Desarrollo de Aplicaciones para Dispositivos Móviles

Práctica 3

Grado en Diseño y Desarrollo de Videojuegos

Curso 2018/2019

Andrés Felipe García

Raquel Gastón Vicente

Índice

Introducción

En esta memoria se explicarán los aspectos más importantes acerca de la implementación de la Práctica 3 de la asignatura, consistente en la utilización del motor de juego desarrollado en clase para el desarrollo de un sencillo *shoot'em up.* Se repasarán los puntos esenciales de la realización de la práctica, tanto la ampliación del motor como la programación de la lógica del videojuego, justificando la toma de decisiones en cada caso.

Descripción general

La aplicación se trata de un juego arcade en el que se pescan tiburones desde una barquita, disparando para ello arpones. Utiliza como esqueleto la estructura básica de un *shoot'em up* clásico, al estilo del "juego de los marcianitos". En este caso, el jugador controla una barca que avanza por el océano disparando arpones a los tiburones, que avanzan en sentido contrario, y tratando de esquivarlos para no perder salud. La partida termina cuando han aparecido un número determinado de tiburones, o bien cuando el jugador pierde toda su salud.

La aplicación está enteramente traducida al inglés y al español.

Requisitos y desafíos

Requisitos mínimos

La aplicación se basa enteramente en el motor de Android explicado en el libro *Mastering Android Game Development* (Portales, 2015). Al iniciarse, da la bienvenida con un menú principal, que muestra el logo del juego y un botón para comenzar la partida. Asimismo, cuenta con la pantalla de juego, que constituye un nivel completo; y una pantalla de fin de partida, en la cual se muestra al jugador si ha ganado o perdido, el tiempo que ha durado su partida y la puntuación obtenida. Además, cuenta con un menú de pausa, accesible desde un botón en la pantalla, o al pulsar el botón de atrás del dispositivo.

La barca controlada por el jugador se controla mediante un joystick virtual invisible, situado en la región izquierda de la pantalla. En la mitad derecha se sitúa un botón, que como se verá más adelante, sirve para realizar un disparo especial.

Existe un tipo de enemigo, que es el tiburón. Los tiburones se generan en la clase `GameController`, con un punto de *spawn* en la parte central superior de la pantalla, y se mueven con un ángulo que varía en el rango de los -30 a 30 grados, es decir, cada uno se mueve en un sentido, pero siempre avanzando hacia abajo. Cuando un tiburón toca la barca, se pierde un 10% de la vida. Si se llega a 0, el jugador pierde la partida. Por su parte, los arpones que dispara la barca eliminan de un solo golpe a los tiburones.

A fin de llevar a cabo las funciones comentadas y otras que se explicarán más adelante (como, por ejemplo, los *power-ups),* los elementos del juego tendrán un sistema de colisiones radiales. Es decir, detectan colisiones en un área circular de un tamaño aproximado al de su *sprite.*

La puntuación del jugador se incrementa por cada enemigo eliminado. En principio, los puntos se suman de uno en uno, pero hay un *power-up* que incrementa ese valor.

En el estado actual de la aplicación, el número de enemigos que tienen que generarse para finalizar la partida con éxito es de 50. Este valor se puede modificar mediante la constante `MAX_ENEMIES` de la clase `GameController`.

Otro de los requisitos básicos que cumple la aplicación es la interfaz gráfica durante la partida. Aparte del *framerate* al que corre el juego, visible en la esquina inferior izquierda, se muestran la salud del jugador (en porcentaje) y la puntuación obtenida, en la esquina superior izquierda.

Desafíos

De cara a implementar los desafíos de la práctica, se ha comenzado por establecer un disparo automático, siguiendo el estilo de los juegos arcade clásicos: la barca dispara un arpón hacia delante cada 0,5 segundos. El botón de disparo de los controles se utiliza ahora para un disparo especial, consistente en tres arpones, cada uno en una dirección (a 30 grados entre ellos). Este disparo solo se puede realizar una vez cada 2 segundos.

Además, se ha creado una pantalla intermedia (fragmento `SpaceShipSelectorFragment`) entre la de inicio y la de juego, en la cual se da al jugador la opción de elegir una barca entre distintos diseños. El *sprite* de la barca durante la partida dependerá de esta selección.

Para terminar con los desafíos plata, se ha personalizado la temática y estética de toda la aplicación. Por un lado, se ha cambiado la temática espacial, con naves y asteroides, por una aventura por el océano, con peces y tiburones; es decir, se han renovado todos los *sprites*. Por otro lado, el diseño de la aplicación se ha modificado acorde: los colores del tema son azules, se ha cambiado la fuente y los botones tienen las esquinas redondeadas.

