



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 — 12 de julio
de 2020

Tutor: José Francisco Díez Pastor y César
García Osorio

Índice general

Índice general	I
Índice de figuras	III
Índice de tablas	IV
Apéndice A Plan de Proyecto Software	1
A.1. Introducción	1
A.2. Planificación temporal	1
A.3. Estudio de viabilidad	6
Apéndice B Especificación de Requisitos	11
B.1. Introducción	11
B.2. Objetivos generales	11
B.3. Catalogo de requisitos	11
B.4. Especificación de requisitos	11
Apéndice C Especificación de diseño	13
C.1. Introducción	13
C.2. Diseño de datos	13
C.3. Diseño procedimental	13
C.4. Diseño arquitectónico	13
Apéndice D Documentación técnica de programación	15
D.1. Introducción	15
D.2. Estructura de directorios	15
D.3. Manual del programador	18

D.4. Compilación, instalación y ejecución del proyecto	22
D.5. Pruebas del sistema	26
Apéndice E Documentación de usuario	31
E.1. Introducción	31
E.2. Requisitos de usuarios	31
E.3. Instalación	31
E.4. Manual del usuario	31
Bibliografía	33

Índice de figuras

A.1. Sprint 1.	3
A.2. Sprint 2.	4
A.3. Sprint 3.	5
D.1. Google services	16
D.2. app/build.gradle	17
D.3. Comando generar clave keyStore	17
D.4. Variables entorno	19
D.5. Comando version Flutter	19
D.6. Máquinas virtuales	20
D.7. Firebase console	21
D.8. Play Store console	22
D.9. Abrir git bash	23
D.10.Comando para clonar el repositorio.	23
D.11.Actualización paquetes	24
D.12.Máquina virtual	25
D.13.Máquina virtual	27
D.14.Código para las pruebas de caja negra	28

Índice de tablas

A.1. Equivalencias <i>Story Points</i> y tiempo estimado	2
A.2. Coste hardware	7
A.3. Coste personal	7
A.4. Coste cuota de la seguridad social	8
A.5. Coste vario	8
A.6. Licencias de bibliotecas y herramientas utilizadas	9
A.7. Fuente del contenido audiovisual	9
D.1. Clases de equivalencia	28
D.2. Derivación en casos de prueba de caja negra	29

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.

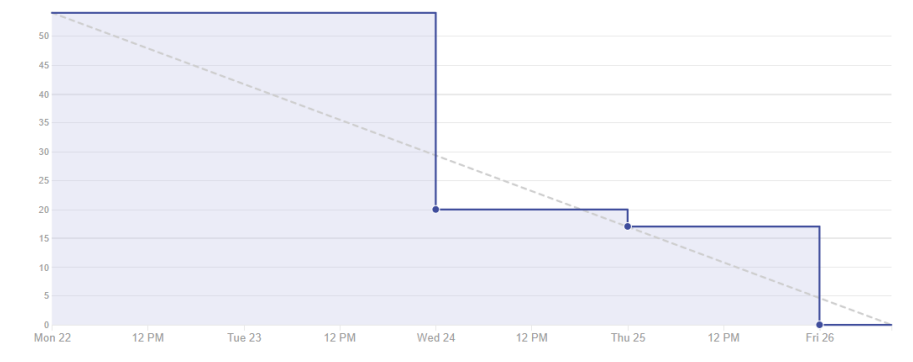


Figura A.1: Sprint 1.

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**

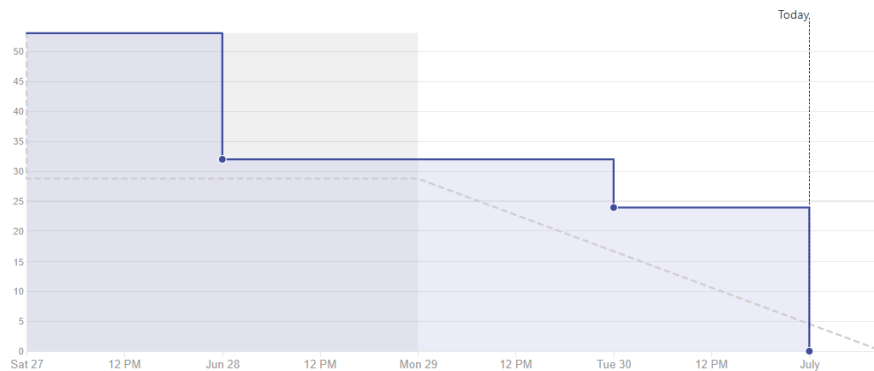


Figura A.2: Sprint 2.

