



UNIVERSIDAD DE BURGOS  
ESCUELA POLITÉCNICA SUPERIOR  
Grado en Ingeniería Informática



**TFG del Grado en Ingeniería  
Informática**

**Flutter  
Documentación Técnica**



Presentado por Samuel Casal Cantero  
en Universidad de Burgos — 14 de julio  
de 2020

Tutor: José Francisco Díez Pastor y César  
García Osorio



---

# Índice general

---

<b>Índice general</b>	<b>I</b>
<b>Índice de figuras</b>	<b>III</b>
<b>Índice de tablas</b>	<b>v</b>
<b>Apéndice A Plan de Proyecto Software</b>	<b>1</b>
A.1. Introducción . . . . .	1
A.2. Planificación temporal . . . . .	1
A.3. Estudio de viabilidad . . . . .	7
<b>Apéndice B Especificación de Requisitos</b>	<b>11</b>
B.1. Introducción . . . . .	11
B.2. Objetivos generales . . . . .	11
B.3. Catalogo de requisitos . . . . .	12
B.4. Especificación de requisitos . . . . .	15
<b>Apéndice C Especificación de diseño</b>	<b>23</b>
C.1. Introducción . . . . .	23
C.2. Diseño de datos . . . . .	23
C.3. Diseño procedimental . . . . .	29
C.4. Diseño arquitectónico . . . . .	30
C.5. Diseño de interfaces . . . . .	33
<b>Apéndice D Documentación técnica de programación</b>	<b>35</b>
D.1. Introducción . . . . .	35
D.2. Estructura de directorios . . . . .	35

D.3. Manual del programador . . . . .	38
D.4. Compilación, instalación y ejecución del proyecto . . . . .	42
D.5. Pruebas del sistema . . . . .	46
<b>Apéndice E Documentación de usuario</b>	<b>51</b>
E.1. Introducción . . . . .	51
E.2. Requisitos de usuarios . . . . .	51
E.3. Instalación . . . . .	52
E.4. Manual del usuario . . . . .	53
<b>Bibliografía</b>	<b>79</b>

---

# Índice de figuras

---

A.1. Sprint 1. . . . .	3
A.2. Sprint 2. . . . .	4
A.3. Sprint 3. . . . .	5
A.4. Sprint 4. . . . .	7
B.1. Boceto sing in . . . . .	12
B.2. Diagrama con los casos de uso . . . . .	16
C.1. Base de datos en firestore . . . . .	24
C.2. Diagrama BD firestore . . . . .	25
C.3. Contenido json de rankinig . . . . .	26
C.4. Contenido json de cuatorows . . . . .	27
C.5. Método para crear la tabla en la base de datos local . . . . .	28
C.6. Diagrama de flujo juego snake . . . . .	29
C.7. Estado Flutter . . . . .	31
C.8. Diseño bloc . . . . .	32
C.9. Estructura de directorios aplicación . . . . .	32
D.1. Google services . . . . .	36
D.2. app/build.gradle . . . . .	37
D.3. Comando generar clave keyStore . . . . .	37
D.4. Variables entorno . . . . .	39
D.5. Comando version Flutter . . . . .	39
D.6. Máquinas virtuales . . . . .	40
D.7. Firebase console . . . . .	41
D.8. Play Store console . . . . .	42
D.9. Abrir git bash . . . . .	43
D.10.Comando para clonar el repositorio. . . . .	43

D.11.Actualización paquetes . . . . .	44
D.12.Máquina virtual . . . . .	45
D.13.Máquina virtual . . . . .	47
D.14.Código para las pruebas de caja negra . . . . .	48
E.1. Tienda con el proyecto . . . . .	53
E.2. Log in page . . . . .	54
E.3. Formulario sign in . . . . .	55
E.4. Política de privacidad . . . . .	55
E.5. Home . . . . .	56
E.6. Banner . . . . .	56
E.7. Menú lateral . . . . .	57
E.8. Mostrar sign out . . . . .	58
E.9. Sign Out . . . . .	58
E.10.Settings . . . . .	59
E.11.Dark Mode . . . . .	60
E.12.About me . . . . .	61
E.13.Mail to . . . . .	62
E.14.Games menú . . . . .	63
E.15.Snake game . . . . .	64
E.16.Snake game over . . . . .	65
E.17.Formulario del ranking . . . . .	66
E.18.Vídeo recompensa . . . . .	67
E.19.Ranking . . . . .	68
E.20.Menú del cuatro en raya . . . . .	69
E.21.Invitar cuatro en raya . . . . .	70
E.22.Compartir clave . . . . .	70
E.23.Sin jugador . . . . .	71
E.24.Formulario para unirse a partida . . . . .	72
E.25.Alerta de clave no válida . . . . .	72
E.26.Cuatro en raya . . . . .	73
E.27.Sorteo del turno . . . . .	74
E.28.Zona superior . . . . .	74
E.29.Tablero . . . . .	75
E.30.Zona personal . . . . .	75
E.31.Envío de mensaje . . . . .	76
E.32.Final del cuatro en raya . . . . .	77

---

# Índice de tablas

---

A.1. Equivalencias <i>Story Points</i> y tiempo estimado . . . . .	2
A.2. Coste hardware . . . . .	8
A.3. Coste personal . . . . .	8
A.4. Coste cuota de la seguridad social . . . . .	9
A.5. Coste vario . . . . .	9
A.6. Licencias de bibliotecas y herramientas utilizadas . . . . .	10
A.7. Fuente del contenido audiovisual . . . . .	10
B.1. Caso de uso 1: Sign in . . . . .	17
B.2. Caso de uso 2: Mostrar el menú . . . . .	18
B.3. Caso de uso 3: Configuración de la aplicación . . . . .	19
B.4. Caso de uso 4: Mostrar el contenido de About . . . . .	19
B.5. Caso de uso 5: Contenido publicitario . . . . .	20
B.6. Caso de uso 6: Juego del snake . . . . .	21
B.7. Caso de uso 7: Juego cuatro en raya online . . . . .	22
D.1. Clases de equivalencia . . . . .	48
D.2. Derivación en casos de prueba de caja negra . . . . .	49





## Apéndice A

---

# Plan de Proyecto Software

---

### A.1. Introducción

En este apartado trabajaremos 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 importante que para cada una de las tareas, sepamos el tiempo que durará aproximadamente, quien es el encargado de hacer la tarea y el dinero que supone hacerla. Por lo que invertir tiempo en la estimación de las horas cada una de las tareas, ayuda a identificar las irregularidades 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. Dependiendo de los requisitos del proyecto, número de integrantes y

del tiempo disponible, se maneja esta horquilla temporal, en mi caso por la falta de tiempo, decidí hacerlos de 5 días.

Estos *sprints* se basan en una reunión, donde se planifican todas las tareas que se tiene como objetivo realizar, en mi caso eran las reuniones eran 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.

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, de que lo que estaba haciendo no me funcionaba, que estaba verde y que si cabía la posibilidad de hacer otro trabajo. Me dieron el visto bueno, pero que tenía que hacer algo diferenciador, ya que no vale cualquier cosa. Por lo que en la reunión de cierre de *sprint* se comentaría como centrar la aplicación.

Entonces debido a la escasez temporal, me decido a hacer una aplicación en *Flutter*, ya que se puede hacer algo de calidad de manera ágil.

Los objetivos en este *sprint* inicial fueron:

- Documentar como hacer aplicaciones en *Flutter*.
- Cursar un curso en [Udemy](#)
- 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](#)

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 el curso me duró más de lo estimado.



Figura A.1: Sprint 1.

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

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

Los objetivos planteados fueron:

- Persistencia de datos, con base de datos local.
- 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**



Figura A.2: Sprint 2.

En la reunión del cierre del seguimiento del sprint se muestra la **v.2.0.0**, viendo que los objetivos propuesto 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 dieron para completar 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. Se me propuso una idea de juego, que 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, por lo que el juego, al final, 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 memoria y anexos.

Las diferentes *issues* se encuentran en [Sprint 3](#)



Figura A.3: Sprint 3.

Al final en la reunión del cierre de este *Sprint*, con los tutores, quedaba 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 que para el siguiente, debería de quedar terminado.

En cuanto a las horas que planifique 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 más horas a otras tareas o que soy demasiado optimista a la hora de hacer la planificación.

Se estimaron unas 58 horas, pero al final 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 que sobreestime para la memoria.

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

La duración de este *Sprint* es de un día menor, porque coincidía el cierre de este 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 amigo para el poster.
- Resolución de algunos problemas.

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



Figura A.4: Sprint 4.

Para el cierre el total de las tareas fueron completadas, se planificaron 41 horas, destinando 33 horas para el desarrollo esperado.

## A.3. Estudio de viabilidad

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

### 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, incluyendo 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 que nos ofrece *Google Ad*, a través de diferentes tipos de *banners* y videos, con los que tener un retorno de dinero. Ya que la idea es que la aplicación este disponible de manera gratuita, en la *play store* para todos aquellos que se la quieran descargar. Aunque esto este disponible, para contabilizar los costes es algo despreciable, 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 los 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, que se encargaba del desarrollo de software y la planificación. El número de horas inicialmente pensado fue de 250 horas, siendo este 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 €.

### Costes varios

Otros costes que también se deben de tener en cuenta en el proyecto

Concepto	Coste (€)
Internet	50
Cuenta Google Play	25
Coste total	75

Tabla A.5: Coste vario

### Coste total proyecto

El coste total del proyecto es la suma de los costes anteriores, que nos da un importe de 5399,9 €, a esto le tenemos que aplicar el correspondiente incremento del impuesto de valor añadido, que es [?] del 21 %, por lo que el coste total del proyecto es de 6533,9 €.

### Beneficios

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

Por lo que la vía de obtener ingresos como freelance, puede ser incrementar el coste del proyecto por un 15 %, de tal manera que nos podamos asegurar

ese beneficio, además de garantizarnos ante un incremento de las horas planificadas, seguir teniendo ese margen de beneficio.

El nuevo coste total del proyecto sería entonces de 6209,89 €, que añadiendo el impuesto de valor añadido, se queda en 7513,96 €.

## Viabilidad legal

Para el completar el proyecto han sido necesarias gran multitud de herramientas, a continuación, en la tabla A.6, se expondrán las principales que fueron utilizadas.

Librería	Versión	Descripción	Licencia
VsCode	1.46.1	Editor de código.	MIT
Android Studio	4.0	Aplicación para virtualización del sistema operativo Android.	Apache 2.0
Android R	10.0+ Api 30	Versión del S.O virtualizado.	Apache 2.0
Flutter	1.17	Framework con el que se ha desarrollado la app	BSD 3-Clause
Dart	2.2.0	Lenguaje de programación desarrollado por Google	BSD
Node.js	12.18	Uso de npm	MIT
Firebase console	5.5	Herramienta de Google gestión sercios.	Apache 2.0

Tabla A.6: Licencias de bibliotecas y herramientas utilizadas

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

## Imágenes y material gráfico

Fuente	Descripción	Licencia
Giphy	Repositorio de contenido visual	Open Source

Tabla A.7: Fuente del contenido audiovisual

## Apéndice *B*

---

# Especificación de Requisitos

---

### B.1. Introducción

En este apéndice se detallan los requisitos del proyecto, como los funcionales y no funcionales. La finalidad es hacer de intermediario entre el cliente y los programadores, con el objetivo de ayudar a entender, comprender y analizar la aplicación.

### B.2. Objetivos generales

Este trabajo final de grado focaliza la aplicación mediante dos puntos de vista totalmente diferentes, dependiendo de las situaciones futuras a las que se someta la aplicación, siempre englobado en el marco de que es un producto que nos 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, ante los equipos de recursos humanos en las empresas, llegado el momento de tener que buscar trabajo.

### B.3. Catalogo de requisitos

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

#### Requisitos funcionles

Por cada uno de los requisitos funcionales, va a tener 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.

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 23.

