Memoria Práctica 1: Switch Dash

Sergio J. Higuera Velasco && Pablo Gómez Calvo

1. Explicación del juego.

Switch Dash es un juego disponible en muchos sitios web de juegos de navegador. El juego consiste en aguantar el mayor tiempo posible sin morir. La mecánica es sencilla, en la parte superior de la pantalla van a ir apareciendo pelotas de color negro o blanco y en la parte inferior hay de una barra que será de uno de esos dos colores antes mencionados. El juego consiste en cambiar de color la barra de la parte inferior de tal manera que cuando las bolas choquen con la barra, ambos sean del mismo color. Si esto ocurre sumaremos puntos y si, por ejemplo, cocha una bola blanca con una barra negra, perderemos la partida.

Para añadir una progresión al juego, según va pasando el tiempo, la velocidad de caída de las bolas va aumentando de manera progresiva.

Para realizar esta práctica se ha utilizado **Android Studio**. Dentro del mismo proyecto, se ha dividido la arquitectura del código de tal manera que este proyecto funcione tanto en un dispositivo móvil como en un equipo de sobremesa.

2. Arquitectura del proyecto.

Para dividir ambos tipos de compilación se han utilizado 2 inicializadores los cuales son ejecutados en función de la plataforma en la que nos encontremos. Para Android se ha creado una actividad y desde ahí se ha inicializado tanto la lógica del juego como el motor que depende de la plataforma. En el caso de PC, lo que se ha hecho es una clase que contiene un único método **main()** que también inicializa la lógica del juego y el motor de la plataforma para la que estamos ejecutando.

Como ya se ha dicho, los inicializadores cargan el motor del juego en función de la plataforma en la que nos encontremos. Ambos motores tienen unas clases comunes que vienen dadas por el módulo **Interfaces**. Este módulo es el que abstrae a la **Lógica** de la plataforma para la que se está ejecutando el juego. Estas interfaces comunes son funcionalidades que todo juego necesita como son:

- **Implementación de Input:** Esta labor cambia bastante con respecto a la plataforma, pero en ambos casos esta parte del motor maneja la captura del Input del jugador. En caso de PC es con ratón y en caso de Android es sobrecargando la función **onTouch** de la plataforma.
- **Tratado de imágenes:** Este apartado es cubierto por las clases de **Graphics** y **Image**. En Android se usa una implementación que utiliza los Bitmaps y los Canvas que las librerías de Android nos proporcionan. En el caso de la implementación de PC, usamos las imágenes de AWT y Swing que usa el lenguaje nativo de java. Aquí se introduce un nuevo actor que es el echo de tener que redimensionar las imágenes ya que los ordenadores y sobre todo los teléfonos móviles tienen resoluciones muy dispares. Para esto lo que se ha hecho ha sido aislar a la lógica y a cualquier futuro usuario del motor sobre los problemas que conlleva escalar una imagen o unas coordenadas, es por eso que la **Lógica** únicamente conoce la existencia de las interfaces, pero las implementaciones de estas llaman a un "redimensionador" de coordenadas por debajo para posteriormente devolver las coordenadas correctas del juego en cuanto a lo que lógica se refiere. Por poner un ejemplo, si en coordenadas lógicas la posición de una bola es (0,100) en el caso de estar en una resolución de 16/9, el motor transforma esa coordenada a la resolución que nosotros tengamos en nuestro dispositivo de tal manera que la lógica siempre siga su estándar y para su entendimiento el juego siempre esté en una resolución de 16/9 aunque en verdad la posición real de esa bola sea (0,45).

- **Bucle de juego e inicializador:** En efecto esta última interfaz lo que encapsula es la inicialización de la ventana y demás para posteriormente entrar de lleno en la ejecución del bucle principal del juego y llamar al **update** a las diferentes Entidades lógicas sin que la plataforma afecte, aunque haya 2 bucles de juego, uno para cada plataforma.

Dentro de la **Lógica** se ha dividido el juego en diferentes estados. Estos estados son Menu, Instructions, Play y Game Over. Cada uno de estos tiene su vector de Entidades y cada una de estas Entidades contiene un **Rectángulo** que indica la posición global de la entidad y un **Sprite** que es utilizado para saber lo que pintar en rectángulo anterior.

Para que todo tenga una mayor estructura jerárquica, existe una clase que se encarga de manejar los cambios de estado y la carga de recursos. Esta clase es **LogicStateManager** y la función principal que tiene es la de hacer los swaps entre estados y cargar todos los recursos necesarios para el juego nada más empezar. Esta clase también llama al update de los estados y almacena los colores que usa el **clear** para el fondo del juego.

Con todo esto explicado, queda la siguiente arquitectura de módulos: