



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



**TFG del Grado en Ingeniería
Informática**

**Flutter Games
Documentación Técnica**



Presentado por Samuel Casal Cantero
en Universidad de Burgos — 20 de julio
de 2020

Tutores: Dr. César Ignacio García Osorio
Dr. José Francisco Díez Pastor

Índice general

Índice general	I
Índice de figuras	III
Índice de tablas	VI
Apéndice A Plan de Proyecto Software	1
A.1. Introducción	1
A.2. Planificación temporal	1
A.3. Estudio de viabilidad	9
Apéndice B Especificación de Requisitos	17
B.1. Introducción	17
B.2. Objetivos generales	17
B.3. Catálogo de requisitos	18
B.4. Especificación de requisitos	21
Apéndice C Especificación de diseño	29
C.1. Introducción	29
C.2. Diseño de datos	29
C.3. Diseño procedimental	35
C.4. Diseño arquitectónico	36
C.5. Diseño de interfaces	39
Apéndice D Documentación técnica de programación	45
D.1. Introducción	45
D.2. Estructura de directorios	45

D.3. Manual del programador	48
D.4. Compilación, instalación y ejecución del proyecto	53
D.5. Pruebas del sistema	57
Apéndice E Documentación de usuario	63
E.1. Introducción	63
E.2. Requisitos de usuarios	63
E.3. Instalación	64
E.4. Manual de usuario	65
Bibliografía	91

Índice de figuras

A.1. Sprint 1.	3
A.2. Sprint 2.	4
A.3. Sprint 3.	6
A.4. Sprint 4.	7
A.5. Sprint 5.	8
A.6. Sprint 6.	9
B.1. Boceto sign in	18
B.2. Diagrama con los casos de uso	22
C.1. Base de datos en firestore	30
C.2. Diagrama BD firestore	31
C.3. Contenido fichero en base de datos para un jugador usado en raking	32
C.4. Contenido fichero de cuatro en raya base de datos	33
C.5. Método para crear la tabla en la base de datos local	34
C.6. Diagrama de flujo juego snake	35
C.7. Estado Flutter	37
C.8. Diseño bloc	38
C.9. Estructura de directorios aplicación	38
C.10. Boceto de diseño	40
C.11. Paleta de colores	42
C.12. Algunos iconos	43
D.1. Google services	46
D.2. app/build.gradle	47
D.3. Comando generar clave keyStore	47
D.4. Variables entorno	49

D.5. Comando version Flutter	49
D.6. Máquinas virtuales	50
D.7. Firebase console	52
D.8. Play Store console	53
D.9. Abrir git bash	54
D.10.Comando para clonar el repositorio.	54
D.11.Actualización de paquetes	55
D.12.Máquina virtual	56
D.13.Máquina virtual	58
D.14.Código para las pruebas de caja negra	59
E.1. Tienda con el proyecto	65
E.2. Log in page	66
E.3. Formulario sign in	67
E.4. Política de privacidad	67
E.5. Home	68
E.6. Banner	68
E.7. Menú lateral	69
E.8. Mostrar sign out	70
E.9. Sign Out	70
E.10.Settings	71
E.11.Dark Mode	72
E.12.About me	73
E.13.Mail to	74
E.14.Games menú	75
E.15.Snake game	76
E.16.Snake game over	77
E.17.Formulario del ranking	78
E.18.Vídeo recompensa	79
E.19.Ranking	80
E.20.Menú del cuatro en raya	81
E.21.Invitar cuatro en raya	82
E.22.Compartir clave	82
E.23.Sin jugador	83
E.24.Formulario para unirse a partida	84
E.25.Alerta de clave no válida	84
E.26.Cuatro en raya	85
E.27.Sorteo del turno	86
E.28.Zona superior	86
E.29.Tablero	87
E.30.Zona personal	87

<i>Índice de figuras</i>	v
--------------------------	---

E.31.Envío de mensaje	88
E.32.Final del cuatro en raya	89

Índice de tablas

A.1. Equivalencias <i>Story Points</i> y tiempo estimado	2
A.2. Coste hardware	10
A.3. Coste personal	11
A.4. Coste cuota de la seguridad social	11
A.5. Coste vario	12
A.6. Horas del proyecto	13
A.7. Licencias de bibliotecas y herramientas utilizadas	14
A.8. Fuente del contenido audiovisual	14
A.9. Fuente del contenido audiovisual	15
A.10.Licencias de bibliotecas y herramientas utilizadas	15
B.1. Caso de uso 1: Sign in	23
B.2. Caso de uso 2: Mostrar el menú	24
B.3. Caso de uso 3: Configuración de la aplicación	25
B.4. Caso de uso 4: Mostrar el contenido de About	25
B.5. Caso de uso 5: Contenido publicitario	26
B.6. Caso de uso 6: Juego del snake	27
B.7. Caso de uso 7: Juego cuatro en raya online	28
D.1. Clases de equivalencia	59
D.2. Derivación en casos de prueba de caja negra	60

Apéndice A

Plan de Proyecto Software

A.1. Introducción

En este apartado se trabaja sobre la planificación del proyecto, de tal manera que se pueda definir, identificar y programar las actividades específicas que se requieren para realizar las tareas del mismo.

La evolución temporal es una de las partes más importantes de todo el proceso de desarrollo, ya que una mala planificación, puede hacer que el proyecto sufra retrasos, de tal manera, que no se llegue a la fecha de entrega prevista, lo que supone un coste, en este caso, el suspenso, pero también el económico, por las horas y recursos destinados a tal fin.

En esta fase es muy apropiado hacer una estimación lo más precisa posible, definir quien es el encargado de hacer la tarea y el dinero que supondría hacerla. Por lo que invertir tiempo en la estimación de las horas cada una de las tareas, ayuda a identificar las irregularidades que se pueden presentar en el futuro.

La viabilidad pone el foco en el coste económico del proyecto como de la parte legal. Es decir, es un reactivo limitante, sobre todo el coste.

A.2. Planificación temporal

El método de trabajo para hacer el seguimiento y la planificación del mismo es *Scrum*, que pretende realizar una gestión ágil del proyecto. Se basa en *sprints*, la duración de estos suele rondar entre los siete y quince días. Además esto depende de los requisitos del proyecto, número de integrantes

y del tiempo disponible. Así se maneja esta horquilla temporal, en mi caso por la falta de tiempo, decidí hacer cada una de las iteraciones en 5 días.

Estos *sprints* se basan en una reunión, donde se planifican todas las tareas que se pretenden realizar. En mi caso dichas reuniones eran con los tutores del trabajo de fin de grado. Además cada día se tiene que hacer el *daily meeting*, pero como el proyecto es unipersonal, no es necesario. Por lo que se puede afirmar que se ha seguido la filosofía ágil.

Hay que aclarar que la estimación del tiempo se realiza mediante los *story points*, que indican la complejidad de la tarea a realizar. Esta herramienta nos la aporta ZenHub, con la siguiente relación según el coste temporal, que podemos ver en la siguiente tabla [A.1](#)

Story Points	Estimación temporal
1	1 hora
2	2 horas
3	3 horas
4	4 horas
5	5 horas
6	6 horas
7	7 horas
8	8 horas
9	9 horas

Tabla A.1: Equivalencias *Story Points* y tiempo estimado

A continuación se detallan cada uno de los *sprints* realizados durante el proyecto:

Sprint 1 (22/06/20 - 26/06/20)

El inicio del proyecto fue a través de correo, explicando la situación a mis tutores. Para comenzar, les sugerí cambiar el tema, ya que había elegido algo de investigación con poca documentación. Me dieron el visto bueno, pero me propusieron hacer algo diferenciador. Por lo que en la reunión de cierre de *sprint* se comentaría como centrar la aplicación.

Debido a la escasez temporal, decidí hacer una aplicación en *Flutter*, ya que se puede diseñar algo de calidad de manera ágil.

Los objetivos en este *sprint* inicial fueron los siguientes:

- Documentar como hacer aplicaciones en *Flutter*.
- Realizar un curso en [Udemy](#)¹
- Crear el repositorio.
- Implementar el cuerpo de la aplicación.
- Investigar sobre *apis* que ofrece el *framework*.

Todas las *issues* realizadas para este *sprint*, están disponibles en [Sprint 1](#)²

La estimación fue de 54 horas, pero que finalmente se destinaron 45,5 horas, debido en gran parte a la reutilización de código del curso, aunque dicha formación duró más tiempo de lo que se había estimado.

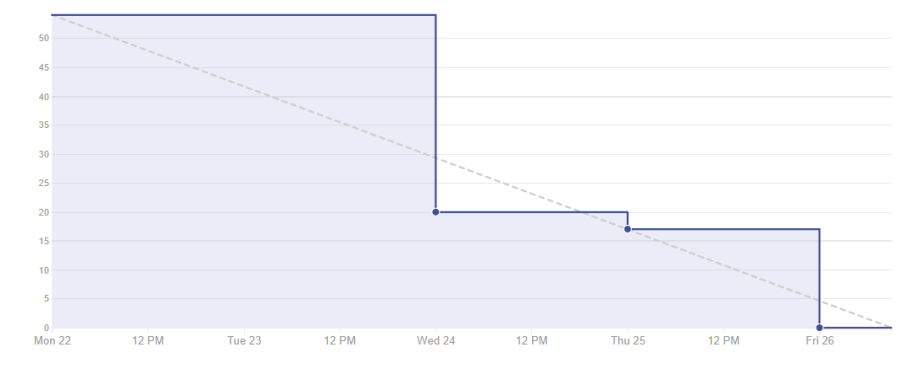


Figura A.1: Sprint 1.

Sprint 2 (27/06/20 - 01/07/20)

En este segundo incremento de la aplicación, se definió que el proyecto iba a ser una colección de juegos, además podría usarse como *portfolio*, por lo que los juegos eran el hilo conductor para tocar el mayor número de herramientas posible.

Los objetivos planteados fueron los siguientes:

- Persistencia de datos, con base de datos local.

¹<https://www.udemy.com/course/flutter-primeros-pasos/>

²https://github.com/scc0034/flutter_serpiente/milestone/1?closed=1

- Conectar con firebase para usar la base de datos que proporciona.
- Formulario para recoger las puntuaciones de los jugadores.
- Hacer la ventana que muestra el ranking de los jugadores.
- Avanzar de manera significativa en la memoria.
- Mejorar el juego del snake.
- Implementar la api de login de firebase, mediante Google.
- Google Ad mobile.

Las diferentes *issues* planificadas están en **Sprint 2**³

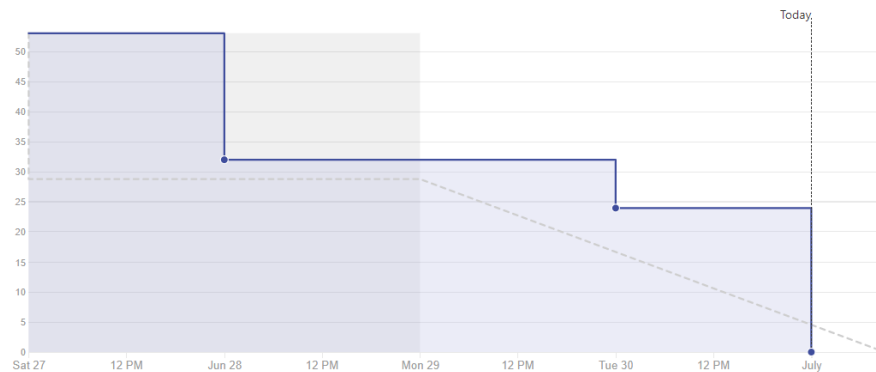


Figura A.2: Sprint 2.

En la reunión del cierre del seguimiento del sprint se muestra la **v.2.0.0**⁴, viendo que los objetivos propuestos para este *sprint* se han logrado a excepción de los anuncios, que por diversos motivos no se muestran.

Se estimaron unas 53 horas, pero se acabaron destinando 45 horas, que sirvieron para completar todas las tareas.

Sprint 3 (02/07/20 - 06/07/20)

En esta iteración de la aplicación se debía de añadir un juego más para que tenga sentido como una colección de juegos. La propuesta de juego

³https://github.com/scc0034/flutter_serpiente/milestone/2?closed=1

⁴https://github.com/scc0034/flutter_serpiente/releases/tag/V.2.0.0/

consistía en mover unas tuberías de agua hasta cerrar un circuito. Descarte esa opción a medida que podía explotar la funcionalidad de tener una base de datos en tiempo real. Finalmente el juego elegido fue el cuatro en raya online. Además de la mejora constante del producto.

Los objetivos planificados fueron:

- Añadir sonidos y música al juego del snake.
- Funcionalidad de las tuberías en el snake.
- Controlar que si hay mejora de puntuación para cada usuario en el snake, pueden hacer un submit con la nueva puntuación.
- Añadir imágenes de medallas al ranking.
- Solucionar el problema de funcionamiento de los anuncios.
- Mejorar el rendimiento del conjunto.
- Crear la interfaz de conexión del cuatro en raya con la base de datos.
- Crear la primera estructura del juego cuatro en raya.
- Realizar las pruebas necesarias para un despliegue de la aplicación en entorno web.
- Solucionar algunos problemas de la play store.
- Continuar con la realización de la memoria y anexos.

Las diferentes *issues* se encuentran en [Sprint 3](https://github.com/scc0034/flutter_serpiente/milestone/3?closed=1)⁵

⁵https://github.com/scc0034/flutter_serpiente/milestone/3?closed=1

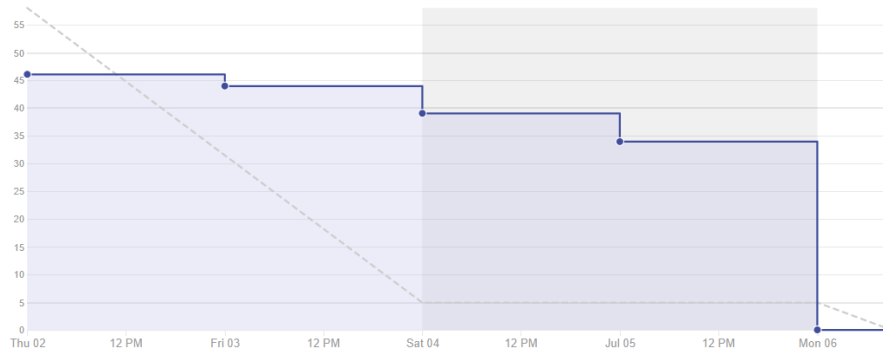


Figura A.3: Sprint 3.

Al final en la reunión del cierre de este *Sprint* con los tutores, quedó claro que gran parte de las tareas fueron completadas. En cuanto al tema del juego, no se pudo hacer más que la estructura del mismo, pero para la siguiente iteración del producto debería de quedar terminado.

En cuanto a las horas planificadas para la documentación de la memoria, solo hice el 17 % de las 12 horas que me propuse. En gran parte fue debido a que destiné algunas horas más a otras tareas o que fui demasiado optimista a la hora de planificar.

Se estimaron unas 58 horas, pero finalmente se emplearon 44,5. Todas las tareas se completaron a excepción del despliegue de la aplicación en el entorno web, debido a unos problemas con las compilaciones, y de las 10 horas no invertidas en la redacción de la memoria.

Sprint 4 (07/07/20 - 10/07/20)

La duración de este *Sprint* es de un día menos, porque coincidía el cierre en sábado. Por lo que se pasa de los cinco días de duración normal, a cuatro. Lo que se plantea es tener el juego terminado además de añadir un chat que aporte valor añadido.

Las pretensiones para este *sprint* son:

- Actualizar el contenido de la *Play Store*
- Reunión de seguimiento el viernes 10 julio.
- Control del turno en el juego online.

- Colocación correcta de la ficha.
- Determinar cuando se produce el fin del cuatro en raya.
- Test de que funciona el juego online correctamente.
- Mejoras generales en el rendimiento.
- Indentar y eliminar los warnings del código.
- Capacidad de mandar mensajes entre los jugadores.
- Reunión con el diseñador del póster, para explicar los requisitos.
- Resolución de algunos problemas.

Las diferentes *issues* se encuentran en [Sprint 4](#)⁶



Figura A.4: Sprint 4.

Para el cierre, todas las tareas presupuestadas fueron realizadas. Se planificaron 41 horas, destinando 33 horas para el desarrollo esperado.

Sprint 5 (11/07/20 - 15/07/20)

Para este *Sprint* la idea es terminar los anexos, pulir el despliegue de la aplicación en la *Play Store* y resolver los problemas finales.

Los objetivos para este *sprint* son:

⁶https://github.com/scc0034/flutter_serpiente/milestone/4?closed=1

- Actualizar el contenido de la *Play Store*
- Planificar el sprint.
- Plan de proyectos anexos.
- Requisitos anexos.
- Diseño anexos.
- Manual del programador anexos.
- Manual de usuario anexos.
- Reunión de seguimiento.
- Solucionar el problema al compilar la memoria, ya que no se ve.
- Problemas menores.

Las diferentes *issues* se encuentran en **Sprint 5**⁷

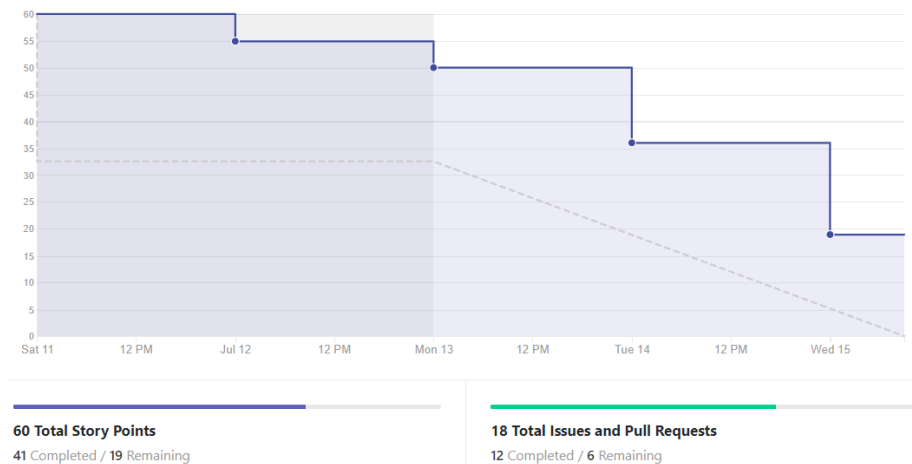


Figura A.5: Sprint 5.

Para el cierre del sprint 5 no todas las tareas se terminaron, debido a que la planificación fue optimista y que las horas empleadas para algunas tareas fue mayor a la estimación.

De las 60 horas planificadas para la semana, 41 horas se emplearon para cerrar el 66 % de las tareas, quedando 19 horas de tareas, que se mueven al siguiente sprint.

⁷https://github.com/scc0034/flutter_serpiente/milestone/5?closed=1

Sprint 6 (16/07/20 - 20/07/20)

Tras el cierre del *sprint* anterior quedó claro que no se pudieron completar todas las tareas, ya que la memoria y anexos llevan mucho tiempo, si se quiere hacer contenido de calidad. Por lo que para este último sprint del proyecto, se pretende terminar toda la documentación, revisar todos los elementos a entregar y la creación de los videos.

El día 20 de Julio, día de la entrega se tiene que hacer la reunión final de cierre, donde se firmarán las dos documentos y se procederá al cierre del proyecto en Github.

Las diferentes *issues* se encuentran en [Sprint 6⁸](https://github.com/scc0034/flutter_serpiente/milestone/6?closed=1)

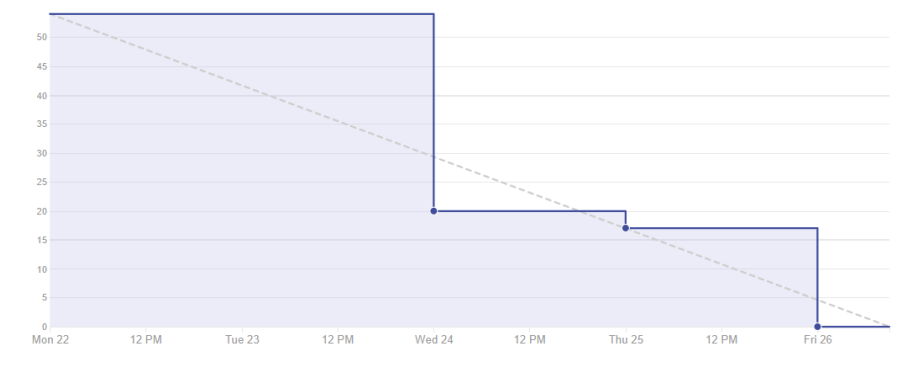


Figura A.6: Sprint 6.

Para el cierre de este *sprint* todas las tareas fueron completadas, ya que también era el cierre del proyecto. El número de horas que se destinaron para este fueron de 38.