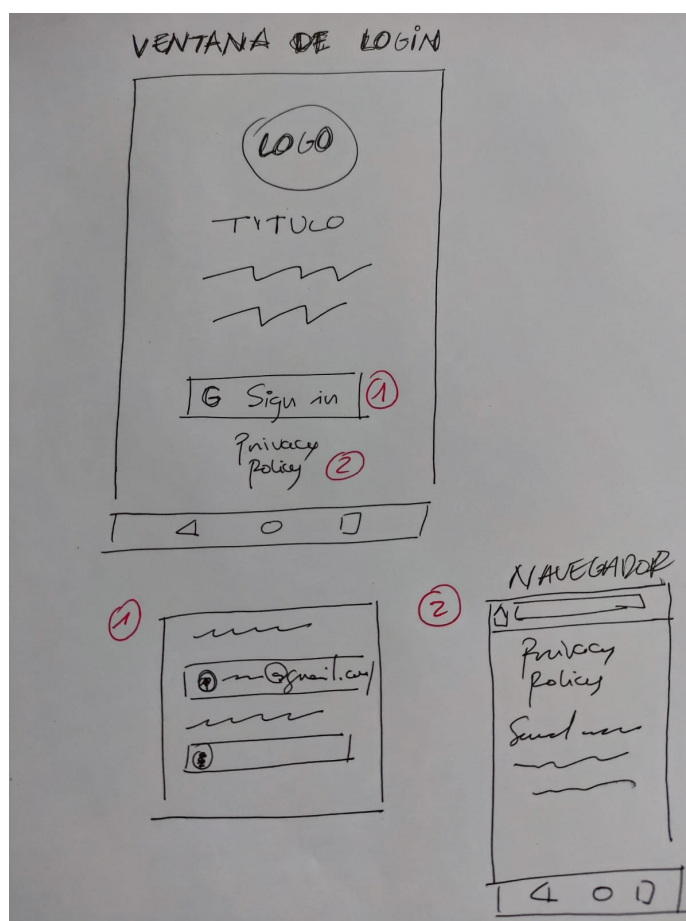


Figura B.1: Boceto sing in

- **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 ser capaz de gestionar algunas de las diferentes capacidades del producto, que el usuario desee.
  - **RF-3.1 Opción *dark mode*:** la aplicación tiene que ser capaz 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 carousel imagenes:** 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*:** mostrar en el *footer* un contenedor de anuncios, proporcionados estos por Google.
- **RF-5.2 Continuar Snake:** el usuario tiene que ser capaz de ver un video, como recompensa, continuar en la partida.
- **RF-6 Juego del snake E.15:** el usuario tiene que ser capaz de jugar a este juego.
  - **RF-6.1 Jugar snake:** la aplicación tiene que ser capaz de trabajar con los choques que se producen, con la comida, pared, bloques, o tuberías.
  - **RF-6.2 Mostrar la puntuación:** la app tiene que ser capaz de mostrar la puntuación que lleva en 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 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 ser capaz de enseñar un menú intermedio con el fin de poder elegir entre invitar o unirse.
  - **RF-7.2 Invitar:** el usuario tiene que ser capaz de 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 debe de ser capaz de ingresar la clave de acceso de la partida.
  - **RF-7.4 Jugar cuatro en raya:** la aplicación tiene que ser capaz de controlar los diferentes eventos se producen durante la partida, como 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 se debe ver 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 una garantía de confianza. Ninguno de los datos deben de ser ofrecidos a terceros, exceptuando a Google.  
Mediante una cifrado de extremo a extremo (SHA-1 [?]) se realizaran las comunicaciones con *Firebase cloud*.
- **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 periódica.
- **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. Este será capaz de ver todo el contenido que tiene asociado a su cuenta.

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.

Diagrama de casos de uso

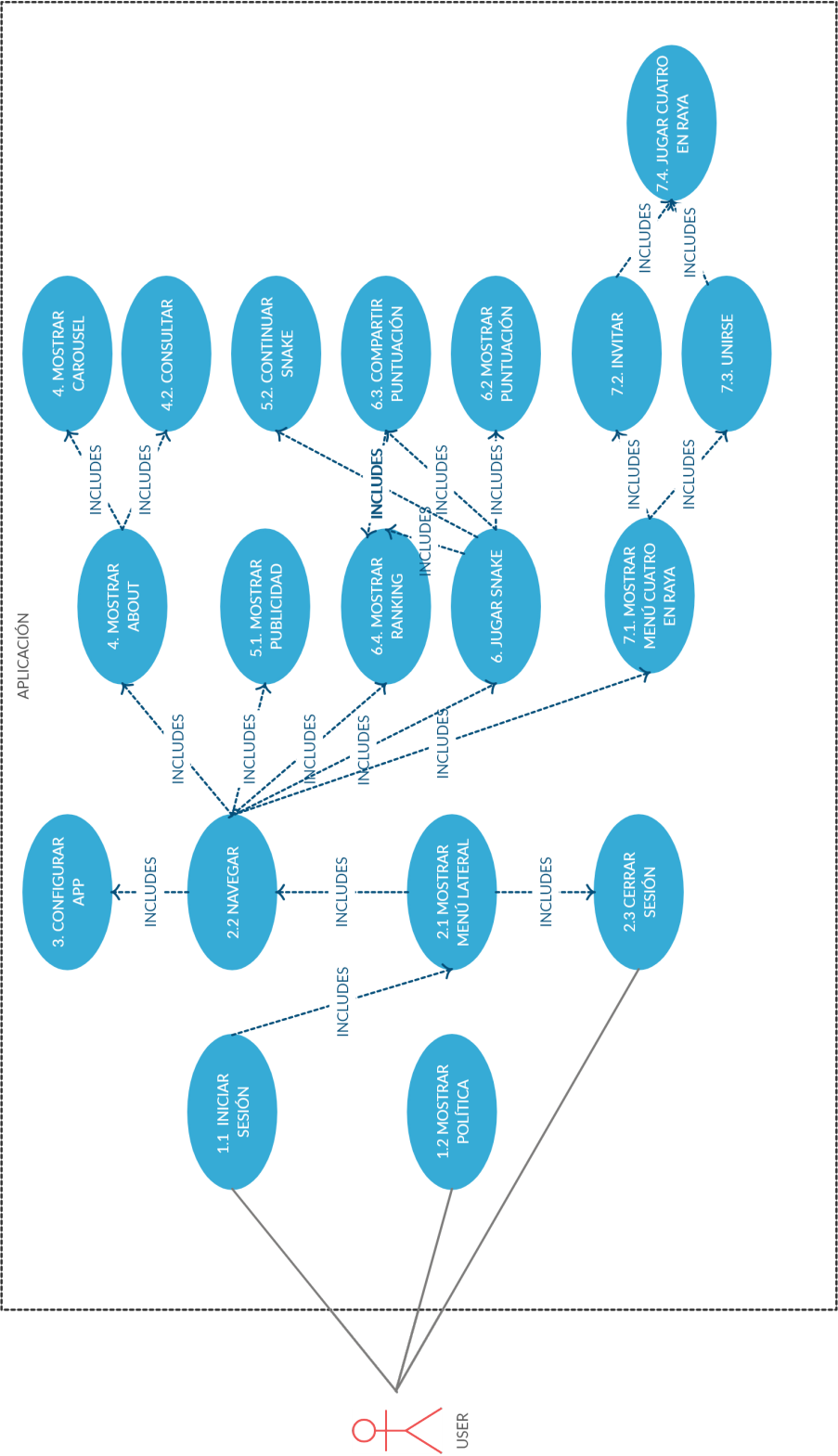


Figura B.2: Diagrama con los casos de uso



## 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 clicka en 'Privacy policy'.
	2        Se muestra el navegador con lo 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 la 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 quien 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. En el caso el botón sea de correo, nos 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 video 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. Si son pulsados, nos deben de abrir el navegador o la 'play store'.
	2	En el juego del snake, ver el video 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 tenemos que elegir el juego del snake.
	2	Intentar conseguir la mayor puntuación posible.
	3	Si morimos, ver video para continuar con la partida.
	4	En el caso de tener mejora de puntuación, mostrar 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	Dar la capacidad a los jugadores de crear o unirse a partidas, con el fin de poder batirse en batallas del cuatro en raya. Durante estas 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 batirnos.
Secuencia normal	Paso    Acción
	1        Abrir el menú lateral para navegar a la pestaña de juegos, una vez dentro tenemos que 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 con 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, como se resolvieron los requisitos funcionales anteriormente expuestos [11](#), 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, tenemos 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 un modelos no relacionales. Esta fue usada para la persistencia externa de datos. Como se puede ver en la siguiente imagen [C.1](#) de la consola de Firebase:

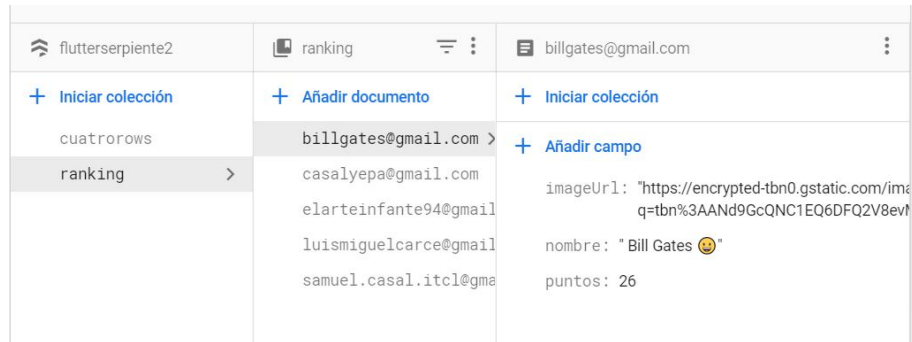


Figura C.1: Base de datos en firestore

- **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 problema, ya que solo queremos 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 (colecciones en este caso) en la base de datos y que no tiene relación entre sí.





Figura C.2: Diagrama BD firestore

Una vez sabemos 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

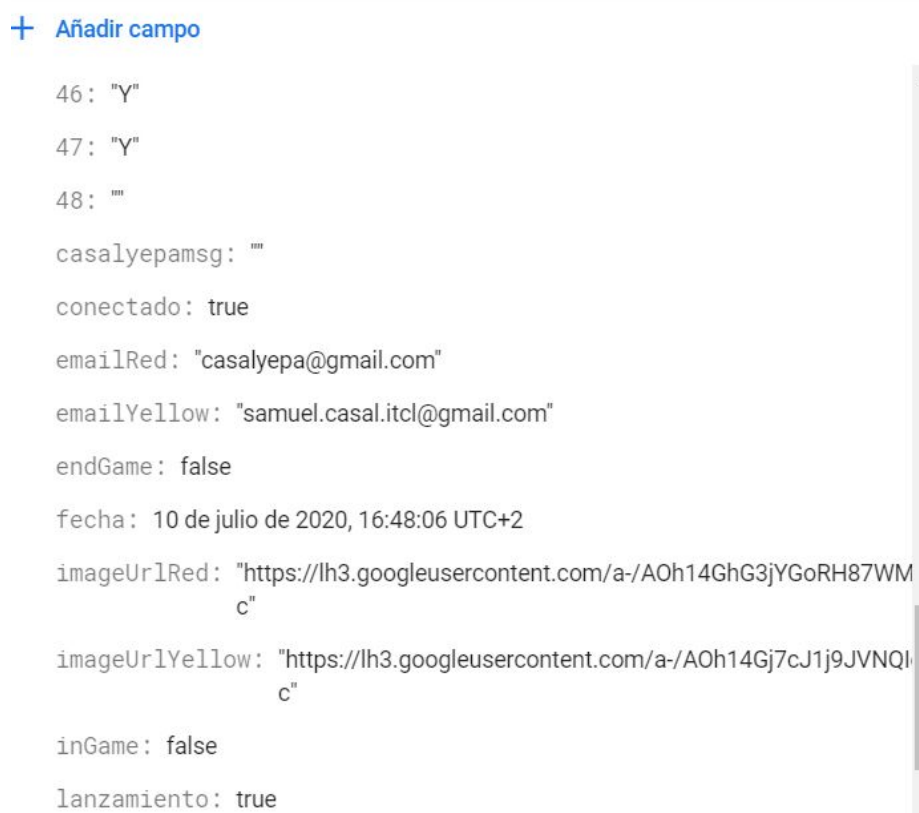
```
imageUrl: "https://encrypted-tbn0.gstatic.com/images?q=tbn%3AAND9GcQNC1EQ6DFQ2V8evM0mhwQkjDrF3NbxeBCK9A&usc"
nombre: " Bill Gates 😊"
puntos: 26
```

Figura C.3: Contenido json de ranking

- **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.

