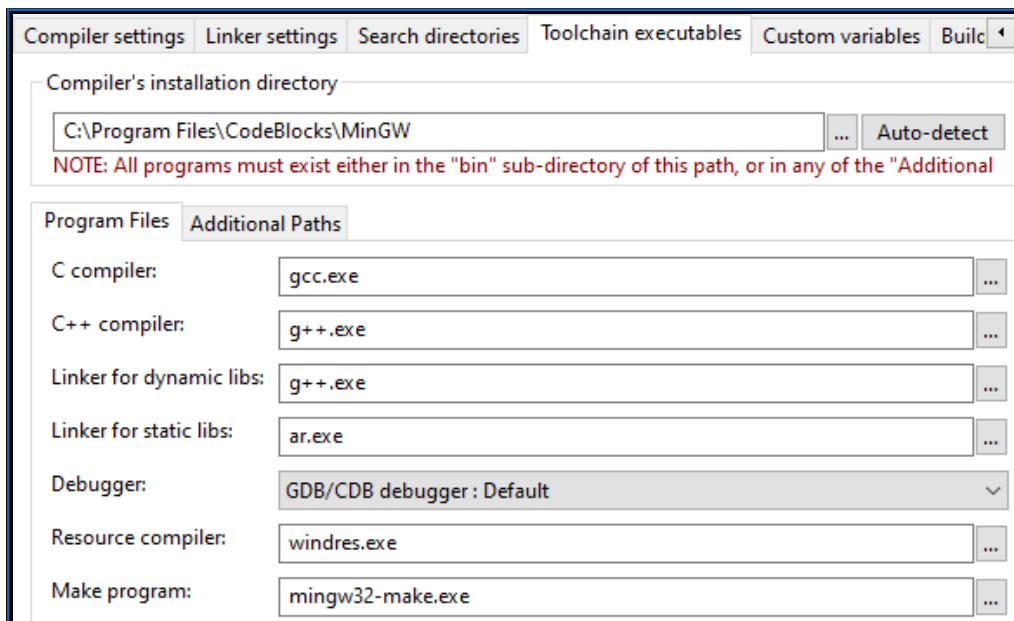
## 5 Entorno de Trabajo

Las instalaciones pertinentes a la hora de realizar este proyecto han sido pocas. Teniendo únicamente que instalar el software de desarrollo CodeBlocks y realizar algunas configuraciones en este junto a unas modificaciones en algunas librerías del sistema.

Para comenzar con su instalación deberemos de dirigirnos a la página oficial de CodeBlocks y descargaremos su binario, seleccionando el enlace .exe junto a su compilador MinGW, ya que si no posteriormente dará una serie de errores de compatibilidad y será más tedioso.

Una vez realizado la descarga, ejecutaremos el archivo y seleccionaremos el botón de siguiente eligiendo la ruta que deseemos. Una vez instalado si deseamos cambiar la apariencia de la interfaz del software habrá que instalar una librería en la opción de *settings*, pero eso será a gusto de cada uno.

Ya habiendo ejecutado el software deberemos comprobar si están las características del compilador adecuadamente, para llegar aquí deberemos abrir la pestaña de *Global Compiler* y seleccionar la opción de *Toolchain executable*, una vez realizado esto comprobaremos que están correctamente los archivos deseados, tal como se muestra en esta imagen:



Cabe añadir que una vez realizado el proyecto, si deseamos ejecutarlo, podremos hacerlo mediante dos opciones, la primera sería mediante la opción gráfica del software de desarrollo, como es CodeBlock o bien mediante comandos gracias a la terminal. Para realizarlo mediante la segunda opción deberemos seguir los siguientes pasos:

Al estar el programa está desarrollado en C++ tendremos que compilarlo mediante g++ y no mediante gcc que se emplearía para C.

```
PS C:\Users\aleja> g++ main.cpp
```

Una vez realizado esto en la consola, se nos habrá generado un archivo ya compilado que podremos ejecutarlo escribiendo el nombre del proyecto junto una terminación inicial, tal como muestro:

```
PS C:\Users\aleja> .\main.exe
```

## 6 Desarrollo del Proyecto

Sobre el desarrollo del proyecto, expondré algunas partes del código para explicar algunas funcionalidades importantes:

Antes de comenzar con la realización del proyecto tenemos que capturar la salida y entrada por la consola mediante un tipo de puntero. Este se le conoce como *Handle* y es un puntero inteligente utilizado para controlar los bloques de memoria o objetos para otros sistemas. En nuestro caso para controlar la pantalla de la consola.

