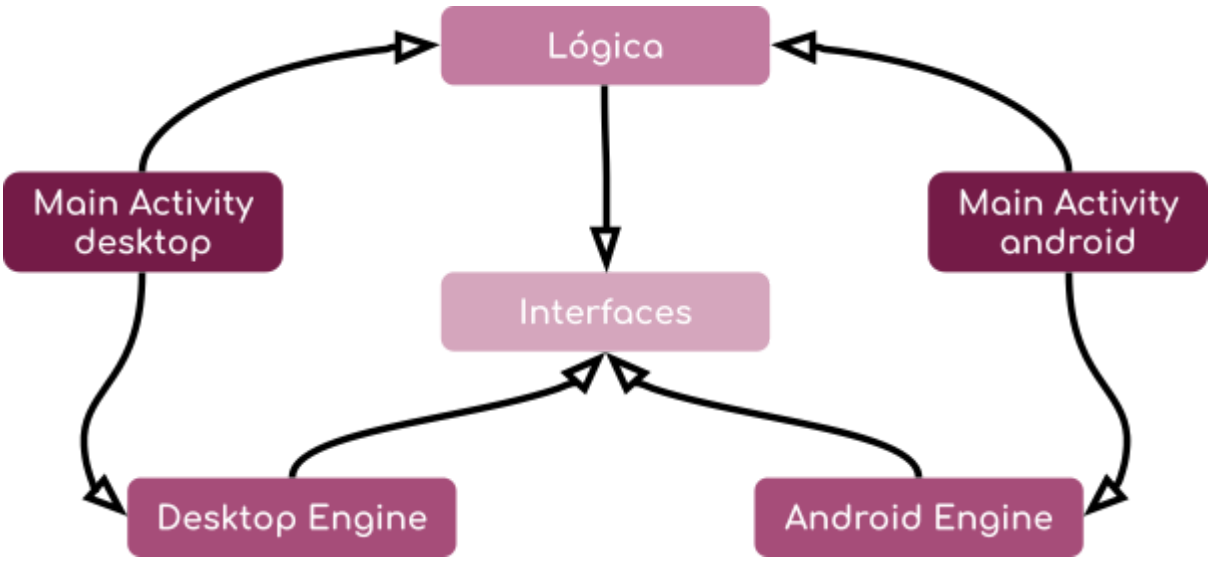
ESTRUCTURA DE LOS MÓDULOS



Interfaces que implementan los motores para que las use la lógica:

- **IAudio->** AudioDesktop, AudioAndroid
- **IEngine->** EngineDesktop,EngineAndroid
- **IFont->** FontDesktop, FontAndroid
- **IGraphics->** GraphicsDesktop, GraphicsAndroid
- **IImage->** ImageDesktop, ImageAndroid
- **ISound->** SoundDesktop, SoundAndroid
- **IInput->** InputDesktop, InputHandler(Desktop/Android), InputAndroid

Interfaces que implementa la lógica para que las usen los motores:

- **IGameObject->** Circles y Board
- **IScene->** EndScene, GameScene, Menu Scene

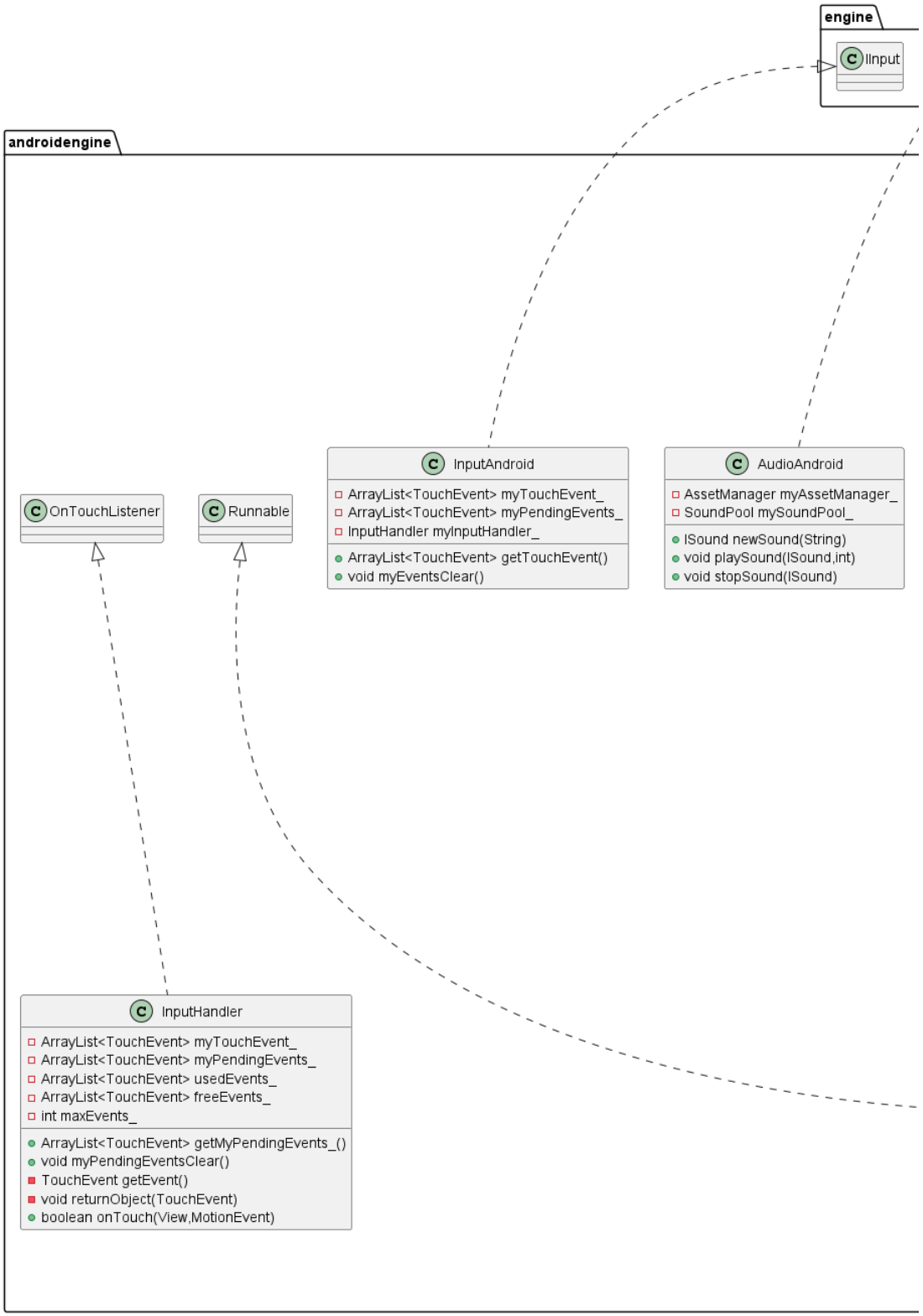
MÓDULOS DE LA PRÁCTICA:

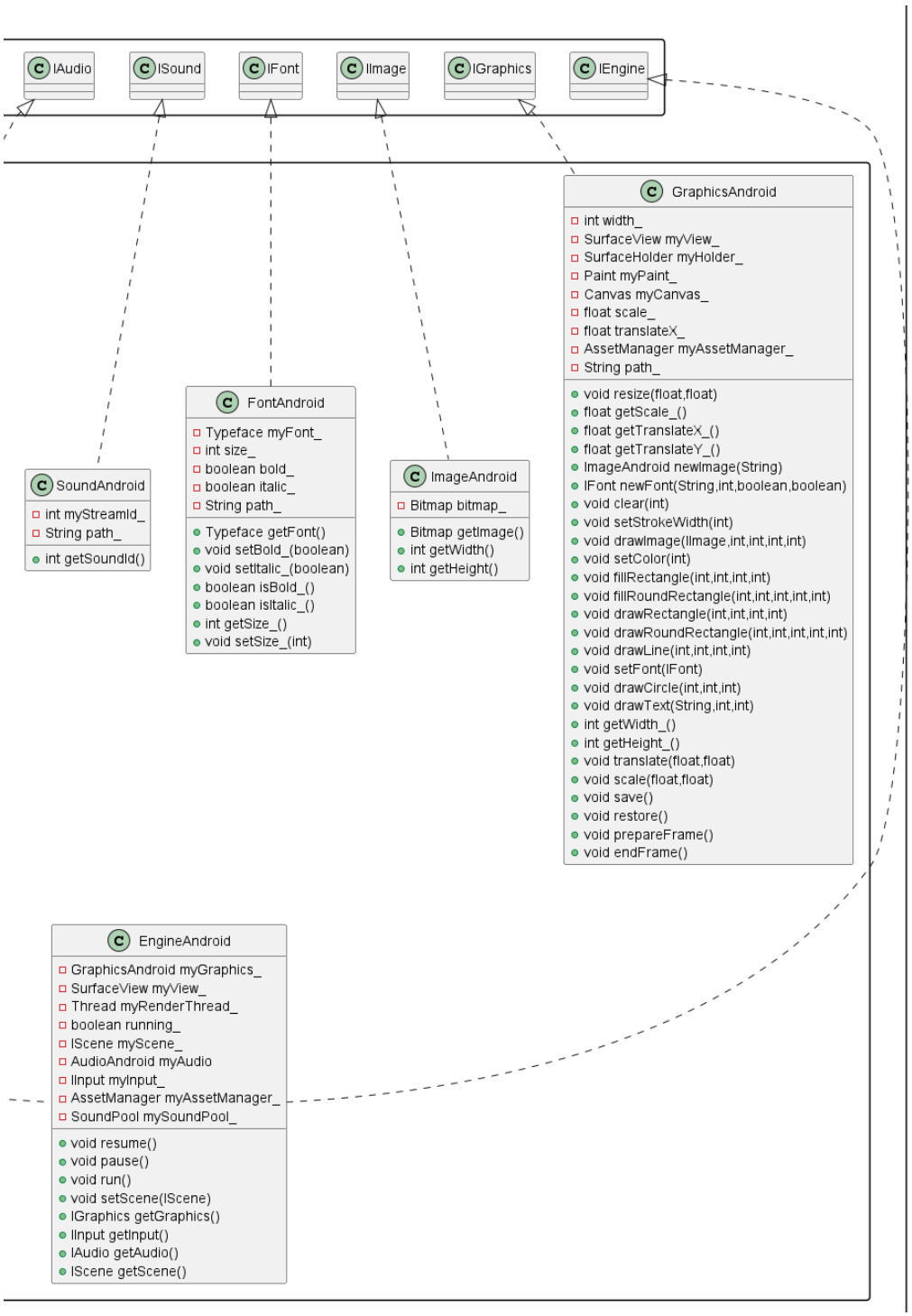
*Para visualizar mejor las imágenes UML mostradas en este documento, consultar la carpeta "UML" de la práctica

- **AndroidEngine:** Módulo que implementa el motor para la ejecución en Android. Tiene las siguientes clases:
    - **AudioAndroid:** Se utiliza para facilitar la creación, gestión y reproducción de objetos SoundAndroid.