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](#)

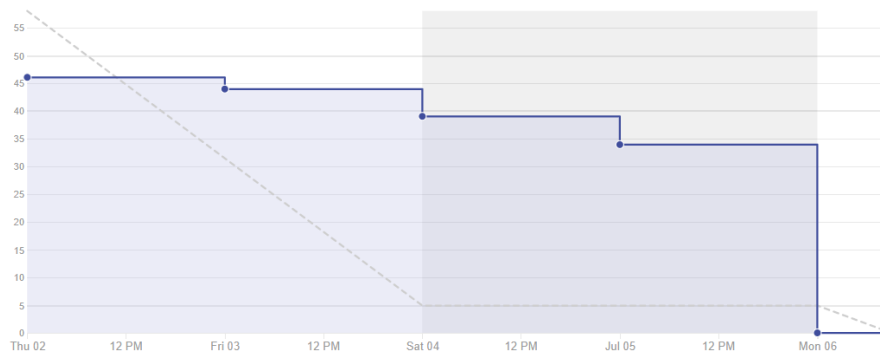


Figura A.3: Sprint 3.

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.

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
- B.2. Objetivos generales
- B.3. Catalogo de requisitos
- B.4. Especificación de requisitos

Apéndice C

Especificación de diseño

- C.1. Introducción
- C.2. Diseño de datos
- C.3. Diseño procedimental
- C.4. Diseño arquitectónico

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"
}

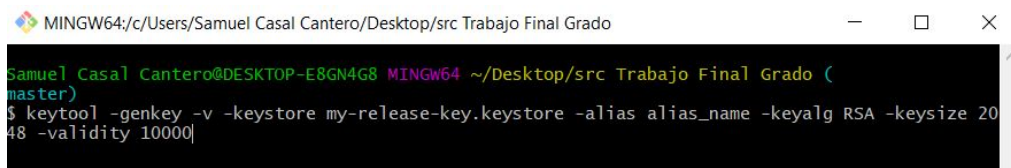
//NUEVO PLAY STORE
signingConfigs {
    release {
        keyAlias keystoreProperties['keyAlias']
        keyPassword keystoreProperties['keyPassword']
        storeFile file(keystoreProperties['storeFile'])
        storePassword keystoreProperties['storePassword']
    }
}

//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 prompt is "Samuel Casal Cantero@DESKTOP-E8GN4G8 MINGW64 ~/Desktop/src Trabajo Final Grado (master)". 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 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:

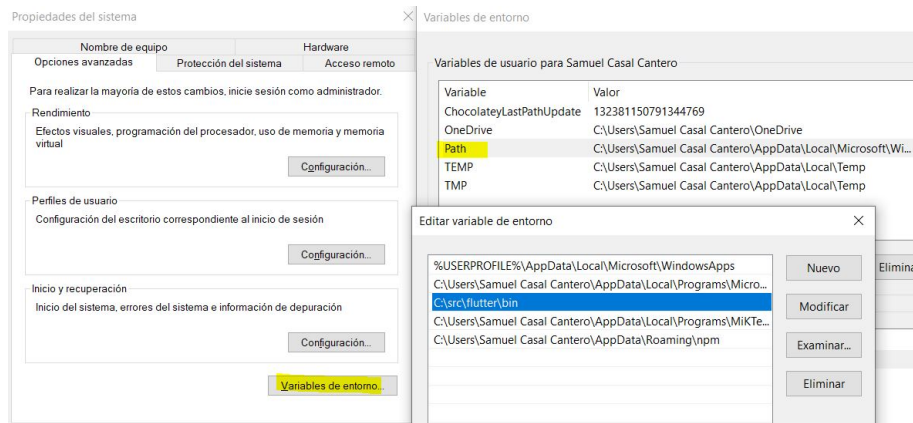


Figura D.4: Variables entorno

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.