Junto a este puntero se le incluirá una un tipo de estructura, llamada *Coord* que su función es implementar unas variables (coordenadas), para el búffer de la consola por pantalla.

```
/**
 * @brief pintarPantalla Captura la salida por pantalla.
 * @param handle Puntero inteligente captura E/S y errores por pantalla.
 * @param coord Estructura que define las coordenadas en un búffer (0,0).
 * @return h, v Devuelve las coordenadas por pantalla.
 */
void pintarPantalla(int h, int v)
{
    COORD coord;

    coord.X = h;
    coord.Y = v;

    SetConsoleCursorPosition(handle, coord); //Pinta coordenadas
}
```

A parte de crear esta función, también tendremos que crear otra para quitar el típico parpadeo por consola que genera el cursor. Esta función será bastante simple, ya que ya hay funciones por defecto para controlar este tipo de características.



```

/**
 * @brief quitarParpadeo Captura el cursor por pantalla y quita el parpadeo que produce.
 * @param dwSize Especifica un número entre 1-100 e indica lo que ocupa este.
 * @param bVisible Se encarga de cambiar la visibilidad del cursor.
 */
void quitarParpadeo()
{
    CONSOLE_CURSOR_INFO cursorSkin; // Estruct info cursor por pantlla
    cursorSkin.dwSize = 2;
    cursorSkin.bVisible = false;

    SetConsoleCursorInfo (handle,&cursorSkin); //Contrl caracts cursor // ALIAS VARIABLE
}

```

Al haber usado estas funcionalidades, tendremos que pasárselo a las características del cursor, pasándole la captura por pantalla y los variables con los datos especificados.

Una vez hecho esto para ir centrándonos más, comenzamos con generar los límites por los que se va a realizar todo. Por lo que gracias al código ASCII implementaremos un símbolo para delimitar la pantalla. Esto se generará mediante dos bucles. Uno para que delimite de forma horizontal y otro para delimitar la zona vertical por la pantalla.

```

void limitarPantalla()
{
    for(int i=2; i<78; i++)
    {
        pintarPantalla(i,3);
        printf ("%c", 205);

        pintarPantalla(i,33);
        printf ("%c", 205);

    }

    for(int i=4; i<33; i++)
    {
        pintarPantalla(2,i);
        printf ("%c", 186);

        pintarPantalla(77,i);
        printf ("%c", 186);

    }
}

```

Una vez realizado estos dos bucles, nos quedarán por rellenar las esquinas por lo que habrá que pintarlas de forma manual pintándolas en las coordenadas correspondientes.

```
// ===== ESQUINAS ===== //
```

```
pintarPantalla(2,3);  
printf ("%c", 201);  
  
pintarPantalla(2,33);  
printf ("%c", 200);  
  
pintarPantalla(77,3);  
printf ("%c", 187);  
  
pintarPantalla(77,33);  
printf ("%c", 188);
```

Concluyendo ya este apartado inicial, pasaremos a explicar las clases introducidas y algunas funcionalidades.

Comencemos por la clase `Nave`, esta clase contará con una función propia, con la funcionalidad de borrar el rastro que genera el cursor a medida que se va desplazando por la pantalla. Esto se hará pintando en las coordenadas del cursores unos blancos al rededor de este. Dando una apariencia de que no deja rastro por pantalla.

```
/**  
 *@brief borrar Función que borra el recorrido de la nave por la pantalla  
 */  
void borrar()  
{  
  
    pintarPantalla(h,v);  
    printf(" ");  
}
```

Otra función a destacar será la del movimiento de de esta. Por lo que capturaremos el teclado con la función `kbhit`(del inglés *key board hit*) y le pasaremos los parametros recogidos por un `switch` para que verifique unas condiciones para su movilidad.

```
void mover()
{
    if(kbhit())
    {
        char key = getch();
        borrar();

        switch( key )
        {
            case IZQUIERDA:
                if( h>3 )
                {
                    h--;
                }
                break;

            case DERECHA:
                if( h+3 < 76 ) //SE LE SUMA SEGUN EL NUM DE CARCTS NAVE
                {
                    h++;
                }
                break;
        }
    }
}
```

Dentro del *switch* las opciones a verificar en los *case* tendrán nombre, pero el valor a comprobar será numérico, en este caso para comprobar la tecla pulsada por el teclado. Esto se debe a que he incluido "*macros*" las cuales sustituyen al número de la tecla requerido.

```
#define IZQUIERDA 75
#define DERECHA 77
```

