Jose Daniel Rave Robayo - Sergio José Alfonso Rojas - Daniel Illanes Morillas

NONOGRAMA

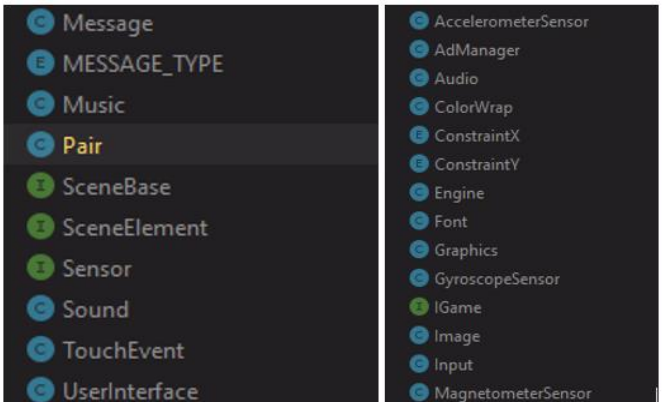
Práctica 2

INTRODUCCIÓN

La práctica consiste en el desarrollo de un único motor para Android y una aplicación de dispositivo móvil que usa este sobre dicha plataforma. Por lo tanto, el proyecto consiste en dos módulos los cuales serán explicados a continuación, al igual que el contenido de sus clases y el uso de estas.

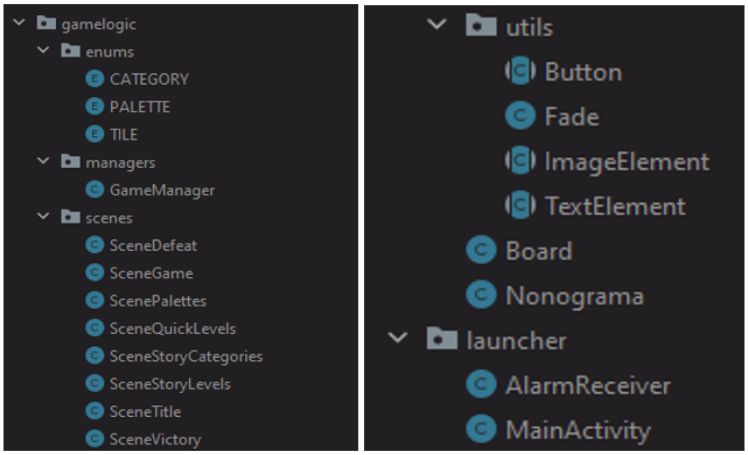
MÓDULOS

- Motor Android
  Contiene un gran grupo de clases que se encargan de gestionar distintas funcionalidades y tareas básicas de una aplicación. Por ejemplo, el sistema de renderizado, audio, entrada de usuario e incluso gestión de anuncios. Funcionan como envolvente para abstraer los motores proporcionados por Android y librerías propias de Java.

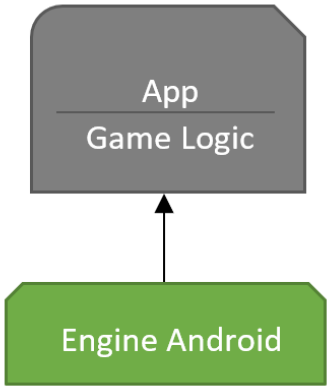


Listado de clases del módulo EngineAndroid

- App
  Básicamente se trata de una estructura de clases que conforman el videojuego usando las clases del módulo previamente mencionado y un lanzador que crea instancias tanto del motor como del juego para ejecutar la aplicación en un dispositivo móvil. En este caso la clase Nonograma es quién se encarga de crear, destruir y gestionar las escenas que definen los estados del juego. Además, controla al GameManager, una clase estática que usa un patrón de diseño Singleton, cuyo único propósito es almacenar datos triviales de la partida de manera global.



Listado de clases del módulo App



Dependencias entre módulos

IMPLEMENTACIÓN

- Motor Android

  El motor con respecto a la Práctica 1 no ha cambiado mucho su estructura, salvo que ahora se trata de una clase sin heredar de ninguna interfaz.

  Implementa nuevas funcionalidades como la entrada de usuario mediante sensores, lanzamiento de dos tipos anuncios (Banner y Reward), y el uso restricciones de posición en pantalla (también conocidas como Constraints).

  A continuación, explicaremos brevemente las clases más relevantes del motor.

