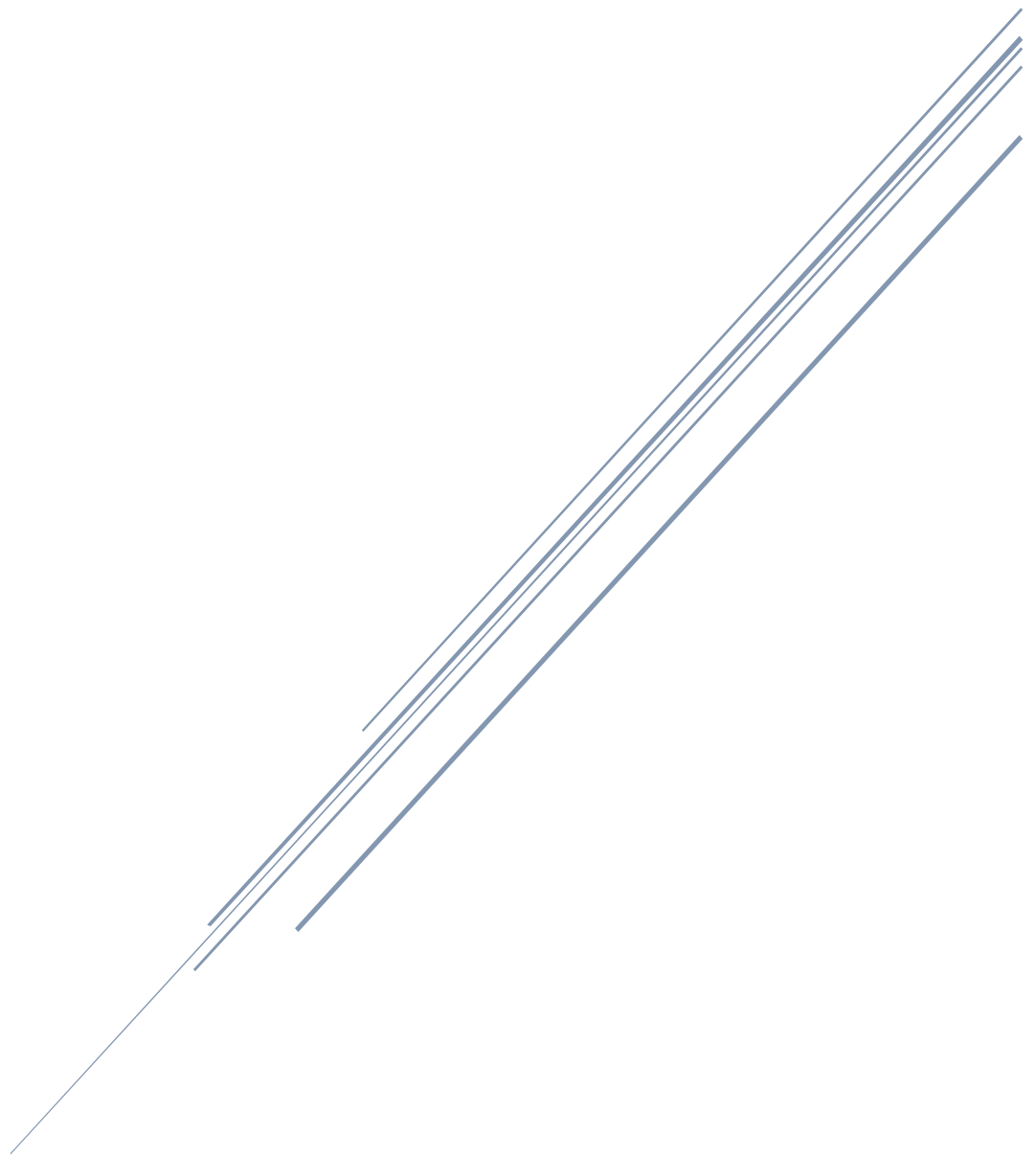


RULE THE WORLD

Trabajo fin de grado 2014

Egoitz Puerta Beldarrain



Escuela universitaria de ingeniería técnica industrial de Bilbao
Grado en ingeniería informática de gestión y sistemas de
información

Resumen

Rule the World es una aplicación para móviles Android. Consiste en introducir al jugador en una realidad aumentada, mediante el uso de su localización, debe de recoger diferentes objetos para darles diferentes usos, como llevarlos equipados, usarlos para construir otros objetos o enviárselos a amigos.

En el siguiente documento se muestra el completo desarrollo de este proyecto, como se ha realizado la gestión, en que partes se ha dividido, la planificación que se ha llevado para realizar el trabajo, el análisis que se hizo de la aplicación, junto con su diseño, como se ha realizado el desarrollo y las pruebas.

Este proyecto ha servido para afianzar conocimientos adquiridos a lo largo del grado, como el desarrollo de bases de datos, seguridad y arquitecturas y algoritmos software. Pero también ha servido para aprender nuevas cosas, como programar para un sistema diferente, utilizar elementos poco vistos en el grado, como la geolocalización y los mapas.

Índice de contenidos

Resumen.....	1
Índice de contenidos	2
Índice de tablas.....	4
Índice de ilustraciones.....	5
Introducción	7
Gestión del proyecto	8
Objetivos	8
Alcance	8
Planificación temporal	9
Riesgos.....	13
Método de trabajo	14
Herramientas a utilizar	14
Antecedentes	16
Captura de requisitos	17
Definición de los requisitos.....	17
Jerarquía de actores	17
Casos de uso	18
Modelo de dominio	18
Descripción de la arquitectura.....	19
Análisis y diseño	20
Transformación de modelo de dominio a Base de datos.....	20
Diagrama de clases	21
Arquitectura del sistema	24
Diagramas de secuencia	25
DESARROLLO	26
VERIFICACIÓN Y EVALUACIÓN	29
CONCLUSIONES Y TRABAJO FUTURO	30
BIBLIOGRAFÍA	32
ANEXO I.- CASOS DE USO EXTENDIDOS	33
Caso de uso: Identificarse	33
Caso de uso: Registrarse	34
Caso de uso: Moverse por el mapa.....	37
Caso de uso: Fabricar objeto	38
Caso de uso: Ver el inventario	39

Caso de uso: Ver amigos	41
Caso de uso: Ver estado y equipo.....	42
Caso de uso: Combatir enemigo	43
ANEXO II.- DIAGRAMAS DE SECUENCIA	45
ANEXO III.- PLAN DE PRUEBAS	52
ANEXO IV.- MANUAL DE USUARIO	59
Instalación	59
Identificación y registro	59
Jugar	59

Índice de tablas

Tareas EDT 1	8
Tareas EDT 2	9
Planificación	10
Plan de Pruebas	52

Índice de ilustraciones

Diagrama EDT	8
Calendario 1	11
Calendario 2	11
Diagrama de Gantt 1	12
Diagrama de Gantt 2	12
Diagrama de Gantt 3	13
Diagrama de Gantt 4	13
Android studio	14
Notepad++	14
MySQL	14
phpMyAdmin	14
SQLite	15
Dia	15
UModel	15
Moqups	15
Word	15
Excel	15
Git	15
Dropbox	15
WinSCP	15
Casos de uso	18
Modelo de domino	19
Diagrama Base de datos Servidor	20
Diagrama Base de datos App	21
Diagrama de clases 1	21
Diagrama de clases 2	22
Diagrama de clases 3	23
Diagrama de clases 4	23
Diagrama de clases 5	24
Pantalla 1	27
Pantalla 2	27
Pantalla 3	27
Pantalla 4	27
Pantalla 5	27
Pantalla 6	27
Caso de usos extendidos	33
CDUE: Identificarse	33
Prototipo 1	34
Prototipo 2	34
Prototipo 3	34
CDUE: Registrarse	34
Prototipo 4	36
Prototipo 5	36
Prototipo 6	36
Prototipo 7	36
CDUE: Moverse en el mapa	37
Prototipo 8	38

Prototipo 9	38
CDUE: Fabricar objetos	38
Prototipo 10	39
Prototipo 11	39
Prototipo 12	39
CDUE: Ver inventario	39
Prototipo 13	40
Prototipo 14	40
CDUE: Ver amigos	41
Prototipo 15	42
Prototipo 16	42
CDUE: Ver estados y equipos	42
Prototipo 17	43
Prototipo 18	43
CDUE: Combatir enemigos	43
Prototipo 19	44
Prototipo 20	44
Prototipo 21	44
DS: Lanzamiento de aplicación	45
DS: FragmentMapa	46
DS: Inventario	47
DS: Equipo	48
DS: Combinaciones	49
DS: Amigos	50
DS: Combates	51
Manual 1	59
Manual 2	60
Manual 3	60
Manual 4	60
Manual 5	61
Manual 6	61
Manual 7	61
Manual 8	62

Introducción

Descripción del trabajo

Se pretende realizar un juego de realidad aumentada para plataformas móviles en el cual, el jugador deberá de elegir un bando y buscar objetos distribuidos por el mundo para poder conseguir mejor equipamiento.

Durante el desarrollo del juego, al jugador se le mostrará un mapa real de su zona donde se depositarán diversos objetos, que llamaremos materias, y que deberán de ser recogidos por el jugador para coleccionarlos, crear objetos de mayor valor, venderlos o perderlos, es decir, dejarlos de nuevo en el mapa. Estas materias se localizarán de manera aleatoria y cercana al jugador, y el valor del objeto encontrado dependerá del nivel del personaje y la franja horaria en la que juegue.

Las materias podrán ser utilizadas como materia prima para fabricar otros objetos de mayor valor. Estos nuevos objetos podrán ser utilizados de nuevo como materia prima para objetos más elaborados, equiparlos o perderlos para dejarlos a amigos en el mapa.

El jugador podrá equiparse determinadas materias que varíaran sus atributos, como la fuerza, la defensa o la velocidad. Estos atributos podrán aumentarse según el personaje suba de nivel, es decir, realizando con éxito ciertas acciones.

Se ideará un sistema de batalla, basado en atributos, donde el jugador podrá elegir qué acciones hacer durante esta para determinar el éxito o pérdida de la batalla.

Mediante el dinero recogido por el personaje a lo largo de las misiones se podrá comprar objetos que le darán ventaja al personaje durante determinado periodo de tiempo, a los que llamaremos bonificadores. Estos objetos harán al personaje inmune a los ataques durante un tiempo, hará que no se tenga que acercarse a los objetos para poder recogerlos o podrá ver que contiene un objeto antes de acercarse a recogerlo, entre otras opciones.

Como objetivos de realización secundarios se dejará los bonificadores previamente citados, así como una tienda para poder obtener dinero en el juego y poder así conseguir más bonificadores. También hay como objetivos secundarios la realización de una página web de publicidad para el juego, así como incrementar la capacidad de gráficos en el juego, para que el jugador pueda personalizar su personaje. Los sonidos y música que acompañen al juego será también objetivo secundario. Por último se podría incluir una pequeña red social, donde los usuarios puedan agregar conocidos y poder enviarse mensajes y retos que deberán de superar.

Antecedentes

Como antecedente existe la aplicación Ingress de NianticLabs, que sugiere algo similar, basándose en la geolocalización del teléfono, calcula elementos cercanos con los que el jugador puede interaccionar. El usuario tiene que recoger energía para poder “hackear” los portales y añadirlos a su bando, generando puntos y energía en su beneficio.

Razones de elección del trabajo, motivación y experiencia previa

La elección de este trabajo fue realizada, además de por el auge que están teniendo las tecnologías móviles en nuestra vida, por preferencias personales, debido a que me gustaría dedicarme al sector de los videojuegos.

La idea surgió a partir de una idea propuesta en el aula de tecnologías móviles de EUITI de Bilbao.

La experiencia previa en el aula de tecnologías móviles se limita a la exploración de este terreno y se pretende explorar en profundidad esta área.

Gestión del proyecto

Objetivos

El objetivo de este proyecto consiste en la realización de un videojuego de realidad aumentada para dispositivos móviles que llame la atención al usuario y le mantenga entretenido durante largos periodos de tiempo y que a la vez pueda interactuar con el medio que le rodea.

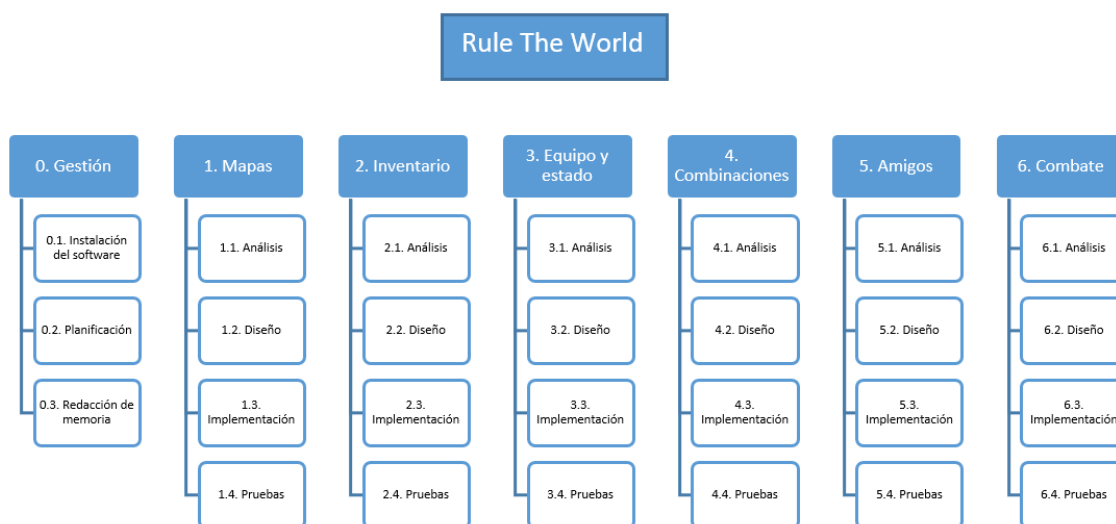
Como objetivos secundarios se plantea la realización de una página web donde se pueda realizar marketing del videojuego, así como dar información y estadísticas a los usuarios sobre sus avances y amigos en el juego. Además de una tienda para obtener beneficios dentro del juego. Como objetivos secundarios se propone la inclusión de medios para mejorar la interacción entre el usuario y el videojuego.

Alcance

Las tareas a desarrollar para llevar con éxito los objetivos se dividieron en función de las funcionalidades que estos objetivos conllevan. Con ello, se pierde la tradicional forma de trabajar, y se opta por una metodología ágil, donde las tareas de diseño o desarrollo se realizan tantas veces como funciones haya en los objetivos. Además se añadieron tareas de gestión y documentación.

En las siguientes tareas los recursos personales, así como la responsabilidad de las tareas recaen exclusivamente sobre el autor, Egoitz Puerta.

A continuación se presenta el diagrama de la estructura de descomposición del trabajo.



La descripción de cada tarea se detalla en las siguientes tablas. La duración de las tareas es estimada, teniendo en cuenta la duración de otras tareas semejantes en proyectos anteriores.

0.1 Instalación y configuración del software necesario	
Duración	3 horas.
Descripción	Instalación y configuración del software necesario para realizar el proyecto.
Antecedentes	
Salida	Software totalmente configurado.
0.2. Planificación	

Duración	5 horas
Descripción	Establecer tareas, con su estimación temporal y organizarlas temporalmente para su realización a tiempo.
Antecedentes	
Salida	EDT y planificación temporal.
0.3. Redacción de la memoria final	
Duración	50 horas
Descripción	Reunir todos los documentos desarrollados a lo largo de todo el proyecto, unificarlos y extraer conclusiones de los resultados.
Antecedentes	Todas las pruebas realizadas.
Salida	Memoria final.

Las siguientes tareas se realizarán por cada función a implementar, véase: Mapas, inventario, equipo y estado, combinaciones, amigos y combates. Como resultado, cada tarea actualiza el documento correspondiente hasta obtener la versión final al finalizar la última de las tareas.

x.1. Análisis	
Duración	5 horas
Descripción	Recopilar las características que debe de cumplir el software para la tarea que va a desarrollarse y como presentarlo de cara al usuario.
Antecedentes	Y 0.2. y análisis anteriormente realizados.
Salida	Casos de usos y modelo de dominio.
x.2. Diseño	
Duración	6 horas
Descripción	Establecer la secuencia lógica de acontecimientos necesarios para la correcta implementación del análisis.
Antecedentes	x.1. y diseños anteriormente realizados.
Salida	Diagrama de clases, diagramas de secuencia.
x.3. Implementación	
Duración	15 horas.
Descripción	Codificación del diseño.
Antecedentes	x.2. y código realizado anteriormente.
Salida	Código.
x.4. Pruebas	
Duración	7 horas.
Descripción	Comprobación y rectificación del código realizado para asegurarse del correcto funcionamiento de las funciones analizadas.
Antecedentes	x.3. y pruebas realizadas anteriormente.
Salida	Prototipo de aplicación funcional.

Planificación temporal

A continuación se muestra una tabla con las tareas a realizar, su duración y fechas de inicio y fin estimadas. Para calcular las fechas se desestimaron los fines de semana y días festivos y la jornada media es de 5 horas. Dada la metodología seleccionada, para pasar de fase se ha de concluir totalmente la fase anterior. Además, por recursos de personal, no se puede asumir más de una tarea simultánea.

TAREA	DURACIÓN	FECHA INICIO	FECHA FIN
0.1. Configuración de software	3	02/06/2014	02/06/2014
0.2. Planificación	5	03/06/2014	03/06/2014
0.3. Redacción de memoria	50	03/06/2014	29/08/2014
1. Mapas	34	04/06/2014	12/06/2014
1.1. Análisis	5	04/06/2014	04/06/2014
1.2. Diseño	7	05/06/2014	06/06/2014
1.3. Implementación	15	06/06/2014	11/06/2014
1.4. Pruebas	7	11/06/2014	12/06/2014
2. Inventario	34	13/06/2014	23/06/2014
2.1. Análisis	5	13/06/2014	13/06/2014
2.2. Diseño	7	16/06/2014	17/06/2014
2.3. Implementación	15	17/06/2014	20/06/2014
2.4. Pruebas	7	20/06/2014	23/06/2014
3. Estado y Equipo	34	24/06/2014	02/07/2014
3.1. Análisis	5	24/06/2014	24/06/2014
3.2. Diseño	7	25/06/2014	26/06/2014
3.3. Implementación	15	26/06/2014	01/07/2014
3.4. Pruebas	7	01/07/2014	02/07/2014
4. Combinación	34	03/07/2014	11/07/2014
4.1. Análisis	5	03/07/2014	03/07/2014
4.2. Diseño	7	04/07/2014	07/07/2014
4.3. Implementación	15	07/07/2014	10/07/2014
4.4. Pruebas	7	10/07/2014	11/07/2014
5. Amigos	34	14/07/2014	22/07/2014
5.1. Análisis	5	14/07/2014	14/07/2014
5.2. Diseño	7	15/07/2014	16/07/2014
5.3. Implementación	15	16/07/2014	21/07/2014
5.4. Pruebas	7	21/07/2014	22/07/2014
6. Combate	34	23/07/2014	04/08/2014
6.1. Análisis	5	23/07/2014	23/07/2014
6.2. Diseño	7	24/07/2014	28/07/2014
6.3. Implementación	15	28/07/2014	01/08/2014
6.4. Pruebas	7	01/08/2014	04/08/2014

A Continuación se muestra la anterior tabla en una vista en calendario, donde cada color representa un tipo de tarea, siendo verde las tareas de gestión, rojas tareas de análisis, amarillas tareas de diseño, azules tareas de implementación y el naranja pruebas.

Junio						
L	M	X	J	V	S	D
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

Julio						
L	M	X	J	V	S	D
7	1	2	3	4	5	6
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Agosto						
L	M	X	J	V	S	D
4	5	6	7	1	2	3
11	12	13	14	8	9	10
18	19	20	21	15	16	17
25	26	27	28	22	23	24
				29	30	31

Septiembre						
L	M	X	J	V	S	D
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

Gestión	Análisis	Diseño	Implementación	Pruebas
Entrega de memoria			Presentación del trabajo	

En el siguiente calendario se muestran la planificación en una vista de calendario, donde los colores muestran las funciones que se van a desarrollar.

