



-

-

-



MEMORIA FINAL DE PROYECTO

Ship Battle

CICLO FORMATIVO DE GRADO SUPERIOR

Desarrollo de Aplicaciones Multiplataforma

AUTOR:

Jorge Correyero Fernández

TUTOR:

Pablo Palacios Vázquez

COORDINADOR:

Jesús Belchi Tendero

1 INTRODUCCIÓN.....	3
2 ALCANCE DEL PROYECTO Y ANÁLISIS PREVIO.....	4
3 ESTUDIO DE VIABILIDAD.....	4
3.1 ESTADO ACTUAL DEL SISTEMA.....	4
3.2 REQUISITOS DEL CLIENTE.....	4
3.3 POSIBLES SOLUCIONES.....	5
3.4 SOLUCIÓN ELEGIDA.....	5
3.5 PLANIFICACIÓN TEMPORAL DE LAS TAREAS DEL PROYECTO <SHIP BATTLE>.....	5
3.6 PLANIFICACIÓN DE LOS RECURSOS A UTILIZAR.....	7
4 ANÁLISIS.....	7
4.1 REQUISITOS FUNCIONALES.....	7
4.2 REQUISITOS NO FUNCIONALES.....	7
5 DISEÑO.....	10
5.1 ESTRUCTURA DE LA APLICACIÓN.....	10
5.2 SISTEMA GUARDADO DE INFORMACIÓN.....	10
5.3 HERRAMIENTAS DE DISTRIBUCIÓN.....	10
5.4 HERRAMIENTAS DE DESARROLLO.....	11
6 DESARROLLO.....	12
6.1 ESTADOS.....	12
6.2 CARPETAS Y ARCHIVOS.....	13
6.3 CLASES Y MÉTODOS.....	15
7 PRUEBAS.....	50
7.1 CASOS DE PRUEBAS.....	50
7.2 CASOS DE PRUEBAS NEGATIVOS.....	62
8 CONCLUSIONES.....	68
9 FUENTES.....	69
9.1 LEGISLACIÓN.....	69
- DAM.....	69
9.2 BIBLIOGRAFÍA.....	69

1 INTRODUCCIÓN

El sector cuaternario de las tecnologías incluyen numerosas empresas que se dedican a diferentes aplicaciones de las tecnologías como la microelectrónica, la robótica, las telecomunicaciones o incluso la informática.

Dentro del sector de la informática, están los videojuegos, los cuales reciben una gran demanda anual para poder satisfacer las necesidades de ocio de los usuarios. Los clientes demandan todo tipo de productos, desde grandes juegos con costes elevados con conexiones a servidores para gran cantidad de jugadores, hasta juegos simples y gratuitos de uno o dos jugadores para disfrutar sin necesidad de conexión a internet. Todo esto hace que el mercado en el que se sitúan tenga una gran variedad de productos y una gran cantidad de clientes actuales y potenciales. Analizando estadísticas sobre el mercado de la industria de los videojuegos a nivel mundial, se puede distinguir un crecimiento anual considerable en el valor del mismo mercado, siendo de 39 mil millones la diferencia entre 2023 y 2024, con China como la mayor potencia dentro de la industria y con mayor número de empresas relacionadas con los videojuegos. A su vez, se puede distinguir un crecimiento del uso de plataformas digitales ('Steam', 'Epic Games', 'GoG Galaxy') en las que se distribuyen y compran los productos a la venta en el mercado de los videojuegos, siendo un 95% adquiridos en formato digital en 2022.

Como consumidor de este tipo de producto desde hace mucho, siempre me ha llamado la atención todo lo que hay detrás, tanto la idea de qué se quiere hacer (temática, tamaño, cliente objetivo...) como todo el proceso de desarrollo para llevarla a cabo (estructuración del proyecto, uso de tecnologías de desarrollo, puesta en marcha...). Por lo tanto, he decidido crear un videojuego de naves llamado "Ship Battle" que se pueda jugar en cualquier momento con un amigo de forma local debido a la carencia de conexión a internet y los bajos requerimientos necesarios que debe tener su ordenador para poder disfrutar de la aplicación.

2 ALCANCE DEL PROYECTO Y ANÁLISIS PREVIO

El propósito general de este proyecto es crear un juego para dos personas que no necesite conexión a internet para jugarlo una vez se descargue a través de una plataforma (como ‘Steam’) para un ordenador de gama baja. No se espera conseguir una gran remuneración del mismo, sino conseguir visibilidad y darse a conocer para atraer ofertas de trabajo relacionadas con el sector. Posterior a la publicación del juego se podrá continuar con más proyectos similares de juegos “indie” o “retro” para agruparlos en una sola aplicación de “Mini Juegos” con un mayor precio de venta incluyendo todos los juegos que se desarrollen. Por el momento, se llevará a cabo el desarrollo e implantación de un sólo juego para analizar la demanda y respuesta de los clientes objetivos.

3 ESTUDIO DE VIABILIDAD

3.1 *Estado actual del sistema*

Actualmente, los videojuegos se encuentran en crecimiento a nivel mundial, incrementando su demanda anualmente de forma considerable. A su vez, las plataformas digitales que se utilizan para distribuir los videojuegos se utilizan cada vez más debido a que permiten al cliente adquirir los productos disponibles desde su hogar sin necesidad de comprarlo en formato físico. Por otro lado, la competencia en la que se sitúa el producto es prácticamente nula, debido a que sólo se puede encontrar un juego parecido en la plataforma ‘Steam’ en la que se pretende distribuir llamado ‘Space Ship Infinity’, aunque este sólo permite jugar un sólo jugador.

El juego se venderá para una sola plataforma, los ordenadores, debido a que resulta muy sencilla la distribución y desarrollo de videojuegos en ella. Para distribuirlo para ordenadores, se usarán plataformas digitales únicamente, por lo que el cliente deberá tener una cuenta e instalado el lanzador para poder comprarlo y descargarlo.

3.2 *Requisitos del cliente*

La demanda de los clientes a los que se pretende llegar requieren un juego que requiera un espacio de almacenamiento muy bajo así como un ordenador de baja calidad para poder jugarlo. También se deberá adaptar a pantallas de diferentes calidades cuando se inicie el juego en pantalla completa debido a que se pretende llegar a clientes de cualquier nivel adquisitivo con pantallas de todas las resoluciones posibles.

3.3 Posibles soluciones

Es posible cambiar el foco a un cliente de mayor nivel adquisitivo para no necesitar preocuparse de la cantidad de paquetes (incrementando el espacio necesario) ni resoluciones de pantalla durante el desarrollo. Como consecuencia, se podría incrementar el precio del videojuego porque se estaría buscando un cliente que pudiera pagar más.

Por otro lado, sería una opción viable seguir enfocándose a un cliente de menor nivel adquisitivo, por lo que durante el desarrollo se tendrá en cuenta la optimización de la aplicación y la adaptación de la misma en diferentes pantallas.

3.4 Solución elegida

Teniendo en cuenta al cliente que se ha decidido enforcar el proyecto, se hará un videojuego de baja calidad pero que necesite unos requerimientos mínimos para poder jugarlo. Debido a los requerimientos que se esperan pedir a los clientes objetivo que quieran comprar el producto, se usará la menor cantidad de paquetes posibles durante el desarrollo y se conseguirá que el juego ocupe un espacio menor a 1 GB. Se optimizará la aplicación lo máximo posible para conseguir un rendimiento máximo sin necesidad de requerir un ordenador de gran potencia y se adaptará a la resolución de la pantalla automáticamente.

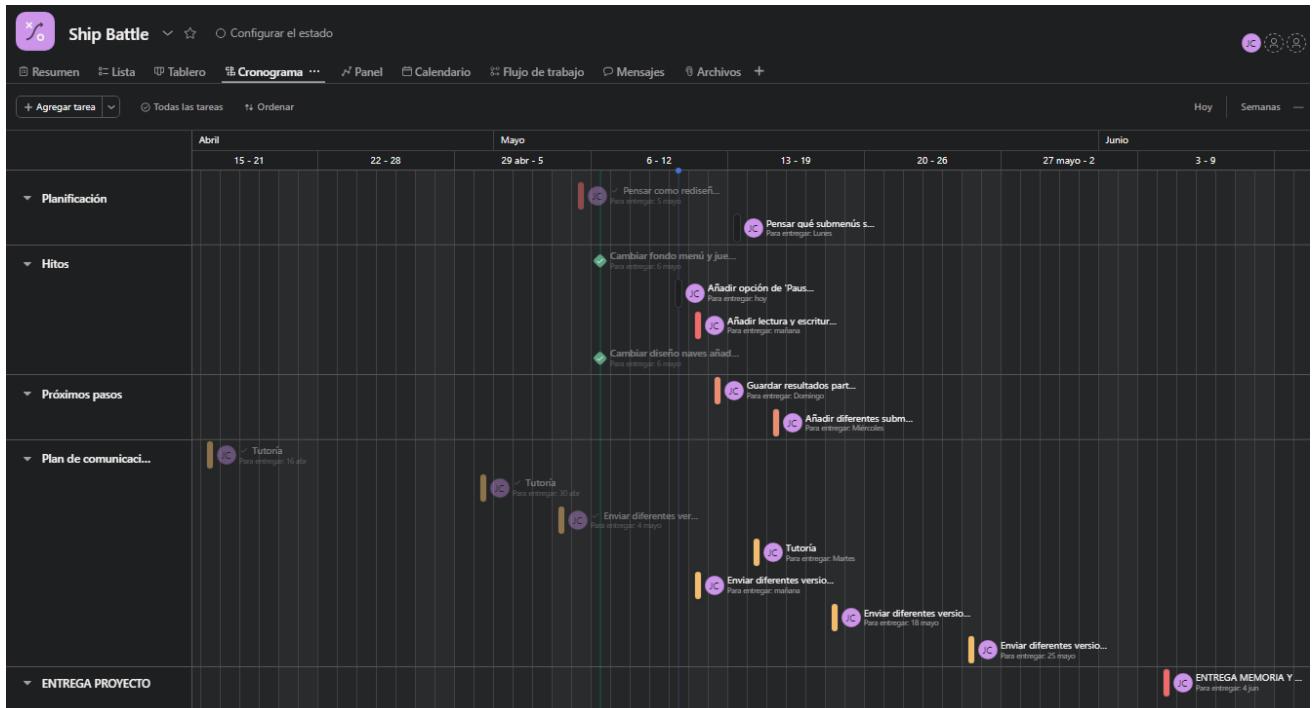
3.5 Planificación temporal de las tareas del proyecto <Ship Battle>

Para la planificación que se realiza durante el proyecto sobre los diferentes cambios e implementaciones de elementos para mejorar la interfaz gráfica y la parte funcional de la aplicación se ha usado ‘Asana’. Con la ayuda de diferentes herramientas que ofrece la prueba gratuita de 30 días se creó un diagrama de Grantt, y una lista de tareas para poder establecer la planificación (pensar ideas), los hitos (cambios), próximos pasos (posibles nuevos hitos), y planes de comunicación sobre el proyecto con el tutor encargado (cada fin de semana desde el comienzo del desarrollo y los martes de tutoría).

Dentro de cada tarea de la *lista* se establece un responsable (en este caso sería yo mismo puesto que no hay más personas implicadas en el desarrollo), fecha de entrega, proyecto asociado a la tarea (‘Ship Battle’), prioridad (Alta, Media, Baja), progreso, y una breve descripción.

The screenshot shows the Asana application interface. On the left, there's a sidebar with sections like 'Resumen', 'Lista', 'Tablero', 'Cronograma', 'Panel', 'Calendario', 'Flujo de trabajo', 'Mensajes', and 'Archivos'. Below these are sections for 'Planificación', 'Hitos', 'Próximos pasos', and 'Plan de comunicaciones'. A specific task is selected: 'Pensar qué submenús se van a añadir en el menú principal'. This task has a due date of 'Lunes' and a priority of 'Alta'. It is assigned to 'Jorge Corey...' and is currently in the 'Trabajo t.' state. To the right of the task list is a detailed view pane. It includes fields for 'Responsable' (set to 'Jorge Corey...'), 'Fecha de entrega' (set to 'Lunes'), 'Proyectos' ('Ship Battle - Planificación'), 'Dependencias' (empty), and 'Prioridad' (empty). There's also a 'Descripción' section with a note about adding more buttons to the main menu. At the bottom of the right pane, there are sections for 'Comentarios' (with a recent comment from 'Jorge Correyero') and 'Colaboradores'.

Por otro lado, dentro de la aplicación ‘Asana’, se crea un **cronograma** (Diagrama de Gantt) automáticamente en relación a las tareas que se añaden dentro de la pestaña *lista*, para poder visualizar de forma gráfica las tareas que se han terminado y las que todavía se tienen pendientes para poder seguir con el desarrollo del juego de forma continua y organizada. También permite ver las fechas programadas que se han establecido para permitir que el tutor tenga un seguimiento del proyecto y la fecha de entrega final.



3.6 Planificación de los recursos a utilizar

Durante el desarrollo del proyecto se usará un ordenador capacitado para poder llevarlo a cabo sin complicaciones. Sólo contará el equipo con un solo desarrollador que se encargará de realizar las pruebas y establecer la planificación y cómo implementar el juego para poder distribuirlo a través de plataformas digitales. Debido a que seré yo mismo dicho desarrollador, no será necesaria la contratación de personal actualmente para la realización de ninguna de las partes del proyecto.

4 ANÁLISIS

4.1 Requisitos funcionales

Primero, para que el cliente pueda obtener del juego, deberá tener una cuenta de 'Steam' y tener instalada dicha plataforma, para posteriormente poder pagar 0,99€ y descargarlo. También es importante tener en cuenta que el juego sólo funciona para un sistema operativo 'Windows', debido a las rutas que utilizan dentro del código para poder acceder a los diferentes archivos dentro de la aplicación, como por ejemplo la música.

```
self.musica.load("music\Musica_Menu.mp3") #carga el .mp3
```

Los requisitos a nivel de *hardware* son realmente bajos ya que el juego ocupa menos de 500MB y se adapta a la resolución de pantalla que tenga el usuario, sin necesidad de cambiarlo en ajustes.

4.2 Requisitos no funcionales

Por otro lado, tenemos los diferentes requisitos no funcionales que se han establecido a lo largo del desarrollo del proyecto.

El diseño general de la aplicación se enfoca en un estilo retro, utilizando imágenes de *pixel art* y el mismo tipo de fuente para todos los textos (*American Captain*), manteniendo colores oscuros o tenues que no dañen la vista del usuario. La disposición de los elementos como botones o *sprites*, se mantienen según en tipo de resolución de pantalla que tenga cada usuario para que la interfaz sea *user friendly* para todo tipo de jugadores.

También se han implementado diferentes funcionalidades, para ayudar al usuario con el manejo de los menús: para la búsqueda del historial de las 10 ultimas partidas (mostrando un texto predeterminado si no hay partidas guardadas), los resultados de un jugador buscado por TAG (indicando si existe o no), y la elección de TAGs (permitiendo que continúe al juego si ha escrito dos TAGs distintos) cuando haga *click* en el botón ‘Jugar’ del menú principal. A su vez, se incluye un ícono de ayuda (un símbolo de interrogación abajo a la izquierda en todos los menús) para que el usuario pueda consultar los controles de cada jugador manteniendo pulsado dicho ícono.

Por el momento, el jugador tan sólo podrá escoger un rango de *FPS* de 30-60-120, y el tamaño de la ventana es completamente automático, sin posibilidad de cambiarlo sin modificar el fichero de variables constantes “*propiedades.py*”, lo cual se podrá implementar en futuras versiones del juego. También podemos ver que la resolución mínima para que los elementos de los menús sean legibles (no se solapen entre ellos) y agradables para el usuario es 1280x720. Podemos ver las diferencias del menú principal para 3 resoluciones posibles a continuación:



Pantalla 1: 1440p (2K): 2560 × 1440.



Pantalla 2: 1080p (HD): 1920 × 1080.



Pantalla 3: 720p (HD): 1280 × 720.

5 DISEÑO

5.1 *Estructura de la aplicación*

Como ya se ha establecido anteriormente en diferentes puntos, el proyecto trata el desarrollo de un videojuego, enfocado puramente al ocio y al entretenimiento para dos personas, para una sola plataforma actualmente, el ordenador. La aplicación constará a nivel gráfico de un menú principal (con diferentes submenús) y la pantalla del juego tras la selección de TAGs. A nivel interno tendrá diferentes carpetas para almacenar ficheros, audios (“music”), fuentes de letra e imágenes (“img”), al igual que las diferentes clases (ficheros .py) que se usen para poder realizar las funcionalidades que se requieran para todos los estados del juego.

5.2 *Sistema guardado de información*

Para poder guardar los ajustes seleccionados en el “Menú Opciones” que sean usados en otro estados de la aplicación como la pantalla del juego, se usarán distintas variables que se pasarán por parámetro cuando se cambie de estado. Inherente al guardado de información, se guardarán las estadísticas de cada jugador y el resultado de cada partida en ficheros de texto cuando se entre en estado “Game Over”.

Por lo tanto, para el guardado de datos no es necesario utilizar una base de datos, lo cual supondría un gasto de mantenimiento y servicio innecesario, y se usarán variables y ficheros para guardar los datos que se necesiten para el correcto funcionamiento del juego.

5.3 *Herramientas de distribución*

La distribución del juego se hará de forma inicial a través la plataforma ‘Steam’. Para realizar la subida a dicha plataforma, habrá que seguir 6 pasos que se indican en su página web a través de ‘Steamworks’: crear una cuenta de ‘Steamworks’ y asociar una cuenta bancaria, descargar el ‘SDK de Steamworks’, crear compilaciones y repositorios de ‘Steam’, configurar las características de ‘Steamworks’, crear la página de la tienda en la que se expondrá el juego y establecer el precio (0,99€), y preparar el lanzamiento revisando lo anterior. Una vez cumplidos estos pasos, habrá que pagar una tarifa de ‘Steam Direct’ 100USD (93€ aproximadamente) para publicar la aplicación.

1. Establecer tu cuenta de Steamworks

El primer paso es registrarte para convertirte en un asociado de Steamworks.

Una vez que tengas configurada tu cuenta y se confirme la información bancaria, fiscal y empresarial, podrás añadir más usuarios y empezar a configurar tu primer producto o "aplicación".

2. Descargar el SDK de Steamworks

Una vez que completes el proceso de incorporación, necesitarás descargar el SDK de Steamworks, el cual contiene todos los scripts y plantillas para crear y cargar tu producto en Steam.

Puedes descargar la versión más reciente del SDK de Steamworks [aquí](#) y puedes leer más sobre el SDK en la [SDK de Steamworks](#) documentación.

3. Crear tus compilaciones y repositorios de Steam

Para poner en marcha tu aplicación en Steam, debes cargar una compilación y configurar algunos ajustes. Este proceso se describe en detalle en [Carga de datos en Steam](#). Tú decides cuándo quieres compilar y probar tu aplicación.

4. Configurar las características de Steamworks

Puedes usar tantas características de Steamworks como deseas. Para obtener más información, consulta [Características](#).

5. Crear tu página de la tienda y los precios

Mientras trabajas para poner en marcha tu aplicación, también puedes preparar la presencia en la tienda de tu juego o software. Si te surge alguna duda durante el proceso, puedes preguntar o buscar ayuda en los foros de la página de la tienda o de marketing. Para obtener más información, consulta [Presencia en la tienda](#).

6. Preparar el lanzamiento

La mayor parte del trabajo de preparar el producto para su lanzamiento en Steam puedes realizarlo tú mismo con las herramientas y la documentación. No obstante, al acercarse el lanzamiento, Valve tendrá que revisar tu página de la tienda, el precio propuesto y la compilación del juego.

Ya sea que estás completando el proceso de registro en Steam Direct o que ya eres un desarrollador establecido de Steamworks, ahora basta con pagar una tarifa de 100 USD (o la cantidad equivalente) por cada nueva aplicación que deseas distribuir en Steam. Los detalles sobre este proceso se encuentran a continuación.

5.4 Herramientas de desarrollo

El código se ha desarrollado únicamente en 'Visual Studio Code' (en 'Python'), con la ayuda de la documentación que proporciona la página de 'PyGame' y 'StackOverflow'.

Por otro lado, las imágenes no tienen *Copyright* debido a que se han creado con el generador de imágenes con IA de Bing ('Copilot | Designer'), a excepción del fondo del juego, el cual ha sido adquirido a través del banco de imágenes gratuitas 'FreePik'. Se han editado las imágenes con tres herramientas de navegador: 'lloveimg' (reescalar imágenes), 'Photopea' (editar capas imágenes) y 'Remove.bg' (quitar fondo).

Por último, la música de los menús y del juego se ha sacado de un vídeo de 'Youtube', cuyo creador permite su uso para cualquier proyecto de forma gratuita. A su vez, los efectos de sonido se han creado grabando sonidos con un micrófono para luego ser editados con 'Mp3cut' con la finalidad de recortar su duración para implementarlo en la clase pygame.mixer.Sound("ruta").

