



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

*Desarrollo de un juego para móviles Android que facilite
el aprendizaje de las banderas de señales*

Grado en Ingeniería Mecánica

ALUMNO: Jesús Fernández Guaza

DIRECTOR: Norberto Fernández García

CURSO ACADÉMICO: 2018-2019

Universida_{de}Vigo



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

*Desarrollo de un juego para móviles Android que facilite
el aprendizaje de las banderas de señales*

Grado en Ingeniería Mecánica

Intensificación en Tecnología Naval

Cuerpo General

UniversidadeVigo

RESUMEN

A pesar de los avances tecnológicos en el ámbito de las comunicaciones navales, equipos radio, telefonía satélite, etc., las banderas de señales siguen teniendo un papel fundamental, pues permiten transmitir mensajes completos comprendidos por cualquier persona, independientemente del idioma y sin necesidad de ningún equipo. Por ejemplo, la bandera “Alpha” indica la presencia de buceadores en las inmediaciones.

Por esta razón el aprendizaje del sistema de señales es fundamental en la formación de cualquier marino. Por ello, la Escuela Naval Militar dota a sus alumnos con una baraja de cartas con las distintas señales. Sin embargo, este método de aprendizaje requiere disponer físicamente de las cartas para estudiar, por lo que, para estudiar en cualquier momento, los alumnos emplean aplicaciones para el móvil. Generalmente, estas aplicaciones disponen de escasos medios de evaluación y no contemplan la interacción entre usuarios. Teniendo en cuenta la importancia de la gamificación en el aprendizaje, se pretende desarrollar una aplicación que facilite que los usuarios aprendan las señales y permita que compitan entre ellos.

La aplicación desarrollada permitiría reemplazar a la “Baraja de Señales”, contribuyendo a mantener el carácter vanguardista de la Armada Española, su Responsabilidad Social en materia de medio ambiente y colaborando en la medida de lo posible en el gran reto al que se enfrenta en la actualidad: la Transformación Digital.

PALABRAS CLAVE

Aplicación móvil, Android, Banderas de señales, Aprendizaje, Gamificación

AGRADECIMIENTOS

En primer lugar, agradecer la paciencia de mis compañeros de camareta, Antonio González-Llanos Flethes y Fernando Quijano Martínez, por ofrecerse voluntarios continuamente a probar la aplicación durante su fase de desarrollo.

A mi tutor, D. Norberto Fernández García, por ser autor de la idea de este proyecto y por ayudarme a conseguir los objetivos marcados.

Agradecer también el apoyo recibido por todos mis compañeros de promoción, al prestarse voluntarios a probar y evaluar la aplicación. Así como por sus sugerencias, muchas de las cuales fueron implementadas en la aplicación.

Agradecer a mi amiga Eva Romero López, estudiante de Doble Grado en Educación Primaria e Infantil en la Universidad de Palencia, por ayudarme a enfocar didácticamente la aplicación y guiarme en el tema de la gamificación. A mi amiga María Carro Sanz, estudiante de Grado en Fotografía en la escuela TAI, adscrita a la Universidad Rey Juan Carlos, por ayudarme en el empleo de herramientas gráficas.

De manera más personal, querría agradecer a mi hermano, Alfredo Fernández Guaza, por ayudarme en mis primeros pasos en Java.

A todos, gracias.

CONTENIDO

Contenido	1
Índice de Figuras	3
Índice de Tablas.....	6
1 Introducción y objetivos	7
1.1 Contextualización.....	7
1.2 Orígenes del Código Internacional de Señales	9
1.3 Señales Navales OTAN	11
1.4 Objetivo del TFG	12
1.5 Organización de la memoria	14
2 Estado del arte	15
2.1 Gamificación: el aprendizaje divertido	15
2.2 Aplicaciones existentes	16
2.2.1 Nautical Flags and Signals.....	16
2.2.2 ICS Flag Trainer	16
2.2.3 Nautical Flags Helper	17
2.2.4 Marinus RIPA	18
2.2.5 Comparativa.....	19
2.3 Herramientas utilizadas.....	19
2.3.1 Formato SVG	19
2.3.2 Inkscape	20
2.3.3 Java	21
2.3.4 Android	23
2.3.5 Entornos de desarrollo para Android.....	25
3 Desarrollo del TFG.....	29
3.1 Requerimientos del Proyecto	29
3.2 Nociones básicas de Android.....	29
3.2.1 Componentes básicos.....	29
3.2.2 Estructura de un proyecto Android.....	30
3.2.3 Niveles de API.....	31
3.3 Ficheros que componen una aplicación	32
3.3.1 AndroidManifest.....	32
3.3.2 build.gradle	34
3.3.3 Ficheros de recursos	35

3.3.4 Actividades	40
3.4 Bluetooth en Android.....	46
3.4.1 Permisos necesarios	46
3.4.2 Dispositivos vinculados	46
3.4.3 Conexión de los dispositivos	47
3.5 Desarrollo de la aplicación.....	49
3.5.1 Splash Screen.....	50
3.5.2 Menú principal.....	51
3.5.3 Modo aprendizaje	53
3.5.4 Modo juego.....	59
3.5.5 Actividades auxiliares.....	68
4 Resultados	73
4.1 Instalación de la aplicación	73
4.2 Funcionamiento de la aplicación.....	74
4.2.1 Aprendizaje de las señales	74
4.2.2 Evaluación de los conocimientos adquiridos.....	75
4.3 Plan de pruebas de la aplicación	75
4.3.1 Herramienta Monkey	76
4.3.2 Búsqueda metódica de errores	77
4.3.3 Resultados de la búsqueda de errores	80
5 Conclusiones y líneas futuras	81
5.1 Conclusiones	81
5.2 Líneas futuras	82
6 Bibliografía.....	83
Anexo I: Criterios de comparación de las aplicaciones.....	88
Anexo II: Relación de señales correspondientes al Código Internacional de Señales	89

ÍNDICE DE FIGURAS

Figura 1-1 Banderas del Código Internacional de Señales [2]	8
Figura 1-2 Símbolos Scott [2]	8
Figura 1-3 Señales del método del semáforo [3].....	9
Figura 1-4 Extracto del libro <i>Signal-book for the Ships of War</i> (1779) [4]	9
Figura 1-5 Extracto de <i>Code of Signals for the Merchant Service</i> (1817) [5]	10
Figura 1-6 Extracto de <i>Comercial Code Flags</i> (1857) [6]	10
Figura 1-7 Señales Navales OTAN [7]	12
Figura 1-8 Baraja de señales.....	12
Figura 2-1 Aprendizaje significativo.....	15
Figura 2-2 Pantallas de Nautical Flags and Signals [11].....	16
Figura 2-3 Pantallas de ICS Flag Trainer [12]	17
Figura 2-4 Pantallas de Nautical Flags Helper [13]	18
Figura 2-5 Marinus RIPA [14]	18
Figura 2-6 Diferencia entre imágenes rasterizadas y vectorizadas [16].....	20
Figura 2-7 Diseño de “Code” con Inkscape	21
Figura 2-8 La plataforma Java [19].....	22
Figura 2-9 Pila de software de Android [22].....	24
Figura 2-10 Xamarin, desarrollo multiplataforma [24].....	25
Figura 2-11 IntelliJ IDEA IDE [25]	26
Figura 2-12 Interfaz de Android Studio	26
Figura 2-13 Sistema operativo dominante por países 2019 [29].....	27
Figura 2-14 Distribución del mercado de Android e iOS en España 2019 [29].....	27
Figura 3-1 Vista <i>Android</i> de los archivos.....	31
Figura 3-2 Vista <i>Project</i> de los archivos	31
Figura 3-3 Extracto del <i>AndroidManifest.xml</i>	33
Figura 3-4 Extracto del <i>build.gradle</i>	34
Figura 3-5 Cantidad relativa de versiones de plataforma Android [26]	35
Figura 3-6 Recursos <i>drawable</i>	36
Figura 3-7 Extracto de <i>ic_a.xml</i>	36
Figura 3-8 Ejemplo de subdirectorios <i>string</i> para múltiples idiomas [36].....	36
Figura 3-9 Extracto de <i>strings.xml</i>	37
Figura 3-10 Resumen de elementos de diseño empleados	38
Figura 3-11 Extracto de <i>activity_main.xml</i>	39
Figura 3-12 Panel de diseño correspondiente a <i>activity_main.xml</i>	39

Figura 3-13 Ejemplo de un <i>intent</i>	40
Figura 3-14 Extracto de <i>DiccionarioActivity.java</i>	41
Figura 3-15 Método <i>findViewById</i> de la <i>BTStartActivity.java</i>	41
Figura 3-16 Ciclo de vida de una actividad [39]	42
Figura 3-17 Método <i>shuffleArray</i> de <i>MetodosGlobales.java</i>	43
Figura 3-18 Método <i>especificarPregunta</i> de <i>MetodosGlobales.java</i>	44
Figura 3-19 Método <i>dobleClick</i> de <i>MetodosGlobales.java</i>	45
Figura 3-20 Extracto de <i>VariablesGlobales.java</i>	45
Figura 3-21 Obtención de los dispositivos vinculados.....	47
Figura 3-22 Clases requeridas para establecer una conexión Bluetooth	47
Figura 3-23 Diagrama de flujo del establecimiento de conexión Bluetooth	48
Figura 3-24 Ejemplo del funcionamiento de un <i>handler</i>	49
Figura 3-25 Funcionamiento de la aplicación	50
Figura 3-26 <i>Splash Screen</i>	51
Figura 3-27 Extracto de <i>SplashActivity.java</i>	51
Figura 3-28 Menú principal.....	52
Figura 3-29 Extracto de <i>MainActivity.java</i>	52
Figura 3-30 Modo aprendizaje	53
Figura 3-31 Extracto de <i>AprendizajeActivity.java</i>	54
Figura 3-32 Diccionario de banderas	54
Figura 3-33 Extracto de <i>DiccionarioActivity.java</i>	55
Figura 3-34 Diseño de cada elemento de <i>CustomListview</i>	55
Figura 3-35 Adivinar las banderas	55
Figura 3-36 Funcionamiento básico de <i>AdivinarActivity</i>	56
Figura 3-37 Método <i>onResume</i> de <i>AdivinarActivity.java</i>	57
Figura 3-38 Método <i>cartaPulsada</i> de <i>AdivinarActivity.java</i>	58
Figura 3-39 Método <i>cargarDatos</i> de <i>AdivinarActivity.java</i>	58
Figura 3-40 Modo juego.....	59
Figura 3-41 Extracto de <i>JuegoActivity.java</i>	59
Figura 3-42 Método <i>determinarDificultad</i> de <i>LocalActivity.java</i>	60
Figura 3-43 Reparto de preguntas por medio del método <i>determinarDificultad</i>	60
Figura 3-44 Método <i>onResume</i> de <i>LocalActivity.java</i>	61
Figura 3-45 Test Bluetooth.....	62
Figura 3-46 Escuchador correspondiente al botón de ayuda de <i>BTStartActivity.java</i>	62
Figura 3-47 Método <i>prepareToConnect</i> de <i>BTStatActivity.java</i>	63
Figura 3-48 Escuchador de los dispositivos enlazados de <i>BTStartActiviy.java</i>	63

Figura 3-49 <i>Radio Button</i> del modo Bluetooth	64
Figura 3-50 Extracto del <i>Handler</i> de <i>BTStartActivity.java</i>	65
Figura 3-51 Pantallas de evaluación.....	66
Figura 3-52 Método <i>onClick</i> de <i>BanderaActivity.java</i>	67
Figura 3-53 Métodos <i>acierto</i> y <i>fallo</i> de <i>BanderaActiviy.java</i>	67
Figura 3-54 Método <i>onResume</i> de <i>BanderaActivity.java</i>	68
Figura 3-55 Ventana <i>PopUp</i>	69
Figura 3-56 Extracto del método <i>onCreate</i> de <i>PopUpActivity.java</i>	69
Figura 3-57 Extracto de <i>styles.xml</i>	69
Figura 3-58 Actividad resultados	70
Figura 3-59 Método <i>onCreate</i> de <i>ResultadoActivity.java</i>	71
Figura 3-60 Método <i>onResume</i> de <i>ResultadoActivity.java</i>	71
Figura 3-61 Método <i>onKeyDown</i> de <i>ResultadoActivity.java</i>	72
Figura 4-1 Generar una APK.....	73
Figura 4-2 Código QR para descargar el archivo APK.....	74
Figura 4-3 Permitir orígenes desconocidos en Android 7.1.1 y versiones anteriores	74
Figura 4-4 Icono de la aplicación ENM Banderas	74
Figura 4-5 Empleo de la herramienta Monkey.....	77
Figura 4-6 Extracto del método <i>onCreate</i> de <i>PopUpActivity.java</i>	78
Figura 4-7 Extracto del <i>switch</i> de <i>JuegoActivty.java</i>	79
Figura 4-8 <i>BroadcastReceiver</i> de <i>BTStartActivity.java</i>	79

ÍNDICE DE TABLAS

Tabla 1-1 Bandera de Señales, Orden Diaria. [8].....	13
Tabla 2-1 Comparativa de aplicaciones existentes.....	19
Tabla 3-1 Versiones Android	32
Tabla 5-1 Comparativa de aplicaciones de aprendizaje de banderas de señales.....	81

1 INTRODUCCIÓN Y OBJETIVOS

En este capítulo se introducirá brevemente el origen del Código Internacional de Señales y de las banderas que lo componen. Posteriormente se detallarán los objetivos y la organización del presente proyecto.

1.1 Contextualización

El Código Internacional de Señales [1] se centra en mensajes de navegación y seguridad, incluyendo un apartado médico. Existen mayoritariamente tres tipos de mensajes: los formados por una sola letra, corresponden a señales comunes, importantes o urgentes; los formados por dos letras, correspondientes a mensajes ordinarios, los cuales a su vez pueden ser seguidos de números que complementen el mensaje; y los mensajes formados por trigramas cuya primera letra es “Mike”, correspondientes a señales médicas.

Este código permite que los buques de todo el mundo sean capaces de comunicarse entre sí de una forma básica sin necesidad de tan siquiera hablar el mismo idioma.

Los mensajes pueden ser transmitidos bien por fonía o por medios visuales:

- Por fonía los mensajes han de ser transmitidos a través de una radio, normalmente a través del servicio móvil marítimo. Para facilitar el correcto entendimiento entre personas con distintos idiomas, cada letra y número se sustituye por una palabra con pronunciación característica.
- La comunicación por señales visuales es más amplia y cabe diferenciar tres métodos: banderas, Scott y semáforo.

La comunicación por banderas consta de cuarenta señales, que se muestran en la Figura 1-1, con las que se representan:

- Caracteres del abecedario inglés.
- Números decimales.
- Tres gallardetes repetidores, los cuales permiten realizar cualquier combinación de cuatro caracteres con un solo juego de banderas.
- El gallardete característico “Code”, se emplea para indicar si el mensaje ha sido recibido e interpretado y en el caso de buques de guerra indica que el mensaje a continuación pertenece al Código Internacional de Señales.

INTERNATIONAL FLAGS AND PENNANTS					
ALPHABET FLAGS			NUMERAL PENNANTS		
Alfa		Kilo		Uniform	
Bravo		Lima		Victor	
Charlie		Mike		Whiskey	
Delta		November		Xray	
Echo		Oscar		Yankee	
Foxtrot		Papa		Zulu	
Golf		Quebec		SUBSTITUTES	
Hotel		Romeo		1st Substitute	
India		Sierra		2nd Substitute	
Juliett		Tango		3rd Substitute	
			CODE (Answering Pennant or Decimal Point)	0	

Figura 1-1 Banderas del Código Internacional de Señales [2]

El método Scott consiste en la transmisión de las letras y números por medio de destellos de distinta duración. En la Figura 1-2 se puede ver qué símbolo corresponde a cada letra y número.

ALPHABET				NUMERALS			
A	•-	N	-•	1	•----	6	-----•
B	----	O	---	2	••----	7	-----••
C	-•••	P	•-••	3	•••----	8	-----•••
D	--•	Q	---•	4	••••-	9	-----••••
E	•	R	•-•	5	•••••	0	-----
F	••••	S	•••				
G	---•	T	-				
H	••••	U	••-				
I	••	V	•••-				
J	•----	W	•-•				
K	-•-	X	-••-				
L	••••	Y	-•••				
M	--	Z	---•				

Figura 1-2 Símbolos Scott [2]

El método del semáforo (Figura 1-3), es un sistema de comunicación en el cual se emplea la posición de los brazos para representar las distintas letras del abecedario. Generalmente se realiza sosteniendo en las manos dos banderas “Oscar”, pero también pueden utilizarse antorchas para realizarlo durante la noche. El aprendizaje de este método es relativamente sencillo pues las señales están ordenadas de manera que recuerdan el movimiento de las agujas de un reloj.



Figura 1-3 Señales del método del semáforo [3]

Antes de la llegada de la radio y el Scott, la señalización por banderas era la única forma que los buques tenían para comunicarse. Incluso a día de hoy este método es preferible frente a otros pues es discreto y a la vez entendible por todos los que estén lo suficientemente cerca. Permite enviar mensajes sin que nadie que no esté en el alcance visual pueda interceptarlos. Además, permite enviar mensajes completos de forma muy simple, por ejemplo, para indicar por radio que estoy con buzos en el agua y que todo el mundo se mantenga a una distancia prudente habría que indicar: el distintivo de llamada del buque, que está empleando buzos y dónde se encuentra. Mientras que si se realiza por banderas simplemente basta con izar la bandera “Alpha”, lo que además permite que personal que se encuentre en las inmediaciones o embarcaciones de menor porte sin equipos radio puedan recibir el mensaje.

1.2 Orígenes del Código Internacional de Señales

Los orígenes del Código Internacional de Señales actual, se remontan al siglo XVII en Europa, concretamente, en Inglaterra. Las primeras señales de banderas entre buques datan de 1653. No obstante el sistema no se formalizó hasta el año 1779 con el *Signal-book for the Ships of War* (en español, Libro de Señales para los Buques de Guerra) [4]. Como se muestra en la Figura 1-4, el sistema constaba de dos tipos de señales: las que correspondían a una sola bandera (el código disponía de veinticinco banderas numeradas entre sí) y las que correspondían a varias banderas formando números que posteriormente se asociaban a un mensaje.

SIGNALS with SINGLE FLAGS.				7
Colour.	Art.	Purport.	SIGNIFICATION.	
	1	Enemy in fight.	NUMERAL FLAGS. —To denote an Enemy in Sight. The Bearing will be shewn occasionally by the proper Com-pafs signal. N. B. The Flag will be shewn with a common Pendant over, by Flag-Officers of the Red Squadron, having occasion to make this signal.	

36 NUMERAL SIGNALS.				
No. of Signal.	Purport.	SIGNIFICATION.	Instruct.	
			Pa.	Art.
148.	Pilot.	Ships having a Pilot or other qualified person on board to lead the Fleet through any intricate Navigation wherein it may be then engaged. N. B. If respecting a part of the Fleet only the suitable distinguishing signals will be made herewith.		
149	Port. for the nearest.	To proceed for the nearest Port. If meant to repair to any assigned Rendezvous, the distinguishing signal thereof will be at the same Time made. - - - -		

Figura 1-4 Extracto del libro *Signal-book for the Ships of War* (1779) [4]

No fue hasta 1817 cuando el capitán Frederick Marryat publicó *Code of Signals for the Merchant Service* (en español, Código de Señales para la Marina Mercante) [5], el primer sistema general de señalización para buques mercantes. Este libro contenía extensas listas que relacionaban combinaciones de banderas con distintos buques de guerra, mercantes, faros, puertos, vocabulario y frases preestablecidas. A modo ilustrativo, como se puede apreciar que en la Figura 1-5, el puerto de Ferrol tenía asignado el código numérico 3016.

HEADLANDS, PORTS, AND HARBOURS. PART IV.		
RENDEZVOUS FLAG		
To be hoisted over the Number.		
		
Nº 1.	Nº 2.	Nº 3.
		
Nº 5.	Nº 6.	Nº 7.
		
Nº 9.	Nº 10.	Telegraph Flag.
		
Union Jack.	First Distress Pendant.	Second Distress Pendant.

2706 Don R.	2871 Eastport, Cape	2954 Earope
2708 Dondra Head	2873 Eastham	2956 Europe Island
2801 Donnegal	2874 Ebeltoft	2957 Eustatia, St.
2803 Dordrecht	2875 Ebro River	2958 Exe River
2804 Dordt	2876 Eckernförde	2960 Exmouth
2805 Doubtful Harbour	2879 Edam	2961 Exmouth Gulf
2806 Doubtless Bay	2890 Eday	2963 Fanborg
2807 Douglas	2891 Edinburgh	2964 Fair Island
2809 Douglas Bay	2893 Egg Harbour	2965 Falkland Isles
2810 Douro River	2894 Egina	2967 Sound
2813 Dover	2895 Egmont I.	2968 Falmouth
2814 Dovey River	2896 Egmont Cape	2970 False Bay
2815 Downs	2897 Bay	2971 False Cape
2816 Dragon's Mouth	2901 Egypt	2973 Falster I.
2817 Dram	2903 Elba	2974 Faou I.
2819 Drogheda	2904 Elbe River	2975 Fans
2830 Drontheim	2905 Elgin	2976 Farewell C., N.Z.
2831 Drummond I.	2906 Elsinour	2978 Greenland
2834 Dublin	2907 Elizabeth Bay	2980 Faro Sound
2835 Duddon	2908 Elizabeth Port	2981 Farquhar Inlet
2836 Ducies I.	2910 Embden	2983 Faversham
2837 Dumbarton	2913 Encounter Bay	2984 Fayal
2839 Dumfries	2914 Endeavour Straits	2985 Fear Cape
2840 Duna River	2915 Engano I.	2986 Pecamp
2841 Dunbar	2916 Engelholm	2987 Fernando Po Isle
2843 Dundalk	2917 England	3012 Ferret Cape
2845 Dundee	2918 English Bay (3014)	3014 Ferro I.
2846 Duudrum	English Channel,	3015 Ferroe Isles
2847 Dungarvon	2930 English Roads	3016 Ferrol
2849 Dungarvon Bay	2931 Entry I., N.Z.	Fez, 3016
2850 Dronmanus Bay	2934 Equator	3017 Fidallah
2851 Dungeness	2935 Equinoctial Line	3018 Fidgee Islands
2853 Dunkirk	2936 Erie	3019 Fifeness

Figura 1-5 Extracto de *Code of Signals for the Merchant Service* (1817) [5]

No fue hasta 1857 cuando el *British Board of Trade* (en español, Ministerio de Comercio Británico) publicó el *Commercial Code Flags* (en español, Código Comercial de Señales), sustituyendo gradualmente el código de Marryat. Este nuevo código empleaba banderas alfabéticas, como las que se muestran en la Figura 1-6, en lugar de las numéricas. Este sistema consistía en una bandera bifurcada, cinco triangulares y trece rectangulares, dando lugar a dieciocho letras, correspondientes al abecedario completo salvo X, Y, Z y las vocales. Las letras X, Y y Z fueron omitidas debido a que son poco comunes en los idiomas europeos, mientras que las vocales fueron omitidas para evitar que palabras censuradas pudieran ser deletreadas.

El código realizaba combinaciones de dos, tres o cuatro banderas para enviar los distintos mensajes, salvo los mensajes de afirmativo, negativo y cuarentena que consistían de una sola bandera. Este código permitía comunicar hasta 70.000 mensajes distintos.



Figura 1-6 Extracto de *Commercial Code Flags* (1857) [6]

En 1887 el *Comercial Code Flags* fue revisado por el *British Board of Trade*. Asimismo, este fue modificado en la Conferencia Internacional de 1889 en Washington D.C. pasando a contener ocho nuevos símbolos, correspondientes a las cinco vocales y a las letras X, Y y Z (omitidas en el *Comercial Code Flags* de 1857).

Tras la Primera Guerra Mundial, en la Conferencia Radiotelegráfica Internacional de 1927 de Washington D.C. se decidió revisar el código y editarlo en siete idiomas: inglés, alemán, español, francés, italiano, japonés y noruego. Esta nueva modificación fue adoptada por la Conferencia Radiotelegráfica Internacional de Madrid en 1932. Esta edición fue publicada oficial y mundialmente el 1 de enero de 1934. Constaba de dos volúmenes, uno de señales visuales y otro radiotelegráfico. En el segundo volumen se incluyeron mensajes relacionados con la aviación y la medicina. Asimismo, se introdujeron trece nuevas banderas y los gallardetes empleados para las letras C, D, E, F y G fueron sustituidos por banderas cuadradas. Estos gallardetes pasaron a representar los números 1, 2, 3, 4 y 5 respectivamente. Se introdujeron otras cinco nuevas banderas para representar los números 6, 7, 8, 9, y 0 así como tres gallardetes para los tres repetidores. Estas banderas son las que se emplean en la actualidad, tal y como se puede ver en Figura 1-1.

Tras la Segunda Guerra Mundial, en 1947, la UIT (*Union Internationale des Télécommunications*, en español, Unión Internacional de Telecomunicaciones) sugirió que el Código Internacional de Señales debería estar bajo la responsabilidad de la IMCO, actual IMO (*International Maritime Organization*, en español, Organización Marítima Internacional). En 1961 el código fue nuevamente revisado, decidiéndose incluir en esta versión los idiomas ruso y griego a los siete anteriores.

En 1964, siguiendo las recomendaciones del SOLAS (*Safety of Live at Sea*, en español, Convenio Internacional para la Seguridad de la Vida Humana en el Mar), se realizaron profundos cambios en el código. Se desestimó el método que contemplaba la transmisión de mensajes palabra por palabra y se juntaron los volúmenes de señales visuales y radiotelegráficas. Este código fue adoptado finalmente en 1965.

En la actualidad, el Código Internacional de Señales depende de la IMO, cuya última edición impresa fue publicada en 2005. [1]

1.3 Señales Navales OTAN

Con el fin de suplir necesidades en el control de las maniobras y despliegues, la OTAN (Organización del Tratado del Atlántico Norte) emplea una combinación del Código Internacional de Señales y banderas específicas de la OTAN, todas ellas recogidas en la Figura 1-7. Para ello dispone de:

- Banderas numerales, distintas a los banderines del Código Internacional de Señales.
- Cuarto repetidor, permite realizar cualquier combinación de cinco caracteres con un solo juego de banderas.
- Banderas indicadoras de unidad: flotilla, escuadrón, división y subdivisión.
- Banderines de gobierno, que alteran el significado de la señal: Preparativo, interrogativo y negativo.
- Banderas de maniobra: cortina, formación, rumbo, giro, estribor, babor, velocidad, puesto y emergencia.
- Banderín especial: designación, indicador de que el texto mostrado a continuación es lenguaje en claro.

FLAGS AND PENNANTS SINGLE & INTERNATIONAL MEANINGS			
A Divers/Friendly underwater demolition personnel down. Intn'l - Diver down. Keep well clear at slow speed.	K Personnel working aloft. Intn'l - I wish to communicate with you.	T Time indicator. Intn'l - Keep clear of me I am engaged in pair trawling.	3 Boat signal. (Steer away)
B Weapons practice. Fuelling or transferring explosives. Intn'l - Taking in, discharging or carrying dangerous goods.	L Radhaz/Hero warning. Intn'l - You should stop your vessel instantly.	U Anchoring, Mooring & Weighing. Intn'l - You are running into danger.	4 ASW Exercises.
C Affirmative. Intn'l - Yes, (Affirmative)	M Medical/Dental Guard Duty ship. Disregard my movements. Intn'l - My vessel is stopped.	V Streaming/Recovering towed sonic devices - not including minesweeping equipment. Intn'l - I require assistance.	5 Breakdown.
D Degaussing. Intn'l - Keep clear. I am manoeuvring with difficulty.	N Your movements not understood. Not keeping Visual watch. Intn'l - No (Negative)	W Flag Hoist information addressee. Intn'l - I require medical assistance.	6 Act at your discretion.
E - Intn'l - I am altering my course to Starboard.	O Man overboard. Intn'l - Man overboard.	X Evolution or Exercise completed. X-ray tack (Signal). Carry out for Exercise the meaning of the signal. Intn'l - Stop carrying out your intentions & watch for my signals.	7 AAW Action Table.
F Flight operations. Intn'l - I am disabled. Communicate with me.	P General recall. Position indicator. Intn'l - Recall. All persons to repair onboard. Vessel about to sail.	Y Acknowledge. OTC's Location. Visual Communication Duty Ship. Intn'l - I am dragging my anchor.	8 Boat signal. (Steer towards)
G Guide. Intn'l - I require a pilot.	Q Boat recall - own boats or those addressed. Intn'l - Vessel is healthy. Request free pratique.	Z - Intn'l - I require a tug.	9 Torpedo Action Table.
H Helicopter operations. Intn'l - I have a pilot onboard.	R Going alongside for Replenishing/ Transfer/Fuelling at sea. Ready Duty ship. MCM Operations. Intn'l -	1st Absentee Indicator - Flag Officer or Sq. Commander.	0 Guard Mail. Military Guard.
I Going alongside in Port/Anchor. Intn'l - Altering my course to Port.	S Flag Hoist Drill signal. Intn'l - My engines are going full speed astern.	2nd Absentee Indicator - Chief of Staff.	PENNANT ONE
J Semaphore message. Intn'l - I am on fire. Dangerous cargo, keep well clear.		3rd Absentee Indicator - Commanding Officer/X. O.	PENNANT TWO
		4th Absentee Indicator - Civil or Military Official.	PENNANT THREE
		1 ASW Action Table	PENNANT FOUR
		2 Surface Action Table	PENNANT FIVE
			PENNANT SIX
			PENNANT SEVEN
			PENNANT EIGHT
			PENNANT NINE
			PENNANT ZERO
			CODE or ANSWER Acknowledgement. Fractions. Use INTERCO.
			CORPEN Stop the turn.
			DESIG Plain Text. Proceeding to Station/Berth. Acknowledge DSL*.
			EMERGENCY Signal(s) flying are to be obeyed as soon as understood.
			FORM - ATION Refuse Boat is required.
			INT Signals not understood.
			NEGATIVE Negative. Flag Hoist Exempt Addressees.
			PREPARATIVE Replenishing (Receiving Vessel). Colours & Sunset. Preparative.
			PORT Indefinite turn to Port. Out of Routine.
			SCREEN -
			SPEED -
			STAR - BOARD Indefinite turn to Starboard SCOPA.
			STATION Take proper or assigned station.
			TURN Water barge is required.
			FLOTILLA SQUADRON DIVISION SUB DIVISION

Figura 1-7 Señales Navales OTAN [7]

1.4 Objetivo del TFG

En la actualidad la Escuela Naval Militar dota a los alumnos de nuevo ingreso con una baraja de cartas (Figura 1-8) con las distintas señales del Código Internacional de Señales, señales OTAN, así como otras banderas y gallardetes. Las cartas de la baraja se encuentran diseñadas de forma que, en el anverso se encuentra la señal representada gráficamente y en el reverso el significado de la misma. Esto permite que los alumnos primero memoricen las distintas señales viendo ambas caras de la carta y después comprueben sus conocimientos viendo solo la señal y relacionándola con su significado y viceversa.



Figura 1-8 Baraja de señales

La baraja está compuesta un total de 76 cartas constituidas por las siguientes señales:

- 26 letras correspondientes al abecedario inglés.
- 10 *pennant* o gallardetes numéricos del Código Internacional de Señales.
- 4 gallardetes repetidores.
- 1 gallardete característico “Code”.
- 10 números OTAN.
- 14 gallardetes especiales.
- 3 banderas especiales.
- 8 insignias de bando a bordo.

Asimismo, la Escuela Naval Militar publica en la Orden Diaria el significado de la bandera izada ese día para el adiestramiento de los alumnos en el palo de señales, tal y como se muestra en la Tabla 1-1.

BANDERA	CÓDIGO NATO – ATP-1 con buque de guerra	INTERCO con buques mercantes (gallardete CODE izado)
ZULU	En puerto: Mantengo guardia de comunicaciones	Necesito remolcador

Tabla 1-1 Bandera de Señales, Orden Diaria [8]

El uso de estos sistemas presenta ciertas deficiencias:

- En el caso de la baraja de señales, esta tiene muchas limitaciones, la más flagrante es que con el uso las cartas se deterioran y pueden perderse ciertas cartas dificultando por tanto el aprendizaje de las mismas. Además, se requiere que el alumno disponga físicamente de las cartas para estudiar, y no siempre las tendrá, cosa que con el móvil no suele pasar, de esta manera el alumno puede aprovechar los tiempos muertos para aprender.
- En el caso del método de la bandera izada diariamente en el palo de señales las limitaciones son obvias. En primer lugar, no despierta interés en el usuario puesto que es un método muy poco dinámico, sólo permite aprender una señal al día. En segundo lugar, es un método ineficiente, pues el alumno debe memorizar la información recogida en la Orden Diaria y al cabo de un tiempo asociar esa información con la señal izada, dado que generalmente no dispone de acceso a los dos recursos simultáneamente. Si bien es cierto que este método permite que el alumno adquiera un conocimiento más práctico, ya que este no asocia el significado a un dibujo sino a una bandera física, con la dificultad que eso conlleva pues las banderas flamean con el viento.