A.3. Estudio de viabilidad

En esta parte se pretende analizar el coste / beneficio de todo el proyecto, como el apartado legal en todo el proceso de desarrollo, en el caso de que se hubiera tenido que realizar en un entorno real.

⁸https://github.com/scc0034/flutter_serpiente/milestone/6?closed=1

Viabilidad económica

La estructura de mi trabajo, es la de un proyecto con precio cerrado, es decir, no soy un trabajador asalariado, sino autónomo, que recibe un encargo, en este caso por parte de la Universidad. La definición que encaja para esto es la de *freelancer*, [?]. Por lo que el coste dependerá de la estimación inicial de las horas del proyecto, así como de otros costes que conozcamos de ante mano, finalmente se incluye un porcentaje de beneficios. En mi caso la estimación inicial de las horas fue de 250 horas.

Otra de las vías para obtener ingresos es mediante los anuncios ofrecidos por *Google Ad*, a través de diferentes tipos de *banners* y vídeos con los que tener un retorno de dinero. La idea es que la aplicación esté disponible de manera gratuita, en la *play store* para todos aquellos que se la quieran descargar.

Aunque la app disponga de publicidad, se desestimará para el cálculo del beneficio, ya que es complicado tener una gran repercusión en las primeras etapas de proyecto, o incluso una vez finalizado.

Coste hardware

Se desglosa el coste de los dispositivos usados para la implementación, suponiendo que la amortización del PC de sobremesa es aproximadamente de 5 años y de 2 años para el *smartphone*, con la duración del proyecto de 1 mes.

Concepto	Coste (€)	Coste amortización (€)
PC sobremesa	1200	20
Dispositivo móvil	450	18,75
Coste total		38,75

Tabla A.2: Coste hardware

Coste software

El coste de las librerías, *cloud services*, entornos de desarrollo, licencias, cursos, máquinas virtuales, entre otros, han sido de uso gratuito, dando lugar a un coste software de 0 euros.

Coste personal

El proyecto fue llevado por un solo trabajador (Samuel Casal Cantero), ya que es un proyecto unipersonal, que se encargaba del desarrollo de software y la planificación. El tiempo inicialmente estimado fue de 250 horas. Siendo el trabajador autónomo se considera el siguiente salario:

Concepto	Precio €/h	horas	Coste (€)
Freelancer	20	250	5000

Tabla A.3: Coste personal

En lo referente a las cuotas de la seguridad social, para el año 2020, tenemos que la cuota mínima a pagar es el resultado de aplicar el 30,3 % al salario mínimo interprofesional [?] que es de 944,4 €, dando lugar a que esta cuota sea de 286,15 €. Dependiendo de la contingencia el desglose es:

Concepto	Coste (€)
Salario bruto del trabajador	944,4
Contingencias comunes (28,3 %)	283,2
Contingencias profesionales (1,1 %)	10,38
Cese de actividad (0,8 %)	7,55
Formación profesional (0,1 %)	0,94
Coste cuota	286,15

Tabla A.4: Coste cuota de la seguridad social

Finalmente vemos que el coste del empleado para este proyecto es de 5286,15 €.

Otros costes

Son otro tipo de gastos que también se deben tener en cuenta.

Concepto	Coste (€)
Internet	50
Cuenta Google Play	25
Maquetador póster	50
8 horas asesoramiento tutores	600
Coste total	725

Tabla A.5: Coste vario

Coste total estimado del proyecto

El coste total estima del proyecto es la suma de los costes anteriores, el cual da un importe de 6011,15 €, a esto le tenemos que aplicar el correspondiente incremento del impuesto de valor añadido, que es [?] del 21 %, ya que se debe realizar una factura, por lo que el coste total estimado del proyecto es de **7273,5 €**.

Margen de beneficios

La idea es que la aplicación se distribuya de manera gratuita a través de la cuenta de *Google Play*, pero a pesar de tener publicidad los beneficios que retornará serán prácticamente nulos.

Por lo que la vía de obtener ingresos como freelance, puede ser incrementar el coste del proyecto un 15 %, con el fin de asegurar ese beneficio. Esto garantiza que ante un incremento de las horas planificadas, siga existiendo margen de beneficio. En el caso de que no lleguemos a gastar todas las horas planificadas, también supondrá un aumento de las ganancias.

El nuevo coste total del proyecto sería: $6011,15 * 0,15 \% = 6912,82$ €, que añadiendo el I.V.A. del 0,21 %, se queda en **8364,5 €** que debería de pagar la Universidad como cliente que realiza el encargo. Esta sería la factura final de todo el proyecto.

Coste real del proyecto

Haciendo la suma de las horas destinadas en cada uno de los *sprints*, vemos en la tabla A.6:

Sprint	Horas
Sprint 1, ver pág. 2	45,5
Sprint 2, ver pág. 3	45
Sprint 3, ver pág. 4	44,5
Sprint 4, ver pág. 6	33
Sprint 5, ver pág. 7	41
Sprint 6, ver pág. 9	38
Horas totales	247

Tabla A.6: Horas del proyecto

El coste real del proyecto sería el mismo que el coste estimado pero con 3 horas de diferencia, ya que se estimaron 250 horas de las que han sido necesarias emplear 247 horas.

Este número de horas hace un coste de : **5951,15 €** 60 euros menos que la estimación inicial.

Beneficios

Para calcular todos los beneficios debemos de ver la diferencia entre el coste total estimado, menos el coste real del proyecto.

- El coste estimado con 250 horas y horquilla de 0.15 % era: 6912,82 €.
- El coste real del proyecto: 5951,15 €.
- Diferencia = 6912,82 € - 5951,15 € = 961.67 €.

Hay que recalcar que las horas como freelance también son beneficios, pero personales, ya que soy el encargado de ejecutar el trabajo final de grado. Pero que para el proyecto imputan como costes.

Los beneficios a fecha 20/07/2020: +961.67 €

Viabilidad legal

Para el completar el proyecto han sido necesarias gran multitud de herramientas, a continuación, en la tabla A.7, se expondrán las principales que fueron utilizadas.

Librería	Versión	Descripción	Licencia
VsCode	1.46.1	Editor de código.	MIT
Android Studio	4.0	App virtualización del SO Android.	Apache 2.0
Android R	10.0+ Api 30	Versión del S.O virtualizado.	Apache 2.0
Flutter	1.17	Framework usado en desarrollar la app	BSD 3-Clause
Dart	2.2.0	Lenguaje prog. desarrollado por Google	BSD
Node.js	12.18	Uso de npm	MIT
Firebase console	5.5	Herramienta de Google gestión servicios.	Gratis
Play Store console	–	Backend developers tienda apps	Licencia pagada
Admob services	–	Herramienta publicidad	Gratis

Tabla A.7: Licencias de bibliotecas y herramientas utilizadas

Aclarar que de las herramientas anteriores, las dos que aparecen como licencia gratuita, es debido a la escalabilidad de aplicación. Hace referencia que a partir de una cantidad de usuarios/consultas tienes que pagar por el servicio para el caso de firebase console. Para el caso de admob, depende del número de clicks en los anuncios, impuestos y regulaciones varias. En mi caso es una aplicación pequeña, siendo el servicio gratuito con los beneficios para Google.

Imágenes y material gráfico

Fuente	Descripción	Licencia
Giphy	Repositorio de contenido visual	Open Source

Tabla A.8: Fuente del contenido audiovisual

Música y sonido

Fuente	Descripción	Licencia
Freesounds ⁹	Repositorio musical visual	CC0 Public Domain

Tabla A.9: Fuente del contenido audiovisual

Herramientas

En Flutter son conocidos como *packages*, paquetes que ofrecen funcionalidades que ya han sido implementadas. Todas ellas se pueden encontrar en la siguiente [web](#)¹⁰. Estos se integran en el fichero `pubspec.yaml`, siendo este muy importante. Para el proyecto se usaron:

Librería	Versión	Descripción	Licencia
shared preferences	0.5.6	Persistencia datos.	BSD
flutter email sender	3.0.1	Envió correos	Apache 2.0
url launcher	5.4.9	Abrir navegador.	Apache 2.0
flutter phone state	0.5.9	Abrir app llamadas	MIT
sqflite	1.3.0	BD local	MIT
path	1.6.4	Rutas	BSD
firebase auth	0.16.1	Conectar con firebase	BSD
google sign in	4.5.1	Sign in Google	BSD
firebase core	0.4.5	API firebase moviles	BSD
firebase core web	0.1.1+2	API web	BSD
firebase admob	0.9.3+2	Integración publicidad	BSD
cloud firestore	0.13.7	Cloud firestore	BSD
audioplayers	0.15.1	Reproductor musical	MIT
flutter share me	0.9.1	Shared	Apache 2.0
bubble	1.1.9+1	Chat	BSD
provider	2.0.1	Herramienta integración	MIT

Tabla A.10: Licencias de bibliotecas y herramientas utilizadas

Licencia aplicación

Estudiando todas las compatibilidades de las licencias, donde más problemas encontramos es en la licencia Apache 2.0, ya que es de las más

¹⁰<https://pub.dev/>

restrictivas. No impide la comercialización (*free-to-play*), que es justo lo que se pretende, pero en el caso de que modifique el código fuente de algunas de las herramientas, si que tendría más problemas, pero no se da el caso, ya que solo se usan.

Revisando los juegos hay que detenerse en:

- **Derecho de autor o *Copyright*:** no se infringe, ya que este protege el código fuente, aplicación y recursos artísticos. En este caso se hizo desde cero toda la aplicación, por lo que no afecta.
- **Propiedad industrial:** no se infringe, no hay patentes encontradas.
- **Trademark o marca registrada:** sin problema, no se puede apropiar de la palabra serpiente / snake o cuatro en raya, por lo que se puede llamar así.

Por lo tanto el trabajo final de grado tiene:

Licencia MIT (Massachusetts Institute Technology) siendo una licencia de uso libre y permitiendo su uso comercial y modificación.

Apéndice *B*

Especificación de Requisitos

B.1. Introducción

En este apéndice se detallan los requisitos del proyecto, los funcionales y no funcionales. El objetivo es hacer de analista entre el cliente y los programadores, para entender, comprender y diseñar la aplicación.

B.2. Objetivos generales

Este trabajo final de grado tiene dos puntos de vista totalmente diferentes, dependiendo de las situaciones futuras a las que se someta la aplicación. Sin olvidar que es una aplicación que encarga la universidad de Burgos, como cliente final.

- **Colección de videojuegos** Tener una gran cantidad de estos mismos, para lograr sacar un rendimiento económico gracias a la publicidad, por lo que es necesario que la aplicación tenga una gran repercusión en el mercado. Esto implica que a futuro se debe de hacer una inversión en publicidad y *marketing*.
- **Portfolio** [?] o escaparate con el que mostrar las capacidades técnicas, herramientas usadas o lenguajes de programación aprendidos.

B.3. Catálogo de requisitos

A continuación se enumeran los requisitos del proyecto extraídos de los generales.

Requisitos funcionles

Cada uno de los requisitos funcionales tiene una correlación con la pantalla en la que se da la operatividad del mismo, excepto en los casos en las que este requerimiento abarca varias ventanas o que no es una funcionalidad con interfaz.

Algunos de los requisitos funcionales, en el momento de ser tratados con el cliente, tuvieron apoyo de unos *sketch* o bocetos conceptuales, con el fin de dar soporte, ayudar a comprender y facilitar la comunicación cliente - analista. Como podemos ver en la imagen B.1, o en el apéndice de diseño 29.

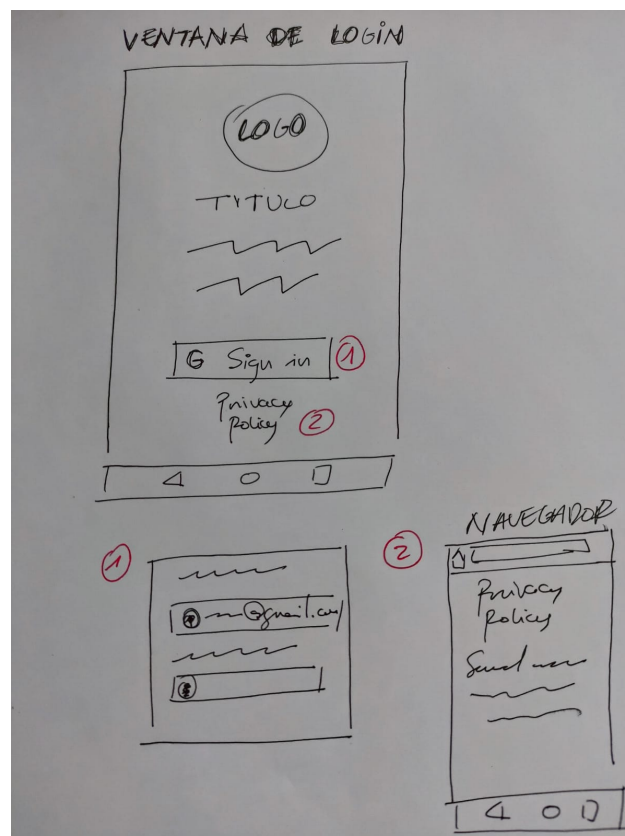


Figura B.1: Boceto sign in

- **RF-1 Ventana de *sign in*: E.2** La aplicación tiene que contener una ventana principal con la que los usuarios puedan iniciar la sesión, con el fin de tener servicios en la nube.
 - **RF-1.1 Botón de inicio sesión:** Mediante un elemento *clickable* se debe de tener acceso a un formulario para poder entrar mediante la cuenta de *Google*.
 - **RF-1.2 Consultar la política de privacidad:** Un enlace en el que se pueda mostrar todo lo referente a la política de privacidad.
 - **RF-1.3 Cerrar sesión:** ser capaz de salir de la aplicación *sign out*, desde cualquier ventana.
- **RF-2 Menú de navegación: E.7:** Se considera necesario tener la capacidad de navegar de una parte a otra dentro de la aplicación, sin tener que pasar entre ventanas, mediante un menú.
 - **RF-2.1 Mostrar el menú lateral** el usuario debe poder entrar al menú mediante un *scroll lateral*.
 - **RF-2.2 Navegar:** la capacidad de navegar entre las diferentes ventanas que ofrece la aplicación.
 - **RF-2.3 Integración con el RF-1.3:** Para que se disponga la salida en cualquier momento de la aplicación es necesario incluir en el menú, los datos de usuario con la capacidad de salir.
- **RF-3 Configurar app E.10** la aplicación tiene que poder gestionar algunas de las diferentes capacidades del producto, que el usuario desee.
 - **RF-3.1 Opción *dark mode*:** la aplicación tiene que tener la capacidad de cambiar el color del sistema, para ahorrar batería o para mejorar el contraste.
 - **RF-3.2 Opciones del juego snake:** algunas de las opciones de este juego deben de ser controlables desde el apartado de ajustes.
- **RF-4 Ventana de *about* E.12** mostrar información sobre el creador de la aplicación.
 - **RF-4.1 Mostrar carrusel imágenes:** enseñar al usuario fotografías de los lenguajes de programación que el creador de la aplicación conoce.
 - **RF-4.2 Consultar mediante *iconbuttons*:** unos botones que redirijan a contenido relevante del creador.
- **RF-5 Publicidad integrada:** la aplicación debe de ser capaz de mostrar anuncios.

- **RF-5.1 Mostrar *banner*:** colocar en el *footer* un contenedor de anuncios, proporcionados por Google.
 - **RF-5.2 Continuar Snake:** en el caso de que finalice la partida, la primera vez que muera la serpiente, el usuario tiene la posibilidad de ver un vídeo, que tiene como recompensa continuar con la partida.
- **RF-6 Juego del snake E.15:** el usuario tiene que tener la posibilidad de jugar.
- **RF-6.1 Jugar al snake:** la aplicación tiene que manejar la lógica del juego como los choques que se producen con la comida, pared, bloques, o tuberías.
 - **RF-6.2 Mostrar la puntuación:** la app tiene que mostrar la puntuación que lleva el jugador durante la partida.
 - **RF-6.3 Compartir la puntuación:** el jugador tiene que ser capaz de subir la puntuación una vez termine la partida. Siempre y cuando esta sea mejor a otra anterior.
 - **RF-6.4 Mostrar el ranking:** el usuario tiene que ser capaz de ver la puntuación del resto de jugadores, así como la que él ha logrado.
- **RF-7 Juego del cuatro en raya E.20:** el usuario tiene que ser capaz de jugar a este juego.
- **RF-7.1 Mostrar menú juego:** la app tiene que enseñar un menú intermedio con dos opciones a elegir entre invitar o unirse.
 - **RF-7.2 Invitar:** el usuario puede invitar a otro jugador, compartiendo una clave de acceso a la partida. Tiene que haber un botón que ayude a hacer esta tarea.
 - **RF-7.3 Unirse:** el usuario mediante un formulario puede ingresar la clave de acceso a la partida.
 - **RF-7.4 Jugar al cuatro en raya:** la aplicación tiene que manejar la lógica de negocio como controlar los diferentes eventos se producen durante la partida. Estos pueden ser: sorteo de inicio, ficha no válida, formación del cuatro en raya, advertir de quien gana...
 - **RF-7.5 Hablar mediante chat:** el usuario tiene que tener la capacidad de mandar mensajes cortos, de no más de 15 caracteres. Después de remitir el mensaje, se tiene que bloquear esta opción durante 10 segundos, para no sobrescribir otros mensajes o que se sature el juego.

Requisitos no funcionales

- **RNF-1 Rendimiento:** la capacidad de dar una calidad de producto homogénea, no debe verse menguada por tiempos de carga, cuelgues en el sistema o cualquier problema / incidencia que lastre el desempeño del producto.
- **RNF-2 Seguridad:** ofrecer la aplicación desde la *Store*, ya que esta tiene soporte integrado de antivirus. Lo que da una garantía de confianza. Ninguno de los datos deben de ser ofrecidos a terceros, exceptuando a Google.
Mediante un cifrado de extremo a extremo (SHA-1 [?]) se realizarán las comunicaciones con *Firebase cloud* y *Play store*.
- **RNF-3 Escalabilidad:** todo el sistema tiene que ser escalable dependiendo del número de usuarios simultáneos que estén usando la app.
- **RNF-4 Usabilidad:** interfaces como las interacciones con el usuario, tienen que ser lo más amigables posibles. (*responsive*)
- **RNF-5 Disponibilidad:** la aplicación tiene que estar operativa en cualquier momento, así como garantizar una frecuencia de actualizaciones periódicas.
- **RNF-6 Plataforma:** dirigido a dispositivos con el sistema operativo Android, desde la versión 10.0 Lollipop.

B.4. Especificación de requisitos

En este apartado se muestran los diagramas de casos de uso, con una breve descripción.

Actores

Solo tenemos un tipo de actor que puede interactuar con la aplicación, que es el usuario estándar.

Como apreciación, los desarrolladores tendrán acceso al *backend*, donde se puede gestionar la plataforma y la tienda. Pero no es un usuario final de la aplicación, por lo que quedan excluidos del diagrama de casos de uso. Ya que esto es algo inherente a su cargo.