  - **Audio:** Se encarga de crear y reproducir audios usando un SoundPoll para sonidos cortos. Contiene una tabla hash que guarda los que se carguen en la lógica.
  - **Font:** Se encarga de cargar fuentes y gestionarlas. Así como alterar su tamaño y estilo.
  - **Color:** Envolvente de la clase Color de Android que almacena los colores necesarios.
  - **Graphics:** Se encarga del renderizado de todos los elementos de la pantalla usando coordenadas mapeadas a un tamaño de juego lógico o Constraints para anclar objetos en esquinas o laterales de la pantalla. Asimismo, se encarga del recalculo de tamaños al girar el dispositivo. Usa los enumerados ConstraintsX y ConstraintsY.
  - **Image:** Envuelve un mapa de bits que definen una imagen cargada desde lógica. Almacena sus atributos básicos, así como métodos para manipularlos.
  - **Input:** Se encarga de gestionar el input mediante una lista de eventos, tales como TOUCH, RELEASE, LONG, MOVE.
  - **Sound:** Envuelve un sonido y permite gestionarlo mediante un SoundPoll(reproducir, pausar, etc.). Está hecho para lanzar sonidos relativamente cortos.
  - **Music:** Envuelve un sonido cuya longitud es considerablemente mayor en comparación con un Sound. Implementa funciones para gestionarlo (reproducir, pausar, loop, volumen, etc.).
  - **AdManager:** Gestiona la inicialización, carga y lanzamiento de dos tipos de anuncios: Banner y Reward. Permite también obtener datos gráficos básicos de estos como su tamaño.
  - **IGame:** Interfaz de gestor de escenas. Abstracción más básica del juego que conoce Engine.
  - **SceneBase:** Interfaz de escena. Abstracción más básica de una escena que maneja IGame.
  - **ISensor:** Interfaz par gestionar sensores. Permite activarlos o desactivarlos en ejecución.
  - **AccelerometerSensor:** Hereda de ISensor. Se encarga de obtener la aceleración del propio dispositivo en todos los ejes y almacenarlos para poder ser usados desde un juego.
  - **GyroscopeSensor:** Hereda de ISensor. Se encarga de obtener la aceleración angular del propio dispositivo en todos los ejes y almacenarlos para poder ser usados desde un juego.

- **MagnetometerSensor:** Hereda de ISensor. Se encarga de obtener datos sobre cambios producidos en el campo magnético de la Tierra. En general obtiene las fuerzas del campo sobre los tres ejes y los almacena para poder ser usados desde un juego.
- **Message:** contenedor de un tipo MESSAGE_TYPE. Se usa para la comunicación entre objetos de juego, anuncios o incluso entre el SO de Android para el uso de notificaciones.

- App

  Con respecto a la Práctica 1, se ha añadido dos modos de juego: partida rápida y modo historia. La partida rápida es básicamente el contenido de la práctica 1. Sin embargo, el modo historia contine cuatro categorías con temáticas distintas cuya dificultad aumenta conforme se vayan desbloqueando progresivamente.

  Además, se ha añadido un sistema de monetización que permite comprar paletas de colores que afectan el estilo de las casillas del tablero. Por cada tablero completado el jugador obtendrá **dinero** como recompensa, siempre y cuando no haya perdido sus tres vidas durante la resolución de éste.

  Por otro lado, se han incorporado **anuncios** al juego. En la parte inferior del tablero existe un anuncio que se muestra constantemente en pantalla (Banner). Durante la partida, el jugador podrá recuperar una **vida** al ver un anuncio de tipo Reward; de la misma manera, al ganar una partida y se obtiene dinero, éste se podrá duplicar al ver este mismo tipo de anuncio.

  Los **sensores** y **notificaciones**, en esta Práctica 2, juegan un papel secundario, ya que los usamos para cambiar el color de la pantalla del título usando la orientación del dispositivo mediante el magnetómetro (el acelerómetro y giróscopo no se han usado en esta práctica). Las notificaciones las lanzamos al salir de la aplicación cada 30 segundos para incitar al jugador a entrar en el juego (y también como método para exhibir dicha funcionalidad en la corrección de esta práctica). De tal manera que gamificamos un poco la entrada al juego.

  En relación a la persistencia, la aplicación contiene un sistema de guardado implementando mediante una mezcla entre **SharedPreferencies** y sistema de ficheros con **checksum invertida**, el cual impide que el jugador pierda sus datos más importantes [1]al cambiar de aplicación, cerrarla o incluso apagar el móvil. Si el jugador se encontraba jugando en alguno de estos últimos casos, el estado del tablero se restaurará al volver a abrir la aplicación. Sin embargo, si el jugador entra a otro nivel e interactúa con este, los datos guardados del nivel anterior se habrán perdido.

  Por último, cabe destacar que el juego ahora permite jugar en horizontal, distribuyendo así los elementos de la escena en una nueva layout para así aprovechar todo el espacio de la pantalla.

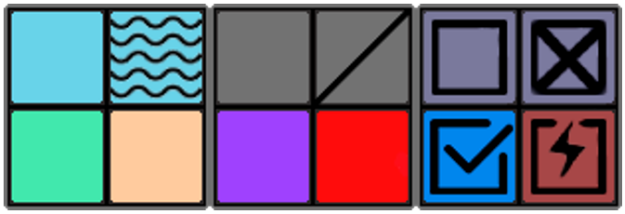
  NOTA: el botón para comprobar si el tablero es correcto, es manual ya que a nivel de diseño nos parecía más intuitivo que el jugador tenga que comprobar su solución y no que el programa la compruebe constantemente incluso si el estado actual del tablero coincide con la solución.

