GAME DESIGN DOCUMENT

ASTEROIDS

VERSIÓN DEL DOCUMENTO 1.0

María Cantón González
Alejandro San Juan Sanchez

21/09/2023

OBJETIVO DEL JUEGO

El juego consiste en una nave espacial cuyo objetivo es disparar y destruir todos los asteroides que se va encontrando en su camino para poder sobrevivir al mismo tiempo que va acumulando puntos.

Se trata de una adaptación del clásico videojuego de arcade lanzado por Atari en 1979.

ESCENAS

Se trata de un juego muy sencillo el cual está formado por una única escena, esta consiste en una vista en 2D del espacio exterior en el cual se desplazará una nave espacial.

También aparecerán otros elementos en la escena como los asteroides que tratarán de destruir la nave al colisionar con ella o las balas emitidas por la nave con el objetivo de defenderse para poder sobrevivir.

Además, en todo momento podremos visualizar un marcador que indicará los puntos obtenidos hasta ese momento durante el juego.

ASSETS Y GAMEOBJECT

1. SPACE

Se trata de la base del juego, sobre este objeto se situarán el resto de los componentes del juego como la nave o los asteroides.

2. SHIP

Es el elemento controlado por el usuario, podrá rotar sobre si mismo, avanzar en la dirección en la que apunta y disparar una ráfaga de balas. Si la nave colisiona con un asteroide se destruirá y volverá a comenzar el juego.

Mientras la nave acelera es posible que esta gire dándole de este modo completa libertad de movimiento al jugador. Es decir, las acciones que puede realizar la nave son independientes entre sí, es decir, el jugador podrá acelerar, girar y disparar de manera simultánea.

La aceleración siempre se aplicará en la dirección a la que apunta la nave.

3. METEOR (PREFAB)

Este elemento es un prefab, es decir, se trata de una plantilla a partir de la cual se podrán crear nuevas instancias del objeto en la escena. Cualquier edición que se realice sobre el prefab se verá reflejada inmediatamente en todas las instancias de este.

Dicho prefab consiste en un asteroide que irá cayendo linealmente de manera aleatoria desde la parte superior de la pantalla a una velocidad constante, la cual aumentará en función del tiempo que aguante el jugador sin ser eliminado.

Los asteroides no serán destruidos directamente al ser alcanzados por un disparo, en lugar de ello se dividirán en dos de un tamaño menor que al ser alcanzados por una bala serán destruidos por completo. Los asteroides no podrán colisionar entre sí.

También existirá otro tipo de objeto denominado miniMeteor, será un prefab idéntico al Meteor pero más pequeño que surge tras la colisión de una bala con un asteroide.

4. BULLET (PREFAB)

Al igual que el el GameObject anterior se trata de un prefab, a partir de una bala podremos obtener el resto.

Este objeto saldrá disparado en la dirección a la que apunte la nave, pudiendo disparar varias balas de manera seguida pulsando el botón indicado.

El objetivo de las balas será colisionar contra los asteroides con el fin de destruirlos.

MECANICAS

El jugador controlará una nave espacial la cual tendrá capacidad de rotación en ambos sentidos. Para moverse por el mapa, podrá acelerar y moverse en la dirección en la que esté orientada la nave.

El movimiento de la nave será inercial, es decir, esta no frenará inmediatamente al dejar de acelerar, sino que la velocidad en la dirección de movimiento ira disminuyendo poco a poco.

Por otro lado, el mapa es un espacio infinito por lo que al atravesar uno de sus limites se aparece por el límite opuesto.

Una de las habilidades principales de la nave es el disparo, al pulsar la tecla de disparo, comenzaran a salir de la nave una serie de balas que viajaran por el espacio hasta que colisiona con un asteroide o se sale del plano.

La nave no estará sola en el espacio, sino que hay asteroides que destrozan cualquier nave con la que impactan, la trayectoria de estos asteroides será una línea recta con una velocidad constante desde que aparecen hasta el momento del impacto. Estos asteroides irán apareciendo de manera automática en la escena y lo harán con mayor frecuencia según vaya avanzando el juego.

