UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Desarrollo de un videojuego arcade en Unity

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Adrián Bernet Medina

Tutor: Javier Lluch Crespo

Curso 2018-2019

Resumen

En este documento se va a redactar el proceso de desarrollo de un videojuego de estilo "arcade" empleando la versión 2018.3.9 del motor gráfico **Unity 3D** y el editor de texto **Visual Studio Community 2017**. El objetivo de este proyecto es que, a partir de un tutorial básico facilitado por la propia desarrolladora de Unity, pueda crearse un videojuego completo con menús, selector de personajes y varios modos de juego.

En este juego, el jugador controlará una nave, con la que deberá ir esquivando y destruyendo enemigos e intentar alcanzar la máxima puntuación. Para ello, dispondrá de diversas mecánicas (como esquivar y realizar un ataque especial), además de la posibilidad de recoger objetos que ayuden al jugador.

Palabras clave: Unity, videojuego, C#.

Abstract

In this document will describe the process of developing an arcade-style videogame using version 2018.3.9 of the **Unity 3D** game engine and the **Visual Studio Community 2017** text editor. The objective of this project is that, starting from a basic tutorial created by the Unity's own developer, create a complete videogame with menus, character selector and several game modes.

In this game, the player will control a ship, with which he must dodge and destroy enemies and try to reach the maximum score. To do this, the player will have some mechanics (such as dodge and make a special attack), as well as the ability to collect objects that help the player.

Keywords : Unity, videogame, C#.

Agradecimientos

A mi familia, en especial a Josephine, por el apoyo constante.

A mis “bio-compas” del piso, por soportarme todo este tiempo.

A mis amigos, tanto los de clase, como los del Gali y los del pueblo.

A Xavi Lluch, mi tutor, por aguantar mi insistencia.

Índice de contenidos

Índice de Figuras

Índice de Tablas

1. Introducción

Desde los últimos años, la industria del videojuego no ha parado de crecer. Es un mercado amplio, llegando al punto de haber superado los ingresos generados tanto por el mercado del cine como el de la música **[1]**. Esta puede ser una de las razonas por las que el interés de entrar en el mundo del desarrollo de videojuegos aumente.

De la misma forma que dicho interés aumenta, aparecen en el mercado diversas herramientas que facilitan el desarrollo de videojuegos, de las cuales algunas tienen una versión gratuita para que cualquier usuario pueda introducirse en el mundo del desarrollo y experimentarlo de primera mano. Algunas de las herramientas que tienen versión gratuita son Unreal Engine **[2]** o Unity 3D **[3]**, herramienta elegida para el desarrollo de este proyecto puesto que es más sencilla de utilizar que la competencia, además de ser la que mejor se ajusta a las necesidades del proyecto **[41]**.

En el caso de este último, además de permitir el acceso a la herramienta a cualquier persona que la desee, también tiene una serie de tutoriales básicos gratuitos como introducción para los usuarios y comprender así el funcionamiento del programa **[4]**.

Llevo jugando a videojuegos muchos años, por lo que cuando se me presentó la oportunidad de poder realizar mi propio juego como trabajo de final de grado fue una oportunidad que no podía rechazar. Siempre he creído en que los videojuegos son el mejor medio para contar historias, puesto que permiten la combinación de múltiples formas de narrativa. Además de hacer al jugador partícipe de la historia que se cuenta, en lugar de ser un mero espectador como si se viese una película o leyese un libro, hace que la inmersión sea superior a cualquier otro medio de transmisión.

Por otro lado, cabe destacar que debido a la desinformación del medio, gran cantidad de gente cree que tienen mero interés lúdico. Sin embargo, al igual que en el cine, música u otros medios, hay bastantes títulos de temáticas variadas que tratan historias profundas o temas de actualidad mediante crítica social. Si a este medio se le diese mayor visibilidad, los prejuicios que tiene la sociedad en general hacia los videojuegos cambiarían.

En esta memoria se relatará el estado del arte de los motores gráficos, desde los primeros que hicieron aparición en la década de los 90 hasta los más actuales. También se hablará sobre la historia del “shoot ‘em up”, un estilo de juego arcade nacido a finales de los años 70, con el popular “Space Invaders” de “Taito”, el cual caracterizará el estilo del juego que se va a desarrollar en esta memoria. En el siguiente punto, se hablará sobre la herramienta empleada para este proyecto, el motor gráfico “Unity 3D”, junto con algunas de sus características principales, como pueden ser la creación de objetos y sus componentes, o la creación de código y su inclusión en el proyecto mediante scripts.

Posteriormente se explicarán las diversas características que se han diseñado para el juego y su justificación, tales como los modos de juego, las mecánicas de las que dispondrá el jugador, entre otros detalles de diseño. Después, en el apartado de desarrollo se explicarán todos los pasos a seguir en el tutorial y posteriormente todas las implementaciones añadidas.

Después, se desarrollarán los problemas encontrados durante las fases de pruebas junto con su posterior resolución. Y por último en el apartado de resultados se mostrará una ejecución del juego, pasando por los distintos menús que tiene el juego.

Para este trabajo, se toma como punto de partida uno de estos tutoriales, concretamente el titulado: “Introduction to Space Shooter” **[5]**, el cual implementa las mecánicas básicas del juego que se va a desarrollar, tales como la creación de un jugador, creación y aparición de

enemigos, sumar puntos, etc. Posteriormente se desarrollarán mecánicas adicionales para hacerlo más complejo y divertido de jugar, como por ejemplo crear un tipo nuevo de nave o implementar un modo de juego diferente.

Por último, el desarrollo de este proyecto tiene diversos objetivos, entre los cuales destaca la profundización en el uso de motores gráficos y el establecimiento de las bases para crear un juego "arcade", con el cual se podrá participar en el concurso de "Minijuegos", organizado por el profesor la asignatura "Introducción a los Sistemas Gráficos Interactivos".

Dichos objetivos a cumplir son:

- Desarrollo completo del tutorial "Introduction to Space Shooter"
- Creación de varios tipos de naves con las que se puedan jugar
- Aumentar las mecánicas del juego
    - Creación de maniobra de evasión
    - Creación de habilidades especiales
- Crear una herramienta que permita al jugador escoger nave
- Desarrollo de distintos modos de juego
- Crear un sistema de menús funcional

2. Estado del arte

Desde el nacimiento del primer videojuego hasta la actualidad, los videojuegos y la tecnología siempre han ido parejos a lo largo de su evolución. Conforme la tecnología ha ido mejorando, los juegos cada vez han ido necesitando de más recursos para funcionar. Es por eso que a lo largo de la vida de los videojuegos ha habido multitud de herramientas y motores gráficos los cuales ya están en desuso, y otros los cuales ya están establecidos en el mercado.

2.1. Id Tech 1

Uno de los primeros motores más importantes de la industria fue "Id Tech 1", también conocido como "Doom Engine". Este motor nació en 1993 y fue desarrollado por la empresa "Id Software", la cual fue la creadora de grandes clásicos de la industria como fueron "Doom" (figura 1) y su secuela, "Doom 2: Hell on Earth". Aunque daba sensación tridimensional, este motor gráfico estaba limitado a crear escenarios en 2D, es decir, no podía existir una sala encima de otra. Por lo tanto, mediante el posicionamiento de objetos y el uso de técnicas para pasar de una sala a otra (mediante "teletransportadores" y similares) se daba al jugador la sensación de que subía o bajaba de piso, pero en realidad se trasladaba a otra sala a la misma altura que la anterior **[6]**.

Figura 1: Captura de "Doom" (1993)

2.2. Quake Engine

En 1996 aparece "Quake", el cual es también desarrollado por "Id Software". Este juego se desarrolló con el motor gráfico "Quake Engine", que fue uno de los primeros motores de la historia en permitir desarrollar juegos tridimensionales íntegramente creados con polígonos. Este juego dejó un gran legado en la industria tanto por el juego per se como por la tecnología empleada. A pesar de que ya existían juegos tridimensionales en aquella época, "Quake" (figura 2) poseía una fluidez en los ordenadores domésticos que la competencia no tenía, además de permitir el juego multijugador en línea. En 1999 publicaron el código fuente de este motor como software libre (licencia GNU GPL), la cual daba permiso a otras desarrolladoras a utilizar y modificar el código fuente, lo cual fue clave en el mundo del desarrollo de videojuegos. Paralelamente a este acontecimiento "Id Software" mejoró este motor con el lanzamiento de la secuela de "Quake", "Quake 2", el cual hacía uso de "Quake 2 Engine", también llamado "Id Tech 2" **[7]**. Este motor ha ido evolucionando hasta llegar a la versión "Id Tech 7", la cual se anunció en 2018 junto con el juego "Doom Eternal" (figura 3) **[8]**.

Figura 2: Captura de "Quake" (1996)

Figura 3: Captura de "Doom Eternal" (2019)

2.3. Source

Aunque la fecha del lanzamiento del motor "Source" queda lejana en 2004, es un motor gráfico desarrollado por "Valve Corporation" que a día de hoy es utilizado por varios juegos que en 2019 siguen activos, tales como "Counter Strike: Global Offensive" estrenado en 2012 o el reciente "Apex Legends", estrenado en febrero de 2019. Este motor gráfico nació como sucesor de "GoldSource", un motor desarrollado a partir del mencionado "Quake Engine" tras la publicación del código fuente **[9]** con el que se desarrolló "Half Life".

La primera aparición del motor "Source" fue en junio de 2004, con el lanzamiento del juego "Counter Strike: Source", y seguido de cerca por el popular "Half Life 2" (figura 4), estrenado en noviembre de ese mismo año **[10]**. Este motor está caracterizado por recibir constantes mejoras y actualizaciones, con lo cual el paso de los años no hace mucho efecto en él. Sin embargo, en 2015 se anunció una nueva versión de este motor, llamada "Source 2", la cual se emplea en juegos de la propia "Valve Corporation" como "Artifact" o "Dota 2", el cual se trasladó del motor "Source" a esta nueva versión (figura 5) **[11]**.

Figura 4: Captura de "Half Life 2" (2004)

Figura 5: Captura de "Dota 2" en el motor "Source 2" (2015)

2.4. CryEngine

Uno de los motores actuales a los que se puede acceder de manera gratuita es "Cry Engine", desarrollado por la empresa alemana "Crytek". Pese a que el nacimiento de este motor data de 2002, no fue hasta 2004 que se utilizó en un videojuego, "Far Cry" (figura 6), desarrollado por la misma compañía y distribuido por "Ubisoft", que compraría la licencia de "Cry Engine" en 2006, el cual utilizaría para desarrollar su propio motor gráfico, "Dunia Engine", empleado principalmente para el resto de juegos de la saga "Far Cry" **[12]**.


Figura 6: Captura de "Far Cry" (2004)

Posteriormente este motor gráfico ha ido recibiendo mejoras y versiones hasta llegar a "Cry Engine V", la cual se puede descargar de manera gratuita desde su página oficial **[13]**. Las distintas versiones de este motor se han empleado en diversos juegos, siendo los más destacados los tres que conforman la saga "Crysis" (figura 7).

Figura 7: Captura de "Crysis 3" (2013)

2.5. Unreal Engine

Otro de los motores más usados por multitud de desarrolladoras es "Unreal Engine" creado por la empresa "Epic Games", el cual también puede obtenerse de manera gratuita. Tuvo su primera aparición en 1998, junto con la salida de "Unreal" (figura 8), el cual empleaba la primera versión de este motor. Su segunda versión data de 2002 junto con el lanzamiento de "America's Army". Esta versión pulía algunos aspectos y reescribía parte del núcleo del motor gráfico, además de hacer compatible el motor con las consolas del momento: "PlayStation 2" de Sony, "GameCube" de Nintendo y "Xbox" de Microsoft. Al igual que su anterior versión, se utilizó en multitud de juegos de diversos géneros como "BioShock" (figura 9), desarrollado por "2K Boston" o "Tom Clancy's Splinter Cell", de "Ubisoft".

Figura 8: Captura de "Unreal" (1998)

Figura 9: Captura de "BioShock" (2007)

Con la llegada de la Xbox 360 y la PlayStation 3 hace aparición la tercera versión de este motor gráfico en 2006, la cual sigue siendo utilizada en multitud de juegos como "BioShock Infinite" (figura 10) o "Unreal Torunament 3". En 2008 este motor recibió una versión intermedia (nombrada "Unreal Engine 3.25/5), la cual añadía mejoras visuales en efectos, tales como fluidos o iluminación y sombras, una IA mejorada, etc.


Figura 10: Captura de "BioShock Infinite" (2013)

En 2014, "Epic Games" anuncia la versión más reciente hasta la fecha de este motor gráfico, "Unreal Engine 4" el cual, dependiendo de la cantidad de ingresos generada, puede hacerse un uso gratuito de este motor, además de tener acceso completo a su código fuente. El juego más popular desarrollado con este motor es "Fortinte" (figura 17), propiedad de la propia "Epic Games" y lanzado en 2017 de manera gratuita.

Figura 11: Captura de "Fortnite" (2017)

Desde la versión "Unreal Engine 3" la potencia que ha adquirido este motor ha hecho que se pueda aplicar en diversos campos además del de los videojuegos. Se han realizado proyectos referentes a arquitectura, simuladores de conducción, entre otros **[14]**.

2.6. Unity Engine

El último motor gráfico a comentar, puesto que también es uno de los más recientes es Unity, motor con el que además se va a realizar el proyecto de esta memoria. Este motor gráfico propiedad de "Unity Technologies" fue estrenado en 2005 y sólo estaría disponible para Mac, presentando dos versiones, ambas de pago.

La versión 2.0 de este motor salió en 2007 añadiendo nuevas características como el sombreado dinámico o la generación de puntos de luz direccionales, pero no fue hasta el año 2009 cuando Unity se estableció firmemente en el mercado, gracias a su cambio de política en la que creaba una versión gratuita para todos los usuarios. Sin embargo, fue con la versión 4.0 con la que se consiguió establecer en el mercado gracias a un incremento en la calidad gráfica del motor, haciendo que algunas de las grandes desarrolladoras como "Sony", "Microsoft" y "Nintendo" mostrasen interés por "Unity", haciendo que fuese compatible con sus consolas **[15]**. Con el paso de los años ha ido recibiendo mejoras y actualizaciones, llegando al punto de poder compilar proyectos para más de 25 plataformas diferentes en la versión de 2018.

Al igual que "Unreal Engine", "Unity" ha desarrollado tanto potencial que se ha empleado en diversos campos además de los videojuegos. Se ha empleado en diversos campos como la arquitectura, industria e incluso cine.

Juegos como "Cuphead" (figura 12) o "Hearthstone" **[16]** o el corto "ADAM: The Mirror" (figura 13) **[17]** son solo unos pocos ejemplos del amplio repertorio de proyectos que se han realizado con este motor.

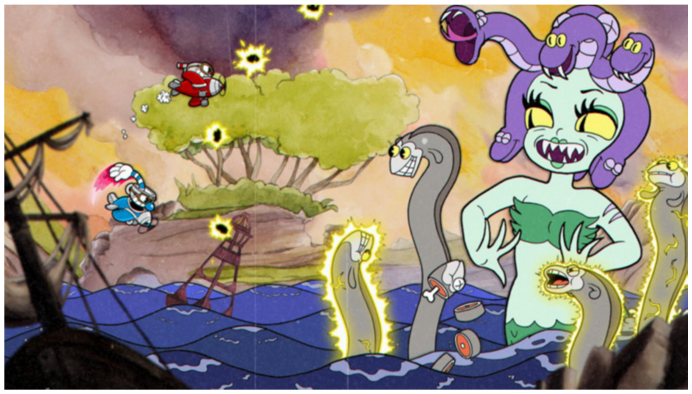

Figura 12: Captura de "Cuphead" (2017)

Figura 13: Fotograma de "Adam: The Mirror" (2017)

Al igual que "Unreal Engine", este motor gráfico tiene actualmente una versión gratuita disponible para todo aquel usuario que quiera utilizarla siempre y cuando no comercialice productos que superen los 100.000€ de beneficio **[18]**.

2.7. Género "Shoot 'em up"

El proyecto juego consistirá en la recreación de un juego estilo "shoot 'em up", el cual apareció originalmente a finales de la década de los 70 y a principios de la década de los 80, los cuales se popularizaron con el clásico "Space Invaders", desarrollado por la empresa "Taito" y estrenado en 1978 (figura 14).

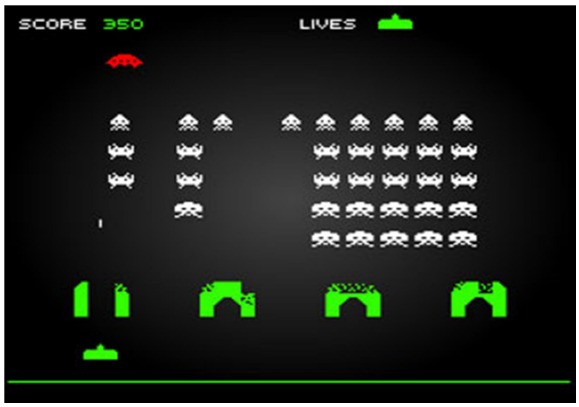

Figura 14: Captura de "Space Invaders" (1978)

Este juego, pese a no ser el primero de su estilo en salir al mercado, sí que fue el que popularizó este estilo, además de los videojuegos en general. Tras él, aparecieron múltiples juegos que fueron ampliando y variando el estilo de juego y las mecánicas. Por ejemplo, el título "Gradius" desarrollado por "Konami" en 1985 (figura 15) introdujo el sistema de ventajas en forma de objetos coleccionables **[19]**.

Figura 15: Captura de "Gradius" (1985)

Este estilo de juegos estuvo en alza durante la década de los 90, en la cual durante sus últimos años comenzó a surgir un subgénero de los "shoot 'em up" llamado "bullet hell", caracterizado por su gran dificultad debido gran número de enemigos en pantalla, haciendo que el jugador tuviese que estar atento tanto de esquivar como disparar. Este subgénero terminó de popularizarse con la llegada de juegos como "Ikaruga", desarrollado por "Treasure" en 2001 (figura 16) **[20]**.

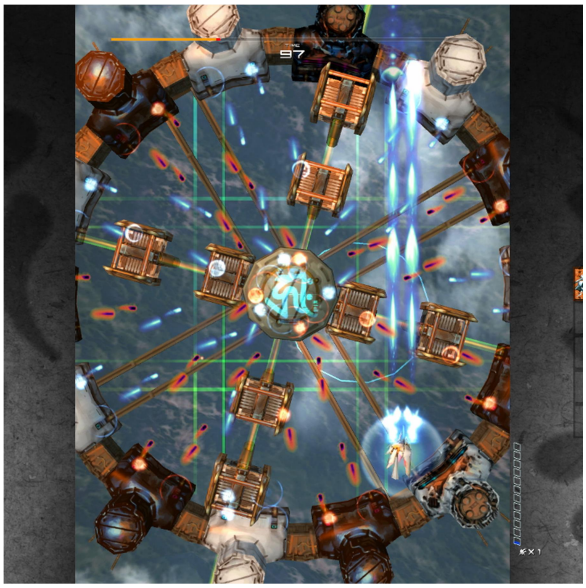

Figura 16: Captura de "Ikaruga" (2001)

Actualmente estos juegos clásicos se están adaptando y relanzando en consolas actuales. Por ejemplo, el ya comentado "Ikaruga" ha salido en diversas plataformas actuales, tales como la Play Station 4, móviles Android o PC, y se ha lanzado recientemente en la consola "Nintendo Switch" **[21]**. Paralelamente a estos relanzamientos hay algunas sagas de las cuales a día de hoy siguen saliendo juegos, como puede ser "Touhou Project", que tiene previsto el próximo juego

de la saga para verano de 2019 **[22]**. Otros juegos, en cambio, utilizan este estilo para ciertas partes del juego concretas, como es el caso de "NieR: automata", publicado en 2017 **[23]**.

Estos dos últimos juegos nombrados, "Touhou" (figura 17) y "NieR: automata" (figura 18), son los que se han tomado como inspiración para la realización y el diseño de este juego, puesto que tienen las mecánicas que se quieren recrear: juegos de acción con muchos enemigos en pantalla, los cuales el jugador deberá destruir y sortear.

Figura 17: Captura de "Touhou: Hidden Star in Four Seasons" (2017)

Figura 18: Captura de "NieR: automata" (2017)

3. Herramientas de desarrollo

Para el desarrollo de este proyecto se han empleado dos herramientas: el motor gráfico "Unity" en su versión 20.18.3.9 junto con el editor de texto "Visual Studio Community 2017" **[24]**, el cual viene con la propia instalación del motor.

Al iniciar Unity se permitirá la creación de un proyecto nuevo, el cual estará formado por una o varias escenas. Estas escenas son el componente principal del proyecto, ya que son cada una de las distintas pantallas del juego, y sobre las cuales se trabaja directamente empleando el editor y se pueden gestionarse de manera independiente.

3.1. Editor de Unity

Como puede observarse en la Figura 19, Unity presenta un editor visual para trabajar. Dicho editor está dividido en diversas secciones que el usuario puede modificar a placer arrastrándolas a la posición del editor donde se quiera colocar la ventana. La principal es la ventana de la escena, que es la que permite modificar los elementos que conforman el juego y se verán en la pantalla. Por otro lado, la pantalla "Game" permite comprobar cómo se vería la pantalla del juego final. Todos los elementos que forman la escena se pueden ver en la ventana "Hierarchy", en la cual aparecen listados todos los objetos junto con su relación. Las propiedades de cada objeto, ya sea su posición, rotación o componentes, se pueden comprobar en la ventana de nombre "Inspector". Por último, la ventana "Project" permite acceder a todo el material del proyecto: materiales, texturas, pistas de audio, scripts, etc. Se pueden crear carpetas en ella para una mejor organización del proyecto.

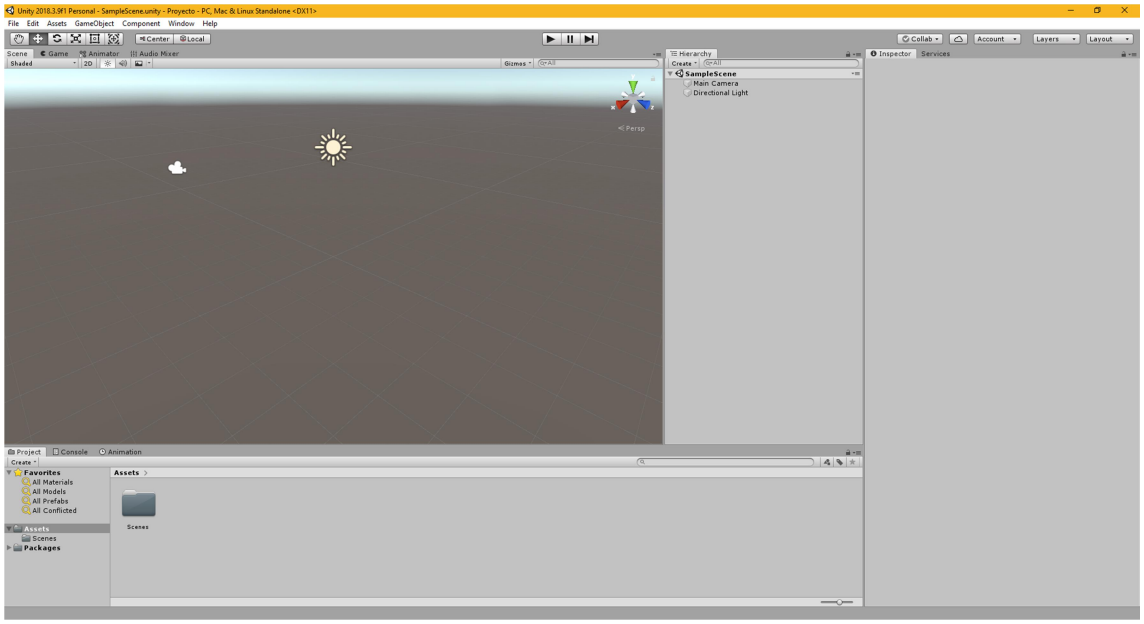

Figura 19: Editor de Unity

3.2. Objetos en Unity

Unity basa toda la creación de escenas en los objetos, llamados "Game Objects", los cuales se pueden crear desde el botón "Create" de la jerarquía o pulsando el botón derecho del ratón sobre la jerarquía y escogiendo la opción "Create". Dichos objetos pueden ser cuerpos geométricos básicos (como cubos o esferas), o bien pueden ser objetos cuya forma sea algún modelo 3D desarrollado con algún programa externo. Hay que tener en cuenta que se pueden crear también objetos vacíos o "Empty Objects", los cuales no tienen ningún componente, salvo la posición, la rotación y la escala (componente de nombre "transform"). Este objeto suele usarse para crear una combinación de elementos diferentes ("hijos") agrupados en un solo objeto ("padre"). De esta manera, se pueden modificar diversas propiedades (como la escala) solamente modificando al padre. Por otro lado, los "hijos" modificarán ciertos elementos en función del objeto padre, como por ejemplo la posición, en la que el origen de coordenadas estará en el centro del objeto padre.

