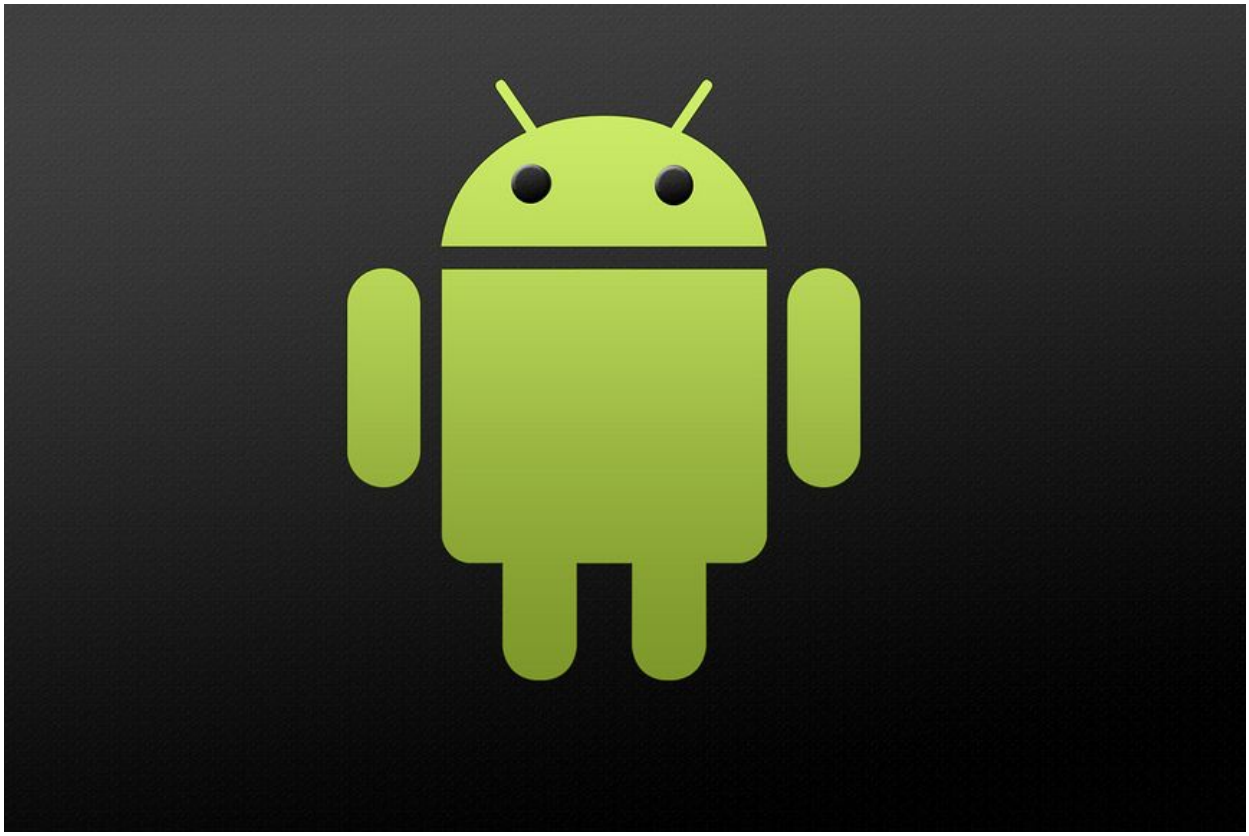

Práctica 3: Juego en Motor

12/2018

Sergio García Aloguín y Guillermo Amigó Urda

Desarrollo de aplicaciones para dispositivos móvil
Diseño y Desarrollo de Videojuegos
Universidad Rey Juan Carlos

Índice

Introducción y funcionamiento básico

Actualmente, el videojuego realizado se compone de un nivel y las respectivas ventanas para la ejecución de la aplicación: Menú principal, partida, menú de pausa y menú de conclusión, donde se muestran puntos y se puede reiniciar la partida.