Por lo que la *key* compartida durante el juego es 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.



The image shows a screenshot of a JSON editor interface. At the top left, there is a blue plus icon followed by the text 'Añadir campo'. Below this, a JSON object is displayed with the following fields and values:

```
{
  "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-/AOh14GhG3jYGoRH87WMc",
  "imageUrlYellow": "https://lh3.googleusercontent.com/a-/AOh14Gj7cJ1j9JVNQIc",
  "inGame": false,
  "lanzamiento": true
}
```

Figura C.4: Contenido json de cuatorows

- **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. Se usa internamente nada más, 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 [24](#).

### C.3. Diseño procedimental

Para 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:

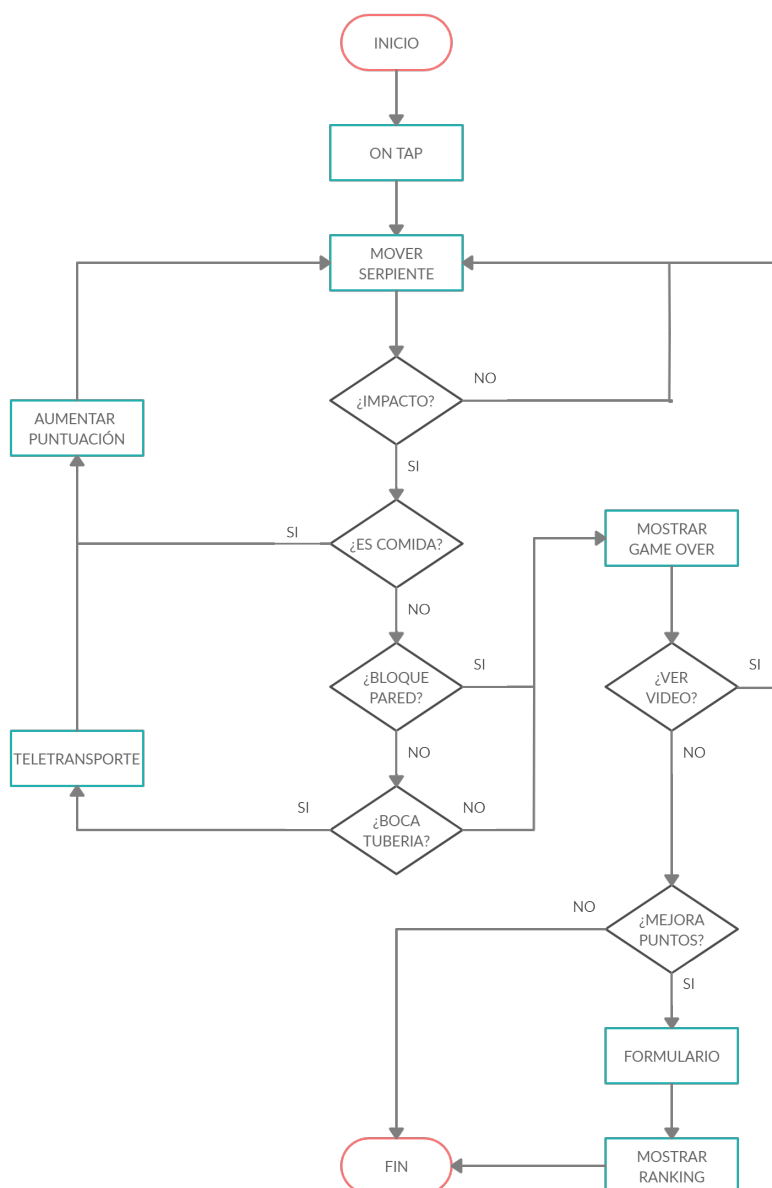


Figura C.6: Diagrama de flujo juego snake

## C.4. Diseño arquitectónico

La arquitectura de la aplicación ha seguido el patrón de diseño BLoC [?]. Significa *Business Logic Component* 31. Fue creado por Paolo Soare y Cong Hu, los dos de Google y presentado en la conferencia de Dart en 2018. Por lo que es algo bastante nuevo.

Para comprender el porque de esta arquitectura es necesario saber como funcionan los estados compartidos entre los componentes en los framework declarativos 30.

### 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, donde se incluye también las llamadas a un 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.

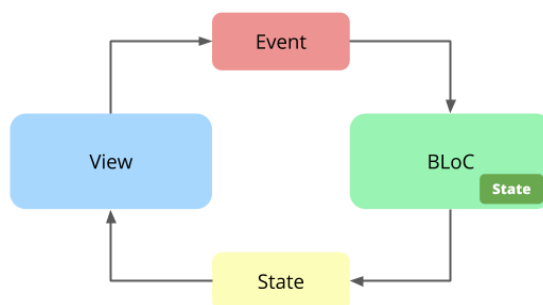


Figura C.8: Diseño bloc

Por lo que una vez sabemos que la arquitectura es BLoC, he pretendido 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](#)

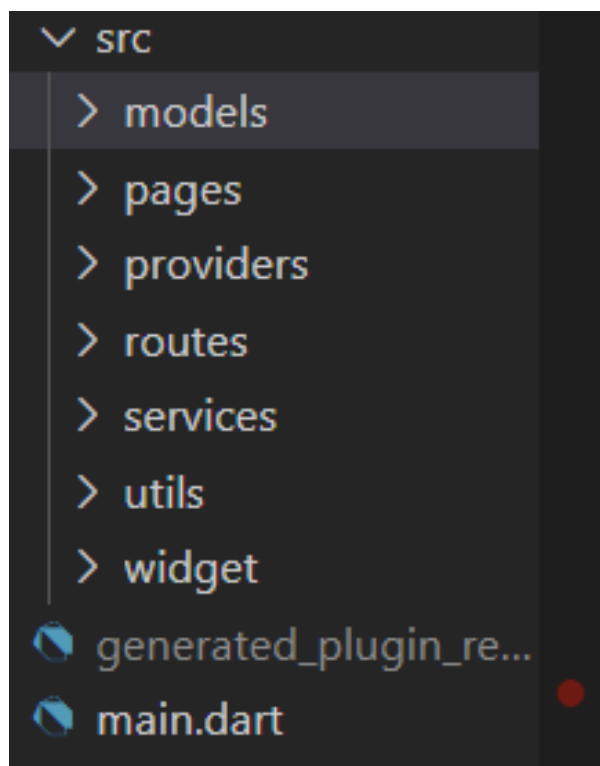


Figura C.9: Estructura de directorios aplicación



- **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 tiene 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 de especificación de requisitos 11, 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 ??.



## 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](#). La estructura de ficheros que sigue es la siguiente, destacando algunos de los más importantes:

- `./` 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 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, ya que cuando lo subamos a la *Play Store*, es algo que debemos de revisar, 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"
}

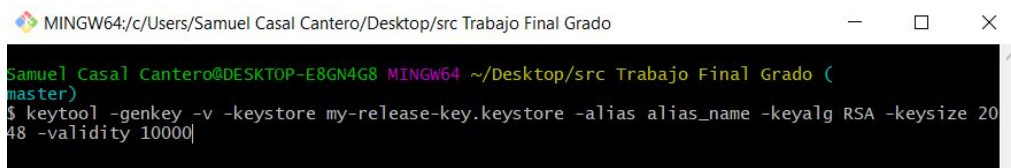
//NUEVO PLAY STORE
signingConfigs {
    release {
        keyAlias keystoreProperties['keyAlias']
        keyPassword keystoreProperties['keyPassword']
        storeFile file(keystoreProperties['storeFile'])
        storePassword keystoreProperties['storePassword']
    }
}

//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.



The screenshot shows a terminal window with the title 'MINGW64/c/Users/Samuel Casal Cantero/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`. The prompt shows the user is 'Samuel Casal Cantero' at 'DESKTOP-E8GN4G8' in a 'MINGW64' environment, with the current directory being '~/Desktop/src Trabajo Final Grado'.

Figura D.3: Comando generar clave keyStore

- **assets**: directorio que contiene los activos 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 las personas futuras que 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](#) 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 ejecutables), por lo que debemos de dejar este en un lugar en

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

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

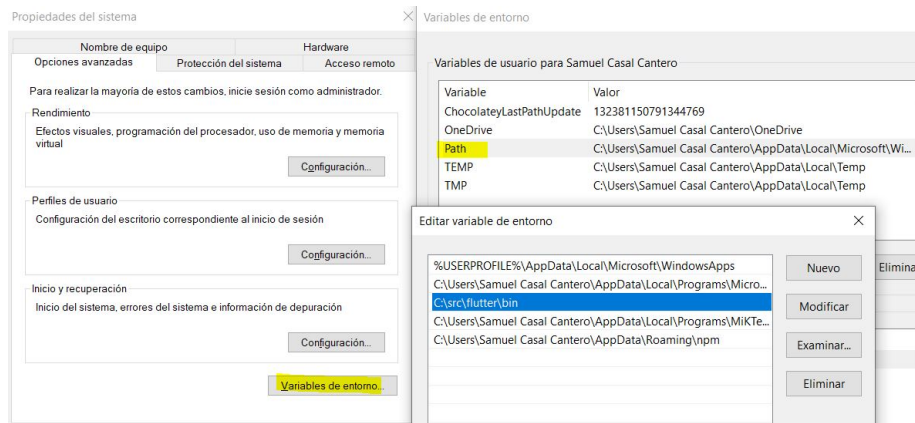


Figura D.4: Variables entorno

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](#) e instalamos, en mi caso uso la versión 3.6.

Una vez lo tengamos instalado creamos las máquinas virtuales, para ello nos vamos a la barra de herramientas Tools>AVD manager y creamos las que sean necesarias, en mi caso tengo dos, con Android R, que es la versión 10 de sistema operativo, ya que es la última estable que nos ofrece el IDE.

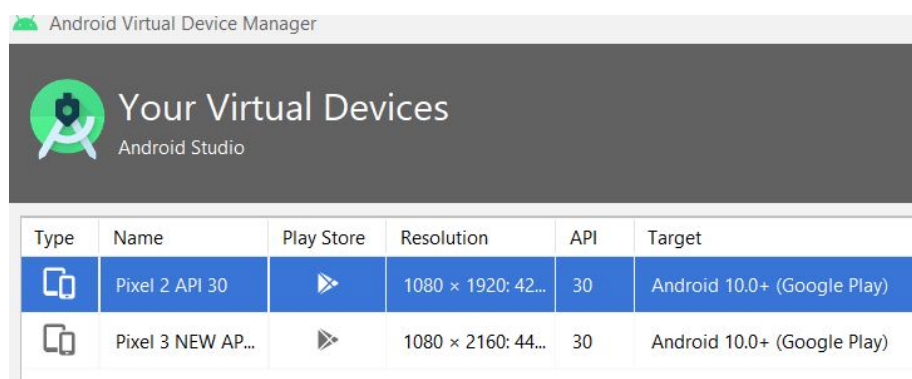


Figura D.6: Máquinas virtuales

## 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](#), en mi caso uso la versión 1.47.

Una vez lo tengamos instalado, procedemos a instalar los siguiente *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 que es lo que hace cada una de estas herramientas:

- [Awesome Flutter Snippets](#). Apache 2.0.
- [Bracket Pair Colorizer 2](#). MIT License.
- [Dart](#). MIT License.
- [Flutter](#). 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](#).