Por otro lado, Unity permite hacer uso de etiquetas para crear agrupaciones de distintos elementos y distinguirlos del resto de elementos. Por ejemplo, el objeto que controla el jugador se le ha puesto la etiqueta "Player", o la cámara principal tiene la etiqueta "MainCamera". Además de las etiquetas predefinidas, el usuario puede crear las que se deseen mediante la opción "Add Tag" y añadiendo el nombre que desee utilizar.

3.2.1. Componentes de los objetos

Dado que Unity posee un motor de físicas y colisiones, es posible dar a un objeto la capacidad de verse afectado por estos factores mediante el añadido de componentes. Por ejemplo, añadir un componente "Rigidbody" a un objeto hará que éste (y sus hijos, si los tuviese) se vea afectado por el motor de físicas. Por otro lado, añadiendo un componente "Collider" se puede ajustar las dimensiones a las que se interpretará la colisión con el objeto, además de poder tener múltiples formas diferentes. Permite dos formas principales de interacción: mediante una colisión normal entre objetos o la elección del "Collider" como "Trigger". Con esta segunda opción, todo aquel objeto que colisione atravesará el objeto, pero en su lugar generará eventos que indican que se ha producido colisión.

Otros dos componentes muy importantes que poseen la mayoría de objetos son el "Mesh Filter" y el "Mesh Renderer". Estos objetos son los que permiten renderizar objetos tridimensionales en la pantalla. Cuando se añade un modelo tridimensional a la escena, el "Mesh Filter" se encarga de analizar la forma del modelo y enviárselo al "Mesh Renderer" para que muestre dicho modelo por pantalla y permitir que otros objetos puedan colisionar con él. Estos no son los únicos componentes que se pueden añadir a un objeto, pero sí son de los más utilizados.

Unity permite el guardado de configuraciones de componentes y jerarquías, agrupados en un solo objeto llamado "Prefab". Esta herramienta es muy útil para cuando se quieren tener múltiples objetos con características idénticas, ya sea por sus componentes o su estructura.

3.3. Programación en Unity

Respecto a la implementación de código, Unity utiliza principalmente C#. Existen dos maneras de crear un script: la primera, pulsando el botón derecho del ratón en cualquier carpeta de la ventana "Project" y seleccionando las opciones "Create" > "C# script" y después asignarlo al objeto de la escena que se desee. Por otro lado, la otra opción consiste en crear el script directamente en el objeto donde se quiera asignar seleccionando el botón "Add Component" en el inspector del objeto de la jerarquía y seleccionando la opción "new Script". Si en el buscador de componentes se escribe algo que no tenga relación con ningún componente, se podrá crear un script con dicho nombre. Todos los scripts deben derivar de la clase "MonoBehaviour", que es la clase principal de "Unity". La documentación de Unity se puede consultar en el enlace **[25]**.

Cuando se crea un script, en él siempre aparecen dos funciones predefinidas: la función "Start()" y la función "Update()". Ambas funciones, aunque no siempre son necesarias, son muy útiles para el correcto funcionamiento del juego.

Todo el código escrito dentro de la función "Start()" se ejecutará siempre en el momento en el que se active el script en la escena. Además de esta función, también existe la función "Awake()", la cual se ejecuta antes de la función "Start()" incluso si el script aún no está activo. Ambas funciones se ejecutarán solamente una vez en todo el tiempo de vida del script. Estas funciones son útiles para inicializar valores de las variables o referencias entre scripts.

En cambio, la función "Update()" se ejecuta después de "Start()" y su ejecución se repite una vez en cada frame mientras el juego siga en ejecución. Sin embargo, el tiempo de llamada a esta función no es regular, es decir, si un frame cuesta más de procesar que otro, la próxima llamada a "Update()" tardará más en realizarse. Esta función es útil para actualizar valores o permitir el movimiento de objetos que no tengan físicas. Por otro lado, existe también la función "FixedUpdate()", la cual realiza todos los cálculos relacionados con las físicas en cada frame. Al contrario que "Update()", el tiempo de llamada a esta función sí que es regular.

Otra herramienta muy útil que tiene Unity tiene relación con las variables públicas, a las cuales "Unity" permite darles un valor concreto desde el editor en lugar de escribirlo en el script. De esta forma, puede cambiarse el valor de ésta rápidamente y de una forma más cómoda.

4. Análisis y diseño de la solución

El juego a desarrollar tendrá un estilo “arcade”, tratando de recrear los juegos clásicos del género aprovechando las capacidades que tiene Unity. Este juego basará todo su funcionamiento a lo largo de los ejes X y Z, de manera que ningún elemento debería tener una altura distinta de 0. Por lo tanto, se puede realizar en un proyecto preconfigurado a 2D, en el cual todos sus elementos serán tridimensionales.

El primer paso para comenzar el desarrollo será obtener el paquete de modelos gratuitos diseñados por la propia “Unity” para la realización del tutorial con el que se comenzará el desarrollo. Para descargarlo, simplemente hay que acceder a la “Asset Store”, una ventana del editor que permite acceder a una tienda desde la que los usuarios pueden descargar modelados, “sprites” e incluso proyectos enteros, tanto gratuitos como de pago. Para acceder a ella simplemente hay que pulsar la tecla Control + 9 o seleccionar “Window” > “Asset Store” en la barra superior del editor.

Para poder descargar material de la “Asset Store” hay que registrarse en la web de Unity y crear una cuenta de manera gratuita. Una vez registrado el usuario puede descargar cualquier elemento que esté subido ahí. Tras buscar el paquete de material del tutorial, se pulsa en el botón rojo en el que aparece “Import”. Tras descargar los materiales, habrá que crear una nueva escena pulsando el botón derecho del ratón > “Create” > “Scene”.

Modos de juego y jugabilidad

El objetivo principal del juego consiste en la obtención de la mayor puntuación posible en la partida. Para ello, el jugador deberá ir disparando y esquivando a los enemigos, que vendrán de frente y desde los laterales. Con esta idea, se han desarrollado dos modos de juego diferentes.

En el primer modo, llamado “modo arcade”, el cual no tendrá fin, el jugador deberá sumar el máximo de puntos hasta perder todos los puntos de vida, momento en el que es derrotado. Conforme el jugador va destruyendo enemigos, éstos aparecen tanto en mayor número y frecuencia, dificultando cada vez más el juego.

En cambio, el segundo modo de juego consistirá en el avance del jugador por el nivel, y, llegado a cierto punto, dejarán de aparecer enemigos comunes y aparecerá en su lugar un enemigo de mayor tamaño, el cual no avanzará hasta el final, sino que se quedará ubicado en la parte superior de la pantalla. Desde esta posición este enemigo se irá desplazando lateralmente de manera aleatoria para incrementar su dificultad. Además, este enemigo tendrá más vida que el resto, por lo que será más difícil de vencer. En caso de que el jugador venza, aparecerá un texto de victoria y se permitirá jugar otra vez o volver al menú principal.

Para dar una mayor variedad al nivel, se han creado dos enemigos finales diferentes. La elección de cual aparecerá se toma de manera aleatoria en el momento antes de la instanciación del enemigo.

A lo largo de la partida cada vez que el jugador destruye un enemigo tendrá la posibilidad de recibir una bonificación, ya sea restaurando vida del jugador o reduciendo el tiempo de reutilización del ataque especial. Pasado un tiempo, dichos objetos comenzarán a desplazarse sobre el eje Z para que, en caso de que el jugador destruyese un enemigo en una posición inaccesible para él, pudiese tener la posibilidad de intentar recoger dicho objeto. Pasado un tiempo, si el jugador no ha recogido un objeto, éste se autodestruirá.

Interfaz

Respecto a la interfaz, se quiere desarrollar como una herramienta poco intrusiva visualmente hablando, debido a la gran cantidad de enemigos que pueden aparecer en pantalla en un momento determinado. La vida del jugador se ha ubicado en la esquina superior izquierda en color rojo, y la barra de reutilización de la evasión estará inmediatamente debajo de la de vida, pintada en amarillo. La barra que indica el tiempo de reutilización de la habilidad especial se ha decidido colocar en la esquina inferior derecha para no sobrecargar el lado izquierdo de la pantalla. Encima de esta barra habrá un texto que indicará si está disponible o si hay que esperar. Por último, la puntuación estará aislada en la esquina superior derecha para que resalte.

Todos los textos que aparezcan en el juego, tanto la interfaz como los menús, harán uso de una fuente de texto de aspecto futurista que vaya acorde a la temática del juego obtenida del siguiente enlace **[26]**.

Menú de pausa

También se implementará un menú de pausa el cual aparecerá cuando el jugador pulse la tecla "Escape". Se ha elegido esta tecla ya que para la mayoría de usuarios es intuitivo, debido a que en gran cantidad de programas y juegos permite que se detenga la ejecución o se pause el programa. Desde este menú de pausa el jugador podrá continuar la partida, ver los controles del juego, ajustar el sonido, reiniciar la partida o salir del juego al menú principal.

Naves y controles del juego

A cada uno de estos niveles se podrá acceder desde la ventana del selector de personajes, en la que se permitirá al usuario elegir entre naves con distinto comportamiento. En este proyecto se desarrollarán dos tipos de naves diferentes: una más rápida que solo podrá disparar hacia delante, y otra más lenta con la que se podrá elegir la dirección con el disparo usando el ratón. En ambos casos se disparará la munición básica con el botón izquierdo del ratón, y se realizará el ataque especial con el botón derecho. Se ha tomado esta decisión puesto que es la forma más cómoda de tener los dos botones en una posición natural. A esto se suma además el hecho de que una de las naves requiere del ratón para poder apuntar correctamente. De cada tipo de nave se han desarrollado dos modelos diferentes para dar variedad estética al juego.

Por último, respecto al movimiento del jugador, se han elegido las teclas "W, A, S, D" debido a que son las que se emplean mayormente en los videojuegos para desplazarse por el mapa. Esta elección se ve fortalecida además por el fácil acceso que se da a la mano izquierda para llegar a la barra espaciadora, la cual permitirá al jugador realizar un esquive si la pulsa cuando la barra amarilla esté completa. Para que el jugador no esté constantemente esquivando, se ha decidido poner un tiempo de reutilización de 0.75 segundos.

Ataque especial

Los ataques especiales se han desarrollado con la intención de compensar los puntos débiles y fortalecer los fuertes. Su reutilización será de 20 segundos, para que no se pueda abusar de esta herramienta con frecuencia. Para la nave de disparo frontal se ha desarrollado una habilidad que dispara desde cada lado unos disparos en forma de abanico que cubren los puntos ciegos de la nave. El ataque especial del otro tipo de nave consiste en una ráfaga de disparos que puede ser manejada con el ratón, al igual que el disparo normal.

En este selector de personajes se podrá acceder al mismo recuadro de explicación de los controles que el que aparece en el menú de pausa de la partida para poder los controles de cada tipo de nave antes de empezar a jugar.

Menú principal

Al selector de personajes se podrá acceder desde el menú principal, que será la primera pantalla que aparecerá al iniciar el juego. Este menú tendrá tres botones, cada uno con una opción diferente: jugar, ajustar el volumen o salir del juego. La idea es que estéticamente sea sencillo pero bonito visualmente. Por lo tanto se escogerá una imagen estática, que se empleará de fondo, junto con algún efecto de partículas para que no sea plano.

Música y efectos de sonido

Cada una de estas pantallas tendrá una pista de audio diferente, que podrá regularse con el menú de opciones. De manera excepcional, el segundo modo de juego tendrá dos canciones diferentes, dependiendo de si el jugador está enfrentándose al jefe o no. En caso de que el jugador complete el nivel, sonará una música de victoria y, en caso de que pierda, la música dejará de sonar. Es recomendable que si la pista de audio tiene que estar repitiéndose constantemente, su formato sea ".WAV", ya que si es ".MP3" se producirá una pausa entre repeticiones.

Además de música el juego contará con diversos efectos de sonido, los cuales podrán ser regulados al igual que la música. Los disparos y las explosiones serán los principales elementos que los tendrán.

Diseño del escenario

El escenario estará formado por una imagen plana o una textura la cual no deje rastro al conectar dos objetos con la misma textura. Esta imagen se colocará en un "quad", que es básicamente un cuadrado de una unidad de largo por una unidad de ancho, mientras que un plano mide diez unidades de largo por diez de ancho. Otra razón por la que se se ha decidido emplear un "quad" en lugar de un plano es debido a que el "quad" tiene una geometría más simple, puesto que está formado únicamente por dos triángulos al contrario que un plano, que está compuesto por doscientos. Esto hace que para elementos sencillos como la creación de fondos, sea más eficiente emplear un "quad" en temas de optimización.

Por último, se ha decidido que la configuración empleada para la cámara en este proyecto sea diferente a la explicada en el tutorial, puesto que la idea es que la cámara esté en un plano picado en lugar de estar enfocada en una vista superior. Por eso, en lugar de emplear una cámara ortográfica se empleará una perspectiva.

La principal diferencia entre una cámara perspectiva y una ortográfica es que la cámara perspectiva muestra el tamaño de los objetos acorde al mundo real, es decir, que conforme los elementos del juego vayan acercándose a la cámara, éstos incrementaran el tamaño y viceversa. Por otro lado, las cámaras ortográficas siempre muestran los objetos del mismo tamaño, independientemente de la distancia en la que se encuentren de la cámara.

Cámara

Dado que si el jugador reduce de manera considerable el tamaño de la pantalla del juego se dará cuenta de que hay un punto en el que la pantalla de juego deja de ser completamente visible, y los elementos del juego (enemigos y jugador) desaparecen de la ventana. A la hora de jugar esto puede ser un inconveniente, ya que no se sabe con seguridad las dimensiones de la pantalla que se van a emplear para jugar. Es por eso que esta funcionalidad debe de ser necesaria en el juego.

Se creará un script que permitirá el reescalado de la ventana manteniendo una relación de aspecto de 16:9, puesto que es el formato que emplean la mayoría de monitores domésticos actuales.

5. Desarrollo de la solución

5.1. Mecánicas básicas

En este apartado de la memoria se va a explicar todo el proceso de desarrollo del tutorial, en el cual se implementarán las mecánicas básicas tales como crear y permitir el movimiento de un objeto jugador, crear enemigos o sumar puntos.

5.1.1. Creación del jugador

Para empezar, hay que crear un "Game Object" que representará al jugador, y se le tiene que añadir un componente "Rigidbody" para que el objeto jugador pueda verse afectado por el motor de físicas del juego. Es importante desactivar la opción "Use gravity" en el editor, ya que si no el objeto que controla el jugador caerá libremente. Sin embargo, añadiendo solamente el "Rigidbody" no será suficiente para el funcionamiento total del jugador, así que se debe agregar un componente "Collider", para detectar colisiones. El tamaño del "Collider" será el espacio que ocupa el cuerpo del jugador en el espacio tridimensional. Es importante ponerle una etiqueta a este objeto, de nombre "Player".

En el contenido del paquete de modelos aparece un "Collider" que se ajusta a la forma de la nave. La forma correcta de añadir dicho componente es creando un "Mesh Collider" y reemplazando su forma por el modelo descargado. Como además se quiere que cuando el jugador colisiones contra algo haya una respuesta, hay que marcar las casillas "Is Convex" y "Is Trigger".

Por último, al modelo se le puede añadir un sistema de partículas que represente el motor de la nave. Para ello simplemente hay que ir a la carpeta de efectos incluida en el paquete, llamada "VFX" y arrastrar el efecto visual al objeto jugador en la jerarquía, de manera que el efecto visual quedará colocado como "hijo" del objeto jugador, que actuará de "padre".

Por último, para que el efecto quede rotado acorde al jugador se debe seleccionar cada uno de los dos componentes que forman el efecto, ir al apartado "Renderer" del sistema de partículas, y cambiar el "Render Alignment" de "View" a "Local". De esta forma rotará en función del eje de coordenadas del padre. Hecho esto, hay que rotar el componente 90º respecto al eje X y eliminar la rotación sobre el eje Y. De esta forma el efecto quedará orientado correctamente, como se puede observar en la figura 20.

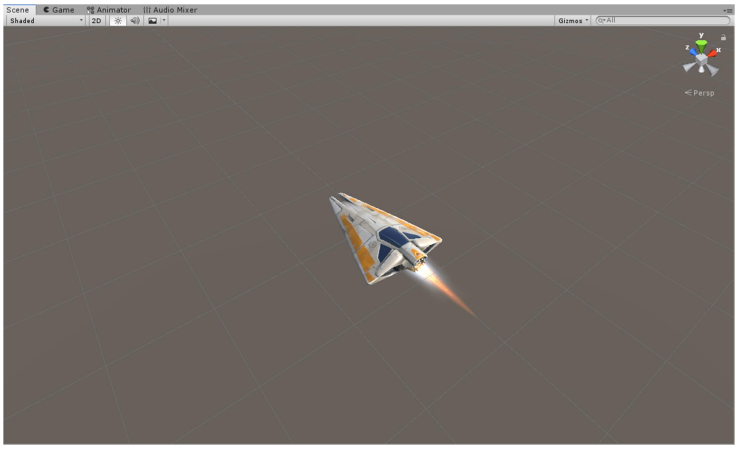

Figura 20: Efectos de partículas implementados sobre la nave jugador

Una vez realizada toda la configuración del objeto jugador, es recomendable colocarlo en la posición del origen (0, 0, 0).

5.1.2. Cámara e iluminación

Antes de configurar el movimiento del jugador, se va a configurar el posicionamiento de la cámara para poder establecer correctamente los límites del movimiento del jugador más adelante. Adicionalmente, también se configurará la iluminación del juego.

Cámara

A continuación, hay que posicionar la cámara en las coordenadas (0, 10, -5) con una vista perspectiva y un campo de visión (en inglés "Field of View") de valor 60. El campo de visión es un valor que, en función de su valor, acerca o aleja la imagen (es equivalente a hacer zoom con una cámara).

Iluminación

El siguiente elemento a tratar es el de la iluminación, cuya configuración variará dependiendo de la ambientación del mapa que se desarrolle para el juego. En el caso del tutorial, se opta por crear una ambientación relacionada con el espacio, por lo que se crean tres focos de luz tenue, que impactan en ambos lados del jugador.

Para configurar estas luces correctamente, hay que comenzar por eliminar la luz ambiental de la escena. Para ello, debe accederse a las opciones de iluminación mediante el botón "Window" de la parte superior del editor, y se selecciona la opción "Rendering" > "Lighting settings". Una vez se accede al menú hay que cambiar el color de la luz ambiental a negro. A parte de esto es recomendable cambiar el color de fondo de la cámara a negro para ajustar las luces de manera más precisa (figura 21).

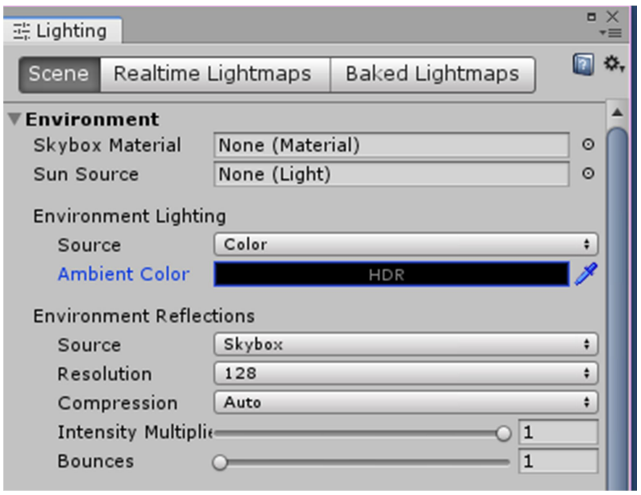

Figura 21: Configuración de la luz ambiental

A continuación, hay que configurar la iluminación principal de la escena. Ésta será una luz direccional, la cual es generada automáticamente con la creación de la escena. Hay que tener en cuenta que las luces direccionales no varían su iluminación dependiendo de la posición, sino por

su rotación. Así que, por comodidad, puede posicionarse en el origen de coordenadas (0, 0, 0) y ha de rotarse 20º en el eje X y -115º en el eje Y. Por último, se reduce la intensidad de la luz de 1 a 0.75.

Después, hay que crear otra luz direccional, que tendrá la función de luz secundaria. Por lo tanto, su intensidad será menor que la intensidad de la luz principal. Esta luz tiene un valor de 0.3 y únicamente una rotación sobre el eje Y de 125º. A esta segunda luz se le puede cambiar el color, para que en lugar de ser blanca sea un azul claro y quede más acorde a la ambientación del fondo, que como se verá más adelante presenta tonos azulados.

Por último, puede crearse una tercera luz direccional, también situada en el origen. Esta luz estará rotada -15º en el eje X y 65º en el eje Y, y tendrá una intensidad de 0.2.

Por último, para facilitar el desarrollo más adelante, es conveniente organizar los elementos en la jerarquía, para que quede lo más clara posible. Para ello se puede crear un "Empty Object", que es un "Game Object" que no tiene ningún componente a excepción del componente "Transform" (encargado de la posición, la rotación y la escala del objeto), e incluir dentro de él los focos de luz creados.

También se puede alejar la agrupación de luces de la zona del editor que se está empleando, ya que como se ha explicado anteriormente, las luces direccionales se ven afectadas sólo por la rotación, y no por la distancia.

5.1.3. Fondo del juego

En esta parte de la memoria se explicará la implementación del fondo del juego, el cual irá desplazándose por la pantalla para dar sensación de movimiento.

Creación del fondo

A continuación se va a crear el fondo del juego. Para esta parte del desarrollo se empleará la imagen incluida en el paquete de materiales del tutorial, pero se puede utilizar cualquier otra imagen para dar una ambientación totalmente diferente. Sin embargo, la creación del fondo siempre gasta la misma estructura, explicada en este apartado.

Por defecto, cuando se crea un "quad" aparece siempre posicionado en el plano (X, Y), así que para posicionarlo correctamente se debe colocar en el origen de coordenadas y rotarlo 90º en el eje X, de manera que quedará colocado en el eje (X, Z), como puede verse en la figura 22.

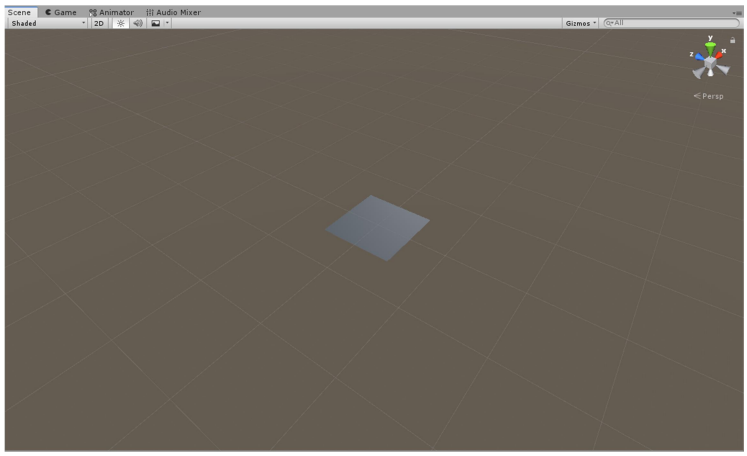

Figura 22: "Quad" orientado sobre el plano (X, Z)

Después hay que arrastrar la imagen que se quiera establecer de fondo hasta el "quad". Al hacer esto, Unity creará automáticamente un material asociado a la imagen que se quiera utilizar y la asignará al "quad".

El siguiente paso es ajustar el "quad" a la pantalla y hacer que no se solape con el jugador. Para hacer esto simplemente hay que aumentar la escala del "quad", hasta que se considere conveniente y reposicionar el quad en el espacio. Colocando el "quad" en la posición (0, -10, 12) y con una escala de (100, 55, 1) el fondo no interfiere con el jugador y cabe correctamente en la pantalla (figura 23).

Ahora, surge un problema con la iluminación del fondo, y es que queda muy oscuro respecto al jugador. Dado que el fondo varía muy poco, no compensa gastar recursos en iluminar de propio el fondo de la escena, por lo que la solución consiste en hacer que el "quad" no se vea afectado por los focos de luz de la escena. Esto se lleva a cabo seleccionando el fondo en la jerarquía y modificando el "shader" del material, aplicándole la configuración "Unlit" > "Texture". Tras hacer esto el fondo se verá bien iluminado respecto al jugador.

Por último, se puede eliminar el "Mesh Collider" del fondo, ya que es innecesario para la función que cumple el "quad" en el juego.