- **EngineAndroid:** Es el motor de juego de Android. Se encarga de inicializar los recursos necesarios para el resto de motores así como gestionar escenas, la interfaz gráfica, la entrada del usuario, el audio, y, en general, la ejecución del juego mediante los métodos: "resume", "pause" y "run".
- **FontAndroid:** Se utiliza para representar y gestionar fuentes de texto en Android a través de un objeto Typeface. Permite cargar una fuente de texto desde el sistema de archivos de la aplicación y ajustar sus atributos, como el tamaño, la negrita y la cursiva.
- **GraphicsAndroid:** Se utiliza para proporcionar métodos encargados de dibujar formas, crear y mostrar imágenes , cargar fuentes de texto y proyectar los cambios en el objeto Canvas. También se encarga de gestionar la escala en caso de que cambie el tamaño de la ventana.
- **ImageAndroid:** Se utiliza para representar imágenes en formato de mapa de bits a través de un objeto Bitmap en Android. También permite obtener propiedades de la imagen, como su alto y su ancho.
- **SoundAndroid:** Se utiliza para cargar sonidos en la aplicación mediante el uso de un SoundPool. La clase ofrece un método para obtener el identificador del sonido cargado, de forma que pueda ser gestionado por AudioAndroid.
- **InputHandler:** Se utiliza para procesar los eventos táctiles de la aplicación guardando información como el tipo de evento que es, o las coordenadas en las que se ha detectado.Utiliza una lista de eventos pendientes y un pool de eventos para administrar la entrada de manera que, los objetos de nuestra clase TouchEvent, puedan ser reutilizables.
- **InputAndroid:** Se encarga de recopilar los eventos pendientes del InputHandler, los agrega a su propia lista y luego los borra del InputHandler. Además permite a otras partes de la aplicación acceder a los eventos registrados.

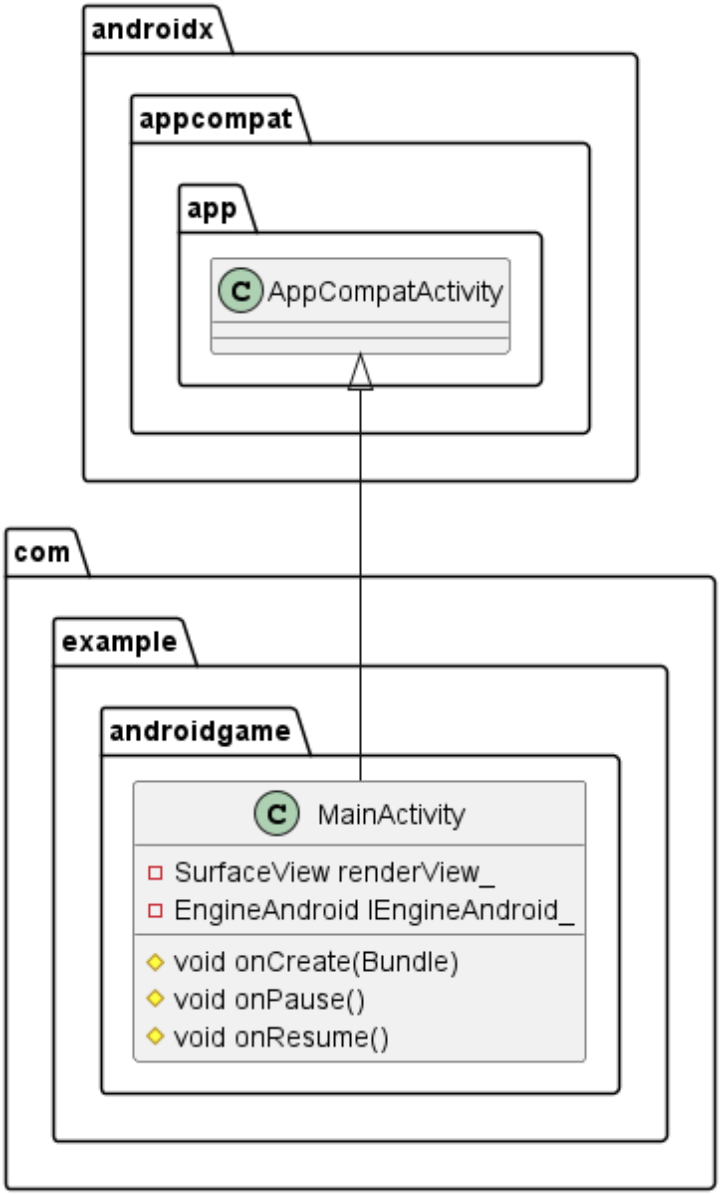
Diagrama UML para AndroidEngine:




IAudio
ISound
IFont
IImage
IGraphics
IEngine
GraphicsAndroid
int width_
SurfaceView myView_
SurfaceHolder myHolder_
Paint myPaint_
Canvas myCanvas_
float scale_
float translateX_
AssetManager myAssetManager_
String path_
void resize(float,float)
float getScale_()
float getTranslateX_()
float getTranslateY_()
ImageAndroid newImage(String)
IFont newFont(String,int,boolean,boolean)
void clear(int)
void setStrokeWidth(int)
void drawImage(IImage,int,int,int,int)
void setColor(int)
void fillRectangle(int,int,int,int)
void fillRoundRectangle(int,int,int,int,int)
void drawRectangle(int,int,int,int)
void drawRoundRectangle(int,int,int,int,int)
void drawLine(int,int,int,int)
void setFont(IFont)
void drawCircle(int,int,int)
void drawText(String,int,int)
int getWidth_()
int getHeight_()
void translate(float,float)
void scale(float,float)
void save()
void restore()
void prepareFrame()
void endFrame()
FontAndroid
Typeface myFont_
int size_
boolean bold_
boolean italic_
String path_
Typeface getFont()
void setBold_(boolean)
void setItalic_(boolean)
boolean isBold_()
boolean isItalic_()
int getSize_()
void setSize_(int)
SoundAndroid
int myStreamId_
String path_
int getSoundId()
ImageAndroid
Bitmap bitmap_
Bitmap getImage()
int getWidth()
int getHeight()
EngineAndroid
GraphicsAndroid myGraphics_
SurfaceView myView_
Thread myRenderThread_
boolean running_
IScene myScene_
AudioAndroid myAudio
IInput myInput_
AssetManager myAssetManager_
SoundPool mySoundPool_
void resume()
void pause()
void run()
void setScene(IScene)
IGraphics getGraphics()
IInput getInput()
IAudio getAudio()
IScene getScene()

- **AndroidGame:** Módulo puente entre la lógica y el motor de Android que se encarga de ejecutar la lógica del juego en el motor AndroidEngine.

Diagrama UML para AndroidGame:



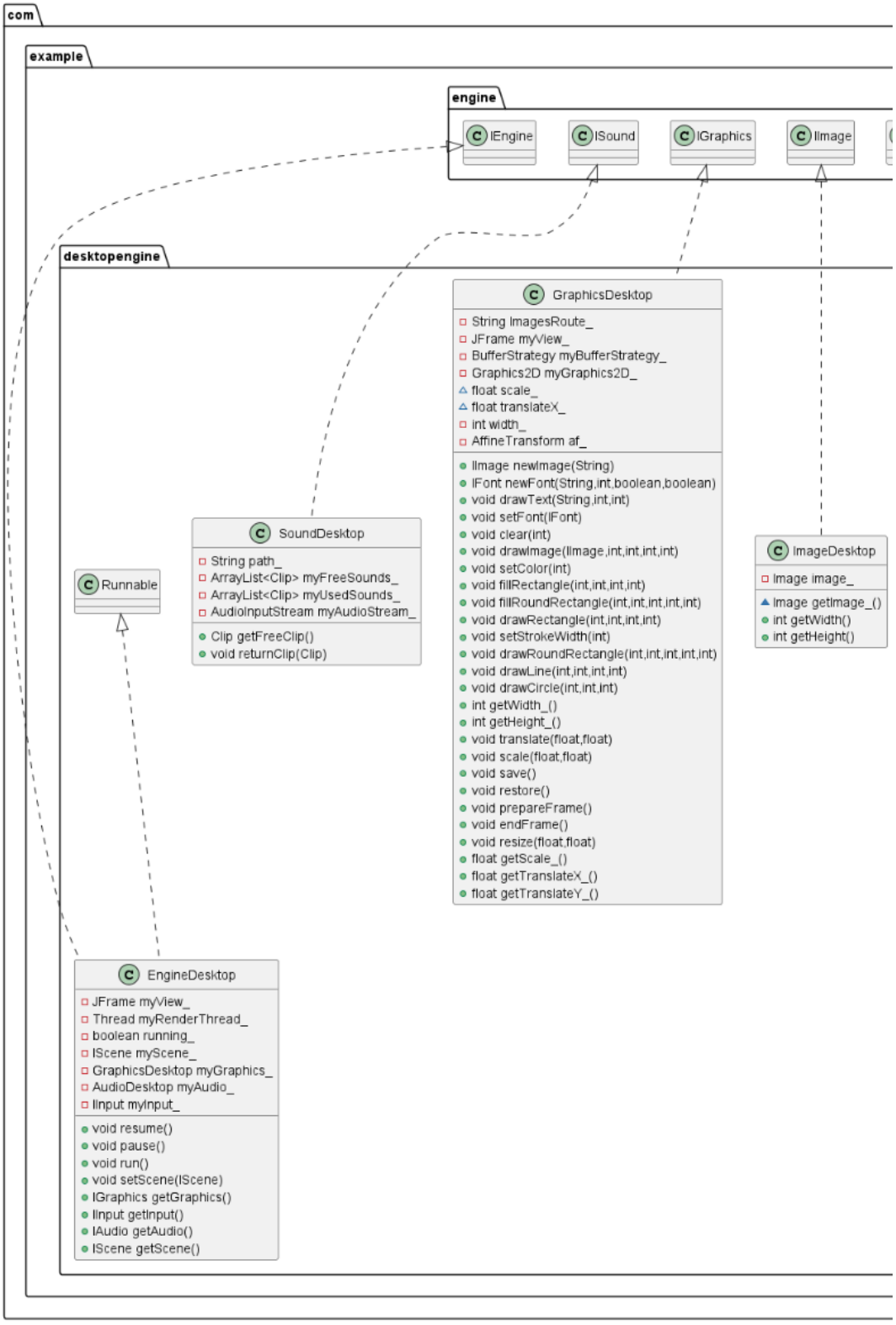
- **DesktopEngine:** Módulo que implementa el motor para la ejecución en PC. Tiene las siguientes clases, análogas a las explicadas anteriormente para AndroidEngine:

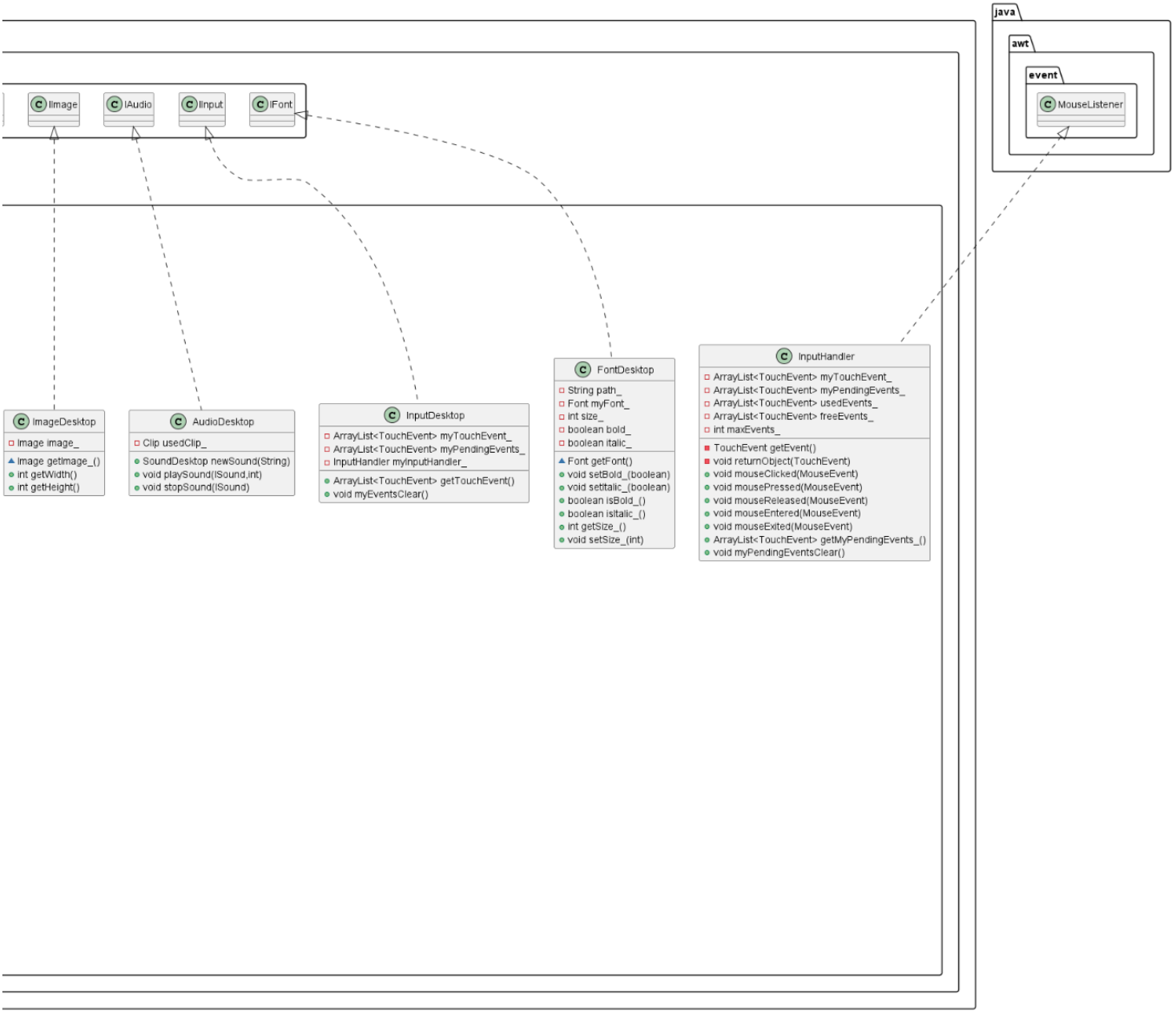
    - **AudioDesktop:** Se utiliza para facilitar la creación, gestión y reproducción de objetos SoundDesktop.
    - **EngineDesktop:** Es el motor de juego de escritorio. Se encarga de inicializar los recursos necesarios para el resto de motores así como gestionar escenas, la interfaz gráfica, la entrada del

usuario, el audio, y, en general, la ejecución del juego mediante los métodos: "resume", "pause" y "run".