El videojuego se compone de un personaje principal (elegible entre varios) que puede disparar proyectiles y usar power ups para derrotar a una serie de enemigos u objetos que intentarán hacer que el jugador pierda vidas. Si el jugador pierde sus vidas, perderá la partida, y en caso de eliminar obstáculos o enemigos, ganará puntos.

Se ha insertado una estética propia afín al videojuego.

La aplicación final consta de varias clases para el funcionamiento el juego, el sonido y la aplicación. A continuación presentaremos las más relevantes a la práctica:

- ***MainMenuFragment, GameEngine, GameFragment, EndGame y ScaffoldActivity***:Serán las encargadas principales de hacer funcionar el juego, siendo las pantallas o el motor donde se implementan métodos como añadir y quitar objetos de escena, o gestionar las **colisiones**.
- ***Contenedor de "space"***: Aquí contendremos todos los GameObjects que serán nuestros personajes, enemigos,balas y otros (items) que serán los objetos visibles dentro del juego.
- ***Clases auxiliares:*** Habrán numerosas clases para que el juego pueda ser llevado a cabo, que se han generado en clase y sirven para tener un control del jugador, o que el motor pinte en escena. Tendrán pocos cambios rspecto a su creación original.
- **SoundManager,ParallaxBackground**: Clases nuevas para ampliar funcionalidades, añadiendo sonido, musica, y fondos interactivos al juego.

Requisitos mínimos

Menú principal

Figura 1: Menú principal

Se ha recuperado los menús que proveía la aplicación original desarrollada en clase, cambiando el aspecto de sus botones y su texto, y añadiendo botones para poder controlar si queremos escuchar los sonidos y la música del juego.

Nivel

Figura 2: Ejecución de una partida

- ***Información en pantalla durante el juego:*** Existen cuatro variables que serán visibles al jugador en todo momento
    - **Número de vidas restantes**
    - **Número de bombas restantes**
    - **Número de naves asesinadas**
    - **Número de puntuación actual**