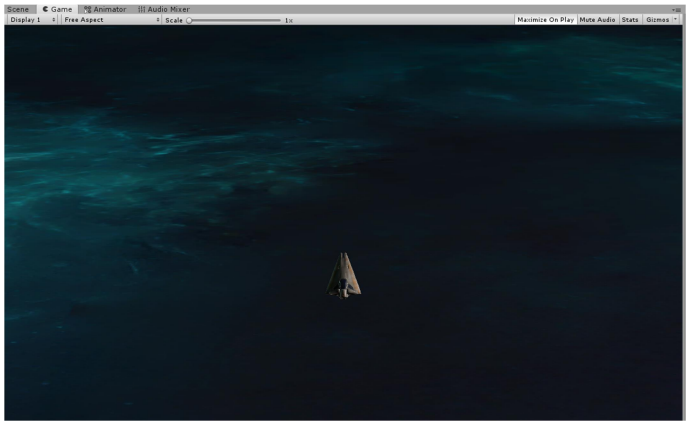

Figura 23: Vista del juego con el fondo y la iluminación configurados

Movimiento del fondo

Para dar al jugador la sensación de que el movimiento de la nave no está limitado a la ventana, se va a mover el fondo de manera cíclica para que produzca la sensación de que la nave avanza a lo largo del mapa, además de moverse en la pantalla.

Hay que comenzar por duplicar el "quad" que se ha diseñado previamente como fondo, y se colocará la copia justo detrás del fondo original, además de colocarla como "hijo" del fondo original en la jerarquía. La posición de la copia puede verse en la figura 24.

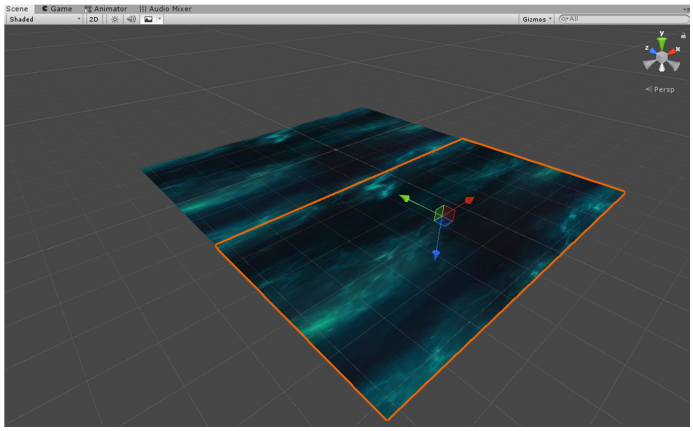

Figura 24: Posicionamiento de la copia (naranja) del fondo

El próximo paso consiste en crear un script para hacer que el fondo se mueva, llamado "BGController.cs". Dicho script irá adjunto al fondo "padre".

Este script basa su funcionamiento en el uso de la función "Mathf.Repeat(float t, float max)", la cual permite mantener el valor de la variable "t" comprendido entre 0.0 y el valor de "max", sin que nunca llegue a igualarse.

El código al comenzar su ejecución obtiene la posición inicial del fondo y después se calcula la nueva posición del fondo multiplicando los segundos de ejecución del juego por la velocidad de desplazamiento. El resultado de la nueva posición tendrá como máximo el valor de la variable establecida.

Como se observa, todos estos cálculos del posicionamiento se colocan en la función explicada anteriormente, "Mathf.Repeat()", de manera que el fondo se verá controlado por las variables definidas por el usuario.

Después de los cálculos de la posición del fondo en el espacio, los aplica de forma que la posición actual del fondo es la suma de la posición inicial y la multiplicación entre el cálculo anterior y el "Vector3.forward", que es equivalente al vector (0, 0, 1). De esta manera el fondo se moverá un máximo de distancia a una determinada velocidad, ambas decididas por el usuario.

Por último, hay que decidir los valores de las variables para que el fondo se mueva. Puesto que la nave tiene que crear sensación de avance, la velocidad a la que el fondo se desplaza debería ser negativa, puesto que tiene que ir contrario al jugador. Por lo tanto, con un valor de -1 el fondo se mueve a un ritmo acorde al del jugador.

También hay que indicar el máximo rango al que puede llegar el fondo antes de que comience a repetirse. Este valor debería ser igual a la altura del fondo, esto es, el valor Y de la escala. En nuestro caso, dicho valor es 55.

De manera adicional, en los efectos de partículas que vienen incluidos en el paquete de materiales del tutorial, hay unos efectos de partículas que se asemejan a estrellas, por lo que se pueden combinar con el fondo para que sea más llamativo visualmente. Su aplicación es muy sencilla: primero hay que añadirlas a la escena arrastrándolas. A continuación hay que reposicionarlas en el espacio en las coordenadas (0,-5,25).

El último paso consiste en modificar algunos parámetros de cada uno de los efectos de partículas que conforman el efecto visual. Para el primer efecto visual (llamado "part_starField"), hay que incrementar el valor del número máximo de partículas de 200 a 600,

además de modificar en la pestaña "Shape" del editor de partículas el valor de la escala a (50, 1, 50). Del otro sistema de partículas (llamado "part_starField_distant), hay que modificar los mismos parámetros, pero con otros valores. El número máximo de partículas será de 1000, y la escala será (10, 1, 50). Una vez completados todos estos pasos, el fondo del juego estará terminado. El resultado será similar al de la figura 25.

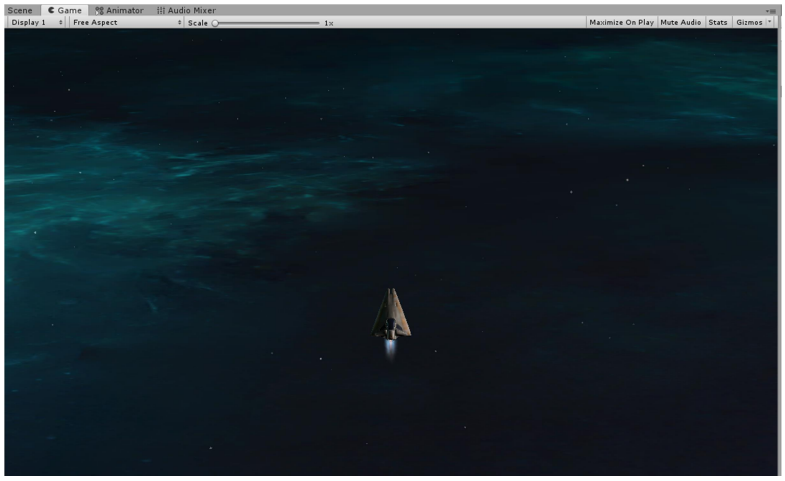

Figura 25: Escena con el fondo terminado

5.1.4. Movimiento del jugador

Para el movimiento del jugador va a emplearse un script, llamado "PlayerController.cs". Dicho script hará que el objeto jugador se mueva por la pantalla mediante físicas usando la velocidad del objeto jugador mediante la llamada "Rigidbody.velocity", que devuelve la velocidad del objeto representada en un vector. Por lo tanto, la mayoría del código de movimiento se implementarán en la función "FixedUpdate()".

"FixedUpdate()" funciona de manera que en las variables "moveX" y "moveZ" se van guardando los valores que devuelve la función "Input.GetAxis()" del eje horizontal y del vertical, respectivamente. "Input.GetAxis()" devuelve un valor comprendido entre -1 y 1 en función del movimiento del jugador. Es decir, "Input.GetAxis("Vertical")" valdrá 1 si el jugador se mueve hacia adelante, 0 si está parado o -1 si el jugador se desplaza hacia atrás. Cada uno de los valores almacenados en las dos variables ha de multiplicarse por la variable velocidad, determinada por el usuario desde el editor.

Tal como está definido el movimiento del jugador actualmente, el jugador podría mover al personaje con libertad, llegando al punto de poder salirse de la pantalla, puesto que no hay implementado ningún rango que limite el movimiento.

Para solucionar esto hay que crear en el mismo script otra clase diferente, llamada "Boundaries", la cual permitirá definir de forma manual unos valores máximos y mínimos de los cuales el jugador no podrá salir, quedando encerrado. Con la etiqueta "[System.Serializable]" puesta antes de la declaración de la clase, lo que se consigue es permitir la edición de los variables definidas en la clase desde el editor de Unity.

Por último, queda comprobar que la posición del jugador está dentro de los rangos establecidos. Para ello, se va a hacer uso de la función "Mathf.Clamp(float valor, float min, float max)". Esta función devuelve la variable "valor" si ésta está comprendida entre los valores "min", y "max".

En caso contrario, devolverá “min” o “max” dependiendo de la posición en la que se encuentre el jugador.

Por lo tanto, hay que hacer una comprobación de que los valores de la posición del jugador tanto del eje X como del eje Z están comprendidos entre los máximos y mínimos definidos anteriormente.

5.1.5. Disparar

Dado que los disparos que realizará el jugador serán distintas instanciaciones de un objeto con unas características determinadas, primero se va a explicar la configuración adecuada de dicho objeto y después se va a relatar el código necesario para que el jugador pueda disparar.

Crear disparos

Para crear el objeto que será el disparo hay que comenzar por crear un “Empty Object”, y después crear dentro de él un “quad”, que se rotará 90º para que esté sobre el plano (X, Z). Al igual que para el fondo, el “quad” será el elemento que llevará el material visual. Tras crear el “quad”, hay que eliminar el componente “Mesh Collider” puesto que se deberá ajustar el “Collider” a la forma del disparo y no al “quad”.

A continuación hay que crear el material para el disparo. Primero hay que crear un material vacío, pulsando el botón derecho del ratón y seleccionando “Create” > “Material”. Una vez creado hay que asignarle la textura a usar. Con el material seleccionado en el inspector, hay que arrastrar la textura hasta la casilla nombrada “Albedo” del material, tal como aparece en la figura 26.

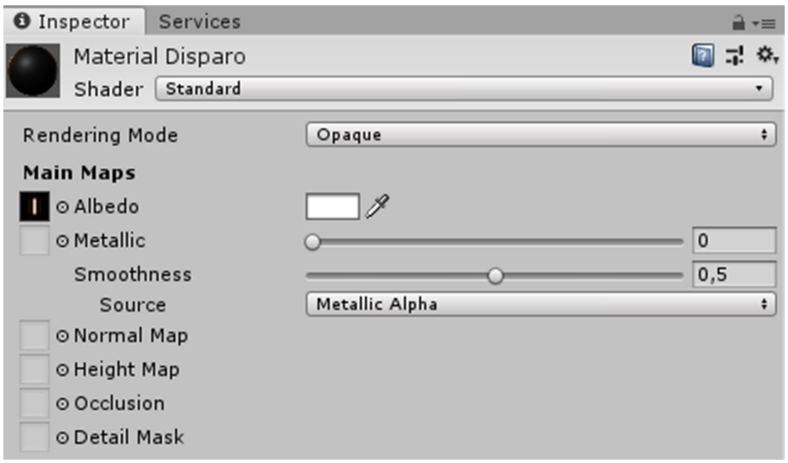

Figura 26: Asignación de la textura al material del disparo

Después hay que arrastrar el material creado al “quad”. Al hacerlo, se ve que parte del “quad” está de color negro, tal y como aparece en la figura 27. Para que únicamente se vea el disparo hay que cambiar la configuración del “Shader” a “Legacy Shaders” > “Particles” > “Additive”. Con esta configuración, el color negro pasa a tener valor 0 y deja de verse en pantalla, como puede apreciarse en la imagen 28.

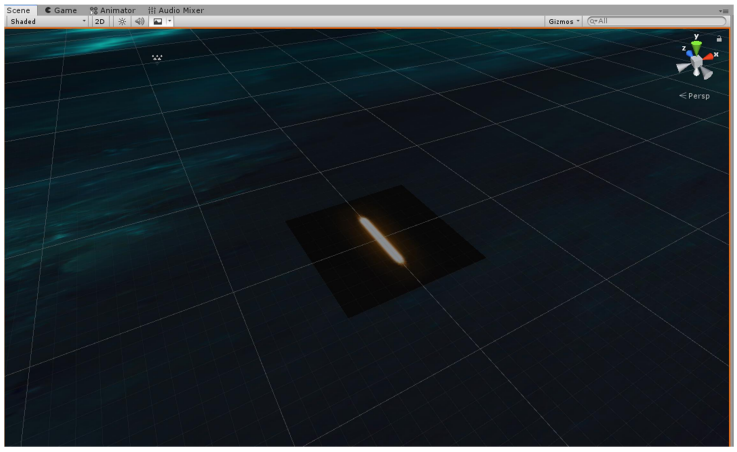

Figura 27: Material del disparo sin la configuración del "Shader" adecuada

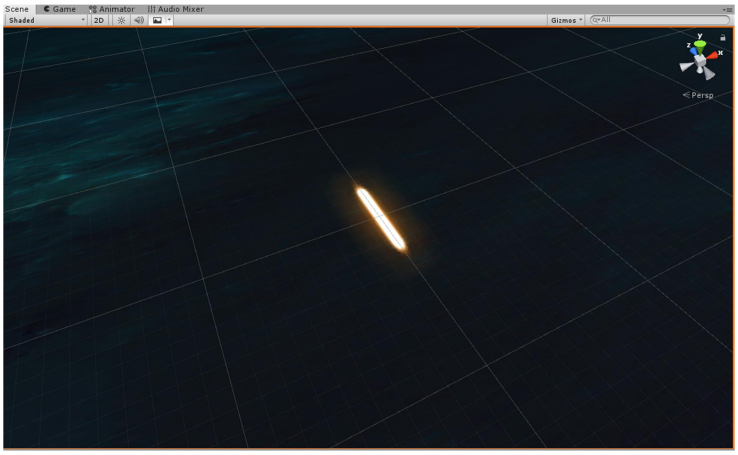

Figura 28: Material del disparo con el "Shader" adecuado

Por último, como se puede ver en la figura 29, se puede duplicar el efecto de disparo para que se generen dos disparos a la vez en lugar de uno solo, aunque en el resultado final del juego esto no tiene ningún impacto significativo.

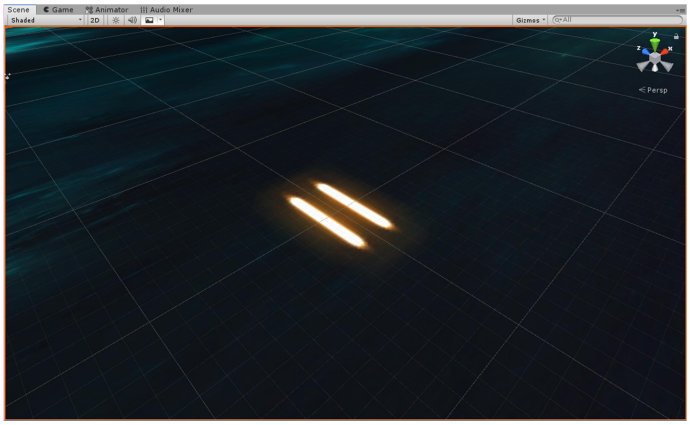

Figura 29: Resultado de duplicar el material visual del disparo del jugador.

Con esto, la parte visual del disparo está implementada. Ahora hay que añadir los componentes correspondientes. El primer elemento a añadir es un "Capsule collider", para que los disparos

tengan la capacidad de impactar con los enemigos cuando estén implementados. Tras ajustar su tamaño tomando como dirección en la que incrementa la cápsula el "eje Z", se debe marcar el "collider" como "Is Trigger", para que pueda interactuar con otros elementos del juego.

A parte hay que añadirle también un "Rigidbody", al que hay que deseleccionar la casilla "Use gravity" para que los disparos no se vean afectados por la gravedad.

Por último, hay que permitir al disparo avanzar por el espacio. Esto se hace mediante un script, llamado "Mover.cs". Este script funcionará de manera que, al iniciarse, hará que el disparo avance hacia delante a velocidad constante, determinada por el usuario. En este caso, se decidió poner una velocidad de 100.

Una vez creado y configurado el disparo, debe seleccionarse el objeto "padre" del disparo y etiquetarlo con una etiqueta creada mediante la opción "Add tag", que se llamará "bolt", y posteriormente puede guardarse como un "Prefab". Para crearlo simplemente hay que arrastrar el componente "padre" del disparo a la carpeta "Assets" del proyecto.

Permitir al jugador disparar

Una vez creado el disparo, debe implementarse la acción de disparar que realizará el jugador. A pesar de que en el tutorial añaden el código necesario en el script "PlayerController.cs", en el desarrollo del proyecto se ha decidido separarlo y escribirlo en otro script, llamado "ShootingFront.cs". De esta manera se tendrá separado el movimiento del jugador de la acción de disparar.

El primer paso consiste en crear un "Empty Object" desde el que se crearán los disparos, el cual habrá que hacer "hijo" del objeto jugador para que siempre se mueva acorde a éste. Tras crearlo, debe colocarse a una distancia conveniente de la nave, en este caso se ha optado por la posición (0, 0, 1.2), de manera que los disparos aparecerán delante de la nave.

A continuación habrá que hacer funcional el script "ShootingFront.cs", el cual basará su funcionamiento principalmente en la función "Instantiate(objeto, posición, rotación)", que permite la instanciación de objetos en la escena, con una posición y rotación determinadas.

Unity permite la creación de referencias de tipo "Game Object", de manera que se puede pasar el disparo creado anteriormente como una referencia para que sea el objeto que se genere cuando dispare el jugador.

Por otro lado, la clase "Transform", es aquella que permite saber la posición, rotación y escala de un "Game Object" dado. Dicho componente se puede pasar como tipo para crear una variable, por lo que las funciones a llamar para obtener la posición y rotación del "Game Object" que hará de cañon son menores y más legibles. Las llamadas "shotSpawn.position" y "shotSpawn.rotation" harán pues referencia a la posición y la rotación del objeto que se haya pasado como variable desde el editor, respectivamente.

Por lo tanto, la idea es instanciar el disparo en la posición del cañón con su respectiva rotación cada vez que el jugador pulse un botón. En este proyecto se decidió que el jugador disparase pulsando y manteniendo el botón izquierdo del ratón, acción que se detecta mediante el parámetro "GetMouseButton()" de la clase "Input". Este parámetro es un booleano que devuelve "true" mientras el botón se mantenga pulsado. La llamada correspondiente para detectar si se ha pulsado el botón izquierdo del ratón es "Input.GetMouseButton(0)".

En conclusión, el script funcionará de manera que si el valor de "Input.GetMouseButton(0)" es "true", se instanciará un disparo.

Por último habrá que declarar también una variable que se pueda modificar desde el editor equivalente a la cadencia de disparo, de forma que habrá cierto tiempo de espera entre disparo y disparo. Además hay que declarar otra variable que guarde el valor del tiempo en el que el jugador podrá volver a disparar.

Por lo tanto, en la condición para saber si se ha pulsado el botón izquierdo hay que comprobar también que el tiempo de espera entre disparos se ha cumplido. Cada vez que se dispara, debe calcularse el nuevo instante de tiempo desde el cual el jugador podrá disparar.

Adicionalmente se puede añadir al jugador un componente "AudioSource" con la casilla "Play On Awake" desactivada, que permita reproducir un sonido de disparo cada vez que se pulse el botón de disparar. Sin embargo, se necesita indicar la reproducción del sonido de disparo mediante código, de manera que con la función "Play()" se permite la reproducción del clip de audio que tenga el "AudioSource".

5.1.6. Límites del juego

Tal y como está el juego actualmente, cuando el jugador dispara, los disparos siguen avanzando infinitamente y acumulándose en la escena. Es por ello que se deben establecer unas fronteras en forma de caja, para que cuando los disparos y los enemigos salgan de ella, se destruyan.

Para crear estos límites hay que comenzar por crear un cubo y colocarlo en el origen de coordenadas. Después se marca la casilla "Is Trigger" de su componente "Box Collider". Ahora hay que ajustar el cubo a las dimensiones del juego.

A continuación hay que añadirle un script, llamado "DestroyByBoundary.cs", el cual funcionará de manera que cuando un disparo o un enemigo colisionen contra alguno de los límites, dichos elementos se destruyan.

Esta función detecta todos los "Colliders" dentro del "Collider" del objeto que tiene la función, y, cuando detecta que alguno de los objetos que están dentro impacta contra alguno de los bordes, se activa esta función, que en este caso destruirá el objeto que colisiona con uno de los bordes de la frontera.

Una vez terminado y añadido el script, hay que desactivar o eliminar los componentes "Mesh Renderer" y "Mesh Filter", puesto que no se necesita el cubo de forma física, sino solo un "Collider" con su forma.

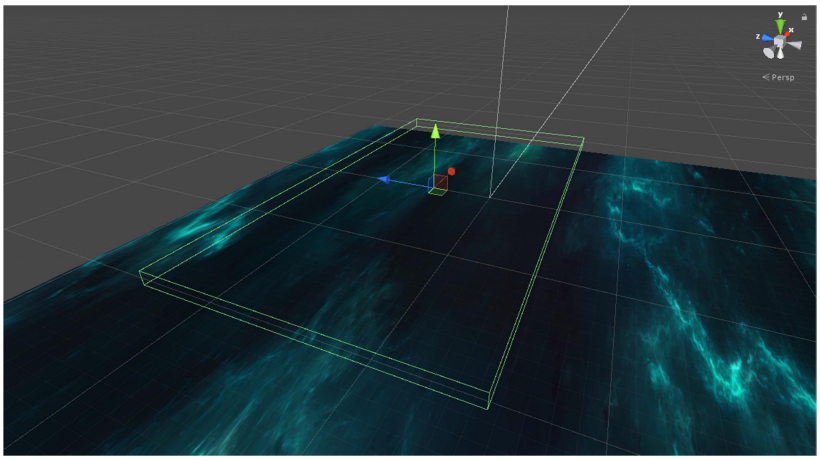

Figura 30: Límites del juego en los que se desarrollará la acción

5.1.7. Enemigos

A continuación se va a explicar la creación de los enemigos del juego, a los cuales el jugador tendrá que hacer frente. Primero se explicará la creación de los enemigos básicos, los cuales en este juego serán asteroides que simplemente avanzarán hacia delante, y después se explicará cómo desarrollar naves enemigas que disparen al jugador.

Cabe destacar que se pueden utilizar otros modelos para crear enemigos, pero para este tutorial se han empleado aquellos ya incluidos en el paquete de materiales por comodidad.

Asteroides

Los asteroides serán los enemigos básicos del juego. Éstos simplemente se desplazarán desde un extremo de la pantalla al otro, utilizando el script "Mover.cs" creado anteriormente.

El primer paso para crear los asteroides es crear un "Empty Object", el cual puede colocarse frente al jugador para hacer comprobaciones fácilmente.

A continuación, hay que coger el modelo del asteroide y colocarlo como "hijo" del objeto que se acaba de crear. Es importante que la posición del "hijo" esté en el centro del eje de coordenadas local, es decir, del "padre". Después hay que añadir en el objeto padre un componente "Rigidbody" con la casilla "Use Gravity" deseleccionada y un "Capsule Collider" acorde al tamaño del modelo usado (figura 31).

Para hacer más llamativos los asteroides, se puede programar el siguiente script, de nombre "RandomRotator.cs", que hará que en cada instancia del asteroide, éste rote sobre sí mismo de forma diferente. Este script funciona afectando directamente a la velocidad angular del objeto, que es la que indica la velocidad a la que éste rota. Para calcular la rotación aleatoria se hace uso de la propiedad "insideUnitSphere" de la clase "Random", la cual devuelve un vector tridimensional aleatorio de una esfera de radio 1. Dicho vector se multiplicará por una variable que defina la velocidad a la que rota el objeto.

Una vez terminada la configuración del aspecto físico del asteroide, se debe crear una interacción cuando un disparo colisione contra uno de estos enemigos. Actualmente, si dos de estos objetos colisinan entre sí, no se producirá ninguna reacción puesto que ambos tienen la casilla "Is Trigger" activada en sus respectivos "Collider". Por lo tanto, para permitir la reacción hay que escribir otro script, esta vez llamado "DestroyByContact.cs".

Esta vez, el funcionamiento del script se basará principalmente en la función "OnTriggerEnter(Collider other)". Dicha función se activa en el momento en el que dos "Colliders" impactan el uno con el otro. Por lo tanto, cuando un disparo o el jugador colisionen con el objeto portador del script, ambos serán destruidos mediante la función "Destroy(objeto)". Sin embargo, hay que tener en cuenta que el propio espacio por el que se mueve el jugador es un "Collider", por lo que hay que hacer que los objetos ignoren los límites creados, ya que si no se destruirán nada más instanciarse. Para solucionar esto, se hará uso de las etiquetas.

Primero hay que seleccionar el objeto "límite" en la jerarquía y después abrir el menú de etiquetas de la parte superior y seleccionando la opción "Add tag" se podrá crear una nueva etiqueta, llamada "límites" y asignarla al objeto. También se debe poner una etiqueta en todos los enemigos que se estén creando, de forma que si por algún casual dos objetos enemigos colisionen entre sí, se ignoren mutuamente y no se destruyan. Para etiquetar a los enemigos, se ha empleado la etiqueta "Enemigo".

