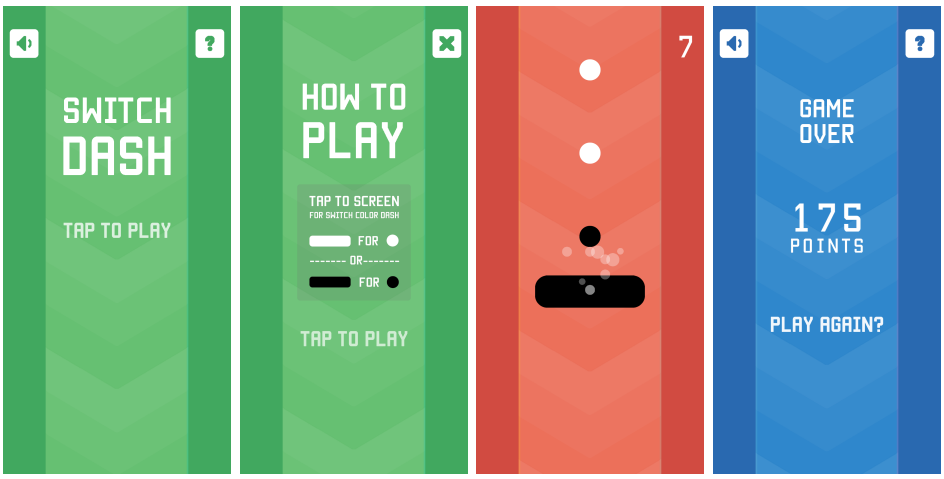
Práctica 1: Clon de *Switch Dash*

Fecha de entrega: 17 de noviembre de 2019, 23:55

Objetivo: Iniciación a la programación en Android y Java. Arquitectura para desarrollo multiplataforma



Switch Dash es un juego en HTML5 disponible en muchos sitios web de juegos para el navegador[1]. La mecánica es muy sencilla. En la parte inferior de la pantalla aparece una *barra* que alterna su color entre blanco y negro cada vez que el jugador pulsa en cualquier punto de la pantalla. De la parte superior caen bolas blancas y negras, y el jugador debe encargarse de que la barra inferior tenga el color de la bola que la golpea en cada momento. Recibe un punto cada vez que lo consigue, y la partida termina cuando falla. La velocidad de caída de las bolas aumenta progresivamente.

1. El juego

La práctica consiste en la implementación de un clon de *Switch Dash* para móvil y ordenadores de escritorio. Al lanzarse, el juego mostrará una pantalla de bienvenida

[1] Por ejemplo en `https://lagged.com/en/g/switch-dash`

animando al jugador a pulsar sobre la pantalla para empezar. Después, se mostrarán unas breves instrucciones y se pasará a la pantalla principal del juego.

En ella, las bolas caen a velocidad creciente hasta que el jugador falla en la captura de una por no haber colocado el color correcto en la barra. En ese momento se mostrará la puntuación final, dando la opción de jugar una nueva partida o volver a ver las instrucciones.

El fondo de la pantalla es siempre el mismo, aunque cambia aleatoriamente de color al principio de cada partida. La pantalla se pinta de un color plano, que es sustituído por otro ligeramente más claro en la zona central que hace las veces de "tablero". Encima de ese tono más claro, se pinta un patrón con forma de "cabezas de flecha" que se desplaza continuamente hacia abajo para dar la sensación de movimiento. La velocidad a la que se desplaza va aumentando al avanzar la partida, y se mantiene a la velocidad final cuando ésta termina.

A continuación se indican posibles valores para algunos de los parámetros más importantes del juego. En los casos donde es necesario, se asume un tamaño de referencia de pantalla de 1080x1920 píxeles:

- Las bolas están separadas entre sí por 395 píxeles y caen todas a la misma velocidad. Cuando se acelera el juego, lo hacen todas las bolas a la vez, y no solo las nuevas.
- La velocidad inicial de las bolas es 430 píxeles por segundo, y la de las flechas del fondo 384.
- Cada vez que se recogen 10 bolas más se incrementa la velocidad en 90 píxeles por segundo.
- El color de las bolas que caen es aleatorio, aunque ligeramente sesgado para favorecer la aparición de *secuencias* del mismo color. Cada bola tiene un 70 % de probabilidades de ser del mismo color que la que tiene debajo de ella.
- La barra del jugador está situada a 1200 píxeles de la parte superior de la pantalla.

2. Requisitos de la implementación

El juego se implementará en Java, y deberán generarse dos versiones diferentes, una para Android y otra para sistemas de escritorio (Windows o GNU/Linux). El código deberá estar correctamente distribuído en paquetes, ser comprensible y estar suficientemente documentado.

