Memoria del proyecto
Space Project

1. Introducción

Space Project es un pequeño proyecto desarrollado para la asignatura de Programación Avanzada con el objetivo de poner en práctica todos los conocimientos aprendidos durante el desarrollo de las clases, y conseguir un proyecto con un código reutilizable, limpio y escalable.

El concepto inicial del proyecto consiste en la creación de un juego de acción espacial en tercera persona. Entrando más en detalle se han definido los siguientes puntos clave:

- El jugador tendrá el control sobre una nave espacial de las siguientes características:
    - La nave podrá moverse por el espacio en las 6 dimensiones.
    - Tendrá a su disposición dos tipos de armas:
        - Un lanzacohetes, en caso de que se haya fijado un objetivo, estos cohetes se dirigirán hacia este. Estos cohetes serán más efectivos contra enemigos sin escudo.
        - Un rayo láser, que será más efectivo ante los escudos de los enemigos.
    - La vida de la nave se compondrá de:
        - Un escudo, que recibirá todo el daño en caso de estar activo y se recargará en base al tiempo.
        - La vida como tal, si esta llega a 0 la nave acabará destruida.
- Los enemigos y obstáculos a los que se enfrentará en jugador consistirán en:
    - El jugador estará rodeado por asteroides en todo momento, estos harán daño a todo objeto que colisione con ellos, además podrán ser destruidos si se les dispara con cualquier tipo de arma.
    - Existirán enemigos inmóviles, con un comportamiento similar al de los asteroides, estos, además atacarán al jugador cuando este entre en su rango de acción.
    - El jugador también deberá de enfrentarse a naves enemigos con un comportamiento similar al del propio jugador.

2. Desarrollo del proyecto

El proyecto se ha desarrollado en la versión 2019.4.13f1 de Unity, haciendo uso del alumbrado lineal para conseguir una mayor fidelidad de los colores mostrados por pantalla. Además, se ha hecho uso de una serie de Add-ons para mejorar el acabado y la calidad del producto final. El listado de estos elementos se compone de:

- TextMesh Pro, es un Add-on utilizado para conseguir texto de alta calidad en la interfaz de usuario.

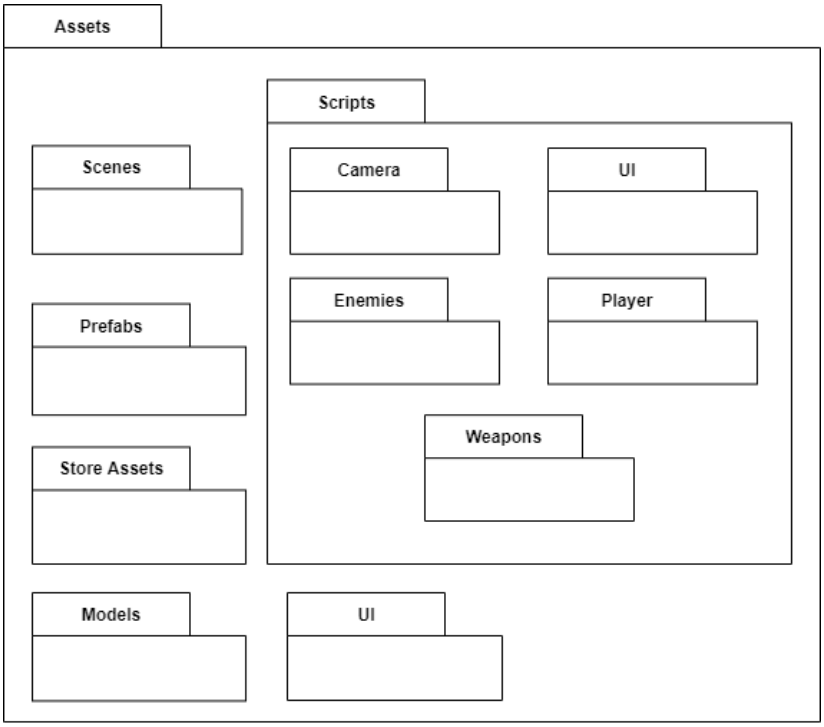
- Post Processing Stack, nos permite añadir una serie de efectos de post procesado a la imagen final, con la intención de mejorar su aspecto final.
- Starfield Skybox, nos proporciona un skybox personalizado que se adapta a la ambientación de nuestro videojuego.

También se han hecho uso de una serie de herramientas externas:

- Git, es un software de control de versiones utilizado para mantener un registro de los cambios realizados en los archivos durante el proceso de desarrollo.
- Blender, es un programa gratuito de modelado utilizado para la creación de los diferentes assets utilizados en el proyecto.

