Universidad de Valladolid

E. U. DE INFORMÁTICA (SEGOVIA)

Grado en Ingeniería Informática de Servicios y Aplicaciones

Simple Android Games

Alumno: Vicente Bermejo Bermejo

Tutor: Luís Ignacio Sebastián Martín

Trabajo de Fin de Grado

TÍTULO DEL TFG: Simple Android Game

TITULACIÓN: Grado en Ingeniería Informática de Servicios y Aplicaciones

AUTOR: Vicente Bermejo Bermejo

TUTOR: Luís Ignacio Sebastián Martín

FECHA: 01 de septiembre de 2013

Índice

Prólogo

Los **juegos** llevan bastante tiempo viviendo entre nosotros. Sin embargo, con la llegada de nuevos dispositivos híbridos (capaces de unir capacidades de teléfonos móviles con funciones que hasta hace no mucho tiempo eran exclusivos de ordenadores), el panorama ha empezado a cambiar. Actualmente, parece que los juegos no están destinados a unos cuantos *frikis*, y nos empieza a parecer normal ver a gente seria y respetable jugando con sus teléfonos móviles en público, a periódicos haciéndose eco de noticias relacionadas con las fortunas que han conseguido los **desarrolladores** de algún pequeño juego y, en definitiva, parece que todo el mundo se ha dado cuenta del tirón que tienen estas aplicaciones para los **dispositivos móviles**.

Los **smartphones** están por todas partes. Posiblemente podamos deducir de ésta idea el resto de factores relacionados con los juegos para móviles.

Los precios del hardware no dejan de bajar y todos los días aparecen en el mercado nuevos teléfonos móviles con mejores prestaciones, por ello, este tipo de dispositivos se han convertido en plataformas ideales para juegos. Cuentan con una gran penetración en el mercado. Mucha gente cambia sus viejos teléfonos por los smartphones de última generación (es más, la propia sociedad incita a ello) y se encuentran con un sinfín de aplicaciones a su disposición.

Antiguamente, la gente que compraba una consola o un ordenador potente lo hacía para jugar; hoy en día tienen, en cierta medida, todas estas funcionalidades de forma gratuita en sus teléfonos móviles. Sin necesidad de invertir más dinero podemos tener en la mano una consola de juegos. Basta con meter la mano en el bolsillo, sacar el teléfono móvil y ya estaremos listos para jugar con un sistema que lo integra todo.

Aparte de la ventaja que supone tener el teléfono, Internet, los juegos y otras aplicaciones en un mismo dispositivo, todos estos factores hacen que los juegos que se desarrollan para teléfonos móviles disfruten de un **mercado de consumidores potenciales muy grande**. Como usuarios, podemos entrar en acción al momento, coger un juego que nos parezca interesante y empezar a jugar de inmediato. No hace falta conectar el teléfono al ordenador para acceder a la tienda a través de Internet y descargarlo.

La gran ventaja de los smartphones de última generación también tiene un impacto en las capacidades que tenemos a nuestra disposición como desarrolladores de juegos. Incluso los dispositivos de clase media son capaces de generar juegos parecidos a los que veíamos en consolas como Xbox y PlayStation. Dada la capacidad del hardware que poseen estas plataformas, podemos desarrollar juegos más elaborados que incluyen simulaciones físicas y que ofrecen elementos innovadores.

Con los nuevos dispositivos también aparecen nuevos métodos de control. Ya hay juegos que utilizan el **GPS** y la **brújula** que tienen la mayoría de dispositivos de este tipo. El **acelerómetro** se ha convertido en un elemento habitual en la mayoría de juegos y las **pantallas táctiles** ahora ofrecen al usuario nuevas formas para interactuar con el juego. Comparado con las consolas de juego clásicas, esto representa un nuevo desafío para los desarrolladores de juegos. Ciertamente, ya se ha trabajado mucho sobre todo esto, pero verdaderamente todas estas funcionalidades se puedan aplicar de manera realmente innovadora, lo que a la postre será atractivo para los usuarios.

Los smartphones se pueden adquirir al contratar una nueva conexión de datos. Ya no se usan sólo como teléfonos móviles, sino también para acceder a los sitios más populares de

Internet. Es muy posible que un usuario que tenga un Smartphone termine conectándose a la Web en algún momento.

El hecho de estar permanentemente en **Internet** abre un nuevo mundo de posibilidades para los juegos. La gente puede desafiar a sus amigos, que están en otros puntos del planeta, a jugar una partida rápida al ajedrez, explorar juntos mundos virtuales o retarles a un combate *a vida o muerte* en alguna ciudad imaginaria. Y todo esto ocurre sobre la marcha, cuando vamos en el autobús o en el tren, o cuando estamos apoyados en la esquina de algún local.

Aún sin usar Internet, estos dispositivos nos ofrecen la posibilidad de retar a nuestros amigos cercanos sin necesidad de conectar engorrosos cables. Tecnologías como **Bluetooth** o **Wi-Fi** son ya de lo más común en cualquier teléfono móvil y permiten una conexión muy rápida entre dispositivos, con todas las posibilidades que eso conlleva.

Aparte de la funcionalidad multijugador, las redes sociales, además se han convertido en un punto de partida para juegos móviles. Los juegos permiten que publiquemos en Twitter nuestras puntuaciones o que informemos a nuestros amigos de los últimos logros que hemos conseguido en una carrera de coches. Aunque las redes sociales también existen en el mundo clásico de las consolas, la penetración en el mercado de servicios como Facebook o Twitter es mucho mayor y quita al usuario la presión de tener que trabajar con varias redes al mismo tiempo.

La gran adaptación de los smartphones implica que gente que nunca había sentido interés por este mundo, de repente, lo haya descubierto. Si bien, su idea de un buen juego puede diferir un poco del ideal que tenga un amante de este género.

Debido al uso que hacen de sus teléfonos móviles, los usuarios casuales suelen recurrir a **juegos cortos** con los que divertirse durante un par de minutos, mientras viajan en el autobús o esperan en la cola de su restaurante favorito. Estos juegos son equivalentes a los pequeños programas realizados en Flash a los que mucha gente juega en clase o en la oficina y que tratan de ocultar a toda velocidad cuando alguien aparece por detrás. Deberemos hacernos una pregunta: ¿cuánto tiempo dedicaremos a jugar con nuestro teléfono móvil? ¿Sería posible imaginar a alguien echando una partida rápida de, por ejemplo, un juego de estrategia en tiempo real en uno de estos dispositivos?

Seguro que habría gente dispuesta a ello, pero este grupo es una pequeña minoría. Basta con echar un vistazo al tipo de juegos que alcanzan la parte alta de los rankings de ventas o descargas en **Google Play** y **Apple Store** para darse cuenta de ello. Los juegos más demandados suelen ser pequeñas aplicaciones que se basan en un hecho común: el tiempo medio que le dedica un jugador es de unos pocos minutos pero la dinámica que sigue el juego les incita a volver a intentarlo.

El hecho de que predominen los juegos rápidos viene en parte definido por la propia naturaleza del dispositivo en el que se ejecuta. No olvidemos que estamos hablando de dispositivos inalámbricos, con una batería limitada y con unas dimensiones de pantalla un tanto reducidas que no favorecen su uso prolongado. Además, los *jugadores por excelencia* siempre contarán con una consola en la que *echar un FIFA*, *ir a la guerra* con Call of Duty o *hacer el gamberro* en una gran ciudad de GTA.

Hemos hablado de los smartphones, pero no debemos olvidar otro tipo de dispositivos móviles que tienen aún mayor potencial para el mundo de los juegos y que está experimentando un claro auge. Nos estamos refiriendo, naturalmente, a las **tablets** o tabletas.

En relación al mundo de los juegos, las tabletas nos ofrecen todas las características de los smartphones añadiéndole, generalmente, mejores prestaciones y una mayor superficie de visualización e interactuación con el usuario, lo que hace que estos aparatos sean más aptos para jugar y abren una pequeña puerta a que juegos de otro tipo tengan cabida en estos dispositivos. Y es que, poca gente tendría la tentación de echar una partida con simulador de carreras de coches en una pantalla de 3 pulgadas, pero posiblemente más gente estaría dispuesta a correr por las calles virtuales de Mónaco en un Fórmula 1 que pueden ver en una pantalla de 7 pulgadas o más.

Uno de los puntos más atractivos para los desarrolladores independientes son los modestos requisitos necesarios que presentan estos dispositivos. En el caso de Android, esta barrera es especialmente baja: basta con conseguir el SDK y ponerse a programar. Ni siquiera hará falta un dispositivo porque se puede usar un emulador (aunque siempre es recomendable contar con un dispositivo físico sobre el que probar nuestras aplicaciones desarrolladas para ver cómo se comportan en un entorno de trabajo real). La naturaleza abierta de Android también genera mucha actividad en la Web. En Internet se puede encontrar información sobre todos los aspectos de la programación de los sistemas. Además, para acceder a ellas no hará falta firmar ningún tipo de acuerdo.

En el momento de comenzar este proyecto, resultaba llamativo que los juegos más populares estaban siendo desarrollados por pequeñas compañías o equipos independientes de desarrolladores. Muchas empresas de renombre aún no habían accedido a este mercado o lo habían hecho sin éxito. Posteriormente algunas de esas empresas vieron el filón que estaban desaprovechando y, a día de hoy, no es difícil encontrar en Google Play juegos de, por ejemplo, la gigante empresa de los videojuegos **Electronic Arts** (EA). De hecho, EA adquirió la compañía PopCap Games[1] que desarrollaba, entre otras cosas, juegos para dispositivos móviles.

El entorno también invita a hacer experimentos y ser innovador. Hay mucha gente que se pasea por Internet en buscar de algún juego que les parezca entretenido, gente que estará encantada de probar nuevas ideas y mecánicas de juego innovadoras. Ya sabemos que la experimentación en las plataformas de juego clásicas, como los ordenadores o las consolas, pueden tener resultados desastrosos y ser un completo fracaso. Sin embargo, Google Play nos permite llegar a un público más grande, que está deseoso de probar nuevas ideas y sin que les cueste nada llegar a ellas. Muchos desarrolladores han obtenido grandes ingresos a través de las descargas que se han hecho desde Google Play (antes Android Market), aunque hay que decir que muy pocos (por no decir ninguno) conocen las reglas que hay que seguir para que un juego de este tipo se convierta en un éxito. Un claro ejemplo es Angry Birds, un juego simple que nadie pensaba que pudiera llegar a niveles tan altos de popularidad.

En lo referente a la innovación, parece que se acercan tiempos aún más revolucionarios, con dispositivos que pueden hacernos cambiar definitivamente la forma en la que nos relacionamos con la tecnología y con el mismo mundo real que nos rodena. Pensemos por

[1] Electronic Arts compra PopCap Games

un momento en **Google Glass,** ¡un ordenador totalmente funcional en unas gafas! (que por cierto, funciona con Android). La sobreposición del mundo virtual en el mundo real con tan solo unos comandos de voz. No tardaremos ni un segundo en darnos cuenta de la gran cantidad que posibilidades que esto nos ofrece. Habrá que ver cómo evoluciona esta tecnología y la aceptación que tienen por parte de los usuarios, pero, por si acaso, vayamos pensando en ello.

1. Identificación del proyecto.

El proyecto se basa en la creación de una aplicación para dispositivos Android (con todo el proceso de desarrollo de software asociado) que sirva de base para la ejecución de juegos simples en el terminal móvil. Así, se pretende conseguir un diseño general que permita realizar partidas multijugador en dispositivos físicamente separados.

Actualmente existe gran variedad de juegos individuales para Android que dan soporte multijugador, la diferencia fundamental respecto a este proyecto radica en que con él se busca crear un soporte que maneje las comunicaciones entre dispositivos, el cual permita ser aplicado en diferentes juegos. De esta manera, los juegos que funcionen bajo el *paradigma* que este proyecto va a definir, ofrecerán la posibilidad de realizar partidas multijugador entre varios usuarios (cada uno con su correspondiente dispositivo) en el que se intercambiarán mensajes de manera síncrona en comunicaciones punto a punto.

Para poner en práctica esta base común, se desarrollará algún juego clásico que funcione bajo Android, con toda la lógica y recursos propios (pantallas, imágenes, sonidos, etc.) asociados al mismo. Obviamente, el juego desarrollado, aunque sea simple y limitado, deberán ser totalmente funcional y *jugable*.

Es decir, la aplicación constará de al menos un juego simple que permitirá que dos usuarios puedan jugar entre sí (cada uno con su correspondiente dispositivo). Además, en función de la lógica propia de cada juego, el usuario podrá jugar individualmente (jugador rival simulado por el dispositivo) o contra otro usuario compartiendo dispositivo.

Es evidente que cada juego tiene su propia lógica específica, por lo que el sistema deberá ser lo suficientemente general como para abarcar el mayor número de tipos de juegos posibles sin perder la esencia básica del proyecto, que no deja de tener un enfoque simplista, pues de otra manera el tiempo requerido para su realización sería completamente inabordable para alguien que carece de experiencia en estas lides. En resumen, no espere ver un juego innovador, con unos gráficos espectaculares o simulaciones físicas realistas y tenga siempre en cuenta el título de este proyecto: *Simple Android Games*.

Por todo ello, este proyecto sirve como excusa para acercarme (aunque sea levemente) al mundo del desarrollo de juegos para dispositivos de algún tipo, un terreno completamente inexplorado por mi hasta ahora y sobre el que tenía cierta curiosidad. Así mismo, el desarrollo de este proyecto también es una gran oportunidad para mejorar y aplicar de manera distinta los pequeños conocimientos sobre programación para dispositivos Android con los que ya contaba.

2. Descripción general del proyecto.

2.1.Objetivos

El objetivo principal del proyecto es desarrollar una aplicación Android para dispositivos móviles (smartphones y tablets) con el fin de dar soporte a juegos simples desarrollados para dicho sistema operativo.

Para ello el proyecto se divide en subobjetivos que se deben alcanzar ordenados según el grado de prioridad, que son:

- Definición de protocolo general de comunicación para juegos: encargado de administrar y gestionar los mensajes que se envían entre los dispositivos.
- Desarrollo de la lógica de, al menos, un juego clásico: encargado de llevar a cabo la lógica específica de cada juego.
- Diseño de elementos de juego: encargado de la gestión de la parte visual de los juegos desarrollados.
- Aplicación específica del protocolo sobre los juegos desarrollados: se encargará de administrar y gestionar cada tipo de comunicación soportada en principio por el sistema. Estos son:
  - Comunicaciones Bluetooth.
  - Comunicaciónes Wi-Fi.

2.2.Cuestiones metodologicas

Para la realización del proyecto se ha seguido un **ciclo de vida clásico o en cascada**, que es el enfoque metodológico que ordena rigurosamente las etapas del proceso para el desarrollo de software, de tal forma que el inicio de cada etapa debe esperar a la finalización de la inmediatamente anterior.

Este ciclo de vida se divide en las siguientes etapas o fases de desarrollo:

- Análisis: En esta fase se analizan las necesidades de los usuarios finales del software para determinar qué objetivos debe cubrir. De esta fase surge una memoria llamada DRS (documento de requisitos del sistema), que contiene la especificación completa de lo que debe hacer el sistema sin entrar en detalles internos, y del DAS (documento de análisis del sistema).
- Diseño: Se descompone y organiza el sistema en elementos que puedan elaborarse por separado, aprovechando las ventajas del desarrollo en equipo si lo hubiera. Como resultado surge el DDS (Documento de Diseño del Software), que contiene la descripción de la estructura relacional global del sistema y la especificación de lo que debe hacer cada una de sus partes, así como la manera en que se combinan unas con otras.
- Codificación: Es la fase en donde se implementa el código fuente, pudiendo hacer uso de prototipos así como de pruebas y ensayos para corregir errores.
- Pruebas: Los elementos, ya programados, se ensamblan para componer el sistema y se comprueba que funciona correctamente y que cumple con los requisitos, antes de ser entregado al usuario final.

- Mantenimiento: En esta fase, que tiene lugar después de la entrega, se asegura que el sistema siga funcionando y adaptándose a nuevos requisitos.

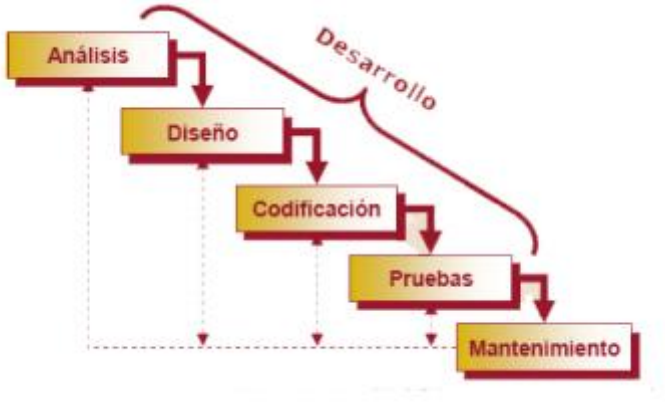


Figura 1. Etapas ciclo de vida en cascada.

También se han utilizado algunas técnicas basadas en **Métrica 3,** que es una metodología de planificación, desarrollo y mantenimiento de sistemas de información, promovida por el Ministerio de Administraciones Públicas del Gobierno de España para la sistematización de actividades del ciclo de vida de los proyectos software en el ámbito de las administraciones públicas.

En cuanto a la programación, se ha utilizado un modelo de **programación orientada a objetos**. Éste es un paradigma de programación que utiliza objetos y sus interacciones para diseñar aplicaciones y programas informáticos. Está basado en varias técnicas, incluyendo herencia, abstracción, polimorfismo y encapsulamiento. De este modo, el lenguaje utilizado será Java, que está orientado a objetos, y que es el principal lenguaje en el que se realizan las aplicaciones Android.

En el aspecto de técnicas utilizadas, se ha escogido **UML**, ya que la programación orientada a objetos es un complemento perfecto de UML.

UML es un *lenguaje de modelado* para especificar o para describir métodos o procesos. Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo. Se puede aplicar en el desarrollo de software entregando gran variedad de formas para dar soporte a una metodología de desarrollo de software (tal como el Proceso Unificado Racional o RUP), pero no especifica en sí mismo qué metodología o proceso usar.

UML no puede compararse con la programación estructurada, pues UML significa Lenguaje Unificado de Modelado, no es programación, sólo se representa la realidad de una utilización en un requerimiento.

2.3.Entorno de la aplicación (marco tecnológico).

El proyecto se enmarca en el entorno tecnológico del software para dispositivos móviles inteligentes (smartphones o tablets). Este entorno tecnológico se ha visto favorecido en los últimos años por la gran expansión que ha sufrido el mercado y la demanda de este tipo de dispositivos. El mercado ya no demanda móviles que sólo puedan realizar y recibir llamadas o SMS, sino que se demandan dispositivos que ofrezcan una amplia variedad de opciones, por ejemplo, que soporten completamente un cliente de correo electrónico, con la funcionalidad completa de un organizador personal, o que permitan la instalación de programas para incrementar el procesamiento de datos y la conectividad. Además, el mundo tecnológico y de la computación avanza hacia dispositivos cada vez más pequeños, inalámbricos y veloces que tengan los recursos necesarios para facilitar un poco más la vida de las personas y acceder a una gran cantidad de información de manera rápida desde cualquier lugar.

Este gran auge viene también motivado, en cierta medida, por la aparición de distintos sistemas que mejoran y ofrecen todas esas posibilidades en estos dispositivos, además de las constantes mejoras aparecidas en cuanto a hardware. Pues bien, dentro de este marco tecnológico reside Android, que es el sistema base de este proyecto.

Android es un sistema operativo basado en GNU/Linux diseñado originalmente para dispositivos móviles, pero que posteriormente se expandió su desarrollo para soportar otros dispositivos tales como tablets, reproductores MP3, netbooks, PCs, televisores, lectores de e-books e incluso, se han llegado a ver implantado en microondas y lavadoras.

Inicialmente fue desarrollado por Android, Inc., aunque Google no tardó que adquirirlo completamente (anteriormente ya había prestado ayuda económica al proyecto). Android fue presentado en 2007 y el primer dispositivo que funcionaba con este sistema salió a la venta a mediados de 2008, por lo que podemos decir que es una tecnología relativamente reciente.

La estructura del sistema operativo Android se compone de aplicaciones que se ejecutan en un framework **Java** de aplicaciones orientadas a objetos, sobre el núcleo de las bibliotecas, en una máquina virtual Dalvik con compilación en tiempo de ejecución. Las bibliotecas escritas en lenguaje C incluyen un administrador de interfaz gráfica (surface manager), un framework OpenCore, una base de datos relacional SQLite, una API gráfica OpenGL ES 2.0 3D, un motor de renderizado WebKit, un motor gráfico SGL, SSL y una biblioteca estándar de C Bionic.

Los componentes principales del sistema operativo de Android:

- **Aplicaciones**: las aplicaciones base incluyen un cliente de correo electrónico, programa de SMS, calendario, mapas, navegador, contactos y otros. Todas las aplicaciones están escritas en lenguaje de programación Java.
- **Marco de trabajo de aplicaciones**: los desarrolladores tienen acceso completo a los mismos APIs del framework usados por las aplicaciones base. La arquitectura está diseñada para simplificar la reutilización de componentes; cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación puede luego hacer uso de esas capacidades (sujeto a reglas de seguridad del

framework). Este mismo mecanismo permite que los componentes sean reemplazados por el usuario.

- **Bibliotecas**: Android incluye un conjunto de bibliotecas de C/C++ usadas por varios componentes del sistema. Estas características se exponen a los desarrolladores a través del marco de trabajo de aplicaciones de Android.
- **Runtime de Android**: Android incluye un set de bibliotecas base que proporcionan la mayor parte de las funciones disponibles en las bibliotecas base del lenguaje Java. Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual Dalvik. Dalvik ha sido escrito de forma que un dispositivo puede correr múltiples máquinas virtuales de forma eficiente. Dalvik ejecuta archivos en el formato Dalvik Executable (.dex), el cual está optimizado para memoria mínima. La Máquina Virtual está basada en registros y ejecuta clases compiladas por el compilador de Java que han sido transformadas al formato .dex por la herramienta incluida "dx".
- **Núcleo Linux**: Android depende de Linux para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, pila de red y modelo de controladores. El núcleo también actúa como una capa de abstracción entre el hardware y el resto de la pila de software.

Esto se puede apreciar de manera más clara en la siguiente imagen, que representa la arquitectura de Android con sus partes diferenciadas. De manera resumida podríamos decir que Android funciona sobre un núcleo Linux con una capa Java intermedia (máquina virtual Dalvik). Es por ello que podemos encontrar bibliotecas principales (incluso aplicaciones) que *se saltan* esa capa Java y compilan directamente sobre el núcleo Linux, y otras bibliotecas que hace una compilación previa en el entorno de ejecución.

Figura 2. Arquitectura Android

En lo referente a Android como plataforma para desarrollar aplicaciones, hay que decir que se sigue un **patrón MVC,** donde las clases Java representan el **controlador**, las **vistas** son archivos XML y el **modelo** vendría dado por los propios datos obtenidos de base de datos o servicios externos.