Junio						
L	M	X	J	V	S	D
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

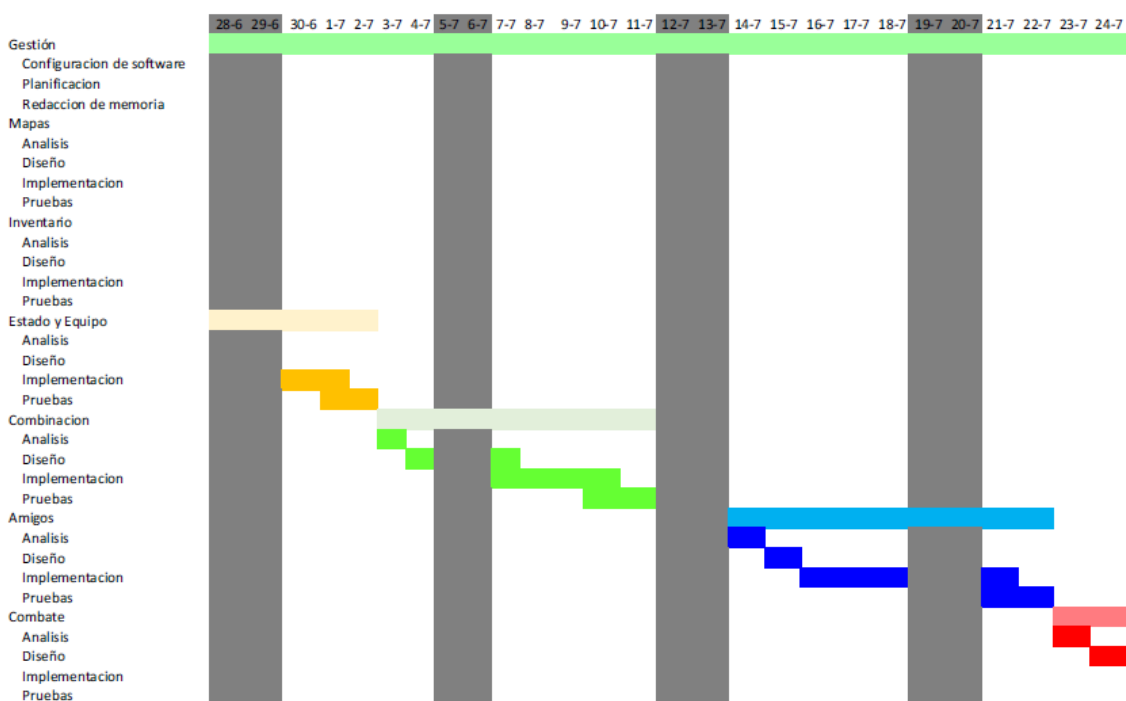
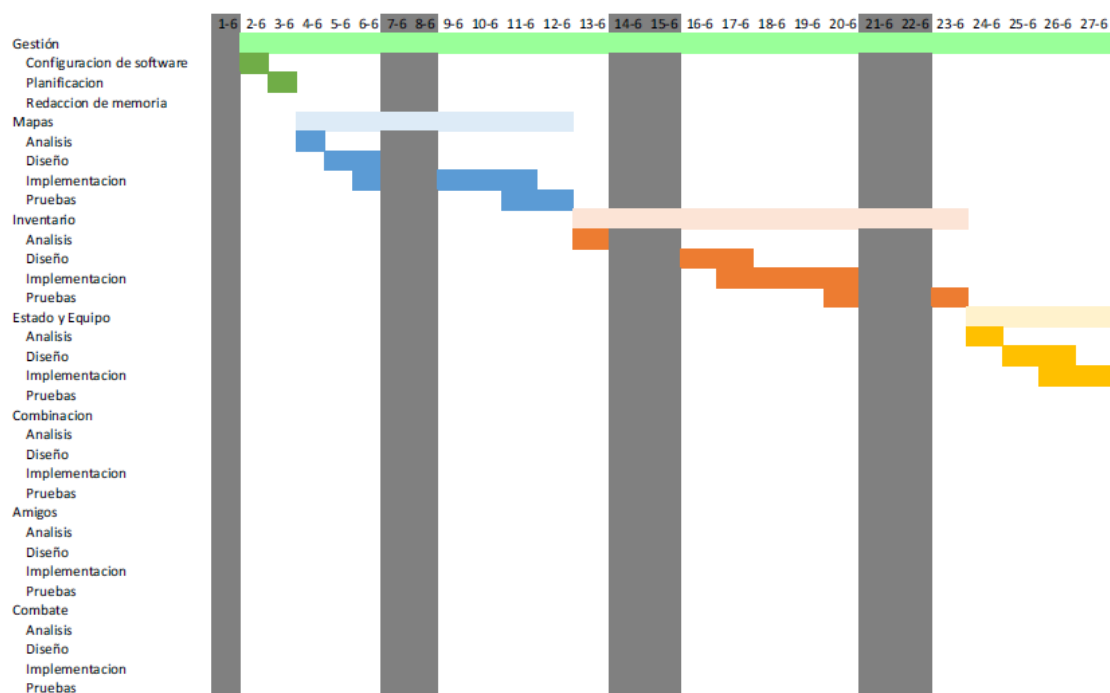
Julio						
L	M	X	J	V	S	D
7	1	2	3	4	5	6
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

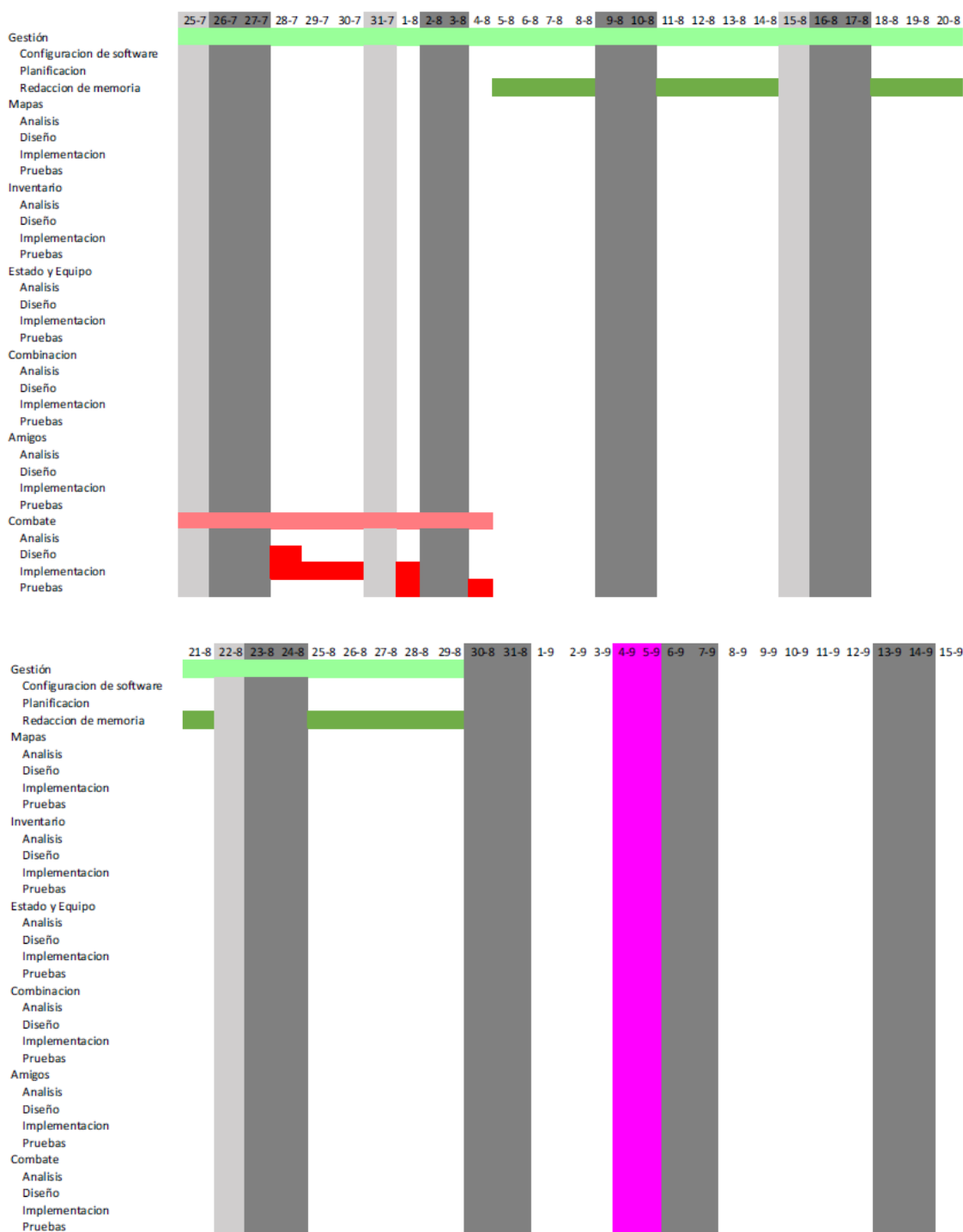
Agosto						
L	M	X	J	V	S	D
4	5	6	7	1	2	3
11	12	13	14	8	9	10
18	19	20	21	15	16	17
25	26	27	28	22	23	24
				29	30	31

Septiembre						
L	M	X	J	V	S	D
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

Gestión	Mapas	Inventario	Equipo y estado	Combinaciones
Amigos	Combates	Entrega de memoria	Presentación del trabajo	

Por último se muestra un diagrama de Gantt con la disposición temporal de cada tarea y su duración de manera que se muestre lineal.





Las horas estimadas para realizar y completar el proyecto son de 262 horas. Estimando como precio de un analista-programador de 20€ por hora, el proyecto podría ser valorado en 5240€, que dividido entre los tres meses que ha durado el proyecto hace una suma de 1747€ al mes.

Riesgos

A continuación se exponen la lista de riesgos que se han contemplado y las medidas que se han tomado para hacer que los riesgos afecten en la menor medida al proyecto.

- Pérdida o daños del equipo: El Workspace y la documentación se encuentra en una carpeta que, gracias a Dropbox, se mantiene sincronizado con la nube, permitiendo el acceso a los datos desde cualquier otro equipo, o la web. Se procurará un nuevo equipo en el que poder continuar el trabajo lo antes posible.
- Pérdida o daños en los datos: Mediante Dropbox, se realizan copias de seguridad de las versiones de los documentos, junto con el código fuente, pudiendo recuperarlas fácilmente. Además se preparará un repositorio git, donde se subirán versiones completas y funcionales del software periódicamente, fácilmente recuperables.
- Pérdida de conexión de internet: Actualmente existen multitud de puntos de conexión a internet, en caso de que sucediese y fuera necesario la conexión, se buscará una alternativa de conexión.
- Enfermedad del desarrollador: Se procurará descanso y una recuperación rápida. Tras la recuperación se realizará un esfuerzo superior para recuperar las horas atrasadas.

Método de trabajo

Como ya se ha mencionado antes, se utilizará una metodología ágil para el desarrollo del proyecto, separándolo en fases, que tendrán como resultado un fragmento funcional del proyecto, al cual, en cada fase, se le irá añadiendo más y más funcionalidad hasta su completitud.

Cada fase se compondrá de un análisis, donde se estudiarán los casos necesarios para que esa fase este funcional; un diseño, donde se realizará una metodología de programación que encaje con el resto de la aplicación y la plataforma donde se desarrollará; una implementación, que convertirá el diseño previo en código funcional; y por ultimo unas pruebas, que servirán para corregir los posibles defectos del código.

Herramientas a utilizar

Desarrollo principal

Para realizar el proyecto se usarán las siguientes herramientas:



Android Studio, herramienta basada en IntelliJ, será la herramienta oficial de desarrollo en Android cuando esté acabada completamente.



Notepad++, editor de texto con resaltador de sintaxis, para desarrollo de ficheros PHP.

Bases de datos

Para el almacenamiento de datos y su gestión se utilizan las siguientes herramientas:



MySQL es el motor de base de datos alojado en el servidor.

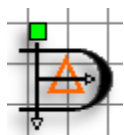
phpMyAdmin es el gestor que nos permitirá, a través de una página web, manejar la base de datos de manera remota.



SQLite es el motor de base de datos que Android maneja internamente en sus aplicaciones.

Diagramas e imágenes

Para crear y diseñar todos los diagramas se usarán las siguientes herramientas:



Dia, como editor de diagramas de modelo de dominio, de clases y de bases de datos.



uModel para editar y crear diagramas de secuencia.



Moqups como editor de prototipos de interfaces gráficas.

Documentación

Para realizar la documentación se han usado las siguientes herramientas:



Microsoft Office Word para redactar y corregir la documentación.



Microsoft Office Excel para realizar los diagramas de Gantt.

Backup y compartición de archivos

Para la realización de los planes de riesgo y compartir los archivos se usaron estas herramientas:



Git, con su interfaz de usuario y github.com para almacenamiento del repositorio para realizar el control de versiones y backup.



Dropbox, semejante a Git, permite tener los documentos en la nube para acceder a ellos en cualquier momento y dispositivo.



WinSCP para enviar los documentos PHP al servidor.

Antecedentes

Se quiere realizar una aplicación con realidad aumentada, que implique al usuario una movilidad e interacción con su entorno.

Actualmente la gran mayoría de los videojuegos y plataformas para estos están diseñados para jugar desde un sofá o silla, con un mando o teclado, no requieren al usuario un esfuerzo físico. La consola de Nintendo, Nintendo Wii y posteriormente Nintendo Wii U, comienzan a exigir cierto esfuerzo físico, estas consolas demandan al usuario una movilidad considerable para poder controlarlas debidamente, haciendo más participe al usuario. La nueva generación de consolas, como Xbox One o Playstation 4 han tratado de integrar el ambiente del usuario dentro de la consola, mediante micrófonos para captar comandos de voz o cámaras para visualizar las acciones del usuario. Sin embargo estas consolas no han conseguido que sus usuarios se levanten del sofá.

Las tecnologías móviles, por el contrario, han experimentado un gran auge, ya que prácticamente cada persona tiene un teléfono o Tablet que lleva consigo, y permiten que el usuario se mueva y esté conectado, además de interactuar con su localización, usar la cámara de fotos o video e incluso conectarse a internet.

Entre todos los sistemas operativos móviles en el mercado, el 88% de los dispositivos llevan Android, lo que quiere decir que si realizamos el proyecto con soporte para Android, casi nueve de cada diez personas podrían llegar a usar nuestra aplicación, frente a otros sistemas como iOS, Windows Phone o Blackberry.

Otro de los grandes motivos por los que decantarse por Android es la gran cantidad de documentación existente para su desarrollo, así como la comunidad que existe de soporte para este. Además, para poder desarrollar aplicaciones para iOS, la otra alternativa, se requiere de un ordenador que lleve iOS, como los ordenadores Mac de Apple, cosa que no se dispone.

Respecto a la aplicación, Android dispone de varios juegos que basan su experiencia en la localización del dispositivo y por ende, del jugador. Algunos de ellos, consiste en huir o perseguir en un lugar cercano, usando como base el mapa. El más famoso de estos juegos puede ser Ingress diseñado por NianticLabs. Consiste en capturar energía a través de portales, situados en lugares emblemáticos de las ciudades, y que los usuarios deben de hackear y convertir para su propio equipo.

Nuestra aplicación pretende realizar algo parecido, pero aportar algo diferente. En nuestro caso, el jugador deberá de recoger diversos objetos que se encuentran en el área cercana a este, para poder con ello, mejorar sus propios objetos que podrá equiparse para mejorar sus estadísticas como la fuerza o la vida. Mejorando esto, hará más difícil que otros usuarios puedan derrotarles y podrán enfrentarse a diversas criaturas para poder obtener mejores armas.

Captura de requisitos

El proyecto estará orientado para adolescentes que posean un Smartphone, por ello, el juego deberá de ser atractivo para los usuarios y que les mantenga entretenidos durante largos periodos de tiempo. El juego deberá de tener acceso a la localización del dispositivo, así como a internet para poder actualizarse. También debe de ser económico energéticamente hablando, ya que gastar la batería rápidamente desvirtuaría completamente la idea de que sea un juego móvil.

Definición de los requisitos

El objetivo del juego es obtener la mayor cantidad de experiencia posible. La experiencia se obtiene de:

- Recoger objetos del mapa.
- Combinar objetos para crear nuevos objetos.
- Luchar contra monstruos.

Al obtener más experiencia subiremos de nivel, lo que nos otorgará obtener mejores objetos. Esto nos ayudará a poder defendernos y combatir mejor a los monstruos ya que podremos equiparnos estos objetos para aumentar nuestras estadísticas. También podremos ayudar a nuestros amigos, dejándoles en el mapa para que ellos puedan obtenerlos.

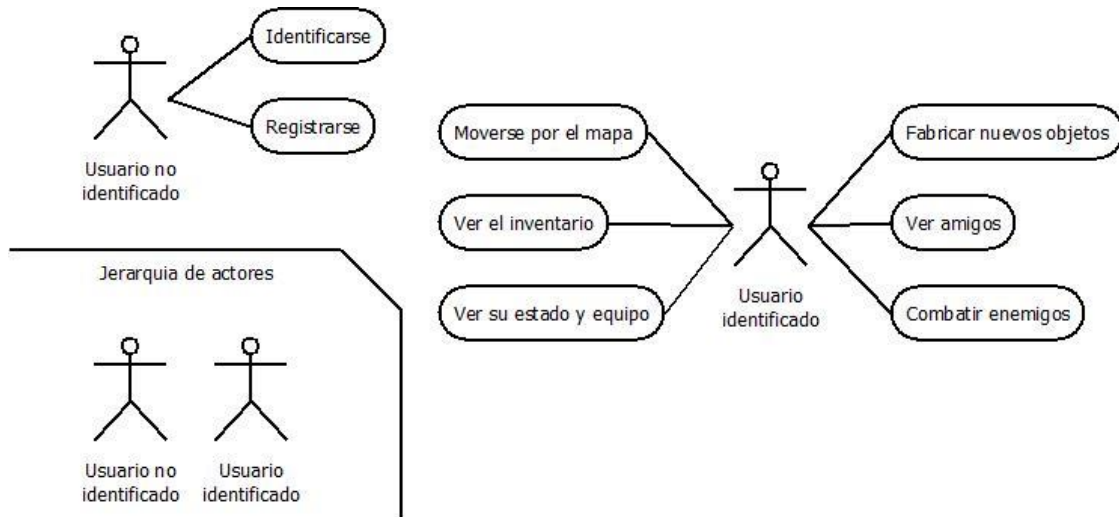
Algunas de las condiciones necesarias para avanzar en el juego son:

- Para recoger una materia, el usuario deberá de desplazarse hasta que la materia este lo suficientemente cerca.
- Al dejar una materia para un amigo, el amigo deberá de acercarse hasta ella para poder recogerla, pero solo podrá hacerlo la siguiente vez que se conecte a la aplicación.
- Al combinar dos objetos, estos dos objetos se restarán del inventario del jugador, independientemente del éxito de la combinación. Si esta combinación es correcta, el objeto resultante se añadirá al inventario del jugador.
- Será el usuario el que decida empezar una pelea, pero el enemigo será elegido al azar entre la lista de enemigos disponibles.
- El daño a realizar o recibido depende de los índices de ataque y defensa. Por ejemplo, si se va a atacar a un enemigo, el daño realizado será la diferencia entre el índice de ataque del jugador menos el índice de ataque del enemigo, si este resulta positivo.
- Al optar por defenderse, ese turno se gozará del doble del índice de defensa, haciendo al enemigo más difícil la posibilidad de herir al jugador.

Jerarquía de actores

Los actores involucrados en el proyecto son:

- Usuario no identificado: Aquel que no ha ingresado unas credenciales para confirmar su identidad, o lo ha hecho de manera errónea.
- Usuario identificado: Aquel usuario que se ha identificado correctamente y tiene acceso al juego.



Casos de uso

El usuario no identificado posee dos casos de uso:

- **Identificarse:** El usuario podrá introducir unas credenciales y si son validadas correctamente, se le permitirá acceder al resto de la aplicación, convirtiendo al actor en un usuario identificado.
- **Registrarse:** Se le permite al usuario introducir unas credenciales para poder tener un futuro acceso a la aplicación. Este registro no implica una identificación. Si el usuario desea entrar en la aplicación deberá de identificarse posteriormente.

