# ARQUITECTURA Y MÓDULOS DE LA PRÁCTICA

Los módulos que encontramos en la práctica son los siguientes:

**Engine:** módulo común para ambas plataformas. Contiene la estructura del motor de manera abstracta sin centrarse en ninguna plataforma.
Contiene las interfaces del propio Engine y las principales partes de este: IGraphics, IInput y IAudioSystem.
A su vez en él hemos creado interfaces de los diferentes elementos con los que trabajará el motor: ISound, IFont, IColor, IImage, IScene. Para facilitar el trabajo en cada una de las plataformas, hemos definido clases bases de estos elementos en este módulo: SoundJ, ColorJ, ImageJ, FontJ; que además de los métodos públicos contienen algunas variables que serán comunes a ambas plataformas.
Por último, señalar que también se encuentra en él la clase Engine. Clase base que aunque no sea funcional, si que implementa los métodos de Run, Resume, Pause, HandleInput, SetScene. Con lo que ya damos un comportamiento común de ejecución del motor para todas las plataformas que se implemente, salvando distancias.

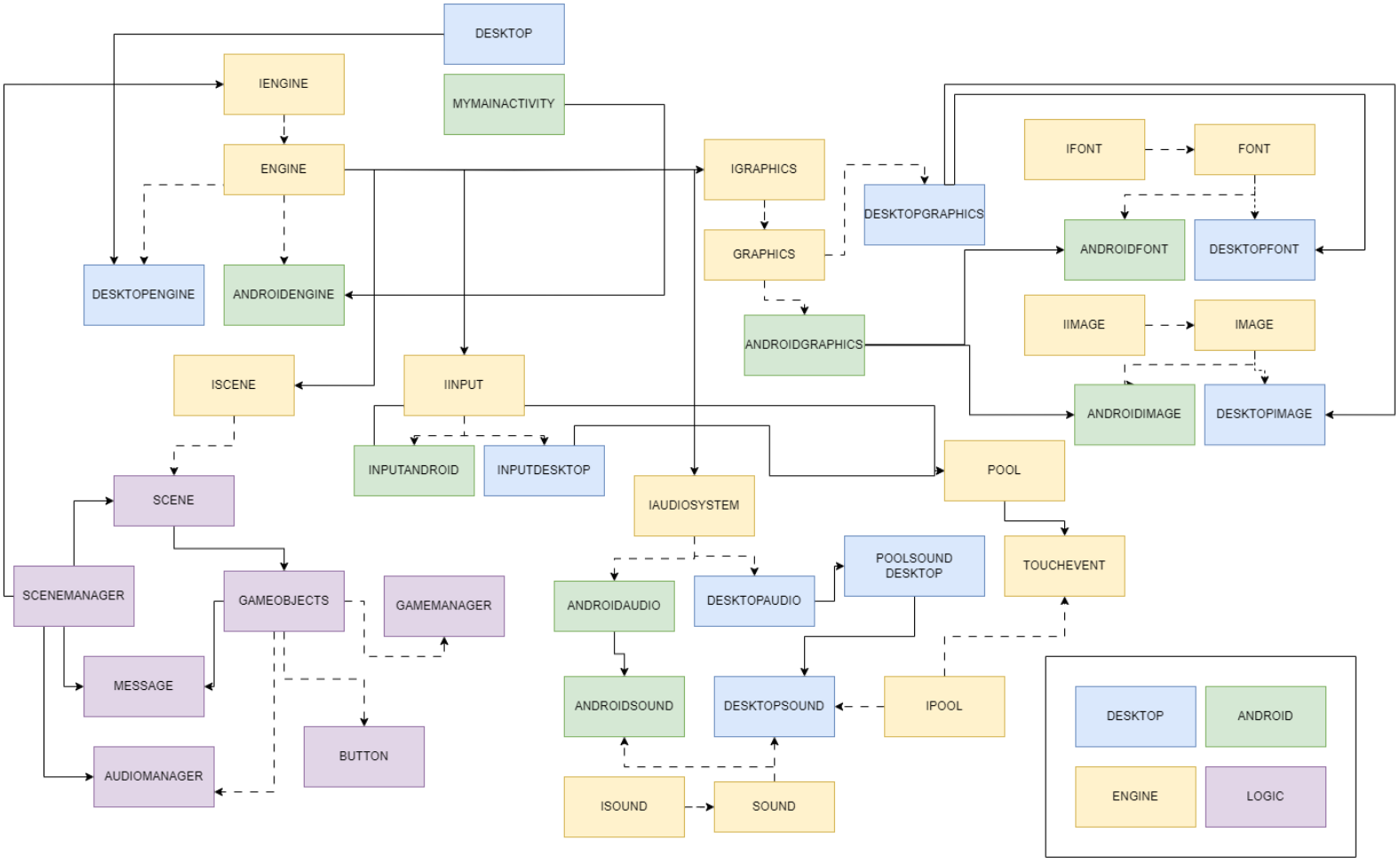
**Src:** módulo donde se contiene toda la lógica del juego y sistemas adicionales; que al ser un motor puramente tecnológico no se ubican en él. Es donde desarrollamos el juego puramente dicho.
Las clases más relevantes son:
- Scene, que hereda de la interfaz IScene y la adapta a nuestras necesidades.
- IGameObject - GameObject, que usamos como entidad común para todos los elementos del juego.
- SceneManager, singleton con el simplificamos la gestión de escenas juego-motor, así como el control de gameobjects y mensajes en la escena.
- AudioManager, que gestiona el audio de cada escena.
- GameManager, que gestiona la escena principal de juego.
- Message, que sirve como plantilla para los mensajes.
- Clases de Row y ButtonArray, que estructuran los botones y los distribuyen para tener una creación de la pantalla de juego más sencilla.