Por último hay que añadir al código la condición de que si la etiqueta del objeto con el que colisiona el portador del script es "Límites" o "Enemigo", no haga ninguna acción.

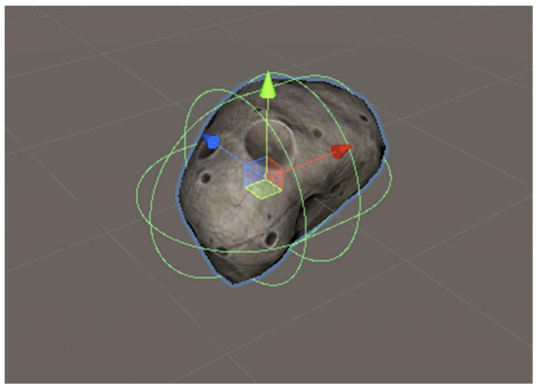

Figura 31: Asteroide junto a su "Collider"

Naves enemigas

Ahora se va a implementar una nave enemiga que avance y dispare al jugador. Lo primero es, al igual que con el asteroide, crear un "Empty Object" y colocar el modelo de la nave como un "hijo" del objeto creado. Además, también hay que añadirle un componente "Rigidbody" con la casilla "Use gravity" deseleccionar y un "Collider" ajustado al tamaño del modelo y con la casilla "Is Trigger" activada (figura 32). También se debe etiquetar la nave con la etiqueta "Enemigo".

A continuación, seleccionando el modelo de la nave (el "hijo" del objeto creado), hay que darle una rotación de 180º para que esté orientado hacia el jugador. Con el paquete de materiales del tutorial también vienen incluidos unos efectos de partículas que dan un efecto similar al motor de una nave enemiga, por lo que se puede añadir simplemente poniéndolo como hijo del "Empty object" creado y colocándolo a una distancia que se crea conveniente del modelo, en este caso, con un valor en Z de 0.25.

El siguiente paso es añadir a la nave enemiga el script "DestroyByContact.cs" creado previamente, y añadir otro script para que pueda disparar. Así pues, hay que desarrollar a continuación un script llamado "WeaponController.cs" que se encargará de realizar esta función.

Este script tiene algunas similitudes con el que se implementó para que el jugador pudiese disparar. Así pues, la instanciación de los disparos tiene la misma estructura, por lo que habrá que crear otro "Empty object" desde el que se instanciarán los disparos del enemigo. Sin embargo, la llamada a la función para disparar será diferente.

En este script se hace uso de la función "InvokeRepeating(String nombre, float t, float p)", que permite comenzar la ejecución en bucle de la función "nombre" a los "t" segundos con una pausa de "p", haciendo que la llamada a esta función solo deba hacerse una vez. Por lo tanto, debe realizarse en la función "Start()".

Por último, al igual que con los disparos del jugador, se creará una fuente de audio con un sonido de disparo, para que cada vez que se produzca un disparo, se reproduzca la pista de audio. Es necesario que la nave tenga añadido desde el editor un componente "AudioSource" junto con el archivo que se quiera reproducir, además de tener la casilla "Play On Awake" deseleccionada.

El siguiente paso consiste en crear un objeto disparo para las naves enemigas, empleando como base los del jugador, pero con unas pequeñas modificaciones en su estructura. La primera, de

manera opcional, consiste en cambiar el color para que el disparo enemigo tenga un color distinto al del jugador. Para ello, simplemente hay que duplicar el material del disparo creado anteriormente, cambiarle la textura a la copia y añadirle el nuevo material a un disparo ya creado y puesto en la escena.

Las modificaciones restantes son dos: hay que añadir a este disparo el script "DestroyByContact.cs", y además se debe cambiar la variable de velocidad del nuevo disparo del script "Mover.cs" de forma que sea negativa para que se desplace en sentido contrario puesto que si se deja el valor actual, cuando el enemigo dispare el objeto irá hacia el lado opuesto. Hechos estos pasos basta con etiquetar el nuevo disparo como "Enemigo", guardarlo como un nuevo "Prefab", y asignárselo a la nave.

Para hacer que la nave se desplace, también se hará uso del script "Mover.cs". Sin embargo, se va a desarrollar otro script, de nombre "EvasiveManeuver.cs" que permitirá a estos enemigos desplazarse lateralmente de manera aleatoria, de manera que su movimiento sea impredecible.

Cuando este script comience su ejecución guardará la velocidad actual de la nave en el eje Z y ejecutará una corrutina que hará que la nave, tras esperar un número de segundos aleatorio comprendido entre dos valores definidos en la variable "startWait", se desplace hacia uno de los lados.

El código entrará en un bucle infinito que calculará un valor aleatorio comprendido entre 1 y el valor de la variable "range", que definirá el desplazamiento lateral que realizará la nave. Este valor aleatorio además será multiplicado por el signo opuesto de la posición actual de la nave. Es decir, si la nave está en una coordenada positiva de X el valor aleatorio será multiplicado por -1 y viceversa. La obtención del signo se realiza mediante la llamada "Mathf.Sign(valor)", por lo que al poner el signo negativo antes de la llamada, se obtiene el signo opuesto. Por lo tanto, la nave enemiga nunca va a salirse de la pantalla al moverse, porque si el enemigo está en una coordenada positiva, irá a una negativa, y viceversa.

Tras esto, se vuelve a calcular de manera aleatoria la rapidez con la que va a efectuar la maniobra, y tras efectuarla vuelve a esperar un valor aleatorio de tiempo antes de realizar la siguiente.

Para que una función pueda ser iniciada en una corrutina, ha de ser tipo "IEnumerator". Además, las esperas que se realizan entre acciones se producen gracias a la orden "yield return new WaitForSeconds()", que permite pausar la ejecución de la corrutina un periodo de tiempo determinado.

```
IEnumerator Evade()
{
    yield return new WaitForSeconds(Random.Range(startWait.x, startWait.y));

    while (true)
    {
        targetManeuver = Random.Range(1, dodge) * - Mathf.Sign(transform.position.x);
        yield return new WaitForSeconds(Random.Range(maneuverTime.x, maneuverTime.y));
        targetManeuver = 0;
        yield return new WaitForSeconds(Random.Range(maneuverWait.x, maneuverWait.y));
    }
}
```

Tabla 1: Código de la función "Evade()" del script "EvasiveManeuver.cs"

Por otro lado, hay que modificar la posición de la nave para que se desplace hasta la posición indicada. Puesto que el movimiento de la nave se realiza mediante físicas, las acciones se

realizan en la función "FixedUpdate". Para ello, se hace uso de la función "Mathf.MoveTowards(float actual, float objetivo, float maxDesp)", la cual desplaza el valor de la variable "actual" hasta la posición "objetivo", con una velocidad máxima de valor "maxDesp". Posteriormente se actualiza la velocidad de la nave, y por último se comprueba que la posición no sobrepasa ninguno de los límites del mapa, de igual manera que se hace con el jugador.

Por último, con la variable "tilt" se indica cuánta inclinación tendrá la nave enemiga mientras se mueve, para hacer un movimiento más realista cuando ésta se desplaza hacia un lado.

```
    void FixedUpdate()
    {
        float newManeuver = Mathf.MoveTowards(rb.velocity.x, targetManeuver, Time.deltaTime *
smoothing);
        rb.velocity = new Vector3 (newManeuver, 0.0f, currentSpeed);
        rb.position = new Vector3
            (
                Mathf.Clamp(rb.position.x, boundary.xMin, boundary.xMax),
                0.0f,
                Mathf.Clamp(rb.position.z, boundary.zMin, boundary.zMax)
            );

        rb.rotation = Quaternion.Euler(0.0f, 0.0f, rb.velocity.x * -tilt);
    }
```

Tabla 2: Código de la función "FixedUpdate()" del script "EvasiveManeuver.cs"

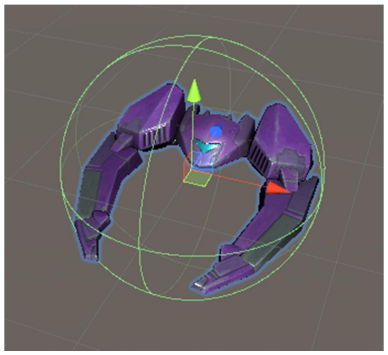

Figura 32: Nave enemiga junto a su "Collider"

Explosiones

Una vez creados los enemigos, se pueden poner efectos de partículas para que el juego quede más llamativo desde un punto de vista visual. En el paquete de materiales del tutorial vienen incluidos ya unos efectos de partículas que cumplen esta función. Así que deben hacerse unas modificaciones al código ya escrito para que sean visibles al destruir a un enemigo.

En el script "DestroyByContact.cs", simplemente hay que añadir una referencia al efecto de partículas que se quieran utilizar y añadir una función "Instantiate()", usada anteriormente para crear los disparos para que se genere el efecto de partículas cuando el objeto se destruya. Además, debe realizarse la comprobación de si el objeto tiene asociado un efecto de partículas, y en caso de que no lo tenga, que el objeto simplemente se destruya. Esta modificación es necesaria puesto que los disparos enemigos no tienen un efecto de partículas asociado.

Adicionalmente también se puede modificar el código para hacer que la nave del jugador tenga también efectos de explosión cuando ésta se destruya. Además, hay que añadir una condición de que el objeto que colisiona con el portador del script tiene la etiqueta “Player”.

Por último, se puede modificar el “Prefab” de las explosiones para añadirles sonido al instanciarse. Simplemente hay que seleccionarlas para que aparezcan los componentes en el inspector, y arrastrar la pista de audio que se quiera hacer sonar hasta el inspector, creando así un componente “AudioSource” de manera automática. Por último, hay que asegurarse de que la casilla “Play On Awake” está activada. Hay que repetir este proceso para cada efecto que se quiera usar como explosión.

Un factor a tener en cuenta es que en este momento, cada vez que se termina la animación de un efecto de explosión, éste no se destruye, sino que sigue en escena. A largo plazo esto puede traer problemas de rendimiento, ya que pueden haber demasiados elementos en la escena. Por ello, puede crearse un sencillo script que permite destruir los objetos de la escena pasado un tiempo determinado. Este script se llamará “DestroyByTime.cs”.

La función “Destroy()”, usada anteriormente, tiene un parámetro opcional que es el tiempo que ha de pasar para que se destruya un objeto. Pasando una variable como valor, el usuario puede definir el tiempo que ha de pasar para que un objeto se destruya, en este caso, las explosiones de los enemigos. Este código hay que ponerlo en los “Prefabs” de todas las explosiones. A la variable se le puede poner un valor de 2 o 3 segundos, para que el efecto pueda verse completo.

Hecho esto, hay que añadir a los enemigos los efectos a usar como explosión. Con esto ya estarían terminados los enemigos, por lo que pueden guardarse como “Prefab” tanto a la nave enemiga como al asteroide.

5.1.8. Generar enemigos

A continuación se va a desarrollar el controlador del juego, que será un “Empty Object” que tendrá aquellos scripts necesarios para el correcto funcionamiento del juego. Dicho objeto tendrá la etiqueta “Game Controller” puesta y se le adjuntará el script llamado “GameController.cs”.

En dicho script se implementará una función para que se generen enemigos en una posición aleatoria de un rango determinado, y avanzarán a lo largo de la escena.

Lo primero será declarar un array de objetos enemigos, que estará compuesto por los diferentes enemigos creados. Después se crea la función “SpawnWaves()” de tipo “IEnumerator” y así hacerla corrutina. Dicha función obtendrá una posición aleatoria entre un valor del eje X y su valor negativo y establecerá la rotación del objeto creado a la posición predeterminada mediante el método “Quaternion.identity”. Por último, se elegirá una posición aleatoria del array de enemigos aleatoria, y se instanciará con la posición y la rotación obtenidas anteriormente.

Todo este código explicado habrá que meterlo en un bucle, de forma que se generarán varios enemigos a la vez con pausas tanto antes de generar el primer enemigo, como pausas entre generación de enemigos.

Por último, hay que hacer que se generen enemigos de forma ilimitada, por lo que todo este código se puede poner en un bucle while(true). Al final de este bucle, se debe poner otra pausa, de manera que los enemigos aparecerán en oleadas de tantos enemigos como iteraciones haga el primer bucle.

5.1.9. Sumar puntos

Una vez creada la jugabilidad básica, hay que darle al jugador algún objetivo. Por eso, en este apartado se va a crear un sistema de puntuación, de manera que al destruir un enemigo se sumen puntos al marcador, y éstos se muestren por pantalla.

Hay que empezar por crear el texto en el que aparecerá la puntuación. Se creará un objeto "texto", el cual creará automáticamente un "Canvas" (que es un panel en el que aparecerán todos los elementos de texto e interfaz del juego), y un "EventSystem", que es el manejador de los objetos de interfaz. Antes de continuar es recomendable modificar la configuración del canvas, de manera que la pestaña "UI scaler mode" tenga el valor "Scale With Screen Size" con una resolución de referencia de 1920 x 1080, que es la resolución de la mayoría de monitores actuales. Esta opción permitirá al "Canvas" ajustarse al tamaño de la pantalla en lugar de tener siempre un tamaño fijo.

Una vez creado el texto y modificado el canvas, se puede ajustar el tamaño del texto y su posición en la pantalla, además de su color. Si se selecciona el objeto texto en la interfaz, aparece un recuadro que permite ajustar la posición del objeto junto con su punto de referencia si se selecciona una de las opciones mientras se mantienen las teclas "ALT" + "Mayus". En este caso se ha ajustado la puntuación a la esquina superior derecha (figura 33).

A continuación hay que ampliar el código del script "GameController.cs" para que actualice el texto junto con la puntuación. Simplemente hay que crear una variable que equivalga al texto de la puntuación, y otra variable que equivalga a la propia puntuación. Para poder referenciar a objetos de tipo "Text" y otros elementos de interfaz de usuario, es necesario importar la librería "UnityEngine.UI".

También hay que establecer el valor inicial del texto de la puntuación a 0 al empezar la partida, además crear una función que se encargue de actualizar dicho texto y otra que se encargue de sumar los puntos y añadirlos al texto.

Ahora toca modificar el script "DestroyByContact.cs", para que cada vez que se destruya un enemigo, éste sume su valor a la puntuación del jugador.

Puesto que el controlador de puntuación está en el script "GameController", se tiene que crear una variable que haga referencia a dicha clase para poder acceder a sus funciones. También hay que crear una variable que sea la puntuación por destruir el objeto.

Hay que añadir una línea antes de "Destroy(other.gameObject)" que sea la llamada a la función para aumentar la puntuación, pasando como parámetro la variable creada. Haciendo esto cada tipo de enemigo puede tener asignado un valor diferente, y sumar más o menos puntos. Por ejemplo, un asteroide puede valer 10 puntos y una nave enemiga 25.

El problema que aparece ahora es que cada instancia de los enemigos ha de ser capaz de encontrar el "GameController" al que enviarle la puntuación cuando se destruyan. Por lo tanto, nada más crear la instancia, ha de usarse la función "GameObject.FindGameObjectWithTag("Etiqueta")" para encontrar un "GameObject" con una etiqueta concreta. Puesto que al controlador se le ha puesto la etiqueta "GameController", será el objeto que encuentren los enemigos cuando se generen. A continuación, si se ha encontrado un objeto con la etiqueta, hay que obtener el componente "GameController", encargado de actualizar la puntuación. Si la búsqueda falla, puede ponerse una excepción por consola. Esta acción debe realizarse cada vez que se instancia un objeto, por lo que debe ponerse en la función "Start()".

Figura 33: Captura del juego con el indicador de puntos en pantalla

5.1.10. Final de partida

Llegados a este punto del tutorial, se han creado todas las mecánicas básicas del juego. Queda solo implementar un sistema de final de partida, para que cuando muera el jugador, se termine el juego.

Hay que empezar por crear dos cuadros de texto nuevos dentro del "Canvas": un texto que muestre un mensaje de fin de partida y otro que dé instrucciones para volver a jugar. Una vez creados, habrá que volver a editar el script "GameController.cs" para hacer que aparezcan los textos cuando la partida termine. Para ello habrá que crear dos variables de texto junto con dos variables booleanas ("gameOver" y "restart") para saber si la partida ha terminado o no.

Al iniciar la partida, ambas variables booleanas estarán a valor "false", y ambos cuadros de texto estarán vacíos. Habrá que crear también una función llamada "GameOver()", que rellenará el cuadro de texto de final de partida con la frase que quiera poner el usuario (en este caso, "Mission Failed!"), y el valor booleano de "gameOver" pasará a "true". El texto que aparecerá en pantalla se puede ver en la figura 34.

Por otro lado, al final de la espera del bucle while, se hará una comprobación de la variable "gameOver" para ver si está a "true" y, en caso de estarlo, rellenará el texto de instrucciones para volver a jugar, además de cambiar el valor de "restart" a "true" y romper la ejecución del bucle while mediante la instrucción "break".

A continuación hay que crear la función "Update()" para comprobar si el valor de "restart" es "true". En el momento en el que lo sea, en caso de que se pulse la tecla R, volverá a cargar la escena del juego con la orden "SceneManager.LoadScene("nombreEscena")". Para realizar esta función y otras relacionadas con las escenas, es necesario importar la librería "UnityEngine.SceneManagement". Después hay que asignar los cuadros de textos a las casillas correspondientes del controlador del juego.

El próximo paso es editar el script "DestroyByContact.cs", de manera que cuando se detecte la colisión con el jugador, se llame a la función "GameOver()" implementada anteriormente. Hecho esto, el tutorial estará terminado, de manera que en los siguientes apartados se explicarán todas las ampliaciones y mejoras realizadas partiendo desde este punto.

Figura 34: Pantalla de "Game Over"

5.2. Aumentar mecánicas del juego

Seguir los pasos anteriores permite crear un funcionamiento básico del juego ya funcional. Sin embargo, es ahora cuando se tiene libertad total para añadir nuevas mecánicas y funcionamientos al juego, para hacerlo más complejo, divertido y variado. Por lo que en esta parte de la memoria se irán explicando los diferentes añadidos que se le han ido haciendo al juego base para hacerlo más entretenido.

5.2.1. Aparición de enemigos por los laterales e incremento de enemigos

Para comenzar, se crearán dos puntos más de aparición de enemigos en los laterales. El primer paso consiste en crear dos variables más de tipo "Vector3", las cuales tendrán los mismos valores pero con el valor de la posición "X" cambiada de signo de manera que cada variable apunte aun extremo de la pantalla. Después habrá que crear una nueva función, llamada "SpawnWavesSide()", que se deberá ejecutar como corrutina en la función "Start()".

Dicha función tendrá una estructura similar a "SpawnWaves()", explicada anteriormente en el desarrollo del tutorial, con una diferencia respecto a la instanciación de los enemigos y el tiempo de espera. En esta función cada 3 segundos se generará un enemigo desde cada extremo de la pantalla, a una altura aleatoria comprendida entre dos rangos.

La siguiente modificación introducida consiste en la ampliación de la función "SpawnWaves()", la cual generará la primera oleada de enemigos de la partida, de manera que éstos sean todos asteroides y así el jugador pueda acostumbrarse a los controles. En este proyecto el último objeto de la lista de enemigos es un asteroide, por lo que invocándolo sólo a él, aparecerán únicamente asteroides.

Posteriormente, cuando la partida ya esté en curso, el número de enemigos por oleada se incrementará en 2 y el tiempo entre aparición de enemigos por oleada se irá reduciendo en 0.1 hasta un mínimo de 0.4 segundos por enemigo.

5.2.2. Añadido de vida al jugador y a los enemigos

Tal como está el juego implementado actualmente, todos los objetos (jugador y enemigos) se destruyen al primer contacto. Es decir, el juego actúa como si todos los elementos solo tuviesen un punto de vida. Por lo tanto, para hacer que las partidas duren más tiempo se va a implementar un sistema de vida, tanto para el jugador como para los enemigos.

Vida del jugador

El primer paso será crear un "Slider" en el "Canvas" para que se muestre de forma visual la vida actual que tiene el jugador. Para ello, primero hay que organizar los elementos de la interfaz, agrupándolos en paneles. Lo primero es crear un "Panel" dentro del "Canvas", y llamarlo "BasicInfo UI". Este panel contendrá todos los elementos de la interfaz relacionados con el juego (el texto de la puntuación, el de final de partida, etc.), por lo que una vez creado hay que insertar dichos elementos dentro del panel.

Posteriormente hay que crear otro panel, que contendrá la información de la interfaz relacionada con el usuario. Dentro de este panel se deberá crear el "Slider" la vida del jugador. A dicho "Slider" hay que eliminarle el componente "Handle Slide Area", poner el valor del "Slider" a 0 para poder ajustar el tamaño del elemento "Fill" del "Slider". Una vez ajustado, hay que cambiar el valor del "Slider" a 1 y ajustar el tamaño del elemento "Fill Area" para que se ajuste al fondo del "Slider". De forma opcional se pueden cambiar los colores tanto del fondo como de la parte que se rellena. Este panel estará desactivado siempre, por lo que la manera de activarlo es mediante una modificación en la función "Start()" del script "ShootingFront.cs".

El "Slider" ha de tener el mismo valor máximo que vida máxima tenga el jugador. Por último, se puede seleccionar la casilla "Whole numbers" para que así el "Slider" solo esté formado por valores enteros.

En el script "PlayerController.cs" se deben tener dos variables públicas que almacenen el valor de la vida máxima y la vida actual del jugador. Esta segunda variable, pese a ser pública puede ocultarse en el editor mediante el atributo "[HideInInspector]". También se debe crear una referencia al "Slider" que se empleará como barra de vida del jugador. Ahora, en la función "Start()", hay que iniciar la vida máxima del jugador como la vida actual. Por último, hay que crear una función, llamada "Hurt(int dmg)", la cual actualizará tanto el valor dela vida actual como el valor de la barra de vida, restando el daño recibido pasado como parámetro.

El último paso es establecer la cantidad de vida que quita cada enemigo al jugador cuando impactan. Para ello habrá que modificar el script "DestroyByContact.cs" añadiendo una variable que representará el daño realizado por el enemigo, y en la detección de colisión con el jugador se deberá llamar a la función "Hurt()" creada anteriormente pasando como parámetro la variable con el daño del enemigo y posteriormente se genere un efecto de partículas de explosión. Para poder tener acceso a todos los atributos y funciones públicos del script "PlayerController.cs" que lleva el jugador, hay que crear una instancia del mismo.

Después de esto debe comprobarse que la vida del jugador sea menor o igual a 0 ya que, en caso de que el jugador haya perdido toda la vida, se llegará al fin de la partida.

Vida de los enemigos

Creado el sistema de vida para el jugador, ahora falta diseñar un sistema de vida para los enemigos. Puesto que el objeto que tiene que quitar vida a los enemigos no es el jugador, sino los disparos que éste realiza, hay que añadir a los disparos un componente que sea el daño que éstos hacen sobre los enemigos mediante un script.

El siguiente paso es volver a editar el script "DestroyByContact.cs", donde se deberán crear dos variables que representen la vida máxima y actual del enemigo, tal y como se ha hecho con el jugador. El siguiente paso es crear una referencia al disparo para poder obtener el valor de la variable del daño, y así poder actualizar la vida actual del enemigo. En caso de que ésta sea menor o igual a 0, se sumará la puntuación y se destruirá el objeto enemigo.

5.2.3. Animaciones del jugador

En este apartado se van a explicar cómo se han desarrollado las animaciones para que, al igual que las naves enemigas, la nave del jugador se incline hacia un lado u otro en función del movimiento. También se explicará la implementación de una animación para que el jugador pueda esquivar y, mientras ésta dure el jugador sea invencible, de manera que se ignorarán las colisiones entre el jugador y los enemigos.

Para este apartado se hará uso del componente "Animator", que es el que permite crear y establecer relaciones entre las distintas animaciones que se creen en Unity.

Para crear una animación y un "Animator", basta con seleccionar el objeto de la jerarquía con el que se quiera hacer la animación, en este caso la nave del jugador, y en la ventana "Animation" seleccionar el botón "Create".

Inclinación durante el movimiento

Primero, se comenzará creando una animación vacía, que será la primera animación que tenga la nave en estado de reposo. Después, hay que crear otra animación para que la nave se incline hacia un lado cuando el jugador se desplace lateralmente. Para ello, simplemente hay que modificar el elemento "Transform.rotation" de la nave, el cual se añadirá a la animación mediante el botón "Add Property" y pulsando en el botón con el símbolo "+".