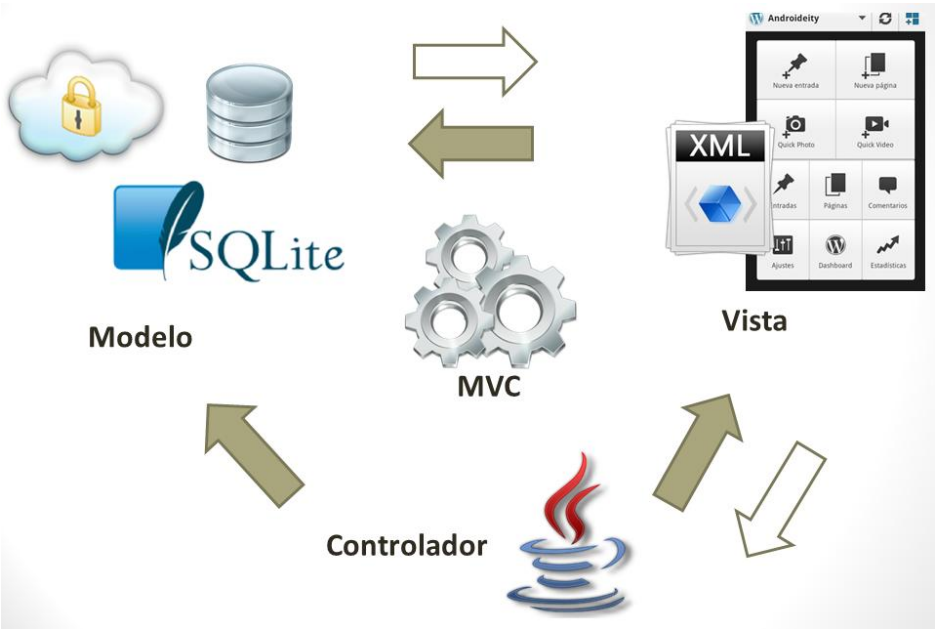


Figura 3. MVC Android

Por otra parte, Android se desarrolla de forma abierta y se puede acceder tanto al código fuente, como a la lista de incidencias donde se pueden ver problemas aún no resueltos y reportar problemas nuevos. Esto unido a que desde su comienzo ha sido altamente personalizable, desemboca en uno de los principales problemas de esta plataforma: la **fragmentación**.

Android es un sistema operativo fragmentado, esto quiere decir que no siempre será posible contar con todas las funcionalidades que ofrece el sistema en un determinado dispositivo. Al ser de código abierto, existen multitud de versiones (y subversiones) de Android; los fabricantes de los dispositivos (incluso los propios usuarios) modifican con frecuencia ciertos aspectos del sistema según sus preferencias o requerimientos. A un desarrollador esta situación le supone, en ocasiones, tomar decisiones que pueden limitar el mercado de su aplicación. Es decir, un desarrollador puede *crear* una nueva aplicación que aproveche las características más nuevas y recientes de la última versión de Android, pero en ese caso, perderá una buena cuota de mercado, ya que con toda probabilidad esas nuevas funcionalidades aún no estarán disponibles en gran parte de dispositivos. Por otra parte, el desarrollador puede tratar de hacer su aplicación compatible con el mayor número de dispositivos posible (con distintas características, tamaños de pantalla, etc), lo que le supondrá un mayor esfuerzo de desarrollo y testing.

Como dato significativo, hay que decir que, según un reciente informe (agosto de 2013)[2] elaborado por IDC (corporación estadounidense centrada en estudios de mercado del mundo de las TI), Android posee en el segundo trimestre de 2013 una cuota de mercado del 79,3% en teléfonos inteligentes, frente al 13,2% de iOS, el 3,7% de Windows Phone o el 2,9% de BlackBerry OS. Respecto al mismo periodo del año anterior, se puede apreciar un crecimiento significativo (69,1% de cuota de mercado). A la luz de estos datos, podemos ver que Android sigue creciendo, es decir, cada vez es mayor el número de dispositivos de este tipo en el mercado, lo que a nosotros (como desarrolladores) nos ofrece al mismo tiempo un mayor número de clientes potenciales.

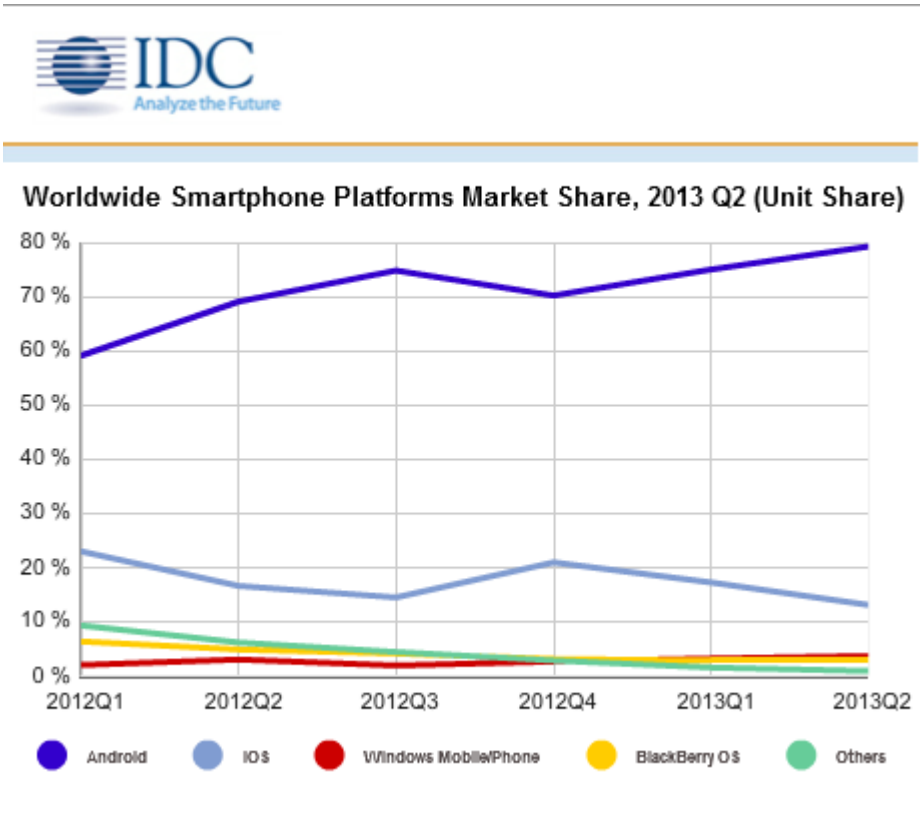


Figura 4. Cuota de mercado smartphones por sistema.

Por todo esto, este proyecto supondrá una nueva, de las innumerables aplicaciones y posibilidades que este sistema y este marco tecnológico ofrecen.

Ya hemos hablado de Android de una manera general, poniendo en claro el entorno tecnológico de base en el que se moverá nuestra aplicación; pero falta relacionarlo con otro aspecto básico que define este proyecto. Esto no es otra cosa que el desarrollo o la programación orientada a obtener como resultado un juego.

[2] Apple Cedes Market Share in Smartphone Operating System Market as Android Surges and Windows Phone Gains, According to IDC

Todo el mundo convendrá que desarrollar un juego no es exactamente lo mismo que desarrollar la típica aplicación CRUD de gestión. Por ello, habrá que tener en cuenta una serie de aspectos a la hora de llevar a cabo el proyecto.

Desarrollar juegos es un trabajo duro, no porque se trate de una ciencia compleja, sino por la cantidad de información que hay que asimilar antes de crear el juego en cuestión. Desde el punto de vista de la programación, debemos prestar atención a cosas tan mundanas como la lectura y escritura de datos en un archivo (I/O), el sistema que se utilizará para controlar el juego, la programación del audio y de los gráficos y el código para trabajar en red. Y sólo estamos hablando de conceptos básicos. Además, tendremos que diseñar la mecánica real del juego, que estará encima de todo esto. El código también necesita su estructura y no siempre resulta obvio generarla. Tenemos que decidir cómo se moverá el mundo de nuestros juegos. ¿Usaremos un dispositivo físico para desarrollar el código o trabajaremos directamente con un simulador? ¿Qué unidades y escalas emplearemos en nuestro juego? ¿Cómo las traduciremos a las medidas de la pantalla?

Hay otro problema que la mayoría de los programadores noveles pasan por alto: hay que dedicar algo de tiempo a diseñar el juego antes de ponerse a programar. De hecho, hay gran cantidad de proyectos que nunca han llegado a ver la luz y se han quedado sólo en una versión de pruebas porque el programador no tenía claro el comportamiento de la versión final. No hablamos de la mecánica de un juego básico donde el jugador disparará a todo lo que se mueva por la pantalla. Ésta es la parte más sencilla: con unas cuantas teclas y el ratón basta para controlar el movimiento del personaje. Nos estamos refiriendo a preguntas como ¿habrá pantalla de presentación? ¿Qué elementos aparecerán en la pantalla del juego? ¿Cómo diseñar la interfaz para que funcione en pantallas de diferentes tamaños? Y, la verdad, no hay una respuesta universal para todo. No hay una respuesta estándar o una regla preestablecida. *Cualquier cosa que funcione será válida*; sin embargo, debemos buscar siempre la solución más sencilla.

En cierto modo, un juego es parecido a cualquier programa que trabaje con una interfaz para el usuario. Si el sistema operativo que se encuentra detrás emplea ventanas, nuestro juego deberá aparecer dentro de alguna de ellas. Hay que tener en cuenta que este modelo lo utilizan la mayoría de los sistemas operativos en la actualidad. La ventana hace las funciones de contenedor. Pensemos en ella como el lienzo donde dibujaremos el contenido del juego.

La mayoría de los sistemas operativos permiten que el usuario interactúe con la ventana presionando alguna tecla o tocando la pantalla (entradas).

El módulo del gestor de ventanas del SO y la aplicación son los responsables de la configuración de la ventana y de asegurarse de que sólo se emplea un único componente IU para rellenarla. Más tarde se dibujará este componente en la ventana y será utilizado para detectar si el usuario ha tocado la pantalla o ha presionado alguna tecla. Para dibujar los componentes de la interfaz de usuario es posible recurrir a la CPU o realizar aceleración por hardware si se trabaja con **OpenGL ES**.

Pero volvamos a lo básico, ¿cómo funciona un juego? ¿Cuál es su estructura?

Cuando se ejecuta un videojuego lo que realmente se hace es simular un universo virtual formado por los elementos que conforman el nivel o la pantalla en la que se encuentre el juego (personajes, enemigos, escenario, objetos con los que interaccionar, etc.).

Se trata de hacer creer al usuario que ese universo que se le está mostrando tiene vida propia. Realmente esta simulación de nuestro universo virtual se hace tomando diferentes **frames** y mostrándolos en pantalla. Es decir, cada cierto tiempo se *hace una foto* del universo que se muestra por pantalla. Al mostrar suficientes *fotos* por segundo el usuario tendrá la sensación de movimiento. El universo creado para el videojuego cobrará vida. El principio es el mismo que el efecto producido cuando vemos pasar rápidamente hojas con dibujos que varían muy poco entre sí.

Ahora bien, un videojuego requiere de la interacción del usuario, que no sabemos cómo va a reaccionar ante las situaciones que se sucedan, es decir, no podemos *hacer fotos* prefijadas del universo en cada situación para ir mostrándolas por pantallas; y aunque esto fuera posible, en ningún caso sería práctico, ya que habría que almacenar gran cantidad de información sobre cada uno de los posibles estados en los que se puede encontrar cada elemento de nuestro universo en cada frame. Esto es intratable.

En lugar de eso, lo que se hace es calcular la posición y el estado de cada elemento en el instante de tiempo que se quiere representar. Una vez realizados estos cálculos, se *dibujan* los elementos en la pantalla. Esto es lo que se conoce como proceso de **renderizado**.

A grandes rasgos, el sistema que siguen todos los juegos que trabajan a alto nivel se rige por los siguientes pasos: Primeramente se inicializan todos los recursos necesarios (imágenes, sonidos, etc.). Seguidamente se inicia el **bucle del juego**, en el que se procesan las entradas del usuario, se actualiza el estado de los elementos del juego y se muestran en pantalla (o se produce la salida que corresponda, por ejemplo se reproduce un sonido). Este bucle se repite hasta el infinito, o hasta que el usuario decida abandonar el juego. Finalmente, cuando termina el bucle principal se limpian y liberan todos los recursos.

Podemos decir que cada *vuelta* del bucle corresponde con un frame o fotograma del juego y que cuantas más *vueltas* del bucle se sucedan en un segundo mayor será **la velocidad de reproducción** a la que se mueve nuestro juego. Evidentemente, si podemos dibujar más fotogramas en un segundo el juego será fluido para el usuario.

Por ello, la mejor forma de determinar la velocidad a la que se mueve nuestro programa es contar el número de fotogramas que puede dibujar por segundo. Estamos hablando de algo llamado **sincronización vertical** o *vsync*. Es algo que está disponible en todos los dispositivos Android del mercado y limita la cantidad máxima de fotogramas por segundo a 60.

Aunque 60 fotogramas por segundo es una velocidad ideal, es algo que es muy difícil de conseguir por problemas de rendimiento. Los dispositivos con pantallas más grandes de lo habitual o con mucha resolución tienen que rellenar muchos píxeles en cada *barrido*, aún en el que caso en el que estén limpiando el contenido de la pantalla. Esta es una de las razones por la que se considera razonable 30 fotogramas por segundos como velocidad efectiva constante. Como curiosidad y para hacernos una idea, se necesitan 24 fotogramas movidos en un segundo para generar una sensación de movimiento continuo a una velocidad normal. 24 fps (fotogramas por segundo) es la velocidad a la que el ojo humano percibe un movimiento continuo. Por esta razón, 24 fps son los que se utilizan en el mundo del cine para las películas. De hecho, se dice que 40 fps es lo máximo que el ojo humano podría percibir, aunque no todo el mundo llega a esa cantidad.

De esta manera estableceremos los límites en los que deben moverse nuestros juegos.

Un aspecto importante, y que podemos pasar por alto a priori, es que el movimiento de los elementos en pantalla debe ser independiente de la velocidad de reproducción.

Supongamos que el dispositivo del usuario puede ejecutar un juego a una velocidad de 60 fotogramas por segundo. Si dicho juego tiene un elemento que se mueve por la pantalla avanzando, por ejemplo, 100 píxeles en el eje X en 100 fotogramas, tardará 1,66 segundos en alcanzar la posición 100 (recordemos que la velocidad es de 60 fotogramas por segundos). Ahora, asumamos que un segundo jugador utiliza el juego en un dispositivo diferente, el cual es capaz de reproducir el juego con una velocidad de 30 fotogramas por segundo. Es decir, cada segundo nuestro objeto avanza 30 píxeles, por lo que necesitaría 3,33 segundos para hacer el mismo recorrido.

Evidentemente, esta situación no es deseable. Sería inaceptable que el movimiento del personaje por la pantalla dependiese de la velocidad del dispositivo. Este hecho cambiaría completamente la experiencia del jugador. Imaginemos por un momento un juego de carreras en el intervienen varios usuarios con distintos dispositivos. En condiciones normales, ganaría siempre el usuario que contara con un dispositivos de gama superior (muy torpe debería ser este usuario para que no sucediera así).

Para evitar que esto ocurra, hay que conseguir que los elementos del juego se muevan independientemente de la velocidad de reproducción del dispositivo. En lugar de mover nuestros elementos del juego una cantidad determinada en cada fotograma, se debe especificar la velocidad del movimiento en unidades por segundo. Pero no basta con decir que un objeto debe moverse una determinada cantidad de píxeles por segundo, evidentemente, se debe proporcionar información sobre la cantidad de tiempo que ha pasado desde la última vez que se movió el elemento del juego. Se debe trabajar, por tanto, con intervalos de tiempo desde la última *actualización* del objeto en pantalla; de manera que si un dispositivo puede *refrescar* la pantalla un mayor número de veces por segundo que otro, un objeto que se mueva en el primer dispositivo avanzará menor cantidad de píxeles en cada actualización de lo que lo hará en el segundo dispositivo. Y de esta manera se consigue la misma velocidad de movimiento en cualquier dispositivo. Evidentemente, esto no es una solución mágica y en dispositivos que no cuenten con unas prestaciones mínimos nos vamos a encontrar los llamados *tirones* en el juego.

Todo lo dicho anteriormente es sólo la idea básica que hay bajo todo juego. Además, tendremos que gestionar cada una de las posibles entradas y salidas del juego, tener en cuenta y manipular la calidad y la compresión del sonido que queramos usar, trabajar con rasters, pixeles y framebuffers para mostrar imágenes, aplicar correctamente composiciones de color en diferentes formatos, diferente número de bits para cada uno, valores alfa en la mezcla de color, etc. (por ejemplo ARGB8888, RGB565, ABGR8888 o BGR565); manejo y posicionamiento de vértices y triángulos para formar las imágenes que queremos representar en pantalla, escalas, rotaciones, uso de matrices para guardar los estados de OpenGL, introducción de texturas, representación de simulaciones, implementación de motor de físicas si tenemos elementos que se mueven por la pantalla para simular velocidades, aceleraciones, etc.; añadir detección de colisiones a cada elemento individual de la pantalla, y, en definitiva, gran cantidad de elementos de muy bajo nivel que pueden hacer de nuestro trabajo una verdadera pesadilla.

Por esta razón, y puesto que no es conveniente *volver a reinventar la rueda*, para este proyecto se usará un **motor de juegos** que nos ahorra tener que implementar a tan bajo nivel, permitiéndonos centrarnos en el propio desarrollo de los aspectos del juego.

En general, un motor de juegos define cómo se deben hacer las cosas, proporcionando modelos fáciles de comprender y especializados en realizar una tarea común, utilizando una arquitectura genérica. En general, un buen motor acelera notablemente el desarrollo inicial del juego.

Para la implementación de los juegos de este proyecto se hará uso de **AndEngine**, que es un motor Java de código abierto para desarrollar juegos en 2D mediante OpenGL para Android y con terminología propia. AndEngine además cuenta con una serie de extensiones que permiten separar funcionalidades para realidad aumentada, multijugador, motor de físicas, scripting y otra serie de elementos que no serán usados en este proyecto.

Por otra parte, no es lo mismo desarrollar un juego en una plataforma clásica, que para un dispositivo móvil. En general, tendremos menos recursos para ejecutar nuestros juegos y una batería limitada. Un aspecto fundamental, por tanto, será el hecho de que hay que realizar una correcta gestión de los recursos a nuestra disposición. Debemos tenerlo muy en cuenta y actuar en consecuencia. Sobre esto existen dos consejos básicos que todo desarrollador de juegos para Android debe conocer: por una parte, no hacer trabajo innecesario y por otra, evitar *pedir* memoria en la medida de lo posible (reutilizar objetos por ejemplo).

3. Descripción general del producto.

Anteriormente se ha comentado en qué consiste el producto final que se pretende obtener de este proyecto y se ha puesto en el contexto en el que se define; en este apartado se dará una visión general del sistema, definiendo sus límites, funcionalidades básicas y los usuarios a los que va dirigido.

Podemos hablar de un producto que se presenta como un **sistema cliente-servidor**, aunque en realidad, a alto nivel, no hay grandes diferencias entre lo que hace el cliente y el servidor. Esto es debido a que se ha optado por que cada uno de los elementos cuente con la lógica del juego correspondiente. Se podría haber desarrollado un cliente ligero que sólo contara con la interfaz gráfica y que intercambiara mensajes, pero esto supondría contar con dos aplicaciones separadas, algo que no tiene mucho sentido si se quiere dar la posibilidad al usuario de jugar partidas individuales sin conexión. Además, al contar con la lógica de negocio en ambos extremos de la comunicación, nos podemos ahorrar unos cuantos intercambios de mensajes y hacer la comunicación más fluida. Mejor incluir toda la funcionalidad en una sola aplicación.

Evidentemente, esto tiene sus inconvenientes. Se ha optado por dar prioridad a comunicaciones rápidas de corto alcance (Wi-Fi y Bluetooth) y a incluir la lógica de cada juego dentro de la propia aplicación Android (necesario para poder hacer uso de esas comunicaciones de corto alcance sin contar con Internet); esta situación hace que sea necesario incluir la misma lógica de negocio en un elemento que inicialmente no estuviera pensado para funcionar en este sistema. Por ejemplo, imaginemos que una vez terminado este proyecto queremos que nuestros juegos estén disponibles para ser jugados en Facebook. Para ello sería necesario incluir esa lógica propia del juego en la aplicación para Facebook. Otra posibilidad sería montar un servidor propio por el que pasaran todas las comunicaciones entre las diferentes aplicaciones que puedan hacer uso de nuestro sistema. Nuevamente estamos hablando de centralizar la lógica de negocio y de aplicaciones que no serían otra cosa que clientes ligeros. En cualquier caso, seguiríamos necesitando que nuestro dispositivo móvil cuente con esa lógica para poder funcionar sin Internet.

Se ha optado por esta vía puesto que el proyecto se enfoca más a Android (que como ya se ha comentado al inicio, cuenta con gran número de clientes potencias).

En cualquier caso, el sistema deja la puerta totalmente abierta a incluir un servidor central que sirva de nexo en Internet entre dispositivos, ya que no sólo se ha tratado de generalizar una base sólida para que podamos desarrollar juegos multijugador para las comunicaciones del tipo *prefijado*, sino que, a alto nivel, el modo de comunicación es totalmente transparente y la inclusión de nuevas formas de comunicación no afectaría en ningún caso a las ya implantadas. Es decir, en cualquier momento podríamos incluir nuevos modos de comunicación a nuestros juegos adaptando, por ejemplo, la biblioteca de Android para comunicaciones NFC a nuestro *paradigma*.

Sin embargo, aquí ya estamos entrando demasiado en temas de diseño que veremos más adelante.

Retomando el tema, a efectos prácticos tenemos un servidor al que se conecta un cliente y a partir de ese momento pueden intercambiar mensajes de forma bidireccional indistintamente. Al usuario de la aplicación no le va a suponer ninguna diferencia actuar como cliente o como servidor (más allá de que el que actúa como servidor será normalmente a quien corresponda el primer turno de juego).

Por tanto, el sistema debe dar la posibilidad de enviar y recibir mensajes tanto a cliente como a servidor, produciendo una reacción por parte del receptor cuando el mensaje llegue a su destino.

Por otra parte, el proyecto presenta dos juegos simples que funcionan por turnos de mecánica clásica. Estos juegos son el **Tres en Raya** (Tic Tac Toe) y el **Hundir la Flota** (Battleship). Estos juegos ofrecen la posibilidad de realizar partidas multijugador mediante el sistema que se acaba de comentar o jugar *contra la máquina*, en lo que sería una simulación del jugador rival. Sobre esto, se aplican conceptos básicos de **teoría de juegos.**

El proyecto se dirige a un amplio espectro de usuarios (con progresión ascendente), que necesitan tener conocimientos muy básicos de manejo de aplicaciones. En general, cualquier usuario que posea un dispositivo móvil compatible con la aplicación debería tener los conocimientos necesarios para manejarla sin problemas, y más teniendo en cuenta que los juegos implementados son muy conocidos y no suponen ninguna novedad.

Además, este proyecto también puede ser de utilidad para desarrolladores de juegos para Android que quieran añadir comunicaciones rápidas de corto alcance para posibilitar partidas multijugador a sus creaciones. Con la base creada, esto se podría realizar sin apenas esfuerzo de desarrollo.

4. Planificación y presupuesto.

Estimación inicial.

Para la estimación inicial presupuestaria se ha utilizado el modelo constructivo de costes **COCOMO** (Modelo Constructivo de Costes).

