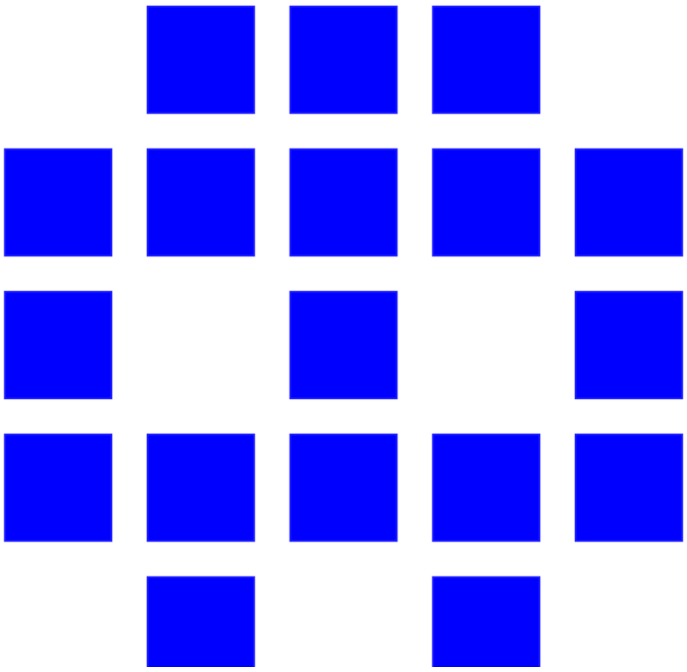
Videojuegos para dispositivos móviles
Resumen Práctica 2

Grupo 11:
Stiven Arias Giraldo
Óscar Fernández Romano
Álvaro Serna Ramírez

Nonogramas



Grado en Desarrollo de Videojuegos
Facultad de Informática

Madrid, 2022-2023

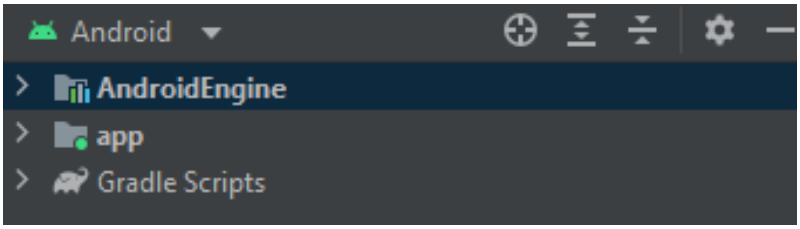
Índice general

Capítulo 1

Arquitectura de módulos

En esta nueva práctica ya no se dispondrá de la capacidad multiplataforma, pues ahora el juego va a funcionar solamente en **Android**. Por ello, se han suprimido todos los módulos relacionados con **Desktop**, así como toda la abstracción mediante interfaces realizada en la práctica anterior. Así pues, se tienen los siguientes módulos.

- **AndroidEngine**. Contiene todas las clases necesarias para poner en funcionamiento el juego. Ya no implementa interfaces como se hacía en la práctica anterior ni contiene ninguna dependencia con cualquier otro módulo. Se explicará más detalles sobre este módulo en capítulos posteriores.
- **app**. Antes se utilizaba para únicamente inicializar el juego creando el **Engine**. Ahora, este módulo contiene toda la lógica del juego, de forma que el módulo de **Logic** ya no existe y ha pasado a ser un **package** de este módulo. Por lo tanto, este módulo contiene todo el juego de la práctica. Tiene una depenencia con **AndroidEngine**.