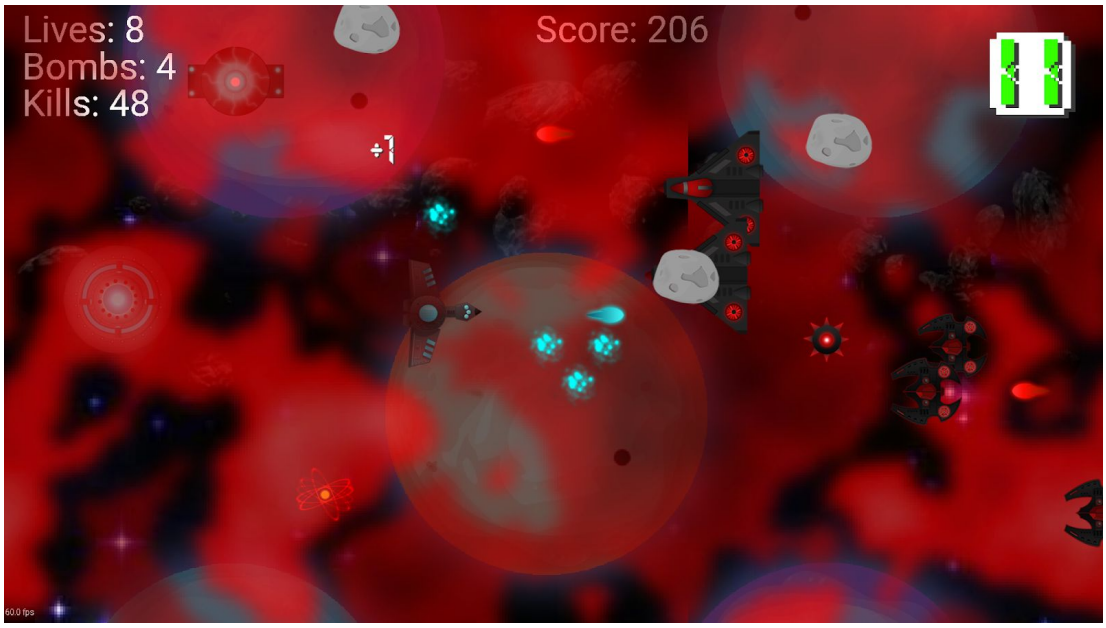

Figura 3: Ejecución de una partida durante el modo Rampage

- ***EXTRA: Modo Rampage:*** Cuando se llega a una puntuación de 200, el juego se hace más difícil, de forma dinámica, se crea:
    - Un nuevo parallax background encima del jugador para entorpecer su vista.
    - Se añaden dos nuevos SmartEnemies
    - Se añaden dos nuevos asteroides.
    - Se añaden dos nuevos Enemies.
    - Y, además las velocidades de algunas naves aumentan para que cada vez el juego sea más difícil.

Menú de pausa

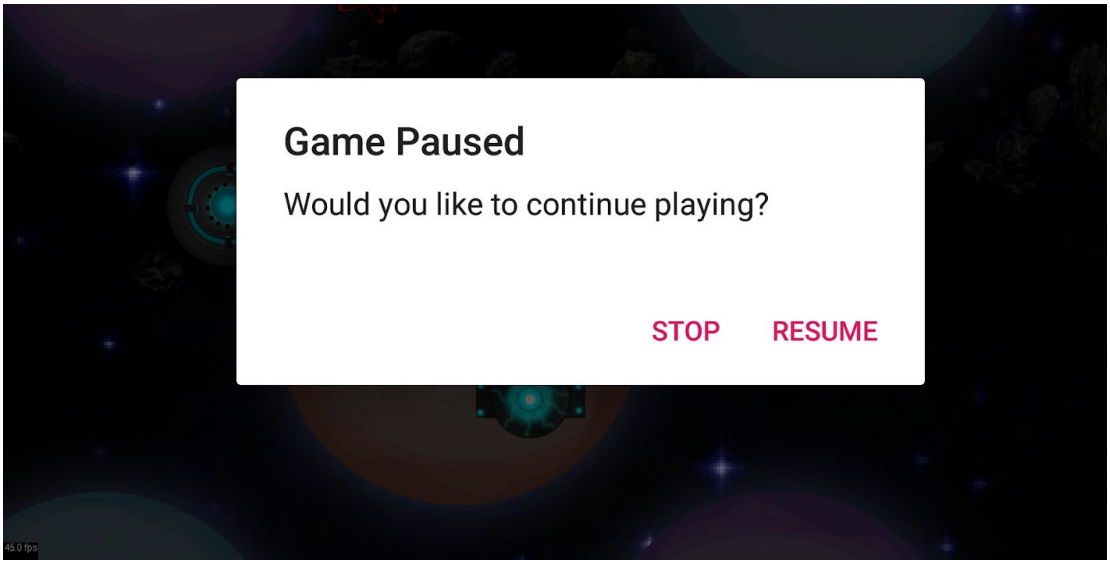

Figura 4: Menú de pausa

Mientras estemos jugando una partida, podemos pulsar el botón de pausa y paralizar el juego.

Desde este menú de pausa se nos va a permitir volver a la selección de personaje para jugar una partida nueva, o continuar la partida que estamos jugando actualmente.