Antes de empezar con el modelo se estimó un tamaño de 5 KLDC (miles de líneas de código). Así, se decidió que el modo de desarrollo a utilizar fuera el **Orgánico** ya que el proyecto no iba a ser tan grande para realizar las estimaciones en función de otro modelo en otro modelo diferente (estimación menor de 50 KLDC).

Para realizar el modelo necesitamos dos variables propias de nuestro proyecto, la cantidad de líneas de código KLDC (dato estimado) y el factor de complejidad total, que se obtiene mediante la multiplicación de los valores evaluados en 15 factores específicos de coste que se observan en la siguiente tabla (se muestran sombreados los valores seleccionados para este proyecto):

	Valor de los factores					
FACTORES	Muy bajo	Bajo	Medio	Alto	Muy alto	Extra alto
Fiabilidad requerida	0,75	0,88	1,00	1,15	1,4	
Tamaño de la base de datos		0,94	1,00	1,08	1,16	
Complejidad del software	0,70	0,85	1,00	1,15	1,30	1,65
Restricciones de tiempo de ejecución			1,00	1,11	1,30	1,66
Restricciones de memoria			1,00	1,06	1,21	1,56
Volatilidad del hardware		0,87	1,00	1,15	1,30	
Restricciones de tiempo de respuesta		0,87	1,00	1,07		
Calidad de los analistas	1,46	1,19	1,00	0,86	0,71	
Experiencia con el tipo de aplicación	1,29	1,13	1,00	0,91	0,82	
Experiencia con el hardware	1,21	1,10	1,00	0,90		
Exp. con el lenguaje de programación.	1,14	1,07	1,00	0,95		
Calidad de los programadores	1,42	1,17	1,00	0,86	0,70	
Técnicas modernas de programación	1,24	1,10	1,00	0,91	0,82	
Empleo de herramientas	1,24	1,10	1,00	0,91	0,83	
Restricciones a la duración del proyec.	1,23	1,08	1,00	1,04	1,10	

Así, para calcular el **factor de ajuste del esfuerzo** (FAE) se multiplican los valores sombreados (distintos de 1):

$$\boldsymbol{FAE} = \prod_{k=1}^{n} Factor_k = 0{,}94 * 0{,}86 * 1{,}13 * 0{,}9 * 0{,}95 = \mathbf{0{,}78103566}$$

Cálculo del **esfuerzo del desarrollo** (E):

$$Esfuerzo = a\ KLDC^{b} \prod_{k=1}^{n} Factor_k = a\ KLDC^{b} * FAE$$

$$\boldsymbol{E = a\ KLDC^{b} * FAE = 3{,}2 * 5^{1{,}05} * 0{,}78 = 13{,}54\ personas/mes}$$

* Siendo a y b constantes del COCOMO que para modo orgánico toman los valores 3,2 y 1,05 respectivamente.

Cálculo **tiempo de desarrollo:**

$$\boldsymbol{TD = c\ Esfuerzo^{d} = 2{,}5 * 13{,}54^{0{,}38} = 6{,}73\ meses}$$

* Nuevamente se tienen dos constantes propias del COCOMO, en este caso c = 2,5 y d =0,38.

Productividad:

$$\boldsymbol{PR = {LDC}/{Esfuerzo} = {5.000}/{13{,}54} = 369{,}17\ LDC/personas\ mes}$$

Personal promedio:

$$\boldsymbol{P = {E}/{T} = {13{,}54}/{6{,}73} = 2{,}013\ LDC/personas\ mes}$$

Atendiendo a estas cifras, será necesario un equipo de 2 personas trabajando a tiempo completo algo menos de 7 meses, pero como el proyecto se realizará por una sola persona, el tiempo se vería incrementado hasta los **13 meses aproximadamente**.

Recursos utilizados

Para la realización de este proyecto se han utilizado diferentes recursos, tanto humanos como hardware y software.

Los recursos empleados en la creación de este proyecto han sido:

- Recursos humanos:
  - Un Analista-Diseñador /Programador /Probador.

- Recursos hardware:
  - Ordenadores:
    - Portátil HP Pavilion HDX9350ES con SO Windows Vista.
  - Móviles:
    - Tablet Nexus 7.
    - Smarphone Samsung Galaxy Ace S5830.
  - Otros:
    - Router Wi-Fi.
    - Impresora HP Deskjet D4260.
    - Conexión a Internet.

- Recursos software:
  - Análisis y diseño: StartUML v5.0.2.1570 y SmartDraw 2012
  - Seguimiento y planificación: OpenProj v1.4.
  - Implementación Android: Eclipse Java con SDK y ADT Android.
  - Documentación y manuales: Paquete Microsost Office 2010.
  - Diseño gráfico: Adobe Photoshop CS6
  - Generación sonido: as3sfxr y bfxr.
  - Motor de juegos AndEngine + AndEngine Multiplayer Extension.

Presupuesto.

A partir de las estimaciones de esfuerzo y tiempo y de los recursos utilizados, se calcula el presupuesto total del proyecto

A continuación se desglosan los costes asociados al uso de los recursos indicado.

Suponiendo que la vida útil de un ordenador personal sea de 4 años (48 meses) y que el proyecto durará en torno a los 13 meses, se tendría un **porcentaje de uso** que estaría alrededor del **25%**. De igual manera calcularemos el coste que supone el uso de la tablet y del Smartphone, aunque estos dispositivos sólo serán usados en la fase de pruebas y en determinados momentos de la fase de implementación (unos 4 meses), lo que sería una porcentaje de uso en torno al 8% de sus vidas útiles (suponiendo también que esta sea de 4 años).

Para la impresora se usará **dos cartuchos de tinta** valorados en **30€ cada uno** a largo del proyecto. En cuanto a la **conexión a internet** parece lógico suponer que no será usada única y exclusivamente para la realización del proyecto, por lo que se considera un **portaje de uso del 50%;** además, su precio corresponde al precio medio de la banda ancha en España según los datos publicados por la Comisión del Mercado de las Telecomunicaciones (CMT). Asimismo, se toma la estimación de 13 meses de duración del proyecto obtenida de los cálculos anteriores del COCOMO para determinar que el coste total derivado de la conexión a internet será de aproximadamente **238,55€ (18,35€/mes x 13 meses)**. El valor del router Wi-Fi está incluido el coste de la conexión a internet (lo proporciona la compañía que facilita el servicio).

HARDWARE	USO (%)	COSTE TOTAL UNITARIO (€)	COSTE POR USO UNITARIO (€)	CANTIDAD	COSTE TOTAL PROYECTO (€)
Ordenador personal	25%	1200 €	300 €	1 unidad	300 €
Tablet	8%	250 €	20 €	1 unidad	20 €
Smarthpone	8%	200 €	16 €	1 unidad	16 €
Conexión a internet	50%	36,7 €/mes	18,35 €/mes	13 meses	238,55€
Impresora	15%	60 €	9 €	1 unidad	9 €
Tinta impresora	100%	30 €	30 €	2 unidades	60 €
				TOTAL:	**643,55 €**

Tabla 1. Presupuesto recursos hardware

StarUML, OpenProj, Eclipse, Android SDK + ADT, AndEngine, as3sfxr y **bfxr** son **software libre** y no implica coste alguno. En cuanto a **Microsoft Office Professional 2010**, éste tiene actualmente un coste de 499€ y se aplica el **25% porcentaje de uso** (el mismo que el del ordenador personal en el que se instala). **SmartDraw 2012** tiene un coste de 225€ y **Adobe Photoshop CS6** de 280€, a ambos se aplica también un porcentaje de uso del 25%. El coste de Windows Vista está incluido en el coste del ordenador personal.

SOFTWARE	USO (%)	COSTE TOTAL UNITARIO (€)	COSTE POR USO UNITARIO (€)	COSTE TOTAL PROYECTO (€)
Eclipse	-	0	-	0
Android SDK-ADT	-	0	-	0
StarUML	-	0	-	0
OpenProj	-	0	-	0
AndEngine	-	0	-	0
as3sfxr	-	0	-	0
bfxr	-	0	-	0
Microsoft Office Professional 2010	25%	499 €	124,75€	124,75€
Photoshop CS6	25%	280€	70€	70€
SmartDraw 2012	25%	225€	56,25€	56,25€
				251€

Tabla 2. Presupuestos recursos software

Se estima la duración de cada fase del proyecto (tarea) en horas tomando como referencia el cálculo anterior obtenido en la estimación mediante el modelo COCOMO. Dicha estimación previa daba como resultado aproximadamente **13 meses**. Con una media de **20 días** trabajado **por mes** y **8 horas cada día** se obtendría **2080 horas de trabajo** que se distribuiría como sigue:

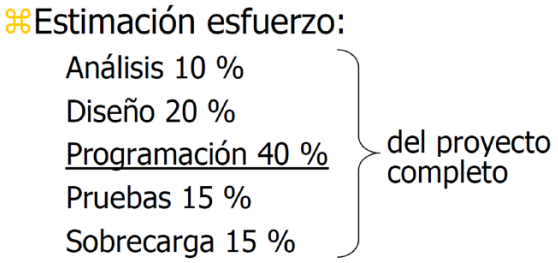


Ilustración 1. División general del esfuerzo

TAREA	%	DURACIÓN (HORAS)
Requisitos del sistema	5	104
Análisis del sistema	10	208
Diseño del sistema	15	312
Implementación	40	832
Pruebas de aplicación	15	312
Documentación adicional	5	104
Sobrecarga	10	208
TOTAL:		**2.080 HORAS**

Tabla 3. División del esfuerzo para el proyecto

Tomando las 2.080 horas de trabajo total y suponiendo un sueldo por hora de 10 € para un titulado de Grado en Informática, se tendría:

	TIEMPO	COSTE
Ingeniero	2080 HORAS	10 €/Hora
	TOTAL:	**20.800€**

Tabla 4. Presupuesto desarrollo

Hay que considerar además otros gastos referentes a material de oficina, entre otros, que se estiman en 150€.

	COSTE
Hardware	643,55 €
Software	251 €
Desarrollo	20.800 €
Otros gastos	150 €
TOTAL	**21.844,55 €**

Tabla 5. Presupuesto total

Seguimiento, calendarización y presupuesto del proyecto.

En esta sección se comenta el seguimiento, la calendarización y el presupuesto del proyecto realizados con la herramienta OpenProj.

Antes del comienzo del proyecto, se hizo una previsión de su duración dividiéndolo en las distintas tareas que lo componen. Además, se estableció como fecha de inicio del mismo el día 27 de mayo de 2013 y teniendo en cuenta la duración de las tareas, el final se establecía por la herramienta el día 12 de agosto de 2013. Por diversas circunstancias, el proyecto como tal se comenzó el día 10 de junio de 2013, por lo que hubo que reajustar la duración de las tareas y el calendario fijado. Dicho calendario fijado establecía trabajo de lunes a domingo de 10:00 a 14:00 y de 15:00 a 17:00, es decir, 6 horas de trabajo diario.

Además, a medida que el proyecto avanzaba se iba actualizando la duración real de las distintas tareas. Un ejemplo significativo de esto es el hecho de que la tarea llamada *Implementación de comunicaciones* se alargó más de lo inicialmente estimado, así que para compensar la pérdida de tiempo se invirtieron más horas diarias (sobreesfuerzo de programador) y se reajustaron las tareas de pruebas haciendo que los distintos tipos de pruebas fueran complementarios, es decir, las tareas se ejecutan en paralelo. Afortunadamente, gracias al trabajo realizado anterior al inicio del proyecto, las tareas de *implementación lógica de juegos* e *Implementación juego*s se finalizaron en menos tiempo del previsto y se pudo compensar en cierta medida la pérdida de tiempo que posteriormente se produciría con la implementación de comunicaciones.

Sobre esto hay que decir que entre las tareas del proyecto no se ha incluido el tiempo de preparación y formación previa en el desarrollo de juegos y en AndEngine, ya que fue un trabajo que se realizó antes de perfilarse oficialmente el proyecto como tal. Hay que decir que dentro de esa preparación previa se incluye tener el funcionamiento de los juegos bastante encaminado. Es decir, si lo queremos considerar parte del proyecto habría que hacer que las tareas relacionadas con lo comentado tuvieran una duración mucho mayor. En cualquier caso, aquí nos centraremos única y exclusivamente en el trabajo realizado entre el inicio del proyecto como tal (10 de junio) y el final del mismo.

No está de más recordar, que el proyecto sigue el ciclo de vida en cascada (clásico), y hasta que no haya finalizado una etapa no se inicia la siguiente, aunque en alguna ocasión se han realizado a la par algunas tareas complementarias (como en el caso de las pruebas antes mencionado).

Las etapas o tareas de requisitos, análisis y diseño son llevadas a cabo por el único integrante del proyecto, que toma el rol analista/diseñador. El trabajo realizado en estas etapas es importante para la realización de las siguientes; por ejemplo, la etapa de análisis se hace a partir de lo realizado en la etapa de requisitos, la de diseño a partir de análisis y la implementación teniendo en cuenta el análisis y diseño realizado.

La etapa de implementación, la cual se subdivide en función de los diferentes subsistemas o partes del sistema, representa el desarrollo de la aplicación (creación del código fuente) a cargo de un programador. La implementación se trata de una etapa larga, ya que en muchas ocasiones hay detalles en los que hay que detenerse para consultar documentación.

Tras la implementación, entra el probador en escena para testear el sistema al completo, primero cada parte por separado y finalmente todo el sistema en conjunto (pruebas de integración).

Por último, existe una tarea de documentación que consiste en recopilar la documentación realizada anteriormente (requisitos, análisis, diseño, etc.) y la creación de manuales de usuario e instalación, etapas importantes no sólo para el mantenimiento posterior, sino para la comprensión del proyecto.

A continuación se muestran todas las tareas del proyecto en dos imágenes.

En la primera imagen se muestran las distintas tareas en las que se ha dividido el proyecto, todas ellas englobadas en la tarea general del proyecto llamada Simple Android Games. Se puede observar de cada tarea su duración, con su fecha de inicio y fin, y los recursos asociados a cada una de ellas.

Task Information

ID	Nombre	Duración	Inicio	Terminado	Predecesores	Nombres del Recurso
1	Simple Android Games	79 days	10/06/13 10:00	27/08/13 17:00		Portatil HP Pavilion HDX9350ES;
2	Planificación y Estimación	1,667 days	10/06/13 10:00	11/06/13 15:00		
3	Planificación temporal	1 day	10/06/13 10:00	10/06/13 17:00		Analista-Diseñador
4	Estimación presupuestaria	0,667 days	11/06/13 10:00	11/06/13 15:00	3	Analista-Diseñador
5	Planificación completada	0 days	11/06/13 15:00	11/06/13 15:00	4	
6	Requisitos	9 days	11/06/13 14:00	20/06/13 15:00	5	
7	Analizar requisitos	3 days	11/06/13 14:00	14/06/13 15:00		Analista-Diseñador
8	Documentar requisitos	5 days	14/06/13 14:00	19/06/13 15:00	7	Analista-Diseñador
9	Revisar requisitos	1 day	19/06/13 14:00	20/06/13 15:00	8	Analista-Diseñador
10	Proceso de requisitos	0 days	20/06/13 15:00	20/06/13 15:00	9	
11	Analisis	17 days	20/06/13 14:00	7/07/13 15:00	10	
12	Modelado analítico del sistema	15 days	20/06/13 14:00	5/07/13 15:00		Analista-Diseñador
13	Revisar modelo analítico	2 days	5/07/13 14:00	7/07/13 15:00	12	Analista-Diseñador
14	Análisis completado	0 days	7/07/13 15:00	7/07/13 15:00	13	
15	Diseño	11 days	7/07/13 14:00	18/07/13 15:00	14	
16	Diseño de clases	3 days	7/07/13 14:00	10/07/13 15:00		Analista-Diseñador
17	Diseño de interfaces	5 days	10/07/13 14:00	15/07/13 15:00	16	Analista-Diseñador
18	Diseño de comunicaciones	3 days	15/07/13 14:00	18/07/13 15:00	17	Analista-Diseñador
19	Diseño completado	0 days	18/07/13 15:00	18/07/13 15:00	18	
20	Implementación	37 days	18/07/13 14:00	24/08/13 15:00	19	
21	Implementación general	4 days	18/07/13 14:00	22/07/13 15:00		Programador
22	Implementación lógica de juego	6 days	22/07/13 14:00	28/07/13 15:00	21	Programador
23	Implementación juego	10 days	28/07/13 14:00	7/08/13 15:00	22	Programador
24	Juego operativo	0 days	7/08/13 15:00	7/08/13 15:00	23	
25	Implementación comunicaciones	15 days	7/08/13 14:00	22/08/13 15:00	24	Programador[150%]
26	Integración de componentes	2 days	22/08/13 14:00	24/08/13 15:00	25	Programador
27	Implementación completada	0 days	24/08/13 15:00	24/08/13 15:00	26	
28	Pruebas	1,667 days	24/08/13 14:00	26/08/13 12:00	27	
29	Pruebas aplicación	0,667 days	24/08/13 14:00	25/08/13 12:00		Probador
30	Pruebas juegos	1,667 days	24/08/13 14:00	26/08/13 12:00	29SS	Probador
31	Pruebas integración	1 day	24/08/13 14:00	25/08/13 15:00	30SS	Probador
32	Pruebas completadas	0 days	25/08/13 15:00	25/08/13 15:00	31	
33	Documentación	2,333 days	25/08/13 14:00	27/08/13 17:00	32	
34	Recopilación de documentos	1 day	25/08/13 14:00	26/08/13 15:00		Analista-Diseñador
35	Manual de usuario	0,667 days	26/08/13 14:00	27/08/13 12:00	34	Analista-Diseñador
36	Manual instalación	0,667 days	27/08/13 12:00	27/08/13 17:00	35	Analista-Diseñador
37	Proceso documentación	0 days	27/08/13 17:00	27/08/13 17:00	36	
38	Final	0 days	27/08/13 17:00	27/08/13 17:00	37	

Figura 5. Listado de tareas del proyecto

En la segunda imagen, se pueden apreciar las mismas tareas junto con un diagrama de Gantt, representando cada barra la tarea correspondiente a su izquierda. La longitud de la barra viene determinada por la duración de la tarea asociada. En el diagrama se puede ver claramente que todas las tareas comienzan cuando termina la inmediatamente anterior, salvo en las tareas de pruebas de implementación que se realizan en paralelo.

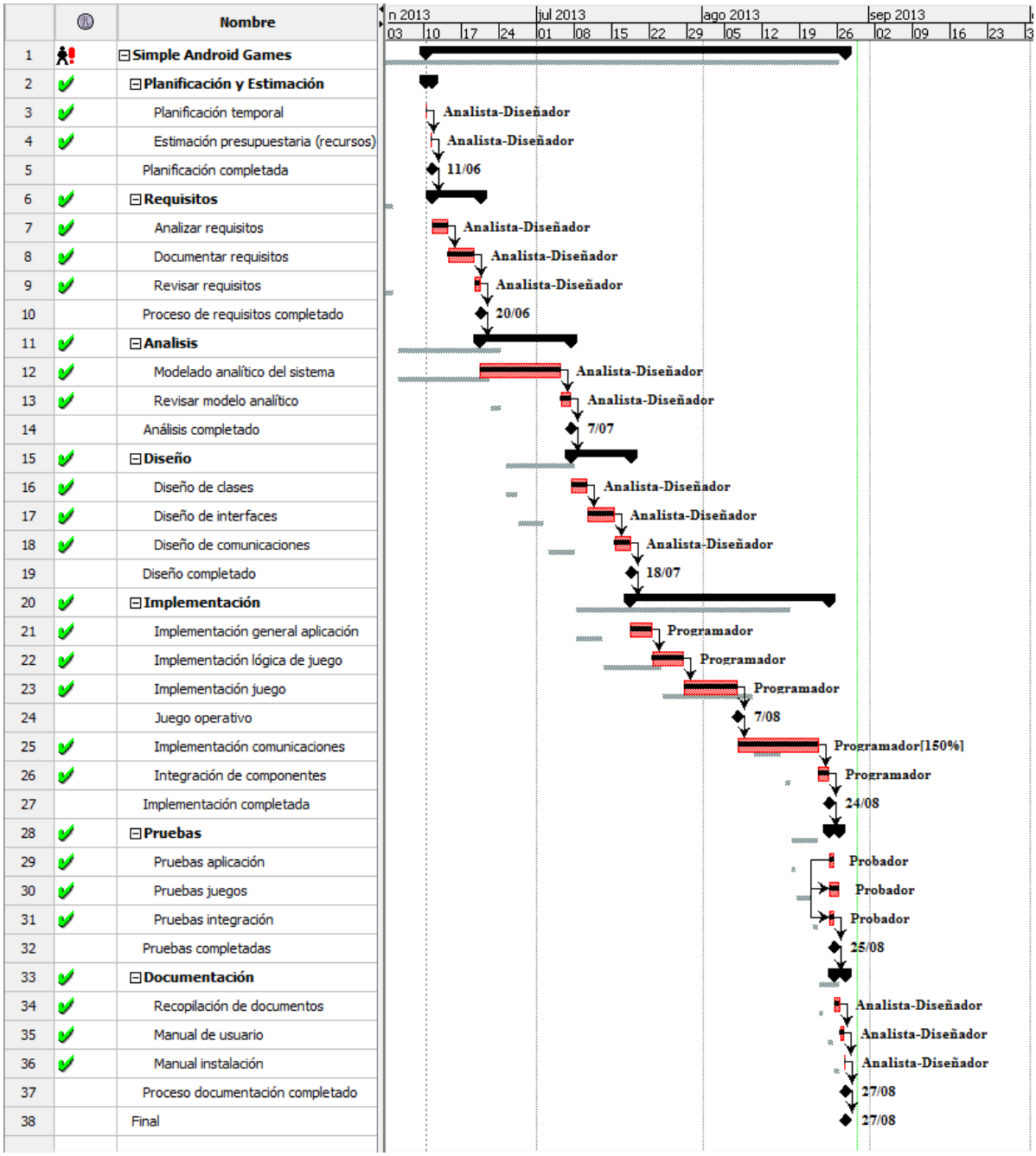


Figura 6. Tareas del proyecto con diagrama de Gantt.

Las variaciones producidas entre las estimaciones y el resultado real son visibles en el diagrama de Gantt mediante una línea base. La línea base representa la estimación.

A continuación se muestra la misma figura con más detalle.