6 DESARROLLO

6.1 Estados

Los estados en los que se puede encontrar la aplicación son los siguientes:

1. Menú “Principal”

1.1. Menú “Selección de TAGs”

1.1.1. Pantalla “Juego”

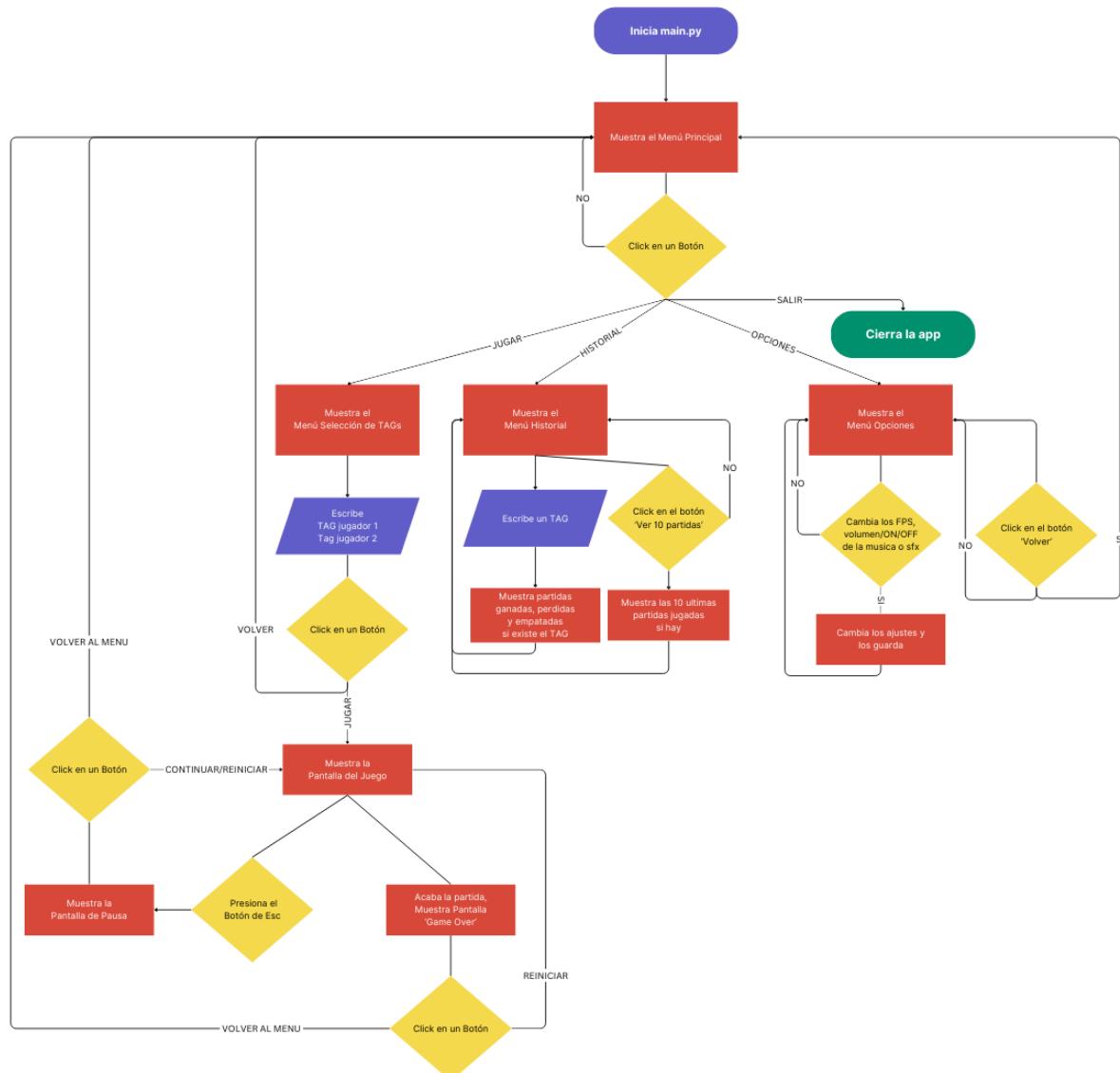
1.1.2. Pantalla “Pausa”

1.1.3. Pantalla “Game Over”

1.2. Menú “Opciones”

1.3. Menú “Historial”

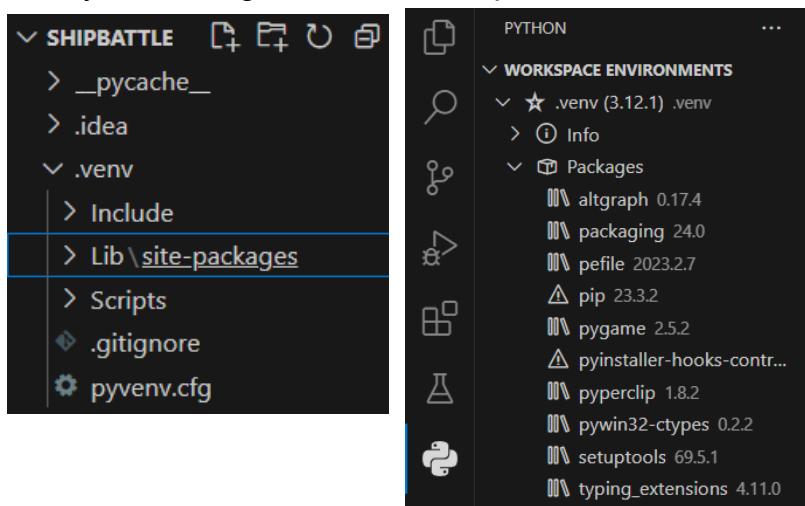
A continuación, se adjunta un diagrama de flujo el cual muestra los diferentes estados. Además de los eventos que surgen y se necesitan para cambiar a cada estado.



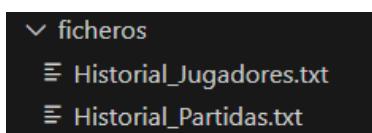
6.2 Carpetas y archivos

Durante el desarrollo del proyecto, se ha organizado en carpetas los diferentes archivos que se van a utilizar dentro del proyecto para poder implementar el uso de rutas relativas cuando se quiera acceder a dichos archivos para que se pueda instalar en juego donde quiera el usuario.

Las primeras carpetas que encontramos dentro del proyecto corresponden a una extensión del ‘Visual Studio Code’ conocida como ‘Python Environment Manager’. Esta extensión permite trabajar en un proyecto sin necesidad de descargarte los paquetes que se necesiten en el propio ‘Python’ del ordenador, sino en un “entorno” propio. Los paquetes que se quieran descargar para el proyecto, se guardan en la carpeta “.venv”, para poder seleccionarla posteriormente como “Entorno de trabajo” (*Workspace Environment*) y permitir al desarrollador trabajar en dicho proyecto con los paquetes que se hayan descargado en dicha carpeta.

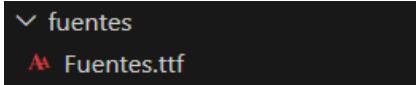


Posteriormente, se encuentra la carpeta de “ficheros”, en la que se guardan dos ficheros de texto para guardar datos. En el fichero “Historial_Jugadores.txt” se guardan los resultados de partidas ganadas, perdidas y empatadas que tiene cada jugador con formato “TAG:NumPartidasGanadas:NumPartidasPerdidas:NumPartidasEmpatadas:”, para poder posteriormente ser utilizado por el menú “Historial”. En el otro fichero, “Historial_Partidas.txt”, se guarda el historial de todas las partidas que se juegan junto con su resultado, la fecha y la hora a la que se termina cada partida, para poder mostrar las 10 últimas partidas en el menú “Historial”.

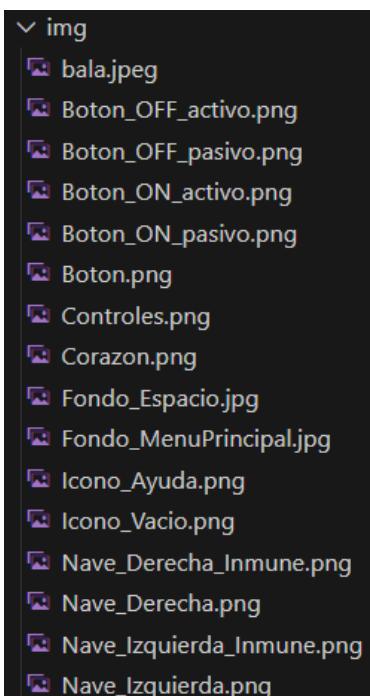


A continuación, la carpeta “fuentes” contiene todo lo necesario para aplicar un mismo tipo de fuente (*American Captain*) a todo el texto que se cree dentro de la aplicación, principalmente con un método propio que devuelve un pygame.font.Font(“fuente.ttf”, tamaño) para simplificar el código.

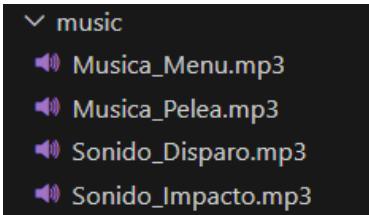
```
def get_font(self, size): #para obtener el tipo de fuente  
    return pg.font.Font("fuentes/Fuentes.ttf",size)
```



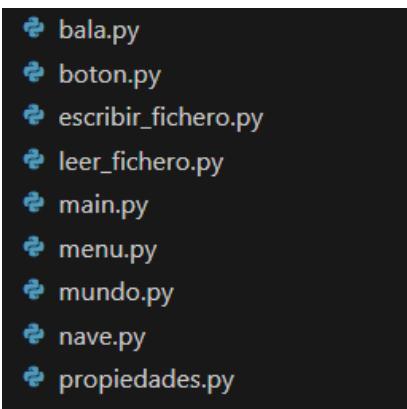
Justo debajo, en la carpeta “img”, se encuentran todas las imágenes que se utilizan para la interfaz del juego, ya sea para los fondos o los diseños de los *sprites*. La primera imagen se utiliza para los *sprites* de las balas que generan las naves. Las siguientes cuatro imágenes corresponden a los botones que se encuentran dentro del menú “Opciones” para activar o desactivar la música y los efectos de sonido. La imagen “Boton.png” se utiliza de fondo en el resto de los botones para los menús. Para mostrar los controles que tiene cada jugador, se ha creado una imagen (“Controles.png”) a modo de ayuda que se muestra al mantener pulsado el ícono de ayuda (“Icono_Ayuda.png”). Durante la partida, se muestra en todo momento la cantidad de vidas que tiene cada jugador en la parte inferior de la pantalla junto con la imagen “Corazon.png”. El fondo que se utiliza para el menú “Principal” es la imagen “Fondo_MenuPrincipal.png”, y para el resto de los fondos (para los submenús y el juego) se usa “Fondo_Espacio.jpg”. Por último, están las imágenes que se utilizan en los *sprites* de las naves, diferenciando entre las imágenes de la nave izquierda y de la derecha para cada estado de inmunidad (cuando es inmune a las balas y cuando no lo es).



La última carpeta es “music”, en la que se han puesto todos los efectos de sonido (“Sonido_Disparo.mp3” para el sonido de las balas, y “Sonido_Impacto.mp3” para cuando hay cualquier tipo de colisión) y música que se reproduce en el menú “Principal” y sus submenús (“Musica_Menu.mp3”), así como la que suena durante el estado de “Juego” (“Musica_Pelea.mp3”).



El resto de los archivos son ficheros de extensión “.py”, los cuales contienen las clases y métodos que emplean los archivos que sean necesarios para poder ejecutar y tener un funcionamiento correcto del juego.



6.3 Clases y métodos

Dentro de cada clase se ha comentado y limpiado el código durante y al final del desarrollo para aumentar la legibilidad del código. A continuación se explica de forma detallada el funcionamiento y estructura de las clases (*imports*, métodos...) que contiene la aplicación, en caso de que los comentarios sean insuficientes para entender el código. Muchas de las variables se inicializan en el constructor de cada clase (`def __init__(self):`) y se establecen como estáticas (`self.`) para que se puedan modificar y utilizar dentro de los métodos.

→ **propiedades.py**

El primer fichero que se ha creado es el que contiene las variables globales y constantes que definen las características del tamaño de pantalla (automáticamente), tamaño de la nave y sus balas, y la velocidad a la que se mueven las naves.

-Imports:

El único *import* necesario para este fichero es ‘tkinter’, para recibir el ancho y alto de la pantalla en la que se ejecuta el programa y guardarla en una variable.

```
1 import tkinter as tk
```

-Variables:

Se crea una variable inicial de tipo Tk (‘tkinter’) para usar los métodos .winfo_screenwidth() y .winfo_screenheight() para obtener y guardar el alto y ancho de la pantalla del usuario. También se crean variables para definir el tamaño de la nave (40x40 píxeles), el tamaño de las balas (10x5 píxeles), y la velocidad de la nave (número de píxeles que se mueve cuando se desplaza la nave).

```
3 #creo una variable de tkinter para obtener la resolucion de la pantalla
4 root = tk.Tk()
5 ANCHO_PANTALLA = root.winfo_screenwidth()
6 ALTO_PANTALLA = root.winfo_screenheight()
7 #ANCHO_PANTALLA = 1280 #var para probar resolucion 720p
8 #ALTO_PANTALLA = 720
9
10 ANCHO_NAVE, ALTO_NAVE = 40, 40 #tamaño nave
11
12 VEL_NAVE = 5 #su vel
13
14 ANCHO_BALA, ALTO_BALA = 10, 5 #tamaño bala
```

→ main.py

La clase la cual se ejecuta para poder iniciar la aplicación es “main.py”. Esta clase tiene el método que por defecto se ejecuta al iniciar el módulo (*if __name__ == “__main__”*). Se crea un objeto de tipo “Main()” para poder ejecutar los dos estados principales: los menús y el juego (si la variable *menu_disp==True* muestra el menú, de lo contrario se ejecuta el juego). Se ejecutan en un bucle infinito del cual sale cuando se presiona la “X” para cerrar la ventana o al presionar el botón “Salir” del menú “Principal”.

```
60 if __name__ == "__main__":
61     M = Main()
62     while True: #bucle infinito para que la aplicación nunca deje de ejecutarse hasta que salga el usuario presionando X de la ventana
63         if M.menu_disp:
64             m1 = M.menu_ejecucion()
65         else:
66             m2 = M.juego_ejecucion()
```

-Imports:

En este módulo importamos varios paquetes útiles que se necesitan para ejecutar todas las funcionalidades del juego. El principal paquete que se usa en todos los módulos es obviamente “pygame”, con métodos muy útiles para el desarrollo de un videojuego, como por ejemplo el de la música (`pygame.mixer.Music()`). “sys” es un módulo necesario para permitir al usuario salir de la aplicación, en este caso al pulsar el icono “X” en la ventana.

```
1 import pygame as pg
2 import sys
3 from propiedades import ANCHO_PANTALLA, ALTO_PANTALLA
4 from mundo import Mundo
5 from menu import Menu
```

-Variables:

Para poder utilizar los métodos propios del paquete de ‘PyGame’, es necesario inicializar los componentes generales y los que tengan relación con el sonido (“mixer”). Se modifica a su vez el nombre de la ventana del juego a “Ship Battle”. Posteriormente, se crean los componentes necesarios para el fondo del juego, uno para la ventana (“pantalla_juego”) y otro para poner la imagen del fondo (“fondo_juego”). Ambos componentes se escalan con respecto al ancho y alto de la pantalla del fichero de “propiedades.py” para poder adaptar el juego automáticamente.

```
7 #inicializo los componentes de pygame y creo los componentes visuales del fondo del juego
8 pg.init()
9 pg.mixer.init()
10
11 pg.display.set_caption("Ship Battle") #nombre de la app
12 pantalla_juego = pg.display.set_mode((ANCHO_PANTALLA, ALTO_PANTALLA + 40)) #creo la ventana del juego con los datos del fichero de propiedades
13 fondo_juego = pg.transform.scale(pg.image.load("img/Fondo_Espacio.jpg"), (ANCHO_PANTALLA, ALTO_PANTALLA + 40)) #y para el fondo del juego
```

-Métodos:

`def __init__(self):`

En el constructor de esta clase, se inicializan variables que se modifican dentro del estado menú “Opciones” (los *FPS* [60 por defecto], el volumen de la música y el volumen de los efectos [al 100% ambos por defecto]), así como los *TAGs* que se introducen en el menú “Selección de *TAG*” para poder pasar dichas variables al estado del “Juego”. Por otro lado, se inicializa un objeto fundamental para poder establecer los *FPS* del juego (`self.FPS = pg.time.Clock()`).

```

16     def __init__(self): #constructor
17         self.tag_j1 = ""
18         self.tag_j2 = ""
19         #cambio de estado
20         self.menu_disp = True #inicia la pantalla en el menu
21         #opciones
22         self.cantidad_FPS = 60 #inicialmente el juego tendrá 60 fps
23         self.volumen_musica = 1.0 #y la musica y efectos de sonido estarán al 100% de volumen
24         self.volumen_sfx = 1.0
25         #objetos para los fps
26         self.FPS = pg.time.Clock()

```

def menu_ejecución(self):

Este es el método que se ejecuta en el “main” al iniciar la aplicación. Al principio crea un objeto tipo “Menu()”, y se le pasa por parámetro la cantidad de *FPS*, el volumen que tiene el objeto pg.mixer.music (volumen de la música redondeado a un decimal), y el volumen que tienen los SFX. Posteriormente, entra en el estado de menú “Principal” (en el que se empieza en la clase “Menu()”) y los cambios que se realizan dentro de la clase en las variables relacionadas con los ajustes del menú “Opciones”. Por último se comprueba si el usuario quiere entrar en el estado “Juego” y se actualiza la pantalla.

```

47     def menu_ejecucion(self): #ejecución menú
48         menu = Menu(self.cantidad_FPS, round(pg.mixer.music.get_volume(), 1), self.volumen_sfx) #se guardan los ajustes elegidos
49
50         while self.menu_disp:
51             menu.mostrar_menu()
52             #valores recibidos en el menú
53             self.cantidad_FPS = menu.FPS
54             self.tag_j1 = menu.tag_j1
55             self.tag_j2 = menu.tag_j2
56             #guarda los ajustes seleccionados en el menu de opciones
57             self.volumen_sfx = menu.volumen_sfx
58             self.menu_disp = menu.menu_disp #comprueba si la variable de la clase mostrar menú, se ha puesto en false para salir
59             pg.display.update()

```

def juego_ejecución(self):

Esta función crea inicialmente un objeto de tipo “Mundo”, al cual se le pasa por parámetro la pantalla en la que se van a colocar todos los componentes visuales (botones, *sprites*...), los *TAGs* de cada jugador, y el volumen escogido para los efectos de sonido. Todos los “while” (a excepción del que se encuentra en el “`__main__`:”) indican en que estado se encuentra la aplicación. En este caso, mientras que no se encuentre en los estados de menús, se ejecuta el estado del juego. Se imprime en todo momento en la variable de la pantalla la imagen del fondo con `pantalla_juego.blit(fondo_juego,(0,0))`. Posteriormente se comprueba si se ha intentado cerrar la ventana para cerrar la ventana, y si es el caso desinicializa los componentes de ‘PyGame’ y cierra el intérprete de ‘Python’. Se añaden a continuación los métodos necesarios para que el usuario interactúe con el juego (mover la nave y disparar) junto con la actualización de pantalla y el volcado de los elementos actualizados. Por último se comprueba si se manda una señal a través de lo que devuelve un método (`mostrarmenu()`) de la clase “Mundo”.

```
27
28     def juego_ejecucion(self): #ejecución de juego
29         mundo = Mundo(pantalla_juego,self.tag_j1,self.tag_j2, self.volumen_sfx)
30
31         while not self.menu_disp: #mientras que no enseñe el menú, muestra el juego
32             pantalla_juego.fill("black")
33             pantalla_juego.blit(fondo_juego,(0,0))
34
35             for event in pg.event.get(): #detecta si se presiona la X de la ventana para salir
36                 if event.type == pg.QUIT:
37                     pg.quit()
38                     sys.exit()
39
40             mundo.nave_mov() #métodos para que se pueda interactuar con el juego y actualizar pantalla/sprites
41             mundo.nave_disp()
42             mundo.update()
43             pg.display.update()
44             self.FPS.tick(self.cantidad_FPS)
45             self.menu_disp = mundo.mostrarmenu() #comprueba si el juego ha acabado
```

→ **menu.py**

Cuando entra en la clase “Menu”, se empieza en el estado menú “Principal”, del cual se puede transitar al resto de estados menú. Todos los elementos se colocan utilizando posiciones relativas al tamaño de pantalla que se ejecuta la aplicación.

-Imports:

Para esta clase se importan varios paquetes: pygame (para los componentes visuales), sys (poder cerrar la ventana), resolución de pantalla del fichero “propiedades.py”, la clase “Boton”, y la clase que lee los ficheros “.txt” (“Leer_Fichero”).

```
1 import pygame as pg
2 import sys
3 from propiedades import ALTO_PANTALLA, ANCHO_PANTALLA
4 from boton import Boton
5 from leer_fichero import Leer_Fichero
```

-Variables:

La única variable global necesaria para casi todos los métodos de la clase es “pantalla_menú”, para poder dibujar los componentes visuales y establecer los diferentes fondos para cada menú.

```
7 #creo la variable necesaria para poner los componentes en pantalla y los fondos
8 pantalla_menu = pg.display.set_mode((ANCHO_PANTALLA, ALTO_PANTALLA + 40))
```

-Métodos:

def __init__(self, fps, vol_musica, vol_sfx):

Para el constructor se dividen las variables que comparten los métodos según el menú que los modifica. Las primeras variables son aquellas que establecen el cambio de estado entre menús, y la que manda la “señal” al “__main__:” para que cambie de estado al “Juego” (self.menu_disp). También se crea una variable para la música en la cual se carga “Musica_Menu.mp3”.

Más adelante se crean las variables para la selección de TAGs, tanto los *strings* para guardar los nombres, como el *string* que muestra los mensajes de error, y las variables que permiten escribir en cada rectángulo dentro del menú y el color que indica si se está escribiendo.

A continuación, se inicializan las variables que se reciben por parámetro para los ajustes que se tenían guardados anteriormente. Dichas variables se modificarán en el menú “Opciones” cuando el usuario interactúe con los elementos relacionados con cada ajuste.