**Desktop Engine:** módulo que extiende el módulo de engine y lo adapta para la plataforma de Desktop. Completa y expande las funcionalidades del Engine para que ahora sí sea funcional en PC.
Implementa Desktop como punto de entrada. DesktopAudio, DesktopImage, DesktopInput, etc; para dotar de funcionalidad a las interfaces de elementos para la plataforma objetivo. Y contiene una PoolSoundDesktop para mejorar el rendimiento evitando generar instancias nuevas por cada llamada para reproducir un sonido.

**Android Engine:** módulo que extiende el módulo de engine y lo adapta para la plataforma de Android. Completa y expande las funcionalidades del Engine para que ahora sí sea funcional en dicha plataforma.

MainActivity realiza la función de punto de entrada. Encontramos clases como AndroidAudio, AndroidImage, AndroidInput, etc; que dotan de funcionalidad a las interfaces de elementos vistas en el módulo Engine para la plataforma de Android.
El input se gestiona mediante InputAndroid e InputHandlerAndroid, usando esta última también una pool.

Para la práctica hemos utilizado el siguiente esquema de clases. En este se pueden ver las clases utilizadas en cada módulo como sus relaciones entre estas:




Visto este esquema ahora explicaremos más detalladamente algunos de sus puntos específicos para nuestra versión de lo solicitado en la práctica.

Para poder unificar todos los elementos del juego y conseguir una gestión más cómoda de estos hemos creado la clase GameObject que sirve como clase base para la gran mayoría de entidades del jeugo; sobretodo de aquellas que requieren actualizaciones en el bucle de juego.

Hemos usado pools para la creación de sonidos en desktop (PoolSoundDesktop), para poder imitar el funcionamiento que tiene la librería SoundPool en android. Así como una pool para guardar tanto en android como en desktop los diferentes eventos de input que se dan en el juego. De esta manera la primera pool descrita se encarga de la creación de DesktopSound mientras que la segunda se encarga de los TouchEvent.

También, un punto clave para el paso de información es el uso de mensajes. Para ello tenemos la clase Message, de la cual heredarán los diferentes tipos de mensajes que queremos usar (modo daltonico, poner color, etc.). Después, dentro de los handler input de los gameobjects, cuando queramos realizar algún tipo de funcionalidad mandaremos un mensaje a través del SceneManager, el cual será explicado más adelante. Este se encargará de mandar el mensaje a todos los gameobjects que estén suscritos a los mensajes de la escena activa, y cada uno de estos al recibirlo harán la función correspondiente. En nuestro caso los únicos gameobjects que recibirán mensajes son los CombinationButton (los botones de colores de input del jugador dentro del juego y de las filas de este), el GameManager y el AudioManager. Los primeros reciben el mensaje si está en modo daltónico o no, mostrando o no los números correspondientes. El GameManager por su parte se encarga de gestionar el estado del juego: entre otras funciones, redirige los inputs del jugador a las columnas de juego y desactiva un botón si es vuelto a pulsar; por lo que necesita recibir mensajes de pulsaciones de botón. Por último el AudioManager informa si se tiene que reproducir o parar un audio del audioSystem correspondiente.

El SceneManager es un singleton que se encarga de la comunicación entre escenas, mandar mensajes y añadir objetos a las escenas. Antes hemos visto el uso para el paso de mensajes y ahora veremos sus otras funcionalidades. Para el cambio de escenas este tiene un método llamado SetScene que es llamado por los gameobjects en su handle input que realizan un paso entre diferentes escenas (botones de elección de dificultad, etc.). Al llamar a este método del SceneManager se hace un cambio de escena activa, se llama al SetScene de esta nueva escena, que se encarga de crear los gameobjects (que a su vez estos gameobjects se añaden a la escena activa en ese momento) y se llama al SetScene de cada engine que al recibir un IScene informamos que se quiere realizar un cambio de escena. Al final del bucle principal comprobamos si se quiere cambiar de escena, y si es así seteamos la escena correspondiente con el método de Engine ManageChangeScene.

**Partes extras/opcionales realizadas:**

De las partes opcionales propuestas por los profesores no hemos realizado ninguna. Pero sí podemos señalar diferentes añadidos que hemos incluido en nuestra práctica:

-Posibilidad de en lugar de quitar colores de la solución al picar sobre ellos, usarlos como entrada de Input además de la línea inferior.

> Para activar esto hay que cambiar el booleano de la clase GameManager `clearInRows` a false.

-Posibilidad de que el tamaño de las filas escale con la proporción altura/número de filas. Permitiendo más filas que las dadas por el enunciado de la práctica, aunque sin scroll.

> Para activar esto hay que cambiar el booleano de la clase GameScene `scaleRowWithSpace` a true.