## Firebase Console:

La consola de [Firebase](#) 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, os tengo que dar permisos de administrador en el caso de que fuera necesario, para ello no dudéis en mandarme un correo [casalyepa@gmail.com](mailto:casalyepa@gmail.com)

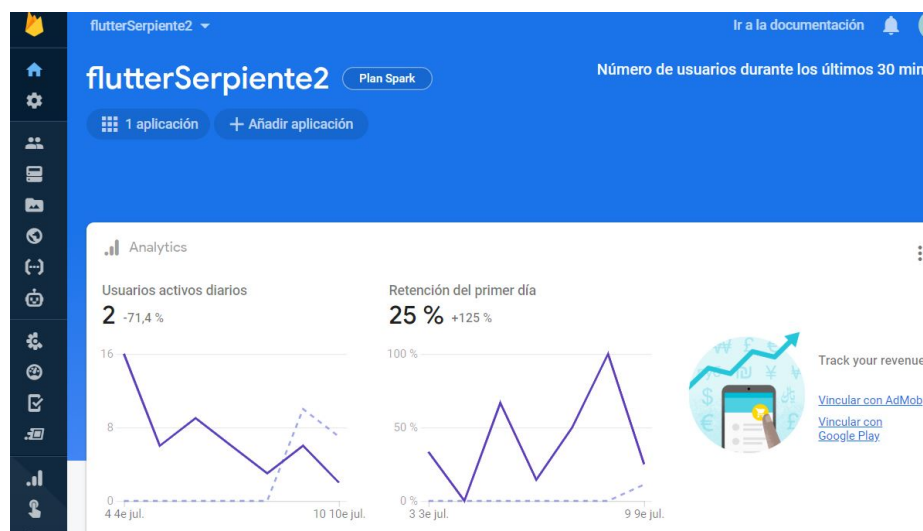


Figura D.7: Firebase console

## Play Store Console:

La consola de [Play Store](#) es la herramienta web que nos permite hacer el despliegue de la aplicación, tanto como para hacer las pruebas 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 tenemos que tener paciencia.

Al igual que con Firebase, os tengo que dar permisos en el caso de que sea necesario hacer nuevos despliegues de la app [casalyepa@gmail.com](mailto:casalyepa@gmail.com)

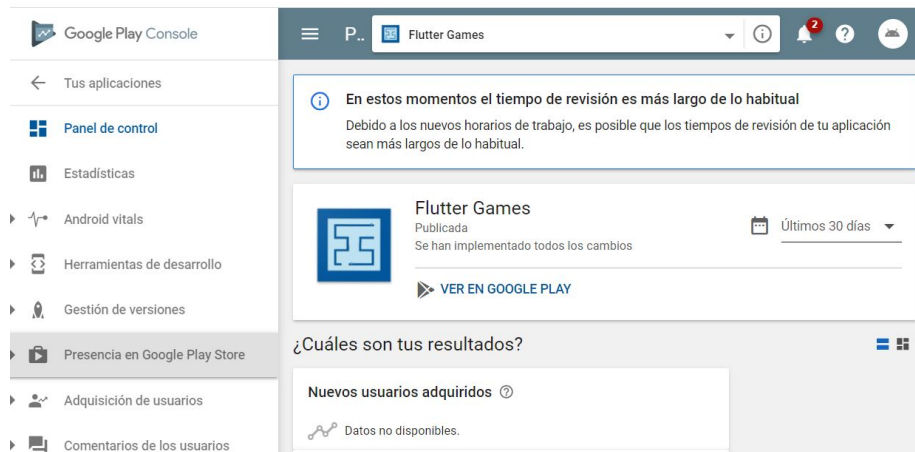


Figura D.8: Play Store console

## 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:

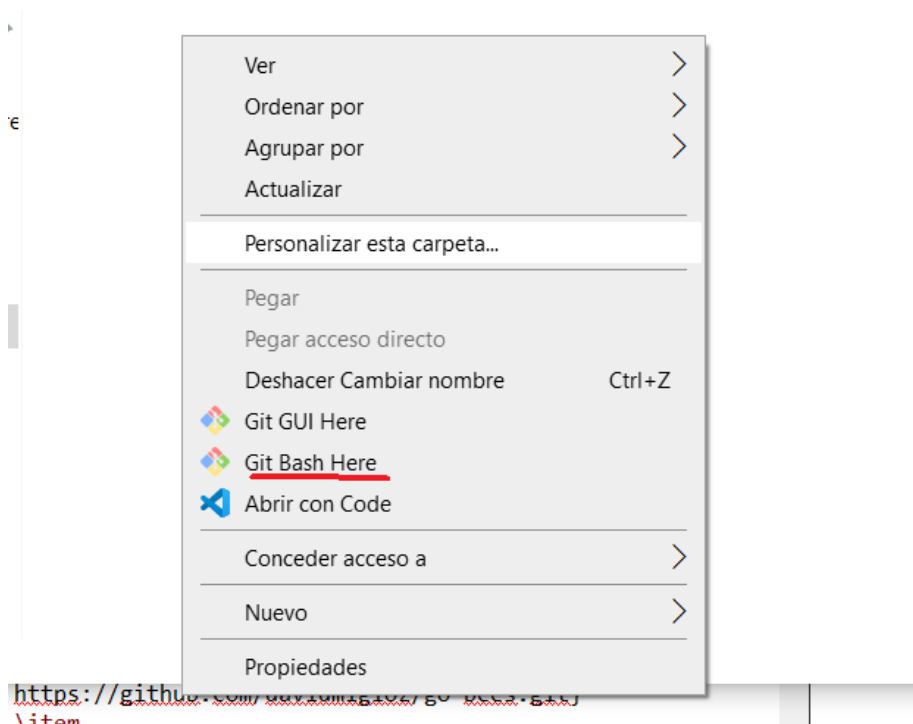


Figura D.9: Abrir git bash

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

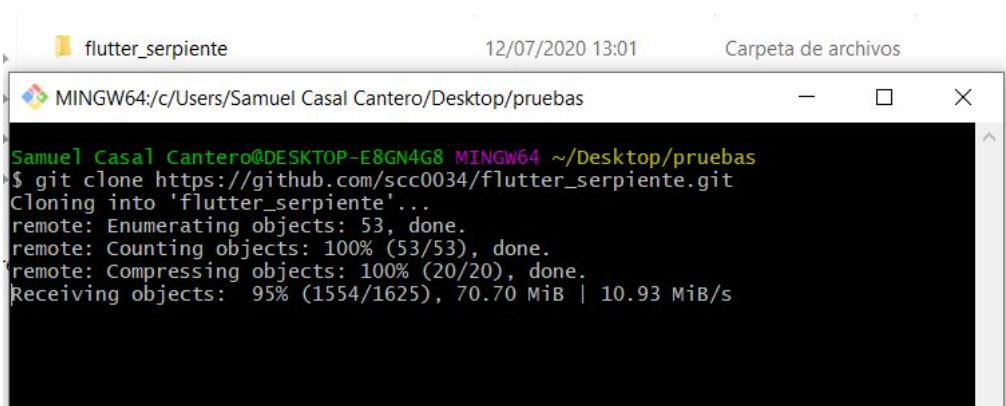


Figura D.10: Comando para clonar el repositorio.

## Instalación de los paquetes:

Una vez tengamos el proyecto clonado, debemos de ejecutar el comando `flutter upgrade`, para traer todos los paquetes, ya que si no las importaciones de cada uno de las páginas 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 paquetes

Una vez lo hagamos debemos de reiniciar Visual Studio Code.

## Ejecución del proyecto:

Podemos hacerlo de tres maneras:

1. Descargar la última versión de subida a la **Play Store**, esto solo es válido para el despliegue de la aplicación, no para desarrollo.
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, debemos de ir a Andorid Studio y arrancar una de las máquinas virtuales **D.12** . Si en la lista tenemos dispositivos, ejecutamos el siguiente comando `flutter run -d emulator-id`, siendo el id, el número de la máquina virtual que nos muestra el comando anterior.
3. Desde Visual Studio Code (debemos de tener las herramientas de visual studio), damos al **F5** nos pide que dispositivo queremos arrancar, en el caso de que tengamos un movil físico conectado, aparecerá en la lista, una vez elijamos donde se lanza, lo veremos ahí.**D.12**.

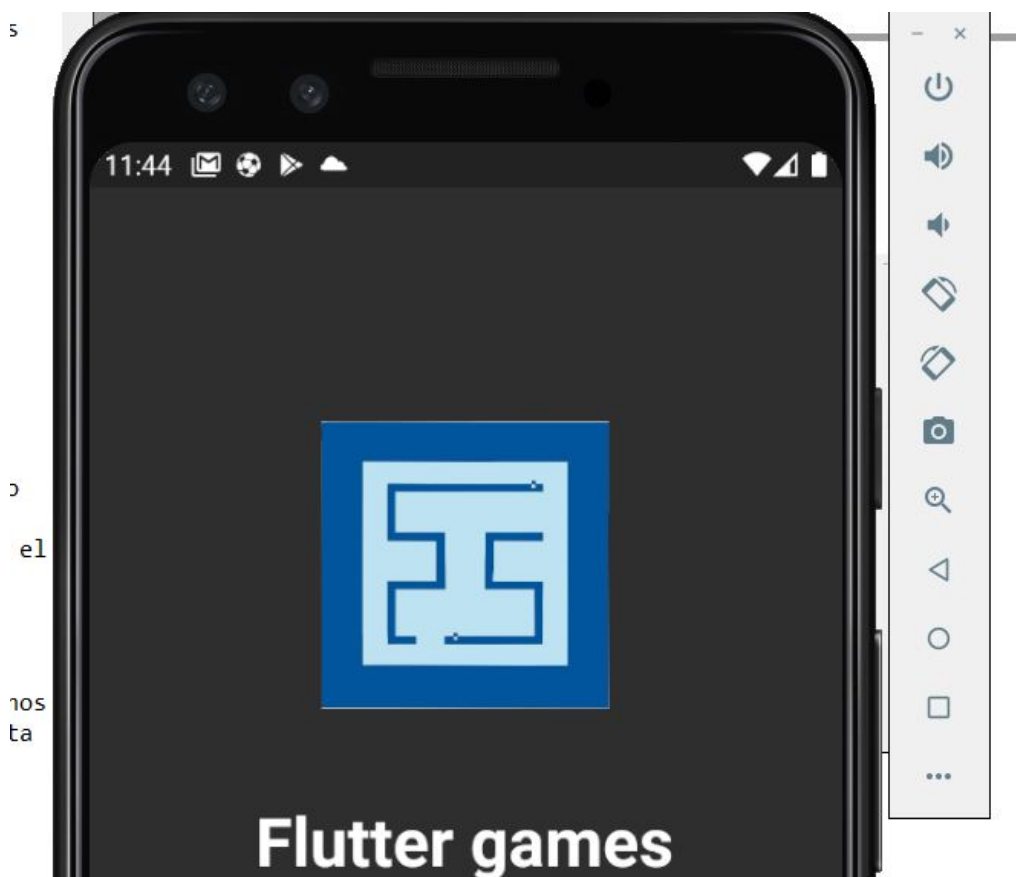


Figura D.12: Máquina virtual

## Generar los apks

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

1. **Para la tienda:** Generar el *bundle* [?] para hacer el despliegue en *Play Store*, para ello el comando que tenemos que escribir en la consola es: `flutter build appbundle`. Nos 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*, debemos mirar que versión de la app tenemos en 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 nos dice la ruta en la que se encuentran los ficheros generados.