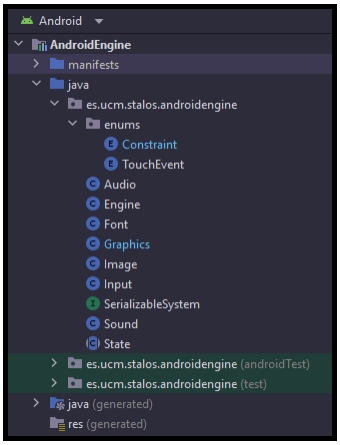
Módulos usados

Capítulo 2

Arquitectura de clases

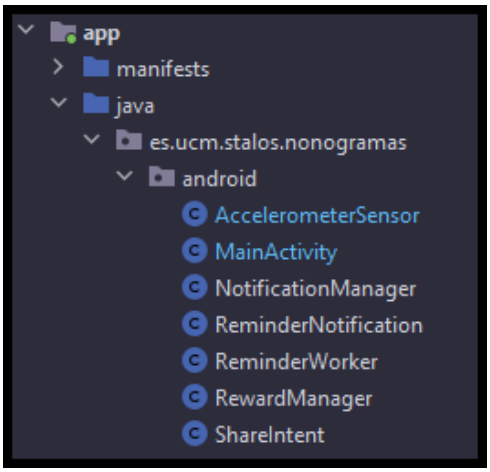
En general, se están usando las mismas clases de la práctica anterior. Además, se han creado y modificado tantas clases como fuese necesario con el objetivo de generar el juego solo para Android. Así pues estas son las clases añadidas en cada módulo:

- **AndroidEngine**. Se ha modificado el nombre de todas las clases que tuviesen relación con el **Engine** de la anterior de práctica suprimiendo el prefijo **Android** en cada una de ellas (Audio, Engine, Font, Graphics, Image, Input y Sound). Además, las clases *Abstract*, creadas anteriormente para implementar código código común a ambas plataformas, han sido eliminadas y todo el código ha pasado a formar parte de las respectivas clases. Por otro lado, las clases **FileReader** y, también, **IFile** han sido eliminadas pues éstas eran una abstracción creada para poder leeer archivos en ambas plataformas. Por otro lado, se han añadido las siguientes clases:
  - **Paquete *enums***. Dentro de este paquete hemos incluido los diferentes *enums* que hemos visto necesario crear: **TouchEvent** (ya existente) y **Constraint**.
    - **Constraint**. Consiste en una serie de tipos que definen la posición en la que se debe pintar un objeto. Fue creado principalmente para cambiar las coordenadas de dibujado al alternar entre modo vertical y horizontal del móvil.
  - **SerializableSystem**. Se trata de una interfaz que contiene los métodos *void saveData()* y *void loadData()*. Está pensada para que se pueda utilizar dentro del motor de forma abstracta y para que en la lógica se puedan implementar los métodos como sea necesario, ya sea como se ha hecho en esta práctica como en cualquier otro tipo de juego.



Contenido del módulo AndroidEngine

- **app**. Como se mencionó anteriormente, dentro de este módulo se encuentra el lanzador del juego y también la lógica. Además se han añadido otras clases para complementar lo requerido en el enunciado de la práctica. Por tanto, lo que era conocido como **Módulo Logic** ahora ha pasado a ser un paquete de este módulo llamado **logic**..
  - **Paquete *android***. Este paquete está pensado para almacenar clases que implementen código relacionado con el SO del móvil o o que implementen código
    - **AccelerometerSensor**. Es la clase que se encarga de manejar la lógica del juego cuando interviene el sensor del acelerómetro del móvil. Hereda de **SensorEventListener**.
    - **NotificationManager**. Clase encargada de manejar las diferentes notificaciones de la aplicación. Ahora mismo solo existe **ReminderNotification**, pero dentro de esta clase se pueden añadir más funciones con el fin de manejar otro tipo de notificaciones. Utiliza **WorkManager** para poder crear temporizadores en las notificaciones.
    - **ReminderNotification**.Clase que contiene toda la información y funcionalidad necesaria para mostrar la notificación requerida por el enunciado. Muestra mensajes de forma aleatoria.
    - **ReminderWorker**. Hereda de **Worker** y se usa para lanzar una **ReminderNotification** a través del tiempo establecido en **NotificationManager**.
    - **RewardManager**. Es la clase encarga de mostrar anuncios por pantalla, de forma que cuando se termine la visualización del anuncio le comunica a **GameState** que debe otorgarle un premio al jugador.
    - **ShareIntent**. Es la clase que contiene la funcionalidad necesaria para lanzar un *Intent* que sea capaz de abrir una aplicación externa para poder compartir información del juego con más personas.