Pantalla de fin de partida

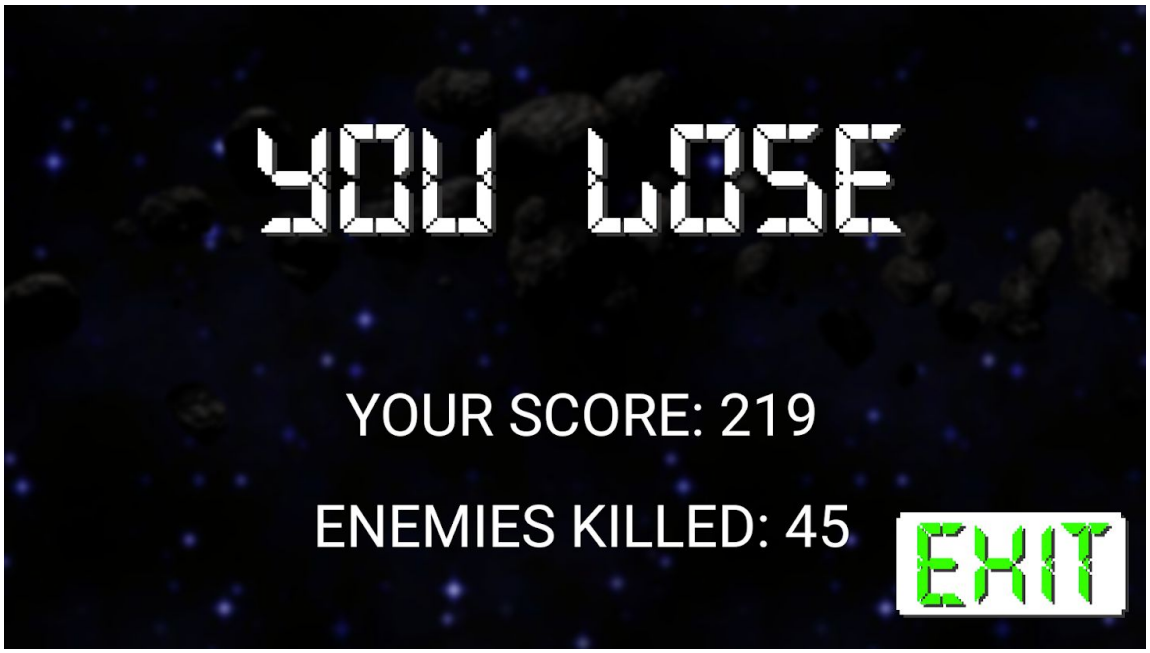

Figura 5: Menú fin de partida

Esta es la actividad para acabar el juego (EndGame), donde se recoge puntuación del jugador y se muestra por pantalla, además también de mostrar el número de enemigos que hemos destruido.

Por último, también nos permite volver a la primera actividad y jugar otra partida.

Controles

Figura 6: Tutorial de los controles de una partida

- **Joystick de movimiento.** Los controles del juego serán una versión modificada ligeramente del JoystickInput Controller, que ofrecía una solución buena para juegos de índole móvil, ya que no ocupan espacio en la pantalla haciéndolos cómodos.

- **Botón de disparo.** Además, si pulsamos en la parte izquierda de la pantalla, dispararemos una bomba triple de la que hablaremos de forma más detallada en esta memoria.

Enemigos

Se han implementado diversos enemigos, en las clases "**Enemy**, **Asteroid** y **SmartEnemy**", que tendrán comportamientos propios y cuyo objetivo será intentar hacer daño al jugador.

- **Enemy**: Enemigo que se mueve y dispara aleatoriamente.
- **Asteroid**: Asteroide que viaja por el mapa en una dirección y desaparece al ser destruido o salir de la escena.
- **SmartEnemy**: Enemigo que dispara apuntando al jugador, y que además tiene 2 vidas.

Todos estos enemigos se gestionan mediante un **EnemySpawner**, que se ocupará de generar dinámicamente enemigos por medio de una Pool para obstruir el avance del jugador.

Mas adelante se explicará de forma más detallada estos enemigos.

Colisiones

Para el desarrollo del disparo y de los enemigos, se ha tenido que implementar un sistema de colisiones. Se ha adherido a los **Sprites** un objeto de la clase **Rect**, que funcionará como una colisión **AABB** gracias al método de los objetos Rect "**intersect**()". Cada Sprite tendrá un método "**OnCollision**()", como se puede ver en cualquier motor de juegos, que permite que cada **GameObject** tenga el comportamiento necesitado al **colisionar**. También tenemos variables para designar el **tipo de Sprite** y ayudarnos a implementar cosas tales como colisión entre enemigos, con asteroides, etc.

Desafíos de plata

Diseño

Figura 7: Estilo visual de letreros y botones

Para dotar a la aplicación de una personalidad única, se ha incorporado un icono de aplicación, como Sprites propios, fondos y demás.

Figura 8: Icono de la aplicación

Disparo múltiple

Existen dos tipos de disparo:

Figura 9: AutoBullet

- **AutoBullet**
  - Es un disparo simple que se mueve de forma horizontal, que se dispara automáticamente cada cierto tiempo.