En cuanto a los desafíos oro, los dos elegidos para optar al sobresaliente son los siguientes: paralaje e ítems que proporcionan poderes.

La paralaje se ha implementado, siguiendo lo explicado en el libro, en la clase `ParallaxBackground`, que extiende a `GameObject`. Los elementos que se han incluido en el juego con efecto de paralaje han sido: un fondo de agua, una imagen con peces que nadan, y otra capa de agua con mayor transparencia para simular olas y cáusticas. Cada una de estas capas se mueve a diferente velocidad, simulando efecto de profundidad en la escena.

Los *power-ups* son objetos de la clase `Item`, que extiende a `Sprite`, al igual que la barca, los arpones o los tiburones. Se han añadido ítems de tres tipos, mediante el uso del enumerado `ItemType`:

- **Corazón** (`ItemType.HealthUp`): reinicia la salud del jugador al 100%.
- **Estrella** (`ItemType.Immortal`): hace al jugador invulnerable durante 5 segundos (valor almacenado en la constante `IMMORTAL_TIME` de la clase `SpaceShipPlayer`).
- **Arpón arco iris** (`ItemType.PowerGun`): incrementa la puntuación obtenida por enemigo eliminado.

Técnicas elegidas

Toda la aplicación se ha implementado en una sola actividad, utilizando fragmentos para las diferentes pantallas, y se ha gestionado la navegación entre estos y el comportamiento del botón de atrás.

- `MainMenuFragment`: es el menú principal. Muestra el título del juego y un botón para comenzar la partida (Figura 1).

Figura 1

- `SpaceShipSelectorFragment`: es una pantalla intermedia para seleccionar la apariencia de la barca (Figura 2). Muestra tres *sprites* y al pulsar sobre uno, envía a la pantalla siguiente el *id* de la imagen elegida. Esto se hace mediante el paso del dato en un *bundle* en la función `startGame` de la actividad.

Figura 2

- `GameFragment`: es la pantalla en la que tiene lugar el juego (Figura 3). Al crearse, inicializa el `GameEngine`, y le va añadiendo los objetos necesarios, atendiendo al orden en el que deben renderizarse.
  En el caso de los elementos con paralaje, se han elegido velocidades más bajas para aquellos que se encuentren a mayor profundidad. La imagen con los peces es una excepción, ya que para simular que van nadando, tienen una velocidad mayor que el agua. El paralaje (`ParallaxBackground`) está implementado utilizando la función de dibujado eficiente descrita en el libro.

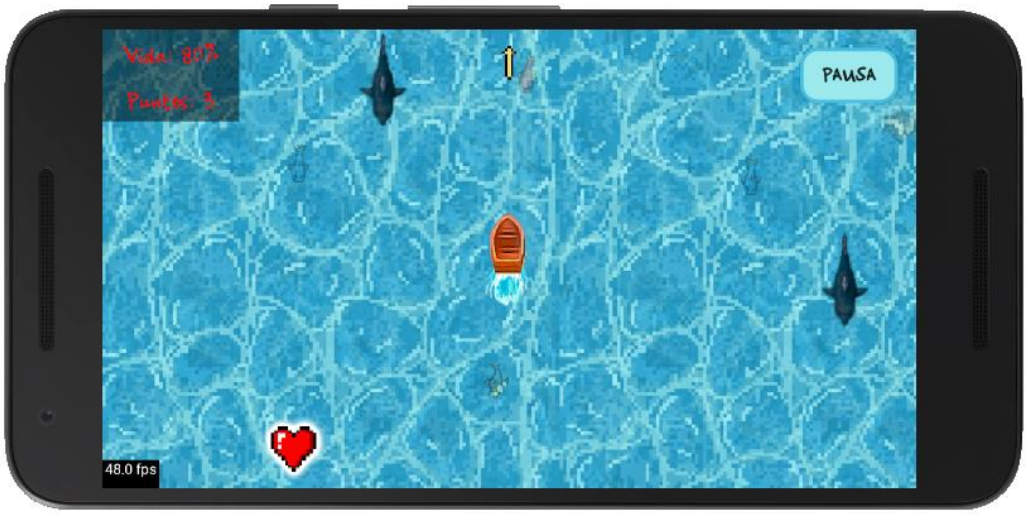

Figura 3

Aparte de objetos del propio juego, se inicializan los elementos de la interfaz: contador de *frames* (`FramesPerSecondCounter`), contador de salud (`PlayerHealthUI`) y contador de puntuación (`PlayerScoreUI`). Estas clases cuentan con una instancia del objeto barca (`SpaceShipPlayer`) para acceder a sus datos en todo momento, y hacen uso de la clase `Canvas` de Android para pintar por pantalla.

