Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

DESARROLLO DE UN JUEGO DE MESA EN APLICACIÓN MÓVIL

Alumno: Ángel Cruz Lupia

Tutor: Prof. D. Carlos Javier Ogayar Anguita
Dpto: Departamento de Informática

Julio, 2023

(Página intencionalmente en blanco)

Universidad de Jaén

Departamento de Informática

Don Carlos Javier Ogayar Anguita, tutor del Proyecto Fin de Carrera titulado: ‘Desarrollo de un Juego de Mesa en Aplicación Móvil’, que presenta Ángel Cruz Lupia, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Julio de 2023

El alumno:

CRUZ LUPIA ANGEL - 15522404A

Firmado digitalmente por CRUZ LUPIA ANGEL - 15522404A
Fecha: 2023.07.10 16:35:15 +02'00'

Ángel Cruz Lupia

El tutor:

OGAYAR ANGUITA CARLOS JAVIER - 26039170L

Firmado digitalmente por OGAYAR ANGUITA CARLOS JAVIER - 26039170L
Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-26039170L, givenName=CARLOS JAVIER, sn=OGAYAR ANGUITA, cn=OGAYAR ANGUITA CARLOS JAVIER - 26039170L
Fecha: 2023.07.10 16:26:37 +02'00'

Carlos Javier Ogayar Anguita

(Página intencionalmente en blanco)

Agradecimientos

Quiero expresar todo mi agradecimiento a mi familia, en especial a mis padres Ángel y Paqui, a mis hermanos, Alba y Joaquín, y mi abuela Rosa, que siempre creyeron en mi y me han apoyado desde el primer hasta el último día en este proceso tan largo llamado Ingeniería.
Sin olvidarme del resto de mi familia, de mis tios, tías y primos, gracias a todos.

Dar las gracias a las personas que me han rodeado todo este tiempo, personas muy cercanas que me he llevado en todo este periodo, amigos de toda la vida y compañeros de carrera, que me han ayudado a tener una carrera universitaria en todo su explendor, subidas, bajadas, decepciones pero muchas alegrias y conocimientos.

Dar las gracias a mi tutor Carlos, por resolver todas las dudas planteadas y ayudarme en las situaciones que lo he necesitado.

Por último, dar las gracias en general al “Grado en Ingeniería Informática”, porque gracias a él he adquirido infinidad de conocimientos, me ha dado la oportunidad de poder vivir en el extranjero y conocer lugares maravillosos, conocer gente estupenda y disfrutar de todo ello.

Gracias de corazón.

FICHA DEL TRABAJO FIN DE TÍTULO	
Titulación	Grado en Ingeniería Informática
Modalidad	Proyecto de Ingeniería
Especialidad (solo TFG)	Sistemas de Información
Mención (solo TFG)	Sin mención
Idioma	Español
Tipo	Específico
TFT en equipo	No
Autor/a	Ángel Cruz Lupia
Fecha de asignación	02/11/2022
Descripción corta	Desarrollo de juego de cartas de creación propia en una aplicación móvil. El juego consistirá en una introducción de jugadores de los cuales se seleccionará de manera aleatoria uno de ellos para que escoja una carta de color rojo (decir la verdad) o de color negro (decir mentira). Seguidamente, se vuelve a hacer una tirada aleatoria para que uno de los jugadores realice una pregunta al primer jugador seleccionado que previamente ha elegido una carta. Después se hace una ronda recorriendo a todos los jugadores restantes los cuales deben de elegir una carta (de color rojo o negro) intentando adivinar cual ha sido la elección del primer jugador basándose en la respuesta a la pregunta realizada y en que el color rojo es “verdad” y el negro es “mentira”; los jugadores que acierten, conseguirán un punto, la tirada finaliza y comienza una nueva tirada.

NORMAS APLICADAS EN ESTE DOCUMENTO	
LOCALES	
TFT-UJA:2017	Normativa de Trabajos Fin de Grado, Fin de Máster y otros Trabajos Fin de Título de la Universidad de Jaén (Normativa marco UJA aprobada en Consejo de Gobierno)
TFT-EPSJ:2017	Normativa sobre Trabajos Fin de Grado y Fin de Máster en la Escuela Politécnica Superior de Jaén (Normativa EPSJ aprobada en Junta de Escuela)
TFT-EPSJ	Criterios de evaluación y normas de estilo para TFG y TFM de la Escuela Politécnica Superior de Jaén
NACIONALES E INTERNACIONALES	
ISO 2145:1978	Documentación - Numeración de divisiones y subdivisiones en documentos escritos
UNE 50132:1994	Traducción de la ISO 2145
APA 6ª edición	Estilo de referencias y citas de APA (American Psychological Association)

NORMAS UTILIZADAS COMO BASE O REFERENCIA	
NACIONALES	
UNE 157001:2014	Criterios generales para la elaboración formal de los documentos que constituyen un proyecto técnico
UNE 157801:2007	Criterios generales para la elaboración de proyectos de sistemas de información
*Estas normas se han utilizado **como base o referencia** para la inclusión de algunos contenidos y definiciones sobre elaboración de proyectos, entendiendo como **proyecto** la documentación consensuada entre una empresa y un cliente, que da lugar al perfeccionamiento de un contrato para la elaboración de una obra o la prestación de un servicio. Por consiguiente, no debe esperarse la aplicación de estas normas en cuanto a la completitud de los contenidos ni a la organización de los mismos.*	

Contenido

1 Especificación del trabajo 13

1.1 Introducción 13
1.2 Objetivos del trabajo 14
1.3 Antecedentes y estado del arte 15
1.4 Requisitos iniciales 18
1.5 Alcance 19
1.6 Hipótesis y restricciones 19
1.7 Estudio de alternativas, viabilidad y elección 21
1.7.1 Plataforma 21
1.7.2 Android Studio 23
1.7.3 draw.io 25
1.7.4 Estadística de uso de las versiones Android 26
1.7.5 Windows 10 27
1.8 Tecnologías utilizadas 27
1.9 Metodología de desarrollo de software 29
1.9.1 Scrum 29
1.9.2 Iteraciones del proyecto: 31
1.10 Estimación del tamaño y esfuerzo 32
1.10.1 Historias de usuario 32
1.11 Diagrama de Gantt 33

2 Diseño inicial 35

2.1 Especificaciones del sistema 36
2.1.1 Historias de usuario 36
2.2 Análisis y diseño del sistema 37
2.2.1 Boceto 37
2.2.1.1 Boceto general inicial 37
2.2.1.2 Boceto pantalla de inicio 38
2.2.1.3 Boceto players 39
2.2.1.4 Boceto cartas 39
2.2.1.5 Boceto pregunta 40
2.2.1.6 Boceto cartas 2 41
2.2.1.7 Boceto puntuación 42
2.2.2 StoryBoards 43
2.2.3 Casos de uso 45

3 Desarrollo 56

3.1 Primera Iteración 56
3.2 Segunda Iteración 60
3.2.1 Actividad “MainActivity” 60
3.2.2 Actividad “introPlayers” (Introducción de jugadores) 62
3.3 Tercera Iteración 69
3.3.1 Actividad “Random Player” (Elección de jugador aleatorio) 69
3.3.2 Actividad “Eleccion_carta_1” 73
3.4 Cuarta Iteración 81

3.4.1 Actividad “Random_Player_2” (Elección de jugador aleatorio) 81
3.4.2 Actividad “Pregunta_Player” 85
3.4.3 Actividad “Elección_Carta_all_players” 87
3.5 Quinta Iteración 94
3.5.1 Actividad “Resultados” 94
3.6 Pruebas finales y documentación 100

4 Conclusiones y trabajos futuros 101

5 Apéndices 103

5.1 Guía original del Trabajo Fin de Título 103
5.2 Instalación y configuración del sistema 104
5.3 Manuales de usuario 108

6 Bibliografía 109

Índice de ilustraciones

Ilustración 1 - Baraja de cartas provenzal de principios del siglo XV. Museo Fournier de Naipes de Álava 15
Ilustración 2 - Juego de cartas en mesa 16
Ilustración 3 - Juego "El solitario" para PC 17
Ilustración 4 - Logo Android 22
Ilustración 5 - Android Studio 24
Ilustración 6 - Programa draw.io 25
Ilustración 7 - Fragmentación Android agosto 2022 26
Ilustración 8 - Diagrama de Gantt 34
Ilustración 9 - Boceto inicial proyecto 37
Ilustración 10 - Boceto pantalla de inicio 38
Ilustración 11 - Boceto Players 39
Ilustración 12 - Boceto cartas 39
Ilustración 13 - Boceto pregunta 40
Ilustración 14 - Boceto cartas 2 41
Ilustración 15 - Boceto Puntuación 42
Ilustración 16 - StoryBoard general 44
Ilustración 17 - Caso de uso pantalla de introducción de jugadores 46
Ilustración 18 - Caso de uso pantalla de Tirada Aleatoria 48
Ilustración 19 - Caso de uso pantalla Elección de carta 49
Ilustración 20 - Caso de uso pantalla de Tirada Aleatoria 2 50
Ilustración 21 - Caso de uso pantalla de Pregunta 51
Ilustración 22 - Caso de uso elección de carta resto de jugadores 53
Ilustración 23 - Caso de uso nueva tirada 55
Ilustración 24 - First fragment (prueba) 56
Ilustración 25 - Codigo XML (prueba) 57
Ilustración 26 - Código java prueba 58
Ilustración 27 - Foto inicial de Inicio de Juego (pantalla de carga) 60
Ilustración 28 - Foto final de Inicio de Juego (pantalla de carga) 61
Ilustración 29 - Código pantalla inicial 62
Ilustración 30 - Foto inicial de la actividad de introducción de jugadores 63
Ilustración 31 - Foto final de la actividad de introducción de jugadores 64
Ilustración 32 - Recyclerview 65
Ilustración 33 - Código clase "AdaptadorPlayer" 65
Ilustración 34 - Estilo para "recyclerview" 66
Ilustración 35 - Foto inicial de la actividad de elección de jugador aleatorio 69
Ilustración 36 - Foto final de la actividad de elección de jugador aleatorio 70
Ilustración 37 - Código JAVA asociado a la actividad "Random Player" 71
Ilustración 38 - Método "siguiente" 72
Ilustración 39 - Foto inicial de la actividad de elección de carta para jugador 1 74
Ilustración 40 - Foto final de la actividad de elección de carta para jugador 1 75
Ilustración 41 - Métodos "onClickButtonRed" y "onClickButtonBacl" 76

Ilustración 42 - Carta roja seleccionada....77
Ilustración 43 - Carta negra seleccionada....77
Ilustración 44 - Código XML de la actividad "Eleccion_carta_1" 78
Ilustración 45 - Código XML de la actividad "Eleccion_carta_1" 78
Ilustración 46 - Código método "siguiente" de la actividad "Eleccion_carta_1" 79
Ilustración 47 - Foto inicial de la actividad de elección de jugador aleatorio 2 81
Ilustración 48 - Foto final de la actividad de elección de jugador aleatorio 2....82
Ilustración 49 - Código asociado a la actividad "Random_Player_2" 83
Ilustración 50 - Método "siguiente" asociado a la actividad "Random_Player_2"....84
Ilustración 51 - Código asociado a la actividad "Pregunta_Player" 85
Ilustración 52 - Foto final de la actividad “Pregunta_Player” 86
Ilustración 53 - Foto inicial de la actividad “Pregunta_Player”....86
Ilustración 54 - Código para la realización del "AlertDialog"....88
Ilustración 55 - Foto del "AlertDialog" final....89
Ilustración 56 - Código asociado a la clase "Elección_carta_all_players" 91
Ilustración 57 - Foto final de la actividad “elección_carta_all_players”....92
Ilustración 58 - Foto inicial de la actividad “resultados”....95
Ilustración 59 - Código de la clase "AdaptadorPlayer"....96
Ilustración 60 - Foto de la actividad “itemplayer”. Diseño para ReyclerView de “resultados” 96
Ilustración 61 - Código XML de la actividad "itemplayer"....97
Ilustración 62 - Foto final de la actividad “resultados” 98
Ilustración 63 - Captura error de intalacion....105
Ilustración 64 - Captura solución de error....106
Ilustración 65 - Captura solución error 2....107

Índice de tablas

Tabla 1. Plantilla Casos de uso 45
Tabla 2. CU-01 - Introducir Jugador 47
Tabla 3. CU-02 - Añadir Jugador 47
Tabla 4. CU-03 - Eliminar Jugador 47
Tabla 5. CU-04 - Siguiente Pantalla 48
Tabla 6. CU-05 - Tirada Aleatoria 48
Tabla 7. CU-06 - Siguiente Pantalla 49
Tabla 8. CU-07 - Elección carta roja 49
Tabla 9. CU-08 - Elección carta negra 50
Tabla 10. CU-09 - Siguiente pantalla 50
Tabla 11. CU-10 - Tirada Aleatoria 51
Tabla 12. CU-11 - Siguiente pantalla 51
Tabla 13. CU-12 - Realizar pregunta 52
Tabla 14. CU-13 - Siguiente pantalla 52
Tabla 15. CU-14 - Elección carta roja 53
Tabla 16. CU-15 - Elección carta negra 54
Tabla 17. CU-16 - Enviar carta 54
Tabla 18. CU-17 - Siguiente pantalla 54
Tabla 19. CU-18 - Nueva partida 55

1 ESPECIFICACIÓN DEL TRABAJO

En este capítulo se presenta la especificación del trabajo, con una estructura y contenidos **inspirados** en los criterios y recomendaciones que establece la norma UNE 157801:2007 - "*Criterios Generales para la elaboración de proyectos de Sistemas de Información*".

A lo largo del documento se utilizarán términos y acrónimos cuya descripción aparecen en el apartado **¡Error! No se encuentra el origen de la referencia.** (**¡Error! No se encuentra el origen de la referencia.**).

1.1 Introducción

En esta documentación se describirán la propuesta, el diseño y el desarrollo de del proyecto planteado.

Este proyecto tratará sobre la realización de una aplicación para dispositivo móvil sobre un juego de mesa multijugador con interacción directa entre la aplicación y los jugadores.

En este documento, se podrá encontrar el inicio de la idea de la realización de la aplicación, así como su desarrollo, implementación y resultado final; demostrando su función en un dispositivo móvil.

El objetivo principal de este proyecto es, plasmar en una aplicación de invención propia, los conocimientos adquiridos y mostrar el resultado final de esta aplicación.

Los motivos de realización de este proyecto son principalmente la motivación de una idea propia y que puede llegar a muchos usuarios, gracias a la plataforma donde se implantará y la sencillez final del juego, el cual "virtualiza" un juego de mesa con cartas tradicionales y jugadores presenciales, pero en este caso, interactuando ambas partes.

1.2 Objetivos del trabajo

Los objetivos principales de este proyecto se vienen definidos sobre la siguiente lista:

- Estudio y diseño del proyecto a partir de una idea propia inicial.
- Analizar las distintas opciones para el desarrollo del proyecto
- Desarrollo del juego de mesa en una aplicación portable para poder jugar en cualquier lugar con un dispositivo móvil: Vuelo, coche (en un viaje largo):
    - Implementación de un juego de mesa de creación personal para dispositivos móviles, haciendo uso de la misma para jugar de manera portátil.
- Desarrollar de un nuevo tipo de juego
    - Crear un nuevo tipo de juego en aplicación móvil que no se tenga constancia de existir en las plataformas de descarga habituales.
- Diseñar las distintas pantallas del juego que se verán reflejadas en el proyecto final.
- Cubrir una alternativa a nivel tecnológico sobre el juego de mesa mencionado. Pudiéndose usar de manera totalmente portátil y poder tener una interacción entre los participantes físicos y la aplicación.

Finalizado el proyecto, obtendremos una aplicación donde quedará reflejado el proyecto y podrá ser instalada por los usuarios que deseen probar este nuevo juego.

1.3 Antecedentes y estado del arte

La ejecución del proyecto viene dada por el interés en crear una aplicación basada en un juego de mesa de creación personal; el cual fue “creado”/”inventado” por un grupo de amigos estudiantes durante un viaje al extranjero, en el cual disponían de una baraja de cartas de póker y durante el viaje fue creado de manera totalmente aleatoria el juego de mesa. Con el único inconveniente de tener que disponer de una baraja de cartas en todo momento, o al menos 2 de ellas.

Si vemos tiempo atrás, los juegos de cartas han sido de los juegos más populares en reuniones de todo tipo desde hace décadas, pero ¿quién inventó los juegos cartas? Nadie tiene una respuesta, tan solo conjeturas de que las cartas tal y como las conocemos, provienen de Asia pero se desconoce cómo llegaron a Europa.

Ilustración 1 - Baraja de cartas provenzal de principios del siglo XV. Museo Fournier de Naipes de Álava

El uso de las cartas es prácticamente incontable: juegos de mesa, juegos de estrategia, la magia (posiblemente de las modalidades más conocidas donde se usan cartas), juegos populares entre los que encontramos “El solitario”, “La brisca española” y juegos que llegan a considerarse deportes, como el “Póker”, donde se usan la baraja de carta Francesa, la más popular en nuestros días.

Ilustración 2 - Juego de cartas en mesa

A día de hoy, existen infinidad de esos juegos que hace años se jugaban en una mesa y a día de hoy disponemos de ellos en apps para nuestros dispositivos:

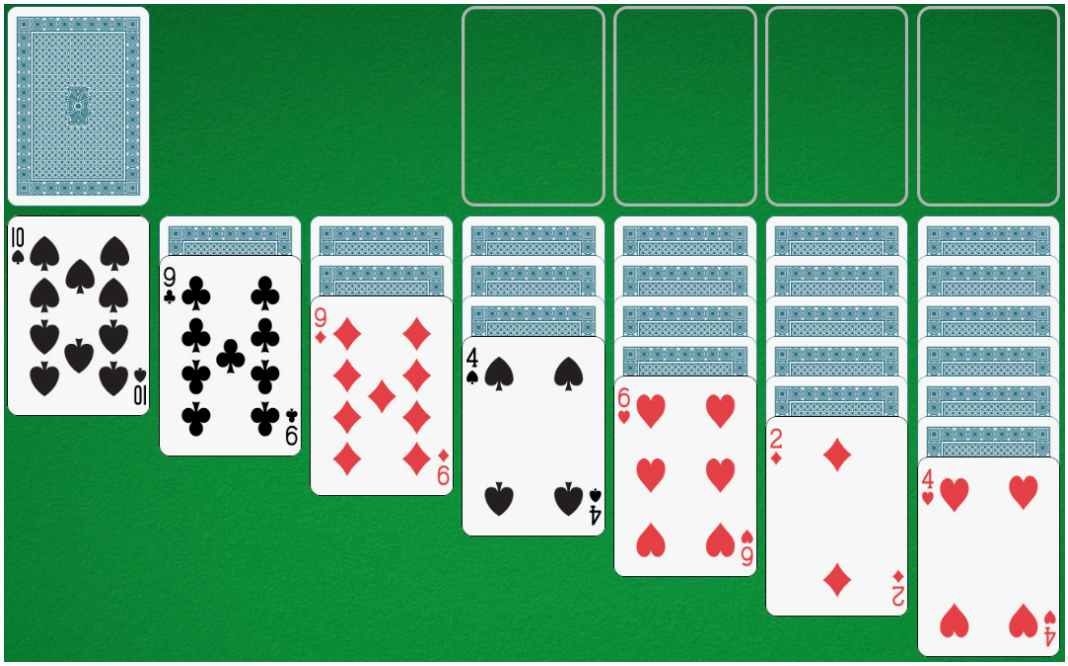


Ilustración 3 - Juego "El solitario" para PC

Haciendo uso de esa última afirmación, en este proyecto se pretende extrapolar el juego de mesa nombrado al principio del proyecto, a un entorno digital en un dispositivo móvil y que pueda ser usada de manera más sencilla, agradable y fácil por parte de los integrantes del juego.

Al no disponer en todo momento de una baraja de cartas ya sea en un viaje, en una reunión familiar, de amigos, universidad, colegio, etc… gracias a esta aplicación dispondríamos del juego mencionado en todo momento con una simple instalación en Android de su instalador.

Las ventajas de disponer el juego sobre la versión de mesa, radican en una mayor portabilidad, conteo de los usuarios, interacción tecnología/juego de mesa clásico (aun siendo de invención propia), disponer de las reglas en todo momento para poder usarlo de manera clara y correcta.

1.4 Requisitos iniciales

Como requisitos iniciales:

- Se debe tener en claro cuáles son las reglas del juego original de mesa, para así, poder jugar de forma efectiva en el entorno móvil (APP).
- Hacer un estudio de cómo se quiere desarrollar, teniendo en cuenta tanto al desarrollador del juego como a los jugadores que harán uso del mismo.
- Los requisitos técnicos por parte del alumno deben ser la capacidad de poder desarrollar de manera efectiva el proyecto indicado, en este caso: conocimiento de lenguajes, uso de las plataformas, capacidad de resolución de problemas durante el proceso.
- Conocimiento sobre las tecnologías estudiadas y seleccionadas para realización del proyecto, haciendo un estudio previo de los métodos que se usarán para su realización a nivel técnico sabiendo que el usuario final puede ser el creador del proyecto así como usuarios externos sin conocimiento alguno de él (serán la mayoría de usuarios finales)
- En cuanto a requisitos materiales, se debe disponer de un dispositivo Android, (Smartphone, Tablet, o emulador de Android) donde poder instalar el “apk” correspondiente a la app. Como sugerencia, el dispositivo debería ejecutar el juego con una solvencia mínima; haciendo uso de las versiones Android 5.0 a Android 12 (en caso de hacer uso de esta plataforma)

Como objetivo final, se estudia la posibilidad de disponer del proyecto en las plataformas principales de descarga de aplicaciones dependiendo de la plataforma para la que se desarrolla

1.5 Alcance

El alcance del proyecto viene plasmado en los siguientes puntos:

1. Entregable de archivo PDF con la documentación asociada al proyecto
    a. Contenido explicativo completo del proyecto, memoria del proyecto. Ordenada a través de un índice para cada uno de los puntos que se desarrollan en relación al proyecto
2. Entregable del proyecto ejecutable con extensión ".apk" creado a través de la plataforma de desarrollo "Android Studio"
3. Manual de juego/instrucciones básico en formato ".pdf" de las reglas del juego y funcionamiento del mismo dentro de la aplicación.
4. Archivo de los "storyboards", ejecutado en el programa Draw.io realizados en el primer análisis del proyecto y que han servido como base para el comienzo del estudio visual del proyecto, para su desarrollo y final implementación en el cliente "Android Studio"
5. Carpeta en formato ".zip" que contiene algunas páginas web que han servido de estudio y ayuda para creación y resolución del proyecto. En el inicio del estudio, en el desarrollo y en la implementación del proyecto.
6. Habilitar un link de descarga privado para realizar la descarga directa a través de "Google Drive"

1.6 Hipótesis y restricciones

///****El TFT se define como una asignatura de 12 créditos, lo que supone que la duración total del proyecto será de 300 horas, incluyendo todas las etapas del ciclo de vida, con la excepción del mantenimiento.

La principal restricción aplicable es la limitación de la duración del trabajo y el desarrollo del mismo realizado por una sola persona.