Por último se crea un objeto tipo “Leer_Fichero” para poder leer los datos que se hayan guardado en los ficheros y se crea una variable para mostrar las 10 últimas partidas.

```

11     def __init__(self, fps, vol_musica, vol_sfx): #constructor
12         #variables menú principal
13         self.menu_disp = True
14         self.menuPrincipal_disp = True #var para mostrar el menú principal
15         self.menuTag_disp = False #var para mostrar el menú de selección de tags
16         self.menuOpciones_disp = False #var para mostrar el menú de opciones
17         self.menuHistorial_disp = False #var para mostrar el menú de historial
18         self.musica = pg.mixer.music
19         self.musica.load("music\Musica_Menu.mp3") #carga el .mp3
20
21         #variables menú TAG
22         self.tag_j1 = "AAA" #var para recibir los tags escritos
23         self.tag_j2 = "BBB"
24         self.texto_err_tag = "" #texto para mostrar error en la selección de tag
25         self.escribir_tag1 = False #y para permitir escribir el tag1 o el tag2
26         self.escribir_tag2 = False
27         self.color_rectangulo1 = pg.Color('navy') #var para definir el color del rectángulo de selección de tag
28         self.color_rectangulo2 = pg.Color('navy') #var para definir el color del rectángulo del segundo tag
29
30         #variables menú opciones
31         self.FPS = fps
32         self.volumen_musica = vol_musica
33         self.volumen_sfx = vol_sfx
34
35         #objeto tipo Leer_Fichero para el menu_historial y
36         self.leer_fich = Leer_Fichero()
37         self.texto_hist_10_partidas = "" #ver el historial de 10 partidas

```

def mostrar_menu(self):

Esta función es la encargada de gestionar los estados de menú en los que se encuentra la aplicación. Al entrar a la función, inicia la música en un bucle infinito para que no deje de sonar mientras se esté en cualquiera de los menús.

Para cada estado, se contiene el método que tiene las funcionalidades y los componentes visuales en un bucle “while”, del cual sale cuando la variable que controla si se encuentra en ese estado se pone a “False” (`self.menuEstado_disp`). La única peculiaridad que tiene uno de los bucles es para el menú “Selección de TAGs”, en el que se vacían los *strings* de los TAGs para poder seleccionar otros nuevos.

```

39     def mostrar_menu(self): #BUCLE MUESTRA CADA MENU CUANDO SE LE INDICA menuXXXX_disp = True
40         self.musica.play(-1, 0.0) #para iniciar la música en un bucle infinito (loops, start_point)
41         while self.menu_disp:
42             while self.menuPrincipal_disp:
43                 if self.tag_j1.__len__() != 0: #vacía los tags anteriores cuando vuelve al menú principal
44                     self.tag_j1 = ""
45                 if self.tag_j2.__len__() != 0:
46                     self.tag_j2 = ""
47                 self.mostrar_Principal()
48             while self.menuTag_disp: #menu selección tag
49                 self.mostrar_Tag()
50             while self.menuOpciones_disp: #menu opciones
51                 self.mostrar_Opciones()
52             while self.menuHistorial_disp: #menu historial
53                 self.mostrar_Historial()
54             self.menuPrincipal_disp = True #si sale del último bucle, volverá a mostrar el menú principal

```

Para explicar cada menú, se va a explicar en profundidad solamente los elementos que no se comparten entre ellos o los que no hayan sido ya explicados, para no repetir explicaciones innecesarias.

def mostrar_Principal(self):

Al iniciar cada estado, se pone la imagen que se requiera en la variable global “pantalla_menu” con respecto al tamaño de pantalla que tenga el usuario. También se crea una variable para detectar la posición del ratón y las colisiones con el resto de componentes.

```
56     def mostrar_Principal(self): #INTERFAZ MENU PRINCIPAL
57         pantalla_menu.blit(pg.transform.scale(pg.image.load("img\Fondo_MenuPrincipal.jpg"),(ANCHO_PANTALLA, ALTO_PANTALLA + 40)), (0, 0))
58         MENU_POS_RATON = pg.mouse.get_pos() #detecta pos del mouse
```

En el menú “Principal”, se dibuja el nombre del juego y el nombre del autor con un color y tamaño de fuente diferentes, con el uso de un método que se explicará más adelante de todos los menús. Posteriormente, se crea un rectángulo invisible sobre la superficie del texto para poder dibujarlo en pantalla (pantalla_menu.blit(nombre, rect)).

```
60     #dibuja textos sin usar la funcion self.escribir_texto() porque son ligeramente diferentes a los textos 'default'
61     nombre_autor = self.get_font(25).render("JORGE CORREYERO FERNANDEZ", True, "#b68f40")
62     nombre_juego = self.get_font(150).render("SHIP BATTLE", True, "#FFFFFF")
63     nombre_autor_rect = nombre_autor.get_rect(centerx=ANCHO_PANTALLA-150, centery=ALTO_PANTALLA-20)
64     nombre_juego_rect = nombre_juego.get_rect(centerx=ANCHO_PANTALLA//2, centery=100)
65
66     pantalla_menu.blit(nombre_autor,nombre_autor_rect)
67     pantalla_menu.blit(nombre_juego,nombre_juego_rect)
```

Después de dibujar los textos, se crean variables para cada botón con la ayuda de la clase que se ha creado e importado anteriormente. Para crear cada botón se le debe pasar los siguientes parámetros: URL imagen de fondo, posición ‘x’ e ‘y’, texto que tenga encima del fondo, tamaño de la letra que se quiera para el texto anterior, el color base del texto y el color al que cambie cuando se tenga el ratón encima del botón (*Hover*).

```
69     #dibuja botones
70     BTN_PLAY = Boton(imagen=pg.image.load("img/Boton.png"), pos=(ANCHO_PANTALLA // 2, ALTO_PANTALLA // 2 -125),txt_entr="JUGAR",font=self.get_font(50), base_color="White", color_flot="Green")
71     BTN_SALIR = Boton(imagen=pg.image.load("img/Boton.png"), pos=(ANCHO_PANTALLA // 2, ALTO_PANTALLA // 2+250),txt_entr="SALIR", font=self.get_font(50),base_color="White", color_flot="Green")
72
73     BTN_OPCIONES = Boton(imagen=pg.image.load("img/Boton.png"), pos=(ANCHO_PANTALLA // 2, ALTO_PANTALLA // 2),txt_entr="OPCIONES",font=self.get_font(50), base_color="White", color_flot="Green")
74     BTN_HIST = Boton(imagen=pg.image.load("img/Boton.png"), pos=(ANCHO_PANTALLA // 2, ALTO_PANTALLA // 2+125),txt_entr="HISTORIAL",font=self.get_font(50), base_color="White", color_flot="Green")
```

A diferencia del texto, todos los botones se dibujan en pantalla con su método *update* y el color del texto se cambia con respecto a la posición del ratón.

```
77     for boton in [BTN_PLAY, BTN_SALIR, BTN_HIST, BTN_OPCIONES]: #hover para los botones y cambiarlos de color
78         boton.cambiarColor(MENU_POS_RATON)
79         boton.update(pantalla_menu)
```

Para detectar los eventos que se producen dentro del menú, se utiliza el método propio del paquete ‘PyGame’, pg.event.get(). Una vez se han guardado los eventos que ocurren, se comprueban los siguientes escenarios: si se ha pulsado “X” para cerrar la ventana, si se ha hecho *click* en alguno de los botones para acceder al resto de los estados de menú (poniendo a “True” la variable necesaria para cambiar al estado siguiente, y a “False” la variable del estado actual), o si se ha hecho *click* en el botón “Salir” para cerrar la ventana. Siempre que se salga de la aplicación, desinicializamos los componentes de ‘PyGame’ para asegurar que se cierra correctamente y no se dejan tareas en segundo plano que consuman recursos al usuario.

```
81     eventos = pg.event.get() #comprueba si se pulsa X para salir o el boton del raton
82     for event in eventos:
83         if event.type==pg.QUIT:
84             pg.quit()
85             sys.exit()
86         if event.type==pg.MOUSEBUTTONDOWN: #detecta si ha clickado algún botón
87             if BTN_PLAY.verInputs(MENU_POS_RATON):
88                 self.menuPrincipal_disp = False
89                 self.menuTag_disp = True
90             if BTN_OPCIONES.verInputs(MENU_POS_RATON):
91                 self.menuPrincipal_disp = False
92                 self.menuOpciones_disp = True
93             if BTN_HIST.verInputs(MENU_POS_RATON):
94                 self.menuPrincipal_disp = False
95                 self.menuHistorial_disp = True
96             if BTN_SALIR.verInputs(MENU_POS_RATON):
97                 pg.quit()
98                 sys.exit()
```

Por último, se utiliza el método (que se explicará más adelante) que crea y permite al usuario interactuar con él, antes de actualizar todos los componentes del menú.

```
100     self.cuadrado_ayuda(MENU_POS_RATON, eventos) #ICONO AYUDA
101
102     pg.display.update()
```

def mostrar_Tag(self):

Se crean los componentes visuales (texto y botones). Aparte, se crean dos variables tipo pygame.Color('color') para asignar colores pasivo y activo (si no está escribiendo o si sí lo está) para los rectángulos de escritura. También se crean dichos rectángulos con el método self.dibujar_rect(pos x, pos y, anchura, altura, color, tamaño borde) y se guarda en una variable para detectar si se hace *click* con el ratón sobre alguno de ellos más adelante.

```
104 def mostrar_Tag(self): #INTERFAZ MENU SELECCION TAGS
105     pantalla_menu.fill("black")
106     pantalla_menu.blit(pg.image.load("img\Fondo_Espacio.jpg"), (0, 0)) #variable para el fondo del menú, posicion inicial img->(0, 0)
107     MENU_POS_RATON = pg.mouse.get_pos() #detecta pos del mouse
108
109     #dibuja botones
110     BTN_PLAY = Boton(imagen=pg.image.load("img\Boton.png"), pos=(ANCHO_PANTALLA // 2, ALTO_PANTALLA // 2+125),txt_entr="JUGAR",font=self.get_font(50))
111     BTN_VOLVER = Boton(imagen=pg.image.load("img\Boton.png"), pos=(ANCHO_PANTALLA // 2, ALTO_PANTALLA // 2+250),txt_entr="VOLVER", font=self.get_font(50))
112
113     color_activo = pg.Color('white') #rect seleccionado, escribiendo
114     color_pasivo = pg.Color('navy') #rect NO seleccionado, no puede escribir
115
116     #dibuja texto para pedir los TAG
117     self.escribir_texto("TAG JUGADOR 1:",80, ANCHO_PANTALLA // 5, ALTO_PANTALLA // 2 -350)
118     self.escribir_texto(self.tag_j1,80, ANCHO_PANTALLA // 5, ALTO_PANTALLA // 2 -250) #cuadrado input tag1
119     rect_tag1 = self.dibujar_rect(ANCHO_PANTALLA // 5 -10, ALTO_PANTALLA // 2 -260, 170,80, self.color_rectangulo1, 2)
120
121     self.escribir_texto("TAG JUGADOR 2:",80, ANCHO_PANTALLA*3 // 5, ALTO_PANTALLA // 2 -350)
122     self.escribir_texto(self.tag_j2,80, ANCHO_PANTALLA*3 // 5, ALTO_PANTALLA // 2 -250) #cuadrado input tag2
123     rect_tag2 = self.dibujar_rect(ANCHO_PANTALLA*3 // 5 -10, ALTO_PANTALLA // 2 -260, 170,80, self.color_rectangulo2, 2)
```

Justo después, se modifica el mensaje de error, indicando si al usuario si ha escogido el mismo *TAG* para ambos jugadores o si ha dejado vacío alguno de ellos. Tras asignar el mensaje de error, se escribe en pantalla con el mismo método que el resto.

```
125 #texto error: APARECE AUTOMATICAMENTE CUANDO SE INTENTA CLICKAR EN JUGAR NO SE HA COMPLETADO ALGUNO DE LOS 2 TAG o SI SE HA SELECCIONADO EL MISMO
126 if self.tag_j1 == self.tag_j2:
127     self.texto_err_tag = "Escoja TAGs distintos para cada jugador."
128 elif self.tag_j1 == "" or self.tag_j2 == "":
129     self.texto_err_tag = "Escriba ambos TAGs antes de continuar."
130 else:
131     self.texto_err_tag = ""
132 self.escribir_texto(self.texto_err_tag,40, ANCHO_PANTALLA // 3, ALTO_PANTALLA // 2 -50)
```

A continuación, se dibujan los botones y se comprueban los eventos.

```
134 for boton in [BTN_PLAY, BTN_VOLVER]: #hover para los botones y cambiarlos de color
135     boton.cambiarColor(MENU_POS_RATON)
136     boton.update(pantalla_menu)
137
138 eventos = pg.event.get()
139 for event in eventos:
140     if event.type==pg.QUIT: #comprueba si se pulsa X para salir o el boton del raton
141         pg.quit()
142         sys.exit()
```

Dentro de los eventos, se comprueba si se ha pulsado alguna tecla (pygame.KEYDOWN). Si está activo alguno de los rectángulos para escribir el *TAG*, escribe la tecla que haya presionado el usuario y la guarda en la variable que corresponda al jugador en mayúsculas. Previamente, se comprueba que no se haya pulsado las teclas ‘tabulador’ ni ‘enter’ ni ‘espacio’ (pygame.K_TAB, pygame.K_RETURN, pygame.K_SPACE respectivamente) para que no pueda añadir espacios en blanco, y si se ha pulsado la tecla para borrar, elimina la última letra que tenga el *TAG*.

```

144 if event.type==pg.KEYDOWN: #detecta si se pulsan teclas
145     if self.escribir_tag1:
146         if event.key == pg.K_BACKSPACE: #si es la tecla de borrar
147             self.tag_j1 = self.tag_j1[0:-1] #borra el ultimo caracter
148         elif self.tag_j1.__len__() < 3 and event.key != pg.K_TAB and event.key != pg.K_SPACE and event.key != pg.K_RETURN: #si no ha superado
149             self.tag_j1 += event.unicode
150             self.tag_j1 = self.tag_j1.upper() #lo que escribe lo pone a mayúsculas
151
152     if self.escribir_tag2: #igual con el segundo tag si tiene seleccionada la caja
153         if event.key == pg.K_BACKSPACE: #si es la tecla de borrar
154             self.tag_j2 = self.tag_j2[0:-1] #elimina la ultima letra
155         elif self.tag_j2.__len__() < 3 and event.key != pg.K_TAB and event.key != pg.K_SPACE and event.key != pg.K_RETURN:
156             self.tag_j2 += event.unicode
157             self.tag_j2 = self.tag_j2.upper()

```

Otros eventos que se detectan son los *clicks* del ratón, los cuales pueden ser en alguno de los botones, en los rectángulos para escribir los *TAGs* o fuera de todos los elementos. Cuando el usuario presiona uno de los rectángulos, le cambia el color a activo y el color del otro rectángulo a pasivo, para mostrar al usuario que está escribiendo. También se habilita o deshabilita la escritura del rectángulo con la variable `self.escribir_tagX`.

```

159 if event.type==pg.MOUSEBUTTONDOWN:
160     if rect_tag1.collidepoint(event.pos): #detecta si la caja del tag 1 se ha seleccionado
161         self.escribir_tag1 = True #habilita la escritura para el tag1
162         self.escribir_tag2 = False #y la deshabilita la escritura para el tag2
163         self.color_rectangulo1 = color_activo #muestra que está seleccionado el rectángulo para el tag1 poniéndolo en blanco
164         self.color_rectangulo2 = color_pasivo #y desactiva el otro poniéndolo en azul
165     elif rect_tag2.collidepoint(event.pos): #o si se selecciona la de tag 2 lo mismo pero para los elementos del tag2
166         self.escribir_tag2 = True
167         self.escribir_tag1 = False
168         self.color_rectangulo2 = color_activo
169         self.color_rectangulo1 = color_pasivo
170     else: #de lo contrario, se ha clickado fuera de ambas, se ponen en pasivo (no escritura) ambas y deshabilita las escrituras
171         self.escribir_tag1 = False
172         self.escribir_tag2 = False
173         self.color_rectangulo1 = color_pasivo
174         self.color_rectangulo2 = color_pasivo
175
176     if BTN_PLAY.verInputs(MENU_POS_RATON) and self.tag_j1.__len__() > 0 and self.tag_j2.__len__() > 0 and self.tag_j1 != self.tag_j2:
177         self.menu_disp = False
178         self.menuPrincipal_disp = False
179         self.menuTag_disp = False
180     if BTN_VOLVER.verInputs(MENU_POS_RATON):
181         self.menuPrincipal_disp = True
182         self.menuTag_disp = False

```

Se añade al final el método para el ícono de ayuda y el `pg.display.update()` para actualizar la pantalla.

def mostrar_Opciones(self):

Se establece el fondo de pantalla, y se crean variables para detectar la posición del ratón y cambiar el color de los rectángulos (pasivo y activo).

Los ajustes que se pueden modificar son los siguientes:

Selección FPS: Se comprueba la cantidad de *FPS* que tiene en todo momento la aplicación para establecer el color activo al rectángulo que corresponda (y pasivo al resto) para mostrar al usuario cuál tiene seleccionado. También se crea y pinta en pantalla el texto para la selección de *TAGs*, así como los rectángulos que rodea cada número.

```
188 def mostrar_Opciones(self): #INTERFAZ MENU OPCIONES (FPS, VOLUMEN MUSICA, VOLUMEN EFECTOS, VIDAS JUGADOR)
189     pantalla_menu.blit(pg.image.load("img/Fondo_Espacio.jpg"), (0, 0)) #variable para el fondo del menú, posicion inicial img->(0, 0))
190     MENU_POS_RATON = pg.mouse.get_pos() #detecta pos del mouse
191
192     color_activo = pg.color('white') #fps seleccionados
193     color_pasivo = pg.color('black') #fps NO seleccionados (no se ve el rectangulo)
194
195     #SELECCION FPS
196     match(self.FPS): #mostrar qué fps están seleccionados con un cuadrado blanco
197         case 30:
198             color_rectangulo30 = color_activo
199             color_rectangulo60 = color_pasivo
200             color_rectangulo120 = color_pasivo
201         case 60:
202             color_rectangulo30 = color_pasivo
203             color_rectangulo60 = color_activo
204             color_rectangulo120 = color_pasivo
205         case 120:
206             color_rectangulo30 = color_pasivo
207             color_rectangulo60 = color_pasivo
208             color_rectangulo120 = color_activo
209
210     #dibujo los textos para la seleccion de fps
211     self.escribir_texto("FPS:",80, ANCHO_PANTALLA // 3, ALTO_PANTALLA // 2 -350)
212     self.escribir_texto("30",80, ANCHO_PANTALLA // 3 +150, ALTO_PANTALLA // 2 -350)
213     self.escribir_texto("60",80, ANCHO_PANTALLA // 3 +230, ALTO_PANTALLA // 2 -350)
214     self.escribir_texto("120", 80, ANCHO_PANTALLA // 3 +320, ALTO_PANTALLA // 2 -350)
215     #rectangulos de seleccion fps
216     rect_30 = self.dibujar_rect(ANCHO_PANTALLA // 3 +145,ALTO_PANTALLA // 2 -355, 70,70, color_rectangulo30, 2) #pos x, pos y, tam x, tam y, color,
217     rect_60 = self.dibujar_rect(ANCHO_PANTALLA // 3 +225, ALTO_PANTALLA // 2 -355, 80,70, color_rectangulo60, 2)
218     rect_120 = self.dibujar_rect(ANCHO_PANTALLA // 3 +315, ALTO_PANTALLA // 2 -355, 100,70,color_rectangulo120, 2)
```

Selección Música ON/OFF: se establece una imagen para cada botón (“ON” y “OFF”) según su estado (activo o pasivo), y se guarda en una variable teniendo en cuenta si el volumen de la música es igual a 0 (si está apagada, “OFF” está activo). Posterior a la elección de imagen, se crean los botones “ON” y “OFF” con su imagen en el estado correspondiente, y se dibuja el texto que indica pará que ajuste son los botones.

```
#SELECCION MUSICA ON/OF
if self.volumen_musica > 0: #establece la imagen de los botones ON/OFF según si el sonido está desactivado (es 0) o no
    imagen_ON_musica = pg.image.load("img/Boton_ON_activo.png")
    imagen_OFF_musica = pg.image.load("img/Boton_OFF_pasivo.png")
else:
    imagen_ON_musica = pg.image.load("img/Boton_ON_pasivo.png")
    imagen_OFF_musica = pg.image.load("img/Boton_OFF_activo.png")
#uso botones para la seleccion de ON/OFF
self.escribir_texto("MUSICA:",80, ANCHO_PANTALLA // 3 - 100, ALTO_PANTALLA // 2 -250)
BTN_ON_MUSICA = Boton(imagen_ON_musica, pos=(ANCHO_PANTALLA // 2 -50, ALTO_PANTALLA // 2 -220),txt_entr="", font=self.get_font(50),base_color="White")
BTN_OFF_MUSICA = Boton(imagen_OFF_musica, pos=(ANCHO_PANTALLA // 2 +50, ALTO_PANTALLA // 2 -220),txt_entr="", font=self.get_font(50),base_color="White")
```