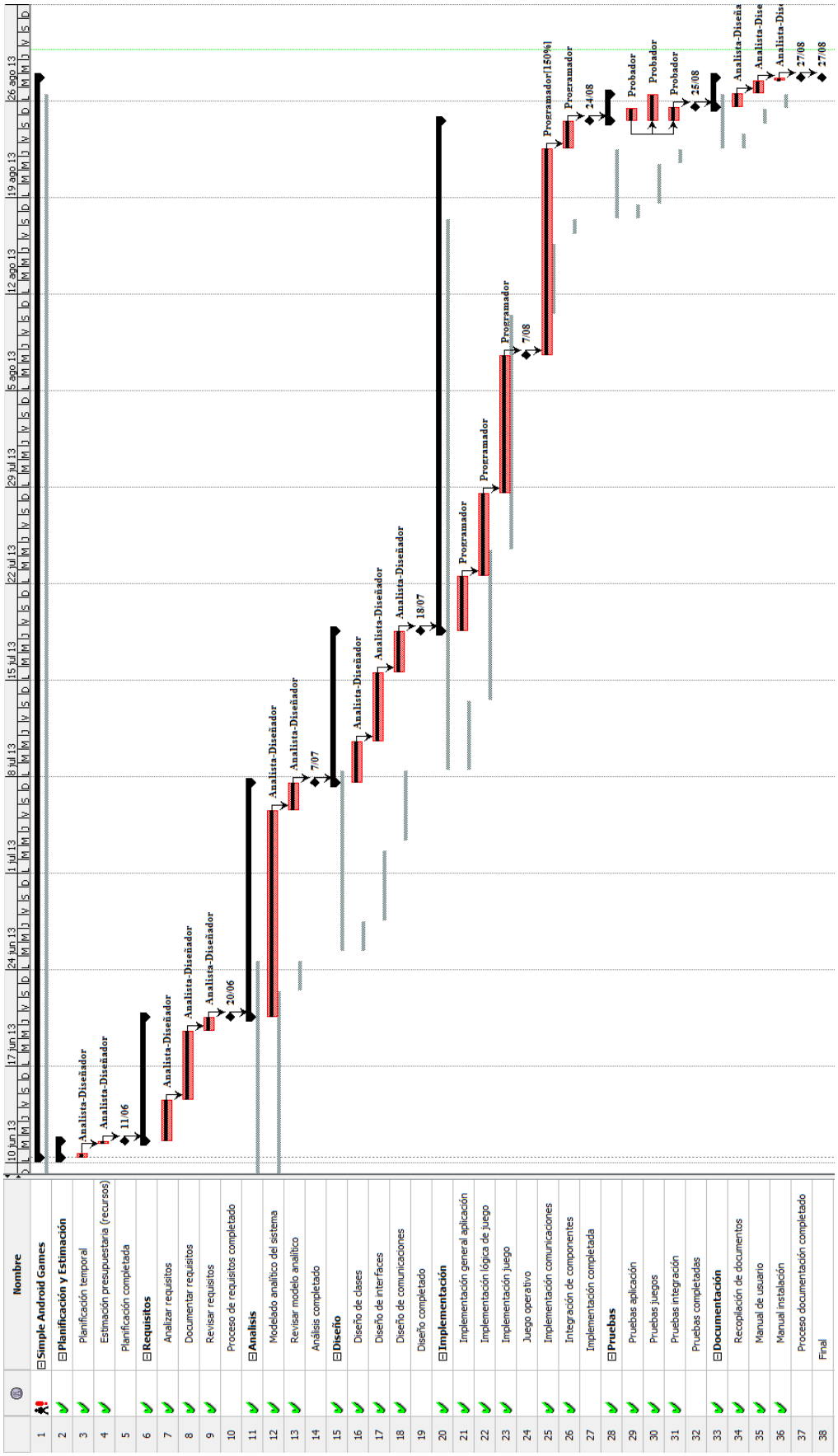


Figura 7. Tareas del proyecto con diagrama de Gantt (detallado).

En lo referente al presupuesto y costes finales del proyecto, se tienen los siguientes costes asociados a cada tarea:

Task Information

ID	Nombre	Costo	Costo Actual	Baseline Costo
1	Simple Android Games	$ 5.085,00	$ 4.749,00	$ 5.768,00
2	Planificación y Estimación	$ 100,00	$ 100,00	$ 160,00
3	Planificación temporal	$ 60,00	$ 60,00	$ 80,00
4	Estimación presupuestaria	$ 40,00	$ 40,00	$ 80,00
5	Planificación completada	$ 0,00	$ 0,00	$ 0,00
6	Requisitos	$ 540,00	$ 540,00	$ 800,00
7	Analizar requisitos	$ 180,00	$ 180,00	$ 160,00
8	Documentar requisitos	$ 300,00	$ 300,00	$ 480,00
9	Revisar requisitos	$ 60,00	$ 60,00	$ 160,00
10	Proceso de requisitos	$ 0,00	$ 0,00	$ 0,00
11	Analisis	$ 1.020,00	$ 1.020,00	$ 1.200,00
12	Modelado analítico del sistema	$ 900,00	$ 900,00	$ 1.040,00
13	Revisar modelo analítico	$ 120,00	$ 120,00	$ 160,00
14	Análisis completado	$ 0,00	$ 0,00	$ 0,00
15	Diseño	$ 660,00	$ 660,00	$ 800,00
16	Diseño de clases	$ 180,00	$ 180,00	$ 160,00
17	Diseño de interfaces	$ 300,00	$ 300,00	$ 320,00
18	Diseño de comunicaciones	$ 180,00	$ 180,00	$ 320,00
19	Diseño completado	$ 0,00	$ 0,00	$ 0,00
20	Implementación	$ 2.109,00	$ 2.109,00	$ 2.280,00
21	Implementación general	$ 228,00	$ 228,00	$ 304,00
22	Implementación lógica de juego	$ 342,00	$ 342,00	$ 608,00
23	Implementación juego	$ 570,00	$ 570,00	$ 988,00
24	Juego operativo	$ 0,00	$ 0,00	$ 0,00
25	Implementación comunicaciones	$ 855,00	$ 855,00	$ 304,00
26	Integración de componentes	$ 114,00	$ 114,00	$ 76,00
27	Implementación completada	$ 0,00	$ 0,00	$ 0,00
28	Pruebas	$ 180,00	$ 180,00	$ 288,00
29	Pruebas aplicación	$ 36,00	$ 36,00	$ 72,00
30	Pruebas juegos	$ 90,00	$ 90,00	$ 144,00
31	Pruebas integración	$ 54,00	$ 54,00	$ 72,00
32	Pruebas completadas	$ 0,00	$ 0,00	$ 0,00
33	Documentación	$ 140,00	$ 140,00	$ 240,00
34	Recopilación de documentos	$ 60,00	$ 60,00	$ 80,00
35	Manual de usuario	$ 40,00	$ 40,00	$ 80,00
36	Manual instalación	$ 40,00	$ 40,00	$ 80,00
37	Proceso documentación	$ 0,00	$ 0,00	$ 0,00
38	Final	$ 0,00	$ 0,00	$ 0,00

Figura 8. Coste por tareas.

Conviene aclarar que el coste del proyecto total no es la suma de todas las tareas anteriormente indicada como se podría pensar, ya que no sólo se incluyen las tareas simples, sino también las tareas formadas por otras subtareas. El coste total del proyecto lo indica la tarea con ID 1 SimpleAndroidGames.

Aunque en la figura aparezca el símbolo del dólar, todos los costes están calculados en euros.

En las siguientes imágenes se puede apreciar el coste de cada una de las tareas principales.

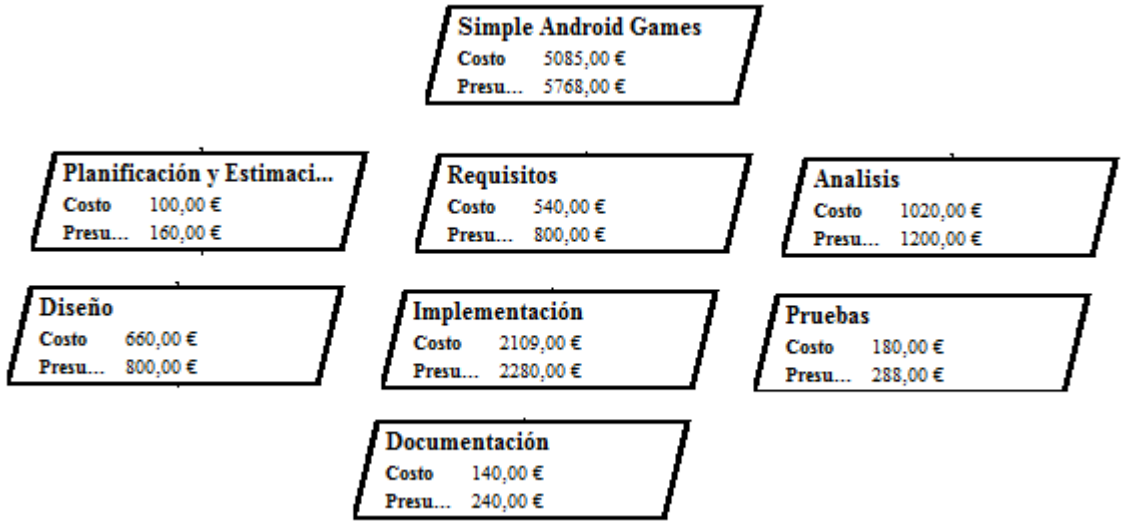


Figura 9. Coste de tareas principales.

A continuación se muestran los costes en relación a los recursos utilizados a lo largo del proyecto, primero recursos material y luego recursos humanos:

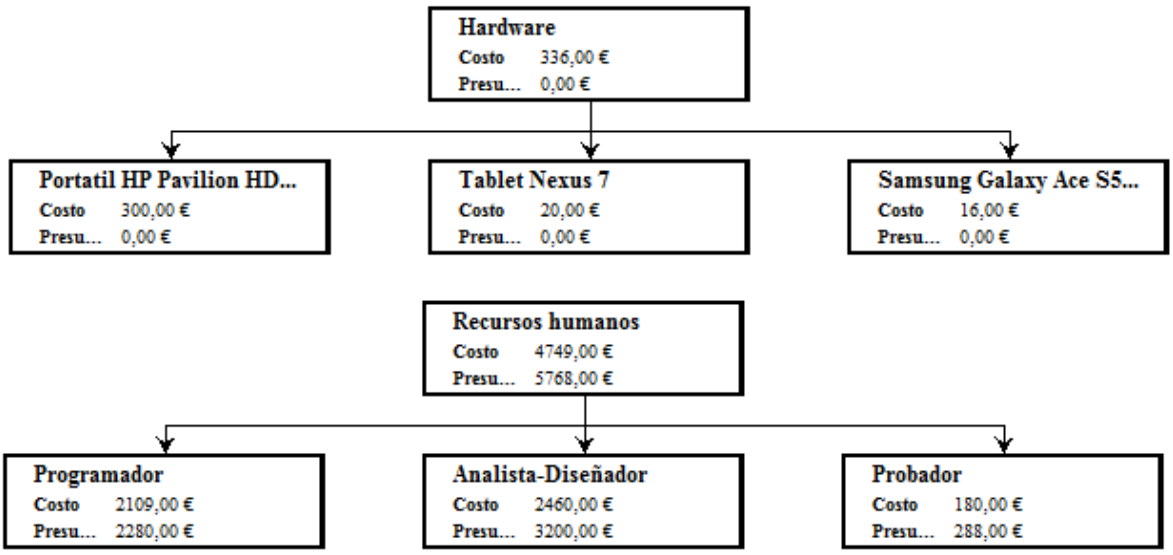


Figura 10. Coste Recursos.

El coste de los recursos hardware se ha calculado de igual manera que en la fase de estimación meciante COCOMO, esto es 25% del precio de compra del ordenador personal y el 8% para la tablet y el smartphone, ya que tras finalizar el proyecto se podrá seguir utilizando estos recursos materiales para otros proyectos u otros usos diversos.

Si se suma el coste de todos los recursos utilizados en el proyecto se puede comprobar que el coste total es exactamente igual a la suma de los costes en función de las tareas, que es el coste total del proyecto. Este coste total se muestra a continuación junto con otra información importante del proyecto.

Simple Android Games

Dates			
Start	10/06/13 10:00	Finish	27/08/13 17:00
Baseline Start	20/05/13 10:00	Baseline Finish	26/08/13 15:00
Actual Start	10/06/13 10:00	Actual Finish	

Duration			
Scheduled	79 days	Remaining	79 days
Baseline	98,667 days	Actual	0 days
		Percent Complete	78%

Work			
Scheduled	488 horas	Remaining	0 horas
Baseline	656 horas	Actual	488 horas

Costs			
Scheduled	5085,00 €	Remaining	336,00 €
Baseline	5768,00 €	Actual	4749,00 €
		Variance	1190,00 €

Figura 11. Información general del proyecto.

Como se puede observar en esta última imagen, aparece un resumen con los datos de la fecha de inicio y fin del proyecto, su duracion en días laborales y horas totales de trabajo y, como dato más importante, el coste total del proyecto (ya visto anteriormente mediante tareas y recursos) que suma un total de **5.085 euros.**

A tenor de los resultados obtenidos, se puede decir que las estimaciones presupuestarias realizadas inicialmente están muy alejadas de los datos obtenidos de la aplicación OpenProj, existiendo una **diferencia de más de 16.000 € entre el presupuesto estimado y el *real***.

Sin duda, una de las razones para de esta gran diferencia es el hecho de no haber incluido toda la parte de formación, preparativos y trabajo que ya había realizado previos al comienzo del proyecto como ya se ha comentado. Prueba de ello es la importante diferencia que también existe en tiempo de desarrollo necesario. Sobre esto, la realidad es que el tiempo que tenía para completar el proyecto era no era prorrogable.

Otra estimación inicial es la que indica la línea base creada al inicio, antes de realizar las modificaciones requeridas antes comentadas. Este valor, como se ve en la figura anterior es de **5.768 euros**.

5. Documentación técnica.

Después de haber explicado de forma general en qué consiste el proyecto y el producto de él obtenido, y haber comentado las cuestiones presupuestarias y de planificación asociadas al proyecto, se procede a explicar la documentación técnica del mismo. Dentro de la documentación técnica se incluyen las diferentes etapas que se han seguido, cumpliendo con el ciclo de vida clásico, para la realización del proyecto (requisitos, análisis, diseño, implementación y pruebas).

Para el cumplimiento de estas etapas se han ido realizando una serie de documentos que se expondrán seguidamente. Estos documentos son importantes no sólo para la realización del proyecto, sino también para que se tenga una base de cualquier parte del proyecto para futuras labores de mantenimiento del mismo.

Los documentos son extensos por lo que cada uno tiene su propio índice asociado.

Documento de Requisitos del Sistema

Índice DRS

Introducción

El proyecto se basa en la creación de una aplicación que sirva como base para ejecutar juegos Android simples. En función de cada juego específico, se debe proveer de la funcionalidad necesaria para que un usuario pueda *jugar contra la máquina* en partidas de un jugador o realizar partidas multijugador en un solo dispositivo. Además, dichos juegos contarán con funcionalidad multijugador provista para que dos usuarios con dispositivos físicamente separados puedan jugar entre ellos.

Para realizar partidas multijugador con dispositivos separados será necesaria una comunicación entre ellos. Dicha comunicación establece que uno de los dispositivos actúe como **servidor** y el otro como **cliente**. Así, el servidor será iniciado y se mantendrá a la espera de recibir peticiones por parte de algún cliente. Por su parte, el cliente al ser iniciado solicitará la dirección o ruta para conectar con el servidor y procederá a realizar dicha conexión. A partir de entonces la conexión quedará abierta para intercambiar mensajes indistintamente entre cliente y servidor en función de cómo se vaya desarrollando el juego.

La conexión podrá ser cancelada desde cualquier extremo, lo que implicará la parada de cliente y servidor.

En cuanto al funcionamiento de los juegos, en principio no hay ningún requisito específico que se deba cumplir para poder establecer comunicaciones. Por tanto, existe libertad en este aspecto para implementar la lógica de cada juego como se quiera o como se necesite.

Los juegos de este proyecto serán de tipo *tablero por turnos*, es decir cada jugador podrá realizar un movimiento por turno (y sólo durante su turno) en un tablero virtual y en función del resultado de ese movimiento mantendrá o no su turno de juego. Una partida de este tipo de juegos finaliza cuando no existen más movimientos posibles (empate), cuando uno de los dos jugadores gana la partida (victoria/derrota) o cuando algún jugador abandona el juego (fin de partida).

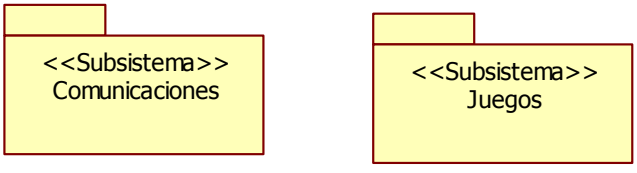


Figura 12. Subsistemas de la aplicación.

De lo dicho anteriormente, se deduce que el proyecto se puede dividir en dos subsistemas (ver figura anterior) tales como **Comunicaciones** y **Juegos**. No obstante, estos subsistemas estarán en cierta medida relacionados puesto que los juegos cuentan con funciones de comunicación. Se hace esta división con el fin de realizar un análisis más claro del sistema general.

Objetivos del sistema

Esta sección contiene una lista con los objetivos que se esperan alcanzar cuando el sistema software a desarrollar esté en explotación, especificados mediante la plantilla para objetivos.

Tabla 6. OBJ-1 Definir protocolo general de comunicación.

OBJ-1	Definición de protocolo general de comunicación
Versión	1.0
Autores	Vicente Bermejo Bermejo
Fuentes	Especificación de requisitos.
Descripción	Se define y gestiona la comunicación entre cliente y servidor.
Subobjetivos	OBJ-1.1 Gestión de Mensajes.
Importancia	Alta.
Urgencia	Alta.
Estado	Validado.
Estabilidad	Alta.
Comentarios	Esta comunicación puede realizase mediante Bluetooth o Wi-Fi.

Tabla 7. OBJ-1.1 Gestión de Mensajes

OBJ-1.1	Gestión de Mensajes
Versión	1.0
Autores	Vicente Bermejo Bermejo
Fuentes	Especificación de requisitos.
Descripción	El sistema deberá gestionar y procesar los diferentes mensajes enviados y recibidos en/por el cliente y el servidor.
Importancia	Media.
Urgencia	Media.
Estado	Validado.
Estabilidad	Alta.
Comentarios	Los mensajes pueden ser de diferentes tipos.

Tabla 8. OBJ-2 Gestión de Componentes

OBJ-2	Gestión de Componentes
Versión	1.0
Autores	Vicente Bermejo Bermejo
Fuentes	Especificación de requisitos.
Descripción	El sistema deberá gestionar los distintos elementos físicos que conforman la comunicación, tales como cliente y servidor.
Subobjetivos	OBJ-2.1 Gestión de Servidor. OBJ-2.2 Gestión de Cliente.
Importancia	Alta.
Urgencia	Media.
Estado	Validado.
Estabilidad	Alta.

Tabla 9. OBJ-2.1 Gestión de Servidor

OBJ-2.1	Gestión de Servidor
Versión	1.0
Autores	Vicente Bermejo Bermejo
Fuentes	Especificación de requisitos.
Descripción	El sistema deberá gestionar el funcionamiento del servidor.
Importancia	Media.
Urgencia	Media.
Estado	Validado.
Estabilidad	Alta.

Tabla 10. OBJ-2.2 Gestión de Cliente

OBJ-2.2	Gestión de Cliente
Versión	1.0
Autores	Vicente Bermejo Bermejo
Fuentes	Especificación de requisitos.
Descripción	El sistema deberá gestionar el funcionamiento del cliente.
Importancia	Media
Urgencia	Media
Estado	Validado.
Estabilidad	Alta.

Tabla 11. OBJ-3 Desarrollo y Gestión de Juegos

OBJ-3	Desarrollo y Gestión de Juegos
Versión	1.0
Autores	Vicente Bermejo Bermejo
Fuentes	Especificación de requisitos.
Descripción	El sistema deberá definir el comportamiento general de los juegos que se ejecutan y gestionar su funcionamiento.
Subobjetivos	OBJ-3.1 Definición de lógica de juego Tic Tac Toe. OBJ-3.2 Gestión juego Battleship.
Importancia	Alta
Urgencia	Media
Estado	Validado.
Estabilidad	Alta.
Comentarios	Ninguno.

Tabla 12. OBJ-3.1 Definición de Lógica de Juego Tic Tac Toe

OBJ-3.1	Definición de Lógica de Juego Tic Tac Toe
Versión	1.0
Autores	Vicente Bermejo Bermejo
Fuentes	Especificación de requisitos.
Descripción	El sistema deberá definir la lógica específica de del juego *Tic Tac Toe* y gestionar su funcionamiento.
Importancia	Alta.
Urgencia	Alta.
Estado	Validado.
Estabilidad	Alta.
Comentarios	Ninguno.

Tabla 13. OBJ-3.2 Definición de Lógica de Juego Battleship

OBJ-3.2	Definición de Lógica de Juego Battleship
Versión	1.0
Autores	Vicente Bermejo Bermejo
Fuentes	Especificación de requisitos.
Descripción	El sistema deberá definir la lógica específica de del juego *Battleship* y gestionar su funcionamiento.
Importancia	Alta.
Urgencia	Alta.
Estado	Validado.
Estabilidad	Alta.
Comentarios	Ninguno.

Catálogo de requisitos del sistema

Esta sección se divide en las siguientes subsecciones en las que se describen los requisitos del sistema.

Requisitos de información

Esta subsección contiene la lista de requisitos de almacenamiento y de restricciones de información que se han identificado, utilizando para especificarlos las plantillas para requisitos de información.

Tabla 14. IRQ-1 Información sobre mensajes

IRQ-1	**Información sobre mensajes**	
Versión	1.0.	
Autores	Vicente Bermejo Bermejo	
Fuentes	Especificación de requisitos.	
Objetivos asociados	OBJ-1.1 Gestión de Mensaje.	
Requisitos asociados	UC-1 Enviar mensaje. UC-2 Enviar mensaje Bluetooth. UC-3 Enviar mensaje Wi-Fi. UC-4 Procesar mensaje.	
Descripción	El sistema deberá almacenar la información correspondiente a los mensajes. En concreto:	
Datos específicos	- **Tipo**: tipo de mensaje. - **Datos**: información enviada en el mensaje.	
Tiempo de vida	**Medio**	**Máximo**
	Indeterminado	Indeterminado
Ocurrencias simultáneas	**Medio**	**Máximo**
	1	10
Importancia	Alta.	
Urgencia	Media.	
Estado	Validado.	
Estabilidad	Alta.	
Comentarios	Ninguno	

Tabla 15. IRQ-2 Información sobre servidor

IRQ-2	**Información sobre servidor**	
Versión	1.0.	
Autores	Vicente Bermejo Bermejo	
Fuentes	Especificación de requisitos.	
Objetivos asociados	OBJ-2.1 Gestión de Servidor.	
Requisitos asociados	UC-5 Iniciar servidor.	
Descripción	El sistema deberá almacenar la información correspondiente al servidor. En concreto:	
Datos específicos	- **Dirección**: localizador de recurso para conectar al servidor.	
Tiempo de vida	**Medio**	**Máximo**
	Indeterminado	Indeterminado
Ocurrencias	**Medio**	**Máximo**

simultáneas	1	1
Importancia	Alta.	
Urgencia	Alta	
Estado	Validado.	
Estabilidad	Alta.	
Comentarios	La dirección puede ser IP (para Wi-Fi) o MAC (para Bluetooth).	

Tabla 16. IRQ-3 Información sobre cliente

IRQ-3	**Información sobre cliente**	
Versión	1.0.	
Autores	Vicente Bermejo Bermejo	
Fuentes	Especificación de requisitos.	
Objetivos asociados	OBJ-2.2 Gestión de Cliente.	
Requisitos asociados	UC-6 Iniciar cliente. UC-7 Conectar con Servidor.	
Descripción	El sistema deberá almacenar la información correspondiente al cliente. En concreto:	
Datos específicos	- **Dirección**: localizador de recurso para el cliente. - **Servidor**: servidor al que está conectado el cliente (en caso de estarlo).	
Tiempo de vida	**Medio**	**Máximo**
	Indeterminado	Indeterminado
Ocurrencias simultáneas	**Medio**	**Máximo**
	1	1
Importancia	Alta.	
Urgencia	Alta	
Estado	Validado.	
Estabilidad	Alta.	
Comentarios	La dirección puede ser IP (para Wi-Fi) o MAC (para Bluetooth).	