## D.5. Pruebas del sistema

Para validar el correcto funcionamiento del producto, se han hecho diferentes tipos de pruebas: unitarias, de integración y de sistema. Hay que destacar que Flutter tiene herramientas para la automatización de test (*Appian* es una de ellas), pero debido a lo ajustado del 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, prácticamente es otro trabajo final de grado, que se puede incluir la aplicación, por eso es otra de las líneas futuras con las que seguir, con metodologías de programación dirigida por los test, como es el caso de TDD: Test-Driven Development.

### Pruebas de unitarias:

Se han realizado de forma simple o *hardcodeando prints* en el propio código y mostrándolos por el terminal en tiempo de ejecución. Sabiendo que la aplicación es sencilla, puedo permitírmelo, pero en el caso de continuar con el desarrollo, sería algo imprescindible. 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](#):

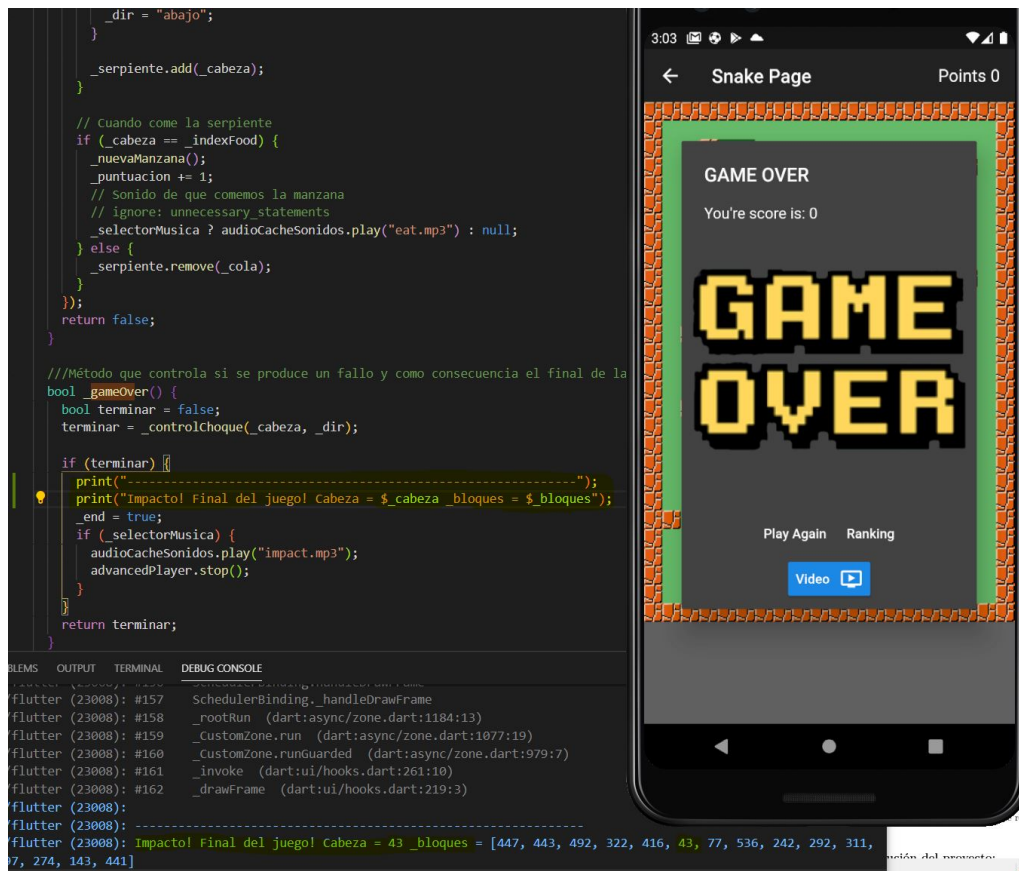


Figura D.13: Máquina virtual

### Pruebas de caja negra:

En otros casos 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 ???. Después la tabla de los diferentes casos de prueba con los resultados obtenidos ???.

```

/// 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 siguientes tablas:

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 el caso de que no se devuelve 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:

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 necesite 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 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*, ya que esta cuenta con el SDK mínimo con el que se ha desarrollado la aplicación, siendo este el 21 [?].
- 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 52.

## 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 debemos permitir dando permisos de la siguiente manera:

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 games**, que es el enlace de descarga para dispositivos móviles Android. La versión disponible es la 6, ya que he tenido que realizar varias pruebas, con el fin de validar la disponibilidad, por eso el número que tiene.

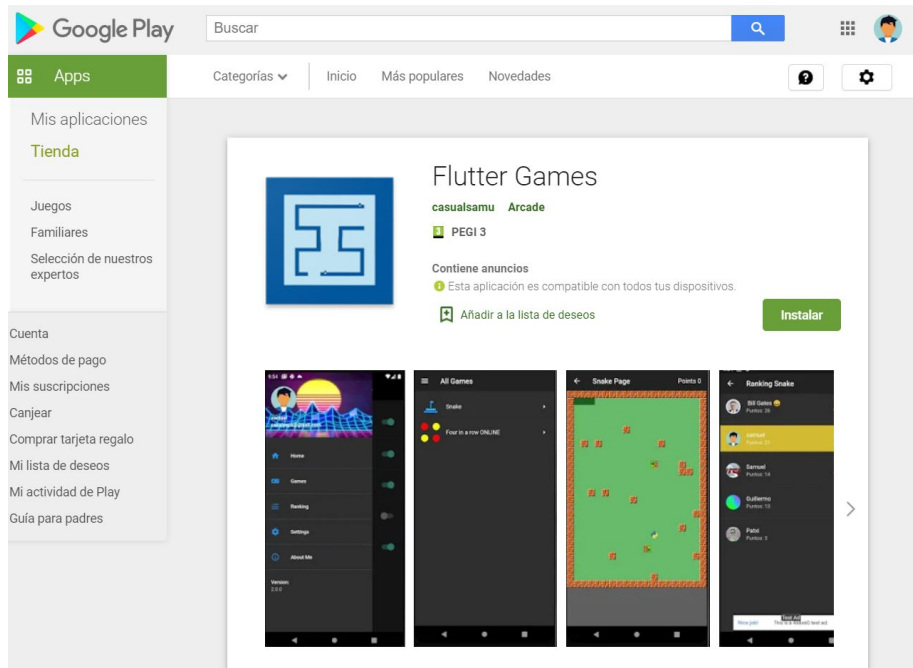


Figura E.1: Tienda con el proyecto

El proceso de instalación es simple, solo tenemos que dar al botón de instalar. En el caso de que nos encontremos en el navegador web, nos deja elegir el dispositivo que tenemos vinculado a nuestra cuenta de *Google*. Si por otra parte nos encontramos desde el terminal, se instalará sin ningún problema.

