Trabajo Fin de Grado
en Ingeniería de la Energía

Plataforma de recogida de acciones de usuario en dinámica de sistemas

Autor: José María Martínez Bellido

Tutor: José María Maestre Torreblanca

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2025

Trabajo Fin de Grado
en Ingeniería de la Energía

Plataforma de recogida de acciones de usuario en dinámica de sistemas

Autor:
José María Martínez Bellido

Tutor:
José María Maestre Torreblanca
Catedrático de Universidad

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2025

Trabajo Fin de Grado: Plataforma de recogida de acciones de usuario en dinámica de sistemas

Autor: José María Martínez Bellido

Tutor: José María Maestre Torreblanca

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2025

El Secretario del Tribunal

A mi familia

A mis amigos

A mis compañeros

Agradecimientos

Este trabajo marca el fin de una época. Siete años de mi vida que he dedicado a estudiar esta carrera. Siete años en los que he conocido personas a las que llevaré conmigo el resto de mi vida, aunque solo sea en mis recuerdos.

Gracias a mi familia, por creer en mí incluso cuando yo no lo hice.

Gracias a mis amigos, acompañarme y apoyarme en mis peores momentos.

Gracias a todos los compañeros, que me han ayudado a lo largo de estos años universitarios, por todos los buenos momentos y las risas.

José María Martínez Bellido

Sevilla, 2025

Resumen

El objetivo principal de este trabajo es el desarrollo e implementación de una interfaz gráfica funcional con el formato de una aplicación móvil para el sistema operativo *Android* con el objetivo de la recopilación de datos durante las partidas jugadas. La aplicación está basada en el juego desarrollado por John D. Sterman durante su estancia en el MIT. Se han descrito detalladamente los pasos seguidos para la configuración de los distintos métodos de inicio de sesión, permitiendo a los usuarios acceder a la aplicación de manera segura y eficiente.

Además, se ha configurado una base de datos donde se almacenarán los resultados de las distintas partidas jugadas por los usuarios, así como otros datos relevantes. Asimismo, se presenta la labor de diseño y programación de la interfaz gráfica, asegurando una experiencia de usuario intuitiva.

Finalmente, el trabajo también expone la comunicación con los servidores de *Firebase*, una plataforma que facilita la autenticación, el almacenamiento y la sincronización de datos en tiempo real.

Índice

Índice de Figuras

Índice de Tablas

Índice de Código

Notación

API	Application Programming Interface
ETSI	Escuela Técnica Superior de Ingeniería
JDK	Java Development Kit
MIT	Massachusetts Institute of Technology
PX	Píxeles
PT	Píxeles Lógicos o Puntos
PPI	Píxeles por Pulgada
SDK	Software Development Kit
UID	Unique Identifier

1 INTRODUCCIÓN

El juego es la forma más elevada de investigación.

Albert Einstein

La mejor forma de aprender es jugando. Es algo que he tenido siempre muy claro. Desde pequeño, me ha resultado más sencillo estar horas disfrutando de títulos como *Minecraft* o *Satisfactory*, donde es fácil aprender sobre circuitos electrónicos, puertas lógicas y diseño de sistemas complejos, que memorizar el temario de cualquier asignatura, por eso he intentado facilitar mis estudios utilizando la gamificación.

Para transformar una idea compleja en varias más simples y volver a unirlas es necesario crear un hilo conductor. Aunque depende de la percepción de cada persona, creo que la forma más adecuada es mediante el juego. Este debió ser el motivo que llevó a John D. Sterman a crear los STRATEGEM, un sistema para enseñar sistemas económicos complejos a estudiantes del MIT.

Otro de los ejemplos que se puede nombrar es el del videojuego *Portal*, que, aunque toca ideas más sencillas, ayuda al entendimiento de nociones como la gravedad, la aceleración o la conservación de la energía, conceptos de física mostrados dentro de un ámbito de la resolución de puzzles. Aunque si se desea explicar temas algo más complejos se debería dirigir más hacia *Kerbal Space Program*, mostrando conceptos como la física de cohetes, la mecánica orbital y temas de ingeniería aeroespacial.

Países como Suecia ya llevan tiempo utilizando videojuegos como herramienta educativa, debido a la facilidad que poseen los estudiantes para asimilar conceptos, haciendo el aprendizaje de estos más interactivo fomentando en el proceso la habilidad para la resolución de problemas [1].

Debido a todo lo expuesto anteriormente es por lo que nace este proyecto. La finalidad es observar las decisiones que se toman con el objetivo de mantener en marcha la producción, camuflado tras un juego de móvil.

El propósito del trabajo es el diseño e implementación de una interfaz sencilla que ayude a la recopilación de datos. Para ello será necesario la creación de un sistema de registro de usuario, ya sea de forma automática con la cuenta de *Google* o rellenando un formulario de nuevo usuario.

La aplicación mostrará una pantalla de bienvenida donde el usuario pueda iniciar sesión y registrarse, una pantalla para la recopilación de datos, así como una pantalla de juego, en el que se mostrará visualmente el funcionamiento del algoritmo de Kondratiev. También existirá una pantalla de ajustes donde realizar las configuraciones de la aplicación.

Por último, se configurará la aplicación para poder permitir la comunicación con la base de datos. Estos datos son requeridos por el Departamento de Ingeniería de Sistemas y Automática, para el estudio de los parámetros obtenidos del juego con respecto a los distintos datos recopilados.

Para poner todo lo anterior en práctica, primero se configurará el emulador *Android*, donde se probarán todos los diseños implementados en la aplicación, por lo que se deberá realizar un pequeño estudio, destacando las resoluciones más usadas por los últimos dispositivos lanzados al mercado.

1.1 Motivación y Objetivos

Los objetivos para realizar en este trabajo son los siguientes:

1. Estudio de la aplicación original y funcionamiento del ciclo de Kondratiev.
2. El diseño la interfaz gráfica para la aplicación, teniendo en cuenta su facilidad de uso y claridad visual. Se debe poder distinguir a simple vista posibles datos mostrados para la retroalimentación de las acciones realizadas por el usuario, la ventana que se usará como input de programa.
3. Implementación de la interfaz gráfica.
4. Configuración de la base de datos.
5. Pruebas de uso de la aplicación observando que el funcionamiento de esta sea el esperado.

1.2 Estructura de la Memoria

La memoria se ha estructurado en los siguientes apartados, donde se explica brevemente de qué trata cada uno.

1. *Introducción.* Contexto de este proyecto y se explican los objetivos a alcanzar.
2. *El Ciclo de Kondratiev.* Explicación del contexto histórico.
3. *Aplicaciones Desarrolladas.* Se muestran las distintas versiones jugables que existen del algoritmo.
4. *Desarrollo Conceptual.* Elección de la temática de la aplicación, recopilación de datos, selección y configuración del dispositivo de simulación.
5. *Base de datos.* Elección y configuración de la base de datos para el guardado de la información de los usuarios.
6. *Funcionalidad de la Aplicación.* Desarrollo y explicación del código necesario para el funcionamiento de la aplicación, así como de las distintas pantallas en las que está dividida.
7. *Pruebas Realizadas.* Se detallan las pruebas realizadas para asegurar el correcto funcionamiento de la aplicación y de su base de datos.
8. *Conclusión.* Recapitulación y crítica al proyecto realizado, señalando aspectos a optimizar con posibles mejoras para futuras versiones.
9. *Referencias.* Recoge las referencias bibliográficas y los recursos utilizados.

2 EL CICLO DE KONDRATIEV

La teoría de los ciclos de onda larga económica o ciclo de Kondratiev, desarrollada por el economista ruso Nikolai Kondratiev en la década de 1920, expone que la economía global está determinada por ciclos de expansión y depresión, los cuales tienen una duración media de 50 años. La teoría gira en torno al autoordenamiento del capital y la dependencia de los sectores productores de capital de la economía, en conjunto, de su propia producción. De esta teoría surge el Modelo Nacional, desarrollado por el MIT como modelo educativo para relacionar una variedad de factores económicos, tales como la inversión de capital, el empleo, la política monetaria y fiscal, la inflación, la productividad y la innovación. La importancia de este modelo reside en la gran cantidad de detalles que representa, pero esto dificulta su explicación. Por este motivo existe el artículo de John D. Sterman, una explicación sencilla para el modelo de onda larga económica [2].

Han existido otros modelos que han intentado explicar los movimientos de la economía, como el de Tinbergen en 1981, el cual planteaba que los ciclos estaban marcados por los tiempos de guerras y factores monetarios como los descubrimientos de yacimientos de oro, o la de Schumpeter en 1939, utilizada durante mucho debido a ser la más completa, la cual está centrada en torno a la innovación tecnológica.

2.1 Hipótesis Dinámica

Para facilitar el entendimiento de este sistema, a continuación, se explicará el modelo de onda larga económica utilizando objetos cotidianos.

El incremento de la demanda sobre un determinado artículo, en ese caso un martillo, supone un aumento en su fabricación. Sin embargo, este crecimiento no es proporcional, dado que el propio martillo es necesario para su fabricación ya que, para poder fijar la cabeza del martillo a su mango, es necesario contar con otro martillo con el cual aplicar la fuerza necesaria.

Además, se debe tener en cuenta la vida útil del martillo. Con su uso y desgaste, los martillos usados en la industria eventualmente se deteriorarán y deberán ser reemplazados. Como resultado, un aumento de la demanda de 100 martillos suponga una producción superior a la cantidad solicitada. Esto genera un bucle de retroalimentación difícil de prever.

También debe considerarse que, aunque los aumentos de demanda pueden ser instantáneos, la producción siempre va por detrás. Este retraso de producción provoca un desequilibrio y la necesidad de un ajuste transitorio de la onda larga, pudiendo provocar plazos de entrega largos y escasez de equipos especializados. En el caso de los demandantes, pueden ocurrir fenómenos de acaparamiento de recursos, como se pudo ver durante la pandemia mundial de COVID-19.

Para hacer más comprensible y poder comunicar la esencia del modelo de onda larga de forma convincente, se desarrolló un modelo el cual solo requería de 30 ecuaciones para su funcionamiento, en comparación con las 1600 ecuaciones necesarias para hacer funcionar el Modelo Nacional. Pero incluso este modelo es demasiado complejo para las personas sin un entendimiento avanzado en modelos matemáticos.

2.2 Juego de simulación

En 1985, el Instituto Internacional de Análisis de Sistemas Aplicados de Laxenburg, el Centro de Políticas de Recursos de la Universidad Dartmouth y el Grupo de Dinámica de Sistemas del MIT, colaboraron en el desarrollo de una serie de juegos de simulación STRATEGEM (Strategic Games for Educating Managers).

Los jugadores deberían gestionar el sector de producción, equilibrando la oferta y la demanda, intentando evitar grades fluctuaciones en la producción. Aunque las decisiones tomadas en cuanto a inversión y producción puedan verse como racionales, pueden producir comportamientos irracionales, como la sobreexpansión y la contradicción en decisiones económicas [3].

Este programa ya ha sido probado con anterioridad con estudiantes y economistas, en la mayoría de casos pequeñas variaciones en la demanda provocaron ciclos de gran amplitud de entre 30 y 70 años, demostrando que las decisiones de inversión pueden llevar a la inestabilidad económica.

3 APLICACIONES DESARROLLADAS

El algoritmo original se creó para ser jugado a mano, usando una plantilla para la recopilación de datos y posteriormente calcular la puntuación obtenida. Era un modelo fundamentalmente educativo, desarrollado como un sistema para entrenar a los estudiantes del Máster en Administración de Empresas del MIT. Tras esto se han desarrollado varias aplicaciones basadas en este algoritmo original.

3.1 Versión en Basic

Como se ha comentado en el Capítulo 2, existe una versión desarrollada en *Basic* por John D. Sterman. Dada la naturaleza del lenguaje de programación, esta versión cuenta con problemas en cuanto al redondeo de algunos valores. Esto va a complicar la adaptación del algoritmo original, habiendo cambios significativos tras varias iteraciones de este. Por eso, aunque funcional y con el objetivo de la recopilación de datos, desde la ETSI se decidió crear una nueva versión del programa en *Matlab*.

Se adjunta la siguiente tabla con los valores resultantes usando el programa original. Esta será de utilidad a la hora de comprobar y comparar estos con los resultados que se obtengan utilizando el con el nuevo algoritmo desarollado.

Año	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	450	450	50	500	500	50	500	100	450	50	50
2	450	450	50	500	500	50	500	100	450	50	50
4	500	500	50	550	500	50	500	91	450	50	60
6	500	550	60	610	500	50	500	82	450	50	70
8	500	600	80	680	500	50	500	74	440	60	80
10	500	660	100	760	510	50	510	67	440	70	90
12	500	720	120	840	530	50	530	63	450	80	140
14	500	770	180	950	560	60	560	59	450	110	140
16	500	820	210	1030	610	60	610	59	480	130	140
18	500	840	220	1060	680	70	680	64	540	140	150
20	500	800	230	1030	750	80	750	73	580	170	140
22	500	720	200	920	840	80	840	91	660	180	130
24	500	560	150	710	940	90	710	100	560	150	120
26	500	500	120	620	1000	100	620	100	500	120	0
28	500	500	0	500	1020	100	500	100	500	0	0
30	500	500	0	500	920	90	500	100	500	0	0

32	500	500	0	500	830	80	500	100	500	0	0
34	500	500	0	500	750	80	500	100	500	0	30
36	500	500	30	530	670	70	530	100	500	30	20
38	500	500	20	520	630	60	520	100	500	20	40
40	500	500	40	540	590	60	540	100	500	40	50
42	500	500	50	550	570	60	550	100	500	50	50
44	500	500	50	550	560	60	550	100	500	50	60
46	500	500	60	560	550	60	550	98	490	60	50
48	500	510	50	560	550	60	550	98	500	50	60
50	500	510	60	570	540	50	540	95	480	60	60
52	500	530	60	590	550	60	550	93	490	60	80
54	500	540	80	620	550	60	550	89	480	70	80
56	500	560	90	650	560	60	560	86	480	80	90
58	500	580	100	680	580	60	580	85	490	90	110
60	500	590	120	710	610	60	610	86	510	100	90
62	500	580	110	690	650	70	650	94	550	100	70
64	500	530	80	610	680	70	610	100	530	80	60
66	500	500	60	560	690	70	560	100	500	60	0
68	500	500	0	500	680	70	500	100	500	0	0
70	500	500	0	500	610	60	500	100	500	0	60

Tabla 3.1 Resultados del Algoritmo de Sterman

Cada columna representa uno de los valores que son calculados dentro del algoritmo, siendo la columna 10 los valores introducidos por el usuario.

0. Nuevos pedidos del sector de bienes

1. Cartera de pedidos no satisfechos del sector de bienes

2. Cartera de pedidos no satisfechos del sector de capital

3. Producción deseada

4. Capacidad de producción

5. Depreciación del stock de capital

6. Producción realizada

7. Fracción de la demanda satisfecha

8. Envíos al sector de bienes

9. Envíos al sector de capital

10. Nuevos pedidos del sector de capital

Hay que añadir que la primera iteración se realiza de forma automática para marcar el punto de inicio del programa.

3.2 Versión en Matlab

Para simplificar la visión económica de la aplicación original, desde el Departamento de Ingeniería de Sistemas y Automática de la ETSI, se decidió modificar la temática hacia temas relacionados con la educación.

La interfaz muestra principalmente 4 datos, Figura 3.1, la cantidad de profesores, el número de estudiantes que se convertirán en profesores, los que se graduarán como ingenieros para trabajar en la industria y el total de estudiantes de la escuela. El dato de los ingenieros demandados por la industria, aunque presente al comienzo de la partida, se ocultará pasados unos turnos eliminando uno de los datos principales de los que disponemos para planificar la cantidad de profesores que deberemos formar.

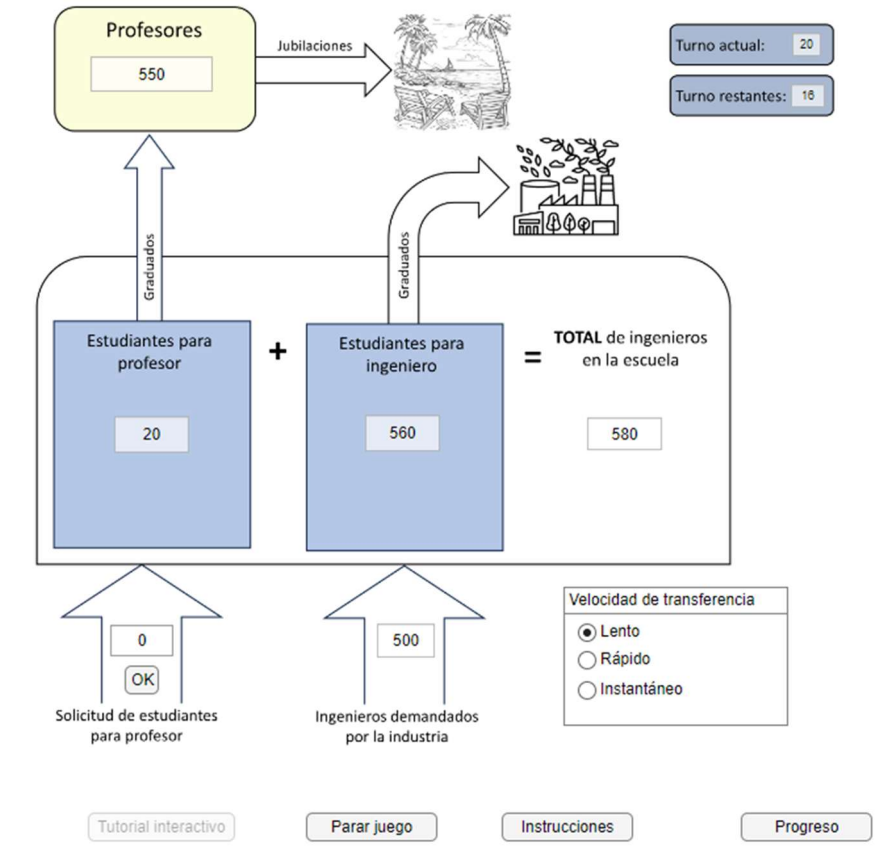

Figura 3.1 Interfaz de la Aplicación Original

Para medir la desviación respecto al valor base del número de estudiantes que se convertirán en profesores, valor a introducir para que la desviación sea 0, se calcula la diferencia entre el número de profesores y el de estudiantes. Esta desviación es la usada para calcular el coste de las operaciones, que actuará como sistema de puntuación. La desviación está graficada en la Figura 3.2 en color rojo.

Aunque *Matlab* es un sistema de cómputo numérico ampliamente usado en la industria, su uso fuera de esta no está tan extendido, esto limita la posibilidad de la recopilación de datos e incluso puede llegar a limitar su uso en aulas, dada la necesidad del uso de ordenadores con esta aplicación instalada para su ejecución. De estas limitaciones surge la idea de la creación de una aplicación móvil que funcione con el algoritmo John D. Sterman.

Figura 3.2 Datos Obtenidos Durante los Primeros 20 Turnos

4 DESARROLLO CONCEPTUAL

Existen una gran cantidad de lenguajes de programación especializados en el desarrollo de aplicaciones móviles. En *Android* de forma nativa se encuentra *Kotlin*, desarrollado por *Google*, es un lenguaje de programación expresivo y conciso que permite reducir los errores de código comunes y se integra fácilmente en las apps existentes [4].

Por la parte de *iOS* tenemos *Swift*, propiedad de *Apple*, lo cual limita el alcance de los dispositivos compatibles a los de la propia compañía.

Debido a lo expuesto anteriormente y dada la necesidad de utilizar la aplicación a múltiples sistemas operativos el lenguaje de programación elegido para el desarrollo de la interfaz es *Dart*, del cual se usará su SDK desarrollado por *Google*, *Flutter*. Se ha elegido este entorno dado ser gratuito, de código abierto y ofrece ventajas como el desarrollo de aplicaciones multiplataforma. Aunque principalmente se usa para la programación de aplicaciones tanto en *Android* como en *IOS*, los programas también pueden ser exportados para su uso en *web*, en *Linux*, incluso en *Windows* y *MacOS*, lo que hará posible el uso de la aplicación en prácticamente cualquier entorno de trabajo.

Al estar desarrollado por *Google*, este nos ofrece una estabilidad y una certeza en que el proyecto podrá seguir siendo utilizado en el futuro, teniendo un soporte y una vida útil alargada en el tiempo.

4.1 Elección de la Temática

La interfaz principal de la aplicación tendrá el aspecto de juego, por lo tanto, será necesario mostrar en pantalla animaciones que representen visualmente lo que ocurre en el código.

Debido a la familiaridad que por lo general tenemos para entender el funcionamiento de la guerra, gracias a todas las películas y series que hemos consumido a lo largo de los años, se ha decidido esta sea la temática del juego. Además, dado que la aplicación está siendo desarrollada como un Trabajo de fin de Grado de la Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla, se ha optado por que el nombre de la aplicación refleje tanto el nombre de la escuela como la temática elegida para la aplicación. Por eso el título elegido ha sido *ETSI Royale*.

4.2 Tipos de Dispositivos

Las pantallas de los dispositivos móviles con los que estamos acostumbrados a interaccionar a diario cuentan con dos resoluciones, que, aunque relacionadas entre sí, están diferenciadas. Estas son la resolución nativa, medida en píxeles, *px*, y la resolución relativa, medida en pixeles lógicos o puntos, *pt*. Mientras que la resolución nativa representa la medida real de píxeles en los ejes coordenados, la resolución relativa no tiene en cuenta la densidad de píxeles por pulgada, *ppi*, de la pantalla [5].

Por esto la resolución relativa de pantalla es la unidad estándar de trabajo en el desarrollo de aplicaciones móviles. Su uso facilita la conversión y la consistencia de la interfaz para los múltiples dispositivos en los que se puede usar ya que es independiente de la de la densidad de píxeles del dispositivo usado. Por eso es importante elegir una resolución relativa de trabajo parecida a las más utilizadas por los dispositivos móviles.

La conversión entre resolución nativa y resolución relativa puede calcularse con la siguiente fórmula teniendo en cuenta que la densidad de píxeles de referencia para *Android* son 160 *ppi*. Los resultados de esta fórmula se redondean al valor entero más cercano.

$$pt = px * (160 / ppi) \quad \textbf{(4-1)}$$

A continuación, se detallan las características de los últimos dispositivos lanzados por *Google* [6], *Samsung* [7], *Apple* [8] y *Xiaomi* [9], los cuales se han tomado como referencia para el análisis de las resoluciones de dispositivos.

Dispositivo	Resolución Nativa	Píxeles por Pulgada	Resolución Relativa
Google Pixel 9	1080 x 2424 px	422 ppi	409 x 767 pt
Google Pixel 9 Pro	1280 x 2856 px	495 ppi	441 x 978 pt
Google Pixel 9 Pro XL	1344 x 2992 px	486 ppi	441 x 978 pt
Samsung Galaxy s25	1080 x 2340 px	416 ppi	416 x 768 pt
Samsung Galaxy s25+	1440 x 3120 px	509 ppi	453 x 975 pt
Samsung Galaxy s25 Ultra	1440 x 3120 px	501 ppi	458 x 976 pt
iPhone 16	1179 x 2556 px	460 ppi	375 x 812 pt
iPhone 16 Pro	1206 x 2622 px	460 ppi	414 x 736 pt
iPhone 16 Pro Max	1320 x 2868 px	460 ppi	453 x 991 pt
Xiaomi 14	1200 x 2670 px	460 ppi	414 x 902 pt
Xiaomi 14 Ultra	1440 x 3200 px	522 ppi	441 x 973 pt
Xiaomi 14T Pro	1220 x 2712 px	446 ppi	439 x 991 pt

Tabla 4.1 Resolución de los Dispositivos Seleccionados

Como se puede observar, la resolución relativa en todos estos dispositivos se encuentra en un rango de resoluciones muy acotado. Por ello se va a tomar la resolución del dispositivo más cercano a la media como referencia para el diseño de la interfaz de la aplicación, en el caso de esta tabla el dispositivo es el *Xiaomi 14T Pro*, con una resolución relativa de 439 x 991 pt. Sin embargo, para facilitar el diseño de la interfaz se ha redondeado la resolución relativa de este dispositivo a 440 x 990 pt.

4.3 Android Studio

Con la información que hemos obtenido en el apartado anterior, podemos empezar a configurar el sistema de emulación del dispositivo *Android*. Este sistema de emulación nos permitirá ver en tiempo real los cambios de código que realicemos, verificando así su correcta funcionalidad.

Dentro de *Android Studio*, en el apartado de *SDK Manager*, podremos configurar un nuevo dispositivo que al igual que el real seleccionado, el emulador contará con una pantalla de 6,67 pulgadas, Figura 4.1. Para el apartado del sistema operativo, *Android Studio* nos proporciona varias opciones a elegir, pero para asegurar el funcionamiento de la aplicación con futuras versiones se ha optado por tomar la versión más reciente en producción de *Android 15*, la *api* 35 denominada *VanillaIceCream,*

Figura **4.2**.

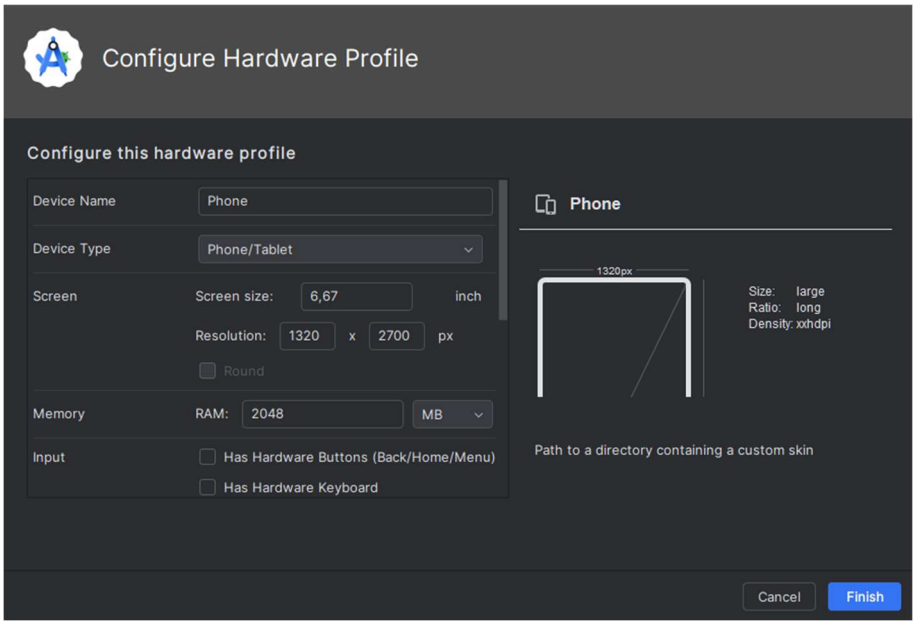

Figura 4.1 Configuración del Dispositivo de Emulación

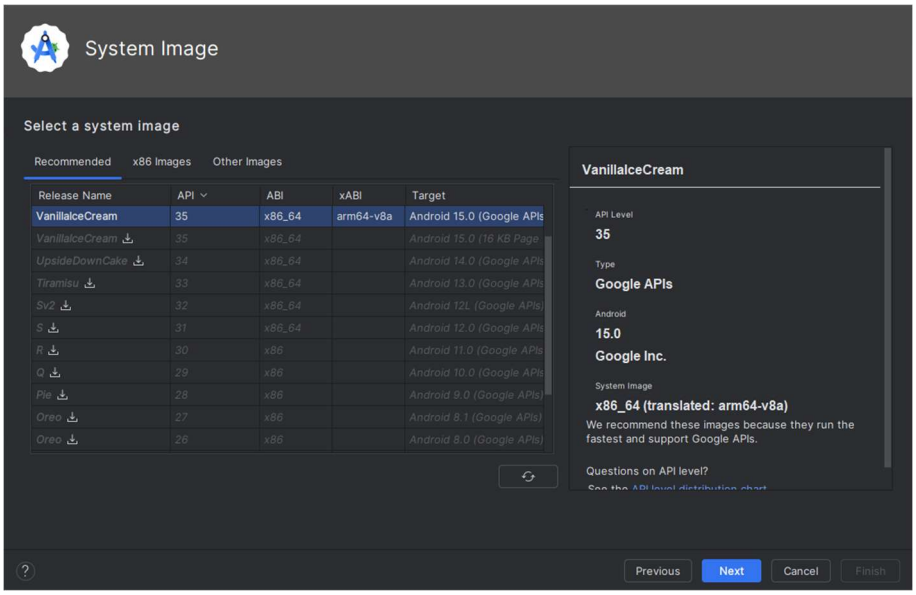

Figura 4.2 Elección de la Versión del Sistema Operativo

5 BASE DE DATOS

La aplicación necesita un lugar donde poder almacenar los datos de todos los usuarios, así como los resultados que estos obtengan en el juego, de ahí el requisito de tener una base de datos. *Firebase*, desarrollada por *Google*, cuenta con muchas herramientas que serán útiles en el desarrollo de la infraestructura de este proyecto, como puede ser la propia base de datos, *Firestore*, o la de autentificación de usuarios, *Authentication*.

5.1 Configuración del Proyecto

Para que *Firebase* pueda comunicarse con la aplicación que estamos desarrollando, debemos agregarla a su *web*. Como se muestra en la Figura 5.1, han habilitado varias opciones para realizar este enlace, tanto para dispositivos que usen *Android*, *iOS* o *web* y para entornos de desarrollo como *Unity* o *Flutter*.

Al elegir esta última opción, nos podemos asegurar que *Firebase* podrá comunicarse con nuestro código independientemente del dispositivo y sistema operativo desde el cual se esté ejecutando la aplicación.

Figura 5.1 Agregar *Firebase* a la Aplicación

Elegida esta opción, *Firebase* nos proporciona una serie de instrucciones a seguir para finalizar la integración. Siguiendo los pasos descritos en la Figura 5.2,deberemos descargar e instalar el cliente y configurarlo añadiendo el usuario y el proyecto en el cual queremos trabajar. Tras esto nos pedirá que ejecutemos una serie de comandos en el directorio raíz de nuestro proyecto y añadir una línea de código que se encargará de inicializar la comunicación con *Firebase* cuando se ejecute la aplicación, visible en el Código 6.1.

Una vez enlazado el proyecto con *Firebase*, podríamos empezar a configurar los distintos servicios que necesitamos para el correcto funcionamiento de la aplicación.

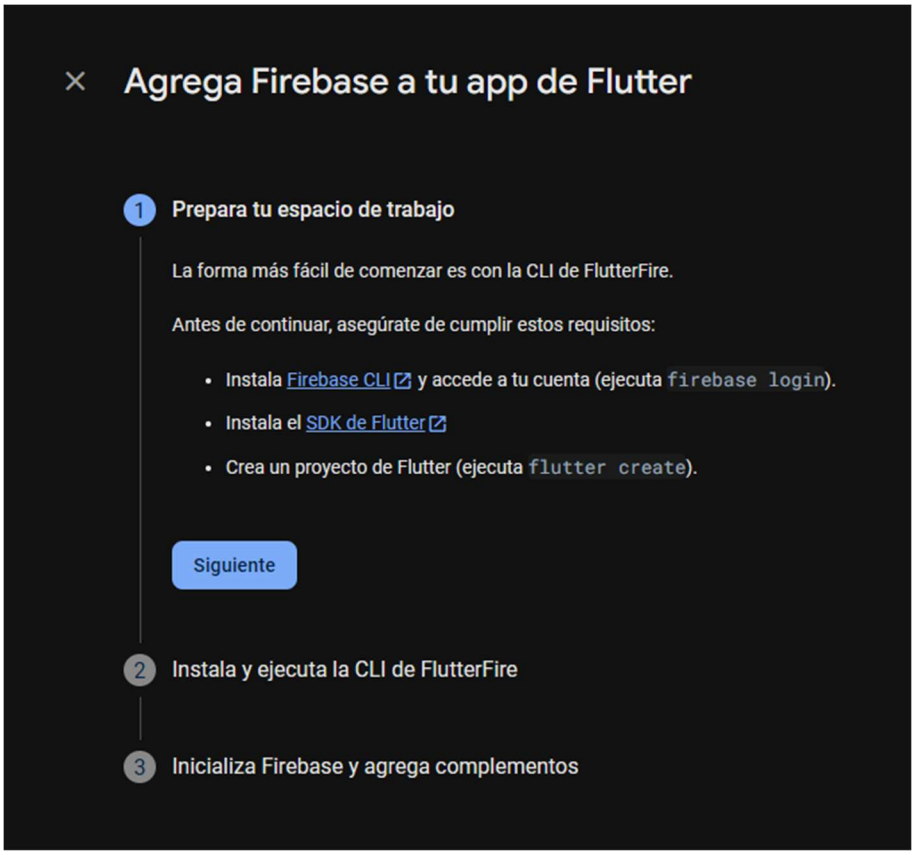

Figura 5.2 Descarga del cliente de *Firebase*

5.2 Authentication

Las entradas en la base de datos deben estar identificadas por una secuencia alfanumérica única. Generar esta desde la aplicación es una tarea complicada, por lo que se va a delegarla a *Authentication*.

Como se puede ver en la Figura 5.3, esto también nos permite ofrecer varias opciones para la identificación de usuarios, de forma nativa usando correo electrónico/contraseña o un número de teléfono, o con el uso de proveedores adicionales usando las cuentas ya existentes de los usuarios en *Google*, *Microsoft*, *GitHub* y varias opciones más. En el caso de esta aplicación y para facilitar el acceso de van a habilitar 2 proveedores, correo electrónico/contraseña, Figura 5.4 y *Google,* Figura 5.5.

Figura 5.3 Modos de Registro

Si bien proveedores de acceso como *Google* pueden proporcionar datos de usuario, se ha decidido que los datos recopilados de los usuarios deberán ser proporcionados por estos, en vez de utilizar los proporcionados por los

proveedores de registro.

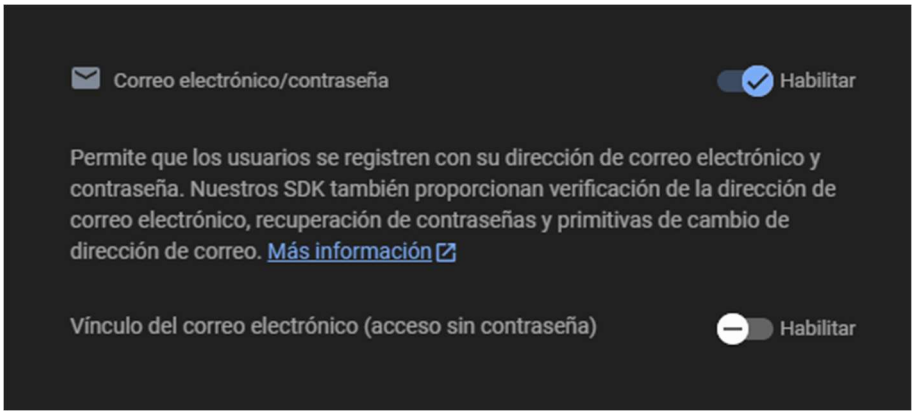

Figura 5.4 Habilitar Registro con Correo Electrónico y Contraseña

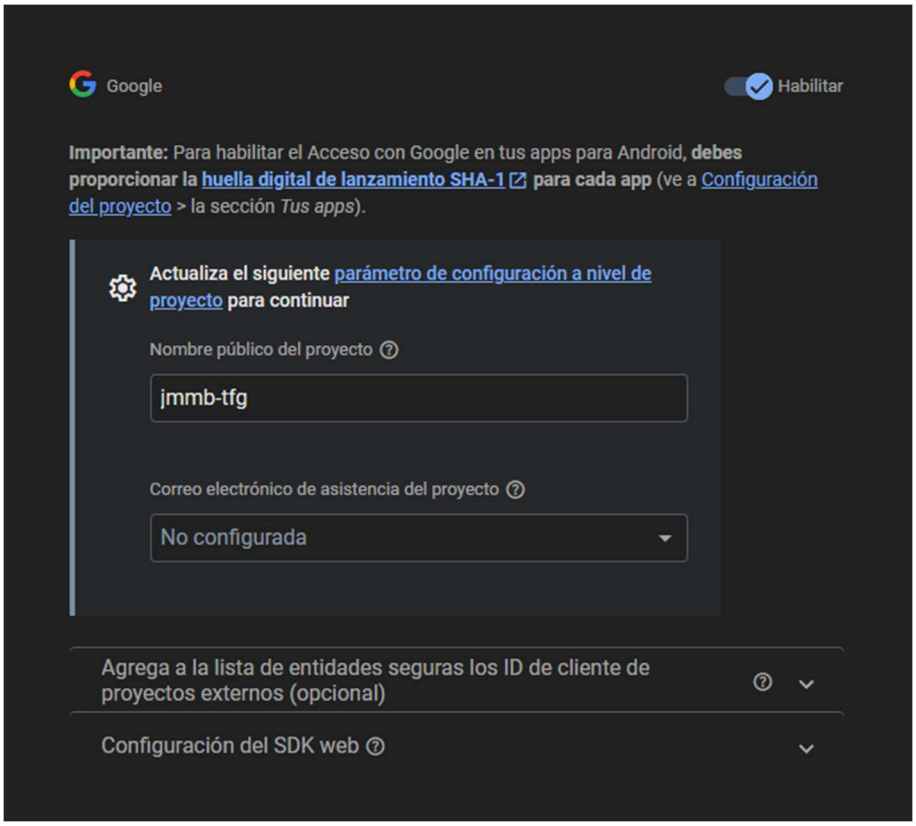

Figura 5.5 Habilitar Registro con *Google*

5.3 Firestore Database

5.3.1 Ubicación

Firebase no permite cambiar la ubicación del servidor donde se va a alojar la base de datos una vez creada. Debido a esto se ha elegido *eu3*, dado que se encuentra en Europa, reduciendo la latencia para las conexiones desde España y a la vez maximizar la disponibilidad para otros países ya que como se puede observar en la Figura 5.6, el servidor es multirregional [10].

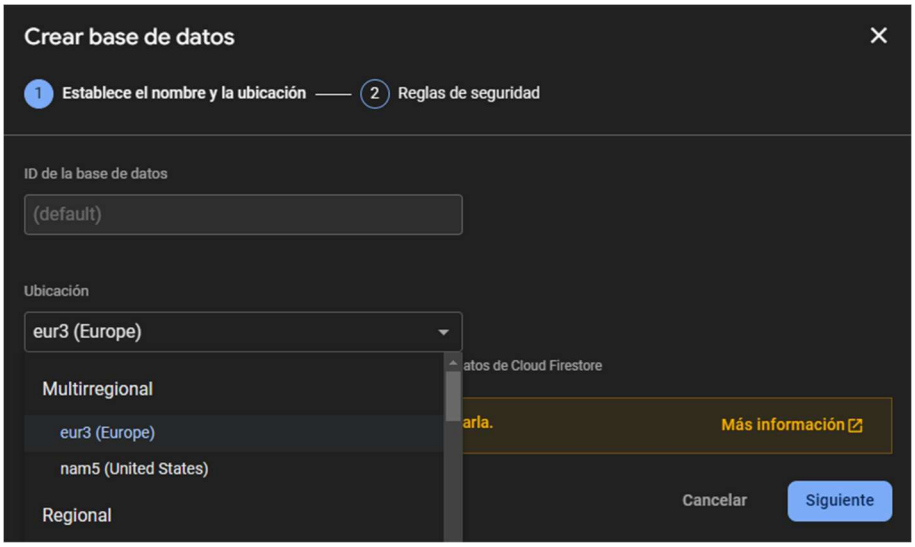

Figura 5.6 Ubicación de la Base de Datos

5.3.2 Seguridad y Protección

Aunque *Firestore* ya es un entorno seguro donde trabajar, la entrada y salida de los datos está controlada por las reglas de acceso que especifiquemos. Por defecto, la base de datos se ejecuta bloqueando todas las operaciones de lectura y escritura, pero eso dejaría inutilizada la base de datos por completo.

Los usuarios de la aplicación deben poder leer y escribir datos dentro de su propio perfil, bloqueando el acceso de estos a los datos de otros usuarios. Los nuevos usuarios no registrado deben tener la posibilidad de crear nuevos perfiles. Para que esto sea posible debemos modificar las reglas de seguridad visibles en la Figura 5.7 [11].

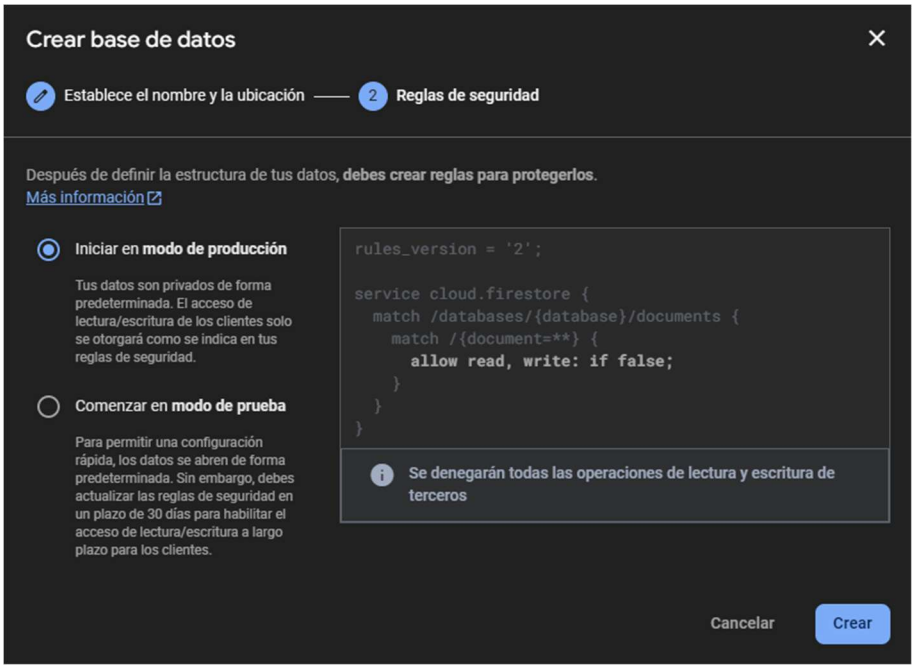

Figura 5.7 Configuración de las Reglas de Seguridad

Utilizando el siguiente código, los usuarios podrán crear, actualizar y leer sus entradas en la base de datos siempre que hayan iniciado sesión y la identificación del usuario, *uid*, sea igual a la que se les asignó al registrarse.

Código 5.1 Reglas de Seguridad de *Firestore*

```
service cloud.firestore {
  match /databases/{database}/documents {
    match /usuarios/{userID} {
      allow create, read, update, read: if isSignedIn() &&
        request.auth.uid == userID;

      match /{subCollection}/{document=} {
        allow create, read, update: if isSignedIn() &&
          request.auth.uid == userID;
      }
    }
  }
  function isSignedIn(){
    return request.auth != null
  }
}
```

5.3.3 Guardado de Datos

Una vez recopilado los datos del usuario, es necesario enviarlos a *Firestore* para su almacenamiento. Esta tarea se realiza con el enlace ya establecido.

Primero se debe seleccionar la colección dentro de la cual se quieren almacenar los datos, en ese caso la colección *usuarios*. Tras esto es necesario acceder a los servicios de *Authentication* para obtener el *uid* del usuario. Se debe comprobar que este existe, ya que de no ser así la función devolvería *null* provocando un error en el sistema. Para finalizar, los datos son enviados a *Firestore*.

Código 5.2 Creación de la Entrada del Usuario en *Firestore*

```
Future<void> userSetup(
  String genero,
  int edad,
  String pais,
  String nivelEducativo,
  String tiempoJuego,
) async {
  if (FirebaseAuth.instance.currentUser != null) {
    var usuarios = FirebaseFirestore.instance.collection('usuarios');
    FirebaseAuth auth = FirebaseAuth.instance;
    String? userID = auth.currentUser!.uid;
    String? email = auth.currentUser!.email;

    usuarios.doc(userID).set({
      'email': email,
      'genero': genero,
      'edad': edad,
      'pais': pais,
      'nivelEducativo': nivelEducativo,
      'tiempoJuego': tiempoJuego,
      'partidas': 0,
    });
    usuarios.doc(userID).collection('partidas');
  }
}
```

Como se desea guardar los datos de todas las partidas jugadas por los usuarios, se ha creado colección dentro del perfil del usuario llamada *partidas*. Dentro de esta colección se irán creando sucesivos documentos donde se alojarán los datos de una partida. Para crear esta colección y que *Firestore* no la elimine, es necesario añadir algún elemento dentro, debido a esto existe la partida 0. Con esta jerarquía se mejora el orden y el entendimiento de la estructura establecida en la base de datos, y se permite recuperar tanto el total de las partidas jugadas por un usuario, como el conjunto de partidas de todos los usuarios.

Código 5.3 Envio de los Datos de Partidas a *Firestore*

```
Future<void> userGameData(
  int score,
  List results,
  int production,
  int demandSatisfied,
) async {
  if (FirebaseAuth.instance.currentUser != null) {
    var usuarios = FirebaseFirestore.instance.collection('usuarios');

    FirebaseAuth auth = FirebaseAuth.instance;
    String? userID = auth.currentUser!.uid;
    DocumentSnapshot userDoc = await usuarios.doc(userID).get();
    int partidasJugadas = userDoc['partidasJugadas'] ?? 0;
    partidasJugadas += 1;

    await usuarios.doc(userID).update({'partidasJugadas':
      partidasJugadas});

    usuarios
        .doc(userID)
        .collection('partidas')
        .doc(partidasJugadas.toString())
        .set({
          'puntuación': score,
          'resultados 0': results[0],
          'resultados 1': results[1],
          'resultados 2': results[2],
          'resultados 3': results[3],
          'resultados 4': results[4],
          'resultados 5': results[5],
          'resultados 6': results[6],
          'resultados 7': results[7],
          'resultados 8': results[8],
          'resultados 9': results[9],
          'resultados 10': results[10],
          'production': production,
          'demandSatisfied': demandSatisfied,
        });
  }
}
```

6 FUNCIONALIDAD DE LA APLICACIÓN

La aplicación se encuentra dividida en pantallas, las cuales tendrán la función de guiar al usuario a la realización de una acción, ya sea completar su registro o modificar la configuración de la aplicación. Estas a su vez están compuestas por widgets, que son los bloques fundamentales de construcción en *Flutter* y se encargan de crear elementos de la interfaz de usuario tanto de forma visual como interactiva [12].

6.1 Función Principal

Para empezar a hablar de la aplicación debemos explicar la funcionalidad de *main.dart* o archivo principal. Este archivo cuenta con la función *void main*, por lo que es el primero en ejecutarse. Este es el encargado de inicializar todos los paquetes, así como la comunicación con la base de datos tal y como se puede observar a continuación.

Código 6.1 Declaración de la Función Principal

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(options:
    DefaultFirebaseOptions.currentPlatform);
  await EasyLocalization.ensureInitialized();
  SystemChrome.setSystemUIOverlayStyle(
    SystemUiOverlayStyle(statusBarIconBrightness: Brightness.light),
  );

  runApp(
    EasyLocalization(
      supportedLocales: supportedLocales,
      path: 'assets/translations',
      fallbackLocale: spanish,
      child: MultiProvider(
        providers: [
          ChangeNotifierProvider(create: (_) => GoogleSignInProvider()),
        ],
        child: ETSIRoyale(),
      ),
    ),
  );
}
```

Tras esto es necesario crear una clase que será llamada por esta función y que se encargará tanto de cargar las preferencias del usuario modificadas en la pantalla de ajustes, como de inicializar la pantalla principal, o en el caso que el usuario tenga la sesión iniciada, mostrar la pantalla de juego.

Esta clase está formada por un *StatefulWidget*, que a diferencia del *StatelessWidget*, permite que la información mostrada por pantalla se vea modificada por el tiempo. Esto es de mucha utilidad ya que nos permite no desperdiciar recursos en elementos que no vayamos a actualizar o que tengan una cuestión meramente estética, como es el caso de algunos elementos que veremos en las distintas pantallas [13].

Dentro de este *StatefulWidget*, se puede observar el widget principal de la clase *ETSIRoyale*. Este es el encargado de construir la interfaz de la aplicación. Como se puede ver, uno de sus argumentos es *initialRoute*, que redirigirá a la pantalla de bienvenida o a la pantalla de juego en función si el usuario ya había iniciado sesión antes de abrir la aplicación o no.

Código 6.2 Widget Principal

```
class ETSIRoyale extends StatefulWidget {
  const ETSIRoyale({super.key});

  @override
  State<ETSIRoyale> createState() => _ETSIRoyaleState();
}

class _ETSIRoyaleState extends State<ETSIRoyale> {
  @override
  void initState() {
    super.initState();
    getSharedPreferences();
  }

  getSharedPreferences() async {
    SharedPreferences prefs = await SharedPreferences.getInstance();
    isLoggedIn = (prefs.getBool('isLoggedIn') ?? false);
    isVibrationActive = (prefs.getBool('isVibrationActive') ?? true);
    isGameTutorialActive = (prefs.getBool('isGameTutorialActive') ??
      true);
    isRealTimeGameActive = (prefs.getBool('isRealTimeGameActive') ??
      true);
    setState(() {});
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      localizationsDelegates: [
        ...context.localizationDelegates,
        CountryLocalizations.delegate,
      ],
      supportedLocales: context.supportedLocales,
      locale: context.locale,
      isLoggedIn ? '/gameScreen' : '/',
      onGenerateRoute: RouteGenerator.generateRoute,
    );
  }
}
```

6.2 Generador de Rutas

Esta clase está dedicada a la navegación por las distintas pantallas de la aplicación. Esto se realiza llamando a la clase *Navigator*. En el Código 6.3 podemos ver tres llamadas al generador de rutas, la primera para retroceder una página, la segunda para desplazarse hasta una página específica, y la tercera para redirigir a la pantalla de bienvenida y borrar el registro de ruta, eliminando la posibilidad que el usuario vuelva hacia atrás. Para que la redirección a una página específica funcione correctamente se le deben proporcionar dos parámetros, un *context* y una ruta previamente establecida en el generador de rutas. El parámetro *context* es parte de la clase *BuildContext* se encarga de localizar el widget dentro del árbol de widgets y proporcionar información sobre su ubicación [14].

Como se puede observar en el Código 6.4, el generador de rutas está gobernado por un *switch case*, el cual siempre que el parámetro proporcionado exista, devuelve una de las pantallas de la aplicación, y en el caso de que no existir, devuelve una pantalla de error.

Código 6.3 Llamadas al Generador de Rutas

```
Navigator.pop(context);
Navigator.pushNamed(context, '/settings');
navigator.pushNamedAndRemoveUntil('/', Route<dynamic> route) => false,);
```

Código 6.4 Generador de Rutas

```
class  RouteGenerator {
  static Route<dynamic> generateRoute(RouteSettings settings) {
    switch (settings.name) {
      case '/':
        return MaterialPageRoute(builder: (_) => WellcomeScreen());
      case '/log_in':
        return MaterialPageRoute(
          builder: (_) => LogInScreen(isLoginPage: true),
        );
      case '/register':
        return MaterialPageRoute(
          builder: (_) => LogInScreen(isLoginPage: false),
        );
      case '/data_collector':
        return MaterialPageRoute(builder: (_) => DataCollectorScreen());
      case '/gameScreen':
        return MaterialPageRoute(builder: (_) => GameScreen());
      case '/settings':
        return MaterialPageRoute(builder: (_) => SettingsScreen());
      case '/results':
        final algorithm = settings.arguments as KondratievAlgorithm;
        return MaterialPageRoute(builder: (_) => ResultsScreen(
          algorithm:algorithm));
      default:
        return _errorRoute();
    }
  }
}
```

6.3 Pantallas

6.3.1 Pantalla de Bienvenida

Esta es la primera pantalla que se mostrará al usuario, donde podrá iniciar sesión y registrarse, ya sea de forma rápida usando los widgets de acceso con *Google* o mediante un correo electrónico y una contraseña.

En la parte superior de la Figura 6.1, se puede observar una ilustración acompañada de un mensaje de bienvenida a la aplicación. A continuación, un botón de color negro permite redirigir al usuario a la pantalla de inicio de sesión. Bajo el separador, podemos encontrar el proveedor de registro del cual se habló en el Apartado 5.2. Para finalizar encontramos un botón que nos redirige a la página de registro donde el usuario puede crear una cuenta usando un correo electrónico y una contraseña, Figura 6.2.

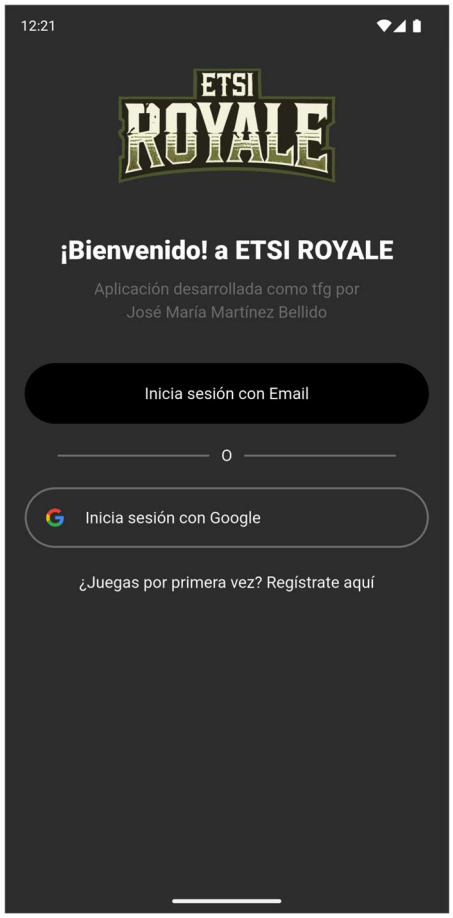

Figura 6.1 Pantalla de Bienvenida

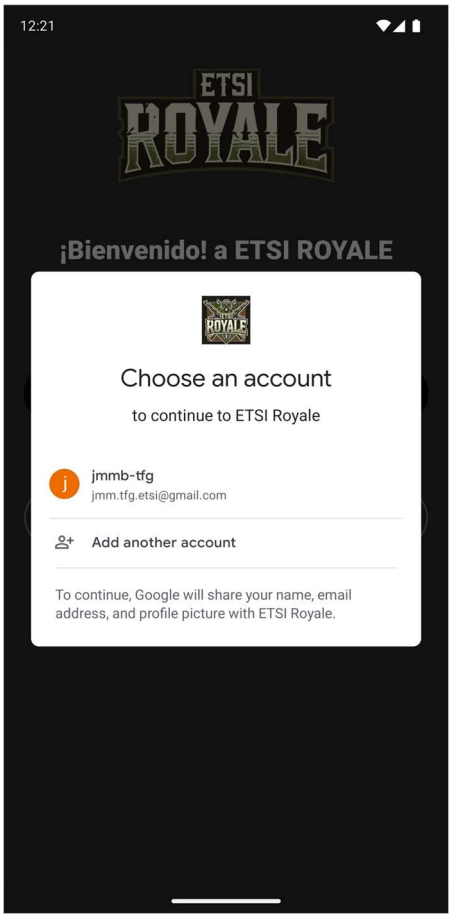

Figura 6.2 Inicio de Sesión con *Google*

Como se puede observar en el Código 6.5, se están utilizando widgets como *ButtonLogInProvidersWidget*, Código 6.20. que no forman parte del conjunto de widgets predeterminados de *Flutter*. Estos han sido creados de manera personalizada debido al diseño que hemos planteado para la aplicación. Más información sobre los widgets personalizados en el Apartado 6.3.7.

Tras hacer pruebas de la aplicación en distintos dispositivos móviles se observó que el título dada a su longitud quedaba dividido en dos filas. Por eso se añadió un condicionante que comparaba el ancho del dispositivo donde se estaba ejecutando la aplicación con el del diseño original, esto nos permite separar el texto en dos filas con una composición visualmente mejor a la que resultaría de no haber hecho esta modificación.

Los elementos *SizedBox(height: marginMedium)* actúan como separadores entre los widgets, proporcionando espacio adicional para mejorar la organización y legibilidad de la interfaz. El parámetro *height* controla la separación entre los distintos widgets.

Código 6.5 Pantalla de Bienvenida

```
class WellcomeScreen extends StatefulWidget {
  WellcomeScreen({super.key});

  @override
  _WellcomeScreenState createState() => _WellcomeScreenState();
}

class _WellcomeScreenState extends State<WellcomeScreen> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: SingleChildScrollView(
        child: Column(
          children: [
            Container(
              height: altoPantallaTotal(context),
              width: anchoPantallaTotal(context),
              color: colorbackground,
              padding: const EdgeInsets.all(marginMedium),
              child: Column(
                mainAxisAlignment: MainAxisAlignment.start,
                children: <Widget>[
                  SizedBox(height: marginMedium * 2),
                  Image(
                    image: AssetImage('assets/images/Logo.png'),
                    height: blockLogoHeight,
                  ),
                  if (anchoPantallaUsable(context) < blockLengthBig)
                    ...[
                    TitleSplitWidget(
                      titleSplit1: 'title_wellcome_split_1',
                      titleSplit2: 'title_wellcome_split_2',
                      subtitle: 'subtitle_wellcome',
                    ),
                  ] else ...[
                    TitleWidget(
                      title: 'title_wellcome',
                      subtitle: 'subtitle_wellcome',
                    ),
                  ],
                  SizedBox(height: marginMedium * 2),
                  ButtonBlackWidget(
                    title: 'text_sign_in_email_long',
                    onPressed: () {
                      Navigator.pushNamed(context, '/log_in');
                    },
                  ),
                  SizedBox(height: marginMedium),
                  SeparatorWidget(text: 'separador'),
                  SizedBox(height: marginMedium),
                  ButtonLogInProvidersWidget(
                    title: 'sign_in_google',
                    icon: 'assets/icons/google.webp',
                    iconHeight: iconHeightGoogle,
                    onPressed: () async {
                      await GoogleSignInProvider().googleLogin();
```

```
                        },
                      ),
                      SizedBox(height: marginSmall),
                      TextButton(
                        onPressed: () {
                          vibration();
                          Navigator.pushNamed(context, '/register');
                        },
                        child: Text(
                          'subtitle_wellcome_register'.tr(),
                          style: fontStyleBodyWhite,
                        ),
                      ),
                    ],
                  ),
                ),
              ],
            ),
          ),
        );
    }
}
```

6.3.2 Pantalla de Inicio de Sesión y Pantalla de Registro

Como se puede observar en la Figura 6.3 y en la Figura 6.4Ambas pantallas se ejecutan usando la misma clase. Aunque los componen los mismos widgets, se diferencian en el texto que estos muestran y en la funcionalidad dada al botón de color negro.

Para poder hacer esta diferencia y como se ha podido observar en el Código 6.6, se ha optado por definir la variable booleana *isLoginPage*, la cual será verdadera cuando el usuario esté intentando entrar en la página de inicio de sesión y falsa si desear acceder a la pantalla de registro. Esto consigue eliminar la redundancia de código entre las dos páginas facilitando también su modificación.

ButtonLoginEmailWidget se encarga de la funcionalidad y diseño de interfaz del botón de color negro, este requiere recibir el parámetro *isLoginPage*, entre otros, para poder ajustar su función al tipo de página de la cual se le ha llamado. Tras esto, se redireccionará al usuario a la pantalla de recopilación de datos. Más información sobre este widget en el Código 6.19.

Una de las funcionalidades más interesantes que se implementó en estas páginas, aunque también se puede encontrar en otras, es el cierre de forma automática del teclado virtual. Este teclado aparece en pantalla al momento de rellenar los datos necesarios para el inicio de sesión o registro y solo podía ser cerrado en el caso de que el usuario pulsara un botón presenten en el propio teclado. Para permitir el cierre de forma automática cuando se pulsa fuera del teclado, se ha envuelto la pantalla en un *GestureDetector*, el cual al accionarse activa *FocusScope.of(context).unfocus()* cerrando así el teclado.

Para los iconos presentes en esta y otras pantallas, se ha usado la librería *Lucide*, la cual consta de una gran cantidad de iconos de forma gratuita [15].

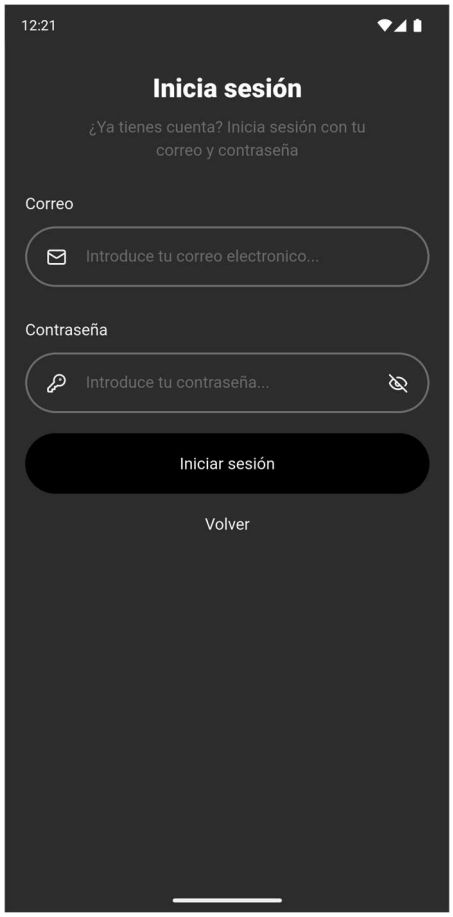

Figura 6.3 Página de Inicio de Sesión

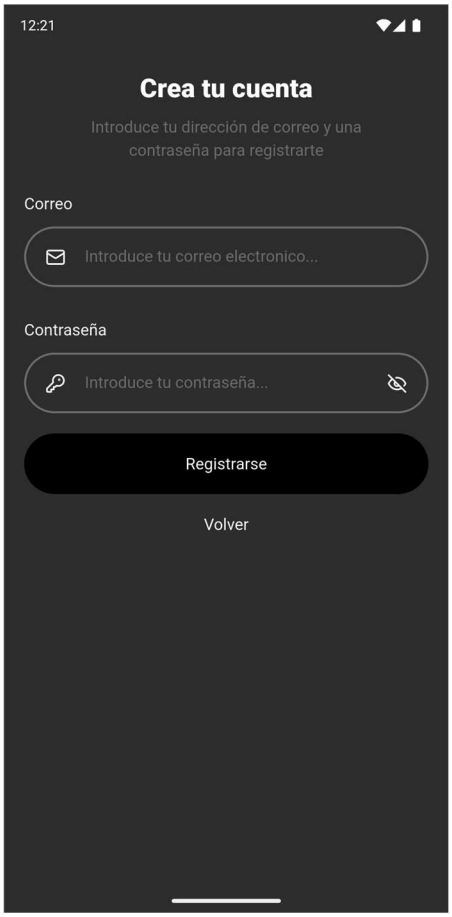

Figura 6.4 Página de Registro

Código 6.6 Pantalla de inicio de sesión y pantalla de registro

```
class _LogInScreenState extends State<LogInScreen> {
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passwordController =
    TextEditingController();

  @override
  Widget build(BuildContext context) {
    String title;
    String subtitle;
    String buttonText;

    if (widget.isLoginPage) {
      title = 'title_login';
      subtitle = 'subtitle_login';
      buttonText = 'button_login';
    } else {
      title = 'title_register';
      subtitle = 'subtitle_register';
      buttonText = 'button_register';
    }

    return Scaffold(
      body: GestureDetector(
        onTap: () {
          FocusScope.of(context).unfocus();
        },
```

```
            child: SingleChildScrollView(
              child: Container(
                height: altoPantallaTotal(context),
                width: anchoPantallaTotal(context),
                color: colorbackground,
                padding: const EdgeInsets.all(marginMedium),
                child: Column(
                  mainAxisAlignment: MainAxisAlignment.start,
                  children: <Widget>[
                    TitleWidget(title: title, subtitle: subtitle),
                    SizedBox(height: marginMedium),
                    TextInputWidget(
                      icon: LucideIcons.mail,
                      text: 'input_email',
                      isEmail: true,
                      controller: _emailController,
                    ),
                    SizedBox(height: marginMedium),
                    TextInputWidget(
                      icon: LucideIcons.keyRound,
                      text: 'input_password',
                      isPassword: true,
                      controller: _passwordController,
                    ),
                    SizedBox(height: marginMedium),
                    ButtonLoginEmailWidget(
                      isLoginPage: widget.isLoginPage,
                      title: buttonText,
                      emailController: _emailController,
                      passwordController: _passwordController,
                    ),
                    ButtonBackWidget(),
                  ],
                ),
              ),
            ),
          ),
        );
      }
    }
```

6.3.3 Pantalla de Recopilación de Datos

Como se ha comentado anteriormente, el objetivo principal de la aplicación es la recopilación de datos tanto propios de los usuarios. Estos datos serán guardados en la base de datos de *Firebase*. Más información sobre este punto en el Capítulo 5.

Debido a esta necesidad, esta página cuenta con 5 widgets encargados de esta recopilación, Figura 6.5. En primer lugar, encontramos un selector de opciones, Código 6.27, seguido de un *TextInputWidget*, Código 6.29, el cual está diseñado para bloquear cualquier entrada de texto que difiera de valores numéricos. Una vez que el usuario ha escrito su edad, el widget devuelve en forma de *controller* la variable que hemos elegido.

Tras esto encontramos 3 botones que se ejecutan utilizando la misma función, Código 6.22, cuyas interfaces visuales y funcionalidades solo difieren debido a los parámetros que se adjuntan cuando son llamados. En las Figura 6.7 y Figura 6.8 se puede observar la funcionalidad de uno de estos botones. El accionamiento de este despliega una pantalla con múltiples opciones para elegir, que tras pulsar uno de ellos, guardará la opción seleccionada a la espera de su envío a la base de datos, lo cual ocurrirá pulsado el botón de continuar y solo una vez se hayan rellenados todos los datos necesarios, Figura 6.6.

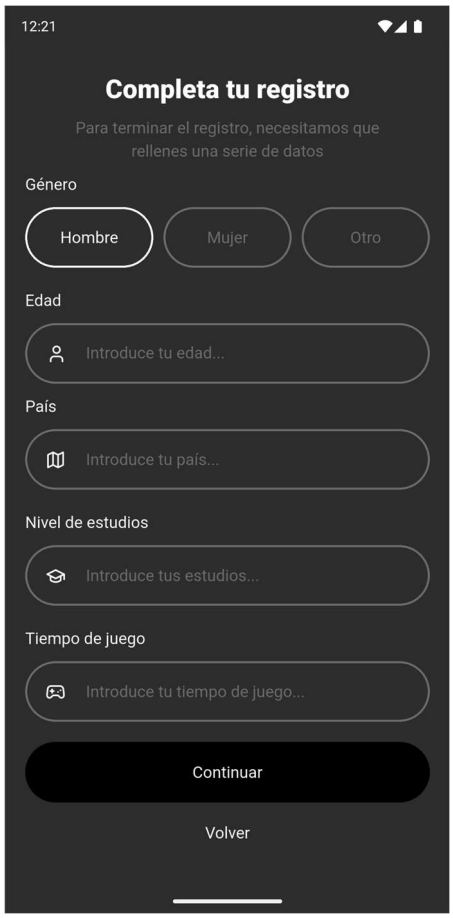

Figura 6.5 Pantalla de Recopilación de Datos

Figura 6.6 Recopilación de Datos Rellena

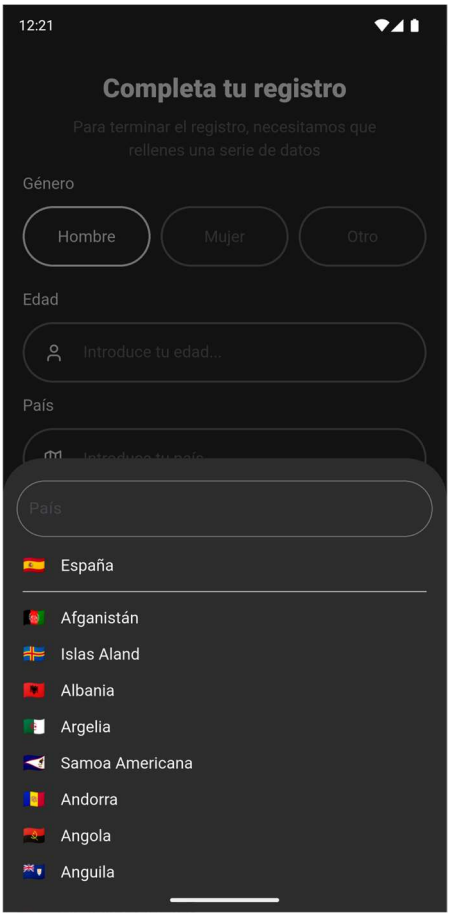

Figura 6.7 Selección de la Nacionalidad

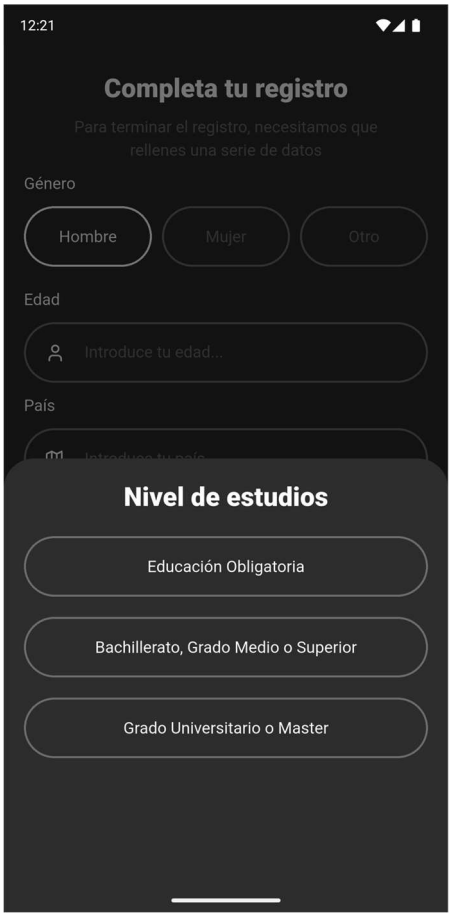

Figura 6.8 Selección del Nivel de Estudios

Código 6.7 Pantalla de Recopilación de datos

```
class DataCollectorScreen extends StatefulWidget {
  DataCollectorScreen({super.key});

  @override
  _DataCollectorScreenState createState() => _DataCollectorScreenState();
}

class _DataCollectorScreenState extends State<DataCollectorScreen> {
  var selectedGender = 'text_gender_male';
  var selectedAge = 'text_input_age';
  var selectedCountry = 'text_input_country';
  var selectedEducationLevel = 'text_input_education_level';
  var selectedPlayTime = 'text_input_play_time';
  final TextEditingController ageController = TextEditingController();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: SingleChildScrollView(
        child: GestureDetector(
          onTap: () {
            FocusManager.instance.primaryFocus?.unfocus();
          },
          child: Container(
            height: altoPantallaTotal(context),
```

```
                    width: anchoPantallaTotal(context),
                    color: colorbackground,
                    padding: const EdgeInsets.all(marginMedium),
                    child: Column(
                      mainAxisAlignment: MainAxisAlignment.start,
                      children: <Widget>[
                        TitleWidget(
                          title: 'title_data_collector',
                          subtitle: 'subtitle_data_collector',
                        ),
                        GenderSelectionWidget(
                          selectedGender: selectedGender,
                          onGenderSelected: (gender) {
                            setState(() {
                              selectedGender = gender;
                            });
                          },
                        ),
                        SizedBox(height: marginSmall),
                        TextInputWidget(
                          title: 'text_age',
                          icon: LucideIcons.user,
                          text: 'text_input_age',
                          isInputNumber: true,
                          controller: ageController,
                        ),
                        ButtonOutlinedWidget(
                          icon: LucideIcons.map,
                          title: 'text_country',
                          text: selectedCountry,
                          onPressed: (result) {
                            setState(() {
                              FocusManager.instance.primaryFocus?.unfocus();
                              selectedCountry = result;
                            });
                          },
                        ),
                        ButtonOutlinedWidget(
                          title: 'text_education_level',
                          icon: LucideIcons.graduationCap,
                          text: selectedEducationLevel,
                          onPressed: (result) {
                            setState(() {
                              FocusManager.instance.primaryFocus?.unfocus();
                              selectedEducationLevel = result;
                            });
                          },
                        ),
                        ButtonOutlinedWidget(
                          title: 'text_play_time',
                          icon: LucideIcons.gamepad2,
                          text: selectedPlayTime,
                          onPressed: (result) {
                            setState(() {
                              FocusManager.instance.primaryFocus?.unfocus();
                              selectedPlayTime = result;
                            });
                          },
                        ),
```

```
                    SizedBox(height: marginSmall),

                    ButtonBlackWidget(
                      title: 'button_continue',
                      onPressed: () {
                        if (selectedGender.isEmpty ||
                            ageController.text.isEmpty ||
                            selectedCountry.isEmpty ||
                            selectedEducationLevel ==
                                'text_input_education_level'.tr() ||
                            selectedPlayTime ==
                              'text_input_play_time'.tr()) {
                          dialogError(context, 'error_data_collector');
                        } else {
                          userSetup(
                            selectedGender.tr(),
                            int.parse(ageController.text),
                            selectedCountry.tr(),
                            selectedEducationLevel.tr(),
                            selectedPlayTime.tr(),
                          );
                          Navigator.pushNamed(context, '/gameScreen');
                        }
                      },
                    ),
                    ButtonBackWidget(),
                  ],
                ),
              ),
            ),
          ),
        );
    }
}
```

6.3.4 Pantalla de Tutorial

Para facilitar el entendimiento tanto de la interfaz como del algoritmo que usa el juego se ha desarrollado un tutorial en forma de animación que se muestra la primera vez que se entra al juego. Como se puede ver en la Figura 6.9 y la Figura 6.10, esta animación oscurece la toda la pantalla excepto la parte que se desea resaltar y además muestra por pantalla un mensaje explicativo. Este mensaje cuenta en su parte inferior con dos botones, el de la izquierda permite saltar el tutorial por completo mientras que el de la derecha sirve para continuar en él. En el caso que al llamar a esta función *isLast* sea positivo, se eliminará la opción para salir del tutorial y el botón para continuar se modificará para permitir terminar el tutorial, procediendo con el mismo funcionamiento que el botón para salir. Más información sobre este widget en el Apartado 6.4.13.

Una vez el tutorial se haya completado, o se haya pulsado la opción de salir, este se desactivará, impidiendo así que vuelva a aparecer la próxima vez que se abra la aplicación. En el caso que se quiera revertir este cambio y el usuario quiera volver a ver el tutorial, deberá acceder a la pantalla de ajustes y reactivarlo. Tras guardar y volver a la pantalla de juego, el tutorial volverá a aparecer. Más información sobre los ajustes en el Apartado 6.3.6.

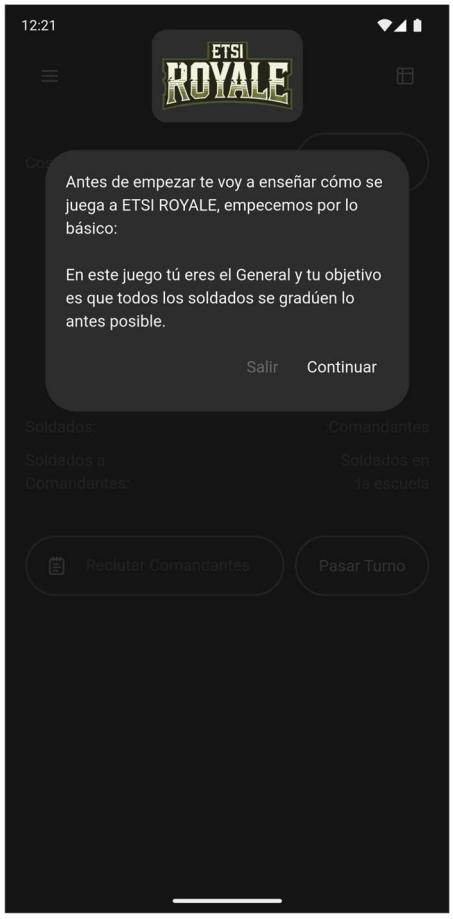

Figura 6.9 Introducción al Tutorial

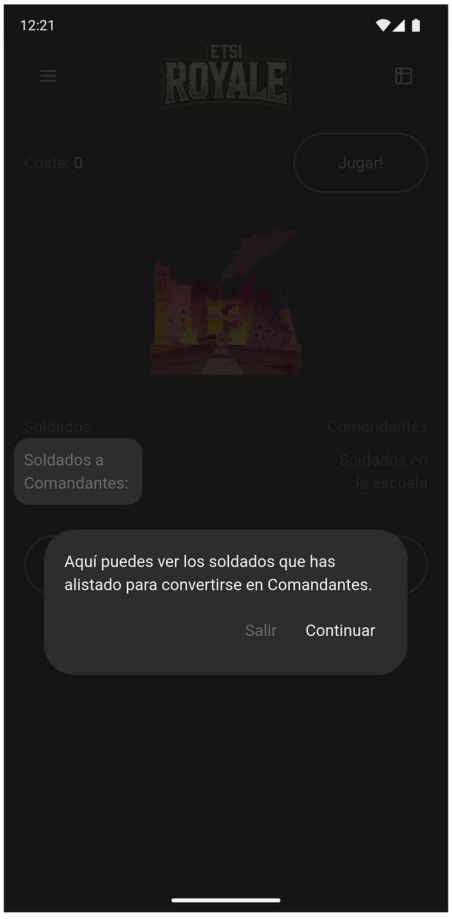

Figura 6.10 Explicación de variables

6.3.5 Pantalla de Juego

Esta es la pantalla principal, deberá ser en la que el usuario se encuentre la mayor parte del tiempo de uso de la aplicación. Será redirigido tras registrarse y rellenar los datos necesarios o en caso de haber jugado con anterioridad, deberá ser la primera pantalla mostrársele para que pueda empezar a jugar de forma inmediata.

Como se puede observar en la Figura 6.11, la pantalla de juego muestra una ilustración, a su izquierda se puede encontrar un botón que dirige al usuario a la pantalla de ajustes y a su derecha el botón dirige hasta la pantalla

de resultado. Esto es seguido por un texto donde se mostrará el coste / puntuación, obtenido en la partida que se esté jugando, así como un botón para iniciar dicha partida. Más adelante encontramos una segunda ilustración representando el campo de batalla [16], cuatro textos que irán acompañado de los datos necesarios para poder desarrollar la partida correctamente. Por último, podemos observar un campo para introducir valores al algoritmo y un botón para poder pasar de turno.

Los datos que se muestran en blanco en la Figura 6.12, son extraídos del Algoritmo de Kondratiev, Código 6.8, mediante el acceso a la matriz de resultados, especificando su fila y columna.

Para mostrar visualmente los movimientos de los soldados y comandantes, se han usado imágenes obtenidas de *Freepik* [16].

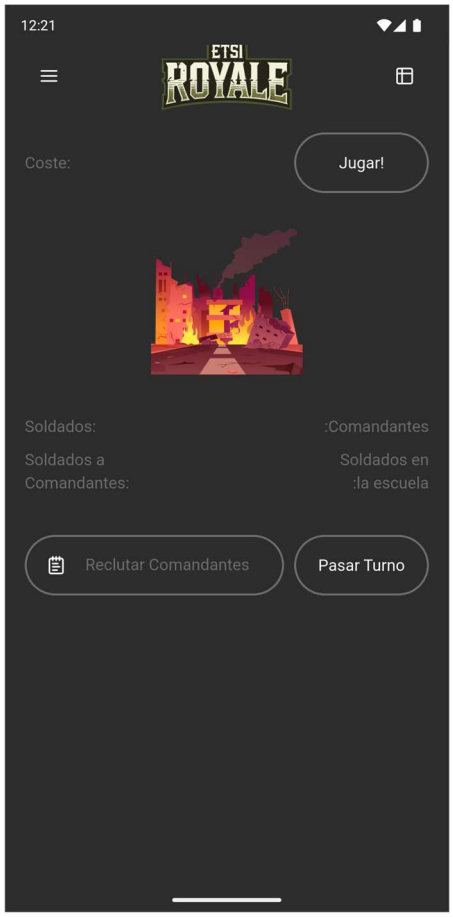

Figura 6.11 Pantalla de Juego

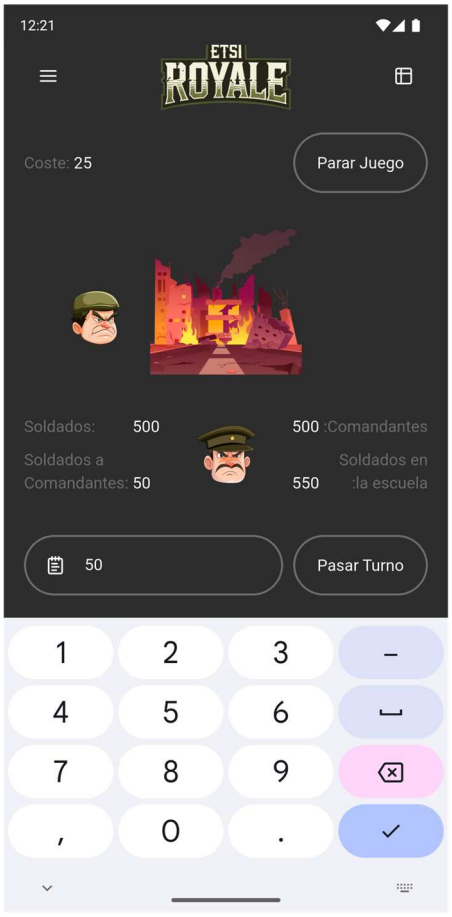

Figura 6.12 Juego en Funcionamiento

Para evitar que el usuario parara la partida debido a una pulsación accidental del botón, se ha implementado un mensaje de conformación. Este aparece siempre al pulsar el botón y muestra dos opciones en la parte inferior, a la derecha para terminar la partida y a la izquierda para volver hacia atrás, Figura 6.13.

También se ha implementado un mensaje al finalizar la partida tras jugar los 70 turno, mostrando la puntuación obtenida en dicha partida, Figura 6.14.

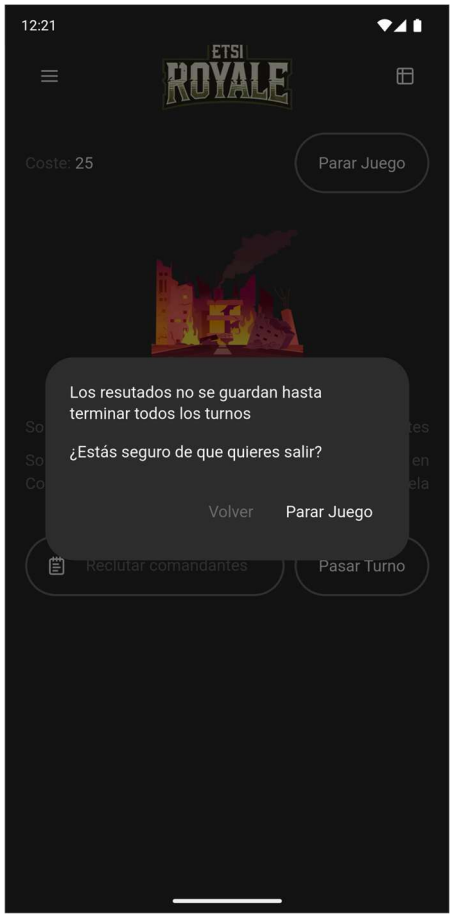

Figura 6.13 Confirmación para Terminar la Partida

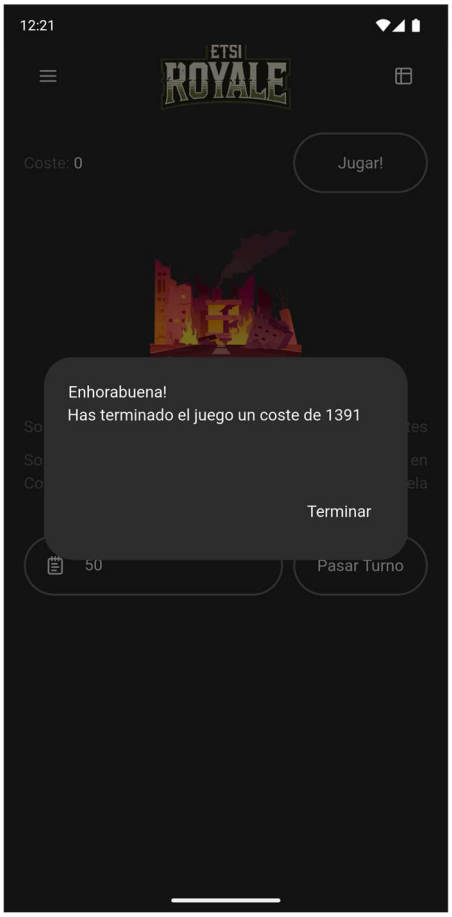

Figura 6.14 Mensaje de Finalización del Juego

Tras realizar pruebas de funcionamiento del algoritmo, nos dimos cuenta de que algunos parámetros tomaban valores negativos, para evitar esto se implementó el reconocimiento de estos usando la función *max*, ya que esta función devuelve el mayor número de entre sus entradas.

Código 6.8 Algoritmo de Kondratiev programado en Dart

```
class KondratievAlgorithm {
  int yearMax = 70;
  int yearAdd = 2;

  int newOrdersGoods = 0;
  int backlogGoods = 0;
  int backlogCapital = 0;
  int desiredProduction = 0;
  int capitalStock = 500;
  int depreciation = 0;
  int production = 0;
  int demandSatisfied = 0;
  int shipmentsGoods = 0;
  int shipmentsCapital = 0;
  int newOrdersCapital = 50;

  List<List<int>> results = List.generate(12, (i) => List.filled(36, 0));
  int score = 0;
  int s2 = 0;
  int year = 0;
```

```
    kondratievAlgorithm(
      bool isGameOn,
      int newOrdersCapital,
      Function resetAlgorithm,
    ) {

      if (isGameOn && year <= yearMax) {
        if (year < 2) {
          newOrdersCapital = 50;
        }

        if (year < 4) {
          newOrdersGoods = 450;
        } else {
          newOrdersGoods = 500;
        }

        backlogCapital = max(0, backlogCapital + newOrdersCapital);
        backlogGoods = backlogGoods + newOrdersGoods;

        desiredProduction = backlogCapital + backlogGoods;
        capitalStock = capitalStock + shipmentsCapital - depreciation;

        int s1 = (desiredProduction - capitalStock).abs() ~/ 100;
        s2 += s1;
        score = 100 * s2 ~/ ((year ~/ yearAdd) + 1);

        if (capitalStock > desiredProduction) {
          production = desiredProduction;
        } else {
          production = capitalStock;
        }
        demandSatisfied = ((production / desiredProduction) * 100).round();

        depreciation = max(0, ((capitalStock * 0.1) / 10).round() * 10);
        shipmentsGoods = max(0,
          ((demandSatisfied / 100 * backlogGoods) / 10).round() * 10,
        );

        shipmentsCapital = max(0, production - shipmentsGoods);

        results[0][year ~/ yearAdd] = newOrdersGoods;
        results[1][year ~/ yearAdd] = backlogGoods;
        results[2][year ~/ yearAdd] = backlogCapital;
        results[3][year ~/ yearAdd] = desiredProduction;
        results[4][year ~/ yearAdd] = capitalStock;
        results[5][year ~/ yearAdd] = depreciation;
        results[6][year ~/ yearAdd] = production;
        results[7][year ~/ yearAdd] = demandSatisfied;
        results[8][year ~/ yearAdd] = shipmentsGoods;
        results[9][year ~/ yearAdd] = shipmentsCapital;
        results[10][year ~/ yearAdd] = newOrdersCapital;

        backlogCapital -= shipmentsCapital;
        backlogGoods -= shipmentsGoods;
      }
    }
  }
```

Código 6.9 Pantalla de Juego

```
class GameScreen extends StatefulWidget {
  const GameScreen({super.key});
  @override
  _GameScreenState createState() => _GameScreenState();
}

saveSharedPreferences() async {
  SharedPreferences prefs = await SharedPreferences.getInstance();
  await prefs.setInt('costeMin', costeMin);
}

class _GameScreenState extends State<GameScreen> {
  TutorialCoachMark? tutorialCoachMark;

  final List<Widget> _generatedContainers = [];
  TextEditingController soldiersController = TextEditingController();
  KondratievAlgorithm algorithm = KondratievAlgorithm();
  String result1 = '';
  String result2 = '';
  String result3 = '';
  String result4 = '';
  String startTitle = 'button_start_game'.tr();
  bool isGameOn = false;
  Completer<void> _completer = Completer<void>();

  List<TargetFocus> targets = [];
  GlobalKey tutorial_1_key = GlobalKey();
  GlobalKey tutorial_2_key = GlobalKey();
  GlobalKey tutorial_3_key = GlobalKey();
  GlobalKey tutorial_4_key = GlobalKey();
  GlobalKey tutorial_5_key = GlobalKey();
  GlobalKey tutorial_6_key = GlobalKey();
  GlobalKey tutorial_7_key = GlobalKey();
  GlobalKey tutorial_8_key = GlobalKey();
  GlobalKey tutorial_9_key = GlobalKey();
  GlobalKey tutorial_10_key = GlobalKey();
  GlobalKey tutorial_11_key = GlobalKey();
  GlobalKey tutorial_12_key = GlobalKey();

  @override
  void initState() {
    Future.delayed(const Duration(seconds: 1), () {
      showCoachMark();
    });
    super.initState();
  }

  void updateText() {
    if (isGameOn) {
      startTitle = 'button_end_game'.tr();
    } else {
      startTitle = 'button_start_game'.tr();
    }
    setState(() {});
  }
```

```
void resetAlgorithm() {
  setState(() {
    startTitle = 'button_start_game'.tr();
    algorithm.newOrdersGoods = 0; // 0
    algorithm.backlogGoods = 0; // 1
    algorithm.backlogCapital = 0; // 2
    algorithm.desiredProduction = 0; // 3
    algorithm.capitalStock = 500; // 4
    algorithm.depreciation = 0; // 5
    algorithm.production = 0; // 6
    algorithm.demandSatisfied = 0; // 7
    algorithm.shipmentsGoods = 0; // 8
    algorithm.shipmentsCapital = 0; // 9
    algorithm.newOrdersCapital = 50; // 10
    algorithm.score = 0;
    algorithm.s2 = 0;
    algorithm.i = 0;
    algorithm.year = 0;

    result1 = '';
    result2 = '';
    result3 = '';
    result4 = '';
  });
}

void stopAnimation() {
  if (!_completer.isCompleted) {
    _completer.complete();
  }
}

void runAlgorithm() async {
  _completer = Completer<void>();

  if (algorithm.year <= algorithm.yearMax) {
    setState(() {
      startTitle = 'button_end_game'.tr();
    });

    int number = int.tryParse(soldiersController.text) ?? 0;
    algorithm.kondratievAlgorithm(isGameOn, number, resetAlgorithm);
    int animationSoldier = algorithm.results[1][algorithm.year ~/ 2];
    int animationCommander = algorithm.results[10][algorithm.year ~/ 2];
    final double imageSize = 60;

    for (int i = 0; i < min(animationSoldier / 100, 10); i += 1) {
      await Future.any([
        Future.delayed(Duration(milliseconds: 500)),
        _completer.future,
      ]);
      if (_completer.isCompleted) return;
      Widget newContainer = AnimatedContainerWidget(
        containerSize: imageSize,
        initialTop: 280,
        initialLeft: -imageSize,
        moveOffset: 180,
        moveAnimationDuration: 1300,
        image: 'assets/images/ImagenSoldado.png',
```

```
            movementeLeft: true,
          );
          setState(() {
            _generatedContainers.add(newContainer);
          });
        }

        for (int i = 0; i < min(animationCommander / 10, 10); i += 1) {
          await Future.any([
            Future.delayed(Duration(milliseconds: 500)),
            _completer.future,
          ]);
          if (_completer.isCompleted) return;
          Widget newContainer = AnimatedContainerWidget(
            containerSize: imageSize,
            initialTop: (altoPantallaTotal(context) - imageSize) / 2 + 40,
            initialLeft: (anchoPantallaTotal(context) - imageSize) / 2,
            moveOffset: 120,
            moveAnimationDuration: 1000,
            image: 'assets/images/ImagenCoronel.png',
            movementeLeft: false,
          );
          setState(() {
            _generatedContainers.add(newContainer);
          });
        }
        setState(() {
          result1 = algorithm.results[1][algorithm.year ~/ 2].toString();
          result2 = algorithm.results[2][algorithm.year ~/ 2].toString();
          result3 = algorithm.results[3][algorithm.year ~/ 2].toString();
          result4 = algorithm.results[4][algorithm.year ~/ 2].toString();
          algorithm.year += algorithm.yearAdd;
        });
      }
    }

    void showCoachMark() {
      initTargets();
      if (isGameTutorialActive) {
        tutorialCoachMark = TutorialCoachMark(
          targets: targets,
          pulseEnable: false,
          colorShadow: Colors.black,
          hideSkip: true,
        )..show(context: context);
      }
    }

    TargetFocus targuetFocus(GlobalKey key, String text, bool isLast) {
      return TargetFocus(
        keyTarget: key,
        shape: ShapeLightFocus.RRect,
        radius: borderRadiusSmall,
        paddingFocus: marginMedium,
        enableOverlayTab: true,
        contents: [
          TargetContent(
            align: ContentAlign.bottom,
            builder: (context, controller) {
```

```
                return CoachMark(
                  text: text.tr(),
                  isLast: isLast,
                  onSkip: () {
                    controller.skip();
                    isGameTutorialActive = false;
                    saveSharedPreferences();
                  },
                  onNext: () {
                    if (isLast) {
                      controller.skip();
                      isGameTutorialActive = false;
                      saveSharedPreferences();
                    } else {
                      controller.next();
                    }
                  },
                );
              },
            ),
          ],
        );
      }

      void initTargets() {
        targets = [
          targuetFocus(tutorial_1_key, "tutorial_1", false),
          targuetFocus(tutorial_2_key, "tutorial_2", false),
          targuetFocus(tutorial_3_key, "tutorial_3", false),
          targuetFocus(tutorial_4_key, "tutorial_4", false),
          targuetFocus(tutorial_5_key, "tutorial_5", false),
          targuetFocus(tutorial_6_key, "tutorial_6", false),
          targuetFocus(tutorial_7_key, "tutorial_7", false),
          targuetFocus(tutorial_8_key, "tutorial_8", false),
          targuetFocus(tutorial_9_key, "tutorial_9", false),
          targuetFocus(tutorial_10_key, "tutorial_10", false),
          targuetFocus(tutorial_11_key, "tutorial_11", false),
          targuetFocus(tutorial_12_key, "tutorial_12", true),
        ];
      }

      @override
      Widget build(BuildContext context) {
        return GestureDetector(
          onTap: () {
            FocusScope.of(context).unfocus();
          },
          child: Scaffold(
            body: Stack(
              children: [
                SingleChildScrollView(
                  child: Container(
                    height: altoPantallaTotal(context),
                    width: anchoPantallaTotal(context),
                    color: colorbackground,
                    padding: const EdgeInsets.all(marginMedium),
                    child: Column(
                      mainAxisAlignment: MainAxisAlignment.start,
                      children: [
```

```
                        SizedBox(height: 15),
                        Row(
                          mainAxisAlignment:
                            MainAxisAlignment.spaceBetween,
                          children: [
                            Align(
                              key: tutorial_11_key,
                              alignment: Alignment.topRight,
                              child: SizedBox(
                                height: blockHeight / 2,
                                child: IconButton(
                                  icon: const
                                    Icon(LucideIcons.alignJustify),
                                  iconSize: iconSize,
                                  color: colorprimary,
                                  padding: const EdgeInsets.all(0),
                                  onPressed: () {
                                    Navigator.pushNamed(context,
                                      '/settings');
                                  },
                                ),
                              ),
                            ),
                            Image.asset(
                              key: tutorial_1_key,
                              'assets/images/Logo.png',
                              height: blockLogoHeight / 5 * 3,
                            ),
                            Align(
                              key: tutorial_12_key,
                              alignment: Alignment.topRight,
                              child: SizedBox(
                                height: blockHeight / 2,
                                child: IconButton(
                                  icon: const Icon(LucideIcons.table2),
                                  iconSize: iconSize,
                                  color: colorprimary,
                                  padding: const EdgeInsets.all(0),
                                  onPressed: () {
                                    setState(() {
                                      Navigator.pushNamed(
                                        context,
                                        '/results',
                                        arguments: algorithm,
                                      );
                                    });
                                  },
                                ),
                              ),
                            ),
                          ],
                        ),
                        SizedBox(height: marginMedium),
                        Row(
                          children: [
                            SizedBox(
                              height: blockHeight,
                              width: anchoPantallaUsable(context)
                                * 2 / 3,
```

```
                    child: Column(
                      mainAxisAlignment:
                        MainAxisAlignment.center,
                      crossAxisAlignment:
                        CrossAxisAlignment.start,
                      children: [
                        Row(
                          children: [
                            Text(
                              'game_puntuation'.tr(),
                              style: fontStyleSubtitle,
                            ),
                            Text(
                              algorithm.score.toString(),
                              style: fontStyleBodyWhite,
                            ),
                          ],
                        ),
                      ],
                    ),
                  ),
                  ButtonOutlinedCenteredWidget(
                    key: tutorial_10_key,
                    width: anchoPantallaUsable(context) / 3,
                    title: startTitle,
                    onPressed: () {
                      vibration();
                      if (isGameOn == true) {
                        dialogshowStopGame(context, () {
                          stopAnimation();
                          isGameOn = false;
                          setState(() {});
                          updateText();
                          resetAlgorithm();
                          soldiersController.clear();
                        });
                      }
                      if (isGameOn == false) {
                        isGameOn = true;
                        setState(() {});
                        updateText();
                        resetAlgorithm();
                        runAlgorithm();
                      }
                    },
                  ),
                ],
              ),
              SizedBox(height: marginMedium * 3 / 2),
              Row(
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
                  SizedBox(
                    key: tutorial_2_key,
                    height: blockHeight * 2.5,
                    width: blockHeight * 2.5,
                    child: Image.asset('assets/images/
                      ImagenBatalla.png'),
                  ),
```

```
                    ],
                  ),
                  SizedBox(height: marginMedium * 2),
                  Row(
                    mainAxisAlignment:
                      MainAxisAlignment.spaceBetween,
                    children: [
                      Column(
                        children: [
                          Row(
                            children: [
                              Column(
                                crossAxisAlignment:
                                  CrossAxisAlignment.start,
                                children: [
                                  Text(
                                    key: tutorial_3_key,
                                    'game_info_soldier'.tr(),
                                    style: fontStyleBodyGray,
                                  ),
                                  SizedBox(height: marginSmall),
                                  Column(
                                    key: tutorial_4_key,
                                    crossAxisAlignment:
                                      CrossAxisAlignment.start,
                                    children: [
                                      Text(
                                        'game_info_commander_1'
                                          .tr(),
                                        style:
                                          fontStyleBodyGray,
                                      ),
                                      Text(
                                        'game_info_commander_2'
                                          .tr(),
                                        style:
                                          fontStyleBodyGray,
                                      ),
                                    ],
                                  ),
                                ],
                              ),
                              Column(
                                mainAxisAlignment:
                                  MainAxisAlignment.end,
                                crossAxisAlignment:
                                  CrossAxisAlignment.start,
                                children: [
                                  Text(
                                    result1,
                                    style: fontStyleBodyWhite,
                                  ),
                                  SizedBox(height: marginSmall),
                                  Text(
                                    '50',
                                    style: fontStyleBodyWhite
                                      .copyWith(
                                      color: colorbackground,
                                    ),
```

```
                                    ),
                                    Text(
                                      result2,
                                      style: fontStyleBodyWhite,
                                    ),
                                  ],
                                ),
                              ],
                            ),
                          ],
                        ),

                        Column(
                          key: tutorial_7_key,
                          crossAxisAlignment: CrossAxisAlignment.end,
                          children: [
                            Row(
                              children: [
                                Text(
                                  result4, // * 4: Profesores
                                  style: fontStyleBodyWhite,
                                ),
                                Text(
                                  key: tutorial_6_key,
                                  'game_info_commander_total'.tr(),
                                  style: fontStyleBodyGray,
                                ),
                              ],
                            ),
                            SizedBox(height: marginSmall),
                            Row(
                              mainAxisAlignment:
                                MainAxisAlignment.start,
                              crossAxisAlignment:
                                CrossAxisAlignment.end,
                              children: [
                                Column(
                                  key: tutorial_5_key,
                                  crossAxisAlignment:
                                    CrossAxisAlignment.end,
                                  children: [
                                    Text('game_info_soldier
                                      _and_commander_1'.tr(),
                                      style: fontStyleBodyGray,
                                    ),
                                    Row(
                                      children: [
                                        Align(
                                          alignment: Alignment
                                            .bottomRight,
                                          child: Text(
                                            result3,
                                            style:
                                              fontStyleBodyWhite,
                                          ),
                                        ),
                                        Text(_info_soldier
                                          _and_commander_2'.tr(),
                                          style:
```

```
                                        fontStyleBodyGray,
                                    ),
                                ],
                            ),
                        ],
                    ),
                ],
            ),
        ],
    ),

    SizedBox(height: marginMedium * 2),

    Row(
        children: [
            Expanded(
                child: TextInputWidget(
                    key: tutorial_8_key,
                    icon: LucideIcons.notepadText,
                    controller: soldiersController,
                    text:
                        'game_recruit_commander_turns'.tr(),
                    isInputNumber: true,
                ),
            ),
            SizedBox(width: marginSmall),
            ButtonOutlinedCenteredWidget(
                key: tutorial_9_key,
                width: anchoPantallaUsable(context) / 3,
                title: 'button_next_turn'.tr(),
                onPressed: () {
                    vibration();
                    if (isGameOn) {
                        if (algorithm.year <=
                            algorithm.yearMax) {
                            stopAnimation();
                            runAlgorithm();
                            // costeMin = algorithm.score;
                        } else if (algorithm.year >
                            algorithm.yearMax) {
                            setState(() {
                                dialogEndGame(context,
                                    algorithm.score);

                                userGameData(
                                    algorithm.score,
                                    algorithm.results,
                                    algorithm.production,
                                    algorithm.demandSatisfied,
                                );
                                isGameOn = false;
                                updateText();
                                resetAlgorithm();
                            });
                        }
                    }
                },
```

```
                                ),
                              ],
                            ),
                          ],
                        ),
                      ),
                    ),
                    ..._generatedContainers,
                  ],
                ),
              ),
            );
        }
    }

    class AnimatedContainerWidget extends StatefulWidget {
        final double initialTop;
        final double initialLeft;
        final double containerSize;
        final double moveOffset;
        final int moveAnimationDuration;
        final String image;
        final bool movementeLeft;

        const AnimatedContainerWidget({
            super.key,
            required this.initialTop,
            required this.initialLeft,
            required this.containerSize,
            required this.moveOffset,
            required this.moveAnimationDuration,
            required this.image,
            this.movementeLeft = false,
        });

        @override
        _AnimatedContainerWidgetState createState() =>
             _AnimatedContainerWidgetState();
    }

    class _AnimatedContainerWidgetState extends State<AnimatedContainerWidget>
    {
        late double top;
        late double left;
        late double opacity;

        @override
        void initState() {
            super.initState();
            top = widget.initialTop;
            left = widget.initialLeft;
            opacity = 1.0;

            Timer(Duration(milliseconds: 0), () {
                setState(() {
                    if (widget.movementeLeft) {
                        left = left = widget.initialLeft + widget.moveOffset;
                    } else {
                        top = widget.initialTop - widget.moveOffset;
```

```
            }
          });

          Timer(Duration(milliseconds: widget.moveAnimationDuration ~/ 2), ()
          {
            setState(() {
              opacity = 0.0;
            });

            Timer(Duration(milliseconds: widget.moveAnimationDuration ~/ 2),
            () {
              if (mounted) {
                setState(() {
                  // Remove the widget after the animation completes
                });
              }
            });
          });
        });
      }

      @override
      Widget build(BuildContext context) {
        return AnimatedPositioned(
          duration: Duration(milliseconds: widget.moveAnimationDuration),
          top: top,
          left: left,
          child: AnimatedOpacity(
            opacity: opacity,
            duration: Duration(milliseconds: widget.moveAnimationDuration ~/
            2),
            child: Image.asset(
              widget.image,
              height: widget.containerSize,
              width: widget.containerSize,
            ),
          ),
        );
      }
    }
```

6.3.6 Pantalla de Ajustes

Cada persona tiene unos gustos particulares, esto provoca que la configuración óptima para un individuo no tiene por qué ser la misma que para otro individuo. Debido a esto se ha configurado la siguiente página de ajustes.

Tanto los iconos como el texto que se pueden ver en las opciones de la Figura 6.15 cambian en función del ajuste seleccionado para una mayor interacción visual del usuario con la aplicación. Estos ajustes son guardados como *SharedPreferences*, lo cual permitirá recuperarlos una vez reiniciada por completo la aplicación.

Esta pantalla también cuenta con un botón de color gris que permite al usuario de la aplicación cerrar sesión sin importar su método de registro. Para asegurar que no se ha pulsado de manera accidental, antes de continuar cerrando la sesión del usuario, como se muestra en la Figura 6.16, se muestra por pantalla un *dialog* de confirmación. Más información sobre este widget en el Código 6.21.

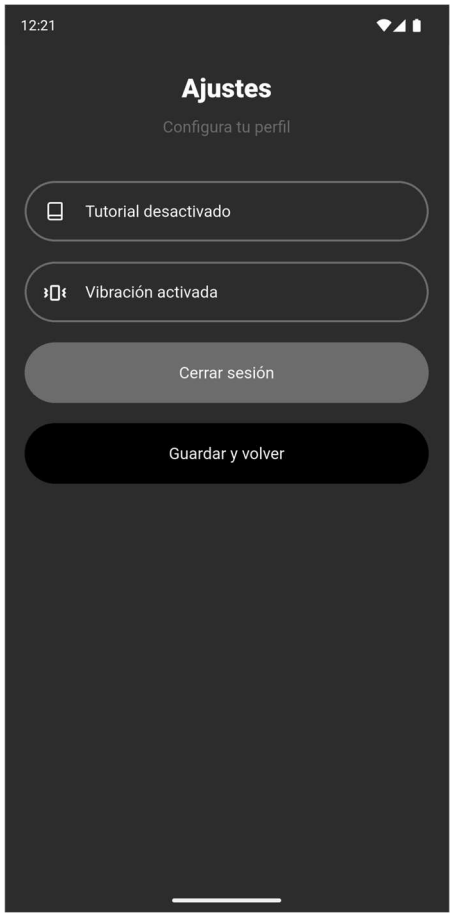

Figura 6.15 Pantalla de Ajustes

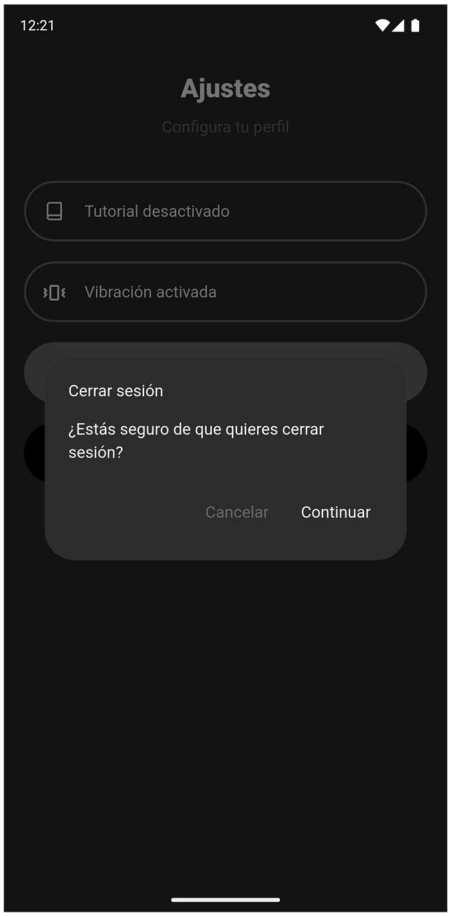

Figura 6.16 Confirmación para Cerrar la Sesión

Código 6.10 Pantalla de Ajustes

```
class SettingsScreen extends StatefulWidget {
  SettingsScreen({super.key});

  @override
  _SettingsScreenState createState() => _SettingsScreenState();
}

saveSharedPreferences() async {
  SharedPreferences prefs = await SharedPreferences.getInstance();
  await prefs.setBool('isVibrationActive', isVibrationActive);
  await prefs.setBool('isGameTutorialActive', isGameTutorialActive);
  await prefs.setBool('isRealTimeGameActive', isRealTimeGameActive);
}

class _SettingsScreenState extends State<SettingsScreen> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: SingleChildScrollView(
        child: Container(
          height: altoPantallaTotal(context),
          width: anchoPantallaTotal(context),
          color: colorbackground,
          padding: const EdgeInsets.all(marginMedium),
          child: Column(
```

```
                    mainAxisAlignment: MainAxisAlignment.start,
                    children: <Widget>[
                      TitleWidget(
                        title: 'title_settings',
                        subtitle: 'subtitle_settings',
                      ),
                      SizedBox(height: marginMedium),
                      ButtonGrayWidget(
                        titleOn: 'settigns_tutorial_on',
                        titleOff: 'settigns_tutorial_off',
                        iconOn: LucideIcons.bookOpen,
                        iconOff: LucideIcons.book,
                        setting: isGameTutorialActive,
                        onPressed: () {
                          isGameTutorialActive = !isGameTutorialActive;
                        },
                      ),
                      ButtonGrayWidget(
                        titleOn: 'settings_vibration_on',
                        titleOff: 'settings_vibration_off',
                        iconOn: LucideIcons.vibrate,
                        iconOff: LucideIcons.vibrateOff,
                        iconBiggerBy: 3.5,
                        setting: isVibrationActive,
                        onPressed: () {
                          isVibrationActive = !isVibrationActive;
                        },
                      ),
                      ButtonLogOutWidget(),
                      ButtonBlackWidget(
                        title: 'button_save_back',
                        onPressed: () async {
                          final navigator = Navigator.of(context);
                          await saveSharedPreferences();
                            navigator.pushReplacement(
                            MaterialPageRoute(builder: (context) =>
                              GameScreen()),
                            );
                        },
                      ),
                    ],
                  ),
                ),
              ),
            );
        }
    }
```

6.3.7 Pantalla de Resultados

En esta pantalla se muestran los resultados obtenidos en la partida. Si se accede a esta pantalla y no se ha comenzado una partida, la tabla se encontrará vacía, como se muestra en la Figura 6.17, pero en el caso que se haya iniciado una, los datos irán apareciendo según vayan pasando los turnos, Figura 6.18. La pantalla cuenta con un sistema de deslizamiento, permitiendo ver más resultados de los que se pueden mostrar por pantalla de forma regular.

Hay que añadir que se puede acceder a esta pantalla en cualquier momento de la partida sin necesidad de pararla, pudiendo así ver los resultados obtenidos en tiempo real. Al pulsar el botón de volver, se regresará a la pantalla de juego en el punto que se dejó y pudiendo continuar la partida.

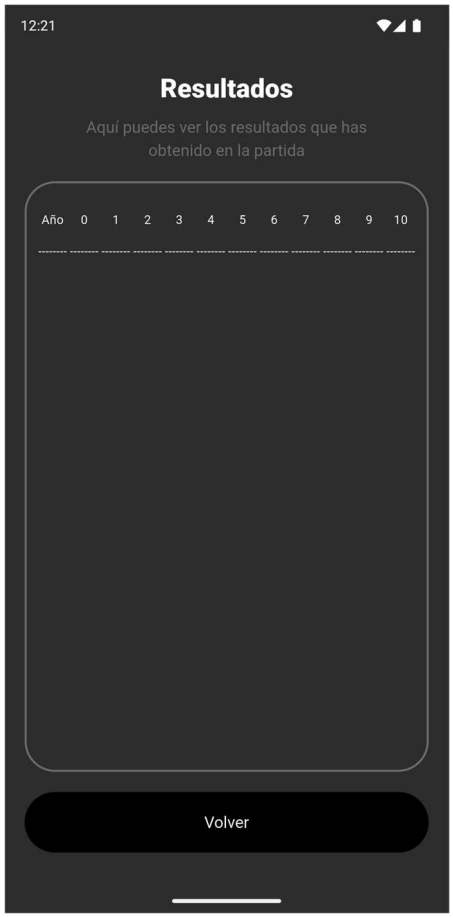

Figura 6.17 Pantalla de Resultados Vacía

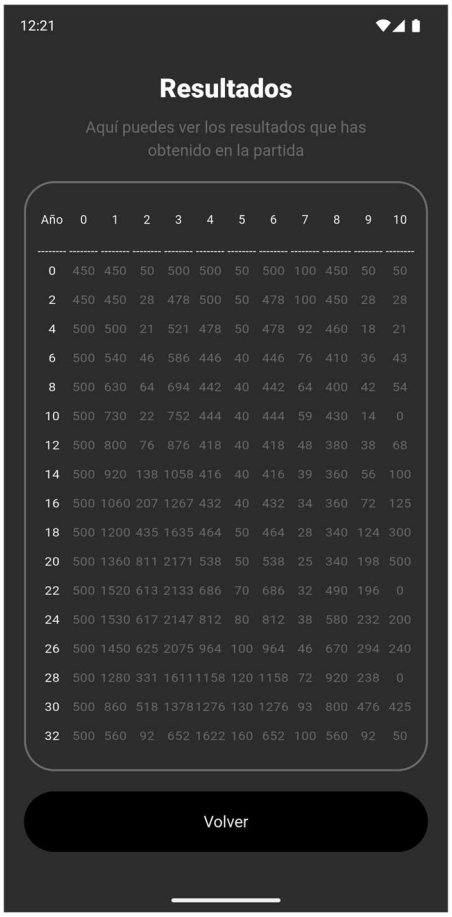

Figura 6.18 Pantalla de Resultados Jugando

Código 6.11 Pantalla de Resultados

```
class ResultsScreen extends StatefulWidget {
  final KondratievAlgorithm algorithm;

  ResultsScreen({super.key, required this.algorithm});

  @override
  _ResultsScreenState createState() => _ResultsScreenState();
}

class _ResultsScreenState extends State<ResultsScreen> {
  late KondratievAlgorithm algorithm;
```

```
@override
void initState() {
  super.initState();
  algorithm = widget.algorithm;
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    body: SingleChildScrollView(
      child: Column(
        children: [
          Container(
            height: altoPantallaTotal(context),
            width: anchoPantallaTotal(context),
            color: colorbackground,
            padding: const EdgeInsets.all(marginMedium),
            child: Column(
              mainAxisAlignment: MainAxisAlignment.start,
              children: [
                TitleWidget(
                  title: 'title_results',
                  subtitle: 'subtitle_results',
                ),
                SizedBox(height: marginMedium),
                Container(
                  height: altoPantallaTotal(context) * 0.65,
                  width: anchoPantallaTotal(context),
                  padding: const EdgeInsets.fromLTRB(marginSmall,
                    marginMedium, marginSmall, marginMedium),
                  decoration: BoxDecoration(
                    borderRadius:
                      BorderRadius.circular(borderRadiusBig),
                    color: colorbackground,
                    border: Border.all(
                      color: colorsecondary,
                      width: blockBorder,
                    ),
                  ),
                  child: GridView.count(
                    padding: const EdgeInsets.all(0),
                    crossAxisCount: 12,
                    mainAxisSpacing: 0,
                    crossAxisSpacing: 0,
                    children: [
                      Center(
                           child: Text(
                             'text_results_year'.tr(),
                             style: TextStyle(
                               fontSize: fontSizeResults,
                               color: colorprimary,
                             ),
                           ),
                         ),
                      for (int i = 0; i <= 10; i++)
                        Center(
                          child: Text(
                            i.toString(),
                            style: TextStyle(
```

```
                                    fontSize: fontSizeResults,
                                    color: colorprimary,
                                  ),
                                ),
                              ),
                              for (int i = 0; i <= 11; i++)
                              Center(
                                child: Text(
                                  '--------',
                                  style: TextStyle(
                                    fontSize: fontSizeResults,
                                    color: colorprimary,
                                  ),
                                ),
                              ),

                                for (int i = 0; i < algorithm.year;
                                  i += 2) ...[
                                  Center(
                                  child: Text(
                                    i.toString(),
                                    style: TextStyle(
                                    fontSize: fontSizeResults,
                                    color: colorprimary,
                                    ),
                                  ),
                                  ),
                                  for (int j = 0; j < 11; j++)
                                  Center(
                                    child: Text(
                                      algorithm.results[j]
                                        [i ~/ 2].toString(),
                                    style: TextStyle(
                                      fontSize: fontSizeResults,
                                      color: colorsecondary,
                                    ),
                                    ),
                                  ),
                                ],
                            ],
                          ),
                        ),
                        SizedBox(height: marginMedium),
                        ButtonBlackWidget(
                          title: 'button_back',
                          onPressed: () {
                            Navigator.pop(context);
                          },
                        ),
                      ],
                    ),
                  ),
                ],
              ),
            ),
          );
    }
}
```

6.4 Widgets Personalizados

A continuación, se muestran los widgets creados dados los requerimientos de interfaz y comunicación de la aplicación. Todos ellos han sido desarrollados de manera personalizada para reducir la repetición de código. Muchos están configurados para retroalimentarse ya sea de forma visual o con variables de trabajo.

6.4.1 AnimatedContainerWidget

Establece la creación, el movimiento y el cambio de opacidad en las distintas animaciones usadas en la pantalla de juego.

Código 6.12 AnimatedContainerWidget

```
    class AnimatedContainerWidget extends StatefulWidget {
      final double initialTop;
      final double initialLeft;
      final double containerSize;
      final double moveOffset;
      final int moveAnimationDuration;
      final String image;
      final bool movementeLeft;

      const AnimatedContainerWidget({
        super.key,
        required this.initialTop,
        required this.initialLeft,
        required this.containerSize,
        required this.moveOffset,
        required this.moveAnimationDuration,
        required this.image,
        this.movementeLeft = false,
      });

      @override
      _AnimatedContainerWidgetState createState() =>
          _AnimatedContainerWidgetState();
    }

    class _AnimatedContainerWidgetState extends State<AnimatedContainerWidget>
{
      late double top;
      late double left;
      late double opacity;

      @override
      void initState() {
        super.initState();
        top = widget.initialTop;
        left = widget.initialLeft;
        opacity = 1.0;

        Timer(Duration(milliseconds: 0), () {
          setState(() {
            if (widget.movementeLeft) {
              left = left = widget.initialLeft + widget.moveOffset;
            } else {
              top = widget.initialTop - widget.moveOffset;
```

```
            }
          });

          Timer(Duration(milliseconds: widget.moveAnimationDuration ~/ 2),
            () {
            setState(() {
              opacity = 0.0;
            });

            Timer(Duration(milliseconds: widget.moveAnimationDuration ~/ 2),
              () {
              if (mounted) {
                setState(() {});
              }
            });
          });
        });
      }

      @override
      Widget build(BuildContext context) {
        return AnimatedPositioned(
          duration: Duration(milliseconds: widget.moveAnimationDuration),
          top: top,
          left: left,
          child: AnimatedOpacity(
            opacity: opacity,
            duration: Duration(milliseconds: widget
              .moveAnimationDuration ~/ 2),
            child: Image.asset(
              widget.image,
              height: widget.containerSize,
              width: widget.containerSize,
            ),
          ),
        );
      }
    }
```

6.4.2 BotomSheetCountryPicker

Se trata de un widget destinado a la selección del país de residencia. Utiliza el paquete del repositorio de Dart llamado *CountryPicker* [17].

Está construido por un *BotomSheet* o bandeja la cual aparece desde abajo y ocupa la mitad de la pantalla. En esta bandeja se muestra una lista de países y sus banderas, así como un buscador en la parte superior. Para que estuviera más integrado con el resto de la interfaz gráfica, se han modificado algunos de sus parámetros de configuración, como el color de fondo, tamaño de la bandera o estilo de texto. La funcionalidad de este widget se puede ver en la Figura 6.7.

Código 6.13 BotomSheetCountryPicker

```
    Future<String?> botomSheetCountryPicker(BuildContext context) async {
      Completer<String?> completer = Completer<String?>();

      showCountryPicker(
```

```
        context: context,
        onSelect: (Country country) {
          completer.complete(country.nameLocalized);
        },
        favorite: ['ES'],
        countryListTheme: CountryListThemeData(
          flagSize: flagSize,
          backgroundColor: colorbackground,
          textStyle: fontStyleBodyWhite,
          bottomSheetHeight: altoPantallaBottomSheet(context),
          inputDecoration: InputDecoration(
            hintText: 'text_country'.tr(),
            labelStyle: fontStyleBodyWhite,
            border: OutlineInputBorder(
              borderSide: BorderSide(color: colorsecondary),
              borderRadius: BorderRadius.circular(borderRadiusBig),
            ),
          ),
        ),
      );
      return completer.future;
    }
```

6.4.3 BottomSheetEducationLevel

Esta pantalla también muestra una bandeja con múltiples opciones entre las cuales el usuario debe seleccionar una, pero en este caso no se usa ningún paquete externo para facilitar la programación, si no que está construido de forma personalizada. El widget se ha creado para mostrar por pantalla 3 niveles de estudios, de los cuales el usuario debe elegir el que lo mejor lo represente. Una vez elegido el nivel, el widget se encarga de devolver usando la clase *Navigator* el valor seleccionado.

Código 6.14 BottomSheetEducationLevel

```
    Future<String?> bottomSheetEducationLevel(BuildContext context) async {
      return await showModalBottomSheet<String>(
        context: context,
        shape: const RoundedRectangleBorder(
          borderRadius: BorderRadius.vertical(
            top: Radius.circular(borderRadiusBig),
          ),
        ),
        backgroundColor: colorbackground,
        builder:
            (context) => SizedBox(
              height: altoPantallaBottomSheet(context),
              child: Padding(
                padding: const EdgeInsets.all(marginMedium),
                child: Column(
                  children: [
                    Text('text_education_level'.tr(),
                      style:fontStyleTitle),
                    SizedBox(height: marginMedium),
                    ButtonOutlinedCenteredWidget(
                      title: 'text_input_education_level_basic',
                      onPressed: () {
```

```
                        Navigator.pop(context,
                      'text_input_education_level_basic');
                      },
                    ),
                    SizedBox(height: marginMedium),
                    ButtonOutlinedCenteredWidget(
                      title: 'text_input_education_level_middle',
                      onPressed: () {
                        Navigator.pop(context,
                      'text_input_education_level_middle');
                      },
                    ),
                    SizedBox(height: marginMedium),
                    ButtonOutlinedCenteredWidget(
                      title: 'text_input_education_level_higher',
                      onPressed: () {
                        Navigator.pop(context,
                      'text_input_education_level_higher');
                      },
                    ),
                  ],
                ),
              ),
            ),
        );
    }
```

6.4.4 BottomSheetPlayTime

Al igual que el widget anterior, este muestra en pantalla una bandeja con múltiples. En este caso, aparecerán 4 opciones relacionadas con el tiempo que dedica el usuario a jugar a videojuegos. Las 3 primeras opciones varían entre jugar una vez al día y jugar una vez al mes, mientras que la última opción representa la ausencia total de tiempo de juego.

Código 6.15 BottomSheetPlayTime

```
    Future<String?> bottomSheetPlayTime(BuildContext context) async {
      return await showModalBottomSheet<String>(
        context: context,
        shape: const RoundedRectangleBorder(
          borderRadius: BorderRadius.vertical(
            top: Radius.circular(borderRadiusBig),
          ),
        ),
        backgroundColor: colorbackground,
        builder:
            (context) => SizedBox(
              height: altoPantallaBottomSheet(context),
              child: Padding(
                padding: const EdgeInsets.all(marginMedium),
                child: Column(
                  children: [
                    Text('text_play_time'.tr(), style: fontStyleTitle),
                    SizedBox(height: marginMedium),
                    ButtonOutlinedCenteredWidget(
```

```
                        title: 'text_input_play_once_day',
                        onPressed: () {
                          Navigator.pop(context,
                            'text_input_play_once_day');
                        },
                      ),
                      SizedBox(height: marginMedium),
                      ButtonOutlinedCenteredWidget(
                        title: 'text_input_play_once_week',
                        onPressed: () {
                          Navigator.pop(context,
                            'text_input_play_once_week');
                        },
                      ),
                      SizedBox(height: marginMedium),
                      ButtonOutlinedCenteredWidget(
                        title: 'text_input_play_once_month',
                        onPressed: () {
                          Navigator.pop(context,
                            'text_input_play_once_month');
                        },
                      ),
                      SizedBox(height: marginMedium),
                      ButtonOutlinedCenteredWidget(
                        title: 'text_input_play_never',
                        onPressed: () {
                          Navigator.pop(context, 'text_input_play_never');
                        },
                      ),
                    ],
                  ),
                ),
              ),
    );
  }
```

6.4.5 ButtonBackWidget

Dentro de la interfaz de la aplicación es necesario que exista algún sistema para poder volver a la pantalla anterior, este widget suple esa necesitad. El widget se encuentra configurado de tal forma siempre volverá a la pantalla anteriormente visitada sin importar donde se coloque. Para ello usa un comando propio de *Flutter* que se comentó en el Apartado 6.2, *Navigator.pop(context)*.

Código 6.16 ButtonBackWidget

```
class ButtonBackWidget extends StatelessWidget {
  const ButtonBackWidget({super.key});

  @override
  Widget build(BuildContext context) {
    return Container(
      height: blockHeight,
      width: blockLengthBig / 2,
      decoration: BoxDecoration(
        borderRadius: BorderRadius.circular(borderRadiusBig),
```

```
          ),
          child: TextButton(
            onPressed: () {
              vibration();
              Navigator.pop(context
            },
            child: Center(
              child: Text("button_back".tr(), style: fontStyleBodyWhite),
            ),
          ),
        );
    }
}
```

6.4.6 ButtonBlackWidget

Algunos botones de color negro que se han podido ver en la interfaz de la aplicación, como en la Figura 6.1, se encuentran usando esta clase. Se encarga de crear la interfaz del botón, así como de su funcionalidad, la cual está configurada con la función *onPressed*, que se puede definir cuando este widget es llamado.

Código 6.17 ButtonBlackWidget

```
class ButtonBlackWidget extends StatelessWidget {
  final String title;
  final Function onPressed;

  const ButtonBlackWidget({
    super.key,
    required this.title,
    required this.onPressed,
  });

  @override
  Widget build(BuildContext context) {
    return OutlinedButton(
      onPressed: () {
        vibration();
        onPressed();
      },
      style: OutlinedButton.styleFrom(
        backgroundColor: colorbutton,
        side: BorderSide(color: colorbutton, width: blockBorder),
        shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(borderRadiusBig),
        ),
        padding: EdgeInsets.zero,
        minimumSize: Size(double.infinity, blockHeight),
      ),
      child: Center(child: Text(title.tr(), style: fontStyleBodyWhite)),
    );
  }
}
```

6.4.7 ButtonGrayWidget

Al igual que *ButtonBlackWidget*, se encarga de la interfaz del botón y de su funcionalidad, pero con una diferencia, se necesitan aportar más parámetros al llamarlo. Esto parámetros se encargan de la colocación de un icono y un texto que irán variando según si el ajuste que controla el botón está activado o desactivado, así como de un ajuste para el tamaño del icono proporcionado, ya que visualmente algunos pueden reflejar ser más pequeños.

Código 6.18 ButtonGrayWidget

```
class ButtonGrayWidget extends StatefulWidget {
  final String titleOn;
  final String titleOff;
  final IconData iconOn;
  final IconData iconOff;
  final double iconBiggerBy;
  bool setting;
  final Function() onPressed;

  ButtonGrayWidget({
    super.key,
    required this.titleOn,
    required this.titleOff,
    required this.iconOn,
    required this.iconOff,
    this.iconBiggerBy = 0,
    required this.setting,
    required this.onPressed,
  });

  @override
  ButtonGrayWidgetState createState() => ButtonGrayWidgetState();
}

class ButtonGrayWidgetState extends State<ButtonGrayWidget> {
  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        SizedBox(
          height: blockHeight,
          child: OutlinedButton(
            onPressed: () {
              vibration();
              widget.setting = !widget.setting;
              widget.onPressed();
              setState(() {});
            },
            style: OutlinedButton.styleFrom(
              side: BorderSide(color: colorsecondary,
                width: blockBorder),
              shape: RoundedRectangleBorder(
                borderRadius: BorderRadius.circular(borderRadiusBig),
              ),
              padding: EdgeInsets.zero,
              minimumSize: Size(double.infinity, blockHeight),
            ),
```

```
                child: Row(
                  children: [
                    SizedBox(
                      width: iconBlockSize,
                      child: Icon(
                        widget.setting ? widget.iconOn : widget.iconOff,
                        size: iconSize + widget.iconBiggerBy,
                        color: colorprimary,
                      ),
                    ),
                    Text(
                      widget.setting ? widget.titleOn.tr()
                        : widget.titleOff.tr(),
                      style: fontStyleBodyWhite,
                    ),
                  ],
                ),
              ),
            ),
            SizedBox(height: marginMedium),
          ],
        );
    }
}
```

6.4.8 ButtonLoginEmailWidget

Este widget está encargado de proveer al usuario con el servicio de inicio de sesión y de registro. Para poder distinguir entre estas funcionalidades, se requiere proporcionar un parámetro booleano *isLoginPage*, el cual será verdadero si se requiere la funcionalidad de inicio de sesión y falso si es la de registro. Esta distinción varía los comandos ejecutados con respecto a *Authentication.*

En el caso que el usuario pulse el botón de inicio de sesión / registro sin haber rellenado los campo de correo electrónico y/o contraseña, se muestra por pantalla un diálogo de error. Si por el contrario el usuario ha rellenado los datos correctamente pero no ha completado el formulario de recopilación de datos, se le redirigirá a esta pantalla en vez de a la pantalla de juego.

Código 6.19 ButtonLoginEmailWidget

```
class ButtonLoginEmailWidget extends StatelessWidget {
  final bool isLoginPage;
  final String title;
  final TextEditingController emailController;
  final TextEditingController passwordController;

  const ButtonLoginEmailWidget({
    super.key,
    required this.isLoginPage,
    required this.title,
    required this.emailController,
    required this.passwordController,
  });

  @override
  Widget build(BuildContext context) {
    return OutlinedButton(
```

```
            onPressed: () async {
              vibration();
              FocusManager.instance.primaryFocus?.unfocus();
              String email = emailController.text;
              String password = passwordController.text;
              if (email.isNotEmpty && password.isNotEmpty) {
                try {
                  isLoginPage
                       ? await Auth().signInWithEmailAndPassword(
                          email: email,
                          password: password,
                       )
                       : await Auth().createUserWithEmailAndPassword(
                          email: email,
                          password: password,
                       );
                  if (FirebaseAuth.instance.currentUser != null) {
                    isLoggedIn = true;
                    if ((await FirebaseFirestore.instance
                              .collection('usuarios')
                              .doc(FirebaseAuth.instance.currentUser!.uid)
                              .get())
                         .exists) {
                      Navigator.pushNamed(context, '/gameScreen');
                    } else {
                      Navigator.pushNamed(context, '/data_collector');
                    }
                  }
                } catch (e) {
                  dialogError(context, e.toString());
                }
              } else {
                dialogError(context, 'error_empty_email_password'.tr());
              }
            },
            style: OutlinedButton.styleFrom(
              backgroundColor: colorbutton,
              side: BorderSide(color: colorbutton, width: blockBorder),
              shape: RoundedRectangleBorder(
                borderRadius: BorderRadius.circular(borderRadiusBig),
              ),
              padding: EdgeInsets.zero,
              minimumSize: Size(double.infinity, blockHeight),
            ),
            child: Center(child: Text(title.tr(), style: fontStyleBodyWhite)),
          );
        }
      }
```

6.4.9 ButtonLogInProvidersWidget

Se encarga de mostrar en la pantalla de bienvenida el botones para el inicio de sesión el proveedor habilitado. Más información sobre este tema en Apartado 5.2.

Para evitar que los usuarios que hayan iniciado sesión puedan evitar la pantalla de recogida de datos, se ha añadido un condicionante a este widget. Antes de redirigirlos a la pantalla de juego, se comprueba que exista una entrada en la base de datos identificada con su *uid*, ya que para existir es necesario que estos hayan rellenado

y enviado correctamente el formulario de recopilación de datos.

Código 6.20 ButtonLogInProvidersWidget

```
class ButtonLogInProvidersWidget extends StatefulWidget {
  final String title;
  final String icon;
  final double iconHeight;
  final Function() onPressed;

  const ButtonLogInProvidersWidget({
    super.key,
    required this.title,
    required this.icon,
    required this.iconHeight,
    required this.onPressed,
  });

  @override
  _ButtonLogInProvidersWidgetState createState() =>
      _ButtonLogInProvidersWidgetState();
}

class _ButtonLogInProvidersWidgetState
    extends State<ButtonLogInProvidersWidget> {
  FirebaseAuth auth = FirebaseAuth.instance;

  @override
  Widget build(BuildContext context) {
    return SizedBox(
      height: blockHeight,
      child: OutlinedButton(
        onPressed: () async {
          vibration();
          var navigator = Navigator.of(context);
          await widget.onPressed();
          if (Auth.currentUser != null) {
            if ((await FirebaseFirestore.instance
                        .collection('usuarios')
                        .doc(auth.currentUser!.uid)
                        .get())
                    .exists) {
              navigator.pushNamed('/gameScreen');
            } else {
              navigator.pushNamed('/data_collector);
            }
          }
        },
        style: OutlinedButton.styleFrom(
          side: BorderSide(color: colorsecondary, width: blockBorder),
          shape: RoundedRectangleBorder(
            borderRadius: BorderRadius.circular(borderRadiusBig),
          ),
          padding: EdgeInsets.zero,
          minimumSize: Size(double.infinity, blockHeight),
        ),
        child: Row(
```

```
                children: [
                  SizedBox(
                    width: iconBlockSize,
                    child: Image.asset(widget.icon,
                      height: widget.iconHeight),
                  ),
                  Text(widget.title.tr(), style: fontStyleBodyWhite),
                ],
              ),
            ),
          );
      }
    }
```

6.4.10 ButtonLogOutWidget

Como se comentó en el Apartado 6.3.6, al ser pulsado el botón, tras asegurándose que existe un *uid* y para contrarrestar las posibles pulsaciones accidentales, se muestra por pantalla un mensaje de confirmación, tras la cual, y si se ha pulsado la opción para continuar, procede a cerrar la sesión activa del usuario. Esta tarea es encargada a *DialogshowLogout*, Código 6.26 En el caso de no encontrar un *uid*, muestra un mensaje de error explicando esta situación, Código 6.25.

Código 6.21 ButtonLogOutWidget

```
class ButtonLogOutWidget extends StatelessWidget {
  final user = FirebaseAuth.instance.currentUser;

  ButtonLogOutWidget({super.key});

  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        SizedBox(
          height: blockHeight,
          child: OutlinedButton(
            onPressed: () {
              vibration();
              if (user != null) {
                DialogshowLogout(context);
              } else {
                dialogError(context, 'error_not_logged_in');
              }
            },
            style: OutlinedButton.styleFrom(
              backgroundColor: colorsecondary,
              side: BorderSide(color: colorsecondary,
                width: blockBorder),
              shape: RoundedRectangleBorder(
                borderRadius: BorderRadius.circular(borderRadiusBig),
              ),
              padding: EdgeInsets.zero,
              minimumSize: Size(double.infinity, blockHeight),
            ),
            child: Center(
```

```
                    child: Text('button_sign_out'.tr(),
                      style: fontStyleBodyWhite),
                  ),
                ),
              ),
              SizedBox(height: marginMedium),
            ],
          );
        }
      }
```

6.4.11 ButtonOutlinedWidget

Para representar por pantalla los datos que el usuario debe introducir en la pantalla de recopilación de datos, Figura 6.5, se ha diseñado este widget. Es necesario aportar un título, texto, icono y función al presionar el botón. El widget crea un título que encabeza el botón para poner en contexto al usuario sobre la funcionalidad de este.

Código 6.22 ButtonOutlinedWidget

```
class ButtonOutlinedWidget extends StatelessWidget {
  final IconData icon;
  final String title;
  final String text;
  final Function(String) onPressed;

  const ButtonOutlinedWidget({
    super.key,
    required this.icon,
    required this.title,
    required this.text,
    required this.onPressed,
  });

  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        SizedBox(
          height: blockTitleHeight,
          child: Align(
            alignment: Alignment.centerLeft,
            child: Text(
              title.tr(),
              style: fontStyleBodyWhite,
              textAlign: TextAlign.left,
            ),
          ),
        ),
        OutlinedButton(
          onPressed: () async {
            vibration();
            String? result;
            switch (title) {
              case 'text_education_level':
                result = await bottomSheetEducationLevel(context);
```

```
                    break;
                  case 'text_play_time':
                    result = await bottomSheetPlayTime(context);
                    break;
                  case 'text_country':
                    result = await botomSheetCountryPicker(context);
                    break;
                }
                if (result != null) {
                  onPressed(result);
                }
              },
              style: OutlinedButton.styleFrom(
                side: BorderSide(color: colorsecondary, width: blockBorder),
                shape: RoundedRectangleBorder(
                  borderRadius: BorderRadius.circular(borderRadiusBig),
                ),
                padding: EdgeInsets.zero,
                minimumSize: Size(double.infinity, blockHeight),
              ),
              child: Row(
                children: [
                  SizedBox(
                    width: iconBlockSize,
                    child: Icon(icon, size: iconSize, color: colorprimary),
                  ),
                  Text(
                    text.tr(),
                    style:
                        (text == 'text_input_country' ||
                          text == 'text_input_education_level' ||
                          text == 'text_input_play_time')
                          ? fontStyleBodyGray
                          : fontStyleBodyWhite,
                  ),
                ],
              ),
            ),
            SizedBox(height: marginSmall),
          ],
        );
    }
  }
```

6.4.12 ButtonOutlinedCenteredWidget

Este widget es semejante al anterior, pero con una diferencia principal, el texto está situado en el centro del botón. Este cambio no supone suficiente diferencia para la creación de un widget nuevo, pero si al añadir la eliminación del icono, así como la del título superior.

Código 6.23 ButtonOutlinedCenteredWidget

```
  class ButtonOutlinedCenteredWidget extends StatelessWidget {
    final String title;
    final Function() onPressed;
```

```
    double width;

    ButtonOutlinedCenteredWidget({
      super.key,
      required this.title,
      required this.onPressed,
      this.width = 0.0,
    });

    @override
    Widget build(BuildContext context) {
      return Column(
        children: [
          SizedBox(
            height: blockHeight,
            width: width == 0.0 ? width = anchoPantallaUsable(context)
              : width,
            child: OutlinedButton(
              onPressed: () {
                vibration();
                onPressed();
              },
              style: OutlinedButton.styleFrom(
                side: BorderSide(color: colorsecondary,
                  width: blockBorder),
                shape: RoundedRectangleBorder(
                  borderRadius: BorderRadius.circular(borderRadiusBig),
                ),
                padding: EdgeInsets.zero,
                minimumSize: Size(double.infinity, blockHeight),
              ),
              child: Center(child: Text(title.tr(),
                style: fontStyleBodyWhite)),
            ),
          ),
        ],
      );
    }
  }
```

6.4.13 CoachMark

El siguiente widget se ha creado para agilizar la creación de los mensajes que aparecen en la pantalla durante el tutorial. Muestra un rectángulo de bordes redondeados con un texto que se adjunta al llamar a la función. También se pueden indicar funciones para los botones para avanzar y desactivar el tutorial.

Código 6.24 CoachMark

```
  class CoachMark extends StatefulWidget {
    CoachMark({
      super.key,
      required this.text,
      this.onSkip,
      this.onNext,
      this.isLast,
```

```
    });

    final String text;
    final void Function()? onSkip;
    final void Function()? onNext;
    bool? isLast = false;
    @override
    State<CoachMark> createState() => _CoachMarkState();
  }

  class _CoachMarkState extends State<CoachMark> {
    @override
    Widget build(BuildContext context) {
      return Padding(
        padding: const EdgeInsets.only(left: marginMedium,
          right: marginMedium),
        child: Container(
          width: 50,
          padding: const EdgeInsets.all(marginMedium),
          decoration: BoxDecoration(
            color: colorbackground,
            borderRadius: BorderRadius.circular(borderRadiusBig),
          ),
          child: Column(
            mainAxisSize: MainAxisSize.min,
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              Text(widget.text, style: fontStyleBodyWhite),
              const SizedBox(height: 10),
              Row(
                mainAxisAlignment: MainAxisAlignment.end,
                children: [
                  if (widget.isLast == false)
                    TextButton(
                      onPressed: () {
                        widget.onSkip!();
                        isGameTutorialActive = false;
                      },
                      child: Text("button_exit".tr(), style:
fontStyleBodyGray),
                    ),
                  TextButton(
                    onPressed: widget.onNext,
                    child: Text(
                      (widget.isLast!
                          ? "button_finish".tr()
                          : "button_continue".tr()),
                      style: fontStyleBodyWhite,
                    ),
                  ),
                ],
              ),
            ],
          ),
        ),
      );
    }
  }
```

6.4.14 DialogError

Este widget está encargado de mostrar por pantalla los distintos errores con los que se puede encontrar un usuario. Cuenta con un botón en la parte inferior derecha para cerrar el cuadro de diálogo. El texto mostrado deberá ser dado como parámetro al momento de la llamada del widget.

Código 6.25 DialogError

```
Future dialogError(BuildContext context, String error) async {
  showDialog<void>(
    context: context,
    builder: (BuildContext context) {
      return AlertDialog(
        title: Text(error.tr(), style: fontStyleBodyWhite),
        backgroundColor: colorbackground,
        actions: <Widget>[
          TextButton(
            child: Text('button_back'.tr(), style: fontStyleBodyWhite),
            onPressed: () {
              vibration();
              isLoggedIn = false;
              Navigator.of(context).pop();
            },
          ),
        ],
      );
    },
  );
}
```

6.4.15 DialogShowLogout

Widget muy parecido *DialogError* pero en este caso se ha configurado para el servicio de cierre de sesión. Muestra un pequeño texto que se puede ver en la Figura 6.16, seguido de dos botones, el primero para volver hacia atrás y el segundo para proceder con el cierre de sesión.

Código 6.26 DialogshowLogout

```
void dialogshowLogout(BuildContext context) {
  showDialog<void>(
    context: context,
    builder: (BuildContext context) {
      return AlertDialog(
        title: Text('title_logout'.tr(), style: fontStyleBodyWhite),
        content: Text('subtitle_logout'.tr(), style: fontStyleBodyWhite),
        backgroundColor: colorbackground,
        actions: <Widget>[
          TextButton(
            onPressed: () {
            vibration();
              Navigator.of(context).pop();
            },
```

```
                child: Text('button_cancel'.tr(), style: fontStyleBodyGray),
              ),
              TextButton(
                child: Text('button_continue'.tr(),
                  style: fontStyleBodyWhite),
                onPressed: () async {
                  vibration();
                  final navigator = Navigator.of(context);

                  for (var providerProfile in user!.providerData) {
                    switch (providerProfile.providerId) {
                      case 'password':
                        await FirebaseAuth.instance.signOut();
                        break;
                      case 'google.com':
                        await Provider.of<GoogleSignInProvider>(
                          context,
                          listen: false,
                        ).logout();
                        break;
                    }
                  }
                  isLoggedIn = false;
                  navigator.pop();
                  navigator.pushNamedAndRemoveUntil(
                    '/',
                    (Route<dynamic> route) => false,
                  );
                },
              ),
            ],
          );
        },
      );
    }
```

6.4.16 GenderSelectionWidget

Como se ha visto en la Figura 6.5, en la pantalla de recogida de datos se muestra un selector triple para elegir el género. Este widget ha configurado para este propósito. Se trata de 3 botones de los cuales solo uno puede estar activo, esto se realiza de forma automática gracias a la variable *selectedGender*. Una vez realizado el cambio de opción, el botón seleccionado cambia de color blanco, mientras que el resto cambia a color gris. Este cambio permite una interacción más visual con el usuario, permitiéndole reconocer la opción pulsada de forma rápida.

Código 6.27 GenderSelectionWidget

```
class GenderSelectionWidget extends StatelessWidget {
  final String selectedGender;
  final Function(String) onGenderSelected;

  const GenderSelectionWidget({
    super.key,
    required this.selectedGender,
    required this.onGenderSelected,
  });
```

```
@override
Widget build(BuildContext context) {
   return Column(
      children: [
         SizedBox(
            height: blockTitleHeight,
            child: Align(
               alignment: Alignment.centerLeft,
               child: Text(
                  'text_gender'.tr(),
                  style: fontStyleBodyWhite,
                  textAlign: TextAlign.left,
               ),
            ),
         ),
         Row(
            children: [
               _buildGenderButton('text_gender_male'),
               SizedBox(width: marginSmall),
               _buildGenderButton('text_gender_female'),
               SizedBox(width: marginSmall),
               _buildGenderButton('text_gender_other'),
            ],
         ),
      ],
   );
}

Widget _buildGenderButton(String gender) {
   return Expanded(
      child: SizedBox(
         height: blockHeight,
         child: OutlinedButton(
            onPressed: () {
               vibration();
               onGenderSelected(gender);
            },
            style: OutlinedButton.styleFrom(
               side: BorderSide(
                  color: selectedGender == gender ? colorprimary
                     : colorsecondary,
                  width: blockBorder,
               ),
               shape: RoundedRectangleBorder(
                  borderRadius: BorderRadius.circular(borderRadiusBig),
               ),
            ),
            child: Text(
               gender.tr(),
               style:
                    selectedGender == gender
                         ? fontStyleBodyWhite
                         : fontStyleBodyGray,
               textAlign: TextAlign.center,
            ),
         ),
      ),
   );
```

```
    }
  }
```

6.4.17 SeparatorWidget

Existe un elemento en la pantalla principal que se encarga de separar visualmente el inicio de sesión con email y el inicio de sesión con *Google*. Este widget es el encargado de construirlo y mostrarlo por pantalla.

Código 6.28 SeparatorWidget

```
class SeparatorWidget extends StatelessWidget {
  final String text;
  const SeparatorWidget({super.key, this.text = ''});

  @override
  Widget build(BuildContext context) {
    return Row(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Container(
          height: blockBorder,
          width: blockSpliterHeight,
          color: colorsecondary,
        ),
        Text(text.tr(), style: fontStyleBodyWhite),
        Container(
          height: blockBorder,
          width: blockSpliterHeight,
          color: colorsecondary,
        ),
      ],
    );
  }
}
```

6.4.18 TextInputWidget

Los usuarios necesitan poder ingresar ciertos datos, en algunos casos estos datos pueden ser introducidos en forma de opciones. Cuando no puede ser así, como es el caso de la pantalla de inicio de sesión o la pantalla de registro, se necesita una manera de mostrar el teclado en la pantalla y recopilar los datos escritos por los usuarios. Tanto el campo de correo como el de contraseña tienen esta necesidad, por ello se han juntado en esta clase diferenciándose solo por el parámetro *isPassword*, el cual modifica la funcionalidad para acometer datos sensibles y ocultarlos a la vista. En el caso que se quiera ver la contraseña introducida, se ha colocado un botón a la derecha del widget que la muestra al ser pulsado, Figura 6.3 y Figura 6.4.

Código 6.29 TextInputWidget

```
class TextInputWidget extends StatefulWidget {
  final String title;
  final IconData icon;
  final bool isPassword;
  final String text;
```

```
    final bool isInputNumber;
    final TextEditingController controller;

    const TextInputWidget({
      super.key,
      this.title = '',
      this.isPassword = false,
      this.isInputNumber = false,
      required this.icon,
      required this.text,
      required this.controller,
    });

    @override
    State<TextInputWidget> createState() => _TextInputWidgetState();
  }

  class _TextInputWidgetState extends State<TextInputWidget> {
    bool _isPasswordVisible = false;

    @override
    Widget build(BuildContext context) {
      return Column(
        children: [
          if (widget.title != '')
            SizedBox(
              height: blockTitleHeight,
              child: Align(
                alignment: Alignment.centerLeft,
                child: Text(
                  widget.title.tr(),
                  style: fontStyleBodyWhite,
                  textAlign: TextAlign.left,
                ),
              ),
            ),
          Container(
            height: blockHeight,
            decoration: BoxDecoration(
              border: Border.all(color: colorsecondary,
                width: blockBorder),
              borderRadius: BorderRadius.circular(borderRadiusBig),
            ),
            child: Row(
              children: [
                SizedBox(
                  width: iconBlockSize - blockBorder,
                  child: Icon(widget.icon, size: iconSize,
                    color: colorprimary),
                ),
                Expanded(
                  child: TextFormField(
                    controller: widget.controller,
                    keyboardType:
                        widget.isInputNumber
                            ? TextInputType.number
                            : TextInputType.emailAddress,
                    obscureText: widget.isPassword
                      ? !_isPasswordVisible : false,
```

```
                        inputFormatters:
                            widget.isInputNumber
                                ? <TextInputFormatter>[
                                    FilteringTextInputFormatter.digitsOnly,
                                    FilteringTextInputFormatter
                                    .deny(RegExp(r'\s')),
                                ]
                                : <TextInputFormatter>[
                                    FilteringTextInputFormatter
                                        .singleLineFormatter,
                                    FilteringTextInputFormatter
                                        .deny(RegExp(r'\s')),
                                ],
                        decoration: InputDecoration(
                          hintText: widget.text.tr(),
                          hintStyle: fontStyleBodyGray,
                          border: InputBorder.none,
                        ),
                        style: fontStyleBodyWhite,
                      ),
                    ),
                    if (widget.isPassword == true)
                      GestureDetector(
                        behavior: HitTestBehavior.translucent,
                        onTap: () {
                          setState(() {
                            _isPasswordVisible = !_isPasswordVisible;
                          });
                        },
                        child: SizedBox(
                          width: iconBlockSize - blockBorder,
                          height: blockHeight,
                          child: Icon(
                            _isPasswordVisible ? LucideIcons.eye
                                : LucideIcons.eyeOff,
                            size: iconSize,
                            color: colorprimary,
                          ),
                        ),
                      ),
                  ],
                ),
              ),
            ],
          );
        }
    }
```

6.4.19 TitleWidget

Al comienzo de cada pantalla se ha colocado un título y un subtítulo que indican al usuario la pantalla en la que se encuentra. Tras recibir estos parámetros de texto, esta clase está encargada de colocarlos con el estilo, el color y el tamaño de letra particular de cada elemento.

Código 6.30 TitleWidget

```
class TitleWidget extends StatelessWidget {
  final String title;
  final String subtitle;

  const TitleWidget({super.key, required this.title, this.subtitle = ''});

  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        SizedBox(height: marginBig - marginMedium),
        ConstrainedBox(
          constraints: BoxConstraints(minHeight: blockTitleHeight),
          child: SizedBox(
            child: ConstrainedBox(
              constraints: BoxConstraints(minHeight: 10),
              child: Text(
                title.tr(),
                style: fontStyleTitle,
                textAlign: TextAlign.center,
              ),
            ),
          ),
        ),
        if (subtitle != '') ...[
          SizedBox(
            height: blockTitleHeight,
            width: blockLengthSubtitle,
            child: Text(
              subtitle.tr(),
              style: fontStyleSubtitle,
              textAlign: TextAlign.center,
            ),
          ),
        ],
      ],
    );
  }
}
```

6.4.20 TitleSplitWidget

Como se comentó en el Apartado 6.3.1, algunos dispositivos móviles se encuentran más restringidos en cuanto al tamaño de pantalla. Debido a esto los títulos largos como el presente en la pantalla de bienvenida pueden ser deformados. Para solucionar con este problema, el texto es dividido en dos filas, aumentando el espacio vertical que ocupa.

Código 6.31 TitleSplitWidget

```
class TitleSplitWidget extends StatelessWidget {
  final String titleSplit1;
  final String titleSplit2;
  final String subtitle;
```

```
    const TitleSplitWidget({
      super.key,
      required this.titleSplit1,
      required this.titleSplit2,
      required this.subtitle,
    });

    @override
    Widget build(BuildContext context) {
      return Column(
        children: [
          SizedBox(height: marginBig - marginMedium),
          SizedBox(
            child: ConstrainedBox(
              constraints: BoxConstraints(minHeight: 10),
              child: Text(
                titleSplit1.tr(),
                style: fontStyleTitle,
                textAlign: TextAlign.center,
              ),
            ),
          ),
          SizedBox(
            child: ConstrainedBox(
              constraints: BoxConstraints(minHeight: 10),
              child: Text(
                titleSplit2.tr(),
                style: fontStyleTitle,
                textAlign: TextAlign.center,
              ),
            ),
          ),
          SizedBox(
            height: blockTitleHeight,
            width: blockLengthSubtitle,
            child: Text(
              subtitle.tr(),
              style: fontStyleSubtitle,
              textAlign: TextAlign.center,
            ),
          ),
        ],
      );
    }
  }
```

7 PRUEBAS REALIZADAS

Una vez configurados ambos servicios de *Firebase*, se procede a la realización pruebas uso, tanto del servicio de inicio de sesión como del de guardado de datos del usuario. Estas pruebas de funcionamiento de la aplicación se han realizado en distintos dispositivos móviles para asegurar su correcto funcionamiento en una distintos escenarios.

En la Figura 7.1, se muestra el resultado de una prueba de juego y en la Figura 7.2 el resultado de un intento de cierre de sesión, donde no se había iniciado sesión con anterioridad, provocando un mensaje de error del sistema.

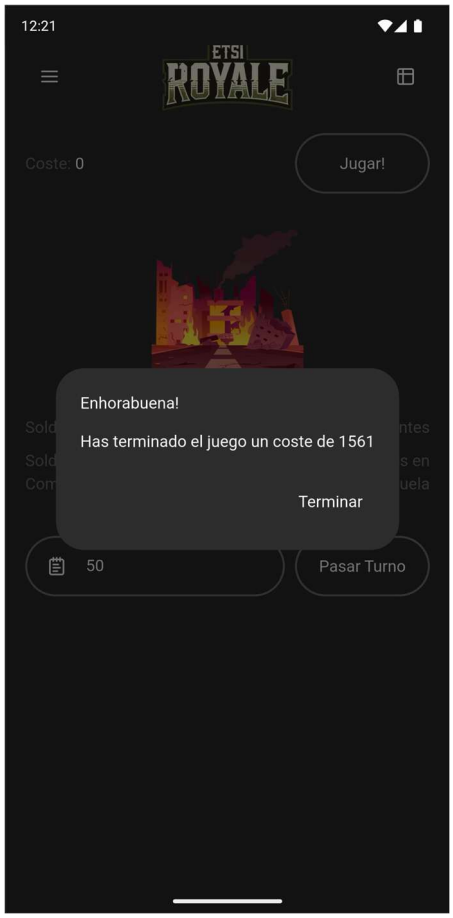

Figura 7.1 Prueba de Juego

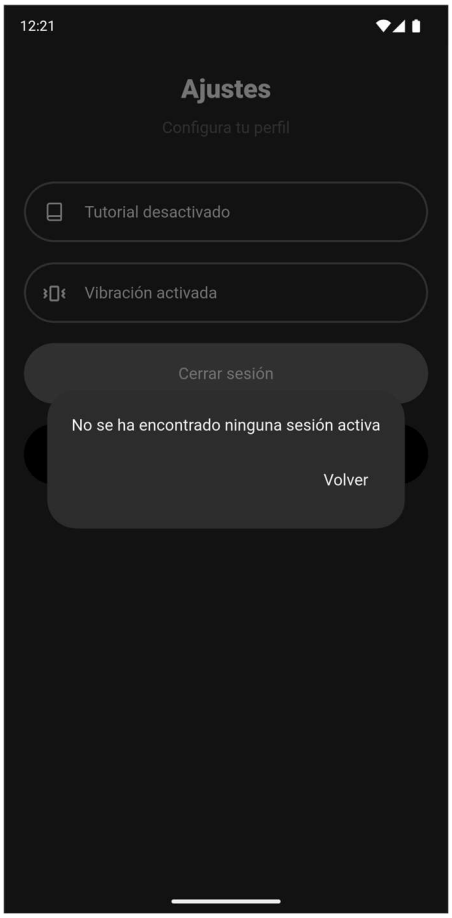

Figura 7.2 Prueba de Cierre de Sesión

Como se puede ver en las siguientes figuras, los servicios de inicio de sesión, registro y guardado de datos funcionan correctamente. *Authentication* guarda los datos de inicio de sesión y genera un *uid* único para cada usuario, Figura 7.3.

Una vez que se ha iniciado sesión, los datos del usuario se guardan en *Firestore* con la estructura establecida. Dentro del documento específico referido con el *uid* del usuario se encuentran los distintos campos rellenados

en la pantalla de recopilación de datos, Figura 7.4, y dentro de la colección de partidas se encuentran los resultados obtenidos por el usuario, Figura 7.5. Como se comentó en el Apartado 5.3.3, la partida 0 es creada para evitar la eliminación automática de esta colección por parte de *Firestore.* Los siguientes documentos ya contienen los datos recopilados durante las partidas jugadas, añadiendo además datos sobre la demanda satisfecha, la producción y la puntuación obtenida.

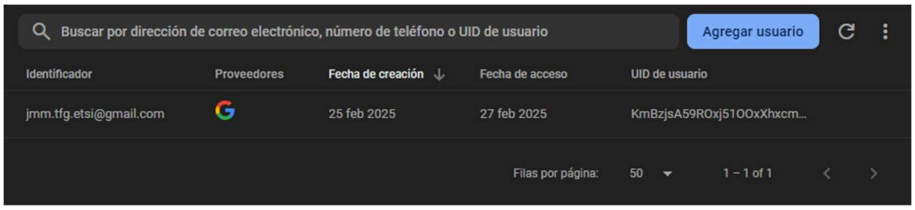

Figura 7.3 Usuarios Registrados en *Authentication*

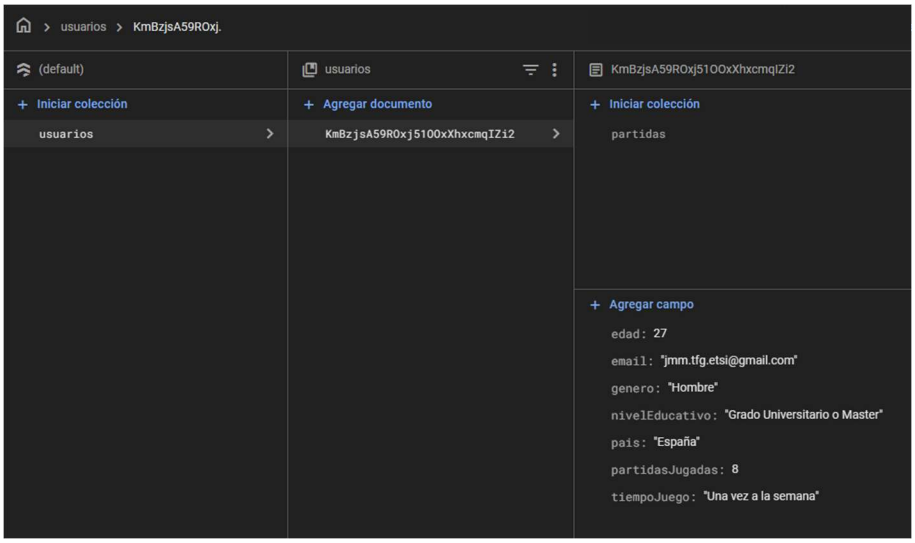

Figura 7.4 Datos del Usuario en *Firestore*

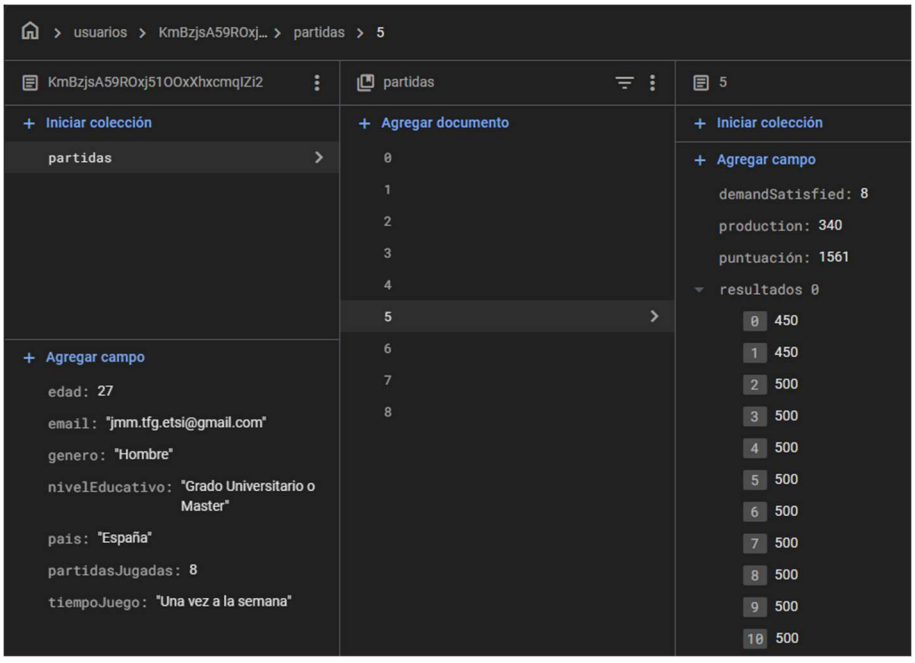

Figura 7.5 Resultados de las partidas en *Firestore*

Tras esto se han recopilado los valores resultantes de una de las partidas jugadas en la que se utilizó los mismos valores que John D. Sterman en su paper [3]. Estos valores han resultado en la siguiente tabla. Se han resaltado los valores no coincidentes con los resultados del algoritmo original.

Año	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	450	450	50	500	500	50	500	100	450	50	50
2	450	450	50	500	500	50	500	100	450	50	50
4	500	500	**60**	**560**	500	50	500	**89**	450	50	60
6	500	550	**80**	**630**	500	50	500	**79**	**430**	**70**	70
8	500	**620**	**90**	**710**	**520**	50	**520**	**73**	**450**	**70**	80
10	500	**670**	**110**	**780**	**540**	50	**540**	**69**	**460**	**80**	90
12	500	**710**	**170**	**880**	**570**	**60**	**570**	**65**	**460**	**110**	140
14	500	**750**	**200**	950	**620**	60	**620**	**65**	**490**	**130**	140
16	500	**760**	210	**970**	**690**	**70**	**690**	**71**	**540**	**150**	140
18	500	**720**	**210**	**930**	**770**	**80**	**770**	**83**	**600**	**170**	150
20	500	**620**	**180**	**800**	**860**	**90**	**800**	**100**	**620**	**180**	140
22	500	**500**	**130**	**630**	**950**	**100**	**630**	**100**	**500**	**130**	130
24	500	**500**	**120**	**620**	**980**	**100**	**620**	100	**500**	**120**	120
26	500	500	**0**	**500**	1000	100	**500**	100	500	**0**	0
28	500	500	0	500	**900**	**90**	500	100	500	0	0
30	500	500	0	500	**810**	**80**	500	100	500	0	0
32	500	500	0	500	**730**	**70**	500	100	500	0	0
34	500	500	**30**	**530**	**660**	**70**	**530**	100	500	**30**	30
36	500	500	**20**	**520**	**620**	**60**	**520**	100	500	**20**	20
38	500	500	**40**	**540**	**580**	60	**540**	100	500	**40**	40
40	500	500	**50**	**550**	**560**	60	**550**	100	500	**50**	50
42	500	500	50	550	**550**	60	550	100	500	50	50
44	500	500	**60**	**560**	**540**	**50**	**540**	**96**	**480**	**60**	60
46	500	**520**	**50**	**570**	550	60	550	**96**	**500**	**50**	50
48	500	**520**	**60**	**580**	**540**	**50**	**540**	**93**	**480**	**60**	60
50	500	**540**	60	**600**	**550**	**60**	**550**	**92**	**500**	**50**	60
52	500	**540**	**90**	**630**	**540**	**50**	**540**	**86**	**460**	**80**	80
54	500	**580**	**90**	**670**	**570**	**60**	**570**	**85**	**490**	**80**	80
56	500	**590**	**100**	**690**	**590**	60	**590**	**86**	**510**	80	90
58	500	580	**130**	**710**	**610**	60	**610**	**86**	**500**	**110**	110
60	500	**580**	**110**	**690**	**660**	**70**	**660**	**96**	**560**	100	90
62	500	**520**	**80**	**600**	**690**	70	**600**	**100**	**520**	**80**	70
64	500	**500**	**60**	**560**	**700**	**70**	**560**	**100**	**500**	**60**	60
66	500	**500**	**0**	**500**	**690**	70	**500**	**100**	**500**	**0**	0

68	500	500	0	500	**620**	**60**	500	100	500	0	0
70	500	500	**60**	**560**	**560**	**60**	**560**	100	500	**60**	60

Tabla 7.1 Resultados del Algoritmo en Dart

Como se puede observar, los resultados distan de los obtenidos por John D. Sterman en sus pruebas. Esto confirma la teoría mencionada en el Apartado 3.1, que exponía la complicación a la hora de reproducir a la perfección el sistema de redondeo empleado en el programa desarrollado en *Basic*.

Tras analizar detenidamente el algoritmo utilizado se ha determinado que el causante es, efectivamente, el redondeo de valores. Se han probado múltiples opciones para intentar solventar este problema, pero ninguna ha conseguido reproducir fielmente los resultados publicados en el paper [3].

Esto pone de manifiesto la importancia de considerar las particularidades del entorno de trabajo, tanto para desarrollar una aplicación original como al intentar replicar otros modelos computacionales. Aunque parezcan insignificantes, los detalles son los que marcan la diferencia.

En definitiva, aunque se ha logrado aproximarse parcialmente a los resultados originales, y se ha comprobado que el resto de sistemas de la aplicación funcionan correctamente, la imposibilidad de replicarlos exactamente resalta la sensibilidad del modelo a pequeños cambios, incluso tratándose de una versión simplificada del algoritmo.

8 CONCLUSIÓN

Concluido con el diseño y la programación de la aplicación, se ha realizado un conjunto de pruebas para corroborar su correcto funcionamiento. La aplicación desarrollada ha sido probada en distintos dispositivos, los cuales contaban con diversas resoluciones de pantalla, lo que ha permitido identificar errores que no se mostraban en el dispositivo emulado.

Un ejemplo de ellos es la ocultación del título de la pantalla de bienvenida, Figura 6.1, el cual aparecía dividido en dos filas de manera aleatoria. Esto se solventó estableciendo unos límites de ejecución en función del tamaño de la pantalla del dispositivo.

La comunicación entre la aplicación y la base de datos también ha sido la causante de algunos problemas. Ya que se trataba de un sistema de configuración nuevo para mí, debía aprender como establecer las comunicación entre ellos.

Ha habido situaciones donde una línea de código ha provocado un fallo en cadena. Esto ha ocurrido debido a que *algorithm.year += algorithm.yearAdd* dentro de Código 6.9 estaba comentada, provocando que el algoritmo no avanzara y repitiera el año 0 de forma continuada. También causaba que la pantalla de visionado de resultados se quedara vacía, ya que no existían datos para representar, la matriz de resultados no mostrara nada.

Aunque todo esto no habría sido posible si no se hubiera arreglado el primer obstáculo. Antes incluso de empezar a programar la aplicación, Flutter no podía ejecutarse. Esto se debía a un problema con el *jdk*, existe un error en el que Android Studio tiene problemas con las versiones superiores a la 17, por lo que al desinstalar la versión más actual y sustituirla por esta, el problema se resolvió.

Uno de los errores que no se ha podido solucionar ha sido el de la interpretación exacta al algoritmo de Sterman original. Este algoritmo al estar programado en *Basic* cuenta con un sistema de redondeo de valores propio del lenguaje, lo cual causa que su traducción entre lenguajes sea difícil y, en este caso, no se haya logrado conseguir una traducción exacta.

8.1 Futuras Líneas de Desarrollo

Aunque la aplicación creada funciona correctamente, el algoritmo usado no es fiel al creado por John D. Sterman, por lo que una mejora en este aspecto sería necesaria.

Otro de los aspectos a desarrollar podría ser la exportación de la aplicación al ecosistema de *Apple*, facilitando así el acceso a los usuarios que dispongan de un dispositivo *iPhone*. Aunque la cuota de mercado de *Apple* en cuanto a móviles es reducida, en comparación con *Android*, en otros mercados como el estadounidense podría dar a una mayor acogida por parte del público.

Esto muestra otra de las mejoras que podría darse, la traducción a otros idiomas. Empezando por el inglés, dada su importancia global, y siguiendo por otros con mayor cantidad de hablantes, como el chino. El sistema para la fácil implementación de idiomas está configurado, ya que es el método con el que se cargan los texto de toda la aplicación, mensajes de error incluidos, y no supondría una gran dificultad añadir uno nuevo. El problema residiría en la configuración de este. Lo ideal sería la detección automática del idioma utilizado en el dispositivo, evitando así que el usuario necesitara realizar el cambio, pero esto podría resultar un problema mayor a causa de los permisos que requeriría la aplicación a la hora de ser instalada. También sería necesaria una modificación de la pantalla de ajustes para permitir el cambio de idioma, pudiendo ser distinto al del dispositivo, y recordando la elección una vez la aplicación se reinicie.

Una idea para incentivar la adopción masiva de la aplicación sería la de introducir un sistema de clasificación, donde se comparara la puntuación obtenida por los distintos usuarios, publicando una lista con los mejores resultados, facilitando así la competición entre los mismos.

Claro que esto resultaría más entretenido para la mayoría de jugadores si se cambiara el sistema de juego de estar basado en turnos a un juego en tiempo real, donde se pudieran ver los cambios hechos sin la necesidad de estar limitado por una serie de turnos.

9 REFERENCIAS

[1] S. University, «Mapping the Landscape of Digital Game-Based Learning in Swedish Compulsory and Upper Secondary Schools: Opportunities and Challenges for Teachers,» [En línea]. Available: https://su.diva-portal.org/smash/record.jsf?pid=diva2%3A1425997. [Último acceso: 4 Mayo 2025].

[2] J. D. Sterman, A simple model of the economic long wave, Cambridge: Massachusetts Institute of Technology, 1983.

[3] J. D. Sterman, STRATEGE-2: A Microcomputer Simulation Game of the Kondratiev Cycle, Cambridge: Massachusetts Institute of Technology, 1995.

[4] Android Developers, «Kotlin,» [En línea]. Available: https://developer.android.com/kotlin. [Último acceso: 12 Enero 2025].

[5] Android Developers, «Cómo brindar compatibilidad con diferentes densidades de píxeles,» [En línea]. Available: https://developer.android.com/training/multiscreen/screendensities. [Último acceso: 12 Enero 2025].

[6] Google, «Especificaciones técnicas Pixel 9, Pixel 9 Pro y Pixel 9 Pro XL,» [En línea]. Available: https://store.google.com/es/product/pixel_9_specs. [Último acceso: 10 Enero 2025].

[7] Samsung, «Galaxy S,» [En línea]. Available: https://www.samsung.com/es/smartphones/galaxy-s/. [Último acceso: 10 Enero 2025].

[8] Apple, «iPhone,» [En línea]. Available: https://www.apple.com/es/iphone/. [Último acceso: 10 Enero 2025].

[9] Xiaomi, «Móviles,» [En línea]. Available: https://www.mi.com/es/product-list/phone/xiaomi/. [Último acceso: 10 Enero 2025].

[10] Firebase, «Ubicaciones multirregionales,» [En línea]. Available: https://firebase.google.com/docs/projects/locations. [Último acceso: 25 Enero 2025].

[11] Firebase, «Lenguaje de reglas de seguridad,» [En línea]. Available: https://firebase.google.com/docs/rules/rules-language. [Último acceso: 25 Enero 2025].

[12] Google, «Cómo agregar un widget de la pantalla principal a tu app creada con Flutter,» [En línea]. Available: https://codelabs.developers.google.com/flutter-home-screen-widgets. [Último acceso: 18 Febrero 2025].

[13] Flutter, «Add interactivity to your Flutter app,» [En línea]. Available: https://docs.flutter.dev/ui/interactivity. [Último acceso: 19 Febrero 2025].

[14] Flutter, «BuildContext class,» [En línea]. Available: https://api.flutter.dev/flutter/widgets/BuildContext-class.html. [Último acceso: 19 Febrero 2025].

[15] Lucide, «Lucide Icons,» [En línea]. Available: https://lucide.dev/. [Último acceso: 23 Febrero 2025].

[16] Freepik, «Freepik,» [En línea]. Available: https://www.freepik.com. [Último acceso: 20 Febrero 2025].

[17] Flutter, «Repositorio de Paquetes de Dart,» [En línea]. Available: https://pub.dev/. [Último acceso: 10 Enero 2025].