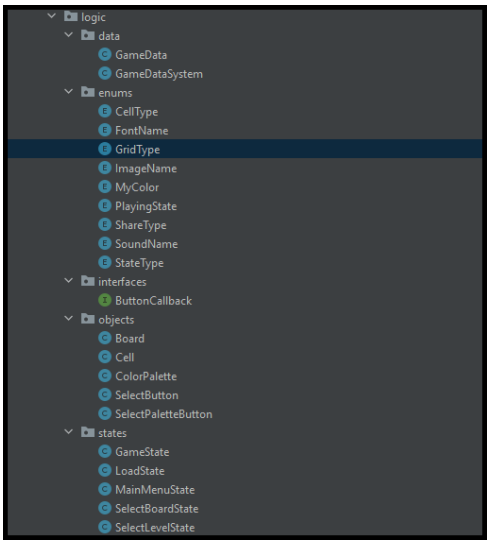
Paquete de **android**

  - **Paquete *logic***. Ahora el anterior módulo **Logic** se encuentra aquí, junto con más clases necesarias para cumplir con los objetivos de la práctica. A su vez, este paquete está compuesto por los siguientes paquetes: **data**, **enums**, **interfaces**, **objects** y **states**.
    - **Paquete data**. Nuevo paquete que contiene aquellas clases relacionadas con la serialización y guardado de datos.
      - **GameData**: Contiene todos aquellos datos que se quieren guardar del juego.
      - **GameDataSystem**: Implementa la interfaz **SerializableSystem** anteriormente mencionada. Así pues, contiene la funcionalidad para serializar los datos y deserializarlos. Además, tiene la capacidad de encriptar los datos mediante **hash-number**.
    - **Paquete *enums***. Al igual que en la prácitca anterior, es el paquete dedicado a contener los diferentes enumerados del juego. Se ha ampliado con las siguientes clases:

- ◇ **FontName**: Contiene información sobre el nombre de aquellas fuentes que se utilizan durante el juego. Es usando tanto para la creación como para el dibujado de las fuentes.
- ◇ **ImageName**: Contiene información sobre el nombre de aquellas imágenes que se utilizan durante el juego. Es usando tanto para la creación como para el dibujado de las imágenes.
- ◇ **MyColor**: Contiene información sobre el nombre de aquellos colores que se utilizan durante la lógica en formato hexadecimal RGBA.
- ◇ **ShareType**: Contiene información sobre las diferentes plataformas a las que enviar un **ShareIntent**.
- ◇ **SoundName**: Contiene información sobre el nombre de aquellos sonidos que se utilizan durante el juego. Es usando tanto para la creación como para la reproducción de sonidos.
- ◇ **StateType**: Nombre de todos aquellos estados que se utilizan durante el juego. Son los mismos que los estados que ya existían anteriormente.

- ○ **Paquete *interfaces***. Este paquete no tiene cambios respecto a la práctica anterior.
- ○ **Paquete *objects***. Contiene los diferentes "*GameObjects*"del juego. Se han añadido las siguientes clases:
  - ◇ **ColorPalette**: Es el objeto que contiene todas la paletas de colores. También es el encargado de manejar los botones relacionados con cada una de las paletas.
  - ◇ **SelectPaletteButton**: Objeto usado por **ColorPalette** para representar una paleta de colores. También posee el *callback* para aplicar el cambio de colores en el juego.

- **Paquete *states***. Este paquete no tiene cambios respecto a la práctica anterior.