Selección volumen Música: la selección porcentual de volumen se ha creado a partir de rectángulos, siendo el volumen +10% por cada rectángulo activo. Primero se escribe el texto que indica lo que modifica esta parte de las opciones. Segundo, se establece la posición inicial (eje de las “x”) que va a tener el rectángulo de cada 10% de volumen (pos_rect = pos_rect + [50*num_rect]). El número de rectángulos activos es 10*volumen_música (self.volumen_musica es un valor entre 1.0 y 0.0), y el número de rectángulos pasivos es el resto de 10 menos los que haya activos. Para dibujar cada rectángulo, se utiliza un bucle “while” (para los rectángulos activos y luego los pasivos) en el que se dibuja cada rectángulo y se añade un array de rects para detectar posteriormente cuándo se *clickea* en cada uno. Finalmente, a la derecha de los cuadrados, se indica la cantidad de volumen (en porcentaje) que tiene en todo momento.

```

233 #SELECCION VOLUMEN MUSICA
234 self.escribir_texto("VOLUMEN MUSICA:",80, ANCHO_PANTALLA // 3 -340, ALTO_PANTALLA // 2 -150)
235
236 pos_rect = 150
237 num_rect_pasivo = int(10 - (10*self.volumen_musica)) #para poder usar el volumen_musica como contador lo paso a entero 1.0 -> 10
238 num_rect_activo = int(10*self.volumen_musica)
239 rect_vol_musica = []
240
241 #cada 10% es un rectangulo para simular una barra de volumen para la musica y los efectos
242 while num_rect_activo > 0: #añade a un array los rect activos
243     rect_vol_musica.append(self.dibujar_rect(ANCHO_PANTALLA // 3 +pos_rect, ALTO_PANTALLA // 2 -155, 50, 70, color_activo, 0)) #color = color_activo
244     pos_rect += 50 #cada vez que dibuja un rect se mueve 50px a la derecha
245     num_rect_activo -=1 #var contador act
246
247 while num_rect_pasivo > 0: #añade a un array los rect pasivos
248     rect_vol_musica.append(self.dibujar_rect(ANCHO_PANTALLA // 3 +pos_rect, ALTO_PANTALLA // 2 -155, 50, 70, color_pasivo, 0)) #color = color_pasivo
249     pos_rect += 50
250     num_rect_pasivo -=1 #var contador pas
251
252 #muestro el % de volumen en el que se encuentra la musica
253 texto_vol_musica_actual = str(int(self.volumen_musica*100)) + "%" #convertir de 10 a 100
254 self.escribir_texto(texto_vol_musica_actual,80, ANCHO_PANTALLA // 3 +700, ALTO_PANTALLA // 2 -150)

```

Selección SFX ON/OFF: se utiliza el mismo procedimiento que se usa con la música para activar y desactivar los efectos de sonido.

```

257 #SELECCION SFX ON/OFF
258 if self.volumen_sfx > 0: #establece la imagen de los botones ON/OFF según si el sonido está desactivado (es 0) o no
259     imagen_ON_sfx = pg.image.load("img/Boton_ON_activo.png")
260     imagen_OFF_sfx = pg.image.load("img/Boton_OFF_pasivo.png")
261 else:
262     imagen_ON_sfx = pg.image.load("img/Boton_ON_pasivo.png")
263     imagen_OFF_sfx = pg.image.load("img/Boton_OFF_activo.png")
264
265 self.escribir_texto("SFX:",80, ANCHO_PANTALLA // 3, ALTO_PANTALLA // 2 -50)
266 BTN_ON_SFX = Boton(imagen_ON_sfx, pos=(ANCHO_PANTALLA // 2 -50, ALTO_PANTALLA // 2 -20),txt_entr="", font=self.get_font(50),base_color="White", colo
267 BTN_OFF_SFX = Boton(imagen_OFF_sfx, pos=(ANCHO_PANTALLA // 2 +50, ALTO_PANTALLA // 2 -20),txt_entr="", font=self.get_font(50),base_color="White", co

```

Selección volumen SFX: para seleccionar el porcentaje de volumen que quiera para los efectos de sonido, se hace igual que la música pero 200 píxeles más abajo.

```

269 #SELECCION VOLUMEN SFX (igual que la musica pero -200 px abajo)
270 self.escribir_texto("VOLUMEN SFX:",80, ANCHO_PANTALLA // 3 -250, ALTO_PANTALLA // 2 +50)
271
272 pos_rect = 150
273 num_rect_pasivo = int(10 - (10*self.volumen_sfx))
274 num_rect_activo = int(10*self.volumen_sfx)
275 rect_vol_sfx = []
276
277 while num_rect_activo > 0: #añade a un array los rect activos
278     rect_vol_sfx.append(self.dibujar_rect(ANCHO_PANTALLA // 3 +pos_rect, ALTO_PANTALLA // 2 +45, 50, 70, color_activo, 0)) #color = color_activo
279     pos_rect += 50 #cada vez que dibuja un rect se mueve 50px a la derecha
280     num_rect_activo -=1 #var contador act
281
282 while num_rect_pasivo > 0: #añade a un array los rect pasivos
283     rect_vol_sfx.append(self.dibujar_rect(ANCHO_PANTALLA // 3 +pos_rect, ALTO_PANTALLA // 2 +45, 50, 70, color_pasivo, 0)) #color = color_pasivo
284     pos_rect += 50
285     num_rect_pasivo -=1 #var contador pas
286
287 #muestro el % de volumen en el que se encuentra la sfx
288 texto_vol_sfx_actual = str(int(self.volumen_sfx*100)) + "%" #convertir de 10 a 100
289 self.escribir_texto(texto_vol_sfx_actual,80, ANCHO_PANTALLA // 3 +700, ALTO_PANTALLA // 2 +50)

```

Previo a la comprobación de eventos, se crea el botón para volver al menú

“Principal” y se imprimen en pantalla todos los botones que se han creado en el método.

```

292 #boton salir del menu opciones
293 BTN_VOLVER = Boton(imagen=pg.image.load("img/Boton.png"), pos=(ANCHO_PANTALLA // 2, ALTO_PANTALLA // 2+250),txt_entr="VOLVER", font=self.get_font(50)
294
295 for boton in [BTN_VOLVER]: #hover para el btn
296     boton.cambiarColor(MENU_POS_RATON)
297     boton.update(pantalla_menu)
298 #dibuja los botones de ON/OFF en su estado actual de musica y sfx
299 BTN_ON_SFX.update(pantalla_menu)
300 BTN_OFF_SFX.update(pantalla_menu)
301 BTN_ON_MUSICA.update(pantalla_menu)
302 BTN_OFF_MUSICA.update(pantalla_menu)

```

Tras guardar los eventos que se capturan la variable “eventos = pg.event.get()”, se comprueba si se ha querido cerrar la ventana. Después, se comprueba si se ha hecho click sobre alguno de los números para cambiar la cantidad de fps.

```

304     eventos = pg.event.get() #comprueba si surgen eventos en teclado o raton
305     for event in eventos:
306         if event.type==pg.QUIT:
307             pg.quit()
308             sys.exit()
309         if event.type==pg.MOUSEBUTTONDOWN:
310             if rect_30.collidepoint(event.pos): #fps puestos a 30
311                 self.FPS = 30
312             elif rect_60.collidepoint(event.pos): #fps puestos a 60
313                 self.FPS = 60
314             elif rect_120.collidepoint(event.pos): #fps puestos a 120
315                 self.FPS = 120

```

Más adelante se comprueban todos los rectángulos que se guardaron en cada array (de música y SFX) para saber si el usuario ha hecho presionado con el ratón alguno de ellos para cambiar el volumen.

```

317 contador = 1
318 for rect_vol_mus in rect_vol_musica: #comprueba si se han clickado los cuadrados para subir/bajar volumen musica
319     if rect_vol_mus.collidepoint(event.pos):
320         self.volumen_musica = contador/10
321         pg.mixer.music.set_volume(self.volumen_musica) #establece el volumen de toda la musica cuando se cambia en el menu opciones
322         contador += 1
323
324 contador = 1 #reset de contador
325 for rect_vol_s in rect_vol_sfx: #comprueba si se han clickado los cuadrados para subir/bajar volumen sfx
326     if rect_vol_s.collidepoint(event.pos):
327         self.volumen_sfx = contador/10
328         contador += 1

```

Los últimos eventos que se comprueban son los botones para apagar y encender la música o los efectos de sonido (siendo un 0% si se apaga y un 10% del volumen si se enciende de manera predeterminada), y el botón para volver al menú principal.

```

if BTN_ON_MUSICA.verInputs(MENU_POS_RATON):
    if self.volumen_musica == 0: #si estaba apagado el sonido
        self.volumen_musica = 0.1
        pg.mixer.music.set_volume(self.volumen_musica) #establece el volumen al minimo

if BTN_OFF_MUSICA.verInputs(MENU_POS_RATON):
    self.volumen_musica = 0.0
    pg.mixer.music.set_volume(self.volumen_musica) #establece el volumen a 0

if BTN_ON_SFX.verInputs(MENU_POS_RATON):
    if self.volumen_sfx == 0: #si el volumen es nulo
        self.volumen_sfx = 0.1 #establece el volumen al minimo

if BTN_OFF_SFX.verInputs(MENU_POS_RATON):
    self.volumen_sfx = 0.0 #pone el volumen a 0

if BTN_VOLVER.verInputs(MENU_POS_RATON): #pulsa el boton volver
    self.menuPrincipal_disp = True #vuelve atras al menu principal
    self.menuOpciones_disp = False

```

Al igual que en el resto de menús, se muestra el ícono de ayuda y se actualiza la pantalla al final.

```

350     self.cuadrado_ayuda(MENU_POS_RATON, eventos)
351
352     pg.display.update()

```

def mostrar_Historial(self):

El último submenú que hay es el menú “Historial”. Se vuelve a poner el “Fondo_Espacio.jpg” y se obtiene la posición del ratón. Posteriormente, se crea el botón para volver al menú “Principal” y el que muestra las 10 últimas partidas jugadas. Al igual que en la selección de TAGs, se crean dos variables de color para los estados activo y pasivo del rectángulo de *input*. En la parte superior izquierda, se dibuja un texto para indicar al usuario que el rectángulo que se dibuja al lado es para introducir un TAG y mostrar su resultado.

```
def mostrar_Historial(self):
    pantalla_menu.blit(pg.image.load("img\Fondo_Espacio.jpg"), (0, 0)) #variable para el fondo del menú, posicion inicial img->(0, 0)
    MENU_POS_RATON = pg.mouse.get_pos() #detecta pos del mouse

    #dibuja botones
    BTN_10_ULT_PART = Boton(imagen=pg.image.load("img/Boton.png"), pos=(ANCHO_PANTALLA // 2, ALTO_PANTALLA // 2+125),txt_entr="VER 10 PARTIDAS",
    BTN_VOLVER = Boton(imagen=pg.image.load("img/Boton.png"), pos=(ANCHO_PANTALLA // 2, ALTO_PANTALLA // 2+250),txt_entr="VOLVER", font=self.get

    color_activo = pg.color('white') #rect seleccionado, escribiendo
    color_pasivo = pg.color('navy') #rect NO seleccionado, no puede escribir

    #texto y rect de seleccion de tag para ver sus resultados
    self.escribir_texto("TAG:", 80, ANCHO_PANTALLA // 5 -100, ALTO_PANTALLA // 4 -160)
    #reutilizo las variables de la clase relacionadas con el TAG1 -> self.tag_j1, self.color_rectangulo1, self.escribir_tag1, self.texto_err_tag
    self.escribir_texto(self.tag_j1,80, ANCHO_PANTALLA // 5+50, ALTO_PANTALLA // 4 -160)
    rect_tag = self.dibujar_rect(ANCHO_PANTALLA // 5+40,ALTO_PANTALLA // 4 -170,170,80,self.color_rectangulo1,2)
```

Con el uso del objeto creado en el constructor se guarda el historial de partidas que tenga el “tag_j1” (si existe o ha introducido alguno). Para mostrar las partidas ganadas, perdidas, empatadas y el texto de error se comprueba si se ha escrito algún TAG, si no ha recibido ningún resultado el método buscar_hist_jugador(self.tag_j1), y de lo contrario a todo lo anterior, se guarda el resultado en variables individuales para mostrarlas después en pantalla. Si no se introduce ningún TAG o no lo encuentra en el fichero “Historial_Jugadores.txt”, se muestra un mensaje de error en pantalla para informar al

usuario.

```
371     texto_res_tag = self.leer_fich.buscar_hist_jugador(self.tag_j1)
372     texto_ganadas = ""
373     texto_perdidas = ""
374     texto_empatadas = ""
375     if self.tag_j1 == "":
376         texto_ganadas = ""
377         texto_perdidas = ""
378         texto_empatadas = ""
379         texto_res_tag = "Escriba el TAG del jugador que quiera saber sus resultados."
380     elif texto_res_tag == "":
381         texto_ganadas = ""
382         texto_perdidas = ""
383         texto_empatadas = ""
384         texto_res_tag = "TAG no encontrado"
385     else:
386         array_texto_res_tag = texto_res_tag.split(":")
387         texto_res_tag = "ESTADÍSTICAS:"
388         texto_ganadas = "GANADAS: " + array_texto_res_tag[1]
389         texto_perdidas = "PERDIDAS: " + array_texto_res_tag[2]
390         texto_empatadas = "EMPATADAS: " + array_texto_res_tag[3]
```

A continuación se muestra en pantalla los resultados del jugador o por el contrario el mensaje de error. También se muestra el historial de las diez últimas partidas en el caso de que el usuario haya pulsado el botón. Se imprimen los botones que tiene el menú con su *hover*.

```

392     self.escribir_texto(texto_res_tag, 30, ANCHO_PANTALLA // 5 -100, ALTO_PANTALLA // 4)
393     self.escribir_texto(texto_ganadas, 30, ANCHO_PANTALLA // 5 -100, ALTO_PANTALLA // 4+100)
394     self.escribir_texto(texto_perdidas, 30, ANCHO_PANTALLA // 5 -100, ALTO_PANTALLA // 4+200)
395     self.escribir_texto(texto_empatadas, 30, ANCHO_PANTALLA // 5 -100, ALTO_PANTALLA // 4+300)
396     #bucle para mostrar por pantalla cada partida en el historial
397     pos_y = -100
398     for txt in self.texto_hist_10_partidas:
399         self.escribir_texto(txt, 30, ANCHO_PANTALLA*3 // 5 +50, ALTO_PANTALLA // 4+pos_y)
400         pos_y += 70
401
402     for boton in [BTN_10_ULT_PART, BTN_VOLVER]: #hover para los botones y cambiarlos de color
403         boton.cambiarColor(MENU_POS_RATON)
404         boton.update(pantalla_menu)

```

Se vuelven a comprobar los eventos que suceden en el menú para saber si se está escribiendo en el rectángulo de *input* para el *TAG*, o si se pulsó alguno de los botones (para ver el historial o volver al menú “Principal”).

```

406     eventos = pg.event.get() #comprueba eventos en el menu
407     for event in eventos:
408         if event.type==pg.QUIT:
409             pg.quit()
410             sys.exit()
411
412         if event.type==pg.KEYDOWN: #detecta si se pulsan teclas
413             if self.escribir_tag1: #si tiene seleccionado el cuadro para escribir el tag
414                 if event.key == pg.K_BACKSPACE: #si es la tecla de borrar
415                     self.tag_j1 = self.tag_j1[0:-1] #borra el ultimo caracter
416                 elif self.tag_j1.__len__() < 3: #si no ha superado la longitud máxima
417                     self.tag_j1 += event.unicode
418                     self.tag_j1 = self.tag_j1.upper() #lo que escribe lo para a mayúsculas
419
420         if event.type==pg.MOUSEBUTTONDOWN:
421             if rect_tag.collidepoint(event.pos): #detecta si la caja del tag se ha seleccionado
422                 self.escribir_tag1 = True #habilita la escritura
423                 self.color_rectangulo1 = color_activo #muestra que está seleccionado el rectángulo para el tag1 poniéndolo en blanco
424             else: #si no se ha clickado en el rect (por ejemplo en un botón o fuera)
425                 self.escribir_tag1 = False #deshabilita la escritura
426                 self.color_rectangulo1 = color_pasivo #y lo enseña desactivado poniéndolo en azul
427             if BTN_10_ULT_PART.verInputs(MENU_POS_RATON):
428                 self.texto_hist_10_partidas = self.leer_fich.buscar_10_ult_partidas()
429                 if self.texto_hist_10_partidas[0] == "": #si la primera linea está vacía, es que no hay historial que enseñar
430                     self.texto_hist_10_partidas[0] = "NO SE HA ENCONTRADO NINGÚN HISTORIAL."
431             #de lo contrario, es que hay historial
432
433             if BTN_VOLVER.verInputs(MENU_POS_RATON):
434                 self.texto_hist_10_partidas = ""
435                 self.menuPrincipal_disp = True
436                 self.menuHistorial_disp = False

```

Y se muestra el ícono de ayuda antes de actualizar la pantalla.

```

438     self.cuadrado_ayuda(MENU_POS_RATON, eventos) #ICONO AYUDA
439
440     pg.display.update()

```

def escribir_texto(self, texto, tam_letra, x , y):

Para poner texto en pantalla, se ha creado un método que recibe el *string* que se quiera mostrar junto con el tamaño de la fuente y la posición.

```
442     def escribir_texto(self, texto, tam_letra, x , y):  
443         texto_pan = self.get_font(tam_letra).render(texto, True, (255,255,255))  
444         pantalla_menu.blit(texto_pan,(x , y))  
def dibujar_rect(self, x , y, x_tam, y_tam, color, borde):
```

En este método se para por parámetro la posición x e y, el ancho y alto del rectángulo, y su color y tamaño de borde. ‘PyGame’ permite crear rectángulos con el método `pygame.Rect(pos_x, pos_y, ancho, alto)` para luego dibujarlo en pantalla y devolverlo para que se detecten eventos (*clicks*).

```
446     def dibujar_rect(self, x , y, x_tam, y_tam, color, borde): #método para dibujar rectángulos  
447         rect = pg.Rect(x , y, x_tam, y_tam)  
448         pg.draw.rect(pantalla_menu, color, rect, borde)  
449         return rect #devuelve rect para poder comprobar las colisiones con el cuadrado
```

def cuadrado_ayuda(self, MENU_POS_RATON, eventos):

Para el ícono que muestra los controles de cada jugador, se crea un botón que muestra la imagen de los controles mientras que el usuario mantenga el ícono pulsado.

```
451     def cuadrado_ayuda(self, MENU_POS_RATON, eventos): #ICONO AYUDA CONTROLES -> (?)  
452         BTN_AYUDA = Boton(pg.image.load("img/Icono_Ayuda.png"), pos=(100, ALTO_PANTALLA-100),txt_entr="", font=self.get_font(50),base_color="White",  
453         BTN_AYUDA.update(pantalla_menu)  
454  
455         btn_pulsado = False #inicialmente no está pulsado de forma predeterminada  
456         for event in eventos:  
457             if event.type==pg.MOUSEBUTTONDOWN:  
458                 if BTN_AYUDA.verInputs(MENU_POS_RATON):  
459                     btn_pulsado = True  
460  
461         while btn_pulsado: #mientras el botón del ratón siga pulsado  
462             imagen_ayuda = pg.image.load("img/Controles.png") #dibuja la imagen con los controles  
463             imagen_ayuda = pg.transform.scale(imagen_ayuda, (500, 281))  
464             pantalla_menu.blit(imagen_ayuda, (ANCHO_PANTALLA//3, ALTO_PANTALLA//3))  
465  
466             eventos = pg.event.get() #comprueba si surgen eventos para salir del bucle  
467             for event in eventos:  
468                 if event.type==pg.MOUSEBUTTONUP: #cuando suelta el botón del ratón  
469                     if BTN_AYUDA.verInputs(MENU_POS_RATON):  
470                         btn_pulsado = False #sale del bucle que muestra la imagen  
471             pg.display.update()
```

def get_font(self, size):

Con esta función se devuelve una variable tipo “Font” de ‘PyGame’ para los botones o escritura de texto.

```
473     def get_font(self, size): #recibe fuentes  
474         return pg.font.Font("fuentes/Fuentes.ttf",size)
```

→ botón.py

Para crear los botones de la interfaz se ha creado una clase para dividir el código y permitir que sea usada por los diferentes estados del videojuego.

Métodos:

def __init__(self, imagen, pos, txt_entr, font, base_color, color_flot):

Cuando se crea un objeto tipo “Botón”, se le deben pasar por parámetro diferentes variables para su creación: una variable de imagen (pg.image.load("imagen")), su posición (en una lista de dos elementos), una variable para el *font* (pg.font.Font("fuente",tamaño)), el color de la fuente inicial, y el color para el *hover*. En el caso de que no se le pase una imagen, el texto se establece como el dibujo para la detección de eventos.

```
2     def __init__(self, imagen, pos, txt_entr, font, base_color, color_flot): #constructor
3         self.imagen = imagen
4         self.x_pos = pos[0]
5         self.y_pos = pos[1]
6         self.font = font
7         self.color_base, self.color_hover = base_color, color_flot #color y color de hover
8         self.txt_entr = txt_entr
9         self.text = self.font.render(self.txt_entr, True, self.color_base) #font
10        if self.imagen is None:
11            self.imagen = self.text #si no tiene img, se le pone el texto como dibujo
12            self.rect = self.imagen.get_rect(center=(self.x_pos,self.y_pos))
13            self.text_rect = self.text.get_rect(center=(self.x_pos, self.y_pos))
```

def update(self, pantalla):

Para poner el botón en pantalla se utiliza este método, debido a que hay diferencia si se le ha pasado una imagen en el constructor o no, al momento de dibujarlo en pantalla.

```
15    def update(self, pantalla): #actualiza el estado del boton
16        if self.imagen is not None:
17            pantalla.blit(self.imagen, self.rect)
18            pantalla.blit(self.text, self.text_rect)
```

def verInputs(self, posicion):

Por otro lado, se detecta si se tiene el ratón encima del botón con el uso de este método para saber si se ha hecho *click* en él.

```
20    def verInputs(self, posicion): #metodo para saber si se pulsa el boton
21        if posicion[0] in range(self.rect.left, self.rect.right) and posicion[1] in range(self.rect.top, self.rect.bottom):
22            return True
23        return False
```

def cambiarColor(self, posicion):

De manera parecida a la detección de *input*, se detecta si el ratón se encuentra encima del ratón para cambiar el color del texto.

```
29     def cambiarColor(self, posicion): #cambia el color para el hover
30         if posicion[0] in range(self.rect.left, self.rect.right) and posicion[1] in range(self.rect.top, self.rect.bottom):
31             self.text = self.font.render(self.txt_entr, True, self.color_hover)
32         else:
33             self.text = self.font.render(self.txt_entr, True, self.color_base)
```

def centrar(self, pantalla):

Para mostrar el botón de manera centrada en pantalla se creó este método por si fuera necesario usarlo.

```
25     def centrar(self, pantalla): #centra el boton en la pantalla
26         ancho, alto = pantalla.get_size()
27         self.image_rect = self.imagen.get_rect(center=(ancho // 2, alto // 2))
```

→ **mundo.py**

Cuando se cambia al estado “Juego”, se crea un objeto tipo “Mundo” para poder utilizar los métodos que permiten mostrar la interfaz y funcionamiento del *gameplay*.