- **FontDesktop:** Se utiliza para representar y gestionar fuentes de texto en la aplicación de escritorio a través de un objeto de Java, Font. Permite cargar una fuente de texto a través del objeto FileInputStream y ajustar atributos, como el tamaño, la negrita y la cursiva.
- **GraphicsDesktop:** Se utiliza para proporcionar métodos encargados de dibujar formas a través de la clase Graphics2D, crear y mostrar imágenes con el objeto Image , cargar fuentes de texto y proyectar los cambios a través del objeto BufferStrategy. También se encarga de gestionar la escala en caso de que cambie el tamaño de la ventana durante la ejecución.
- **ImageDesktop:** Se utiliza para representar imágenes a través de un objeto Image de Java. También permite obtener propiedades de la imagen, como su alto y su ancho, o la imagen en sí.
- **SoundDesktop:** Se utiliza para cargar sonidos en la aplicación de escritorio. Se hace uso de una pool propia con dos listas que guardan objetos de tipo Clip donde una gestiona los clips libres y la otra los que están en uso (simulando el comportamiento descrito en Android). La clase ofrece un método para obtener objetos de la pool que se encuentren libres, así como un método para devolverlos a la pool de forma que puedan ser reutilizados.
- **InputHandler:** Se utiliza para procesar los eventos de ratón de la aplicación de escritorio guardando información como el tipo de evento que es, o las coordenadas en las que se ha detectado.Utiliza una lista de eventos pendientes y un pool de eventos para administrar la entrada de manera que, los objetos de nuestra clase TouchEvent, puedan ser reutilizables.
- **InputDesktop:** Al igual que en Android, se encarga de recopilar los eventos pendientes del InputHandler, los agrega a su propia lista y luego los borra del InputHandler. Además permite a otras partes de la aplicación acceder a los eventos registrados.

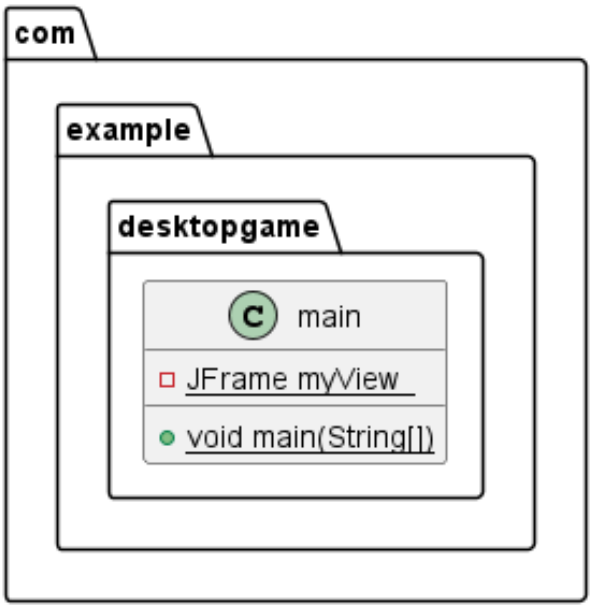
Diagrama UML para DesktopEngine:





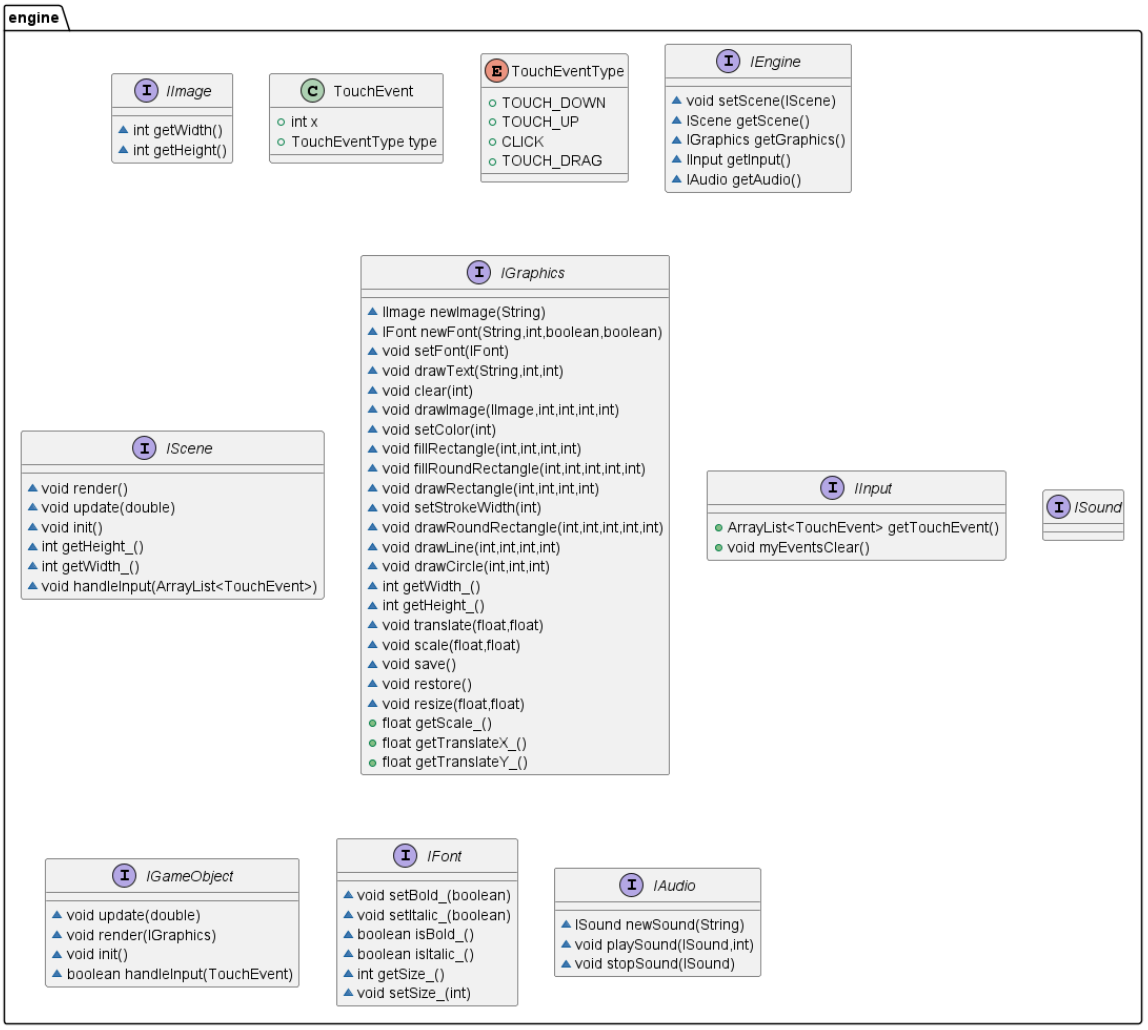
- **DesktopGame:** Módulo puente entre la lógica y el motor de PC, que se encarga de ejecutar la lógica del juego en el motor DesktopEngine.