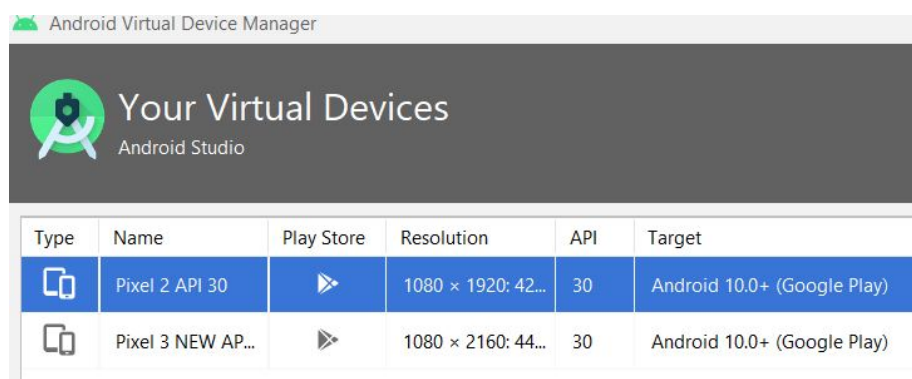


Figura D.6: Máquinas virtuales

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

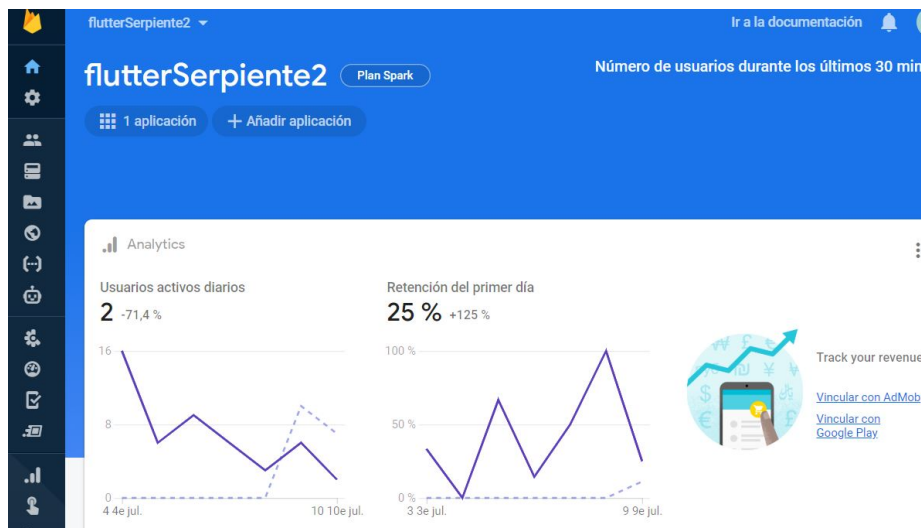


Figura D.7: Firebase console

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

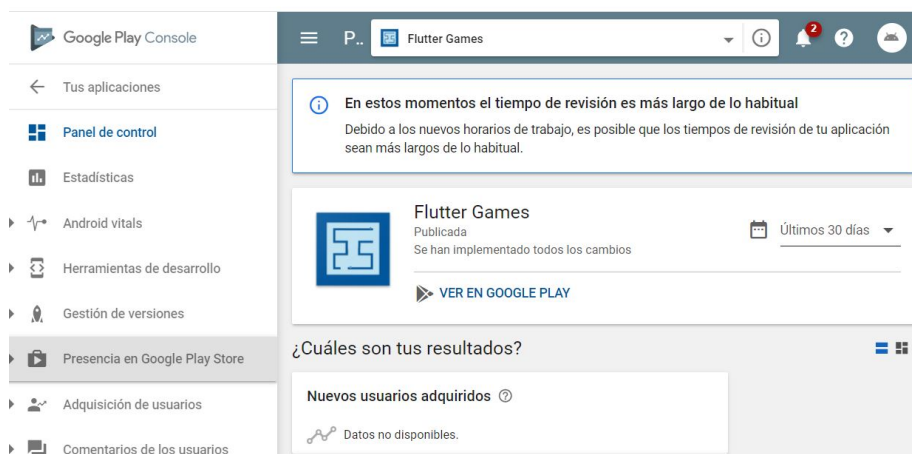


Figura D.8: Play Store console

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:

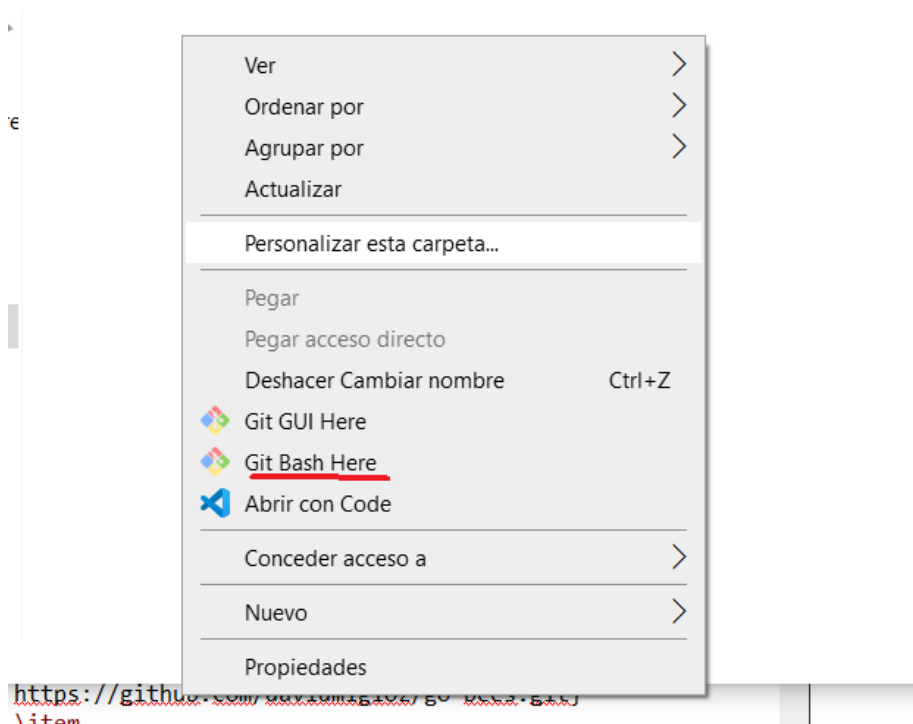


Figura D.9: Abrir git bash

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

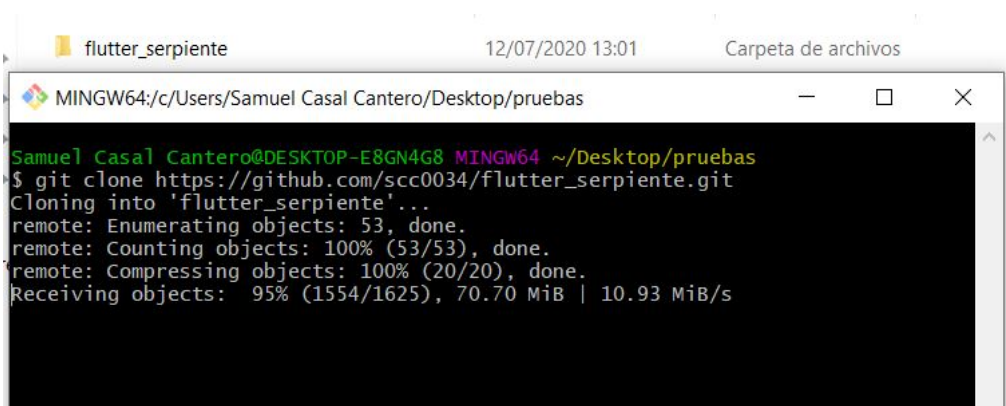


Figura D.10: Comando para clonar el repositorio.

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**.