Cada uno de los rombos que aparecen en la línea de tiempo representa un punto clave de la animación. Así pues, hay que copiar los rombos que aparecen en el instante 0:00 y pegarlos en el instante 0:30. Esta franja de tiempo representa el tiempo que le tomará a la nave realizar la animación completa. Por último, hay que modificar la rotación de la nave sobre el eje Z en el punto copiado cambiando el valor a 30º. Esta animación se activará cuando el jugador se mueva hacia la izquierda. Los resultados de las animaciones pueden verse en las figuras 35 y 36.

Unity genera automáticamente los estados intermedios de una animación dados los puntos clave de la misma, por lo que solo hay que modificar aquellos instantes en los que la animación tiene un estado significativo.

Después hay que crear otra animación idéntica que se iniciará cuando la nave se mueva hacia la derecha. Esta animación tendrá la inclinación con un valor de -30º.

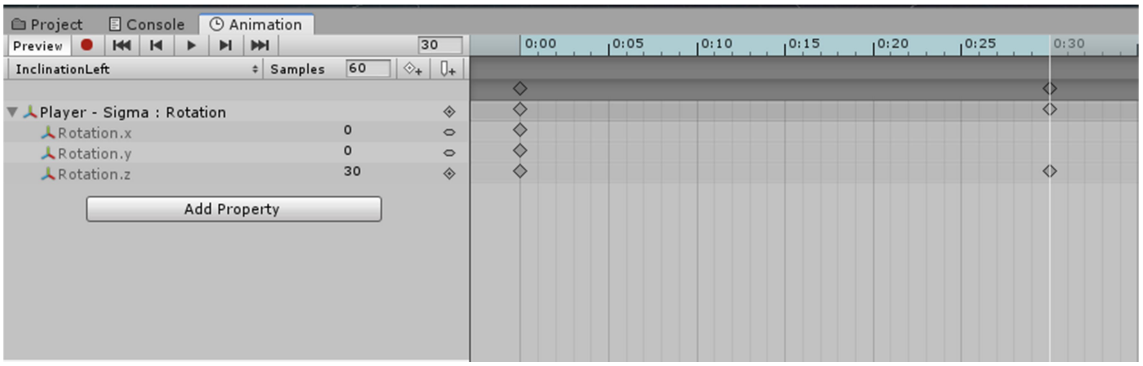

Figura 35: Estructura de la animación cuando el jugador se desplaza a la izquierda

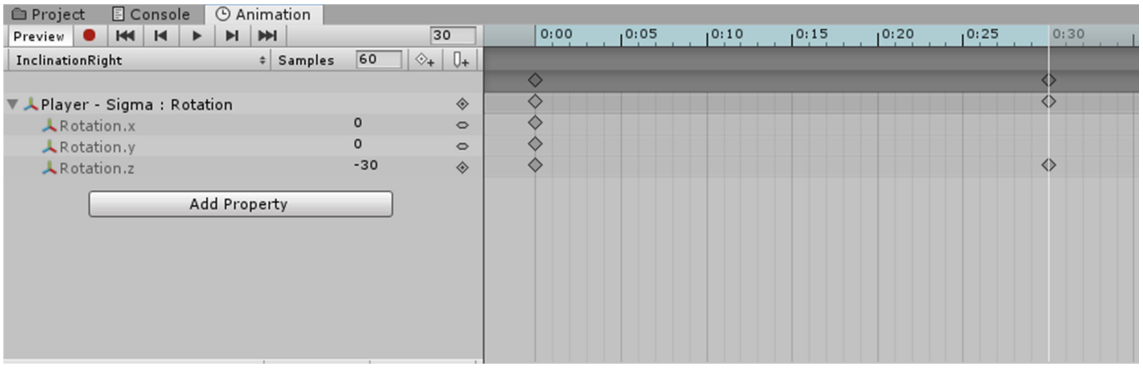

Figura 36: Estructura de la animación cuando el jugador se desplaza a la derecha

Esquivar

Hechas las animaciones mientras el jugador se mueve, ahora se va a implementar una animación de esquive que hará que mientras ésta dure el jugador sea invencible.

Primero, hay que modificar el script "PlayerController.cs" para añadir una variable pública (de nombre "isInvincible"), que indique si el jugador es invencible o no. Dicha variable tendrá un valor predefinido a "false". También deberá crearse una referencia al componente "Animator" para que el objeto jugador pueda realizar las animaciones.

Adicionalmente hay que modificar también la función "OnTriggerEnter()" del script "DestroyByContact.cs", de manera que al haber una colisión, además de comprobar que la etiqueta del objeto sea la del jugador se compruebe también que el valor de la variable "isInvincible" esté a "false". Por lo tanto, mientras la variable "isInvincible" valga "true", el jugador ignorará todo tipo de colisión.

Una vez hechas modificaciones, hay que crear las animaciones, dependiendo del lado al que se mueva el jugador, al igual que las inclinaciones. De la misma forma que para las animaciones anteriores, hay que modificar el componente "transform.rotation", de manera que a mitad de la animación esté orientado a 180º en el eje Z, y al terminar la animación la rotación sea de 360º ya que si se dejase a 0º la nave no daría una vuelta completa, sino que se volvería a colocar en la posición principal repitiendo el mismo giro. Para esta animación se ha establecido el punto intermedio en el segundo 0:14 y el final de la animación en el segundo 0:28.

Sin embargo, hay que añadir un componente más: la variable booleana creada anteriormente. Desde el instante inicial hasta el final de la animación tendrá valor a "true". El valor de esta variable puede añadirse de la misma forma que se ha añadido el componente

"transform.rotation", con la diferencia de que se activará o desactivará la casilla en los instantes de tiempo que correspondan. Es decir, en el instante 0:00 estará activada y en el instante 0:29, pasará a estar desactivada. En las figuras 37 y 38 se puede ver la estructura de la animación de la evasión hacia los distintos lados.

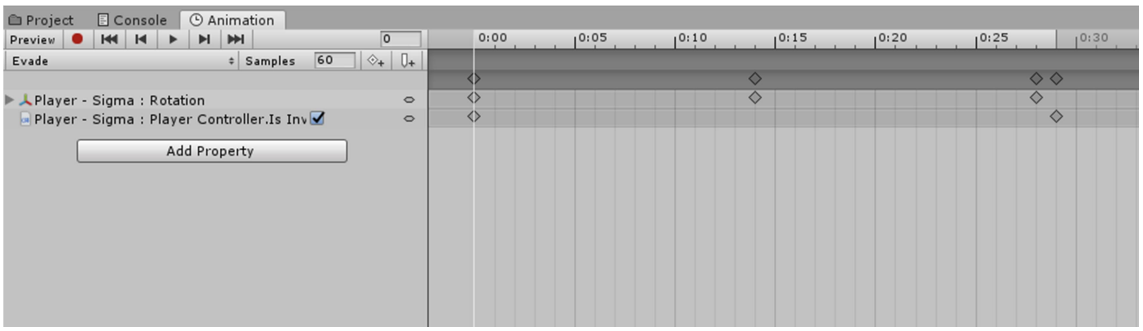

Figura 37: Animación de evasión hacia la izquierda

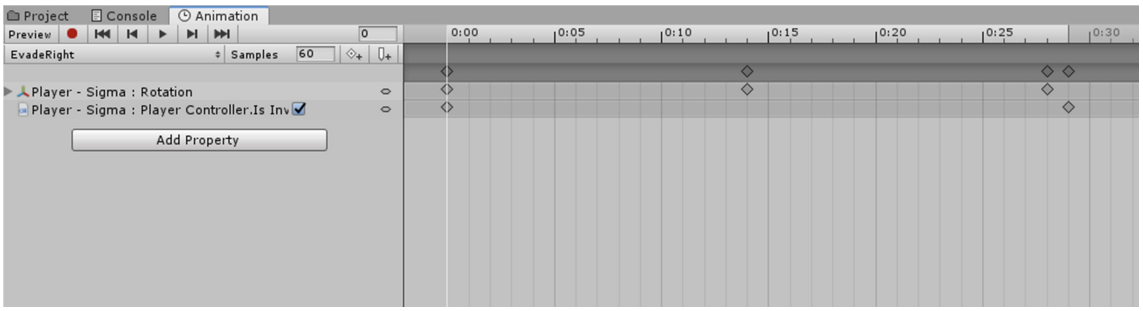

Figura 38: Animación de evasión hacia la derecha

Con esto, ambas animaciones estarán terminadas, pero ahora hay que implementarlas para que se reproduzcan a lo largo de la partida. Esta configuración se realiza en el "Animator", que es el administrador de las animaciones.

El primer paso es añadir todas las animaciones en el "Animator", seleccionar con el botón derecho la animación que equivalga a la nave en estado de reposo y elegir la opción "Set as layer default state". Haciendo esto el "Animator" sabrá que el estado por defecto será dicha animación. También se debe hacer doble "click" en cada una de las otras animaciones y comprobar que la casilla "Loop time" esté desactivada.

A continuación, hay que crear cuatro parámetros (dos booleanos y dos "Triggers") pulsando en el botón con el símbolo "+" ubicado arriba a la izquierda de la ventana del "Animator". Los dos booleanos se usarán para saber hacia qué lado se está moviendo la nave (por lo que se optó por ponerles los nombres de "InclinationLeft" e "InclinationRight"). Siempre que se esté moviendo a la izquierda o a la derecha, el booleano valdrá "true" y la animación estará activa hasta que valga "false", momento en el que volverá al estado inicial. Estas condiciones han de ponerse en las transiciones desde el estado por defecto hacia cada una de las animaciones de inclinación. Por último, hay que comprobar que las transiciones (tanto de ida como de vuelta) de la inclinación al estado inicial tienen la casilla "Has Exit Time" desactivada.

Por otro lado, los "Triggers" se usan para saber si la nave está esquivando hacia un lado u otro. En el momento en el que el "Trigger" se activa, la animación da comienzo, hasta el momento en el que se desactive, que se volverá a la animación inicial. Hay que destacar que las transiciones desde la animación inicial o de inclinación tienen la casilla "Has Exit Time" desactivada, pero las transiciones de vuelta a estos estados sí que la tienen activa.

Como puede observarse en la figura 39, que se ve el resultado de las transiciones y los parámetros

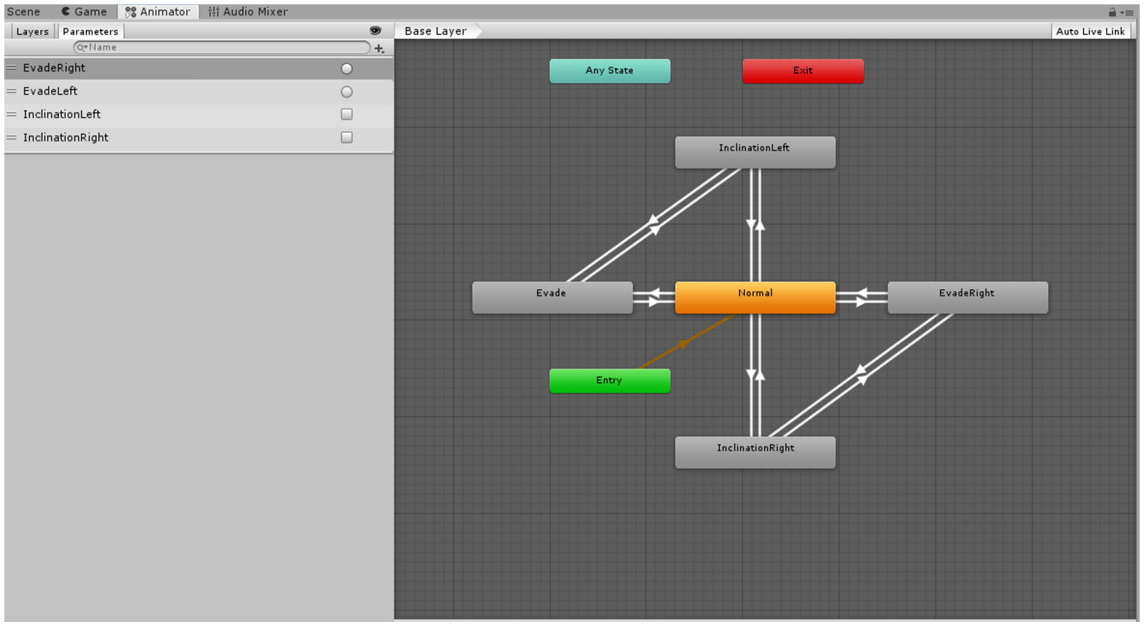

Figura 39: "Animator" de la nave del jugador junto con los parámetros

Una vez terminada esta configuración, hay que volver a hacer modificaciones en el script "PlayerController.cs" para establecer las condiciones en las que los valores de los parámetros asignados tendrán un valor u otro.

Después, en la función "Update()" hay que crear dos condiciones, de forma que dependiendo de la dirección hacia la que se esté moviendo el jugador cuando éste pulse la barra espaciadora, el valor del parámetro correspondiente sea "true" y así se inicie la animación. En caso contrario, el valor del parámetro vuelve a "false". El valor del parámetro booleano se puede modificar mediante la función "Animator.SetBool("parámetro", true)". Para ello hay que crear dos variables, una que será el tiempo máximo a esperar y otra que será el tiempo actual que queda para poder volver a esquivar.

De igual manera que se han creado condiciones para cambiar el valor de los parámetros de inclinación, aquí se activa el "Trigger" en el momento en el que el jugador pulsa la barra espaciadora, además de haber esperado el tiempo de reutilización. Se puede activar un "Trigger" mediante el uso de la función "Animator.SetTrigger("parámetro")".

Al igual que se creó un objeto "Slider" con la vida actual del jugador, se ha creado otro ubicado justo debajo de la barra de vida, que indica el tiempo restante hasta poder volver a esquivar. En el momento en el que vale 0 o menos, el jugador puede volver a hacer la animación.

5.2.4. Ataque especial

Para que el jugador tenga más mecánicas con las que jugar, se va a explicar cómo crear un ataque especial que el jugador ejecutará pulsando el botón derecho del ratón. Dicha habilidad tendrá un tiempo de reutilización, durante el cual el jugador no podrá usarla.

Para ello, hay que comenzar creando un nuevo tipo de munición, que será la munición especial. Tendrá las mismas propiedades que las balas normales de la nave, pero con la diferencia de que éstas sólo se instanciarán en el momento de emplear el ataque especial.

Para crear los disparos especiales se ha creado una copia del "Prefab" de los disparos normales, con la diferencia de que se ha eliminado uno de los haces de luz que forman la bala. El haz restante se ha centrado en el origen de coordenadas y se ha aumentado la escala a 2.

Ahora hay que hacer modificaciones al objeto jugador. Primero a los elementos que lo componen, y después al script que permite disparar al jugador, "ShootingFront.cs", para crear referencias a los cañones que dispararán el ataque especial, además de escribir el código para que se realice el ataque.

Al igual que se creó un punto de referencia para que los disparos saliesen de él, ahora se deben crear dos puntos más para que los disparos aparezcan desde ellos, el izquierdo y el derecho. Cada uno de ellos estará con una orientación inicial de 0º. Ambos estarán en la misma posición sobre el eje Z pero en valores opuestos respecto al eje X.

Dado que ahora cuando el jugador se mueve lateralmente hace que la nave se incline, el problema viene con que los cañones desde los que se disparan las balas especiales se inclinan también, haciendo que éstas salgan disparadas inclinadas, variando su altura en el eje Y en lugar de avanzar solo por el plano (X, Z). Por eso, a cada uno de estos cañones hay que añadirles un sencillo script, de nombre "LockRotation.cs" para fijar su orientación y evitar que roten junto con la nave cuando ésta se incline, ya que de lo contrario el valor de los disparos sobre el eje Y variará. Partiendo de la solución que explican en el enlace de referencia **[27]**, nada más comenzar la partida se obtiene la rotación inicial de ambos objetos, y en cada actualización de frame restablece las coordenadas iniciales de X y Z. Este script se debe poner en todos los objetos desde los que se instancian disparos, tanto los especiales como los normales.

El siguiente paso es crear un "Slider" en el "Canvas" para que se muestre de forma visual si el jugador puede usar el disparo especial o no. Dicho "Slider" se colocará como "hijo" en el mismo panel en el que se ha puesto la barra de vida del jugador. Encima de dicha barra puede crearse un cuadro de texto que indique por escrito si el ataque especial está listo o no.

Con este último elemento, la interfaz de usuario estará terminada, y el resultado se puede ver en la imagen 40.

Imagen 40: interfaz de usuario completa

Ahora hay que editar el script "ShootingFront.cs" para poder realizar el disparo con las nuevas balas creadas. Para ello, hay que comenzar por crear dos variables de tipo "Vector3" y guardar en cada una de ellas la rotación inicial de los cañones, es decir, los valores de las variables que contienen la rotación inicial del cañón izquierdo y la del cañón derecho valdrán las dos (0, 0, 0). También hay que crear una variable que sea el tiempo de reutilización y otra que indique el tiempo máximo de reutilización, además de una referencia al "Slider", a los objetos que harán de cañones y a los disparos de cada uno de estos cañones.

A continuación hay que crear una función de tipo “IEnumerator”, para poder ejecutar la función en una corrutina. Dicha función colocará ambos objetos con sus respectivas rotaciones iniciales mediante la función “Quaternion.Euler(vector3)”, que devuelve una rotación pasado un vector como parámetro.

Después, hay un bucle iterativo que, para un número determinado de disparos determinados por el usuario en una variable pública, instanciará dos disparos en direcciones opuestas y ambos cañones rotarán un número determinado de grados, en función del número de disparos indicado. Tras un tiempo de pausa entre disparo y disparo, el bucle se vuelve a repetir.

Tras haber terminado el bucle todas las iteraciones, se establece un tiempo de reutilización y se establece el tiempo de espera del jugador para poder volver a usar el ataque junto con el valor del “Slider”, que irá acorde al tiempo de espera. Por último, se restablece la rotación inicial de los cañones.

Para iniciar la ejecución de la función hay que comprobar que se ha pulsado el botón derecho del ratón, para lo que se emplea la función “GetMouseButtonDown(1)”, que devuelve “true” si dicho botón se ha pulsado. A esta condición se ha de añadir además otra que indique si el tiempo de reutilización de la habilidad es menor o igual a 0, momento en el que el jugador podrá usarla.

Si ambas condiciones se cumplen, se podrá ejecutar en una corrutina la función explicada con anterioridad. Después de haya terminado la ejecución de la función, hay que actualizar el valor de la variable del tiempo de reutilización, de manera que en cada frame el valor se irá actualizando con la función “Time.deltaTime”, valor que indica el tiempo transcurrido entre el frame anterior y el actual. Por último, hay que actualizar el valor del “Slider” mediante la comprobación de que si el valor de reutilización es menor o igual a 0, el valor del Slider será 0, y en caso contrario sea el valor actual de la variable. Debe ajustarse el valor máximo del “Slider”, que tiene que coincidir con el valor del tiempo de reutilización puesto. Puestos estos valores, el jugador podrá disparar con el botón derecho un ataque especial y saber cuándo puede reutilizarlo.

5.2.5. Creación de un segundo tipo de nave

Habiendo terminado ya el desarrollo de una nave completa, a continuación se va a explicar cómo crear un segundo tipo de nave, de manera que el jugador tendrá la opción para escoger con cuál jugar. Este tipo de nave comparte ciertas similitudes con la ya creada, pero tendrá ciertos aspectos que la harán diferente de jugar.

Para comenzar, hay que guardar todas las modificaciones y configuraciones nuevas que se le han añadido a la nave del jugador. Para guardar dichas configuraciones hay que seleccionar el objeto jugador en la jerarquía y pulsar en el botón ubicado en la parte superior del inspector de nombre “overrides” > “Apply all”. Una vez hecho esto, hay que borrar el objeto de la escena y duplicarlo. Esta copia del “Prefab” del objeto jugador creada será la base para el nuevo tipo de nave que podrá controlar el jugador.

A continuación hay que colocar en la escena el nuevo objeto que se ha creado para poder modificarlo. Puede comenzarse por cambiar el modelo visual, para que sea distinto al aspecto que tiene la primera nave. El modelo que se ha empleado aquí es el mismo modelo que el de las naves enemigas, con la diferencia de que se ha empleado el material alternativo que viene incluido en el paquete de materiales del tutorial. Como puede verse en la figura 41, tanto la nueva nave como las naves enemigas tendrán la misma forma, pero la nave del jugador será de color rojo y las enemigas de color morado.

Para cambiar el modelo basta con reemplazar el modelo del "Prefab" con el que se quiera usar. Una vez esto hecho, hay que reposicionar los puntos de aparición de los disparos. Esta nave tendrá un punto de disparo tanto para los disparos normales como para el ataque especial. En función del momento, rotará a velocidades diferentes.

Esta nave, al contrario que la anterior, solo necesitará un objeto que haga de cañón, el cual será empleado tanto por el disparo normal como por el especial. Por lo tanto, se pueden borrar de aquí los dos "GameObjects" adicionales que tenía la otra nave.

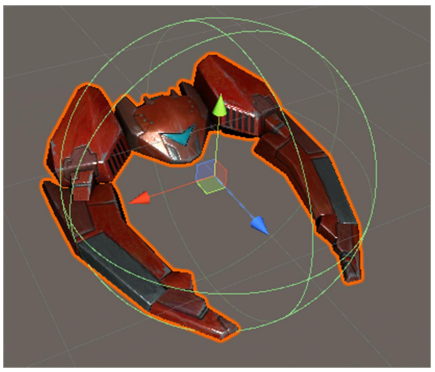

Figura 41: Segundo modelo de nave jugador junto a su "Prefab"

El siguiente paso es permitir a la nave disparar. Para ello, se creará el script "ShootRotator.cs", el cual tiene ciertos elementos idénticos que el script "ShootingFront.cs", tales como la función "Update()", por lo que se puede copiar literal de un script a otro. También puede copiarse la función "SpecialWeaponShot()" para emplearla como base pero, como se ha dicho anteriormente, el contenido de esta función será diferente al actual.

Para esta nave lo que se quiere implementar es una gran cantidad de disparos que vayan en la dirección en la que apunte el jugador. Por ello, primero se modificará el valor de velocidad de movimiento del cañón a la indicada por el usuario desde el editor y después el bucle instanciará un disparo por iteración junto con una breve pausa entre instanciaciones.

También es necesario duplicar la interfaz de usuario creada para la otra nave, y asignar la copia a la nueva nave mediante la referencia explicada anteriormente. Esta interfaz también estará desactivada por defecto, y se activará cuando inicie la partida y se haya elegido esta nave en concreto. Esta es la opción más cómoda ya que se pueden realizar modificaciones a cada tipo de nave y poder cambiar los valores establecidos sin afectar al resto de elementos.

El último paso que queda para terminar esta nave es permitir el apuntado con el ratón al jugador. Para implementar esta mecánica se ha hecho uso del script encontrado en **[28]**, que permite la rotación de un objeto a lo largo del eje Y a una velocidad determinada. En este proyecto, a este script se le ha puesto el nombre de "RotateCannon.cs" y se deberá añadir al objeto que hace de cañón, para que éste gire acorde al movimiento que haga el jugador con el ratón.

El primer paso es crear un plano con la normal sobre el eje Y (0, 1, 0) que intersecte con la posición del objeto portador del script. A continuación se crea un rayo desde la posición del cursor con la función "Camera.ScreenPointToRay", para después comprobar el punto en el que el rayo colisiona con el plano creado. La posición del ratón en la pantalla puede obtenerse mediante la función "Input.mousePosition".

El siguiente paso consiste en comprobar la intersección entre el rayo creado con el plano a una distancia determinada (en este caso 0, o lo que es lo mismo, con la pantalla de la cámara). En caso de que el rayo sea paralelo, devolverá "false". En caso de cumplirse, se guarda el punto de intersección en una variable. En caso de que el rayo colisione con el plano a una distancia

determinada, se calcula dicho punto pasando la distancia como parámetro con la función "GetPoint(distancia)".

Después se calcula la rotación a aplicar al objeto como la diferencia entre el punto obtenido anteriormente con la posición del portador del script mediante la función "LookRotation()".

Por último, se aplica la rotación mediante la función "Quaternion.Slerp()", que permite al objeto transitar desde su rotación actual a la nueva rotación a una velocidad concreta. Toda esta explicación se puede ver reflejada en la tabla 3.

```
public class RotateCannon : MonoBehaviour
{
    public float speed;

    void FixedUpdate()
    {
        Plane playerPlane = new Plane(Vector3.up, transform.position);

        Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

        float hitdist = 0.0f;

        if (playerPlane.Raycast(ray, out hitdist))
        {
            Vector3 targetPoint = ray.GetPoint(hitdist);

            Quaternion targetRotation = Quaternion.LookRotation(targetPoint -
transform.position);

            transform.rotation = Quaternion.Slerp(transform.rotation, targetRotation, speed *
Time.deltaTime);
        }
    }
}
```

Tabla 3: Script "RotateCannon.cs"

Por último, hay que modificar el script "ShootingRotator.cs" para hacer que la velocidad de rotación cambie dependiendo de si se está usando la habilidad especial o no. Hay que comenzar por crear una referencia al script, y dos variables: una que guarde la velocidad de rotación inicial y otra que valga la velocidad de rotación cuando se use el ataque especial.