En la pantalla de juego podrá verse en todo momento la puntuación actual, reiniciándose esta en el momento en el que un asteroide colisiona con la nave y vuelve a comenzar el juego.

SCRIPTS

- ### PLAYER

  En este script se definirá el movimiento de la nave para ello se utilizarán las variables '*thrustForce*' y '*rotationSpeed*' que indicarán la fuerza de empuje y velocidad de rotación respectivamente. Para recibir el movimiento que el jugador desea proporcionarle a la nave se utilizará la función '*GetAxis*' de la librería *Input.*

  Le añadiremos fuerza a la nave utilizando la función *AddForce* en la dirección en la que apunta el vector (hacia la derecha ya que la nave está alineada a la derecha), multiplicado por el valor del empuje proporcionado por el jugador por el factor de fuerza para poder darle más o menos potencia al pulsar la tecla.

  Para el giro, con la función *Rotate* habrá que indicarle el vector sobre el que va a aplicar el giro y el ángulo de giro (valor que detectamos a través del teclado) multiplicado por el factor de rotación.

Por otro lado, en este script se instanciarán nuevos objetos de tipo bala, esto ocurrirá si el jugador pulsa la tecla 'Espacio', al pulsar esa tecla se creará un nuevo GameObject *'bullet*' llamando al método '*GetPooledObject*()' de la clase ObjectPool, con el fin de reciclar balas. Si la llamada a dicho método devuelve una bala, la bala se activará y saldrá disparada de la nave.

Este script también proporciona una sensación de espacio infinito, ya que cuando la nave sale de la pantalla por uno de los limites de esta, aparecerá por el opuesto. Para ello se han utilizado las variables *'xBorderLimit'* e *'yBorderLimit',* que definen los límites de la pantalla.

Finalmente el método '*OnCollisionEnter(Collision collision)',* compara con el método '*compareTag*' si la nave ha colisionado con algún tipo de asteroide y en el caso de ser así el juego comenzará de nuevo. Para ello la variable 'SCORE' que da la puntuación se reiniciará y cargará de nuevo la escena inicial con el método '*LoadScene*'.

- **BULLET**

Para configurar las balas utilizaremos dos parámetros que nos proporcionarán la velocidad y el tiempo de vida de esta, dichos parámetros serán las variables '*speed*' y '*maxLifeTime*' respectivamente.

Por un lado, la bala debe salir disparada siempre en una misma dirección, siendo la dirección de la nave la que dirija la dirección de la bala, para ello se utilizará el vector 'targetVector'.

En el método *Update*, se buscará "teletransportar" el objeto de tal manera que al ejecutar dicho método en cada frame se consiga una sensación de movimiento, para ello se utilizará el método *Translate* el cual necesitará un vector de translación (*'targetVector'* indicado anteriormente) y un factor de velocidad (variable '*speed*').

Por otro lado, el método *IsTheBulletOutOfGameArea()* devolverá un booleano que indicará si la bala está dentro de los limites de la pantalla, en el caso de que se encuentre fuera del limite la bala será desactivada.

Después nos encontramos con el método 'OnCollisionEnter(Collision collision)' que en este caso si una bala colisiona con un enemigo se llamará al método 'IncreaseScore()' para aumentar la puntuación y a continuación el asteroide se dividirá en dos asteroides más pequeños instanciados a partir del prefab 'miniMeteor' que saldrán disparados en direcciones opuestas. También se destruirá el meteorito con el cual ha impactado la bala inicialmente y se desactivará la bala.

También puede darse el caso de que la bala colisione con uno de los asteroides más pequeños, en este caso también aumentará el contador, de destruirá el mini asteroide y se desactivará la bala.

- **ENEMYSPAWNER**

El objetivo del script será generar asteroides, por tanto, utilizará el prefab de estos. Los asteroides serán generados en base a una ratio en el tiempo, para ello se utilizará la variable '*spawnRatePerMinute*' que indicará el número de asteroides que caerán por minuto.