Teniendo en cuenta la limitación de tiempo y recursos (humano), se desea conseguir el desarrollo y funcionamiento completo de los objetivos principales, así como la finalización del proyecto. Pudiendo tener un mantenimiento posterior.

Una vez planteado previamente la limitación por tiempo al definirse como una asignatura de 12 créditos, se empieza a hacer un estudio inicial de todas las posibilidades para la realización del proyecto, teniendo en cuenta varios puntos propuestos:

- Tipo de juego a realizar y plataforma final de uso
- Al elegir una sola plataforma en la que se desarrollará, en este caso particular dispositivos con Android, se debe limitar el trabajo al estudio, desarrollo e implementación del mismo para un resultado efectivo en esa plataforma dejando así descartado otras plataformas.
- Limitación de llegar a usuarios que solamente usen la plataforma Android, dejando descartados a usuarios que usen por ejemplo iOs
- Se plantea el posible tiempo necesario para el desarrollo en la plataforma Android, haciendo búsqueda de los clientes (programas) principales para la implementación del proyecto. Teniendo en consideración Android Studio, NetBeans o Eclipse como entorno de trabajo.
- Se decide la tecnología a usar para el desarrollo, como alternativas se dispone de JAVA o KOTLIN sobre Android.
- Se plantean las restricciones sobre dispositivos antiguos los cuales ejecutan una versión de Android desfasada y que puede acarrear problemas de ejecución del proyecto.
- El proyecto debe ajustarse a unos estándares de la plataforma, decidiendo su ejecución como un formato vertical o un formato horizontal.
- Estudio de estructuración dentro del proyecto, se buscan alternativas para un boceto inicial. Como alternativas: “draw.io”, “Dibujos de google” (Google Drawings), entre otros.

- Implementación del diseño, estudiando la viabilidad de tener correlación entre el desarrollo de la funcionalidad (tecnologías de desarrollo mencionada anteriormente, JAVA/KOTLIN) y el desarrollo de diseño.

1.7 Estudio de alternativas, viabilidad y elección

Como primer punto, se estudió cual sería la plataforma donde mejor podría ejecutarse el proyecto: La decisión fue sobre la primera opción pensada, debido a: realización de pruebas, realizar el desarrollo y ejecutar la versión final del mismo; se decidió así por el Sistema Operativo **Android**.

1.7.1 Plataforma

Para poder diseñar y desarrollar una aplicación, primero debemos elegir cual será la plataforma para la cual nuestra aplicación será compatible y pueda ejecutarse.

A día de hoy, los dos sistemas operativos más usados en el mundo son Android (de código abierto**)** e iOs.

Durante unos años también estuvo comercializándose el sistema operativo Windows Phone, que actualmente está totalmente descontinuado *(desde 2017 para Windows Phone y 2019 para Windows Phone 8.1)* y no es usado en ningún dispositivo que esté actualmente en el mercado.

Una vez hayamos planteado las posibilidades de elección, debemos conocer que Android es el sistema operativo más usado, y al ser de código abierto y comerciable por parte de Google *(propietaria de Android)*, diferentes marcas como **Samsung**, la propia **Google con los Pixel**, **Xiaomi**, **Huawei**, **Motorola** (entre otros), hacen uso de este S.O para darle “vida” a su hardware, ya sea en Smartphone o Tablets.

Por otro lado, encontramos iOs, el cuál tan solo podemos encontrarlo bajo la marca Apple, en sus dispositivos iPhone o iPads, sin posibilidad de encontrarlo en ninguna otra marca de hardware.

Por esa razón, el sistema operativo elegido en el que se basa el proyecto será Android, con la intención de llegar a más usuarios y la compatibilidad con diferentes dispositivos hardware que usan este S.O.

Ilustración 4 - Logo Android

De manea resumida, la elección viene dada por l

- Android, Sistema Operativo (S.O) más usado en dispositivos móviles (Smartphones, Tablets…), se descarta otras plataformas como iOs, por desarrollo, implementación y compatibilidades.

1.7.2 Android Studio

Para la implementación principal del proyecto, en la parte funcional (JAVA) como en la parte de diseño (XML) , se estudian varias alternativas de programas IDE, como por ejemplo: Android Studio, NetBeans, Eclipse…

Finalmente, se decide como alternativa el cliente Android Studio:

Android Studio es el entorno de desarrollo integrado (IDE) oficial que se usa en el desarrollo de apps para Android. Basado en el potente editor de código y las herramientas para desarrolladores de IntelliJ IDEA

Android Studio es: *https://developer*.android.com/studio/intro?hl=es-419

Para este proyecto la versión instalada que hemos utilizado ha sido la versión **4.1.2**

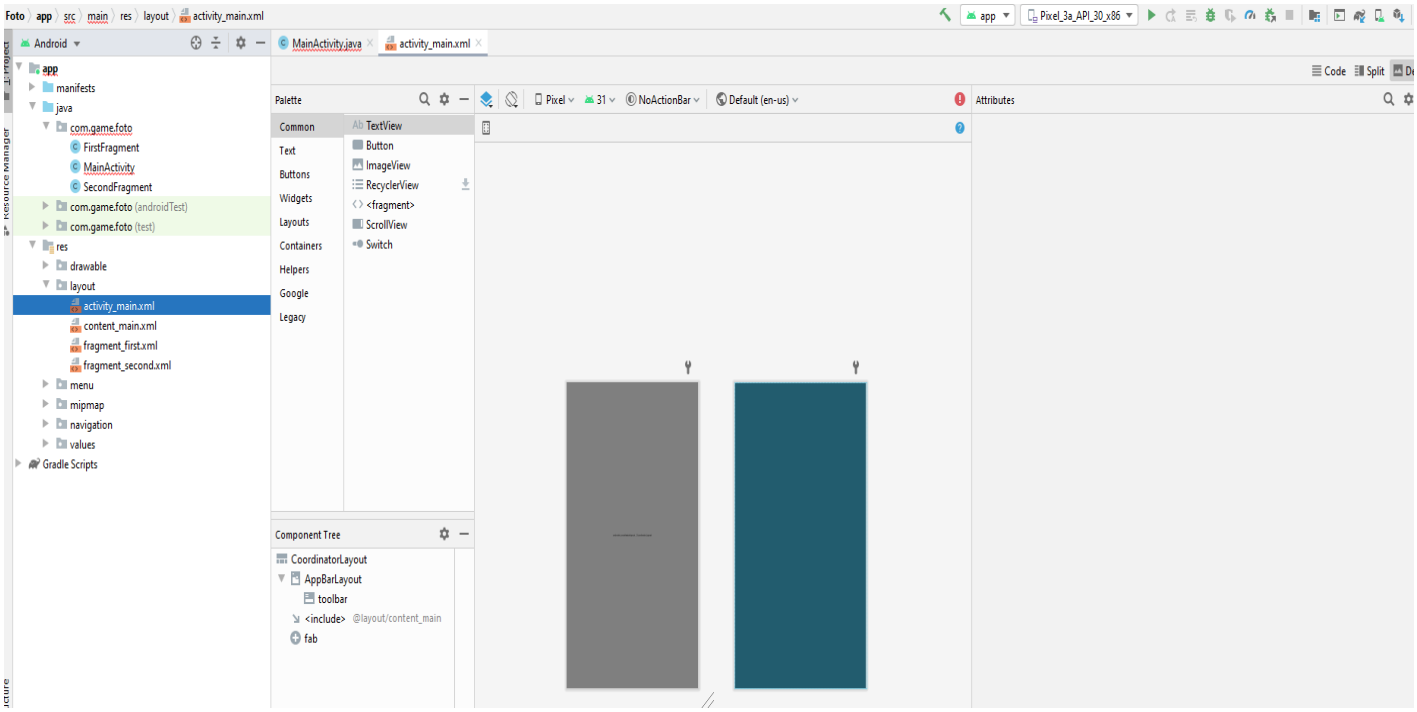


Ilustración 8 - Android Studio

Se decide por la tecnología JAVA por tener mayor conocimiento sobre ella respecto a la tecnología de KOTLIN. Pudiendo ampliar y mejorar el conocimiento sobre ella (JAVA):

- Se elige la versión JAVA 1.8 (JAVA 8) para mayor compatibilidad con Android, siendo así la versión más usada para el desarrollo de aplicaciones.

1.7.3 draw.io

Draw.io es un software utilizado para diseñar diagramas de forma gratuita y offline, aunque también tiene una versión completamente funcional en el navegador web, y además, facilita la integración con múltiples plataformas y programas .

Para el boceto de diseño se elige esta herramienta “draw.io”, por tener un conocimiento más amplio sobre ella, siendo similar a otras alternativas como “Google Drawings”

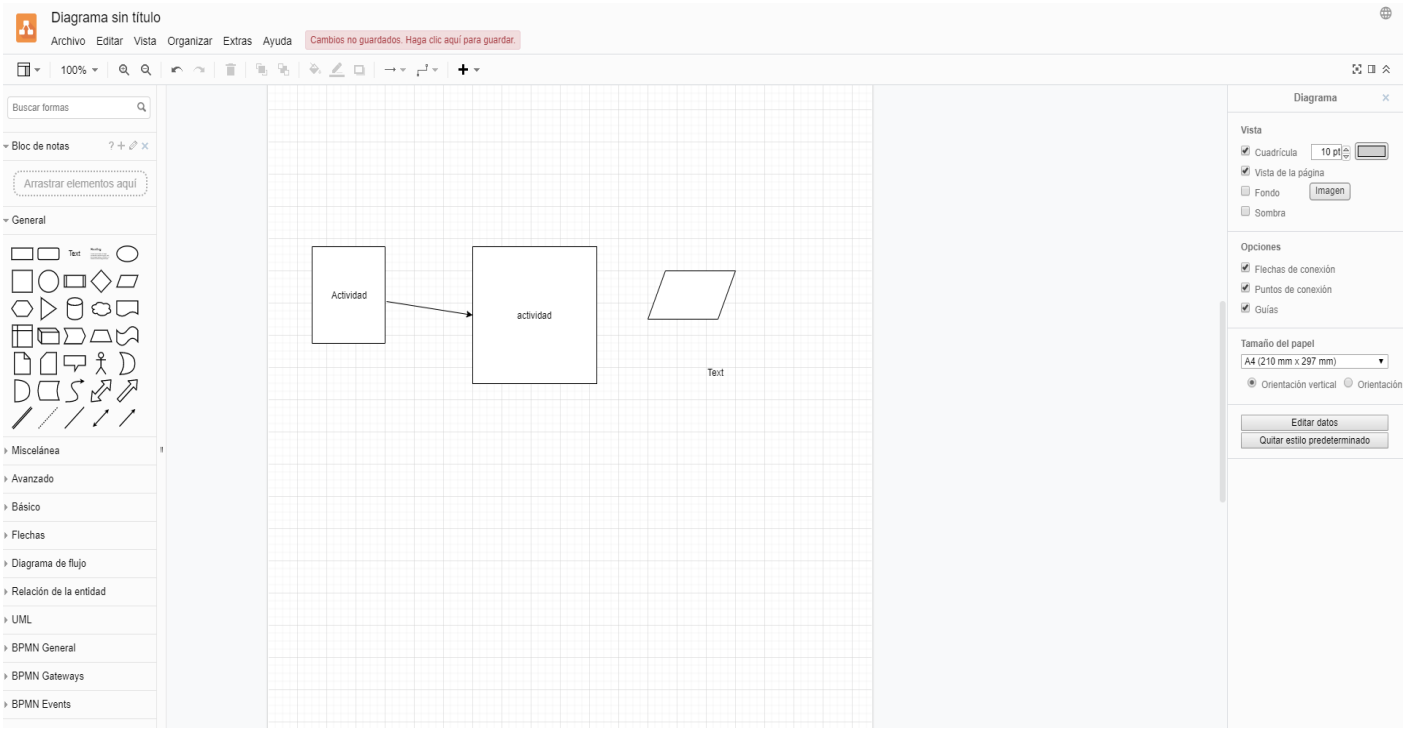


Ilustración 9 - Programa draw.io

1.7.4 Estadística de uso de las versiones Android

Después del estudio realizado sobre las distintas versiones disponibles y en uso de Android en la actualidad, a continuación, se detallan las versiones y fragmentación del S.O y el dispositivo final donde podrá ejecutarse el proyecto:

Dispositivo Android actual (posterior a 2017), en el que se esté ejecutando una versión de Android actualizada o como mínimo la versión más usada por los usuarios.

- Fragmentación de Android en agosto de 2022:

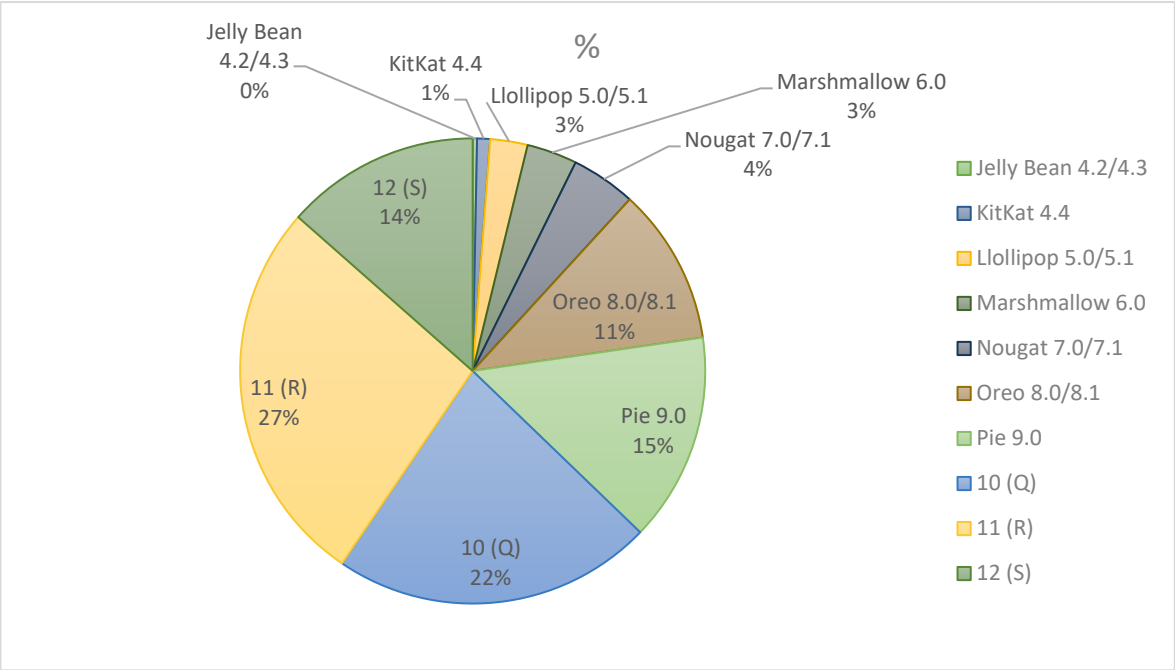


Ilustración 10 - Fragmentación Android agosto 2022

1.7.5 Windows 10

La plataforma para el desarrollo sería Windows 10, al disponer más compatibilidades ambos sistemas operativos (Android/Windows)

Gracias a la elección de Windows como sistema de instalación del software "Android Studio", disponemos también facilidad para la conexión de dispositivos externos Android para para la instalación de drivers.

El uso de las alternativas anteriores descritas como Android Studio y draw.io son completamente compatibles con Windows 10.

1.8 Tecnologías utilizadas

Se han usado las siguientes tecnologías:

- Software para la implementación, creación y ejecución: Android Studio
    - Versión Android Studio 4.1.2
    - Gradle Versión 6.5
    - Versiones de desarrollo:
        - Versión de compilación: compileSdkVersion: 31 (API Android 31)
        -
        - Versión mínima compatible: Android 5.0 (Lollipop) (minSdkVersion: 21)
        - Versión Android de ejecución óptima: Android 12 (targetSdkVersion: 31)
- Lenguaje de programación: JAVA asociado a lenguaje XML para creación de la interfaz de la app.
    - Versión 1.8 (JAVA 8)

        - Versión más utilizada para desarrollo de aplicaciones en el Sistema Operativo Android

- Librerías/dependencias:
    - implementation 'androidx.appcompat:appcompat:1.4.1'
    - implementation 'com.google.android.material:material:1.6.0'
    - implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    - testImplementation 'junit:junit:4.+'
    - androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    - androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
    - implementation "androidx.recyclerview:recyclerview:1.2.1"
    - // For control over item selection of both touch and mouse driven selection
    - implementation 'androidx.recyclerview:recyclerview-selection:1.1.0'
    - implementation 'com.github.bumptech.glide:glide:4.13.0'
    - annotationProcessor 'com.github.bumptech.glide:compiler:4.13.0'
- Herramienta draw.io, instalada en Windows 10 para diseño inicial de “Storyboards”

El material usado para la creación del proyecto es el siguente:

- Ordenador con Windows 10, con la Herramienta (cliente) “Android Studio” y draw.io.
    - Versión 12.4.2
- Smartphone con S.O. Android para poder ejecutar el proyecto de manera rápida y cómoda, buscando la ejecución en un dispositivo real, prescindiendo de la ejecución virtual del cliente “Android Studio”
    - Google Pixel 4a,
        - Android 12 (actualizado a Android 13 durante el desarrollo)

1.9 Metodología de desarrollo de software

Para el desarrollo de la aplicación, la metodología de desarrollo software escogido ha sido un hibrido entre “Agil” (Agile) **SCRUM** y “Desarrollo guiado por pruebas” (Test Driven Development (TDD)).

Se han elegido estos métodos debido a que al ser una aplicación para dispositivos Android, se debe tener un seguimiento de las tareas que se van realizando, en este proyecto en concreto la funcionalidad de cada pantalla (actividad) de la aplicación

Priorizando principalmente el funcionamiento correcto de la aplicación, testeando en reuniones (en este caso en pequeñas porciones de tiempo que el desarrollador destine para ello, ya que no es un proyecto en grupo) el proceso que se ha llevado a cabo durante el desarrollo de una parte del proyecto

Se deben tener finalizadas entregas del proyecto lo antes posible en plazo priorizando lo anteriormente mencionado, la funcionalidad, por ejemplo:

En una de las “Activity”, se debe realizar una tirada para seleccionar de manera aleatoria al jugador que primero elige su carta; para ello debemos tener disponible en la “Activity” anterior, todos los jugadores que van a realizar la partida. Si el resultado no es el esperado, modificar con otra tarea asignada el funcionamiento.

1.9.1 Scrum

Scrum es un marco de gestión de proyectos de metodología ágil que ayuda a los equipos a estructurar y gestionar el trabajo mediante un conjunto de valores, principios y prácticas.

1. **Roles:**

- **Product Owner o responsable del producto:**

Persona a cargo de la lista de trabajo pendiente del producto (product backlog). Principal autoridad del proyecto. Define objetivos y requisitos del proyecto. Está en contacto directo con las necesidades del usuario y transmitir la información a su equipo.

- **Scrum Master:**

  Persona que dirige los diferentes eventos de Scrum. Considerado el gerente del proyecto. Promueve las reuniones para estar actualizado sobre el desarrollo del proyecto y planificar en esas reuniones planes revisiones y análisis retrospectivo del Sprint.

- **Equipo Scrum:**

  Integrantes del equipo que trabajan sobre el proyecto y se auto-organizan para lograr el objetivo.

2. Etapas del SCRUM

En SCRUM se trabaja con Sprints, es decir, el proyecto se divide en pequeñas partes para poder abordarlas de forma más rápida y eficiente. Un proyecto puede estar compuesto por varios sprints que cuando se concluyen dan el resultado esperado.

1. **Sprint**: Es la primera fase del SCRUM, dónde se describe las tareas que se van a asignar a cada persona del equipo y el tiempo para esa tarea.
2. **Scrum team meeting**: Son reunions habituales para revisar y evaluar el trabajo realizado sobre las tareas asignadas y los posibles inconvenientes y soluciones que se darán a futuro.
3. **Backlog**: Repaso de las tareas y su evolución por parte del Product Owner. Con el objetivo de evaluar las tareas y resolver los inconvenientes que hayan surgido en ellas

4. **Sprint review**: Reuiones para en el fin de la iteración para evaluar los objetivos obtenidos y revisar que debe ser modificado.
5. **Retrospective**: Reunion final tras la finalización del sprint, donde se entrega el proyecto y se evalúan los fallos y dificultades encontradas durante el sprint propuesto para mejorarlas en futuros sprint.

1.9.2 Iteraciones del proyecto:

El proyecto se ha dividido en 8 iteraciones las cuales cada una tiene relación entre sí para entender el correcto funcionamiento de la aplicación. Planteándose que si en una de las iteraciones posteriores es necesario que se modifique o mejore una de las anteriores, no haya conflicto en realizar esa modificación.

1. Planteamiento inicial de la aplicación y comienzo de la documentación.
2. Desarrollo de las primeras actividades (Inicio e Introducción de jugadores)
3. Desarrollo de las actividades intermedias (Tirada aleatoria para primer jugador y selección de carta)
4. Desarrollo de las actividades intermedias (tira aleatoria pregunta jugador, pregunta, eleccion carta resto jugadores)
5. Desarrollo de la actividad final (muestra final del contenido de la partida y comienzo de una nueva partida)
6. Pruebas finales, realización y finalización de la documentación

1.10 Estimación del tamaño y esfuerzo

Ya que el presente proyecto es un TFT, no existen restricciones de tipo económico, sino de tipo temporal (un número aproximado de horas). Por consiguiente, los cálculos de tamaño del proyecto están supeditados el tiempo disponible. En cuanto al esfuerzo, se dispone de tan un solo efectivo (la persona autora del trabajo).

1.10.1 Historias de usuario

Las historias de usuario son las unidades de trabajo más pequeñas en los marcos ágiles.

Una historia de usuario se define como una explicación general e informal de una función de software. El propósito de una historia de usuario es articular y cómo un elemento de trabajo entregará un valor al usuario final. Ya sea de manera externa a un cliente o simplemente a usuarios dentro de un entorno conocido.

Las historias de usuario se añaden en los sprint de nuestra metodología SCRUM.

Las historias de usuario suelen expresarse con una frase simple con la siguiente estructura:

“Como [perfil], [quiero] [para].”

- “Como (perfil), ¿para quién desarrollamos el proyeco?
- “Quiere”: ¿Qué se está intentado lograr?
- “Para”: Motivo por el que se realiza

Se usarán puntos de historia que asignarán un valor relevante a las historias de usuario para el equipo de trabajo que está realizando el proyecto a través de SCRUM.

Los puntos de historia se definen como una unidad de medida utilizada principalmente en la gestión de proyectos ágiles de la metodología Scrum. Se utilizan

para estimar la carga de trabajo global de los equipos, con el fin de planificar cada sprint o iteración.

Los valores vendrán dados por: ***0, 0.5, 1, 2, 3, 5, 8, 13, 20, 40, 100*** ; para facilitar su uso.

Historias de usuario en el apartado

1.11 Diagrama de Gantt

La planificación temporal podemos estructurarla basándonos en las iteraciones que se han realizado para así poder ver la programación que se ha seguido en todo el proyecto y el tiempo empleado para cada iteración.