El usuario identificado cuenta con varios casos de uso:

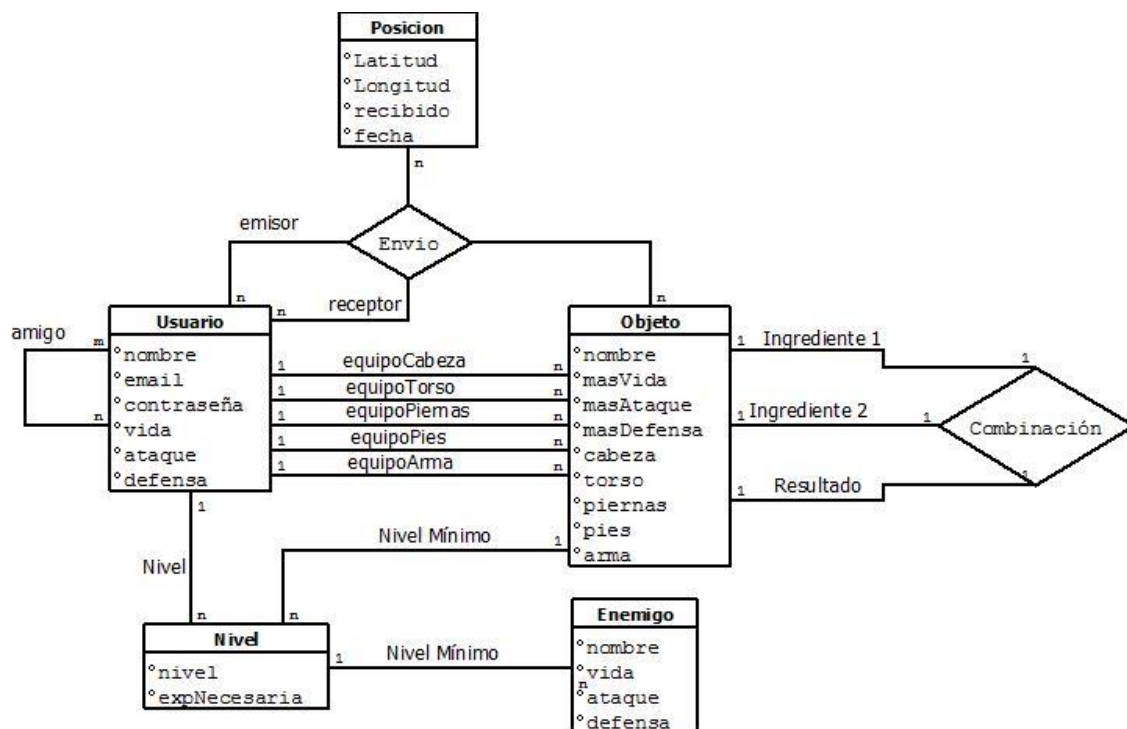
- **Moverse por el mapa:** El usuario puede desplazarse por su entorno, cambiando la posición y de esta manera obtener nuevos objetos.
- **Ver el inventario:** El usuario puede consultar todos los objetos contenidos en su inventario, así como ver qué características poseen los objetos y dejarlos en el mapa.
- **Ver su estado y equipo:** El usuario puede consultar su estado, como la vida, la fuerza o el nivel en el que se encuentra; y su equipo, para poder equiparse diferentes objetos.
- **Fabricar nuevos objetos:** El usuario podrá seleccionar hasta un máximo de dos objetos para combinarlos y crear nuevos objetos.
- **Ver amigos:** El usuario podrá ver su lista de amigos y enviarle alguno de los objetos que tenga en su inventario, así como añadir nuevos amigos.
- **Combatir enemigos:** El usuario podrá entrar en combate con un enemigo elegido al azar y tendrá que pelear contra él.

Los casos de uso extendido se encuentran en el Anexo I de este mismo documento.

Modelo de dominio

La aplicación debe de almacenar los datos de los diferentes objetos de los que se dispone, la información de los usuarios, tanto los que permitan el acceso a la aplicación, como los que informan de las estadísticas y los amigos de cada uno de los usuarios. Además, una colección de objetos, monstruos y un sistema de niveles.

Para almacenar todos estos datos se ideó el siguiente modelo de dominio.



Descripción de la arquitectura

La aplicación corresponderá a diferentes tipos de arquitecturas de software:

- Arquitectura cliente-servidor, donde el cliente, todos los dispositivos móviles, piden datos y confirmaciones al servidor, mediante una capa de red. Esto permite que se centralicen los datos y que alimentando una base de datos, no sea necesario actualizar todos los clientes.
- Arquitectura Modelo-vista-controlador, donde la interfaz sea controlada por los controladores y alimentada por el modelo, pero nunca al revés. Esto otorga modularidad al sistema, permitiendo cambiar, la interfaz o el modelo sin tener que modificar el otro.
- Arquitectura dirigida por eventos, para que sea el usuario el que tenga el control sobre qué es lo que quiere hacer y cuando. Toda acción de usuario genera un evento, que es controlado y produce una reacción en el resto de la aplicación.

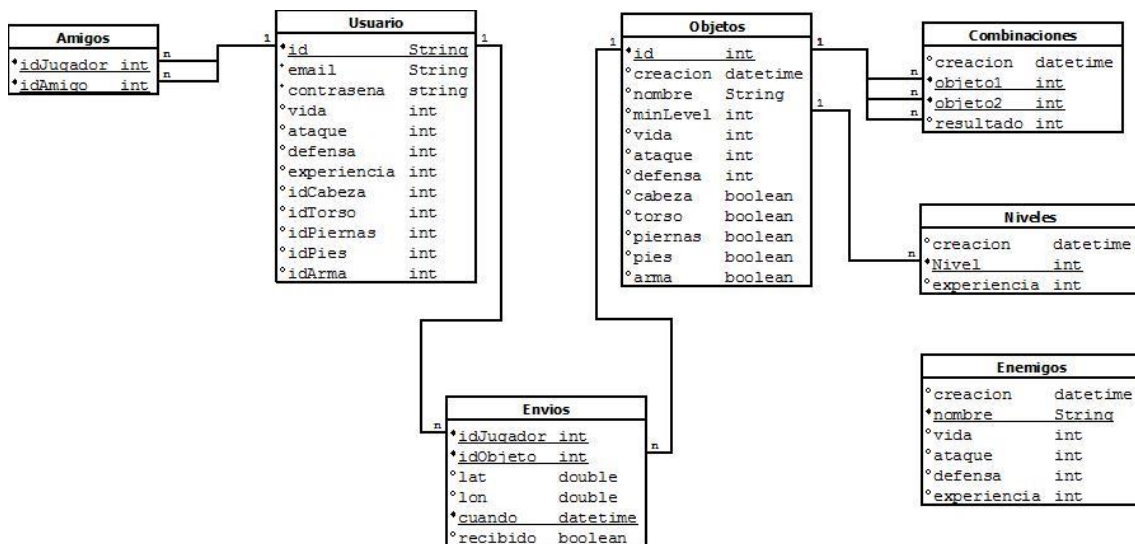
Análisis y diseño

Transformación de modelo de dominio a Base de datos

Este modelado de los datos, se transformó en dos bases de datos, casi idénticas, una en cada dispositivo móvil y otra en el servidor, que ejercería de central y sobre la que los clientes pedirían actualizaciones.

La base de datos del servidor, es donde se almacenan todos los datos necesarios, los datos descargables (objetos, combinaciones, niveles y enemigos) poseen un atributo extra que indica la fecha de creación, esto es para que, cuando el terminal cliente mande la petición de descarga, esta deba de incluir la fecha de la última actualización, y se le enviaran la diferencia de datos entre los que había en dicha fecha de actualización y la actual.

El siguiente diagrama muestra el diagrama de base de datos del servidor.



La base de datos de los terminales contiene menos información ya que no necesita ser almacenada, el id, email y contraseña del usuario aquí no es necesaria y la tabla usuario (llamada Estadísticas) solo almacena los datos de ese único usuario. La tabla de envios no existe y la tabla de amigos solo contiene una lista de nuestros amigos. Y en la tabla objetos se añade el campo cantidad, que al incluir un nuevo objeto, ese se inicializa a cero y crece según se recogen objetos en el juego.

También se puede apreciar la falta de claves primarias y extranjeras, esto es para evitar problemas a la hora de insertar o definir tablas. Estas claves están realizadas en la base de datos principal, la del servidor y por ello no es necesario repetirlas en el cliente, por eso se deja al sistema que las administre como mejor le convenga.

Objetos	Estadísticas	Enemigos
*id int	*vida int	*nombre String
*nombre String	*ataque int	*vida int
*minLevel int	*defensa int	*ataque int
*cantidad int	*experiencia int	*defensa int
*vida int	*idCabeza int	*experiencia int
*ataque int	*idTorso int	
*defensa int	*idPiernas int	
*cabeza boolean	*idPies int	
*torso boolean	*idArma int	
*piernas boolean		
*pies boolean		
*arma boolean		

Niveles	Amigos
*Nivel int	*nombre String
*experiencia int	

Combinaciones
*objeto1 int
*objeto2 int
*resultado int

Diagrama de clases

A continuación se muestran todas las clases que son necesarias para el funcionamiento del software en un dispositivo Android, junto con sus atributos y las funciones que cada una de ellas lleva. Están separadas en tres paquetes, el controlador de la interfaz, donde están todos los Fragments y Activities, el modelo, donde los controladores piden la información a la base de datos que necesitan las interfaces, y los datos y conexiones externas, donde está la clase que controla la base de datos de la aplicación y las clases que heredan de AsyncTask, responsables de la conexión con el servidor externo.

BaseDeDatos
<pre> -ld: SQLiteDatabase +BaseDeDatos(context:Context) +onCreate(db:SQLiteDatabase) +onUpgrade(db:SQLiteDatabase,oldVersion:int, newVersion:int) +cargarNivel(nivel:int,exp:int) +cargarObjeto(id:int,nombre:String,minLevel:int, vida:int,defensa:int,ataque:int, cabeza:int,torso:int,piernas:int, pies:int,arma:int) +cargarCombinacion(obj1:int,obj2:int,rdo:int) +getObjetoRandom(): Objeto +sumarAlInventario(objeto:Objeto) +getListObjetos(): LinkedList<String> +getObjeto(id:int): Objeto +getCantidad(id:int): int +restarCantidad(id:int,cant:int) +getIdObjeto(nombre:String): int +getStat(stat:nombreEstadistica): int +getObjetosEquipables(parte:nombreEstadistica): String[] +equipar(parte:nombreEstadistica,nombreObjeto:String): Objeto +getObjetosUtilizables(): String[] +combinar(ing1:String,ing2:String): String +añadirExperiencia(puntos:int) +getExperienciaNecesaria(): int +getNivel(): int +getListAmigos(): LinkedList<String> +añadirAmigo(nombre:String) +obtenerEnemigoAleatorio(): Enemigo +cargarEnemigo(vida:int,ataque:int,defensa:int, exp:int,nombre:String) </pre>

<<AsyncTask<String, Void, JSONObject>>> GestionActualizacionesRemoto
<pre> -c: Context -barProgressDialog: ProgressDialog -ip: String -ur: String +GestionActualizacionesRemoto(pc:Context) #onPostExecute() #doInBackground(params:String[]): JSONObject #onPostExecute(jsonObject:JSONObject): void </pre>

<<AsyncTask<String, Void, JSONObject>>> GestionAmigosRemoto
<pre> -c: Context -barProgressDialog: ProgressDialog -ip: String -ur: String +GestionAmigosRemoto(pc:Context) #onPostExecute() #doInBackground(params:String[]): JSONObject #onPostExecute(jsonObject:JSONObject): void </pre>

<<AsyncTask<String, Void, JSONObject>>> GestionUsuariosRemoto
<pre> -c: Context -barProgressDialog: ProgressDialog -ip: String -ur: String +GestionUsuariosRemoto(pc:Context) #onPostExecute() #doInBackground(params:String[]): JSONObject #onPostExecute(jsonObject:JSONObject): void </pre>

Equipo
-c: Context
-mEquipo: Equipo
-statusJugador: Stats
-Equipo()
+getEquipo(pci:Context): Equipo
+getStats(): Stats
+getObjetosEquipables(parte:Stats.nombreEstadistica): String[]
+equipar(parte:Stats.nombreEstadistica,nombreObjeto:String)
+añadirExperiencia(puntos:int)
+getExperienciaNecesaria(): int
+getNivel(): int

Descargas
-c: Context
-mDescargas: Descargas
-Descargas()
+getDescargas(pci:Context): Descargas
+cargarObjetos(objetos:JSONArray)
+cargarCombinaciones(combinaciones:JSONArray)
+obtenerJugadoresServidor(): CharSequence[]
+cargarEnvios(envios:JSONArray)
+cargarNiveles(niveles:JSONArray)
+cargarEnemigos(enemigos:JSONArray)

Objeto
-id: int
-nombre: String
-minLevel: int
-vida: int
-ataque: int
-defensa: int
-cabeza: boolean
-torso: boolean
-piernas: boolean
-pies: boolean
-arma: boolean
+Objeto(pid:int,pnombre:String,pminLevel:int,pVida:int,pAtaque:int,pDefensa:int,pCabeza:boolean,pTorso:boolean,pPiernas:boolean,pPies:boolean,pArma:boolean)
+getNombre(): String
+getId(): int
+getVida(): int
+getAtaque(): int
+getDefensa(): int
+isCabeza(): boolean
+isTorso(): boolean
+isPiernas(): boolean
+isPies(): boolean
+isArma(): boolean

Materia
-latitud: double
-longitud: double
-objeto: Objeto
+Materia(lat:double,lng:double,pObjeto:Objeto)
+getLatitud(): double
+getLongitud(): double
+getObjeto(): Objeto

Amigos
-c: Context
-mAmigos: Amigo
-envios: LinkedList<Materia>
-Amigos()
+getAmigos(pci:Context): Amigos
+getAmigos(): LinkedList<String>
+añadirAmigo(nombre:String)
+obtenerEnvios(): LinkedList<Materia>
+enviarAmigo(nombreObjeto:String,nombreAmigo:String)
+añadirEnvio(lEnvio:LinkedList<Materia>)

Stats
+nombreEstadisticas: enum = (VIDA, DEFENSA, ATAQUE, EXPERIENCIA, CABEZA, PECHO, PIERNAS, PIES, ARMA)
-vida: int
-ataque: int
-defensa: int
-experiencia: int
-cabeza: Objeto
-pecho: Objeto
-piernas: Objeto
-pies: Objeto
-arma: Objeto
+Stats(c:Context)
+getVida(): int
+getDefensa(): int
+getAtaque(): int
+getCabeza(): Objeto
+getPecho(): Objeto
+getPiernas(): Objeto
+getPies(): Objeto
+getArma(): Objeto
+actualizar(parte:nombreEstadistica,o:Objeto)
+añadirExperiencia(puntos:int)

Jugador
-DISTANCIA_MINIMA_RECORRER: float = 1000
-DIFERENCIA: double = -0.005
-SALTO: double = 0.010
-c: Context
-mJugador: Jugador
-mLocation: Location
-distanciaRecorrida: int
-materiasEnElMundo: LinkedList<Materia>
-Jugador()
+getMijugador(pci:Context): Jugador
+getMijugador(): Jugador
+actualizarPosicion(location:Location)
+recogerMateriaOroana(location:Location)
+actualizarDistancia(location:Location)
+obtenerLocalizacion(): Location
+distanciaRecorrida()
+getMaterias(): LinkedList<Materias>
+añadirObjeto(objeto:Objeto)

Combates
-mCombate: Combate
-c: Context
-Combates()
+getCombates(pci:Context): Combates
+getEnemigo(): Enemigo

Enemigo
-vida: int
-defensa: int
-ataque: int
-experiencia: int
-nombre: String
+Enemigo(vida:int,ataque:int,defensa:int,nombre:String,exp:int)
+getVida(): int
+getAtaque(): int
+getDefensa(): int
+getNombre(): String
+getExp(): int
+recibirAtaque(dafior:int): int

Objetos
-mObjetos: Objetos
-c: Context
-Objetos()
+getObjetos(pci:Context): Objetos
+getListaObjetos(): LinkedList<String>
+obtenerInfoObjeto(id:int): Objeto
+obtenerCantidad(id:int): int
+tirarObjeto(c:Objeto)