3. Diagrama de paquetes

A continuación, pasaremos a explicar la agrupación del proyecto en paquetes.



a. Scenes

El paquete “Scenes” contiene los entornos y menús del juego. Además, en esta carpeta almacenaremos los perfiles que contienen la información de postprocesado aplicada a la cámara en cada una de las escenas.

b. Prefabs

En el paquete “Prefabs” almacenamos las copias de los objetos que utilizaremos en la escena con sus componentes y propiedades preasignadas. De este modo, se nos permitirá instanciar lo diferentes assets en tiempo de ejecución con una configuración base.

c. StoreAssets

El paquete “StoreAssets” contiene los materiales externos utilizados en el proyecto. De esta manera mantenemos separados los assets de terceros de los propios del proyecto.

d. Models

En el paquete “Models” almacenamos todos los modelos 3D en formato obj para su utilización en el proyecto, así como los materiales que se aplicarán a cada uno de estos objetos.

e. UI

El paquete “UI” contiene todos los sprites que se utilizarán para componer la interfaz de usuario.

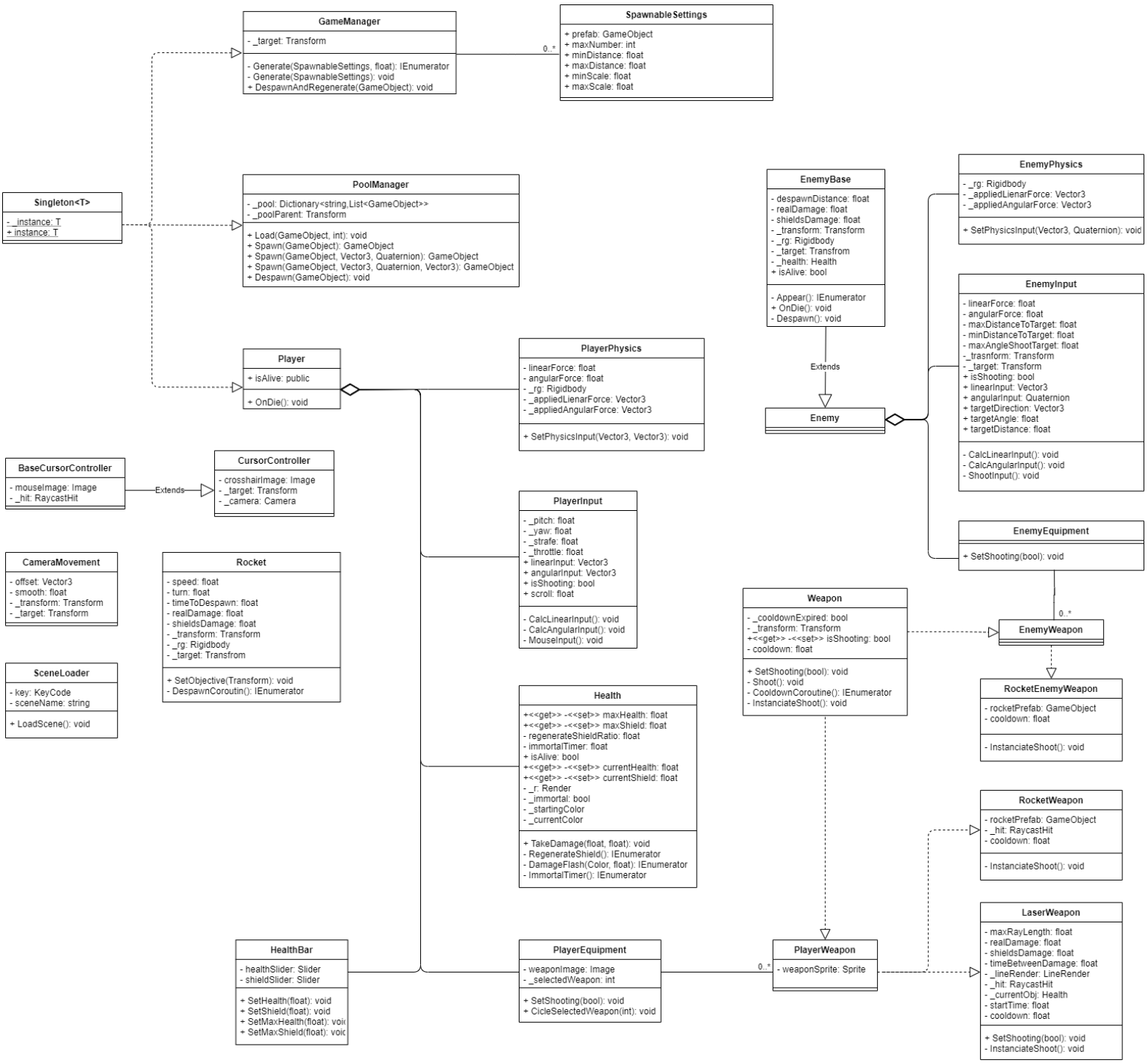
f. Scripts

En el paquete “Scripts”, como su propio nombre indica, almacenamos todo el código del proyecto. Dentro de esta carpeta tenemos ordenados los diferentes scripts según su categoría y a que parte del sistema hacen referencia. Entrando más en detalle:

- La raíz de la carpeta contiene elementos generales, como son el GameManager, el PoolManager y el Singleton, así como otros que hacen referencia a más de un sistema.
- El paquete Camera, contiene el código encargado de controlar tanto la propia cámara como los diferentes tipos de cursores que se utilizarán.
- El paquete Enemies, contiene todo comportamiento relacionado con los enemigos, su manera de interactuar con el entorno y su lógica interna.
- El paquete Player, contiene todo comportamiento relacionado con el jugador, su manera de interactuar con el usuario, su lógica interna y su manera de interactuar con el entorno.
- El paquete UI, contiene el código con el que se interactuará desde una interfaz, en este caso únicamente contamos con la funcionalidad de cambiar entre escenas.
- El paquete Weapons, contiene toda la lógica relacionada con las armas y su manera de interactuar con el entorno, estar armas serán utilizadas tanto por el jugador, como de los enemigos.