-Imports:

Dentro de este módulo se importa el paquete de ‘PyGame’, la clase “Escribir_Fichero”, “Nave” y “Boton”, junto con las propiedades necesarias para la pantalla y los *sprites* de las naves. También se importa la clase “Menu” para mostrar el “ícono de ayuda” en el menú “Pausa”.

```
1  import pygame as pg
2  from escribir_fichero import Escribir_Fichero
3  from nave import Nave
4  from propiedades import ALTO_PANTALLA, ANCHO_PANTALLA, ANCHO_NAVE
5  from boton import Boton
6  from menu import Menu
```

-Métodos:

```
def __init__(self, pantalla, tag_j1, tag_j2, volumen_sfx):
```

Al crear el objeto, se le pasa por parámetro la pantalla en la que se ponen los elementos de la interfaz, y las variables modificadas por otros menús (*TAGs* y volumen efectos). Dentro del constructor se crean diferentes variables necesarias para cambiar de estado, crear *sprites* (naves), guardar los *TAGs*, guardar el último tiempo de colisión, poner música y efectos *SFX*, poner las imágenes que muestran las vidas (corazones) y al final generar el mundo (*sprites* de las naves).

```
10     def __init__(self, pantalla, tag_j1, tag_j2, volumen_sfx): #constructor recibe pantalla del juego, y los tags escogidos en el menu
11         self.ganador = "" #var para definir quien gana
12         self.coli_tiempo = pg.time.get_ticks() #var para saber cuando fue la ultima vez que colisiona algun sprite
13         self.pantalla = pantalla
14         self.nave1 = pg.sprite.GroupSingle() #se crea un grupo de sprite unico para cada nave
15         self.nave2 = pg.sprite.GroupSingle()
16         self.fin_partida = False
17         self.volver_menu = False
18         self.tag_j1 = tag_j1 #tag j1
19         self.tag_j2 = tag_j2 #tag j2
20
21         self.musica_fondo = pg.mixer.music #variable para general la musica
22         self.musica_fondo.load("music/Musica_Pelea.mp3") #carga .mp3
23         self.musica_fondo.play(-1)
24         self.parar_musica = True
25
26         #var para efectos de sonido
27         self.volumen_sfx = volumen_sfx
28         self.sonido_impact = pg.mixer.Sound("music/Sonido_Impacto.mp3") #carga .mp3 para el sonido de impacto
29         self.sonido_impact.set_volume(self.volumen_sfx) #establece el volumen elegido en opciones
30
31         #img vidas
32         self.imagen_v1 = pg.image.load('img/Corazon.png').convert() #carga la imagen
33         self.imagen_v1 = pg.transform.scale(self.imagen_v1, (50, 50)) #la escala
34         self.pantalla.blit(self.imagen_v1, (ANCHO_PANTALLA-120, ALTO_PANTALLA-80))
35         self.mask = pg.mask.from_surface(self.imagen_v1)
36
37         self.imagen_v2 = pg.image.load('img/Corazon.png').convert() #carga la imagen
38         self.imagen_v2 = pg.transform.scale(self.imagen_v2, (50, 50)) #la escala
39         self.pantalla.blit(self.imagen_v2, (0, ALTO_PANTALLA-80))
40
41         self.pausa = False #var para pausar el juego
42
43         self.generando_mundo() #se genera el mundo al final del constructor
```

```
def generando_mundo(self):
```

En este método se crean dos objetos tipo “Nave” (en la derecha “nave2” y en la izquierda “nave1”) y se añaden a su grupo de *sprite* individual que se creó en el constructor.

```
45     def generando_mundo(self): #crea el mundo
46         pos_x1 = 0
47         pos_y1 = ALTO_PANTALLA // 2
48         pos_nave1 = (pos_x1, pos_y1) #pone la nave1 en el medio izquierdo
49         self.nave1.add(Nave(pos_nave1, "nave1", self.volumen_sfx)) #lo añade al grupo de sprite unico
50         pos_x2 = ANCHO_PANTALLA - ANCHO_NAVE
51         pos_y2 = ALTO_PANTALLA // 2
52         pos_nave2 = (pos_x2, pos_y2) #pone la nave2 en el medio derecha
53         self.nave2.add(Nave(pos_nave2, "nave2", self.volumen_sfx)) #y lo añade a su grupo de sprite unico
```

def update(self):

Dentro del método `update()`, se decide el estado en el que se encuentra el juego (Pantalla “Juego”, “Game Over” o “Pausa”). Para parar la música una única vez cuando se pausa o se acaba la partida, se usa una variable auxiliar (`self.parar_musica`) para evitar que el juego se quede “colgado” al intentar parar la música varias veces. Cuando entra en estado “Game Over”, aparte de parar la música, se crea una variable tipo “Escribir_Fichero” para guardar los resultados de cada jugador y la partida que se ha terminado.

```
55 def update(self):
56     if not self.fin_partida and not self.pausa:
57         self.partida()
58
59     while not self.fin_partida and self.pausa: #tanto el menu de 'pausa' como el menu de 'game over' los meto en un bucle para evitar retardo en la
60         if self.parar_musica: #para la música una vez al entrar en "pausa"
61             self.musica_fondo.pause() #se para la musica de pelea
62             self.parar_musica = False
63         self.menu_pausa()
64
65     while self.fin_partida:
66         if self.parar_musica: #para la música y guarda los resultados una única vez al entrar en "gameover"
67             self.musica_fondo.pause() #se para la musica de pelea
68             self.parar_musica = False
69             escribir_resultado = Escribir_Fichero(self.tag_j1,self.tag_j2,self.ganador)
70             escribir_resultado.escribir_historial_jugador(self.tag_j1) #escribe/actualiza las partidas ganadas:perdidas:empatadas de ambos jugadores
71             escribir_resultado.escribir_historial_jugador(self.tag_j2)
72             escribir_resultado.escribir_historial_partida() #y guarda el resultado de la partida
73
74         self.game_over()
```

def nave_mov(self):

Mientras que no se haya acabado la partida, el juego detecta las teclas que se han pulsado en el teclado para mover las naves de cada jugador. Si se pulsan las teclas ‘W’, ‘A’, ‘S’ o ‘D’ se mueve la nave de la izquierda (“nave1”), y si se pulsan las flechas del teclado se mueve la nave de la derecha (“nave2”), siempre y cuando no se salgan de la pantalla por ninguno de sus laterales.

```

76     def nave_mov(self):
77         if not self.fin_partida: #mientras que no acabe la partida
78             teclas = pg.key.get_pressed() #detecta las teclas pulsadas y se mueve acorde
79
80             #movimientos nave1 -> W A S D
81             if teclas[pg.K_a]:
82                 if self.nave1.sprite.rect.left > 0:
83                     self.nave1.sprite.izq()
84             if teclas[pg.K_d]:
85                 if self.nave1.sprite.rect.right < ANCHO_PANTALLA:
86                     self.nave1.sprite.der()
87             if teclas[pg.K_w]:
88                 if self.nave1.sprite.rect.top > 0:
89                     self.nave1.sprite.arr()
90             if teclas[pg.K_s]:
91                 if self.nave1.sprite.rect.bottom < ALTO_PANTALLA-127:
92                     self.nave1.sprite.abj()
93
94             #movimientos nave1 -> ↑ ← ↓ →
95             if teclas[pg.K_LEFT]:
96                 if self.nave2.sprite.rect.right > 50:
97                     self.nave2.sprite.izq()
98             if teclas[pg.K_RIGHT]:
99                 if self.nave2.sprite.rect.right < ANCHO_PANTALLA:
100                    self.nave2.sprite.der()
101             if teclas[pg.K_UP]:
102                 if self.nave2.sprite.rect.top > 0:
103                     self.nave2.sprite.arr()
104             if teclas[pg.K_DOWN]:
105                 if self.nave2.sprite.rect.bottom < ALTO_PANTALLA-127:
106                     self.nave2.sprite.abj()

```

def nave_disp(self):

Al igual que con el movimiento de las naves, permite disparar si no se ha acabado la partida. A su vez, se detecta si se ha pulsado los botones de disparo de cada nave y el tiempo en el que se han pulsado. Se permite disparar en intervalos de tiempo para dejar espacio entre balas, de manera que las naves puedan esquivar ráfagas de balas moviéndose entre bala y bala.

```

108     def nave_disp(self):
109         if not self.fin_partida: #mientras que no acabe la partida
110             teclas = pg.key.get_pressed() #detecta las teclas que se pulsan
111             ticks = pg.time.get_ticks()
112             segs = int(ticks/20) #cantidad de balas por disparo
113
114             if segs % 12 == 0: #intervalo entre disparos
115                 if teclas[pg.K_LCTRL]:
116                     self.nave1.sprite.disparar()
117                 if teclas[pg.K_RCTRL]: #CAMBIAR TECLA DE DISPARO A K_RCTRL PARA WINDOWS, K_RSHIFT PARA LINUX
118                     self.nave2.sprite.disparar()

```

def nave_disp(self):

Este método muestra en pantalla los componentes visuales del estado “Juego”. Primero, se ponen en pantalla las imágenes de corazones en la parte inferior, junto con los contadores de vida de cada jugador.

```
120     def partida(self):
121         #dibujo las imagenes de corazón
122         self.pantalla.blit(self.imagen_v1, (ANCHO_PANTALLA-120, ALTO_PANTALLA-80))
123         self.pantalla.blit(self.imagen_v2, (0, ALTO_PANTALLA-80))
124
125         #y el num de vidas que queda a cada jugador
126         numv1 = "X " + str(self.nave1.sprite.vida)
127         numv2 = "X " + str(self.nave2.sprite.vida)
128         vidas_p1 = pg.font.Font("fuentes/Fuentes.ttf", 40).render(numv1, True, "#FFFFFF")
129         vidas_p1_rect = vidas_p1.get_rect(centerx=ANCHO_PANTALLA-40,centery=ALTO_PANTALLA-50)
130         self.pantalla.blit(vidas_p1, vidas_p1_rect)
131         vidas_p2 = pg.font.Font("fuentes/Fuentes.ttf", 40).render(numv2, True, "#FFFFFF")
132         vidas_p2_rect = vidas_p2.get_rect(centerx=80,centery=ALTO_PANTALLA-50)
133         self.pantalla.blit(vidas_p2, vidas_p2_rect)
```

Segundo, se detecta si las naves se encuentran en un estado de inmunidad (después de producirse una colisión) para mostrar en mitad de la pantalla un contador regresivo de 3 segundos. A continuación, se actualiza el estado de los *sprites* de naves y sus balas, para ponerlos en la pantalla junto al resto de elementos.

```
135     #dibujo el contador de inmunidad
136     if not self.no_inmunidad(): #si son inmunes dibuja el contador
137         t_inmune = int(((pg.time.get_ticks() - self.colis_tiempos)/1000) -4) * (-1)
138         contador_inmunidad = pg.font.Font("fuentes/Fuentes.ttf", 40).render(str(t_inmune), True, "#FFFFFF")
139         contador_inmunidad_rect = contador_inmunidad.get_rect(centerx=ANCHO_PANTALLA//2,centery=ALTO_PANTALLA//2)
140         self.pantalla.blit(contador_inmunidad, contador_inmunidad_rect)
141
142         self.nave1.update() #actualiza el estado del sprite (posición, movimiento...)
143         self.nave1.draw(self.pantalla) #lo pinta en la pantalla
144         self.nave2.update()
145         self.nave2.draw(self.pantalla)
146
147         self.nave1.sprite.grupo_balas.update() #lo mismo con el grupo de sprites de balas de cada bando
148         self.nave1.sprite.grupo_balas.draw(self.pantalla)
149
150         self.nave2.sprite.grupo_balas.update()
151         self.nave2.sprite.grupo_balas.draw(self.pantalla)
```

Al final de la función, se actualizan todos los componentes que se encuentran en pantalla, se detectan las colisiones que se producen, al igual que se comprueba si se pulsa la tecla ‘Esc’ para entrar en estado de “Pausa”.

```
153     pg.display.flip()
154     self.detectar_colision() #cada vez que actualiza la pantalla detecta colisiones
155
156     teclas = pg.key.get_pressed()
157     if teclas[pg.K_ESCAPE]: #si detecta que se ha pulsado escape
158         self.pausa = True #pausa el juego
```

def game_over(self):

Al cambiar al estado “Game Over”, se cubre la pantalla de negro para tapar los elementos del estado anterior. Tras poner el fondo, se dibujan los diferentes elementos (texto y botones) que componen la interfaz.

```
160 def game_over(self):
161     POS_RATON = pg.mouse.get_pos()
162     self.pantalla.fill("black")
163
164     GAME_OVER = pg.font.Font("fuentes/Fuentes.ttf", 70).render("GAME OVER", True, "#8B0000")
165
166     if self.ganador != "EMPATE!":
167         texto_victoria = self.ganador + " WINS!"
168     else:
169         texto_victoria = self.ganador
170     player = pg.font.Font("fuentes/Fuentes.ttf", 40).render(texto_victoria, True, "#FFFFFF")
171     player_rect = player.get_rect(centerx=ANCHO_PANTALLA // 2, centery=100)
172
173     GAME_OVER_RECT = GAME_OVER.get_rect(center=(ANCHO_PANTALLA // 2, 200))
174
175     self.pantalla.blit(GAME_OVER, GAME_OVER_RECT)
176     self.pantalla.blit(player, player_rect)
177
178     BTN_REPLAY = Boton(imagen=pg.image.load("img/Boton.png"), pos=(ANCHO_PANTALLA // 2, ALTO_PANTALLA // 2), txt_entr="REINICIAR", font=self.get_font())
179     BTN_VOLVER = Boton(imagen=pg.image.load("img/Boton.png"), pos=(ANCHO_PANTALLA // 2, ALTO_PANTALLA // 2+250), txt_entr="VOLVER MENU", font=self.get_font())
180
181     for boton in [BTN_REPLAY, BTN_VOLVER]:
182         boton.cambiarColor(POS_RATON)
183         boton.update(self.pantalla)
```

Como en el resto de menús, se capturan los eventos que ocurren para saber si el usuario ha pulsado algún botón. Si presiona el botón “Reiniciar”, se vuelve a generar el mundo (para reiniciar los *sprites*), se reinicia la música y se cambian las variables de estado para volver al juego. De lo contrario, si presiona “Volver”, se para la música por completo y se vuelve al estado de menú “Principal”.

```
185     eventos = pg.event.get()
186     for event in eventos:
187         if event.type == pg.MOUSEBUTTONDOWN:
188             if BTN_REPLAY.verInputs(POS_RATON):
189                 self.generando_mundo()
190                 self.musica_fondo.set_pos(0.0) #si se reinicia la partida, se vuelve a poner la musica desde el segundo 0.0
191                 self.musica_fondo.unpause()
192                 self.fin_partida = False
193                 self.volver_menu = False
194                 self.parar_musica = True
195             if BTN_VOLVER.verInputs(POS_RATON):
196                 if self.musica_fondo.get_busy():
197                     self.musica_fondo.stop() #si se elige salir al menú, la música se para por completo
198                     self.fin_partida = False #sale del menu de game over
199                     self.volver_menu = True
200     pg.display.update()
```

def menu_pausa(self):

Cuando el usuario presiona ‘Esc’ en su teclado, el juego se pausa por completo y se muestra un menú para que el usuario elija: seguir con la partida (“Continuar”), reiniciar la partida por completo (“Reiniciar”) o volver al menú “Principal” (“Volver menú”).

```

202     def menu_pausa(self):
203         POS_RATON = pg.mouse.get_pos()
204         self.pantalla.fill("black")
205
206         TXT_PAUSA = pg.font.Font("fuentes/Fuentes.ttf", 70).render("JUEGO EN PAUSA", True, "#8B0000")
207         self.pantalla.blit(TXT_PAUSA,TXT_PAUSA.get_rect(center=(ANCHO_PANTALLA // 2, 100)))
208
209         BTN_RESUME = Boton(imagen=pg.image.load("img/Boton.png"), pos=(ANCHO_PANTALLA // 2, ALTO_PANTALLA // 2 -150),txt_entr="CONTINUAR",font=self.get_font())
210         BTN_REPLAY = Boton(imagen=pg.image.load("img/Boton.png"), pos=(ANCHO_PANTALLA // 2, ALTO_PANTALLA // 2),txt_entr="REINICIAR",font=self.get_font())
211         BTN_VOLVER = Boton(imagen=pg.image.load("img/Boton.png"), pos=(ANCHO_PANTALLA // 2, ALTO_PANTALLA // 2+150),txt_entr="VOLVER MENU", font=self.get_font())
212
213         for boton in [BTN_RESUME, BTN_REPLAY, BTN_VOLVER]:
214             boton.cambiarColor(POS_RATON)
215             boton.update(self.pantalla)

```

Al igual que en el menú anterior, se detectan los botones que pulsa con el ratón el usuario y se cambia de estado o se para la música en función de cuál presione. También se muestra el ícono de ayuda y se actualiza la pantalla.

```

217     eventos = pg.event.get()
218     for event in eventos:
219         if event.type == pg.MOUSEBUTTONDOWN:
220             if BTN_RESUME.verInputs(POS_RATON):
221                 self.musica_fondo.unpause()
222                 self.parar_musica = True
223                 self.pausa = False
224             if BTN_REPLAY.verInputs(POS_RATON): #si se reinicia la partida
225                 self.generando_mundo() #se vuelve a generar el mundo
226                 self.musica_fondo.set_pos(0.0)
227                 self.musica_fondo.unpause()
228                 self.parar_musica = True
229                 self.pausa = False
230             if BTN_VOLVER.verInputs(POS_RATON):
231                 self.musica_fondo.stop() #si se elige salir al menú, la música se para por completo
232                 self.pausa = False #sale del menu de pausa
233                 self.volver_menu = True
234
235             self.cuadrado_ayuda(POS_RATON, eventos)
236             pg.display.update()

```

def get_font(self, size):

Como en la clase “Menu”, se crea un método para recibir la fuente que se quiera aplicar al texto.

```

239     def get_font(self, size): #para obtener el tipo de fuente
240         return pg.font.Font("fuentes/Fuentes.ttf",size)

```

def detectar_colision(self):

Las naves y las balas del juego se han creado como extensión de la clase ‘`pg.sprite.Sprite`’ de ‘`PyGame`’. Por lo tanto, se utiliza un método del mismo paquete que permite detectar colisiones entre *sprites*. Se crean variables que comprueban colisiones entre grupos de *sprites* para detectar: si las balas de la “nave1” colisionan con la “nave2”, si las balas de la “nave2” colisionan con la “nave1” o si la “nave1” colisiona con la “nave2”.

```

242     def detectar_colision(self):
243         nave1_coli_nave2 = pg.sprite.groupcollide(self.nave2, self.nave1.sprite.grupo_balas, False, False) #detecta colisiones
244         nave2_coli_nave1 = pg.sprite.groupcollide(self.nave1, self.nave2.sprite.grupo_balas, False, False) #y viceversa
245         coli_naves = pg.sprite.groupcollide(self.nave1, self.nave2, False, False) #detecta si ambas naves se chocan

```

Antes de comprobar si se ha producido una colisión, se comprueba que no haya un estado de inmunidad (si acaba de empezar la partida o se han golpeado hace menos de 3 segundos). Posteriormente se observa si ha habido alguna de las colisiones creadas anteriormente (Ej: colision = True, cuando sí han chocado), y se restan las vidas que correspondan. Cuando hay algún tipo de colisión, también se actualiza el tiempo que ha habido desde la última (`self.coli_tiempo`) y se reproduce el sonido de impacto. Por último, se comprueba que ninguna de las vidas haya llegado a cero y de ser así, se cambia de estado y se actualiza el ganador (o si empatan).

```

247 if self.no_inmunidad(): #si no hay inmunidad
248     if colis_naves:
249         self.nave1.sprite.restar_vida()
250         self.nave2.sprite.restar_vida()
251         self.sonido_impact.play() #reproduce el efecto de sonido de impacto
252         self.coli_tiempo = pg.time.get_ticks() #restablece el tiempo de la ultima colision al actual
253         if self.nave2.sprite.vida < 1 and self.nave1.sprite.vida < 1: #si no quedan vidas a alguno de los dos, se acaba la partida
254             self.fin_partida = True
255             self.ganador = "EMPATE!"
256         elif self.nave1.sprite.vida < 1:
257             self.fin_partida = True
258             self.ganador = self.tag_j1
259         elif self.nave2.sprite.vida < 1:
260             self.fin_partida = True
261             self.ganador = self.tag_j2
262     #colision nave - bala
263     elif nave1_coli_nave2: #si se detecta colision
264         self.sonido_impact.play() #reproduce el efecto de sonido de impacto
265         self.coli_tiempo = pg.time.get_ticks() #restablece el tiempo de la ultima colision al actual
266         self.nave1.sprite.restar_vida() #le quita 1 vida
267         if self.nave1.sprite.vida < 1: #si no quedan vidas, se acaba la partida
268             self.fin_partida = True
269             self.ganador = self.tag_j1
270     elif nave2_coli_nave1: #idem pero balas nave2 con la propia nave2
271         self.sonido_impact.play() #reproduce el efecto de sonido de impacto
272         self.coli_tiempo = pg.time.get_ticks()
273         self.nave2.sprite.restar_vida()
274         if self.nave2.sprite.vida < 1:
275             self.fin_partida = True
276             self.ganador = self.tag_j2