Combinaciones
-c: Context
-mCombinaciones: Combinaciones
-Combinaciones()
+getCombinaciones(pci:Context): Combinaciones
+getObjetosUtilizables(): String[]
+combinar(ing1:String,ing2:String): String

```

<<Fragment>>
MapaFragment

-EXPERIENCIA RECOGER: int = 10
-view View
-VAL MATERIA: int = 30
-FRESH: long = 5000
-REQUEST: LocationRequest
-mLocationClient: LocationClient
-listener: Localizador
-hashMap: HashMap<String, Marker>
-materiaEnCola: LinkedList<Material>

newInstance(): MapaFragment
+MapaFragment()
+onCreate(savedInstanceState: Bundle)
+onResume()
+organizarMateriaEnCola()
+onCreateView(inflater: LayoutInflater, container: ViewGroup, savedInstanceState: Bundle)
+connectarInternetes()
+getMap(): GoogleMap
+inicializarMapa()
+actualizarMapa()
+anadirMaterial()
+anadirMaterial(mMaterial: Material, color: float)
+eliminarMaterial(mMaterial: Material)
+anadirMaterialEnCola(mMaterial: Material)
+onLocationListener(location: Location)
+onStatusChanged(provider: String, status: int, extras: Bundle)
+onProviderEnabled(provider: String)
+onProviderDisabled(provider: String)
+onConnected(bundle: Bundle)
+onDisconnected()
+onConnectionFailed(connectionResult: ConnectionResult)
+onlyLocationChange(location: Location)

<<Fragment>>
CombatFragment

-vida_3: int
-atque_3: int
-defensa_3: int
-ei: Enemy
-barraVida: ProgressBar
newInstance(): CombatFragment
+CombatFragment()
+onCreate(savedInstanceState: Bundle)
+onCreateView(inflater: LayoutInflater, container: ViewGroup, savedInstanceState: Bundle)
+onResume()
+iniciarCombate()
+renderise()
+terminarRes(String, exp: int)
+ronda(accion: int)

<<ActionBarActivity>>
RegistroActivity

+onCreate(savedInstanceState: Bundle)
+register(alias: String, email: String, pass: String)
+onCreateOptionsMenu(menu: Menu): boolean
+onOptionsItemSelected(item: MenuItem): boolean

<<ActionBarActivity>>
ObjetoDetailActivity

-on: Objecto
+onCreate(savedInstanceState: Bundle)

<<ActionBarActivity>>
MainActivity

-REQUEST CODE RECOVER PLAY SERVICES: int = 1001
-on: MainActivity
+onNavigateToDrawerFragment(): NavigationDrawerFragment
+onAmigoFragment(): AmigoFragment
+onCombatFragment(): CombatFragment
+onLaboratorioFragment(): LaboratorioFragment
+onEquipOfragment(): EquipOfragment
+onInventoryFragment(): InventoryFragment
+onMapaFragment(): MapaFragment

+onCreate(savedInstanceState: Bundle)
+actualizar()
+identificar()
+onResume()
+onPause()
+onNavigateDrawerItemSelected(position: int)
+desconectar()
+setAmigoFragment(ft: FragmentTransaction)
+setBatallaFragment(ft: FragmentTransaction)
+setLaboratorioFragment(ft: FragmentTransaction)
+setEquipOfragment(ft: FragmentTransaction)
+setInventoryFragment(ft: FragmentTransaction)
+setMapaFragment(ft: FragmentTransaction)
+onConnected()
+onLocationChange()
+restoreActionBar()
+onCreateOptionsMenu(menu: Menu)
+onOptionsItemSelected(item: MenuItem)
+showPlayServices(): boolean
+showErrorDialog(code: int)
+getMapaFragment(): MapaFragment

<<Fragment>>
EquipOfragment

-vida: int
-vidaExtra: int
-atque: int
-atqueExtra: int
-defensa: int
-defensaExtra: int
newInstance(): EquipOfragment
+EquipOfragment()
+onCreate(savedInstanceState: Bundle)
+onCreateView(inflater: LayoutInflater, container: ViewGroup, savedInstanceState: Bundle)
+onResume()
+sumaRestadificas(o: Objecto)

<<Fragment>>
AmigoFragment

newInstance(): AmigoFragment
+AmigoFragment()
+onCreate(savedInstanceState: Bundle)
+onCreateView(inflater: LayoutInflater, container: ViewGroup, savedInstanceState: Bundle)
+onResume()

<<ActionBarActivity>>
ObjetoDetailActivity

-on: Objecto
+onCreate(savedInstanceState: Bundle)

<<Fragment>>
NavigationDrawerFragment

-STATE SELECTED POSITION: String = selected navigation drawer position
-REF USER LEARNED DRAWER: String = navigation drawer learned
-onClickBack: NavigationDrawerClickListener
+onDrawerToggle: ActionBarDrawerToggle
+onDrawerLayout: DrawerLayout
+onDrawerListView: ListView
+onFragmentContainerView: View
+onItemSelected(position: int)
+onFromSavedInstanceState: boolean
+onUserLearnedDrawer: boolean
+onNavigationDrawerFragment()
+onCreate(savedInstanceState: Bundle)
+onResume()
+onCreate(savedInstanceState: Bundle)
+onCreateView(inflater: LayoutInflater, container: ViewGroup, savedInstanceState: Bundle)
+onDrawerOpen()
+setUp(fragmentId: int, drawerLayout: DrawerLayout)
+selectedItem(position: int)
+onAttach(activity: Activity)
+onDetach()
+onSaveInstanceState(outState: Bundle)
+onConfigurationChanged(newConfig: Configuration)
+onOptionsItemSelected(item: MenuItem)
+showGlobalContextActionBar()
+getActionBar(): ActionBar

<<ListFragment>>
InventoryFragment

-lista: LinkedList<String>
newInstance(): InventoryFragment
+InventoryFragment()
+onCreate(savedInstanceState: Bundle)
+onCreateView(inflater: LayoutInflater, container: ViewGroup, savedInstanceState: Bundle)
+onResume()
+onItemClickListener(listView: View, position: int, id: long)

<<Fragment>>
LaboratorioFragment

-EXPERIENCIA LABO: int = 20
+useIngredientel: boolean
+useIngrediented: boolean
newInstance(): LaboratorioFragment
+LaboratorioFragment()
+onCreate(savedInstanceState: Bundle)
+onCreateView(inflater: LayoutInflater, container: ViewGroup, savedInstanceState: Bundle)
+onResume()

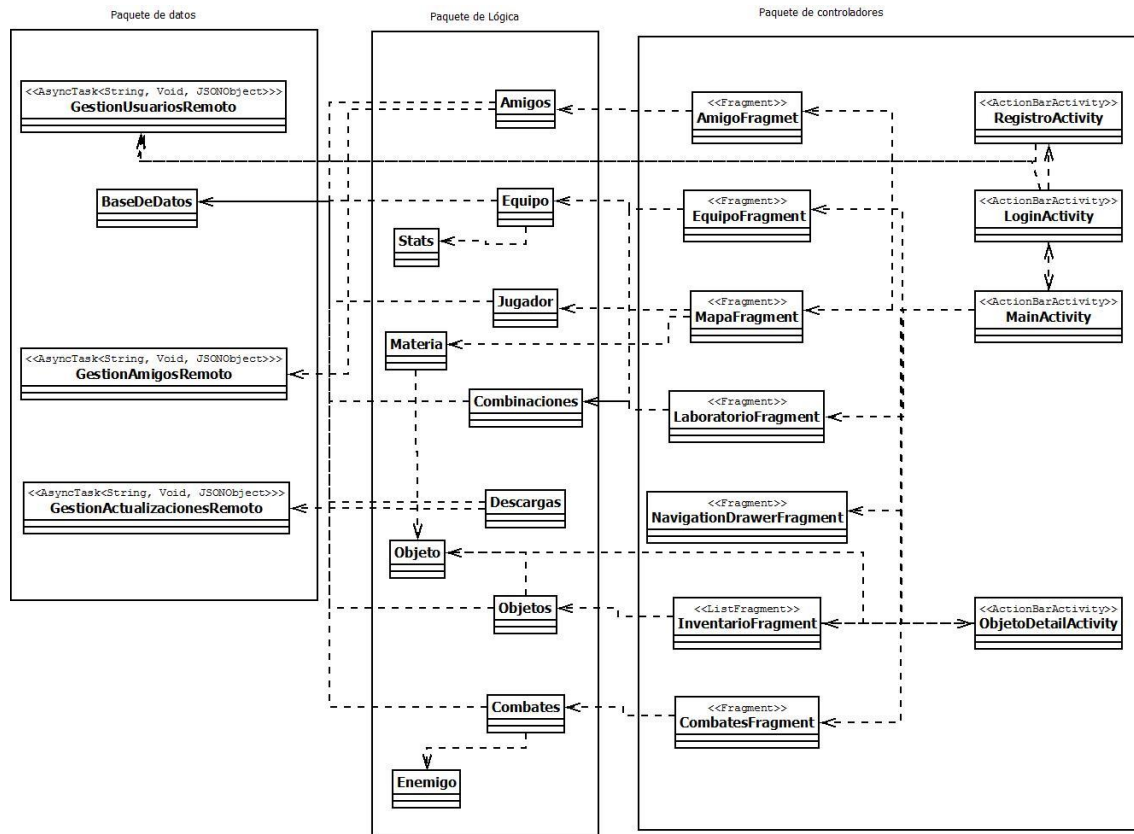
<<ActionBarActivity>>
LoginActivity

+onCreate(savedInstanceState)
+identificar()
+onCreateOptionsMenu(menu: Menu): boolean
+onOptionsItemSelected(item: MenuItem): boolean
+hash plainText: String: String
  
```

En el servidor se establecieron 4 archivos php que contienen las funciones necesarias para la realización de las tareas necesarias y requeridas, como la identificación de usuarios, la descarga de actualizaciones o la búsqueda de nuevos amigos.

Conexion.php	actualizaciones.php	amigos.php	Usuarios.php

Y a continuación se mostrarán las relaciones de dependencia entre las clases. Se puede observar que todas las dependencias van en un único sentido, desde los controladores hasta los contenedores de información, es decir, la base de datos, no existe ninguna llamada a los controladores, salvo que sea entre ellos, en el caso de las Activity, para poder lanzarse entre ellas.



Arquitectura del sistema

La interfaz de usuario en Android está completamente separada del código, dejándola en unos archivos XML concretos. Estos archivos XML configuran la apariencia de la aplicación y otorgan las características e identificadores a cada uno de los elementos que componen las interfaces. Luego estos son cargados en las diferentes clases que controlan dichas interfaces, como las Activity o los Fragment. A su vez, una Activity puede contener uno o varios Fragments, permitiendo de esta manera modularidad en la interfaz.

Como se ha visto en el diagrama de clases, se cumple arquitectura modelo-vista-controlador. La interfaz, no se comunica directamente con los datos, son los controladores, las Activity y los Fragments los que realizan esta función, haciendo que si se quiere cambiar la disposición o el aspecto de esta interfaz, el código siga siendo igual de valido.

Android se comunica con bases de datos externas a través de un servicio web, en nuestro caso PHP, y JSON como medio de transmisión de datos, por ser más ligero que XML, y estas conexiones se realizan en un segundo plano a la aplicación, por si la comunicación falla, la aplicación no quede congelada o inestable.

En este caso, cada móvil ejerce como un cliente, que crea una comunicación con un servidor remoto, compuesto por un servicio PHP, que accede a su vez a la base de datos MySQL, contenida en la misma máquina. Esta comunicación las peticiones se realizan mediante el protocolo HTTP y los métodos POST y se responden con archivos que contienen datos en formato JSON.

Con el fin de ahorrar tiempo y consumo de datos en los dispositivos, se opta también por instalar una base de datos local, donde almacenar los recursos necesarios para el correcto funcionamiento de la aplicación. Estos datos serán descargados al inicializar la aplicación y serán actualizados con la regularidad necesaria, que gracias al modelo-vista-controlador queda aislada de la interfaz, solo accesible desde las clases del modelo.

Por último el sistema está enfocado a eventos mediante Listeners que generan los eventos necesarios para el control de la aplicación. Estos Listener se encuentran en los botones y listas que el usuario tiene a su disposición para realizar las acciones que correspondan, pero además, dispone de Listener que escuchan los cambios en su posición para actualizar los datos en el mapa, o un Listener táctil que al deslizar el dedo desde la derecha hasta el centro de la pantalla, aparezca el menú despegable que permite al usuario cambiar entre las pantallas. Esto le da al usuario una sensación de control, quitándole la parte lineal.

Diagramas de secuencia

Las aplicaciones Android se centran en las Activity, clases que controlan la interacción con el usuario, y son las que regulan en ciclo de la aplicación. La Activity pasa por varios estados desde que se crea hasta que el usuario puede usarla, y pasa por otros cuando la aplicación se minimiza, se cierra o simplemente se cambia de Activity dentro de la propia aplicación. Según el momento en el que se realicen las acciones, estas tendrán una consecuencia u otra. Los Fragments tienen también su propio ciclo de vida según son creados y generan diferentes disparadores, tal y como las Activity realizan. Es por esto que hay que tener sumo cuidado con las acciones a realizar.

La Activity principal se marcó como pilar del desarrollo, siendo esta la que controlase si el móvil tiene los requisitos necesarios, es decir, disponer de la API de Google para Google Maps, si el usuario está identificado o no y ejecutar las descargas de datos necesarias para el funcionamiento. También hospeda casi todos los Fragments de la aplicación, permitiendo al usuario alternar entre ellos según necesite o le convenga.

En cada etapa se desarrolló y diseño un fragmento del software, siguiendo las observaciones anteriormente descritos. Esto ofrecía realizar el diseño de una manera eficaz y poder corregirlo de manera más cómoda debida a la gran modularidad que este sistema ofrece. Al poder modificar los diagramas, se pueden observar nuevas mejoras y actualizaciones que antes no parecían adecuadas o simplemente no se encontraron.

En el anexo II se encuentran los diferentes diagramas de secuencia que explican el funcionamiento secuencial de cada acción en la aplicación.

DESARROLLO

Lo primero que se ejecuta al iniciar la aplicación es MainActivity. Esta comprueba si están instalados los Google Play Services, necesarios para la aplicación. Si no lo están, la aplicación se cierra. Por el contrario, se pasa a comprobar si ya hay un usuario identificado en la aplicación mediante SharedPreferences, una herramienta de Android. Si no lo está, se cede la ejecución a la Activity LoginActivity, donde el usuario podrá identificarse o registrarse. Por último, si tras las comprobaciones, MainActivity sigue ejecutándose, se envía una petición al servidor central con la última fecha de actualización, y si es la primera vez que se ejecuta la aplicación, esta fecha no es enviada. En el servidor se genera un JSON que contiene la diferencia de datos entre los datos del teléfono y del servidor, por si se ha incluido algún nuevo objeto o nivel.

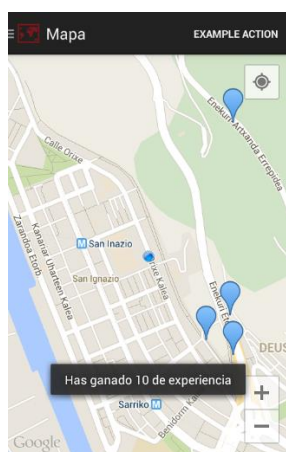
Tras tener todas las comprobaciones y los datos al día, deja en ejecución el primer Fragment, MapaFragment, que contiene el mapa, que carga y muestra varias materias aleatorias en función del lugar donde se encuentre el dispositivo y que además es el encargado de actualizar la posición.

Si el usuario quisiera cambiar de pantalla, para realizar otras cosas, dispone del NavigationDrawerFragment que se muestra deslizando el dedo desde el margen izquierdo de la pantalla hasta el centro y donde aparecen las diversas funcionalidades de la aplicación.

La primera fase instauró la Activity inicial, MainActivity, junto con su NavigationDrawerLayout, descargaba los datos y se ponía en funcionamiento el mapa contenido en su correspondiente Fragment, buscando la última ubicación conocida del dispositivo, y colocando varias Materias alrededor de esta, con diversos objetos que previamente se han descargado o actualizado. (Ver Pantalla 1 más adelante). Se creó la clase Jugador para almacenar la ubicación y las materias que hay disponibles en el mapa y los controladores tuvieran acceso a la base de datos local junto con varias funciones para conocer lo andando y que puedan obtener nuevas materias en caso de que el usuario se mueva lo suficiente.

La siguiente fase consistió en incluir el Fragment que introduzca al usuario la lista de sus pertenencias (ver Pantalla 2 más adelante), así como que permita ver los detalles de cada objeto del que dispone, como la vida que puede aumentarle o en que parte puede equiparse ese objeto. También se generó la clase Objetos para la conexión con la base de datos.

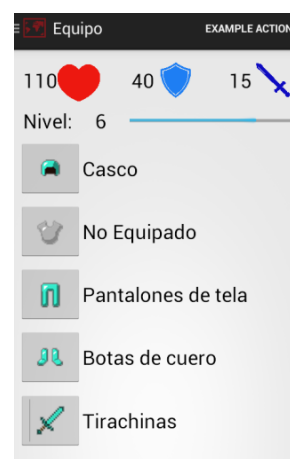
La tercera fase incluye el estado y el equipo del jugador. Aquí se muestra la información del usuario, que vida posee, en qué nivel esta, y que objetos tiene equipados, así como cambiarlos, (ver Pantalla 3 a continuación). La clase Equipo interviene aquí como vínculo con la base de datos.



Pantalla 1



Pantalla 2

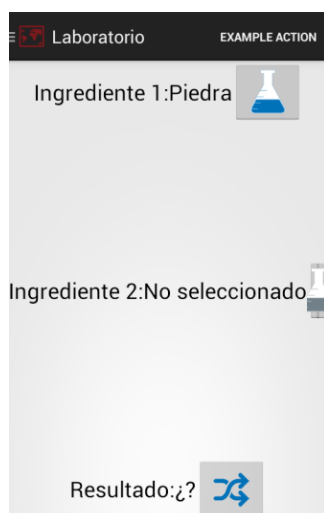


Pantalla 3

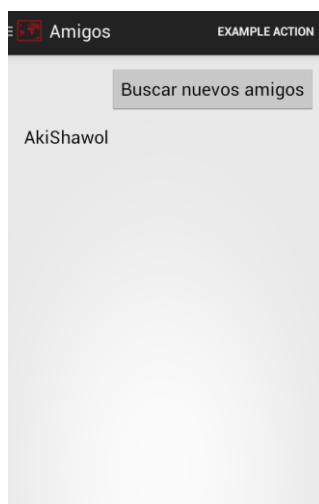
La fase de las combinaciones es la cuarta, donde el usuario puede seleccionar hasta dos objetos de los que disponga y fusionarlos para obtener un nuevo objeto, no siempre con éxito, (ver Pantalla 4, más adelante). Las formulas están en la base de datos y se creó la clase Combinaciones para cumplir este vínculo. Aunque algunas de las funciones ya estaban implementadas en las anteriores fases.

La penúltima fase es la social, donde aparecen nuestro amigos, podemos buscar y añadir nuevos amigos y enviarles algunos de los objetos que tengamos, dejándoselos en la posición donde nos encontremos para que ellos puedan recogerlos posteriormente (ver Pantalla 5 más adelante). Esta es la única fase que tiene relación directa con el servidor, ya que así se requiere para tener una interacción más fluida entre usuarios. También se creó la clase Amigos para vincularlo con la base de datos local.

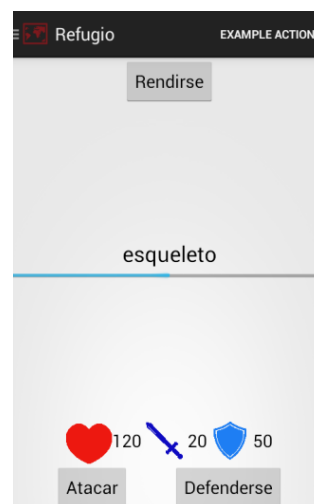
Por último, la fase de combates, muestra un enemigo al que se debe de vencer, usando el arma que tengamos equipada en ese momento, defendernos o huir, (ver Pantalla 6 a continuación). Aquí se creó una última clase Combates, para ayudar a la interfaz y como vínculo a los datos.



Pantalla 4



Pantalla 5



Pantalla 6

El primer inconveniente encontrado fue que para el acceso a la base de datos era irremediablemente necesario el contexto de la aplicación. Al poner las clases MAE entre los controladores de las interfaces y la base de datos era necesario incluir como parámetro el contexto actualizado a cada momento. Se sopeso el incluirlo en los parámetros de cada método desarrollado, sin embargo, tras una investigación, algunos desarrolladores expertos sugirieron incluir este parámetro en la llamada a la MAE en lugar de parámetro de la operación, haciendo de esta manera, más genérico y desligando la operación de la necesidad de incluir un parámetro más.

Otro problema experimentado fue el control del mapa y los Listener necesarios para la localización de la posición. En un principio se sopeso el externalizar todos los Listener en una clase aparte del Fragment, sin embargo, esto requería un acceso a los controladores de las interfaces desde elementos externos a ellos. Esto, debido al diseño elegido, no era viable y podía generar graves problemas en la estabilidad de la aplicación. Finalmente se optó por no hacer esa escisión, dejándolo todo en el Fragment, facilitando la comunicación entre los métodos de los Listener y el Fragment.

El mapa también mostró problemas a la hora de ser usado. El mapa pertenece a una API de Google, el cual controla muy bien todas las licencias y permisos que otorga. Para este proyecto se tuvo que pedir una clave para esta API, así como obtener la huella SHA1 del proyecto. Esto, si no sigues los tutoriales necesarios, puede volverse complicado de hacer. También, a veces los dispositivos no recogen las posiciones de manera correcta, lo que puede generar errores a la hora de lidiar con las posiciones. Esto se arregló poniendo diferentes comprobaciones a la hora de tratar las coordenadas.

El último problema encontrado fueron las imágenes. No se tuvo tiempo, tratando de llevar al día la planificación, de investigar adecuadamente como descargar o almacenar las imágenes adecuadamente, sobre todo las que son dinámicas, como las de los objetos, al contrario que las imágenes de los iconos, que fueron fácilmente integradas en la aplicación, tratando que fuera agradable a la vista del usuario. Esto queda como pendiente para implementar en la aplicación.

VERIFICACIÓN Y EVALUACIÓN

Las pruebas fueron definidas como un check list, donde cada fase tenía una serie de cosas que hacer con varios escenarios posibles. El éxito o fracaso de las pruebas se veía en la completitud de la lista. La manera de determinar si un hito era correcto era arbitrario. La fase de prueba no finalizaba hasta que todo el check list era verde. Las check list, eran inclusivas, se tenía que validar, no solo que la nueva funcionalidad funcionaba, sino que la ya desarrollada seguía funcionando. Esto hacía check list más largos en cada fase.

Algunos ejemplos de hitos en la check list son:

- La aplicación no se detiene automáticamente.
- La aplicación muestra el Drawer en cada uno de los Fragments.
- La aplicación desconecta y conecta adecuadamente.
- La aplicación realiza la descarga de datos necesarios.
- Las interfaces se ajustan a diversos tamaños de dispositivos.

Era difícil y muy costoso la realización de pruebas unitarias programadas, requiriendo tiempo y esfuerzo que podía invertirse en otras cosas. La check list ofrecía la visión más relevante del funcionamiento de la aplicación y de cara al usuario la más funcional. Por el contrario, siempre se pueden escapar casos de prueba haciendo el sistema algo menos fiable.

También se realizaron pruebas con usuarios, que reportaban feedback, sobre interfaz y funcionamiento, buscando mejorar algunos aspectos que desde la programación no se hayan probado, o se hayan pasado por alto.

En el Anexo II se encuentra el plan de pruebas final, con los resultados obtenidos de este.

CONCLUSIONES Y TRABAJO FUTURO

Como se ha explicado a lo largo del documento, se ha creado un juego para dispositivos Andorid que consiste en recoger diversos objetos de su entorno, usarlos, crear nuevos objetos, dejárselos a amigos y combatir contra diversas criaturas.