Se representará a través del Diagrama de Gantt:

El diagrama de Gantt, muy usado en la gestión de proyectos, es un gráfico de barras horizontales que se usa para ilustrar el cronograma de un proyecto, programa o trabajo. Es una forma de visualizar la programación de un proyecto, de dar seguimiento a los logros y de estar siempre familiarizado con el cronograma del trabajo. Cada barra de un diagrama de Gantt representa una etapa del proceso (o una tarea del proyecto) y su longitud, la duración de la tarea.

Ilustración 11 - Diagrama de Gantt

2 DISEÑO INICIAL

El diseño inicial del proyecto viene dado por el desarrollo de un juego de mesa real hacia una aplicación para dispositivos móviles Android.

Se debe tener en cuenta que el principal objetivo del proyecto es plasmar el juego real en la aplicación, teniendo en cuenta las posibles casuísticas que pueden darse en el desarrollo del diseño como del funcionamiento interno del proyecto.

El diseño de una aplicación desde el punto de vista del usuario final, debe de ser fácil de usar, intuitiva, poco cargada estéticamente para aplicaciones con juegos “planos” como en este caso y amigable en el diseño.

Este diseño debe de “fusionarse” con la funcionalidad del propio juego.

Se debe de tener en cuenta que, dependiendo del juego, puede ser usado por personas tanto de edades tempranas (siempre con la autorización previa por parte de padres o tutores) así como de personas en edad más avanzada que quieran usar esta aplicación, lo cual lleva todo a la “sencillez y facilidad”.

2.1 Especificaciones del sistema

2.1.1 Historias de usuario

Las especificaciones del sistema vendrán dadas por las siguientes historias de usuario:

HISTORIAS DE USUARIO	PUNTOS DE HISTORIA
Abrir aplicación	**0.5**
Interfaz de usuario por pantalla	**3**
Introducir nuevo jugador	**2**
Eliminar jugador	**2**
Siguiente pantalla	**0.5**
Jugador aleatorio	**2**
Escoger carta primer jugador	**3**
Escoger carta resto de jugadores	**4**
Interfaz de usuario selección de carta	**4**
Puntuación de jugadores	**3**
Nueva partida	**0.5**

En estas historias de usuario que plasman las principales funciones que se deben seguir para el futuro desarrollo de la aplicación, vendrás incluidas implícitamente dentro de las iteraciones que veremos más adelante.

Con los puntos de historia de cada historia de usuario, se puede calcular un promedio de cuantos puntos de historia serán necesarios por iteraciones.

Se debe dividir el total de los puntos de historia sobre el total de iteraciones del proyecto:

24.5 puntos de historia / 6 iteraciones = 4.08 ≈: 4 puntos de historia/iteración

2.2 Análisis y diseño del sistema

Se analiza las características del juego de mesa original, en el cual nos encontramos con 2 cartas que serán los únicos ítems físicos que podemos rescatar desde un inicio para el diseño, así como las personas implicadas en el juego.

2.2.1 Boceto

Los bocetos en la mayoría de casos de desarrollos y diseños de aplicaciones son necesarios para entender y tener una base de qué es lo que se quiere plasmar, desarrollar y mejorar del diseño de la aplicación.

2.2.1.1 Boceto general inicial

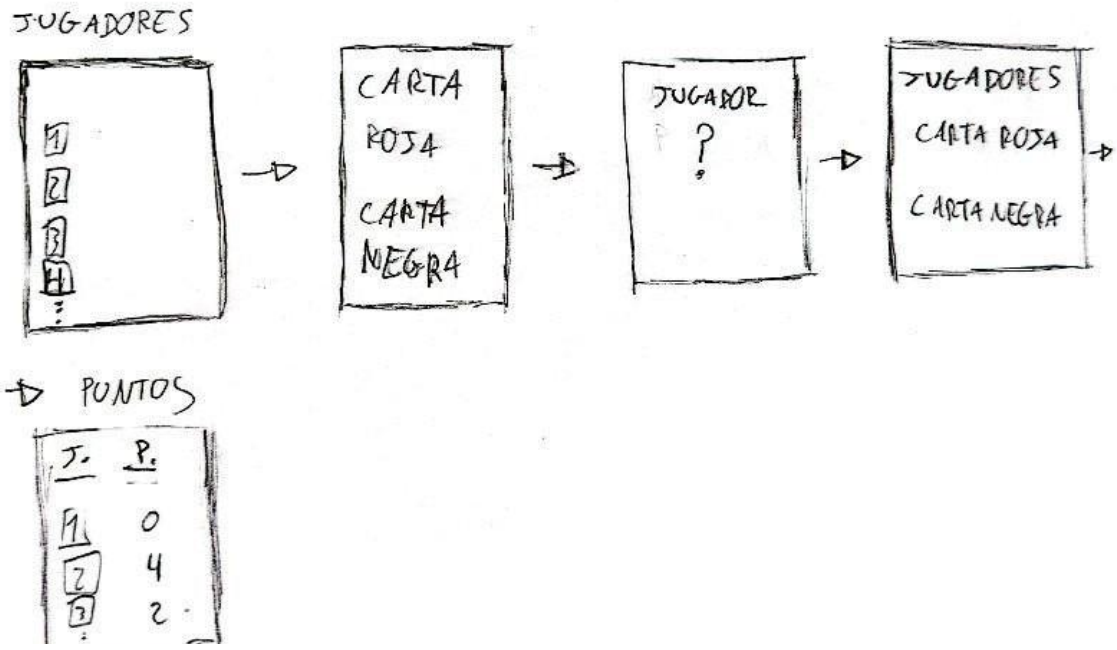


Ilustración 12 - Boceto inicial proyecto

Se plantean las pantallas que tendrá el juego de inicio. Se añadirán o eliminarán pantallas según avance el desarrollo.

2.2.1.2 Boceto pantalla de inicio

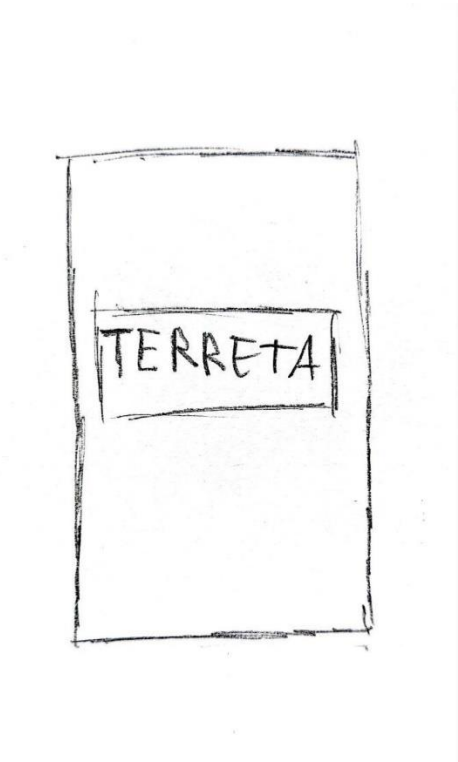


Ilustración 10 - Boceto pantalla de inicio

El boceto inicial del inicio viene dado por una imagen o texto del título del juego, en este caso el nombre “TERRETA” que corresponde al nombre del juego.

La intención de esta pantalla será que sea la primera del juego, a modo de inicio.

2.2.1.3 Boceto players

Ilustración 14 - Boceto Players

Este boceto representa los jugadores de la partida, puede haber un número N de jugadores, con distintos nombres, así como un botón para la siguiente pantalla

2.2.1.4 Boceto cartas

Ilustración 15 - Boceto cartas

Boceto de las cartas que debe elegir un jugador de los anteriores. Dispondrá de 2 cartas, roja y negra, donde cada una estará asociada a “verdad” o “mentira” respectivamente. Se debe revisar el manual de juego para comprender mejor el funcionamiento.

2.2.1.5 Boceto pregunta

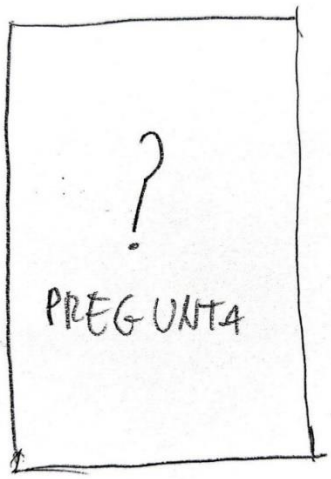


Ilustración 13 - Boceto pregunta

Uno de los jugadores anteriormente disponibles, realizará una pregunta al otro jugador

2.2.1.6 Boceto cartas 2

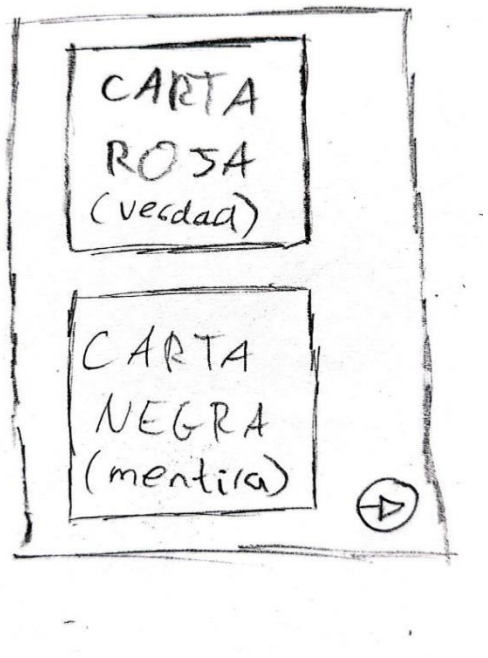


Ilustración 14 - Boceto cartas 2

Boceto de las cartas que debe elegir cada uno de los jugadores restantes. Dispondrán de 2 cartas, roja y negra, donde cada una estará asociada a “verdad” o “mentira” respectivamente. Se debe revisar el manual de juego para comprender mejor el funcionamiento.

2.2.1.7 Boceto puntuación

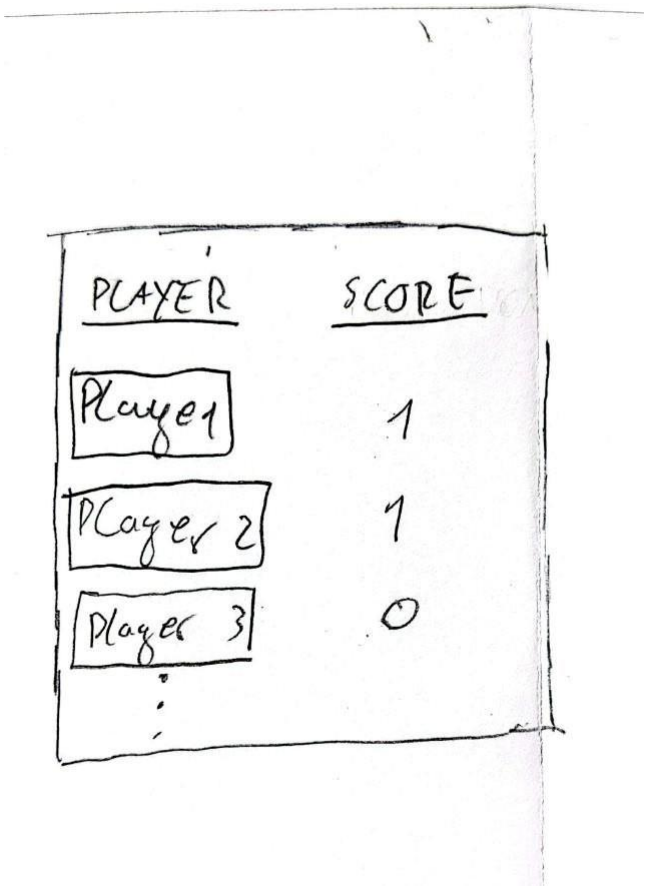


Ilustración 15 - Boceto Puntuación

Boceto de la pantalla de puntos una vez finalizado el juego, si el jugador X ha adivinado la carta del primer jugador, obtendrá puntos y se verán reflejados en pantalla

2.2.2 StoryBoards

A continuación, se muestra un diseño inicial presentados en “storyboards” a través del programa draw.io

Como se puede observar en los distintos storyboards, se ha querido plantear una interfaz sencilla y limpia donde el principal objetivo es el funcionamiento del juego.

Así, se estudió la manera de hacer la interfaz de cada evento (pantalla), muy legible para que, cualquier usuario medio/bajo pueda usarla. Descartando así cualquier “input” que distraiga al usuario del objetivo principal, que en este caso sería la jugabilidad.

Conociendo previamente los distintos casos que pueden darse dentro de las normas del juego de mesa original, se plantea cada storyboard enfocándose exclusivamente en la función de la misma, como, por ejemplo: En la escena de “Tirada Aleatoria”, el objetivo de esta escena radica en conocer un jugador aleatorio de los ya introducidos anteriormente.

Los casos de uso que se definen en el siguiente apartado están basados en el storyboard general que se muestra a continuación:

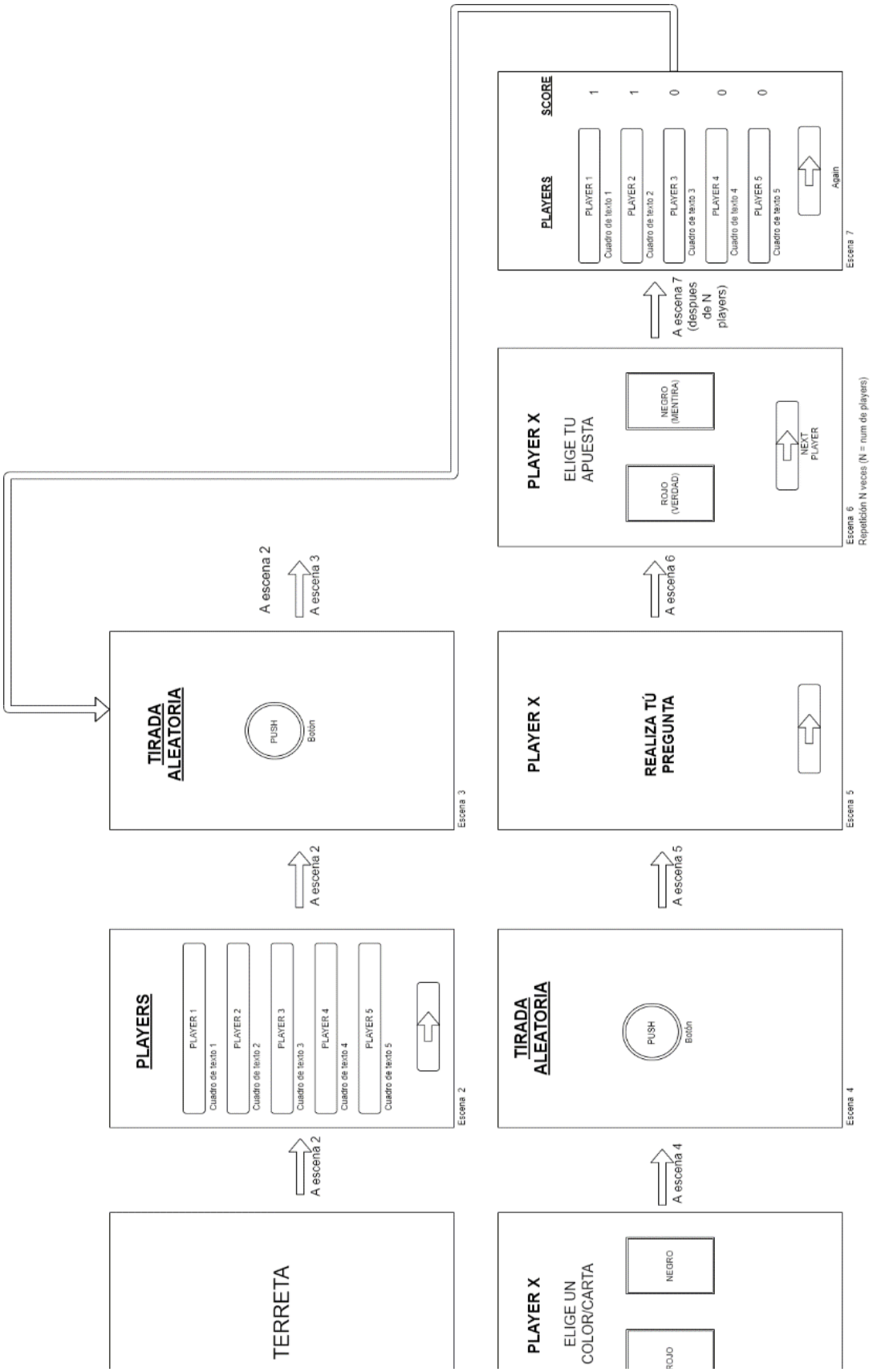


Ilustración 19 - StoryBoard general

2.2.3 Casos de uso

En este apartado se definirán los casos de uso que se han definido y que tendrán lugar en el proyecto. Se podrán añadir o eliminar según el avanzo de proyecto.

Los casos de uso serán realizados por todos los jugadores que intervengan en el juego.

Para cada caso de uso usaremos:

CU-XX	
Caso de uso	
Usuario	
Descripción	
Precondición	
Secuencia	

Tabla 1. Plantilla Casos de uso

- **Caso de uso**: Nombre del caso de uso que se puede realizar
- **Usuario**: Jugador que realiza la acción
- **Precondición**: Que condición previa tiene que darse para realizar el caso de uso
- **Secuencia**: Ejecución del caso de uso

Casos de uso:

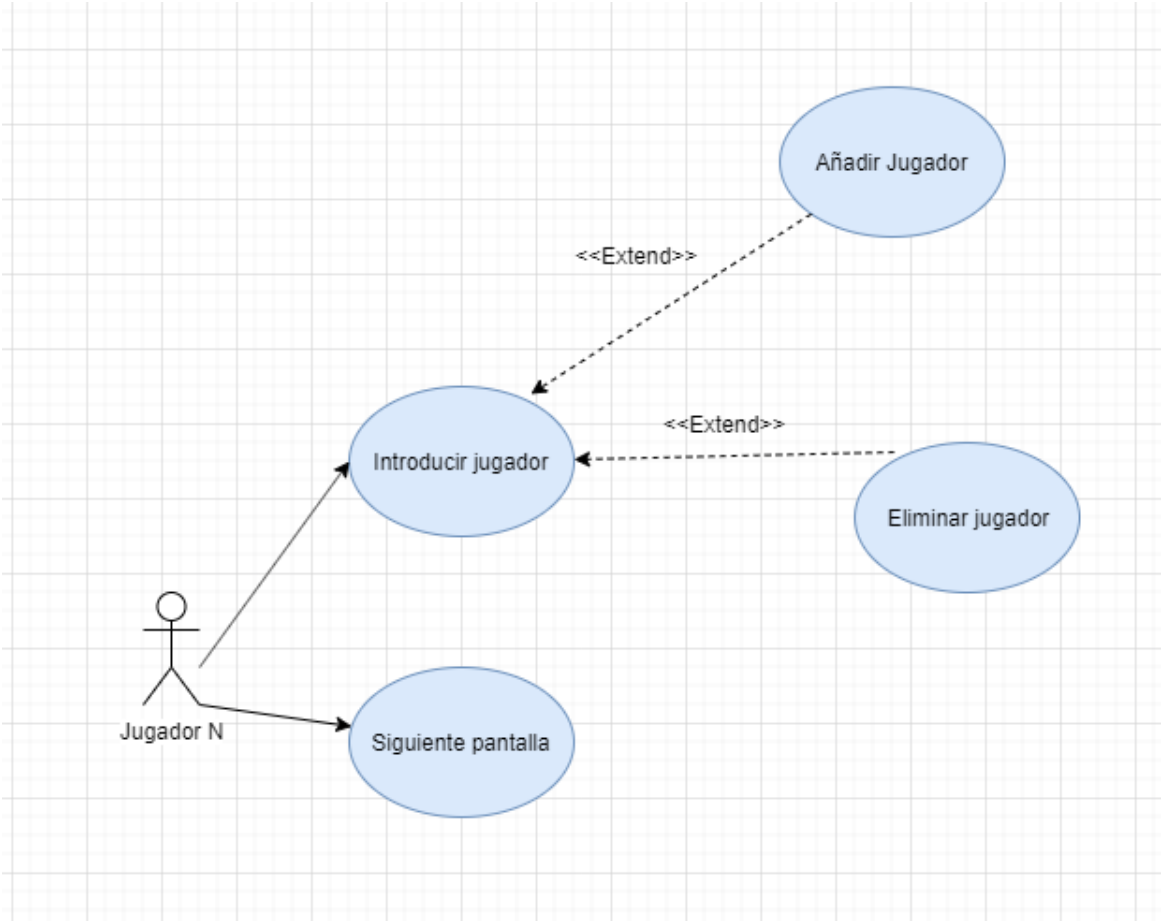


Ilustración 20 - Caso de uso pantalla de introducción de jugadores

CU-01	
Caso de uso	Introducir Jugador
Usuario	Jugador N (quien desee dentro de los posibles)
Descripción	Uno de los jugadores introduce los N jugadores que comenzarán la partida
Precondición	No hay
Secuencia	El jugador debe escribir en el cuadro de texto

Tabla 2. CU-01 - Introducir Jugador

CU-02	
Caso de uso	Añadir Jugador
Usuario	Jugador N (quien desee dentro de posibles)
Descripción	Se añade un jugador a la lista de jugadores
Precondición	Debe haber escrito un nombre de usuario en el cuadro de texto
Secuencia	El jugador pulsa en el botón de añadir jugador y la lista se actualiza con el nuevo jugador

Tabla 3. CU-02 - Añadir Jugador

CU-03	
Caso de uso	Eliminar Jugador
Usuario	Jugador N (quien desee dentro de posibles)
Descripción	Se elimina un jugador de la lista de jugadores
Precondición	Debe estar introducido previamente el jugador a eliminar
Secuencia	El jugador pulsa el botón de eliminar jugador y la lista se actualiza con un jugador menos (el eliminado)

Tabla 4. CU-03 - Eliminar Jugador

CU-04	
Caso de uso	Siguiente pantalla
Usuario	Jugador N que ha estado introduciendo los jugadores
Descripción	Se ejecuta la siguiente pantalla que sucede a la actual
Precondición	Debe de haber un mínimo de jugadores para que el juego pueda iniciarse
Secuencia	Jugador N pulsa el botón de “next” y la aplicación ejecuta la siguiente pantalla