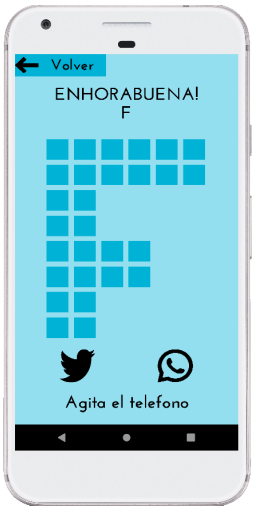
Paquete de **logic**

Capítulo 3

Información Adicional

- **Orientación**: El videojuego funciona tanto en vista vertical como en vista horizontal sin utilizar bandas a los lados. Para ello, se ha hecho uso del *enum* **Constraint**. De esta forma, en cada uno de los estados se están organizando los objetos en función de la orientación del móvil. Para ello se hace uso de la función *protected void togglePortraitLandscape(boolean isLandscape)*, la cual es llamada tanto desde los inicializadores de los estados como desde *onResume()* del **Engine** cuando se reinicia el juego. Además, se están guardando todos los datos necesarios para poder mantener el estado del juego.

Orientación vertical



Orientación horizontal

- **Cambio el tamaño lógico original**. Hemos estado investigando al respecto y el tamaño 400x600 lógico no nos parecía el más adecuado, pues tiene una relación de aspecto de 2:3. Por ello, hemos decidido usar 360x640, ya que mantiene una relación de 16:9, la cual es la más utilizado a nivel mundial en los móviles.
- **Vidas**. Hemos diseñado un sistema de 3 vidas donde si el jugador pierde alguna de las vidas, éstas pueden ser recuperadas mediante anuncios (gestionados por **RewardManager**). Además, si el jugador pierde todas las vidas, se mostrará la pantalla ed *Game Over* donde el jugador podrá recupera una y solo una vida para continuar con el estado del tablero.
- **Sensor utilizado: Acelerómetro**. Hemos utilizado el sensor para que cuando te hayas pasado un nivel en el modo historia pases al siguiente nivel agitando el dispositivo. Cuando te encuentras en el último nivel del paquete no cambia al siguiente. En el modo aleatorio esto tendrá el efecto de cargar un nuevo nivel con el mismo tamaño de tablero.

- **Modo Historia**. Para el modo historia hemos optado por utilizar un sistema de dificultad organizado por diferentes tamaños de tablero (4x4, 5x5, 8x8, 10x5, 8x8, 10x10, 15x10). Cada paquete de niveles tiene a su vez 20 niveles, de forma que va a estar todo bloqueado para ir abriendo niveles a lo largo del juego. Además, cada paquete está relacionado con una paleta de colores, de forma que al completar un paquete se desbloquea una paleta.
- **Paleta de colores**: Hemos implementado una paleta de colores, con diferentes conjuntos de colores que se desbloquean según completas paquetes del modo historia. Estas paletas cambian el aspecto general de la aplicación tanto de botones como el fondo y se mantiene durante la ejecución e incluso después de cerrar el juego. Los colores de la paleta sólo se pueden cambiar mediante los botones correspondientes en el**GameState**.
- **Interacción social**: Para la interacción social se permite al usuario enviar mensajes a través de **Twitter** y de **WhatsApp** informando sobre el nivel que se acaban de pasar. Por otro lado, en el mensaje que se genera se ha añadido también un enlance a nuestro repositorio, con la idea de emular nuestra propia página web del videojuego. Es así porque creemos que sería buena idea compartir dicho enlace con el fin de obtener más publicidad de nuestro juego. Otra opción sería con un enlace que envíe directamente a **PlayStore** para descargar la aplicación.
- **Correcciones importantes: Práctica 1**
  - **Qué es MAX en Input::TouchEvent??**: Se ha eliminado este tipo *enum* dentro de TouchEvent.
  - **Sonidos e Imágenes** : Tal como se nos recomendó, se ha utilizado un **HashMap** para almacenar estos datos, así como las fuentes. Además, se está haciendo uso de **R.string.someValue** de forma que estamos almacenando información sobre la ruta de los archivos en la carpeta **res/values** que viene por defecto en Android. De esta forma evitamos también el problema respecto a usar *path-names* directamente en el código.
  - **Ventana/Vista**: Ahora **MainActivity** es quien se encarga de pasarle una vista al **Engine** de forma que esta implementación no se desarrolle en su interior.
  - **Incoherencia de nombres en enums**: Se han cambiado los nombres para que sean más apropiados, no solo en los enumerados, sino en general.