Al iniciar la ejecución, en la función "Start()", hay que guardar en una variable el valor inicial de la velocidad de rotación que tiene el cañón. Por otro lado, al iniciar la función "SpecialWeaponShot()" la rotación del cañón cambia su valor por el de la otra variable. Al final de la ejecución de la misma, la velocidad de rotación vuelve a su valor inicial.

5.2.6. Implementación del menú de pausa

En este apartado se va a explicar la implementación de un menú de pausa, desde el cual el jugador podrá continuar el juego, reiniciar la partida, salir o abrir el menú de opciones, que se desarrollará más adelante. El proceso de implementación se ha basado en el vídeo "PAUSE MENU in Unity" **[29]**.

Para empezar, hay que crear un nuevo panel en el "Canvas", que contendrá todos los botones del menú. Dicho panel tendrá un color oscuro, pero no tendrá la transparencia al máximo, de manera que en el fondo se podrá ver el juego pausado.

A continuación hay que crear dentro del panel un botón. Tras ajustar el tamaño de dicho botón, hay que cambiar el color del fondo del botón a uno más oscuro en el componente "Image (Script)". También se puede modificar el texto del botón cambiando el contenido del cuadro de texto, además de cambiar el color o tamaño de la fuente.

Después, con el botón seleccionado, hay que modificar el valor de transparencia en el apartado "Normal Color" para que tenga el valor mínimo. En el apartado "Highlighted Color" hay que modificar el valor de la transparencia para que sea un poco visible, y en el apartado "Pressed Color" hay que aumentar aún más el valor de la transparencia, sin ponerle el máximo valor. Por último, hay que cambiar el valor de la pestaña "Navigation" de "Automatic" a "None" para que no se quede seleccionado permanentemente.

Con estos ajustes el fondo del botón se verá o no dependiendo de si se coloca el cursor sobre el ratón o si se pulsa sobre el botón. En caso de que no se hagan ninguna de las dos acciones, el fondo no será visible.

Una vez configurado el botón, puede copiarse tantas veces como botones se necesiten. Para este proyecto se han creado cinco botones, uno para continuar, otro con los controles, un tercero para reiniciar la partida, un cuarto para abrir el menú de opciones y el último para salir de la partida. Estarán posicionados tal y como se puede ver en la figura 42.

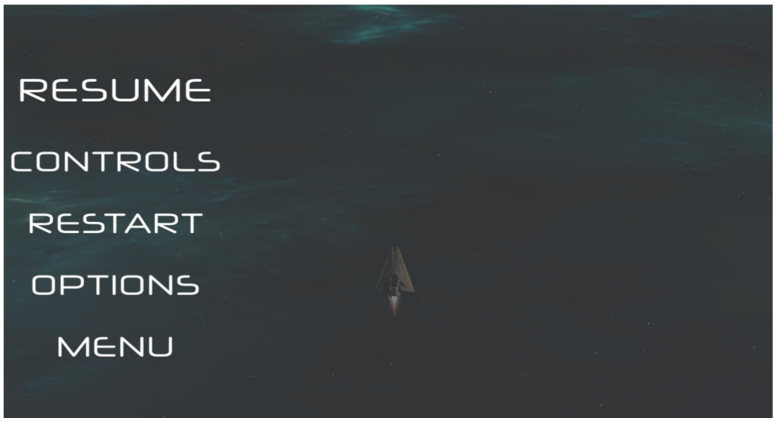

Figura 42: menú de pausa

A continuación hay que crear un nuevo panel que será el menú de opciones, el cual contendrá tres "Sliders" para poder ajustar el volumen de la música (figura 43). Aunque dicha implementación se explicará más tarde, la parte visual puede crearse ahora y así configurar correctamente el sistema de menús. El botón de opciones deberá realizar dos acciones: desactivar el menú de pausa de la escena y activar el menú de opciones de manera que quede visible para el jugador.

Para ello, habrá que crear dos acciones en el recuadro "On Click()". La primera acción se implementará arrastrando el panel desde la jerarquía hasta el recuadro correspondiente y seleccionar la función "GameObject" > "SetActive()" y desmarcar la casilla. El proceso para activar el panel de opciones es el mismo, pero marcando la casilla que aparece al lado del objeto.

Por último, en el panel de opciones deberá añadirse un botón para volver al menú de pausa. Por lo tanto, su configuración será la contraria al botón de opciones del menú.

Figura 43: Menú de opciones

También se pueden crear dos paneles adicionales, uno por tipo de nave, con las instrucciones de los controles del juego. Aunque el correcto funcionamiento se explicará más adelante, se pueden configurar algunos aspectos del botón que lleva a los controles. Este botón, al igual que el de las opciones, deberá ocultar el menú de pausa de la misma forma que se ha configurado el otro botón. Sin embargo, la activación del panel con las instrucciones correctas se explicará más adelante.

Ambos paneles deberán tener también un botón de retorno al menú de pausa, que tendrá una configuración similar al del panel de opciones. En la figura 44 se puede ver el panel con los controles de la nave de disparo frontal.

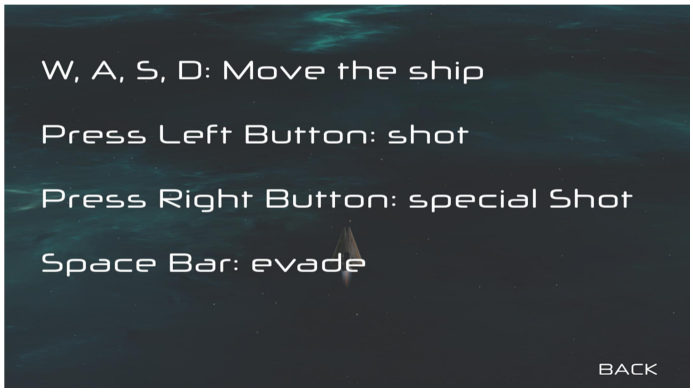

Figura 44: Panel de controles del juego de la nave de disparo frontal

Una vez creados estos cuatro paneles, deben desactivarse de la escena y crear un script llamado "PauseMenu.cs", el cual se deberá añadir al "Canvas". Este script permitirá la activación y desactivación del menú de pausa, así como la detención del juego en el instante en el que se abra el menú.

En dicho script habrá que comenzar por crear una variable booleana iniciada a "false", la cual indicará si el juego está pausado o no. También se deberán crear dos referencias para los menús de opciones y de pausa y una tercera para la interfaz de usuario. Dichas referencias se usarán para activar y desactivar los distintos menús en función de donde se encuentre el jugador. Dicho

script comprobará en cada frame si se ha pulsado la tecla escape y, en el momento que se pulse, y se comprobará el valor de la variable booleana creada. Si ésta vale “true”, querrá decir que el juego está actualmente pausado, y ejecutará la función “Resume()”. En caso contrario, se ejecutará la función “Pause()”.

La función “Pause()” se ejecutará cuando se vaya a pausar el juego. Así pues, se desactivará la interfaz del juego (tanto la puntuación como la información del jugador) y se activará el menú de pausa para que sea visible. También se cambiará el valor de la variable booleana a “true” y se detendrá el juego en el frame que se pulsó la tecla poniendo a 0 el valor del atributo “Time.timeScale”, que modifica la velocidad a la que transcurre el tiempo en el juego.

Para ocultar la interfaz del jugador hay que crear dos referencias y dos variables booleanas para cada una de las interfaces. Dichas variables estarán inicializadas a “false”, y en la función “Pause()” se comprobará cuál de las dos interfaces está activa mediante el uso del parámetro “GameObject.activeSelf”, que devolverá “true” si el “GameObject” está activo. Dependiendo de cuál de las dos interfaces esté activa, el valor de las variables booleanas cambiará o no.

Por otro lado, la función “Resume()” será contraria a “Pause()”, de manera que desactivará el panel del menú de pausa y el de opciones, activará la interfaz de usuario y establecerá el valor de “Time.timeScale” a 1 para que el tiempo transcurra de manera normal. En esta función se comprobará además cuál de las dos variables booleanas está activa, de manera que se volverá a activar la interfaz correspondiente y volviendo a colocar el valor de la variable a “false”.

La función que activará el botón de reiniciar partida será “Restart()”. Dicha función establece el valor de “Time.timeScale” a 1, pone el valor de la variable “GameIsPaused” a false y carga la escena que se está ejecutando.

Por último, se ha implementado una función que permite salir del juego y volver al menú principal. Dicha función cargará la escena correspondiente al menú principal. Sin embargo, de momento esta función no funcionará correctamente ya que no hay creado una escena con el menú al que volver.

Tras colocar los objetos de la jerarquía en las referencias correspondientes en el editor, hay que asignar a cada botón la acción implementada mediante script. Para ello simplemente hay que seleccionar uno de los botones, y en el recuadro llamado “On click()” hay que pulsar el símbolo “+”. Después hay que seleccionar el “Canvas” y arrastrarlo al recuadro debajo de la pestaña “Runtime Only”.

Después, hay que pulsar en la pestaña “No function”, y elegir la función del script “PauseMenu.cs” que corresponda al botón. Hay que repetir esta acción con cada uno de los botones implementados.

Una vez creado todo el menú de pausa, puede crearse una animación (figura 45) para suavizar el oscurecimiento de la pantalla. Al igual que para las animaciones del jugador, hay que comenzar por crear un “Animator” para el panel que contenga el menú de pausa de la misma forma que se explicó anteriormente. En este componente se debe cambiar la configuración de la pestaña “Update Mode” a la opción “Unscale Time”, para que no se vea afectada por las modificaciones del valor de “Time.timeScale”.

La animación a implementar simplemente consiste en el oscurecimiento de la pantalla hasta el valor que se haya definido previamente. Para implementarla simplemente hay que colocarse en el segundo 0:30 y reducir la opacidad a 0, de manera que no se verá la pantalla oscurecida.

El siguiente paso consiste en invertir las posiciones de los cuadrados, de forma que la imagen pasará de estar clara a oscurecida. Y, por último, hay que seleccionar la animación y deseleccionar la casilla “Loop Time”, para que solo se reproduzca la animación una vez.

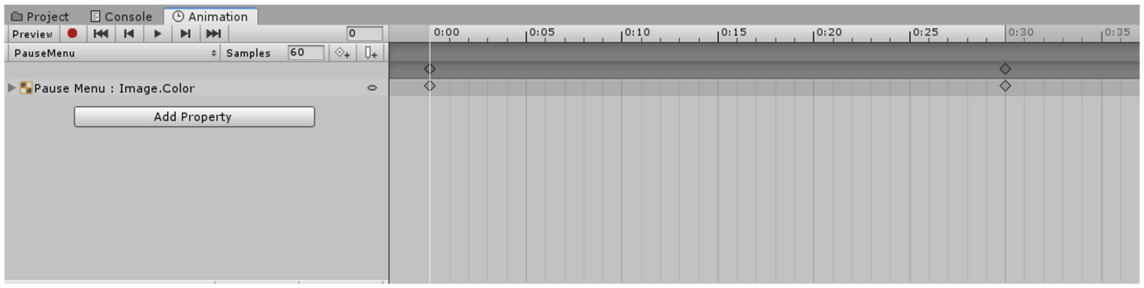

Figura 45: Animación del menú de pausa

Por último, hay que modificar los scripts "ShootingFront.cs" y "ShootingRotator.cs" para que no se ejecuten ciertas acciones mientras el juego está pausado. Dicha modificación consiste en colocar todo el código de la función "Update()" dentro de una condición, que compruebe que el juego está pausado mediante la variable booleana empleada en el script "PauseMenu.cs".

5.2.7. Ajustar dimensiones de la cámara

En este apartado de la memoria se va a redactar cómo permitir el redimensionamiento de la pantalla del juego. Para ello habrá que utilizar dos elementos: una cámara auxiliar y un script, de nombre "CameraController.cs". Todo el proceso para solucionar este problema aparece en el enlace [30].

El primer paso consiste en crear el script nombrado anteriormente, el cual habrá que añadir a la cámara principal. Dicho script calculará, partiendo de los valores pasados como parámetros para el cálculo de la relación de aspecto, la relación entre las dimensiones actuales de la ventana y la relación de aspecto. En caso de ser inferior a 1, se añadirán franjas negras en la parte superior e inferior de la pantalla (con el constructor "Rect"), además de reducir el tamaño de la ventana de juego, pero conservándola por completo.

Por otro lado, si la relación es superior a 1, se añadirán franjas negras a los laterales de la ventana, conservando la resolución pasada como parámetro.

Aunque en la fuente original realizan el cálculo de la relación al inicio al colocarlo en la función "Start()", aquí se ha decidido colocarlo en la función "Update()" para que, en caso de redimensionar la pantalla durante la ejecución del juego, ésta pueda ajustarse a la pantalla correctamente en cualquier momento.

Por último, hay que crear una cámara secundaria para que muestre el color de las franjas cuando éstas aparezcan. En cuanto a la posición, para que la cámara no pueda interferir con el juego, se ha posicionado a una altura de 15 unidades en el eje Y con una rotación de -90º para que mire hacia arriba. Por otro lado, su configuración consiste en hacer que muestre el color del fondo negro y, la modificación más importante: cambiar el valor de profundidad a -2, ya que de otra forma se podría solapar la imagen de la cámara principal con la cámara auxiliar.

5.2.8. Crear objetos “bonificadores”

En este apartado se va a explicar cómo crear e implementar un sistema de objetos, los cuales el jugador podrá coger y obtener beneficios de ellos (figura 46). Se explicará cómo crear un objeto que permita al jugador recuperar vida (en caso de que le faltase) y un segundo objeto que reducirá el tiempo de reutilización del ataque especial.

Hay que comenzar por crear un cubo, el cual tendrá la escala reducida a 0.25 en los tres ejes. A dicho cubo habrá que añadirle además un componente “Rigidbody” con la casilla “Use Gravity deseleccionada” y marcar su “Collider” como “IsTrigger”.

El siguiente paso consiste en crear una nueva etiqueta para diferenciar estos objetos del resto de elementos de la escena. En este caso se ha usado la etiqueta “Drop”.

Después, hay que añadirle al objeto el script “DestroyByTime.cs”, para que, si pasa cierto tiempo y el jugador no lo ha cogido, desaparezca de la escena.

A continuación, hay que crear dos scripts sencillos para implementar las funcionalidades del objeto. Dos de ellos (“PickDrop.cs” y “MoveDown.cs”), serán comunes a todos los objetos que se quieran implementar. El primero, “PickDrop.cs”, tiene una función que hace que solo se destruya si el objeto con el que colisiona tiene la etiqueta “Player”.

El segundo, llamado “MoveDown.cs”, es similar al script “Mover.cs” pero con la diferencia de que aquí el movimiento se produce en una corrutina, de manera que el objeto se quedará quieto en su sitio un tiempo determinado y pasado ese tiempo comenzará a avanzar a velocidad constante.

Por último, habrá que implementar un script que permita al jugador recuperar vida, llamado “PickHealth.cs”, el cual llevarán los objetos que permitan curar, y otros dos script que permita reducir el tiempo de reutilización del ataque especial, el cual llevarán adjunto el “bonificador” correspondiente.

Por último, habrá que implementar los distintos scripts con las ventajas que se quieran dar al jugador. Primero se explicará cómo implementar un script para que el jugador recupere vida, llamado “PickHealth.cs”. Este script comprueba si el objeto que colisiona tiene la etiqueta “Player”, le suma la vida que se haya indicado que recupera el objeto. Posteriormente comprueba si la vida del jugador es superior al máximo, en cuyo caso se reduce el valor de la vida a dicho máximo, y en caso contrario deja la vida con el valor actual.

El otro tipo de bonificador consistirá en una reducción del tiempo de reutilización del ataque especial. Por comodidad, se han implementado dos scripts diferentes, uno por tipo de ataque especial. Esto permite una mejor gestión de los errores de excepción por ausencia de objetos, ya que si se juega con una nave, la otra estará desactivada, junto con todos sus elementos de la interfaz. Además, permite que cuando un jugador coge el objeto, la bonificación que obtenga varíe en función de la nave que esté usando. Ambos scripts funcionan igual, con la única diferencia en los valores pasados a las variables desde el editor. Dichos scripts se llaman “PickCDRFront.cs” y “PickCDRCannonRotator.cs”.

En ambos scripts, primero se crea una referencia a cada uno de los scripts que permiten disparar. En caso de que dicha referencia no obtenga un valor, el script termina. En caso contrario, accede a la variable que contiene el tiempo para reutilizar el ataque especial y reducir su valor una cantidad de tiempo establecida en una variable.

Una vez creados ambos objetos, éstos han de guardarse como “Prefab”. El último paso que hay que realizar ahora es modificar el script “DestroyByContact.cs” para que, cuando se destruya un enemigo, aparezca en su posición uno de estos objetos con cierta probabilidad.

En la función “OnTriggerEnter”, después de que se destruya el objeto enemigo, hay que generar un número aleatorio entre 0 y 1 mediante la función “Random.Range()”. En caso de que dicho número sea menor a 0.2, se volverá a calcular un segundo valor aleatorio comprendido entre 0 y 0.7. Si este segundo valor aleatorio está comprendido entre 0 y 0.1, el objeto que aparecerá será un objeto que recupera salud. En caso de que el resultado esté entre 0.2 y 0.3, se instanciará un objeto de reducción de tiempo de reutilización de la habilidad.

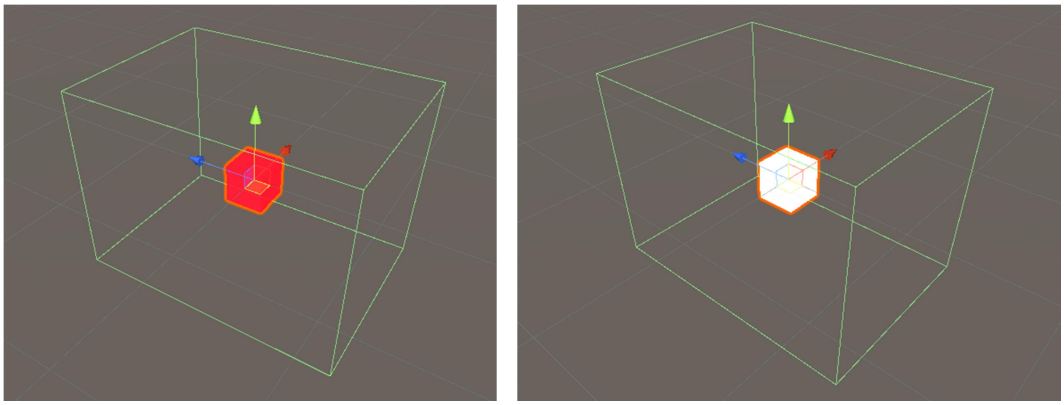

Figura 46: Objetos bonificadores junto a sus “Collider”

5.2.9. Música y configuración del menú de audio

Hasta el momento, el juego es funcional y tiene diversos efectos de sonido. Sin embargo, aunque en el tutorial explican cómo añadir música, se ha decidido omitir esa parte del tutorial para explicarlo aquí, junto con la configuración de un menú de opciones de sonido.

Música

Lo primero es crear un “Empty Object” llamado “AudioManager”, el cual estará compuesto de un componente “AudioSource”. Hay que comprobar que las casillas “Play On Awake” y “Loop” de dicho componente están las dos activadas. Después, puede guardarse la configuración actual del objeto en un “Prefab” para poder usar la configuración en otras pantallas del juego. Después, solo hay que arrastrar la pista de audio que se quiera poner como música de fondo en este componente.

Por último, se editará la función “GameOver()” del script “GameController.cs”, añadiendo la función “AudioSource.Stop()”, la cual permite detener una pista de audio que esté activa. Con esta función y una referencia al objeto “AudioManager” se detendrá la música que suene cuando el jugador pierda la partida.

Menú de opciones de sonido

Previamente debe haberse implementado el menú de las opciones, que en este documento se ha explicado junto con la creación del menú de pausa.

El primer paso para poder controlar el volumen consiste en crear un script, llamado "SoundOptions.cs", el cual tendrá seis funciones: tres que guardarán el valor asignado de cada "Slider" para cada volumen, y tres que cogerán dicho valor para aplicarlo a las fuentes de audio. Para ello se va a utilizar la clase "PlayerPrefs", que permite guardar y utilizar preferencias establecidas por el usuario y guardarlas para posteriores sesiones de juego. Es decir, la configuración de sonido que cree el usuario se guardará para la próxima partida que juegue, sin necesidad de volverla a poner.

Por un lado, las funciones para guardar el volumen (llamadas "SetMasterVolume()", "SetMusicVolume()" y "SetSFXVolume()") requerirán del valor del "Slider" como parámetro para poder guardarlo posteriormente con cada cambio que haga el jugador. Dichas funciones comienzan las tres estableciendo el valor de cada canal de sonido a 1, y después se comprueba si existe un valor guardado en las preferencias para dicha variable. En caso de existir, el valor se actualiza con el valor del "Slider" pasado como parámetro.

El componente "AudioListener" al que accede la función "SetMasterVolume()" es el componente que permite que el jugador escuche el audio de la escena. Vienen siempre incluidos en la cámara principal que se genera junto con la escena, y es recomendable que solo haya uno en la escena. Así que lo que hace la función es aumentar o reducir el volumen de los sonidos que detecta el "AudioListener".

Por otro lado, las funciones para obtener el volumen (llamadas "GetMasterVolume()", "GetMusicVolume()" y "GetSFXVolume()") se encargan de aplicar a las fuentes de sonido el valor guardado en las preferencias del jugador. Al inicio de su ejecución comprueban si existe una preferencia con el valor del volumen a modificar. Si existe, se guarda como variable y se devuelve. En caso de que no exista preferencia se devuelve la variable con valor 1.

Por último, hay que crear otro script, llamado "SliderController.cs", el cual permitirá asignar a cada "Slider" un canal de audio, de manera que su valor permitirá modificar un volumen en concreto.

Hay que crear una variable de tipo String, que será donde el usuario asigne el nombre a cada "Slider". Se comprobará que dicho String no está vacío, puesto que en función del nombre asignado, el Switch hará una operación u otra. Los casos del Switch son tres, uno para el volumen maestro, otro para la música y un tercero para los efectos. En función del "Slider" que se modifique, el volumen de uno de los canales subirá o bajará.

Una vez se haya añadido a cada "Slider" este script, habrá que ponerle a cada uno un nombre de los que aparecen en cada case del Switch. También hay que crear una nueva función en el apartado "On value Changed" de cada uno de los "Sliders" mediante el botón "+". Hay que arrastrar al recuadro que requiere un objeto el panel que contiene el menú opciones, y seleccionar la función "Set" correspondiente a realizar del script "SoundOptions". Es decir, en caso de estar editando el "Slider" del volumen maestro, la función a realizar ha de ser "SoundOptions.SetMasterVolume".

El último paso consiste en asignar el valor del slider como volumen de cada "AudioSource", haciendo distinción de si es un efecto de sonido o música. Para ello, habrá que crear dos scripts sencillos, llamados "MusicVolume.cs" y "SFXVolume.cs", que realizarán esta tarea.

El script "MusicVolume.cs" habrá que añadírselo al objeto "AudioManager.cs", para posteriormente guardarlo como "Prefab".

El siguiente script, "SFXVolume.cs", cumplirá la misma función que el script anterior. Sin embargo, deberán llevarlo todos los objetos del juego que tengan un "AudioSource" que equivalga a algún efecto de sonido, es decir, los portadores del script serán las dos naves que pueden ser controladas por el jugador y los tres efectos de partículas de las explosiones.

5.2.10. Creación del selector de personajes

Una vez creadas todas las mecánicas a nivel jugable, hay que permitir al jugador darle la capacidad de decidir con qué nave jugar. Para ello se va a desarrollar un selector de personajes en una nueva escena, que se llamará "CharacterSelection".

En dicha escena, hay que ajustar la cámara, iluminación y los elementos decorativos de la escena. En este caso se ha utilizado como fondo un modelo 3D que muestra un pasillo de aspecto futurista que puede verse en la figura 47, obtenido de **[31]** junto con una luz puntual que apunta a la posición donde están las naves ubicadas.