Diagrama UML para DesktopGame:



- **Engine:** Módulo abstracto que define las interfaces, descritas en el enunciado de la práctica, para comunicar la lógica con los motores
    - IAudio
    - IEngine
    - IFont
    - IGraphics
    - IImage
    - ISound
    - IInput
    - IGameObject
    - IScene

Diagrama UML para Engine:



- **GameLogic:** Módulo que implementa la lógica del juego. Contiene las siguientes clases:

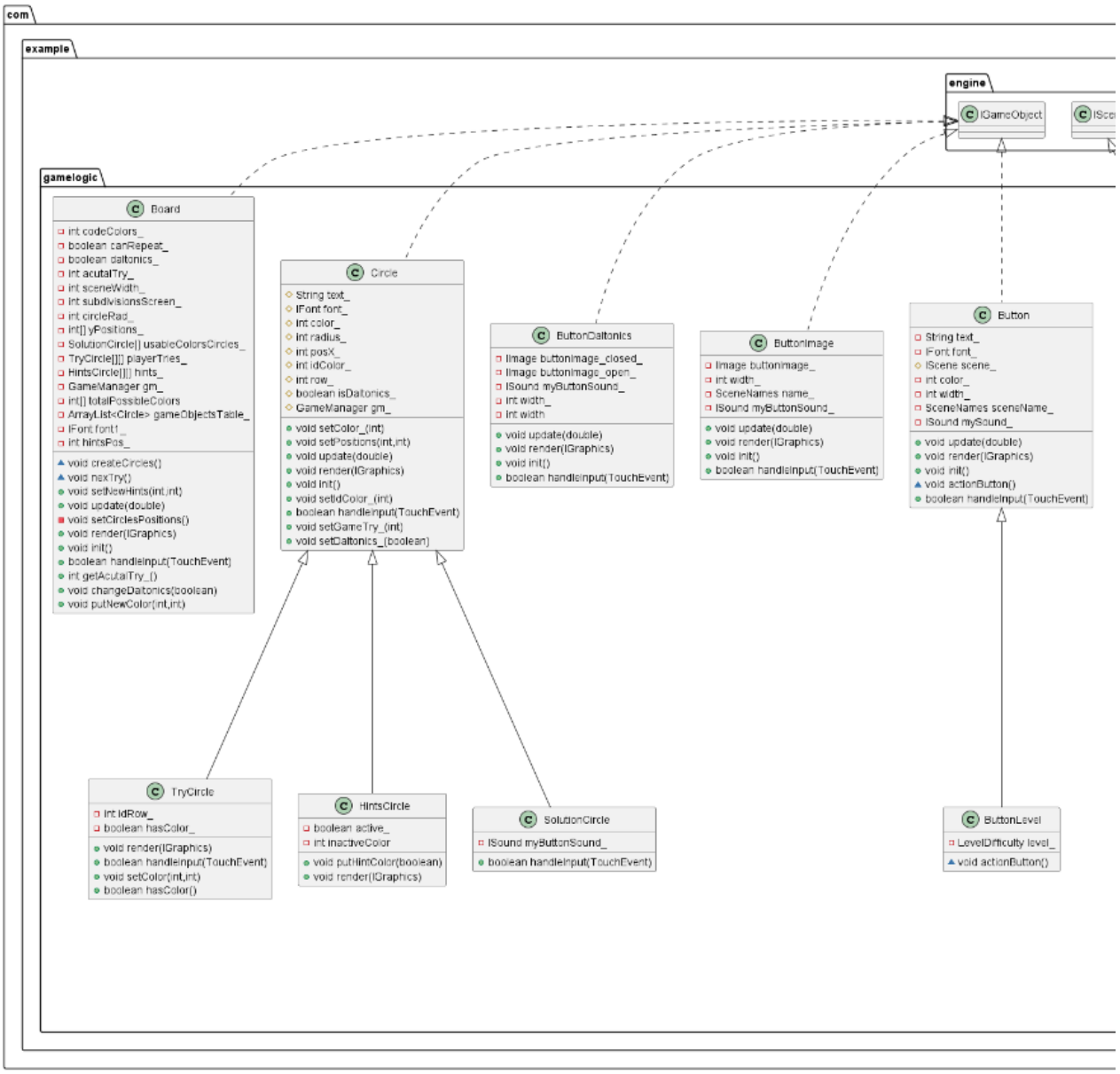
    - **Board:** Clase que gestiona la lógica del tablero, los intentos, y contiene los círculos de la solución y los colores disponibles
    - **Button:** Implementación general de los botones
        - **ButtonDaltonics:** botón que activa/desactiva los números encima de los círculos
        - **ButtonLevel:** botón que cambia de niveles
        - **ButtonImage:** Implementación general de los botones con imagen