Tabla 5. CU-04 - Siguiente Pantalla

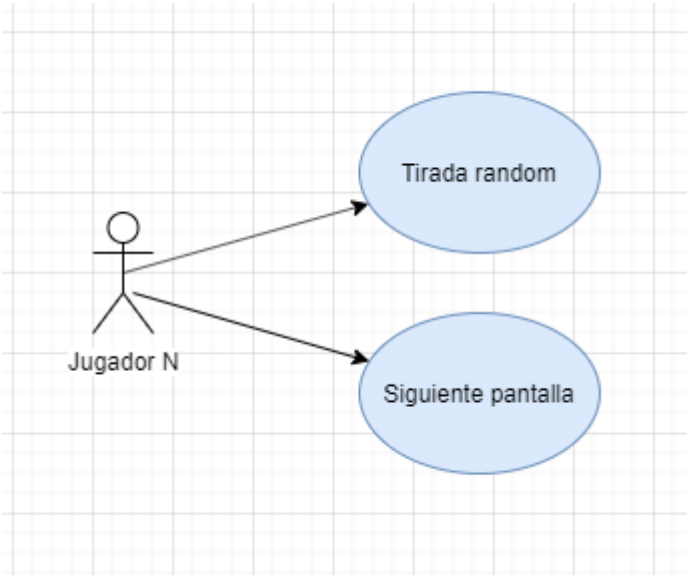


Ilustración 18 - Caso de uso pantalla de Tirada Aleatoria

CU-05	
Caso de uso	Tirada Aleatoria
Usuario	Jugador N que ha estado introduciendo los jugadores en la pantalla anterior
Descripción	Escoge de manera aleatoria un jugador de los introducidos anteriormente
Precondición	Haber jugadores para elegir
Secuencia	El jugador N pulsa el botón de “Tirada Aleatoria”, y aleatoriamente se muestra un jugador de los introducidos anteriormente

Tabla 6. CU-05 - Tirada Aleatoria

CU-06	
Caso de uso	Siguiente pantalla
Usuario	Jugador N que ha estado introduciendo los jugadores en la pantalla anterior
Descripción	Se ejecuta la siguiente pantalla que sucede a la actual

Precondición	Debe de haber ejecutado la tirada aleatoria
Secuencia	Jugador N pulsa el botón de “next” y la aplicación ejecuta la siguiente pantalla

Tabla 7. CU-06 - Siguiente Pantalla

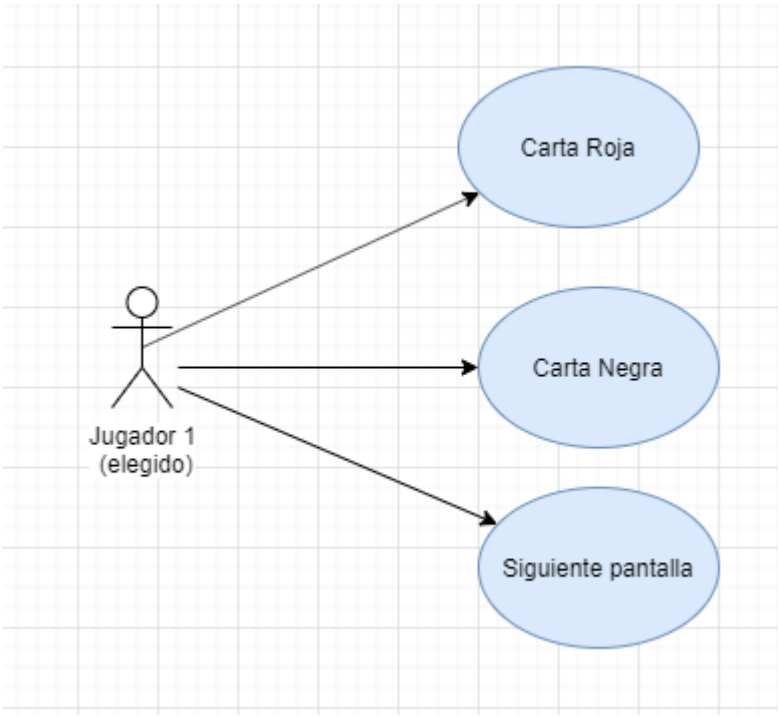


Ilustración 19 - Caso de uso pantalla Elección de carta

CU-07	
Caso de uso	Elección carta roja
Usuario	Jugador 1, elegido aleatoriamente
Descripción	Elección de carta roja, opción que elige jugador 1
Precondición	Haber sido elegido en la tirada aleatoria
Secuencia	Jugador 1 selecciona el botón de carta roja y se asigna a su elección

Tabla 8. CU-07 - Elección carta roja

CU-08	
Caso de uso	Elección carta negra
Usuario	Jugador 1, elegido aleatoriamente

Descripción	Elección de carta negra, opción que elige jugador 1
Precondición	Haber sido elegido en la tirada aleatoria
Secuencia	Jugador 1 selecciona el botón de carta negra y se asigna a su elección

Tabla 9. CU-08 - Elección carta negra

CU-09	
Caso de uso	Siguiente pantalla
Usuario	Jugador 1 que elige carta
Descripción	Se ejecuta la siguiente pantalla que sucede a la actual
Precondición	El jugador 1 debe haber elegido una de las cartas (roja o negra)
Secuencia	Jugador N pulsa el botón de “next” y la aplicación ejecuta la siguiente pantalla

Tabla 10. CU-09 - Siguiente pantalla

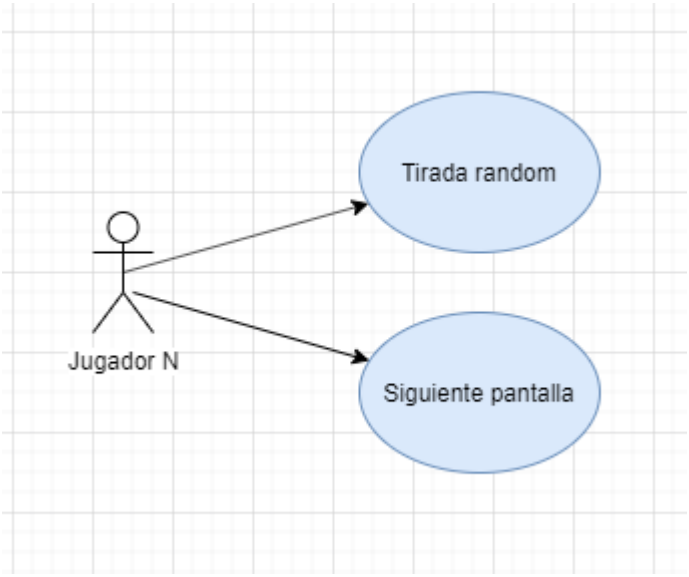


Ilustración 20 - Caso de uso pantalla de Tirada Aleatoria 2

CU-10	
Caso de uso	Tirada Aleatoria

Usuario	Jugador 1 que ha elegido carta anteriormente
Descripción	Escoge de manera aleatoria un jugador de los introducidos excluyendo al Jugador 1
Precondición	Haber jugadores para elegir
Secuencia	El jugador 1 pulsa el botón de “Tirada Aleatoria”, y aleatoriamente se muestra un jugador de los introducidos anteriormente

Tabla 11. CU-10 - Tirada Aleatoria

CU-11	
Caso de uso	Siguiente pantalla
Usuario	Jugador 1 que ha elegido carta anteriormente
Descripción	Se ejecuta la siguiente pantalla que sucede a la actual
Precondición	Debe de haber ejecutado la tirada aleatoria
Secuencia	Jugador N pulsa el botón de “next” y la aplicación ejecuta la siguiente pantalla

Tabla 12. CU-11 - Siguiente pantalla

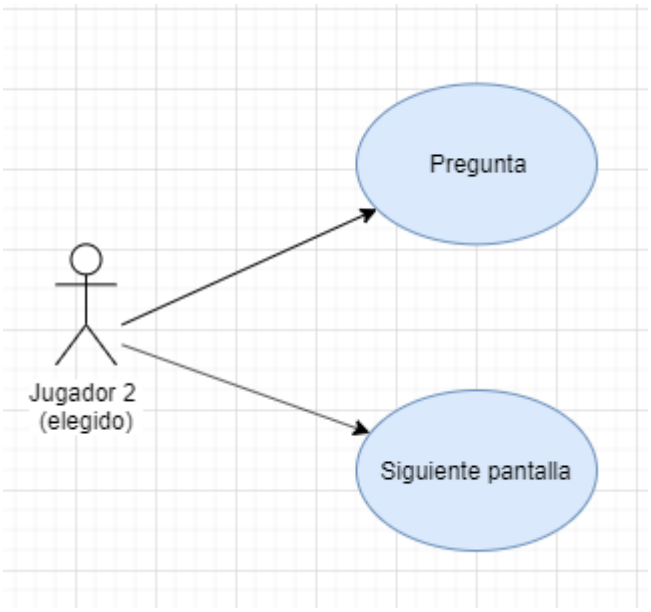


Ilustración 21 - Caso de uso pantalla de Pregunta

CU-12

Caso de uso	Realizar pregunta
Usuario	Jugador 2 que ha sido seleccionado de manera aleatoria
Descripción	Se realiza una pregunta aleatoria sobre el Jugador 1 elegido en la primera tirada
Precondición	Haber sido elegido para preguntar
Secuencia	Jugador 2 realiza pregunta en voz alta hacia el jugador 1

Tabla 13. CU-12 - Realizar pregunta

CU-13	
Caso de uso	Siguiente pantalla
Usuario	Jugador 2 que ha sido seleccionado de manera aleatoria
Descripción	Se ejecuta la siguiente pantalla que sucede a la actual
Precondición	Debe de haber realizado la pregunta
Secuencia	Jugador N pulsa el botón de “next” y la aplicación ejecuta la siguiente pantalla

Tabla 14. CU-13 - Siguiente pantalla

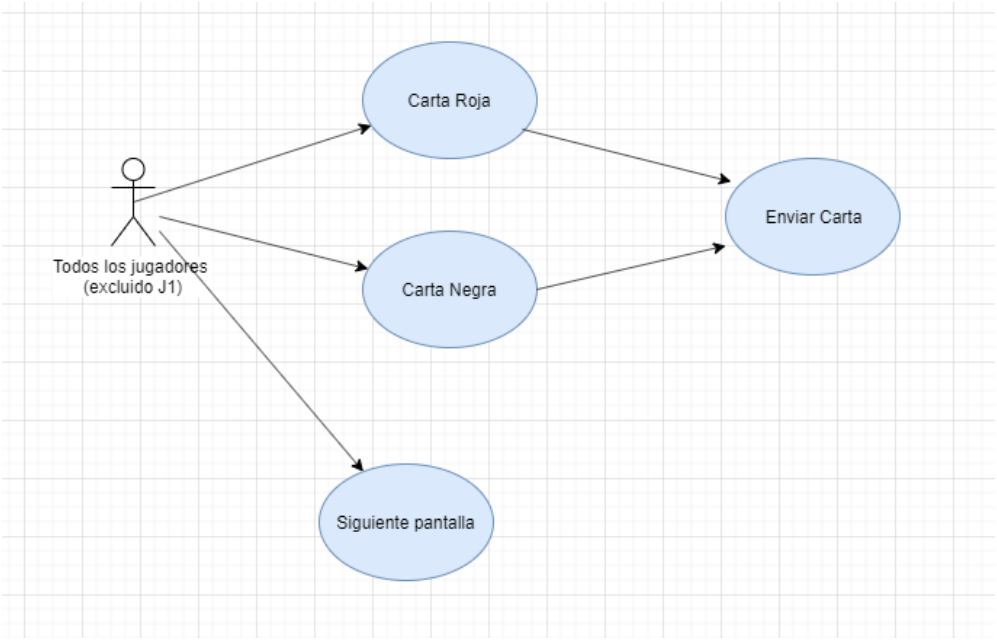


Ilustración 22 - Caso de uso elección de carta resto de jugadores

CU-14	
Caso de uso	Elección carta roja
Usuario	Jugador N, por orden de jugadores
Descripción	Elección de carta roja, opción que elige jugador N correspondiente
Precondición	No ser el jugador 1 elegido previamente en primera tirada aleatoria
Secuencia	Jugador N selecciona el botón de carta roja y se asigna a su elección, para averiguar la elección del Jugador 1

Tabla 15. CU-14 - Elección carta roja

CU-15	
Caso de uso	Elección carta negra
Usuario	Jugador N, por orden de jugadores
Descripción	Elección de carta roja, opción que elige jugador N correspondiente
Precondición	No ser el jugador 1 elegido previamente en primera tirada aleatoria
Secuencia	Jugador N selecciona el botón de carta roja y se asigna a su elección, para averiguar la elección del Jugador 1

Tabla 16. CU-15 - Elección carta negra

CU-16	
Caso de uso	Enviar carta
Usuario	Jugador N, por orden de jugadores
Descripción	Se envía la elección de la carta elegida
Precondición	Jugador N debe haber elegido una carta previa (roja o negra)
Secuencia	Jugador N pulsa el botón de “Enviar carta” y la aplicación ejecuta el envío y pasa al siguiente jugador N+1

Tabla 17. CU-16 - Enviar carta

CU-17	
Caso de uso	Siguiente pantalla
Usuario	Último jugador N en escoger carta
Descripción	Se ejecuta la siguiente pantalla que sucede a la actual
Precondición	Todos los jugadores deben haber elegido una carta previa (roja o negra)
Secuencia	Último Jugador N pulsa el botón de “next” y la aplicación ejecuta la siguiente pantalla de puntuación

Tabla 18. CU-17 - Siguiente pantalla

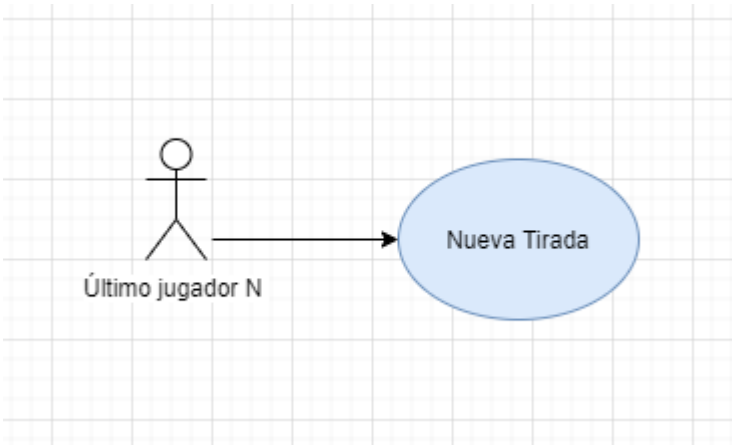


Ilustración 23 - Caso de uso nueva tirada

CU-18	
Caso de uso	Nueva partida
Usuario	Último jugador N en escoger carta
Descripción	Se ejecuta una nueva partida
Precondición	Haber completado una partida completa
Secuencia	Último Jugador N pulsa el botón de “Nueva Tirada”. Vuelve a pantalla de Tirada aleatoria y comienza una nueva partida

Tabla 19. CU-18 - Nueva partida

3 DESARROLLO

Para el desarrollo de la aplicación se seguirán como guía las iteraciones mencionadas anteriormente de manera explicativa

3.1 Primera Iteración

Como primera iteración, se realiza el planteamiento inicial y la elección de forma de desarrollo.

Al elegir el software “Android Studio”, se realizó una primera toma de contacto con el programa, realizando pequeñas pruebas tanto de java como de XML, hasta entender el funcionamiento principal del software.

Seguidamente, se intentan crear avances que veremos la primera “Activity”, probando el lenguaje JAVA y el funcionamiento entre las clases “.java” generadas junto con los layout (.xml) que se asocian a cada clase java:

Diseño:

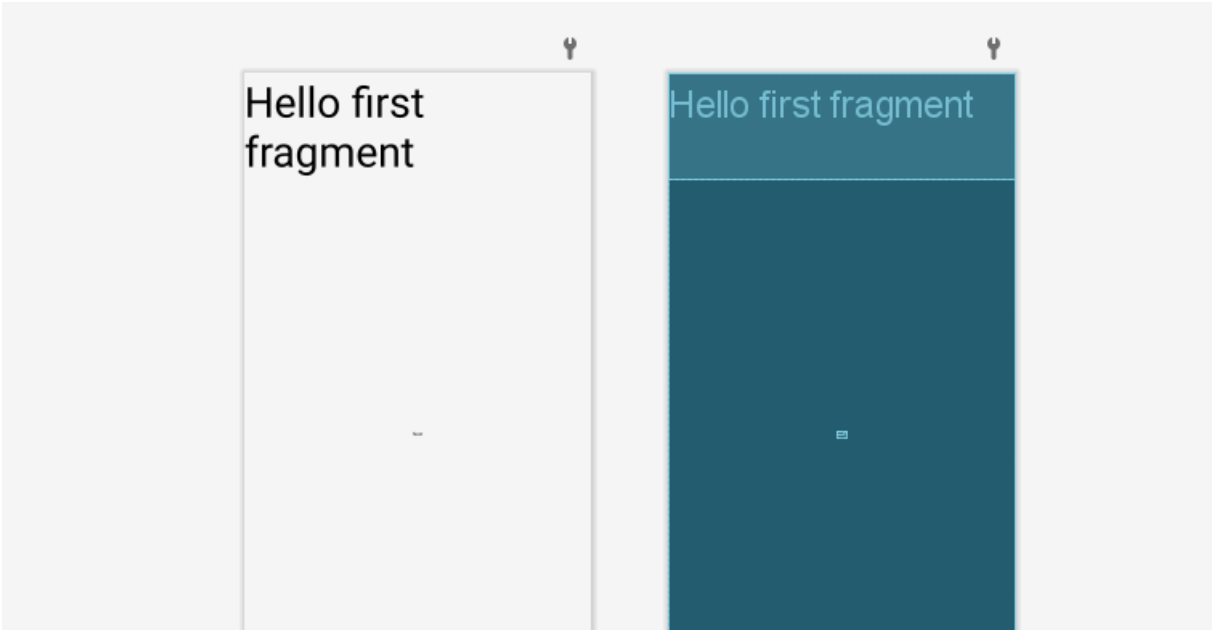


Ilustración 27 - First fragment (prueba)

Asociado al código XML:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".FirstFragment">

    <TextView
        android:id="@+id/textview_first"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello first fragment"
        android:textSize="50dp"
"
        app:layout_constraintBottom_toTopOf="@id/button_first"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Ilustración 28 - Codigo XML (prueba)

Se puede observar un color rojo asociado a la etiqueta TextView, es interesante mencionar este error ya que es un error que se debe al tener un layout de tipo *"constraintLayout*" en el que cada elemento debe estar "limitado" y no suelto por el Layout, ya que este tipo de Layout, es un tipo asociado a limitaciones, restricciones y condiciones de diseño, pero sin embargo, ayuda a tener un control sobre cada elemento dentro del Layout. Es el tipo que más usaremos en las siguientes iteraciones

En referencia al código .java:

```java
import ...

public class FirstFragment extends Fragment {

    @Override
    public View onCreateView(
            LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState
    ) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_first, container, attachToRoot: false);
    }

    public void onViewCreated(@NonNull View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);

        view.findViewById(R.id.button_first).setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                //instrucciones que se asociarian a un boton creado, en el método oncClick
            }
        });
    }
}
```

Ilustración 29 - Código java prueba

Observamos que en esta iteración lo principal es asociar el código java a la Activity que necesitamos, en este caso al Activity “fragment_first.xml”.

De esta manera, debemos de “inflar” (asociar) con un a variable creada de tipo LayoutInflater el java con el XML.

Con la línea de código:

return inflater.inflate(R.layout.fragment_first, container, false);

También Podemos observar el método findViewById(R.id….), el cual nos servirá en siguiente pasos para buscar y asociar cada elemento del diseño XML en caso de necesitar realizar alguna funcionalidad a través del código java.

Respecto al diseño general que se quiere obtener desde un principio, es una primera actividad en el que aparezca el logo de la aplicación, que sirva como bienvenida a la aplicación que veremos en la segunda iteración

3.2 Segunda Iteración

3.2.1 Actividad “MainActivity”

Para la primera actividad “MainActivity.java” asociada a “activity_main”, debemos destacar que esta actividad es principalmente visual para el inicio del juego, como hemos mencionado anteriormente, se pretende tener una imagen principal de inicio de juego, que se mostrará a través de una imagen:

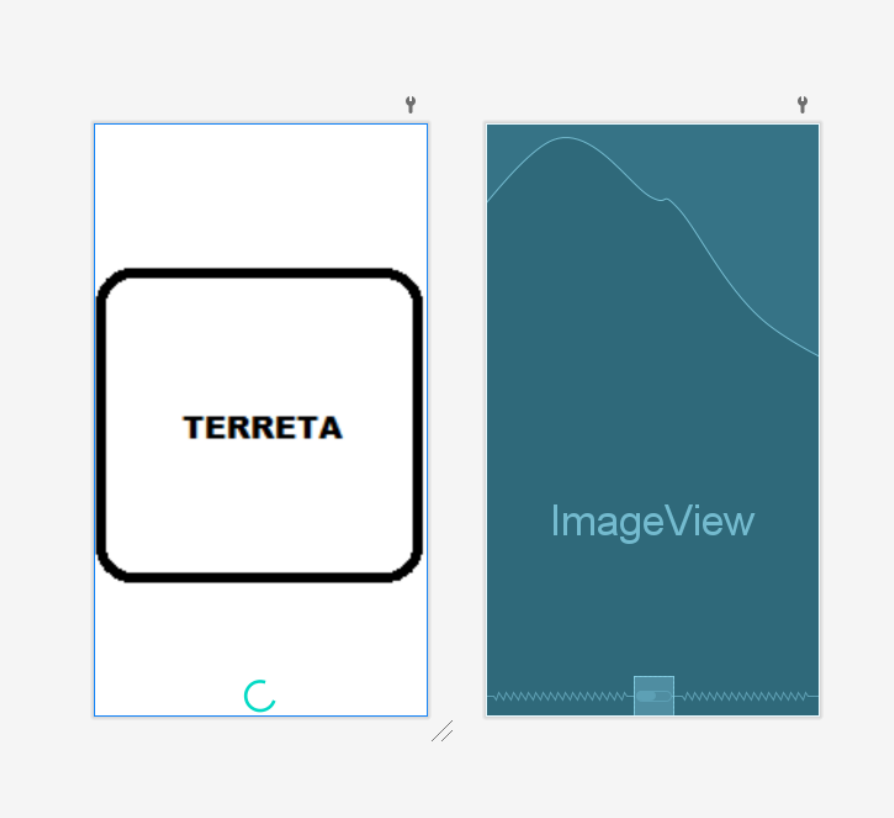


Ilustración 30 - Foto inicial de Inicio de Juego (pantalla de carga)

Ilustración 28 - Foto final de Inicio de Juego (pantalla de carga)

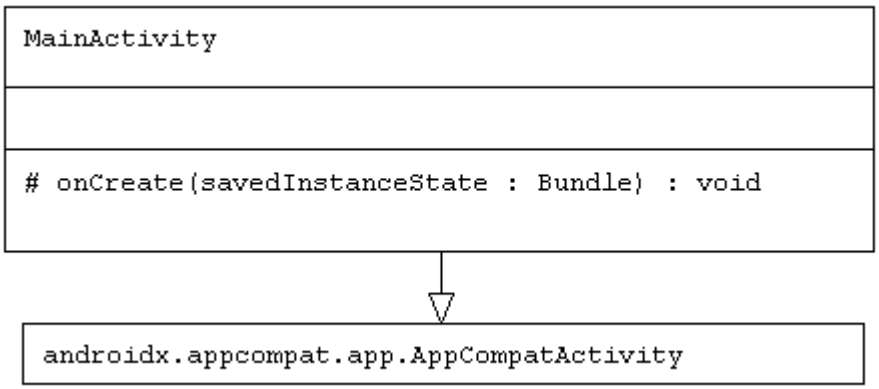
Para el diseño del logo, se han usado diferentes herramientas online y programas instalados en Windows (Canva.com para el diseño y escalado, Paint para recorte y fondo)

En el código java se destaca la variable *"intent"* de tipo *"Intent",* que nos ayudará a pasar a la siguiente actividad de manera automática una vez haya transcurrido un periodo de tiempo predefinido en código:

```
TimerTask tarea = () -> {
        Intent intent = new Intent( packageContext: MainActivity.this, introPlayers.class);
        startActivity(intent);
        finish();
};

Timer tiempo = new Timer();
tiempo.schedule(tarea, delay: 5000);
```

Ilustración 32 - Código pantalla inicial

UML de la clase asociada a la actividad:



3.2.2 Actividad "introPlayers" (Introducción de jugadores)

En la actividad "introPlayers" se buscaba la introducción de los jugadores que participarán en el juego, de manera clara y que no se pudiesen modificar una vez iniciado el juego.