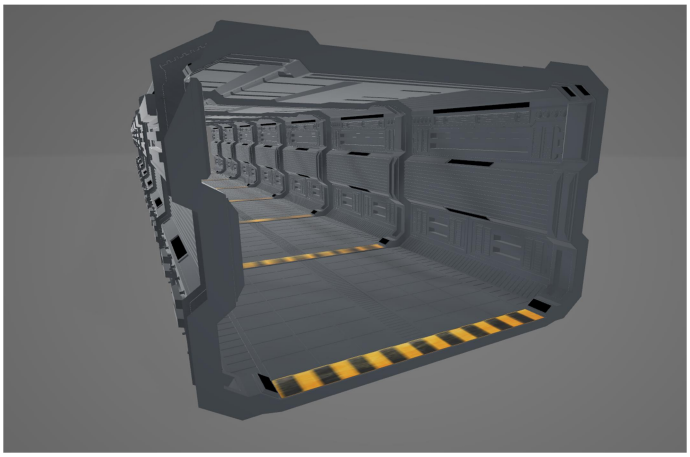

Figura 47: modelo 3D empleado en la escena "Character Selection"

Tras crear la escena el primer paso consiste en crear un "Empty Object" en el que se colocarán todas las naves que puede elegir el jugador. Dichas naves deben posicionarse en el centro y ser añadidas en el objeto hay que desactivarles varios componentes innecesarios en esta escena. Dichos componentes son: el componente "AudioSource" y los scripts "PlayerController.cs" y aquellos que permiten disparar, en este caso "ShootingFront.cs" y "ShootingRotator.cs".

Una vez hecho esto hay que añadir un script sobre el "Empty Object" creado, llamado "CharacterSelection.cs". Dicho script permitirá la selección de una de las naves para después jugar con ella. Para ello, primero habrá que crear un array de "GameObjects", del cual se definirá la longitud al inicio de la ejecución de la escena. Con el parámetro "transform.childCount" se puede calcular cuántos "hijos" tiene un objeto y así poder saber la longitud que debe tener el array. Una vez creado, hay que añadir los objetos "hijo" en cada una de sus posiciones mediante el uso de la función "GetChild(i)", que coge el objeto "hijo" en la posición "i" de la jerarquía.

Después, hay que desactivar todos los objetos que están en el array, de manera que solo esté activo uno de los objetos, dependiendo de un índice creado como variable, llamado "index".

Por último puede añadirse una función para hacer que los elementos del selector de personajes roten a lo largo del eje Y a una velocidad determinada, elegida por el usuario.

El siguiente paso consiste en implementar un panel en el que colocar tres botones: uno para incrementar el índice y pasar al siguiente elemento de la lista, otro para ir al elemento anterior y un tercer botón para confirmar la nave con la que se querrá utilizar (figura 48). También puede crearse de manera adicional un botón para que el jugador pueda volver al menú principal, el cual en este proyecto se ha ubicado en la esquina inferior derecha de la pantalla. Pero como de momento no se ha implementado dicho menú, se puede crear el botón y añadirle la función

"Back()", de manera que luego solo se tenga que poner el nombre de la escena del menú principal en la llamada a "LoadScene()". El resultado será similar al de la figura X.

Figura 48: Panel del selector de personajes

A continuación hay que crear tres funciones públicas en el script "CharacterSelection.cs" llamadas "LeftButton()", "RightButton()" y "ConfirmButton()", las cuales deberán ir asignadas al botón correspondiente del panel.

La primera función contendrá las acciones a realizar al pulsar el botón izquierdo del selector. Comenzará desactivando el modelo que está visible y después reducirá en 1 el valor de la variable "index" y comprobará si éste es menor que 0. En caso de cumplirse, el valor de la variable pasará a ser la última posición del array. Por último, se activará el modelo correspondiente a la posición que indique el valor "index" en el array.

Por otro lado, la función "RightButton()" tiene el mismo funcionamiento, cambiando que en vez de reducir el valor de "index" lo incrementa y que, en caso de que el valor de dicha variable sea mayor que el tamaño de la lista, pase a valer 0.

Por último, "ConfirmButton()" hará uso de la clase "PlayerPrefs" para guardar una nueva preferencia, que será el valor actual de la variable "index". Dicha referencia ha de cargarse al inicio de la función "Start()" para que el jugador pueda utilizar después el personaje seleccionado. Sin embargo, al pulsar el botón de confirmar han de ocurrir dos acciones más aparte de ejecutar esta función, por lo que deberán crearse dos acciones más a parte de la ya explicada. Al pulsarse debe desactivarse el panel de selector de personaje, además de activar un segundo panel que permitirá elegir el modo de juego al que jugar.

Este nuevo panel (figura 49) constará de tantos botones como modos de juego se vayan a implementar. Dicho panel estará desactivado al iniciar la escena. De momento, como solo hay un único mapa jugable, se va a crear solo un botón junto con otro que permita volver al selector de personajes. Dicho botón simplemente desactivará este panel y activará el panel del selector de personajes.

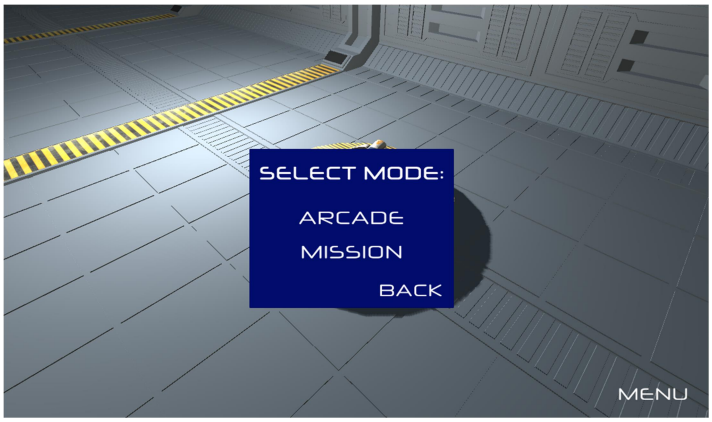

Figura 49: Panel de selección de modo de juego

Por último, habrá que copiar el objeto creado y copiarlo en la escena donde esté desarrollado el juego. Habrá que activar todos los componentes desactivados en la escena anterior, y colocar la velocidad de rotación a 0, además de desactivar todos los "hijos" que haya en el objeto.

5.2.11. Menú principal

Para crear el menú principal habrá que crear una nueva escena en la que se añadirá un panel a pantalla completa. Para el fondo del menú, se ha utilizado una imagen obtenida de **[32]**, la cual habrá que importar y modificar la propiedad "Texture Type" a "Sprite (2D and UI)" y después colocarla como textura en el panel.

Posteriormente hay que añadir el efecto de partículas "Starfield", el cual hay que copiar de la escena del juego y se debe añadir en la jerarquía. En el menú "Renderer" de cada uno de los efectos se cambiará el valor de "Order in layer" a un valor mayor que el que tenga el "Canvas" en la misma propiedad. Por último habrá que modificar en el "Canvas" la opción de "Render mode" a "Screen Space – Camera" y añadiendo la cámara de la escena en la casilla "Render Camera".

El siguiente paso consiste en crear dentro del panel tres botones y un cuadro de texto, que se usará para poner el título. A este panel habrá que añadirle un script para dar funcionalidad al botón de "jugar", que simplemente cargará la escena de selector de personajes explicada anteriormente, y también al botón "salir del juego", que cerrará el juego cuando se pulse mediante la función "Application.Quit()".

Una vez puesto el script en el panel, hay que añadir a cada botón la función del script correspondiente, como se ha explicado ya en el documento previamente.

Para implementar la acción del botón que permita acceder al menú de opciones, primero hay que copiar el panel ya creado en el menú de pausa de la otra escena y pegarlo en el "Canvas" de la escena del menú principal. Hay que recordar que dicho panel estará desactivado desde el inicio.

Tras eso, hay que configurar el botón de opciones para que active el panel correspondiente y desactive el menú principal y el título. Así mismo se deberá implementar un botón que realice las acciones inversas, es decir, desactive el panel de las opciones y active el menú principal y el título.

Hay que recordar añadir esta escena al recuadro “Build Settings”, y posicionarla en la primera posición de la lista, haciendo que tenga valor 0. Esto hará que al iniciar el juego, esta escena sea la que inicie en primer lugar (figura 50).

Figura 50: Menú principal

Una vez terminada la escena con el menú, hay que terminar de configurar los botones de las otras dos escenas, de manera que permitan transitar a esta escena al pulsarlos. Por un lado, en el script “ModeSelection.cs” deberá modificarse el nombre de la escena a la que se hace referencia en la función “Back()”, poniendo el nombre correspondiente a la escena del menú principal.

Por otro lado, se debe actualizar también el botón del menú de pausa, añadiendo el nombre correspondiente a la escena del menú principal en la función “LoadMenu()” del script “PauseMenu.cs”.

5.2.12. Reemplazar Modelos de las naves

En este apartado se va a explicar cómo crear dos naves nuevas utilizando las ya creadas anteriormente como base. Los modelos empleados se han descargado de los siguientes enlaces **[33]** y **[34]** y pueden verse en la figura 51.

Lo primero de todo es importar los modelos que tengan extensión “.fbx” y después añadir dichos modelos al proyecto, pulsando con el botón derecho en la carpeta “Assets”, seleccionar la opción “Import new Asset” y escoger aquellos modelos a añadir.

Con los pasos anteriores se obtiene el modelo completo pero sin texturas, las cuales se añadirán mediante el botón “Extract Materials” de la pestaña “Materials”. Después hay que elegir la carpeta en la que guardar dichos materiales y ajustar la escala del modelo, puesto que debe tener un tamaño acorde a la escena. Esto puede hacerse mediante la opción “Scale Factor” en la pestaña “Model”.

El siguiente paso consiste en crear una copia del “Prefab” de cada nave en los que se eliminará el modelo actual y se añadirán los nuevos modelos. Una vez hecho este cambio se debe reajustar el tamaño del “Collider” y guardar los cambios hechos.

Por último, se deben añadir las nuevas naves al objeto “Character Selection”, tanto de la escena del juego como en el selector de personajes.

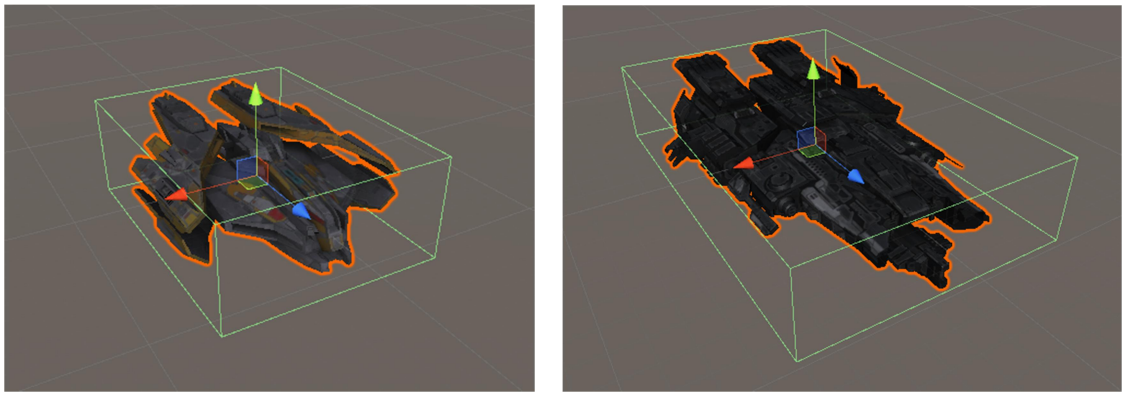

Figura 51: modelos nuevos empleados junto con sus "Collider"

5.2.13. Añadir instrucciones con los controles al juego

Tras añadir todos los modelos al objeto "CharacterSelector" en las dos escenas, habrá que hacer una serie de modificaciones al script "CharacterSelection.cs" para mostrar los controles de cada nave correctamente.

El primer paso consiste en organizar las naves en la jerarquía, de forma que aquellas naves con las que se pueda apuntar estén en posiciones pares y el otro tipo de naves en las impares.

A continuación hay que crear dos botones y dos paneles, uno por tipo de nave, y escribir en cada uno de los paneles las instrucciones correspondientes. Cada uno de los botones abrirá uno de los paneles, además de desactivar la interfaz de selector de personajes. Estos paneles tendrán además un botón para cerrarse y volver a mostrar los controles.

En cuanto al script "CharacterSelection.cs" se deberán crear dos referencias, una para cada botón y en las funciones "Start()", "LeftButton()" y "RightButton()" establecer una condición que compruebe si el índice de la lista es par o impar, y dependiendo del resultado se active un botón u otro. De esta manera cada vez que se cambie de índice en el selector de personajes al pulsar uno de los botones, el panel de las instrucciones cambiará y se mostrará acorde a la nave activa.

Por último, hay que crear un script el cual irá añadido al objeto "CharacterSelector" de la escena del juego, llamado "ControlsManager.cs", el cual tendrá una función llamada "ControlSelect()". Este script tendrá referencias al índice del script "CharacterSelection.cs", y al botón creado en el menú de pausa con las opciones. Por lo tanto, cuando se pulse el botón de los controles se desactivará el menú de pausa y la función comprobará el valor del índice y, en función de su valor, se activará un panel de instrucciones u otro. Esta función también se debe asignar al botón "Back" del selector de modo de juego en la escena "Character Selection".

5.2.14. Creación de un modo de juego finito

Hasta este punto del desarrollo se han implementado diversas mecánicas sobre un modo de juego el cual no tiene fin. Sin embargo, este tipo de juegos ofrecen una gran diversidad en cuanto al desarrollo y diseño de niveles del mismo. Es por eso que en este apartado va a diseñarse un modo de juego que esta vez sí que tenga final. Dicho modo de juego estará implementado en otra escena, a la cual se podrá acceder mediante un botón que se añadirá al panel del selector de modo de juego, el cual hasta este momento sólo tenía un botón operativo.

Tras crear el botón en el panel, habrá que modificar el script "ModeSelection.cs", al cual habrá que añadirle una función de nombre "MissionMode()" que permitirá cargar la escena donde se esté desarrollando este nuevo modo de juego mediante la función "SceneManager.LoadScene("Nombre")".

Crear nueva escena

Para comenzar el desarrollo de este nuevo modo de juego, puede crearse un duplicado de la escena de juego existente y cambiarle el nombre la nueva escena por el que le corresponda. Una vez creada la nueva escena, el primer paso consistirá en cambiar el material del fondo y así hacer cada modo visualmente distinto. En este caso la textura empleada ha sido sacada de **[35]**, de donde se han descargado el "Normal Map" y el "Diffuse Map".

El "Normal map" es un tipo de imagen que permite dar la sensación de relieve, mientras que el "Diffuse Map" es el que le da color al "Normal Map".

Una vez descargadas e importadas dichas texturas al proyecto, hay que crear un material nuevo de nombre "Snow" en el que habrá que añadir el "Normal map" en la casilla del mismo nombre en el apartado "Main Maps". Adicionalmente se debe colocar el "Diffuse Map" en la casilla "Albedo" del mismo apartado para darle color a la textura.

Tras haber creado el material debe asignarse al objeto "Background", tanto al "padre" como al "hijo", de forma que ahora el fondo será blanco.

Por último, se puede modificar el efecto de partículas que anteriormente hacía de estrellas para que ahora imite una ventisca. Para ello, hay que comenzar por modificar la posición del efecto de partículas, que estará posicionado en las coordenadas (-20, 0, 5) para que las partículas aparezcan desde el lateral. Después hay que modificar cada uno de los dos efectos que forman el efecto.

En el primer efecto (part_starField), se debe modificar el color del efecto a blanco, además de la velocidad del apartado "Start speed", cuyos valores serán 2 y 10, respectivamente. También habrá que modificar el valor de "Max Particles" a 100.

En el segundo efecto se debe modificar, a parte del color a blanco, el apartado "Start Speed" que tendrá unos valores de 6 y 12, y un valor en "Max Particles" de 1500.

Una vez configurado el escenario, hay que crear dos componentes "AudioSource" en el objeto "GameObject", de manera que tendrá tres componentes de este tipo. El componente ya existente será el que controle la música de juego, el segundo se utilizará para controlar la música que suene durante el combate contra el jefe, y el tercero se empleará para la música de victoria.

Modificaciones a scripts

El siguiente paso consiste en crear un nuevo script llamado “MissionController.cs”, el cual será una copia idéntica del script “GameController.cs” empleado anteriormente. De esta manera se tendrá un controlador del juego para cada tipo de modo diferente, y poder editar cada uno de ellos de manera más cómoda. Tras crear la copia habrá que eliminar el script “GameController.cs” del objeto “GameController” y añadirle en su lugar el nuevo script.

En este script se deberán crear una serie de variables para controlar el número de oleadas de enemigos que se generan en la función “SpawnWaves()”, además de una variable que represente el número máximo de oleadas a superar por el jugador, cuyo valor se establecerá de manera aleatoria al inicio de la partida.

Tras instanciar todas las oleadas, se detendrá la música y tras un tiempo de espera, se instanciará el jefe y empezará a sonar otra música diferente.

En caso de vencer, se iniciará la función “Victory()”, que tendrá un funcionamiento similar a “GameOver()” pero reemplazando el texto “Mission Failed!” por “Mission Completed!” e iniciando la reproducción de la música de victoria.

Otro script que hay que replicar es “DestroyByContact.cs”, cuya copia se llamará “MissionDestroyByContact.cs”. En este script, que deberán llevarlo todos los enemigos de este modo de juego, son necesarias algunas modificaciones en las que habrá que sustituir las referencias a “GameController” por referencias a “MissionController” en la función “Start()”.

Otra modificación que hay que hacer es la implementación de una nueva función en el script “PauseMenu.cs” llamada “RestartMission()”, que permitirá reiniciar el nivel recargando la escena actual. Dicha función será idéntica a “Restart()” pero cambiará el nombre del nombre pasado como parámetro a la función “LoadScene()”, por lo tanto se debe asignar al botón “Restart”.

Enemigos

Puesto que para este modo de juego se ha decidido cambiar la ambientación del mapa, se crearán nuevos enemigos a partir de los ya creados.

Dron

Para comenzar hay que crear un duplicado del “Prefab” de uno de los asteroides, de manera que se reemplace el modelo por el del dron, obtenido de **[36]** y al cual se habrá reducido su escala a 0.1. Posteriormente se ha añadido el script “EvasiveManeuver.cs” para que además de avanzar se desplace hacia los lados y se tendrá que eliminar el script “RandomRotator.cs”. Tras aplicar estos cambios y ajustar el “Collider” a uno que se ajuste mejor a la forma del objeto, habrá que guardarlo como “Prefab”. Este enemigo se puede ver en la figura 52.

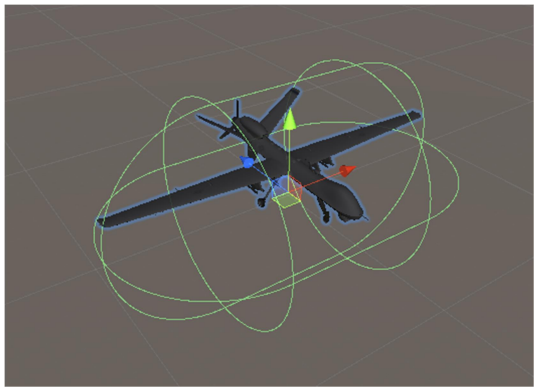

Figura 52: Modelo del dron junto a su "Collider"

Nave enemiga

Para crear este enemigo basta con crear una copia del "Prefab" de la nave enemiga ya creada y reemplazar su modelo por el nuevo, obtenido de **[37]** y con un factor de escala de 0.2. Tras eso hay que ajustar el punto desde el que se generan los disparos, además del "Collider" y la rotación del modelo. El modelo se puede ver en la figura 53.

Por último, debe reemplazarse el efecto de partículas del motor de la nave enemiga por el mismo efecto que utiliza la nave del jugador.

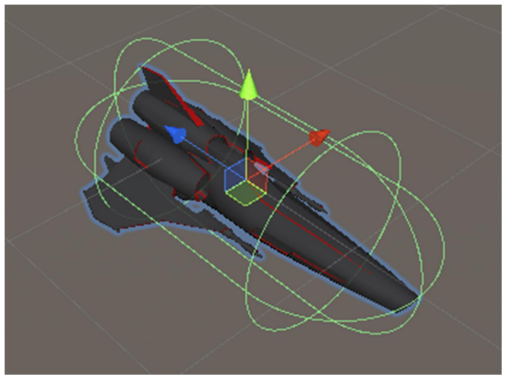

Figura 53: Modelo de la nave enemiga junto a su "Collider"

Enemigos laterales

Para crear el último enemigo estándar de este modo de juego hay que crear una copia del "Prefab" de uno de los enemigos de los laterales del otro modo de juego y reemplazar el modelo por el obtenido en el enlace **[38]**. Una vez añadido el nuevo modelo se le debe modificar el factor de escala a 0.3 y añadirle el efecto de partículas del motor del jugador, ajustándolo al modelo, que aparece en la figura 54.

Una vez creado, hay que guardar la configuración en un nuevo "Prefab", y crear una copia de éste en la que se modificará la orientación del modelo.

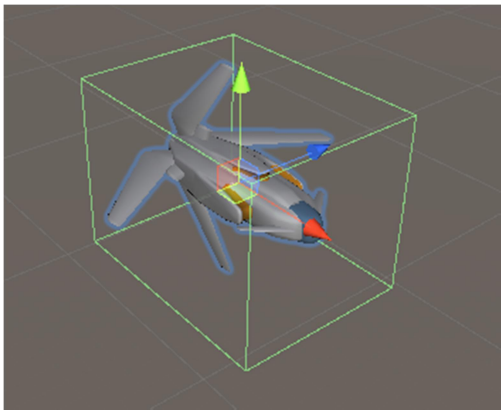

Figura 54: Modelo del enemigo lateral junto a su "Collider"

Es importante recordar añadir el script "MissionDestroyByContact.cs" empleado en todos los enemigos de este modo en sustitución del script "DestroyByContact.cs".

Jefes

Los jefes serán los últimos enemigos que deberá derrotar el jugador antes de terminar la partida. Por lo tanto, deben tener un comportamiento diferente al resto de enemigos a los que se enfrenta el jugador el cual se creará mediante variaciones de los scripts existentes específicas para este tipo de enemigo.

Pese a que durante la partida se deberá enfrentar solamente a un jefe se han creado dos distintos para darle variedad a cada partida que se juegue. Los modelos empleados para cada jefe se han obtenido de **[39]** y de **[40]** y se pueden ver en la figura 55. Una vez incluidos ambos modelos en el proyecto, se debe ajustar la escala de cada uno de ellos de manera que quepan en la pantalla y sean de un tamaño superior al jugador.

Para comenzar, basta con crear una copia del "Prefab" de la nave enemiga creada anteriormente, a la cual reemplazaremos el modelo por uno de los nuevos y ajustaremos el "Collider".

Estos enemigos tendrán 4 puntos de disparo diferentes: dos en el centro y dos en las extremidades. Respecto al resto de componentes, sólo deben mantenerse el "AudioSource" y el "Rigidbody" junto con los scripts "SFX Volume.cs" y "EvasiveManeuver.cs".

Una vez configurados estos componentes hay que añadirle algunos nuevos para el correcto funcionamiento del enemigo. Para ello, se crearán tres scripts nuevos: "BossDestroyByContact.cs", "MechaWeaponController.cs" y "MoverBoss.cs".

El primero de estos scripts, "BossDestroyByContact.cs", es una copia de "MissionDestroyByContact.cs". Sin embargo, en este script se elimina la generación de objetos y se añade una llamada a la función "Victory()" del script "MissionController.cs".

El segundo script a crear, "MechaWeaponController.cs" será el encargado de controlar los patrones de ataque del enemigo. Para cubrir el mayor espacio posible, este script ejecutará dos corrutinas con las que instanciará los disparos. La primera corrutina, de nombre "Fire()", se encargará de generar disparos en arco desde los objetos centrales, cubriendo un ángulo de 120º. Por otro lado, la corrutina "FireSides()" instanciará disparos en línea recta desde los objetos situados a los lados. El tiempo de espera entre ráfagas se generará de manera aleatoria, con un valor comprendido entre dos rangos para hacerlo más impredecible.

El último script a crear para este enemigo, “MoverBoss.cs”, el cual realiza una función particular. Este script se encarga de mover al enemigo desde su punto de aparición hasta una posición concreta, en este caso, la parte superior de la pantalla del jugador.

Para esta implementación se comienza calculando la posición inicial y la distancia inicial al destino mediante el uso de la función “Vector3.Distance(inicio, destino)”. Una vez calculado, se inicia una corrutina que está activa mientras la distancia actual del enemigo y el destino sea superior a 0 en la que se va calculando la distancia recorrida, la posición actual y su nueva posición mediante la función “Vector3.Lerp(inicio, destino, t)”, que calcula la interpolación entre dos vectores y obtiene el punto correspondiente a “t”. Es decir, si “t” vale 0, el objeto estará en la posición “inicio”, y si vale 1 estará en la posición “destino”. El bucle se ejecutará sin pausa mientras no se haya llegado al destino.