Para ello se utilizó un sistema de desarrollo basada en técnicas ágiles, haciendo que cada fase tuviera su propias subfases de análisis, diseño, desarrollo y prueba. Convirtiendo la aplicación en una más fácilmente manejable y modular.

Se logró una mayor comprensión de los sistemas móviles, concretamente Andorid, a la vez que un mayor entendimiento del ciclo de vida de una aplicación, junto de los servicios PHP y los archivos JSON.

Respecto a la gestión, la planificación a penas se vio afectada, fue muy ajustada, con algunos retrasos en algunas tareas y algunos adelantos en otras, haciendo que el tiempo se complementase bastante bien. Se suprimieron ciertos aspectos respecto a la propuesta original, reduciendo algunos elementos, como el contexto del juego, dejándolo como tarea secundaria.

Algunos objetivos mencionados en las ideas iniciales fueron suprimidos y cambiados por otros. Por ejemplo se suprimió la idea de elegir bando al principio del juego, y fue cambiado el sistema de combate ideado.

No se conocen grandes juegos que usen la realidad aumentada como principal aliciente en el juego. En dispositivos móviles existen varios juegos que consisten en ir o huir de algo. En el juego Ingress de Niantic Labs, modifican el aspecto del mapa, dejándolo más uniforme visualmente y haciendo que se resalten los puntos que ellos creen necesarios.

Los usuarios que han probado la aplicación han coincidido en que le falta calidad de gráficos, incluir más fotos, y fondos y quizás algo de música o ambientación.

La aplicación puede crecer en varios aspectos. Algunos de las futuras características a desarrollar podrían ser los siguientes:

- Incorporar soporte para varios idiomas.
- Incorporar imágenes en las descripciones de los objetos.
- Crear sistema de lucha entre usuarios.
- Incluir música ambiental o de fondo, junto con sonidos que avisen de éxitos o fracasos.
- Hacer que no todos los objetos sean encontrados en el mapa.
- Incluir más objetos, niveles, combinaciones y adversarios.
- Incorporar una nueva fase que le permita al usuario realizar varias misiones en función de su nivel y que le otorgue experiencia y monedas para poder canjear en un futuro.
- Incorporar un mercado donde poder canjear monedas por objetos más raros.

Android está preparado para soportar varios idiomas y que las aplicaciones cambien este según el idioma del teléfono. La aplicación se ha basado en este principio y se han extraído la mayoría de los textos que contiene a un archivo XML para este fin, para que solo haya que crear tantos XML como idiomas se quieran implementar y traducirlos. Para esta tarea se estima que cerca de 15 o 20 horas se podrían dedicar a terminar de extraer los textos y traducirlos mediante Google Translate.

Los puntos intermedios mencionados podrían hacerse en unos pocos días, ya que solo requieren investigar sobre cómo obtener, almacenar y mostrar imágenes desde un servicio web y ejecutar sonido ambiental. También depende de la originalidad y la capacidad de obtener todo ese material multimedia. El tiempo estimado para estas tareas puede ser aproximadamente de 25 a 35 horas.

Los últimos puntos requieren de más esfuerzo. Crear una nueva fase para introducir misiones, requeriría introducir modificaciones en las bases de datos, en los métodos de descarga de información y de actualización del servidor, además de introducir un nuevo Fragment con su interfaz gráfica.

Para el último punto, además de todo lo anterior, debería de poder realizarse desde la web, por lo que habría que desarrollarla, con los niveles necesarios de seguridad.

Para estas dos últimas características, el trabajo a realizar ya aumentaría considerablemente, llegando a alcanzar las 40 horas por característica.

Respecto al mantenimiento que este software supondría, no presenta muchos comentarios a lo largo del código, pero está altamente modularizado y utiliza nombres significativos, por lo que es fácil encontrar y seguir el error.

BIBLIOGRAFÍA

Matías S. Zavia (2014) Android cede algo de cuota de mercado a iOS en España. Windows Phone crece muy lento. Consultado el 5 de Agosto de 2014, en [HTTP://WWW.XATAKAMOVIL.COM/MERCADO/IOS-Y-WINDOWS-PHONE-LE-QUITAN-CUOTA-DE-MERCADO-A-ANDROID-EN-ESPANA-CON-MUCHA-TIMIDEZ](http://www.xatakamovil.com/mercado/ios-y-windows-phone-le-quitano-cuota-de-mercado-a-android-en-espana-con-mucha-timidez)

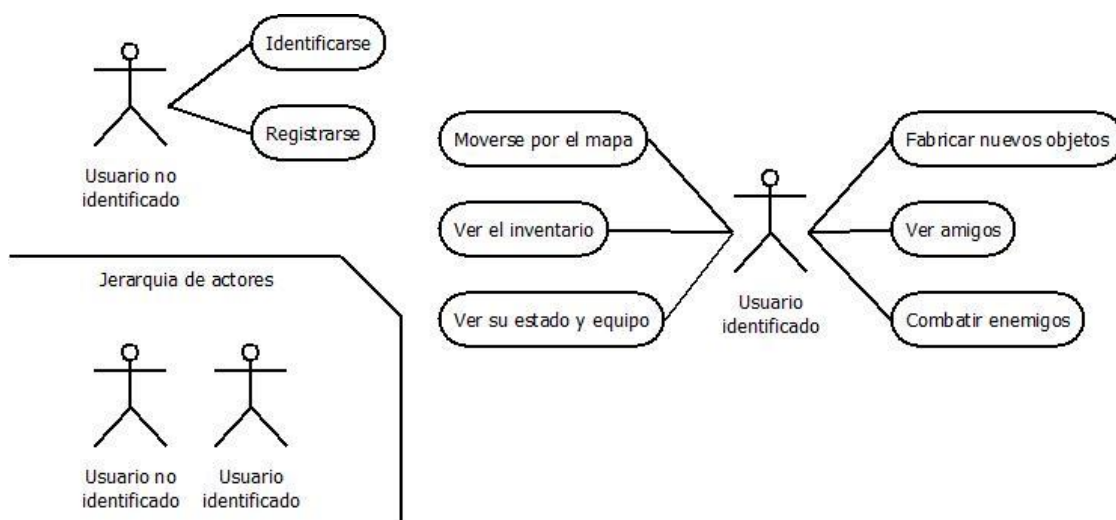
Cliente-servidor. Consultado el 5 de Agosto de 2014, en WIKIPEDIA.ORG

Arquitectura dirigida por eventos. Consultado el 5 de Agosto de 2014, en WIKIPEDIA.ORG

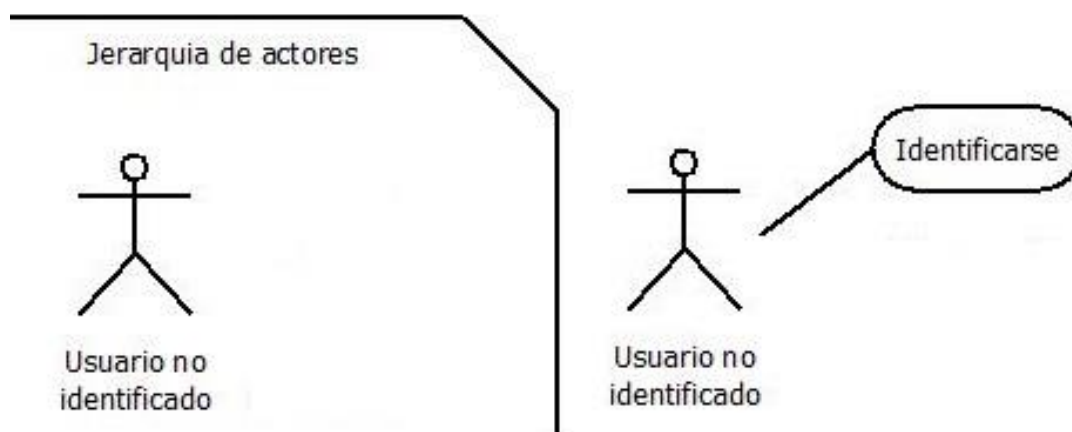
Modelo vista controlador. Consultado el 5 de Agosto de 2014, en WIKIPEDIA.ORG

Android Open Source Project (2014), Starting an Activity. Consultado el 5 de Agosto, en DEVELOPER.ANDROID.COM

ANEXO I.- CASOS DE USO EXTENDIDOS



Caso de uso: Identificarse



Descripción

El usuario deberá introducir unas credenciales para poder identificarse como tal y poder guardar y jugar el juego.

Actores implicados

Usuario no identificado

Precondiciones

Tener conexión a internet en el dispositivo.

Requisitos no funcionales

El sistema ha de ser capaz de soportar multitud de dispositivos físicos y conexiones.

Flujo de eventos

El usuario abre la aplicación.

La aplicación detecta que no hay datos de identificación en el dispositivo. Abre la ventana de identificación. (Prototipo 1)

El usuario introduce los datos de identificación.

Se envían al servidor.

[Si los datos no son correctos o no coinciden]

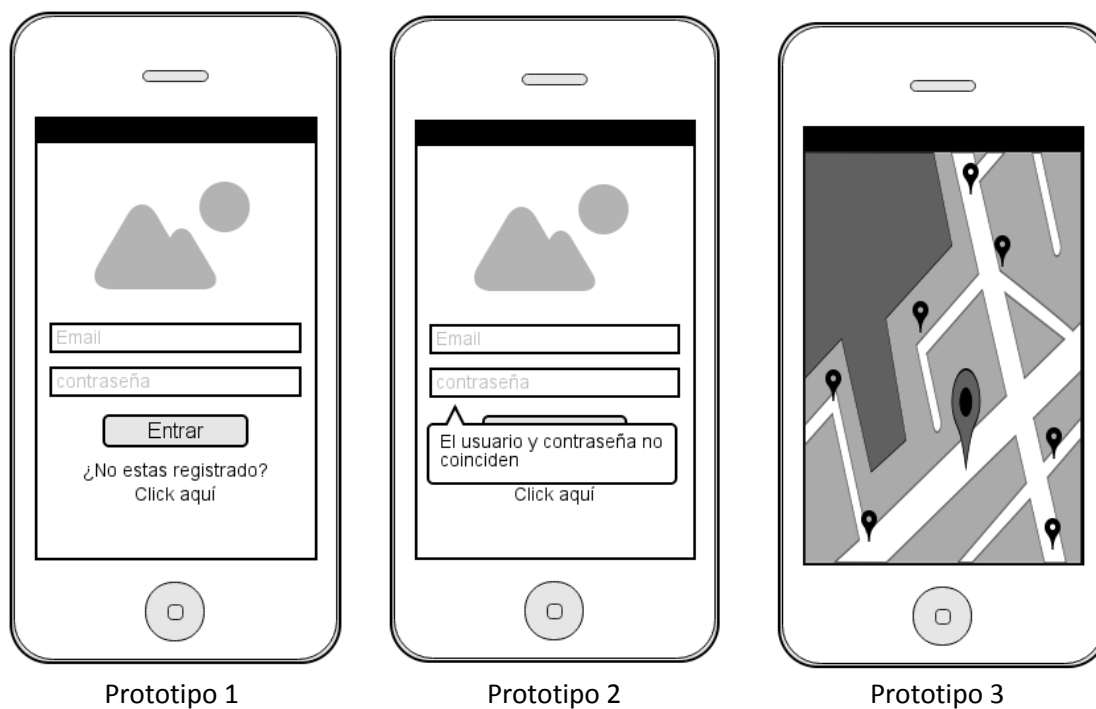
Se devuelve un mensaje de error, que se mostrara en pantalla. (Prototipo 2)

[Si los datos son correctos]

Se abre la pantalla inicial del juego, se descargan los datos relevantes para la aplicación y se carga el mapa. (Prototipo 3)

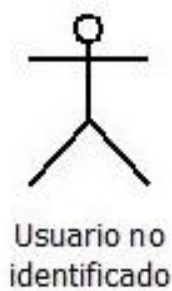
Post condiciones

Interfaz gráfica



Caso de uso: Registrarse

Jerarquia de actores



Descripción

El usuario proporcionará unos datos necesarios para una posterior identificación de su persona en otros, o el mismo, dispositivos, que le permitirá guardar su progreso en el juego.

Actores implicados

Usuario no identificado.

Precondiciones

El dispositivo móvil debe de poseer conexión a internet.

Requisitos no funcionales

El sistema ha de ser capaz de soportar multitud de dispositivos físicos y conexiones.

Flujo de eventos

El usuario inicia la aplicación.

La aplicación detecta que no hay datos de identificación y abre la pantalla de identificación. (Prototipo 4)

El usuario pulsa sobre el botón de registro.

Se abre la pantalla de registro y el usuario introduce los datos requeridos. (Prototipo 5)

Los datos se envían al servidor y se verifican.

[Si los datos no son válidos]

Se devuelve un mensaje de error explicando el error que se le muestra al usuario. (Prototipo 6)

[Si los datos son válidos]

Se devuelve una confirmación. (Prototipo 7)

Se cierra la pantalla de registro y se vuelve a la identificación. (Prototipo 4)

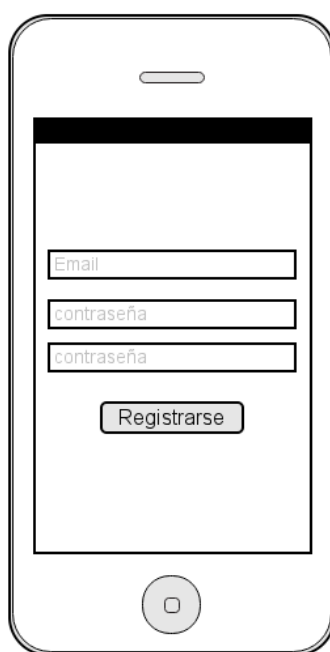
Post condiciones

Se habrá generado un nuevo usuario en el sistema.

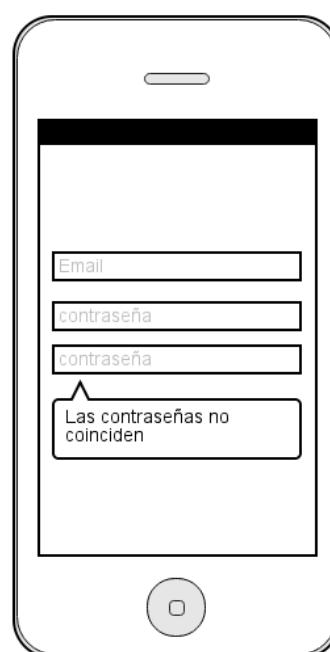
Interfaz gráfica



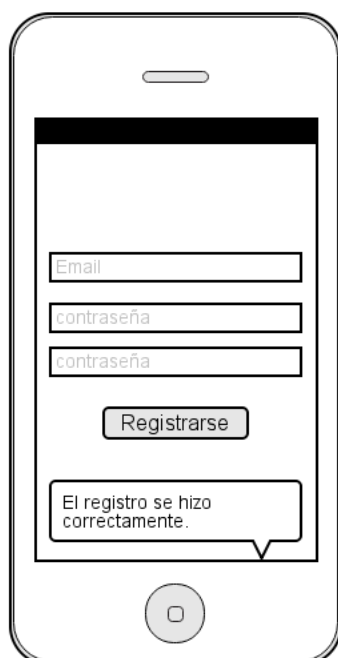
Prototipo 4



Prototipo 5



Prototipo 6



Prototipo 7

Caso de uso: Moverse por el mapa



Descripción

El usuario se desplaza.

Actores implicados

Usuario identificado

Precondiciones

El usuario debe de disponer de un sistema de geolocalización y la aplicación instalada.

Requisitos no funcionales

El sistema ha de ser capaz de soportar multitud de dispositivos físicos y conexiones.

Flujo de eventos

El usuario se desplaza y el dispositivo capta el cambio de posición. (Prototipo 8)

Se comprueba si hay alguna materia cerca de la nueva posición.

[Si hay una materia]

Se elimina del mapa y se agrega el objeto correspondiente al inventario del usuario. (Prototipo 9)

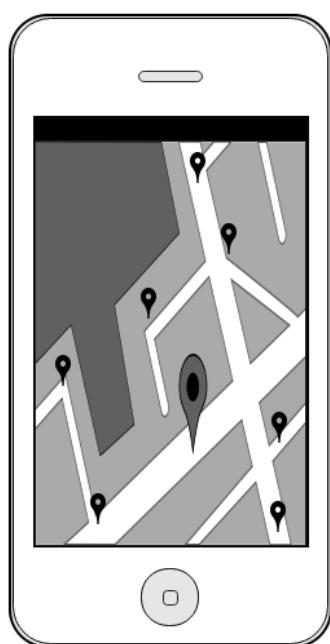
[Si se ha recorrido cierta distancia]

Se cargan más materias en el mapa.

Post condiciones

El usuario habrá cambiado de posición, puede haber añadido un nuevo objeto y cargado más materias en el mapa.

Interfaz gráfica

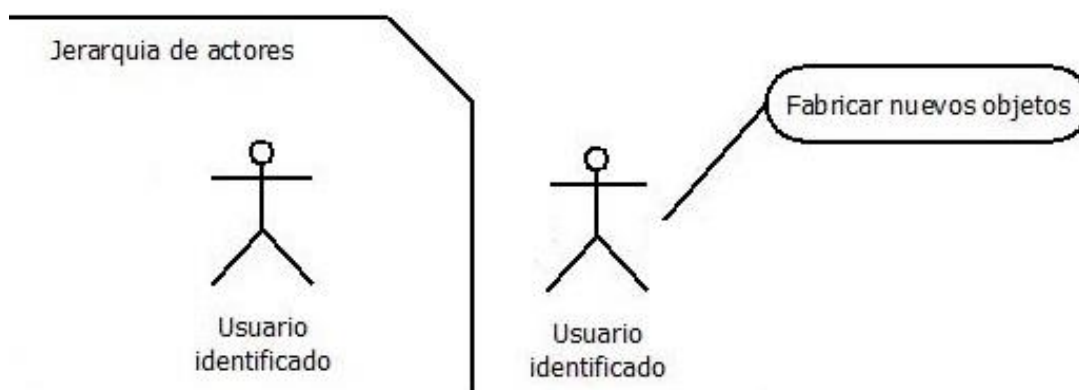


Prototipo 8



Prototipo 9

Caso de uso: Fabricar objeto



Descripción

El usuario podrá crear más y mejores objetos a partir de combinaciones de las que ya tiene en su inventario, pudiendo así, ampliarlo.

Actores implicados

Usuario identificado.

Precondiciones

El usuario debe de disponer de las materias necesarias para hacer una nueva combinación.

Requisitos no funcionales

El sistema ha de ser capaz de soportar multitud de dispositivos físicos y conexiones.

Flujo de eventos

El usuario selecciona hasta dos objetos a combinar. (Prototipo 10)

Se comprueba si en la base de datos esa combinación existe.

[Si existe]

Se añade el resultado al inventario del usuario.

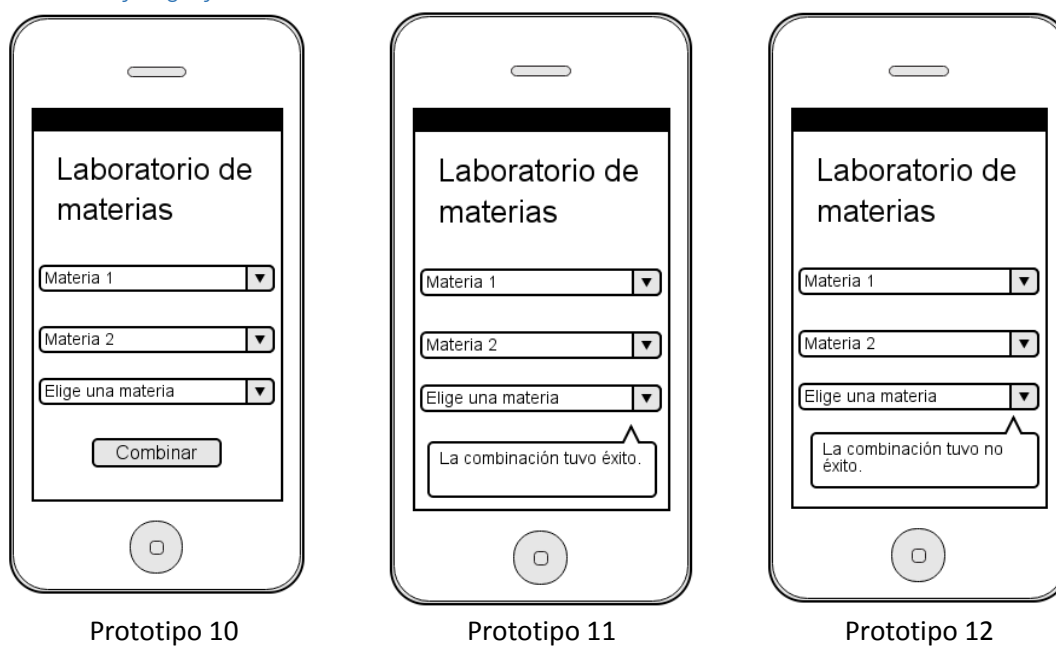
Se eliminan las materias usadas en la combinación.