Para ello se usan los elementos *"Editext"* para la introducción y modificación del nombre antes de pasar a la siguiente actividad. Los nombres serán guardados en un array de tipo Player, una clase llamada Player.java en la que se estructura de manera similiar a un "Singleton" la estructura del jugador:

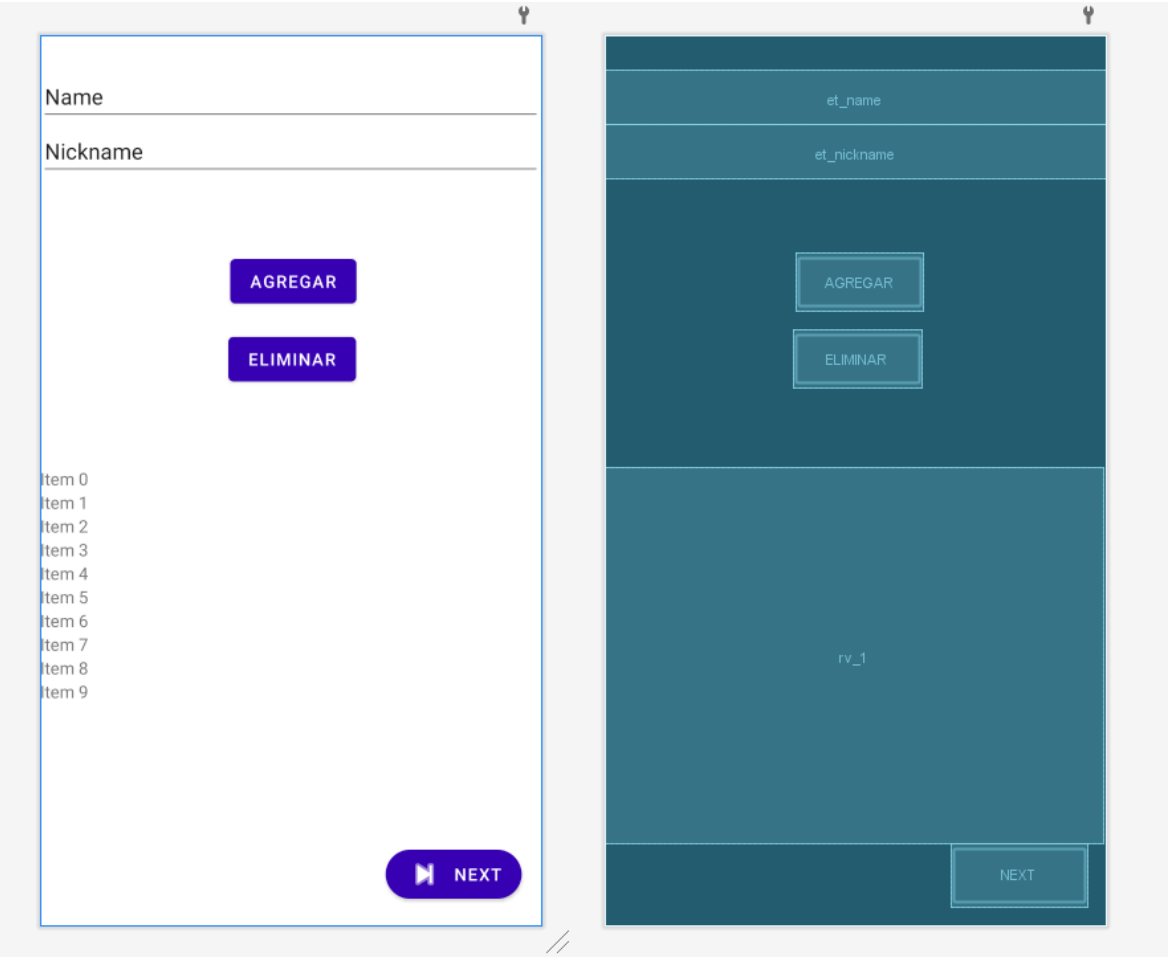


Ilustración 30 - Foto inicial de la actividad de introducción de jugadores

Ilustración 31 - Foto final de la actividad de introducción de jugadores

En esta actividad destacamos la introducción de los jugadores en un cuadro de tipo “ReyclerView”, un tipo de elemento de Android donde poder ejecutar listas dinámicas, con un Layout de fondo de tipo “ConstraintLayout” de nuevo, para que tengamos un lugar fijo en caso de tener un número de jugadores elevado y pueda mostrarse sin problemas en la pantalla.

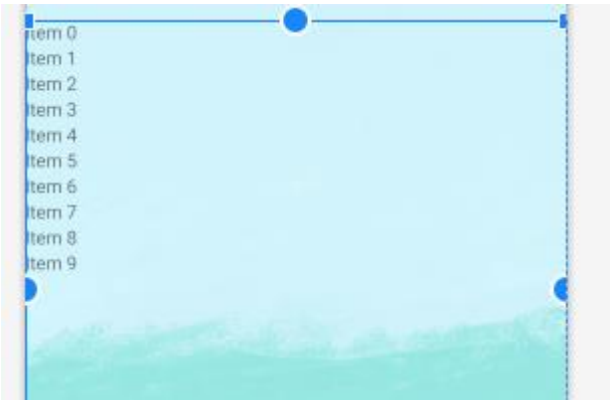


Ilustración 32 - Recyclerview

En el código java destacamos una clase “AdaptadorPlayerHolder”, la cual toma como referencia un activity “itemplayer_intro”, el cual da diseño a los jugadores que aparecerán dentro del Recyclerview.

Esta clase AdaptadorPlayerHolder, extiende de RecyclerView.Adapter

```
public class AdaptadorPlayer extends RecyclerView.Adapter<AdaptadorPlayer.AdaptadorPlayerHolder> {
    @NonNull
    @Override
    public AdaptadorPlayerHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        //crear un objeto de la clase AdaptadorPlayer
        return new AdaptadorPlayerHolder(getLayoutInflater().inflate(R.layout.itemplayer_intro,parent, attachToRoot: false));
    }

    @Override
    public void onBindViewHolder(@NonNull AdaptadorPlayerHolder holder, int position) {
        holder.imprimir(position);
    }

    @Override
    public int getItemCount() { return Player.getInstance().getArray().size(); }
    //crear cada elemento
    class AdaptadorPlayerHolder extends RecyclerView.ViewHolder implements View.OnClickListener{
        TextView tv1;
        //TextView tv2;
        public AdaptadorPlayerHolder(@NonNull View itemView) {
            super(itemView);
            tv1=itemView.findViewById(R.id.textView_name);
          //  tv2=itemView.findViewById(R.id.textView_puntuacion );
            //para capturar cuando hagamos click en la clase
            itemView.setOnClickListener(this);
```

Ilustración 36 - Código clase "AdaptadorPlayer"

El fichero .XML, referenciado anteriormente con el nombre “itemplayer_intro”, dispone de la siguiente estructura para dar forma a los jugadores dentro del recyclerview:

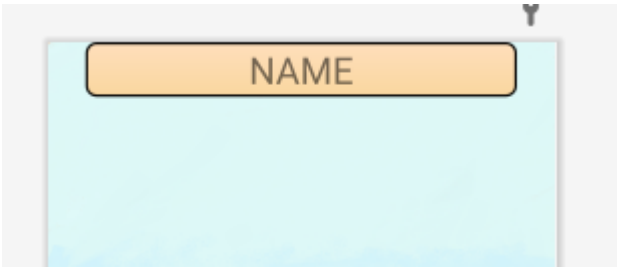
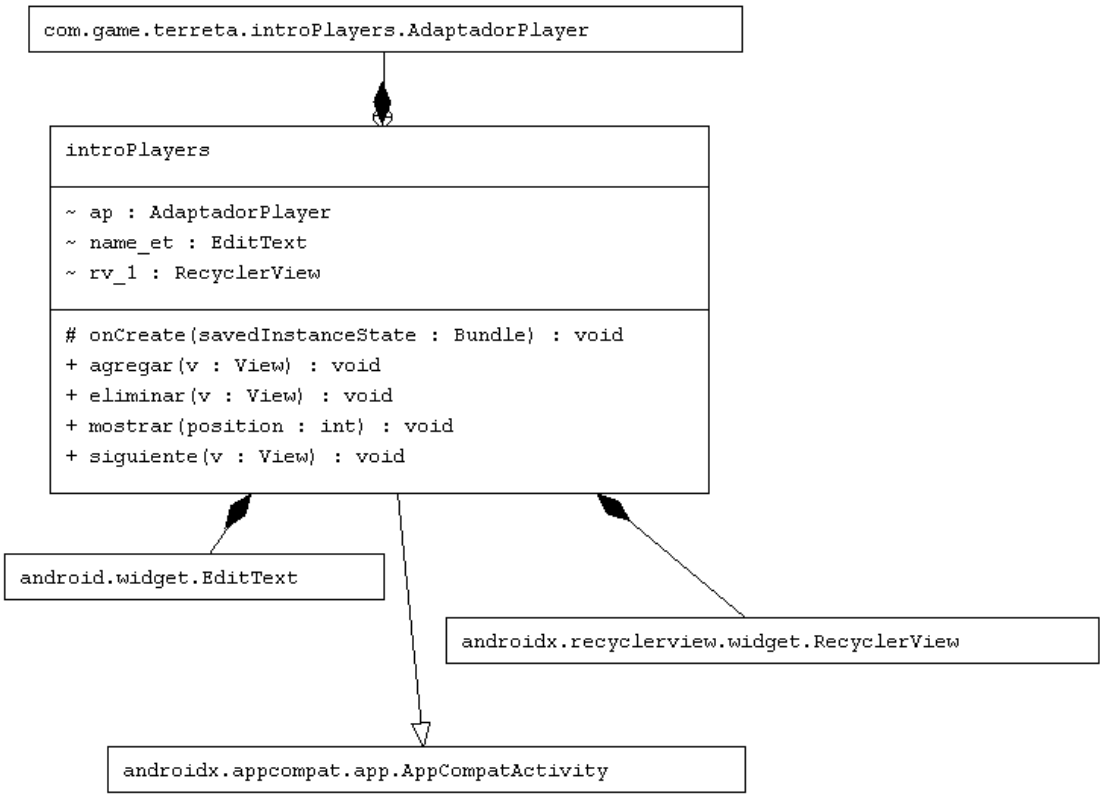


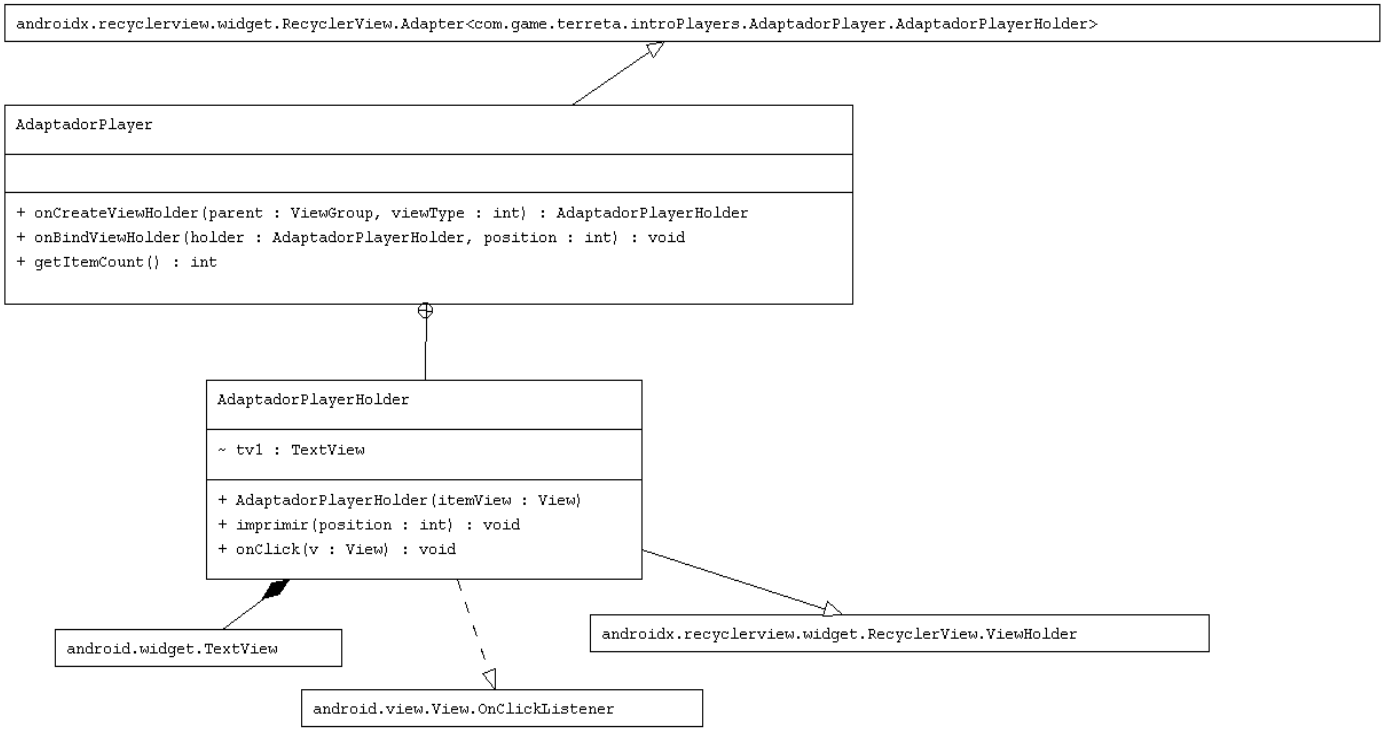
Ilustración 34 - Estilo para "recyclerview"

Disponemos de un LinearLayout englobando a un ConstraintLayout en el que se encuentra el TextView, donde aparecerá el nombre del jugadores que introduzcamos a la primera pantallla de nuestro juego.

UML de la clase asociada a la actividad:



UML de la clase asociada a la actividad:



3.3 Tercera Iteración

3.3.1 Actividad “Random Player” (Elección de jugador aleatorio)

La siguiente actividad nombrada como “random_player”, se realizará la una tirada aleatoria para elegir el primer jugador que elegirá carta y al que se le hará por consiguiente la pregunta en las actividades posteriores.

En esta actividad se implementa como objeto principal un botón aleatorio que trabaja sobre el array de tipo “Player” mencionado en la actividad anterior de introducción de jugadores.

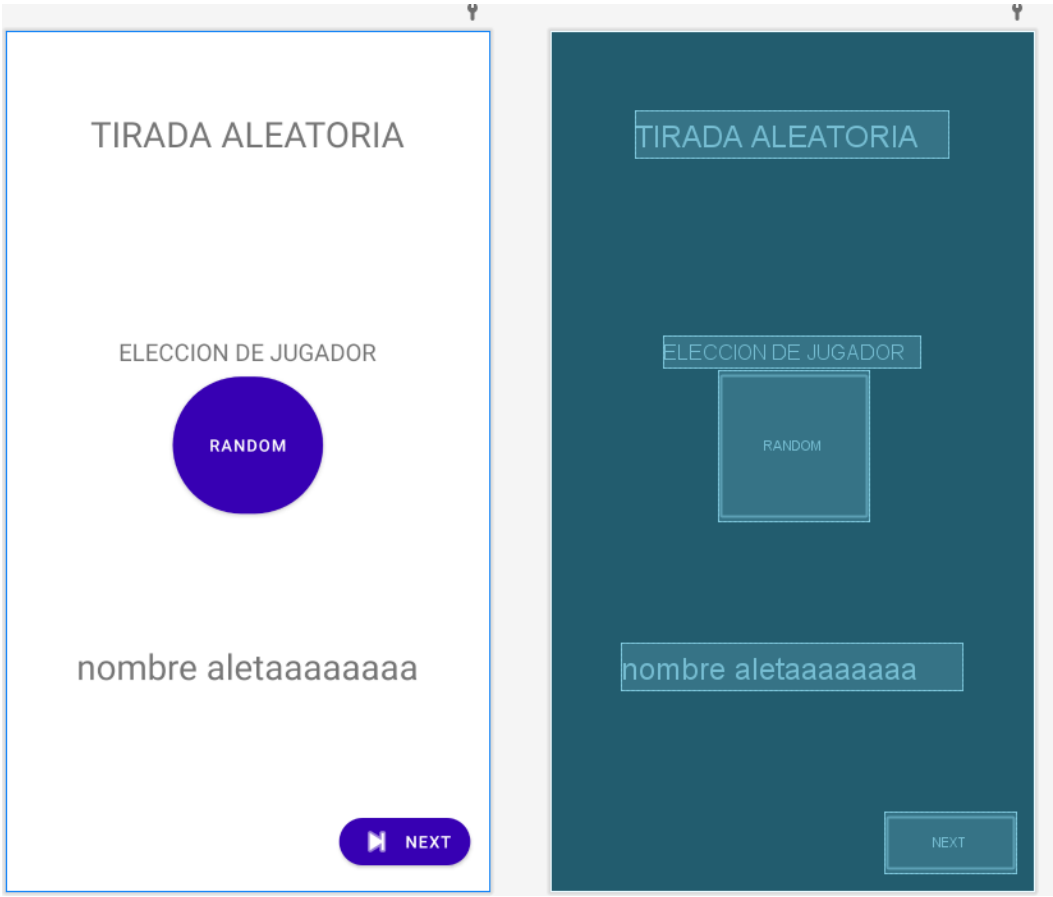


Ilustración 35 - Foto inicial de la actividad de elección de jugador aleatorio

Ilustración 36 - Foto final de la actividad de elección de jugador aleatorio

En el código java asociado a esta activity, podemos destacar el método *"newRandomPlayer",* el cual selecciona el jugador aleatorio dentro del array de tipo Player que ya hemos mencionado y al cual podemos acceder a través de una instancia y los métodos internos *"get",* para poder trabajar sobre cada jugador.

```java
public void newRandomPlayer(View v){

    int tamanio = Player.getInstance().getArray().size();
    System.out.printf("El tamaño es este:  %s", tamanio);
    int indiceAleatorio = numeroAleatorioEnRango(0, Player.getInstance().getArray().size() ); //obtenemos un número aleatroio para seleccionar el índice del array de players
    System.out.printf("ALEATORIO ES  %s", indiceAleatorio);
    System.out.printf("HEREEEEEEEEE");
    players.getArray();

    for(int i=0; i<players.getArray().size();i++){
        System.out.print(players.getArray().get(i).getNombre() );
        if(i == indiceAleatorio){
            name_tv.setText(Player.getInstance().getArray().get(i).getNombre());
            positionNext_randomPlayer = i;
        }
    }
```

Ilustración 40 - Código JAVA asociado a la actividad "Random Player"

Destacar también el método *"siguiente",* que nos servirá para pasar a la siguiente actividad cuando presionemos el botón *"next",* pero en este caso contamos con una particularidad que nos será útil durante todo el juego, y es guardar el "jugador aleatorio 1", para poder usarlo tanto en la actividad de segunda tirada aleatoria como en la actividad de elección de carta por parte de todos los jugadores restantes.

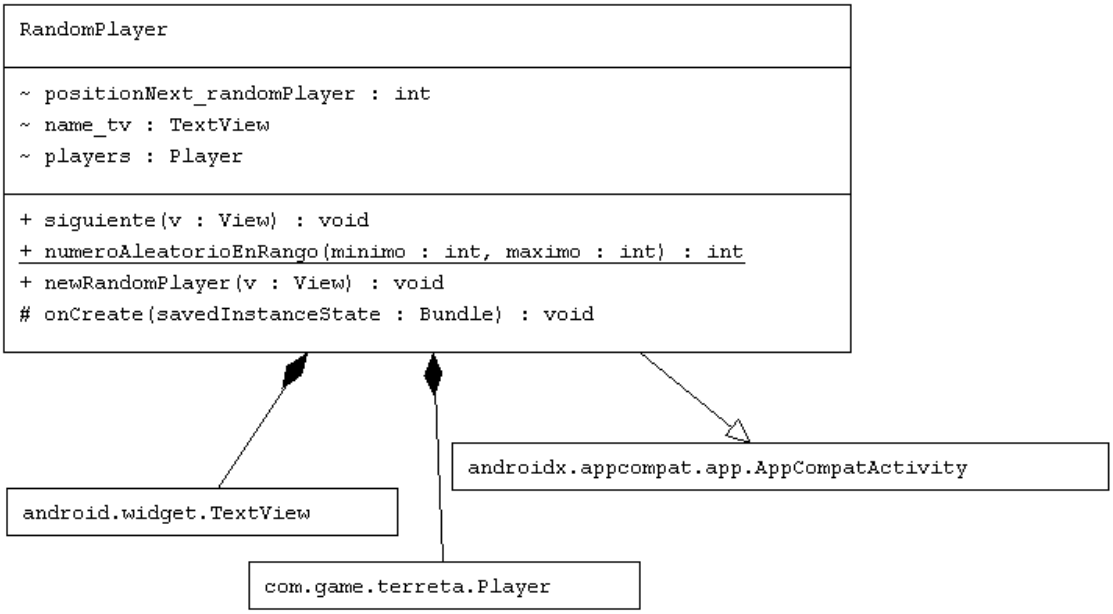
Esto se hará a través de una variable de tipo *"Bundle"* que usara un método interno en este caso *"putInt"* donde le pasaremos como datos la variable que queremos enviar a la siguiente actividad así como una *"key"* asociada, para poder llamar a esa variable en la actividad siguiente.

Seguido de ello, la variable de tipo *"Intent"* que nos sirve para pasar a la siguiente actividad, contiene un método *"putExtras"* donde se le pasará como variable la anterior de tipo Bundle, esto nos servirá para rescatar lo mencionado antes, gracias a un método *"getExtras"* en la siguiente actividad

```
public void siguiente(View v){

    Intent siguiente = new Intent( packageContext: this, eleccion_carta_1.class);
    Player.getInstance().getArray();

    Bundle position_randomplayer_1 = new Bundle();
    position_randomplayer_1.putInt("randomplayer_1_key",positionNext_randomPlayer);

    siguiente.putExtras(position_randomplayer_1);
    startActivity(siguiente);
}
```

Ilustración 38 - Método "siguiente"

UML de la clase asociada a la actividad:



3.3.2 Actividad “Eleccion_carta_1”

En la actividad “elección_carta_1” se desarrolla una de las claves del juego, y es la elección de la carta preferida por el primer jugador aleatorio que hemos recopilado en la actividad anterior de “Random Player”. Una vez elegida la carta, guardaremos también la elección junto con el “jugador 1” como se hizo anteriormente para así, poder comparar en las próximas pantallas frente a los demás jugadores.