Tabla 17. IRQ-4 Información sobre juegos por turnos

IRQ-4	**Información sobre juegos por turnos**	
Versión	1.0.	
Autores	Vicente Bermejo Bermejo	
Fuentes	Especificación de requisitos.	
Objetivos asociados	OBJ-3 Desarrollo y Gestión de Juegos. OBJ-3.1 Desarrollo y Gestión de Juego Tic Tac Toe. OBJ-3.2 Desarrollo y Gestión de Juego Battleship.	
Requisitos asociados	UC-8 Iniciar partida. UC-9 Realizar movimiento. UC-10 Comprobar finalización. UC-11 Simular movimiento. UC-12 Finalizar juego.	
Descripción	El sistema deberá almacenar la información correspondiente al juego por turnos. En concreto:	
Datos específicos	- **Tablero de juego**: representa un tablero virtual sobre el que colocar los elementos del juego. - **CPU**: representa un jugador rival controlado por la máquina.	
Tiempo de vida	**Medio**	**Máximo**
	Indeterminado	Indeterminado
Ocurrencias simultáneas	**Medio**	**Máximo**
	1	1
Importancia	Alta.	
Urgencia	Alta	
Estado	Validado.	
Estabilidad	Alta.	
Comentarios	Ninguno	

Requisitos no funcionales

Tabla 18. NFR–1 Internacionalización

NFR–1	Internacionalización
Versión	1.0
Autores	Vicente Bermejo Bermejo
Descripción	El sistema deberá proveer de un soporte para permitir varios idiomas.
Importancia	Baja.
Urgencia	Baja
Estado	Validado.
Estabilidad	Alta.
Comentarios	Se deben separar todas las cadenas de texto de la parte del código donde aparecen.

Tabla 19. NFR–2 Interfaz simple.

NFR–2	Interfaz simple.
Versión	1.0
Autores	Vicente Bermejo Bermejo
Descripción	La aplicación (y cada juego) contará con una interfaz sencilla y amigable.
Importancia	Media
Urgencia	Baja.
Estado	Validado.
Estabilidad	Alta.
Comentarios	Ninguno.

Tabla 20. NFR–3 Respuesta rápida en situaciones de juego (Rendimiento).

NFR–3	Respuesta rápida en situaciones de juego (Rendimiento).
Versión	1.0
Autores	Vicente Bermejo Bermejo
Descripción	En la medida de lo posible, el sistema deberá tener una respuesta rápida y evitar que los juegos presenten ralentizaciones o saltos en simulaciones o animaciones.
Importancia	Alta.
Urgencia	Media.
Estado	Validado.
Estabilidad	Alta.
Comentarios	La vista no debe bloquearse.

Tabla 21. NFR–4 Adaptación a cambios.

NFR–4	Adaptación a cambios.
Versión	1.0
Autores	Vicente Bermejo Bermejo.
Descripción	El sistema deberá fácilmente a posibles cambios. Especialmente deberá ser posible añadir nuevos juegos a la aplicación y formas de comunicación sin que esto suponga un trastorno para el funcionamiento del sistema.
Importancia	Alta.
Urgencia	Media.
Estado	Validado.
Estabilidad	Alta.

Tabla 22. NFR–5 Separación de vista y lógica de negocio.

NFR–5	Separación de vista y lógica de negocio.
Versión	1.0
Autores	Vicente Bermejo Bermejo.
Descripción	La lógica de negocio de la aplicación (y de cada juego) deberá ser independiente de la vista.
Importancia	Media
Urgencia	Media.
Estado	Validado.
Estabilidad	Alta.
Comentarios	AndEngine dificulta un poco la implementación de este requisito.

Requisitos funcionales

Esta sección contiene la lista de requisitos funcionales que se han identificado, dividiéndose en distintos apartados que se describen a continuación.

Tabla 23. REQ-1 Usuario juega partidas.

Identificador	REQ-1
Definición	Usuario juega partidas.
Autores	Vicente Bermejo Bermejo
Dependencias	UC-1 Alta incidencia.
Descripción	El usuario de la aplicación puede jugar partidas de varios tiempos.
Importancia	Alta.
Comentarios	Ninguno.

Tabla 24. REQ-2 Cliente conecta y se comunica con servidor.

Identificador	REQ-2
Definición	Cliente conecta y se comunica con servidor.
Autores	Vicente Bermejo Bermejo
Dependencias	UC-1 Alta incidencia.
Descripción	Un dispositivo (cliente) se puede conectar con otro (servidor) para jugar partidas multijugador mediante una comunicación bidireccional.
Importancia	Alta.
Comentarios	La comunicación supone un intercambio de mensajes.

Tabla 25. REQ-3 El sistema simula jugadores.

Identificador	REQ-3
Definición	El sistema simula jugadores.
Autores	Vicente Bermejo Bermejo
Dependencias	UC-1 Alta incidencia.
Descripción	El sistema cuenta con una IA (inteligencia artificial) para simular jugadores rivales.
Importancia	Alta.
Comentarios	La CPU genera movimientos para las partidas de un jugador. Los movimientos generados deben ser *lógicos* para el usuario.

Definición de Actores

Este apartado contiene una lista con los actores que se han identificado, especificados mediante la plantilla para actores de casos de uso.

Tabla 26. ACT-1 Usuario.

ACT-1	Usuario
Versión	1.0.
Autores	Vicente Bermejo Bermejo
Descripción	Este actor representa a la persona que juega e interacciona con la aplicación.
Comentarios	Ninguno.

Tabla 27. ACT-2 Servidor

ACT-2	Servidor
Versión	1.0.
Autores	Vicente Bermejo Bermejo
Descripción	Este actor representa al dispositivo que actúa como servidor en el proceso de comunicación.
Comentarios	Aunque forme parte del sistema, tomaremos al servidor como un actor externo cuando nos estamos centrando en el análisis únicamente del cliente, para facilitar su comprensión.

Tabla 28. ACT-3 Cliente

ACT-3	Cliente
Versión	1.0.
Autores	Vicente Bermejo Bermejo
Descripción	Este actor representa al dispositivo que actúa como cliente en el proceso de comunicación.
Comentarios	Aunque forme parte del sistema, tomaremos al cliente como un actor externo cuando nos estamos centrando en el análisis únicamente del servidor, para facilitar su comprensión.

Diagramas de casos de uso

Este apartado contiene los diagramas de casos de uso del sistema que se hayan realizado. Se dividen los casos de usos en distintos niveles de profundidad y en función de los subsistemas anteriormente definidos.

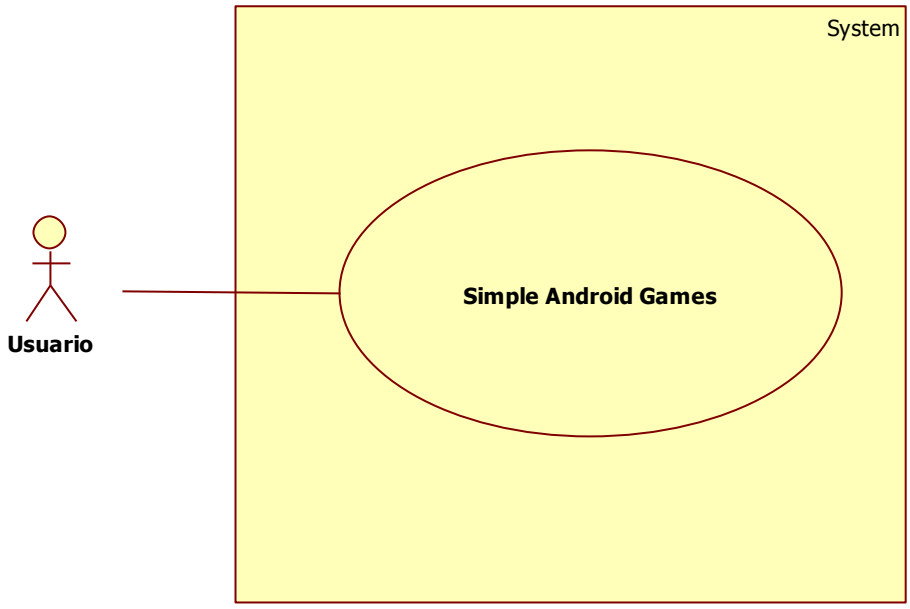


Figura 13. Diagrama de casos de uso nivel 1

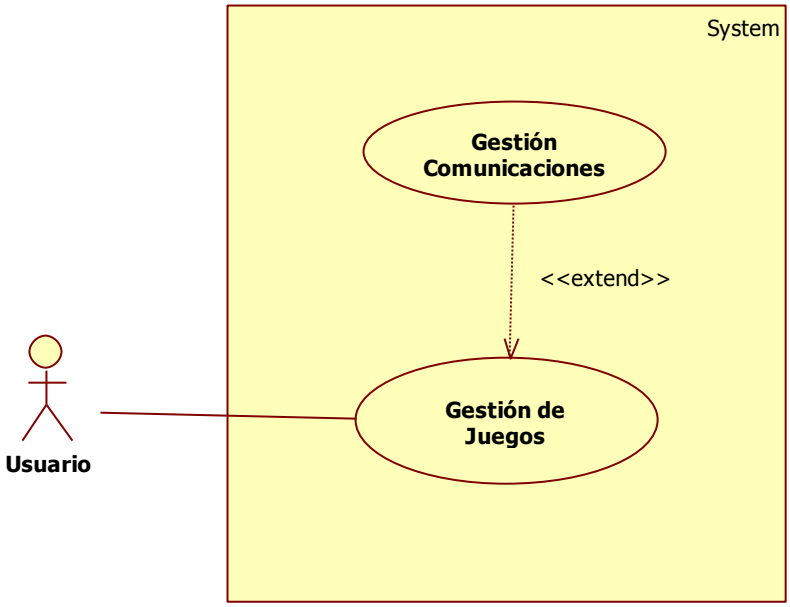


Figura 14. Diagrama de casos de uso nivel 2

Diagramas de Casos de Uso del Subsistema Comunicaciones.

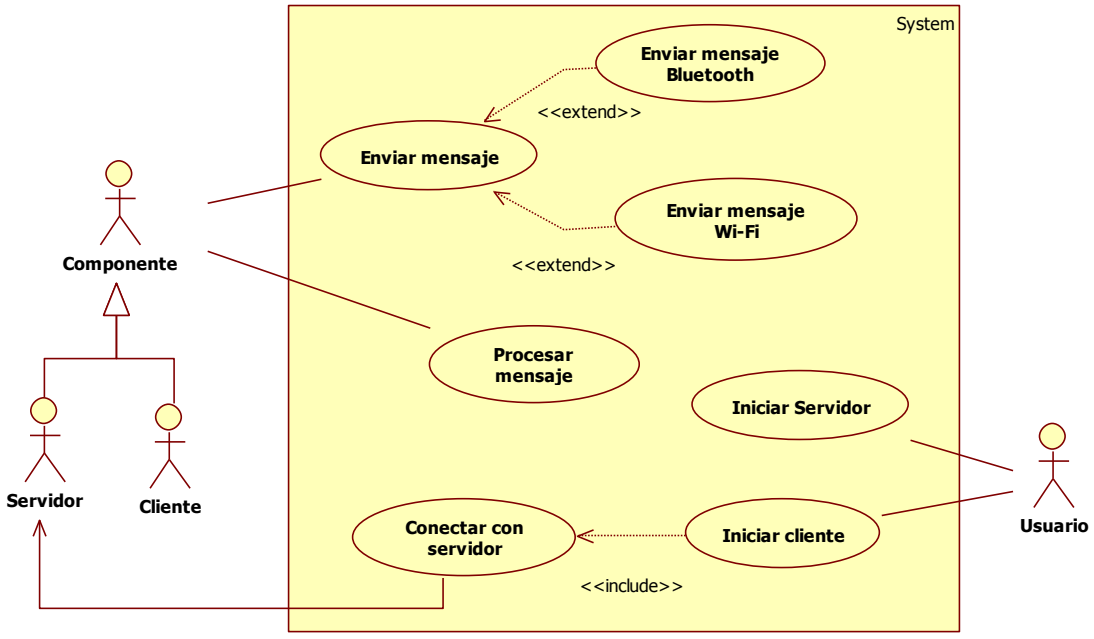


Figura 15. Diagrama de casos de uso nivel 3. Comunicaciones.

Diagramas de Casos de Uso del Subsistema Juegos

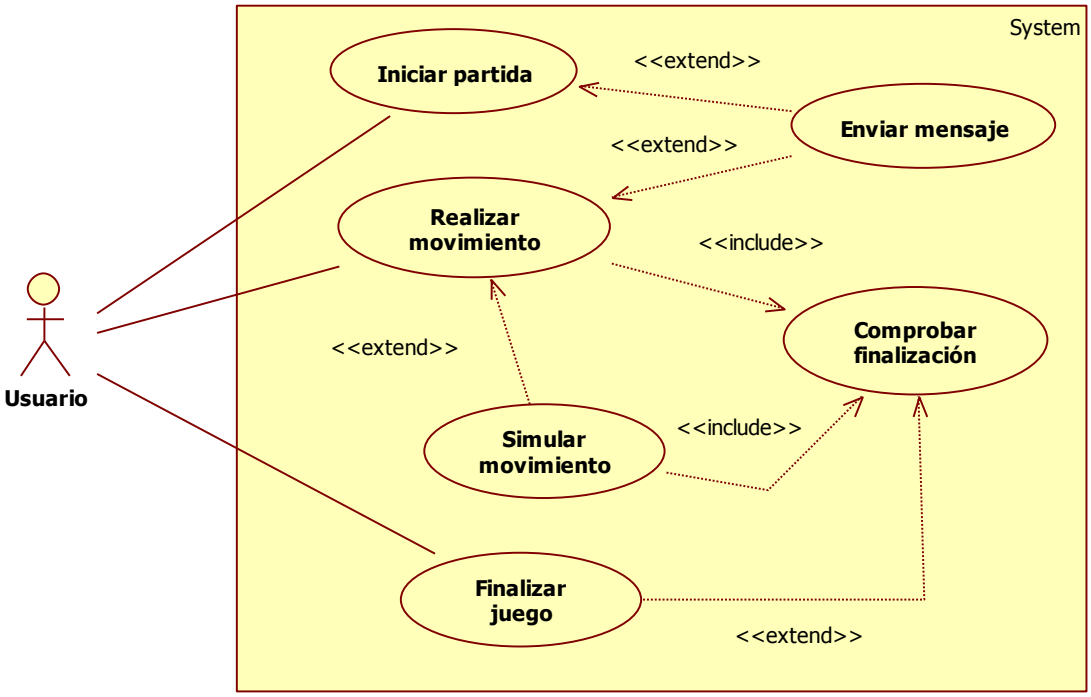


Figura 16. Diagrama de casos de uso nivel 3. Juegos

Especificación de casos de uso.

Casos de uso del Subsistema Comunicaciones.

Tabla 29. UC-1 Enviar mensaje

UC-1	Enviar mensaje	
Versión	1.0	
Autores	Vicente Bermejo Bermejo	
Fuentes	Especificación de requisitos	
Objetivos asociados	OBJ-1 Definición de protocolo de comunicación. OBJ-1.1 Gestión de mensajes. OBJ-2 Gestión de Componentes.	
Requisitos asociados	IRQ-1 Información sobre mensaje.	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando sea necesario enviar un mensaje.	
Precondición	Cliente conectado a servidor.	
Secuencia normal	**Paso**	**Acción**
	p1	El sistema inicia el proceso de envío de mensaje.
	p2	El sistema crear el mensaje en función de su estado actual.
	p3	El sistema ejecuta el caso de uso UC-2 Enviar mensaje Bluetooth si la conexión es de tipo Bluetooth.
Postcondición	Mensaje enviado correctamente al receptor.	
Excepciones	**Paso**	**Acción**
	p3	Si la conexión previamente establecida es de tipo Wi-Fi, se ejecuta el caso de uso UC-3 Enviar mensaje Wi-Fi.
Rendimiento	**Paso**	**Cota de tiempo**
	p3	20 segundos.
Frecuencia	Alta.	
Importancia	Alta.	
Urgencia	Alta	
Estado	Validado.	
Estabilidad	Alta.	
Comentarios	La cota de tiempo de los pasos indicados dependerá de la calidad de la conexión establecida entre cliente y servidor.	

Tabla 30. UC-2 Enviar mensaje Bluetooth

UC-2	Enviar mensaje Bluetooth	
Versión	1.0	
Autores	Vicente Bermejo Bermejo	
Fuentes	Especificación de requisitos	
Objetivos asociados	OBJ-1 Definición de protocolo de comunicación. OBJ-1.1 Gestión de mensajes. OBJ-2 Gestión de Componentes.	
Requisitos asociados	IRQ-1 Información sobre mensaje.	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando sea necesario enviar un mensaje a través de una conexión Bluetooth.	
Precondición	Mensaje creado. Datos de destinatario conocidos.	
Secuencia normal	**Paso**	**Acción**
	p1	El sistema prepara la conexión.
	p2	El sistema envía el mensaje mediante conexión Bluetooth al destinatario indicado.
Postcondición	Mensaje enviado correctamente al receptor.	
Rendimiento	**Paso**	**Cota de tiempo**
	p2	15 segundos.
Frecuencia	Alta.	
Importancia	Alta.	
Urgencia	Alta	
Estado	Validado.	
Estabilidad	Alta.	
Comentarios	La cota de tiempo de los pasos indicados dependerá de la calidad de la conexión establecida entre cliente y servidor.	

Tabla 31. UC-3 Enviar mensaje Wi-Fi

UC-3	Enviar mensaje Wi-Fi	
Versión	1.0	
Autores	Vicente Bermejo Bermejo	
Fuentes	Especificación de requisitos	
Objetivos asociados	OBJ-1 Definición de protocolo de comunicación. OBJ-1.1 Gestión de mensajes. OBJ-2 Gestión de Componentes.	
Requisitos asociados	IRQ-1 Información sobre mensaje.	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando sea necesario enviar un mensaje mediante una conexión Wi-Fi.	
Precondición	Mensaje creado. Datos de destinatario conocidos.	
Secuencia normal	**Paso**	**Acción**
	p1	El sistema prepara la conexión Wi-Fi.
	p2	El sistema envía el mensaje mediante conexión Wi-Fi al destinatario indicado.
Postcondición	Mensaje enviado correctamente al receptor.	
Rendimiento	**Paso**	**Cota de tiempo**
	p2	15 segundos.

Frecuencia	Alta.
Importancia	Alta.
Urgencia	Alta
Estado	Validado.
Estabilidad	Alta.
Comentarios	La cota de tiempo de los pasos indicados dependerá de la calidad de la conexión establecida entre cliente y servidor.

Tabla 32. UC-4 Procesar mensaje

UC-4	**Procesar mensaje**	
Versión	1.0	
Autores	Vicente Bermejo Bermejo	
Fuentes	Especificación de requisitos	
Objetivos asociados	OBJ-1 Definición de protocolo de comunicación. OBJ-1.1 Gestión de mensajes. OBJ-2 Gestión de Componentes.	
Requisitos asociados	IRQ-1 Información sobre mensaje.	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando se reciba un mensaje.	
Precondición	Nuevo mensaje recibido.	
Secuencia normal	**Paso**	**Acción**
	p1	El sistema inicia el procesado de mensajes.
	p2	El sistema analiza el mensaje (su tipo y su contenido).
	p3	El sistema ejecuta las acciones correspondientes al mensaje recibido.
Postcondición	Mensaje procesado correctamente.	
Excepciones	**Paso**	**Acción**
	p2	Si el mensaje recibido es de un tipo desconocido o su información es errónea, el sistema cancela la operación. A continuación este caso de eso queda sin efecto.
Rendimiento	**Paso**	**Cota de tiempo**
	p3	20 segundos.
Frecuencia	Alta.	
Importancia	Alta.	
Urgencia	Alta	
Estado	Validado.	
Estabilidad	Alta.	
Comentarios	Las acciones que realiza el sistema tras analizar el mensaje serán las propias del juego que corresponda, o acciones de desconexión de cliente o servidor.	

Tabla 33. UC-5 Iniciar servidor

UC-5	**Iniciar servidor**	
Versión	1.0	
Autores	Vicente Bermejo Bermejo	
Fuentes	Especificación de requisitos	
Objetivos asociados	OBJ-1 Definición de protocolo de comunicación. OBJ-2 Gestión de Componentes. OBJ-2.1 Gestión de Servidor.	
Requisitos asociados	IRQ-2 Información sobre servidor.	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el usuario desee iniciar el servidor.	
Precondición	Servidor no iniciado.	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario inicia el proceso de activación del servidor.
	p2	El sistema inicia el servidor e informa al usuario de su activación.
	p3	El servidor se mantiene a la espera de conexiones de clientes.
Postcondición	Servidor iniciado y a la espera de nuevas peticiones.	
Excepciones	**Paso**	**Acción**
	p2	Si la activación del servidor falla, se informa al usuario y se cancela la operación. A continuación este caso de uso queda sin efecto
Rendimiento	**Paso**	**Cota de tiempo**
	p3	Indeterminado
Frecuencia	Alta.	
Importancia	Alta.	
Urgencia	Alta	
Estado	Validado.	
Estabilidad	Alta.	
Comentarios	El servidor puede ser Bluetooth o Wi-Fi.	

Tabla 34. UC-6 Iniciar cliente

UC-6	Iniciar cliente	
Versión	1.0	
Autores	Vicente Bermejo Bermejo	
Fuentes	Especificación de requisitos	
Objetivos asociados	OBJ-1 Definición de protocolo de comunicación. OBJ-2 Gestión de Componentes. OBJ-2.2 Gestión de Cliente.	
Requisitos asociados	IRQ-3 Información sobre cliente.	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el usuario desee iniciar el cliente.	
Precondición	Cliente no iniciado.	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario inicia el proceso de activación del cliente.
	p2	El sistema inicia el cliente e informa al usuario de su activación.
	p3	Se ejecuta el caso de uso UC-7 Conectar con Servidor.
Postcondición	Cliente iniciado.	
Excepciones	**Paso**	**Acción**
	p2	Si la activación del cliente falla, se informa al usuario y se cancela la operación. A continuación este caso de uso queda sin efecto.
Rendimiento	**Paso**	**Cota de tiempo**
	P2	2 segundos
Frecuencia	Alta.	
Importancia	Alta.	
Urgencia	Alta	
Estado	Validado.	
Estabilidad	Alta.	
Comentarios	El cliente puede ser Bluetooth o Wi-Fi.	