A continuación se muestra el diagrama de casos de uso **B.2:**

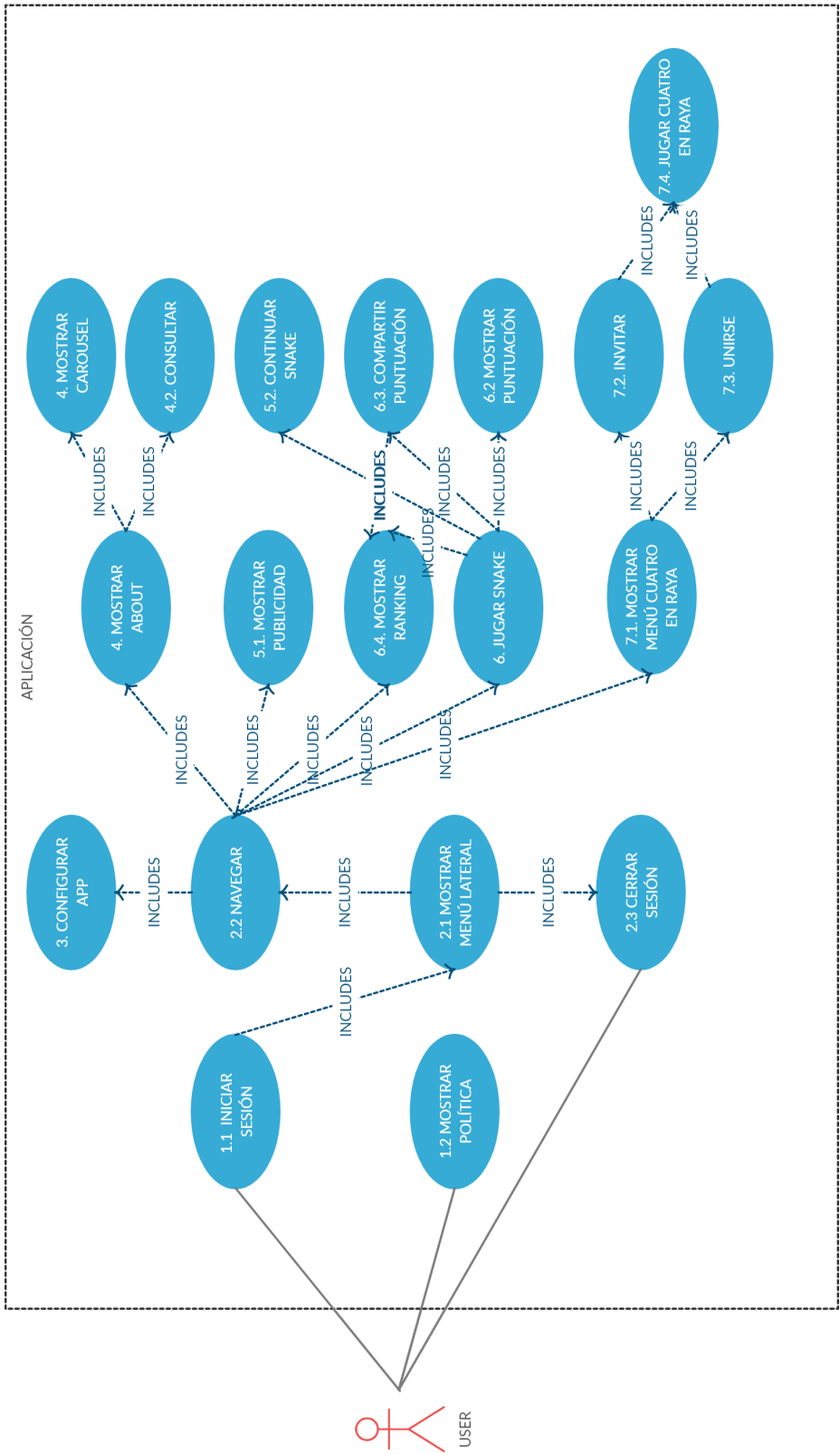


Figura B.2: Diagrama con los casos de uso

Casos de uso

Caso de uso 1: Sign in	
Descripción	Permite al usuario acceder a la aplicación mediante su cuenta de Google. Como consultar la política de privacidad.
Requisitos	RF-1, RF-1.1, RF-1.2
Precondiciones	Tener la aplicación descargada en el terminal
Secuencia normal	Paso Acción
	1 EL usuario <i>clicka</i> en 'Privacy policy'.
	2 Se muestra el navegador con la política.
	3 Se pulsa en el botón de 'Sign in'.
	4 Se indica que cuenta de Google usar, entrando dentro de la aplicación.
Postcondiciones	Ninguna.
Excepciones	No disponer de cuenta de Google
Importancia	Alta
Urgencia	Alta
Frecuencia	Alta

Tabla B.1: Caso de uso 1: Sign in

Caso de uso 2: Mostrar el menú		
Descripción	Da la capacidad al usuario de navegar entre las ventanas de la aplicación, como salir de la misma en el caso de que así lo requiera.	
Requisitos	RF-2, RF-2.1, RF-2.2, RF-2.3, RF-1.3	
Precondiciones	Haber realizado el requisito funcional primero.	
Secuencia normal	Paso	Acción
	1	Deslizamiento horizontal derecho o pulsación en el menú hamburguesa, para mostrar el menú lateral.
	2	Elegimos una de las diferentes opciones del menú.
	3	Navegamos a la ventana escogida o salimos de la aplicación.
Postcondiciones	Ninguna.	
Excepciones	El menú no se encuentra disponible dentro de los juegos	
Importancia	Alta	
Urgencia	Alta	
Frecuencia	Alta	

Tabla B.2: Caso de uso 2: Mostrar el menú

Caso de uso 3: Configuración de la aplicación	
Descripción	Dotar al usuario de las herramientas necesarias para poder hacer cambios de alguna de las partes del sistema.
Requisitos	RF-3, RF-3.1, RF-3.2
Precondiciones	Haber realizado el requisito funcional uno.
Secuencia normal	Paso Acción
	1 Escoger del menú lateral la parte de ‘settings’.
	2 Una vez dentro, usar los ‘sliders’ activar o desactivar las funciones indicadas.
Postcondiciones	Ver que los cambios se producen.
Excepciones	Ninguna.
Importancia	Media
Urgencia	Media
Frecuencia	Alta

Tabla B.3: Caso de uso 3: Configuración de la aplicación

Caso de uso 4: Mostrar el contenido de about	
Descripción	Disponer de la opción de ver quién es el creador de la aplicación y navegar a recursos.
Requisitos	RF-4, RF-4.1, RF-4.2
Precondiciones	Haber realizado el requisito funcional uno.
Secuencia normal	Paso Acción
	1 Escoger del menú lateral la parte de ‘About me’.
	2 Pulsar en los iconbuttons para mostrar el contenido seleccionado en el navegador. Si el botón es el de correo, se tiene que abrir la aplicación correspondiente para mandar un feedback.
Postcondiciones	Ninguna.
Excepciones	Ninguna.
Importancia	Baja
Urgencia	Baja
Frecuencia	Baja

Tabla B.4: Caso de uso 4: Mostrar el contenido de About

Caso de uso 5: Contenido publicitario		
Descripción	Durante la ejecución, en ciertas ventanas tiene que estar disponible el contenido publicitario, así como vídeo recompensas para poder continuar con la partida del snake.	
Requisitos	RF-5, RF-5.1, RF-5.2	
Precondiciones	Haber realizado el requisito funcional uno. Producir un ‘game over’ en el juego del snake.	
Secuencia normal	Paso	Acción
	1	Disponible en las ventanas apropiadas. Al ser pulsadas, se debe abrir el navegador o la ‘play store’.
	2	En el juego del snake, ver el vídeo hasta completar el tiempo requerido del mismo, para que nos den otra vida.
Postcondiciones	Continuar con la partida del snake.	
Excepciones	Ninguna.	
Importancia	Alta	
Urgencia	Alta	
Frecuencia	Alta	

Tabla B.5: Caso de uso 5: Contenido publicitario

Caso de uso 6: Juego del snake		
Descripción	La aplicación tiene que permitir a los usuarios pasar un buen rato jugando al snake.	
Requisitos	RF-6, RF-6.1, RF-6.2, RF-6.3, RF-6.4	
Precondiciones	Ninguna.	
Secuencia normal	Paso	Acción
	1	Abrir el menú lateral para navegar a la pestaña de juegos, una vez dentro se elige el juego del snake.
	2	Intentar conseguir la mayor puntuación posible.
	3	Si morimos, se puede ver un vídeo para continuar con la partida.
	4	En el caso de tener una mejor puntuación, se mostrará el formulario para compartirla.
Postcondiciones	Mostrar el ranking del snake.	
Excepciones	Ninguna.	
Importancia	Alta	
Urgencia	Alta	
Frecuencia	Alta	

Tabla B.6: Caso de uso 6: Juego del snake

Caso de uso 7: Juego cuatro en raya online	
Descripción	Ofrecer a los jugadores la oportunidad de crear o unirse a partidas, con el fin de poder batirse en batallas del cuatro en raya. Durante dichas partidas tiene que ser posible el envío de mensajes cortos.
Requisitos	RF-7, RF-7.1, RF-7.2, RF-7.3, RF-7.4, RF-7.7
Precondiciones	Tener un compañero con el que batirse.
Secuencia normal	Paso Acción
	1 Abrir el menú lateral para navegar a la pestaña de juegos, una vez dentro se debe elegir el juego del cuatro en raya.
	2 Dependiendo del rol que tomemos para el juego, podemos compartir un código o ingresar este para comenzar la partida.
	3 Si no se encuentra el rival, la partida finalizará.
	4 Poder mandar mensajes cortos durante el juego.
	5 Si se da la situación de final de partida, mostrar que jugador es el ganador.
Postcondiciones	Ninguna.
Excepciones	Ninguna.
Importancia	Alta
Urgencia	Alta
Frecuencia	Alta

Tabla B.7: Caso de uso 7: Juego cuatro en raya online

Apéndice C

Especificación de diseño

C.1. Introducción

En este apéndice se recoge el diseño de las interfaces, la manera en la que se resolvieron los requisitos funcionales anteriormente expuestos [17](#) y el manejo de los datos o la estructura de los mismos.

C.2. Diseño de datos

En el tratamiento de los datos, se ha optado por hacerlo de dos formas diferentes, esto es debido, a que se necesita persistencia local y externa. Dependiendo de eso, aparecen dos diseños de datos diferentes.

- **FireStore:** es la base de datos integrada en Firestore. Esta no sigue el modelo clásico, ya que es noSQL [?], es decir, al igual que mongoDB [?], esta se gestiona mediante ficheros `.json`. A diferencia SGBDR (Sistema gestor de bases de datos relacionales), la manera de trabajar de Firestore, es mediante modelos no relacionales. Esta fue usada para la persistencia externa de datos. Esta idea se puede ver en la siguiente imagen [C.1](#) de la consola de Firebase:

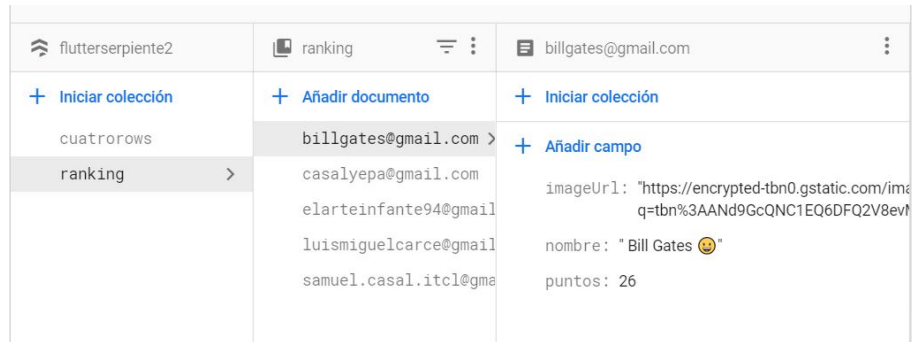


Figura C.1: Base de datos en firestore

- **Sqlflite:** esta base de datos si que sigue el modelo tradicional del SGBDR. En mi caso fue necesario usar este paquete [?], que internamente funciona con sqlite. Usado para almacenar datos en forma local, ya que no era necesario que los datos salieran del terminal. Una de las cosas a tener en cuenta de esto, es que si el usuario borra la caché de la aplicación o la desinstala se borran los ficheros correspondientes. Esto no debe de ser un problema, ya que solo es necesario almacenar ciertos valores.

Diagrama entidad relación

- **Firestore:** la distribución en la base de datos es mediante colecciones, ver imagen C.2. Como ejemplo podemos tener la colección para almacenar los datos del ranking, y para cada una de las entradas del ranking un fichero `.json` que haga referencia a los datos de cada usuario. Aunque esta no siga el modelo relacional, si que se podría implementar un diagrama entidad-relación, pero no se hace, porque es similar a tener dos tablas en la base de datos y que no tiene relación entre sí (colecciones en este caso).



Figura C.2: Diagrama BD firestore

Una vez que se sabe como es el funcionamiento de la base de datos noSQL, las dos colecciones necesarias para la aplicación fueron:

- **ranking:** se encarga de almacenar la mejor puntuación de cada uno de los jugadores del snake [C.3](#). Para distinguir cada uno de los documentos, se usa como clave primaria el correo del usuario, que también será el nombre que identifique a cada uno de estos ficheros.

Las variables que almacena son: nombre(String), imageUrl(String) y puntuación(int).

+ Añadir campo

```
imageUrl: "https://encrypted-tbn0.gstatic.com/images?q=tbn%3AAND9GcQNC1EQ6DFQ2V8evM0mhwQkjDrF3NbxeBCK9A&usc"
nombre: " Bill Gates 😊"
puntos: 26
```

Figura C.3: Contenido fichero en base de datos para un jugador usado en raking

- **cuatrorows:** esta colección guarda cada una de las partidas online del juego cuatro en raya C.4. Los nombres de los documentos se generan de manera única en la base de datos, por lo que no puede haber dos partidas iguales.

Lo que hace que la *key* compartida durante el juego sea la misma que el nombre del documento dentro de esta colección. Los campos que tiene este documentos son muchos y variados, pero los más destacados son:

- Posición de cada una de las fichas, de tipo String, y los valores que toma son Y, R o *null*, dependiendo de la ficha que se encuentre en la celda.
- Datos de los jugadores, tanto como para el que crea la partida como el que recibe la invitación, estos son: nombre, imageUrl, correos... Todos de tipo String.
- Flags para controlar si se producen ciertos eventos, como puede ser el final de la partida, si se ha mostrado el mensaje, lanzamiento de la moneda para el sorteo de quien inicia o el mensaje escrito por cada uno de los jugadores. Todas ellas son de tipo String, exceptuando las que puedan tomar valores de verdadero o falso.

+ Añadir campo

```
46: "Y"
47: "Y"
48: ""
casalyepamsg: ""
conectado: true
emailRed: "casalyepa@gmail.com"
emailYellow: "samuel.casal.itcl@gmail.com"
endGame: false
fecha: 10 de julio de 2020, 16:48:06 UTC+2
imageUrlRed: "https://lh3.googleusercontent.com/a-/AOh14GhG3jYGoRH87WM
c"
imageUrlYellow: "https://lh3.googleusercontent.com/a-/AOh14Gj7cJ1j9JVNQI
c"
inGame: false
lanzamiento: true
```

Figura C.4: Contenido fichero de cuatro en raya base de datos

- **Sqlflite:** usada para la persistencia interna de datos. No tiene diagrama de entidad relación ya que solo consta de una tabla. Como se puede ver en la imagen C.5, este método se encarga de crear el recurso de la tabla, en el caso de que no existan, (cuando se instala la aplicación en el terminal).

```
Future _onCreate(Database db, int version) {  
    return db.execute('''CREATE TABLE $_tableName(  
        id INTEGER PRIMARY KEY AUTOINCREMENT,  
        value INTEGER,  
        nombre TEXT,  
        createTime DATETIME)  
    ''');  
}
```

Figura C.5: Método para crear la tabla en la base de datos local

Cada uno de los campos de esta tabla significan:

- **id:** identificador de tipo entero autoincrementable. Solo se utiliza internamente, al final estas variables que almacena se van a identificar por el nombre.
- **value:** valor que toma esta variable de tipo entero, puede ser 0 o 1, para las de tipo *bool* o valores numéricos. Se hace así con el fin de abarcar estos dos tipos de datos.
- **nombre:** es de tipo String, usada para reconocer a cada una de las variables en la tabla.
- **createTime:** fecha en la que se añade a la tabla una nueva variable. El tipo de dato que almacena es *datetime*.

Un ejemplo de uso, lo podemos encontrar dentro de la aplicación en el apartado de *settings*, donde cada vez que se cambia el valor de un *slider* se actualiza en la base de datos. Esto se hace para que cuando el usuario vuelva a la aplicación, la configuración de la última vez que estuvo se siga manteniendo.

Otra de las formas en las que se almacenan los datos es mediante un fichero en local. Esto solo es usado para la configuración del menú, ya que cada vez que se crea el *Widget* del menú lateral, lee el *.json* para sacar los datos necesarios. Esta lógica de negocio se podría haber usado de la misma forma para solventar el problema de sqllite [30](#).

C.3. Diseño procedimental

En la gran mayoría de procesos importantes se han hecho diagramas de flujo, con el fin de resolver y comprender algunas de las lógicas de negocio más importantes. Un ejemplo de esto es el siguiente diagrama C.6 para la partida del snake:

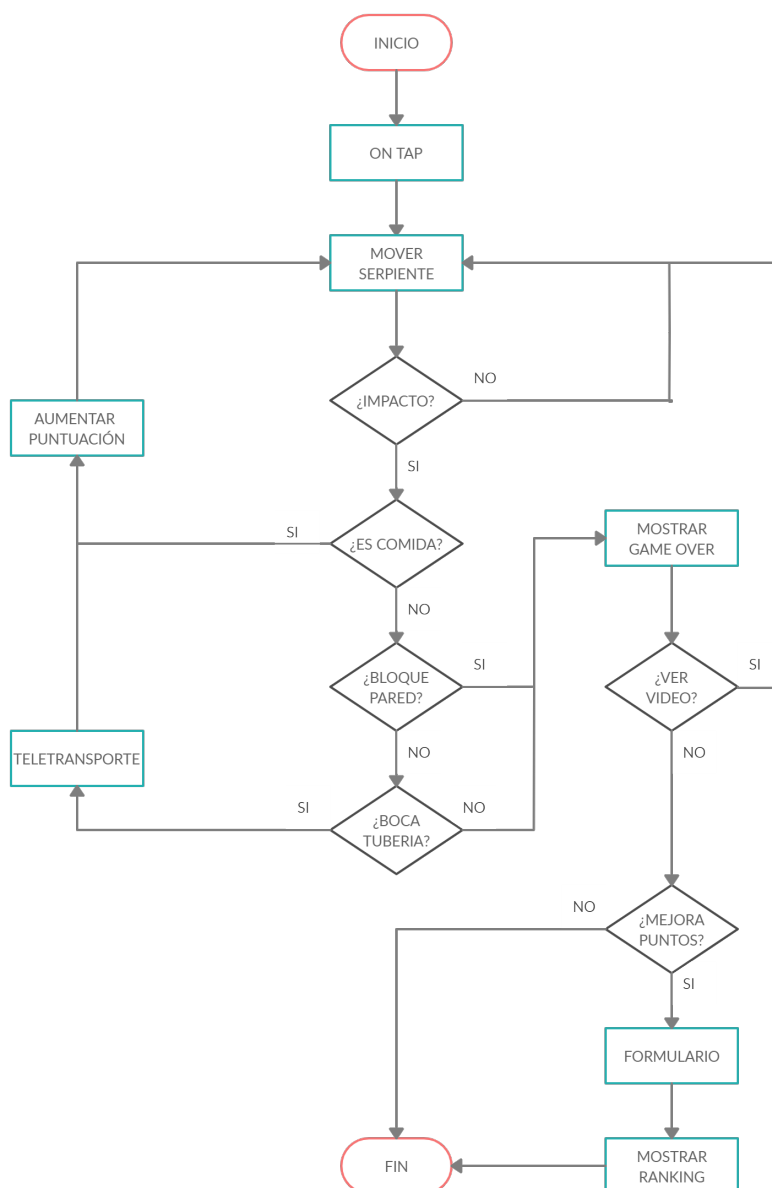


Figura C.6: Diagrama de flujo juego snake

C.4. Diseño arquitectónico

La arquitectura de la aplicación ha seguido el patrón de diseño BLoC [?]. Significa *Business Logic Component* 37. Fue creado por Paolo Soare y Cong Hu, ambos de Google y presentado en la conferencia de Dart en 2018. Por lo que es algo bastante nuevo.

Para comprender el por qué de esta arquitectura es necesario saber como funcionan los estados compartidos entre los componentes en los framework declarativos 36.

Framework declarativo

Los framework declarativos son aquellos donde las vistas se crean y actualizan en base a los datos con los que la vista esta enlazada, de tal forma que, cuando estos datos cambian de valor, se actualizan con una nueva renderización. Es decir, cada uno de estos componentes tiene un estado.

Los frameworks declarativos más conocidos son: *React.js*, *Angular*, *dart.js* o *Flutter*

En el caso de flutter, los componentes son conocidos como Widgets, y pueden tener 3 estados:

- **Sin estado:** no guardan información.
- **Con estado local:** los datos son del widget como puede ser la posición del scroll.
- **Con estado global:** para compartir datos entre diferentes widgets. Como puede ser la sesión de usuario.

Un ejemplo de declaración de widget que tiene el estado dinámico C.7, para este caso extiende de *statefulwidget*:

```
class RankPage extends StatefulWidget {  
  bool ads = false;  
  RankPage({this.ads});  
  
  @override  
  RankPageState createState() => RankPageState(ads: this.ads);  
}  
  
class RankPageState extends State<RankPage> {  
  bool ads = false;
```

Figura C.7: Estado Flutter

BLoC

Este patrón lo que pretende es que los componentes sean intermediarios entre las vistas y el modelo. Esta basado en la programación reactiva, utilizando el patrón observer, en Flutter es llamado *Streams*, lo dota de gran versatilidad.

Los objetivos cuando se presentó este patrón en la conferencia de Google eran tres:

- **Centralizar la lógica de negocio:** pretende crear aplicaciones sin una arquitectura definida, con componentes de gran tamaño, con toda la lógica de negocio, dónde se incluye también las llamadas a una web service o una API, todo dentro del widget.