Se muestra el resultado de la operación. (Prototipo 11 y 12)

Post condiciones

Si el usuario ha realizado correctamente la combinación se añadirá una nueva materia a su inventario. Tanto si se ha realizado correctamente como si no, las materias usadas en la combinación se perderán.

Interfaz gráfica



Caso de uso: Ver el inventario



Descripción

El usuario podrá ver una lista con todas las materias de las que dispone y su cantidad. También podrá deshacerse de ellas.

Actores implicados

Usuario identificado.

Precondiciones

El usuario deberá de disponer de un sistema de geolocalización.

Requisitos no funcionales

El sistema ha de ser capaz de soportar multitud de dispositivos físicos y conexiones. El usuario debe tener instalada la aplicación en el dispositivo móvil.

Flujo de eventos

Se muestra una lista con las materias que el usuario dispone y la cantidad. (Prototipo 13)

[El usuario selecciona una materia]

Se muestra un panel de detalles de la materia. (Prototipo 14)

[Si el usuario selecciona tirar materia]

El usuario pierde una unidad de esa materia.

Post condiciones

El usuario habrá dejado una materia en el lugar donde se encontraba.

Interfaz gráfica

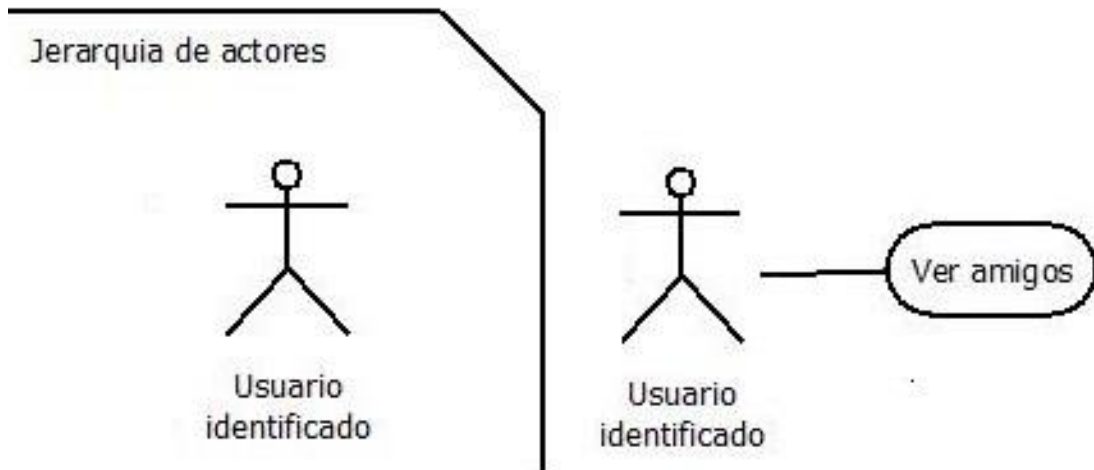


Prototipo 13



Prototipo 14

Caso de uso: Ver amigos



Descripción

El usuario podrá ver una lista con otros usuarios que ha incluido como amigos. Mediante esta opción se podrá enviar materias a otros amigos.

Actores implicados

Usuario identificado.

Precondiciones

El usuario deberá de disponer de un sistema de geolocalización e internet.

Requisitos no funcionales

El sistema ha de ser capaz de soportar multitud de dispositivos físicos y conexiones.

Flujo de eventos

Se obtiene la lista de amigos del usuario. (Prototipo 15)

[Si el usuario pulsa sobre un amigo]

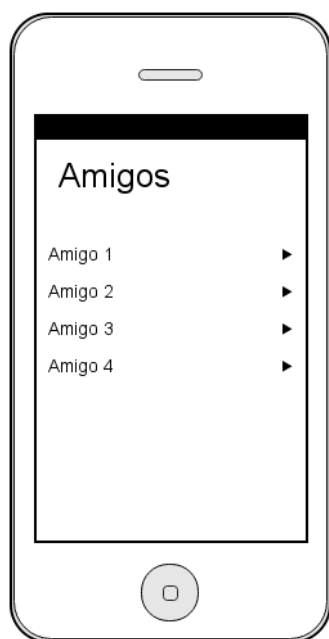
Se abre una pantalla para seleccionar una materia. (Prototipo 16)

Se envía el amigo y la información de la materia al servidor.

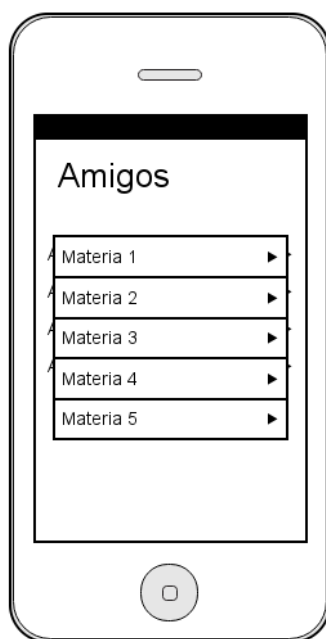
Post condiciones

En caso de que haya enviado una materia a un amigo, el usuario habrá perdido una materia. El usuario que ha recibido la materia, la encontrara la siguiente vez que se identifique.

Interfaz gráfica



Prototipo 15



Prototipo 16

Caso de uso: Ver estado y equipo



Descripción

El usuario puede ver las estadísticas de su personaje y los objetos que tiene equipado, así como cambiar estos objetos.

Actores implicados

Usuario identificado.

Precondiciones

Requisitos no funcionales

El sistema ha de ser capaz de soportar multitud de dispositivos físicos y conexiones.

Flujo de eventos

El usuario elige la opción de ver estado.

Se carga la pantalla con los datos deseados. (Prototipo 17)

[El jugador pulsa sobre el botón de cambiar equipo]

Se muestra una lista con los objetos que el usuario posee y que son equipables en esa parte. (Prototipo 18)

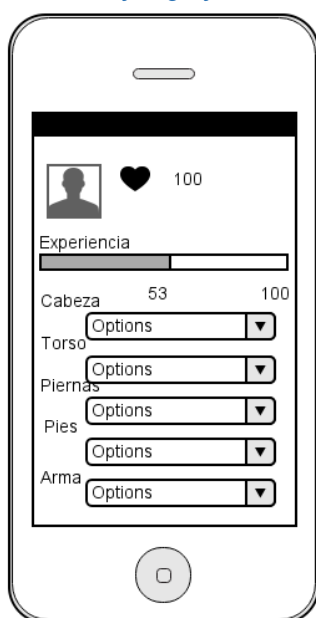
El usuario selecciona una opción.

Se modifican las estadísticas del jugador en función del nuevo objeto.

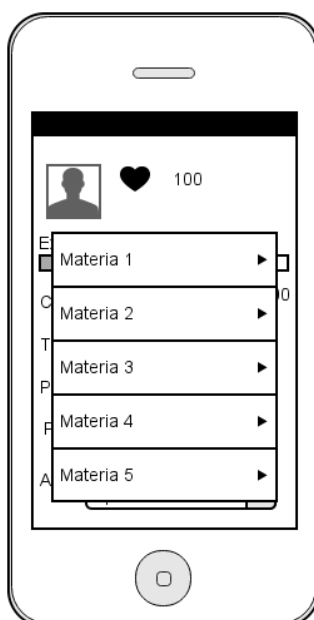
Post condiciones

El usuario ha cambiado una pieza de su inventario y sus estadísticas han cambiado.

Interfaz gráfica

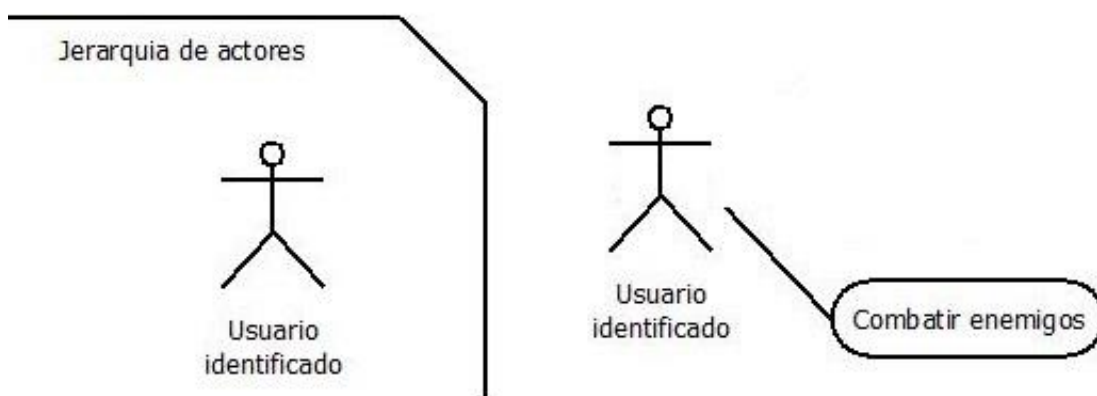


Prototipo 17



Prototipo 18

Caso de uso: Combatir enemigo



Descripción

El usuario puede atacar a un monstruo.

Actores implicados

Usuario identificado.

Precondiciones

El usuario deberá de disponer de un sistema de geolocalización e internet.

Requisitos no funcionales

El sistema ha de ser capaz de soportar multitud de dispositivos físicos y conexiones. El usuario debe tener instalada la aplicación en el dispositivo móvil.

Flujo de eventos

El usuario inicia una batalla.

El sistema elige un enemigo y le asigna estadísticas al azar. (Prototipo 19)

{Mientras el usuario y el enemigo tengan vitalidad positiva.}

El usuario elige una acción a realizar:

[El usuario elige atacar]

Se calcula un número aleatorio que suma a su índice de ataque. Se infringe ese daño al enemigo menos su índice de defensa.

[El usuario elige defenderse]

Se calcula un número aleatorio que aumenta su índice de defensa en el próximo ataque del enemigo.

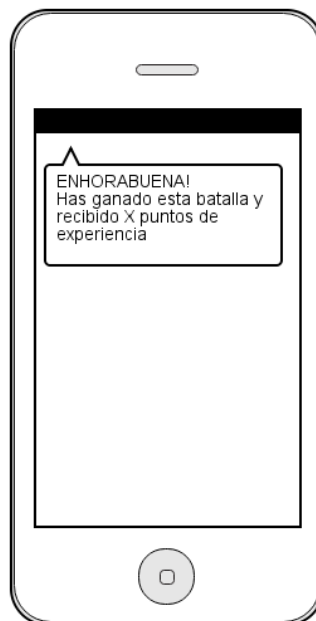
El enemigo ataca. Se calcula un número aleatorio y se suma su índice de ataque y se resta el índice de defensa del usuario. Se resta la vitalidad resultante al usuario.

Se otorga la experiencia al usuario en función del número de turnos empleados y la salud restante. (Prototipo 20 y 21)

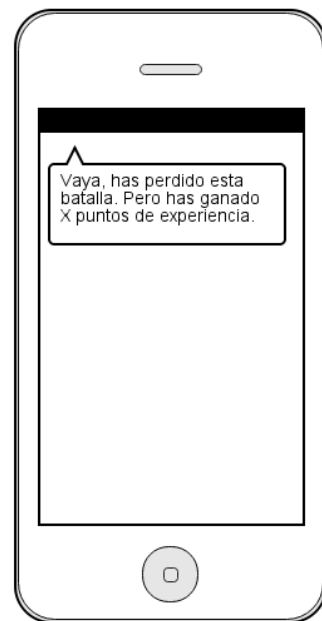
Interfaz gráfica



Prototipo 19



Prototipo 20

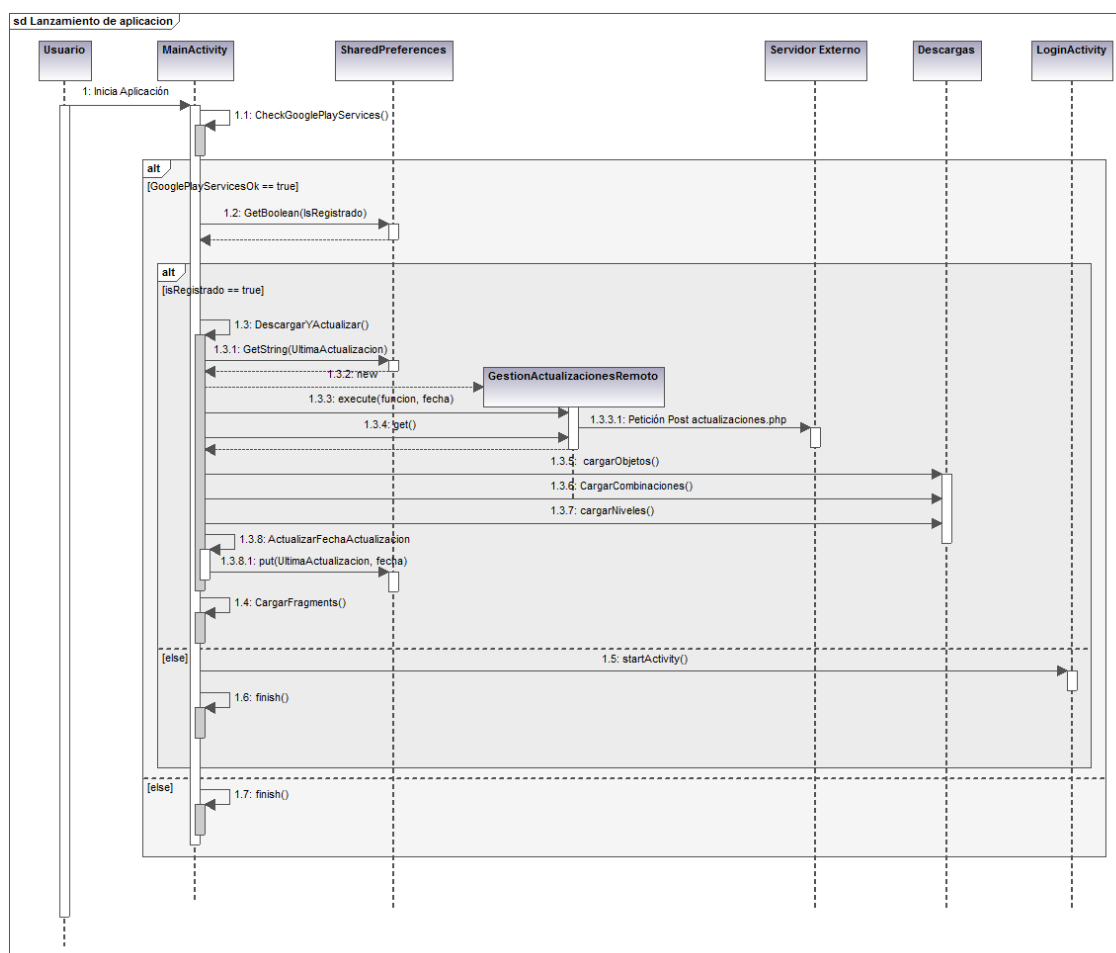


Prototipo 21

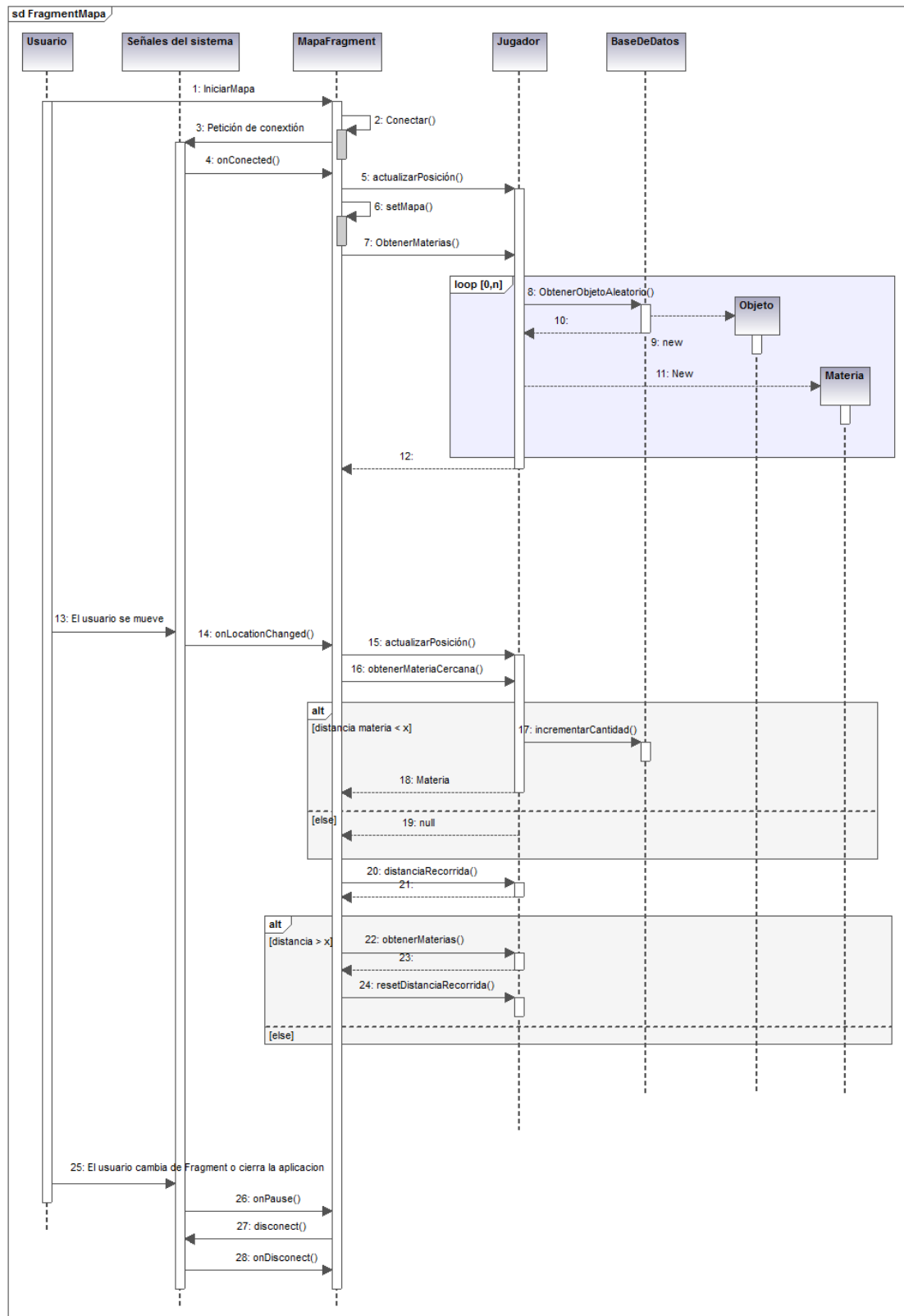
ANEXO II.- DIAGRAMAS DE SECUENCIA

A continuación se expondrán varios diagramas de secuencia que explicarán ligeramente la secuencia de llamadas a implementar.

Aquí se muestran las llamadas que se realizan cuando la aplicación se inicia. Se realizan varias comprobaciones y si todas tienen éxito se descargan los datos y se finaliza de cargar la pantalla correspondiente, sino se puede dar el caso de que la aplicación se cierre y no se pueda ejecutar o se le pida al usuario que introduzca sus credenciales.