En esta actividad se implementan como principal dos botones de estilo “ToggleButton” los cuales tienen asociados una alternancia de “activado” o “desactivado” frente a los “Button”. En este caso es la opción perfecta para la elección de carta, la cual se activa o se desactiva la carta contraria cuando está seleccionada una de ellas:

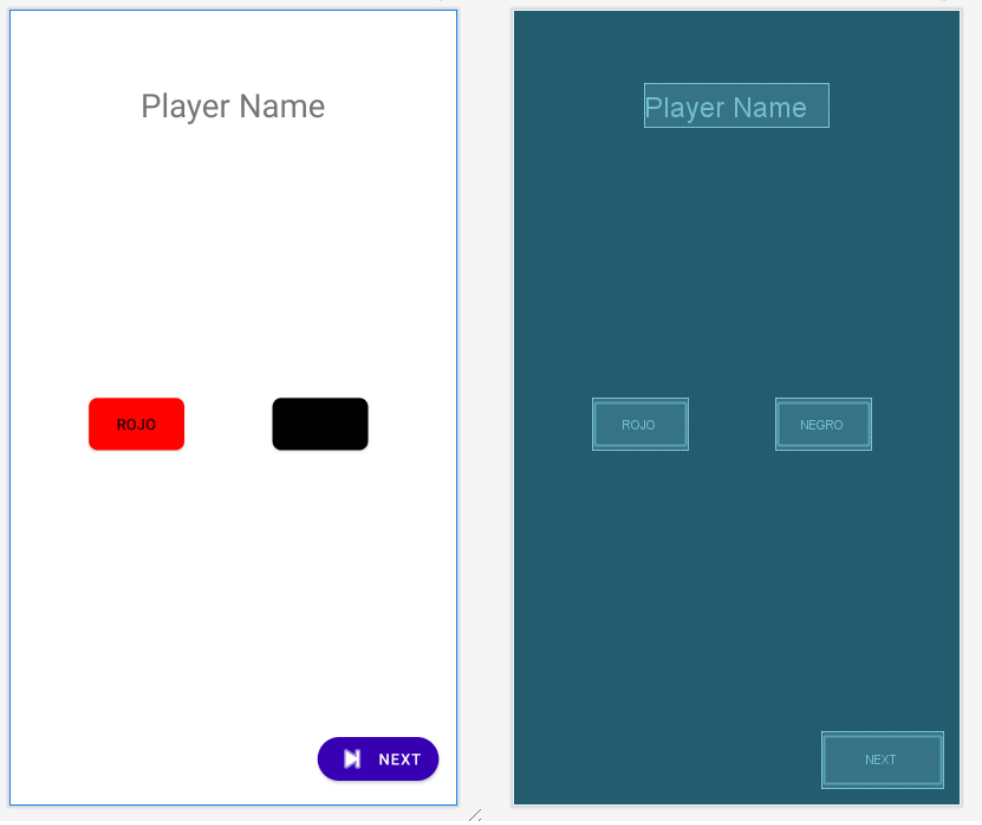


Ilustración 39 - Foto inicial de la actividad de elección de carta para jugador 1

Ilustración 40 - Foto final de la actividad de elección de carta para jugador 1

En el código Java destacamos como se ha mencionado anteriormente, la funcionalidad de los “TogleButton”:

```
    public void onClickButtonRed(View v) {
//
        ToggleButton clickedButton = (ToggleButton) v;

        if (clickedButton.getId() == R.id.tv_boton_carta_roja) {
            redCardButton.setChecked(true);
            blackCardButton.setChecked(false);
            redCardControl = true;
            blackCardControl = false;

        }

//        System.out.println("Estado de ROJO: " + redCardControl);
//        System.out.println("Estado de NEGRO : " + blackCardControl);
    }
```

```
    public void onClickButtonBlack(View v) {
//
        ToggleButton clickedButton = (ToggleButton) v;

        if (clickedButton.getId() == R.id.tv_boton_carta_negra) {
            blackCardButton.setChecked(true);
            redCardButton.setChecked(false);
            blackCardControl = true;
            redCardControl = false;
        }
//        System.out.println("Estado de ROJO: " + redCardControl);
//        System.out.println("Estado de NEGRO : " + blackCardControl);
    }
```

Ilustración 44 - Métodos "onClickButtonRed" y "onClickButtonBacl"

En estos métodos se activan o desactivan los ToggleButton dependiendo de la acción seleccionada sobre la carta, para así poder guardar la elección del primer jugador.

De manera visual lo veríamos de la siguiente manera:

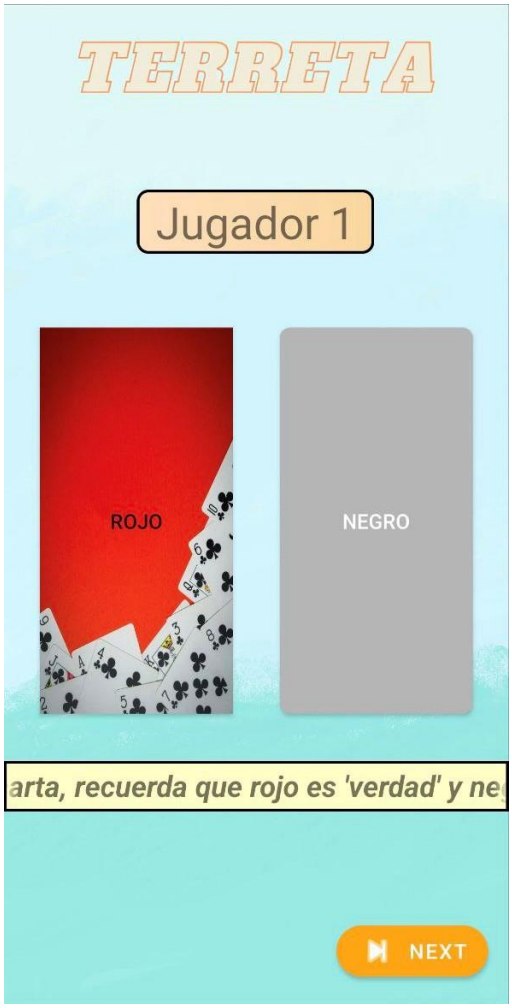


Ilustración 45 - Carta roja seleccionada

Ilustración 46 - Carta negra seleccionada

En esta actividad es necesario destacar también este funcionamiento visual sobre los “ToggleButton”, los cuales adquieren un fondo de color gris cuando el estado es “desactivado” o adquieren el color de la carta cuando están activados.

Se ha trabajado en esto sobre los XML del activity.

Como ejemplo, veremos el funcionamiento de la carta roja:

```
<ToggleButton

    android:id="@+id/tv_boton_carta_roja"
    android:layout_width="150dp"
    android:layout_height="300dp"
    android:layout_marginEnd="8dp"
    android:background="@drawable/button_red_selector"

    android:checked="true"

    android:hapticFeedbackEnabled="false"
    android:isScrollContainer="false"

    android:onClick="onClickButtonRed"

    android:text="ROJO"
    android:textOff="ROJO"
    android:textOn="ROJO"
    app:layout_constraintBottom_toTopOf="@+id/button_next_3"
    app:layout_constraintEnd_toStartOf="@id/tv_boton_carta_negra"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/tv_pregunta_firstplayer"
    app:layout_constraintVertical_bias="0.266" />
```

Ilustración 44 - Código XML de la actividad "Eleccion_carta_1"

Se decidió usar el fondo del “ToggleButton” para realizar esta acción, con un xml de tipo “drawable” llamado *“button_red_selector”:*

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item  android:state_checked="true">
        <bitmap android:src="@drawable/carta_roja"
            />
    </item>
    <item android:drawable="@drawable/button_red_default"  />
</selector>
```

Ilustración 45 - Código XML de la actividad "Eleccion_carta_1"

En cuanto al código java, como en todas las “activity”, disponemos del método “siguiente” para poder realizar la transición a la siguiente actividad, aquí volvemos a puntualizar que necesitamos “pasar”, a la siguiente actividad los datos recogidos sobre el primer jugador elegido, en este caso su elección de carta, para después hacer comparación con la elección de cada uno de los jugadores restantes.

Se realizará dependiendo de la acción que tome el jugador, si carta roja o carta negra:

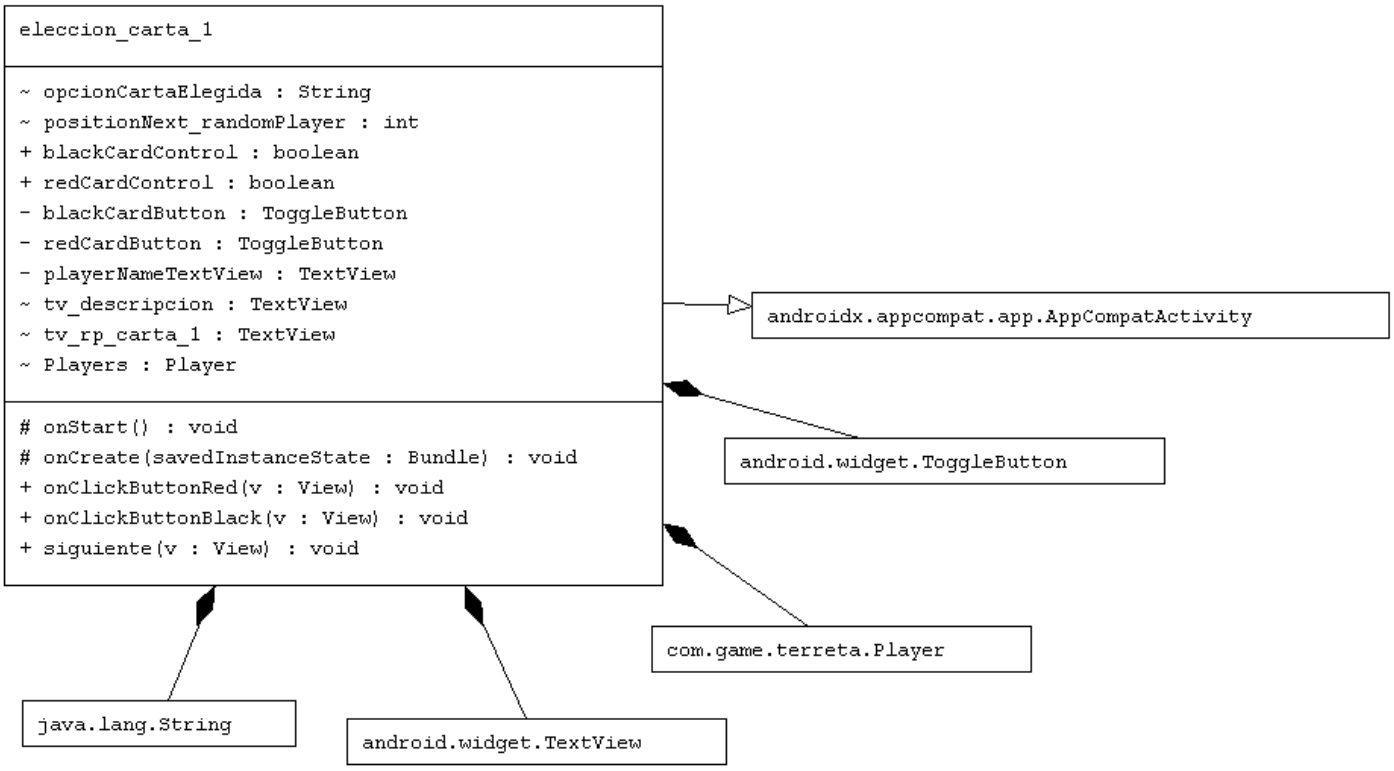
```
public void siguiente(View v){

    Intent siguiente = new Intent( packageContext: this,RandomPlayer_2.class);
    Player.getInstance().getArray();
    Bundle bundleToNext = new Bundle();

    bundleToNext.putInt("randomplayer_1_key",positionNext_randomPlayer);
    bundleToNext.putBoolean("redCardControl_1_key",redCardControl);
    bundleToNext.putBoolean("blackCardControl_1_key",blackCardControl);

    if(redCardControl){
        opcionCartaElegida = "rojo";
    }else if (blackCardControl){
        opcionCartaElegida = "negro";
    }
    bundleToNext.putString("opcionCartaElegida_key",opcionCartaElegida);
    siguiente.putExtras(bundleToNext);

    startActivity(siguiente);
}
```

Ilustración 46 - Código método "siguiente" de la actividad "Eleccion_carta_1"

De nuevo hacemos uso de “Bundle”, para asociar “keys” y recuperar estos datos en la siguiente actividad.

También un “slider” de fondo amarillo de derecha a izquierda, el cual nos aporta un recordatorio sobre la información de cada carta, rojo “verdad”, negro “mentira”. Servirá para recordar al jugador que debe decir cuando seleccione la carta.

UML de la clase asociada a la actividad:



3.4 Cuarta Iteración

3.4.1 Actividad “Random_Player_2” (Elección de jugador aleatorio)

En esta actividad encontraremos un patrón similar a la actividad “random_player_1”, donde se realiza una tirada aleatoria para seleccionar uno de los jugadores del array de tipo “Player” con el que disponemos durante toda la partida.

En esta actividad la particularidad es que debemos descartar al primer jugador aleatorio seleccionado, ya que él no puede realizarse la pregunta así mismo

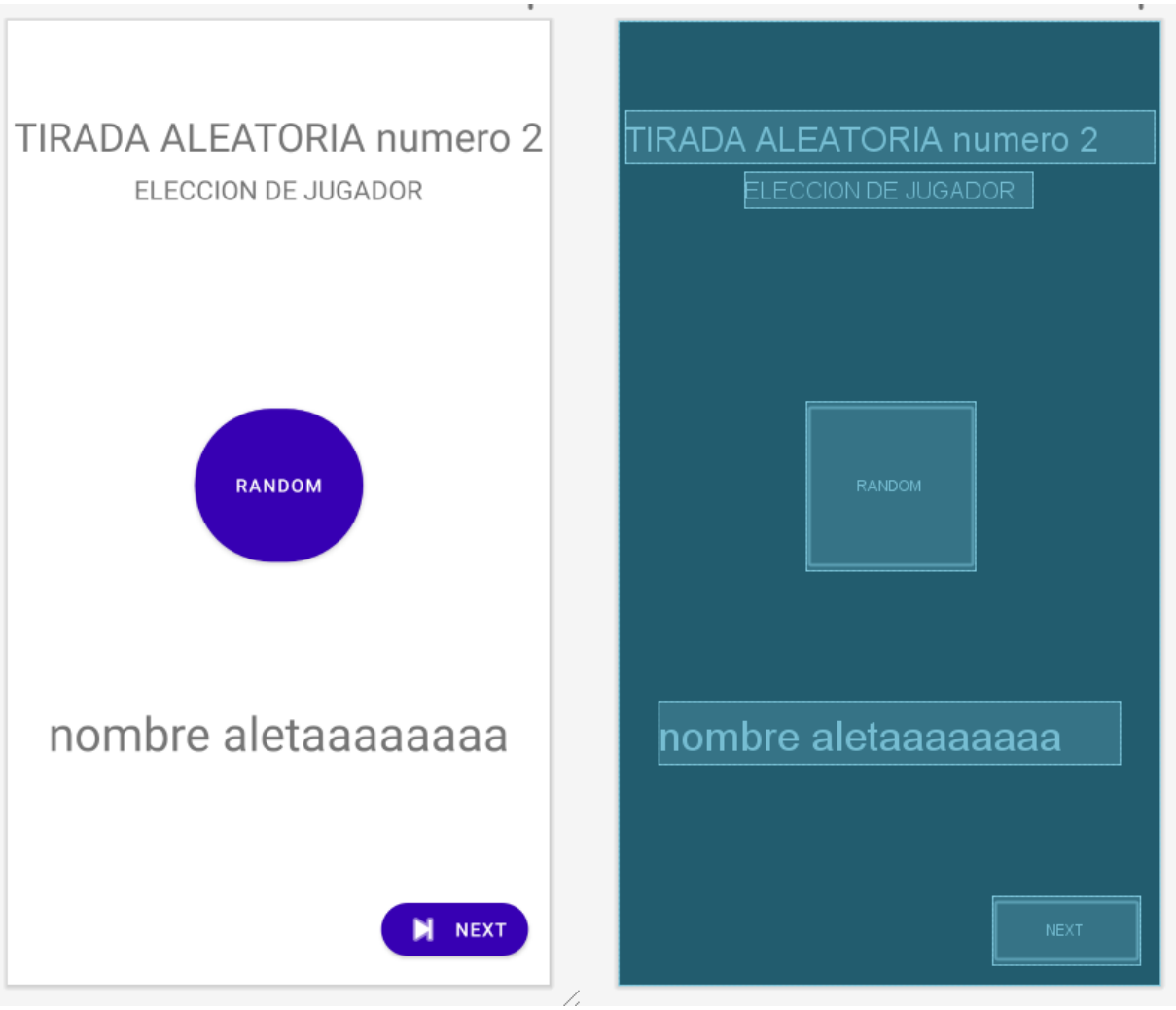


Ilustración 47 - Foto inicial de la actividad de elección de jugador aleatorio 2

Ilustración 48 - Foto final de la actividad de elección de jugador aleatorio 2

Para realizar la elección aleatoria, debemos descartar el primer jugador aleatorio elegido, para ellos haremos uso del “Bundle” y se recopilarán los “extras” gracias al método *“getIntent().getExtras()”* y a través de las “keys” que anteriormente asociamos a cada dato que necesitaremos en esta ocasión.

Vemos estos datos en el código java asociado:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_random_player_2);
    name_tv = findViewById(R.id.textView_ranName_2);


    LinearLayoutManager llm= new LinearLayoutManager( context: this);
    Bundle extras = getIntent().getExtras();
    if (extras != null) {
        int randomPlayer = extras.getInt( key: "randomplayer_1_key");
        if (randomPlayer >= 0) {
            for(int i=0; i<Player.getInstance().getArray().size();i++){
                System.out.print(Player.getInstance().getArray().get(i).getNombre() );
                if(i == randomPlayer){
                    name_toCompare = Player.getInstance().getArray().get(i).getNombre();
                }
            }

        }
    }
    intoCompare = extras.getInt( key: "randomplayer_1_key");
    opcionCartaElegida = extras.getString( key: "opcionCartaElegida_key");

//        System.out.println("EL NOMBRE ESSSSSSSSSSSSS   " + name_toCompare);

}
```

Ilustración 49 - Código asociado a la actividad "Random_Player_2"

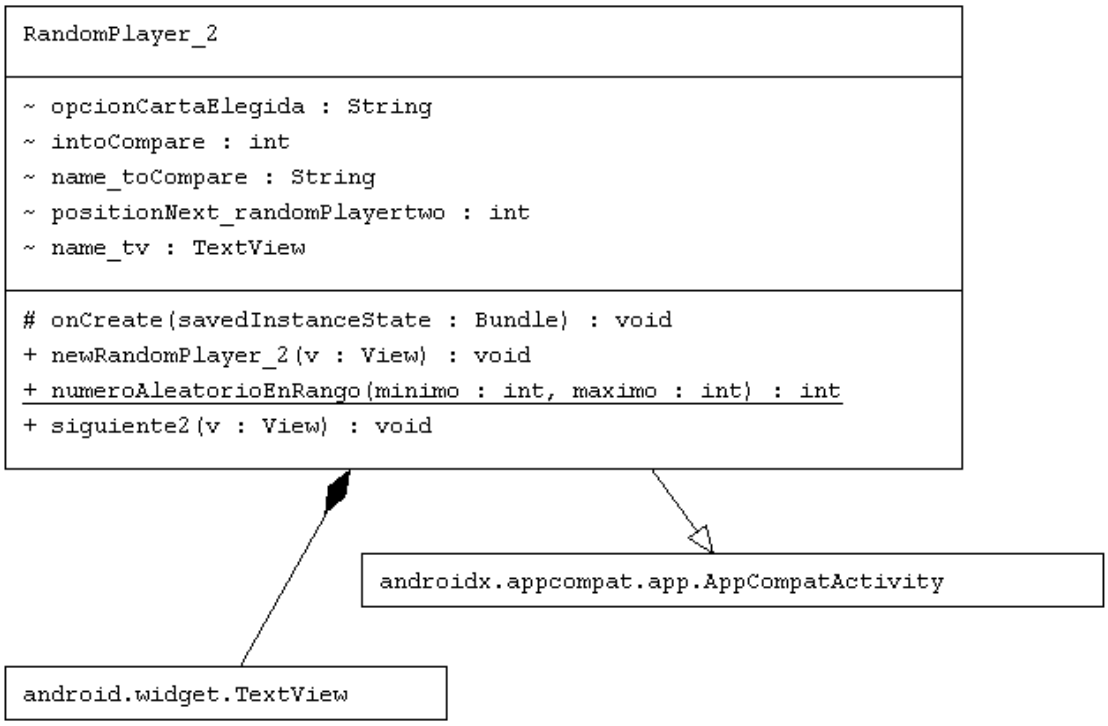
Realizando el método *"newRandomPlayer_2"*, recorremos el array y nunca mostraremos el jugador aleatorio 1, lo descartamos a través de una variable *"name_toCompare"*.

En esta activity, seguimos guardando nuevos datos que necesitaremos en la siguiente, en este caso el nombre del jugador aleatorio 2, que realizará la pregunta al jugador 1 (elegido la primera vez), en la siguiente actividad:

```
public void siguiente2(View v){
    Intent siguiente2 = new Intent( packageContext: this,pregunta_player.class);
    Player.getInstance().getArray();

    Bundle position_randomplayer_2 = new Bundle();
    position_randomplayer_2.putInt("randomplayer_1_key",intoCompare);
    position_randomplayer_2.putInt("randomplayer_2_key",positionNext_randomPlayertwo);
    position_randomplayer_2.putString("opcionCartaElegida_key",opcionCartaElegida);
    siguiente2.putExtras(position_randomplayer_2);
    startActivity(siguiente2);
}
```

Ilustración 50 - Método "siguiente" asociado a la actividad "Random_Player_2"

UML de la clase asociada a la actividad:



3.4.2 Actividad “Pregunta_Player”

En la actividad “pregunta_player”, obtendremos a través de los extras, el nombre del jugador aleatorio 2 que hemos obtenido en la tirada aleatoria de la actividad anterior.

Este jugador realizará la pregunta como bien muestra visualmente un texto y un GIF que se ha asociado a través de la web con la función *“Glide” a un elemento “ImageView”* y “parseando” la URL que asignamos desde la web gracias a la clase “URI” que se encarga de descomponer “urls” .