Tabla 35. UC-7 Conectar con Servidor

UC-7	Conectar con Servidor	
Versión	1.0	
Autores	Vicente Bermejo Bermejo	
Fuentes	Especificación de requisitos	
Objetivos asociados	OBJ-1 Definición de protocolo de comunicación. OBJ-2 Gestión de Componentes. OBJ-2.1 Gestión de Servidor. OBJ-2.2 Gestión de Cliente.	
Requisitos asociados	IRQ-3 Información sobre cliente.	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el usuario conectar con un servidor actuando como cliente.	
Precondición	Cliente iniciado.	
Secuencia normal	**Paso**	**Acción**
	p1	El sistema solicita al usuario la dirección del servidor.
	p2	El usuario facilita al sistema la dirección en la que se puede conectar con el servidor.
	p3	El sistema se conecta con el servidor.
Postcondición	Conexión realizada entre cliente y servidor.	
Excepciones	**Paso**	**Acción**
	p3	Si la conexión entre cliente y servidor falla, se informa al usuario y se cancela la operación. A continuación este caso de uso queda sin efecto.
Rendimiento	**Paso**	**Cota de tiempo**
	p3	10 segundos.
Frecuencia	Alta.	
Importancia	Alta.	
Urgencia	Alta	
Estado	Validado.	
Estabilidad	Alta.	
Comentarios	La dirección puede ser IP o MAC.	

Casos de uso del Subsistema Juegos.

Tabla 36. UC-8 Iniciar partida

UC-8	**Iniciar partida**	
Versión	1.0	
Autores	Vicente Bermejo Bermejo	
Fuentes	Especificación de requisitos	
Objetivos asociados	OBJ-3 Desarrollo y Gestión de Juegos. OBJ-3.1 Desarrollo y Gestión de Juego Tic Tac Toe. OBJ-3.2 Desarrollo y Gestión de Juego Battleship.	
Requisitos asociados	IRQ-4 Información sobre juegos por turnos.	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el usuario quiera iniciar una partida.	
Precondición	Ninguna partida iniciada.	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario solicita iniciar una partida.
	p2	El sistema solicita el tipo de partida al usuario.
	p3	El usuario indica el tipo de partida que desea iniciar.
	p4	El sistema carga los recursos del juego y genera una nueva partida desde cero.
	p5	El sistema informa al usuario de que la partida se ha iniciado.
Postcondición	Partida iniciada.	
Excepciones	**Paso**	**Acción**
	p4	Si el tipo de partida seleccionada por el usuario es de un jugador, el sistema también inicia un jugador simulado (CPU).
	p4	Si el tipo de partida seleccionada por el usuario es multijugador, se ofrece al usuario la posibilidad de iniciar un cliente o un servidor.
Rendimiento	**Paso**	**Cota de tiempo**
	p4	2 segundos.
Frecuencia	Alta.	
Importancia	Alta.	
Urgencia	Alta	
Estado	Validado.	
Estabilidad	Alta.	
Comentarios	En general la partida puede ser de un jugador o multijugador (en el mismo dispositivo, por conexión Wi-Fi o Bluetooth).	

Tabla 37. UC-9 Realizar movimiento

UC-9	**Realizar movimiento**	
Versión	1.0	
Autores	Vicente Bermejo Bermejo	
Fuentes	Especificación de requisitos	
Objetivos asociados	OBJ-3 Desarrollo y Gestión de Juegos. OBJ-3.1 Desarrollo y Gestión de Juego Tic Tac Toe. OBJ-3.2 Desarrollo y Gestión de Juego Battleship.	
Requisitos asociados	IRQ-4 Información sobre juegos por turnos.	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el usuario quiera realizar un movimiento en un juego por turnos.	
Precondición	Partida iniciada. El usuario está en posesión del turno de juego.	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario indica el movimiento que desea realizar.
	p2	El sistema comprueba que el movimiento es válido.
	p3	El sistema ejecuta las acciones de juego correspondientes a ese movimiento.
	p4	El sistema ejecuta el caso de uso UC-10 Comprobar finalización.
	p5	El sistema informa al usuario del resultado del movimiento generado.
Postcondición	Movimiento realizado con éxito	
Excepciones	**Paso**	**Acción**
	p2	Si el movimiento indicado por el usuario no es válido, se informa al usuario, se cancela la operación y se le ofrece al usuario la posibilidad de realizar otro movimiento.
Rendimiento	**Paso**	**Cota de tiempo**
	p3	5 segundos.
Frecuencia	Alta.	
Importancia	Alta.	
Urgencia	Alta	
Estado	Validado.	
Estabilidad	Alta.	
Comentarios	Entre las acciones de juego se incluye la ejecución del caso de uso UC-1 Enviar mensaje en caso de que la partida sea de tipo multijugador Bluetooth o Wi-Fi.	

Tabla 38. UC-10 Comprobar finalización

UC-10	Comprobar finalización	
Versión	1.0	
Autores	Vicente Bermejo Bermejo	
Fuentes	Especificación de requisitos	
Objetivos asociados	OBJ-3 Desarrollo y Gestión de Juegos. OBJ-3.1 Desarrollo y Gestión de Juego Tic Tac Toe. OBJ-3.2 Desarrollo y Gestión de Juego Battleship.	
Requisitos asociados	IRQ-4 Información sobre juegos por turnos.	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso después de realizarse un movimiento.	
Precondición	El usuario ha seleccionado un movimiento válido.	
Secuencia normal	**Paso**	**Acción**
	p1	El sistema iniciar el proceso de comprobación de finalización de partida.
	p2	El sistema comprueba que tras el movimiento realizado aún no existe ganador de la partida.
	p3	El sistema comprueba que aún existen más movimientos posibles.
	p4	El sistema indica que el juego no ha finalizado. La partida continúa normalmente.
Postcondición	Informar de juego finalizado o no finalizado	
Excepciones	**Paso**	**Acción**
	p2	Si existe ganador se ejecuta el caso de uso UC-12 Finalizar juego y se cancela este caso de uso.
	p3	Si no hay más movimientos posibles se ejecuta el UC-12 Finalizar juego y se cancela este caso de uso.
Rendimiento	**Paso**	**Cota de tiempo**
	p2	1 segundo.
	p3	2 segundo.
Frecuencia	Alta.	
Importancia	Alta.	
Urgencia	Media	
Estado	Validado.	
Estabilidad	Alta.	
Comentarios	Ninguno.	

Tabla 39. UC-11 Simular movimiento.

UC-11	**Simular movimiento.**	
Versión	1.0	
Autores	Vicente Bermejo Bermejo	
Fuentes	Especificación de requisitos	
Objetivos asociados	OBJ-3 Desarrollo y Gestión de Juegos. OBJ-3.1 Desarrollo y Gestión de Juego Tic Tac Toe. OBJ-3.2 Desarrollo y Gestión de Juego Battleship.	
Requisitos asociados	IRQ-4 Información sobre juegos por turnos.	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el turno de juego corresponda al sistema en una partida de un jugador.	
Precondición	Partida iniciada. La *CPU* está en posesión del turno de juego.	
Secuencia normal	**Paso**	**Acción**
	p1	El sistema inicia el proceso de simulación de movimiento.
	p2	El sistema analiza el estado del juego y calcula un movimiento válido.
	p3	El sistema ejecuta las acciones de juego correspondientes a ese movimiento.
	p4	El sistema ejecuta el caso de uso UC-10 Comprobar finalización.
	p5	El sistema informa al usuario del resultado del movimiento generado.
Postcondición	Movimiento realizado con éxito.	
Rendimiento	**Paso**	**Cota de tiempo**
	p2	2 segundos.
Frecuencia	Alta.	
Importancia	Alta.	
Urgencia	Alta	
Estado	Validado.	
Estabilidad	Alta.	
Comentarios	Todo movimiento simulado debe ajustarse a los estándares de cada juego específico.	

Tabla 40. UC-12 Finalizar juego.

UC-12	Finalizar juego.	
Versión	1.0	
Autores	Vicente Bermejo Bermejo	
Fuentes	Especificación de requisitos	
Objetivos asociados	OBJ-3 Desarrollo y Gestión de Juegos. OBJ-3.1 Desarrollo y Gestión de Juego Tic Tac Toe. OBJ-3.2 Desarrollo y Gestión de Juego Battleship.	
Requisitos asociados	IRQ-4 Información sobre juegos por turnos.	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando una partida debe finalizar.	
Precondición	El usuario ha cancelado el juego o éste ha llegado a su fin.	
Secuencia normal	**Paso**	**Acción**
	p1	El sistema iniciar el proceso de finalización de partida.
	p2	El sistema determina el resultado de la partida e informa al usuario.
	p3	El sistema pone el juego en *estado de finalización*.
Postcondición	Informar de juego finalizado o no finalizado	
Excepciones	**Paso**	**Acción**
	p2	Si existe ganador se ejecuta el caso de uso UC-12 Finalizar juego y se cancela este caso de uso.
	p3	Si no hay más movimientos posibles se ejecuta el UC-12 Finalizar juego y se cancela este caso de uso.
Rendimiento	**Paso**	**Cota de tiempo**
	p2	1 segundo.
	p3	2 segundo.
Frecuencia	Alta.	
Importancia	Alta.	
Urgencia	Media	
Estado	Validado.	
Estabilidad	Alta.	
Comentarios	El resultado de la partida puede ser victoria, derrota o empate. Llamamos estado de finalización de una partida a aquel estado en el que la partida no puede ser proseguir, ha llegado a su fin.	

Matriz de Rastreabilidad objetivos/requisitos

Esta sección contiene una matriz *objetivos–requisitos*, de forma que para cada objetivo se pueda conocer con qué requisitos está asociado.

	OBJ–1	OBJ–1.1	OBJ–2	OBJ–2.1	OBJ–2.2	OBJ–03	OBJ–3.1	OBJ–3.2
IRQ–01		•						
IRQ–02				•				
IRQ–03					•			
IRQ–04						•	•	•
UC–01	•	•	•					
UC–02	•	•	•					
UC–03	•	•	•					
UC–04	•	•	•					
UC–05	•		•	•				
UC–06	•		•		•			
UC–07	•		•	•	•			
UC–08						•	•	•
UC–09						•	•	•
UC–10						•	•	•
UC–11						•	•	•
UC–12						•	•	•

Resumen

TIPO	ID	Descripción
OBJETIVOS	OBJ - 1	Definición de protocolo de comunicación
	OBJ – 1.1	Gestión de Mensajes
	OBJ – 2	Gestión de componentes.
	OBJ – 2.1	Gestión de Servidor.
	OBJ – 2.2	Gestión de Cliente.
	OBJ – 3	Desarrollo y Gestión de juegos.
	OBJ – 3.1	Desarrollo y Gestión de juego Tic Tac Toe.
	OBJ – 3.2	Desarrollo y Gestión de juego Battleship.
REQUISITOS GENERALES	REQ-1	Usuario juega partidas.
	REQ-2	Cliente conecta y se comunica con servidor.
	REQ-3	El sistema simula jugadores.
REQUISITOS INFORMACION	IRQ - 1	Información sobre mensajes.
	IRQ – 2	Información sobre servidor.
	IRQ – 3	Información sobre cliente.
	IRQ – 4	Información sobre juegos por turnos.
REQUISITOS FUNCIONALES	UC - 1	Enviar mensaje.
	UC – 2	Enviar mensaje Bluetooth,
	UC – 3	Enviar mensaje Wi-Fi.
	UC – 4	Procesar mensaje.
	UC – 5	Iniciar servidor.
	UC – 6	Iniciar cliente.
	UC – 7	Conectar con servidor.
	UC – 8	Iniciar partida.
	UC – 9	Realizar movimiento.
	UC – 10	Comprobar finalización.
	UC – 11	Simular movimiento.
	UC – 12	Finalizar juego.
REQUISITOS NO FUNCIONALES	NFR–1	Internacionalización.
	NFR–2	Interfaz simple.
	NFR–3	Respuesta rápida en situaciones de juego.
	NFR–4	Adaptación a cambios.
	NFR–5	Separación de vista y lógica de negocio.

Documento de Análisis del Sistema

Índice DAS

Introducción

En este documento se explicará el proyecto desde un punto de vista de análisis. Para empezar analizaremos el modelo estático del sistema en el que profundizaremos en el diagrama de tipos de objetos, examinando con detalle los tipos de objetos y las asociaciones entre objetos. Después pasaremos al modelo de comportamiento del sistema, en el cual, como su nombre indica, examinaremos el comportamiento o actuación de los casos de uso explicados en el anterior documento. Para ello, empezaremos con los diagramas de secuencia, pasando por los diagramas de colaboración y terminaremos estudiando los diagramas de estados del sistema y de actividad más importantes para la comprensión de la aplicación. Por último, repasaremos la matriz de rastreabilidad, para tener una idea de donde encajar los objetos y asociaciones analizados anteriormente, es decir, qué requisito de información y caso de uso se ajustan a un objeto determinado y qué requisito puede ajustarse a una asociación determinada.

Como ya se ha comentado previamente en la fase de requisitos, existen dos subsistemas, tales como, Comunicaciones y Juegos. Es por ello que cada sección será dividida a su vez en estos subsistemas para facilitar la comprensión del sistema en su conjunto.

Modelo estático del sistema

Esta sección describe el modelo estático del sistema formado por la descripción de los diferentes tipos de objetos y asociaciones obtenidos durante la fase de análisis.

Diagrama de tipos de objetos

En este apartado se presentan los diagramas UML de los paquetes, subsistemas y tipos de objetos resultantes de la fase de análisis.

Para facilitar el análisis del diagrama de clases y evitar que sea demasiado grande, se dividirá en varias partes. Los diagramas mostrarán las clases e interfaces del sistema indicando en ellas sólo los atributos principales.

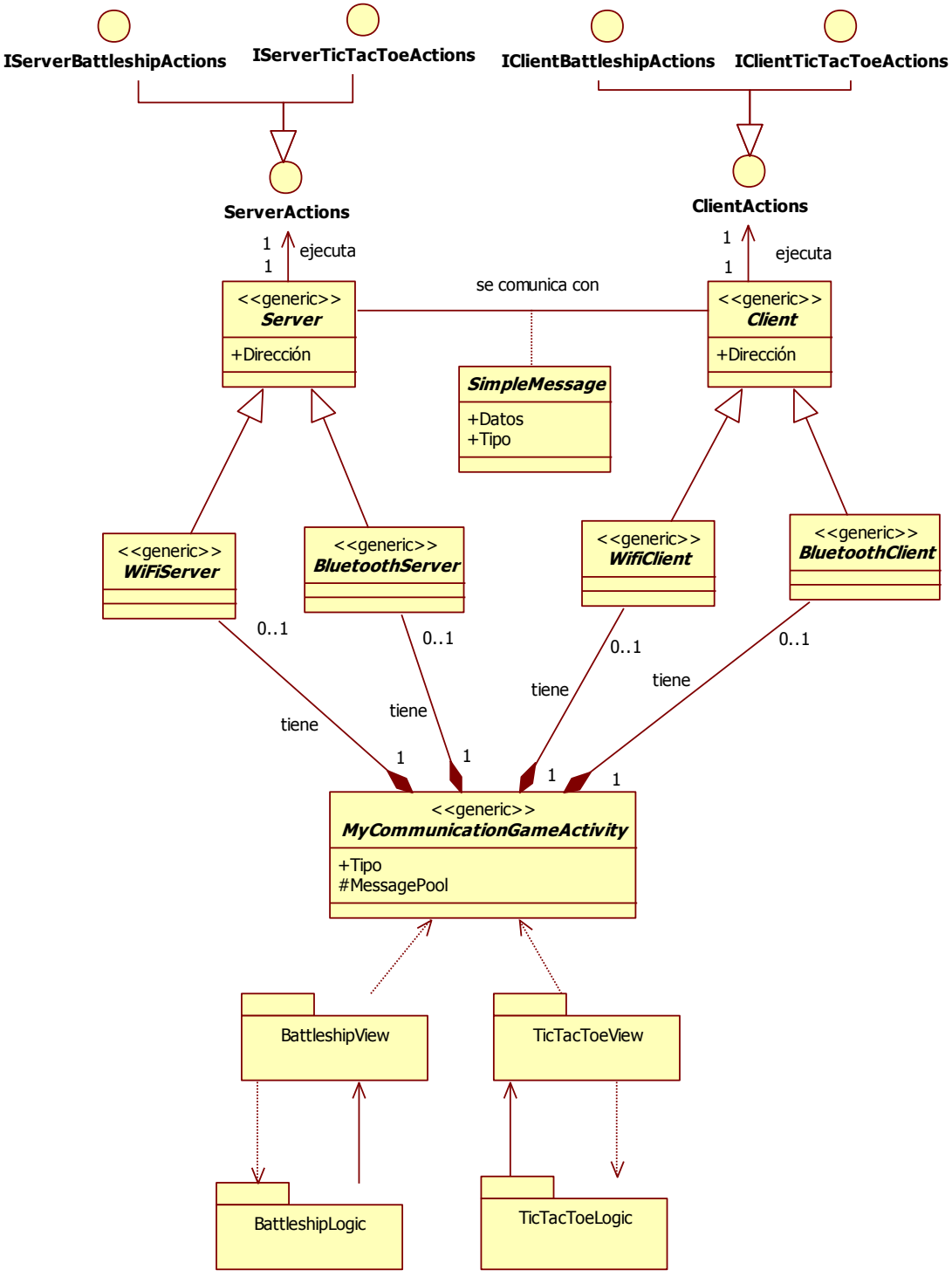


Figura 17. Diagrama de tipos de objetos. Parte comunicaciones.

La primera de la parte del diagrama (Figura 17) se centra en la parte de las comunicaciones y representa como paquetes las partes relativas a la lógica de los juegos y a la presentación por pantalla de los mismos. No se muestra en el diagrama pero cada tipo específico de mensaje es una subclase de *SimpleMessage*.

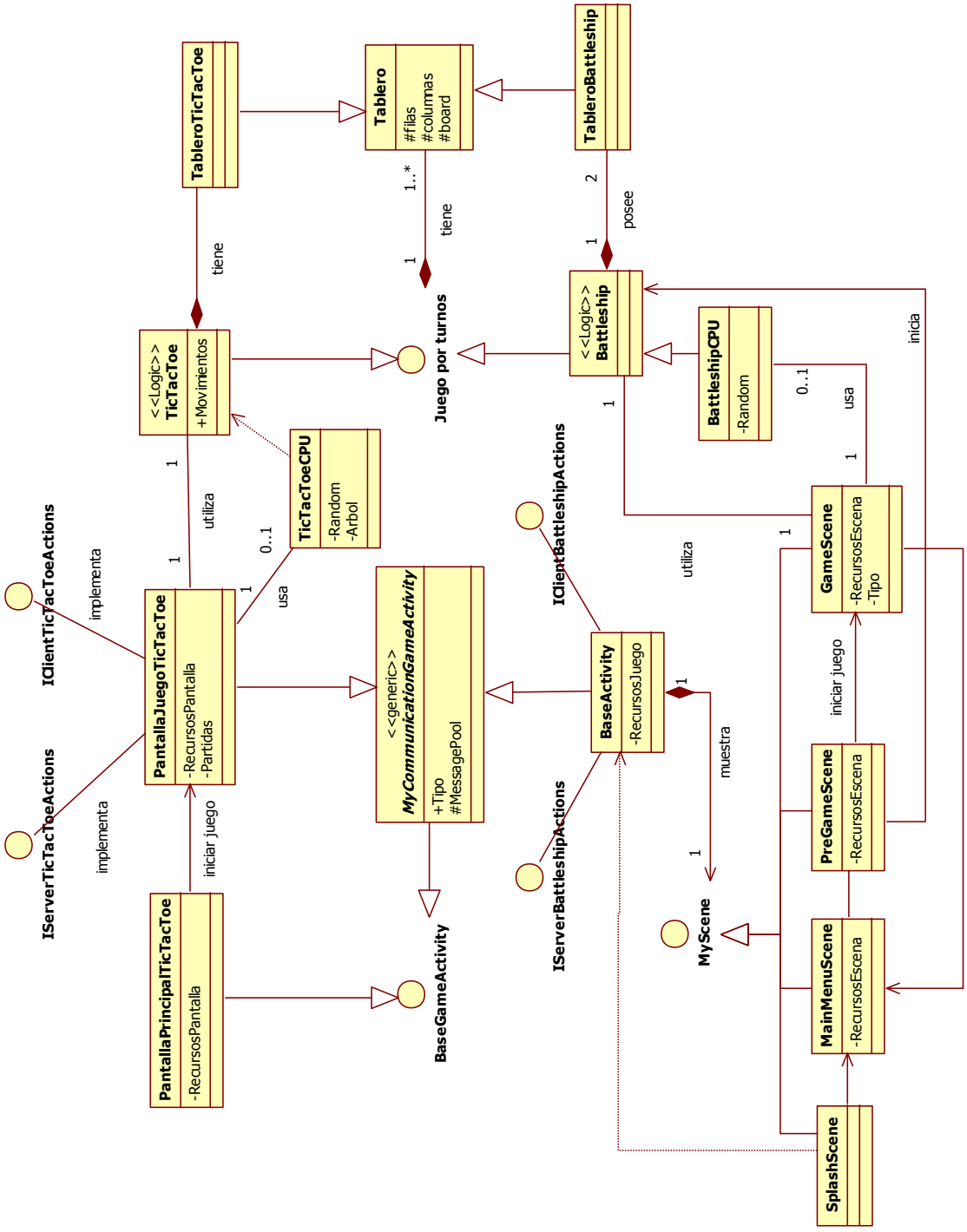


Figura 18. Diagrama de tipos de objetos. Parte juegos.

Para mayor claridad se divide este diagrama en dos subdiagramas, uno para la parte referida al juego *TicTacToe* y otro para el juego *Battleship*.

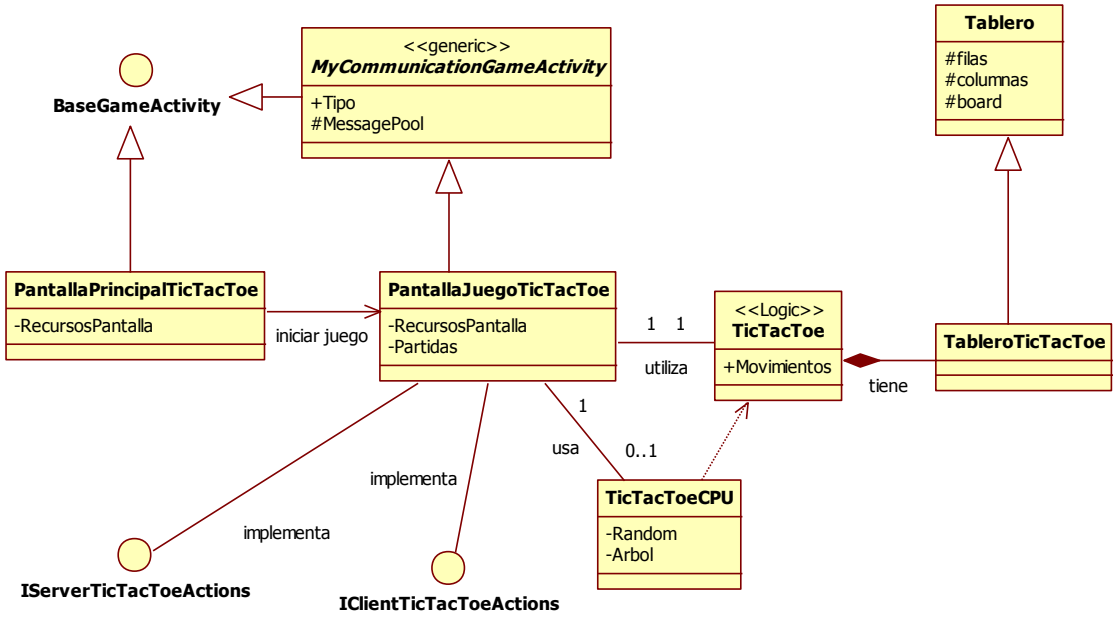


Figura 19. Diagrama de tipos de objetos. Parte TicTacToe.