Por último pero no menos importante explicare otra de las funciones de la clase Nave. Esta función está creada para cuando la Nave sea golpeada. En ella he querido introducir un función Mutex, a parte algunas más. Esta función es un objeto utilizado para la implementación de hilos. He añadiendo esto para poder libiar los procesos generados en el prrograma y no depender tanto de *Sleeps*.

```

/**
 * @brief naveGolpeada Función que controla las variables de colisión de la nave y su comportamiento.
 * @param semaforo.lock Marca el inicio de ejecución hasta que se cierre para ejecutarlo mediante un Hilo.
 * @param semaforo.unlock Concluye el final de ejecución junto al Lock para la estructura mutex.
 * @return Devuelve las vidas restantes, genera una colisión por pantalla y vuelve a pintar la nave.
 */
void naveGolpeada()
{
    semaforo.lock();

    if (corazon == 0)
    {
        borrar();
        pintarPantalla (h,v);
        printf ("  %c  ",176);

        Sleep(200);

        borrar ();

        pintarPantalla (h,v);
        printf ("%c",176);

        borrar();
        vida --;
        corazon =3;
        mostrarVidas();
        pintar();
    }
    semaforo.unlock();
}

```

Sobre la la clase *Ovni* tiene las mismas funciones que la clase *nave*, a excepción de algunas características que expondré posteriormente junto a la clase *Disparo*.

Para la clase *Ovni* y la Clase *Disparo* tuve que crear una lista de punteros y a su vez un índice de esta para poder posteriormente llegar a desplazarme por esta.

```

/**
 * @brief Creamos una lista que contenga punteros y un índice para poder recorrerlo posteriormente
 */
list<Ovni*> O; //Lista de Punteros
list<Ovni*>::iterator indexO; //Creacion de un índice para recorrer la lista

list<Disparo*> D;
list<Disparo*>:: iterator indexD;

```

Una vez generado estas listas con punteros para estas dos clases, introducimos en sus coordenadas que hemos decidido poner los Ovnis.



```
/**  
 *@brief Añadimos un nuevo Ovni en la lista junto a unas coordenadas individuales.  
 */  
O.push_back(new Ovni(10,6) );  
O.push_back(new Ovni(13,6) );  
O.push_back(new Ovni(16,6) );
```

Introducido ya los Ovnis deseados, tuvimos que generar un b ucle para recorrer esta lista y dentro de este b ucle llamar a sus distintas funciones. Si estas se llegan a cumplir, como la funci n de Ovni golpeado, se eliminar  el enlace y pasar  al siguiente.

```
for(indexO=O.begin(); indexO!=O.end(); indexO++)  
{  
  
    (*indexO)->mover();  
    (*indexO)->choque(nave);  
  
    if((*indexO)->llegaOvni() )  
    {  
  
        pintarPantalla( (*indexO)->X(), (*indexO)->Y());  
        printf(" ");  
  
        delete(*indexO);  
        indexO = O.erase(indexO);  
    }  
}
```

Para la clase *Disparo*, dispondr  al igual de la clase *Ovni* de un b ucle para recorrer su lista, pero a diferencia de lo realizado anteriormente, en este caso utilizaremos la funci n kbhit, ya implementada anteriormente en mi c digo para comprobar si se est  pulsando la tecla requerida.

```

if(kbhit() )
{
    char key = getch();

    if( key == 32 )
    {
        D.push_back(new Disparo( nave.X()+2,nave.Y()-1 ) );
    }
}

```

Una vez realizado esto por cada vez que se presione la tecla espaciadora, con el número 32, se insertará dentro de la lista de *Disparo* junto a unas coordenadas para que estas concuerden con la figura de la *Nave*.

Al igual que con el búcle del Ovni creamos un mismo búcle para la misma funcionalidad que este.

Una vez generado estos búcles para que se puedan recorrer sus listas tendremos que comprobar si entre estas dos listas hay una coincidencia de coordenadas para poder determinar que ha habido una colisión entre estos dos.

```

/**
 *@brief Bucla que recorre toda las listas de Ovni y Disparo.
 *@return Si concuerdan las coordenadas elimina ambos y genera puntos al jugador.
 */
for(indexO=O.begin(); indexO!=O.end(); indexO++)
{
    for(indexD = D.begin(); indexD != D.end(); indexD++)
    {
        if( (*indexO)-> X() == (*indexD)-> X() && (*indexO)-> Y() == (*indexD)-> Y() )
        {
            pintarPantalla( (*indexD)-> X(), (*indexD)-> Y() );
            printf(" ");
            delete(*indexD);
            indexD = D.erase(indexD);

            pintarPantalla( (*indexO)-> X(), (*indexO)-> Y() );
            printf(" ");
            delete(*indexO);
            indexO = O.erase(indexO);

            puntos+=100;
        }
    }
}

```

## 7 Pruebas

Se generarán una serie de pruebas en busca de errores y comprobando los resultados obtenidos de estos:

<b>Escenario</b>	<b>Resultado Esperado</b>	<b>Resultado Obtenido</b>	<b>Solución</b>
Introducir valores numéricos al inicio del juego.	Continuar con la ejecución y mostrar el valor introducido.	Se ha obtenido el resultado esperado	
Introducir blancos al comienzo del inicio del juego.	Que el nombre del jugador sean blancos	No se ejecuta el programa hasta que no recibe un valor.	Generar una condición para que si introduce un blanco no te permita ejecutar el programa.
Introducir un valor mayor del tamaño previsto	Que salte algún tipo de error.	El programa se ejecuta hasta la longitud especificada	Implementar una condición para que solo se pueda introducir un tamaño específico.
Generar una gran secuencia de disparos.	Que no se produzca ningún tipo de error.	Llega un punto en su proceso que termina el programa, debido un error en la memoria.	Poner una pausa para que el usuario no pueda generar tantos disparos consecutivos
Introducir un símbolo por teclado	No producirse ningún tipo de error, ya que lo representaría como ASCII.	Se escribe el símbolo escrito como nombre de jugador	

## 8 Conclusión

Sobre este apartado, voy a hablar sobre las dificultades encontradas, las conclusiones a las que he llegado por mí mismo y las ideas a futuro que me gustaría introducir en este.

**➔ Dificultades:**

Sobre las dificultades encontradas en este proyecto, tengo que destacar la parte del aprendizaje, ya que este lenguaje de programación es nuevo para mí. Y he tenido que investigar y documentarme sobre este.

También tengo que incluir el software de desarrollo que he decidido usar para el proyecto, ya que entre los que había es de los más complicados de usar, ya que no da ningún tipo de facilidad en cuanto a escribir el código, como podría darlo otro tipo de software como el de Visual Code, que puedes implementar una extensión con un simple click o la ayuda de auto-relleno de código. CodeBlocks por desgracia si quieres implementar algo nuevo ya sea cambiar su entorno gráfico o implementar cualquier tipo de cambio tienes que modificar sus librerías tanto en sus carpetas de origen donde esté instalado como en las carpetas de tu Sistema Operativo.

A la hora de haber querido documentar mi código en un formato parecido al JavaDoc, estuve investigando y descubrí únicamente el software llamado Doxygen, el cuál se usaba hace bastante tiempo en Windows NT o Windows 7. A la hora de descargarmelo en su página oficial estaban los mirrors caídos por lo que tuve que investigar más.

**➔ Alcance y limitaciones:**

El alcance de este proyecto, era desarrollar un videojuego en un lenguaje nuevo, que no había dado en clase, el proyecto en sí me suponía un gran reto. Sobre su alcance inicial ha podido llegar a ser mayor del que tenía pensado. En cuanto a los objetivos que tenía en mente, como a los que he querido ir añadiendo a la hora de ir realizándolo. También cabe mencionar que este lenguaje de programación cuenta con una gran documentación al respecto y por tanto estoy bastante agradecido por ello.




En cuanto, a las limitaciones principales, las librerías me han supuesto un problema, a la hora de implementarlas ya que tenía que hacerlo de manera manual, y saber qué y dónde tenía que realizarlo. La falta de tiempo, ya que quería haber realizado una mejora del proyecto, pero no he llegado a poder realizarlo. El debugger al código me ha supuesto una limitación también, ya que apenas daba información por lo que tenía que buscar el mensaje que me salía por la consola en Internet, para poder saber el tipo de error que me estaba saliendo.



### ➔ Futuras investigaciones:

En el proyecto se ha intentado realizar todas las funcionalidades e incluir nuevas a este, pero no se ha podido llegar a lo deseado, ya que una gran parte del tiempo invertido en este proyecto ha ido dirigido a la investigación y aprendizaje de este lenguaje.

Por este motivo, en investigaciones a futuro tengo la intención de realizar las siguientes funcionalidades a destacar:

-  Implementar a los Ovnis la característica de generar disparos de una forma aleatoria, pudiendo crear una experiencia más interactiva para el usuario que juegue.
-  Introducir una tabla de puntuaciones al concluir el juego que muestre las mejores puntuaciones.
-  Terminar de desarrollar el este juego con la librería Allegro, la cual permitiría el uso de imágenes y sonidos WAV.

### ➔ Conclusión:

Este Proyecto Final para el Grado Superior de Desarrollo de Aplicaciones Multiplataforma, se ha desarrollado en un lenguaje de programación nuevo, el cuál no he llegado a dar anteriormente. Por lo que me ha llegado a suponer un reto , en el cuál he llegado a aprender bastante y me ha ayudado a fijar más mis bases.

Estoy contento de poder haber realizado este proyecto y la experiencia adquirida de este, al haberme enfrentado de manera independiente.

## 9 Bibliografía

La información en la que me he basado para realizar este proyecto y de donde he aprendido este nuevo lenguaje de programación:

-  [Lenguaje C++](#)
-  [C++ 3Wschools](#)
-  [Vídeos aprendizaje en C++](#)
-  [Excepciones C++](#)
-  [CodeBlocks](#)
-  [GitHub](#)
-  [GitKraken](#)
-  [Notion](#)
-  [Allegro](#)
-  [APIS C++](#)
-  [Defines C++](#)