La idea es que con este principio de inversión de la dependencia, estas clases solo tengan la lógica, permitiendo que la aplicación escale mejor o que se pueda integrar con diferentes tecnologías.

- **Centralizar los cambios de estados:** cada uno de los componentes es encargado de trabajar con los eventos que se producen, ya que estos son los que van a modificar los datos, y por lo tanto, el estado del componente.

El problema de esto es que renderiza Widgets que no han sufrido cambios, es decir, se cambia un padre y también se tienen que renderizar los hijos, lo que implica una carga de trabajo mayor.

- **Tener un mapa del formato de la vista:** los datos formatean la presentación, de tal manera que la vista se renderiza dependiendo de los datos. Lo que hace que sean reutilizables.

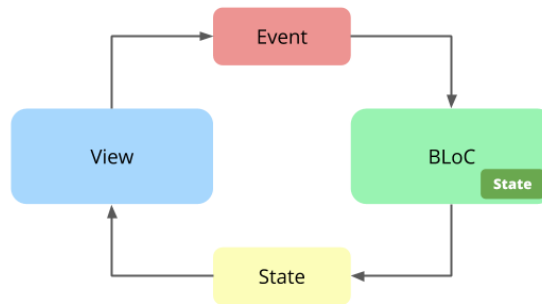


Figura C.8: Diseño bloc

Por lo que una presentada la arquitectura es BLoC, se pretendió tener internamente una estructura de directorios ordenada, dependiendo de cada uno de los componentes que integre, como se puede ver en la imagen [C.9](#)

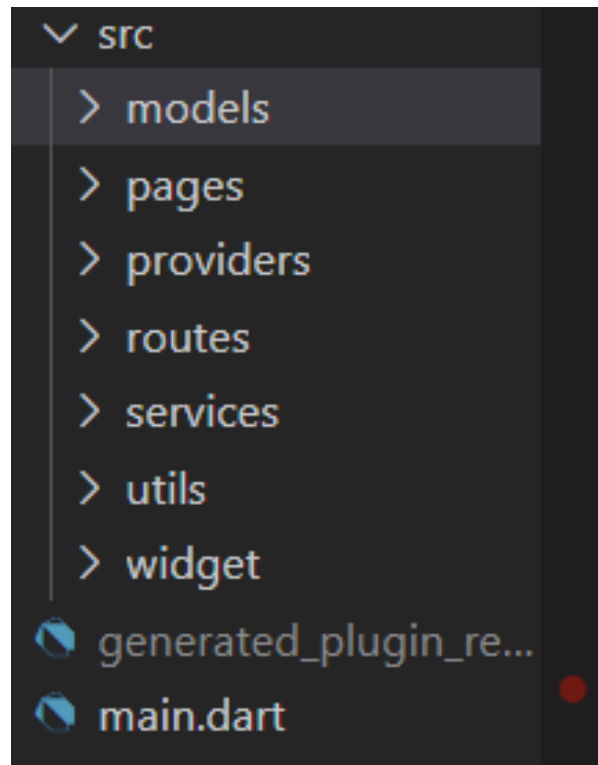


Figura C.9: Estructura de directorios aplicación

- **main.dart:** fichero del que se lanza toda la aplicación, es decir, es el componente raíz de la misma, de la que cuelgan el resto.
- **models:** ficheros que representan algún modelo de datos.
- **pages:** ficheros que contienen la vista de la aplicación, así como la lógica de negocio para cada una de estas.
- **providers:** ficheros que proveen (*future binding*) de contenido internamente.
- **routes:** directorio con el fichero del componente de las rutas. Facilita la navegabilidad, ya que las rutas ahora se pueden hacer mediante llamadas por nombre.
- **services:** componentes que proveen de servicios externos, como es el caso de Firebase.
- **utils:** componentes que tienen utilidades, de tal manera que puedan ser reutilizados.
- **widget:** ficheros que contienen componentes para que puedan ser reutilizados.

C.5. Diseño de interfaces

Como se comenta en el apéndice del especificación de requisitos [17](#), la mayor parte de las interfaces se diseñaron con el cliente, a través de bocetos, con la finalidad de facilitar la comunicación y el entendimiento cliente-analista. Un ejemplo de esto es la siguiente imagen [C.10](#).

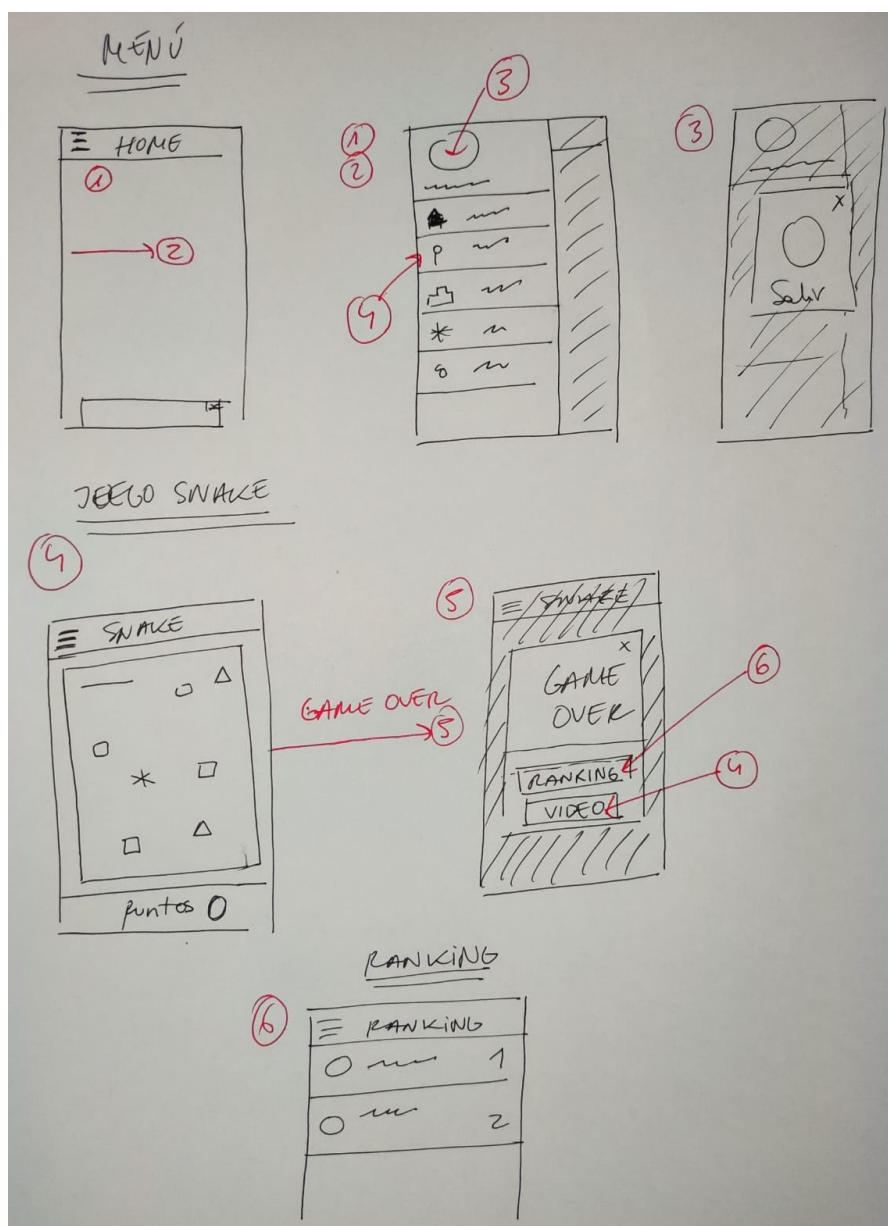


Figura C.10: Boceto de diseño

El diseño final de cada una de las interfaces de la aplicación se puede encontrar en el apéndice Manual de usuario [63](#).

Material Design

Gran parte de los componentes o widgets de la aplicación son provistos por la compañía Google y su paquete de diseño llamado: Material Design. Podemos encontrar en la [web¹](https://material.io/develop/flutter) toda la documentación. Por lo que esto nos ha garantizado tener unos patrones de diseño según el estándar de Android.

Para los juegos habría que testear si el diseño responsive se garantiza, ya que el diseño de los tableros está en contenedores que no son dinámicos. Lo que implica que con la gran heterogeneidad del ecosistema, encontremos fallos.

Paleta de colores

La paleta de colores es la ofrecida por Material Design Palette,. Se supone que garantiza un gran contraste para facilitar la lectura, como el reconocimiento correcto de los contornos. Como consecuencia lo dota de gran usabilidad.

En el caso que la paleta de colores no sea del agrado del usuario final, se puede meter dentro del apartado ‘settings’, para poner la aplicación de color negro, esto incrementa el contraste, lo que es una mejora de usabilidad.

¹<https://material.io/develop/flutter>

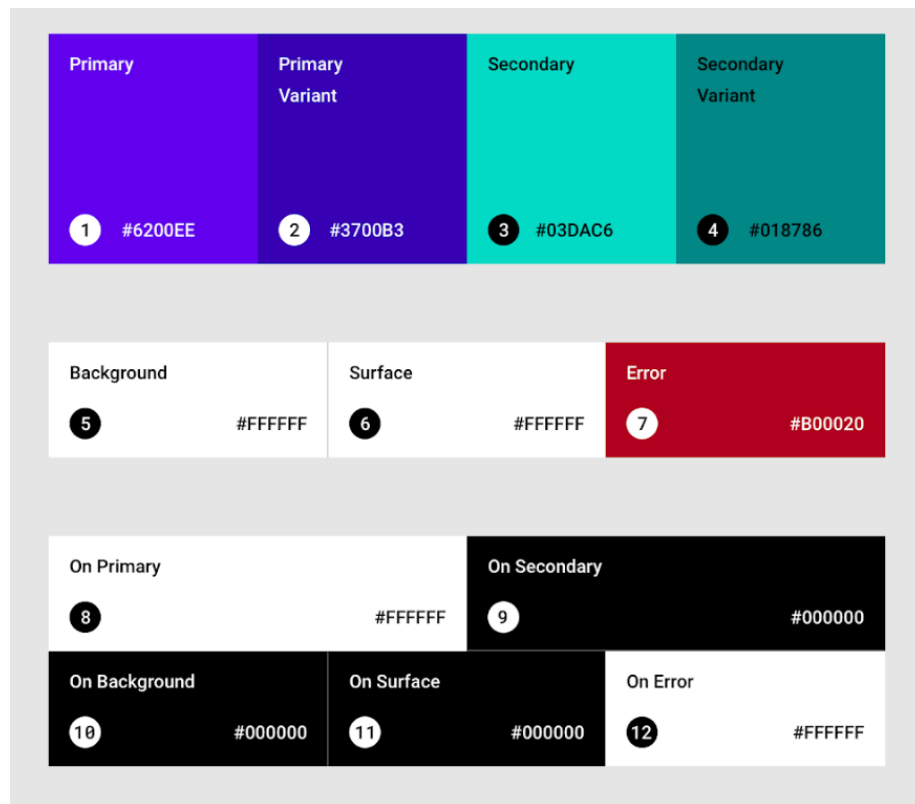


Figura C.11: Paleta de colores

Iconos

Los iconos son provistos por Google, también dentro del material design. La lista de los iconos se puede encontrar [aquí](https://material.io/resources/icons/?style=sharp)², un fragmento de esta lista [C.12](#):

²<https://material.io/resources/icons/?style=sharp>

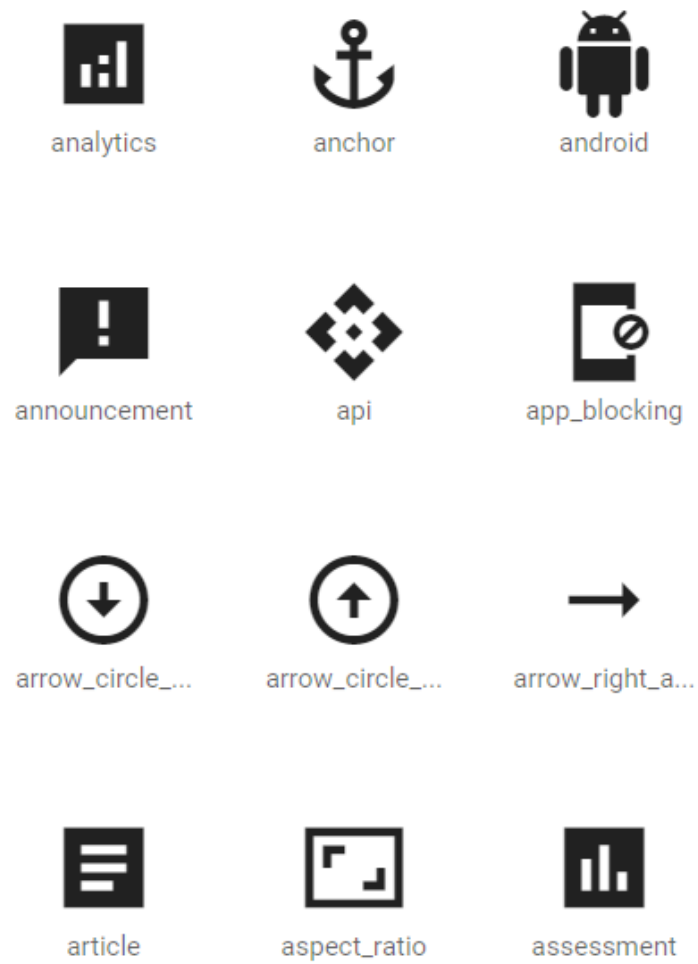


Figura C.12: Algunos iconos

Apéndice *D*

Documentación técnica de programación

D.1. Introducción

En este anexo se describe la documentación técnica de programación para este proyecto. Incluye los primeros pasos que son la instalación del proyecto, la estructura de la aplicación o finalmente como compilarlo, desplegarlo o los diferentes tipos de configuraciones realizados. La idea es poder facilitar a los futuros desarrolladores una guía con la que poder comenzar, en el caso de que quisieran continuar con el trabajo.

D.2. Estructura de directorios

El repositorio se encuentra alojado en [Github](https://github.com/scc0034/flutter_serpiente)¹. La estructura de ficheros más relevantes que tiene el proyecto es la siguiente:

- `./` Directorio raíz del que cuelgan todas los demás ficheros. Este contiene uno de los archivos más importantes, que es `pubspec.yaml`. Este archivos se usa para hacer las importaciones de los paquetes con las funcionalidades que queramos dar a nuestra aplicación.
- **build:** Este directorio contiene todo lo relativo a las compilaciones, es decir, tanto como para hacer las pruebas en local de la aplicación, o crear los *releases* que creamos oportunos. Además

¹https://github.com/scc0034/flutter_serpiente

contiene todo lo relativo a las conexiones con Android Studio y Firebase, ya que necesita hacer las llamadas a este para lanzar los emuladores con la máquina virtual correspondiente. Dentro de esta estructura algunos de los ficheros más importantes son:

- **key.properties:** propiedades de la key, ya que esta nos permite desplegar la aplicación en la *Play Store*. Es algo que no se tiene que perder ni modificar, ya que es de sumo valor.
- **app/google-services.json:** fichero que descargamos desde Firebase, para que la aplicación tenga las conexiones con este *Cloud service*, es decir, contienen las claves de conexión. En el caso de que tengamos que lanzar la app con otro de servicio de Firebase, podemos hacerlo cambiando este fichero.

```
"client": [  
  {  
    "client_info": {  
      "mobilesdk_app_id": "1:66474218378:android:dfca726c209b7a0ea06e73",  
      "android_client_info": {  
        "package_name": "com.ubu.flutter_snake"  
      }  
    },  
    "oauth_client": [  
      {  
        "client_id": "66474218378-c32kj0tepp9bleapoigc3ekmb5jpui5l.apps.googleusercontent.com",  
        "client_type": 1,  
        "android_info": {  
          "package_name": "com.ubu.flutter_snake",  
          "certificate_hash": "8aa991f820f74731872ee34b4f46e34b219c9b1a"  
        }  
      },  
      {  
        "client_id": "66474218378-ulikh2ufgq5m6o87ups31q1da50kk8d4.apps.googleusercontent.com",  
        "client_type": 3  
      }  
    ]  
  }  
]
```

Figura D.1: Google services

- **app/build.gradle:** Fichero que contiene lo necesario para hacer las compilaciones, ya que como vemos tiene el SDK mínimo y máximo con el que trabaja (limitando el número de dispositivos que son compatibles). El número de versión para cuando sea subido a la *Play Store*, es algo que se debe revisar siempre, ya que si no vamos a tener problemas de versionado. Además de los parámetros usados en la clave como podemos ver en la siguiente imagen.

```

defaultConfig {
    // TODO: Specify your own unique Application ID (https://developer.android.com/studio/run/application-id)
    //applicationId "com.example.flutter_snake"
    applicationId "com.ubu.flutter_snake"
    minSdkVersion 21 // Version de kitkat
    // minSdkVersion 16 // muy viejo
    targetSdkVersion 28
    versionCode 6
    versionName "6.0"
}

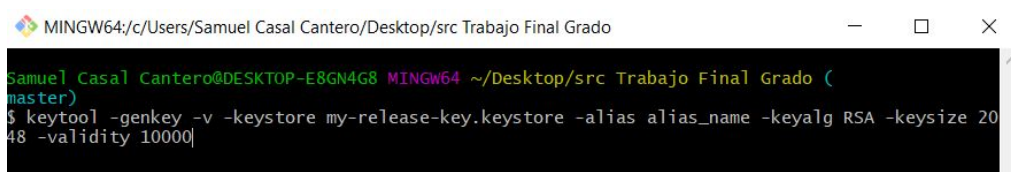
//NUEVO PLAY STORE
signingConfigs {
    release {
        keyAlias keystoreProperties['keyAlias']
        keyPassword keystoreProperties['keyPassword']
        storeFile file(keystoreProperties['storeFile'])
        storePassword keystoreProperties['storePassword']
    }
}

//FINAL PLAY STORE

```

Figura D.2: app/build.gradle

- **app/key/keysake.jks**: clave cifrada generada mediante el comando **D.3**, esta no se puede perder, ya que sin ella es imposible desplegar la aplicación en la *Play Store*. Es conveniente hacer alguna copia de seguridad en local.



A screenshot of a terminal window with a black background and green text. The window title is "MINGW64:/c/Users/Samuel Casal Cantero/Desktop/src Trabajo Final Grado". The prompt shows the user is "Samuel Casal Cantero@DESKTOP-E8GN4G8" in the directory "~/Desktop/src Trabajo Final Grado". The command entered is: `$ keytool -genkey -v -keystore my-release-key.keystore -alias alias_name -keyalg RSA -keysize 2048 -validity 10000`.

Figura D.3: Comando generar clave keyStore

- **assets**: directorio que contiene los archivos de imágenes y audio de la aplicación.
- **data/menu-opt.json**: fichero *json* con la estructura del menú *drawer*, con los nombres de ruta, nombre de icono y nombre de la página.
- **docs/latex**: memoria y anexos del trabajo final de grado.
- **lib**: contiene los ficheros que se compilan para crear la aplicación. La estructura interna de directorios es la siguiente:

- **main.dart**: fichero principal del que cuelga toda la aplicación.
- **models**: contiene los modelos para crear las instancias de los objetos de la base de datos.
- **pages**: cada una de las interfaces o páginas de la aplicación.
- **providers**: proveedores de algunos servicios internos.
- **routes**: rutas de la aplicación.
- **services**: instancias a los servicios externos de la aplicación.
- **utils**: utilidades.
- **widgets**: elementos gráficos reutilizables.

D.3. Manual del programador

Este manual del programador tiene como objetivo ayudar a personas que en un futuro estén interesadas en la continuación del proyecto, o simplemente que tengan ganas de aprender como se ha realizado. Para ello debemos de realizar las instalaciones de las siguientes herramientas:

- Flutter.
- Android Studio.
- Visual Studio Code.
- Git.
- Firebase Console.
- Play Store Console.

Flutter:

Es un SDK de código abierto creado por Google. Ofrece la versatilidad de crear aplicaciones móviles tanto como para *iOS* y *Android*, pero también para el entorno web. Internamente el lenguaje de programación es Dart.js, un *framework*, que también es *Open Source* desarrollado por Google, ya que pretende mejorar algunas de las carencias que tiene *javascript*.

Para la instalación de Flutter debemos de ir a su [web oficial](https://flutter.dev/docs/get-started/install)² y descargarlo, dependiendo del sistema operativo que tengamos usaremos una versión u otra. (v.17.5 *stable*).

Al descomprimir el fichero nos daremos cuenta de que tenemos un directorio (no ejecutable), por lo que debemos de dejar este en un lugar en

²<https://flutter.dev/docs/get-started/install>

concreto para después añadirlo a las variables de usuario. En mi caso lo dejé en la siguiente ruta de mi ordenador personal `C:\src\flutter`.