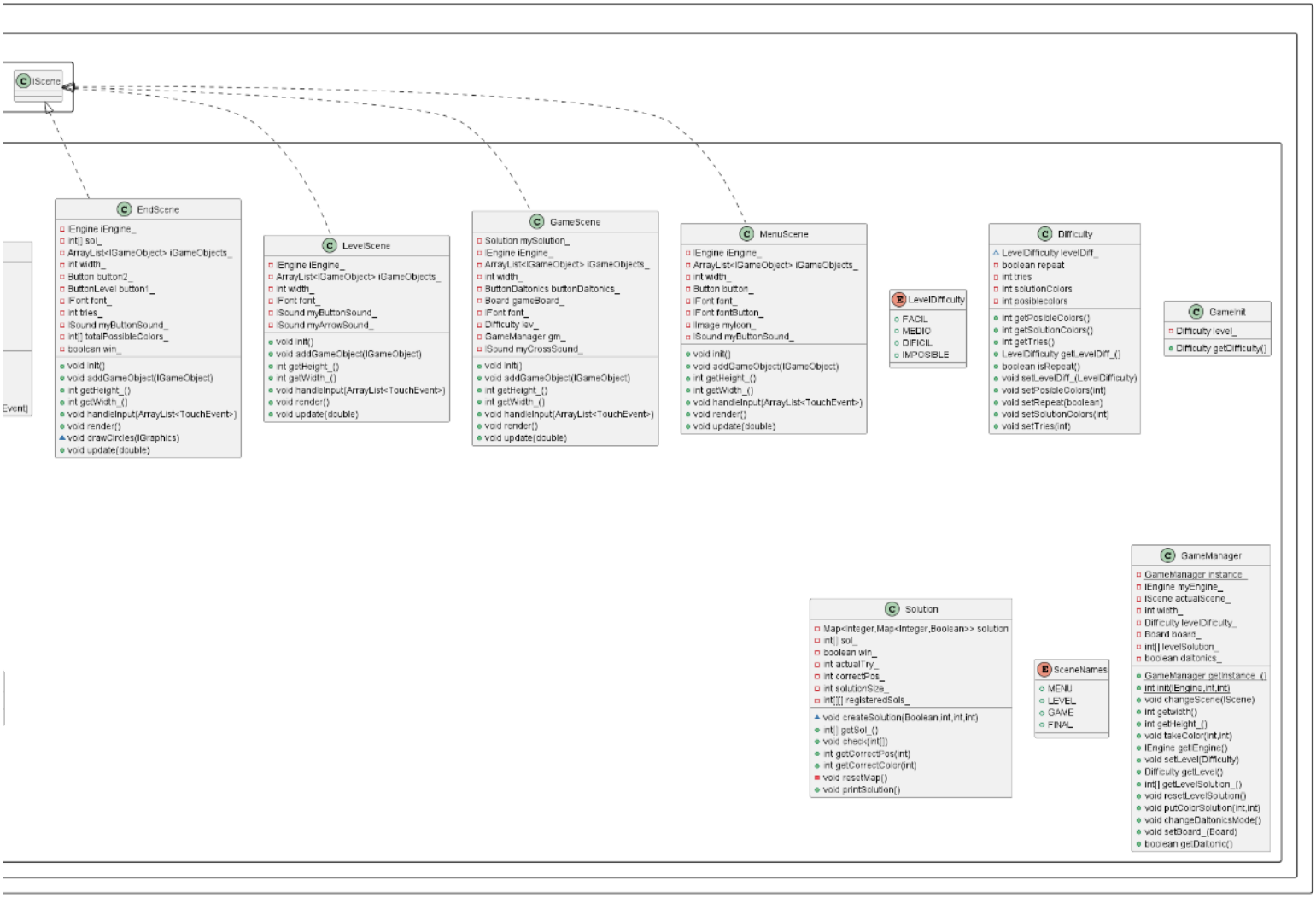
    - **Circle:** Game object que implementa de forma general los círculos para el tablero del juego

        - **HintsCircle:** círculos blancos/negros de las pistas
        - **TryCircle:** círculos de los intentos del jugador
        - **SolutionCircle:** círculos de colores para los intentos

- **EndScene:** Escena del final (ganar/perder) del juego
- **GameScene:** Escena de partida
- **MenuScene:** Escena inicial del juego
- **LevelScene:** Escena de selección de nivel
- **GameManager**
- **Solution:** Implementa la creación de la solución y comprueba los intentos del jugador
- **GameInit:** Almacena las variables de cada dificultad de partida y las distintas dificultades que puede tener el juego
- **SceneNames:** Enum con los nombres de las distintas escenas

Diagrama UML para GameLogic:




IScene
EndScene
LevelScene
GameScene
MenuScene
LevelDifficulty
FACIL
MEDIO
DIFICIL
IMPOSIBLE
Difficulty
GameInit
Solution
SceneNames
MENU
LEVEL
GAME
FINAL
GameManager