```

def no_inmunidad(self):

Este método devuelve “True” y modifica los estados de inmunidad de las naves a “False” si han transcurrido más de tres segundos desde el tiempo actual y el tiempo del último impacto. En caso contrario, vuelve las naves inmunes a las colisiones.

```

278     def no_inmunidad(self):
279         if pg.time.get_ticks() - self.coli_tiempo > 3000:
280             self.nave1.sprite.inmunidad = False
281             self.nave2.sprite.inmunidad = False
282             return True
283         else: #si el tiempo actual - tiempo ultima colision < 3000ms es inmune
284             self.nave1.sprite.inmunidad = True
285             self.nave2.sprite.inmunidad = True
286             return False

```

→ nave.py

Como ya se ha mencionado antes, las naves heredan de la clase “Sprite” propia del módulo de ‘PyGame’ para hacer uso de métodos y funcionalidades útiles para desarrollar el videojuego.

```
5   class Nave(pg.sprite.Sprite):
```

-Imports:

Para las naves, se importa ‘PyGame’, las propiedades establecidas para el tamaño y velocidad de las naves y sus balas, y la propia clase “Bala” para crear más adelante un grupo de *sprites* del mismo.

```
1  import pygame as pg
2  from propiedades import ANCHO_NAVE, ALTO_NAVE, VEL_NAVE, ANCHO_BALA, ALTO_BALA
3  from bala import Bala
```

-Métodos:

```
def __init__(self, pos, bando, volumen_sfx):
```

Cuando se crea un objeto de este tipo, se inicializan primero las variables de la clase de la que hereda (“Sprite”). Después se le asigna la posición inicial que recibe por parámetro, una variable para el efecto de sonido con el volumen que se recibe, una variable para guardar el “bando” al que pertenece esta nave, establecer el fondo del *sprite* a negro e inicializar la variable para la inmunidad. Posteriormente, se comprueba el bando al que pertenece la nave para asignarle la imagen de la carpeta “img” y poder crear una variable “mask” para que se puedan detectar las colisiones. También se guarda la velocidad de la nave, se crea el grupo de balas y se inicializa el número de vidas a 3.

```
6  def __init__(self, pos, bando, volumen_sfx): #constructor
7      super().__init__()
8      self.x = pos[0]
9      self.y = pos[1]
10     self.disparo_sonido = pg.mixer.Sound("music\Sonido_Disparo.mp3") #var para el sonido de bala
11     self.disparo_sonido.set_volume(volumen_sfx)
12     self.image = pg.Surface([ALTO_NAVE, ANCHO_NAVE])
13     self.image.fill("black")
14     self.image.set_colorkey("black")
15     self.bando = bando
16     self.inmunidad = False
17
18     if bando == "nave1": #si es nave2 es la nave de la izquierda, por lo que se pinta de forma distinta
19         self.image = pg.image.load("img/Nave_Izquierda.png")
20         self.image = pg.transform.scale(self.image, (ALTO_NAVE, ANCHO_NAVE))
21     >     ''' DIBUJO SVG ...
22     else: #si es nave1 es la nave de la derecha
23         self.image = pg.image.load("img/Nave_Derecha.png")
24         self.image = pg.transform.scale(self.image, (ALTO_NAVE, ANCHO_NAVE))
25     >     ...
26
27     self.rect = self.image.get_rect(topleft = pos)
28     self.mask = pg.mask.from_surface(self.image) #objeto tipo mask para las colisiones
29     self.velocidad_nave = VEL_NAVE
30
31     self.bando = bando #bando necesario para saber si es izquierda o derecha
32     self.grupo_balas = pg.sprite.Group() #se crea el grupo de balas
33     self.vida = 3 #número de vidas
```

def 'mov'(self):

Los siguientes cuatro métodos son los que permiten cambiar la posición de la nave dentro de la clase “Mundo” cuando se pulsa alguna de las teclas asignadas para ello.

```
42     # definimos los movimientos de la nave
43     def izq(self): #izquierda
44         self.rect.x -= self.velocidad_nave
45     def der(self): #derecha
46         self.rect.x += self.velocidad_nave
47     def arr(self): #arriba
48         self.rect.y -= self.velocidad_nave
49     def abj(self): #abajo
50         self.rect.y += self.velocidad_nave
```

def disparar(self):

Al utilizar este método, se añade un objeto tipo “Bala” a su grupo creado en el constructor, al que se le pasa la posición de la que parte la bala, su tamaño y a qué bando pertenece. Tras añadirlo al grupo, se reproduce el efecto de sonido del disparo.

```
52     def disparar(self): #método para disparar
53         pos_nave = (self.rect.centerx, self.rect.centery - ANCHO_BALA // 2) #se pone la bala delante de la nave
54         self.grupo_balas.add(Bala(pos_nave, ANCHO_BALA, ALTO_BALA, self.bando))
55         self.disparo_sonido.play() #se empieza el sonido de disparo
```

def restar_vida(self):

Cuando se produce cualquier tipo de colisión, se le quita una vida a la variable self.vida y se vacía el grupo de *sprite* de las balas (de aquel que no recibe el impacto).

```
57     def restar_vida(self):
58         if self.vida > 0:
59             self.grupo_balas.empty()
60             self.vida = self.vida - 1
```

def restar_vida(self):

Por último, al actualizar el estado de la nave, se le asigna una imagen u otra según su estado de inmunidad (mostrando un escudo azul si es inmune) y se dibuja en pantalla.

```
62     def update(self): #método para actualizarla posición de la nave y redibujarla donde y cómo se encuentre
63         if self.inmunidad: #si es inmune
64             if self.bando == "nave1": #si es nave2 es la nave de la izquierda, por lo que se pinta de forma distinta
65                 self.image = pg.image.load("img/Nave_Izquierda_Inmune.png")
66                 self.image = pg.transform.scale(self.image, (ALTO_NAVE, ANCHO_NAVE))
67             else: #si es nave1 es la nave de la derecha
68                 self.image = pg.image.load("img/Nave_Derecha_Inmune.png")
69                 self.image = pg.transform.scale(self.image, (ALTO_NAVE, ANCHO_NAVE))
70         elif self.inmunidad == False: #si NO es inmune
71             if self.bando == "nave1": #si es nave2 es la nave de la izquierda, por lo que se pinta de forma distinta
72                 self.image = pg.image.load("img/Nave_Izquierda.png")
73                 self.image = pg.transform.scale(self.image, (ALTO_NAVE, ANCHO_NAVE))
74             else: #si es nave1 es la nave de la derecha
75                 self.image = pg.image.load("img/Nave_Derecha.png")
76                 self.image = pg.transform.scale(self.image, (ALTO_NAVE, ANCHO_NAVE))
77
78         self.rect = self.image.get_rect(topleft=(self.rect.x, self.rect.y))
```

→ bala.py

De misma manera que con las naves, las balas también heredan de la clase “Sprite” para poder hacer uso de sus funciones.

-Imports:

Únicamente es necesario el módulo ‘PyGame’ y el ancho de la pantalla para esta clase.

```
1 import pygame as pg  
2 from propiedades import ANCHO_PANTALLA
```

Métodos:

def __init__(self, pos, bando, volumen_sfx):

Para crear un objeto de este tipo, se le pasa por parámetro la posición inicial, el tamaño que se quiera la bala, y el bando al que pertenece. Dentro del propio constructor, se inicializan las variables que se reciben por parámetro, y se crea la imagen y la máscara del *sprite*. Por otro lado, se asigna la velocidad de las balas según el bando al que pertenecen.

```
5     def __init__(self, pos, ancho, alto, bando):  
6         super().__init__()  
7         self.x = pos[0] #toma las posiciones de las balas  
8         self.y = pos[1]  
9  
10        self.image = pg.image.load('img/bala.jpeg') #carga la imagen  
11        self.image = pg.transform.scale(self.image, (ancho, alto)) #la escala  
12        self.rect = self.image.get_rect(topleft = pos) #la coloca  
13        self.mask = pg.mask.from_surface(self.image) #crea un objeto mask para pintar la bala y detectar las colisiones  
14  
15        if bando == "nave1": #si la bala pertenece a nave1 se mueve hacia la der  
16            self.move_speed = 20  
17        elif bando == "nave2": #si la bala pertenece a nave2 se mueve hacia la izq  
18            self.move_speed = (-20)
```

def mover_bala(self):

Esta función mueve la bala en el eje de las ‘x’ con respecto a la velocidad asignada en el constructor.

```
20     def mover_bala(self): #pos de la bala respecto a la vel de la bala  
21         self.rect.x += self.move_speed
```

def update(self):

Al actualizar las balas, se mueven con el método creado anteriormente. Aparte, si la posición respecto al eje de las ‘x’ es fuera de la pantalla “mata” el *sprite* para no sobrecargar la memoria.

```
23     def update(self): #método para pintar la bala en la pantalla en movimiento
24         self.mover_bala()
25         self.rect = self.image.get_rect(topleft=(self.rect.x, self.rect.y))
26
27         if self.rect.x <= 0 or self.rect.x >= ANCHO_PANTALLA: #si salen de la pantalla se "mata" el sprite
28             self.kill()
```

→ **leer_fichero.py**

Para leer de los ficheros que guardan la información que necesita el menú “Historial”, se ha creado una clase con dos métodos para hacerlo más sencillo y permitir en un futuro realizar otras lecturas si fuera necesario. Para abrir un fichero en ‘Python’ se utiliza la función `open("fichero","modo")`.

-Métodos:

def buscar_10_ult_partidas(self):

Primero, se abre el fichero de texto “Historial_Partidas.txt” en modo lectura (“r”). Una vez abierto, se guardan todas las líneas que tenga el fichero en un *array* (o lista en ‘Python’) al igual que la variable en la que se guardarán las 10 últimas líneas.

Posteriormente, se recorren las 10 últimas líneas (o si hay menos de diez las que haya) que se han guardado en la variable *lineas*. Se accede a ellas con el tamaño que tenga el *array* menos la variable contador que se creó fuera.

Para finalizar, se cierra el fichero para liberar memoria, y se devuelve la lista que contiene las 10 últimas líneas del historial de partidas.

```
5     def buscar_10_ult_partidas(self):
6         fichero_jugadores = open("ficheros/Historial_Partidas.txt","r")
7         i = 1 #var contador
8
9         lineas = fichero_jugadores.readlines() #guardo las líneas en una lista de str
10        lineas_10 = ["","","","","","","","","","",""] #establezco la lista de 10 líneas con str vacíos de forma predeterminada
11
12        while i <= 10 and i <= lineas.__len__(): #recorro 10 líneas del fichero o si tiene menos las que haya
13            lineas_10[i-1] = lineas[lineas.__len__()-i] #guarda cada linea en la lista
14            i+=1 #pasa a la siguiente linea
15
16        fichero_jugadores.close() #cierro el fichero
17        return lineas_10 #devuelve todas las líneas
```

def buscar_hist_jugador(self, jugador):

Para buscar los resultados de un jugador, se abre el fichero "Historial_Jugadores.txt" en modo lectura. Se guardan de nuevo todas las líneas que contenga el fichero el una variable, y se recorren para buscar el TAG del jugador que se ha pasado por parámetro. Debido a que las líneas del fichero tienen un formato propio (TAG:NumPartidasGanadas:NumPartidasPerdidas:NumPartidasEmpatadas), se divide cada línea con .split(":") para obtener un array de strings.

```
ficheros >  ≡ Historial_Jugadores.txt
1    WWW:1:2:0:
2    AAA:3:1:0:
3    SSS:0:1:0:
```

Si encuentra el TAG, guarda la línea en una variable, cierra el fichero, y la devuelve con un *return*. De lo contrario, cierra el fichero y devuelve una variable vacía.

```
19  def buscar_hist_jugador(self,jugador):
20      fichero_jugadores = open("ficheros/Historial_Jugadores.txt","r")
21      i = 0 #var contador
22      historial_jugador = ""
23
24      lineas = fichero_jugadores.readlines() #guardo las lineas en una lista de str
25      for l in lineas: #recorro toda la lista de lineas
26          array_linea = l.split(":") #separa la linea en otra lista (jugador:p_ganados:p_perdidos:\n)
27
28          if jugador == array_linea[0]: #si coincide el tag del jugador con el que hay en el fichero
29              historial_jugador = l
30              fichero_jugadores.close()
31              return historial_jugador #devuelve la linea completa [str()]
32
33          i+=1 #pasa a la siguiente linea
34
35      fichero_jugadores.close()
36      historial_jugador = ""
37      return historial_jugador #devuelve una var vacía la cual indica que no se ha encontrado
```

→ **escribir_fichero.py**

Cuando se guarda el resultado de cada partida, se guarda a su vez la fecha y hora a la que termina. También se guardan los resultados de cada jugador o si ya existen en el fichero, se actualizan.

-Imports:

Es necesario importar **datetime** para obtener la fecha y hora actuales.

```
1  from datetime import datetime
```

-Métodos:

```
def __init__(self, tag_j1, tag_j2, ganador):
```

Dentro del constructor, se guardan los *TAGs* de cada jugador y quien gana (o si empatan) con las variables recibidas por parámetro. Aparte, se crea una lista con los días del mes en español para guardar la fecha en el mismo idioma que el resto de la aplicación.

```
4     def __init__(self, tag_j1, tag_j2, ganador): #constructor el ganador, y los tags escogidos en el menu
5         self.tag_j1 = tag_j1
6         self.tag_j2 = tag_j2
7         self.ganador = ganador
8         self.meses = ["ENERO", "FEBRERO", "MARZO", "ABRIL", "MAYO", "JUNIO", "JULIO", "AGOSTO", "SEPTIEMBRE", "OCTUBRE", "NOVIEMBRE", "DICIEMBRE"]
```

```
def escribir_historial_partida(self):
```

Al terminar cada partida, se guarda el resultado en el fichero “Historial_Partidas.txt”. Primero, obtengo la fecha y hora actuales con un método propio de esta misma clase.

A continuación, abro el fichero “Historial_Partidas.txt” en modo *append*, para añadir una nueva línea al final. Según quien haya ganado o si han empatado, se crea un *string* indicando el resultado junto con la fecha y hora obtenidas anteriormente. Si hubiese algún problema al crear la frase, se guarda un mensaje de error.

Por último se escribe en el fichero la cadena de caracteres creada y se cierra el fichero.

```
10    def escribir_historial_partida(self):
11        fecha_hora = self.get_fecha_hora_actual() #recibo la lista con la fecha y hora
12        fichero_partidas = open("ficheros/Historial_Partidas.txt", "a")
13        texto_resultado = ""
14        #escribo el texto que voy a guardar en el fichero de historial con el resultado de la partida y su fecha y hora correspondientes
15        if self.ganador == "EMPATE!":
16            texto_resultado = self.tag_j1 + " & " + self.tag_j2 + " EMPATAN EL " + fecha_hora[0] + " a las " + fecha_hora[1] + "\n"
17        elif self.ganador == self.tag_j1:
18            texto_resultado = self.ganador + " GANA A " + self.tag_j2 + " EL " + fecha_hora[0] + " a las " + fecha_hora[1] + "\n"
19        elif self.ganador == self.tag_j2:
20            texto_resultado = self.ganador + " GANA A " + self.tag_j1 + " EL " + fecha_hora[0] + " a las " + fecha_hora[1] + "\n"
21        else:
22            texto_resultado = "Hubo un ERROR al escribir el resultado de la partida.\n"
23
24        fichero_partidas.write(texto_resultado)
25
26        fichero_partidas.close()
```

```
def escribir_historial_jugador(self, jugador):
```

Para escribir el historial de cada jugador, se vuelve a abrir un fichero (“Historial_Jugadores”) en modo *append*. Se crean las variables necesarias para indicar las partidas que ha ganado, perdido y empatabo. Después, se utiliza una función creada dentro de la clase (que se explicará más adelante) para buscar el *TAG* dentro del archivo, indicando en la primera posición del *array* recibido la línea en la que se encuentra (o -1 si no se ha encontrado), y en la siguiente una lista con todas las líneas del historial. A continuación, se vacía el fichero al abrirlo en modo escritura y cerrarlo.

```

28     def escribir_historial_jugador(self, jugador):
29         fichero_jugadores = open("ficheros/Historial_Jugadores.txt", "a")
30         #var para historial de partidas del jugador
31         p_ganadas = 0
32         p_empatadas = 0
33         p_perdidas = 0
34
35         encontrado_lineas = self.buscar_jugador(jugador) #busca el tag del jugador en el fichero Historial_Jugadores.txt
36         linea_encontrado = encontrado_lineas[0] #en que linea se encuentra el jugador buscado
37         lista_lineas = encontrado_lineas[1]
38
39         open('ficheros/Historial_Jugadores.txt', 'w').close() #vacia el contenido del fichero

```

A continuación, se comprueba si se ha encontrado al jugador dentro del fichero, para poder modificar su historial según lo que haya en la variable `self.ganador`. Para ello, se modifica la línea en la que se encuentra, sustituyéndola con el nuevo registro. Si no se ha encontrado (`linea_encontrado == -1`), se crea un nuevo registro con el *TAG* del jugador y sus partidas ganadas, perdidas y empatadas, y lo añade al final de la lista que contiene las líneas del fichero (`lista_lineas`).

Al terminar de modificar el *array*, se escriben todas las líneas en el fichero del historial de jugadores y se cierra.

```

41     if linea_encontrado == -1: #si no ha sido encontrado la linea_encontrado == -1
42         if self.ganador == "EMPATE!":
43             p_empatadas += 1
44         elif self.ganador == jugador:
45             p_ganadas += 1
46         else:
47             p_perdidas += 1 #tras establecer si es o no el que gana la partida
48             #(jugador:p_ganadas:p_perdidos:p_empatadas:\n) --> formato
49             linea = str(jugador) + ":" + str(p_ganadas) + ":" + str(p_perdidas) + ":" + str(p_empatadas) + ":\n" #crea la linea con el f
50             lista_lineas.append(linea) #y lo añade al final de la lista de lineas
51     else:
52         linea_jugador = lista_lineas[linea_encontrado].split(":") #vuelvo a dividir la linea que quiero en una lista de str
53         p_empatadas = int(linea_jugador[3]) #partidas empatabadas
54         p_ganadas = int(linea_jugador[1]) #partidas ganadas
55         p_perdidas = int(linea_jugador[2]) #partidas perdidas
56         if self.ganador == "EMPATE!":
57             p_empatadas = p_empatadas + 1
58         elif self.ganador == jugador:
59             p_ganadas = p_ganadas + 1
60         else:
61             p_perdidas = p_perdidas + 1 #tras modificar las partidas ganadas/perdidas/empatadas
62
63             linea = str(jugador) + ":" + str(p_ganadas) + ":" + str(p_perdidas) + ":" + str(p_empatadas) + ":\n" #crea la linea con el f
64             lista_lineas[linea_encontrado] = linea #sustituye el historial anterior por el nuevo
65
66     for l in lista_lineas:
67         fichero_jugadores.write(l) #escribe todos los historiales en el fichero
68
69     fichero_jugadores.close() #lo cierra al final

```

def buscar_jugador(self, jugador):

Cuando se quiera buscar la línea en la que se encuentra un *TAG* dentro del fichero “Historial_Jugadores.txt” se utiliza este método.

Tras abrir el archivo y guardar todas las líneas que haya en una variable, se recorren buscando el nombre que se recibe por parámetro, junto con una variable contador para saber en qué línea se encuentra. Cuando hay una coincidencia, se guarda el número de línea (desde 0, hasta número de líneas -1) y junto con todas las líneas en una lista, y se devuelve con un *return*. En el caso de que no se encuentre, se devuelve en número de línea como “-1” (para indicar que no existe) con el contenido del fichero.

```

71     def buscar_jugador(self,jugador):
72         fichero_jugadores = open("ficheros/Historial_Jugadores.txt","r")
73         i = 0 #var contador
74
75         lineas = fichero_jugadores.readlines() #guardo las lineas en una lista de str
76         for l in lineas: #recorro toda la lista de lineas
77             array_linea = l.split(":") #separo la linea en otra lista (jugador:p_ganados:p_perdidos:p_perdidos:\n)
78
79             if jugador == array_linea[0]: #si coincide el tag del jugador con el que hay en el fichero
80                 lista = [i, lineas] #guarda en una lista la linea en la que se encuentra (0 = linea 1, 1 = linea 2, 2 = linea 3...)
81                 fichero_jugadores.close()
82                 return lista #si lo encuentra envía la linea en la que se ha encontrado junto al resto de lineas
83
84             i+=1 #pasa a la siguiente linea
85
86         fichero_jugadores.close()
87         lista = [-1, lineas] #si no encuentra al jugador
88         return lista #devuelve todas las lineas y el número que indica que no se ha encontrado

```

def get_fecha_hora_actual(self):

Para formatear la fecha y hora actuales, se ha utilizado `datetime.now.strftime()`. Posteriormente, he dividido la fecha y la hora de la variable que recibe el *string* del método anteriormente mencionado, para cambiar el mes de inglés a español. Una vez modificado, se guarda la fecha y la hora en un *array* y se devuelve al finalizar el método.

```

90     def get_fecha_hora_actual(self):
91         fecha_hora = datetime.now().strftime('%d %m %Y, %H:%M:%S')
92
93         hora = fecha_hora[12:fecha_hora.__len__()] #cojo la hora dada por la var fecha_hora (Hora:Min:Seg)
94
95         fecha = list(fecha_hora[:10]) #recibe la fecha de la var fecha_hora
96
97         mes = int(fecha[3] + fecha[4]) #cojo los dos digitos del mes de la variable fecha_hora
98         fecha[3] = "de "
99         fecha[4] = self.meses[mes-1] + " de" #lo actualizo al nombre en español de la lista del constructor
100        fecha = "".join(fecha) #guardo la fecha formateada
101
102        fecha_hora_formateada = ["","","] #creo una lista vacia con los dos elementos [fecha, hora]
103        fecha_hora_formateada[0] = fecha
104        fecha_hora_formateada[1] = hora
105        return fecha_hora_formateada #los guardo en la lista y la devuelvo

```

7 PRUEBAS

En este apartado se van a definir varios casos de prueba para comprobar el correcto funcionamiento de la aplicación. Principalmente, se van a probar los cambios de estado, así como los resultados de partidas y su correcto guardado de historial.