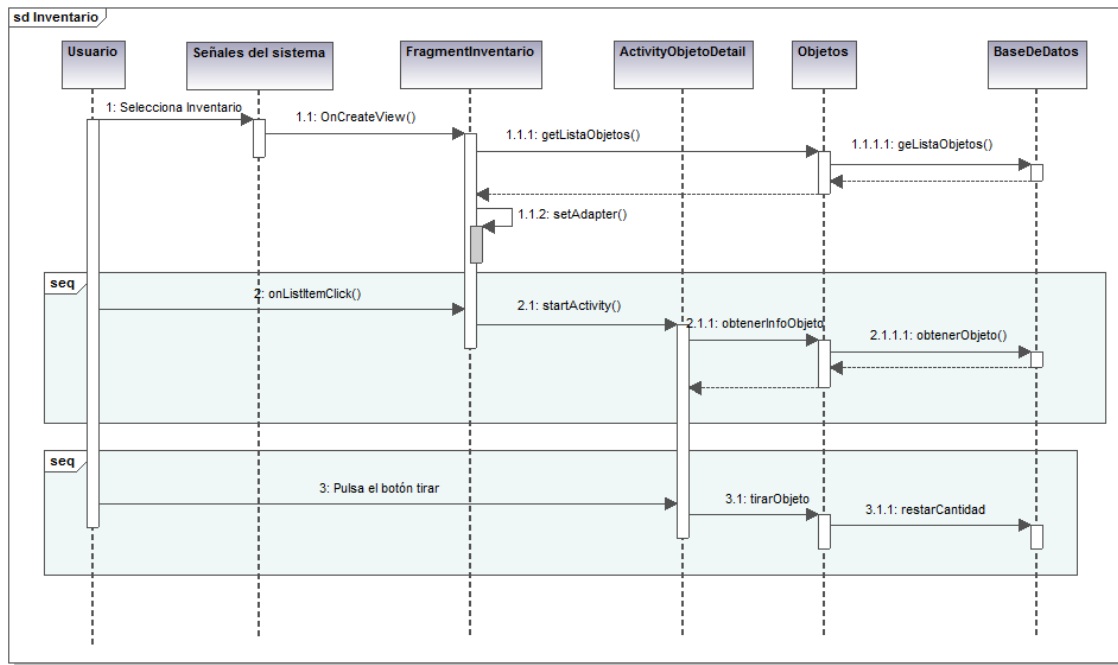
En el siguiente se muestra la secuencia en tres casos, (1) cuando se carga el Fragment que contiene el mapa. Esto ocurre cada vez que abrimos la aplicación exitosamente o dentro de la aplicación navegamos hasta la sección del mapa. La segunda parte (2) muestra que sucede cuando el usuario se mueve estando el mapa activo. Y la última de ellas (3) se muestra que llamadas se suceden cuando se cierra el Fragment, esto es, se cambia a otro Fragment o la aplicación se cierra con este Mapa activo.



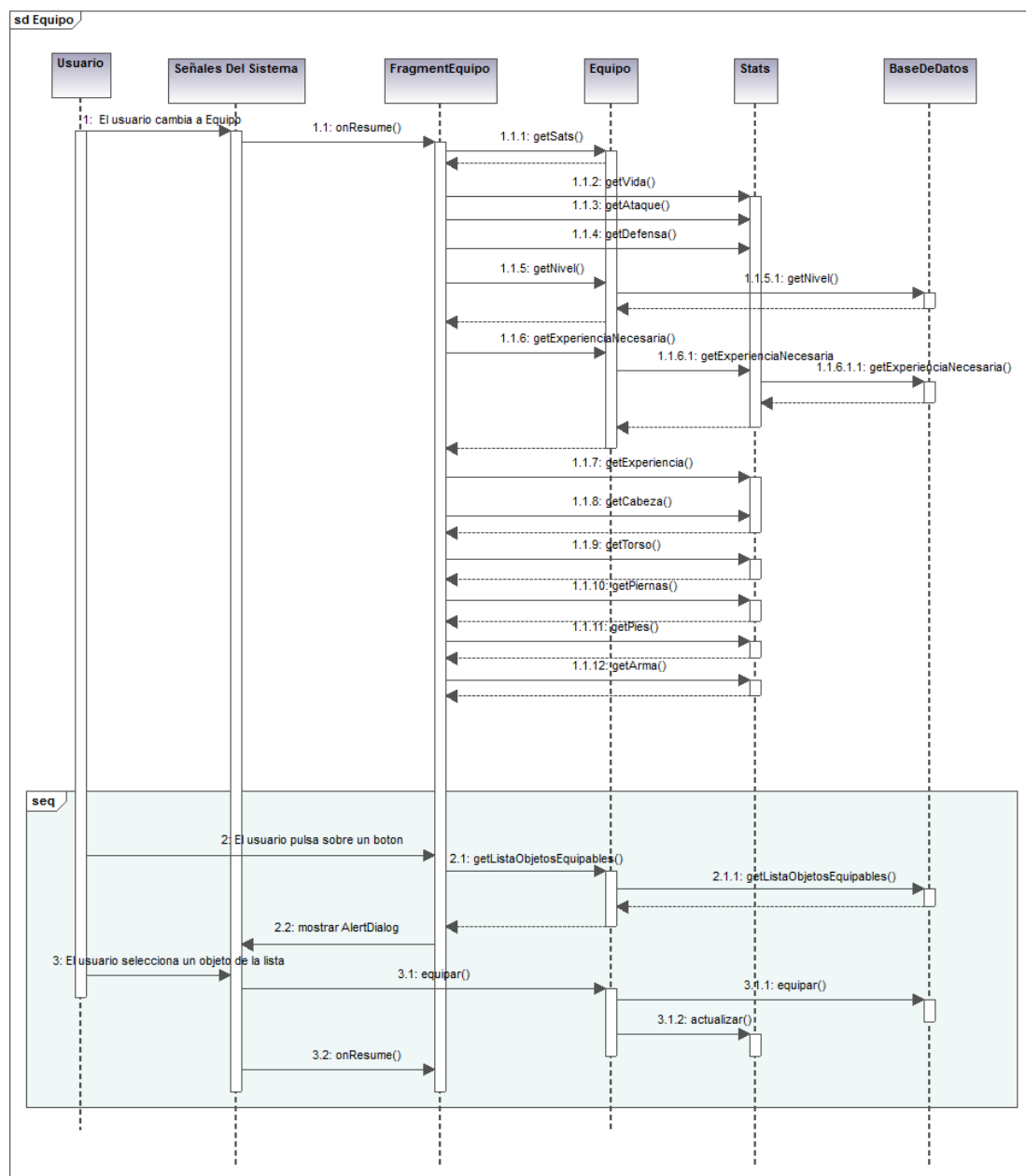
Aquí cabe destacar que no existe ninguna clase llamada Señales del sistema. Esta línea de vida se colocó para comprender más claramente las llamadas que el sistema realiza sin que nosotros lancemos estas llamadas. Son frecuentes en las creaciones y destrucciones de

Fragments y Activitis y en los Listeners. Obviamente no están marcadas todas las llamadas que se realizan, solo las más significativas para el caso.

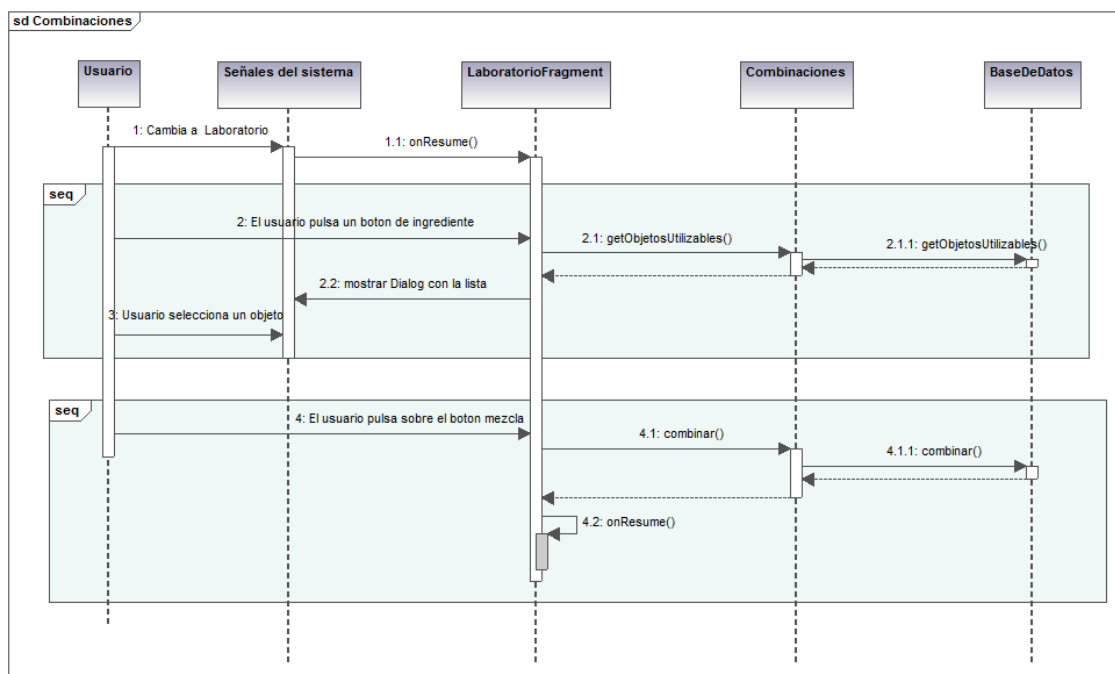
El siguiente diagrama describe el comportamiento del Fragment que contiene la lista de objetos que tiene el jugador. También se hace referencia a lo que sucede cuando se pulsa sobre un objeto. Los cuadros “seq” son los que hacen las separaciones en las acciones del usuario.



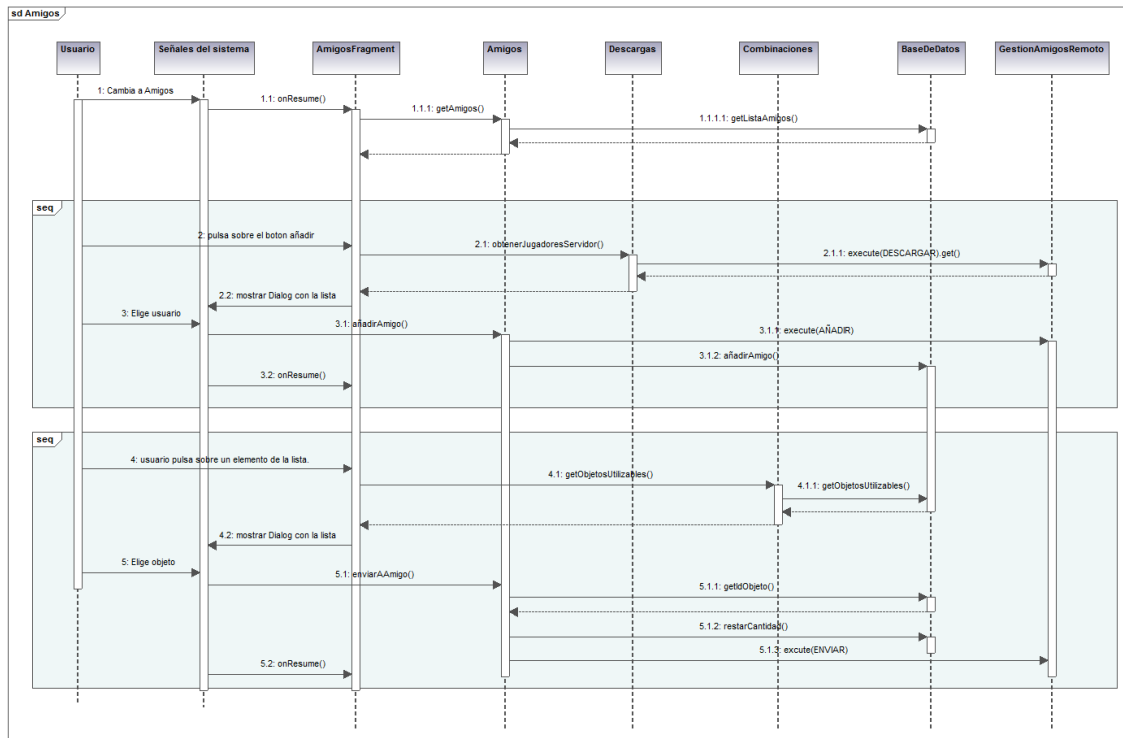
Ahora se mostrará la secuencia de acciones seguidas en la carga del Fragment que muestra el equipo y el estado del jugador. Como en el anterior diagrama, el cuadro “seq” contiene la acción del usuario hace cuando pulsa el botón de cambiar de equipo. Tras finalizar, se aprecia que se llama al método onResume() esto hace que el Fragment se resetee, aplicando los cambios que se han producido.



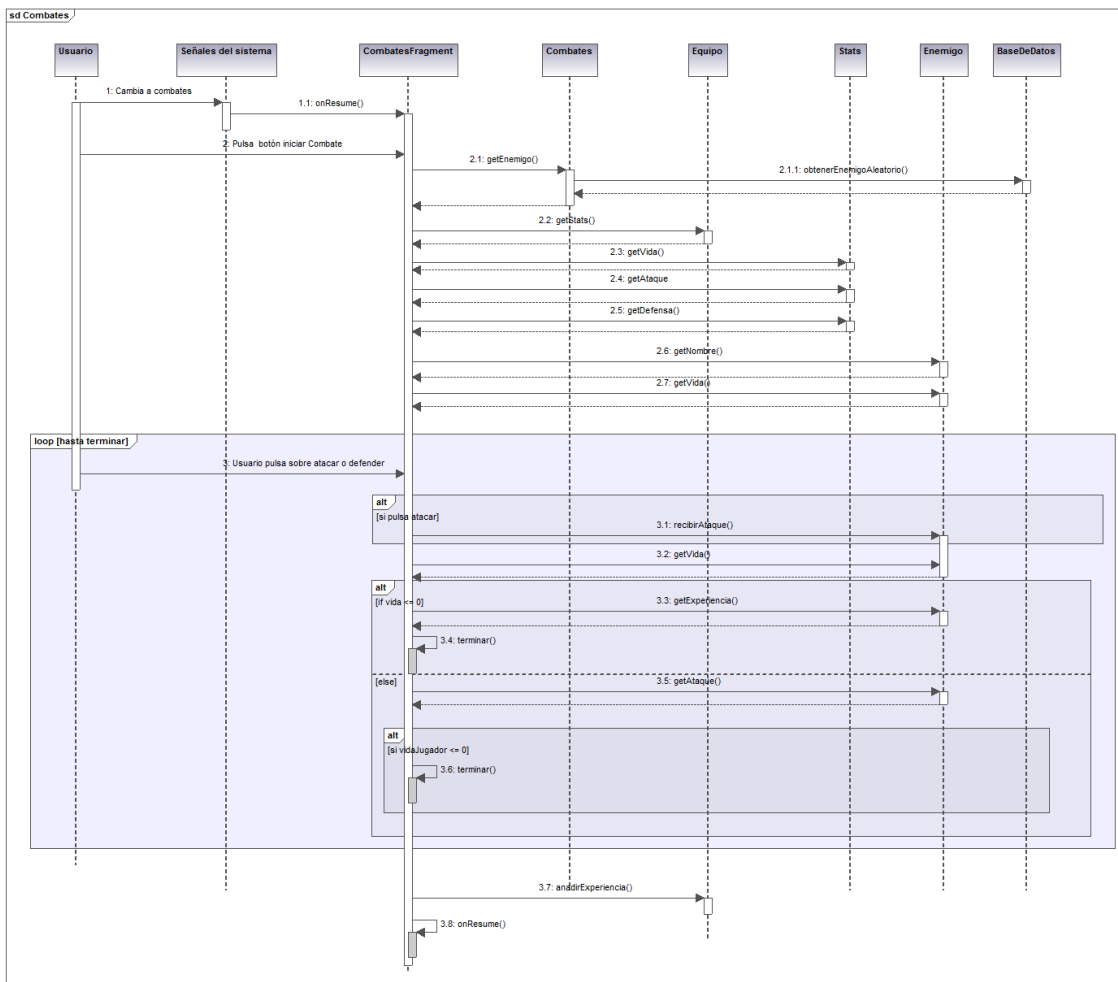
En el siguiente se aprecia el diagrama dirigido a la carga de datos del Fragment que se encarga de permitir al usuario realizar combinaciones. Se puede apreciar también llamada al onResume que resetea la interfaz.



El siguiente diagrama muestra la secuencia seguida en el Fragment que muestra al usuario sus amigos. Se puede apreciar que usa la librería de combinaciones para obtener la lista de objetos que se pueden utilizar. También se aprecia el uso de la clase `GestionAmigosRemoto`, que extiende de la clase `AsyncTask` y permite ejecutar tareas en segundo plano, necesario para la conexión al servidor. En este caso hay una conexión al servidor en medio de la acción dado que esta lista puede ser muy dinámica y cambiar mucho y no hay sentido para tener todos los usuarios en el dispositivo.



Por último se muestra la secuencia seguida en el Fragment correspondiente al combate. En este diagrama lo más importante es el sistema de turnos ideado para el juego, donde las acciones que decide tomar el usuario inician el turno, se realiza la comprobación para saber si el enemigo puede continuar y este ataca en caso afirmativo, tras lo cual se comprueba el estado del jugador y su posibilidad de continuar. Si en algún caso, la vida de alguno de las partes llega a cero, el combate se termina, bloqueando la interacción del usuario con los botones de las acciones.



ANEXO III.- PLAN DE PRUEBAS

En la siguiente tabla se detalla el plan de pruebas llevado a cabo en la última iteración del proyecto, con los resultados que se han obtenido.

01. Identificación incorrecta	
Descripción	Se introducen credenciales incorrectas y no existentes
Entrada proporcionada	Credenciales incorrectas
Salida esperada	Error en identificación.
Salida obtenida	Error en identificación.
Observaciones / cambios	
02. Identificación correcta	
Descripción	Se introducen credenciales correctas
Entrada proporcionada	Credenciales correctas
Salida esperada	Acceso y carga de la interfaz principal del juego
Salida obtenida	Acceso y carga de la interfaz del juego.
Observaciones / cambios	
03. Registro erróneo	
Descripción	Se introducen datos erróneos (email sin @, contraseña demasiado corta)
Entrada proporcionada	Datos erróneos
Salida esperada	Error en registro.
Salida obtenida	Error en registro.
Observaciones / cambios	
04. Registro correcto.	
Descripción	Se introducen datos correctos.
Entrada proporcionada	Datos correctos
Salida esperada	Registro realizado y vuelta a la pantalla de identificación.
Salida obtenida	Registro realizado y vuelta a la pantalla de identificación.
Observaciones / cambios	
05. Desconexión	
Descripción	Se borra la sesión del sistema y se espera que el usuario tenga que volver a identificarse.
Entrada proporcionada	El usuario pulsa sobre el botón "Desconexión".
Salida esperada	Mostrar la pantalla de identificación.
Salida obtenida	Muestra la pantalla de identificación y al reiniciar la aplicación también.
Observaciones / cambios	
06. Uso de aplicación sin internet	
Descripción	Se lanza la aplicación sin conexión a internet
Entrada proporcionada	Lanzamiento de la aplicación sin WIFI ni datos móviles.
Salida esperada	Mensaje de error y cierre de la aplicación
Salida obtenida	Cierre inesperado de la aplicación.
Observaciones / cambios	Se debe de recoger la excepción que trata la falta de conexión a internet para que lance un mensaje de advertencia antes de cerrar la aplicación.
07. Cambio entre Fragments sin error	
Descripción	Se realizan cambios entre los Fragments.
Entrada proporcionada	Cambio por todos los Fragments que hay en la aplicación.