Una vez realizado el paso anterior debemos de actualizar las variables de usuario, apuntando al directorio bin de la ruta anterior, como vemos en la siguiente imagen:

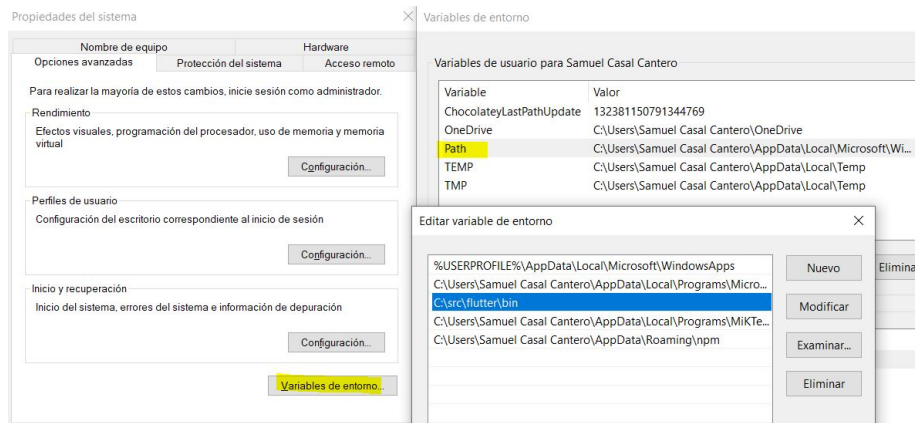


Figura D.4: Variables entorno

Por defecto la rama de Flutter es la *stable*, de las ramas disponibles: *stable*, *beta*, *dev*, *master*. Pero si en el caso de necesitar la misma versión de Flutter, el comando necesario es:

```
Samuel Casal Cantero@DESKTOP-E8GN4G8 MINGW64 ~/Desktop/src Trabajo Final Grado (master)
$ flutter --version
Flutter 1.17.5 • channel stable • https://github.com/flutter/flutter.git
Framework • revision 8af6b2f038 (12 days ago) • 2020-06-30 12:53:55 -0700
Engine • revision ee76268252
Tools • Dart 2.8.4

Samuel Casal Cantero@DESKTOP-E8GN4G8 MINGW64 ~/Desktop/src Trabajo Final Grado (master)
$ flutter version v.1.17.5]
```

Figura D.5: Comando version Flutter

Android Studio:

Es el IDE oficial de Google, que remplacea a Eclipse, basado en *IntelliJ IDEA*, publicado de manera gratuita con Licencia Apache 2.0.

En nuestro caso, no programaremos en este IDE, ya que es complicado trabajar con Flutter directamente, solo lo vamos a utilizar para la creación de las máquinas virtuales para emular el sistema operativo Android. Por lo

que descargamos de la [web oficial Android Studio](https://developer.android.com/studio)³ e instalamos, en mi caso utilicé la versión 3.6.

Una vez instalado, creamos las máquinas virtuales. Para ello se va a la barra de herramientas Tools>AVD manager y se crean las que sean necesarias,. En mi caso tengo dos, ambas con Android R, que es la versión 10 de sistema operativo, ya que es la última estable que nos ofrece el IDE.

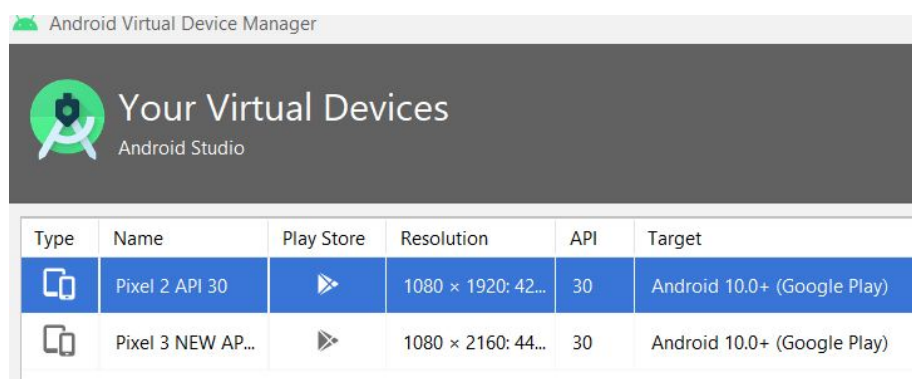


Figura D.6: Máquinas virtuales

Visual Studio Code:

Es el editor de código creado por Microsoft, permite mucha versatilidad y es eficiente para la edición del código en flutter. Es de código abierto bajo la licencia MIT.

Lo podemos descargar de la [web VS Code](https://code.visualstudio.com/)⁴, en mi caso uso la versión 1.47.

Una vez lo tengamos instalado, procedemos a instalar los siguientes *Snippets* que nos ayudan a la creación del código, ya que son como los atajos de teclado. En cada uno de los enlaces, podemos ver una descripción de cada una de estas herramientas:

- [Awesome Flutter Snippets](https://marketplace.visualstudio.com/items?itemName=Nash.awesome-flutter-snippets)⁵. Apache 2.0.

³<https://developer.android.com/studio>

⁴<https://code.visualstudio.com/>

⁵<https://marketplace.visualstudio.com/items?itemName=Nash.awesome-flutter-snippets>

- [Bracket Pair Colorizer 2](#)⁶. MIT License.
- [Dart](#)⁷. MIT License.
- [Flutter](#)⁸. MIT License.

Git:

Es un software para el control de las versiones, que nos permite trabajar mediante comandos desde *Git Bash*, la licencia es GNU-GPL v2. La última versión estable con la que he trabajado es 2.27 y la podemos encontrar [web oficial](#)⁹.

Firebase Console:

La consola de [Firebase](#)¹⁰ es la que nos permite el control de todas las características de *Cloud Services*, desde la autenticación de los usuarios, base de datos o AdMob services, entre otras muchas.

Al estar registrado con mi cuenta personal, tendría que dar permisos de administrador en el caso de que fuera necesario, para ello pueden mandarme un correo a casalyepa@gmail.com¹¹

⁶<https://marketplace.visualstudio.com/items?itemName=CoenraadS.bracket-pair-colorizer-2>

⁷<https://marketplace.visualstudio.com/items?itemName=Dart-Code.dart-code>

⁸<https://marketplace.visualstudio.com/items?itemName=Dart-Code.flutter>

⁹<https://git-scm.com/>

¹⁰<https://console.firebase.google.com/?hl=es>

¹¹<mailto:casalyepa@gmail.com>

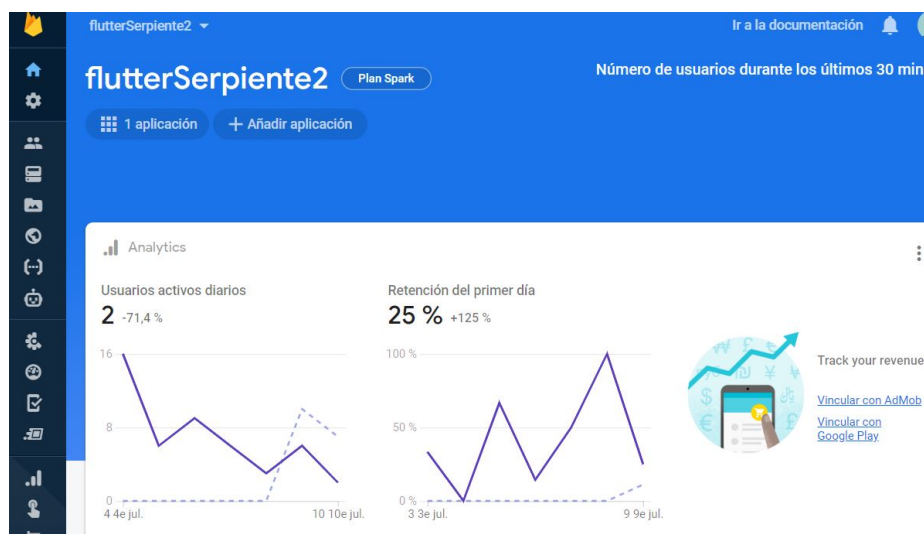


Figura D.7: Firebase console

Play Store Console:

La consola de **Play Store**¹² es la herramienta web que nos permite hacer el despliegue de la aplicación, hacer las pruebas alfa y beta, o para subirla y que todo el mundo se la pueda descargar.

La revisión de cada una de las nuevas versiones puede tardar en ser validada, por lo que se debe tener paciencia.

Al igual que con Firebase, tendría que dar permisos en el caso de que sea necesario hacer nuevos despliegues de la app, correo a **casalyepa@gmail.com**¹³

¹²<https://play.google.com/apps>

¹³<mailto:casalyepa@gmail.com>

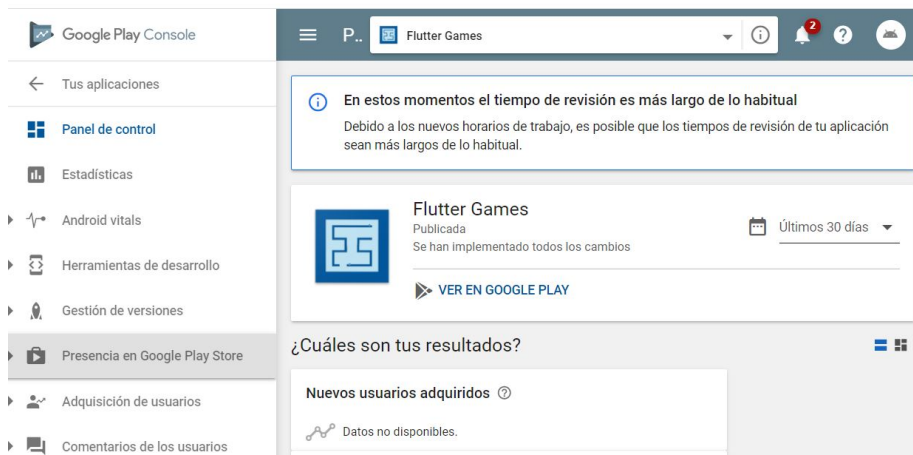


Figura D.8: Play Store console

D.4. Compilación, instalación y ejecución del proyecto

En este apartado se muestran los pasos necesarios para hacer una copia del proyecto en Github y poder trabajar en local desplegando la aplicación en los emuladores.

Clonación del proyecto:

1. Nos colocamos en el directorio donde queramos tener el proyecto y abrimos *git bash* mediante el botón derecho:

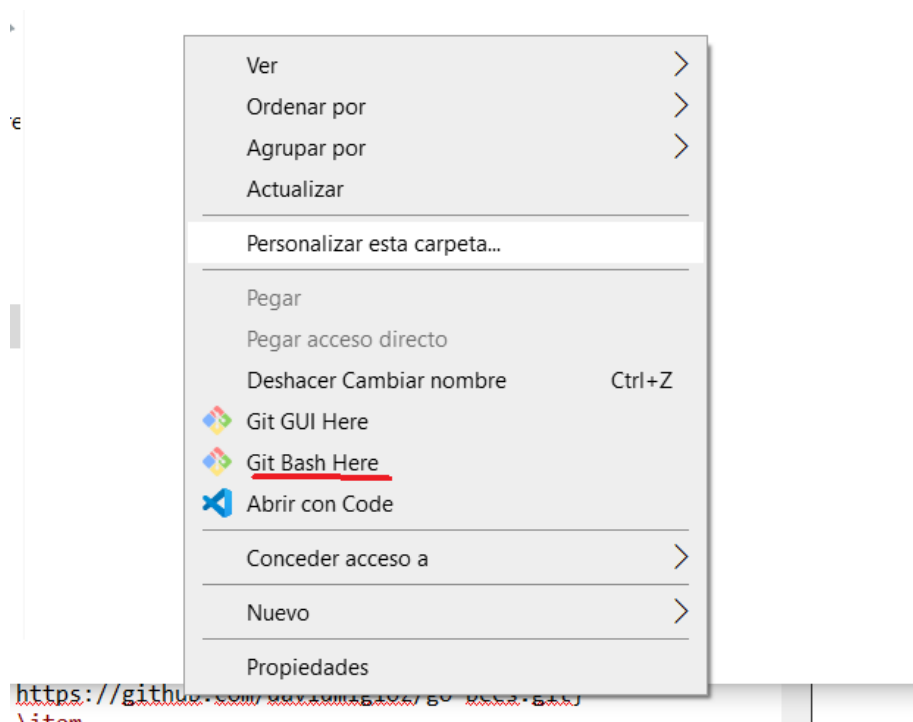


Figura D.9: Abrir git bash

2. Procedemos a clonar el proyecto con el comando : `git clone https://github.com/scc0034/flutter_serpiente.git` D.10

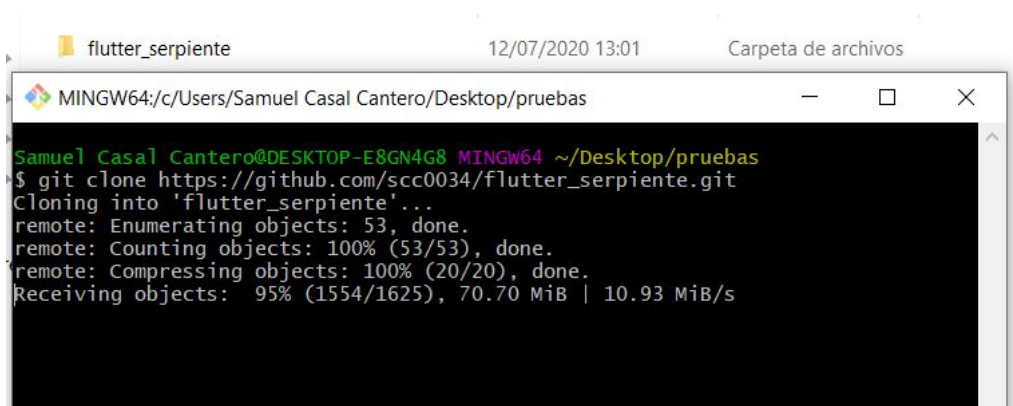


Figura D.10: Comando para clonar el repositorio.

Instalación de los paquetes:

Una vez esté el proyecto clonado, se debe ejecutar el comando `flutter upgrade`, para traer todos los paquetes, ya que si no en las importaciones de cada uno de los ficheros van a aparecer como erróneas.



```
Samuel Casal Cantero@DESKTOP-E8GN4G8 MINGW64 ~/Desktop/src Trabajo Final Grado (master)
$ flutter upgrade
Flutter is already up to date on channel stable
Flutter 1.17.5 • channel stable • https://github.com/flutter/flutter.git
Framework • revision 8af6b2f038 (12 days ago) • 2020-06-30 12:53:55 -0700
Engine • revision ee76268252
Tools • Dart 2.8.4
```

Figura D.11: Actualización de paquetes

Una vez realizado debemos reiniciar Visual Studio Code.

Ejecución del proyecto:

Se puede hacer de tres maneras diferentes:

1. Descargar la última versión subida a la [Play Store](https://play.google.com/store/apps/details?id=com.ubu.flutter_snake)¹⁴, esto solo es válido para el despliegue de la aplicación, no para desarrollo. Por los tiempos de despliegue.
2. Desde la consola de git, `flutter devices`, para ver que dispositivos tenemos disponibles, en el caso de que dentro de la lista no se muestre nada, se debe de ir a Android Studio y arrancar una de las máquinas virtuales [D.12](#). Si la lista presenta dispositivos, ejecutamos el siguiente comando `flutter run -d emulator-id`, siendo el id, el número de la máquina virtual que muestra el comando anterior.
3. Desde Visual Studio Code (se debe tener las herramientas de visual studio), se da al **F5**, se mostrará una lista con los dispositivos encontrados en el sistema. En el caso de que se disponga de un móvil físico conectado, aparecerá también en la lista anterior. Finalmente arrancará la aplicación en el terminal seleccionado.[D.12](#).

¹⁴https://play.google.com/store/apps/details?id=com.ubu.flutter_snake

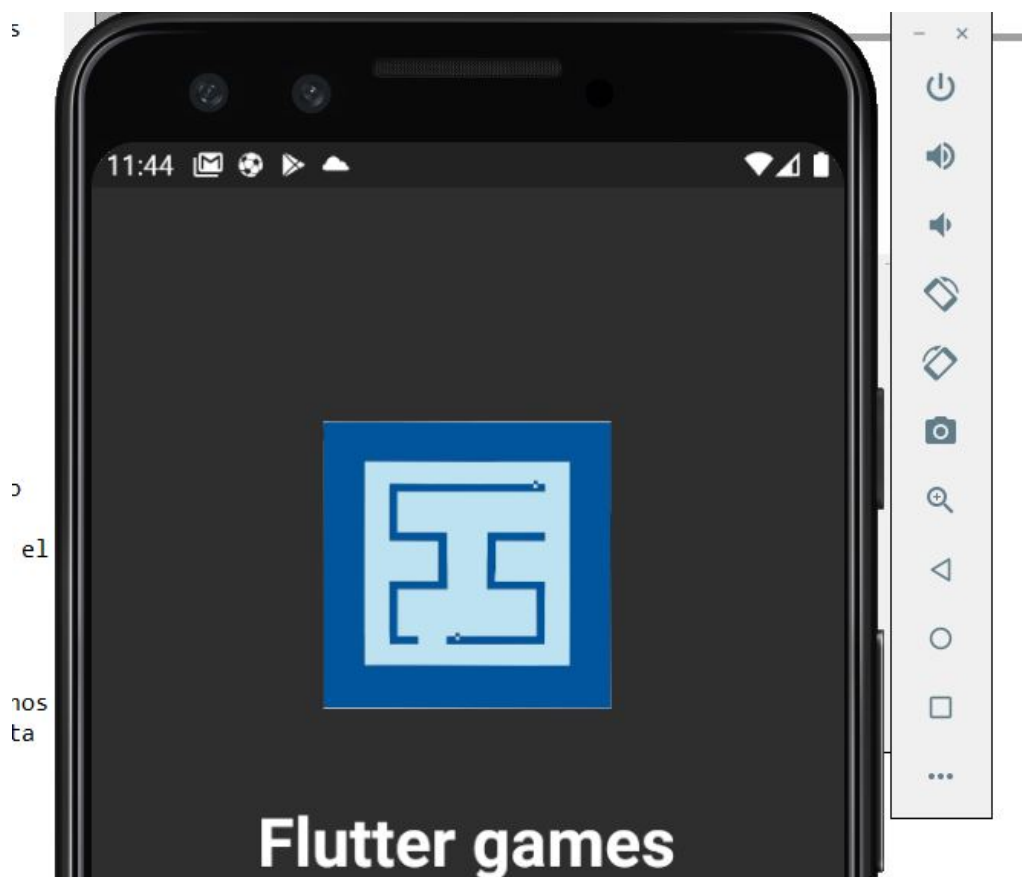


Figura D.12: Máquina virtual

Generar los apks

Hay dos opciones para generar los ficheros, dependiendo del destino de cada uno de ellos, es decir, las opciones que se encuentran son la siguientes:

1. **Para la tienda:** Generar el *bundle* [?] para hacer el despliegue en *Play Store*, para ello el comando que se ha de escribir en la consola es: `flutter build appbundle`. Se genera un archivo *.aab*, de tal forma que dependiendo de que dispositivo descargue la app, desde la tienda, se baje unos archivos u otros, con el fin de que la aplicación pese menos. Antes de generar el nuevo *bundle*, se debe mirar que versión de la app tiene la tienda, para cambiar la versión de la aplicación en local a una mayor, de tal forma que al subirla no se den problemas de

incompatibilidad. En la salida por pantalla del comando nos dice en que ruta se encuentran los ficheros generados.

2. **Apks:** Generar las apks para cada uno de los sistemas: `flutter build apk -releases`. En la ejecución del comando se muestra la ruta donde se encuentran los ficheros generados.

D.5. Pruebas del sistema

Para validar el correcto funcionamiento del producto, se han realizado diferentes tipos de pruebas: unitarias, de integración y de sistema. Hay que destacar que Flutter tiene herramientas para la automatización de test, pero debido del tiempo de que se disponía para finalizar el proyecto, no se han realizado mediante estas, ya que no se pueden grabar los test como con *Selenium*, por poner un ejemplo similar.

Es decir, se puede considerar como línea de trabajo futura, con metodologías de programación dirigida por los test, como es el caso de Test-Driven Development.