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

## E.4. Manual del 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 54.
- Home 56.
- About 60.

- Settings 59.
- Menú de juegos 62.
- Snake 63.
- Ranking 67.
- Cuatro en raya online menú 68:.
  - Invitar 69.
  - Unirse 71.
  - Juego 73.

## 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 esta permitido el acceso mediante anonimato.

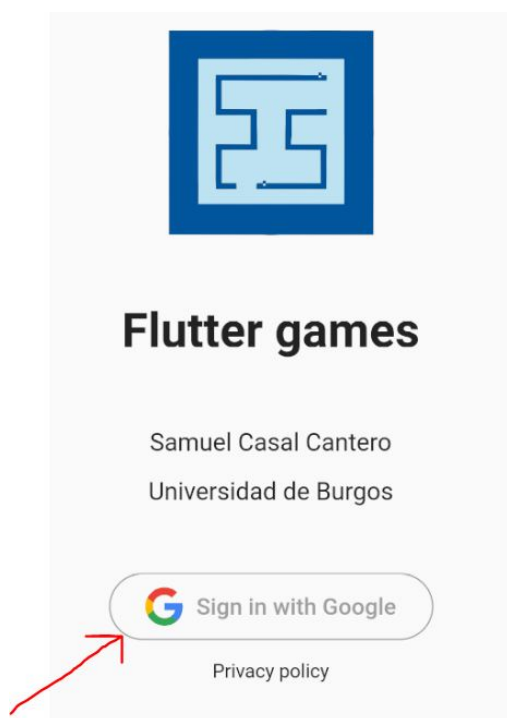


Figura E.2: Log in page

Cuando pulsamos en el botón de *sign in with Google*, nos aparecerá la ventana siguiente E.3, donde nos pedirá que ingresemos el usuario de nuestra cuenta de *Google*. También se puede consultar la política de privacidad E.4

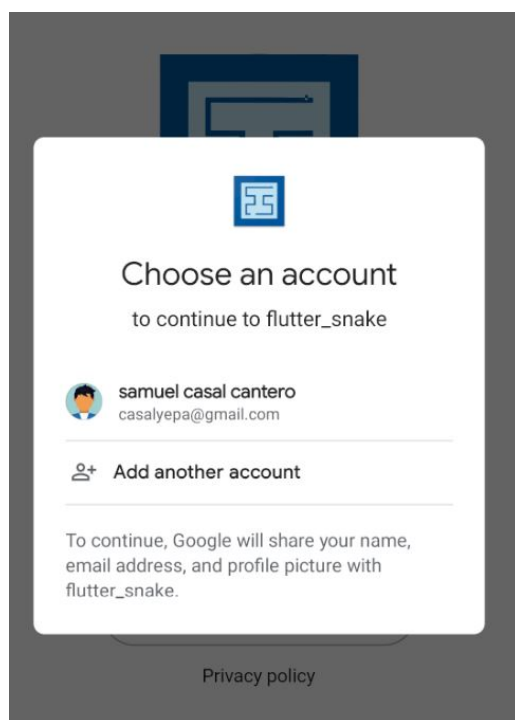


Figura E.3: Formulario sign in

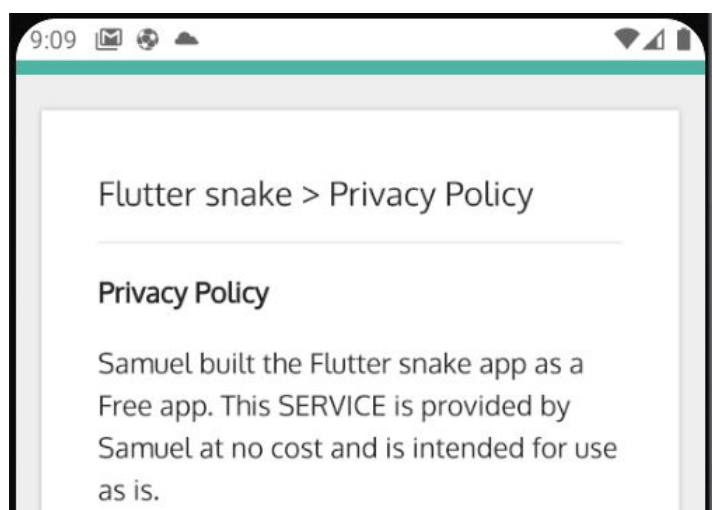


Figura E.4: Política de privacidad

Una vez completemos este proceso, la ventana siguiente a la que nos redirige la aplicación es el Home [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 E.7. Otra de las formas de entrar a este es pulsando el menú hamburguesa.

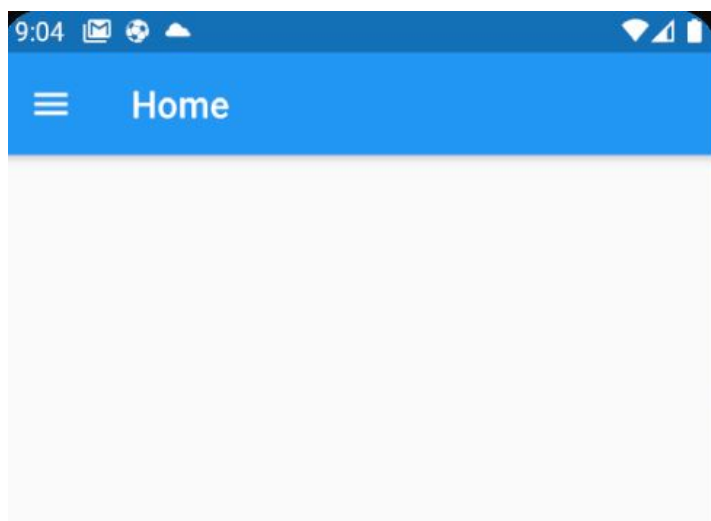


Figura E.5: Home

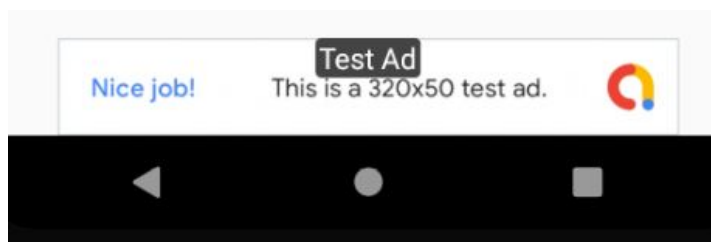


Figura E.6: Banner

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



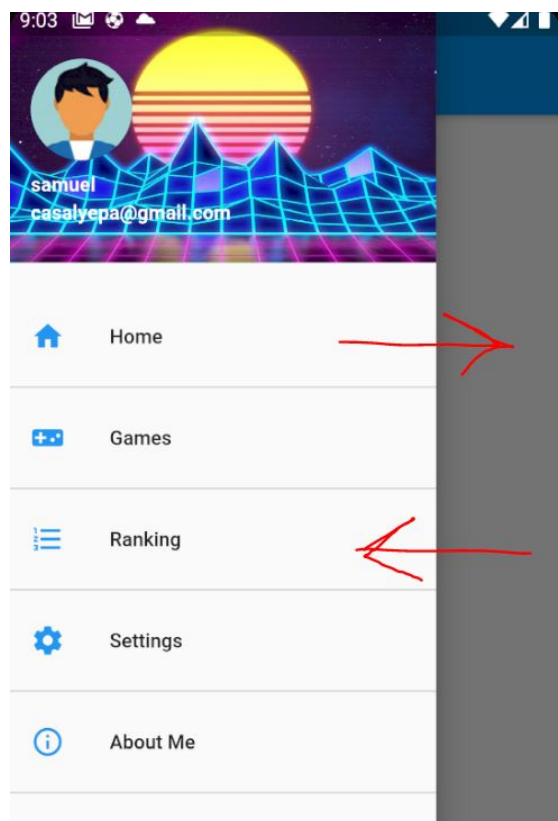


Figura E.7: Menú lateral

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

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

En el caso de que queramos salir de la sesión que tenemos abierta, debemos de pulsar en nuestra imagen E.8 y nos sale un cuadro de alerta con las opciones disponibles, pulsaremos en *Sign out* E.9.

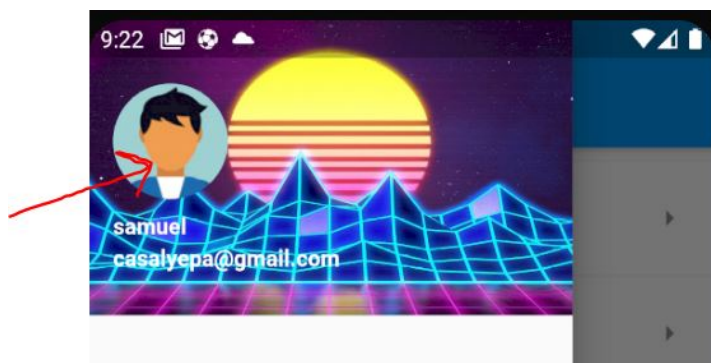


Figura E.8: Mostrar sign out

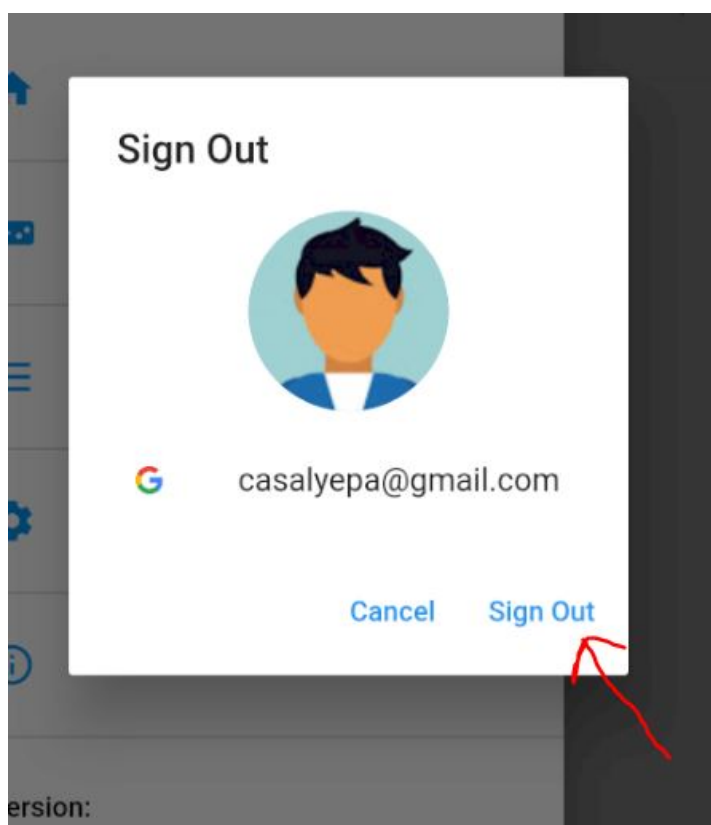


Figura E.9: Sign Out

## Settings

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

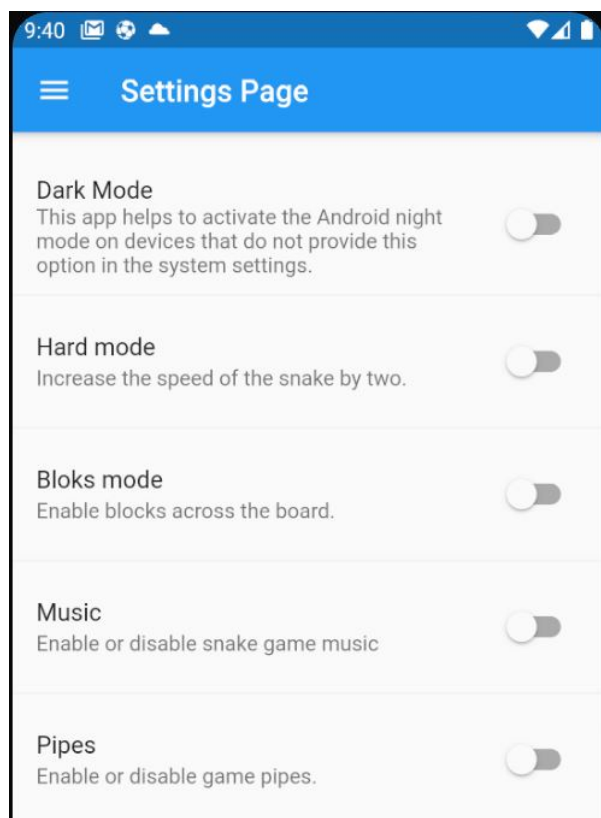


Figura E.10: Settings

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 problemas de dificultad visual E.11.

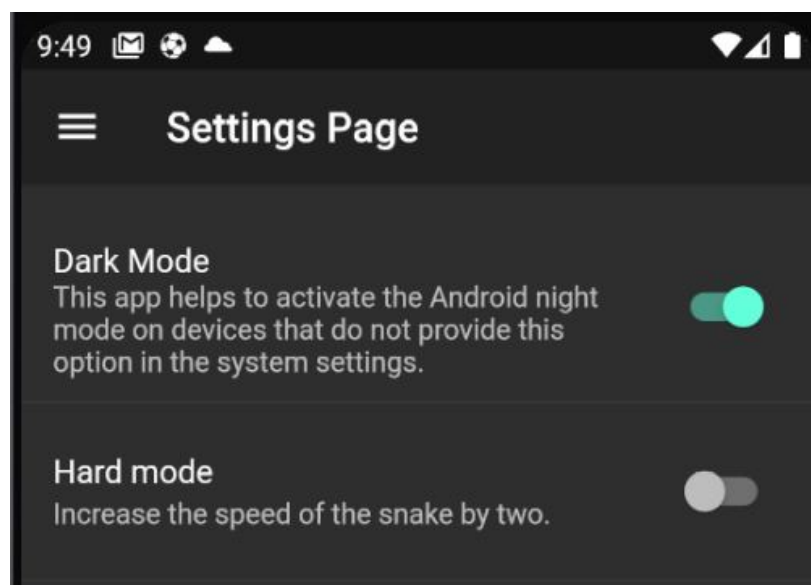


Figura E.11: Dark Mode

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

## About me

Pretende mostrar la información referente al creador de la aplicación, como muestra el diseño [E.12](#). Se aprecia un *corousel*, con imágenes de los lenguajes de programación conocidos y debajo de este unos icono botones, que enlazan con las páginas de interés.