Salida esperada	No errores entre los Fragments.
Salida obtenida	No errores.
Observaciones / cambios	
08. Descarga de datos (no hay datos)	
Descripción	Al iniciar la aplicación se realiza la petición de descarga de nuevos elementos.
Entrada proporcionada	No hay datos a actualizar.
Salida esperada	La aplicación arranca sin problemas.
Salida obtenida	La aplicación arranca sin problemas.
Observaciones / cambios	
09. Descarga de datos (Hay datos)	
Descripción	Al iniciar la aplicación se realiza la petición de descarga de nuevos elementos.
Entrada proporcionada	Hay varios datos para actualizar.
Salida esperada	La aplicación arranca sin problemas y los datos aparecen en la aplicación.
Salida obtenida	La aplicación arranca y los datos están.
Observaciones / cambios	
10. Carga del mapa	
Descripción	Se carga el mapa con varias materias alrededor, se establecen los Listener y el mapa se actualiza con el movimiento del jugador.
Entrada proporcionada	Abrir el Fragment del mapa
Salida esperada	El mapa con varias materias alrededor del jugador.
Salida obtenida	El mapa con varias materias alrededor del jugador.
Observaciones / cambios	
11. Carga de materias en el mapa	
Descripción	El usuario se desplaza por el mapa determinados metros y se necesita cargar más materias.
Entrada proporcionada	Movimiento del usuario.
Salida esperada	Recarga de las materias en el mapa.
Salida obtenida	
Observaciones / cambios	
12. Recogida de materias lejos	
Descripción	El usuario quiere obtener una materia. Pero se encuentra demasiado lejos para poder recogerla.
Entrada proporcionada	Movimiento del usuario.
Salida esperada	La materia no es recogida.
Salida obtenida	La materia no es recogida.
Observaciones / cambios	
13. Recogida de materias cerca	
Descripción	El usuario quiere obtener una materia que se encuentra lo suficientemente cerca.
Entrada proporcionada	El usuario se mueve.
Salida esperada	La materia se recoge y se añade el objeto al inventario.
Salida obtenida	La materia se recoge y es añadida al inventario.
Observaciones / cambios	
14. Carga de inventario (ningún objeto)	
Descripción	El usuario cambia a la pantalla del inventario. El inventario esta vacío y aun no se ha recogido ningún objeto.

Entrada proporcionada	Cambio de pantalla.
Salida esperada	Mostrar la pantalla correctamente.
Salida obtenida	La pantalla se muestra correctamente.
Observaciones / cambios	
15. Carga de inventario (con algún objeto)	
Descripción	El usuario cambia a la pantalla del inventario. El inventario no está vacío.
Entrada proporcionada	Cambio de pantalla.
Salida esperada	Mostrar la pantalla correctamente.
Salida obtenida	La pantalla se muestra correctamente.
Observaciones / cambios	
16. Carga ventana de detalles de objeto	
Descripción	Se quiere mostrar los detalles de un objeto.
Entrada proporcionada	El usuario pulsa sobre un objeto de la lista.
Salida esperada	Nueva pantalla con la descripción y características del objeto.
Salida obtenida	Pantalla con la descripción y características del objeto.
Observaciones / cambios	
17. Tirar objeto	
Descripción	Se tirar un objeto que se encuentra en el inventario.
Entrada proporcionada	Pulsar sobre tirar en un determinado objeto.
Salida esperada	La cantidad del objeto se reduce en una unidad y se muestra una marca en el mapa.
Salida obtenida	La cantidad del objeto se reduce en una unidad.
Observaciones / cambios	No se ha logrado que la marca aparezca donde se encuentra el usuario a la hora de tirar el objeto.
18. Se carga el Fragment de equipo y estado	
Descripción	Se pretende cargar la lista del equipo y el estado.
Entrada proporcionada	Cambio de pantalla.
Salida esperada	Colocación de los datos del jugador. Poner la barra de nivel y el equipo.
Salida obtenida	Colocación de los datos, la barra de experiencia y el equipo que se lleva.
Observaciones / cambios	
19. Se modifica cada parte del inventario	
Descripción	Se realiza un cambio en cada una de las partes del equipo.
Entrada proporcionada	Cambio en el equipo.
Salida esperada	Se guarda el cambio, y las estadísticas del jugador se actualizan en función del objeto.
Salida obtenida	Se guarda el cambio al volver a entrar y las estadísticas son actualizadas en el momento.
Observaciones / cambios	
20. Al seleccionar una parte del inventario solo se muestran los adecuados	
Descripción	Cuando se selecciona cambiar una parte del inventario, solo se muestran los elementos que son aptos para incluirse en esa parte del equipo.
Entrada proporcionada	Cambio en el equipo.
Salida esperada	Lista de objetos capaces de equiparse en esa área.
Salida obtenida	Lista de objetos correcta.
Observaciones / cambios	

21. Al recoger una materia se aumenta la experiencia	
Descripción	Tras recoger una materia, la experiencia tiene que aumentar.
Entrada proporcionada	Recoger una materia.
Salida esperada	La experiencia aumenta 10 puntos.
Salida obtenida	La experiencia aumenta 10 puntos.
Observaciones / cambios	
22. Carga el Fragment de Combinaciones	
Descripción	Se cambia de Fragment y se ha de cargar el Fragment que contiene la interfaz de combinaciones.
Entrada proporcionada	Cambio de pantalla.
Salida esperada	Carga de la pantalla correctamente.
Salida obtenida	Carga de la pantalla correctamente.
Observaciones / cambios	
23. Se cargan los objetos utilizables	
Descripción	Al pulsar sobre los botones de los ingredientes para la combinación, se deben de mostrar los objetos que se pueden usar, aquellos con más de un objeto en el inventario.
Entrada proporcionada	Pulsar sobre los botones.
Salida esperada	Lista con los objetos con una o más unidades.
Salida obtenida	Lista con objetos de más de una unidad.
Observaciones / cambios	
24. Se realiza una combinación exitosa (A+B)	
Descripción	Se eligen dos objetos en los ingredientes y se pulsa sobre el botón de mezcla. La combinación se encuentra en la base de datos con el objeto resultante.
Entrada proporcionada	Dos objetos que pueda resultar una combinación correcta.
Salida esperada	Un nuevo objeto, la pérdida de los objetos usados como ingredientes y la obtención de 20 puntos de experiencia.
Salida obtenida	Se obtiene un nuevo objeto, se pierden los dos objetos y se obtiene la experiencia.
Observaciones / cambios	
25. Se realiza una combinación exitosa (B+A)	
Descripción	La operación es la misma que la anterior, pero con los objetos invertidos.
Entrada proporcionada	La misma que la anterior, pero con los objetos invertidos.
Salida esperada	Un nuevo objeto, la pérdida de los objetos usados como ingredientes y la obtención de 20 puntos de experiencia.
Salida obtenida	Se obtiene un nuevo objeto, se pierden los dos objetos y se obtiene la experiencia.
Observaciones / cambios	
26. Se realiza una combinación fallida	
Descripción	Se eligen dos objetos y se pulsa sobre el botón de mezcla. La combinación específica no está en la base de datos y origina un fallo.
Entrada proporcionada	Dos objetos que no puedan originar una combinación.
Salida esperada	La pérdida de los dos objetos usados y un mensaje anunciando el fallo.
Salida obtenida	Perdida de los objetos y un mensaje.
Observaciones / cambios	

27. Se carga el Fragment de amigos (sin amigos)	
Descripción	Se carga el Fragment que contiene la lista de los amigos vacia.
Entrada proporcionada	Cambio de pantalla.
Salida esperada	Lista vacia.
Salida obtenida	Lista vacia.
Observaciones / cambios	
28. Se carga el Fragment de amigos (con amigos)	
Descripción	Se carga el Fragment que contiene la lista de los amigos.
Entrada proporcionada	Cambio de pantalla.
Salida esperada	Lista de amigos.
Salida obtenida	Lista de amigos.
Observaciones / cambios	
29. Al añadir un amigo se muestran los usuarios necesarios.	
Descripción	Al añadir un amigo se muestran todos los usuarios que hay en la aplicación, excluyendo los que ya son amigos.
Entrada proporcionada	El usuario.
Salida esperada	Lista de usuarios excluyendo los amigos del usuario.
Salida obtenida	Lista de usuarios excluyendo los amigos del usuario.
Observaciones / cambios	
30. Al añadir el amigo se muestra en la lista	
Descripción	Tras añadir y seleccionar un usuario, este se añade a la lista.
Entrada proporcionada	Un usuario del sistema que no es amigo.
Salida esperada	Se añade a la lista del servidor y del dispositivo.
Salida obtenida	Se añade a la lista del servidor y del dispositivo.
Observaciones / cambios	
31. Enviar una materia a un amigo	
Descripción	El usuario renuncia a un objeto para cedérselo a usuario. Este usuario deberá de recogerlo la siguiente vez que se identifique.
Entrada proporcionada	Las coordenadas del usuario, el objeto y el usuario destino.
Salida esperada	Se perderá una unidad del objeto. El usuario destino encontrara en el lugar donde el usuario se encontraba una materia con el objeto seleccionado.
Salida obtenida	El usuario pierde un objeto. El usuario destino no encuentra la materia.
Observaciones / cambios	
32. Se carga el Fragment de combates	
Descripción	Se carga la nueva pantalla.
Entrada proporcionada	Cambio de pantalla.
Salida esperada	Carga nueva pantalla.
Salida obtenida	Carga nueva pantalla.
Observaciones / cambios	
33. Se empieza un combate	
Descripción	Se escoge un enemigo al azar, y se muestra su información en la pantalla al igual que le da al usuario la oportunidad de realizar su primer turno.
Entrada proporcionada	Pulsación del botón.
Salida esperada	La pantalla se carga con la información del enemigo (nombre y barra de vida). Se permite al usuario elegir una acción.

Salida obtenida	La pantalla se carga con la información de un enemigo al azar y se permite al usuario seleccionar una acción.
Observaciones / cambios	
34. Se ataca (ambos sobreviven)	
Descripción	Cuando la batalla esta iniciada, una opción es atacar. Esto causará que el daño se refleje en la barra de vida.
Entrada proporcionada	Botón de atacar.
Salida esperada	Se descontará el daño realizado al enemigo de su vida y se le descontará al jugador el daño infligido.
Salida obtenida	El enemigo y el jugador sufren daño.
Observaciones / cambios	
35. Se ataca (el enemigo muere)	
Descripción	Cuando la batalla esta iniciada, una opción es atacar. Esto causará que el daño se refleje en la barra de vida.
Entrada proporcionada	Botón de atacar.
Salida esperada	Se descontará el daño realizado al enemigo de su vida y esta quedara en 0 o menos. La batalla se detiene. Se aumenta la experiencia del jugador.
Salida obtenida	El enemigo queda con vida 0 o negativa. La batalla se detiene. El usuario recibe experiencia.
Observaciones / cambios	
36. Se ataca (el jugador muere)	
Descripción	Cuando la batalla esta iniciada, una opción es atacar. Esto causará que el daño se refleje en la barra de vida.
Entrada proporcionada	Botón de atacar.
Salida esperada	Se descontará el daño realizado al enemigo de su vida, el usuario recibe el daño y su vida queda a 0 o negativa. La batalla se detiene. El usuario recibe experiencia.
Salida obtenida	El enemigo recibe daño. El usuario recibe daño. La vida del usuario queda a 0 o negativa.
Observaciones / cambios	
37. Se defiende (ambos viven)	
Descripción	Con la batalla iniciada, el usuario decide defenderse. Esto hace que el usuario no pueda hacer daño a cambio de obtener una defensa más alta.
Entrada proporcionada	Botón de defensa.
Salida esperada	El usuario recibe el daño correspondiente a la diferencia entre el ataque y la defensa.
Salida obtenida	El usuario pierde menos daño por disponer de una defensa más alta que el ataque.
Observaciones / cambios	
38. Se defiende (el usuario pierde)	
Descripción	Con la batalla iniciada, el usuario decide defenderse. Esto hace que el usuario no pueda hacer daño a cambio de obtener una defensa más alta.
Entrada proporcionada	Botón de defensa.
Salida esperada	El usuario recibe el daño correspondiente a la diferencia entre el ataque y la defensa. El usuario queda con vida 0 o negativa. Recibe puntos de experiencia.

Salida obtenida	El usuario pierde menos daño por disponer de una defensa más alta que el ataque. Queda con vida 0 o negativa. Recibe puntos de experiencia.
Observaciones / cambios	

ANEXO IV.- MANUAL DE USUARIO

Instalación

Para la instalación del juego se necesita disponer de un dispositivo Android que permita conexión a internet y WIFI que permita la geolocalización. También tiene que permitir instalar aplicaciones de fuentes desconocidas, para ello, vamos a Ajustes, al menú de seguridad y chequeamos Fuentes desconocidas. Esto permite instalar aplicación que no provengan del Play Store, como es el caso.

Tras todo esto. Hay que pasar el archivo *.apk a la memoria del teléfono. Desde nuestro navegador de archivos, ir a la carpeta donde guardamos el archivo y pulsarlo, nos dará la opción de instalarlo. Le damos acceso para los permisos y se instalará.

Identificación y registro

La aplicación necesita de una identificación para poder identificar a los usuarios. Si no se dispone de esta identificación, se puede realizar un registro de manera rápida, indicando el nombre del jugador o seudónimo, un email y contraseña.

Con las credenciales podemos identificarnos en la aplicación.

Jugar

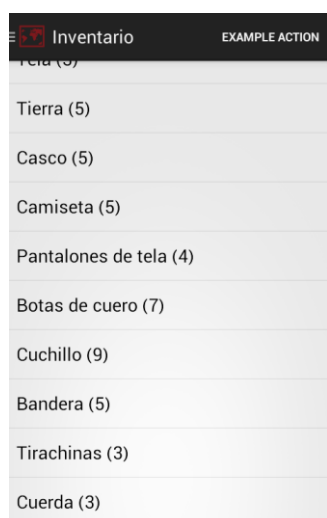
El objetivo del juego es ganar experiencia. Para esto, se puede hacer recogiendo los marcadores que contienen diversos objetos, creando nuevos objetos y combatiendo enemigos.



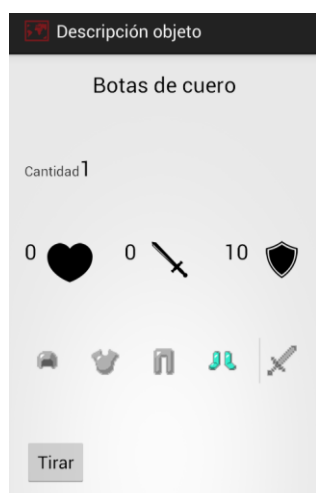
Lo primero que se carga en el juego es el mapa local, con varios marcadores sobre el mapa. Estos marcadores son Materias y contienen objetos que nos pueden servir para más adelante. Nos deberemos de desplazar hasta el punto marcado en el mapa para poder recoger la Materia y obtener de esta manera el objeto que contiene. Si caminamos lo suficiente, nuestro radar detectará más Materias próximas y nos las hará visibles en el mapa para que podamos ir a recogerlas.



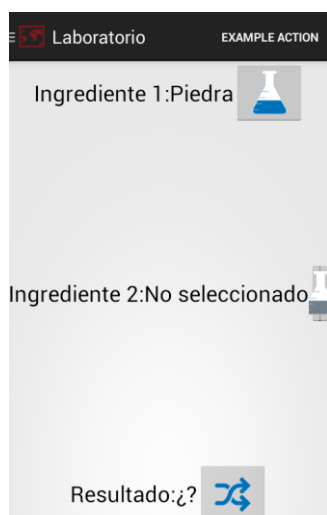
Para cambiar entre pantallas, hay que desplazar el dedo desde el lateral izquierdo de la pantalla hasta el centro de la pantalla y aparecerá el menú de navegación. Con este podremos volver al mapa, además de ir al inventario, al laboratorio, a la arena o incluso ver a nuestros amigos.



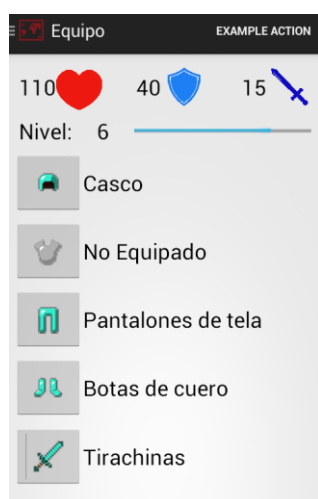
En el apartado de inventario nos aparecerán todos los objetos que hemos recolectado junto con la cantidad. Esto nos puede servir para saber que tenemos y si necesitamos alguna cosa en concreto.



Si pulsamos sobre uno de los objetos que hemos recogido, nos aparecerán los detalles de dicho objeto, como si se puede equipar y en tal caso, cuanta vida, fuerza o protección nos daría. Disponemos de un botón de tirar, por si nos fuera necesario desprendernos de alguno de los objetos que hemos recogido. Para volver a la lista de objetos solo hemos de pulsar el botón atrás del teléfono.

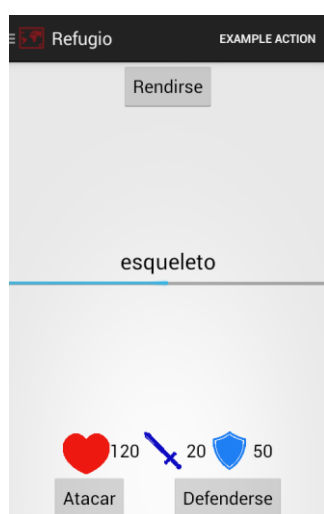


En el laboratorio, podremos seleccionar nuestros objetos que hayamos encontrado para poder crear nuevos y mejorados objetos. Para ello solo hemos de pulsar sobre los botones de vasos de precipitados para seleccionar que objeto queremos y luego pulsar sobre el botón de Mezclar. Por desgracia perderemos los objetos que hayamos usado, pero si hemos elegido correctamente, dispondremos de un objeto nuevo en nuestro inventario. Recuerda que crear nuevos objetos te otorga experiencia, pero desperdiciar los objetos no. Usa los objetos con cabeza y lógica, un trozo de tela y un palo pueden anudarse y formar una bandera, pero juntar dos piedras no sirve para gran cosa.

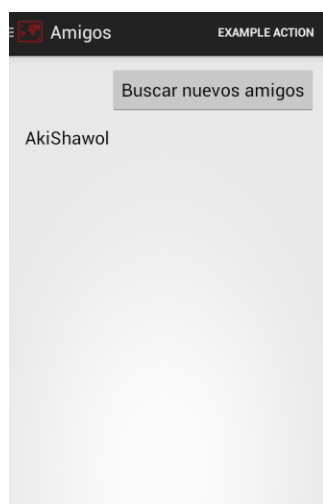


Disponemos de una sección para saber cuál es tu estado, donde se muestra que vida dispones, y que fuerte estas, además de ver lo resistente que eres a los daños. También podremos ver cuanta experiencia has adquirido y en qué nivel estas. Según vayas adquiriendo más experiencia, aumentarás de nivel, esto te permitirá aumentar el nivel del radar y localizar Materias y usar objetos de rangos superiores.

Además, en esta misma sección, puedes cambiar tu equipo, los objetos que tienes asignados a cada parte del cuerpo y que te dan bonificadores a tu estado, como más vida o fuerza. Normalmente el arma aumenta el daño que realizamos al enemigo, mientras que las armaduras nos hacen más resistentes a los ataques.



En la arena se pondrá en valía tu habilidad como guerrero. Aquí podrás luchar contra diferentes y terroríficos monstruos. Podrás ver en todo momento el daño que le has hecho al enemigo y decidir si atacar o defenderte, pero has de tener en cuenta que, escojas lo que escojas, el enemigo siempre atacará después. Planifica bien tu batalla. No debes de preocuparte si tu vida llega a 0, nunca morirás en la arena, solo no conseguirás tantos puntos de experiencia. Y si te ves muy apurado, siempre puedes rendirte, es una pequeña vergüenza, pero se valora más guerrero vivo que muerto, tenlo en cuenta a la hora de planificar tu estrategia.



Por último, pero no menos importante, los amigos. Siempre debes de mantener el contacto con tus amigos cercanos y puedes ayudarles a crecer, para ello, solo debes de regalarles alguno de tus objetos, quizás alguno que ellos no dispongan aun o que necesiten con urgencia. Pero claro, no se puede enviar los objetos tan fácilmente, tu amigo habrá de venir a recoger el objeto para que pueda llevárselo. Así podéis aprovechar y ponerlos al día de vuestros avances.

Así que recuerda, recoge todas las Materias que puedas, prueba a combinar todos los objetos que quieras y se valiente en la arena.