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_pregunta_player);

    Bundle extras = getIntent().getExtras();

    position_randomplayer_1 = extras.getInt( key: "randomplayer_1_key");
    opcionCartaElegida = extras.getString( key: "opcionCartaElegida_key");
    tv_rp_2 = findViewById(R.id.tv_pregunta_allplayer);

    tv_rp_2.setText(Player.getInstance().getArray().get(extras.getInt( key: "randomplayer_2_key")).getNombre());

    String url = "https://i.gifer.com/BYUH.gif" ;
    ImageView imageGif = findViewById(R.id.questionGif);

    Uri uri = Uri.parse(url);
    Glide.with(getApplicationContext()).load(uri).into(imageGif);

}
```

Ilustración 54 - Código asociado a la actividad "Pregunta_Player"

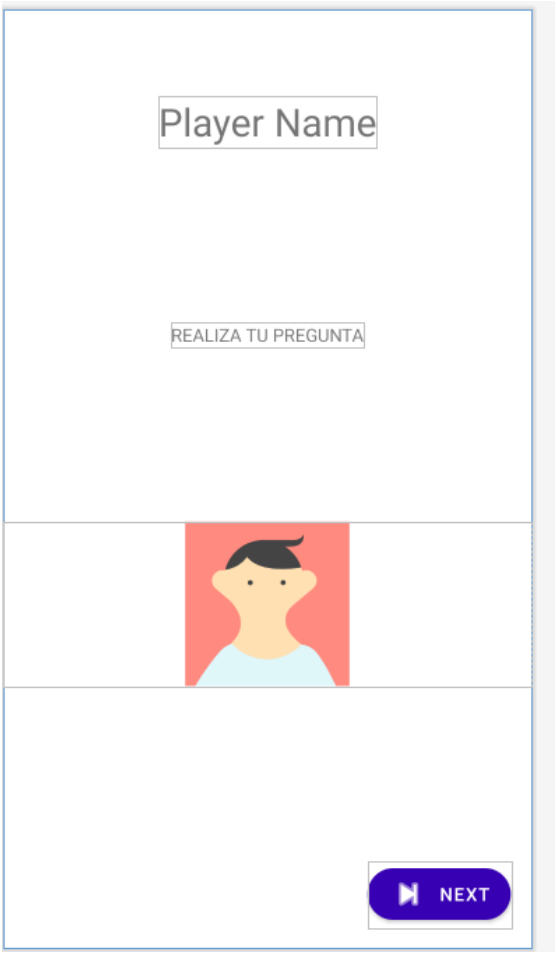


Ilustración 56 - Foto inicial de la actividad "Pregunta_Player"

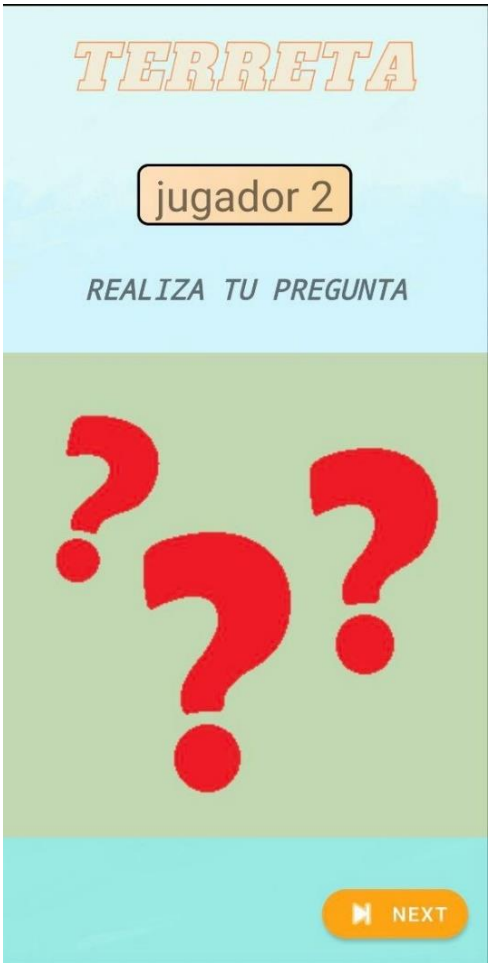
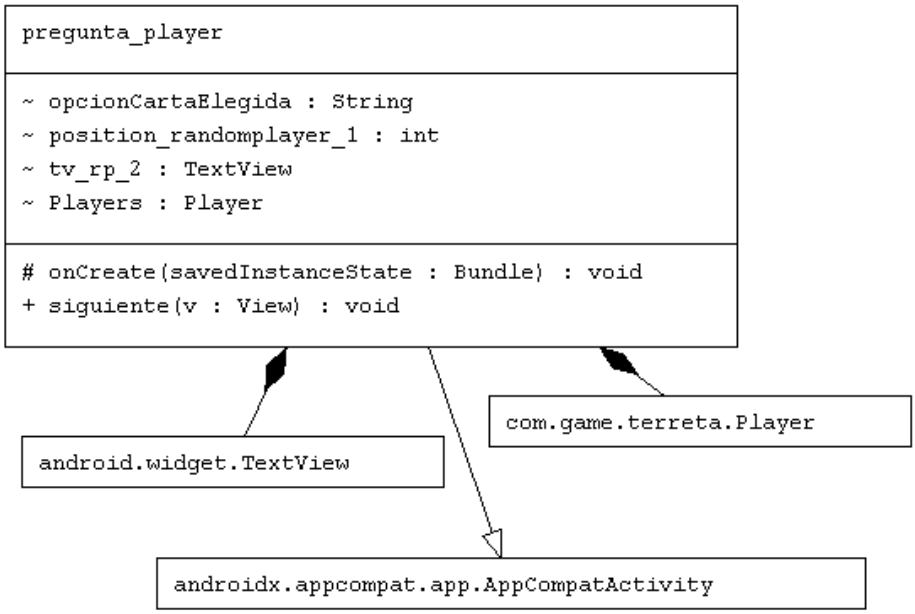


Ilustración 55 - Foto final de la actividad "Pregunta_Player"

UML de la clase asociada a la actividad:



3.4.3 Actividad “Elección_Carta_all_players”

En esta actividad, se seguirá un patrón parecido al de la actividad “Elección carta 1”, pero el proceso de la gestión de los jugadores será similar al de la actividad “Random Player 2”, donde debemos recorrer todos los jugadores para que elijan la carta que ellos creen que el “jugador 1” ha elegido, recordando que cada carta (roja o negra), estaba asociada a una opción.

Para ello debemos de descartar al “jugador 1”, ya que es el jugador al que deben adivinar su elección.

Para ello, recogemos de nuevo los datos necesarios de las actividades anteriores gracias de nuevo al “Bundle” junto a la función “getExtras”.

Necesitaremos conocer de nuevo un identificador para el “jugador 1” elegido y controlar que cuando sea su turno no pueda elegir.

Para evitar que el jugador de la primera tirada vuelva a elegir carta, se ha utilizado una funcionalidad de “pop-up”, *“AlertDialog”* el cuál avisará cuando sea el turno del “jugador 1” elegido de forma aleatoria en la primera tirada.

Código java usado para la realización del “AlertDialog “:

```
if(recorrer != Player.getInstance().getArray().size()){
    if (name_toCompare.equals(Player.getInstance().getArray().get(recorrer).getNombre())) {
        AlertDialog.Builder builder = new AlertDialog.Builder( context: this);
        builder.setTitle("NO ELIGE")
                .setMessage("El jugador " + Player.getInstance().getArray().get(recorrer).getNombre() + " no elige carta, por favor pulsa en Siguiente Jugador")
                .setPositiveButton( text: "SIGUIENTE JUGADOR", new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        // Acciones a realizar al hacer clic en Aceptar
                        allplayerList(v);
                    }
                })
                .setNegativeButton( text: "Cancelar", new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        // Acciones a realizar al hacer clic en Cancelar
                        allplayerList(v);
                        dialog.dismiss(); // Cerrar el diálogo
                    }
                });

        AlertDialog dialog = builder.create();
        dialog.show();
    }
    tv_allnames.setText(Player.getInstance().getArray().get(recorrer).getNombre());
}
```

Ilustración 57 - Código para la realización del "AlertDialog"

Controlamos a través del nombre del jugador guardado en el array de tipo “Player”, cuando coincida el nombre con el jugador de la primera tirada aleatoria, se mostrará el “pop-up” y al seleccionar el botón de “Siguiente Jugador”, volverá a ejecutar el método “allplayerList”, donde se recorre cada jugador de la partida.

Cuando tenga lugar esta casuística, se mostrará el siguiente mensaje en pantalla:

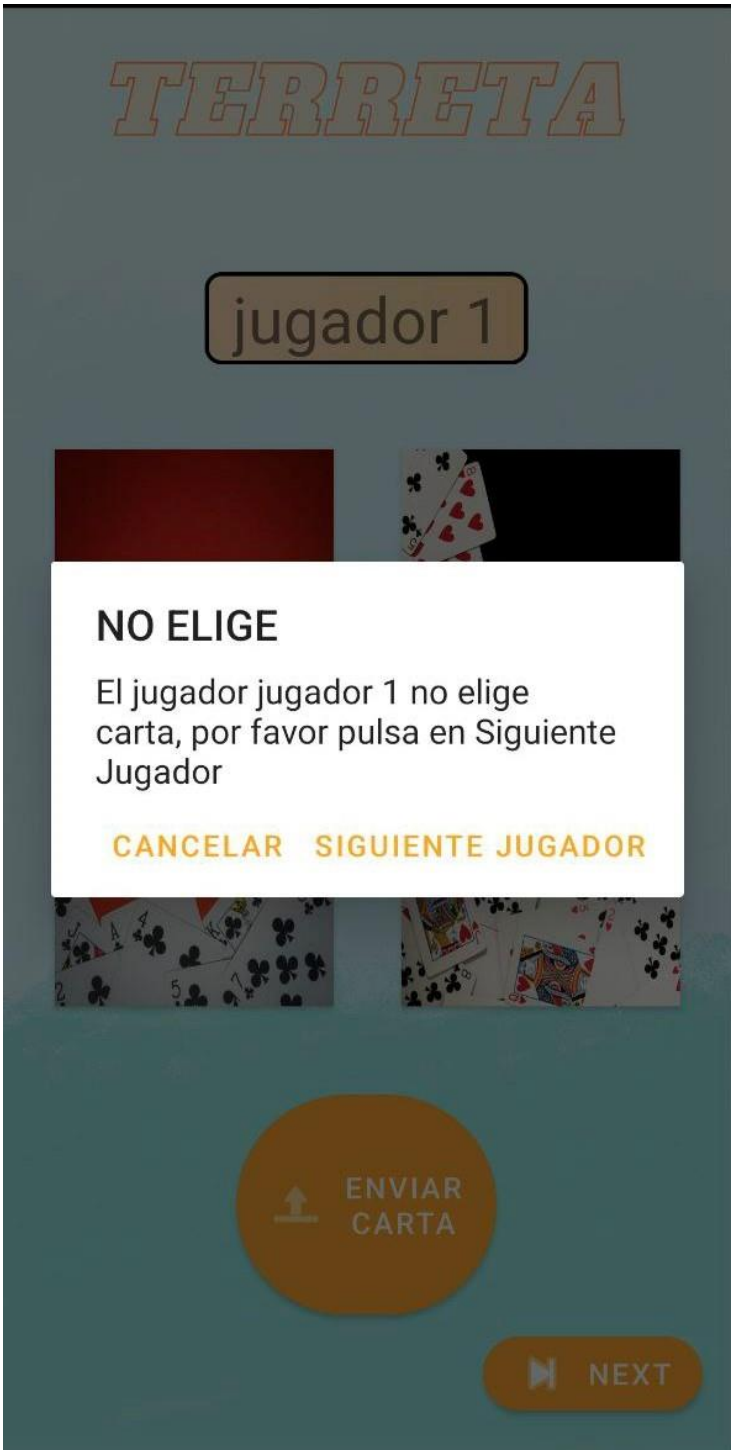


Ilustración 55 - Foto del "AlertDialog" final

El código anterior, controla cuando sea el primer jugador del array guardado, es decir, en este caso mostrado “jugador 1”, pero cuando el jugador que ha elegido

carta previamente no coincida con el primero, debemos también mostrar el “AlertDialog”, para ello reutilizamos el mismo planteamiento anterior pero dentro del método “allplayerList”, donde se recorre todo el array de jugadores y guarda la elección de carta de cada jugador:

Código método *"allplayerList"*:

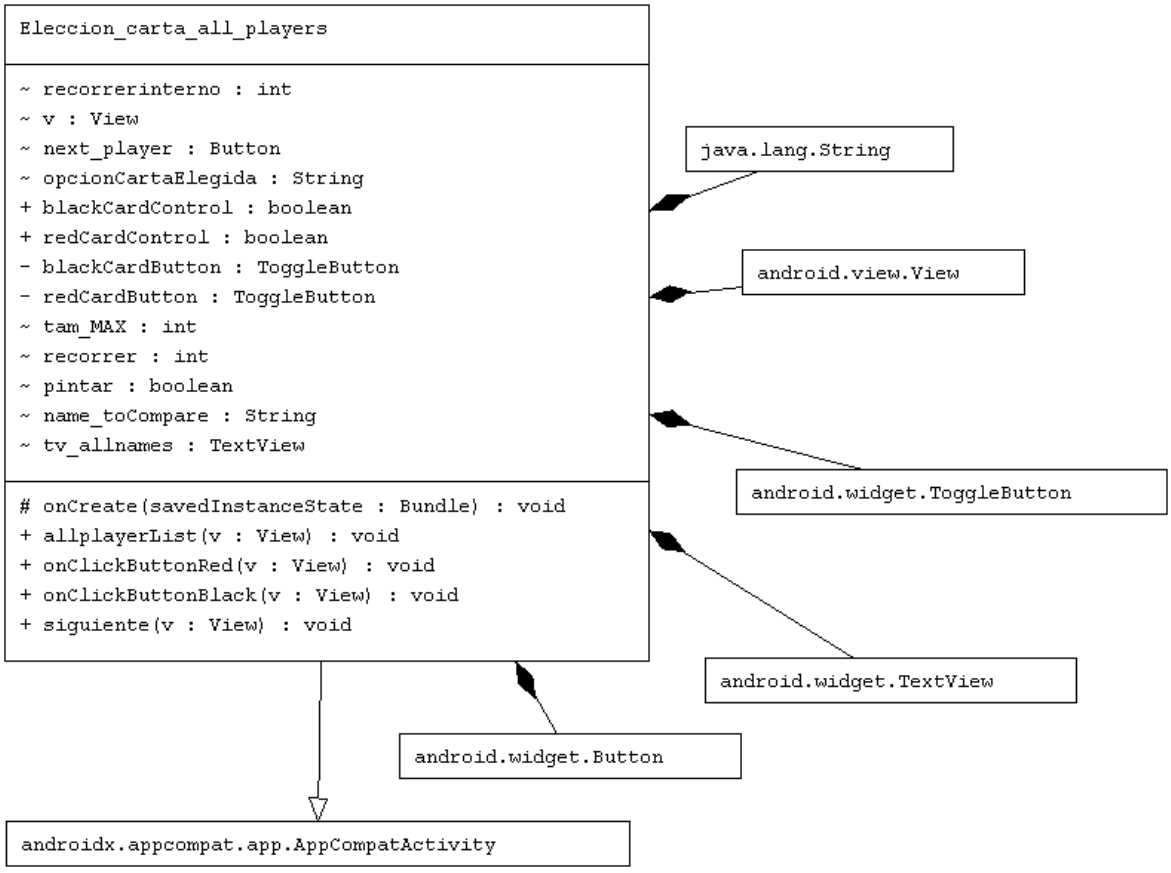
```
public void allplayerList(View v){

    recorrer++;
-------------------------------------------------------------------------------------------
        if(recorrerinterno < Player.getInstance().getArray().size()){
            if(recorrer != Player.getInstance().getArray().size()){
                if (name_toCompare.equals(Player.getInstance().getArray().get(recorrer).getNombre())) {
                    AlertDialog.Builder builder = new AlertDialog.Builder( context: this);
                    builder.setTitle("NO ELIGE")
                            .setMessage("El jugador " + Player.getInstance().getArray().get(recorrer).getNombre() + " no elige carta, por favor pulsa en Siguiente Jugador")
                            .setPositiveButton( text: "SIGUIENTE JUGADOR", new DialogInterface.OnClickListener() {
                                public void onClick(DialogInterface dialog, int id) {
                                    // Acciones a realizar al hacer clic en Aceptar
                                    allplayerList(v);
                                }
                            })
                            .setNegativeButton( text: "Cancelar", new DialogInterface.OnClickListener() {
                                public void onClick(DialogInterface dialog, int id) {
                                    // Acciones a realizar al hacer clic en Cancelar
                                    allplayerList(v);
                                    dialog.dismiss(); // Cerrar el diálogo
                                }
                            });

                    AlertDialog dialog = builder.create();
                    dialog.show();
                }
                tv_allnames.setText(Player.getInstance().getArray().get(recorrer).getNombre());
            }

            //-------------------------------------------------------------------------------------------------------------------------------------------
//              System.out.println("MUESTRAAAAAA: " + Player.getInstance().getArray().get(recorrerinterno).getNombre());
            if(name_toCompare.equals(Player.getInstance().getArray().get(recorrerinterno).getNombre())){
                System.out.printf("ESTE NO SE HACEE "+ Player.getInstance().getArray().get(recorrerinterno).getNombre()  + "/n") ;
            }else{
                if(!redCardControl && !blackCardControl){
                    Toast.makeText( context: this, text: "DEBES ELEGIR UNA CARTA",Toast.LENGTH_SHORT).show();
//              break;
                }else{
                    if(redCardControl){
                        if(opcionCartaElegida.equals("rojo")){
                            Player.getInstance().getArray().get(recorrerinterno).setPuntuacion(+1);
//                              System.out.println("HA ELEGIDO RRRRRRRROOOOJOOOO");
//                              System.out.println("jugador: " + Player.getInstance().getArray().get(recorrerinterno).getNombre() + " puntuacion: " + Player.getInst
                        }
                    }
                    if(blackCardControl){
                        if(opcionCartaElegida.equals("negro")){
                            Player.getInstance().getArray().get(recorrerinterno).setPuntuacion(+1);
//                              System.out.println("HA ELEGIDO NEEEEEEGROOOOOOOOOOOOOOO");
//                              System.out.println("jugador: " + Player.getInstance().getArray().get(recorrerinterno).getNombre() + " puntuacion: " + Player.getInst
                        }
                    }
                }
            }

        //cierre del if:
        recorrerinterno++;
        if(recorrerinterno == Player.getInstance().getArray().size()) {
            Toast.makeText( context: this, text: "NO HAY MAS JUGADORES",Toast.LENGTH_SHORT).show();
        }
    }else{

//          System.out.println("FIIIINNNNNN DEL RECORRIDOOO");
```

Ilustración 59 - Código asociado a la clase "Elección_carta_all_players"

Una vez terminen de elegir todos los jugadores, se mostrará un mensaje de aviso de que no hay más jugadores disponibles para seguir eligiendo carta, así pasaríamos a la actividad final donde nos mostrará la puntuación actual.

Ilustración 57 - Foto final de la actividad "elección_carta_all_players"

UML de la clase asociada a la actividad:



3.5 Quinta Iteración

3.5.1 Actividad “Resultados”

Esta será la actividad final del juego, donde se mostrarán las puntuaciones conseguidas por los jugadores en cada tirada. No son puntuaciones únicas por cada tirada, sino que son acumulables cada vez que se repita el juego con los mismos jugadores.

Como concepto inicial de la actividad se pensó en una muestra de las puntuaciones asociadas a los jugadores cada vez, para comprobar solamente quien había ganado esa tirada, pero al tener un array de tipo “Player” donde los jugadores no se pueden modificar, las puntuaciones pueden ser acumulables y así, poder realizar tantas tiradas como se desee con estos jugadores, para obtener una puntuación no solo de “1 punto”

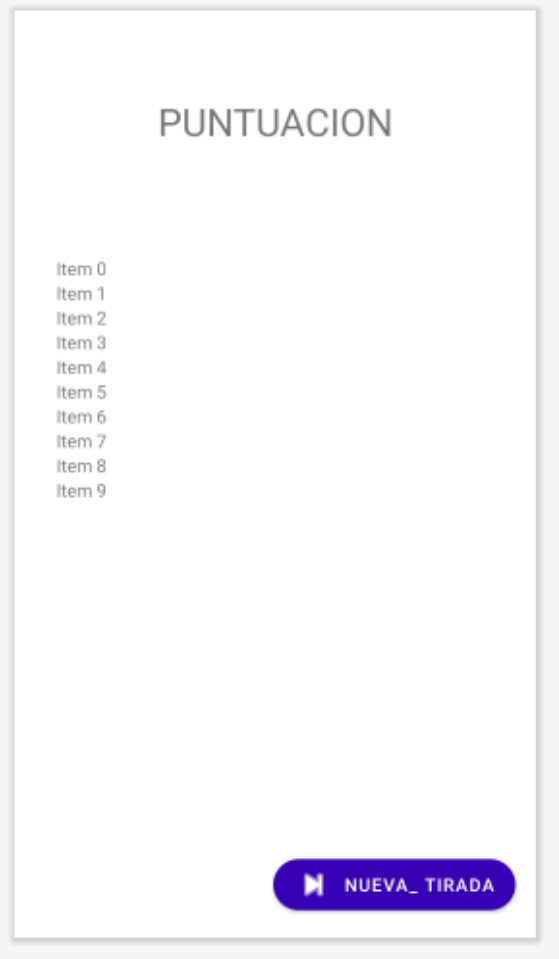


Ilustración 58 - Foto inicial de la actividad "resultados"

En esta actividad, se vuelve a usar la funcionalidad de los "RecyclerView" para mostrar la lista dinámica de los jugadores de toda la partida y sus puntuaciones.

En el código volvemos a usar la clase *"AdaptadorPlayer"*, pero en este caso referencia la actividad *"itemplayer"*, similar a *"itemplayer_intro"*, pero con las puntuacione. Dará el diseño a los jugadores que aparecerán dentro del Recyclerview.

Java de la clase “AdaptadorPlayer”

```
//-----------------------------------------------------------------------------------------------
public class AdaptadorPlayer extends RecyclerView.Adapter<Resultados.AdaptadorPlayer.AdaptadorPlayerHolder> {
    @NonNull
    @Override
    public Resultados.AdaptadorPlayer.AdaptadorPlayerHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        //crear un objeto de la clase AdaptadorPlayer
        return new Resultados.AdaptadorPlayer.AdaptadorPlayerHolder(getLayoutInflater().inflate(R.layout.itemplayer,parent, attachToRoot: false));
    }

    @Override
    public void onBindViewHolder(@NonNull AdaptadorPlayerHolder holder, int position) {
        holder.imprimir(position);
    }

    @Override
    public int getItemCount() { return Player.getInstance().getArray().size(); }
    //crear cada elemento
    class AdaptadorPlayerHolder extends RecyclerView.ViewHolder implements View.OnClickListener{
        TextView tv1;
        TextView tv2;
        public AdaptadorPlayerHolder(@NonNull View itemView) {
            super(itemView);
            tv1=itemView.findViewById(R.id.textView_name);
            tv2=itemView.findViewById(R.id.textView_puntuacion );
            //para capturar cuando hagamos click en la clase
            itemView.setOnClickListener(this);
        }
```

Ilustración 62 - Código de la clase "AdaptadorPlayer"

```
    public void imprimir(int position){
        //asignamos cada nombre al reciclerview
        tv1.setText(/*players.get(position).getNombre()*/ Player.getInstance().getArray().get(position).getNombre());
        //asignamos cada puntuacion a su nombre en el recyclerview, pasando previamente a string la puntuacion
        String puntuacion = String.valueOf(Player.getInstance().getArray().get(position).getPuntuacion());
        tv2.setText(puntuacion);

    }
```

Ilustración 60 - Foto de la actividad “itemplayer”. Diseño para ReyclerView de “resultados”

Código XML del activity “itemplayer”:

```
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/androi
    xmlns:app="http://schemas.android.com/apk/res-auto"

    android:layout_width="350dp"
    android:layout_height="wrap_content"
    android:background="@drawable/textview_background_intro"
    android:orientation="horizontal"
    android:layout_gravity="center_horizontal"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    >

    <TextView
        android:id="@+id/textView_name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="NAME"
        android:textSize="30sp"
        android:layout_marginLeft="15dp"
        app:layout_constraintHorizontal_chainStyle="spread_inside"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintHorizontal_bias="0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toStartOf="@id/textView_puntuacion"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/textView_puntuacion"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="POINTS"
        android:textSize="30sp"
        android:layout_marginRight="15dp"
        app:layout_constraintHorizontal_chainStyle="spread_inside"
        app:layout_constraintHorizontal_bias="1"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toEndOf="@id/textView_name"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Ilustración 61 - Código XML de la actividad "itemplayer"

Se usa de nuevo un “constraintlyout”, que nos aporta más funcionalidad y restricción para el formato, ya que al ser dentro de un “recyclerview”, se prefiere limitar el espacio de cada jugador y que se vea distinga de forma clara.