El objetivo del presente proyecto consiste en el diseño e implementación de un juego para dispositivos móviles Android que facilite el aprendizaje de las banderas de señales mediante el empleo de técnicas de gamificación y uso de las nuevas tecnologías en el aprendizaje. Solventando por tanto las limitaciones de disponibilidad, motivación y durabilidad presentes en los métodos actuales.

- Disponibilidad: pues es muy probable que el usuario siempre tenga acceso al móvil.
- Motivación: proporcionando actividades que supongan un reto para el usuario.
- Durabilidad: al contrario de las cartas de la baraja, las banderas de la aplicación ni se deterioran ni se pierden.

La futura implantación de este sistema como sustituto de la actual baraja de señales, no solo permitiría ahorrar el coste anual de dotar a los alumnos de nuevo ingreso con dicho material, sino que

también reduciría el consumo y desperdicio de material. Todo ello acorde a las exigencias impuestas respecto al comportamiento ambiental de las Bases del Ministerio de Defensa, instrucción 56/2011 [9].

1.5 Organización de la memoria

El presente trabajo se estructura en los siguientes capítulos:

- Capítulo 1: Introducción y objetivos. Se realiza una introducción para contextualizar el presente trabajo, así como exponer los objetivos del mismo.
- Capítulo 2: Estado del arte: Se describen otras aplicaciones del mismo ámbito que este proyecto, así como las herramientas y recursos empleados en su realización.
- Capítulo 3: Desarrollo del TFG. En primer lugar, se explica someramente las bases de la programación en Android. Posteriormente se pasa a detallar el desarrollo de la aplicación, dejando constancia de las decisiones tomadas para conseguir los objetivos propuestos.
- Capítulo 4: Resultados. Se detalla el proceso de distribución, el funcionamiento, así como las pruebas realizadas a la aplicación.
- Capítulo 5: Conclusiones y líneas futuras. Se expone una valoración global y se plantean las posibles líneas de actuación futuras con el fin de continuar con el desarrollo de la aplicación creada en este trabajo.
- Finalmente se anexan elementos que puedan ser de interés para el desarrollo del proyecto, como son los criterios empleados en la valoración de las aplicaciones o la relación de señales del Código Internacional de Señales empleada en el proyecto.

2 ESTADO DEL ARTE

En este capítulo se introduce brevemente el concepto de la gamificación, justificando su relevancia en este ámbito. Se analizan y comparan aplicaciones cuyo objetivo es el aprendizaje de las banderas de señales. Finalmente se describirán las herramientas empleadas en el proyecto.

2.1 Gamificación: el aprendizaje divertido

Marín y Hierro [10] explican la gamificación como “una técnica, un método y una estrategia al mismo tiempo. Parten del conocimiento de los elementos que hacen atractivos a los juegos e identifican, dentro de una actividad, tarea o mensaje determinado, en un entorno de NO-juego, aquellos aspectos susceptibles de ser convertidos en juego o dinámicas lúdicas”. Con esto se pretende conseguir un vínculo especial con los usuarios, incentivando un cambio en su comportamiento o transmitir un mensaje o contenido, creando una experiencia significativa y motivadora.

Por ello la realización de este proyecto se ha basado en la metodología didáctica de la gamificación, pues es la que más se adecua a los objetivos de nuestra aplicación. La gamificación surge de la utilización de tecnologías lúdicas en el ámbito educativo. A través de esta aplicación, los estudiantes se inician en el aprendizaje activo de las banderas y su significado. La motivación nace de la necesidad de resolver un problema, fomentándose una sana competitividad no solo con el resto sino también con uno mismo, se podría decir que ayuda a la superación. Asimismo, dota a los alumnos con retos progresivamente más difíciles que requieren de una mayor concentración y asimilación de los conceptos.

Esta metodología tiene en cuenta el aprendizaje significativo [10], el cual es necesario para conseguir que los conocimientos que adquieren los estudiantes sean duraderos. El aprendizaje significativo consiste en asociar información nueva con información ya conocida. Por ello en nuestra aplicación se parte de que el usuario conoce previamente las letras (A, B, C...), posteriormente asocia las palabras correspondientes a cada letra (“Alpha”, “Bravo”, “Charlie”), a continuación, asocia esa palabra con la representación visual de la bandera y finalmente con el significado. En la Figura 2-1 se puede ver un ejemplo del proceso de aprendizaje significativo donde el estudiante parte del conocimiento de la letra A y finaliza asociando este conocimiento al resto de conceptos relacionados.



Figura 2-1 Aprendizaje significativo

Otro punto a tratar es la socialización de las tecnologías. Pese a la dura crítica que se hace a estas formas de aprendizaje, caracterizándolas de aislantes e individualistas, la realidad es bien distinta. La

gamificación y, en concreto, esta aplicación permite que los alumnos comparen sus resultados, analicen sus fallos y aprendan de sus compañeros.

2.2 Aplicaciones existentes

Debido al amplio público con la necesidad de aprender las banderas y señales del Código Internacional de Señales existen múltiples aplicaciones disponibles para dispositivos móviles, tanto iOS como Android, tanto gratuitas como de pago. En las opciones disponibles, cabe destacar las siguientes aplicaciones de la Google Play Store y de la App Store.

2.2.1 *Nautical Flags and Signals*

Esta aplicación dispone de cuatro menús:

- **Banderas:** en este apartado la aplicación muestra de forma ordenada las banderas del Código Internacional de Señales, señales compuestas por múltiples banderas, señales específicas de la OTAN, señales de regatas, banderas del *Yacht Club Officer* y banderas de aviso de tormenta.
- **Semáforo:** en esta pestaña se presenta la señalización por el método del semáforo.
- **Traductor:** este apartado permite al usuario escribir un texto, el cual es mostrado posteriormente en banderas o en semáforo.
- **Publicaciones:** en este último apartado la aplicación redirecciona al usuario a las publicaciones de los Guardacostas de los Estados Unidos de América del Código Internacional de Señales, normas de navegación y ayudas a la navegación.

Cabe destacar que es multiplataforma (disponible tanto en iOS como en Android), dispone de una interfaz sencilla, Figura 2-2, y es completamente gratuita. [11]

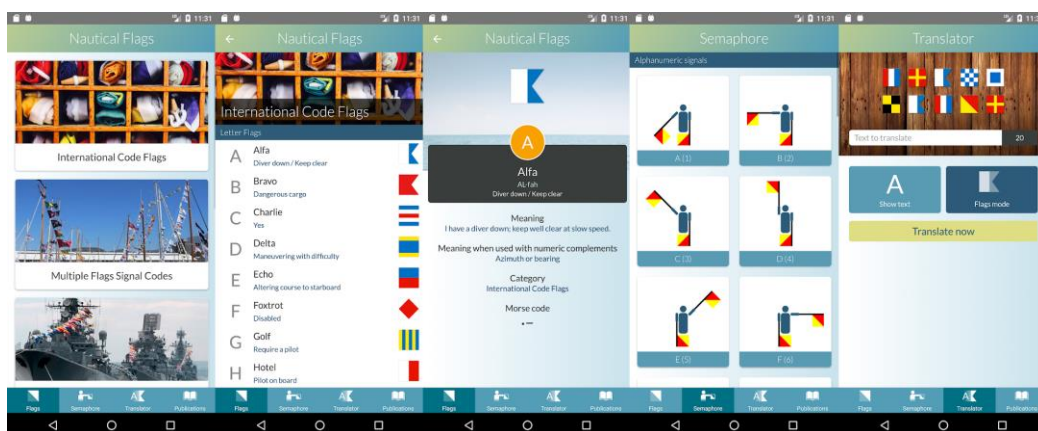


Figura 2-2 Pantallas de Nautical Flags and Signals [11]

2.2.2 *ICS Flag Trainer*

ICS Flag Trainer es una aplicación desarrollada por DGB Nautical [12], exclusivamente para la plataforma Android. Esta no es más que una de las múltiples aplicaciones que ha realizado esta empresa. Dispone de aplicaciones de aprendizaje que cubren los principales temas de la náutica: navegación, estabilidad, meteorología, maniobras, mareas, etc. En la Figura 2-3, se pueden apreciar las distintas pantallas de ICS Flag Trainer.

Esta consta de dos bloques principales, uno centrado en el aprendizaje de las banderas del Código Internacional de Señales y otro bloque en el que por medio de un test evalúa al usuario el conocimiento de las señales.

En el modo de aprendizaje la aplicación nos muestra una matriz con todas las señales y al seleccionar una señal se accede a la definición de la señal. En esta pantalla se muestra la señal, el

significado de la misma en el Código Internacional de Señales, el símbolo en Scott y la señalización por el método del semáforo. Cabe destacar que la aplicación muestra el mensaje en diversos idiomas: inglés, francés y alemán, pero no en castellano.

En el modo evaluación existen tres submodos de operación. En uno pide al usuario que relacione el Código Scott con la letra correspondiente, en otro que relacione la señal mostrada con la letra correspondiente y un tercer submodo en el que muestra un mensaje y le pide al usuario que lo relacione con la señal correspondiente. En cualquiera de los submodos la aplicación permite acceder a un menú de ayuda en caso de que el usuario falle, en el cual básicamente le redirige a la señal correcta en el modo aprendizaje para así asimilar su error.

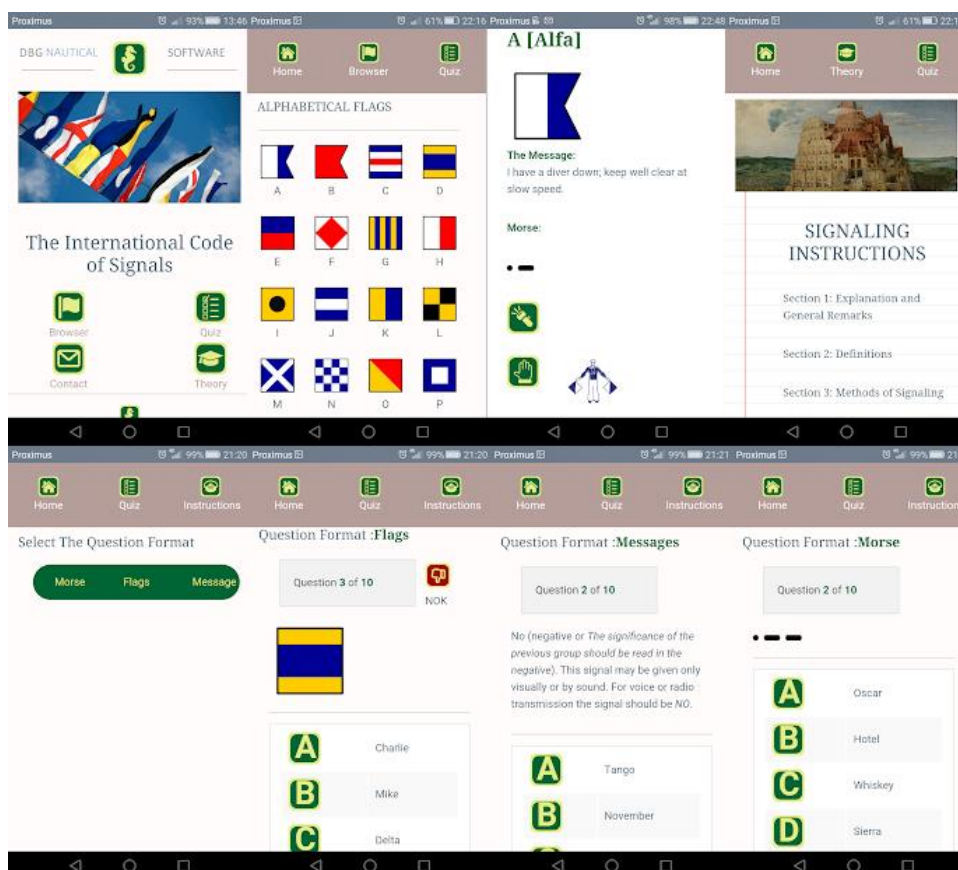


Figura 2-3 Pantallas de ICS Flag Trainer [12]

2.2.3 Nautical Flags Helper

Como puede verse en la Figura 2-4, es una aplicación muy sencilla y solamente dispone de dos modos, aprendizaje y prueba.

En el modo de aprendizaje le muestra al usuario las banderas del Código Internacional de Señales, permitiéndole elegir que las señales sean mostradas en orden alfabético o de forma aleatoria. Al seleccionar una bandera muestra al usuario el significado, la definición, la representación según el código de semáforo y Scott correspondiente. También permite mostrar los caracteres alfanuméricos y que al seleccionarlos se muestre la señal correspondiente.

El modo de prueba es la mejor parte de esta aplicación. Permite al usuario autoevaluarse de las banderas alfabéticas o numéricas, tanto por separado como en conjunto. Asimismo, permite personalizar la evaluación permitiendo al usuario elegir si quiere relacionar el nombre de la bandera con la imagen, el significado de la bandera con el nombre o el nombre de la bandera con el significado. [13]



Figura 2-4 Pantallas de Nautical Flags Helper [13]

2.2.4 Marinus RIPA

Esta aplicación es la más completa de la Google Play Store. Como puede apreciarse en la Figura 2-5, cuenta con un menú en el que se puede seleccionar entre las distintas materias de estudio y con un apartado de test en el que el usuario puede autoevaluarse de los conocimientos adquiridos en las distintas materias.

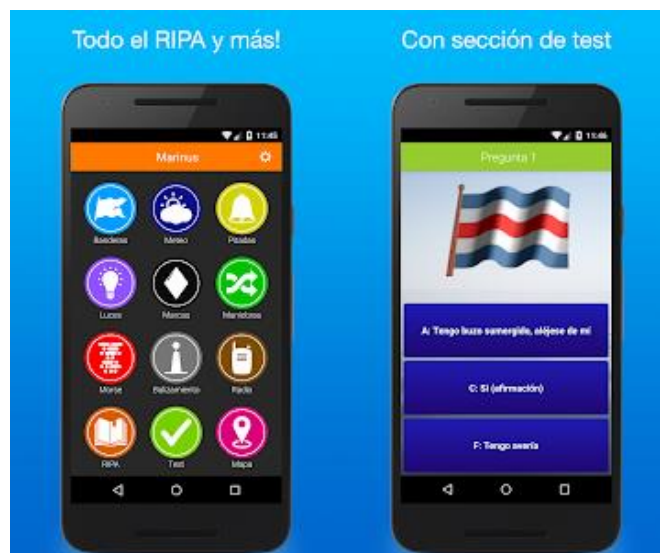


Figura 2-5 Marinus RIPA [14]

La aplicación está centrada en la obtención de las distintas titulaciones náuticas. Por ello se organiza a través de las distintas materias requeridas para una navegación segura:

- Banderas
- Meteorología
- Pitadas
- Luces
- Marcas de día
- Maniobras
- Scott
- Balizamiento
- Radio
- RIPA (Reglamento Internacional para la Prevención de Abordajes)

Además, dentro del apartado de test permite evaluar al usuario de las materias mencionadas anteriormente, ya sea por separado o en conjunto.

El principal inconveniente de esta aplicación es el precio ya que la versión completa cuesta 9,99€ [14], si bien es cierto que se puede descargar la aplicación gratuita Marinus Lite que sigue conteniendo el mismo menú con las distintas materias de estudio pero carece del apartado de test y muestra al usuario anuncios continuamente.

2.2.5 Comparativa

Tras comparar las aplicaciones por medio de los criterios descritos en el Anexo I: Criterios de comparación de las aplicaciones, se obtienen los resultados mostrados en la Tabla 2-1.

	Nautical Flags and Signals	ICS Flag Trainer	Nautical Flags Helper	Marinus RIPA
Contenido	3	2	3	4
Idioma	1	2	1	4
Interacción	3	3	2	4
Evaluación	1	2	2	2
Plataforma	4	1	1	4
Precio	4	4	4	1
Valoración Final	16/24	14/24	13/24	19/24

Tabla 2-1 Comparativa de aplicaciones existentes

Como se puede ver, el mayor inconveniente generalizado que tienen estas aplicaciones es su escasa capacidad a la hora de evaluar el conocimiento del usuario, pues en caso de disponer de capacidad de evaluación sólo cuentan con un tipo test. Con el fin de tener un hueco en el mercado de usuarios de estas aplicaciones se desea realizar una aplicación centrada en suplir esas carencias de evaluación, parte imprescindible del aprendizaje, así como emplear técnicas propias de la gamificación y el aprendizaje significativo.

2.3 Herramientas utilizadas

2.3.1 Formato SVG

SVG [15] (*Scalable Vector Graphics*, en español, Gráficos Vectoriales Redimensionables) es un formato de gráficos vectoriales bidimensionales en XML (*Extensible Markup Language*, en español, Lenguaje de Marcado Extensible), empleados para almacenar datos estructurados. SVG soporta tres tipos de objetos gráficos: formas vectoriales, imágenes y texto. Las siglas de SVG resumen sus características básicas:

- *Scalable*, implica que la imagen puede ser aumentada y reducida uniformemente, Permitiendo mostrar una misma imagen en pantallas de distinta resolución con distintos tamaños.
- *Vector*, contiene objetos geométricos tales como líneas, rectángulos, círculos, elipses, etc. Aportando una gran flexibilidad frente a las imágenes rasterizadas, en las cuales se

almacena la información para cada pixel de la imagen. Además, este formato permite combinar imágenes ráster con información vectorial.

- *Graphics*, la mayoría de los lenguajes XML representan información de texto o datos. El formato SVG suple esta carencia, permitiendo la incorporación de imágenes y gráficos vectoriales y/o rasterizados al meta-lenguaje XML.

La diferencia entre imágenes rasterizadas y vectorizadas se puede apreciar a simple vista, tal y como se muestra en la Figura 2-6.

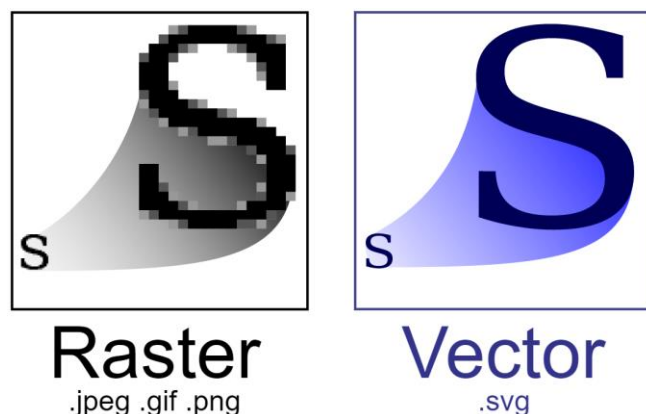


Figura 2-6 Diferencia entre imágenes rasterizadas y vectorizadas [16]

Las ventajas de las imágenes vectoriales sobre las rasterizadas son evidentes. El tamaño del archivo suele ser menor que si la imagen fuese rasterizada y permite ampliar la imagen sin deteriorar la calidad. Sin embargo, tiene un gran inconveniente, ya que no se puede representar cualquier contenido por medio de vectores. Estos gráficos se encuentran limitados a objetos formados por elementos geométricos, por lo que, en el caso de que un objeto sea muy complejo, el tamaño del archivo vectorizado aumenta considerablemente. Otro inconveniente es la imposibilidad de cargar solo una parte del objeto gráfico, hay que cargar el objeto entero.

Teniendo en cuenta que las banderas de señales están compuestas por elementos geométricos simples, se opta por usar este formato para el almacenamiento de las distintas imágenes. Esto permite emplear imágenes de buena resolución sin que eso suponga un aumento excesivo de la memoria requerida.

2.3.2 Inkscape

Inkscape [17] es un editor de gráficos vectoriales de código abierto, empleado en durante el desarrollo del proyecto para la creación de los ficheros SVG correspondientes a las banderas de señales.

Emplea SVG como formato nativo, y proporciona a sus usuarios una herramienta de elaboración de gráficos vectoriales escalables. Originalmente fue desarrollado para GNU/Linux, pero en la actualidad es una herramienta multiplataforma capaz de operar tanto en Windows como Mac OS X u otros derivados de Unix. Inkscape surge en 2003 basándose en Sodipodi, software libre también destinado a la edición de gráficos vectoriales.

Una de las mayores preocupaciones de los creadores de Inkscape es dotar al usuario de una herramienta de dibujo potente. Pese a la cantidad de menús, botones y parámetros, Inkscape es una herramienta fácil de utilizar, especialmente en la creación y modificación de imágenes formadas por patrones sencillos como son las banderas de señales. En la Figura 2-7, se puede apreciar la interfaz de Inkscape durante el diseño de una de las señales.

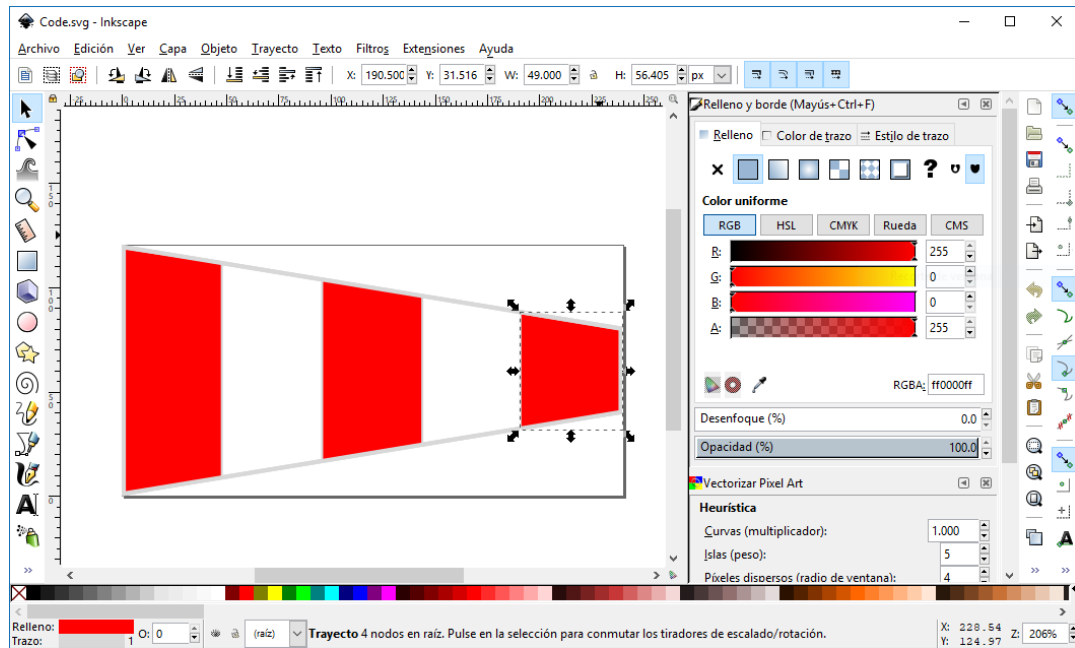


Figura 2-7 Diseño de “Code” con Inkscape

2.3.3 Java

Java [18] es un lenguaje de programación orientado a objetos el cual permite que las aplicaciones una vez compiladas puedan ser ejecutadas en una máquina virtual Java sin estar afectadas por el tipo de arquitectura de la computadora.

El origen de Java se remonta a los años 90, cuando Patrick Naughton, ingeniero de Sun Microsystems, junto con James Gosling y Mike Sheridan comenzaron a trabajar en el conocido como *Green Project*. Su objetivo era desarrollar una tecnología que permitiese programar la siguiente generación de dispositivos inteligentes. Para ello, buscaban un lenguaje de programación que fuese sencillo de aprender y emplear.

Al principio se consideró C++ como el lenguaje a emplear. Este no resultó ser completamente adecuado dando paso a que fuese modificado. El resultado fue C++--, un lenguaje basado en C++, pero con algunas características añadidas y otras eliminadas. Finalmente se abandonó este lenguaje y se optó por crear un nuevo lenguaje desde cero, Oak. Eventualmente, por cuestiones de propiedad intelectual, se cambia el nombre de Oak por Java. El resultado de este proyecto fue un lenguaje similar a C o C++ pero que no estaba ligado a un tipo de CPU en concreto.

En 1991 Java ejecutaba sus primeros programas, pero no fue hasta 1992 cuando Sun comenzó a comercializar productos con esta tecnología, aunque de manera poco fructífera. En ese mismo año termina la etapa de desarrollo y por ende el *Green Project*. Afortunadamente el lanzamiento de Java coincidió con el origen de la Web, herramienta gestada en el CERN (*Conseil Européen pour la Recherche Nucléaire*, en español, Consejo Europeo para la Investigación Nuclear). La Web marcó un antes y un después en el uso de Internet, revolucionando totalmente la concepción del mismo. La naturaleza del lenguaje de Java lo convertía en un perfecto candidato para emplearse en Internet, caracterizado en el momento por su baja velocidad y por la heterogeneidad de los dispositivos conectados a la misma. Tras dos años se consigue adaptar el lenguaje a su nuevo fin. En enero de 1995 Sun formó la empresa Java Soft, con el fin de dedicarse al desarrollo de productos basados en Java. En la SunWorld Exposition de mayo de 1995, se mostraron los avances tecnológicos del lenguaje Java, siendo un rotundo éxito. Netscape, el navegador más utilizado en Internet del momento, mostró un gran interés por Java, favoreciendo enormemente el desarrollo e implementación de dicho lenguaje. [18]

Las principales características del lenguaje de programación Java son las siguientes:

- **Sintaxis:** similar a C y C++, pero con una gestión de la memoria dinámica automática.
- **Interpretado:** un programa escrito en Java se compila a código *bytecode* y es este código el que es interpretado por la *Java Virtual Machine* (Máquina Virtual Java en español) o JVM.
- **Multiplataforma:** una vez compilado el programa Java, el fichero *bytecode* obtenido se ejecuta sobre la máquina virtual siendo independiente de la plataforma.
- **Seguro:** la JVM es la encargada de controlar que dicho programa no ejecute operaciones no autorizadas sobre los recursos del sistema.

La plataforma Java [19] se compone de las siguientes capas fundamentales, mostradas en la Figura 2-8. Empezando desde la base consta de:

- *Device Hardware:* consiste en el soporte físico sobre el cual se ejecuta Java.
- *OS, Drivers:* el Sistema Operativo u OS por sus siglas en inglés (*Operating System*), es el software principal de un sistema informático, el cual gestiona los recursos de hardware y provee servicios a las aplicaciones. Asimismo, los *drivers* o controladores de dispositivo, son los programas informáticos que permiten que el sistema operativo interactúe con los distintos periféricos.
- *Porting Layer for Platform Integrator:* esta capa se encarga de compatibilizar la instalación de Java en las distintas combinaciones posibles de hardware, OS y *drivers*.
- *Java VM:* en esta capa se encuentra la JVM, componente principal de Java. Esta se comunica con el hardware a través del OS y los *drivers*, abstrayendo al usuario de los mismos. Esto lo hace por medio de la interpretación del *bytecode*.
- *Java Profile:* es una herramienta integrada en el paquete de productos de Oracle Java la cual permite monitorizar la ejecución del código *bytecode* en el nivel de la Java VM.
- *Java APIs:* está compuesta por las distintas librerías nativas empleadas normalmente en las aplicaciones. Una API [18] (*Application Programming Interface*, en español, Interfaz de Programación de Aplicaciones) es una colección de componentes que dota a los programadores de medios para desarrollar aplicaciones Java. Las distintas APIs permiten realizar todo tipo de operaciones como puede ser procesar un XML por medio de la API JAXP (*Java API for XML Processing*). [20]
- *Partner Apps/APIs:* son las librerías y aplicaciones desarrolladas por terceros que se ejecutan en Java.

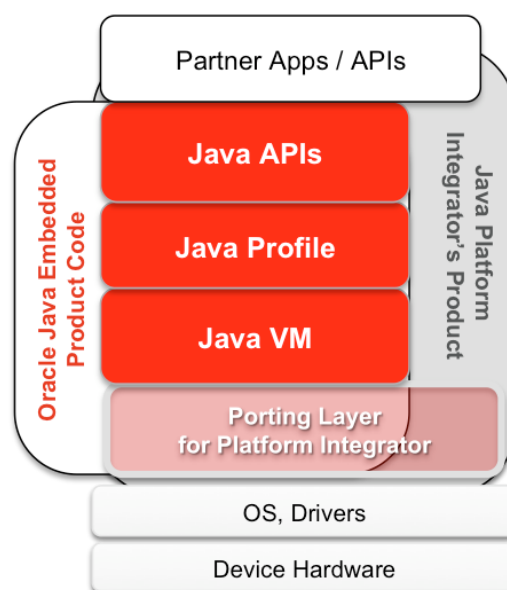


Figura 2-8 La plataforma Java [19]

2.3.4 Android

Android es un sistema operativo de código abierto basado en Linux, cuya programación se realiza mayormente en Java. Su cometido principal es ser empleado en dispositivos móviles con pantallas táctiles como son los teléfonos inteligentes, tabletas, relojes inteligentes, televisores, etc.

Antes de la llegada de Android y sistemas operativos similares, los primeros dispositivos solamente podían ser programados a bajo nivel, exigiendo que los programadores comprendiesen la totalidad del hardware que manejaban. Actualmente, los sistemas operativos permiten abstraer al programador del funcionamiento físico del dispositivo, si bien es cierto que esta abstracción limita al programador la libertad de acceder al hardware del dispositivo. Esta necesidad de abstracción y simplificación de la programación de los dispositivos fue el detonante de la aparición de Android.

Android surge de Android Inc., empresa fundada en 2003 cuyo objetivo era desarrollar un sistema operativo para cámaras digitales. En 2004 Android Inc. decidió cambiar el enfoque y dirigirlo hacia los dispositivos móviles. Este nuevo enfoque fue un gran acierto, despertando el interés de uno de los grandes de Internet, Google, el cual compró la empresa en 2005. Pero no fue hasta finales de 2007 cuando la Open Handset Alliance, una alianza de 34 compañías, presenta Android, favoreciendo enormemente la posición de este sistema operativo en el mercado de la telefonía móvil. [21]

Desde entonces Android no ha parado de desarrollarse, pasando por múltiples versiones con las cuales se incorporaban multitud de utilidades y mejoras desde Android 1.0 (septiembre de 2008) hasta Android 9.0 *Pie* (agosto de 2018). Esta última versión incorpora inteligencia artificial en funciones como son la batería adaptativa o brillo adaptable; mejoras en la gestión de la privacidad; mejoras de seguridad; etc.

La arquitectura de la plataforma Android está constituida por capas, tal y como se muestra en la Figura 2-9, permitiendo que las aplicaciones sean independientes del hardware:

- *Kernel* de Linux: constituye la base de la plataforma. Se encarga de que tanto el software como el hardware funcionen correctamente e interactúen entre sí. Para ello administra la memoria RAM, el procesador, así como la gestión del acceso al resto de periféricos como puede ser el teclado, Bluetooth, cámara ...
- Capa de abstracción de hardware o HAL (*Hardware Abstraction Layer*): permite el acceso a la información del hardware del dispositivo desde capas más altas. La capa está compuesta por bibliotecas de los distintos componentes *hardware*.
- Entorno de ejecución de Android o ART (*Android RunTime*): es la máquina virtual sobre la que se ejecutan las aplicaciones de Java. Esta remplace a Dalvik (empleada desde Android 2.2 Froyo) en dispositivos a partir de Android 5.0 Lollipop. A diferencia de su predecesor, que iba compilando la aplicación conforme se navegaba por ella, este compila todos los elementos y los almacena en una caché desde el momento en que se instala aplicación. Este nuevo sistema, al evitar que una aplicación tenga que compilarse continuamente cada vez que es ejecutada reduce el uso del procesador y, por tanto, aumenta la duración de la batería.
- Bibliotecas nativas de C/C++ o NDK (*Native Development Kit*, en español, Sistema de Desarrollo Nativo): son las bibliotecas nativas generadas por los desarrolladores de Android, destinadas a los componentes o servicios centrales del sistema como son el ART y el HAL basándose en código nativo.
- *Java API Framework*: contiene las funciones del sistema operativo Android, las cuales están disponibles mediante APIs escritas en Java, permitiendo crear las aplicaciones. Los principales componentes son: *View System*, contiene los distintos elementos que conforman la interfaz de usuario; *Content Providers* o proveedores de contenido, empleados para administrar el acceso a los datos de las distintas aplicaciones; y otros administradores como

es el *Activity Manager* o administrador de actividad, el cual gestiona el ciclo de vida de las aplicaciones.