Para el desarrollo se hará uso de Android Studio, utilizando un único proyecto con varios *módulos*. La mayor parte del código *deberá ser común* y aparecer una única vez, compartida entre ambas versiones. Deberá existir así una *capa de abstracción* de la plataforma, que se implemente dos veces, una para Android y otra para escritorio. La implementación del juego deberá hacer uso únicamente de la capa de abstracción y de las funcionalidades del lenguaje que estén disponibles en ambas plataformas.

Dado que el punto de entrada de la aplicación es diferente en cada plataforma (en Android se necesita una `Activity` y en escritorio un `main()`) se permitirá también la existencia de dos módulos distintos (minimalistas) de "arranque del juego".

La aplicación se debe adaptar a *cualquier* resolución de pantalla. Independientemente de su relación de aspecto, el juego *no* deberá deformarse, sino aparecer *centrado* en la pantalla con "bandas" arriba y abajo o a derecha e izquierda. A modo de ejemplo, las

figuras mostradas al principio tienen una resolución de 1080x2220 (relación de 9/18.5), aunque la lógica espera una resolución de 1080x1920 (9:16). El espacio sobrante arriba y abajo se rellena con el fondo. Los botones *no* se desplazan y aunque para la lógica se colocan en 30x30, visualmente aparecen más abajo por la "banda" (invisible) superior.

3. Consejos de implementación

Para abstraer la plataforma, podéis definir los siguientes interfaces:

- `Image`: envuelve una imagen de mapa de bits para ser utilizada a modo de *sprite*:
  - `int getWidth()`: devuelve el ancho de la imagen.
  - `int getHeight()`: devuelve el alto de la imagen.
- `Graphics`: proporciona las funcionalidades gráficas mínimas sobre la ventana de la aplicación:
  - `Image newImage(String name)`: carga una imagen almacenada en el contenedor de recursos de la aplicación a partir de su nombre.
  - `void clear(int color)`: borra el contenido completo de la ventana, rellenándolo con un color recibido como parámetro.
  - `void drawImage(Image image, ...)`: recibe una imagen y la muestra en la pantalla. Se pueden necesitar diferentes versiones de este método dependiendo de si se permite o no escalar la imagen, si se permite elegir qué porción de la imagen original se muestra, si se especifica valor *alfa* para mezcla con transparencia, etcétera.
  - `int getWidth()`, `int getHeight()`: devuelven el tamaño de la ventana.
- `Input`: proporciona las funcionalidades de entrada básicas. El juego no requiere un interfaz complejo, por lo que se utiliza únicamente la pulsación sobre la pantalla (o *click* con el ratón).
  - `class TouchEvent`: clase que representa la información de un toque sobre la pantalla (o evento de ratón). Indicará el tipo (pulsación, liberación, desplazamiento), la posición y el identificador del "dedo" (o botón).
  - `List<TouchEvent> getTouchEvents()`: devuelve la lista de eventos recibidos desde la última invocación.
- `Game`: interfaz que aglutina todo lo demás. En condiciones normales, `Graphics` e `Input` serían *singleton*. Sin embargo, al ser *interfaces* y no existir en Java precompilador no es tan sencillo. El interfaz `Game` puede ser el encargado de mantener las instancias:
  - `Graphics getGraphics()`: devuelve la instancia del "motor" gráfico.
  - `Input getInput()`: devuelve la instancia del gestor de entrada.

Para independizar el código de la lógica de la resolución del dispositivo (o de la ventana) podéis ampliar la clase `Graphics` para que reciba un *tamaño lógico* (de "canvas") y que todas las posiciones se den en ese *sistema de coordenadas*. También podéis plantear el desarrollo de clases adicionales que proporcionen un mayor nivel de abstracción, como

por ejemplo una clase `Sprite`. En particular, para facilitar la puesta en marcha de la aplicación en cada plataforma, es posible que queráis ampliar `Game` para incorporar la idea de *estado* de la aplicación.

Al ser interfaces, observad que *no* se indica cómo se crearán las instancias. Así, por ejemplo, la clase que implemente `Graphics` para la versión de escritorio podría necesitar recibir en su constructor la ventana de la aplicación, y la versión de Android el `SurfaceView` y `AssetManager`. La puesta en marcha de la aplicación tendrá que ser diferente en cada plataforma y estar encapsulada, en la medida de lo posible, en los módulos correspondientes. No obstante, la *carga* de los recursos no debe programarse dos veces, y debe formar parte del módulo de lógica.

Para pintar una imagen añadiendo transparencia, en la versión de escritorio podéis usar `Graphics2D::setComposite()` y `Paint::setAlpha()` en Android. Si en la versión para móvil tenéis problemas de rendimiento, minimizad la creación de objetos (analizad la posibilidad de cachearlos con *pools*) y usad, si es posible, *superficies hardware* (`SurfaceHolder::lockHardwareCanvas()`).