Por otro lado, también se va a comprobar que los errores que pueden suceder durante la ejecución del aplicativo se manejan correctamente (casos de prueba negativos) y orientan al usuario como corresponda.

7.1 Casos de pruebas

- 23-5-2024/Jorge Correyero Fernández/Versión a probar → 13.
- Caso de prueba: **C001**
- Descripción:

Al iniciar la aplicación, se entra en el menú opciones y se modifican los ajustes.

- Condiciones de ejecución:

Se debe iniciar la aplicación ejecutando “main.py” y entrar en el menú “Opciones”.

- Entrada:

Hacer *click* en alguno de los números que modifican los *FPS*, apagar o encender la música al igual que los efectos de sonido, cambiar el porcentaje de volumen de ambas opciones.

- Resultado esperado:

Se marca con un rectángulo blanco los *FPS* seleccionados cambia el estado de los botones *ON/OFF* al hacerles *click*, se indican el porcentaje de volumen seleccionado al lado de la barra correctamente. Se guardan todos los ajustes para el resto de la ejecución hasta que se vuelvan a cambiar.

- Resultado obtenido:

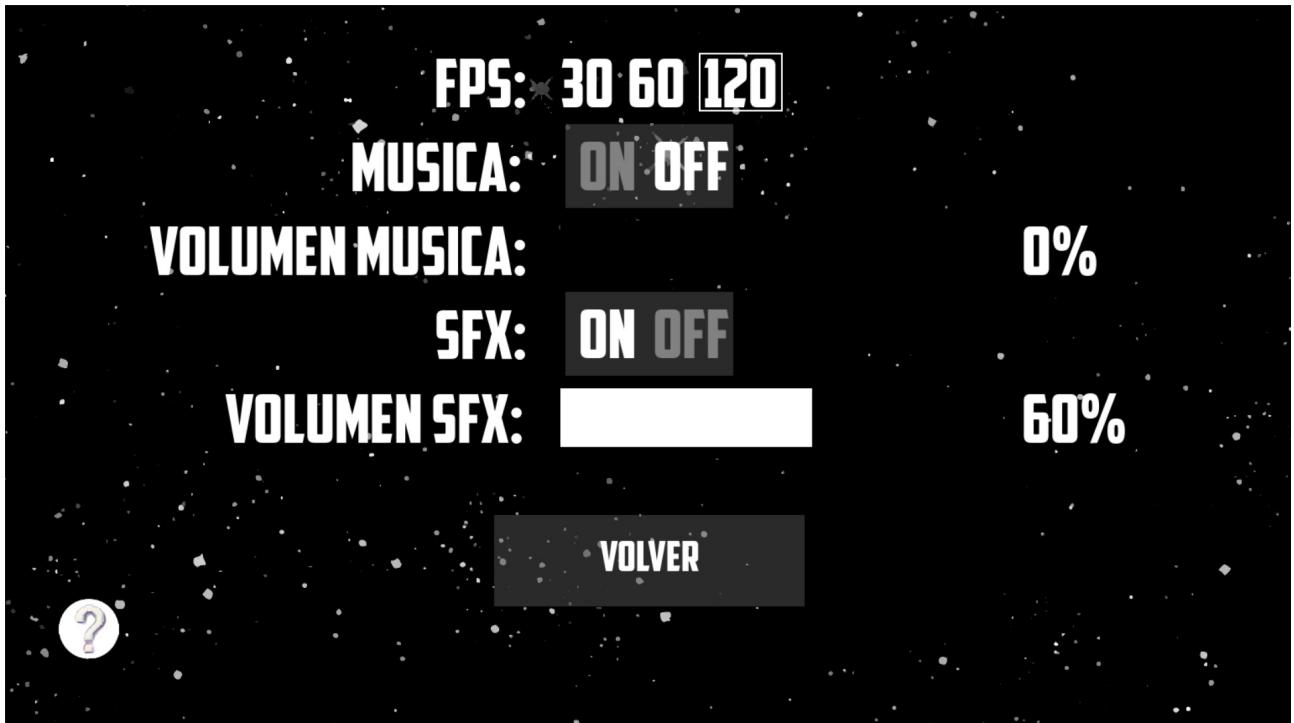


Imagen 1: Se mantienen las opciones escogidas al cambiar de estados

- Evaluación:

Funciona correctamente la modificación y guardado de opciones de este menú.

- 23-5-2024/Jorge Correyero Fernández/Versión a probar → 13.
- Caso de prueba: **C002**
- Descripción:

Se muestra la imagen de los controles al mantener pulsado el ícono que aparece en los menús abajo a la izquierda.

- Condiciones de ejecución:

Se debe iniciar la aplicación ejecutando “main.py” y entrar en todos los menús para comprobar que funciona en todos ellos.

- Entrada:

Mantener pulsado con el botón izquierdo del ratón el ícono de ayuda.

- Resultado esperado:

Aparece en pantalla una imagen que muestra los controles para cada jugador.

- Resultado obtenido:

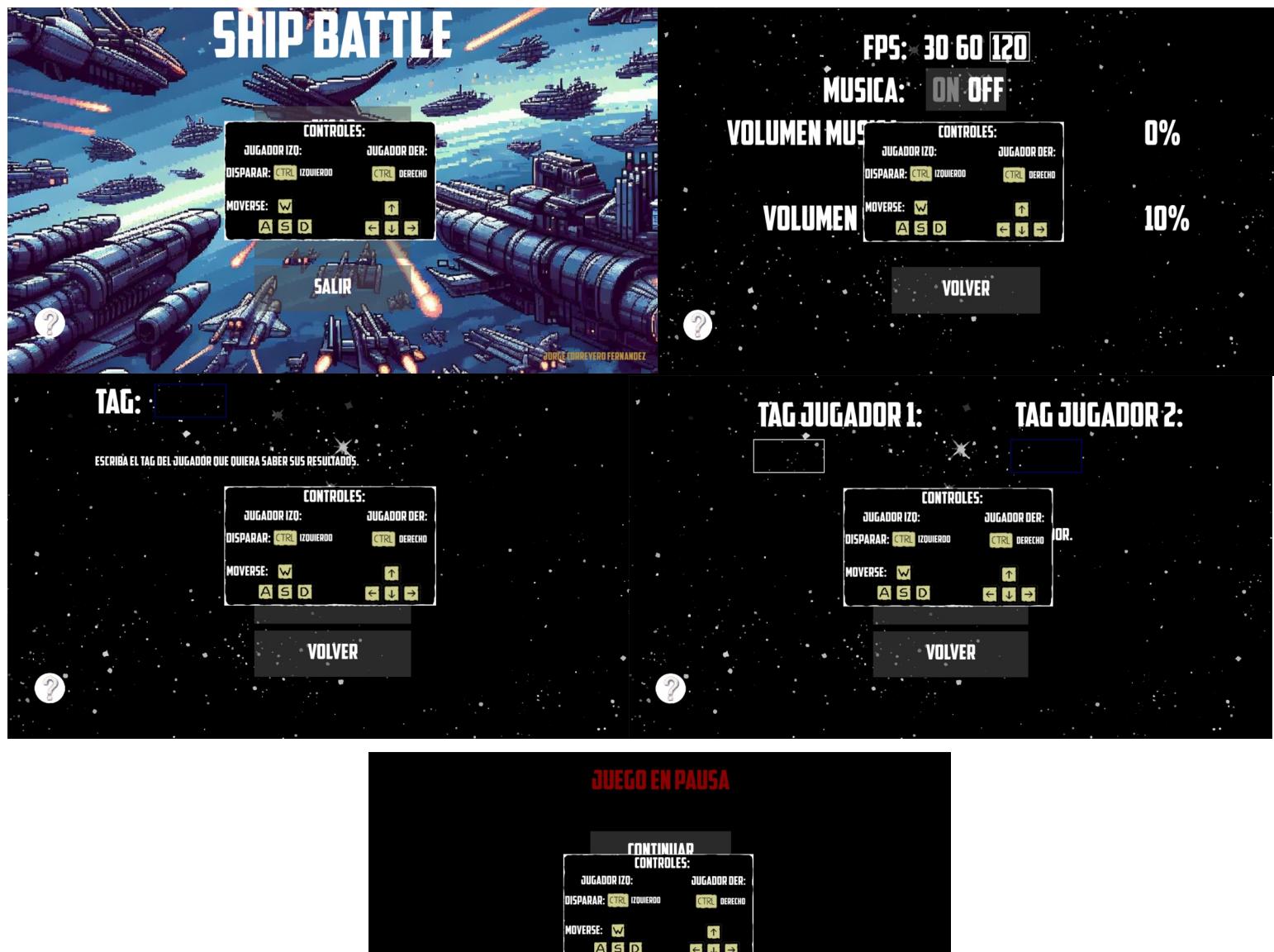


Imagen 2: Se muestra la imagen de los controles.

- Evaluación:

Se muestra correctamente el panel de controles y también desaparece al levantar el botón izquierdo del ratón en cualquier lugar de la pantalla.

-
- 23-5-2024/Jorge Correyero Fernández/Versión a probar → 13.
 - Caso de prueba: **C003**
 - Descripción:

Cuando se introducen ambos TAGs distintos permite al usuario continuar al estado de “Juego”.

- Condiciones de ejecución:

Se debe iniciar la aplicación ejecutando “main.py” y entrar en el estado menú “Selección de TAGs”.

- Entrada:

Introducir distintos TAGs en cada rectángulo de *input* y hacer *click* en el botón “jugar”.

- Resultado esperado:

No muestra mensaje de error y permite entrar a la pantalla del juego.

- Resultado obtenido:



Imagen 3: Se introducen los TAGs y se pulsa "jugar"

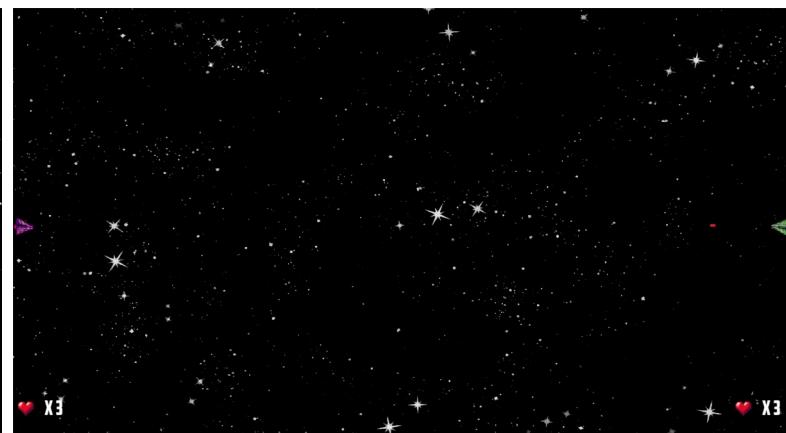


Imagen 4: Entra en el estado "Juego"

- Evaluación:

Entra correctamente en el estado “Juego” si se introducen ambos TAGs y son distintos.

-
- 23-5-2024/Jorge Correyero Fernández/Versión a probar → 13.
 - Caso de prueba: **C004**
 - Descripción:

Se debe iniciar la aplicación ejecutando “main.py”, entrar en el estado menú “Selección de TAGs”, y escribir ambos TAGs antes de presionar el botón “jugar” para entrar en estado de “Juego”.

- Condiciones de ejecución:

Descripción de las condiciones de ejecución que se deben cumplir antes de iniciar el caso de prueba, por ejemplo, que se haya realizado correctamente el login en el sistema, ...

- Entrada:

Presionar los botones “ctrl” de cada jugador.

- Resultado esperado:

Se resta la vida del jugador que corresponda y ambos entran en estado de inmunidad.

- Resultado obtenido:

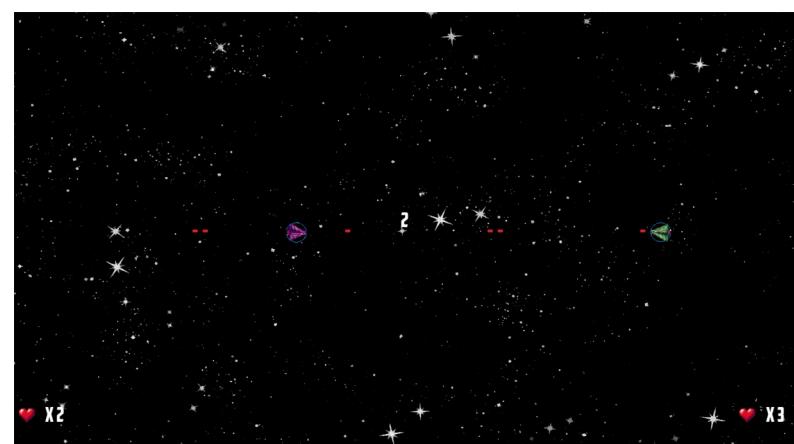


Imagen 6: Balas de la “nave2 con “nave1”

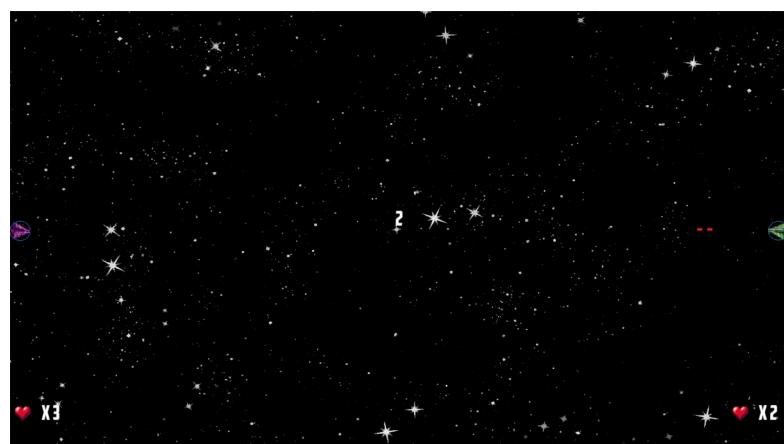


Imagen 5: Balas de “nave1” con “nave2”

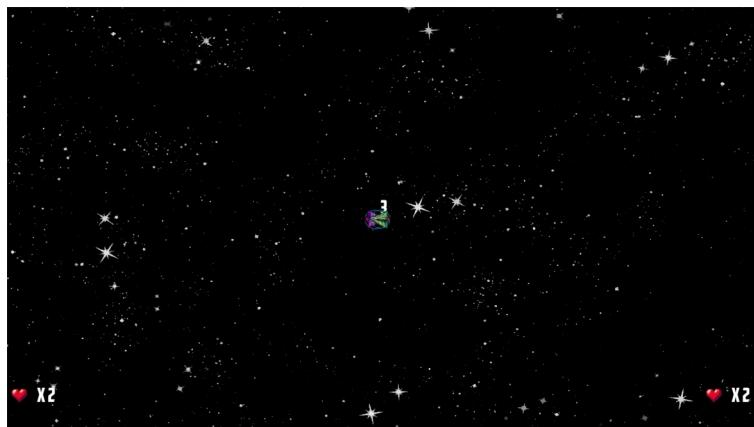


Imagen 7: “nave1” y “nave2” chocan



Imagen 8: Cambia la imagen de ambas naves

- Evaluación:

Las colisiones funcionan perfectamente y se restan las vidas que corresponden. Aparte, vuelve a aparecer el contador de inmunidad y se cambia la imagen de ambas naves

-
- 23-5-2024/Jorge Correyero Fernández/Versión a probar → 13.
 - Caso de prueba: **C005**

- Descripción:

Se comprueba que se puede pausar el juego y funciona correctamente el botón “continuar”.

- Condiciones de ejecución:

Se debe iniciar la aplicación ejecutando “main.py”, entrar en el estado menú “Selección de TAGs”, y escribir ambos TAGs antes de presionar el botón “jugar” para entrar en estado de “Juego”.

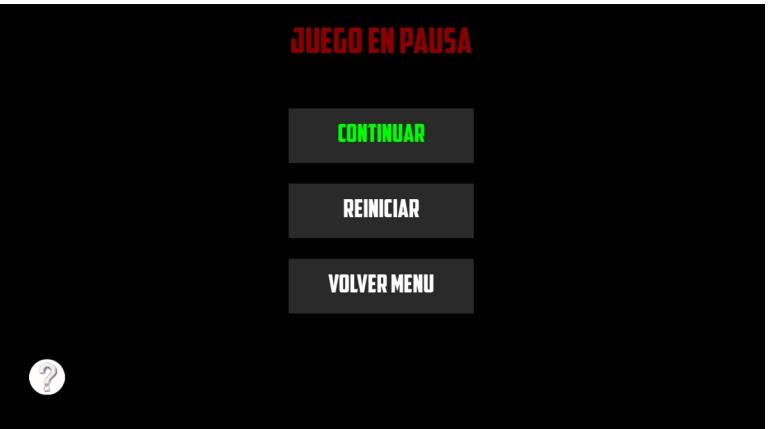
- Entrada:

Presionar la tecla “Esc” y hacer *click* en el botón “continuar” del menú.

- Resultado esperado:

Se pausa el juego y se reanuda al pulsar el botón “continuar” en el estado que estuviese previamente.

- Resultado obtenido:



JUEGO EN PAUSA

CONTINUAR

REINICIAR

VOLVER MENU

?

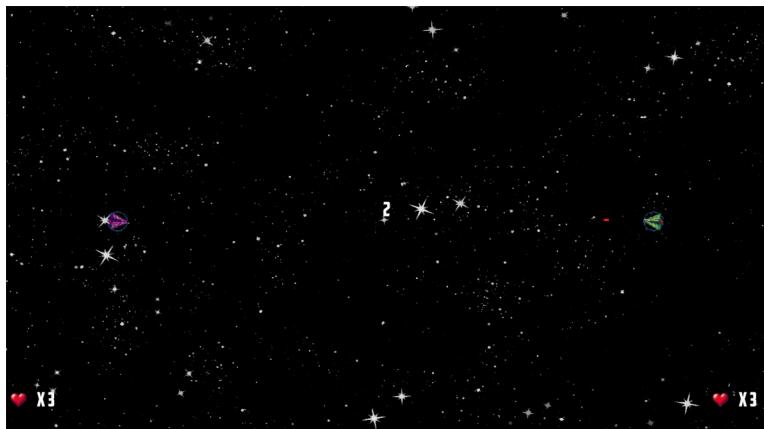


Imagen 9: Se pulsa "continuar"

Imagen 10: Reanuda el estado de "Juego"

- Evaluación:

Cuando se entra al juego, se mueven las naves hacia delante y se pausa inmediatamente el juego (en 1 segundo aproximadamente), se mantiene el estado previo a la pausa (se puede ver porque el contador está a 2).

- 23-5-2024/Jorge Correyero Fernández/Versión a probar → 13.
- Caso de prueba: **C006**
- Descripción:

Se comprueba que se puede pausar el juego y funciona el botón “reiniciar” como debe.

- Condiciones de ejecución:

Se debe iniciar la aplicación ejecutando “main.py”, entrar en el estado menú “Selección de TAGs”, y escribir ambos TAGs antes de presionar el botón “jugar” para entrar en estado de “Juego”.

- Entrada:

Presionar la tecla “Esc” y hacer *click* en el botón “reiniciar” del menú.

- Resultado esperado:

Se pausa la partida, y se reinicia el estado de “Juego” al presionar el botón.

- Resultado obtenido:



Imagen 11: Se pulsa "reiniciar"



Imagen 12: Reinicia y entra en el estado de "Juego"

- Evaluación:

Cuando se pausa y reinicia el juego se restablece la posición de las naves, reinicia el estado de inmunidad inicial, y vuelve a poner 3 vidas a cada nave.

-
- 23-5-2024/Jorge Correyero Fernández/Versión a probar → 13.
 - Caso de prueba: **C007**
 - Descripción:

Se comprueba que se puede pausar el juego y funciona el botón “volver menu” correctamente.

- Condiciones de ejecución:

Se debe iniciar la aplicación ejecutando “main.py”, entrar en el estado menú “Selección de TAGs”, y escribir ambos TAGs antes de presionar el botón “jugar” para entrar en estado de “Juego”.

- Entrada:

Presionar la tecla “Esc” y hacer *click* en el botón “volver menu” del menú.

- Resultado esperado:

Tras pausar el juego y hacer *click* en el botón, vuelve al estado menú “Principal”.

- Resultado obtenido:



Imagen 13: Se pulsa “volver menu”



Imagen 14: Entra en el estado menú "Principal"

- Evaluación:

Cambia de estado correctamente cuando se quiere volver al menú “Principal” desde el menú de “Pausa”.

-
- 23-5-2024/Jorge Correyero Fernández/Versión a probar → 13.
 - Caso de prueba: **C008**
 - Descripción:

Se comprueba que ambas naves pueden perder o empatar si sus vidas llegan a 0.

- Condiciones de ejecución:

Se debe iniciar la aplicación ejecutando “main.py”, entrar en el estado menú “Selección de TAGs”, y escribir ambos TAGs antes de presionar el botón “jugar” para entrar en estado de “Juego”.

- Entrada:

Presionar la tecla “ctrl” hasta tener 0 vidas en cada jugador o para empatar chocar las naves moviéndolas hacia delante.

- Resultado esperado:

Entrar en estado “Game Over” y mostrar el mensaje correspondiente al resultado.

- Resultado obtenido:

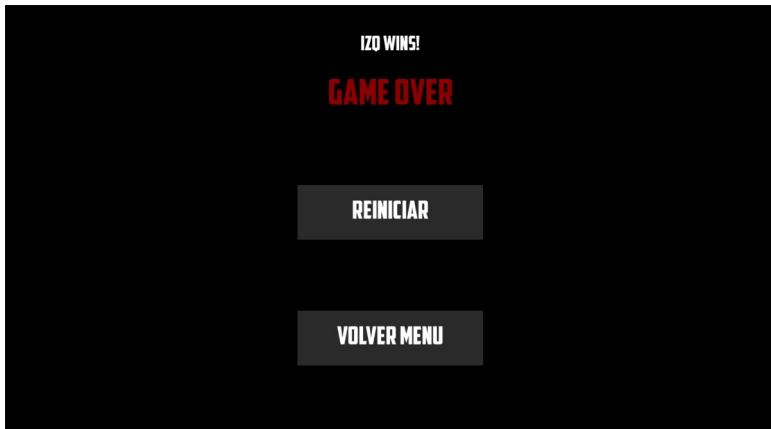


Imagen 16: Gana "nave1"

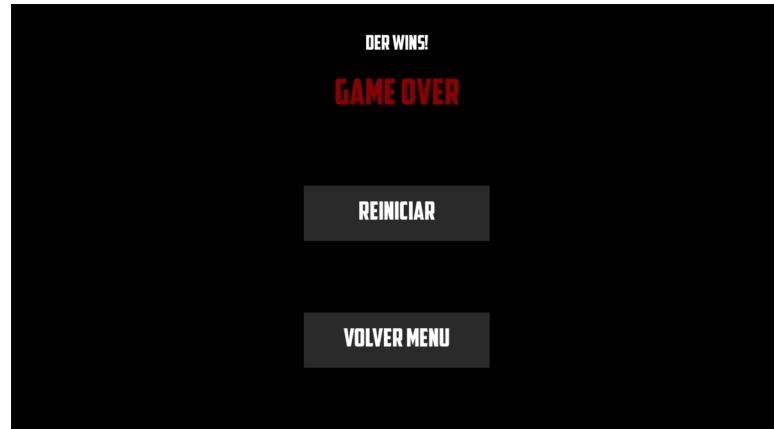


Imagen 15: Gana "nave2"

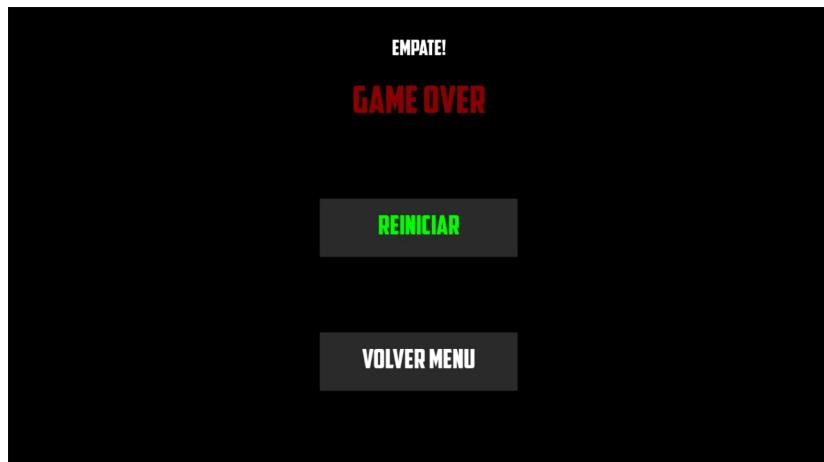


Imagen 17: Empatan ambos jugadores

- Evaluación:

Se cambia de estado y establece el ganador (o empate) correctamente.

- 23-5-2024/Jorge Correyero Fernández/Versión a probar → 13.
- Caso de prueba: **C009**
- Descripción:

Se prueba el botón “reiniciar” del estado “Game Over”.

- Condiciones de ejecución:

Tras finalizar el estado “Juego”, entrar en “Game Over”.

- Entrada:

Hacer *click* en el botón “reiniciar”.

- Resultado esperado:

Vuelve al estado “Juego”, reiniciando el estado inicial de las naves.

- Resultado obtenido:

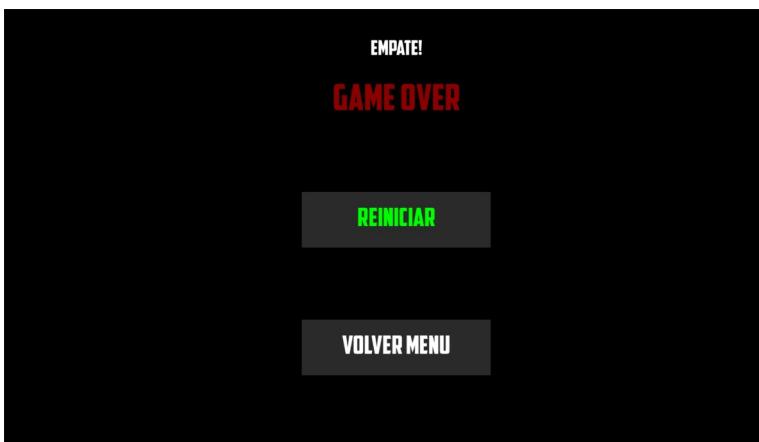


Imagen 19: Se presiona el botón "reiniciar"

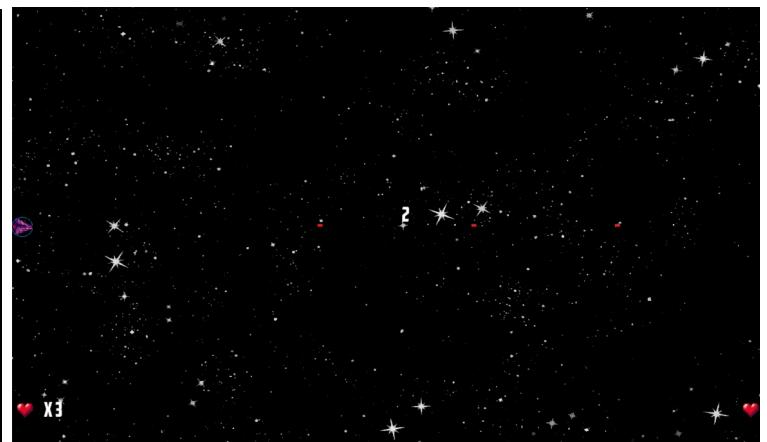


Imagen 18: Se vuelve al estado inicial de "Juego"

- Evaluación:

Se reinicia la partida correctamente desde el estado “Game Over”.

- 23-5-2024/Jorge Correyero Fernández/Versión a probar → 13.
- Caso de prueba: **C010**
- Descripción:

Se prueba el botón “volver menu” del estado “Game Over”.

- Condiciones de ejecución:

Descripción de las condiciones de ejecución que se deben cumplir antes de iniciar el caso de prueba, por ejemplo, que se haya realizado correctamente el login en el sistema, ...

- Entrada:

Hacer *click* en el botón “volver menu”.

- Resultado esperado:

Valor esperado para el correcto funcionamiento del proyecto.

- Resultado obtenido:

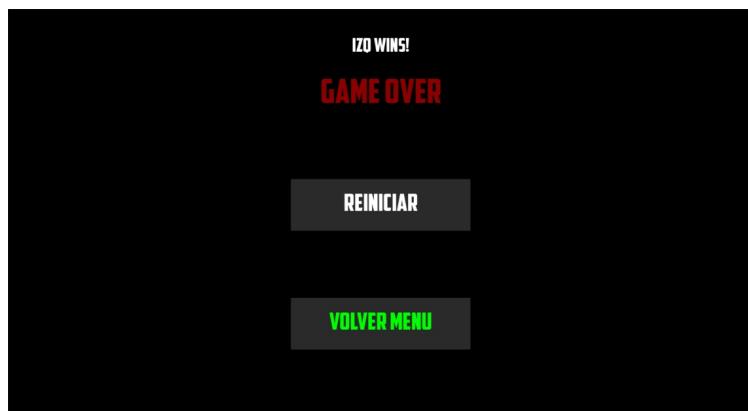


Imagen 20: Se pulsa el botón "volver menu"



Imagen 21: Se entra en el estado menú "Principal"

- Evaluación:

Se cambia de estado correctamente

- 23-5-2024/Jorge Correyero Fernández/Versión a probar → 13.
- Caso de prueba: **C011**
- Descripción:

Se prueba la búsqueda de resultados de un TAG que haya jugado al menos una partida.

- Condiciones de ejecución:

Se debe iniciar la aplicación ejecutando “main.py” y entrar en el menú “Historial”.

- Entrada:

Hacer *click* en el rectángulo de *input* y escribir el TAG de un jugador.

- Resultado esperado:

Se muestran las partidas ganadas, perdidas y empatadas (estadísticas) del jugador que se ha buscado.

- Resultado obtenido:



Imagen 22: Se busca las estadísticas de "IZQ"

```
ficheros >  ≡ Historial_Jugadores.txt
1   AAA:1:0:0:
2   BBB:0:1:0:
3   IZQ:2:1:5:
4   DER:1:2:5:
5   |
```

Imagen 23: Contenido fichero "Historial_Jugadores.txt"

- Evaluación:

Funciona correctamente la búsqueda y se muestra el resultado correctamente.

- 23-5-2024/Jorge Correyero Fernández/Versión a probar → 13.
- Caso de prueba: **C012**
- Descripción:

Se prueba la búsqueda del historial de las 10 últimas partidas.

- Condiciones de ejecución:

Se debe iniciar la aplicación ejecutando “main.py” y entrar en el menú “Historial”.

- Entrada:

Hacer *click* en el botón “ver 10 partidas”.

- Resultado esperado:

Aparece en pantalla las 10 últimas partidas a la derecha. En este caso solo se han jugado 9, por lo que aparecen las 9 últimas.

- Resultado obtenido:



Imagen 24: Muestra en pantalla las 10 últimas

```
ficheros > Historial_Partidas.txt
1 AAA GANA A BBB EL 23 de MAYO de 2024 a las 15:54:05
2 IZQ & DER EMPATAN EL 23 de MAYO de 2024 a las 16:18:03
3 IZQ & DER EMPATAN EL 23 de MAYO de 2024 a las 16:20:16
4 IZQ & DER EMPATAN EL 23 de MAYO de 2024 a las 20:54:52
5 DER GANA A IZQ EL 23 de MAYO de 2024 a las 21:44:22
6 IZQ GANA A DER EL 23 de MAYO de 2024 a las 21:45:04
7 IZQ & DER EMPATAN EL 23 de MAYO de 2024 a las 21:47:04
8 IZQ & DER EMPATAN EL 23 de MAYO de 2024 a las 22:07:24
9 IZQ GANA A DER EL 23 de MAYO de 2024 a las 22:12:51
10
```

Imagen 25: Contenido fichero "Historial_Partidas.txt"

- Evaluación:

Funciona correctamente la búsqueda y se muestra el resultado correctamente.

7.2 Casos de pruebas negativos

- 23-5-2024/Jorge Correyero Fernández/Versión a probar → 13.
- Caso de prueba: **CN001**.
- Descripción:

Se comprueba que no se muestran partidas ganadas, perdidas o empatadas hasta que el usuario introduzca un TAG.

- Condiciones de ejecución:

Se debe iniciar la aplicación ejecutando “main.py” y entrar en el menú “Historial”.

- Entrada:

No se debe introducir por teclado en el apartado TAG ningún carácter o de lo contrario intentar introducir “espacios”.

- Resultado esperado:

Mensaje en pantalla indicando al usuario que introduzca el TAG del jugador que quiera saber sus resultados.

- Resultado obtenido:



Mensaje en pantalla: “ESCRIBA EL TAG DEL JUGADOR QUE QUIERA SABER SUS RESULTADOS”

- Evaluación:

Se captura correctamente el error y se indica al usuario que debe hacer.

- 23-5-2024/Jorge Correyero Fernández/Versión a probar → 13.
- Caso de prueba: **CN002**
- Descripción:

Se comprueba que al no existir el *TAG* introducido, se le notifica al usuario.

- Condiciones de ejecución:

Se debe iniciar la aplicación ejecutando “main.py” y entrar en el menú “Historial”.

- Entrada:

Escribir en el recuadro del *input* un *TAG* que no exista en el fichero “Historial_Jugadores.txt”.

- Resultado esperado:

Mensaje en pantalla indicando al usuario que el *TAG* del jugador no existe.

- Resultado obtenido:



Mensaje en pantalla: “TAG NO ENCONTRADO”

- Evaluación:

Se captura correctamente el error y se indica al usuario que no existe el *TAG* introducido.

- 23-5-2024/Jorge Correyero Fernández/Versión a probar → 13.
- Caso de prueba: **CN003**
- Descripción:

Se comprueba que no se muestra ningún historial si el usuario no ha jugado ninguna partida.

- Condiciones de ejecución:

Se debe iniciar la aplicación ejecutando “main.py”, entrar en el menú “Historial” y tener el fichero “Historial_Partidas.txt” vacío.

- Entrada:

Hacer *click* en el botón ver 10 partidas.

- Resultado esperado:

Valor esperado para el correcto funcionamiento del proyecto.

- Resultado obtenido:



Mensaje en pantalla: “NO SE HA ENCONTRADO NINGÚN HISTORIAL”

- Evaluación:

Se captura correctamente el error y se indica al usuario que no hay ningún historial de partidas.

-
- 23-5-2024/Jorge Correyero Fernández/Versión a probar → 13.
 - Caso de prueba: **CN004**
 - Descripción:

Se introduce un mismo nombre de jugador para ambos, lo cual no debería hacerse para poder guardar resultados individuales.

- Condiciones de ejecución:

Se debe iniciar la aplicación ejecutando “main.py” y presionar el botón “jugar” del menú “Principal”.

- Entrada:

Introducir el mismo TAG para ambos jugadores y presionar el botón “jugar”.

- Resultado esperado:

No permite continuar al estado “Juego”.

- Resultado obtenido:



Mensaje en pantalla: “ESCOJA TAGS DISTINTOS PARA CADA JUGADOR”

- Evaluación:

No se permite al usuario continuar correctamente y se le indica en pantalla que escoja nombres diferentes para cada jugador. Ocurre lo mismo al no introducir ningún TAG.

-
- 23-5-2024/Jorge Correyero Fernández/Versión a probar → 13.
 - Caso de prueba: **CN005**
 - Descripción:

No se permite continuar al estado “Juego” si no se han introducido ambos TAGs.

- Condiciones de ejecución:

Se debe iniciar la aplicación ejecutando “main.py” y presionar el botón “jugar” del menú “Principal”.

- Entrada:

Introducir un solo TAG y presionar el botón “jugar”.

- Resultado esperado:

No permite continuar al estado “Juego”.

- Resultado obtenido:



Mensaje en pantalla: “ESCRIBA AMBOS TAGS ANTES DE CONTINUAR”

- Evaluación:

No se permite al usuario continuar correctamente y se le indica en pantalla que debe escribir los dos *TAGs* antes de continuar.

8 CONCLUSIONES

El desarrollo de un juego puede suponer un gran trabajo si lo realiza una sola persona, debido a que es la encargada de mejorar, arreglar e implementar el código. Para la búsqueda de errores o comprobar ciertos funcionamientos del videojuego, se han puesto *prints* para comprobar en la terminal diversos valores de variables. Por otro lado, se decidió no utilizar una base de datos porque suponía un mayor consumo de memoria al ordenador del usuario.

Sin duda alguna, la función que más problemas ha supuesto es `.pause()` de `pygame.mixer.music` para el estado de “Game Over”. Se arregló el estado “Game Over” debuggeando el método con diversos *breakpoints*, para concluir que la función `self.musica_fondo.pause()` se debía ejecutar una sola vez al cambiar de estado.

Durante el desarrollo, se han ido guardando diferentes versiones del juego según se iban realizando cambios, llegando a tener 13 en total. Esto se ha hecho para evitar perder el progreso y solucionar errores que podían surgir al implementar o modificar código.

Tras finalizar, se ha revisado la memoria y el código del juego varias veces para optimizarlo lo máximo posible. Tras ello, se puede concluir que se han cumplido perfectamente los objetivos para permitir que la aplicación pueda ser ejecutada en ordenadores de bajo rendimiento. Esto se debe a que el juego ocupa tan solo 64,6 MB, y consume apenas 34,2 MB de RAM. Aun con ello, la resolución mínima de la pantalla deberá ser de 720p.

Name	Status	CPU	Memory	Disk	Network	GPU
▼ Python		3,4%	34,2 MB	0,1 MB/s	0 Mbps	0%
Ship Battle						

Captura 1: Juego en el administrador de tareas

9 FUENTES

9.1 Legislación

- DAM

Enseñanzas mínimas: Real Decreto 450/2010, de 16 de abril (BOE 20/05/2010)

http://pdf/IFCS02/titulo/RD20100450_TS_Desarrollo_Aplicaciones_Multiplataforma.pdf

Curriculum: [D. 3/2011, de 13 de enero \(BOCM 31/01/2011\)](#)

http://pdf/IFCS02/curriculo/D20110003_TS_Desarrollo_Aplicaciones_Multiplataforma.pdf

Definición de procedimientos de control y evaluación:

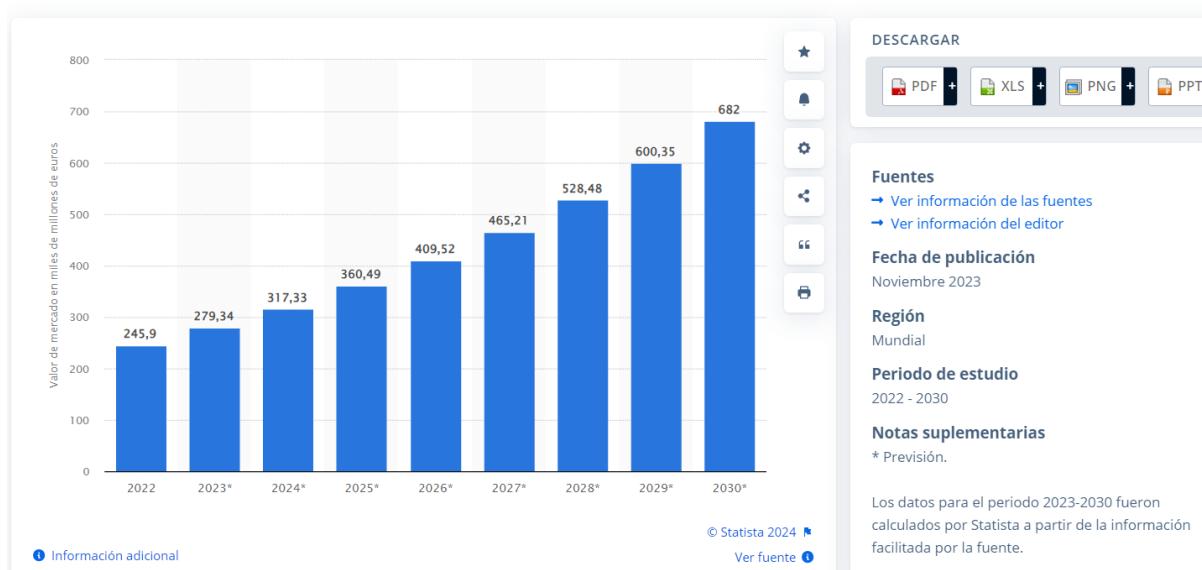
- <http://www.xperta.es/es/descripcion.asp>
- <http://www.xperta.es/es/aquienvadirigido.asp>
- <http://churriwifi.wordpress.com/2010/04/10/gestion-de-incidencias/>
- http://es.wikipedia.org/wiki/Control_de_versiones

9.2 Bibliografía

Estadísticas videojuegos:

<https://es.statista.com/temas/9150/industria-mundial-del-videojuego/>

Evolución del valor de mercado de la industria del videojuego en el mundo entre 2022 y 2030
(en miles de millones de dólares)



<https://vandal.elespanol.com/noticia/1350758853/casi-el-95-de-las-ventas-de-videojuegos-en-2022-fueron-en-formato-digital/>

Space Ship Infinity:

https://store.steampowered.com/app/1482050/Space_Ship_Infinity/

Sectores tecnológicos:

<https://retos-operaciones-logistica.eae.es/tecnologia-y-sectores-de-produccion/#:~:text=Se%20trata%20entre%20otras%20de,las%20telecomunicaciones%20y%20la%20biotecnología.>

Diagrama Gantt:

<https://app.asana.com/0/1207284324975564/board>

Publicación juego Steam:

<https://partner.steamgames.com/steamdirect?l=latam>

Diagrama de Flujo:

https://www.canva.com/es_es/pizarra-online/diagramas-flujo/

Consulta código:

<https://stackoverflow.com>

Música juego y menús:

<https://www.youtube.com/watch?v=5bn3Jmvep1k>

Generar Imágenes con IA:

<https://www.bing.com/images/create?toWww=1&redig=DDA9833D58D149B28398193306311B00>

Imagen Fondo:

https://www.freepik.es/vector-gratis/fondo-galaxia-acuarela_21643353.htm#fromView=image_search_similar&page=1&position=17&uuid=94c97f65-08cb-42eb-b659-323a2e24d6d6

Quitar Fondo Imágenes:

<https://www.remove.bg/es/upload>

Editar Imágenes:

<https://www.photopea.com>

Reescalar Imágenes:

<https://www.iloveimg.com/es/redimensionar-imagen>

Editar Audios:

<https://mp3cut.net/es/>

Documentación PyGame:

<https://www.pygame.org/news>

Documentacion PyGame_menu:

https://pygame-menu.readthedocs.io/en/4.3.9/_source/widgets_scrollbar.html