- Aplicaciones del sistema: son un conjunto de aplicaciones centrales como pueden ser mensajería, calendario, contactos, mapas, etc. Las aplicaciones funcionan como tal para los usuarios y a su vez permiten que un desarrollador pueda emplear sus capacidades a través de su propia aplicación. [22]

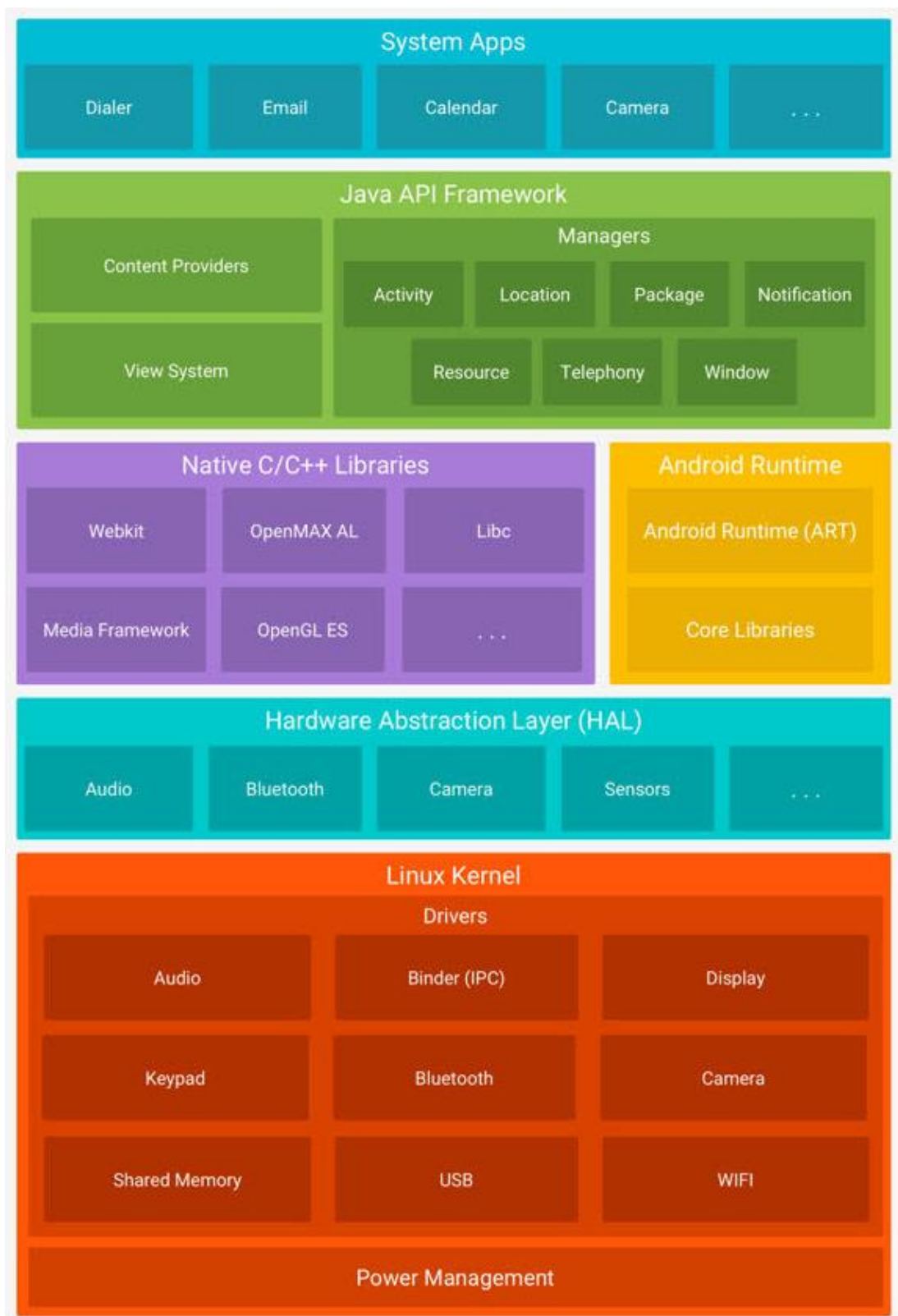


Figura 2-9 Pila de software de Android [22]

2.3.5 Entornos de desarrollo para Android

En este apartado se expondrán brevemente las características de los principales entornos integrados de desarrollo o IDE (*Integrated Development Enviroment*) disponibles en la actualidad para la implementación de aplicaciones Android, justificando la elección del entorno finalmente empleado en el desarrollo de nuestra aplicación.

2.3.5.1 Titanium Appcelerator SDK

Titanium Appcelerator SDK [23] es un entorno que ofrece a los desarrolladores la posibilidad de implementar aplicaciones móviles multiplataforma de forma nativa. A través de JavaScript permite abstraer la mayor parte de las APIs de las plataformas de Android e iOS.

Al ser una aplicación multiplataforma permite que el desarrollador no tenga que programar varias versiones por separado dependiendo del sistema operativo de destino. Aunque este concepto libera al desarrollador de mucha carga de trabajo también complica ligeramente el desarrollo de aplicaciones.

Titanium Appcelerator SDK está disponible en tres versiones, una gratuita y dos de pago. Las versiones de pago incluyen muchas más funcionalidades como son la depuración de código, sistema de pruebas, monitorización y analítica, etc.

2.3.5.2 Xamarin

Xamarin [24] es una herramienta de desarrollo multiplataforma de código abierto ofrecida por Microsoft como parte del paquete Visual Studio. Por medio de Xamarin es posible emplear el lenguaje C Sharp (C#), un lenguaje orientado a objetos desarrollado por Microsoft para el desarrollo de aplicaciones, Xamarin permite dirigirse a las plataformas iOS, Android y Windows Phone. Consigue rendimiento, funcionalidades e interfaces nativas pero empleado un código común para todos, Figura 2-10.

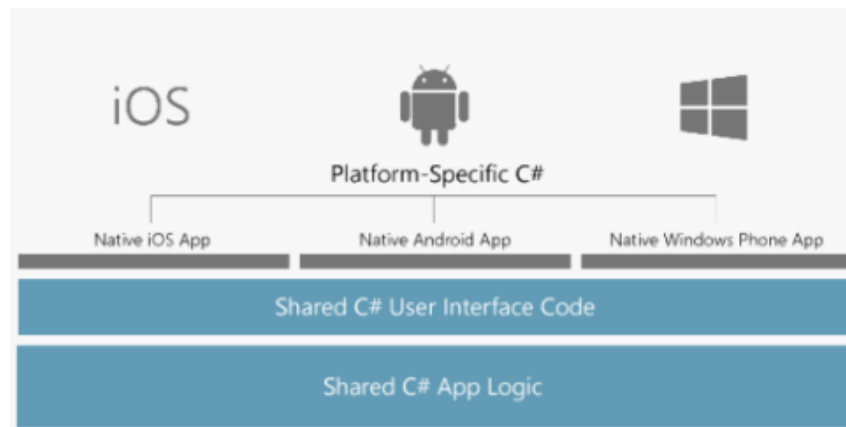
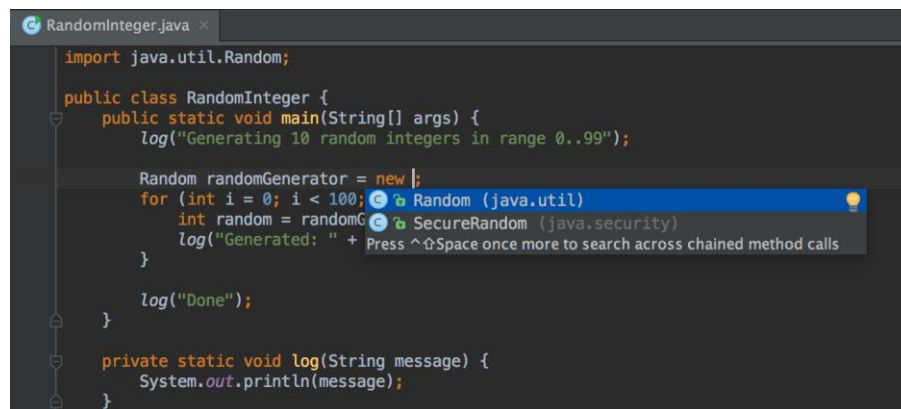


Figura 2-10 Xamarin, desarrollo multiplataforma [24]

2.3.5.3 IntelliJ IDEA

El entorno IntelliJ IDEA [25] está disponible en dos versiones: una de pago y una gratuita de código abierto.

Este entorno dispone entre otras cosas de: un editor de código inteligente, con función de autocompletado (Figura 2-11) e indicación de errores; un diseñador de interfaz, que permite arrastrar y soltar los elementos; herramientas para el tratado de bases de datos; y un control de versiones compatible con GitHub. La versión gratuita, a pesar de ser bastante completa tiene ciertas limitaciones, forzando que los usuarios opten finalmente por la versión de pago.



```

import java.util.Random;

public class RandomInteger {
    public static void main(String[] args) {
        log("Generating 10 random integers in range 0..99");

        Random randomGenerator = new Random (java.util)
        for (int i = 0; i < 100; i++) {
            int random = randomGenerator.nextInt(10);
            log("Generated: " + random);
        }

        log("Done");
    }

    private static void log(String message) {
        System.out.println(message);
    }
}

```

Figura 2-11 IntelliJ IDEA IDE [25]

2.3.5.4 Android Studio

Según Google, Android Studio [26] es el IDE oficial para la implementación de aplicaciones Android. Este entorno ha sido creado exclusivamente para el desarrollo de aplicaciones Android a partir de la versión de código abierto de IntelliJ IDEA, por lo que amplía las características de la versión gratuita de IntelliJ IDEA. Proporciona todas las herramientas necesarias para el desarrollo de aplicaciones en todo tipo de dispositivos Android, ya sean móviles, tabletas, radios de coche, relojes inteligentes, etc.

Algunas de sus principales ventajas son: la incorporación de un editor de código inteligente, que va sugiriendo al desarrollador a medida que este escribe el código; plantillas de código, con patrones de desarrollo estándar; control de versiones integrado; y un emulador, en el que se puede probar la aplicación sin necesidad de un dispositivo real, tal y como se muestra en la Figura 2-12.

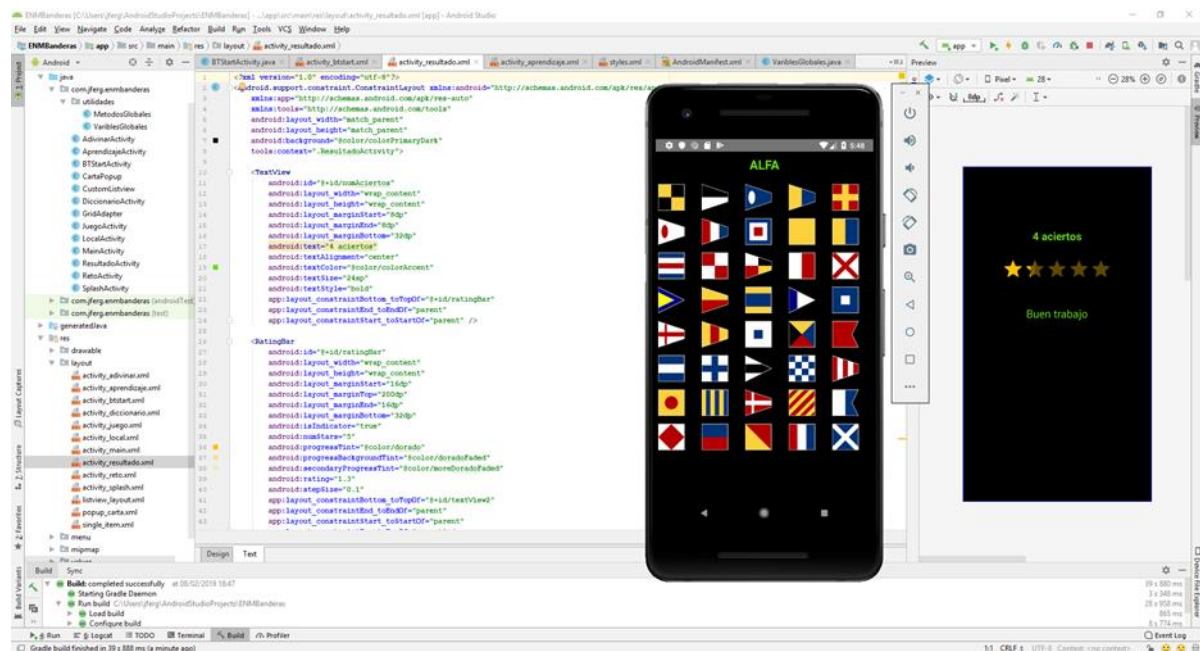


Figura 2-12 Interfaz de Android Studio

Además, debido a su gran aceptación en el ámbito de desarrolladores de aplicaciones Android Studio cuenta con innumerables guías y plantillas de códigos en Internet, lo que facilita enormemente el desarrollo de aplicaciones.

2.3.5.5 Elección

Una vez analizadas las distintas opciones disponibles para el desarrollo de aplicaciones Android, se pasa a seleccionar la opción más adecuada para el desarrollo de este trabajo.

Teniendo en cuenta las elevadas tasas exigidas por convertirse en desarrollador de Apple para poder publicar en su App Store, 99\$ anuales [27], frente a las de Google por publicar en Google Play, un único pago de 25\$ [28]. Por lo que económicamente Apple no es una opción. Además, Android permite instalar aplicaciones no indexadas en la Google Play Store, por lo que tan siquiera es necesario pagar una licencia de programador para distribuir una aplicación. Basta con habilitar los “orígenes desconocidos” e instalar una aplicación distribuida localmente. Del mismo modo, al optar por un desarrollo nativo sobre uno multiplataforma se evitan ciertos inconvenientes como puede ser la dificultad a la hora de añadir librerías de terceros en entornos multiplataforma.

Asimismo, Android es el sistema operativo con el cual se abarcaría a un mayor número de usuarios, pues actualmente es el claro dominador del mercado mundial, como se puede ver en la Figura 2-13. Además tal y como se muestra en la Figura 2-14, Android dispone actualmente de una cuota de mercado en nuestro país superior al 70%. [29]

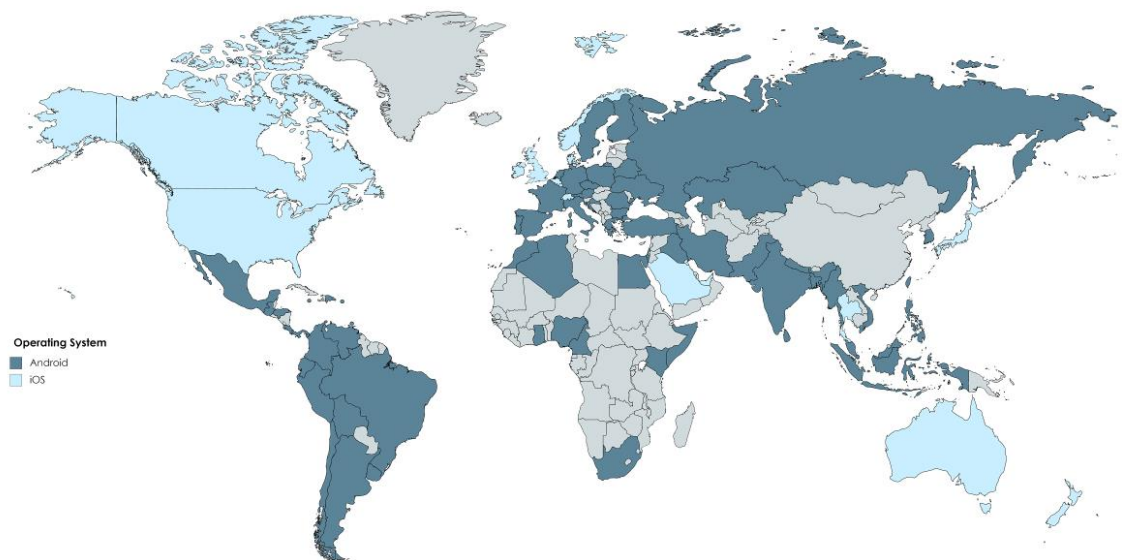


Figura 2-13 Sistema operativo dominante por países 2019 [29]

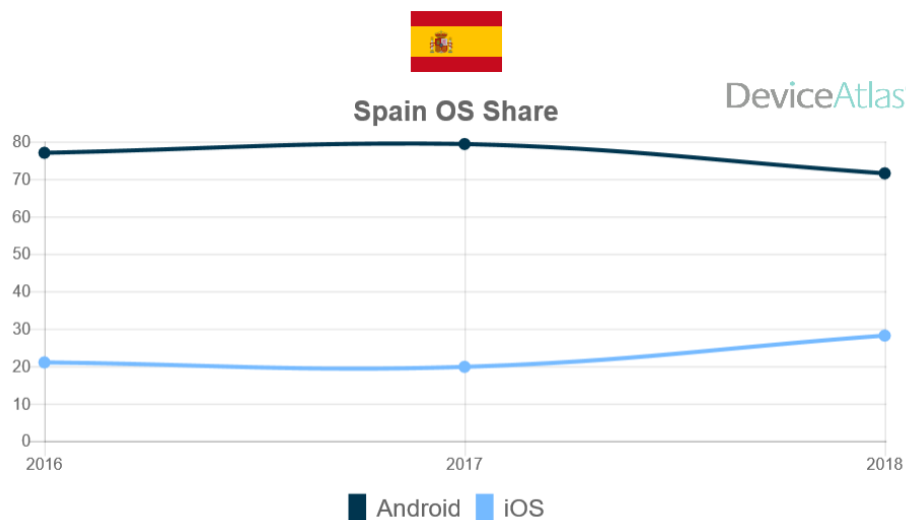


Figura 2-14 Distribución del mercado de Android e iOS en España 2019 [29]

Por las razones anteriormente descritas se descarta dar cobertura a los usuarios de iOS, centrándose el presente trabajo en el desarrollo exclusivo para dispositivos Android, por lo que, por simplicidad, se descarta la necesidad de un entorno multiplataforma.

Descartados los entornos multiplataforma Titanium Appcelerator SDK y Xamarin, queda decidirse entre IntelliJ IDEA y Android Studio.

A pesar de que la versión de pago de IntelliJ IDEA es el IDE más recomendado en múltiples foros [30] como la mejor alternativa, tiene como principal inconveniente su elevado precio (149€ la licencia individual durante el primer año [25]). El siguiente IDE mejor valorado es Android Studio, esta buena valoración viene dada en gran parte por el carácter oficial de este IDE y su gran similitud con la versión de código abierto de IntelliJ IDEA, en la cual se basa. Teniendo en cuenta lo expuesto anteriormente, a lo que hay que añadir la gran comunidad de desarrolladores con la que cuenta, se opta por usar Android Studio.

3 DESARROLLO DEL TFG

En este capítulo se detallarán los requerimientos a cumplir por el proyecto, se presentarán algunas nociones básicas de desarrollo de aplicaciones Android, finalmente, se justificarán las decisiones tomadas y se explicarán las partes más relevantes del código de la aplicación.

3.1 Requerimientos del Proyecto

Se desea realizar una aplicación que permita al usuario aprender y comprobar de manera autónoma las distintas banderas del Código Internacional de Señales. Para ello la aplicación debe contar con dos grandes bloques diferenciados claramente entre sí. Uno centrado en el aprendizaje de las señales, en el cual el usuario pueda acceder a una biblioteca donde se encuentren todas las señales con sus respectivos nombres y significados; y un segundo en el que, por medio de test y juegos, el usuario practique y compruebe lo aprendido anteriormente.

Para ello se requiere que la aplicación disponga de:

- Base de datos, en la cual se relacione la información correspondiente a las banderas, nombres y significados del Código Internacional de Señales.
- Interfaz de usuario intuitiva, que permita al usuario moverse por las diversas pantallas que conforman la aplicación de una forma fluida y sencilla.
- Mecanismos de generación de preguntas, los cuales permitan crear preguntas de forma aleatoria a partir de la información almacenada en la aplicación con el fin de evaluar al usuario.
- Conexión Bluetooth, permitiendo competir con otros usuarios.

3.2 Nociones básicas de Android

Antes de desarrollar una aplicación para este sistema operativo hay que tener en cuenta una serie de nociones básicas sobre el mismo. En este apartado se resumirán los componentes básicos que conforman una aplicación Android, la estructura de directorios que tiene un proyecto desarrollado en Android y una somera explicación sobre las distintas versiones de Android.

3.2.1 Componentes básicos

Los principales componentes disponibles en una aplicación Android son los siguientes:

- *Activity* (actividad): es cada una de las pantallas mostradas al usuario de la aplicación. Generalmente una aplicación está conformada por varias actividades, cuyo conjunto forman la interfaz que se muestra al usuario.

- *Intent* (intención): representa una intención de realizar una acción. Puede ser empleada para lanzar una actividad o para comunicarse e intercambiar información entre las diversas actividades o servicios.
- *Layout* (diseño): definen la estructura visual de la interfaz de usuario.
- *Fragment* (fragmento): es una porción de la interfaz de usuario que puede ser añadida y eliminada de la aplicación independientemente del resto de elementos de la actividad, así como reutilizarse en varias actividades. Esto permite diseñar diversas configuraciones de pantalla en función de su orientación y tamaño sin necesidad de duplicar el código. Para ello, basta con emplear los distintos fragmentos para las distintas configuraciones posibles.
- *Content Provider* (proveedor de contenidos): administra el acceso a un conjunto de datos estructurado, se emplea para compartir datos entre aplicaciones.
- *Service* (servicio): es un proceso ejecutado en un segundo plano, sin necesidad de interactuar con el usuario.

3.2.2 Estructura de un proyecto Android

La estructura de directorios de Android, permite tener los archivos correspondientes a cualquier proyecto siempre de una manera ordenada y tipificada. Esta organización es muy estricta y es tediosa para el usuario, por lo cual Android Studio permite al usuario visualizarla de diversas formas, entre las que cabe destacar la vista **Android** y **Project**.

En la vista **Android**, Android Studio muestra los archivos de un proyecto con la distribución mostrada en la Figura 3-1. Esta vista no representa la jerarquía real de los archivos almacenados en el disco, sino que organiza los archivos en módulos y tipos de archivos, facilitando la navegación. Dentro de cada módulo de aplicación los archivos se estructuran en los siguientes grupos:

- *manifests*: contiene el archivo *AndroidManifest*. Este archivo contiene la información esencial de la aplicación. Entre otras cosas recoge:
 - La versión de la aplicación.
 - Las actividades que van a emplearse.
 - Los permisos necesarios para acceder a otras aplicaciones, así como los permisos para que otras aplicaciones accedan a esta y los permisos para acceder a partes protegidas de una API.
- *java*: contiene los distintos archivos de código fuente escritos en Java.
- *res*: contiene los recursos de la aplicación, formados en su mayoría por ficheros XML. A su vez este está formado por varios directorios:
 - *drawable*: en este directorio se almacenan las imágenes ya sean JPG, PNG o SVG. Cabe reseñar que los nombres de estos archivos deben estar compuestos únicamente por números y las letras del abecedario inglés en minúsculas.
 - *layout*: en este directorio se almacenan los ficheros cuya función es describir el aspecto de la interfaz de usuario de las distintas actividades.
 - *menu*: en este directorio se encuentran los ficheros XML con los menús correspondientes a las distintas actividades.
 - *mipmap*: este directorio funciona de la misma manera que el directorio *drawable*, con la única salvedad de que en este directorio únicamente se emplazan las imágenes que no se usan dentro de la aplicación, como es el caso del icono de la aplicación.
 - *values*: en este directorio se almacenan diversos recursos XML que contienen valores simples como son: *strings* (cadenas de caracteres), *arrays* (vectores), enteros, estilos y colores. Al ser recursos XML podrían ser almacenados juntos en un único documento XML, sin embargo, para facilitar su uso se separan en distintos ficheros acorde con su contenido. [31]

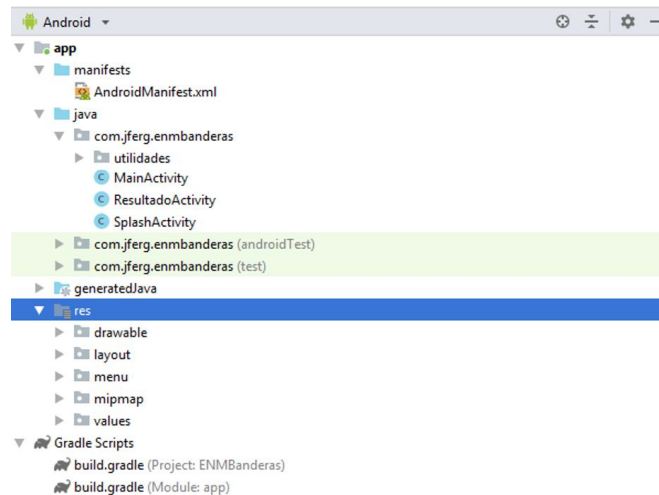


Figura 3-1 Vista *Android* de los archivos

En el caso de que se quiera ver la estructura real de los archivos del proyecto, basta con cambiar la vista a **Project** (Figura 3-2). Al seleccionar esta vista se muestran más archivos y directorios, entre los que cabe resaltar los siguientes:

- *build*: contiene archivos generados por la compilación.
 - *src*: contiene en su interior la carpeta *main* en la que se almacenan los archivos de código y recursos descritos en la vista **Android**:
 - *AndroidManifest*
 - *java*
 - *res*
- *build.gradle*: en este fichero se definen las principales configuraciones de compilación, las cuales se explicarán con mayor detalle más adelante.
 - *compileSdkVersion*: versión API empleada para compilar la aplicación.
 - *minSdkVersion*: nivel de API mínima necesaria para ejecutar la aplicación.
 - *targetSdkVersion*: nivel de API para el que ha sido probada la aplicación. [32]

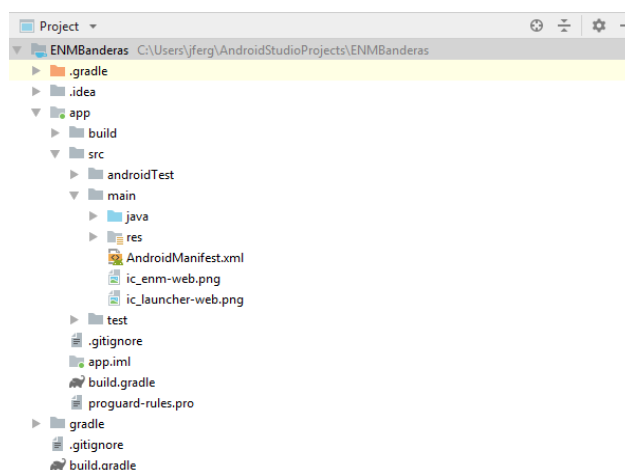


Figura 3-2 Vista *Project* de los archivos

3.2.3 Niveles de API

Antes de comenzar a programar es necesario elegir la versión del sistema sobre la que se quiere desarrollar la aplicación, aunque si fuese necesario esta elección puede ser modificada posteriormente a través del fichero *build.gradle*.

Cada versión incluye nuevas clases y métodos, al mismo tiempo que mantiene la mayoría de los métodos y clases anteriores. Por lo que una aplicación puede ser ejecutada por todos los dispositivos que dispongan de una versión igual o superior. Por ello es necesario conocer los requerimientos de nuestro proyecto y escoger un nivel de API que nos permita realizarlo abarcando el mayor número posible de dispositivos.

Las distintas actualizaciones de Android se describen alternativamente de tres formas: versión, nombre comercial y nivel de API, tal y como se muestra en la Tabla 3-1.

Versión	Nombre Comercial	Nivel de API
Android 1.0	<i>Apple Pie</i>	1
Android 1.1	<i>Banana Bread</i>	2
Android 1.5	Cupcake	3
Android 1.6	Donut	4
Android 2.0–2.1	Eclair	5–7
Android 2.2–2.2.3	Froyo	8
Android 2.3–2.3.7	Gingerbread	9–10
Android 3.0–3.2.6	Honeycomb	11–13
Android 4.0–4.0.5	Ice Cream Sandwich	14–15
Android 4.1–4.3.1	Jelly Bean	16–18
Android 4.4–4.4.4	KitKat	19–20
Android 5.0–5.1.1	Lollipop	21–22
Android 6.0–6.0.1	Marshmallow	23
Android 7.0 - 7.1.2	Nougat	24–25
Android 8.0 - 8.1	Oreo	26–27
Android 9.0	Pie	28

Tabla 3-1 Versiones Android

Las API van desde el nivel 1, correspondiente a Android 1.0 de septiembre de 2008 hasta el nivel 28, correspondiente a Android 9.0 Pie de agosto de 2018. Cabe destacar que no todas las API recibieron un nombre comercial como es el caso de Android 7 Nougat, cuyas versiones 7.0 y 7.1 se corresponden con las APIs 24 y 25. Así como que las versiones de Android 1.0 y Android 1.1 recibieron el nombre comercial Apple Pie y Banana Bread a posteriori.

Android permite que una aplicación sea compatible con una o más versiones de la plataforma Android. Esto lo realiza a través del valor del nivel de API, comparando el nivel de API de la aplicación con el del sistema Android en el que se desea instalar. Los atributos que determinan la compatibilidad entre distintas versiones se declaran en el archivo *build.gradle*.