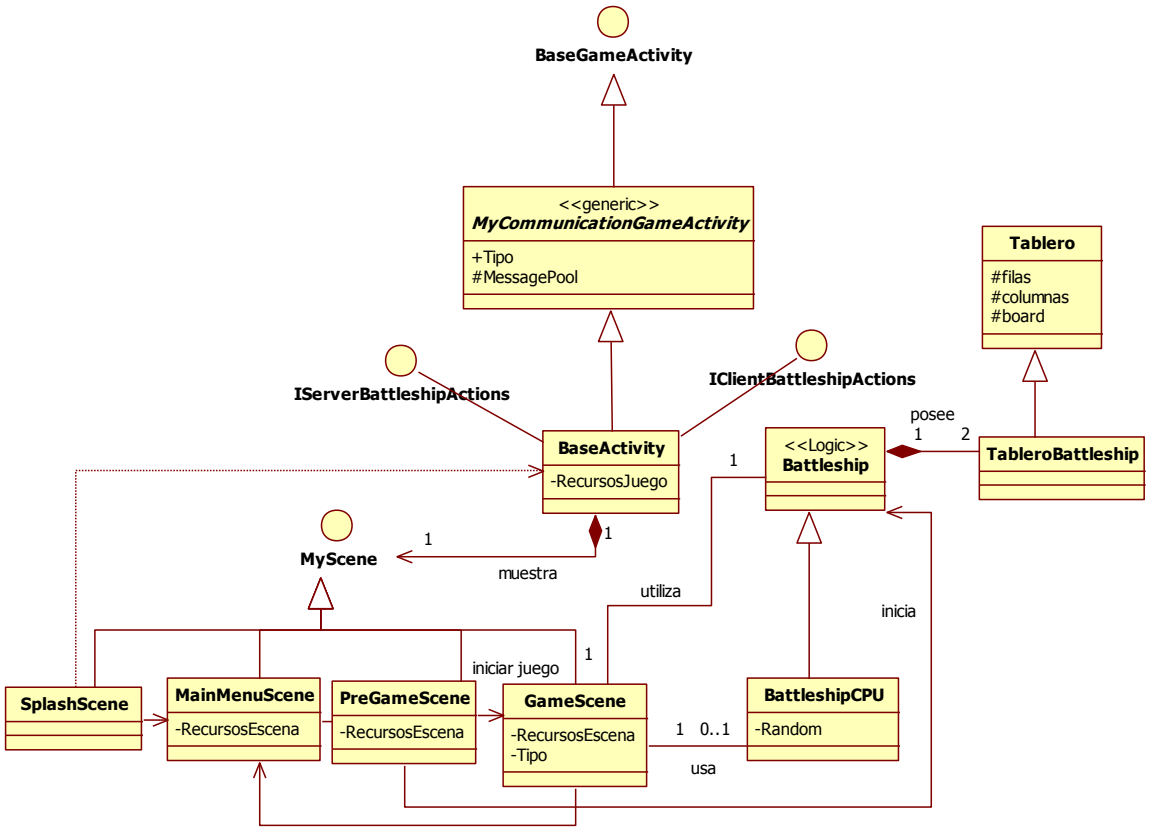


Figura 20. Diagrama de tipos de objetos. Parte Battleship.

Tipos de Objetos

Tipo de objeto Server

Descripción del tipo de Objeto Server

Tipo	Server
Versión	1.0
Autores	Vicente Bermejo Bermejo (desarrollador)
Requisitos Asociados	- IRQ-2 Información sobre Servidor. - REQ-2 Cliente conecta y se comunica con servidor.
Descripción	Este tipo abstracto representa el servidor del dispositivo al que se pueden conectar clientes.
Supertipos	*AndEngineMultiplayer.Server (*)*
Subtipos	WifiServer. BluetothServer
Comentarios	Se trata de una clase abstracta y genérica que puede ser especificada en función del tipo de conexión.

Atributos del tipo de Objeto Server

Atributo constante	Server::Dirección
Descripción	Secuencia de caracteres que se usa para localizar e identificar al servidor en una red de comunicaciones.
Tipo OCL	Cadena de caracteres.
Comentarios	Ninguno.

Tipo de Objeto WiFiServer

Descripción del tipo de Objeto WifiServer

Tipo	WiFiServer
Versión	1.0
Autores	Vicente Bermejo Bermejo (desarrollador)
Requisitos Asociados	- IRQ-2 Información sobre Servidor. - REQ-2 Cliente conecta y se comunica con servidor.
Descripción	Este tipo concreto representa el servidor de tipo Wi-Fi.
Supertipos	Server
Subtipos	TicTacToeWifiServer BattleshipWifiServer
Comentarios	Se trata de una clase abstracta y genérica que puede ser especificada en función del tipo de acciones servidor que deba ejecutar.

Tipo de Objeto BluetoothServer

Descripción del tipo de Objeto BluetoothServer

Tipo	BluetoothServer
Versión	1.0
Autores	Vicente Bermejo Bermejo (desarrollador)
Requisitos Asociados	- IRQ-2 Información sobre Servidor. - REQ-2 Cliente conecta y se comunica con servidor.
Descripción	Este tipo concreto representa el servidor de tipo Bluetooth.
Supertipos	Server
Subtipos	TicTacToeBluetoothServer BattleshipBluetoothServer
Comentarios	Se trata de una clase abstracta y genérica que puede ser especificada en función del tipo de acciones servidor que deba ejecutar.

Tipo de objeto Client

Descripción del tipo de Objeto Client

Tipo	Client
Versión	1.0
Autores	Vicente Bermejo Bermejo (desarrollador)
Requisitos Asociados	- IRQ-3 Información sobre Cliente. - REQ-2 Cliente conecta y se comunica con servidor.
Descripción	Este tipo abstracto representa el cliente del dispositivo que se conecta a un servidor.
Supertipos	*AndEngineMultiplayer.ServerConnector (*)*
Subtipos	WifiClient. BluetothClient.
Comentarios	Se trata de una clase abstracta y genérica que puede ser especificada en función del tipo de conexión.

Atributos del tipo de Objeto Client

Atributo constante	Client::Dirección
Descripción	Secuencia de caracteres que se usa para localizar e identificar al cliente en una red de comunicaciones.
Tipo OCL	Cadena de caracteres.
Comentarios	Ninguno.

Tipo de Objeto WiFiClient

Descripción del tipo de Objeto WifiClient

Tipo	WiFiClient
Versión	1.0
Autores	Vicente Bermejo Bermejo (desarrollador)
Requisitos Asociados	- IRQ-3 Información sobre Cliente. - REQ-2 Cliente conecta y se comunica con servidor.
Descripción	Este tipo concreto representa un conector cliente de tipo Wi-Fi.
Supertipos	Client
Subtipos	TicTacToeWifiClient BattleshipWifiClient
Comentarios	Se trata de una clase abstracta y genérica que puede ser especificada en función del tipo de acciones cliente que deba ejecutar.

Tipo de Objeto BluetoothClient

Descripción del tipo de Objeto BluetoothClient

Tipo	BluetoothClient
Versión	1.0
Autores	Vicente Bermejo Bermejo (desarrollador)
Requisitos Asociados	- IRQ-3 Información sobre Cliente. - REQ-2 Cliente conecta y se comunica con servidor.
Descripción	Este tipo concreto representa un conector cliente de tipo Bluetooth.
Supertipos	Client
Subtipos	TicTacToeBluetoothClient BattleshipBluetoothClient
Comentarios	Se trata de una clase abstracta y genérica que puede ser especificada en función del tipo de acciones cliente que deba ejecutar.

Tipo de Objeto SimpleMessage

Descripción del tipo de Objeto SimpleMessage

Tipo	SimpleMessage
Versión	1.0
Autores	Vicente Bermejo Bermejo (desarrollador)
Requisitos Asociados	- IRQ-1 Información sobre mensajes. - IRQ-2 Información sobre servidor. - IRQ-3 Información sobre cliente. - REQ-2 Cliente conecta y se comunica con servidor.
Descripción	Este tipo abstracto representa el mensaje básico que se transmite en la comunicación entre cliente y servidor.
Supertipos	*AndEngineMultiplayer.Message (*)*
Subtipos	ReadyMessage. RestartMessage ResultMessage SetCellMessage …
Comentarios	Es la clase base de la que hereda cada tipo de mensaje específico.

Atributos del tipo de Objeto SimpleMessage

Atributo constante	SimpleMessage::Tipo
Descripción	Indica el tipo específico del mensaje.
Tipo OCL	Entero
Comentarios	Ninguno.

Atributo variable	SimpleMessage::Datos
Descripción	Representa la información enviada en el mensaje.
Tipo OCL	Entero
Comentarios	Puede ser un conjunto de atributos.

Tipo de Objeto MyCommunicationGameActivity

Descripción del tipo de Objeto MyCommunicationGameActivity

Tipo	MyCommunicationGameActivity		
Versión	1.0		
Autores	Vicente Bermejo Bermejo (desarrollador)		
Requisitos Asociados	- IRQ-2 Información sobre servidor. - IRQ-3 Información sobre cliente. - REQ-2 Cliente conecta y se comunica con servidor.		
Descripción	Este tipo abstracto representa la actividad base para crear juegos que hagan uso de la funcionalidad de comunicaciones.		
Supertipos	*AndEngine.SimpleBaseGameActivity (*)*		
Subtipos	PantallaJuegoTicTacToe BattleshipBaseActivity …		
Componentes	**Nombre**	**Tipo OCL**	**Mult.**
	WiFiServer	WifiServer	0..1
	WiFiClient	WifiClient	0..1
	BluetoothServer	BluetoothServer	0..1
	BluetoothClient	BluetoothClient	0..1
Comentarios	Se trata de una clase abstracta y genérica que puede ser especificada en función del tipo de juego y los conectores que se quieran usar.		

Atributos del tipo de Objeto MyCommunicationGameActivity

Atributo variable	MyCommunicationGameActivity::Tipo
Descripción	Representa el tipo de conexión usada para las comunicaciones.
Tipo OCL	Entero
Comentarios	Niguno

Atributo constante	MyCommunicationGameActivity::MessagePool
Descripción	Agrupamiento de mensajes que permiten su reutilización, mejorando la eficiencia.
Tipo OCL	AndEngineMultiplayer:MessagePool
Comentarios	Es un requisito de uso de AndEngineMultiplayer.

Tipo de Objeto PantallaJuegoTicTacToe

Descripción del tipo de Objeto PantallaJuegoTicTacToe

Tipo	PantallaJuegoTicTacToe
Versión	1.0
Autores	Vicente Bermejo Bermejo (desarrollador)
Requisitos Asociados	- IRQ-4 Información sobre juegos por turnos. - REQ-1 Usuario juega partidas. - REQ-3 El sistema simula jugadores.
Descripción	Este tipo concreto representa la pantalla de juego de TicTacToe.
Supertipos	MyCommunicationGameActivity.
Subtipos	Ninguno.
Comentarios	Ninguno.

Atributos del tipo de Objeto PantallaJuegoTicTacToe

Atributo variable	PantallaJuegoTicTacToe::RecursosJuego
Descripción	Representa los recursos del juego.
Tipo OCL	Objeto
Comentarios	Los recursos pueden ser imágenes de diferentes tipos, sonidos, animaciones…

Atributo variable	PantallaJuegoTicTacToe::Partidas
Descripción	Número de partidas seguidas disputadas.
Tipo OCL	Entero.
Valor Inicial	0
Comentarios	Ninguno.

Tipo de Objeto PantallaPrincipalTicTacToe

Descripción del tipo de Objeto PantallaPrincipalTicTacToe

Tipo	PantallaPrincipalTicTacToe
Versión	1.0
Autores	Vicente Bermejo Bermejo (desarrollador)
Requisitos Asociados	- IRQ-4 Información sobre juegos por turnos. - REQ-1 Usuario juega partidas.
Descripción	Este tipo concreto representa la pantalla principal de TicTacToe, anterior a que se inicie la partida.
Supertipos	*AndEngine.BaseGameActivity*.
Subtipos	Ninguno.
Comentarios	Ninguno.

Atributos del tipo de PantallaPrincipalTicTacToe

Atributo variable	PantallaPrincipalTicTacToe::RecursosJuego
Descripción	Representa los recursos del juego.
Tipo OCL	Objeto
Comentarios	Los recursos pueden ser imágenes de diferentes tipos, sonidos, animaciones…

Tipo de Objeto TicTacToe

Descripción del tipo de Objeto TicTacToe

Tipo	TicTacToe		
Versión	1.0		
Autores	Vicente Bermejo Bermejo (desarrollador)		
Requisitos Asociados	- IRQ-4 Información sobre juegos por turnos. - REQ-1 Usuario juega partidas.		
Descripción	Este tipo concreto representa la lógica de juego de TicTacToe.		
Supertipos	Ninguno.		
Subtipos	Ninguno.		
Componentes	**Nombre**	**Tipo OCL**	**Mult.**
	Tablero	TableroTicTacToe	1
Comentarios	Ninguno		

Atributos del tipo de Objeto TicTacToe

Atributo variable	TicTacToe::Movimientos
Descripción	Indica el número de movimientos realizados en la partida.
Tipo OCL	Entero
Valor Inicial	0
Comentarios	Ninguno.

Tipo de Objeto TicTacToeCPU

Descripción del tipo de Objeto TicTacToeCPU

Tipo	TicTacToeCPU		
Versión	1.0		
Autores	Vicente Bermejo Bermejo (desarrollador)		
Requisitos Asociados	- IRQ-4 Información sobre juegos por turnos. - REQ-3 El sistema simula jugadores.		
Descripción	Este tipo concreto representa la simulación de movimientos por parte de la CPU.		
Supertipos	Ninguno		
Subtipos	Ninguno.		
Componentes	**Nombre**	**Tipo OCL**	**Mult.**
	Tablero	TableroTicTacToe	1
Comentarios	Ninguno		

Atributos del tipo de Objeto TicTacToeCPU

Atributo constante	TicTacToeCPU::Random
Descripción	Generador de números aleatorios para tareas de simulación.
Tipo OCL	Random
Comentarios	Ninguno.

Atributo constante	TicTacToeCPU::Arbol
Descripción	Estructura de datos para determinar qué movimiento de los posibles es mejor.
Tipo OCL	Random
Comentarios	Ninguno.

Tipo de Objeto Tablero

Descripción del tipo de Objeto Tablero

Tipo	Tablero
Versión	1.0
Autores	Vicente Bermejo Bermejo (desarrollador)
Requisitos Asociados	- IRQ-4 Información sobre juegos por turnos. - REQ-1 Usuario juega partidas.
Descripción	Este tipo abstracto representa un tablero bidimensional para juegos.
Supertipos	Ninguno.
Subtipos	TableroTicTacToe TableroBattleship
Comentarios	Ninguno.

Atributos del tipo de Objeto Tablero

Atributo constante	Tablero::filas
Descripción	Número de filas del tablero.
Tipo OCL	Entero.
Comentarios	Ninguno.

Atributo cosntante	Tablero::columnas
Descripción	Número de columnas del tablero.
Tipo OCL	Entero.
Comentarios	Ninguno

Atributo constante	Tablero::Board
Descripción	Representa las celdas del tablero.
Tipo OCL	Array[Entero][Entero]
Comentarios	Ninguno.

Tipo de Objeto TableroTicTacToe

Descripción del tipo de Objeto TableroTicTacToe

Tipo	TableroTicTacToe
Versión	1.0
Autores	Vicente Bermejo Bermejo (desarrollador)
Requisitos Asociados	- IRQ-4 Información sobre juegos por turnos. - REQ-1 Usuario juega partidas.
Descripción	Este tipo concreto representa el tablero del juego TicTacToe.
Supertipos	Tablero
Subtipos	Ninguno.
Comentarios	Ninguno.

Tipo de Objeto TableroBattleship

Descripción del tipo de Objeto Battleship

Tipo	TableroBattleship		
Versión	1.0		
Autores	Vicente Bermejo Bermejo (desarrollador)		
Requisitos Asociados	- IRQ-4 Información sobre juegos por turnos. - REQ-1 Usuario juega partidas.		
Descripción	Este tipo concreto representa el tablero del juego Battleship.		
Supertipos	Tablero		
Subtipos	Ninguno.		
Comentarios	Ninguno.		

Tipo de Objeto BaseActivity

Descripción del tipo de Objeto BaseActivity

Tipo	PantallaJuegoTicTacToe		
Versión	1.0		
Autores	Vicente Bermejo Bermejo (desarrollador)		
Requisitos Asociados	- IRQ-4 Información sobre juegos por turnos. - REQ-1 Usuario juega partidas. - REQ-3 El sistema simula jugadores.		
Descripción	Este tipo concreto representa la actividad base del juego Battleship.		
Supertipos	MyCommunicationGameActivity.		
Subtipos	Ninguno.		
Componentes	**Nombre**	**Tipo OCL**	**Mult.**
	CurrentScene	MyScene	1
Comentarios	Sólo existe una única actividad para este juego, sobre ella se van cargando diferentes escenas.		

Atributos del tipo de Objeto BaseActivity

Atributo variable	BaseActivity::RecursosJuego
Descripción	Representa los recursos del juego.
Tipo OCL	Objeto
Comentarios	Los recursos pueden ser imágenes de diferentes tipos, sonidos, animaciones…

Tipo de Objeto SplashScene

Descripción del tipo de Objeto SplashScene

Tipo	SplashScene
Versión	1.0
Autores	Vicente Bermejo Bermejo (desarrollador)
Requisitos Asociados	- IRQ-4 Información sobre juegos por turnos. - REQ-1 Usuario juega partidas.
Descripción	Este tipo concreto representa la escena inicial del juego Battleship.
Supertipos	MyScene
Subtipos	Ninguno.
Comentarios	Esta escena carga todos los recursos que serán usados por el juego.

Tipo de Objeto MainMenuScene

Descripción del tipo de Objeto MainMenuScene

Tipo	MainMenuScene
Versión	1.0
Autores	Vicente Bermejo Bermejo (desarrollador)
Requisitos Asociados	- IRQ-4 Información sobre juegos por turnos. - REQ-1 Usuario juega partidas.
Descripción	Este tipo concreto representa la escena de menú del juego Battleship.
Supertipos	MyScene
Subtipos	Ninguno.
Comentarios	Esta escena permite seleccionar el modo de juego.

Atributos del tipo de Objeto MainMenuScene

Atributo constante	MainMenuScene:RecursosEscena
Descripción	Representa los recursos específicos de la escena.
Tipo OCL	Objeto.
Comentarios	Los recursos pueden ser imágenes de diferentes tipos, sonidos, animaciones…

Tipo de Objeto PreGameScene

Descripción del tipo de Objeto PreGameScene

Tipo	PreGameScene
Versión	1.0
Autores	Vicente Bermejo Bermejo (desarrollador)
Requisitos Asociados	- IRQ-4 Información sobre juegos por turnos. - REQ-1 Usuario juega partidas.
Descripción	Este tipo concreto representa la escena previa al juego, que permite al usuario colocar los barcos en el tablero.
Supertipos	MyScene
Subtipos	Ninguno.
Comentarios	Ninguno.

Atributos del tipo de Objeto PreGameScene

Atributo constante	PreGameScene:RecursosEscena
Descripción	Representa los recursos específicos de la escena.
Tipo OCL	Objeto.
Comentarios	Los recursos pueden ser imágenes de diferentes tipos, sonidos, animaciones…

Tipo de Objeto GameScene

Descripción del tipo de Objeto GameScene

Tipo	GameScene
Versión	1.0
Autores	Vicente Bermejo Bermejo (desarrollador)
Requisitos Asociados	- IRQ-4 Información sobre juegos por turnos. - REQ-1 Usuario juega partidas.
Descripción	Este tipo concreto representa la escena donde se desarrolla el juego Battleship.
Supertipos	MyScene.
Subtipos	Ninguno.
Comentarios	Ninguno.

Atributos del tipo de Objeto GameScene

Atributo constante	GameScene::RecursosEscena
Descripción	Representa los recursos específicos de la escena.
Tipo OCL	Objeto.
Comentarios	Los recursos pueden ser imágenes de diferentes tipos, sonidos, animaciones…

Atributo constante	GameScene::Tipo
Descripción	Modo de juego de partida.
Tipo OCL	Entero
Comentarios	Ninguno.

Tipo de Objeto Battleship

Descripción del tipo de Objeto Battleship

Tipo	**Battleship**		
Versión	1.0		
Autores	Vicente Bermejo Bermejo (desarrollador)		
Requisitos Asociados	- IRQ-4 Información sobre juegos por turnos. - REQ-1 Usuario juega partidas.		
Descripción	Este tipo representa la lógica del juego Battleship		
Supertipos	Ninguno.		
Subtipos	BattlshipCPU		
Componentes	**Nombre**	**Tipo OCL**	**Mult.**
	Tablero / TableroRival	TableroBattlship	2
Comentarios	Ninguno.		

Tipo de Objeto BattleshipCPU

Descripción del tipo de Objeto BattleshipCPU

Tipo	**BattleshipCPU**
Versión	1.0
Autores	Vicente Bermejo Bermejo (desarrollador)
Requisitos Asociados	- IRQ-4 Información sobre juegos por turnos. - REQ-1 Usuario juega partidas. - REQ-3 El sistema simula jugadores.
Descripción	Este tipo concreto representa la simulación de movimientos por parte de la CPU para el juego Battleship.
Supertipos	Battlship
Subtipos	Ninguno
Comentarios	Ninguno.

Atributos del tipo de Objeto BattleshipCPU

Atributo constante	**BattleshipCPU::Random**
Descripción	Generador de elementos aleatorios para generación de movimientos.
Tipo OCL	Random
Comentarios	Ninguno.

Asociaciones

Asociación **Se comunica con** (**Server**, **Client**)

Descripción de la **Envía** (**Server**, **Client**)

Asociación	**Se comunicacion entre Server y Client**
Versión	1.0
Autores	Vicente Bermejo Bermejo
Requisitos Asociados	- IRQ-1 Información sobre mensajes. - IRQ-2 Información sobre servidor. - REQ-2 Cliente conecta y se comunica con servidor.
Descripción	Esta asociación representa la comunicación entre servidor y cliente.
Comentarios	La comunicación implica un intercambio de mensajes.

Roles de la Asociación **Se comunica con** (**Server, Client**).

Rol	Server **juega rol** extremo de la comunicación **en** Se comunica con (Server, Client)
Descripción	El servidor se comunica con el cliente.
Tipo OCL	Server
Multiplicidad	1
Comentarios	El servidor puede envía y recibir mensajes.

Rol	Client **juega rol** extremo de la comunicación **en** Se comunica con (Server, Client)
Descripción	El cliente se comunica con el servidor.
Tipo OCL	Client
Multiplicidad	1
Comentarios	El cliente puede envía y recibir mensajes.

Atributos de la Asociación **Se comunica con (Server, Client)**.

Atributo constante	**SimpleMessage::Tipo**
Descripción	Indica el tipo específico del mensaje.
Tipo OCL	Entero
Comentarios	Ninguno.

Atributo variable	**SimpleMessage::Datos**
Descripción	Representa la información enviada en el mensaje.
Tipo OCL	Entero
Comentarios	Puede ser un conjunto de atributos.

