**Memoria Móviles**

***Monetización***

El juego tiene un sistema de monedas que pueden ser conseguidas por el jugador jugando niveles. Cuando el jugador gana un nivel se le aporta un número de monedas y puede utilizar esas monedas para conseguir objetos cosméticos en forma de paletas. En cualquier momento el jugador puede cambiar la paleta utilizada en la escena de juego y desbloquear paletas siempre que tenga la cantidad necesaria de monedas. También se puede conseguir monedas en cualquier momento si giras sobre ti mismo mientras sostienes el móvil, de esta manera el jugador puede jugar Just Dance mientras consigue monedas en nuestro juego.

***Anuncios***

En la aplicación hay un banner situado en la parte superior de la pantalla que está presente continuamente en cuanto se carga.

El jugador también puede ver anuncios opcionales desde la escena de juego para recuperar vidas. Esto solo se puede realizar si el jugador no tiene el número máximo de vidas.

***Notificaciones push***

El juego recordará al usuario que lleva un tiempo sin jugar cuando no esté jugando mediante una notificación. El tiempo establecido es de 30 segundos después de que se cierre la aplicación para facilitar el testeo. No queríamos que tuvieras que estar esperando 30 minutos solo para que fuera más “realista” y fastidiarte la corrección.

***Sensores***

Hemos implementado un sensor de Android que te permite obtener monedas ilimitadas al girar. ¿Quién no ha querido alguna vez en su vida poder menear su móvil para conseguir todas las monedas que quiera?

***Guardado y cargado de archivos***

En un inicio habíamos pensado en cargar y guardar los archivos de los niveles e informacion del jugador desde la carpeta assets por su fácil acceso, sin embargo, descubrimos que la capeta assets, es únicamente de lectura y no de escritura, por lo que tuvimos que tomar la decisión de dejar en assets los archivos iniciales de carga en assets y posteriormente, con el juego instalado, cambiar su nuevo lugar de almacenamiento al interno de la propia aplicación del móvil y desde ahí gestionar la carga y descarga de datos.

***Nuevos Assets***

Con las nuevas funcionalidades y mecánicas incluidas en esta práctica se han implementado una nueva colección de assets propios para favorecer la estética del juego.

***Detección de modificaciones, Hash y checksum***

Sabiendo que esta práctica, contiene el uso de anuncios y otras formas de negocio, la seguridad de que los archivos no sean modificados con el propósito de favorecer al usuario injustamente, hemos elegido usar los hash y checksum como método de seguridad.

¿Por qué hemos usado este método? Porque pese a no ser el más fácil, si es de los más eficientes y seguros. La serialización tiene graves problemas de compatibilidad en ejecución, no es eficiente en tiempo y espacio y en Java evita el flujo estándar, aparte de otros inconvenientes a la hora de crear clases serializables. La encriptación era otra opción pero claramente suponía demasiado coste en guardado y carga y otros inconvenientes en datos locales.

Por ello, hemos decidido usar checksum y en este caso con MD5. Primero al guardar un archivo desde el juego se le hace un checksum md5 y se guarda como “clave” en otro archivo separado para mayor seguridad. Posteriormente, al cargar ese archivo, se vuelve a hacer un checksum sobre el mismo y se compara con el último guardado. Si se corresponde procede a cargar el archivo, si no, se detecta un fallo de seguridad y se hace una carga de seguridad de un estado anterior del archivo.

**Resumen implementación**

Esta práctica está formada por dos módulos:

* App: contiene la lógica del juego.
* EngineApp: contiene al engine.

**Lógica (App)**

Contiene toda la lógica del juego y el envío de notificaciones, también es la responsable de crear al Engine en su onCreate().

**Main Activity**

Actividad de la aplicación, obtiene su vista de un xml.

Este xml contiene un AdView donde se crea el banner de la aplicación y un SurfaceView donde se dibuja el juego. Ambos están dentro de un RelativeLayout para que se mantenga la relación en todos los dispositivos.

Inicializa el modulo de Android responsable de los anuncios, inicializa el engine, crea la escena principal (MainMenuScene) y la añade al engine.

Tambien es la responsable cuando se para la aplicación de llamar al resto de componentes para que guarden sus estados guardando la partida guardada. La aplicación no se reinicia cuando se cambia de orientación por un cambio en el Manifest de la actividad.

También se encarga de añadir al WorkManager una notificacion que es enviada 30 segundos después de cerrar la actividad.

**QuickGameScene**

Escena de juego de la práctica anterior con ligeras modificaciones para que sea compatible con los arreglos del Engine que serán posteriormente entregados en la reentrega de la práctica 1.

**CellBase**

Clase base de las celdas que permite una implementación más estructurada dependiendo de si estamos en QuickGameScene o HistoryModeScene.

