Asteroids

Game Design Document

Gabriel de Oliveira Trindade

Fundamentos de Videojuegos
Departamento de Lenguajes y Sistemas Informáticos e Ingeniería de Software
Escuela Técnica Superior De Ingenieros Informáticos
Universidad Politécnica de Madrid
Septiembre 2024

1. Juego

El juego 2D, *Asteroids*, consiste en destruir unos meteoritos que atacan al jugador, representado como una nave. Apretando el espacio, el jugador puede defenderse disparando misiles que destrozan dichos meteoritos. Cada vez que elimina un asteroide, aumenta un punto su puntuación, que aparece siempre en el lado inferior de la pantalla. A medida que pasa el tiempo, el *spawn rate* de meteoritos va en aumento, incrementando la dificultad del juego.

Figura 1: Captura *in-game* del juego.

El juego está disponible en el enlace a mi repositorio de GitHub. El ejecutable está incluido en un comprimido zip en el apartado de *Releases*. El juego ha sido desarrollado en Unity 2022.3 y compilado en Linux para Windows y Linux (*I use arch btw...*).

2. Escenas

El juego solo cuenta con una escena principal, ilustrada en la figura 1, que cuenta con los GameObjects de la nave, el espacio, los meteoritos y las balas. En el siguiente apartado se detalla adecuadamente.

3. Assets y GameObjects

Dentro de la carpeta *Assets* contamos con las subcarpetas de *Prefabs*, *Scenes*, *Scripts* y *Sprites*.

En *Prefabs* están pre configurados las figuras 2D de la bala y el meteorito. La bala está compuesta por su script y un *Box Collider* para chocar contra los meteoritos. El meteorito a su vez cuenta con un *Rigidbody* que permite que caiga hacia abajo para atacar a la nave, y un *Sphere Collider* para que afecte a la nave, o sea afectada por la bala.

En *Scenes* solo está la escena principal del videojuego, que es donde se desarrolla toda la acción.

En *Scripts* se encuentra la lógica del juego, detallada en el apartado 5.

Y por último, en *Sprites* se encuentran las imágenes que se utilizan en el juego.

Toda esta estructura de carpetas y sus ficheros se encuentra en el repositorio de GitHub.

4. Mecánicas e Interacción

El jugador puede defenderse de los asteroides, que únicamente aparecen desde la parte superior de la pantalla, disparando mediante su pistola con la tecla espacio. El jugador puede desplazarse para esquivar dichos asteroides con las flechas del teclado o las teclas WASD.

5. Scripts

Contamos con cuatro scripts para poder implementar la lógica del juego:

- Bullet
- BulletPool
- EnemySpawner
- Ship

5.1. Bullet y BulletPool

El script de la bala, Bullet.cs, controla que esta solo tenga efecto dentro de los límites de la pantalla (no sume puntuación cuando choca con un asteroide que está fuera de la pantalla, ya sea porque no ha desaparecido todavía o no ha entrado en el juego aún), y aumenta la puntuación cuando una bala colisiona con el asteroide.

Se ha desarrollado el extra del Patrón de Diseño *Object Pooling* para la clase *Bullet*, en el fichero BulletPool.cs. Este recibe por el editor de Unity el *Prefab*, para poder instanciarlos solo él, y el acceso a la *pool* ha sido diseñado mediante el patrón *Singleton*, para asegurar que solo esta clase es capaz de generarlos mediante su método definido (*RequestBullet*).

5.2. EnemySpawner

El script de la aparición de asteroides, EnemySpawner.cs, determina en su función *Update* en que parte de la pantalla aparecer. También calcula la cantidad de apariciones por segundo, que va a aumentando cada vez que transcurre el tiempo.

5.3. Player

El script que controla la nave, Player.cs, contiene una variable estática que maneja la puntuación del usuario. Es actualizada por el script de la bala para poder mostrarlo en la interfaz de usuario.

En la función *Start* obtenemos los bordes de la pantalla donde se está jugando, y cogemos el componente *Rigidbody* para poder actualizar la posición de la nave.

En *FixedUpdate*, que está diseñada para hacer los cálculos de movimientos físicos más fiables para cada intervalo, tenemos la lógica de movimiento de la nave, que recoge del input recibido por teclado (ajustado al *deltaTime*) y multiplicado por factores de fuerza para hacerlo más veloz, y así rotar y desplazar la nave.

En la función *Update*, recreamos la sensación de espacio infinito, detectando si la nave supera los límites de la pantalla para poder teletransportarlo al límite contrario. Posteriormente, invoca balas de la *pool* si ha detectado que el usuario presiona el espacio.

Por último, en *OnCollisionEnter*, se detalla la lógica de choque con el meteorito. Si sucede, se reincia la puntuación y se comienza de nuevo.