Por otro lado, para añadir complejidad al juego, para que cada vez aparezcan más asteroides se utilizará la variable '*spawnRateIncrement*'.

Se generarán nuevos asteroides solamente si ha pasado el tiempo suficiente entre el tiempo actual y la generación del último asteroide. En el caso de que la condición anterior se cumpla, a partir de la función *Instantiate* pasando como parámetros el prefab del asteroide, la posición desde donde se lanzará el mismo y el giro obtendremos nuevos asteroides.

Para indicar desde donde generaremos los asteroides utilizaremos la variable 'spawnPosition', vector que utilizará un numero aleatorio comprendido entre los límites de la pantalla como componente x, y un valor fijo que permita no ver el origen del asteroide como componente y.

Finalmente, al igual que ocurría con las balas, los asteroides se autodestruirán tras pasar el tiempo indicado por la variable *'maxTimeLife'* a través de la función *Distroy* para evitar generar basura en la escena y optimizar el juego.

- ### OBJECTPOOL

  El objetivo de este script es utilizar el Object Pooling para reutilizar las balas que ya han sido usadas, es decir en vez de crear balas que luego serán destruidas simplemente la bala se desactivará una vez haya sido utilizada y cuando se necesite una bala de nuevo se reciclará una de las balas desactivadas y se activará de nuevo.

  Para ello tendremos una lista de GameObject donde se agruparán todos los objetos reciclados listos para volver a ser utilizados denominada '*pooledObjects*' y un entero '*amountToPool*' que indicará la cantidad de balas que reciclaremos.

  Al comenzar el juego se inicializará la lista y se instanciaran en ella partiendo del prefab de la bala, tantas balas como indique la variable *'amountToPool'*. Estos objetos no estarán activos en un inicio por tanto no se podrán visualizar en el juego.

  Por último, el método *GetPooledObject()*, recorrerá la lista de balas recicladas y devolverá una de ellas siempre y cuando no esté activa en la escena, si no devolverá null.

- ### PAUSECONTROL

  Este script controlará el menú de pausa del juego.

  Para ello partiremos de una variable booleana 'gameIsPaused' inicializada a falso y un GameObject 'pauseMenuCanvas' que será el menú de pausa propiamente dicho.

  Al iniciar el juego este menú no estará activo ya que así lo indicará la función 'SetActive', y durante el juego al pulsar la tecla *Esc*, este se pausará o reanudará dependiendo del estado en el que se encontrase.

  Para pausar el juego se utilizará el método *PauseGame(),* este método activará el menú con la función 'SetActive' y pondrá la variable '*gameIsPaused*' a true.

  Por otro lado para reanudar el juego se utilizará el método *ResumeGame()* totalmente opuesto al anterior, desactivará el menú y pondrá la variable *'gameIsPaused'* a false.

INTERACCION

El juego cuenta con unos controles muy sencillos a través de los cuales el usuario podrá manejar la nave.

Los controles son los siguientes:

- **Botones de Giro (Izquierda/Derecha ó A/D):** Los botones de giro harán que la nave gire sobre si misma y apunte en una dirección distinta. Al pulsar la tecla "Derecha/D" la nave girará en el sentido de las agujas del reloj, y al pulsar la tecla "Izquierda/A" esta girará en el sentido contrario.
- **Botones de Acelerar (Arriba/Abajo ó W/S):** Los botones de acelerar harán que la nave avance en la dirección en la que apunta. La tecla "Arriba/W" hará que la nave vaya hacia adelante, mientras que las teclas "Abajo/S" equivalen a la marcha atrás.
- **Botón de Disparo (Barra espaciadora):** Al pulsar la barra espaciadora se generará una bala que viajará en línea recta en la dirección en la que apunte la nave.

LINK

En el siguiente link se encuentra el repositorio software del videojuego, así como la demo ejecutable del mismo.

https://github.com/SanJuan28/Microgame.git