  A continuación, explicaremos brevemente las clases más relevantes del juego.

  - **SceneTitle:** Escena inicial del juego que presenta el título y tres botones. El botón de "Quick Game" lleva a la pantalla de elección de niveles aleatorios. El botón "Story" lleva a la pantalla de elección de categoría. Y por último el botón "Palettes" lleva a la pantalla de elección de paletas.
  - **SceneQuickLevels:** Escena de selección de nivel que presenta la posibilidad de generar niveles de tamaño: 4x4, 5x5, 5x10, 8x8, 10x10 y 10x15.
  - **SceneStoryCategory:** Escena de selección entre cuatro categorías. Cocina, medieval, océano y animales.

[1] Estos son: dinero, niveles desbloqueados de cada categoria, paletas desbloqueadas, paleta seleccionada.

- **SceneStoryLevels:** Escena de selección de un nivel dentro de una categoria concreta. Existen 20 niveles por categoria con distintos tamaños.
- **SceneGame:** Escena actual del juego. Permite comprobar el estado del tablero, volver al menú de selección o incluso recuperar una vida mediante un anuncio.
- **SceneVictory:** Escena de victoria que muestra el resultado final. Permite tres interacciones : ver un anuncio para poder duplicar el doble de dinero conseguido por completar el tablero, volver a la pantalla de elección de niveles, o compartir la victoria del juego en un post en Twitter (usando los Intents de Android).
- **SceneDefeat:** Escena de derrota que permite volver a la pantalla de elección de niveles.
- **ScenePalettes:** Escena de elección de paletas que permite elegir entre tres distintas.



Paletas a elegir, siendo la central la default.

- **Board:** Clase que almacena la información de un tablero de Nonograma, con las operaciones necesarias para generar un tablero válido (dado el tamaño o dado un fichero de texto), dibujarse, actualizarse, hacer comprobaciones propias del nonograma y compararse con otros Board. **El tablero puede tener más columnas que filas.** Además, tiene implementación para guardar su contenido en fichero.
- **Button:** Clase para generar un botón sencillo con una fuente, una imagen y un callback al ser pulsado.
- **Fade**: Animación simple de difuminado (in y out) para transiciones entre escenas.
  Nota: En general el fade se realiza de toda la pantalla teniendo en cuenta el tamaño del anuncio situado en la parte inferior de la pantalla. En emulador el difuminado es correcto, sin embargo en dispositivos físicos el tamaño del anuncio banner es distinto y genera un espacio en blanco que no conseguimos solucionar, sin romper el funcionamiento en el emulador.
- **Tile**: Enumerado auxiliar para identificar las casillas del nonograma.
- **Nonograma**: Clase que se encarga de inicializar, actualizar, renderizar, pausar, resumir y guardar escenas. Además, incorpora la posibilidad de comunicar escenas entre ellas mediante mensajes, así como de pasar entre ellas mediante una pila.

  Contiene datos de lógica para que el guardado de partida sea siempre que se salga de la aplicación o ésta se cierre por completo, y que el guardado y restaurado de tablero mediante fichero ocurra solamente si la escena actual es SceneGame. Además, contiene métodos para la generación de un Hash para prevenir manipulación externa de los ficheros de guardado. En caso de que ocurra, se perderán TODOS los datos guardados.

- **AlarmReciever**: se encarga de construir una notificación. En el MainActivity se crea dicha notificación al salir de la aplicación y se lanza cada 30 segundos.
- **GameManager:** obtiene datos de la partida[1] básicos. Implementa la posibilidad de guardar dichos datos usando SharedPreferencies. Asimismo, incorpora la funcionalidad de lanzar Intents a Twitter (Nota: realizamos esta tarea en el GameManager ya que al ser algo tan especifico que puede lanzarse desde cualquier sitio, no pensamos en crear una clase propia que incluya dicho método).
- **ImageElement y TextElement:** clases auxiliares como prueba para crear sistema de interfaz ajeno al pintado general el juego(sin interacción con el usuario). Se han usado únicamente para la creación de la moneda fijada en la parte superior derecha de la pantalla. Por falta de tiempo, no se siguieron extendiendo en la práctica.

Nota final: si se pretende probar todos los niveles para comprobar que funcionan correctamente, establecer un número del 1 al 20 a lado de la cadena de texto que indica su categoria.

```
//Levels Unlocked
//Poner x de mPreferences.getInt("kitchen", x); de 1 a 20 y reinstalar si se quiere iniciar el juego
//con esa cantidad de niveles debloqueados
int kitchenIndex = mPreferences.getInt( s: "kitchen",  i: 0);
int medievalIndex =mPreferences.getInt( s: "medieval",  i: 0);
int oceanIndex = mPreferences.getInt( s: "ocean",  i: 0);
int animalIndex = mPreferences.getInt( s: "animal",  i: 0 );
```

Trozo de código del método Restore() del GameManager