3.3 Ficheros que componen una aplicación

En este apartado se detallarán los ficheros más importantes empleados en el desarrollo del proyecto entre los que se incluyen:

- *AndroidManifest*
- *build.gradle*
- Ficheros de recursos
- Actividades

3.3.1 *AndroidManifest*

Como ya se explicó anteriormente, este archivo (Figura 3-3) proporciona al sistema Android la información esencial requerida para el funcionamiento de la aplicación.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3   xmlns:tools="http://schemas.android.com/tools"
4   package="com.jferg.enmbanderas">
5
6   <uses-permission android:name="android.permission.VIBRATE" />
7   <uses-permission android:name="android.permission.BLUETOOTH" />
8
9   <application
10     android:allowBackup="true"
11     android:icon="@mipmap/ic_launcher"
12     android:label="ENM Banderas"
13     android:roundIcon="@mipmap/ic_launcher_round"
14     android:supportsRtl="true"
15     android:theme="@style/Theme.AppCompat.Light.NoActionBar">
16     <activity
17       android:name=".MainActivity"
18       android:screenOrientation="portrait" />
19
20     <activity...>
21
22     <activity...>
23
24     <activity...>
25
26     <activity...>
27
28     <activity...>
29
30     <activity...>
31
32     <activity...>
33
34     <activity...>
35
36     <activity...>
37
38     <activity...>
39
40     <activity...>
41
42     <activity...>
43
44     <activity...>
45
46     <activity...>
47
48     <activity...>
49
50     <activity...>
51
52     <activity android:name=".SplashActivity">
53       <intent-filter>
54         <action android:name="android.intent.action.MAIN" />
55
56         <category android:name="android.intent.category.LAUNCHER" />
57       </intent-filter>
58     </activity>
59   </application>
60 </manifest>
```

Figura 3-3 Extracto del *AndroidManifest.xml*

Entre los distintos parámetros recogidos en el fichero, cabe destacar los siguientes:

- *package*: es el nombre del paquete Java de nuestra aplicación, empleado para referenciar la aplicación en la Google Play Store (línea 4). Cabe reseñar que la aplicación no es indexable pues no se dispone de una licencia de desarrollador.
- *uses-permission*: por defecto una aplicación Android no tiene permisos asociados, es decir que no puede hacer nada que afecte de manera negativa a la experiencia del usuario o los datos del dispositivo. Por ello para habilitar las funciones protegidas se debe solicitar al usuario los permisos necesarios, como puede ser el empleo del Bluetooth. Estos permisos se declaran en el archivo *AndroidManifest.xml* por medio de etiquetas *<uses-permission>* como se puede observar en las líneas 6 y 7.
- *application*: permite configurar los parámetros básicos de la aplicación como es el icono (mostrado en la Figura 4-4), el nombre, el tema empleado, etc. Todo ello dentro de la etiqueta *<application></application>*. Asimismo, dentro de esta etiqueta han de añadirse todas las actividades de nuestra aplicación, a las cuales también se les puede asignar distintos atributos. Por ejemplo, en las líneas 16-18 se declara la *MainActivity*, actividad correspondiente al menú principal. Asimismo, para simplificar el desarrollo se fuerza que nuestro dispositivo siempre tenga una disposición en modo retrato por medio del atributo *android:screenOrientation="portrait"*. Finalmente, es importante declarar en este

apartado cual es la actividad que ha de ser lanzada en el momento de abrir la aplicación, por medio de la categoría *LAUNCHER*, en este caso es *SplashActivity*, líneas 53 a 59. [33]

3.3.2 *build.gradle*

En este fichero, mostrado en la Figura 3-4, se definen los parámetros necesarios para la compilación del código.

- *compileSdkVersion*: indica la versión del SDK empleada para compilar el proyecto, generalmente esta debe de ser la misma que la *targetSdkVersion*. Este parámetro determina las características que pueden ser empleadas a la hora de compilar el código. En el caso de nuestro proyecto el valor seleccionado es 28, correspondiente al SDK más moderno, puesto que este parámetro sólo afecta a la hora de compilar y no en el paquete APK (*Android Application Package*, en español, Paquete de Aplicación Android). Esto permite beneficiarse de las características de compilación del SDK más moderno, como puede ser la detección de APIs obsoletas.
- *minSdkVersion*: determina el nivel de API mínimo necesario para ejecutar la aplicación. En caso de que el dispositivo sea de un nivel inferior, Android impide incluso que el usuario instale la aplicación.
- *maxSdkVersion*: es un atributo que normalmente no se utiliza, pues limita la aplicación a una determinada API, por lo que impide el funcionamiento de la aplicación en dispositivos cuya versión sea superior a la indicada.
- *targetSdkVersion*: este parámetro no tiene que ver con la forma de compilar la aplicación ni con las APIs empleadas. La utilidad de este parámetro es indicar que la aplicación ha sido probada en una determinada API, por lo cual en el caso de que la API de la plataforma fuese superior a la versión declarada en este atributo el sistema pasaría a habilitar comportamientos de compatibilidad que garanticen el correcto funcionamiento de la aplicación. En caso de no configurarse, su valor es el mismo que el asignado a *minSdkVersion*. A efectos prácticos se configura este parámetro acorde a la última versión de la API disponible, asegurando así que la aplicación funcione de la mejor manera posible en los dispositivos Android más modernos.
- *versionCode*: este valor hace referencia al número de versión de la aplicación, que ha de ser distinto cada vez que esta se publique.
- *versionName*: cadena de caracteres cuyo único propósito es mostrar un valor de versión a los usuarios. [34]

```

1  apply plugin: 'com.android.application'
2
3  android {
4      compileSdkVersion 28
5      defaultConfig {
6          applicationId "com.jferg.enmbanderas"
7          minSdkVersion 21
8          targetSdkVersion 28
9          versionCode 1
10         versionName "1.0"
11         testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
12     }

```

Figura 3-4 Extracto del *build.gradle*

Teniendo en cuenta las estadísticas realizadas por Google, basadas en el número de descargas de aplicaciones de la Google Play Store realizadas en un periodo de 7 días a finales del mes de octubre de 2018, se puede inferir la cantidad relativa de dispositivos que emplean diferentes versiones, tal y como se muestra en la Figura 3-5. Si bien no es un dato exacto pues Google no tiene en cuenta para esta

estadística al mercado chino (no dispone de acceso a la Google Play Store), es un parámetro bastante acertado a la hora de determinar la cantidad de usuarios reales que podrán utilizar una aplicación.

Debido a las ventajas que ofrece Android 5.0 Lollipop frente a versiones anteriores, principalmente el empleo exclusivo del tiempo de ejecución ART, mejorando sustancialmente el rendimiento y la capacidad de respuesta de las aplicaciones, proporcionando al uso un comportamiento más fluido. Incluye también APIs para gestionar el uso de Bluetooth de bajo consumo. Asimismo, incluye APIs enfocadas a la gestión de múltiples redes.

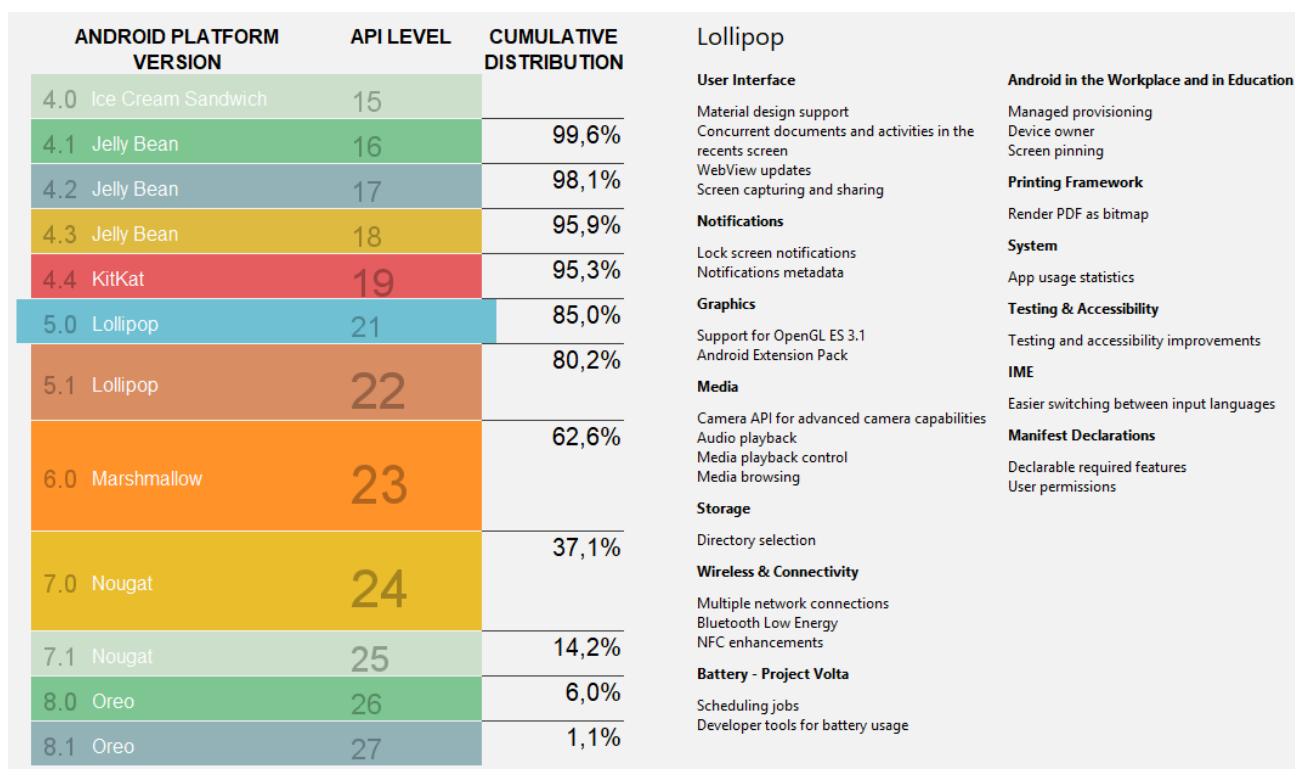


Figura 3-5 Cantidad relativa de versiones de plataforma Android [26]

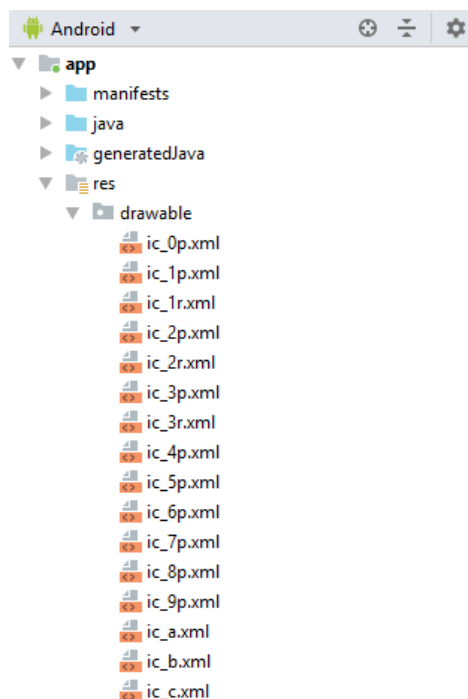
Como puede ser inferido de lo anteriormente expuesto y del código de la Figura 3-4, se decide que el nivel de API mínimo requerido para la ejecución de esta aplicación es el 21, correspondiente a Android 5.0. Lo que permite que la aplicación sea compatible con la mayoría de dispositivos Android, concretamente un 85%. [35]

3.3.3 Ficheros de recursos

En Android existen una gran variedad de archivos de recursos, de entre los que cabe destacar los archivos: *drawable*, *values* y *layout*. Estos se detallan a continuación.

3.3.3.1 Recursos *drawable*

Los recursos *drawable* son las imágenes ya sean JPG, PNG o SVG, empleadas en la aplicación. En el caso de esta aplicación se emplea mayoritariamente el formato SVG debido a las ventajas respecto a espacio de almacenamiento y escalabilidad que se obtienen con este formato frente a los otros mencionados anteriormente. Las imágenes correspondientes a las banderas del Código Internacional de Señales en formato SVG se almacenan en el directorio *drawable* en formato XML, tal y como se muestra en la Figura 3-6, permitiendo que estas puedan ser posteriormente llamadas por las diversas actividades.

Figura 3-6 Recursos *drawable*

Cada imagen correspondiente a las banderas del Código Internacional de Señales se describe en un fichero como el que corresponde a la Figura 3-7, el cual representa la bandera “Alpha”.

Figura 3-7 Extracto de *ic_a.xml*

3.3.3.2 Recursos *values*

Dentro del directorio *values* se encuentran los ficheros que contienen los valores empleados a lo largo del código, ya sean colores, estilos o cadenas de caracteres. Al encontrarse todos los valores en un mismo directorio el desarrollador puede modificar estos valores más cómodamente. Asimismo, este concepto puede ser empleado para desarrollar una aplicación en múltiples idiomas, pues basta con generar distintos archivos *string* donde se almacenen las cadenas de caracteres que conforman las palabras y frases correspondientes a los distintos idiomas en distintos directorios en función del idioma, tal y como se muestra en la Figura 3-8. Si bien es cierto que esta funcionalidad no ha sido empleada en el proyecto, pues sólo se emplean etiquetas en castellano. [36]

```
MyProject/
  res/
    values/
      strings.xml
    values-es/
      strings.xml
    values-fr/
      strings.xml
```

Figura 3-8 Ejemplo de subdirectorios *string* para múltiples idiomas [36]

En la Figura 3-9, se pueden ver algunos ejemplos de cadenas de caracteres o *string-arrays* empleados en la aplicación desarrollada. Entre ellos el nombre de la aplicación (línea 2), los nombres de las banderas (línea 4), los caracteres empleados para representar los símbolos Scott (línea 47), los significados del Código Internacional de Señales (línea 90), etc. Los nombres, significados del Código Internacional de señales y símbolos Scott introducidos en el fichero *strings* se relacionan en el Anexo II: Relación de señales correspondientes al Código Internacional de Señales.

```
1  <resources>
2      <string name="app_name">ENM Banderas</string>
3
4      <string-array name="bandera"...>
46
47      <string-array name="scott"...>
89
90      <string-array name="significado">
91          <item>Buzo en el agua.</item>
92          <item>Cargando o descargando mercancías peligrosas.</item>
93          <item>Afirmación SI.</item>
94          <item>Estoy maniobrando con dificultad.</item>
95          <item>Caigo a estribor.</item>
96          <item>Tengo avería, comuniqué conmigo.</item>
97          <item>Necesito práctico.</item>
98          <item>Tengo práctico a bordo.</item>
99          <item>Caigo a babor.</item>
100         <item>Tengo incendio y llevo a bordo mercancías peligrosas.</item>
```

Figura 3-9 Extracto de *strings.xml*

3.3.3.3 Recursos *layout*

Los ficheros *layout* contienen la información con respecto al diseño visual de la aplicación en formato XML. Definen la estructura visual de la interfaz de usuario. Estos diseños son empleados a través de las actividades como se explicará más adelante. [37]

Existe una gran variedad de elementos empleados en el diseño del aspecto de las interfaces de usuario. A continuación, se relacionan los distintos elementos empleados en el proyecto actual relativos al diseño, mostrados en la Figura 3-10:

- *TextView*: permite mostrar cadenas de textos modificables a través del código.
- *Button*: consiste en un botón, empleado para detectar pulsaciones por parte del usuario y actuar en consecuencia.
- *ImageView*: permite mostrar una imagen, la cual se puede determinar desde el código.
- *Toast*: muestra al usuario un mensaje de notificación en la pantalla, el cual desaparece al cabo de unos pocos segundos sin que el usuario tenga que hacer nada.
- *ListView*: permite generar un grupo de vistas en forma de listado de elementos, los cuales pueden ser seleccionables.
- *GridView*: funciona de igual manera que el anterior, pero con la salvedad de que los elementos se muestran en forma de tabla.

Los elementos *ListView* y *GridView* requieren de un adaptador, el cual recupera los datos de un conjunto y genera los objetos que posteriormente serán listados o tabulados. Volviendo al ejemplo de la Figura 3-10, cada objeto mostrado dentro del *ListView* y *GridView* consta de dos valores, *Item X* y *Sub Item X*.

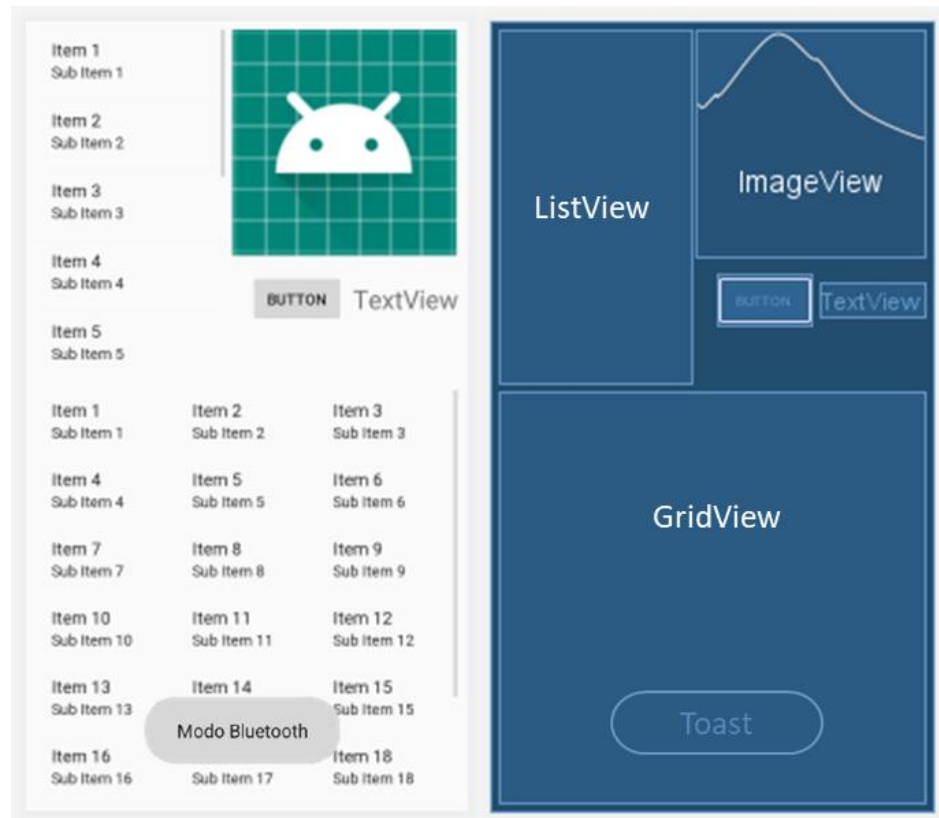


Figura 3-10 Resumen de elementos de diseño empleados

Los distintos elementos y controles que componen la interfaz gráfica se pueden disponer y organizar en la misma por medio de un *layout*, dependiendo de la distribución de elementos requerida en nuestra aplicación. Entre los *layout* disponibles cabe destacar los siguientes:

- *Linear*: apila todos los elementos hijos (elementos contenidos en el interior de otro, al que se denomina padre) en sentido vertical u horizontal, acorde a la orientación que se determine.
- *Relative*: permite especificar la posición de los elementos hijos en relación al padre u otros objetos hijo, indicando las distancias entre los distintos elementos.
- *Constraint*: al igual que el anterior permite posicionar los elementos en función de los demás, pero a diferencia del anterior este permite hacerlo de forma porcentual, siendo de especial utilidad para el uso en pantallas de distinto tamaño, pues emplea referencias relativas. [37]

Los distintos elementos empleados se declaran y describen en formato XML, tal y como se muestra en la Figura 3-11, donde se expone un ejemplo de distintos controles y vistas (*Button*, *ImageView*) organizados a través de un *ConstraintLayout* (línea 2). A continuación, se especifican los parámetros relativos al aspecto general de la pantalla como son el ancho y el alto (líneas 6 y 7), forzando que la actividad ocupe toda la pantalla. Finalmente, se definen los distintos elementos que componen el *layout* como son los botones (líneas 11-26) y las imágenes (líneas 41-53) que componen la interfaz.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <android.support.constraint.ConstraintLayout
3     xmlns:android="http://schemas.android.com/apk/res/android"
4     xmlns:app="http://schemas.android.com/apk/res-auto"
5     xmlns:tools="http://schemas.android.com/tools"
6     android:layout_width="match_parent"
7     android:layout_height="match_parent"
8     tools:layout_editor_absoluteY="25dp"
9     android:background="@color/colorPrimaryDark">
10
11     <Button
12         android:id="@+id/baprendizaje"
13         android:layout_width="0dp"
14         android:layout_height="wrap_content"
15         android:layout_marginStart="32dp"
16         android:layout_marginTop="32dp"
17         android:layout_marginEnd="32dp"
18         android:layout_marginBottom="24dp"
19         android:text="Aprende"
20         android:textSize="30sp"
21         app:layout_constraintBottom_toTopOf="@+id/bjuego"
22         app:layout_constraintEnd_toEndOf="parent"
23         app:layout_constraintStart_toStartOf="parent"
24         app:layout_constraintTop_toBottomOf="@+id/imageView2" />
25
26     <Button...>
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41     <ImageView...>
42
43
44
45
46
47
48
49
50
51
52
53     <ImageView...>
```

Figura 3-11 Extracto de *activity_main.xml*

Asimismo, Android Studio incorpora un panel de diseño, mostrado en la Figura 3-12, el cual permite emplazar los distintos elementos de una forma más intuitiva para el usuario. Permite que el usuario confeccione el *layout* sin necesidad de escribir en XML los parámetros de los distintos elementos, sino que permite seleccionar y arrastrar dichos elementos, generándose automáticamente el fichero XML a partir de las acciones del desarrollador.

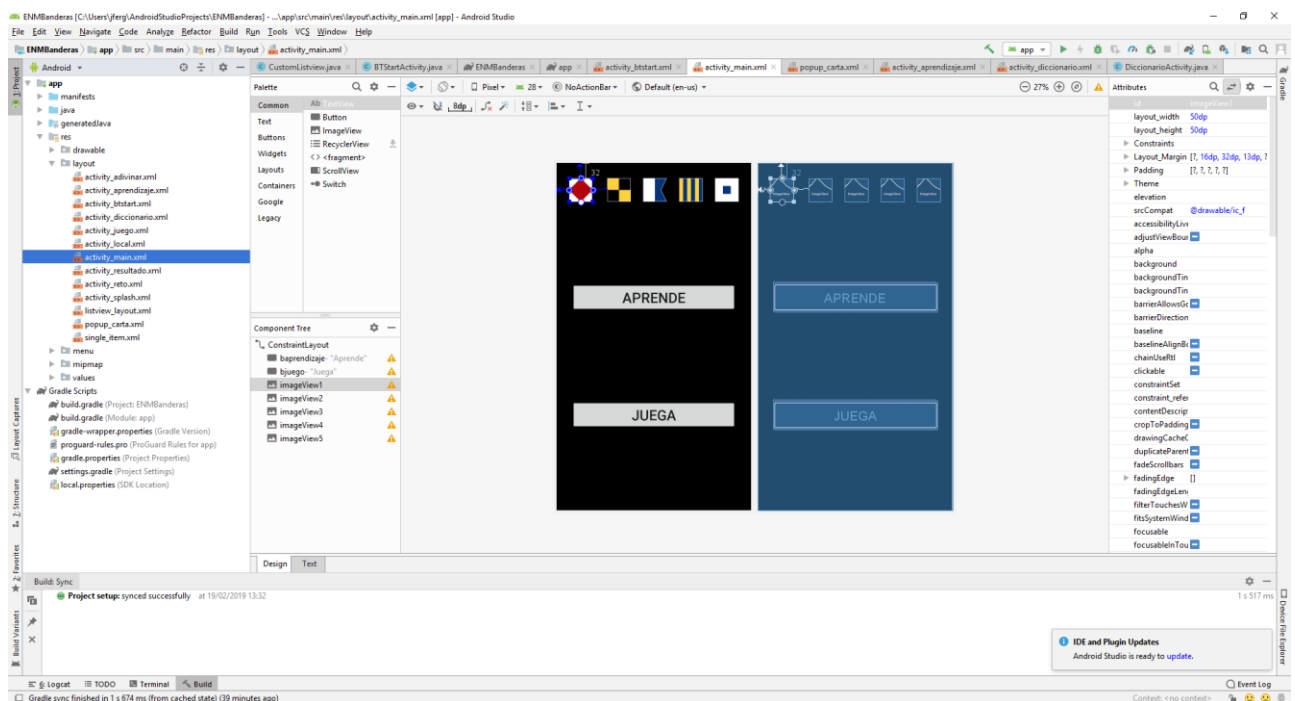


Figura 3-12 Panel de diseño correspondiente a *activity_main.xml*

3.3.4 Actividades

Las actividades son componentes de la aplicación que muestran una pantalla al usuario, con la cual se puede interactuar.

Como se mostró anteriormente, las actividades han de ser declaradas en el *AndroidManifest*, así como declarar la actividad principal que se lanza al inicio de la aplicación. Una vez dentro de una actividad, esta puede iniciar otra actividad llamando al método *startActivity*, junto con un *intent* que describa la actividad a iniciar. En la Figura 3-13 se muestra gráficamente este proceso de salto entre actividades, en el cual cuando el usuario pulsa el botón correspondiente de la actividad *AprendizajeActivity* inicia la actividad *DiccionarioActivity*. Este mecanismo permite vincular las distintas actividades que conforman la aplicación. [38]

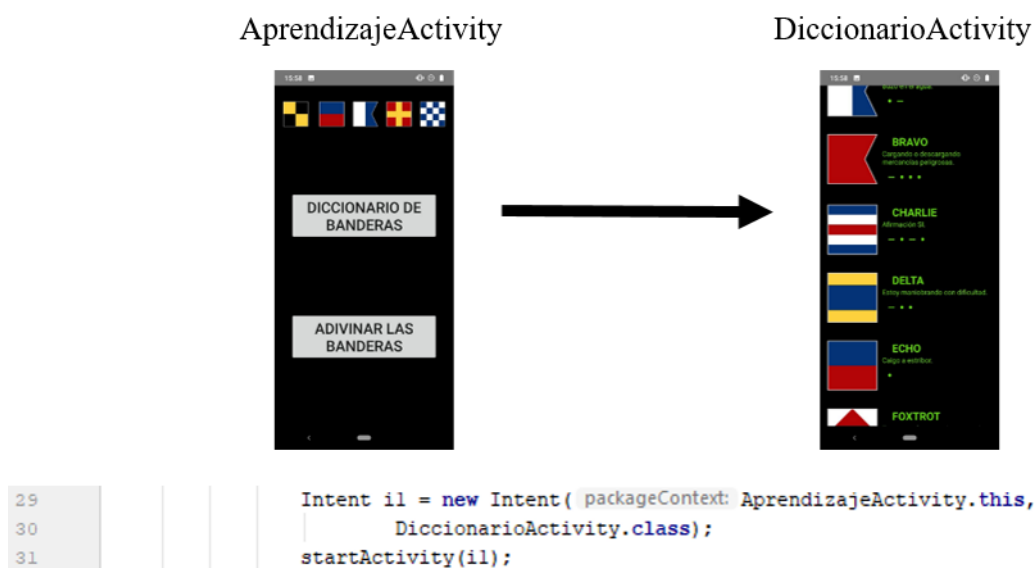


Figura 3-13 Ejemplo de un *intent*

Cada vez que se inicia una actividad se detiene la anterior y es preservada por el sistema en una pila de actividades, por lo que cuando el usuario finaliza de utilizar una actividad y pulsa el botón de retroceso la actividad sale de la pila, se destruye y se reanuda la actividad anterior. Volviendo al ejemplo de la Figura 3-13, en el caso de encontrarse en la *DiccionarioActivity*, al pulsar el retroceso, el sistema destruiría la actividad *DiccionarioActivity* reanudando la anterior, *AprendizajeActivity*.

Las distintas actividades están compuestas por diversos atributos, los cuales aportan las funcionalidades requeridas en la aplicación. A continuación, se pasa a detallar los distintos atributos que confeccionan una actividad estándar, para ello se empleará el código de la Figura 3-14 correspondiente al archivo *DiccionarioActivity.java*.

En primer lugar, se determina el paquete (línea 1), permitiendo identificar la ruta correspondiente al archivo dentro de nuestro proyecto.

Posteriormente, se gestionan las clases importadas que son necesarias para el funcionamiento de la actividad en cuestión, estas pueden ser añadidas manualmente, aunque generalmente el editor las añade automáticamente conforme se emplean métodos de las distintas clases. Por ejemplo, en la línea 4, por medio de *import android.widget.ListView* se importa la clase que permite mostrar elementos listados.

En la línea 8 se declara el nombre de la clase que conforma actividad, el cual ha de coincidir con el nombre del fichero Java. Dentro de esta clase se definen los métodos y variables o atributos. Dentro del método *onCreate* se detalla el *layout* asociado a la actividad, a través del *SetContentView* (en este caso *activity_diccionario*, línea 19), así como el funcionamiento de la actividad. Para ello es necesario

obtener diversas variables (líneas 21-25) previamente declaradas (líneas 10-14); identificar los distintos objetos correspondientes al *layout* y asignarles los recursos correspondientes (línea 27), a través de *findViewById*; y finalmente realizar las operaciones requeridas con los recursos asignados, en este caso listar los contenidos de los vectores iniciales a través de un adaptador (líneas 28-30).

```
1 package com.jferg.enmbanderas;
2
3 import android.support.v7.app.AppCompatActivity;
4 import android.os.Bundle;
5 import android.widget.ListView;
6 import com.jferg.enmbanderas.utilidades.VariblesGlobales;
7
8 public class DiccionarioActivity extends AppCompatActivity {
9
10     ListView lst;
11     String[] bandera;
12     String[] significado;
13     String[] scott;
14     Integer[] imgid;
15
16     @Override
17     protected void onCreate(Bundle savedInstanceState) {
18         super.onCreate(savedInstanceState);
19         setContentView(R.layout.activity_diccionario);
20
21         imgid = VariblesGlobales.iimgid;
22
23         bandera = getResources().getStringArray(R.array.bandera);
24         significado = getResources().getStringArray(R.array.significado);
25         scott = getResources().getStringArray(R.array.scott);
26
27         lst = (ListView) findViewById(R.id.ListView);
28         CustomListview customListview = new CustomListview(context: this, bandera, significado,
29             scott, imgid);
30         lst.setAdapter(customListview);
31     }
32 }
```

Figura 3-14 Extracto de *DiccionarioActivity.java*

Aparte del método *onCreate*, descrito anteriormente, una actividad puede contener también otros métodos. Un método no es más que una porción de nuestro código que puede ser llamado y empleado desde otra parte del código, esto permite reutilizar y ordenar el código. A grandes rasgos existen dos tipos de métodos según el valor devuelto, *void* si no devuelven nada o aquellos que devuelven una variable de cualquier tipo. Un ejemplo de esto es el método mostrado en la Figura 3-15, el cual se emplea para identificar los distintos elementos que conforman el aspecto visual de la actividad *BTStartActivity*. Como se puede ver este no devuelve ningún valor al ser un *void*, simplemente ejecuta el código que contiene al ser llamado.

```
195 private void findViewByIdes() {
196     listDevice = (Button) findViewById(R.id.btListDevice);
197     status = (TextView) findViewById(R.id.tvstatus);
198     listView = (ListView) findViewById(R.id.lvlistview);
199     listViewPregunta = (ListView) findViewById(R.id.lvPregunta);
200     score = (TextView) findViewById(R.id.tvScore);
201     scoreEne = (TextView) findViewById(R.id.tvScoreEne);
202 }
```

Figura 3-15 Método *findViewByIdes* de la *BTStartActivity.java*

3.3.4.1 Ciclo de vida de una actividad

Un factor a tener en cuenta es que no todos los métodos pueden ser determinados por el usuario, existen unos métodos predefinidos encargados de la gestión del ciclo de vida de las actividades. El comportamiento de estos métodos y el ciclo de vida de una actividad se muestran en la Figura 3-16 y se detallan a continuación.

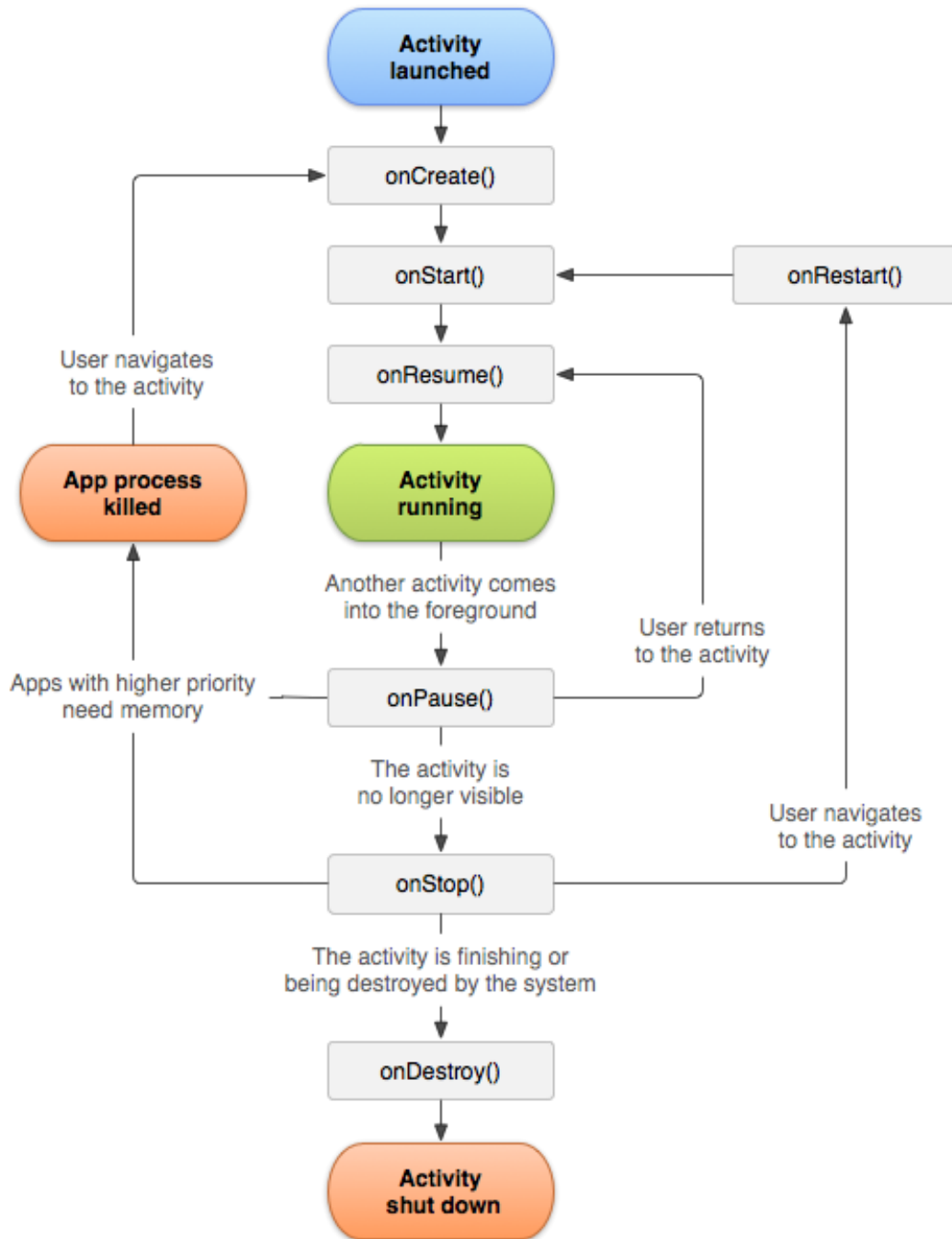


Figura 3-16 Ciclo de vida de una actividad [39]

En el momento que se llama a una actividad por medio de un *intent* se inicia el ciclo de vida y el sistema llama a los siguientes métodos por orden:

- *onCreate*: es el primer método en ser llamado, así como el lugar en el que se declara el *layout* asociado.

- *onStart*: se llama justo a continuación del anterior. En el caso de que la aplicación se encuentre en segundo plano, es el método que se ejecuta al volver a estar esta en primer plano.
- *onResume*: es el último método llamado antes de que la actividad acceda a la pantalla. Es el mejor método en el que colocar elementos gráficos que cambien, ya que este siempre se ejecuta justo antes de que la actividad vuelva a acceder a la pantalla.

Una vez que el usuario finaliza una actividad, bien sea porque pulsa el botón de retroceso o porque la actividad es llamada por otra opción, se ejecutan sucesivamente los siguientes elementos:

- *onPause*: es el primer método en ser llamado cuando hay otra actividad de fondo y la actividad actual se encuentra a punto de irse de la pantalla. Este método es el único capaz de detectar que la actividad o la aplicación se está cerrando, por lo tanto, es crucial para guardar información importante antes de que se cierre la actividad.
- *onStop*: este método es llamado una vez que la aplicación está fuera de la pantalla y por tanto el usuario ha dejado de interactuar con ella.
- *onDestroy*: es el último método que se llama antes de que el mecanismo de gestión automática de memoria de la máquina virtual (*garbage collector*, en español, recolector de basura) elimine la actividad por completo. En este método debe pararse todo proceso que este corriendo de fondo. [39]

3.3.4.2 Clases auxiliares

Con el fin de reutilizar el código empleado y mantener un orden en los archivos se crea un directorio dentro de la carpeta Java de la aplicación. En este se almacenan los ficheros Java que contienen las variables y métodos que se reutilizan a lo largo de diversas actividades y estructuras de código empleadas de manera recurrente a lo largo de las actividades de la aplicación.

3.3.4.2.1 Métodos Globales

En el archivo *MetodosGlobales.java*, se encuentran una serie de métodos que pueden ser llamados desde cualquier parte de la aplicación, de este modo basta con escribir una única vez el código del método que se quiere implementar y simplemente llamarlo cuando se requiera.

Un ejemplo de este tipo de métodos es el *void shuffleArray* (Figura 3-17), el cual ordena aleatoriamente un *array* de números enteros empleando el algoritmo de Fisher-Yates [40]. Este método es empleado tanto para la creación de preguntas como de respuestas, pues a diferencia de la elección puramente aleatoria de un vector, este método evita que se generen preguntas o respuestas duplicadas, que no se extraen valores aleatorios de un intervalo, sino que se desordena el vector del cual posteriormente se obtienen las preguntas y opciones.

```
1 package com.jferg.enmbanderas.utilidades;
2
3 import java.util.Random;
4
5 public class MetodosGlobales {
6     public void shuffleArray(int[] shuffleArray) {
7         //Empleando el algoritmo de Fisher-Yates
8         Random rnd = new Random();
9         for (int i = shuffleArray.length - 1; i >= 0; i--) {
10             int index = rnd.nextInt( bound: i + 1);
11             int temp1 = shuffleArray[index];
12             shuffleArray[index] = shuffleArray[i];
13             shuffleArray[i] = temp1;
14         }
15     }
```

Figura 3-17 Método *shuffleArray* de *MetodosGlobales.java*

Asimismo, todas las pantallas emplean el mismo método para generar las preguntas. El método *especificarPregunta* (Figura 3-18) tiene tres argumentos de entrada: un entero *resp*, cuyo valor representa la pregunta que quiero generar y por tanto la respuesta correcta; un vector de enteros *opciones*, este vector contiene todos los valores que enteros que representan a las banderas, de tal forma que este método altera las posiciones de sus miembros hasta que las primeras posiciones del vector contengan las opciones generadas para la pregunta; y un entero *numOpciones*, que determina el número de opciones que se desea generar.

En un primer lugar llama al método *shuffleArray* (línea 23), desordenando el vector de enteros *opciones*.

A continuación, determina en función de la pregunta que se desee generar cuales son los valores límites para generar respuestas, esto se emplea para evitar que genere preguntas con respuestas obvias (pues las banderas se encuentran almacenadas en las posiciones 0 a 25 y los gallardetes de la 26 a 39).

Posteriormente, por medio de un bucle *for* (líneas 33-44), comprueba que las primeras posiciones del vector *opciones* no coincidan con valores por fuera de los límites considerados como válidos (es decir pertenece al mismo tipo de bandera) o con el valor de la respuesta (*resp*), en caso de que coincidan se salta esa posición, comprobando la siguiente. Esto lo realiza hasta que se alcanza la posición del vector correspondiente al valor determinado al inicio como número de opciones requeridas *numOpciones*, momento en el que sale del bucle de generación de opciones por medio de un *break* (líneas 41-43).

El resultado del bucle *for* anterior es el vector *opciones*, cuyas posiciones iniciales contienen opciones válidas para la pregunta que se quiere generar, pero sin contener la respuesta, por ello al final del método se sustituye aleatoriamente uno de los valores iniciales del vector por el valor de la respuesta correcta (líneas 46-48).

```

19 public void especificarPregunta(int resp, int[] opciones, int numOpciones) {
20     int inic;
21     int fini;
22
23     shuffleArray(opciones);
24     //Discrimina entre banderas del abecedario y los gallardetes
25     if (resp > 25) {
26         inic = 26;
27         fini = 39;
28     } else {
29         inic = 0;
30         fini = 25;
31     }
32     //Genera opciones dentro del intervalo pero que no sean la respuesta
33     int posPreg = 0;
34     for (int i = 0; i < opciones.length; i++) {
35         if (opciones[i] >= inic && opciones[i] <= fini) {
36             if (opciones[i] != resp) {
37                 opciones[posPreg] = opciones[i];
38                 posPreg = posPreg + 1;
39             }
40         }
41         if (posPreg > (numOpciones-1)) {
42             break;
43         }
44     }
45     //Se incluye la respuesta en una de las opciones
46     Random rand = new Random();
47     int rdn = rand.nextInt(numOpciones); //
48     opciones[rdn] = resp;
49 }

```

Figura 3-18 Método *especificarPregunta* de *MetodosGlobales.java*

El último método empleado globalmente es *dobleClick*, mostrado en la Figura 3-19. Se trata de un método *boolean* (devuelve valores verdadero o falso). Este método es llamado siempre se realiza una pulsación y devuelve un valor que determina si la pulsación ha de considerarse válida o no, permitiendo actuar en consecuencia. La utilidad de este método es evitar posibles fallos por múltiples pulsaciones que generen más respuestas de las requeridas.

A grandes rasgos, este método comprueba dos posibles casos:

- En caso de que la diferencia de tiempo desde la última llamada sea superior al tiempo determinado, se puede considerar que es una pulsación válida y por tanto devuelve el valor lógico verdadero (línea 23).
- En caso de haber sido llamado en un tiempo inferior, se entiende ha habido múltiples pulsaciones sobre un mismo ítem, provocando las consiguientes llamadas al método y por tanto devuelve el valor lógico falso (línea 27).

En función del valor devuelto, la activad de origen puede determinar si aceptar o ignorar la pulsación, evitando cargar múltiples actividades. Se determinó que 500 milisegundos es un tiempo de espera aceptable entre pulsaciones válidas y que no compromete la experiencia del usuario.

```
51 private static long mHoraUltClick;
52 public boolean dobleClick(){
53     //Evitar que afecte si usuario hace doble click
54     if (SystemClock.elapsedRealtime() - mHoraUltClick < 500){
55         return true;
56     }
57     else {
58         mHoraUltClick = SystemClock.elapsedRealtime();
59         return false;
60     }
61 }
```

Figura 3-19 Método *dobleClick* de *MetodosGlobales.java*

3.3.4.2 Variables Globales

De la misma manera, en el archivo *VariablesGlobales.java* (Figura 3-20), se encuentran las variables empleadas en múltiples actividades. Un ejemplo de esto puede ser una variable empleada para determinar la actividad actual (línea 7), de manera que cuando una actividad puede ser llamada desde múltiples actividades sea capaz de diferenciar el origen y actuar en consecuencia, como es el caso de *ResultadoActivity*.

```
5 public class VariablesGlobales {
6     //Permite saber el origen de una pantalla compartida en la app
7     public static int pantallaActual=1;
8     //0 Resultado
9     //1 Bandera (Imagen->Bandera)
10    //2 Adivinar
11    //3 Imagen (Bandera->Imagen)
12    //4 Significado (Significado->Bandera)
13
14    //Permite almacenar las imagenes correspondientes a las banderas
15    public static final Integer[] iimgid = {R.drawable.ic_a, R.drawable.ic_b, R.drawable.ic_c,
16        R.drawable.ic_d, R.drawable.ic_e, R.drawable.ic_f, R.drawable.ic_g, R.drawable.ic_h,
17        R.drawable.ic_i, R.drawable.ic_j, R.drawable.ic_k, R.drawable.ic_l, R.drawable.ic_m,
18        R.drawable.ic_n, R.drawable.ic_o, R.drawable.ic_p, R.drawable.ic_q, R.drawable.ic_r,
19        R.drawable.ic_s, R.drawable.ic_t, R.drawable.ic_u, R.drawable.ic_v, R.drawable.ic_w,
20        R.drawable.ic_x, R.drawable.ic_y, R.drawable.ic_z, R.drawable.ic_0p, R.drawable.ic_1p,
21        R.drawable.ic_2p, R.drawable.ic_3p, R.drawable.ic_4p, R.drawable.ic_5p,
22        R.drawable.ic_6p, R.drawable.ic_7p, R.drawable.ic_8p, R.drawable.ic_9p,
23        R.drawable.ic_code, R.drawable.ic_1r, R.drawable.ic_2r, R.drawable.ic_3r};
```

Figura 3-20 Extracto de *VariablesGlobales.java*

Asimismo, se puede emplear para cargar desde los ficheros de recurso las constantes que van a ser empleadas a lo largo de la ejecución de la aplicación, como pueden ser las imágenes de las banderas. Para ello se declaran todas las banderas en un vector de *Integer*. Esto se utiliza para facilitar el acceso desde las distintas clases a los distintos recursos, como es el caso de *Integer[] iimgid* (líneas 15-23), en este se almacenan los atributos *Integer*, empleados a modo de referencia de las distintas imágenes de las banderas, permitiendo acceder a los recursos cómodamente por medio de una posición en un vector.

3.4 Bluetooth en Android

La aplicación generada en este proyecto requiere una conexión Bluetooth para el intercambio de datos con otros dispositivos con el fin de interactuar con otros usuarios. Debido a la complejidad a la hora de emplear estas funcionalidades se explica por separado en este apartado el proceso de conexión empleado en la aplicación.

Por medio de las APIs de Android Bluetooth se pueden realizar todas las tareas necesarias para:

- Activar y desactivar el Bluetooth del dispositivo.
- Consultar los dispositivos Bluetooth sincronizados por medio del adaptador local.
- Establecer los distintos canales de RFCOMM, (*Radio Frequency Communication*, en español, Comunicación por Radio Frecuencia).
- Activar el modo descubrimiento y conectarse a otros dispositivos.
- Administrar múltiples conexiones.
- Transferir datos entre los distintos dispositivos conectados.

Con el fin de simplificar la aplicación y evitar operaciones tediosas al usuario, solo se podrá conectar a dispositivos ya vinculados, es decir, la vinculación de nuevos dispositivos debe realizarse de forma nativa por medio de los ajustes del Bluetooth. Esto simplifica enormemente la programación pues solo es necesario listar los dispositivos vinculados por medio de una consulta al adaptador local, generar un método para que el dispositivo funcione como servidor, otro como cliente y un último método encargado de la transmisión y recepción de los mensajes entre los dispositivos. [41]

3.4.1 Permisos necesarios

Para emplear Bluetooth es necesario añadir al *AndroidManifest* los permisos necesarios para el acceso al hardware. Debido a las simplificaciones empleadas sólo es necesario declarar `<uses-permission android:name="android.permission.BLUETOOTH" />`, permitiendo solicitar y aceptar conexiones, así como transferir datos. [41]

3.4.2 Dispositivos vinculados

La forma de obtener los dispositivos Bluetooth ya conocidos y vinculados es muy sencilla y se muestra en la Figura 3-21. Se hace a través del adaptador Bluetooth *bluetoothAdapter* mediante la llamada al método *getBondedDevices*, línea 109. Una vez obtenidos los dispositivos conectados se puede consultar tanto el nombre de los dispositivos como su dirección MAC (*Medium Access Control*). A pesar de que la dirección MAC es lo único que Android necesita para establecer una conexión, su empleo no es cómodo para el usuario, por lo que lo mejor es mostrarle el nombre del dispositivo y emplear la dirección MAC internamente para establecer la conexión. La obtención tanto de la MAC como del nombre del dispositivo se realiza por medio de los métodos *getAdress* y *getName* respectivamente. [41]

```
109 Set<BluetoothDevice> pairedDevices = bluetoothAdapter.getBondedDevices();
110 String[] strings = new String[pairedDevices.size()];
111 btArray = new BluetoothDevice[pairedDevices.size()];
112 int index = 0;
113
114 if (pairedDevices.size() > 0) {
115     for (BluetoothDevice device : pairedDevices) {
116         btArray[index] = device;
117         strings[index] = device.getName();
118         index++;
119     }
120     ArrayAdapter<String> arrayAdapter = new ArrayAdapter<>(getApplicationContext(),
121         android.R.layout.simple_list_item_1, strings);
122     listView.setAdapter(arrayAdapter);
123 }
124 ServerClass serverClass = new ServerClass();
125 serverClass.start();
```

Figura 3-21 Obtención de los dispositivos vinculados

Una vez obtenidos los dispositivos vinculados, se muestra al usuario una lista con los nombres de dichos dispositivos, en la cual puede seleccionar de una forma cómoda con cual quiere conectarse.

3.4.3 Conexión de los dispositivos

El intercambio de información se realiza por medio del protocolo Bluetooth de Android, este está mediante una arquitectura cliente-servidor. La conexión entre dos dispositivos por medio de Bluetooth es gestionada a través de un *BluetoothSocket*, siendo este el punto de conexión a través del cual se puede intercambiar datos con otro dispositivo por medio de los flujos de entrada y salida *InputStream* y *OutputStream*.

Para establecer una conexión entre dos dispositivos es necesario que cada dispositivo tenga los siguientes tres componentes: servidor, cliente y conexión, siendo gestionados por hilos de ejecución (*threads*) independientes (Figura 3-22).

```
213 //Clases requeridas por el BluetoothSocket
214 private class ServerClass extends Thread {...}
257
258 private class ClientClass extends Thread {...}
292
293 private class SendReceive extends Thread {...}
```

Figura 3-22 Clases requeridas para establecer una conexión Bluetooth

A continuación, se pasa a detallar la secuencia lógica seguida para realizar una conexión:

- La clase servidor crea un *socket* para ponerse en escucha, en este caso se emplea el tipo de *socket* Bluetooth RFCOMM, también conocido como *Serial Port Profile* (SPP), puesto que es el *socket* más comúnmente empleado. Una vez abierto el *socket* se pone a la espera de recibir una solicitud de conexión.
- Mientras tanto, en el otro dispositivo se inicializa la clase cliente, en la que crea un nuevo dispositivo Bluetooth partiendo de su propia dirección MAC y del *socket* Bluetooth del otro dispositivo y se realiza una solicitud de conexión, quedando a la espera de respuesta.
- Al recibir y aceptar la conexión por parte del servidor, este notifica al cliente que su solicitud de conexión ha sido aceptada al mismo tiempo, abre la conexión en un *socket*.
- Al recibir el cliente la notificación de que su solicitud de conexión ha sido aceptada sale de su espera.

- A partir de ese momento el cliente y el servidor obtienen los flujos de entrada y salida de sus respectivos *sockets* y el hilo encargado de la conexión puede empezar a enviar y recibir flujos de *bytes* con el otro dispositivo. [41]

En la Figura 3-23 se resume el proceso necesario para establecer una conexión Bluetooth entre dos dispositivos.

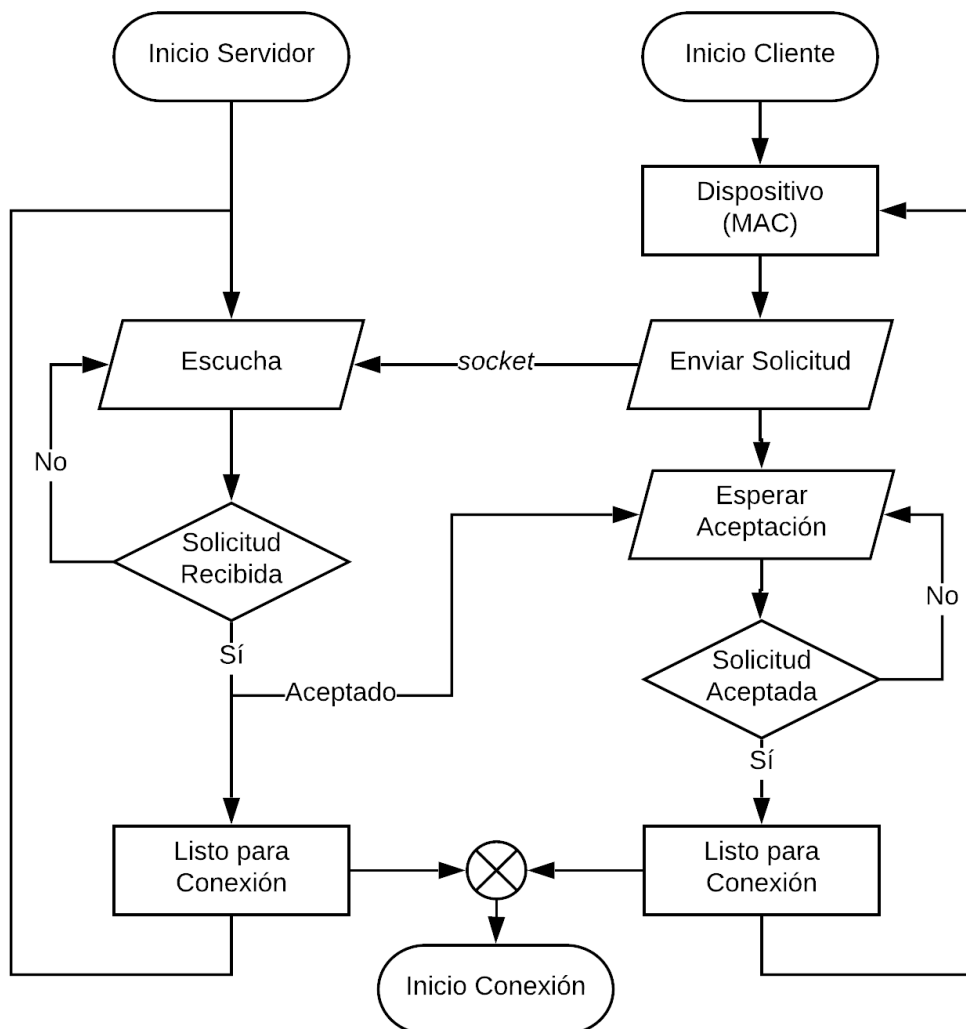


Figura 3-23 Diagrama de flujo del establecimiento de conexión Bluetooth

Con el objetivo de minimizar la necesidad de interacción por parte del usuario a la hora de iniciar la conexión Bluetooth ambos dispositivos inician la clase servidor, la clase cliente se inicia en el momento en el que un usuario decida comenzar la conexión con otro dispositivo. En caso de que ambos envíen la solicitud simultáneamente la aplicación rechaza la conexión y muestra un mensaje de error, permitiendo que vuelvan a intentar iniciar el proceso.

Debido a que las tres clases creadas corren en distintos hilos es necesario de un *handler* para pasar los datos entre sí. Un *handler* permite enviar y procesar objetos *Message* de un *thread* a otro, sirviendo de puente entre el hilo principal que corre la actividad y el resto de hilos [42]. En este ejemplo se emplea un *handler* para que ambos dispositivos conozcan el estado actual de la conexión y por tanto puedan mostrárselo al usuario a través de la interfaz. En la Figura 3-24 se muestra el funcionamiento del *handler* empleado, el cual cambia el texto de estado mostrado al usuario (líneas 121-132) y pasa a la clase encargada de la conexión el mensaje recibido (líneas 133-137). [43]

```
115 Handler handler = new Handler(new Handler.Callback() {
116     @Override
117     public boolean handleMessage(Message msg) {
118         switch (msg.what) {
119             case STATE_LISTENING:
120                 status.setText("Listenning");
121                 break;
122             case STATE_CONNECTING:
123                 status.setText("Connecting");
124                 break;
125             case STATE_CONNECTED:
126                 status.setText("Connected");
127                 break;
128             case STATE_CONNECTION_FAILED:
129                 status.setText("Connection failed");
130                 break;
131             case STATE_MESSAGE_RECEIVED:
132                 byte[] readBuff = (byte[]) msg.obj;
133                 String tempMsg = new String(readBuff, offset: 0, msg.arg1);
134                 msg_box.setText(tempMsg);
135                 break;
136         }
137         return true;
138     }
139 });
```

Figura 3-24 Ejemplo del funcionamiento de un *handler*

3.5 Desarrollo de la aplicación

En este apartado se detallará el funcionamiento de la aplicación generada. Para ello se explicarán las actividades que la componen:

- *SplashActivity*: pantalla introductoria, mostrada al comienzo de la aplicación, redirige al usuario al menú principal.
- *MainActivity*: menú principal, permite al usuario seleccionar entre el modo aprendizaje o evaluación.
- *AprendizajeActivity*: menú del modo aprendizaje, permite acceder a las dos actividades orientadas al estudio de las distintas banderas de señales.
 - *DiccionarioActivity*: listado empleado para el estudio de las banderas de señales.
 - *AdivinarActivity*: actividad enfocada al aprendizaje significativo, el objetivo es que el usuario relacione los nombres con las imágenes de las señales.
- *JuegoActivity*: menú del modo juego, permite acceder a las dos modalidades desarrolladas.
 - *LocalActivity*: modalidad de evaluación local, encargada de la generación de las preguntas.
 - *BTStartActivity*: actividad encargada de gestionar la conexión Bluetooth, así como las preguntas enviadas y recibidas.
 - Actividades que evalúan la capacidad de relacionar distintos elementos de las banderas de señales, basándose en los principios del aprendizaje significativo:
 - *BanderaActivity*: relacionar imagen con el nombre.
 - *ImagenActivity*: relacionar nombre con imagen.
 - *SignificadoActivity*: relacionar el significado con la imagen.
- Actividades auxiliares:
 - *PopUpActivity*: ventana emergente que, en caso de error, muestra la información de la señal seleccionada.
 - *ResultadoActivity*: actividad encargada de reflejar el resultado obtenido al finalizar las diversas actividades de evaluación.

La estructura general de la aplicación por actividades se muestra en la Figura 3-25.

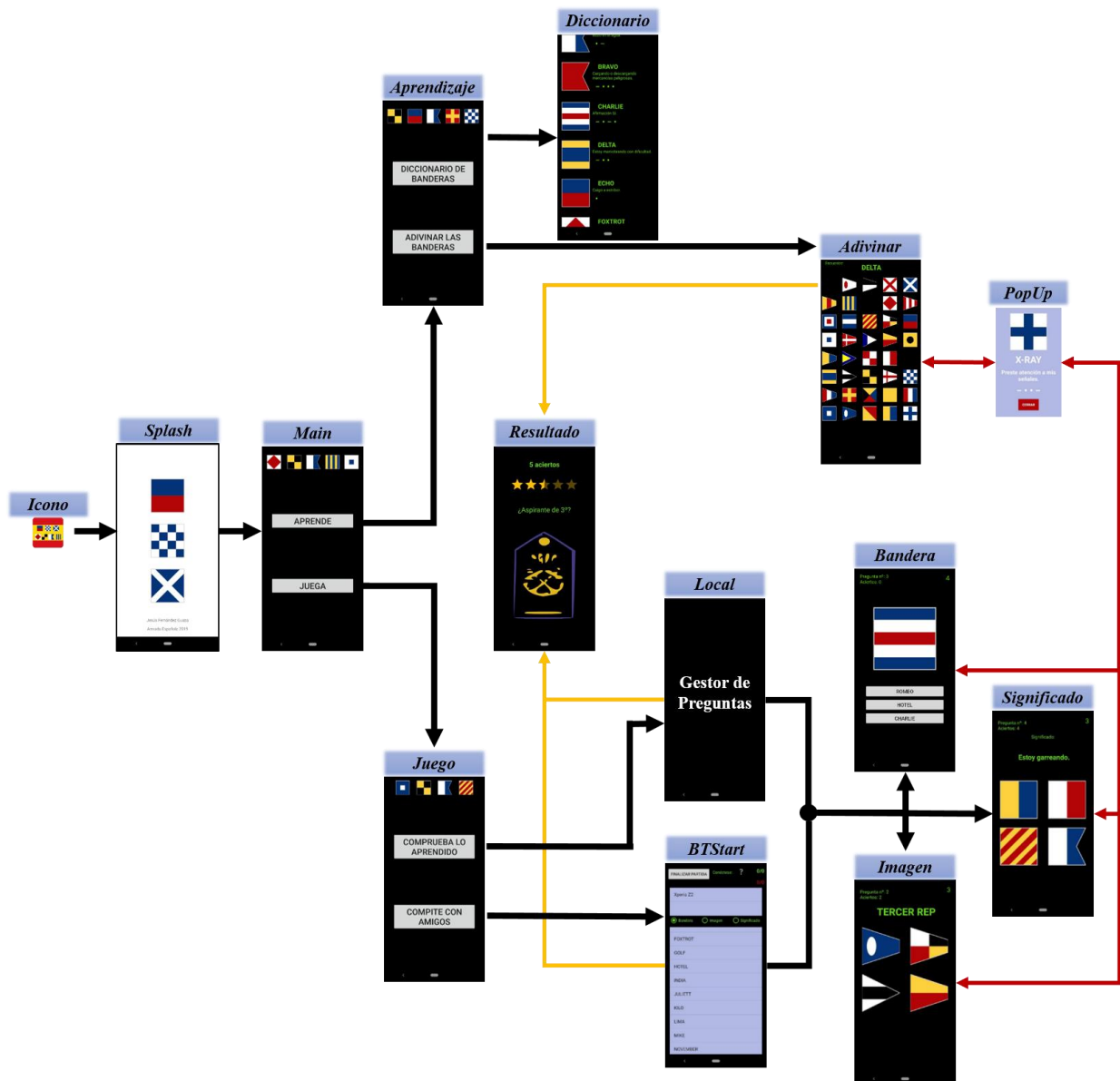


Figura 3-25 Funcionamiento de la aplicación

En los siguientes apartados se describe el funcionamiento de la aplicación ENM Banderas desarrollada en este proyecto, describiendo cada actividad.

3.5.1 Splash Screen

El *Splash Screen* es un elemento visual que contribuye a la imagen final de la aplicación, siendo comparable a la portada de un libro. Este insignificante detalle es importante, pues al existir una elevada competencia en el mundo de las aplicaciones no basta con el mero hecho de tener una buena aplicación, sino que esta ha de ser capaz de transmitir buenas sensaciones al usuario. Esta técnica es especialmente útil en el caso de grandes aplicaciones ya que durante el tiempo que se muestra la pantalla introductoria permite que el dispositivo cargue la aplicación. Aunque no es este el caso de la aplicación desarrollada en este proyecto, no se debe olvidar que nunca hay una segunda oportunidad para causar una primera buena impresión.

Los *Splash Screen* son elementos sencillos generalmente compuestos por un fondo claro, así como un logotipo y un breve texto. En el caso de esta aplicación se ha optado por mostrar un logotipo en el que aparecen las siglas de la ENM en banderas, así como un texto en el pie de la pantalla donde se

puede leer el nombre del desarrollador y “Armada Española 2019”, tal y como se muestra en la Figura 3-26. Esta es la primera actividad mostrada al usuario tras la pulsación del icono de la aplicación y sólo se muestra al usuario por un corto periodo de tiempo.

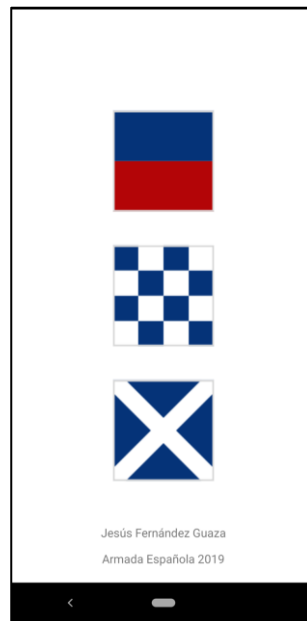


Figura 3-26 Splash Screen

Como puede apreciarse en la Figura 3-27, la implementación de esta actividad es muy sencilla. Sólo requiere el método *onCreate*, en el cual se declara el *layout* correspondiente. Finalmente, en las líneas 22 a 27, mediante el empleo de un *handler*, se retrasa la ejecución del *intent* encargado de cargar la actividad correspondiente al menú principal durante un segundo, tiempo durante el cual se mostrará al usuario la *Splash Screen* de la aplicación antes de que el *intent* inicie la siguiente actividad.

```
10 public class SplashActivity extends Activity {
11     private final int DURACION_SPLASH = 1000;
12
13     @Override
14     public void onCreate(Bundle savedInstanceState) {
15         super.onCreate(savedInstanceState);
16         setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
17         this.getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
18             WindowManager.LayoutParams.FLAG_FULLSCREEN);
19
20         setContentView(R.layout.activity_splash);
21
22         new Handler().postDelayed(() -> {
23             Intent intent = new Intent( packageContext: SplashActivity.this, MainActivity.class);
24             startActivity(intent);
25             finish();
26             }, DURACION_SPLASH);
27     }
28 }
29
30 }
```

Figura 3-27 Extracto de *SplashActivity.java*

3.5.2 Menú principal

En la Figura 3-28, se muestra una captura de pantalla del menú principal, que se compone en la actividad *ActivityMain.java*. En la parte más superior se puede leer “FLAGS” en banderas y debajo se encuentran los dos botones encargados de redirigir al modo aprendizaje o juego.

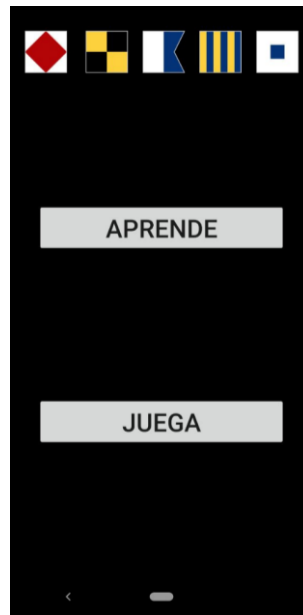


Figura 3-28 Menú principal

El código empleado para el funcionamiento de esta actividad, mostrado en la Figura 3-29, tiene una estructura muy similar al empleado en las actividades correspondientes a los modos aprendizaje y juego. Tras determinar el *layout* correspondiente (línea 16) se declaran las variables que definen los botones correspondientes a los dos modos (líneas 18-19), así como un *Event Listener* o Escuchador de Eventos (líneas 20-21). Este último se trata de un objeto tipo *View*, llamando al método *onClick* (línea 25) cada vez que se realiza una pulsación sobre los objetos en los que fue declarado. [44]

```

12 public class MainActivity extends AppCompatActivity implements View.OnClickListener {
13     @Override
14     protected void onCreate(Bundle savedInstanceState) {
15         super.onCreate(savedInstanceState);
16         setContentView(R.layout.activity_main);
17
18         Button baprendizaje = findViewById(R.id.baprendizaje);
19         Button bjuego = findViewById(R.id.bjuego);
20         baprendizaje.setOnClickListener(this);
21         bjuego.setOnClickListener(this);
22     }
23
24     @Override
25     public void onClick(View v) {
26         MetodosGlobales doble = new MetodosGlobales();
27         if(doble.dobleClick()) {
28             return;
29         }
30         switch (v.getId()) {
31             case R.id.baprendizaje:
32                 Intent i1 = new Intent( packageContext MainActivity.this, AprendizajeActivity.class);
33                 startActivity(i1);
34                 return;
35             case R.id.bjuego:
36                 Intent i2 = new Intent( packageContext MainActivity.this, JuegoActivity.class);
37                 startActivity(i2);
38                 return;
39             default:
40
41         }
42     }
43 }

```

Figura 3-29 Extracto de *MainActivity.java*

Al realizar una pulsación, se comprueba si esta es válida por medio del método *dobleClick*, perteneciente a la clase *MétodosGlobales.java* (líneas 26-27). Si la pulsación es válida, se evalúa por medio de un *switch* (línea 30) cuál de los objetos declarados ha sido seleccionado, iniciando a continuación la actividad correspondiente por medio de un *intent*.

3.5.3 Modo aprendizaje

La actividad correspondiente al modo aprendizaje es muy similar a la anterior. Sólo se diferencia en que las banderas mostradas conforman la palabra “LEARN” y dos botones redireccionan al usuario a las actividades correspondientes al diccionario de banderas y adivinar las banderas.

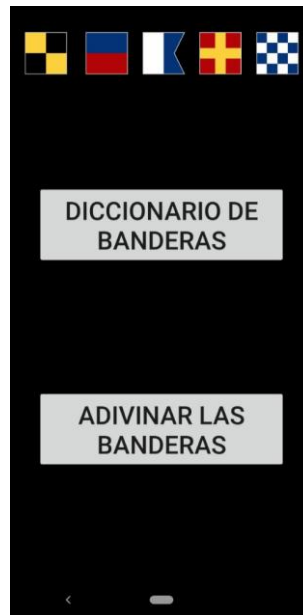


Figura 3-30 Modo aprendizaje

El funcionamiento de esta actividad es similar al anterior. La única diferencia es que en el caso de que el usuario quiera acceder al modo adivinar las banderas la actividad realiza unas operaciones necesarias para el correcto funcionamiento del modo adivinar, líneas 43 a 47 de la Figura 3-31:

- Modifica el valor de dos atributos pertenecientes a la clase variables globales:
 - Un *boolean* (línea 43) empleado para indicar a la otra actividad que es la primera vez que se ha creado, pues la actividad adivinar se inicializa y finaliza cíclicamente.
 - El entero (línea 44) encargado de registrar la actividad de origen, utilizado en la actividad encargada de mostrar el resultado.
- Asimismo, en la línea 47 se añade un *flag*. Los atributos *flag* se emplean para definir el funcionamiento de las actividades, en este caso *FLAG_ACTIVITY_NO_HISTORY*, se encarga de desactivar el historial de la pila de actividades Android. De esta forma, al pulsar el usuario el botón de retroceso, destruyendo la actividad *AdivinarActivity*, Android carga la última actividad que tiene en el historial, en este caso *AprendizajeActivity*. Esto permite alterar el funcionamiento de la pila de actividades Android acorde con las necesidades del desarrollador, siendo de gran utilidad en casos como el presente en el cual una actividad necesita alterar el historial. [45]

```

36      switch (v.getId()){
37          case R.id.bdiccionario:
38              Intent i1 = new Intent( packageContext: AprendizajeActivity.this,
39                                  DiccionarioActivity.class);
40              startActivity(i1);
41              return;
42          case R.id.badivinar:
43              VariablesGlobales.comienzo=true;
44              VariablesGlobales.pantallaActual=2;
45              Intent i2 = new Intent( packageContext: AprendizajeActivity.this,
46                                  AdivinarActivity.class);
47              i2.addFlags(Intent.FLAG_ACTIVITY_NO_HISTORY);
48              startActivity(i2);
49              return;
50          default:
51      }

```

Figura 3-31 Extracto de *AprendizajeActivity.java*

3.5.3.1 Diccionario de banderas

En esta actividad se listan todas las banderas del Código Internacional de Señales por orden alfabético. Este listado incluye para cada señal, la imagen correspondiente a la bandera, el nombre, el significado y el símbolo Scott correspondiente.

A pesar de que la aplicación tiene como fin el aprendizaje significativo de las banderas, esta actividad se basa en un aprendizaje mecanicista, pues permite que el usuario memorice los conceptos mediante la ejecución de acciones repetitivas. En el caso de esta actividad, el usuario puede leer y estudiar el significado de las banderas de forma estructurada, como se puede ver en la Figura 3-32. Pese a que este método de aprendizaje no proporciona conocimientos duraderos, sienta las bases necesarias para el aprendizaje significativo. Del mismo modo que se memorizan las tablas de multiplicar (aprendizaje mecanicista) y posteriormente se aprende a resolver problemas que requieran multiplicar (aprendizaje significativo). [10]



Figura 3-32 Diccionario de banderas

Como se puede ver en la Figura 3-33, este listado se hace por medio de un elemento *ListView*. Los distintos elementos listados se conforman por medio de la clase *CustomListview*. Esta clase sirve de adaptador, incluyendo en cada elemento la imagen, el nombre, significado y símbolo Scott de cada bandera.

```
21      imgid = VariablesGlobales.iimgid;
22      bandera = getResources().getStringArray(R.array.bandera);
23      significado = getResources().getStringArray(R.array.significado);
24      scott = getResources().getStringArray(R.array.scott);
25
26      lst= (ListView) findViewById(R.id.ListView);
27      CustomListview customListview=new CustomListview( context: this,bandera,significado,
28      |          scott,imgid);
29      lst.setAdapter(customListview);
```

Figura 3-33 Extracto de *DiccionarioActivity.java*

Cada elemento conformado en el *CustomListview* tiene el aspecto mostrado en la Figura 3-34. Esto permite simplificar enormemente el listado de los elementos, pues solo es necesario generar la estructura correspondiente a uno y esta se repite tantas veces como sea necesario.

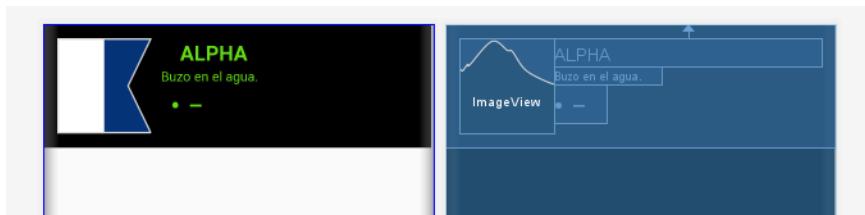


Figura 3-34 Diseño de cada elemento de *CustomListview*

3.5.3.2 Adivinar las banderas

Esta actividad se centra plenamente en el aprendizaje significativo. El objetivo de esta actividad es que el usuario vaya relacionando el nombre de la bandera con las imágenes mostradas, lo que permite aprender las banderas correspondientes a cada señal del Código Internacional de Señales.

Para ello muestra las banderas ordenadas aleatoriamente por medio de un *GridView*, de la misma manera que con el *ListView* de la actividad anterior, pero empleando un adaptador personalizado descrito en la clase *GridAdapter*. Los elementos de este adaptador están compuestos únicamente por un elemento *ImageView*. En caso de que el usuario seleccione una bandera es el adaptador el encargado de devolver la posición de esa bandera, posición que sirve para comprobar si la elección del usuario es correcta. En la Figura 3-35 se puede ver el funcionamiento de esta actividad. En este caso le pide al usuario que seleccione la imagen correspondiente a la bandera “Delta”.



Figura 3-35 Adivinar las banderas

Para conseguir este efecto es necesario implementar múltiples métodos los cuales se resumen en la Figura 3-36 y se detallan a continuación.

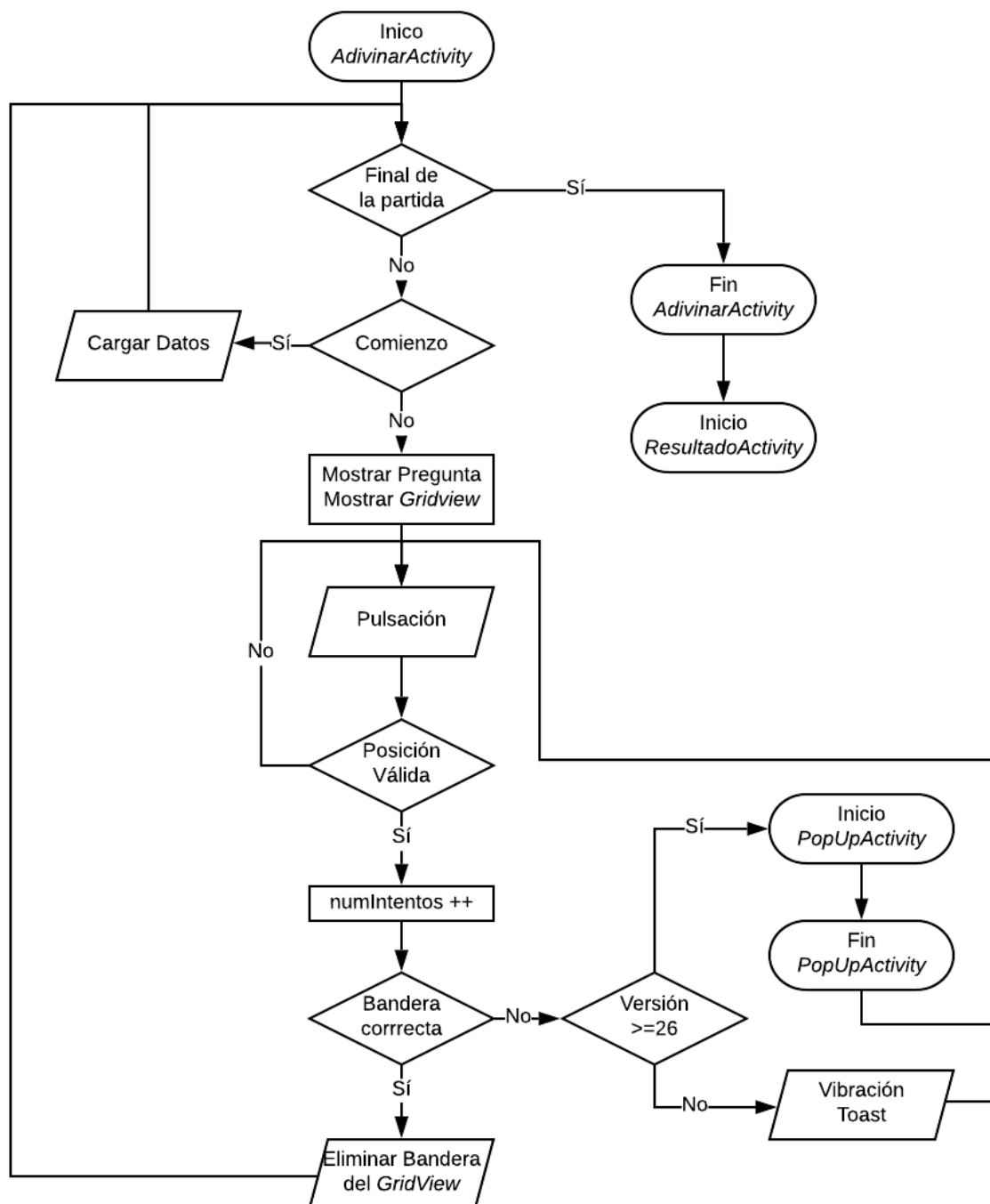


Figura 3-36 Funcionamiento básico de AdivinarActivity

Tras declarar en el método *onCreate* el *layout* correspondiente a la actividad y los atributos de la misma, se pasa a ejecutar el método *onResume* (Figura 3-37), en el cual se realizan las siguientes acciones:

- En primer lugar, se comprueba si se ha llegado al final de la partida, en cuyo caso se llama al método *finPartida* (líneas 44-46) finalizando la actividad y dando paso a la pantalla encargada de mostrar el resultado obtenido, *ResultadoActivity*.
- En caso de que no se haya llegado al final de la partida, se evalúa si ha de llamarse al método *cargarDatos* (líneas 48-51), encargado de asignar a los atributos los valores necesarios para reiniciar una partida.

- Asimismo, aunque por simplicidad del diagrama no se muestre en la Figura 3-36, se implementa un *SwipeRefreshLayout* (líneas 52-53). Este método consiste en la detección de un deslizamiento vertical dactilar, el cual redirige el código al método *cargarDatos*, generando una nueva partida. [46]
- A continuación, se asignan los recursos *TextView* y *GridView* (líneas 61-67) correspondientes al nombre de la bandera a preguntar y la matriz de imágenes a relacionar con el nombre anterior.
- Posteriormente, en las líneas 68-72 se declara un *listener* por medio del adaptador empleado en el *GridView*. Este es el encargado de detectar las pulsaciones y de devolver la posición correspondiente a la imagen pulsada. Una vez detectada la pulsación llama al método *cartaPulsada*.

```

41      @Override
42      protected void onResume() {
43          super.onResume();
44          if (numAciertos == imgidVar.length) {
45              finPartida();
46          }
47          else {
48              if (VariablesGlobales.comienzo) {
49                  VariablesGlobales.comienzo = false;
50                  cargarDatos();
51              }
52              SwipeRefreshLayout swipeRefreshLayout = findViewById(R.id.swipeRefreshLayout);
53              swipeRefreshLayout.setOnRefreshListener(() -> { cargarDatos(); });
54
55              //Pone una pregunta por orden
56              TextView tvPregunta = (TextView) findViewById(R.id.textView1);
57              tvPregunta.setText(bandera[numAciertos]);
58
59              //Pone el gridview
60              GridView gridView = (GridView) findViewById(R.id.griview);
61              GridAdapter gridAdapter = new GridAdapter( context: this, imgidVar);
62              gridView.setAdapter(gridAdapter);
63              gridView.setOnItemClickListener((parent, v, position, id) -> {
64                  cartaPulsada(position);
65              });
66          }
67      }

```

Figura 3-37 Método *onResume* de *AdivinarActivity.java*

El método *cartaPulsada* (Figura 3-38), es declarado junto con un atributo que corresponde a la posición en el índice de la bandera pulsada. El funcionamiento de este método es el siguiente:

- En primer lugar, comprueba que la pulsación se haya realizado sobre una imagen (línea 96), registrando el intento como válido (línea 97), este valor es empleado más adelante en *ResultadoActivity* para mostrar el número de intentos necesarios para completar la actividad.
- Posteriormente, se comprueba si el nombre de la bandera mostrada corresponde con la imagen seleccionada (línea 99), dando pie a otras dos opciones:
 - El nombre corresponde con la imagen seleccionada, esta desaparece y muestra el nombre de la siguiente bandera a buscar por el usuario, para ello se sustituye el índice correspondiente a la bandera correcta, por uno que referencia a una imagen sin contenido (línea 103) y se vuelve al comienzo de la actividad.
 - En caso de que se seleccione la imagen incorrecta, el dispositivo realizará distintas acciones dependiendo de la versión Android: si la versión del dispositivo igual o superior a la Android 8.0 Oreo, correspondiente a la API 26 (línea 108), se muestra

una ventana emergente en la que se muestra la información de la bandera pulsada, esto se realiza mediante una llamada a la *PopUpActivity*. Debido al elevado uso de memoria que requiere la presente actividad para mostrar todas las banderas, en caso de que la versión sea inferior a Android 8.0 Oreo, se estima que el dispositivo puede no tener la memoria suficiente para cargar la actividad encargada de la ventana emergente junto con esta actividad. Por ello limita la interacción a una vibración para alertar al usuario de que se ha equivocado y un mensaje *Toast* en el que se muestra el nombre de la bandera que ha seleccionado.

```

95 public void cartaPulsada(int position){
96     if(indexArray[position]<imgidVar.length){
97         VariablesGlobales.numeroIntentos= VariablesGlobales.numeroIntentos+1;
98         //Comprueba si ha acertado la pregunta
99         if(numAciertos==indexArray[position]){
100             VariablesGlobales.numeroAciertos= VariablesGlobales.numeroAciertos+1;
101             //Evita que el usuario pueda pulsar una carta que ya ha sido pulsada
102             indexArray[position]=imgidVar.length;
103             imgidVar[position]=R.drawable.nulo;
104             recreate();
105         }
106     else{
107         //Solucionar Problemas de versiones
108         if (Build.VERSION.SDK_INT >= 26) {
109             popUp(position);
110             Toast.makeText( context: AdivinarActivity.this, text: "Error",
111                 Toast.LENGTH_SHORT).show();
112         } else {
113             ((Vibrator) getSystemService(VIBRATOR_SERVICE)).vibrate( milliseconds: 150);
114             Toast.makeText( context: AdivinarActivity.this,bandera[indexArray[position]],
115                 Toast.LENGTH_SHORT).show();
116         }
117     }
118 }
119 }

```

Figura 3-38 Método *cartaPulsada* de *AdivinarActivity.java*

El método *cargarDatos* (Figura 3-39), como ya se ha mencionado anteriormente, tiene como función generar los valores de los atributos necesarios para comenzar una nueva partida. Para ello genera un vector que actúa de índice (*indexArray*), el cual posteriormente se ordena aleatoriamente y se emplea para generar y referenciar los punteros de las imágenes del vector *imgidVar* (línea 135). Este vector contiene las referencias correspondientes a las imágenes de banderas, pero estas se encuentran en el mismo orden que el *indexArray*, lo que permite que se muestren posteriormente en el *GridView* en posiciones pseudo-aleatorias.

```

126 private void cargarDatos() {
127     //Generar los arrays de imagenes e indice y despues volver a barajarlos
128     for (int i = 0; i < imgid.length; i++) {
129         indexArray[i] = i;
130     }
131     MetodosGlobales shuffle = new MetodosGlobales();
132     shuffle.shuffleArray(indexArray);
133
134     for (int i = 0; i < imgid.length; i++) {
135         imgidVar[i] = imgid[indexArray[i]];
136     }
137     VariablesGlobales.numeroIntentos=0;
138     VariablesGlobales.numeroAciertos=0;
139     recreate();
140 }

```

Figura 3-39 Método *cargarDatos* de *AdivinarActivity.java*

3.5.4 Modo juego

Una vez que el usuario tiene unas nociones básicas sobre las banderas de señales, este puede empezar a autoevaluarse, así como a asentar los conocimientos adquiridos. Para ello la aplicación dispone de dos modos de funcionamiento, mostrados en la Figura 3-40: uno en local, donde el usuario puede comprobar los conocimientos adquiridos; y uno multijugador donde, por medio de Bluetooth, puede retar y competir contra otro usuario de la aplicación. De la misma manera que en los otros menús las banderas dan una pista sobre el punto en el que se encuentra la actividad, en este caso, en el modo juego se puede leer “PLAY”.

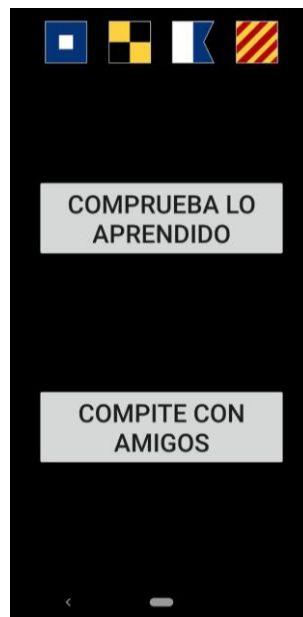


Figura 3-40 Modo juego

El funcionamiento de esta actividad es exactamente igual que *AprendizajeActivity* o *MainActivity*. La única diferencia reside en el funcionamiento del *switch* que evalúa qué botón fue pulsado y actúa en consecuencia, correspondiente a la Figura 3-21. En caso de pulsar el botón del modo local, registra en una variable el que se encuentra en esa pantalla e inicia la actividad encargada del juego en local, *LocalActivity*. En el caso de que se pulse el botón del modo multijugador, registra la pantalla, así como un valor que determina que aún no se ha conectado por Bluetooth, para que al iniciarse la siguiente actividad (*BTStartActivity*) comience el proceso de conexión.

```
35      switch (v.getId()){
36          case R.id.blocal:
37              Intent i1 = new Intent( packageContext: JuegoActivity.this, LocalActivity.class);
38              startActivity(i1);
39              VariablesGlobales.pantallaActual=1;
40              return;
41          case R.id.bbluetooth:
42              VariablesGlobales.pantallaActual=3;
43              VariablesGlobales.conectado =false;
44              Intent i2 = new Intent( packageContext: JuegoActivity.this, BTStartActivity.class);
45              startActivity(i2);
46              return;
47          default:
48      }
```

Figura 3-41 Extracto de *JuegoActivity.java*

3.5.4.1 Test local

La actividad que gestiona el test local es una actividad un tanto peculiar, puesto que su interacción con el usuario es nula, siendo el *layout* asociado una pantalla vacía. La utilidad de esta actividad es generar las preguntas y llamar a las distintas actividades correspondientes a las distintas pantallas de evaluación. Esto se debe a que la actividad *LocalActivity* redirige al usuario a distintos formatos de test con distinta dificultad en función del número de aciertos que vaya teniendo. Esto lo hace por medio del método *determinarDificultad*, mostrado en la Figura 3-42, el cual se ejecuta cada vez que el usuario responde a una pregunta. Para ello emplea un mecanismo similar a AIMD (*Additive Increase Multiplicative Decrease*, en español, Incremento Aditivo Decremento Multiplicativo), en el que se emplea un contador que aumenta de forma aditiva con cada acierto y se divide a la mitad en cada error. Esto permite que el usuario se enfrente a preguntas con una dificultad ascendente de forma que le suponga un reto conseguir acertar todas y que no le parezca muy sencillo evitando que pierda el interés.

```

72 private void determinarDificultad() {
73     //Similar a AIMD
74     if(VariablesGlobales.yoAcerte){
75         dificultad++;
76     }else{
77         dificultad=dificultad/2;
78         if(dificultad<0){
79             dificultad=0;
80         }
81     }
82     VariablesGlobales.yoAcerte=false;
83 }

```

Figura 3-42 Método *determinarDificultad* de *LocalActivity.java*

De esta manera, el usuario recibe siempre comienza con preguntas fáciles y estas se van complicando acorde este va acertando. Esto genera un reparto de preguntas permite que usuarios con muchos fallos reciban preguntas más simples, mientras que conforme se reduce el número de errores la dificultad aumenta, tal y como se puede ver en la Figura 3-43.

Reparto de preguntas en función del resultado

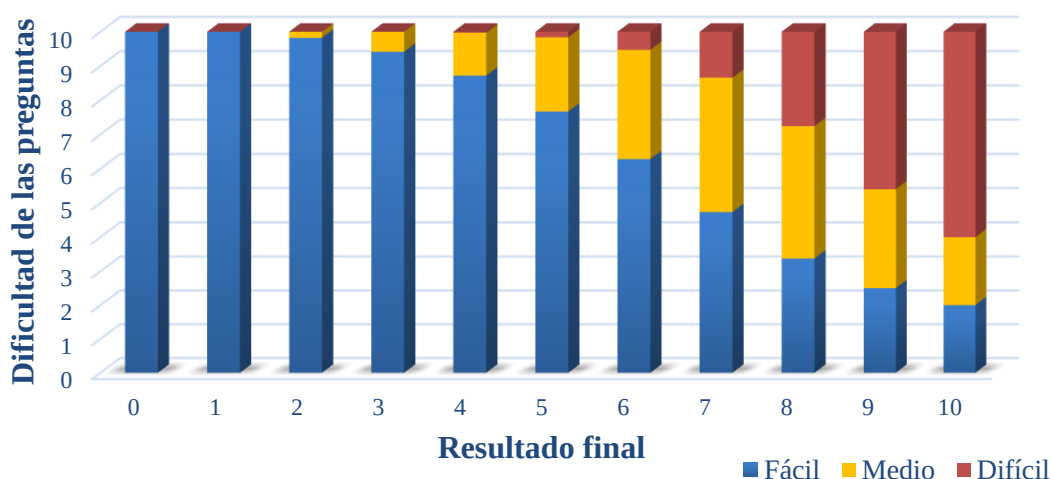


Figura 3-43 Reparto de preguntas por medio del método *determinarDificultad*

El funcionamiento del código empleado por la *LocalActiviy* para llamar a las distintas actividades correspondientes a los test se muestra en la Figura 3-44. En las líneas 47 a la 68, en función del valor del contador de aciertos se evalúan las distintas opciones:

- Si el valor es inferior a 2 inicia la actividad *BanderaActivity*, correspondiente al test más fácil.
- Si el valor del contador es 2 o 3 inicia *ImagenActivity*, test de dificultad intermedia.
- Si el valor es mayor o igual a 4 inicia *SignificadoActivity*, que contiene el test de mayor dificultad.

Asimismo, esta actividad no solo se encarga de decidir la dificultad de los test a realizar el usuario, también es la encargada de determinar cuándo finaliza la actividad, bien sea porque el usuario quiera salir (líneas 39-41) o bien porque ya se han contestado el número de preguntas preestablecido llamando a *ResultadoActivity* para mostrar el resultado final del test al usuario (líneas 48-51). También es la actividad encargada de llamar a *PopUpActivity* en caso de que el usuario hubiese respondido erróneamente a la pregunta anterior (líneas 44-46).

```
36      @Override
37      protected void onResume() {
38          super.onResume();
39          if(VariablesGlobales.exit){
40              finish();
41          }
42          else {
43              determinarDificultad();
44              if(VariablesGlobales.mostrarPopUp){
45                  startActivity(new Intent( packageContext: LocalActivity.this, PopUpActivity.class));
46              }
47              else {
48                  if (VariablesGlobales.numeroIntentos == preguntasMax) {
49                      this.finish();
50                      startActivity(new Intent( packageContext: LocalActivity.this,
51                          ResultadoActivity.class));
52                  } else {
53                      VariablesGlobales.mPregunta = preguntasArray[VariablesGlobales.numeroIntentos];
54                      //Aquí se podría determinar que test se abre
55                      if(dificultad<2){
56                          startActivity(new Intent( packageContext: LocalActivity.this,
57                              BanderaActivity.class));
58                      }
59                      else if(dificultad<4){
60                          startActivity(new Intent( packageContext: LocalActivity.this,
61                              ImagenActivity.class));
62                      }
63                      else {
64                          startActivity(new Intent( packageContext: LocalActivity.this,
65                              SignificadoActivity.class));
66                      }
67                  }
68              }
69          }
70      }
```

Figura 3-44 Método *onResume* de *LocalActivity.java*

3.5.4.2 Test Bluetooth

En este apartado se detallará únicamente el funcionamiento de la interfaz de usuario de la actividad, omitiendo por simplicidad el funcionamiento de la conexión Bluetooth (explicado anteriormente en la sección 3.4 Bluetooth en Android).

A diferencia de la actividad correspondiente al test local, esta permite al usuario interactuar con otros usuarios de la aplicación por medio de la conexión Bluetooth. Para ello, una vez conectados, puede enviar y recibir retos, que consisten en una pantalla de evaluación y una señal, ambas determinadas por el usuario. Esto fomenta el aprendizaje y una sana rivalidad entre los jugadores, los

cuales pueden gestionar sus propias normas conforme a la dificultad de las preguntas y, a diferencia del modo local, pueden enviar preguntas que consideren que el oponente no sabe, forzando que este finalmente aprenda todas las señales. Para ello la actividad consta de una interfaz muy básica (Figura 3-45) que permite al usuario:

- Obtener ayuda por medio de un botón de ayuda, guiando al usuario en caso de que no esté familiarizado con el funcionamiento de la actividad.
- Conectarse al dispositivo de su oponente.
- Seleccionar la dificultad de la pregunta enviada. En caso de no seleccionar ninguna la dificultad predeterminada es la fácil correspondiente a asociar la imagen de la bandera al nombre.
- Seleccionar dentro de un listado la pregunta a enviar.
- Conocer el estado de la partida, por medio de un marcador en la esquina superior derecha.
- Finalizar o salir de la partida.

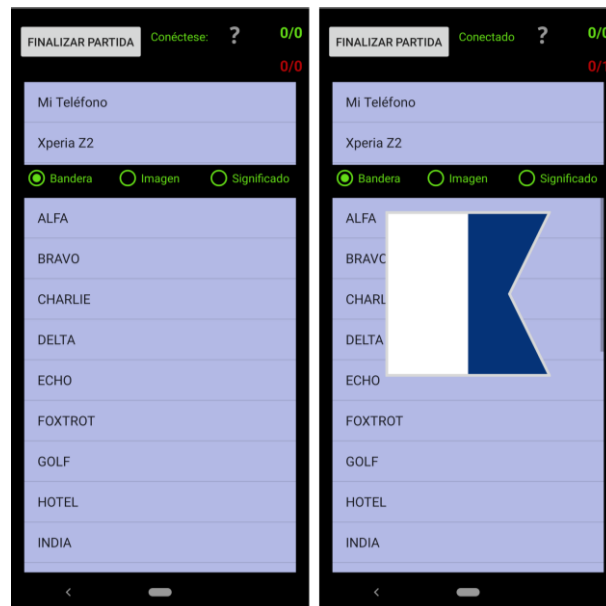


Figura 3-45 Test Bluetooth

Debido a la complejidad de esta actividad se decide incluir un botón de ayuda. En función del punto en el que se encuentre la actividad esta ayuda muestra distintos mensajes *Toast* que guían al usuario. Para ello se declara un escuchador (Figura 3-46) en el cual, al pulsar sobre el botón de ayuda, se valora si se ha realizado la conexión o no, respondiendo en cada caso con un *Toast* mostrando las instrucciones de uso para cada caso.

```

198  imgAyuda.setOnClickListener((v) -> {
201      if(VariablesGlobales.conectado) {
202          Toast.makeText( context: BTStartActivity.this,
203              text: "Seleccione y envíe un reto al otro dispositivo, o espere su turno",
204              Toast.LENGTH_LONG).show();
205      }
206      else {
207          Toast.makeText( context: BTStartActivity.this,
208              text: "Vincule y seleccione el dispositivo BT con el que quiere enlazar",
209              Toast.LENGTH_LONG).show();
210      }
211  });

```

Figura 3-46 Escuchador correspondiente al botón de ayuda de *BTStartActivity.java*

Tras declarar los elementos que conforman la pantalla de la Figura 3-45, se pasa a listar los dispositivos vinculados. Esto se hace por medio del método *prepareToConnect*, mostrado en la Figura 3-47. Cabe destacar que, por simplicidad de la aplicación, no es posible conectarse a dispositivos que no se encuentren previamente vinculados por Bluetooth, por lo que el usuario debe realizar previamente esta acción de forma nativa a través de los ajustes Bluetooth del dispositivo. Para listar los dispositivos se genera un vector de cadenas de caracteres en el cual se almacenan los nombres de los dispositivos enlazados (línea 170), la obtención de los nombres de los dispositivos se realiza por medio del adaptador Bluetooth (línea 177) [41]. Tras obtener los dispositivos conectados se listan por medio de un *Listview* (líneas 180-182). Finalmente se inicia la clase del servidor, lo que permite poner el dispositivo a la espera de una petición.

```
168 private void prepareToConnect() {
169     Set<BluetoothDevice> pairedDevices = bluetoothAdapter.getBondedDevices();
170     String[] strings = new String[pairedDevices.size()];
171     btArray = new BluetoothDevice[pairedDevices.size()];
172     int index = 0;
173
174     if (pairedDevices.size() > 0) {
175         for (BluetoothDevice device : pairedDevices) {
176             btArray[index] = device;
177             strings[index] = device.getName();
178             index++;
179         }
180         ArrayAdapter<String> arrayAdapter = new ArrayAdapter<>(getApplicationContext(),
181             android.R.layout.simple_list_item_1, strings);
182         listView.setAdapter(arrayAdapter);
183     }
184     ServerClass serverClass = new ServerClass();
185     serverClass.start();
186 }
```

Figura 3-47 Método *prepareToConnect* de *BTStatActivity.java*

Una vez listados los dispositivos, el usuario puede seleccionar uno de ellos para conectarse. Esto llama al *listener* encargado de escuchar los eventos en esa lista pasando a ejecutar el código mostrado en la Figura 3-48. En cuyo caso comprueba que no se encuentre ya conectado (línea 225) y posteriormente llama a la clase cliente, enviando la solicitud de conexión al dispositivo seleccionado.

Para establecer la conexión solo es necesario que un usuario envíe la petición al otro, da igual quien lo haga, pues ambos inician la clase del servidor. En caso de que ambos dispositivos envíen la petición simultáneamente, se genera un error en la conexión forzando que esta falle. Esto no es nada grave, simplemente basta con notificar al usuario de que la conexión falló y que es necesario que vuelva a intentar conectarse enviando otra petición, lo que se lleva a cabo en el *handler* empleado para la gestión de los múltiples hilos.

```
222 listView.setOnItemClickListener((parent, view, position, id) -> {
225     if(!VariablesGlobales.conectado) {
226         ClientClass clientClass = new ClientClass(btArray[position]);
227         clientClass.start();
228
229         status.setText("Conectando");
230     }
231 });
```

Figura 3-48 Escuchador de los dispositivos enlazados de *BTStartActiviy.java*

Una vez conectados los dispositivos, los usuarios pueden comenzar a enviarse preguntas entre sí. Antes de seleccionar la pregunta a enviar pueden elegir el tipo de pregunta que desean enviar, seleccionando entre las distintas pantallas disponibles, determinando así la dificultad del test. Para ello

se hace uso de un grupo de botones (*Radio Button*) [47]. Este tipo de control se emplea en casos como este, en el que sólo se permite seleccionar una opción entre una lista cerrada de alternativas. El comportamiento de estos controles es el mismo que el de cualquier otro botón: se realizan acciones por medio de un *listener*. Para notificar el modo de evaluación seleccionado por el usuario se envía un mensaje al otro dispositivo con el código correspondiente al modo seleccionado.

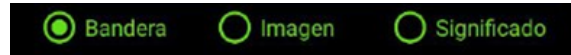


Figura 3-49 *Radio Button* del modo Bluetooth

Una vez el usuario ha seleccionado el tipo de pregunta, el siguiente paso es seleccionar la pregunta concreta a realizar. Para ello basta con seleccionar la pregunta de entre la lista de banderas generada en la parte inferior de la pantalla. Al pulsar sobre una se llama al *listener* correspondiente, devolviendo la posición de la bandera pulsada, enviando este valor al otro dispositivo. Con el fin de saber si el otro ha terminado de contestar desde el momento que se envía la pregunta hasta que se responde se muestra la bandera seleccionada, como se puede ver en la pantalla de la izquierda de la Figura 3-45. Para hacer esto se emplea el método *bringToFront* [48], este permite traer al frente los distintos elementos que componen la interfaz de usuario, por lo que al realizar una pregunta se cambia el contenido de la imagen por el de la bandera correspondiente y se trae al frente y al responder se llama al frente el listado de banderas, ocultando la imagen detrás.

Al recibir cualquier mensaje en el otro dispositivo este se gestiona por medio del *handler* (Figura 3-50). En este manejador se gestionan diversos aspectos en función del valor recibido (línea 321):

- Dificultad de las preguntas: entre la línea 342 y la 350 se comprueban los valores correspondientes a la selección del tipo de pregunta, estos son los comprendidos desde longitud del vector de las preguntas más 4 y más 6. Almacenando el modo de pregunta recibido en el atributo *modoPregunta*.
- Pregunta realizada: en caso de que el valor del mensaje corresponda con alguna posición del vector *bandera* (línea 353) significa que esa es la pregunta recibida, por lo que llama por medio de un *switch* a la actividad correspondiente al modo de pregunta descrito en el punto anterior (líneas 354-364).
- Otros parámetros: asimismo este manejador gestiona otros mensajes de utilidad como son:
 - Pregunta fallada: señal enviada automáticamente por el dispositivo cada vez que el usuario falla una pregunta, indicando al otro dispositivo el resultado obtenido. El valor de esta señal es la longitud del vector *bandera* (líneas 322-325).
 - Pregunta acertada: señal enviada automáticamente por el dispositivo cada vez que el usuario acierta una pregunta, indicando al otro dispositivo el resultado obtenido. El valor de esta señal es la longitud del vector *bandera* más uno (líneas 326-333).
 - Final deliberado de la partida: en caso de que el otro usuario decida finalizar la partida por medio del botón que se encuentra en la esquina superior izquierda de la pantalla se ejecuta el método *finPartida* (línea 335), encargado de finalizar la actividad y de llamar a la actividad responsable de mostrar el resultado.
 - Final inesperado de la partida: mensaje recibido en caso de que el otro usuario abandone la partida debido a que finalizó la actividad, simplemente muestra un *Toast* indicando que el otro usuario finalizó la conexión (líneas 337-341). A diferencia del otro este no fuerza una llamada a la actividad de resultados.

Finalmente, se incorpora un mecanismo de turnos de forma que inicialmente cualquiera de los dos usuarios pueda enviar una pregunta al otro. Al enviar la pregunta se bloquea la posibilidad de enviar otra pregunta hasta haber contestado una, es decir tras haber respondido a la pregunta del oponente. Esto evita múltiples problemas como son:

- Que un usuario envíe más preguntas que el otro.

- Recibir una pregunta mientras se está respondiendo a otra.
- Llegar a bloquear el dispositivo del otro usuario. Saturándolo mediante el envío de una gran cantidad de preguntas que a su vez generan un gran número de peticiones de inicio de actividad en el otro dispositivo, pudiendo llegar a considerarse un ataque por denegación de servicio.

```

319 case STATE_MESSAGE_RECEIVED:
320     byte[] readBuff = (byte[]) msg.obj;
321     int pregRecibida = byteArrayToInt(readBuff);
322     if(pregRecibida==bandera.length){
323         listViewPregunta.bringToFront();
324         Toast.makeText( context: BTStartActivity.this, text: "Falló",
325             Toast.LENGTH_SHORT).show();
326     }else if(pregRecibida==bandera.length+1){
327         VariablesGlobales.susAciertos= VariablesGlobales.susAciertos+1;
328         scoreEne.setText(VariablesGlobales.susAciertos + "/"
329             + VariablesGlobales.susIntentos);
330         listViewPregunta.bringToFront();
331         Toast.makeText( context: BTStartActivity.this, text: "Acertó",
332             Toast.LENGTH_SHORT).show();
333     }
334     else if(pregRecibida==bandera.length+2){
335         finPartida();
336     }
337     else if(pregRecibida==bandera.length+3){
338         listViewPregunta.bringToFront();
339         Toast.makeText( context: BTStartActivity.this, text: "Fin de la conexión",
340             Toast.LENGTH_SHORT).show();
341     }
342     else if(pregRecibida==bandera.length+4){
343         modoPregunta=0;
344     }
345     else if(pregRecibida==bandera.length+5){
346         modoPregunta=1;
347     }
348     else if(pregRecibida==bandera.length+6){
349         modoPregunta=2;
350     }
351     else if (pregRecibida >= 0 && pregRecibida < bandera.length) {
352         VariablesGlobales.mPregunta = pregRecibida;
353         switch (modoPregunta){
354             case 0:
355                 startActivity(new Intent( packageContext: BTStartActivity.this,
356                     BanderaActivity.class));
357                 break;
358             case 1:
359                 startActivity(new Intent( packageContext: BTStartActivity.this,
360                     ImagenActivity.class));
361                 break;
362             case 2:
363                 startActivity(new Intent( packageContext: BTStartActivity.this,
364                     SignificadoActivity.class));
365         }
366     }
367     break;

```

Figura 3-50 Extracto del *Handler* de *BTStartActivity.java*

En el momento en que uno de los dos usuarios decide finalizar la partida tiene dos opciones:

- Pulsar el botón “FINALIZAR PARTIDA”, mostrado en la esquina superior izquierda. Este botón comprueba que la partida se encuentre igualada, es decir que ambos jugadores

tengan el mismo número de intentos, en cuyo caso manda la petición de fin de partida, redirigiendo a los usuarios a la pantalla de resultados.

- Salir de la actividad, pulsando el botón de retroceso, mandando el mensaje correspondiente a fin de conexión para informar al usuario del otro dispositivo.

3.5.4.3 Pantallas de evaluación

La aplicación consta de tres actividades distintas empleadas para la evaluación de los conocimientos del usuario. El funcionamiento básico de las tres pantallas es muy similar. Solamente necesitan recibir una referencia con la posición de la pregunta para generar el test en sí, lo que permite elegir fácilmente cualquiera de ellas sin mayor complicación.

Las tres actividades generan preguntas de distinta índole y dificultad. A continuación, se resume el comportamiento de todas ellas, listadas en orden creciente de dificultad y cuyas pantallas correspondientes se muestran por orden de izquierda a derecha en la Figura 3-51.

- *BanderaActivity*: en esta actividad el usuario debe seleccionar la palabra que se asocia a la bandera mostrada entre un listado de tres alternativas.
- *ImagenActivity*: en esta actividad el usuario debe seleccionar la bandera que se asocia a la palabra mostrada entre 4 alternativas distintas
- *SignificadoActivity*: en esta actividad el usuario debe seleccionar la bandera que se asocia al significado mostrado entre 4 alternativas distintas



Figura 3-51 Pantallas de evaluación

Puesto que las tres funcionan de una forma muy similar, sólo se explicará el funcionamiento de *BanderaActivity*, siendo este extrapolable a las otras dos actividades.

Una vez declarado el *layout*, los elementos y los *listeners* correspondientes la actividad espera a que el usuario pulse alguna de las opciones disponibles, momento en el que ejecuta el método *onClick*, mostrado en la Figura 3-52. Este método, tras asegurarse de que la pulsación es válida (líneas 52-53), obtiene la respuesta seleccionada por medio de la identidad del objeto sobre el que se realizó la pulsación (líneas 55-65). Posteriormente, en función de si la respuesta es correcta o incorrecta llama al método *acierto* o *fallo* respectivamente (líneas 66-74). Asimismo, el método *acierto* se encuentra rodeado por un *try-catch*, puesto que este método contiene una excepción (necesaria para pausar el hilo). [49]

```
50 public void onClick(View v) {
51     //Evitar que afecte si el usuario hace doble click
52     MetodosGlobales doble = new MetodosGlobales();
53     if (doble.dobleClick()) {return;}
54     miContador.cancel();
55     switch (v.getId()) {
56         case R.id.radio_button1:
57             ans = 0;
58             break;
59         case R.id.radio_button2:
60             ans = 1;
61             break;
62         case R.id.radio_button3:
63             ans = 2;
64             break;
65     }
66     if (resp == opciones[ans]) {
67         try {
68             acierto();
69         } catch (InterruptedException e) {
70             e.printStackTrace();
71         }
72     } else {
73         fallo();
74     }
75 }
```

Figura 3-52 Método *onClick* de *BanderaActivity.java*

En la Figura 3-53 se describe el funcionamiento de los métodos *acierto* y *fallo*.

Al ejecutar el método *acierto* se incrementa el valor del contador de aciertos y se notifica a la actividad que lanza las distintas pantallas de test que se escogió la respuesta correcta (líneas 80-81). Asimismo, el método *acierto* es un tanto peculiar, pues requiere el uso de una *InterruptedException* [50] (línea 79), permitiendo pausar la ejecución del hilo por medio del método *sleep*, en este caso 300 milisegundos (línea 82) [42]. Este retraso se utiliza para que el usuario sea capaz de percibir que la respuesta era la correcta. Finalmente, se llama al método *finish*, finalizando la actividad y volviendo a la actividad desde la que gestionan las distintas preguntas.

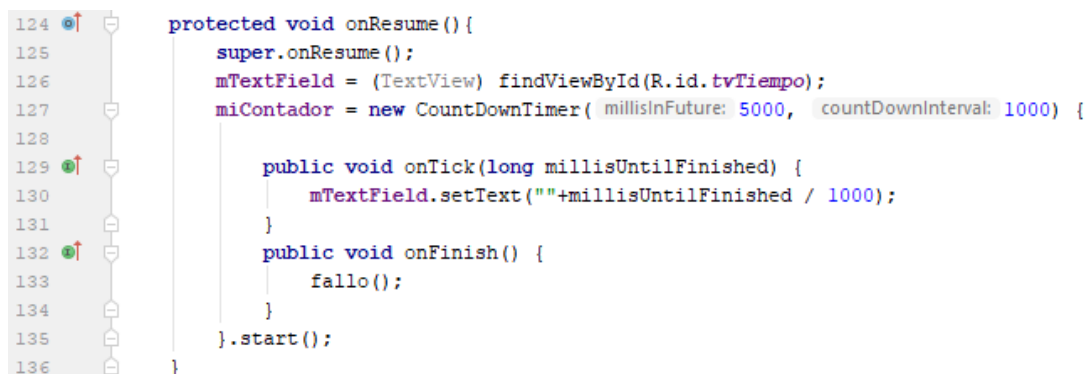
El método *fallo* funciona de una manera similar, lo único es que no requiere pausar la ejecución del hilo, pues el usuario recibe la notificación del error por medio de la *PopUpActivity*, llamada desde la actividad que gestiona las preguntas.

```
79 private void acierto() throws InterruptedException {
80     VariablesGlobales.numeroAciertos = VariablesGlobales.numeroAciertos + 1;
81     VariablesGlobales.yoAcerte=true;
82     Thread.sleep( millis: 300);
83     finish();
84 }
85
86 private void fallo() {
87     VariablesGlobales.mPopUp = resp;
88     VariablesGlobales.mostrarPopUp = true;
89     finish();
90 }
```

Figura 3-53 Métodos *acierto* y *fallo* de *BanderaActiviy.java*

Asimismo, para que la actividad sea más dinámica se añade un límite de tiempo para responder, esto se hace por medio de un temporizador mostrado en la esquina superior izquierda de la interfaz de usuario. La implementación de un temporizador requiere el uso de la clase *CountDownTimer* [51]. En este caso el temporizador se implementa en el método *onResume* (Figura 3-54) para que se inicie en el

instante anterior del acceso a la pantalla por parte de la actividad. Al declarar el temporizador (línea 127) se determina el tiempo para finalizar el temporizador y llamar al método *onFinish* (línea 132), así como el intervalo de tiempo entre llamadas al método *onTick* (línea 129), en este caso se fijó el tiempo total en 5 segundos con actualizaciones cada segundo. De este modo a los 5 segundos, si no se ha elegido ninguna alternativa se llama al método *fallo* (línea 133), de la misma manera, cada segundo se actualiza el valor del *TextView* que informa al usuario del tiempo restante (línea 130). Finalmente, hay que tener en cuenta que si no se detiene el temporizador este ejecutará el método *onFinish* al finalizar el tiempo, por ello, es necesario cancelarlo al pulsar sobre una de las opciones, esto se hace por medio del método *cancel* (mostrado en la Figura 3-52 línea 54), evitando que el temporizador llame al método *fallo* al cabo de los 5 segundos si el usuario ha seleccionado alguna opción.



```

124 protected void onResume() {
125     super.onResume();
126     mTextField = (TextView) findViewById(R.id.tvTiempo);
127     miContador = new CountdownTimer( millisInFuture: 5000, countDownInterval: 1000) {
128
129         public void onTick(long millisUntilFinished) {
130             mTextField.setText(""+millisUntilFinished / 1000);
131         }
132         public void onFinish() {
133             fallo();
134         }
135     }.start();
136 }

```

Figura 3-54 Método *onResume* de *BanderaActivity.java*

3.5.5 Actividades auxiliares

Además de las actividades que conforman la estructura básica de la aplicación es necesario el uso de otras que pueden ser llamadas desde cualquiera de las otras actividades con el fin de mostrar al usuario de una manera más intuitiva sus errores y resultados. Para ello se realizaron dos actividades auxiliares: un *pop-up* o ventana emergente por medio de la cual, en caso de fallo, se muestra la información de la bandera correspondiente; y otra actividad, mostrada al finalizar una actividad evaluable, en la que se refleja el resultado obtenido.

3.5.5.1 PopUp

Esta actividad consiste en una pantalla emergente mostrada al usuario cada vez que este responde incorrectamente. Se incluye además aquí una vibración encargada de alertar al usuario. En esta actividad se muestra toda la información correspondiente a la señal que se le indique por medio del valor registrado en *VariablesGlobales.mPopUp*. A diferencia del resto de actividades, esta actividad no ocupa la totalidad de la pantalla, provocando la sensación de que esta emerge de la anterior. Asimismo, es posible salir de esta pantalla bien pulsando el botón “CERRAR” o la parte de la pantalla que no corresponde a la actividad, en el caso de la Figura 3-55 sería la porción de la pantalla en negro.

Para conseguir las funcionalidades detalladas en el párrafo anterior es necesario en primer lugar declarar dentro del *AndroidManifest* los permisos necesarios para el uso del método *vibrate* [52]. Asimismo, dentro del método *onCreate* de *PopUpActivity* (Figura 3-56), el uso del método vibrar requiere comprobar la versión de Android sobre la que se está ejecutando la aplicación, pues el método empleado en la línea 34 se encuentra obsoleto a partir de Android 8.0, por ello se comprueba la versión (línea 30) y posteriormente se ejecuta el método correspondiente.



Figura 3-55 Ventana *PopUp*

Asimismo, con el fin de conseguir que la ventana sólo ocupe una porción de la pantalla, se obtiene el alto y ancho empleando los métodos *widthPixels* y *heightPixels* de la clase *DisplayMetrics* [53]. Posteriormente se declara el tamaño de la ventana empleada por la actividad, en este caso se optó por una ventana cuyas dimensiones fuesen el 80% del ancho original y el 70% del alto (línea 43).

```
29 //Vibrar en funcion del SDK
30 if (Build.VERSION.SDK_INT >= 26) {
31     ((Vibrator) getSystemService(VIBRATOR_SERVICE)).vibrate
32     (VibrationEffect.createOneShot( milliseconds: 150, amplitude: 10));
33 } else {
34     ((Vibrator) getSystemService(VIBRATOR_SERVICE)).vibrate( milliseconds: 150);
35 }
36
37 DisplayMetrics metr = new DisplayMetrics();
38 getWindowManager().getDefaultDisplay().getMetrics(metr);
39
40 int width = metr.widthPixels;
41 int height = metr.heightPixels;
42
43 getWindow().setLayout((int) (width*0.8),(int) (height*.7));
```

Figura 3-56 Extracto del método *onCreate* de *PopUpActivity.java*

Asimismo, se añade el atributo *android:theme="@style/AppTheme.Custom"* al declarar la actividad en el *AndroidManifest*. Esto asigna el tema *Custom*, almacenado en el archivo *styles.xml* (Figura 3-57). Por medio de diversos ítem se puede modificar el comportamiento de la actividad: permitiendo que la parte de la pantalla no empleada en la actividad sea transparente, mostrándose parte de la actividad predecesora (línea 12); o que al realizar una pulsación fuera de la actividad esta se finalice (línea 13). [54]

```
11 <style name="AppTheme.Custom">
12     <item name="android:windowIsTranslucent">true</item>
13     <item name="android:windowCloseOnTouchOutside">true</item>
```

Figura 3-57 Extracto de *styles.xml*

3.5.5.2 Resultados

Una vez que el usuario finalice cualquiera de los métodos de evaluación es necesario que sea informado de alguna manera de los resultados obtenidos. En función de la pantalla de origen se personaliza la interfaz dando un aspecto más dinámico. En la Figura 3-58 se muestran de izquierda a derecha las tres configuraciones básicas en función de la actividad de origen: adivinar las banderas, test local y test Bluetooth. Mostrando una configuración del texto distinto, acorde con el tipo de evaluación realizada.



Figura 3-58 Actividad resultados

Además, se opta por personalizar la aplicación, de manera que incite a superarse a sí mismo. Por ello, teniendo en cuenta el público hacia el que va dirigida la aplicación se decide darle un toque propio de la Escuela Naval Militar, como son los empleos de sus alumnos. La actividad premia asignando las divisas correspondientes a los distintos empleos, aumentando antigüedad conforme mejoran los resultados obtenidos. Asimismo, muestra un mensaje y una valoración por estrellas en función de los resultados.

Para realizar las anteriores acciones en el método *onCreate* (Figura 3-59) se implementa un *switch* en el cual se evalúa el modo de origen por medio de la *VariablesGlobales.pantallaActual* (líneas 30-43), determinando a continuación el texto a mostrar en cada caso.

En el método *onResume* (Figura 3-60), se implementa el código relativo a las estrellas mostradas, la frase de felicitación y un recurso gráfico en función del resultado.

Con el fin de conseguir una actividad más vistosa, se decide premiar al usuario con un número determinado de estrellas en función del resultado obtenido. Para ello se emplea un elemento *RatingBar* [55], que consiste en un sistema de valoración formado por un determinado número de estrellas, en este caso 5. Es un elemento en el que basta con declarar la cantidad de estrellas que se desea mostrar por medio de un *float* (número decimal), tal y como puede verse en la línea 61.

Asimismo, se desea mostrar una frase y una imagen acordes al resultado obtenido. Para ello se genera un vector de imágenes y uno de frases. Para seleccionar la posición de los valores a mostrar se emplea el valor barra de estrellas del apartado anterior. Debido a que este es un valor decimal antes hay que convertirlo a un valor entero (línea 55).

```
21 @Override
22 protected void onCreate(Bundle savedInstanceState) {
23     super.onCreate(savedInstanceState);
24     setContentView(R.layout.activity_resultado);
25
26     mNumeroIntentos= VariablesGlobales.numeroIntentos;
27     mNumeroAciertos= VariablesGlobales.numeroAciertos;
28
29     TextView textView = (TextView) findViewById(R.id.numAciertos);
30     switch(VariablesGlobales.pantallaActual) {
31         case 1:
32             textView.setText(mNumeroAciertos + " aciertos");
33             return;
34         case 2:
35             textView.setText("Acertaste las " + mNumeroAciertos + " banderas en "
36                             + mNumeroIntentos + " intentos ");
37             return;
38         case 3:
39             textView.setText(" Yo : " + mNumeroAciertos + "/" + mNumeroIntentos +
40                             "\nOtro: " + VariablesGlobales.susAciertos + "/" + VariablesGlobales.susIntentos);
41             return;
42         default:
43     }
44 }
```

Figura 3-59 Método *onCreate* de *ResultadoActivity.java*

```
46 @Override
47 protected void onResume() {
48     super.onResume();
49     Integer[] empleo = {R.drawable.nulo, R.drawable.ic_premio1, R.drawable.ic_premio2,
50                         R.drawable.ic_premio3, R.drawable.ic_premio4, R.drawable.ic_premio5};
51     String TextoPremios[] = getResources().getStringArray(R.array.listaPremios);
52     int quePremio;
53
54     mNumeroEstrellas=((float) mNumeroAciertos)/((float) mNumeroIntentos)*5;
55     quePremio = Math.round(mNumeroEstrellas);
56
57     RatingBar estrellas = (RatingBar) findViewById(R.id.ratingBar);
58     ImageView ivdelPremio = (ImageView) findViewById(R.id.ivPremio);
59     TextView tvdelPremio = (TextView) findViewById(R.id.tvPremio);
60
61     estrellas.setRating(mNumeroEstrellas);
62     tvdelPremio.setText(TextoPremios[quePremio]);
63     ivdelPremio.setImageResource(empleo[quePremio]);
64 }
```

Figura 3-60 Método *onResume* de *ResultadoActivity.java*

Debido a que *ResultadoActivity* es la última actividad mostrada al finalizar una evaluación, se desea que al cerrarla se restaure el valor de todos los contadores para evitar posibles fallos. Por ellos se opta por alterar el funcionamiento del botón de retroceso. Para ello se emplea el método *onKeyDown*, mostrado en la Figura 3-61. Este método evalúa la tecla pulsada y en caso de que sea el botón retroceso, correspondiente al evento *KEYCODE_BACK* (línea 69) pone todos los contadores a cero y finaliza la actividad (líneas 71-76). [56]

```
66      @Override
67      public boolean onKeyDown(int keyCode, KeyEvent event)
68      {
69          if ((keyCode == KeyEvent.KEYCODE_BACK))
70          {
71              VariablesGlobales.pantallaActual=0;
72              VariablesGlobales.numeroIntentos=0;
73              VariablesGlobales.numeroAciertos=0;
74              VariablesGlobales.susIntentos=0;
75              VariablesGlobales.susAciertos=0;
76              finish();
77          }
78          return super.onKeyDown(keyCode, event);
79      }
```

Figura 3-61 Método *onKeyDown* de *ResultadoActivity.java*

4 RESULTADOS

En este capítulo se detalla el proceso de distribución, el funcionamiento y las pruebas realizadas a la aplicación desarrollada en el presente proyecto.

4.1 Instalación de la aplicación

Antes de comenzar a usar la aplicación es necesario instalarla. Dado que no se dispone de cuenta de desarrollador de la Google Play Store es necesario obtener el archivo APK.

La generación del archivo APK es un proceso trivial. Una vez desarrollada y depurada la aplicación, basta con seleccionar en la barra herramientas de Android Studio *Build* → *Build Bundle(s) / APK(s)* → *Build APK(s)*, tal y como se muestra en la Figura 4-1. Esto generará un archivo APK dentro de la carpeta “\app\build\outputs\apk\debug”.

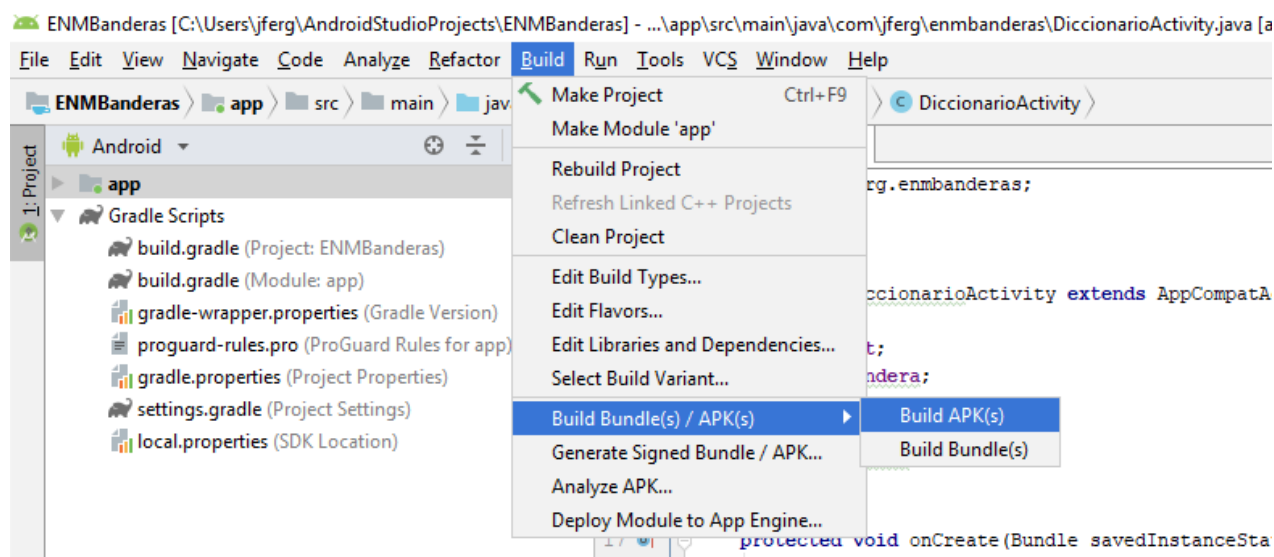


Figura 4-1 Generar una APK

Tras obtener el archivo APK que contiene el instalador de la aplicación es necesario distribuirlo a los futuros usuarios. Con el fin de facilitar la distribución se sube este archivo a Google Drive y se comparte mediante un enlace, el cual posteriormente se transforma a un código QR (*Quick Response Code*, en español, Código de Respuesta Rápida) por medio de las APIs ofrecidas por Google. El resultado obtenido es la Figura 4-2, que permite mediante un escaneo rápido con cualquier aplicación capaz de leer códigos QR redirigir al usuario al enlace de descarga del APK de la aplicación. [57]



Figura 4-2 Código QR para descargar el archivo APK

A continuación, el usuario tiene que activar “orígenes desconocidos” para poder instalar la aplicación, pues esta no se encuentra en la Google Play Store. Esto es un mecanismo de seguridad que evita que aplicaciones maliciosas se instalen en el dispositivo, por ello es recomendable desactivarlo una vez instalada la aplicación. Dependiendo de la versión de Android del dispositivo esta característica se activa de dos modos distintos. Para un dispositivo con Android 8.0 o superior basta con habilitar la instalación desde una ubicación específica, mientras que para los dispositivos con versiones anteriores es necesario activar esta característica en ajustes (Figura 4-3). [58]

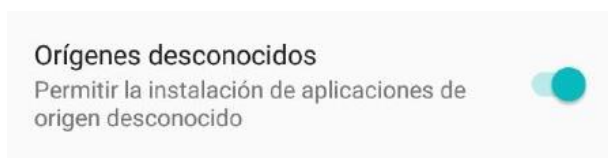


Figura 4-3 Permitir orígenes desconocidos en Android 7.1.1 y versiones anteriores

Una vez instalada la aplicación se genera el icono. A partir de este momento la aplicación funciona como una aplicación más del dispositivo.



Figura 4-4 Icono de la aplicación ENM Banderas

4.2 Funcionamiento de la aplicación

La aplicación ENM Banderas se compone de dos apartados fundamentales los cuales dividen la aplicación en dos grandes bloques: aprendizaje y juego. Esta separación se muestra ya en el menú principal donde el usuario decide a cuál de los dos acceder.

4.2.1 Aprendizaje de las señales

El objetivo principal de este modo es sentar las bases del conocimiento de las banderas de señales, para ello se subdivide en dos actividades:

- Una basada en el aprendizaje mecanicista, donde puede acceder a la información de las banderas correspondientes al Código Internacional de Señales. Esta pantalla sirve para sentar las bases necesarias para aprender las banderas de señales, así como para solventar dudas respecto a las mismas.

- Una segunda actividad fundamentada en el aprendizaje significativo, donde el usuario relaciona el nombre de la bandera mostrado con la imagen correspondiente, lo que permite asentar los conocimientos garantizando así que estos sean duraderos.

4.2.2 Evaluación de los conocimientos adquiridos

Una vez que el usuario se considera listo para evaluar sus conocimientos es el momento ideal para comprobar el dominio adquirido en la materia. Cabe reseñar que, aunque este apartado evalúa los conocimientos del usuario, el fin último es que este aprenda, razón por la cual, cuando este falla, aparece una ventana emergente con la respuesta. En función de si el usuario desea autoevaluar sus conocimientos o retar a un conocido existen dos modalidades:

- El modo de evaluación local, donde el usuario puede autoevaluar sus conocimientos por medio de una serie de tests cuya dificultad se adapta a los resultados obtenidos por el usuario. Al depender la dificultad de la prueba de los conocimientos del usuario se espera que al usuario le suponga un desafío obtener la máxima puntuación sin que este tenga la sensación de que es imposible y, por tanto, se vea tentado a abandonar el proceso de aprendizaje.
- El modo multijugador por Bluetooth permite que dos usuarios compitan entre sí. Esto genera una experiencia mucho más enriquecedora, ya que no solo es una motivación para que el usuario se esfuerce en aprender las banderas para ganar, sino también le supone un reto, pues el adversario es capaz de conocer y por tanto de intentar explotar sus debilidades y fallos más comunes. Asimismo, el que la conexión se establezca por medio de Bluetooth dota de una gran libertad en el uso de la aplicación pues esta no se encuentra limitada por factores externos como puede ser la ausencia de redes WiFi o acceso a Internet.

4.3 Plan de pruebas de la aplicación

El correcto funcionamiento de una aplicación es fundamental para conseguir una aceptación del usuario, pues esta no puede limitarse exclusivamente a tener unas elevadas prestaciones, sino que también es crucial que la interacción hombre máquina sea fluida. Por tanto, es fundamental que la aplicación carezca tanto de errores de funcionamiento como de diseño o de concepto.

Es evidente que esto se ha convertido en el objetivo principal de los desarrolladores, debido principalmente al cambio de política de Google a la hora de priorizar las búsquedas en la Play Store. Este cambio en el algoritmo de listado de las aplicaciones hace que se muestren primero las mejor calificadas y con un mejor rendimiento, en lugar del listado anterior basado en la coincidencia de búsqueda. De esta manera Google penaliza fuertemente a aquellas que sean de mala calidad, contengan fallos de funcionamiento o posible software malicioso [59]. Esto fuerza a los desarrolladores a ser muy meticulosos a la hora de probar su aplicación con el fin de conseguir una aplicación depurada y eficiente, obteniendo un buen puesto en el momento de la búsqueda y por tanto un mayor número de descargas.

Debido a esta concienciación existen múltiples herramientas como puede ser Monkey o Espresso [60] diseñadas para detectar los posibles fallos de las aplicaciones. A pesar de la existencia de herramientas que comprueben el funcionamiento de la aplicación, la única manera de encontrar los pequeños fallos que entorpecen la experiencia del usuario es comprobando meticulosamente su comportamiento actividad a actividad.

Con el fin de depurar la aplicación desarrollada en el presente proyecto se opta por emplear la herramienta Monkey, así como probar la manualmente el correcto funcionamiento de todas las actividades.

4.3.1 Herramienta Monkey

Monkey es una herramienta que funciona a través de comandos de terminal basada en el teorema del mono infinito [61]. La herramienta se basa en la generación de acciones pseudoaleatorias, correspondientes a acciones que un usuario podría ejecutar como puede ser una pulsación en la pantalla, deslizar la pantalla arriba y abajo, cambiar la orientación del dispositivo, etc. Por medio de estas iteraciones se puede probar el correcto funcionamiento de la aplicación. [62]

El empleo de esta herramienta es muy sencillo, basta con conectar el dispositivo Android en el que se quiere probar la aplicación y ejecutar unos comandos de terminal. El proceso consta de los siguientes pasos:

1. Conectar por USB el dispositivo en el que se encuentra instalada la aplicación con la depuración por USB activada.
2. Acceder a la herramienta Monkey. Para emplear esta herramienta desde Windows es necesario hacerlo a través de ADB (*Android Debug Bridge*, en español, Conector de Depuración de Android). La herramienta ADB [63] se encuentra dentro de la carpeta SDK de Android Studio, dentro de platform-tools.
3. Una vez en el directorio correspondiente a la herramienta ADB se introduce el siguiente comando:

```
adb shell monkey -p com.jferg.enmbanderas -v --ignore-crashes --pct-syskeys 0 5000 > registro.txt
```

Este comando llama a la herramienta Monkey a través de la *shell* (en español, intérprete de comandos) e introduce diversos argumentos, condicionando la ejecución de Monkey:

- *-p com.jferg.enmbanderas*: limita el uso de la herramienta a nuestra aplicación.
 - *-v*: determina el nivel de información mostrada al usuario, en este caso se emplea el nivel configurado por defecto, este solamente muestra una notificación de inicio y fin, así como los resultados obtenidos.
 - *--ignore-crashes*: fuerza que la herramienta siga funcionando tras encontrar fallos.
 - *--pct-syskeys 0*: determina el porcentaje de comandos enviados correspondientes con pulsaciones de las teclas retroceso, inicio, bloqueo, etc, en este caso se limita el uso de estas teclas asignando un porcentaje de comandos del 0%.
 - *5000*: determina el número de acciones a realizar por Monkey.
 - *> registro.txt*: genera el archivo de texto donde se guardan los resultados para consultar los posibles fallos encontrados.
4. Ejecutar el comando y esperar a que Monkey encuentre fallos. El tiempo depende principalmente del número de acciones, en este caso, el tiempo oscila los 20 segundos.
 5. Al encontrar un fallo se muestra en el terminal y queda guardado en el archivo de registro como es el caso de la Figura 3-17.

Como se puede ver en la Figura 4-5, la herramienta encontró un fallo en la actividad *BTStartActivity*, donde, al seleccionar cualquier opción de las preguntas a enviar antes de establecer una conexión, la aplicación intenta llamar a una referencia que no existe, causando un error que fuerza el cierre de la misma. Una vez detectado el error este es muy sencillo de solventar, simplemente hay que deshabilitar el *listener* hasta que se establezca la conexión.

A pesar de la enorme utilidad de esta herramienta a la hora de detectar errores, ha de tenerse en cuenta que solo es útil si se emplea con el objetivo encontrar errores que generen un fallo en la ejecución del código. Si se quiere encontrar errores que no generen interrupciones en el código como puede ser un patrón en la generación de respuestas, respuestas erróneas que cuenten como correctas, fallos en el contador, etc, es necesario probar la aplicación actividad a actividad de manera manual.