4. Diseño de clases

A continuación, se mostrará un diagrama general de las clases y sus relaciones en UML para facilitar una visión global del proyecto. En los siguientes apartados entraremos a definir en detalle cada una de las clases y las relaciones que existen entre ellas.



a. Singleton

La clase Singleton tiene el objetivo de restringir la creación de objetos, de este modo las clases que requieren una única instancia extienden de esta. En nuestro caso serán las clases de GameManager, PoolManager y Player.

b. GameManager

El GameManager se encarga de controlar la instanciación y destrucción de los objetos haciendo uso del PoolManager.

Durante el ciclo de comienzo de la escena, esta clase se encarga de instanciar una serie de Assets con diversas propiedades, esta primera instancia viene definida por un Array de objetos SpawnableSettings.

Además, Esta clase se encarga de la destrucción y regeneración de los GameObjects. Cada objeto destruido, se regenera pasado un tiempo, de este modo, mantenemos estable el número de elementos en escena y generamos un bucle de juego estable.

c. PoolManager

El PoolManager es el encargado de instanciar y destruir los objetos de la escena. Para ello cuenta con un Array que nos permite reutilizar los elementos activándolos y desactivándolos según sea necesario. De este modo, en lugar de crear nuevos elementos, lo cual sería muy cotoso computacionalmente, reutilizamos los ya instanciados sin aumentar la carga en el sistema.

d. CursorController

El CursorController se encarga de controlar la posición de los diversos cursores que aparecen en pantalla.

Esta clase hereda de BaseCursorController, que es una clase utilizada para mantener el ratón dentro de los confines de la pantalla de juego, ocultarlo y asignarle una imagen que mantenga su posición con respecto al mismo. Además de esto, CursorController extiende su clase padre para encargarse de mantener la posición de una mirilla dinámica que ofrecerá al jugador una referencia sobre el punto al que se está dirigiendo.

e. CameraMovement

La clase CameraMovement es la encargada de mantener la posición de la cámara con respecto al jugador.

f. SceneLoader

La clase SceneLoader es la encargada de controlar la transición entre escenas.

g. Player

La clase Player es la encargada de controlar el comportamiento del jugador. Para ello se ha dividido en sus diferentes componentes individuales, de este modo contamos con:

- PlayerInput, encargada de recibir los inputs del usuario y transformarlos en los parámetros necesarios para ser usados en el resto de los componentes de Player.
- PlayerPhysics, encargada de recibir los parámetros de velocidad linear y angular procedentes del input y aplicarlas al Rigidbody del objeto.
- PlayerEquipment, encargada de recibir los parámetros relativos al equipamiento y las armas del jugador, y actuar en consecuencia.
- Health, se encarga de controlar la salud del jugador, así como la regeneración de los escudos.
- HealthBar, se encarga de mostrar en la interfaz el estado de la salud del jugador.

La propia clase Player se encarga de organizar y hacer de medidador entre los componentes.

h. Enemy

La clase Enemy se encarga de controlar los comportamientos de los diversos enemigos. De manera similar a la clase Player, se han dividido en sus diferentes componentes individuales, de este modo contamos con:

- EnemyInput, encargada de realizar los cálculos que definirán cual será comportamiento del enemigo según la posición del jugador y transformarlos en los parámetros necesarios para ser usados en el resto de los componentes de Enemy.
- EnemyPhysics, encargada de recibir los parámetros de velocidad linear y angular procedentes del input y aplicarlas al Rigidbody del objeto.
- EnemyEquipment, encargada de recibir los parámetros relativos al equipamiento y las armas del enemigo, y actuar en consecuencia.
- Health, se encarga de controlar la salud del enemigo, así como la regeneración de los escudos.

La propia clase Enemy se encarga de organizar y hacer de medidador entre los componentes.

Enemy, además, hereda de BaseEnemy, que se encarga de definir los comportamientos básicos de todo enemigo, es decir, aquel que no tenga ningún tipo de respuesta ante su entorno.

i. Weapon

Todas las armas del juego son en una implementación de la clase Weapon, esta constituye la base tanto para las armas utilizadas por los enemigos, como para las del jugador. Existe esta separación entre armas con el objetivo de encapsular los diferentes comportamientos entre sí, de este modo, aunque estos hagan uso de herramientas muy similares, nos evitamos que un enemigo haga uso de la lógica de las armas de un jugador y viceversa.

j. Rocket

La clase Rocket se encarga de controlar el comportamiento de los cohetes, estos además de el comportamiento base de control de colisiones, destrucción, velocidad y dirección, mantienen la referencia a un GameObject objetivo, y dirigen el cohete al mismo mientras este se mantenga activo.