En esta pantalla hay un botón de pausa (Figura 4), mediante el cual se puede abandonar la partida.

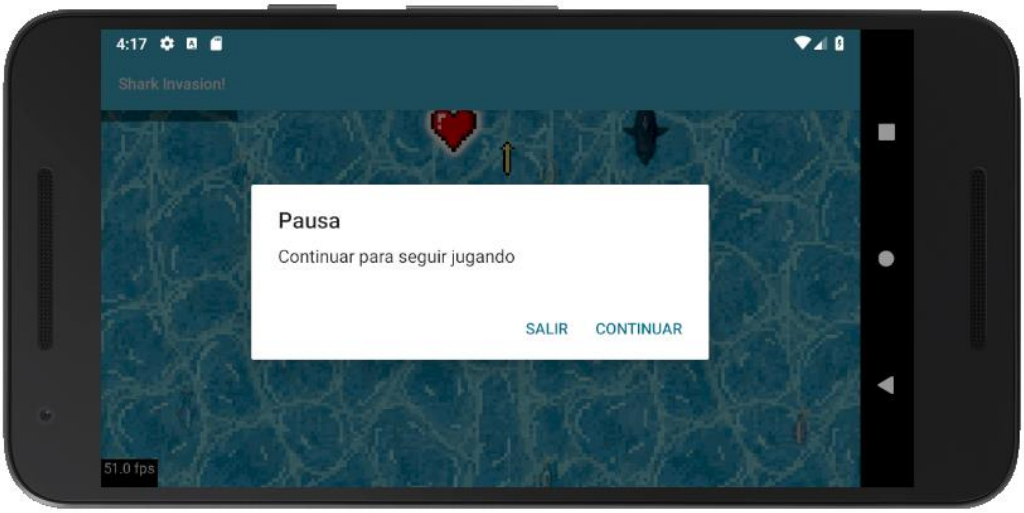

Figura 4

Más adelante se entrará en profundidad acerca de los objetos empleados en la partida.

- `EndOfGameFragment`: es la pantalla que se muestra tras finalizar la partida. Al crearse, recibe un *bundle* con los datos necesarios, a saber: si el jugador ha ganado (Figura 5) o perdido (Figura 6), la puntuación obtenida y el tiempo transcurrido. También cuenta con un botón para volver al menú principal.

Figura 5

Figura 6

Antes de repasar los objetos usados en el juego, se explicará brevemente la jerarquía de clases que se ha utilizado (Diagrama 1).