Simbolo del sistema

```
C:\Users\jferg\AppData\Local\Android\Sdk\platform-tools>adb shell monkey -p com.jferg.enmbanderas -v --ignore-crashes --pct-syskeys 0 5000 > registro.txt
args: [-p, com.jferg.enmbanderas, -v, --ignore-crashes, --pct-syskeys, 0, 5000]
arg: "-p"
arg: "com.jferg.enmbanderas"
arg: "-v"
arg: "--ignore-crashes"
arg: "--pct-syskeys"
arg: "0"
arg: "5000"
data="com.jferg.enmbanderas"
arg="--ignore-crashes" mCurArgData="null" mNextArg=4 argwas="--ignore-crashes" nextarg="--pct-syskeys"
arg="--pct-syskeys" mCurArgData="null" mNextArg=5 argwas="--pct-syskeys" nextarg="0"
data="0"
// CRASH: com.jferg.enmbanderas (pid 22390)
// Short Msg: java.lang.NullPointerException
// Long Msg: java.lang.NullPointerException: Attempt to invoke virtual method 'void com.jferg.enmbanderas.BTStartActivity$SendReceive.write(byte[])' on a null object reference
// Build Label: xiaomi/daisy/daisy_sprout:9/PKQ1.180917.001/V10.0.3.0.PDLMIXM:user/release-keys
// Build Changelist: V10.0.3.0.PDLMIXM
// Build Time: 1547456519000
// java.lang.NullPointerException: Attempt to invoke virtual method 'void com.jferg.enmbanderas.BTStartActivity$SendReceive.write(byte[])' on a null object reference
// at com.jferg.enmbanderas.BTStartActivity$3.onItemClick(BTStartActivity.java:154)
// at android.widget.AdapterView.performItemClick(AdapterView.java:318)
// at android.widget.AbsListView.performItemClick(AbsListView.java:1192)
// at android.widget.AbsListView$PerformClick.run(AbsListView.java:3169)
// at android.widget.AbsListView$3.run(AbsListView.java:4123)
// at android.os.Handler.handleCallback(Handler.java:873)
// at android.os.Handler.dispatchMessage(Handler.java:99)
// at android.os.Looper.loop(Looper.java:193)
// at android.app.ActivityThread.main(ActivityThread.java:6694)
// at java.lang.reflect.Method.invoke(Native Method)
// at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:493)
// at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:858)
```

Figura 4-5 Empleo de la herramienta Monkey

4.3.2 Búsqueda metódica de errores

Con el objetivo de encontrar errores se decide probar todas las actividades de la aplicación realizando metódicamente las siguientes comprobaciones:

- Revisión de los elementos gráficos que conforman la actividad.
 - Se detectaron dos imágenes de banderas con colores que no se correspondían con los del Código Internacional de Señales.
 - Se detectaron múltiples erratas en los significados de las señales.
- Realizar acciones ilógicas.
 - En la actividad encargada del Bluetooth no se cerraba la conexión al finalizar la actividad, provocando que el otro usuario fuese capaz de seguir enviando preguntas, pudiendo llegar a saturar el dispositivo.
 - Pulsar el botón “FINALIZAR PARTIDA” del modo multijugador antes de que se establezca la conexión. Al generar el mismo error que el encontrado por la herramienta Monkey se adoptó la misma solución, deshabilitar el *listener* hasta establecer la conexión.
- Realizar múltiples pulsaciones en todos los elementos.
 - Por medio de esta comprobación se detectó que ciertas acciones como es cambiar de actividad o pulsar una respuesta tenían un comportamiento errático. A raíz de esto se desarrolló el método global *dobleClick*.
- Acceder a la actividad tras haber accedido a otras actividades.
 - Se solventó un fallo en el restablecimiento de contadores al iniciar la actividad encargada del modo Bluetooth al volver del modo adivinar las banderas.
 - Se detectó que al finalizar la actividad no se estaban cerrando de forma adecuada los *sockets* Bluetooth, por lo que no se podían realizar nuevas conexiones al entrar por segunda vez. Para solventar esto se corrigió el cierre de los *sockets*, teniendo en cuenta que para cerrarlos correctamente es necesario determinar si el dispositivo se encuentra actuando de cliente o servidor, cerrando los *sockets* correspondientes en cada caso.

- Cambio de orientación de la pantalla.
 - Se bloqueó la disposición de todas las actividades en modo retrato, pues no se desarrolló ningún *layout* horizontal.
- Interacciones bruscas con elementos deslizantes.
 - No se detectó ningún fallo en ninguno de los listados.
- Comprobar funcionamiento de temporizadores.
 - Se detectó que si se salía del test pulsando el botón de retroceso los temporizadores no se detenían. La solución adoptada fue llamar al método *cancel* al pulsar el botón de retroceso, deteniendo el temporizador al salir del test.
- Activar y desactivar la vibración.
 - La aplicación funciona correctamente.
- Fallos que dependen de la versión de Android del dispositivo.
 - Un *smartphone* actual (2018), con Android 9.0 Pie (API 28), Xiaomi Mi A2 Lite, con una pantalla de 5,84 pulgadas. En el que no se detectaron fallos debido a la versión de Android empleada.
 - Un *smartphone* medio (2016), con Android 8.0 Oreo (API 26), Xiaomi Mi5, con una pantalla de 5,15 pulgadas. Al abrir la actividad correspondiente a la ventana emergente se muestra el siguiente mensaje de error:
java.lang.IllegalStateException:

Only fullscreen opaque activities can request orientation

Esto es debido a que en esta versión de Android existe un error que no permite restringir la orientación de la pantalla si esta es transparente, como es el caso de *PopUpActivity*. Con el fin de evitar este problema que sólo ocurre en esa actividad, en el método *onCreate* (Figura 4-6) se comprueba la versión, restringiendo la orientación en modo retrato sólo si la aplicación no se está ejecutando sobre Android 8.0.

```

29         if (Build.VERSION.SDK_INT != 26) {
30             setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
31         }

```

Figura 4-6 Extracto del método *onCreate* de *PopUpActivity.java*

- En un *smartphone* antiguo (2014), con Android 6.0.1 Marshmallow (API 23), un Sony Xperia Z2 con una pantalla de 5,2 pulgadas. Sin detectar inconvenientes de compatibilidad más allá de las limitaciones de memoria del Sony Xperia Z2. Lo que obligó a alterar el funcionamiento de *AdivinarActivity*, en la cual en función de la versión de Android del dispositivo se estima la memoria disponible del dispositivo, considerando que si la versión de este es igual o superior a Android 8.0 (correspondiente a la versión más baja sobre el que fue probado sin causar fallos por falta de memoria), se estima que no dispone de memoria suficiente y por tanto en vez de llamar a *PopUpActivity* al fallar simplemente se notifica al usuario por medio de un *Toast* y una vibración.
- Asimismo, reseñar que debido a que el método empleado para vibrar es distinto a partir de la versión de API 26, pues el empleado anteriormente se encuentra obsoleto [52], es necesario comprobar la versión del dispositivo para emplear el método correspondiente.
- Modificar el estado del Bluetooth.

- Causaba el cierre si se accedía a la actividad *BTStartActivity* con el Bluetooth apagado. La solución adoptada es evitar el acceso a la misma hasta que este recurso se encuentre disponible. Para ello en la actividad correspondiente al submenú de juego, *JuegoActivity*, se comprueba el estado del Bluetooth antes de iniciar la actividad (Figura 4-7), mostrando un mensaje *Toast* indicando al usuario que encienda el Bluetooth para poder usar esa actividad en caso de que este desactivado (líneas 47-48).

```
45 BluetoothAdapter bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
46 if (!bluetoothAdapter.isEnabled()) {
47     Toast.makeText(context: JuegoActivity.this, text: "Antes active el Bluetooth",
48         Toast.LENGTH_LONG).show();
49 }
50 else {
51     Intent i2 = new Intent(packageContext: JuegoActivity.this, BTStartActivity.class);
52     startActivity(i2);
53 }
```

Figura 4-7 Extracto del switch de *JuegoActivity.java*

- Si durante la conexión el usuario apagaba el Bluetooth, se generaba un fallo causando el cierre de la aplicación, por ello se implementó un método *BroadcastReceiver* [64]. Este está continuamente escuchando cambios en el estado del Bluetooth [41]. En caso de que durante la actividad se detecte un cambio de estado, se comprueba que el Bluetooth no esté apagado (líneas 527-530) ni que se esté apagando (líneas 532-536), en cuyo caso se fuerza el final de la actividad, evitando que se produzca el error en la aplicación, y se alerta al usuario, por medio de un *Toast* con el motivo por lo que se finalizó la actividad.

```
518 private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
519     @Override
520     public void onReceive(Context context, Intent intent) {
521         final String action = intent.getAction();
522
523         if (action.equals(BluetoothAdapter.ACTION_STATE_CHANGED)) {
524             final int state = intent.getIntExtra(BluetoothAdapter.EXTRA_STATE,
525                 BluetoothAdapter.ERROR);
526             switch (state) {
527                 case BluetoothAdapter.STATE_OFF:
528                     Toast.makeText(context: BTStartActivity.this,
529                         text: "Bluetooth desactivado", Toast.LENGTH_LONG).show();
530                     finish();
531                     break;
532                 case BluetoothAdapter.STATE_TURNING_OFF:
533                     Toast.makeText(context: BTStartActivity.this,
534                         text: "Se está apagando el Bluetooth", Toast.LENGTH_LONG).show();
535                     finish();
536                     break;
537             }
538         }
539     }
540 };
```

Figura 4-8 *BroadcastReceiver* de *BTStartActivity.java*

- Envío de peticiones Bluetooth simultáneas.
 - Causa un error en el proceso de enlace, pero no fuerza ningún cierre ni nada dañino, solamente requiere que se vuelva a enviar la petición de conexión, por lo que simplemente se muestra un mensaje que informa de un fallo de conexión para que el usuario vuelva a realizarla.

- Pérdida de la conexión Bluetooth por distancia.
 - Se pierde la conexión temporalmente hasta que ambos dispositivos vuelven a estar en alcance, por lo que no causa mayor problema.

4.3.3 Resultados de la búsqueda de errores

Tras comprobar en multitud de ocasiones, con múltiples dispositivos, las diferentes posibilidades intentando forzar fallos de todo tipo, se puede afirmar con cierta seguridad que la aplicación es lo suficientemente robusta como para que esta funcione de forma adecuada.

5 CONCLUSIONES Y LÍNEAS FUTURAS

En este capítulo se expondrán las conclusiones obtenidas a lo largo del presente proyecto y se esbozarán algunas de las posibles líneas futuras del mismo.

5.1 Conclusiones

El resultado final es una simple y ligera aplicación (el APK ocupa apenas 2 MB), cuyo objetivo principal es que el usuario aprenda de una manera interactiva el Código Internacional de Señales. A diferencia de las aplicaciones existentes, esta aplicación ha específicamente sido diseñada para su uso en el ámbito de la Escuela Naval Militar. Teniendo en cuenta que los usuarios involucrados en el proceso de aprendizaje se conocen, por lo que estos pueden retarse entre sí, y dado que los usuarios potenciales se encuentran físicamente cerca el medio más idóneo es por medio para la conexión es el Bluetooth pues no depende de factores externos. Asimismo, la aplicación procura despertar una sana rivalidad entre sus usuarios que incite al aprendizaje.

Con el fin de comprobar la utilidad de la aplicación actual se vuelve a realizar la comparativa de aplicaciones del capítulo 2 (Acorde a los criterios del Anexo I: Criterios de comparación de las aplicaciones), incluyendo una quinta columna con la aplicación desarrollada, ENM Banderas. El resultado se muestra en la Tabla 5-1.

	Nautical Flags and Signals	ICS Flag Trainer	Nautical Flags Helper	Marinus RIPA	ENM Banderas
Contenido	3	2	3	4	3
Idioma	1	2	1	4	3
Interacción	3	3	2	4	2
Evaluación	1	2	2	2	4
Plataforma	4	1	1	4	1
Precio	4	4	4	1	4
Valoración Final	16/24	14/24	13/24	19/24	17/24

Tabla 5-1 Comparativa de aplicaciones de aprendizaje de banderas de señales

5.2 Líneas futuras

Con respecto al futuro desarrollo de la aplicación se proponen las siguientes líneas futuras.

- Incluir más funcionalidades como pueden ser:
 - Completar las señales de la actual baraja, incluyendo las banderas OTAN e insignias de mando a bordo, esto requeriría generar los archivos SVG correspondientes a: las banderas numerales y de gobierno; el cuarto repetidor; los banderines de unidad, de gobierno y banderín especial; y las insignias. Esto hace un total 36 imágenes, lo que casi duplica la cantidad de señales actuales. Asimismo, habría que introducir el significado correspondiente a las 36 nuevas señales, así como el significado OTAN de las 26 letras y de la bandera “Code”. Por lo que alteraría sutilmente el funcionamiento de la aplicación pues a partir de este momento ha de discernir entre los significados del Código Internacional de Señales y los significados OTAN.
 - Apartados teóricos marítimos de distinta índole: RIPA, balizamiento, luces, navegación, astronomía, etc, con su correspondiente apartado de evaluación. De tal manera que los alumnos tengan todas las herramientas que necesitan para su formación en una sola aplicación.
 - Apartado de publicaciones, donde se redirija al usuario a las distintas publicaciones de referencia, de manera que este se familiarice con el uso de esta documentación, cosa que es indispensable tras su egreso.
 - Modo profesor que permita asignar tareas a sus alumnos y monitorizar los resultados obtenidos por estos, dotando a la Escuela Naval Militar de una cómoda herramienta para la evaluación de asignaturas como Adiestramiento a Flote.
 - Un registro de los errores más repetidos por el usuario de forma que este reciba una valoración más precisa y pueda centrarse en el estudio de aquellos conceptos que tiene menos arraigados.
 - Modo global que permita que los usuarios compitan entre sí, tanto a nivel local por medio del Bluetooth, como mundialmente a través de Internet. Se podrían emplear para ello las funcionalidades ofrecidas por redes sociales como Facebook [65].
- Añadir los idiomas francés e inglés, esto permitiría que la ampliación fuese destinada a un abanico de usuarios mucho mayor. Asimismo, no sólo serviría para aprender el significado de las banderas, sino que también se podría emplear como una herramienta de refuerzo para practicar un segundo idioma.
- Mejorar el acabado estético, de forma que la aplicación sea más intuitiva y vistosa.

Del mismo modo se proponen otras líneas futuras derivadas del desarrollo de la aplicación:

- Reutilizar el código de la aplicación, enfocándolo al aprendizaje de las orlas de los alumnos y profesores de la Escuela Naval Militar, así como de las dotaciones de las unidades. El funcionamiento sería fundamentalmente el mismo, pero alterando nombre e imágenes de las banderas por los de los integrantes de la unidad.
- Generar un servidor de aplicaciones conectado a la Intranet de Defensa, donde colgar esta aplicación, así como otras aplicaciones móviles de interés general para la Armada, como pueden ser aplicaciones de ayuda a la navegación, gestión de personal, etc.

6 BIBLIOGRAFÍA

- [1] International Maritime Organization, International Code of Signals, London: International Maritime Organization, ISBN-13: 978-92-801-0110-2, 2005.
- [2] National Imagery and Mapping Agency (US), International Code of Signals, Bethesda, Maryland: National Imagery and Mapping Agency (US), 2003.
- [3] A. Villalba Fernandez, «La bandera de buceo y el Código Internacional de Señales,» Buzos Argentinos, 21 Marzo 2012. [En línea]. Disponible: <https://web.archive.org/web/20140222004821/http://www.buzosargentinos.org.ar/colaboradores/nota.php?id=86>. [Último acceso: 26 Diciembre 2018].
- [4] H. Edles, Signal-book for the Ships of War, London: H.M. Stationery Office, 1779.
- [5] F. Marryat, Code of Signals for the Merchant Service, 10 ed., London: Admiralty, 1847.
- [6] Cabinet of Treasures, «Etsy,» [En línea]. Disponible: <https://www.etsy.com/es/listing/118839743/banderas-de-senal-maritima-internacional>. [Último acceso: 26 Diciembre 2018].
- [7] S. Mehling, «Creative Live Blog,» 20 Septiembre 2018. [En línea]. Disponible: <https://www.creativelive.com/blog/wp-content/uploads/2015/12/Cmyh7YS.jpg>. [Último acceso: 27 Diciembre 2018].
- [8] J. Sánchez García de Leonardo, *ORDEN DIARIA N° 022 PARA EL MARTES 22 DE ENERO DE 2019*, Marín, Pontevedra: Escuela Naval Militar, 2019.
- [9] Ministerio de Defensa, Instrucción 56/2011, de 3 de agosto, del Secretario de Estado de Defensa sobre sostenibilidad ambiental y eficiencia energética en el ámbito del Ministerio de Defensa., Boletín Oficial del Ministerio de Defensa, 2011.
- [10] I. Marín y E. Hierro, Gamificación. El poder del juego en la gestión empresarial y la conexión con los clientes., Ediciones Urano, ISBN-13: 978-8496627833, 2013.
- [11] A. Ramos, «Google Play,» Google Commerce Ltd, 14 Noviembre 2018. [En línea]. Disponible: <https://play.google.com/store/apps/details?id=com.nauticalflags>. [Último acceso: 27 Diciembre 2018].
- [12] DBG Nautical Brussels, «Google Play,» Google Commerce Ltd, 4 Octubre 2018. [En línea]. Disponible: <https://play.google.com/store/apps/details?id=com.dbg.ics&rdid=com.dbg.ics>.

[Último acceso: 26 Diciembre 2018].

- [13] Pirate Cats, «Google Play,» Google Commerce Ltd, 6 Noviembre 2018. [En línea]. Disponible: https://play.google.com/store/apps/details?id=com.napolovd.nautical.flagshelper&hl=en_US. [Último acceso: 27 Diciembre 2018].
- [14] Marinus, «Marinus,» 22 Agosto 2016. [En línea]. Disponible: <http://www.marinus.es/ES/>. [Último acceso: 27 Diciembre 2018].
- [15] World Wide Web Consortium's SVG Working Group, «W3C Recomendation,» W3C, 16 Agosto 2011. [En línea]. Disponible: <https://www.w3.org/TR/SVG11/>. [Último acceso: 27 Diciembre 2018].
- [16] R. Velasco, «Imágenes SVG: Qué son y en que se diferencian de las JPG,» SOFTZone, 15 Agosto 2016. [En línea]. Disponible: <https://www.softzone.es/2016/08/15/imagenes-svg-se-diferencian-las-jpg/>. [Último acceso: 27 Diciembre 2018].
- [17] Equipo Inkscape, «Inkscape,» [En línea]. Disponible: <https://inkscape.org/>. [Último acceso: 28 Diciembre 2018].
- [18] TechMetrix Research, «Java Application Servers Report,» 29 Diciembre 2010. [En línea]. Disponible: <https://web.archive.org/web/20101229090912/http://www.fscript.org/prof/javapassport.pdf>. [Último acceso: 2 Enero 2019].
- [19] Oracle, «Oracle Java Platform Integrator,» Oracle, 2017. [En línea]. Disponible: <https://www.oracle.com/technetwork/java/embedded/overview/ojpi-1972433.html>. [Último acceso: 04 Febrero 2019].
- [20] Oracle, «Introduction to JAXP,» Oracle Technology Network, 2008. [En línea]. Disponible: <https://www.oracle.com/technetwork/java/intro-140052.html>. [Último acceso: 1 Marzo 2019].
- [21] Open Handset Alliance, «Industry Leaders Announce Open Platform for Mobile Devices,» Open Handset Alliance, 7 Noviembre 2007. [En línea]. Disponible: http://www.openhandsetalliance.com/press_110507.html. [Último acceso: 3 Enero 2019].
- [22] Android Developers, «Arquitectura de la plataforma,» Google, 26 Octubre 2018. [En línea]. Disponible: <https://developer.android.com/guide/platform>. [Último acceso: 25 Febrero 2019].
- [23] Axway, «Appcellerator,» Axway, 3 Enero 2019. [En línea]. Disponible: <https://www.appcelerator.com/>. [Último acceso: 4 Enero 2019].
- [24] Microsoft, «Visual Studio Tools para Xamarin,» Microsoft, 1 Noviembre 2018. [En línea]. Disponible: <https://visualstudio.microsoft.com/es/xamarin/?rr=https%3A%2F%2Fbbvaopen4u.com%2F>. [Último acceso: 4 Enero 2019].
- [25] JetBrains, «IntelliJ IDEA,» Jet Brains, Enero 2019. [En línea]. Disponible: <https://www.jetbrains.com/idea/>. [Último acceso: 16 Febrero 2019].
- [26] Android Developers, «Documentation for app developers,» Google, 26 Octubre 2018. [En línea]. Disponible: <https://developer.android.com/docs/>. [Último acceso: 04 Febrero 2019].
- [27] Apple Inc., «Apple Developer Program,» Apple, 27 Septiembre 2018. [En línea]. Disponible: <https://developer.apple.com/programs/how-it-works/>. [Último acceso: 04 Enero 2019].

- [28] Google, «Cómo utilizar Play Console,» Google, [En línea]. Disponible: <https://support.google.com/googleplay/android-developer/answer/6112435?hl=es>. [Último acceso: 4 Enero 2019].
- [29] Jkielty, «Android v iOS market share 2019,» DeviceAtlas, 8 Enero 2019. [En línea]. Disponible: <https://deviceatlas.com/blog/android-v-ios-market-share>. [Último acceso: 17 Febrero 2019].
- [30] Slant, «What are the best IDEs for Android development?,» Slant, 16 Febrero 2019. [En línea]. Disponible: <https://www.slant.co/topics/1321/~best-ides-for-android-development>. [Último acceso: 16 Febrero 2019].
- [31] Android Developers, «Conoce Android Studio,» Google, 26 Octubre 2018. [En línea]. Disponible: <https://developer.android.com/studio/intro/?hl=es-419>. [Último acceso: 25 Febrero 2019].
- [32] Android Developers, «Proyectos recientes,» Google, 26 Octubre 2018. [En línea]. Disponible: <https://developer.android.com/studio/projects/>. [Último acceso: 25 Febrero 2019].
- [33] Android Developers, «Manifiesto de la app,» Google, 26 Octubre 2018. [En línea]. Disponible: <https://developer.android.com/guide/topics/manifest/manifest-intro?hl=es-419>. [Último acceso: 25 Febrero 2019].
- [34] Android Developers, «<uses-sdk>,» Google, 26 Octubre 2018. [En línea]. Disponible: <https://developer.android.com/guide/topics/manifest/uses-sdk-element?hl=es-419>. [Último acceso: 25 Febrero 2019].
- [35] Android Developers, «Android 5.0 API,» Google, 26 Octubre 2018. [En línea]. Disponible: <https://developer.android.com/about/versions/android-5.0.html>. [Último acceso: 25 Febrero 2019].
- [36] Android Developers, «Admisión de diferentes idiomas,» Google, 26 Octubre 2018. [En línea]. Disponible: <https://developer.android.com/training/basics/supporting-devices/languages?hl=es-419>. [Último acceso: 25 Febrero 2019].
- [37] Android Developers, «Diseños,» Google, 26 Octubre 2018. [En línea]. Disponible: <https://developer.android.com/guide/topics/ui/declaring-layout?hl=es-419>. [Último acceso: 25 Febrero 2019].
- [38] Android Developers, «Iniciar otra activity,» Google, 26 Octubre 2018. [En línea]. Disponible: <https://developer.android.com/training/basics/firstapp/starting-activity?hl=es-419>. [Último acceso: 25 Febrero 2019].
- [39] Android Developers, «Actividades,» Google, 26 Octubre 2018. [En línea]. Disponible: <https://developer.android.com/guide/components/activities?hl=es-419>. [Último acceso: 2019 Febrero 2019].
- [40] D. E. Knuth, The Art of Computer Programing, Tercera ed., vol. II, Boston: Addison-Wesley, ISBN-13: 978-0-201-89684-8, 1997.
- [41] Android Developers, «Bluetooth,» Google, 26 Octubre 2018. [En línea]. Disponible: <https://developer.android.com/guide/topics/connectivity/bluetooth?hl=es-419>. [Último acceso: 26 Febrero 2019].
- [42] Android Developers, «Thread,» Google, 26 Octubre 2018. [En línea]. Disponible: <https://developer.android.com/reference/java/lang/Thread>. [Último acceso: 27 Febrero 2019].

- [43] Android Developers, «Handler,» Google, 26 Octubre 2018. [En línea]. Disponible: <https://developer.android.com/reference/android/os/Handler>. [Último acceso: 26 Febrero 2019].
- [44] Android Developers, «View.OnClickListener,» Google, 26 Octubre 2018. [En línea]. Disponible: <https://developer.android.com/reference/android/view/View.OnClickListener>. [Último acceso: 27 Febrero 2019].
- [45] Android Developers, «Intent,» Google, 26 Octubre 2018. [En línea]. Disponible: <https://developer.android.com/reference/android/content/Intent>. [Último acceso: 26 Febrero 2019].
- [46] Android Developers, «Adding Swipe-to-Refresh To Your App,» Google, 26 Octubre 2018. [En línea]. Disponible: <https://developer.android.com/training/swipe/add-swipe-interface>. [Último acceso: 27 Febrero 2019].
- [47] Android Developers, «RadioButton,» Google, 26 Febrero 2018. [En línea]. Disponible: <https://developer.android.com/guide/topics/ui/controls/radiobutton>. [Último acceso: 28 Febrero 2019].
- [48] Android Developer, «View,» Google, 26 Octubre 2018. [En línea]. Disponible: <https://developer.android.com/reference/android/view/View>. [Último acceso: 3 Marzo 2019].
- [49] Android Developers, «Exception,» Google, 26 Octubre 2018. [En línea]. Disponible: <https://developer.android.com/reference/java/lang/Exception>. [Último acceso: 27 Febrero 2019].
- [50] Android Developers, «InterruptedException,» Google, 26 Octubre 2018. [En línea]. Disponible: <https://developer.android.com/reference/java/lang/InterruptedException>. [Último acceso: 27 Febrero 2019].
- [51] Android Developers, «CountDownTimer,» Google, 26 Octubre 2018. [En línea]. Disponible: <https://developer.android.com/reference/android/os/CountDownTimer>. [Último acceso: 3 Marzo 2019].
- [52] Android Developers, «VibrationEffect,» Google, 26 Octubre 2018. [En línea]. Disponible: <https://developer.android.com/reference/android/os/VibrationEffect>. [Último acceso: 27 Febrero 2019].
- [53] Android Developers, «DisplayMetrics,» Google, 26 Octubre 2018. [En línea]. Disponible: <https://developer.android.com/reference/android/util/DisplayMetrics>. [Último acceso: 27 Febrero 2019].
- [54] Android Developers, «Estilos y temas,» Google, 26 Octubre 2018. [En línea]. Disponible: <https://developer.android.com/guide/topics/ui/themes?hl=es-419>. [Último acceso: 27 Febrero 2019].
- [55] Android Developers, «RatingBar,» Google, 26 Octubre 2018. [En línea]. Disponible: <https://developer.android.com/reference/android/widget/RatingBar>. [Último acceso: 2019 Febrero 2019].
- [56] Android Developers, «Handle keyboard actions,» Google, 26 Octubre 2018. [En línea]. Disponible: <https://developer.android.com/training/keyboard-input/commands>. [Último acceso: 27 Febrero 2019].
- [57] Android Developers, «Preparar el lanzamiento,» Google, 26 Octubre 2018. [En línea]. Disponible: <https://developer.android.com/studio/publish/preparing?hl=es-419>. [Último acceso: 26 Febrero 2019].

- [58] Android Developers, «Opciones de distribución alternativas,» Google, 26 Octubre 2018. [En línea]. Disponible: <https://developer.android.com/distribute/marketing-tools/alternative-distribution>. [Último acceso: 26 Febrero 2019].
- [59] Android Developers Blog, «Improving discovery of quality apps and games on the Play Store,» Google, 29 Junio 2018. [En línea]. Disponible: <https://android-developers.googleblog.com/2018/06/improving-discovery-of-quality-apps-and.html>. [Último acceso: 24 Febrero 2019].
- [60] Android Developers, «Probar tu app,» Google, 26 Octubre 2018. [En línea]. Disponible: <https://developer.android.com/studio/test>. [Último acceso: 26 Febrero 2019].
- [61] X. Yang, S. F. Chien and T. O. Ting, Bio-Inspired Computation in Telecommunications, Waltham: Morgan Kaufmann, ISBN-13: 978-0128017432, 2015.
- [62] Android Developers, «UI/Application Exerciser Monkey,» Google, 26 Octubre 2018. [En línea]. Disponible: <https://developer.android.com/studio/test/monkey>. [Último acceso: 26 Febrero 2019].
- [63] Android Developers, «Android Debug Bridge,» Google, 26 Octubre 2018. [En línea]. Disponible: <https://developer.android.com/studio/command-line/adb?hl=es-419>. [Último acceso: 2 Marzo 2019].
- [64] Android Developers, «BroadcastReceiver,» Google, 26 Octubre 2018. [En línea]. Available: <https://developer.android.com/reference/android/content/BroadcastReceiver>. [Último acceso: 4 Marzo 2019].
- [65] Facebook Developers, «Inicio de sesión con Facebook para Android: inicio rápido,» Facebook, 2019. [En línea]. Available: https://developers.facebook.com/docs/facebook-login/android/?locale=es_ES. [Último acceso: 03 Marzo 2019].

ANEXO I: CRITERIOS DE COMPARACIÓN DE LAS APLICACIONES

El objeto de este anexo es detallar los criterios empleados a la hora de realizar la comparativa de las distintas aplicaciones estudiadas en el presente proyecto.

Para obtener una valoración lo más objetiva posible para ello se tienen en cuenta los siguientes parámetros, valorándose estos del 1 al 4 acorde con los criterios establecidos a continuación:

- Contenido: este apartado valora la cantidad y calidad del contenido didáctico.
 1. Sólo banderas de señales.
 2. Además de lo anterior incluye el significado de las señales
 3. Incluye los contenidos de los puntos 1 y 2 así como el método Scott o semáforo.
 4. Además de lo anterior añade otras materias de interés.
- Idioma: valora el idioma o idiomas en las que se encuentra disponible.
 1. Un único idioma que no es el castellano.
 2. Múltiples idiomas, pero ninguno en castellano.
 3. Disponible en castellano.
 4. Disponible en múltiples idiomas entre los que se encuentra el castellano.
- Interacción: tiene en cuenta el grado de depuración de la interfaz de usuario.
 1. Interfaz con errores.
 2. Interfaz poco intuitiva y/o compleja.
 3. Aplicación intuitiva y depurada.
 4. Aplicación sencilla, depurada y personalizable.
- Evaluación: pondera los distintos métodos empleados para determinar los conocimientos adquiridos.
 1. No dispone de nada para que el usuario evalúe sus conocimientos.
 2. Dispone de un tipo test para evaluar los conocimientos.
 3. Dispone de varios medios para evaluar los conocimientos del usuario.
 4. Permite no solo la autoevaluación sino la competición con otros usuarios.
- Plataforma: valora la disponibilidad de la aplicación en diversos sistemas operativos.
 1. Sólo disponible para Android/iOS.
 2. No evaluable.
 3. No evaluable.
 4. Disponible para Android e iOS.
- Coste: tiene en cuenta el gasto requerido por el usuario para adquirir la aplicación
 1. De pago, cuyo valor sea mayor a 1€.
 2. De pago, pero asequible, no superior a 1€.
 3. Gratuito, pero versión *lite* o reducida.
 4. Gratuito.

ANEXO II: RELACIÓN DE SEÑALES CORRESPONDIENTES AL CÓDIGO INTERNACIONAL DE SEÑALES

Índice	Bandera	Significado	Scott
0	ALFA	Buzo en el agua.	• —
1	BRAVO	Cargando o descargando mercancías peligrosas.	— • • •
2	CHARLIE	Afirmación SI.	— • — •
3	DELTA	Estoy maniobrando con dificultad.	— • •
4	ECHO	Caigo a estribor.	•
5	FOXTROT	Tengo avería, comuníqueme conmigo.	• • — •
6	GOLF	Necesito práctico.	— — •
7	HOTEL	Tengo práctico a bordo.	• • • •
8	INDIA	Caigo a babor.	— • — —
9	JULIETT	Tengo incendio y llevo a bordo mercancías peligrosas.	• — — —
10	KILO	Deseo comunicar con usted o invitación para transmitir.	• — •
11	LIMA	Pare su buque inmediatamente.	• — • •
12	MIKE	Mi buque está parado y sin arrancada.	— —
13	NOVEMBER	Negación NO.	— •
14	OSCAR	Hombre al agua.	— — —
15	PAPA	En puerto, regresar a bordo para hacerse a la mar.	• — — •
16	QUEBEC	Mi buque está sano.	— — • —
17	ROMEO	Recibido.	• — •
18	SIERRA	Estoy dando atrás.	• • •
19	TANGO	Manténgase alejado de mí.	—
20	UNIFORM	Se dirige usted hacia un peligro.	• • —
21	VICTOR	Necesito auxilio.	• • • —
22	WHISKEY	Necesito asistencia médica.	• — —
23	X-RAY	Preste atención a mis señales.	— • • —
24	YANKEE	Estoy garreando.	— • — —
25	ZULU	Necesito remolcador.	— — • •

26	NADAZERO	Pennant 0	— — — — —
27	UNAONE	Pennant 1	• — — — —
28	BISSOTWO	Pennant 2	• • — — —
29	TERRATHREE	Pennant 3	• • • — —
30	KARTERFOUR	Pennant 4	• • • • —
31	PANTAFIVE	Pennant 5	• • • • •
32	SOXISIX	Pennant 6	— • • • •
33	SETTESEVEN	Pennant 7	— — • • •
34	OKTOEIGHT	Pennant 8	— — — • •
35	NOVENINE	Pennant 9	— — — — •
36	CODE	Separador decimal.	
37	Primer Repetidor	Repita el significado de la primera señal.	
38	Segundo Repetidor	Repita el significado de la segunda señal.	
39	Tercer Repetidor	Repita el significado de la tercera señal.	