4. Recursos suministrados

Se proporcionan diferentes ficheros de imágenes (`.png`) con los recursos gráficos necesarios para el desarrollo del juego. Todos son adaptaciones de los utilizados en el juego de HTML5 y están escalados pensando en un tamaño de ventana de 1080x1920:

- `arrowsBackground.png`: imagen parcialmente transparente con "flechas" hacia abajo. Aparece en la parte central de la ventana, ocupando todo el espacio a lo alto, y se desplaza hacia abajo para crear la sensación de movimiento. Sus colores son demasiado opacos, por lo que hay que pintarlo en la pantalla de forma parcialmente transparente.

- `backgrounds.png`: diferentes colores de fondo de la zona central de la pantalla ("tablero" de juego). Los laterales se pintan con un color plano de "borrado" de la pantalla. Para cada uno de los colores de fondo de este fichero podéis usar, respectivamente, 0x41a85f, 0x00a885, 0x3d8eb9, 0x2969b0, 0x553982, 0x28324e, 0xf37934, 0xd14b41 y 0x75706b.

- `balls.png`: *sprites* de las bolas. El fichero es una *sprite sheet* de dos filas, una con las imágenes en blanco y otra en negro. Aparecen 10 modelos de bolas diferentes, aunque es suficiente con utilizar una. Cada bola ocupa 100x100 píxeles, enmarcada en un cuadrado de 128x128 y alineada abajo.

- `buttons.png`: botones de la aplicación. No todos se utilizan. Como se ha dicho, en las pantallas de introducción, instrucciones y fin de partida se situan en $y = 30$.

- `gameOver.png`: texto de "*Game over*" para la pantalla de fin de partida. Se puede colocar, por ejemplo, en $y = 364$.

- `howToPlay.png`: texto de "*How to Play*" para la pantalla de instrucciones. Se puede colocar, por ejemplo, en $y = 290$.

- `instructions.png`: muestra esquemáticamente las instrucciones del juego. Se puede colocar, por ejemplo, en $y = 768$.

- `playAgain.png`: texto de "*Play again?*" para la pantalla de fin de juego. Se puede colocar, por ejemplo, en $y = 1396$ y se muestra oscilando entre totalmente visible y totalmente transparente para destacar el mensaje al usuario.
- `players.png`: barras blanca y negra del jugador. Las imágenes están alineadas arriba en cada uno de los rectángulos de la *sprite sheet*. Como se ha dicho, se situa en $y = 1200$.
- `scoreFont.png`: letras para escribir texto (los puntos y el mensaje "Points" en la ventana de fin de partida). Es una fuente de ancho fijo. Cada *glifo* (representación gráfica de un carácter) ocupa un espacio de 125x160 píxeles, aunque hay mucho espacio en blanco entre ellos. La zona realmente ocupada está centrada en ese rectángulo y ocupa aproximadamente 93x112.
- `switchDashLogo.png`: logo del juego para la pantalla de bienvenida. Se puede colocar en $y = 356$.
- `tapToPlay.png`: mensaje "*Tap to play*" mostrado en la pantalla de bienvenida ($y = 950$) y de instrucciones ($y = 1464$). En ambos casos se muestra oscilando entre totalmente visible y totalmente transparente para destacar el mensaje al usuario.
- `white.png`: imagen blanca para hacer el efecto de "flash" cuando se cambia de ventana.

5. Partes opcionales

Siempre que los requisitos básicos de la práctica funcionen correctamente y estén bien implementados, se valorará positivamente la incorporación de características adicionales como por ejemplo:

- Inclusión de efectos de sonido.
- En la versión de escritorio:
  - Uso de pantalla completa.
  - Uso del teclado con el mismo efecto que pulsar con el ratón.
  - Pausa del juego (sin consumir recursos) cuando la aplicación pierda el foco.

6. Entrega de la práctica

La práctica debe entregarse utilizando el mecanismo de entregas del campus virtual, no más tarde de la fecha y hora indicada en la cabecera de la práctica.

Sólo un miembro del grupo deberá realizar la entrega, que consistirá en un archivo `.zip` con el proyecto completo de Android Studio eliminando los ficheros temporales. Se añadirá también un fichero `alumnos.txt` con el nombre completo de los alumnos y un pequeño `.pdf` indicando la arquitectura de clases y módulos de la práctica y una descripción de las partes opcionales desarrolladas, si ha habido alguna.

El `.zip` deberá tener como nombre los nombres de los integrantes del grupo con la forma `Apellidos1_Nombre1-Apellidos2_Nombre2.zip`. Por ejemplo para el grupo formado por Miguel de Cervantes Saavedra y Santiago Ramón y Cajal, el fichero se llamará `DeCervantesSaavedra_Miguel-RamonYCajal_Santiago.zip`.

Bibliografía

- Beginning Android Games, Third Edition, Mario Zechner and J. F. DiMarzio, Apress, 2016.
- Developing games in Java, David Brackeen, Bret Barker, Lawrence Vanhelsuwe, New Riders.