Pruebas unitarias:

Se han realizado de forma simple o introduciendo impresiones explícitas en el propio código y mostrándolos por el terminal en tiempo de ejecución. En el caso de continuar con el desarrollo sería imprescindible trabajar con test automáticos. Han validado pequeños módulos o fragmentos de código, con el fin de garantizar de que estas pequeñas unidades funcionen. Uno de los ejemplos es [D.13](#):

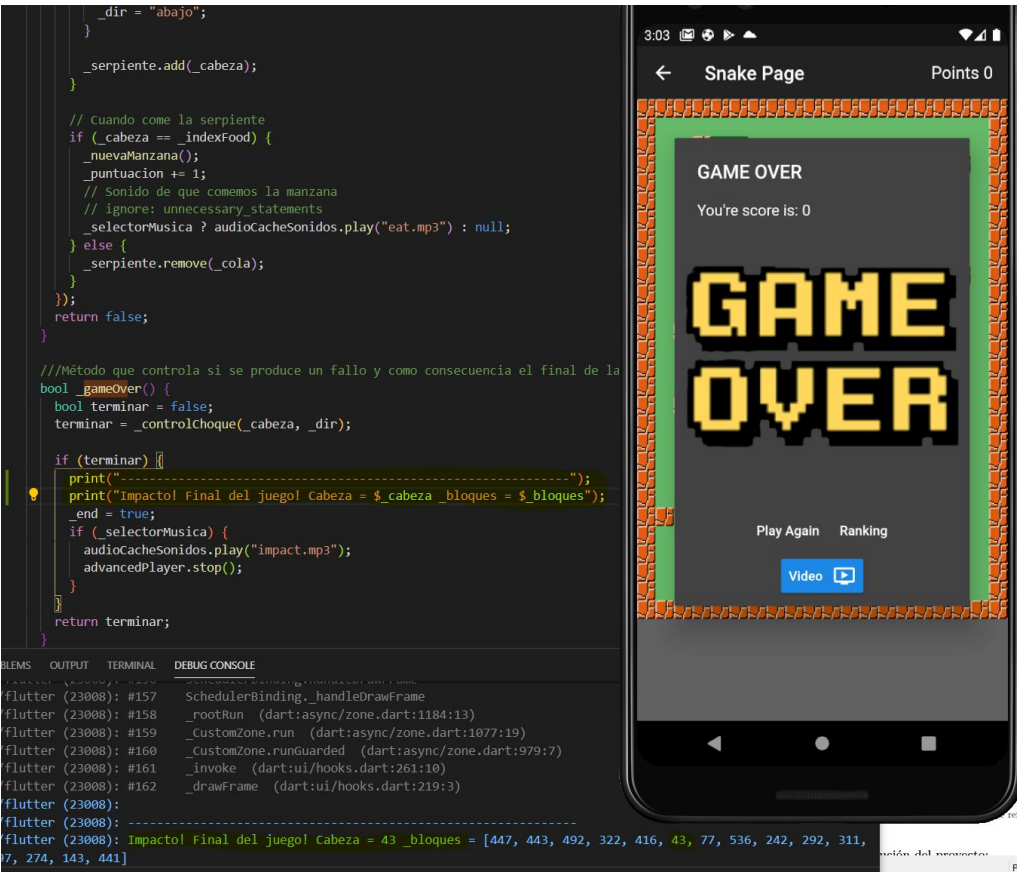


Figura D.13: Máquina virtual

Pruebas de caja negra:

En ocasiones se han realizado casos de prueba de caja negra para algunas líneas de código D.14. Para realizar las pruebas es necesario construir las tablas con clases de equivalencia válidas y no válidas D.1. Después la tabla de los diferentes casos de prueba con los resultados obtenidos D.2.

```

/// Método para controlar si se produce un impacto con algo
/// p es el punto donde se mira si tenemos el choque
/// dir es la dirección del choque para el caso de la tubería
bool _controlChoque(int p, String dir) {
    if(p<0 && p>nCasillas){
        print("Error, punto fuera del tablero.");
    }

    if(dir !=null || dir.compareTo("der") != 0 || dir.compareTo("izq") != 0 ||
    dir.compareTo("abajo") != 0 || dir.compareTo("arriba") != 0){
        print("Error en las direccion.");
    }

    if (_pared.contains(p) || _bloques.contains(p)) {
        return true;
    }
    // Choque contra la propia serpiente
    if (_serpiente.sublist(0, _serpiente.length - 2).contains(p)) {
        return true;
    }
    // Control de choque contra tuberia
    if (_tuberia.contains(p) && dir == null) {
        return true;
    }
    if (dir != null && _tuberia.contains(p)) {
        if (SnakeModel.dirOpuesta[dir] != _tuberiaDir[_tuberia.indexOf(p)]) {
            return true;
        }
    }
    return false;
}

```

Figura D.14: Código para las pruebas de caja negra

Dando como resultado las dos tablas siguientes:

La primera de las tablas define las clases de equivalencia que son válidas y las que no.

Condición entrada	Clase válida	Clase no válida
p dentro del tablero	1. $p > 0$ AND $p < n$ celdas	2. $p < 0$ AND $p > n$ celdas
dir debe ser válida	3. null, der, izq, abajo, arriba	4. dir no válida

Tabla D.1: Clases de equivalencia

La segunda tabla es una comprobación de lo que sucede para cada una de las clases de equivalencia. Dependiendo de si se produce choque con alguno de los elementos se devuelve true, en caso contrario false.

Entrada	Clase cubierta	Resultado
p = 10	1(clase válida)	return true/false
p = -2	2	Mensaje error
p = 600	2	Mensaje error
dir = null	3(clase válida)	return true/false
dir = der	3(clase válida)	return true/false
dir = izq	3(clase válida)	return true/false
dir = abajo	3(clase válida)	return true/false
dir = arriba	3(clase válida)	return true/false
dir = diag	4	Mensaje error

Tabla D.2: Derivación en casos de prueba de caja negra

Tras realizar los casos de caja negra, como es el caso anterior, me he dado cuenta de que es imprescindible la automatización de los test. Ya que nos ahorra gran cantidad de tiempo, y en el momento que la aplicación falle, veremos si se produce el error.

Pruebas de integración y sistema:

Hay que confirmar que el conjunto de módulos anteriormente validados, funcionen todos de manera interrelacionada unos con otros. Para ello lo que se ha hecho es un testeo de funcionamiento en el terminal físico, lo que conlleva un *testing* simultáneo de la interfaz.

Muchos de los errores son más fáciles de encontrar así, porque en la máquina virtual no funciona de la misma manera, ya que el rendimiento no es el mismo, o las simulaciones son parciales. Uno de los ejemplos que fue necesario hacer *in situ*, fue el juego online, a la hora de tener que validar que se producían las conexiones entre los terminales.

En estas pruebas de interfaz, se encontraron algunos campos que no eran visibles de manera correcta o que su usabilidad era reducida. Lo que supuso la realización de la mejora correspondiente.

Pruebas de despliegue de la aplicación

Este tipo de pruebas es algo muy complejo de controlar, ya que, hasta desplegamos la aplicación en la tienda, algunos de los errores permanecen enmascarados, durante el proceso de implementación. Esto es debido a trabajar con máquinas virtuales, capadas en algunos aspectos o simplemente que es un compilado de la aplicación instalada, sin tener que pasar por la tienda. O simplemente que no es el entorno real de usuario.

Por lo que para ello sería muy recomendable hacer unas pruebas de beta cerrada, ya que la plataforma de *Play Store Console*, tenemos opciones para ello.

Apéndice *E*

Documentación de usuario

E.1. Introducción

En este apartado se recoge todo lo que un usuario necesita conocer para poder ejecutar la aplicación en su teléfono móvil Android personal y los requisitos mínimos necesarios.

E.2. Requisitos de usuarios

Los requerimientos necesarios para poder ejecutar la aplicación en el teléfono son:

- Disponer de un terminal que al menos tenga la versión de Android en *Lollipop*. Esto se debe a que cuando se compila la aplicación en Flutter, algunos de los servicios necesitan al menos que el SDK sea el 21 [?]. Limitando por abajo el número de sistemas operativos compatibles.
- Es necesario disponer de conexión a Internet, tanto como para bajarse la aplicación en *Store*, como para usar los servicios, ya que es necesario logearse con *Google*
- La clasificación de contenido, es PEGI 3 [?].
- En el caso de que la descarga se realice mediante la tienda oficial, los países para los que la aplicación está disponible son: España, Francia, Portugal e Irlanda. En el caso de que no sea así, será necesario descargar desde Github, como se explica en la página 64.

E.3. Instalación

La instalación se puede hacer a través de dos métodos diferentes:

GitHub:

Desde el repositorio donde está el proyecto en el apartado de las *releases* [?], podemos encontrar la última de las versiones compiladas, con el fin de descargarla.

Al ser una aplicación de orígenes desconocidos, ya que no se descarga de la tienda se deben de seguir los siguientes pasos para la instalación:

1. Ir a los ajustes del terminal.
2. Apartado de privacidad o seguridad.
3. Activar el *slider* de ‘Orígenes desconocidos’.
4. Ejecutar el fichero .apk que acabamos de descargar.
5. Instalar y abrir la aplicación.

Play Store:

La manera más cómoda de hacerlo, es dirigirse [Flutter games¹](https://play.google.com/store/apps/details?id=com.ubu.flutter_snake), que es el enlace de descarga para dispositivos móviles Android. La versión disponible es la 6, ya que se han tenido que realizar varias pruebas con el fin de validar la disponibilidad, por eso el número de versión que tiene.

¹https://play.google.com/store/apps/details?id=com.ubu.flutter_snake

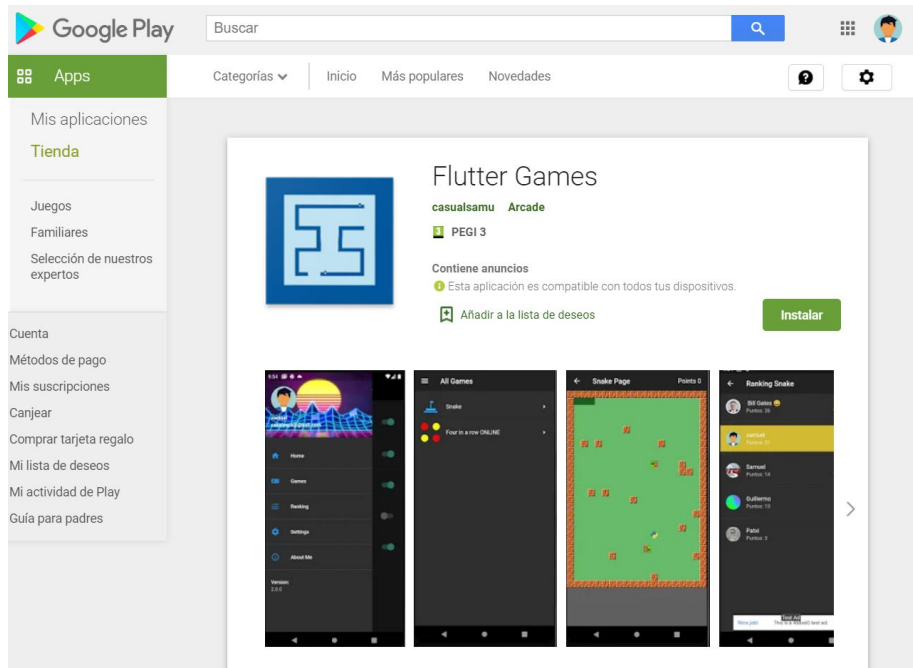


Figura E.1: Tienda con el proyecto

El proceso de instalación es simple, solo se tiene que dar al botón de instalar. En el caso de que nos encontremos en el navegador web, deja elegir el dispositivo que tenemos vinculado a nuestra cuenta de *Google*. No sucede lo mismo si nos encontramos desde el terminal, ya se instalará sin ningún problema.

En el caso de que se lancen nuevas versiones del producto, las actualizaciones se realizarán de forma automática, en el caso de estar activadas.

E.4. Manual de usuario

Se pretende mostrar el funcionamiento de cada una de las ventanas que están disponibles en la aplicación, con el fin de poder informar a los usuarios del funcionamiento de cada una de estas.

Las ventanas de las que consta la aplicación son las siguientes:

- Log in, ver pág. 66.
- Home, ver pág. 68.
- About, ver pág. 72.

- Settings, ver pág. 71.
- Menú de juegos, ver pág. 74.
- Snake, ver pág. 75.
- Ranking, ver pág. 79.
- Cuatro en raya online menú, ver pág. 80:.
 - Invitar, ver pág. 81.
 - Unirse, ver pág. 83.
 - Juego, ver pág. 85.

Log in

Es la primera ventana que nos encontramos cuando lanzamos la aplicación. Es necesario que nos registremos siempre con la cuenta de *Google* que tengamos disponible, ya que muchos de los servicios están en la nube, no está permitido el acceso mediante anonimato.

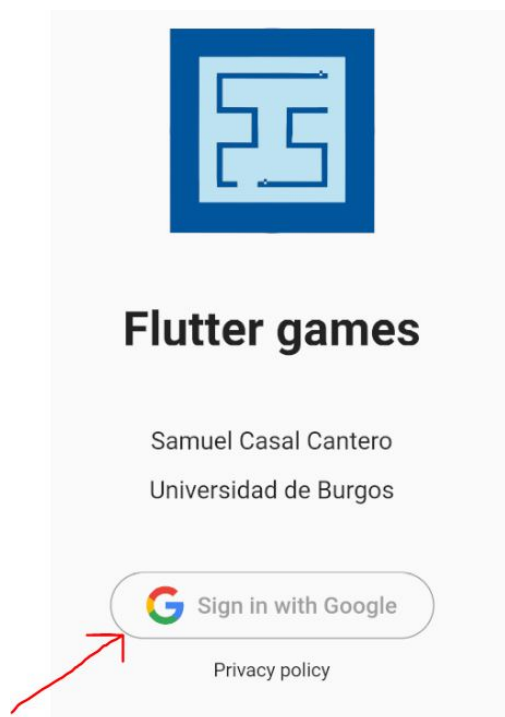


Figura E.2: Log in page

Cuando pulsamos en el botón de *sign in with Google*, nos aparecerá la ventana siguiente (ver figura E.3), dónde nos pedirá que ingresemos el

usuario de nuestra cuenta de *Google*. También se puede consultar la política de privacidad, como se puede ver en la figura E.4

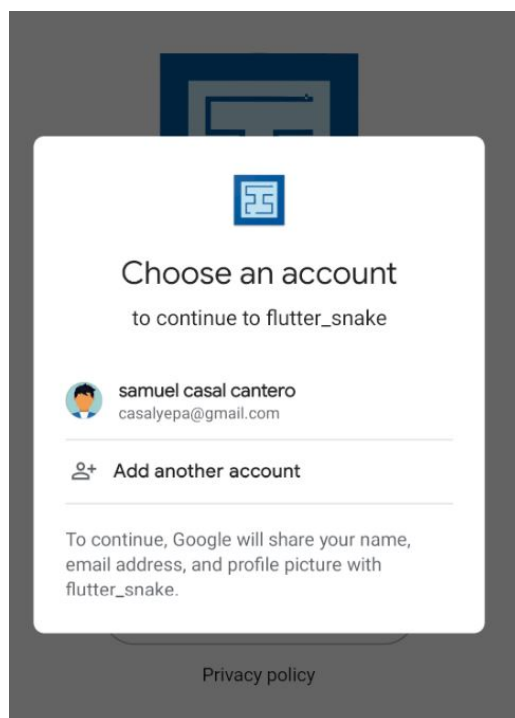


Figura E.3: Formulario sign in

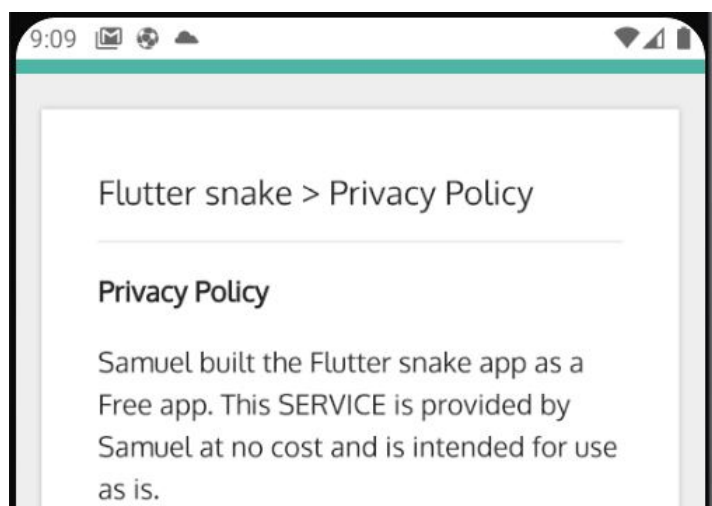


Figura E.4: Política de privacidad

Una vez completemos este proceso, la ventana siguiente a la que nos dirige la aplicación es el Home, ver figura E.5.

Home

Esta página, es donde se muestra el póster del proyecto, además de tener el acceso al menú lateral de la aplicación. Para acceder a este tenemos que deslizar lateralmente a la derecha, y para cerrarlo, tenemos que hacer el proceso inverso deslizando hacia la izquierda o pulsando fuera del menú, ver figura E.7. Otra de las formas de entrar a este es pulsando el menú hamburguesa.

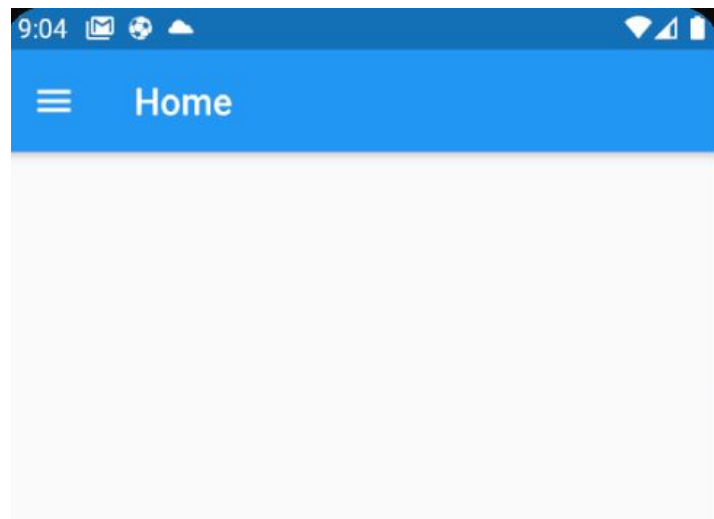


Figura E.5: Home

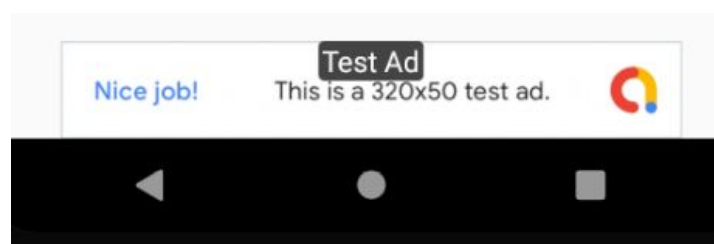


Figura E.6: Banner

Como podemos ver en la figura E.6, tenemos un *banner* donde se nos muestra la publicidad, en el caso de que pinchemos en él, nos abrirá el navega-

dor para mostrarnos más datos referentes al producto que se está anunciando. Es algo que aparecerá durante el resto de la aplicación dependiendo de la página en la que nos encontremos.

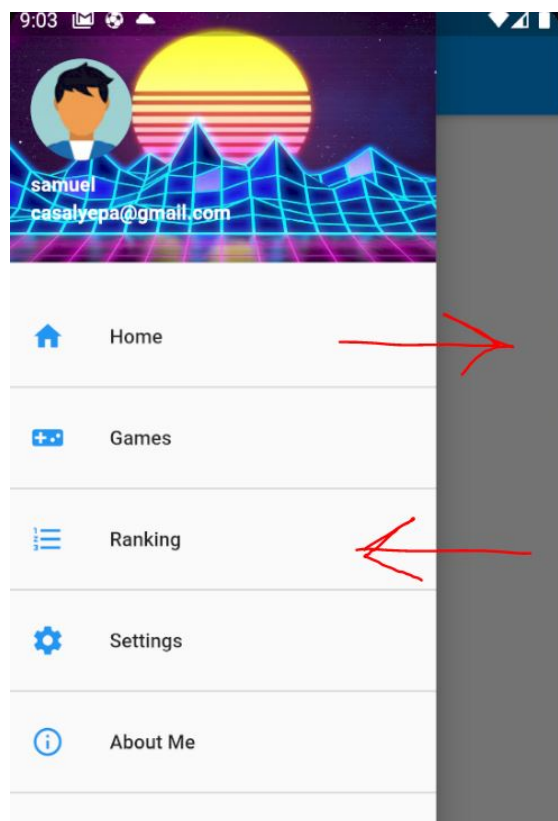


Figura E.7: Menú lateral

Como se aprecia en el menú, ver figura E.7, se encuentra la siguiente lista de opciones, dependiendo de cual sea la que se pulse nos redirigirá a la ventana correspondiente:

- Home, ver figura E.5.
- Games, ver figura E.14.
- Ranking, ver figura E.19.
- Settings, ver figura E.10.
- About me, ver figura E.12.