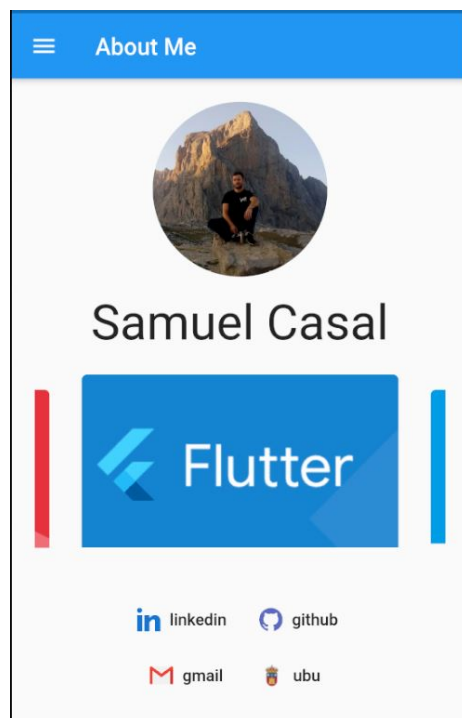


Figura E.12: About me

La lista de *iconbuttons*, con los enlaces:

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

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

Para 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 nos 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 [E.14](#).

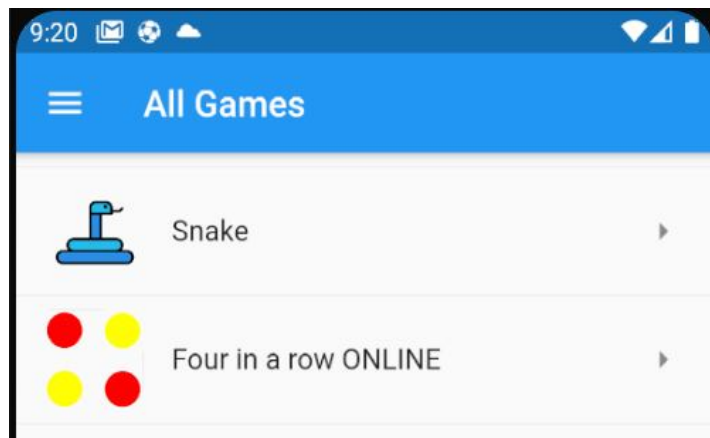


Figura E.14: Games menú

## Snake

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

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

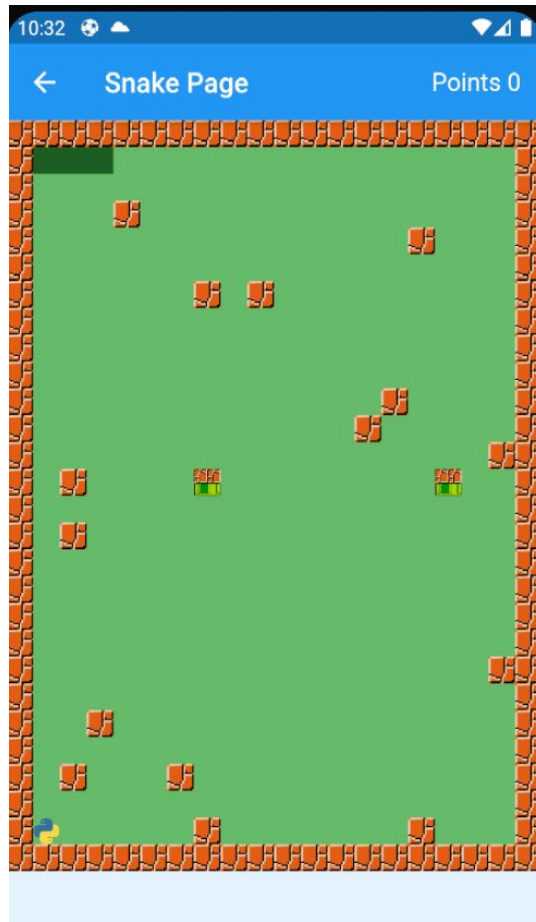


Figura E.15: Snake game

El juego finaliza, desplegando la ventana [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* E.16, tenemos 3 opciones de las que 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 la mejor obtenida en anteriores partidas, para el jugador que se encuentra logueado, se nos mandará al formulario E.17 de ingreso de datos. En el caso de que no tengamos mejora, se puede ver el ranking E.19.

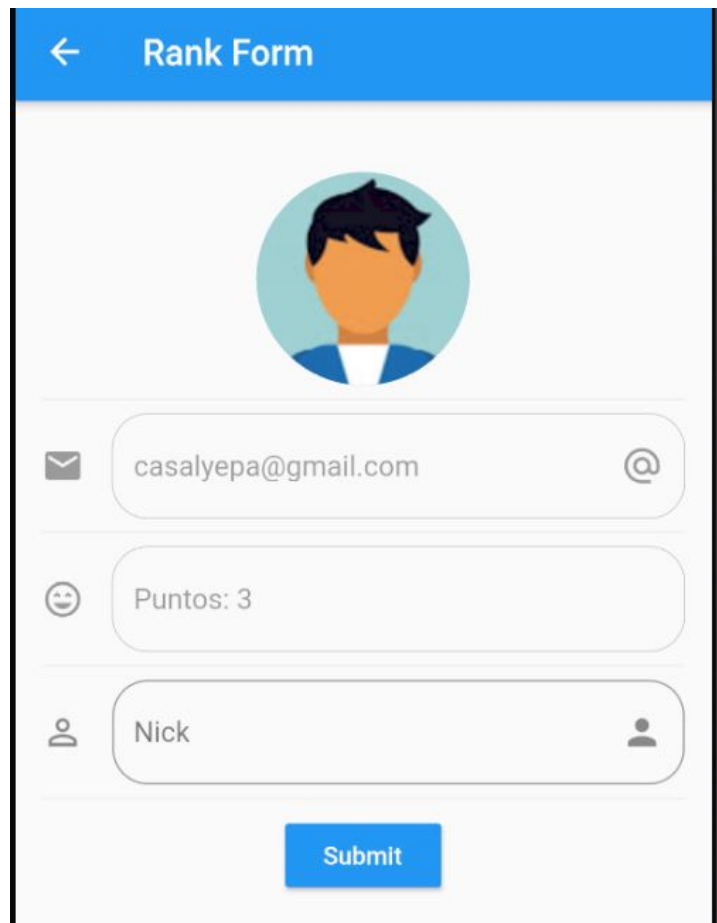
The image shows a mobile application screen titled "Rank Form". At the top is a blue header bar with a white back arrow icon and the text "Rank Form". Below the header is a circular profile picture placeholder showing a person with dark hair and an orange face. Underneath the profile picture are three 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 and contains the text "Puntos: 3". 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 is a blue button with the white text "Submit".

Figura E.17: Formulario del ranking

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

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

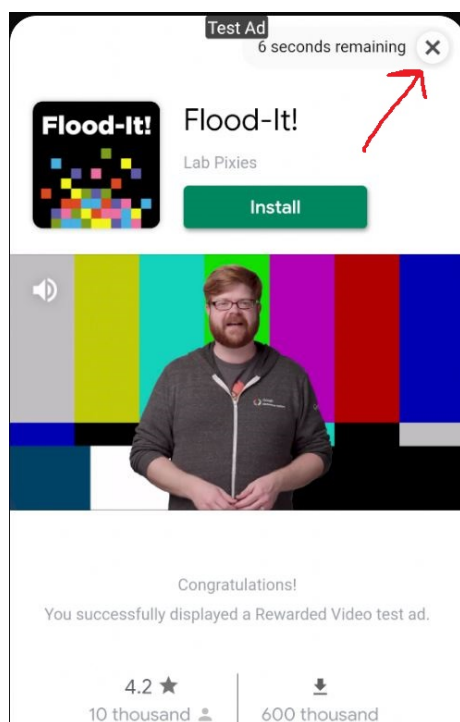


Figura E.18: Vídeo recompensa

Es necesario que se permanezca 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 [E.15](#). Según que usuario entre a ver el contenido de este [E.19](#), se verá una línea de color dorado que indicando el nivel obtenido.

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

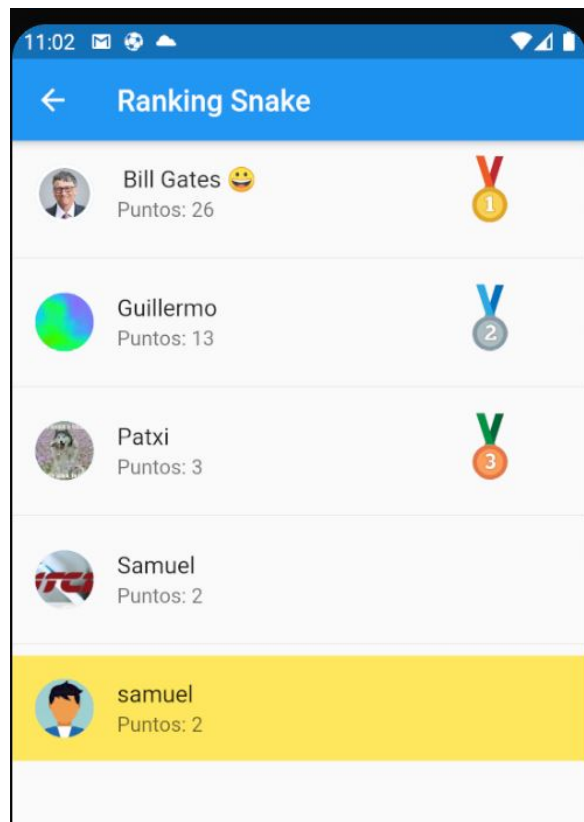


Figura E.19: Ranking

## Menú juego cuatro en raya online

Es la pantalla inicial del juego [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 que se una (*join game*), a la partida creada.