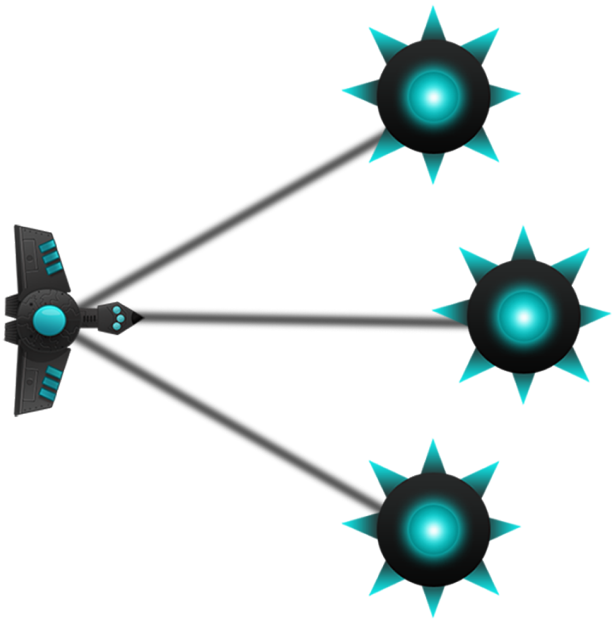

Figura 10: Disparo triple

- **Bombs (Disparo triple)**
  - Es un disparo triple, que al pulsar la mitad izquierda de la pantalla, spawneará tres bombas como las que vemos, creando un gran daño de forma radial.

Varios tipos de naves

Figura 11: Menú de selección de nave

Antes de comenzar una partida, se permitirá usar una serie de naves, a elegir entre cuatro de ellas. Esto se realiza creando una nave con uno de 4 diferentes sprites en la actividad **SelectCharacterFragment**(), clase por la que se pasa antes de empezar la partida.

Desafíos de Oro

Se ha desarrollado el desafíos de oro disponible, siendo explicado a continuación.