En el caso de que se quiera salir de la sesión que tenemos abierta, se debe pulsar en nuestra figura E.8 y aparecerá un cuadro de alerta con las opciones disponibles, se pulsará en *Sign out*, ver figura E.9.

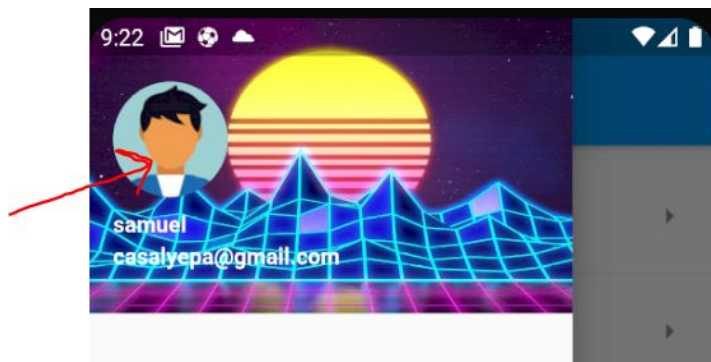


Figura E.8: Mostrar sign out

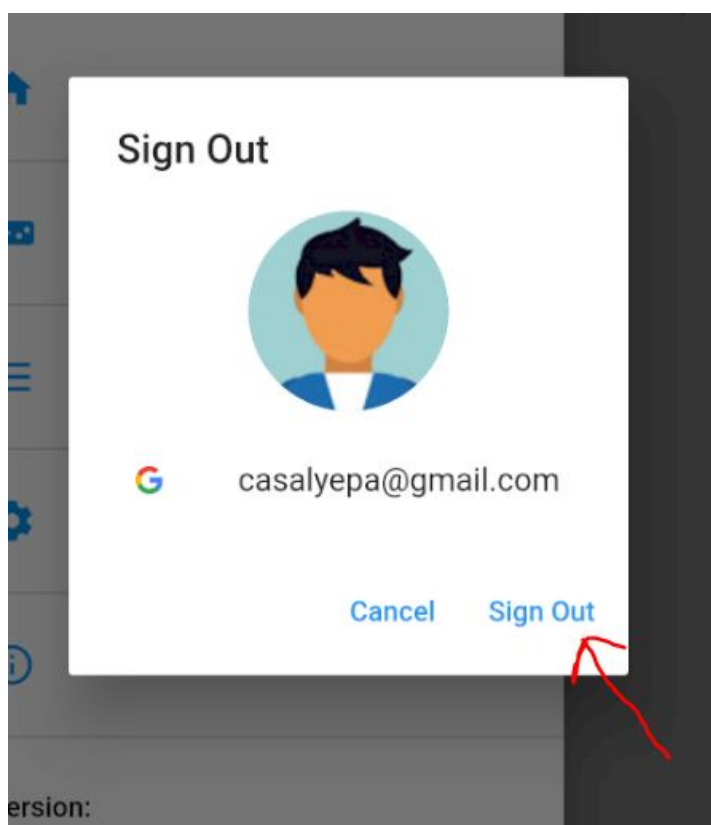


Figura E.9: Sign Out

Settings

Este apartado del menú, es para manejar las opciones referentes a la aplicación, como podemos ver en la figura E.10. Cada una de estas, es un *slider* o deslizador, que si está activado, quiere decir que está a *true* o verdadero.

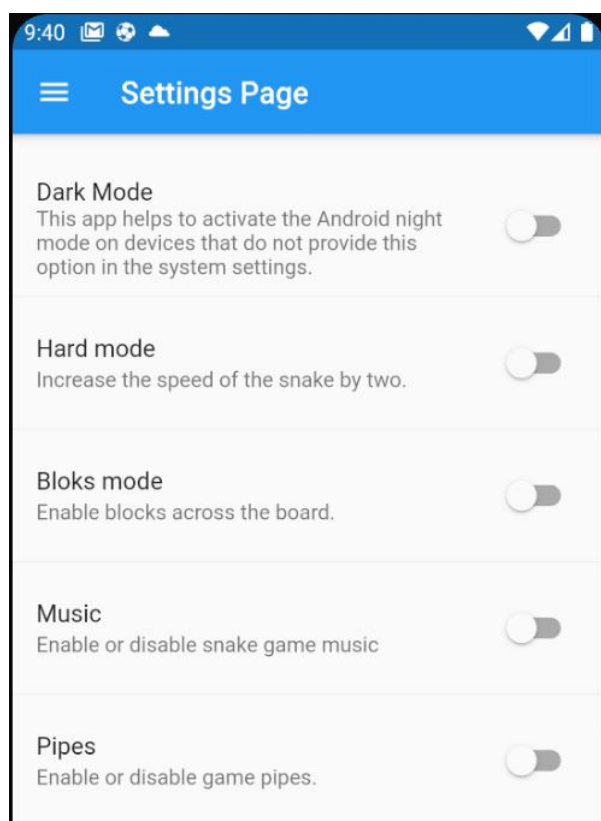


Figura E.10: Settings

Una descripción de lo que hacen cada uno de los ajustes:

- **Dark mode:** cambia el color de la aplicación a modo oscuro, con el fin de ahorrar batería o para personas que tengan dificultades visuales, ver figura E.11.

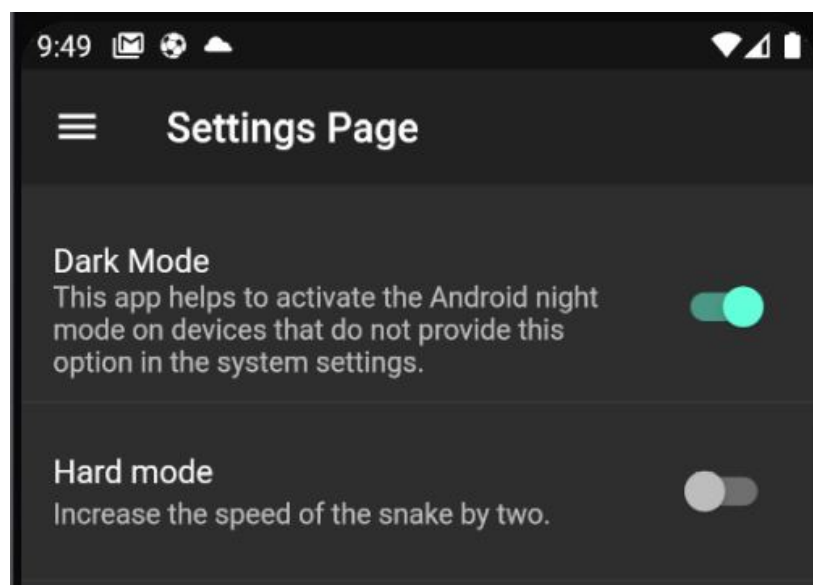


Figura E.11: Dark Mode

- **Hard mode:** aumenta la velocidad del snake, ver figura E.15, para que sea el doble de rápido.
- **Blocks mode:** reparte bloques a través del tablero del snake, en figura E.15, para tener una dificultad añadida.
- **Music:** habilita o deshabilita la música durante las partidas del snake, ver figura E.15.
- **Hard mode:** crea dos tuberías por las cuales la serpiente puede teletransportarse en el tablero, durante el snake, en la figura E.15.

About me

Pretende mostrar la información referente al creador de la aplicación, como muestra el diseño, ver figura E.12. Se aprecia un carrusel, con imágenes de los lenguajes de programación conocidos y debajo de dicho elementos, unos icono botones, que enlazan con las páginas de interés.

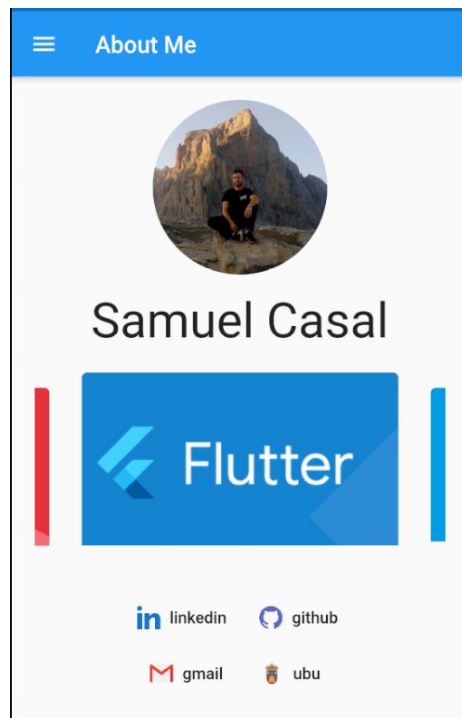


Figura E.12: About me

La lista de *iconbuttons*, con los enlaces:

- [Linkedin](#) [?].
- [Github](#) [?].
- [Gmail](#) [?].
- [Universidad de Burgos](#) [?].

Cada una de las opciones abrirá la aplicación correspondiente en el caso de que se encuentre disponible en el terminal, de no ser así se mostrará en el navegador.

En el caso de *Gmail*, nos abre la vista [E.13](#), para mandar un correo al creador de la aplicación, con algún tipo de *feedback*.



Figura E.13: Mail to

Menú de juegos

En este apartado se muestra una lista con todos los juegos de la colección que están disponibles, la finalidad es que el usuario pueda decidir de una manera sencilla a qué quiere jugar. Por lo que es tan simple como pulsar en la opción correspondiente, ver figura [E.14](#).

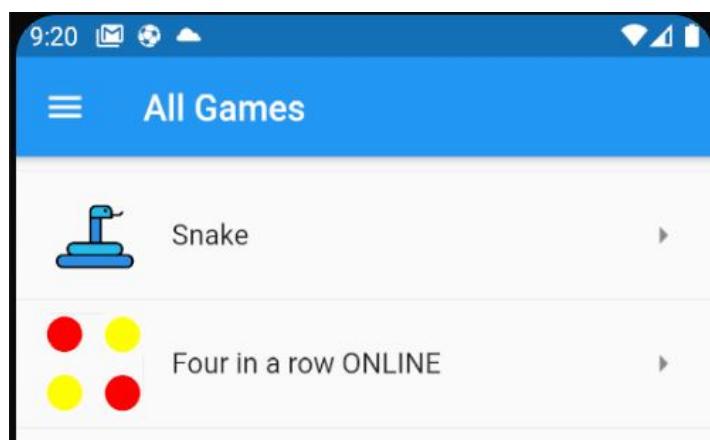


Figura E.14: Games menú

Snake

Es el juego de la serpiente, figura E.15. Cuando se entra el juego permanece a la espera hasta que se toca la pantalla (*onTab*). Una vez se produzca esto comienza la serpiente a moverse. El diseño del tablero, depende de las opciones que se tengan seleccionadas en settings, en la figura E.10.

El juego consiste en lograr la mayor puntuación posible comiendo lenguajes de programación que están repartidos de forma aleatoria por toda la superficie disponible del tablero.

El marcador de los puntos se encuentra arriba a la derecha de la pantalla.

Los controles de desplazamiento son:

- **Arriba:** deslizamiento vertical superior.
- **Abajo:** deslizamiento vertical inferior.
- **Derecha:** deslizamiento lateral derecho.
- **Izquierda:** deslizamiento lateral izquierdo.

Cabe destacar que el menú no está disponible para las ventanas de los juegos, porque puede que se interfiera con la navegabilidad del sistema.

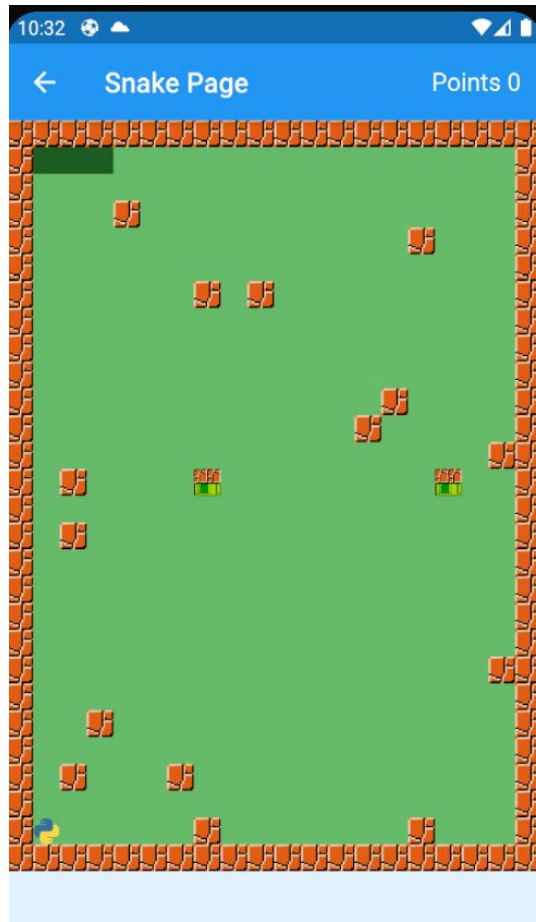


Figura E.15: Snake game

El juego finaliza desplegando la ventana, ver figura [E.16](#), cuando se da una de las siguientes situaciones:

- Impacto en la pared que rodea el tablero.
- Choque contra uno de los bloques repartidos en la superficie.
- No entrar en la tubería por la apertura.
- Cabeza de la serpiente choque contra el cuerpo de la misma.



Figura E.16: Snake game over

Cuando se produce el *game over*, ver figura E.16, tenemos 3 opciones de las cuales se debe elegir una:

1. **Play Again:** vuelves a jugar la partida con el contador de los puntos a cero.
2. **Submit score:** si se da la situación de una mejora de puntuación, con respecto a anteriores partidas, para el jugador que se encuentra logueado, se enviará al formulario, figura E.17 de ingreso de datos. En el caso de que no tengamos mejora, se puede ver el ranking, en la figura E.19.

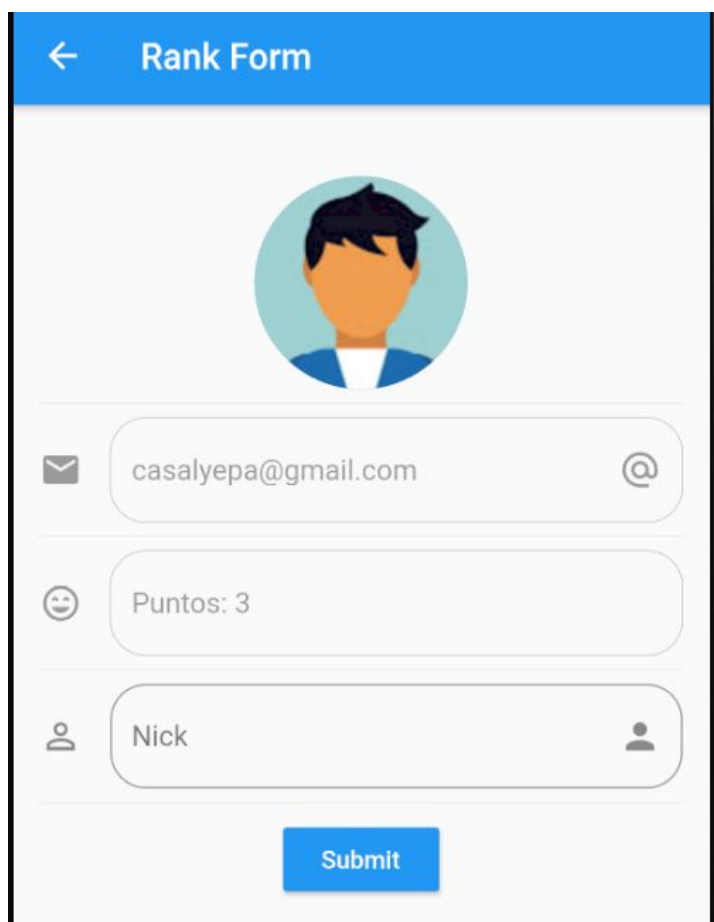
The image shows a mobile application screen titled "Rank Form". At the top, there is a blue header bar with a white back arrow icon on the left and the text "Rank Form" in white. Below the header, there is a circular profile picture placeholder showing a person with dark hair and an orange face. Underneath the profile picture, there are three input fields. The first field has an envelope icon on the left, contains the text "casalyepa@gmail.com", and has an @ symbol icon on the right. The second field has a smiley face icon on the left, contains the text "Puntos: 3", and has no icon on the right. The third field has a person icon on the left, contains the text "Nick", and has a person icon on the right. At the bottom of the form, there is a blue button with the text "Submit" in white.

Figura E.17: Formulario del ranking

En el formulario se encuentran el campo de correo y puntuación bloqueadas como medida de seguridad, por lo que se deja la opción de añadir un *nick*. Una vez pulsado el botón de *Submit*, se redirigirá a la pantalla de ranking, ver figura [E.19](#).

3. **Vídeo:** permite ver un vídeo para volver a continuar con la partida. Esta opción solo está disponible una vez por juego.

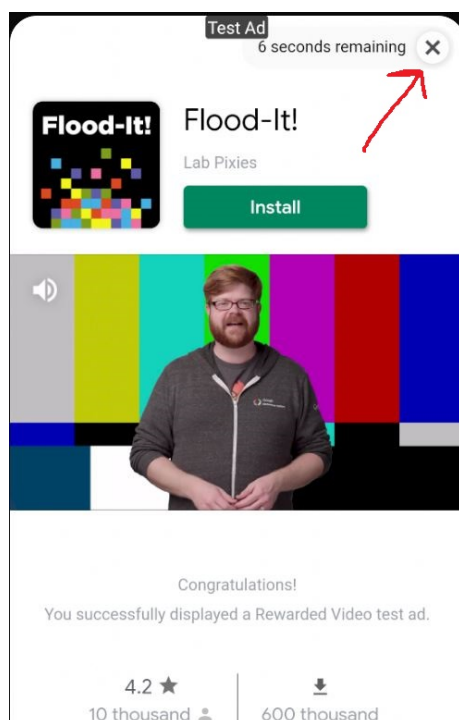


Figura E.18: Vídeo recompensa

Es necesario que se permanezcan los segundos indicados para volver a continuar con el juego.

Tras el impacto, la partida decide que dirección toma la serpiente de forma eficiente para evitar que se produzcan situaciones extrañas.

Ranking

Es una clasificación de jugadores, dependiendo de la puntuación lograda en el juego del snake, ver figura E.15. Según qué usuario entre a ver el contenido de este, ver figura E.19, se verá una línea de color dorado indicando el nivel obtenido.

En el podio de los tres primeros jugadores aparecerá una insignia en forma de medalla, destacando el logro conseguido.

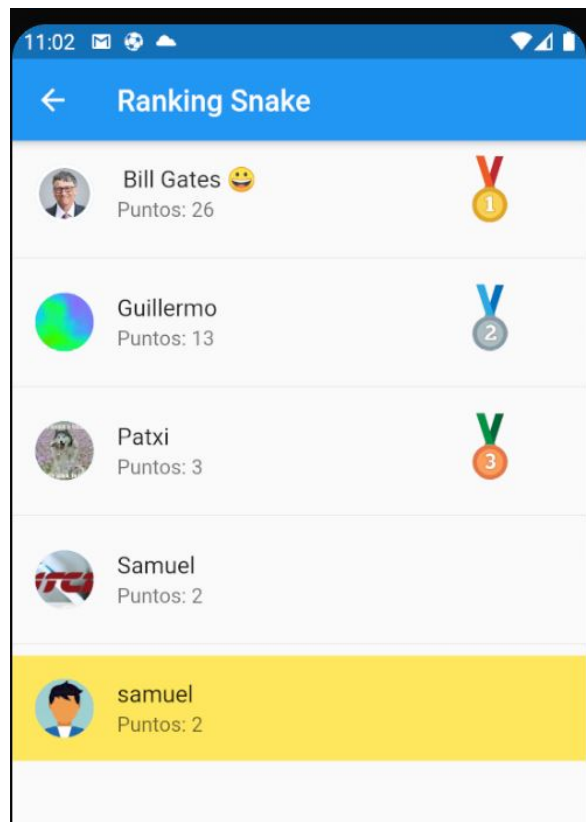


Figura E.19: Ranking

Menú juego cuatro en raya online

Es la pantalla inicial del juego, ver figura E.20, ya que siendo en modo online, uno de los jugadores debe de ser el creador de la partida (*invite friend*) y el otro el que se una a la partida creada (*join game*).

Por lo que se encuentran dos botones que redirigen a:

- Invitar a un amigo, ver figura E.21.
- Unirse a una partida creada, figura E.24.