**CellHistoryMode**

Clase que hereda de CellBase y gestiona el estado de una celda del tablero con las nuevas especificaciones de la Práctica 2.

**CellQuickGame**

Clase que hereda de Cellbase y gestiona el estado de una celda del tablero en QuickGame, esta implementación es la misma que en la práctica anterior.

**HistoryModeScene**

Escena de juego con la nueva implementación de la Práctica 2, incluye vidas, monedas que se consiguen al pasarse un nivel, paletas de colores que se pueden desbloquear gastando monedas y un botón que se puede pulsar para recuperar vidas visualizando un anuncio. El jugador también puede compartir un mensaje en Twitter al pasarse el nivel.

La escena crea todos los elementos en su constructora al igual que en la práctica 1. Se han añadido más botones para las nuevas funcionalidades.

Cada vez que la escena detecta input antes de hacer nada comprueba el número de vidas y procesa el input acorde a ello. Por ejemplo, no permite que se pulse en visualizar anuncio para recuperar una vida si la vida esta al 100% o no permite cambiar celdas si el jugador no tiene vida.

Cuando el jugador pulsa en la última celda correcta, se cambia el estado a win == true y se renderiza la pantalla de victoria, donde el jugador puede volver hacia atrás o compartir mediante un intent un mensaje en Twitter.

También se encarga de cargar el progreso del nivel y guardarlo si el jugador sale del nivel antes de resolverlo. Si el jugador se queda sin vidas sus datos de guardado se reinician.

**HistoryModeMenu**

Menu en el que hay dos botones: el botón de atrás donde vuelves a la escena anterior y el botón de temas que te permite pasa al menú de selección de temas. La imagen del modo desafío ha sido mantenida en caso de que en algún punto sea implementado.

**InputButton**

Hereda de Interactive y permite la gestión del input en la escena con métodos sencillos que permiten especificar un área de input utilizable en las escenas.

**IntentWorkRequest**

Clase que hereda de Worker y permite especificar una tarea para el WorkManager, es utilizado por la actividad para mandar notificaciones.

**LevelSelection**

Selecciona un tamaño de tablero para QuickGameScene.

**MainMenuScene**

Escena que carga todos los recursos del juego, llamada por el engine alser configurada como la primaryScene. Esta es la única escena que tiene permitido cargar recursos de imágenes, sonido, etc. También carga la partida guardada del jugador que incluye su número de monedas, su progreso y paletas de colores desbloqueadas. Estos parámetros se van pasando al resto de escenas mediante un AtomicReference.

**ThemeModeLevels**

Escena que selecciona un nivel de una categoría en concreto.

**ThemeModeMenu**

Escena que permite la selección de un tema/categoría.

**Engine**

**AdManager**

Se encarga de crear el rewardedAd, el canal de notificaciones y de enviar intents. Este AdManager puede ser accedido desde cualquier escena solo en su handleInput().

**AudioAndroid**

Permite la reproducción de sonidos tanto cortos como largos. Su implementación es la misma que en la Práctica 1, pero ahora solo se puede acceder a él desde el handleInput() de la escena.

**EngineApp**

Contiene la hebra principal del juego. Crea en su constructora todos sus componenetes y los guarda, permitiendo el acceso a ellos desde la lógica en diferentes métodos. En caso de que la aplicación sea interrumpida inesperádamente llama a la escena actual para que tenga la oportunidad de guardar su estado en caso de que sea necesario. También avisa a la escena actual en caso de que se cambie la orientación de la pantalla.

**EventHandler**

Estructura que guarda información sobre un evento de input. Igual que en la práctica 1, pero se ha añadido la posibilidad del evento de Long Touch.

**FontAndroid**

Guarda información de una fuente. Igual que en la práctica 1.

**ImageAndroid**

Guarda un Bitmap de una imagen- Igual que en la práctica 1.

**InputAndroid**

Maneja el input, permitiendo que la escena le pregunte en su handleInput() datos como la posición actual del ratón y si está pulsado. Ahora se realiza una conversión de coordenadas lógicas a puntos de la verdadera pantalla.

**RenderAndroid**

Renderiza el surface, es igual que en la práctica 1, pero ahora se realiza aquí la conversión de coordenadas lógicas a tamaño del surface y se escala el texto dependiendo del tamaño del surface.

**Scene**

Interfaz de la que heredan todas las escenas de la lógica, sus distintos métodos son llamados por el SceneManager y cumplen distintas funcionalidades.

**SceneMngrAndroid**

Guarda un stack con las escenas del juego y actualiza la escena actual.

**SoundApp**

Guarda la información de un sonido. Igual que en la práctica 1.

**Utils**

Clase que realiza el Checksum MD5 para la gestión de archivos de guardado.

**Vector2D**

Clase que permite definir un punto lógico. Utilizado por las escenas para posicionar objetos como los botones. Igual que en la práctica 1.