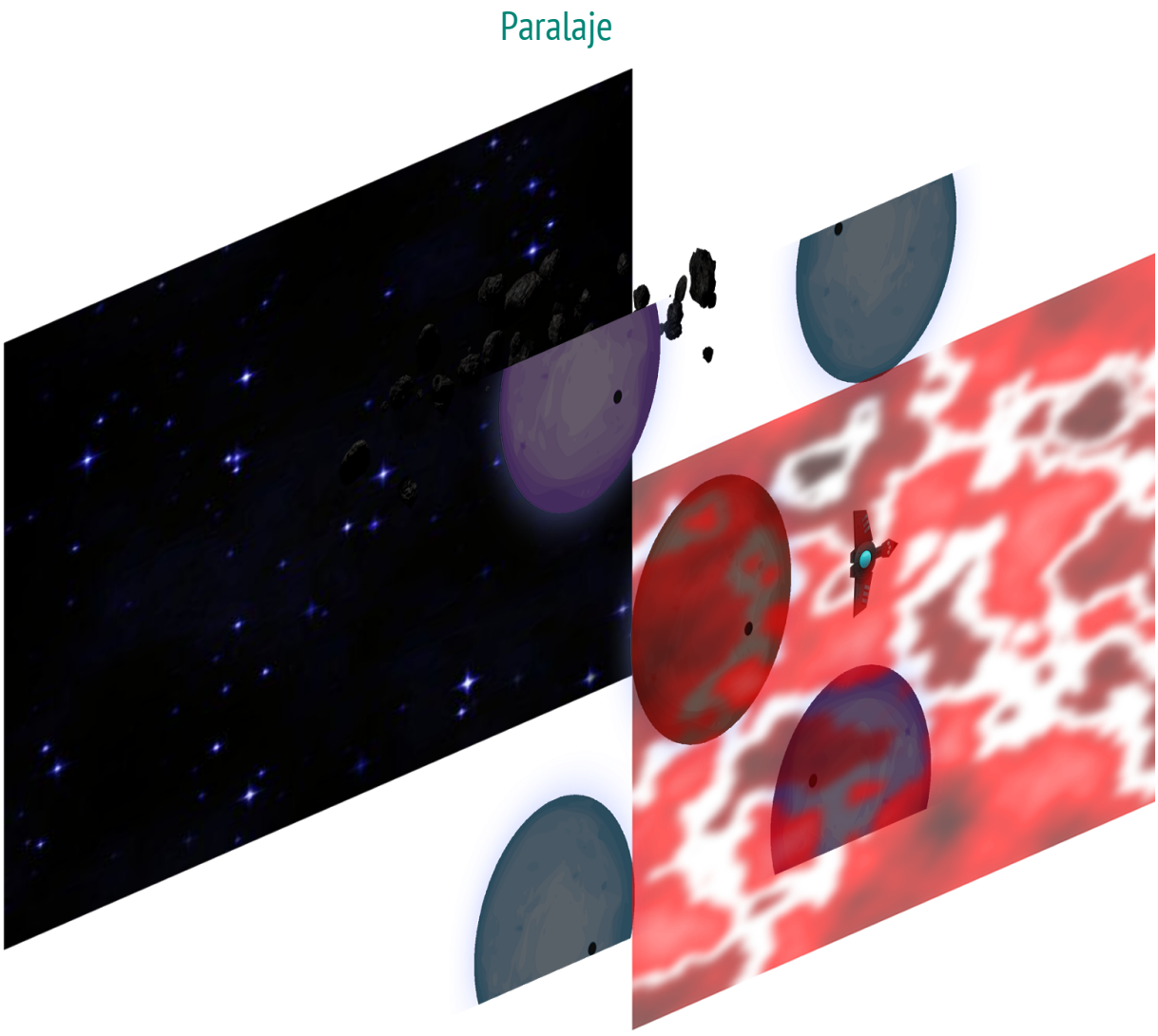

Figura 12: Desglose de las 5 capas del parallax

Gracias a la clase **ParallaxBackground**, tendremos un paralaje lateral en juego. Esto serán sprites en movimiento en la pantalla, guardados en una matriz y movidos en el método **OnDraw()**.

Múltiples Enemigos

El **videojuego** consta de una serie de **enemigos**, principalmente **dos**. Aunque han sido explicados previamente aquí se hará una explicación más exhaustiva de ellos:

Figura 13: Enemy

- **Enemigo normal:** En la clase **Enemy** y usando las balas de la clase **bulletEnemy,** funcionarán como el jugador y su **autodisparo**, generando balas cada unidad de tiempo. Estas balas se generarán en direcciones completamente aleatorias, así como el movimiento del enemigo, generados por Randoms. En su colisión se destruyen, y atraviesan meteoritos.

Figura 14: SmartEnemy

- **Enemigo inteligente:** En la clase **SmartEnemy** y usando **SmartBullet** como sus balas, estas clases compondrán un enemigo cuyas balas van en la **última dirección conocida del jugador**, como si tuvieran **inteligencia propia** para apuntar y disparar al jugador. En su colisión se destruyen, y atraviesan

meteoritos. Además, este enemigo dispone de dos vidas, por lo tanto tendremos que dispararle dos veces para poder matarlo.

Figura 15: Asteroid

- Cabe mencionar los **meteoritos**, que aunque no son un enemigo, sí son un obstáculo que puede afectar al jugador. Estos meteoritos viajan **aleatoriamente** por el mapa y pueden rebotar entre sí (método **Ricochet**()). Serán destruidos si el jugador los dispara.

Ítems

Se han creado tres ítems diferentes que el jugador podrá coger para beneficiarse.

Estos tres ítems heredan de la clase padre “Item”, cuyo funcionamiento es siempre el mismo: spawnean en un punto aleatorio del mapa, y cuando son cogidos por el jugador, vuelven a aparecer en otro punto aleatorio.

Estos son los 3 ítems existentes y su comportamiento al detectar que el usuario los ha cogido:

Figura 16: Recarga de bombas

- **Recarga de bombas:** El usuario solo puede tener hasta un máximo de 4 bombas en su inventario. Cuando el usuario coge este objeto, le otorga nuevas bombas hasta un máximo de cuatro.