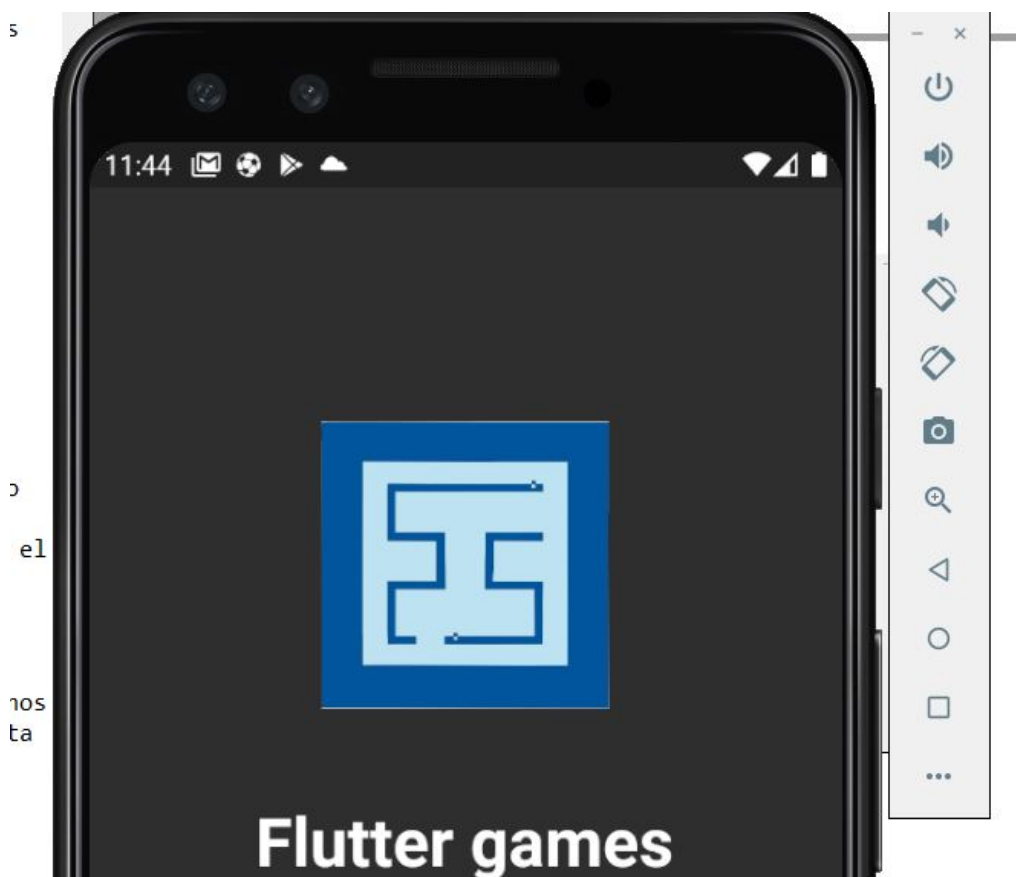


Figura D.12: Máquina virtual

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. 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. 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](#):

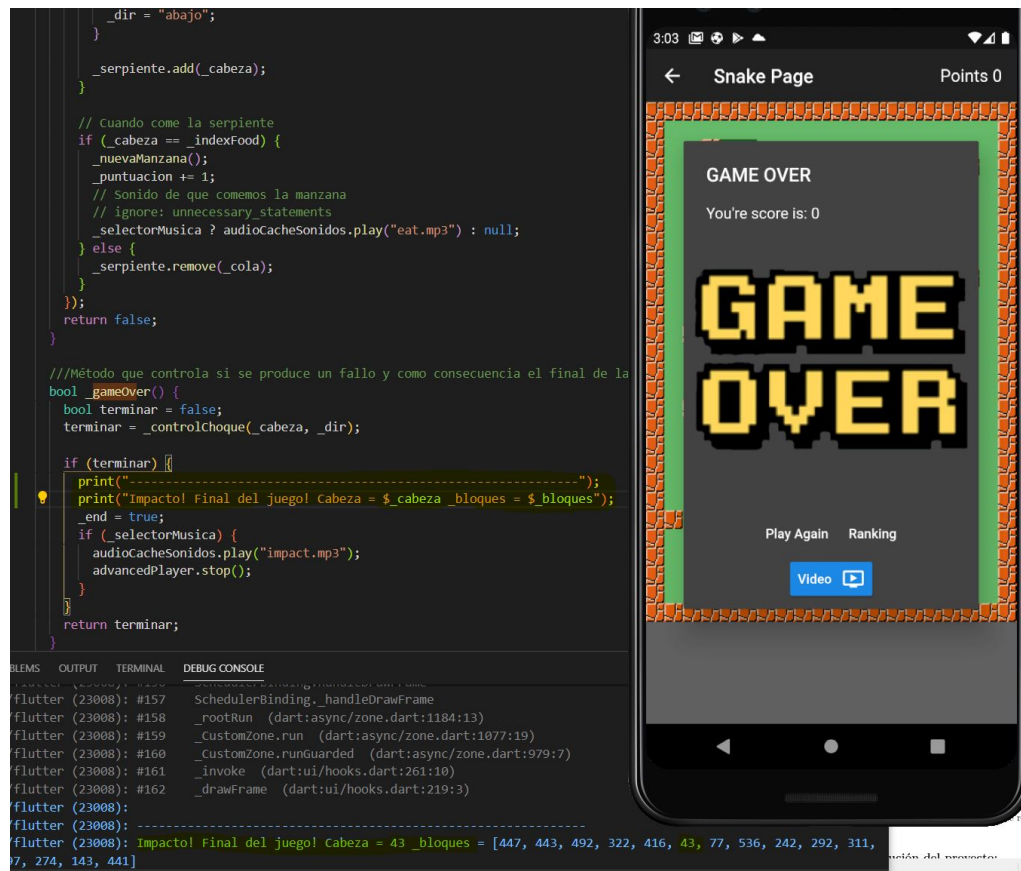


Figura D.13: Máquina virtual

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

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.

Apéndice E

Documentación de usuario

- E.1. Introducción
- E.2. Requisitos de usuarios
- E.3. Instalación
- E.4. Manual del usuario

Bibliografía