Por lo que tenemos dos botones que nos redirigen a:

- Invitar a un amigo [E.21](#).
- Unirse a una partida creada [E.24](#).

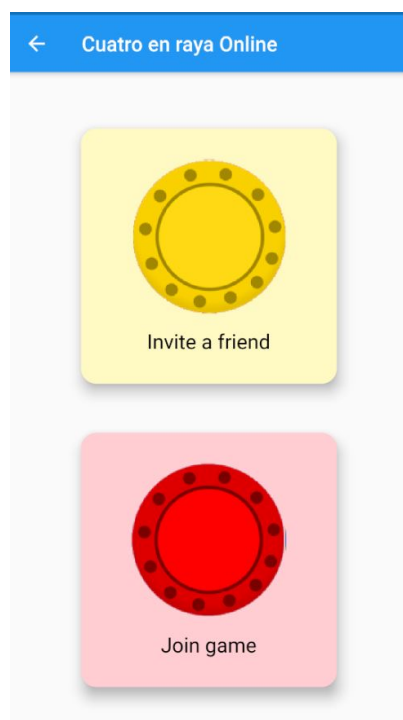


Figura E.20: Menú del cuatro en raya

## 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 tenemos 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 E.20. Es el mismo efecto que si se da a la pestaña de volver atrás.

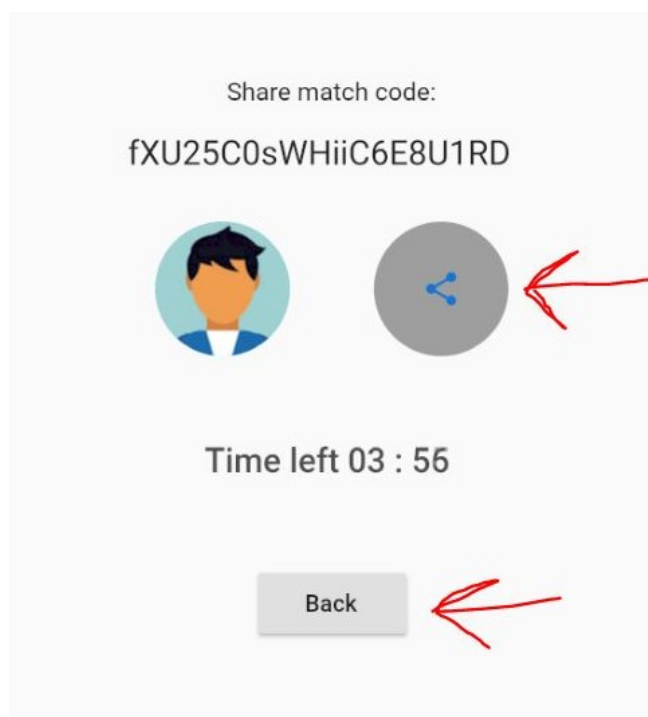


Figura E.21: Invitar cuatro en raya

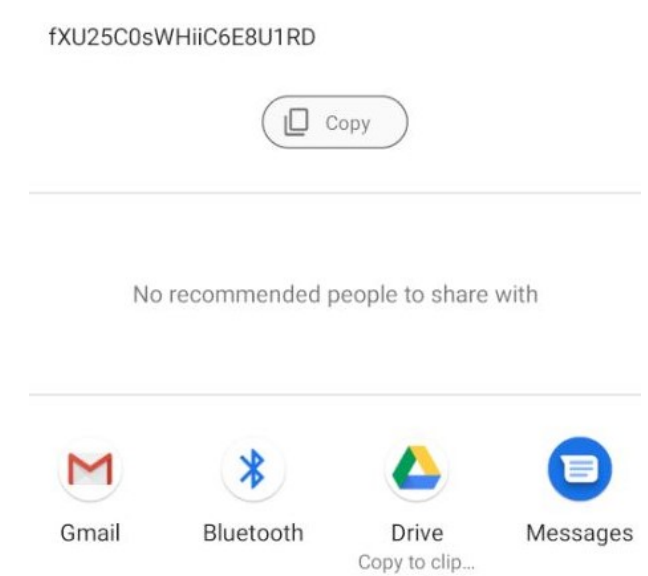


Figura E.22: Compartir clave

Cuando se pulsa en el botón de compartir nos sale un cuadro de diálogo 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 invitado, saliendo el cuadro de diálogo E.23.

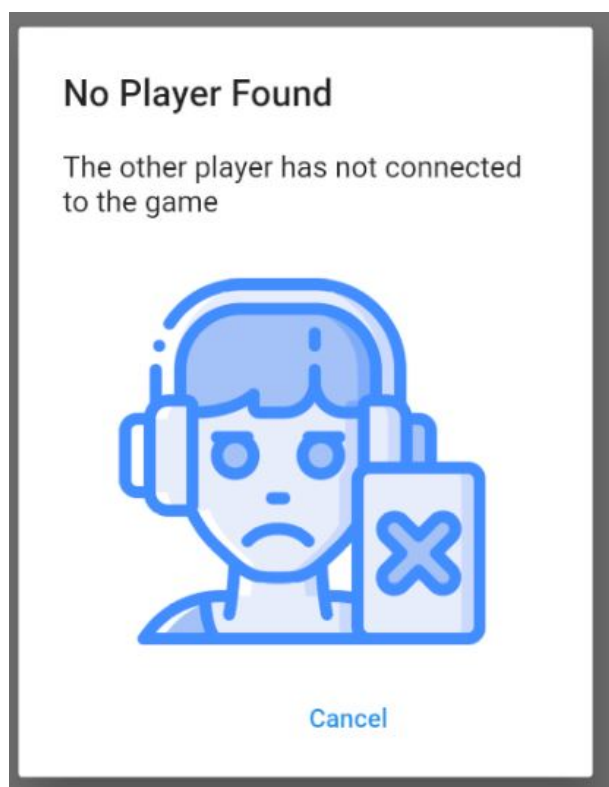


Figura E.23: Sin jugador

## Unir

Pantalla con un formulario de entrada de datos 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 E.25.

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

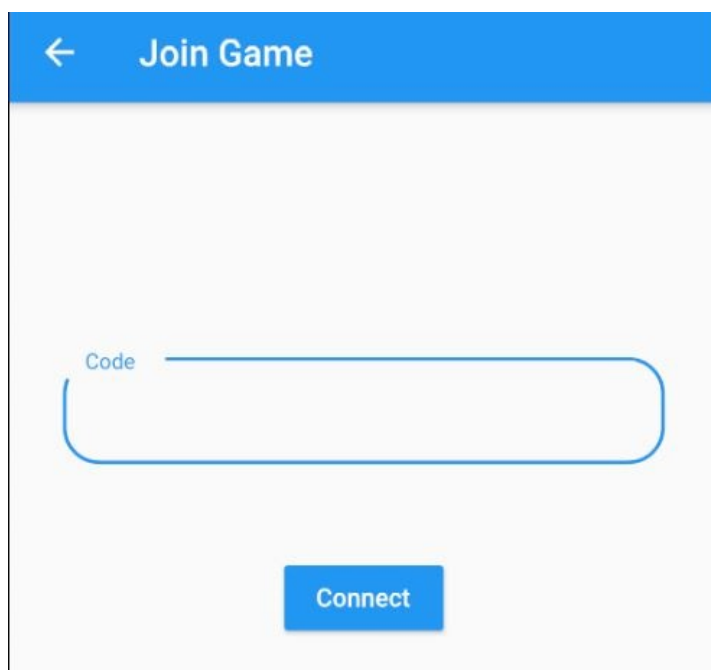


Figura E.24: Formulario para unirse a partida

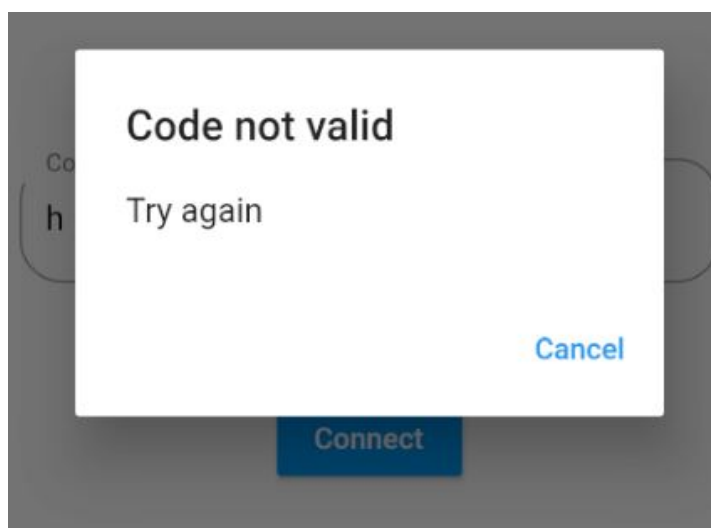


Figura E.25: Alerta de clave no válida



## Juego

Página con el contenido del juego 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 E.27.

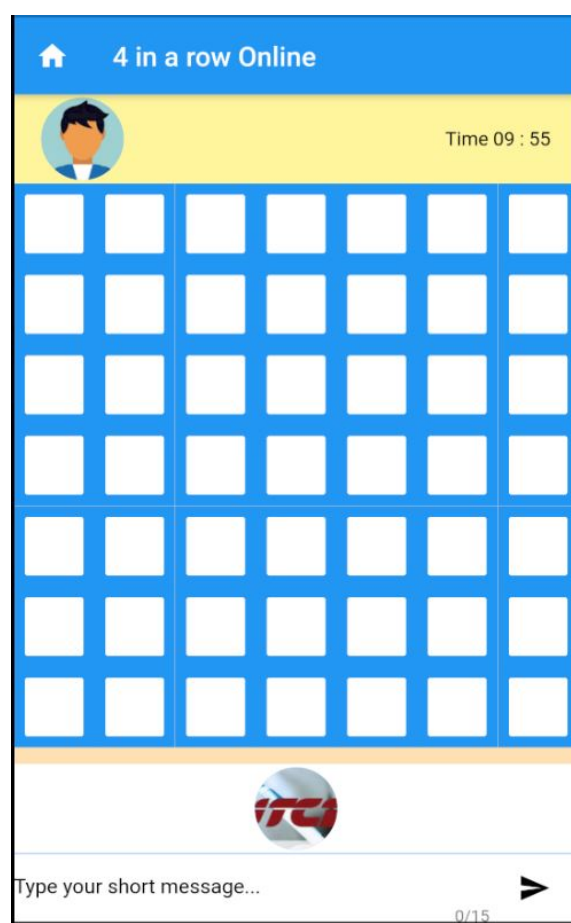


Figura E.26: Cuatro en raya

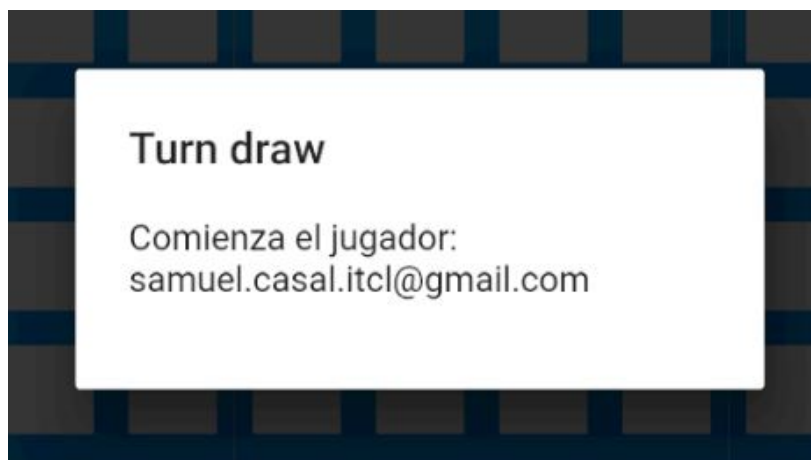


Figura E.27: Sorteo del turno

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

- Cabecera superior 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 nos 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.

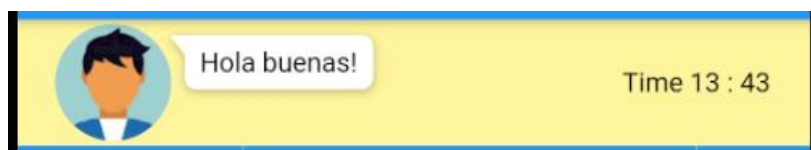


Figura E.28: Zona superior

- En la parte central E.29 tenemos 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.

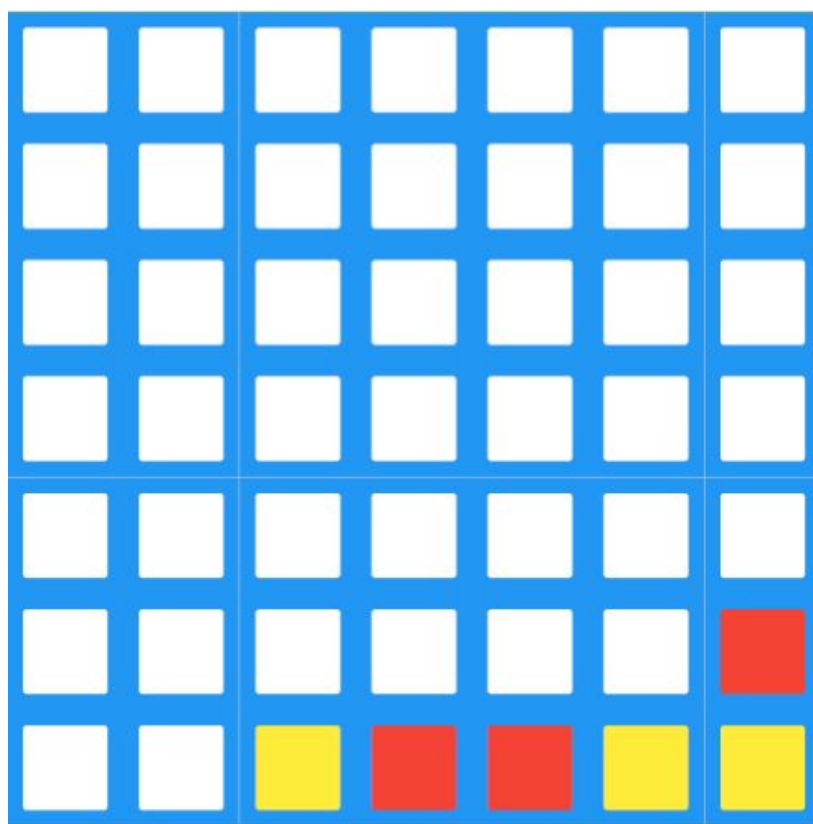


Figura E.29: Tablero

- En la parte inferior [E.30](#), se encuentra la imagen de del aliado, es decir, puede ser el jugador que se unió o que creo 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.



Figura E.30: Zona personal

- Campo para introducir mensajes [E.31](#), este está limitado a 15 caracteres, porque 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.

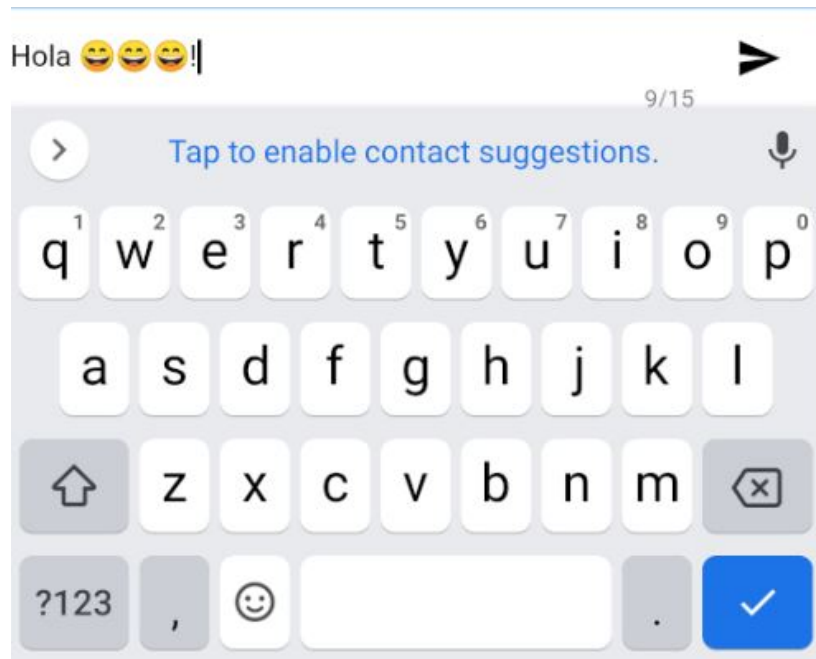


Figura E.31: Envío de mensaje

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 que consiguió ganar la partida, como se puede ver en la imagen siguiente [E.32](#)



Figura E.32: Final del cuatro en raya



---

## **Bibliografía**

---