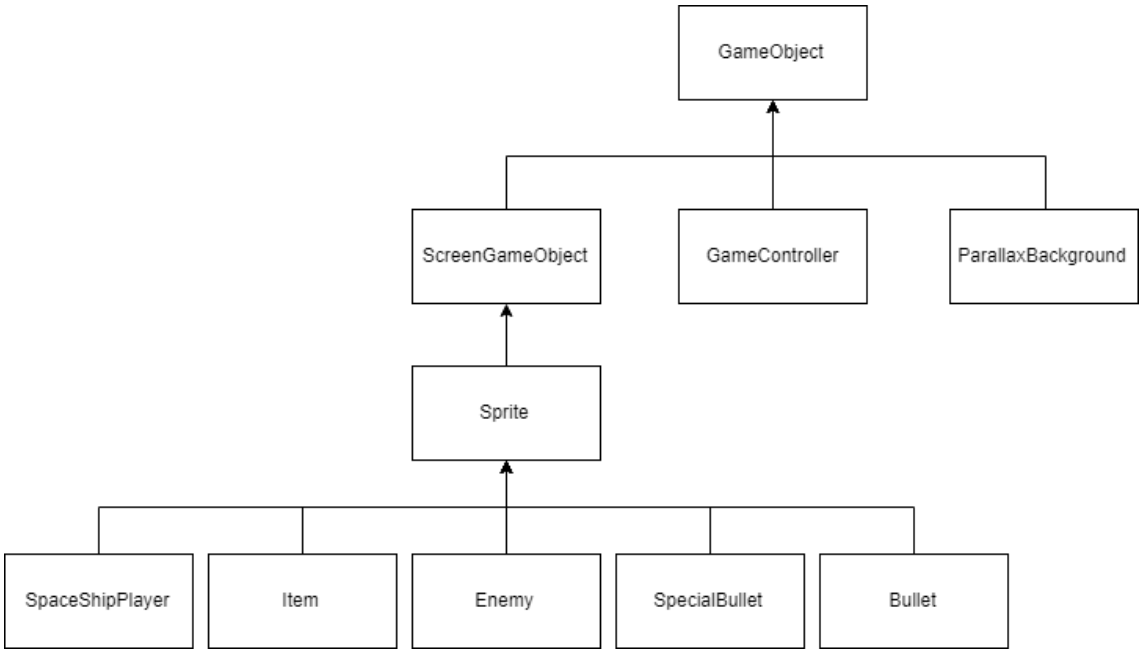

Diagrama 1

El primer dato a tener en cuenta es que se ha creado una clase `ScreenGameObject` que extiende a `GameObject` para aquellos objetos que aparezcan en pantalla, a diferencia de `ParallaxBackground` (clase utilizada para el paralaje) y `GameController` (clase utilizada para generar enemigos e ítems y gestionarlos). Cada `ScreenGameObject` tiene un área de colisión de forma circular, cuyo diámetro se corresponde con el valor máximo entre su alto y su ancho.

La clase Sprite extiende a `ScreenGameObject` y es utilizada para elementos con representación gráfica en pantalla. Esto nos da la oportunidad de crear elementos que se encuentren en el escenario, pero sean invisibles, como zonas donde se disparan *triggers*. En el caso de nuestra aplicación, no existe ninguno con estas características, pero ya está preparada para su posible implementación.

Los objetos que heredan de la clase Sprite en nuestro juego son:

- `SpaceShipPlayer`: es la barca, controlada por el usuario. Como atributos, destacan la salud y la puntuación, variables públicas para su acceso desde las clases de la interfaz, ya que el libro desaconseja el uso de *getters* y *setters.* Contiene una piscina de arpones normales (`Bullet`) y otra de arpones especiales (`SpecialBullet`), mediante las cuales se genera un número limitado de objetos que se reutilizarán a lo largo de la partida, evitando instanciarlos y destruirlos repetidamente.
  La función `onCollision` se encarga de restar puntos de salud cuando tiene lugar una colisión con un enemigo.
- `Item`: son creados en el `GameController` y su tipo, definido mediante el enumerado `ItemType`, se elige aleatoriamente en el momento de su creación desde la clase `GameController`. Como se ha mencionado antes, pueden ser de tres tipos (Figura 7). El método `onCollision` llama a una función u otra de la clase `SpaceShipPlayer` en función de dicho tipo.

- `Enemy`: son los tiburones, enemigos del jugador. Esta clase, al igual que `Item`, `Bullet` y `SpecialBullet`, sigue el patrón de las piscinas de objetos.
- `Bullet`: se utiliza para los arpones normales de la barca, es decir, los que se disparan de manera automática. En su implementación de la función `onCollision`, comprueba si se colisiona contra un enemigo, y en caso afirmativo, lo elimina, se elimina a sí misma (volviendo a la piscina) y aumenta la puntuación del jugador.
- `SpecialBullet`: funciona de manera similar a `Bullet`, con la diferencia de que se mueve tanto en X como en Y y su *sprite* está rotado.

Para terminar, cabe mencionar que el *joystick* y el botón de disparo se han implementado como se ha visto en clase: mediante el uso del *layout* `view_joystick`. Se detecta dónde se ha tocado la pantalla y hacia dónde se ha desplazado el dedo.

Figura 7

Conclusiones

Esta práctica ha resultado muy útil para controlar la noción de fragmento en el contexto de una aplicación de Android. Por su parte, el desarrollo de un motor de juego puede resultar reinventar la rueda, en especial existiendo motores muy potentes, como por ejemplo Unity. No obstante, este ejercicio ha reforzado nuestros conocimientos acerca de programación de videojuegos en general, y nos ha servido para mejorar nuestras prácticas sobre encapsulación y patrones.

Referencias

dafont.com. [En línea] https://www.dafont.com/es/ampersand.font.

opengameart.org. [En línea] https://opengameart.org/content/heart-pixel-art.

opengameart.org. [En línea] https://opengameart.org/content/fish-sprite-sheet.

rpgmakerweb.com. [En línea] https://forums.rpgmakerweb.com/index.php?threads/looking-for-sea-mammal-sprites.67753/.

stackoverflow.com. [En línea] https://stackoverflow.com/questions/11483345/how-do-android-screen-coordinates-work.

android.com. [En línea] https://developer.android.com/guide/components/fragments?hl=es-419.

aldominium. *YouTube.* [En línea] https://www.youtube.com/watch?v=1OP1jLxgsrA.

Garcia, Fran. *YouTube.* [En línea] https://www.youtube.com/watch?v=K3HlCvlNkbI.

Matei, Sergiu. *ArtStation.* [En línea] https://www.artstation.com/artwork/LWYvk.

Portales, Raul. 2015. *Mastering Android Game Development.* s.l. : Packt Publishing Ltd., 2015.