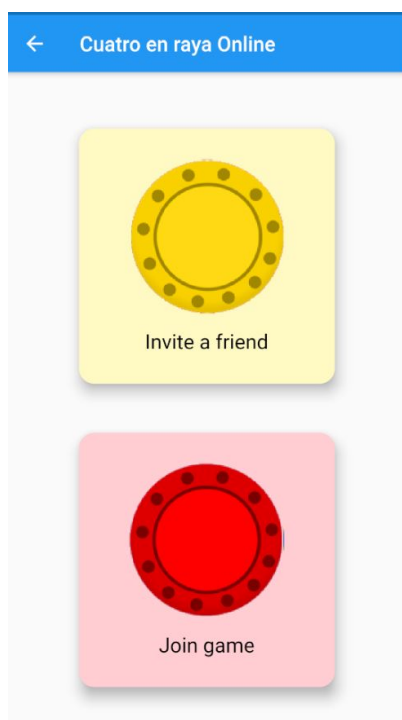


Figura E.20: Menú del cuatro en raya

Invitar

Cuando se entra en la pantalla, se genera una clave única, que es necesaria para hacer la conexión por parte del invitado. Para ello hay un botón *share* que facilita el compartir la *key*.

Hay disponible un botón de volver, que lo que hace es destruir la clave para salir a la pantalla anterior, ver figura E.20. Es el mismo efecto que si se da a la pestaña de volver atrás.

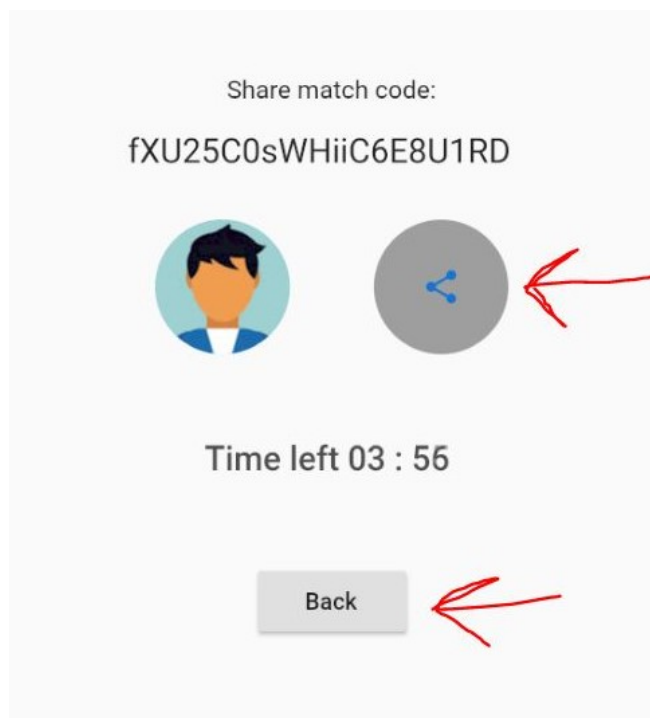


Figura E.21: Invitar cuatro en raya

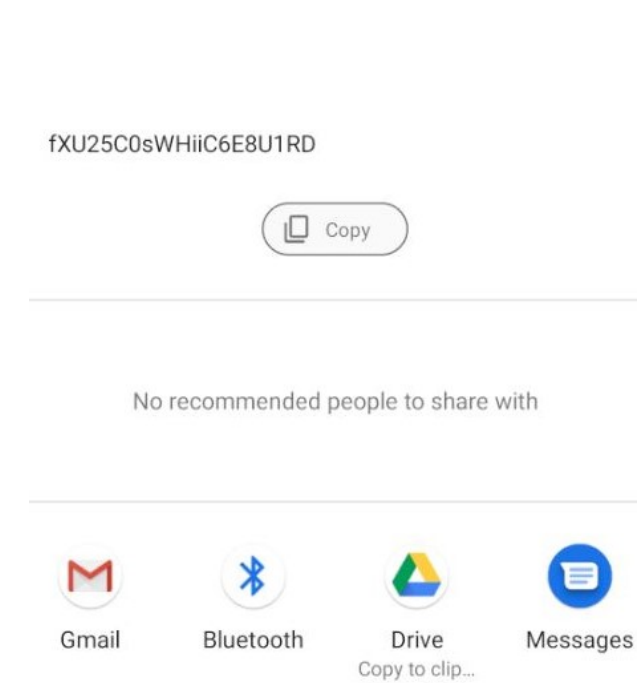


Figura E.22: Compartir clave

Cuando se pulsa en el botón de compartir aparece un cuadro de diálogo, ver figura E.22 mostrando las aplicaciones que facilitan el envío de la clave.

En el caso de que el contador del tiempo restante *time left*, llegue a cero es porque se da por hecho que no se recibe respuesta por parte del invitado, saliendo el cuadro de diálogo, ver figura E.23.

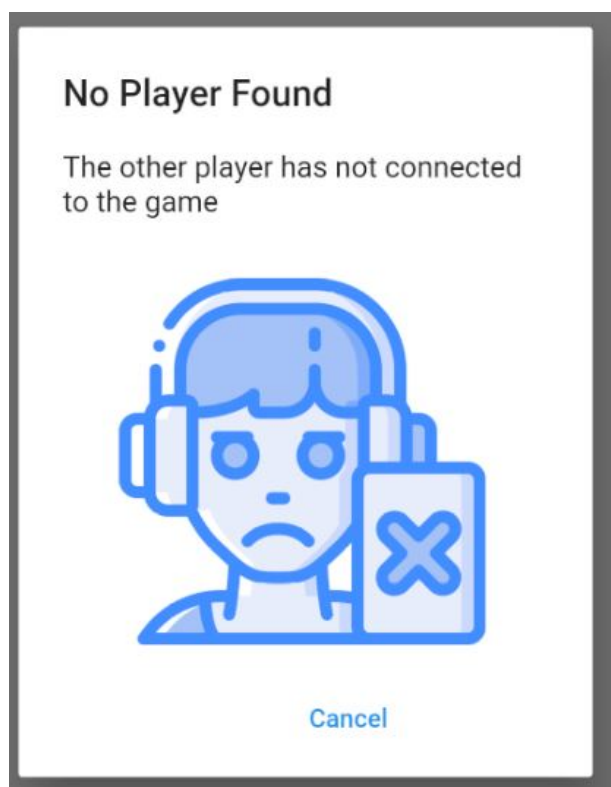


Figura E.23: Sin jugador

Unir

Pantalla con un formulario de entrada de datos, ver figura E.24, se usa para escribir la clave recibida por parte del compañero o amigo.

Si se da la situación de que no es la clave correcta se avisará por pantalla, ver figura E.25.

En el caso de que tengamos una clave válida, se redirigirá hacia la pantalla del juego, ver figura E.26.

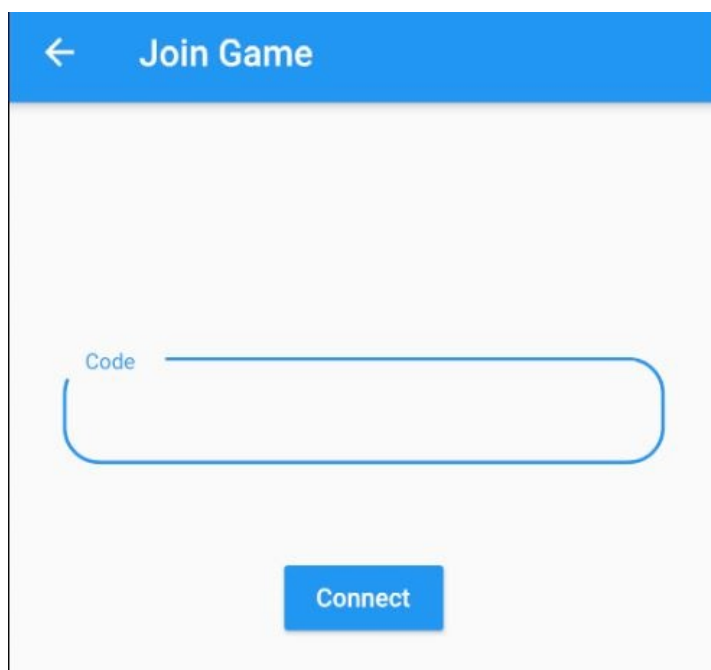


Figura E.24: Formulario para unirse a partida

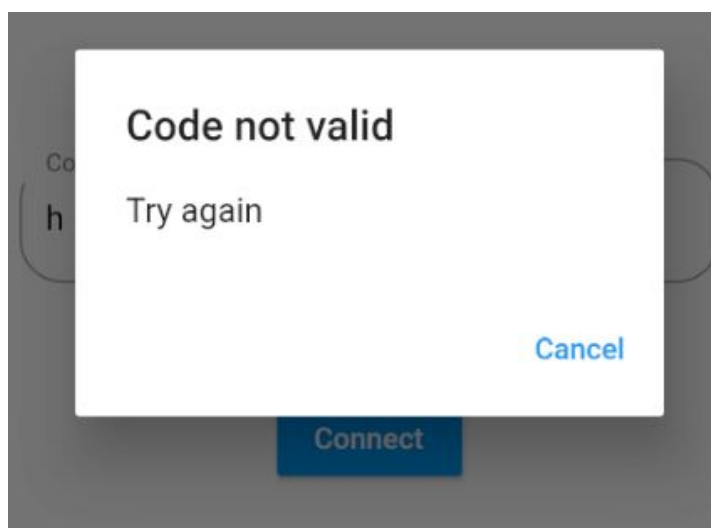


Figura E.25: Alerta de clave no válida

Juego

Página con el contenido del juego, ver figura E.26. Este consiste en formar líneas de longitud cuatro, en cualquiera de las direcciones posibles: diagonal, vertical y horizontal. Nada más entrar en esta pantalla, se produce el sorteo, para saber quién de los dos participantes comienza a jugar, ver figura E.27.

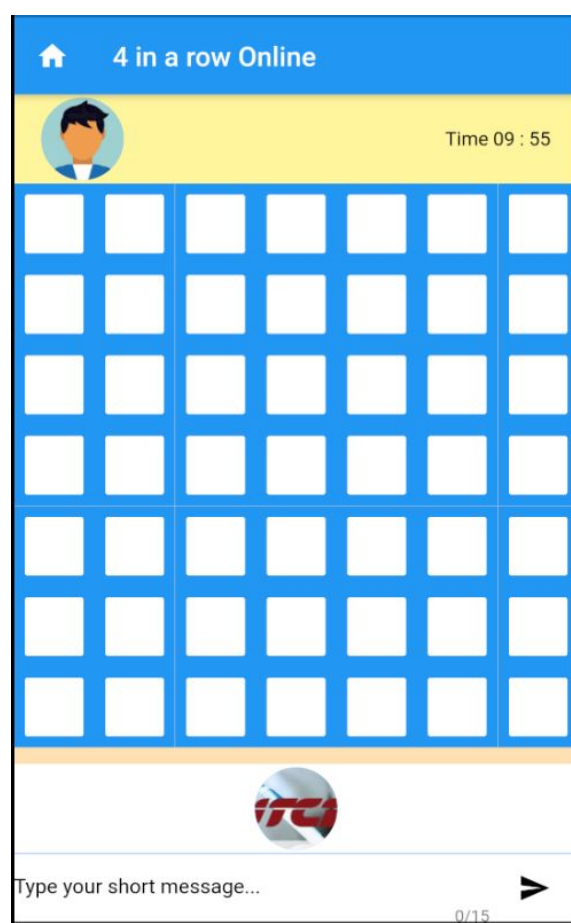


Figura E.26: Cuatro en raya

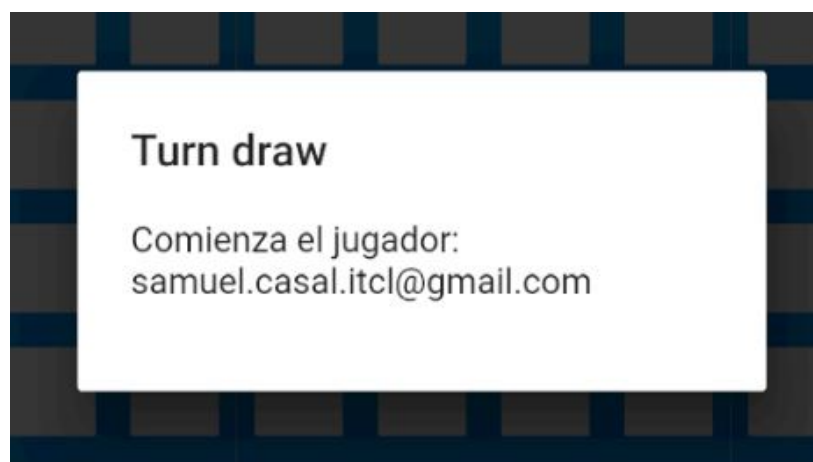


Figura E.27: Sorteo del turno

La distribución de la pantalla E.26 es la siguiente, comenzando por la parte de arriba hasta abajo:

- Cabecera superior, ver figura E.28, se encuentra el rival a batirse, representado mediante una imagen, siendo esta la de la cuenta de *Google*. Dependiendo del color de fondo, se indica si está en su turno o no.

En la parte central hay un espacio para mostrar los mensajes recibidos por parte del contrincante.

En la parte lateral derecha se encuentra el contador del tiempo transcurrido.

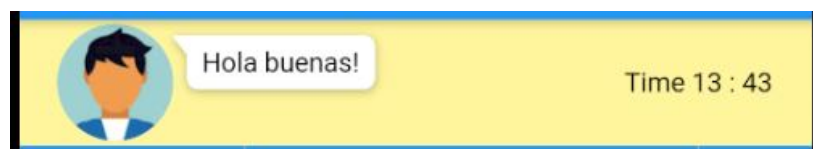


Figura E.28: Zona superior

- En la parte central, ver figura E.29, se dispone el tablero de siete filas por siete columnas. En el caso de que sea nuestro turno, se debe pinchar en la pantalla, donde sea posible colocar una ficha, es decir, que si debajo de la celda se encuentra vacío no será posible colocar la ficha y se deberá proceder a buscar otro sitio en el que se pueda.

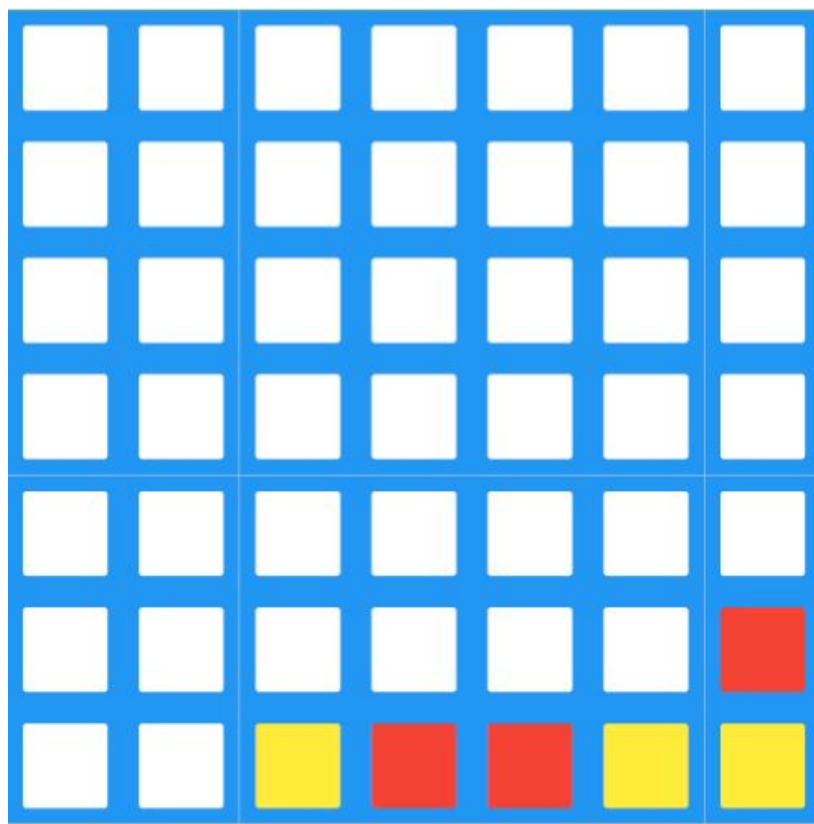


Figura E.29: Tablero

- En la parte inferior, ver figura E.30, se encuentra la imagen del aliado, es decir, puede ser el jugador que se unió o que creó la partida, dependiendo de quién sea el que haya tomado ese rol.

Si esta zona se encuentra con el color de fondo distinto de blanco indica que es el turno de este jugador.



Figura E.30: Zona personal

- Campo para introducir mensajes, ver figura E.31, está limitado a 15 caracteres, ya que está destinado para mensajes breves o emoticonos.

A la derecha de este campo, se encuentra el botón de enviar. Una vez se envía el mensaje, este se deshabilita, para no saturar la línea. Pasados 10 segundos volverá a estar operativo.

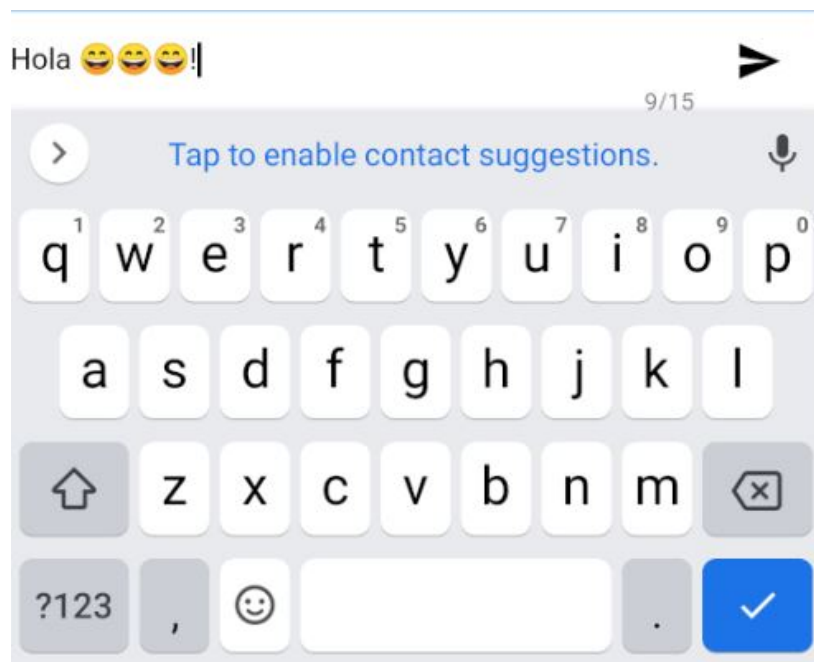


Figura E.31: Envío de mensaje

En el momento en que se da la situación de fin de juego, salta la alerta para mostrar quién de los dos jugadores es el ganador de la partida, como se puede ver en la imagen siguiente, ver figura [E.32](#)

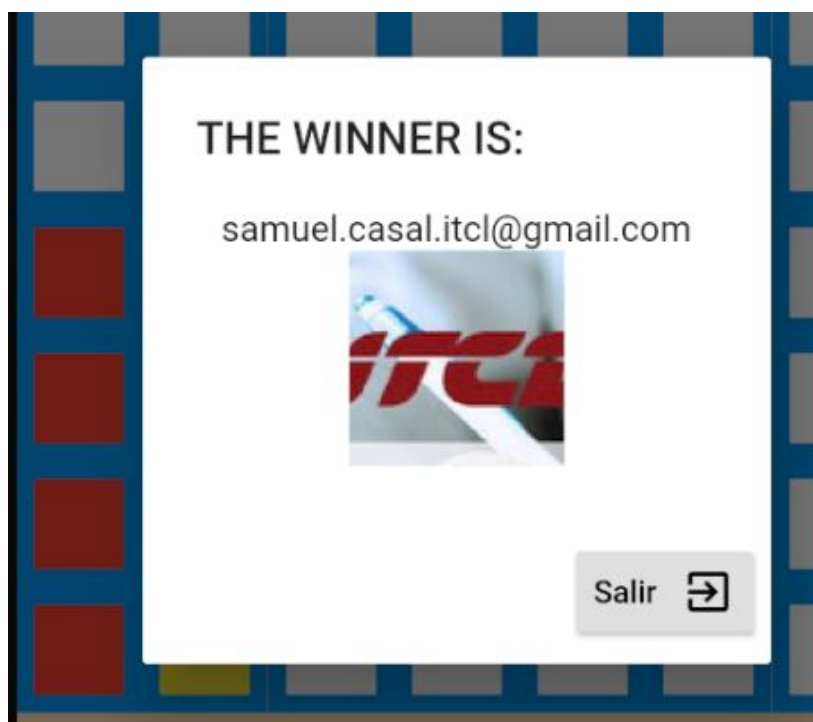


Figura E.32: Final del cuatro en raya

Bibliografía