Una vez llegue a su destino, el enemigo detiene su avance y simplemente se desplazará hacia los lados debido al script “EvasiveManeuver.cs”.

Por último, hay que crear un duplicado del “Prefab” de los disparos enemigos para crear una variante exclusiva de estos enemigos. Estos disparos tendrán el script “MissionDestroyByContact.cs” añadido y estéticamente serán rojos y con la escala duplicada al doble. Además, se moverán a una velocidad menor que el resto de disparos, con un valor de 55.

Tras estas modificaciones, se deben asignar correctamente todos los objetos en las correspondientes instanciaciones en el script “MissionController.cs” del objeto “GameController”. También se deberán dar valores (a criterio del usuario) a las distintas variables del script.

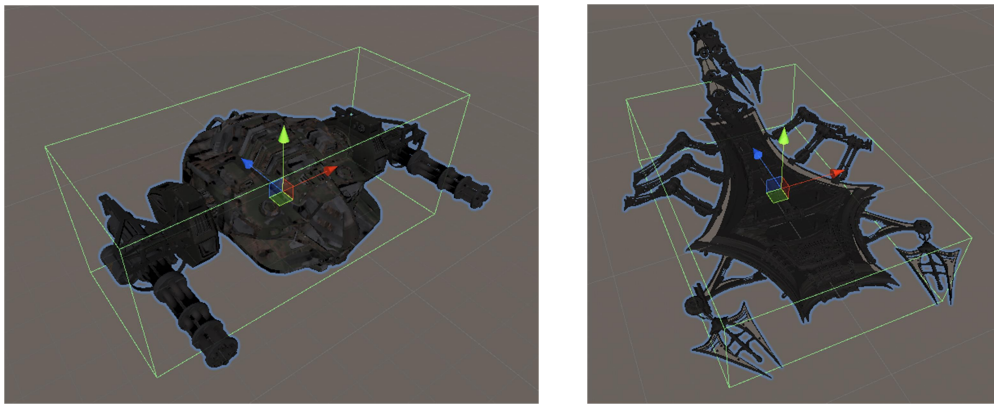

Figura 55: Modelos de los dos jefes creados, con sus respectivos “Collider”

6. Pruebas

A la hora de realizar la primera fase de pruebas, apareció un error el cual no permitía el correcto funcionamiento del juego cuando se empezaba una partida nueva tras salir de la anterior desde el menú de pausa. El problema resultó ser que la función que realizaba el botón para ir al menú no colocaba el valor correcto de la variable que indica si el juego está pausado o no, por lo tanto, al iniciar la nueva partida, el juego asumía que la partida todavía estaba pausada. Hacer la asignación correcta en la función "LoadMenu()" del script "PauseMenu.cs" antes de la carga de la escena en la que se encuentra el menú resolvía el problema.

Otro error muy importante que apareció durante la fase de pruebas fue un fallo que permitía al usuario modificar los valores de los indicadores de vida y tiempo. A pesar de que los valores internos no variaban y eran correctos, podía darse el caso de que el usuario pulsase sobre uno de estos indicadores, por lo que la información mostrada era errónea. Su corrección fue muy sencilla, ya que "Unity" elegir si se quiere el usuario modifique o no los valores del "Slider" mediante la casilla "Interactable".

Un tercer error surgido durante el desarrollo era uno que permitía usar dos veces el ataque especial si el usuario pulsaba muy deprisa el botón derecho del ratón. Para solucionarlo se optó por poner una variable que pasaba de 1 a 0 si se pulsaba el botón derecho junto con una condición a la hora de instanciar la función del disparo, en la cual se especificó que dicha variable debía tener un valor mayor a 0 para poder ejecutarla. De esta manera, cuando el jugador pulsa por primera vez el botón derecho del ratón, se decrementa el valor de la variable, por lo que no se puede volver a llamar a la función. Al final de ésta, cuando se establece el tiempo de reutilización, se vuelve a incrementar el valor a 1.

Sin embargo, los problemas más comunes durante el desarrollo han sido meramente estéticos, como la correcta posición de los textos en pantalla de "Game Over" y nombres de los botones. Estos errores simplemente se corrigen aumentando el tamaño del cuadro de texto o reduciendo el tamaño de la letra.

7. Resultados

Una vez terminado todo el desarrollo y compilado el proyecto, el resultado será un juego "arcade" con cuatro naves diferentes a elegir. Cuando se inicie el juego, lo primero que encontrará el jugador es el menú principal (figura 56) con tres opciones: "jugar", gestionar las opciones de sonido o salir del juego.

Figura 56: Menú principal en el juego compilado

En caso de que el jugador seleccione el botón "Options", se le redirigirá a una pantalla con tres "Sliders" con los que gestionar el volumen del juego (figura 57). Pulsando el botón "Back", se volverá al menú principal.

Figura 57: Ventana de opciones desde el menú principal

Si el jugador selecciona “Play”, se le redirigirá a la escena de selección de personaje, en el que podrá elegir entre diferentes naves con las que jugar. Actualmente se dispone de cuatro naves, dos por cada estilo de juego, que se dividen en “disparo frontal” y “disparo rotatorio”.

Se puede elegir entre naves mediante los botones que están a ambos lados del botón “Confirm”, y en caso de que se quiera volver al menú principal, hay que pulsar el botón “Menu”. En las imágenes 58 y 59 se muestran las dos naves de disparo frontal.

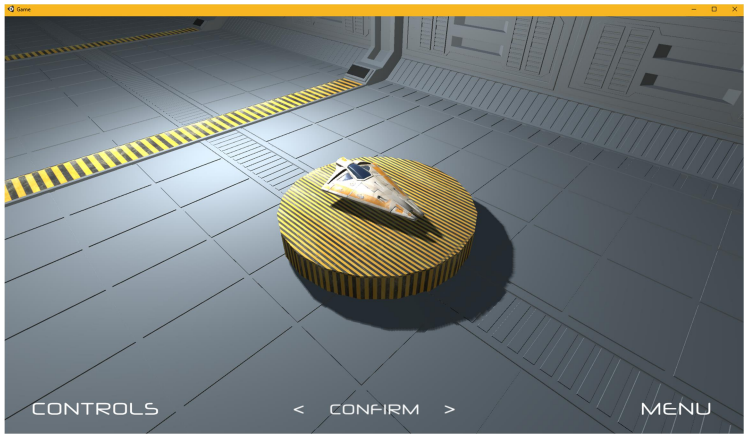

Figura 58: Primera nave de disparo frontal

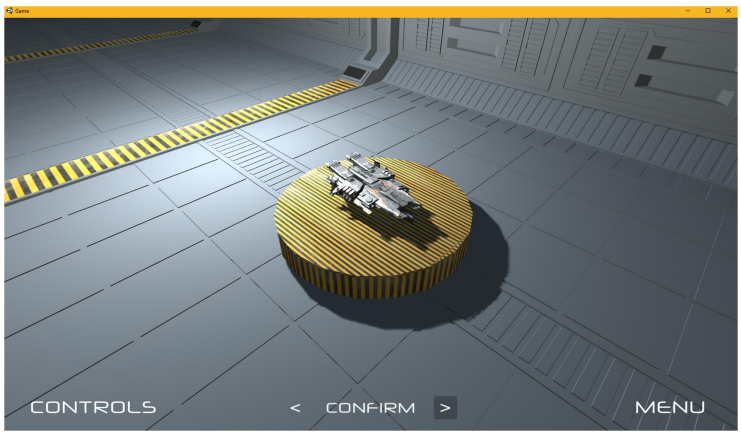

Figura 59: Segunda nave de disparo frontal

Por otro lado, las figuras 60 y 61 muestran las dos naves de disparo rotatorio.

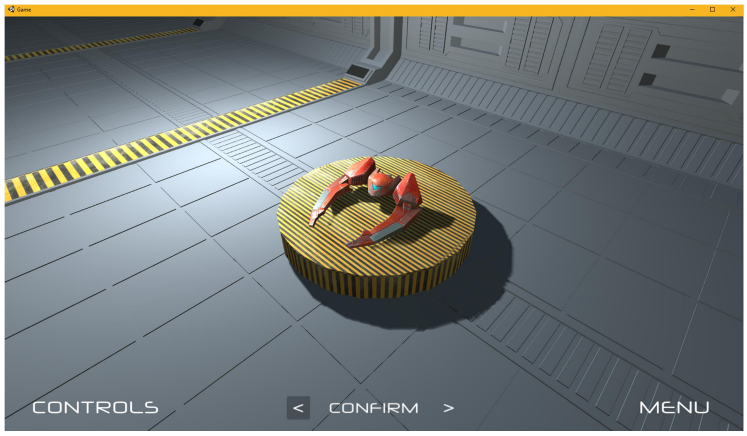

Figura 60: Primera nave de disparo rotatorio

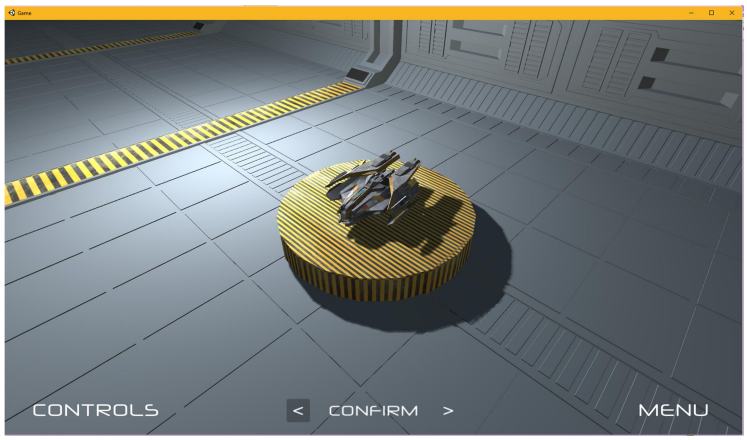

Figura 61: Segunda nave de disparo rotatorio

Pulsar sobre el botón “Controls” abre una ventana con la explicación de la nave que se muestra en pantalla. Puesto que tienen controles diferentes, dependiendo de la nave que esté seleccionada se mostrarán unos controles u otros, como puede verse en las figuras 62 y 63, en las que se muestran los controles de las naves de disparo frontal y rotatorio, respectivamente. En ambas ventanas pulsar el botón “Back” permite volver a la selección de personaje.

Figura 62: Controles de la nave de disparo frontal

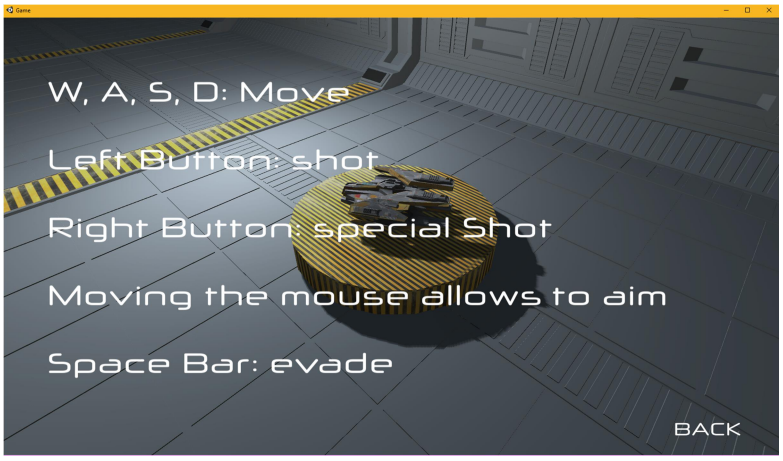

Figura 63: Controles de la nave de disparo rotatorio

Tras seleccionar una nave, hay que pulsar sobre el botón “Confirm”, el cual abrirá un panel con un selector de niveles, el cual puede verse en la figura 64. Dichos niveles son dos: “Arcade”, que consiste en un modo de juego infinito cuyo objetivo es obtener la mayor cantidad de puntos posible, y “Mission”, en el cual hay que superar el nivel derrotando a un enemigo final tras avanzar por el escenario. Por último, pulsar el botón “Back”, permite volver a la selección de nave.

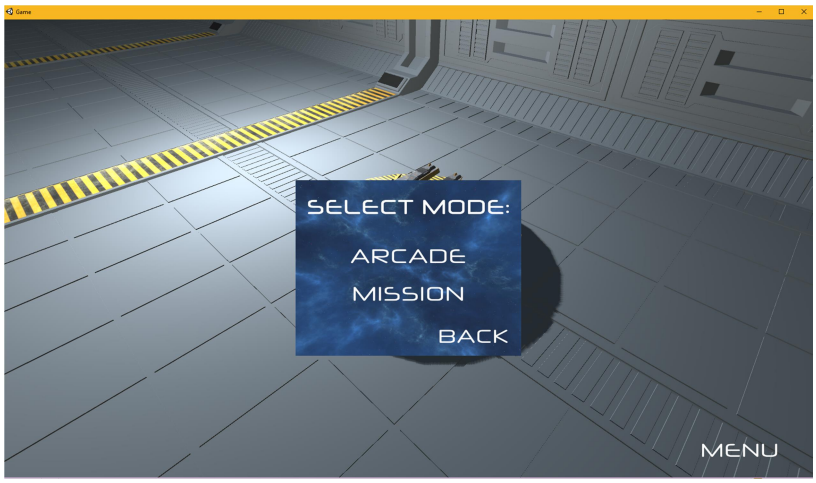

Figura 64: Selector de niveles del juego compilado

En la figura 65 se muestra una captura del modo de juego “arcade” en ejecución. Como puede verse, los enemigos se irán generando aleatoriamente de manera que pueden aparecer naves junto a asteroides.

Figura 65: Captura del modo "Arcade" en el juego compilado

Por otro lado, la figura 66 muestra una captura del modo de juego "Mission", en la cual pueden verse los enemigos venir tanto de frente como de los lados, además de un par de objetos recolectables, uno de cada tipo.

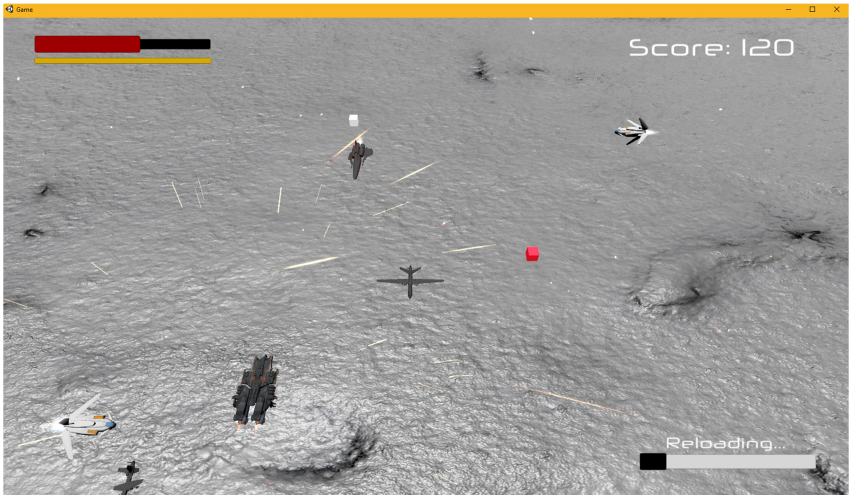

Figura 66: Captura del modo "Mission" en el juego compilado

Una vez se superan las oleadas en el modo "Mission", se llega a una pelea contra el enemigo final, como se ve en la figura 67. En caso de vencer al enemigo, se mostrará la pantalla de victoria.

Figura 67: Captura del modo "Mission" junto con el enemigo final

Ambos modos de juego tienen un menú de pausa idéntico, el cual se puede ver en la figura 68. Este menú se puede abrir con el botón "escape" y muestra cinco botones diferentes. El primer botón, "Resume", sirve para continuar la partida. "Restart" se emplea para reiniciar el juego, y "Controls" para mostrar los controles de forma idéntica a como se muestran en las figuras 63 y 64. Por otro lado, "Options" permite abrir un menú de opciones idéntico al de la figura 57. Por último, el botón "Menu" permite volver al menú principal de la figura 56.

Figura 68: Menú de pausa del juego compilado

8. Conclusiones

Este proyecto me ha dado la posibilidad de experimentar el desarrollo de videojuegos con la herramienta “Unity 3D” de primera mano. Así mismo, me ha ayudado también a diferenciar las distintas fases por las que pasa un juego a lo largo del proceso. Por otro lado, también ha permitido explorar, bastantes de las herramientas de las que dispone Unity.

En cuanto al juego, pese a no tener mucha variedad en cuanto al número de niveles disponibles, se han implementado dos modalidades diferentes con las que dar mayor variedad al juego. También se han implementado todas las mecánicas necesarias para su correcto funcionamiento, por lo que siguiendo las explicaciones de esta memoria se pueden crear en el futuro tantos niveles como se deseen gracias a la variedad que puede dar este tipo de juegos.

Por último, los objetivos establecidos al inicio de la memoria se han cumplido satisfactoriamente. Gracias a la amplia documentación de la que dispone “Unity”, todas las tareas que se habían propuesto se han podido resolver fácilmente. Gracias a aspectos como la agrupación de componentes en un solo objeto o la configuración de los botones para abrir menús y elegir niveles facilitan mucho el desarrollo de cara al usuario.

Pese a que “Unreal Engine” y “CryEngine” son su competencia directa y son capaces de mostrar mayor potencia gráfica que “Unity”, éste presenta muchas más facilidades en otros aspectos como por ejemplo en cuanto a implementación de código, por lo que se hace más intuitivo a desarrolladores noveles.

9. Bibliografía

9.1. Referencias de fuentes electrónicas

[1] Página web de la noticia: https://www.europapress.es/portaltic/videojuegos/noticia-industria-videojuegos-factura-1530-millones-euros-espana-2018-20190507115604.html

[2] Página web oficial de Unreal Engine: https://www.unrealengine.com/en-US/

[3] Página web oficial de Unity 3D: https://unity.com/es

[4] Página web de los tutoriales oficiales de Unity: https://unity3d.com/es/learn/tutorials

[5] Página web del tutorial empleado como base: https://unity3d.com/es/learn/tutorials/projects/space-shooter/introduction?playlist=17147

[6] "Doom Engine": https://www.neoteo.com/los-motores-graficos-mas-importantes-de-la-histori/

[7] "Quake Engine": https://www.pcgamer.com/the-legacy-of-quake-20-years-later/

[8] "Doom Eternal" utiliza "Id Tech 7": https://arstechnica.com/gaming/2018/08/doom-eternal-reveals-new-powers-puts-hell-back-on-earth/

[9] Motor gráfico "GoldSrc": https://developer.valvesoftware.com/wiki/Goldsource

[10] Motor gráfico "Source": https://developer.valvesoftware.com/wiki/Source

[11] "Dota 2" será el primer juego en emplear el motor gráfico "Source 2": https://www.ign.com/articles/2015/09/09/dota-2-now-valves-first-ever-source-2-game

[12] Creación del motor gráfico "Dunia Engine": https://vandal.elespanol.com/ware/articulos/3090/la-evolucion-del-dunia-engine-el-motor-grafico-de-far-cry-new-dawn

[13] Motor gráfico "Cry Engine V": https://www.cryengine.com/

[14] Motor gráfico "Unreal Engine": https://hotgates.eu/this-is-unreal/

[15] Motor gráfico "Unity": https://www.vidaextra.com/industria/unity-el-motor-de-desarrollo-capaz-de-partir-la-historia-de-los-videojuegos-en-dos

[16] Juegos desarrollados con "Unity": https://unity3d.com/games-made-with-unity

[17] Corto animado "ADAM: The Mirror": https://elchapuzasinformatico.com/2017/11/director-distrito-9-usara-motor-grafico-unity-proxima-pelicula/

[18] Versiones actuales de "Unity": https://store.unity.com/es/

[19] Historia de los juegos estilo “shoot ‘em up”: https://www.hobbyconsolas.com/reportajes/mejores-videojuegos-retro-shoot-em-juegos-naves-83808

[20] Nacimiento de los juegos “Bullet Hell”: https://www.usgamer.net/articles/curtains-for-you-the-history-of-bullet-hell

[21] Análisis “Ikaruga” para Nintendo Switch: https://www.nintenderos.com/2018/06/analisis-ikaruga/

[22] Fecha de lanzamiento de “Touhou 17”: https://www.dualshockers.com/zun-reveals-touhou-17-wily-beast-weakest-creature/

[23] Página oficial de “NieR: automata”: https://www.platinumgames.com/games/nier-automata?age-verified=e72806708b

[24] Página web oficial de Visual Studio: https://visualstudio.microsoft.com/es/

[25] Manual de Unity: https://docs.unity3d.com/Manual/index.html

[26] Enlace de descarga de la fuente de texto: https://www.1001fonts.com/neuropol-x-free-font.html

[27] Página de obtención del script para bloquear la rotación de un gameObject: https://answers.unity.com/questions/176235/how-to-freeze-child-gameobject-from-rotation.html

[28] Obtención del script para la rotación con el uso del ratón: http://wiki.unity3d.com/index.php?title=LookAtMouse

[29] Tutorial “PAUSE MENU in Unity”: https://www.youtube.com/watch?v=JivuXdrIHK0

[30] Obtención del script para el reescalado de la cámara: http://gamedesigntheory.blogspot.com/2010/09/controlling-aspect-ratio-in-unity.html?m=1

[31] Enlace de descarga del modelo del selector de personajes: https://www.cgtrader.com/items/2002986/download-page

[32] Enlace de descarga de la imagen empleada en el menú principal: https://www.cgtrader.com/free-3d-models/space/other/spacebox-collection

[33] Enlace de descarga del primer modelo: https://www.cgtrader.com/items/688922/download-page

[34] Enlace de descarga del segundo modelo: https://www.cgtrader.com/items/623682/download-page

[35] Enlace de descarga del material del nuevo mapa: https://texturehaven.com/tex/?c=terrain&t=snow_02

[36] Enlace de descarga del modelo del dron: https://www.cgtrader.com/items/800272/download-page

[37] Enlace de descarga del modelo de la nave:
https://www.cgtrader.com/items/213917/download-page

[38] Enlace de descarga del modelo de la nave lateral:
https://www.cgtrader.com/items/214232/download-page

[39] Enlace de descarga del modelo del primer jefe:
https://www.cgtrader.com/items/51333/download-page

[40] Enlace de descarga del modelo del segundo jefe:
https://www.cgtrader.com/items/764746/download-page

[41] Comparativa “Unreal Engine” vs “CryEngine” vs “Unity”:
https://medium.com/@thinkwik/cryengine-vs-unreal-vs-unity-select-the-best-game-engine-eaca64c60e3e

10. Controles e interfaz

Los controles del juego son los siguientes:

- W, A, S, D: movimiento de la nave
- Barra espaciadora: esquivar
- Botón izquierdo del ratón: disparar (algunas naves permiten apuntar con el ratón).
- Botón derecho del ratón: ataque especial

Como se ha ido explicando a lo largo de la memoria, en el juego hay dos tipos de naves: las de disparo frontal y las de disparo rotatorio. Cada una tiene una serie de características diferentes, las cuales se van a comentar a continuación.

Disparo frontal:

- Tiene 5 puntos de vida (mostrada en la barra roja de la esquina superior izquierda)
- Se desplaza más deprisa que la nave de disparo rotatorio
- Hace un punto de daño por disparo

Disparo rotatorio:

- Tiene 7 puntos de vida (mostrada en la barra roja de la esquina superior izquierda)
- Se desplaza más despacio
- Hace 2 puntos de daño por disparo

Aspectos comunes:

- El tiempo de reutilización del ataque especial es de 20 segundos, y se muestra en la barra blanca de la esquina inferior derecha.
- El tiempo de reutilización de la evasión es de 0.75 segundos y se muestra en la esquina superior izquierda, debajo de la vida.
- La interfaz (figura 69) es idéntica en ambos tipos de nave

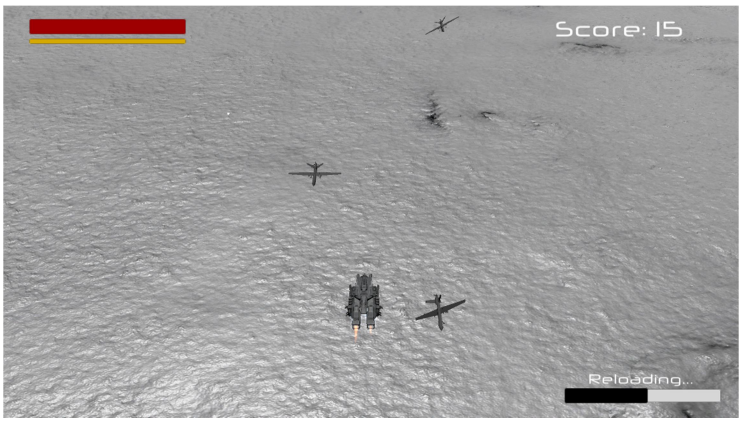

Figura 69: Interfaz de usuario