Figura 17: Duplicador de disparos por segundo

- **Duplicador de disparos por segundo:** ofrece la capacidad al recogerlo de dar disparo más rápido al jugador, reduciendo a la mitad el tiempo entre disparos, durante 10 segundos.

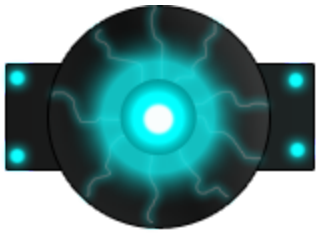

Figura 18: Restaurador de vida

- **Restaurador de vida:** Un jugador tiene 10 vidas. Cuando recoge este objeto, le restaura todas las vidas hasta las diez.

Efectos de sonido y música

Gracias al libro *Mastering Android Game Development*, se ha implementado un sistema de **sonido**, que permite introducir efectos de **sonido**, **música** y activarla o desactivarla en el juego.

Los **sonidos** se generan en eventos (objetos **GameEvents**) y tendrán asociado sonidos. La música será solo una que se reproducirá durante la ejecución del juego.

La clase principal es SoundManager, que será usada en **ScaffoldActivity** para inicializar y controlar todo el sonido de la aplicación. En SoundManager, cargaremos los sonidos, y los ejecutaremos cuando un **GameEvent** lo pida.

Los **gameEvents** se lanzarán desde cualquier clase, en nuestro caso las llamarán los Sprites al colisionar con otros Sprites (disparos con enemigos, etc.)

El juego tendrá dos botones al principio para hacer mute sonido o música de forma independiente, a fin de poder jugar con ninguna, una o dos de las opciones.

EXTRA: Explosiones

En todo videojuego, el feedback es muy importante, es por eso que se ha implementado un Sprite Explosión, que renderiza un sprite en una posición dada durante unos segundos.

Toma como argumentos un drawable, y una posición x e y, de esta manera siendo muy personalizable.

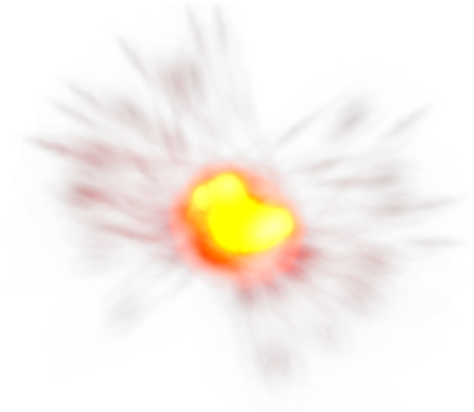

Figura 19: Explosión de nave

- Una explosión cuando destruimos una nave. Es más grande que el resto, para que el jugador sepa efectivamente que hemos destruido una nave.

Figura 20: Destrucción de bala enemiga

- Destrucción de bala enemiga

Figura 21: Impacto en nave

- Daño en nuestra nave

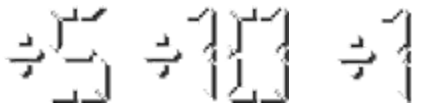

Figura 22: Indicadores de puntuación obtenida

- Indicador de la puntuación obtenida

Conclusiones

Gracias a las clases y el libro *Mastering Android Game Development*, se ha podido realizar completamente la práctica propuesta, que ha servido para aprender sobre cómo funciona un juego por dentro, así como implementaciones, consejos de programación en videojuegos y sobre todo en conceptos sobre videojuegos en móvil.

Una preocupación que quizás se ha quedado más sin tratar es el tema del rendimiento, ya que se ha tenido que dedicar gran tiempo a mejorar el rendimiento de la práctica.

En resumen, gracias a los conocimientos adquiridos en la asignatura se puede decir que somos capaces de crear un videojuego de 0 para Android.

Bibliografía

https://developer.android.com/docs/

https://stackoverflow.com/

Mastering Android Game Development

Apuntes de clase