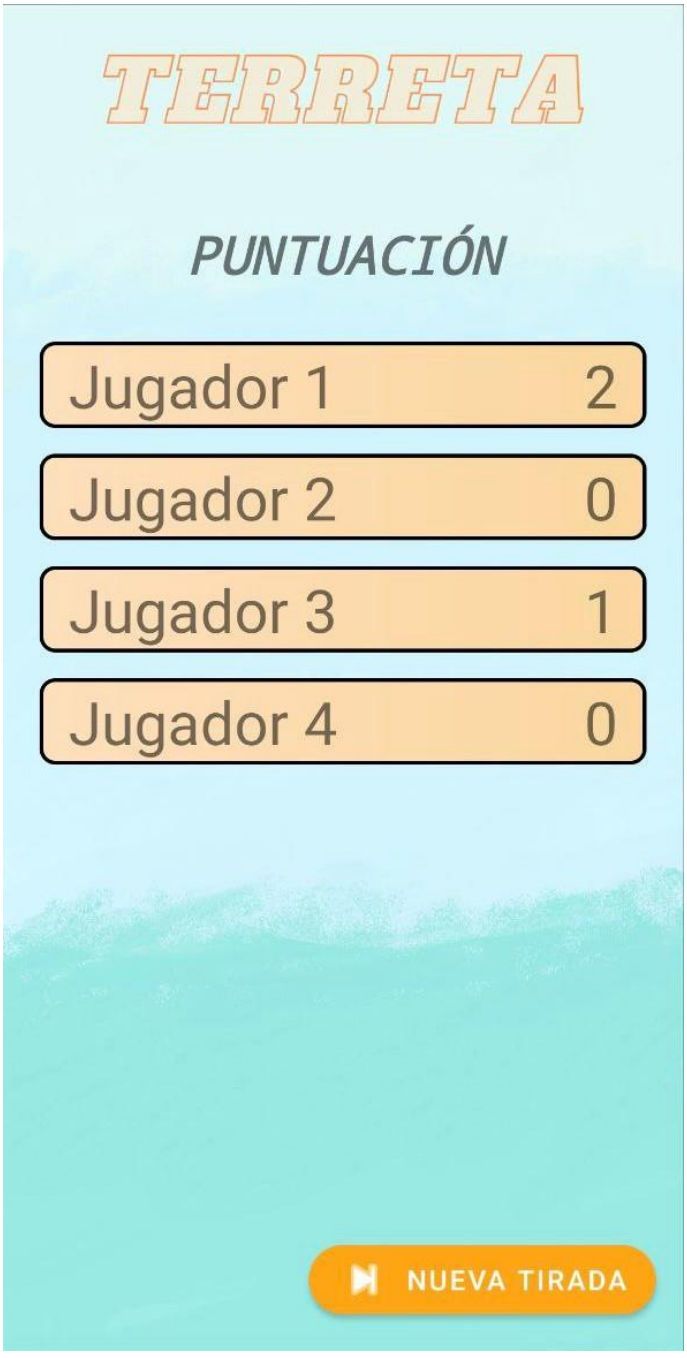
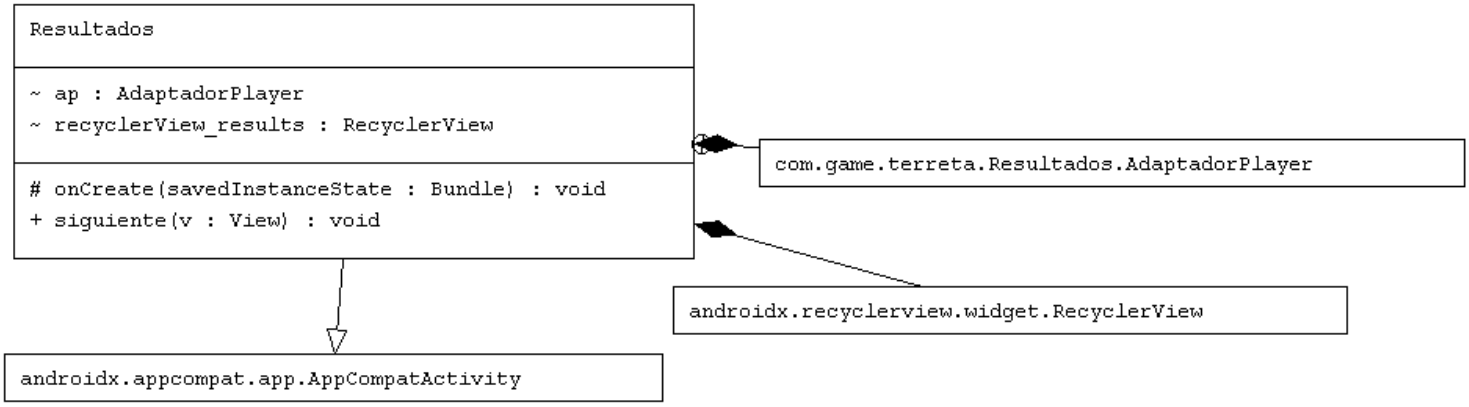
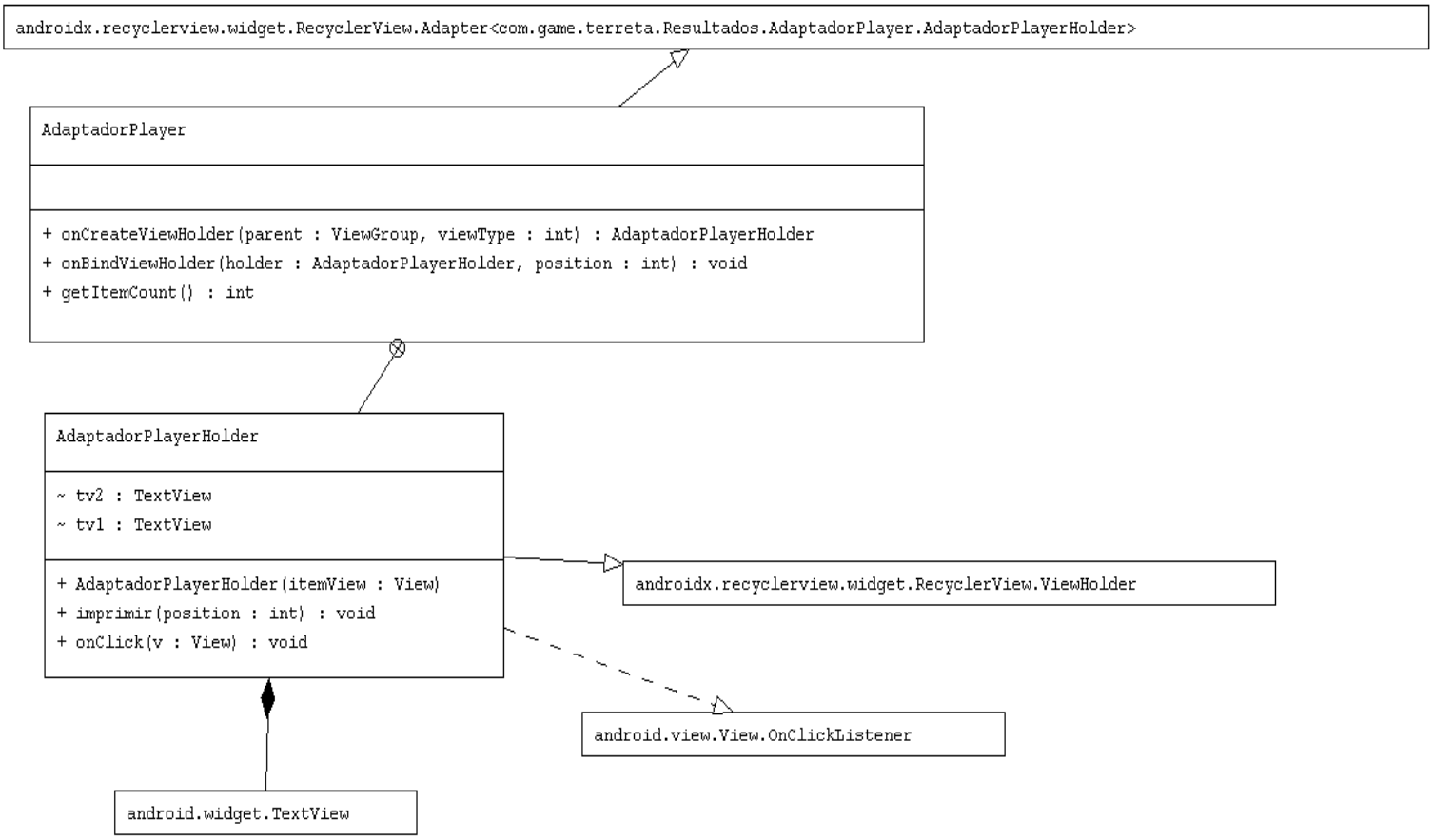


Ilustración 62 - Foto final de la actividad “resultados”

UML de la clase asociada a la actividad:



UML de la clase “AdaptadorPlayer” y “AdaptadorPlayerHolder”:



3.6 Pruebas finales y documentación

Para las pruebas finales se han realizado varios tipos partidas, para probar la funcionalidad correcta de la aplicación, y en caso de detectar algún error, solventarlo antes de terminar el proyecto.

Se busca que la aplicación funcione correctamente sin errores graves aparentes.

Por ejemplo: desde jugar con un solo jugador para probar que no podría jugarse hasta con un número de jugadores elevado para comprobar la capacidad de gestión de la aplicación.

Algunos de fallos encontrados sobre las pruebas realizadas son:

- Gestión errónea en las elecciones de las cartas, tanto para el primer jugador como para el resto de jugadores en las actividades posteriores
    - Error: Al seleccionar una carta (roja o negra), no se “inhabilitaba” la contraria.
- Recorrido de todos los jugadores erróneo en la segunda elección de carta (resto de jugadores excepto el primer jugador escogido aleatoriamente)
    - Error: Durante el recorrido, el primer jugador elegido aleatoriamente, volvía a aparecer en la segunda elección de carta, obviando el control sobre él y podía volver a elegir. Se corrige y se adapta con un “AlertDialog”
- Apariencia de los recyclerview, para las actividades que lo usaban.
    - Error: Recyclerview sin centrar, los nombres de los jugadores no se mostraban correctamente en la pantalla, se “salían” de ella. La lista dinámica no desparecía por debajo de los botones principales, se superponía.

4 CONCLUSIONES Y TRABAJOS FUTUROS

Una vez se ha conseguido la finalización del proyecto, se destacan los conocimientos adquiridos por parte del alumno en una disciplina nueva como es Android, a través del cliente IDE “Android Studio”.

Conocimientos como diseño de interfaces para dispositivos móviles (Android), programación conjunta de lenguajes de programación con el diseño. Estructuración de un proyecto, planificación y desarrollo del mismo; usando metodologías ágiles para grupos con un solo integrante.

Sobre el proyecto finalizado, se han alcanzado todas las etapas y pantallas que se requerían para tener un juego funcional y jugable en una aplicación.

Se ha diseñado desde el principio la introducción de los jugadores que realizarán el juego, así como cada una de las funciones en cada pantalla, destacando la pantalla de tiradas aleatorias y elecciones de carta para cada jugador en la partida iniciada. Este apartado ha supuesto un reto en cuanto al traspaso y gestión de

información entre código programado y la interfaz visual que adquiere la aplicación; el cual puede ser algo lioso cuando no se ha trabajado nunca con esta herramienta y el lenguaje JAVA asociado a Android.

Como mejoras hay una infinidad de mejoras para el proyecto, tales como la introducción de nuevas pantallas y funcionalidades para los jugadores como hacia la los usuarios externos, como por ejemplo compartir los resultados en una red social y dar visibilidad al juego para que nuevos usuarios puedan llegar a conocer el juego y descargarlo.

Con el tiempo se podría realizar un nuevo estudio sobre el ya realizado para ampliar las reglas y funciones del juego, inventando o mejorando las ya existentes, ya que en este proyecto el objetivo principal ha sido disponer del juego de mesa inicial en un entorno digital, portátil y extrapolado a las nuevas generaciones, pero con un enfoque sencillo para todos los públicos.

De cara al futuro también se pretende comercializarlo en las principales tiendas online de “apps” de manera gratuita, pero con anuncios, para así poder financiar un posible proyecto más grande y complejo en el que seguir mejorando el juego.

5 APÉNDICES

5.1 Guía original del Trabajo Fin de Título

1. (Cód.: 21/22-3639) Desarrollo de un Juego de Mesa en Aplicación Móvil

Tutor del TFG: **CARLOS JAVIER OGAYAR ANGUITA**
Modalidad: Proyecto de Ingeniería | Tipo: TFG Específico
Número máximo de estudiantes: 1 (1 asignados)
Idioma: Castellano
Asignado al alumno con DNI:15522404A

Este trabajo consiste en la realización de una aplicación móvil sobre un juego de mesa multijugador. El juego podrá tener unas reglas y dinámicas ya conocidas, o bien ser originales. Además, deberá contar con elementos multimedia para hacerla lo más visual posible. Adicionalmente, podrá incorporar funcionalidades relacionadas con redes sociales, como rankings, capturas de pantalla, etc.

Conocimientos Previos Es conveniente tener conocimientos de desarrollo de aplicaciones web y/o de aplicaciones móviles.

Objetivos del TFG

2. Elegir un juego de mesa multijugador o bien definir las reglas y dinámicas de uno completamente nuevo.
3. Implementar el juego de mesa en una aplicación portable para poder jugar, por ejemplo, en un vuelo o en un viaje de coche largo.
4. Desarrollar la idea propia hasta llegar al máximo número de personas posibles que puedan disfrutar del mismo, es decir, elaborar un plan de difusión y posible comercialización.

Metodología a Desarrollar

1. En caso de elegir un juego existente, elaborar una revisión del estado del arte de las implementaciones actualmente disponibles.
2. Seleccionar y utilizar una metodología basada en Ingeniería del Software para el análisis de requisitos, diseño, implementación, prueba y documentación. Se recomienda utilizar una metodología incremental.
3. Detallar la estimación temporal y de costes de desarrollo.
4. Realizar el desarrollo del nuevo software.
5. Realizar pruebas para el prototipo y documentar los resultados.
6. Redacción de la documentación final requerida, incluyendo un posible modelo de negocio.

Documentos y Formatos de Entrega

- Documentación generada durante el proceso, de acuerdo a las buenas prácticas de la ingeniería del software. Esto se aplicará tanto en los proyectos de Ingeniería como en los trabajos teóricos o experimentales, es decir, siempre que se genere algún tipo de desarrollo, tanto software como webs, scripts, etc.

- Para el formato de documentación de los **Proyectos de Ingeniería** se utilizará la norma **UNE 157801**. Tal y como especifican las normas de estilo de la EPSJ: '*Los Proyectos de Ingeniería deberán presentar una estructura y contenido adaptados a la norma **UNE 157001** - Criterios generales para la elaboración de proyectos - u otra derivada de ésta, en función de la naturaleza del proyecto*'. '*…para el caso de los sistemas de información informatizados (sistemas informáticos, sistemas de información geográfica, etc.) está la norma derivada de la* anterior **UNE 157801** - *Criterios generales para la elaboración de proyectos de sistemas de información*'. Para ello, se puede utilizar también la Norma CCII-N2016-02 (**Norma Técnica para la realización de la Documentación de Proyectos en Ingeniería Informática**) del Consejo General de Colegios Profesionales de Ingeniería en Informática, que incluye e implementa la norma **UNE 157801**.
- Memoria del trabajo, incluyendo manuales y anexos, en formato PDF.
- Código fuente completo y ejecutables del prototipo o software desarrollado.
- Documentos y archivos adicionales a los que se haga mención expresa en la memoria.
- Vídeo de demostración de la aplicación o de la experimentación realizada.
- Anexos para la presentación del trabajo:
    1. Informe favorable del tutor.
    2. Autorización de publicación, si procede.

5.2 Instalación y configuración del sistema

Para la instalación del juego, tendremos un fichero con extensión ".apk", el cual tendremos que instalar en nuestro smartphone y seguir los siguientes pasos:

1- Descargar el ".apk", normalmente lo encontraremos en "/descargas"

2- Instalar aplicación

Posibles errores/inconvenientes:

Es probable que muchos de los usuarios no puedan instalar aplicaciones que no sean descargadas desde la "Play Store", y tengamos un mensaje de error de seguridad como este:

Ilustración 63 - Captura error de intalacion

Presionaremos en “Ajustes”, y nos llevará a la siguiente pantalla:

Ilustración 64 - Captura solución de error

Activaremos esa función “Autorizar de esta fuente” con el interruptor, debe quedar asi:

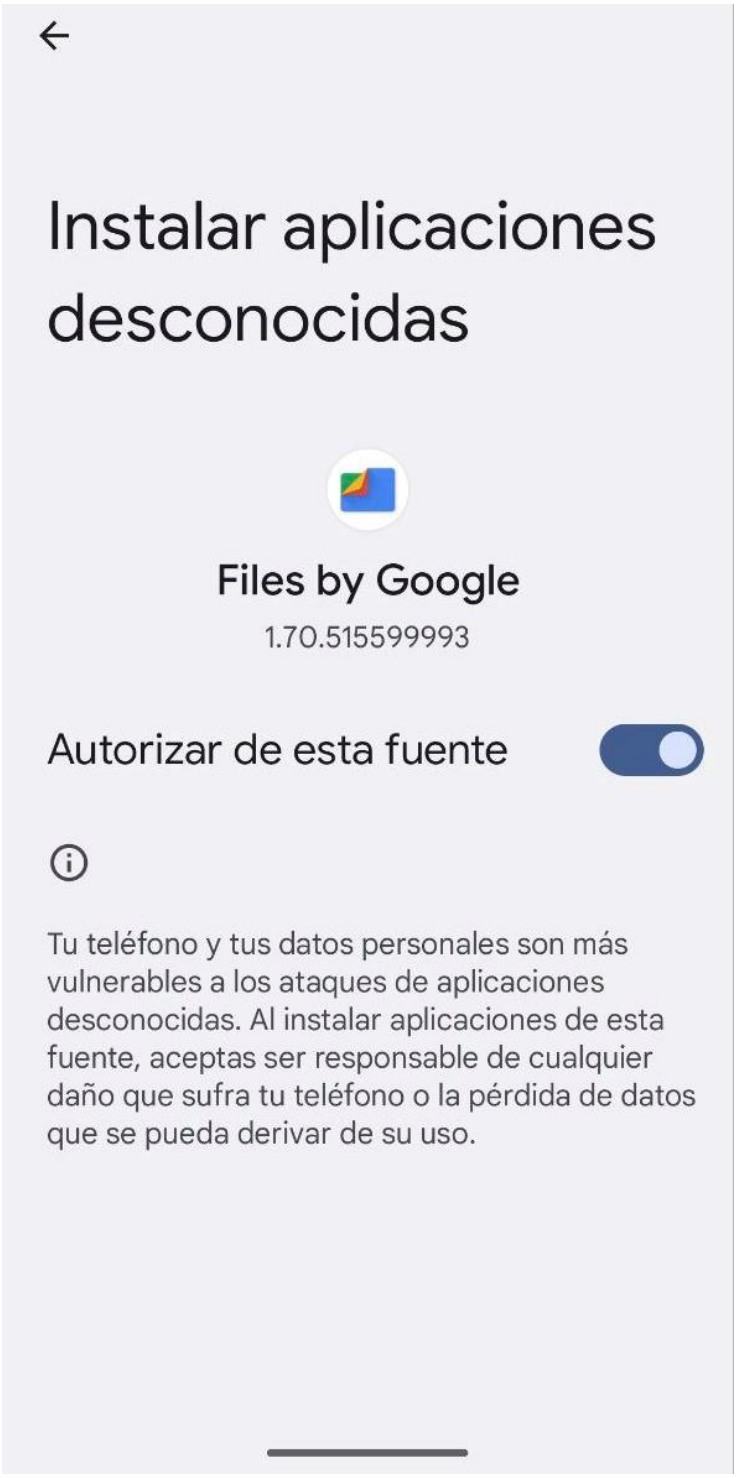


Ilustración 65 - Captura solución error 2

Una vez realizado esto, volveremos donde se encuentra el archivo ".apk" y lo podremos instalar sin problema.

5.3 Manuales de usuario

Para tener información sobre el funcionamiento del juego, por favor consultar el fichero pdf “Instrucciones.pdf”

6 BIBLIOGRAFÍA

[1] Sylvain HEBUTERNE. (2018) *Desarrolle una aplicación android. Programación en Java con android studio.* Eni-

[2] Ángel Arias. (2015). *Aprende a Programar para Android*. CreateSpace Independent Publishing Platform.

[3] José Dimas Luján Castillo. (2017). *ANDROID STUDIO Aprende a desarrollar aplicaciones.* RCLIA|#RC LIBROS.

[4] Jesús Tomás. (2019). *El gran libro de Android.* MARCOMBO S.A;

[5] Adam Gerber, Clifton Craig, David Selvaraj. (2018). *Learn Android Studio. Build Android Apps Quickly and Effectively.* APRESS.

[6] Paul Deitel, Harvey Deitel. (2013) *JAVA. COMO PROGRAMAR.* Pearson

[7] Pressman, R. (2010). *Ingeniería del Software.* McGraw-Hill. Pearson

[8] **<https://www.codeproject.com/Tips/1060314/Simple-Singleton-Pattern-in-Android>**

[9] **http://www.androidcurso.com/index.php/**

[10] "Curso aprender a programación Java desde cero" < https://www.aprenderaprogramar.com/index.php?option=com_content&view=category&id=68&Itemid=188>

[11] **<https://developer.android.com/training/basics/firstapp?hl=es-419>**

[12] Códigos de los colores: <https://encycolorpedia.es/>