Asociación **Tiene** (**MyCommunicationGameActivity**, **WiFiServer**).

Descripción de la Asociación **Tiene** (**MyCommunicationGameActivity**, **WiFiServer**).

Asociación	**Tiene entre MyCommunicationGameActivity y WiFiServer**
Versión	1.0
Autores	Vicente Bermejo Bermejo
Requisitos Asociados	- IRQ-2 Información sobre servidor. - IRQ-4 Información sobre juegos por turnos. - REQ-2 Cliente conecta y se comunica con servidor.
Descripción	Esta asociación representa la tenencia de un servidor Wi-Fi en la clase MyCommunicationGameActivity
Comentarios	Ninguno.

Roles de la Asociación **Tiene** (**MyCommunicationGameActivity**, **WiFiServer**).

Rol	MyCommunicationGameActivity **juega rol** tenedor **en** Tiene (MyCommunicationGameActivity, WiFiServer)
Descripción	La activity base para juegos multijugador tiene Servidor Wi-Fi.
Tipo OCL	MyCommunicationGameActivity
Multiplicidad	1
Comentarios	Ninguno.

Rol	WiFiServer **juega rol** tenencia **en** Tiene (MyCommunicationGameActivity, WiFiServer)
Descripción	WiFiServer es tenido por MyCommunicationGameActivity.
Tipo OCL	WiFiServer
Multiplicidad	0…1
Comentarios	Ninguno.

Asociación **Tiene** (**MyCommunicationGameActivity**, **BluetoothServer**).

Descripción de la Asociación **Tiene** (**MyCommunicationGameActivity**, **BluetoothServer**).

Asociación	**Tiene entre MyCommunicationGameActivity y BluetoothServer**
Versión	1.0
Autores	Vicente Bermejo Bermejo
Requisitos Asociados	- IRQ-2 Información sobre servidor. - IRQ-4 Información sobre juegos por turnos. - REQ-2 Cliente conecta y se comunica con servidor.
Descripción	Esta asociación representa la tenencia de un servidor Bluetooth en la clase MyCommunicationGameActivity
Comentarios	Ninguno.

Roles de la Asociación **Tiene** (**MyCommunicationGameActivity**, **BluetootServer**).

Rol	MyCommunicationGameActivity **juega rol** tenedor **en** Tiene (MyCommunicationGameActivity, BluetoothServer)
Descripción	La activity base para juegos multijugador tiene Servidor Bluetooth.
Tipo OCL	MyCommunicationGameActivity
Multiplicidad	1
Comentarios	Ninguno.

Rol	BluetoothServer **juega rol** tenencia **en** Tiene (MyCommunicationGameActivity, BluetoothServer)
Descripción	BluetoothServer es tenido por MyCommunicationGameActivity.
Tipo OCL	BluetoothServer
Multiplicidad	0…1
Comentarios	Ninguno.

Asociación **Tiene** (**MyCommunicationGameActivity**, **WiFiClient**).

Descripción de la Asociación **Tiene** (**MyCommunicationGameActivity**, **WiFiClient**).

Asociación	**Tiene entre MyCommunicationGameActivity y WiFiClient**
Versión	1.0
Autores	Vicente Bermejo Bermejo
Requisitos Asociados	- IRQ-3 Información sobre cliente. - IRQ-4 Información sobre juegos por turnos. - REQ-2 Cliente conecta y se comunica con servidor.
Descripción	Esta asociación representa la tenencia de un cliente Wi-Fi en la clase MyCommunicationGameActivity
Comentarios	Ninguno.

Roles de la Asociación **Tiene** (**MyCommunicationGameActivity**, **WiFiClient**).

Rol	MyCommunicationGameActivity **juega rol** tenedor **en** Tiene (MyCommunicationGameActivity, WiFiClient)
Descripción	La activity base para juegos multijugador tiene Cliente Wi-Fi.
Tipo OCL	MyCommunicationGameActivity
Multiplicidad	1
Comentarios	Ninguno.

Rol	WiFiClient **juega rol** tenencia **en** Tiene (MyCommunicationGameActivity, WiFiClient)
Descripción	ClientServer es tenido por MyCommunicationGameActivity.
Tipo OCL	ClientServer
Multiplicidad	0…1
Comentarios	Ninguno.

Asociación **Tiene** (**MyCommunicationGameActivity**, **BluetoothClient**).

Descripción de la Asociación **Tiene** (**MyCommunicationGameActivity**, **BluetoothClient**).

Asociación	**Tiene entre MyCommunicationGameActivity y BluetoothClient**
Versión	1.0
Autores	Vicente Bermejo Bermejo
Requisitos Asociados	- IRQ-3 Información sobre cliente. - IRQ-4 Información sobre juegos por turnos. - REQ-2 Cliente conecta y se comunica con servidor.
Descripción	Esta asociación representa la tenencia de un cliente Bluetooth en la clase MyCommunicationGameActivity
Comentarios	Ninguno.

Roles de la Asociación **Tiene** (**MyCommunicationGameActivity**, **BluetootServer**).

Rol	MyCommunicationGameActivity **juega rol** tenedor **en** Tiene (MyCommunicationGameActivity, BluetoothClient)
Descripción	La activity base para juegos multijugador tiene Cliente Bluetooth.
Tipo OCL	MyCommunicationGameActivity
Multiplicidad	1
Comentarios	Ninguno.

Rol	BluetoothClient **juega rol** tenencia **en** Tiene (MyCommunicationGameActivity, BluetoothClient)
Descripción	BluetoothClient es tenido por MyCommunicationGameActivity.
Tipo OCL	BluetoothClient
Multiplicidad	0…1
Comentarios	Ninguno.

Asociación **Utiliza** (**PantallaJuegoTicTacToe**, **TicTacToe**).

Descripción de la Asociación **Utiliza** (**PantallaJuegoTicTacToe**, **TicTacToe**).

Asociación	**Utiliza entre PantallaJuegoTicTacToe y TicTacToe**
Versión	1.0
Autores	Vicente Bermejo Bermejo
Requisitos Asociados	- IRQ-4 Información sobre juegos por turnos. - REQ-1 Usuario juega partidas.
Descripción	Esta asociación representa la utilización de la lógica del juego TicTacToe en su pantalla de juego principal.
Comentarios	Ninguno.

Roles de la Asociación **Utiliza (PantallaJuegoTicTacToe, TicTacToe)**.

Rol	**PantallaJuegoTicTacToe** juega rol **utilizador** en **Utiliza (PantallaJuegoTicTacToe, TicTacToe)**
Descripción	La pantalla principal del juego utiliza la lógica del mismo para actualizarse.
Tipo OCL	PantallaJuegoTicTacToe
Multiplicidad	1
Comentarios	La pantalla muestra los cambios producidos en la lógica de juego.

Rol	**TicTacToe** juega rol **utensilio** en **Utiliza(PantallaJuegoTicTacToe, TicTacToe)**
Descripción	TicTacToe es utilizado por PantallaJuegoTicTacToe .
Tipo OCL	TicTacToe
Multiplicidad	0…1
Comentarios	La pantalla también transmite las entradas de usuario a la lógica de juego.

Asociación **Usa** (**PantallaJuegoTicTacToe**, **TicTacToeCPU**).

Descripción de la Asociación **Usa** (**PantallaJuegoTicTacToe**, **TicTacToeCPU**).

Asociación	**Usa entre PantallaJuegoTicTacToe y TicTacToeCPU**
Versión	1.0
Autores	Vicente Bermejo Bermejo
Requisitos Asociados	- IRQ-4 Información sobre juegos por turnos. - REQ-3 El sistema simula jugadores.
Descripción	Esta asociación representa el uso de un elemento específico para simular movimientos en el juego TicTacToe.
Comentarios	Ninguno.

Roles de la Asociación **Utiliza (PantallaJuegoTicTacToe, TicTacToeCPU)**.

Rol	**PantallaJuegoTicTacToe** juega rol **usador** en **Utiliza (PantallaJuegoTicTacToe, TicTacToeCPU)**
Descripción	La pantalla principal del juego usa un simulador de movimientos TicTacToe.
Tipo OCL	PantallaJuegoTicTacToe
Multiplicidad	1
Comentarios	La pantalla muestra los movimientos generados por el simulador (CPU).

Rol	**TicTacToeCPU** juega rol **elemento usado** en **Utiliza(PantallaJuegoTicTacToe, TicTacToeCPU)**
Descripción	TicTacToeCPU es usado por PantallaJuegoTicTacToe .
Tipo OCL	TicTacToeCPU
Multiplicidad	0…1
Comentarios	Ninguno.

Asociación **inicia** (**PantallaPrincipalTicTacToe**, **PantallaJuegoTicTacToe**).

Descripción de la Asociación **inicia** (**PantallaPrincipalTicTacToec, PantallaJuegoTicTacToe**).

Asociación	**inicia entre PantallaPrincipalTicTacToec y PantallaJuegoTicTacToe)**
Versión	1.0
Autores	Vicente Bermejo Bermejo
Requisitos Asociados	- IRQ-4 Información sobre juegos por turnos. - REQ-1 Usuario juega partidas.
Descripción	Esta asociación representa el inicio de una partida TicTacToe.
Comentarios	Ninguno.

Roles de la Asociación **Utiliza (PantallaJuegoTicTacToe, TicTacToeCPU)**.

Rol	**PantallaPrincipalTicTacToe** juega rol **iniciador** en **inicia (PantallaPrincipalTicTacToec, PantallaJuegoTicTacToe)**
Descripción	La pantalla principal inicia la pantalla de juego.
Tipo OCL	PantallaPrincipalTicTacToe
Multiplicidad	1
Comentarios	La pantalla muestra los movimientos generados por el simulador (CPU).

Rol	**PantallaJuegoTicTacToe** juega rol **elemento iniciado** en **inicia (PantallaPrincipalTicTacToec, PantallaJuegoTicTacToe)**
Descripción	La pantalla de juego es iniciada por la pantalla principal.
Tipo OCL	PantallaJuegoTicTacToe
Multiplicidad	1
Comentarios	Ninguno.

Asociación **Tiene** (**TicTacToe**, **TableroTicTacToe**).

Descripción de la Asociación **Tiene** (**TicTacToe**, **TableroTicTacToe**).

Asociación	Tiene **entre TicTacToe y Tablero TicTacToe**
Versión	1.0
Autores	Vicente Bermejo Bermejo
Requisitos Asociados	- IRQ-4 Información sobre juegos por turnos. - REQ-1 Usuario juega partidas.
Descripción	Esta asociación representa el hecho de que la lógica del juego TicTacToe tenga un tablero de tipo TicTacToe.
Comentarios	Ninguno.

Roles de la Asociación **Tiene** (**TicTacToe**, **TableroTicTacToe**).

Rol	TicTacToe **juega rol** tenedor **en Tiene (TicTacToe, TableroTicTacToe).**
Descripción	La lógica del juego TicTacToe tiene un TableroTicTacToe
Tipo OCL	TicTacToe
Multiplicidad	1
Comentarios	Ninguno.

Rol	TableroTicTacToe **juega rol** elemento tenido **en Tiene** (TicTacToe, TableroTicTacToe)
Descripción	Los TableroTicTacToe son tenidos por TicTacToe.
Tipo OCL	TableroTicTacToe
Multiplicidad	1
Comentarios	Ninguno.

Asociación muestra (BaseActivity, MyScene).

Descripción de la Asociación muestra (BaseActivity, MyScene)

Asociación	**<nombre asociación> entre <participantes>**
Versión	1.0
Autores	Vicente Bermejo Bermejo
Requisitos Asociados	- IRQ-4 Información sobre juegos por turnos. - REQ-1 Usuario juega partidas.
Descripción	Esta asociación representa la actividad base del juego Battleship muestra escenas.
Comentarios	Sólo puede mostrar una única escena al mismo tiempo.

Roles de la Asociación muestra (BaseActivity, MyScene)

Rol	BaseaActivity **juega rol** mostrador **en muestra (BaseActivity, MyScene)**
Descripción	BaseActivity muestra escenas.
Tipo OCL	BaseActivity
Multiplicidad	1
Comentarios	Ninguno.

Rol	MyScene **juega rol** muestra **en muestra (BaseActivity, MyScene)**
Descripción	MyScene es mostrada en BaseActivity
Tipo OCL	MyScene
Multiplicidad	1
Comentarios	Una escena no puede existir sin una BaseActivity en la que mostrarse.

Asociación inicia (PreGameScene, GameScene)

Descripción de la Asociación inicia (PreGameScene, GameScene).

Asociación	**inicia entre PreGameScene y GameScene**
Versión	1.0
Autores	Vicente Bermejo Bermejo
Requisitos Asociados	- IRQ-4 Información sobre juegos por turnos. - REQ-1 Usuario juega partidas.
Descripción	Esta asociación representa el hecho de que PreGameScene inicia GameScene
Comentarios	Ninguno.

Roles de la Asociación inicia (PreGameScene, GameScene).

Rol	PreGameScene **juega rol** GameScene **en inicia (PreGameScene, GameScene).**
Descripción	PreGameScene inicia la pantalla de juego.
Tipo OCL	PreGameScene
Multiplicidad	1
Comentarios	PreGameScene representa la escena en la que se colocan los barcos propios en el tablero.

Rol	GameScene **juega rol** iniciado **en inicia (PreGameScene, GameScene).**
Descripción	GameScene es iniciada por PreGrameScene.
Tipo OCL	GameScene
Multiplicidad	1
Comentarios	GameScene es la escena principal del juego (donde se realizan los movimientos).

Asociación inicia (PreGameScene, Battleship)

Descripción de la Asociación inicia (PreGameScene, Battleship)

Asociación	**inicia entre PreGameScene y Battleship**
Versión	1.0
Autores	Vicente Bermejo Bermejo
Requisitos Asociados	- IRQ-4 Información sobre juegos por turnos. - REQ-1 Usuario juega partidas.
Descripción	Esta asociación representa el hecho de que PreGameScene inicia Battleship.
Comentarios	Ninguno.

Roles de la Asociación inicia (PreGameScene, Battleship).

Rol	PreGameScene **juega rol** GameScene **en inicia (PreGameScene, Battleship)**
Descripción	PreGameScene inicia e inicializad la lógica del juego Battleship.
Tipo OCL	PreGameScene
Multiplicidad	1
Comentarios	PreGameScene representa la escena en la que se colocan los barcos propios en el tablero.

Rol	GameScene **juega rol** iniciado **en inicia (PreGameScene, Battleship)**
Descripción	Battleship es iniciado e inicializado por PreGrameScene.
Tipo OCL	Battleship
Multiplicidad	1
Comentarios	Ninguno.

Asociación **Utiliza** (**GameScene**, **Battleship**).

Descripción de la Asociación **Utiliza** (**GameScene**, **Battleship**).

Asociación	**Utiliza (GameScene , Battleship)**
Versión	1.0
Autores	Vicente Bermejo Bermejo
Requisitos Asociados	- IRQ-4 Información sobre juegos por turnos. - REQ-1 Usuario juega partidas.
Descripción	Esta asociación representa la utilización de la lógica del juego Battleship en su escena de juego principal.
Comentarios	Ninguno.

Roles de la Asociación **Utiliza (GameScene, Battleship)**.

Rol	**GameScene** juega rol **utilizador** en **Utiliza (GameScene , Battleship)**
Descripción	La escena de juego utiliza la lógica de Battleship para actualizarse.
Tipo OCL	GameScene
Multiplicidad	1
Comentarios	La escena muestra los cambios producidos en la lógica de juego.

Rol	**Battleship** juega rol **utensilio** en **Utiliza(GameScene , Battleship)**
Descripción	Battleship es utilizado por BaseActivity.
Tipo OCL	Battleship
Multiplicidad	0…1
Comentarios	La escena transmite las entradas de usuario a la lógica de juego.

Asociación **Usa** (**GameScene**, **BattleshipCPU**).

Descripción de la Asociación **Usa** (**GameScene**, **BattleshipCPU**).

Asociación	**Usa (GameScene, BattleshipCPU)**
Versión	1.0
Autores	Vicente Bermejo Bermejo
Requisitos Asociados	- IRQ-4 Información sobre juegos por turnos. - REQ-3 El sistema simula jugadores.
Descripción	Esta asociación representa el uso de un elemento específico para simular un jugador simulado por la CPU en el juego Battleship.
Comentarios	Ninguno.

Roles de la Asociación **Utiliza (GameScene, BattleshipCPU)**.

Rol	**GameScene** juega rol **usador** en **Utiliza (GameScene, BattleshipCPU)**
Descripción	La escena principal del juego usa un simulador BattleshipCPU.
Tipo OCL	GameScene
Multiplicidad	1
Comentarios	La pantalla muestra los movimientos generados por el simulador (CPU).

Rol	**BattleshipCPU** juega rol **elemento usado** en **Utiliza(GameScene, BattleshipCPU)**
Descripción	BattleshipCPU es usado por GameScene.
Tipo OCL	BattleshipCPU
Multiplicidad	0…1
Comentarios	Ninguno.

Asociación **Posee** (**Battleship**, **TableroBattleship**).

Descripción de la Asociación **Posee** (**Battleship**, **TableroBattleship**).

Asociación	**Posee entre Battleship y TableroBattleship**
Versión	1.0
Autores	Vicente Bermejo Bermejo
Requisitos Asociados	- IRQ-4 Información sobre juegos por turnos. - REQ-1 Usuario juega partidas.
Descripción	Esta asociación representa el hecho de que la lógica del juego Battleship tenga un tablero de tipo Battleship.
Comentarios	Ninguno.

Roles de la Asociación **Posee** (**Battleship**, **TableroBattleship**).

Rol	**Battleship juega rol** poseedor **en** posee **(Battleship , TableroBattleship).**
Descripción	La lógica del juego Battleship tiene un Tablero Battleship
Tipo OCL	Battleship
Multiplicidad	1
Comentarios	Ninguno.

Rol	Tablero**Battleship juega rol** posesión **en** posee (**Battleship** , Tablero**Battleship**)
Descripción	Los TableroBattleship son tenidos por TicTacToe.
Tipo OCL	TableroTicTacToe
Multiplicidad	1
Comentarios	Ninguno.

Modelo de comportamiento del sistema

Diagramas de secuencia del sistema (DSS)

Diagramas de Secuencia subsistema Comunicaciones.

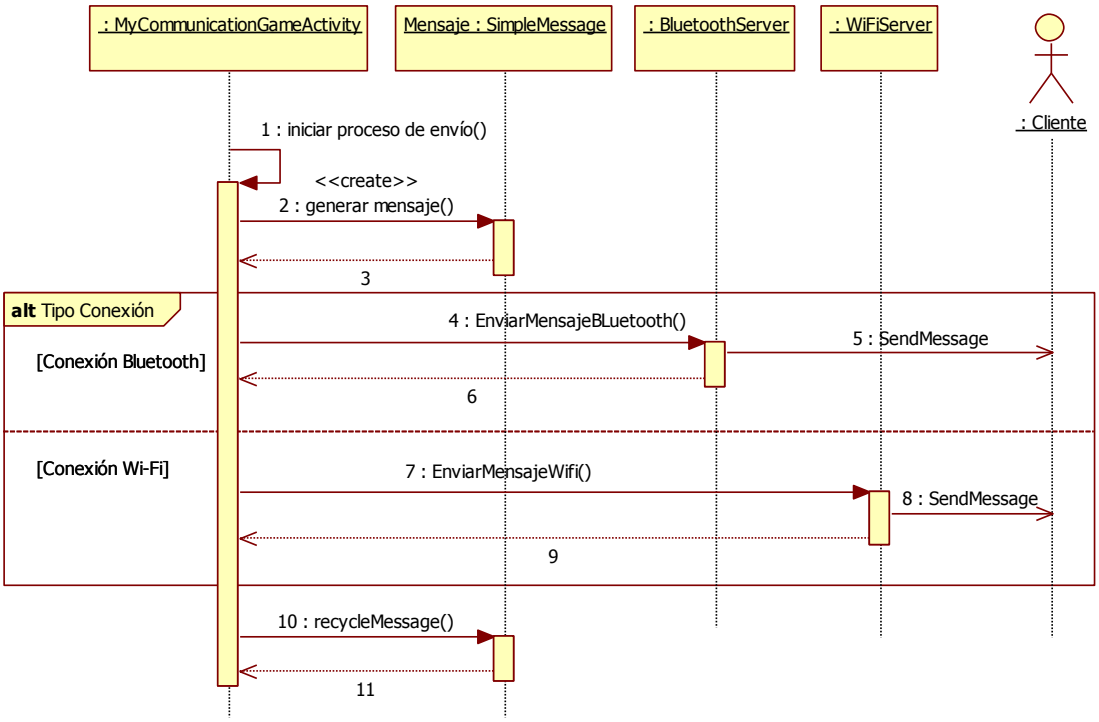


Figura 21. Diagrama Secuencia Enviar mensaje (se servidor a cliente)

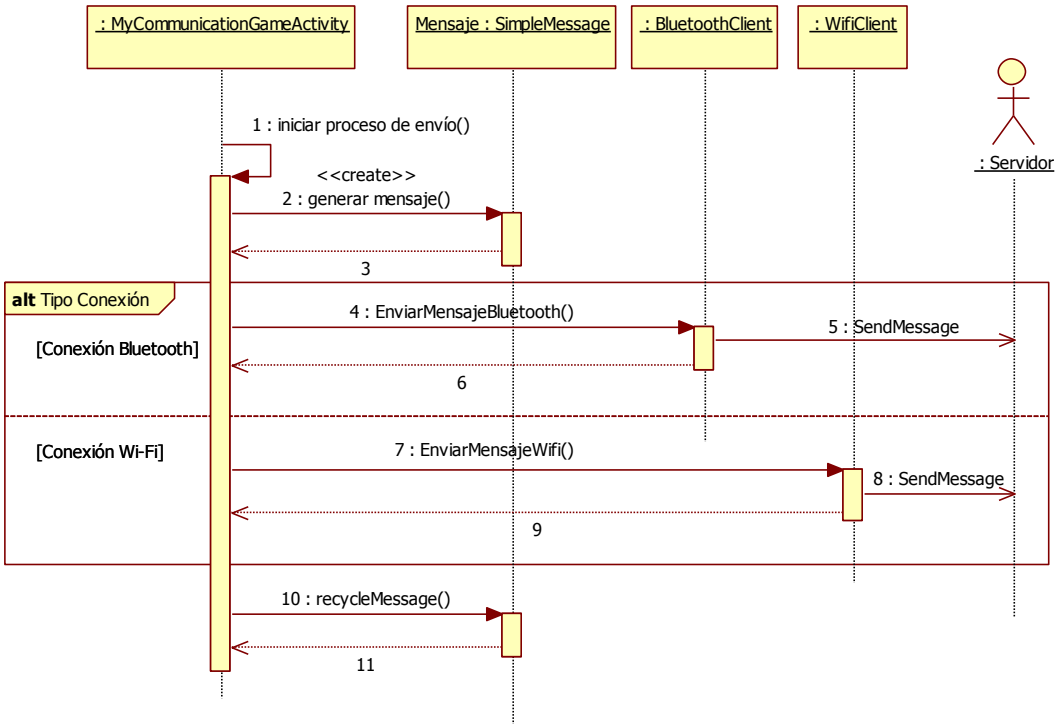


Figura 22. Diagrama Secuencia Enviar mensaje (de cliente a servidor)

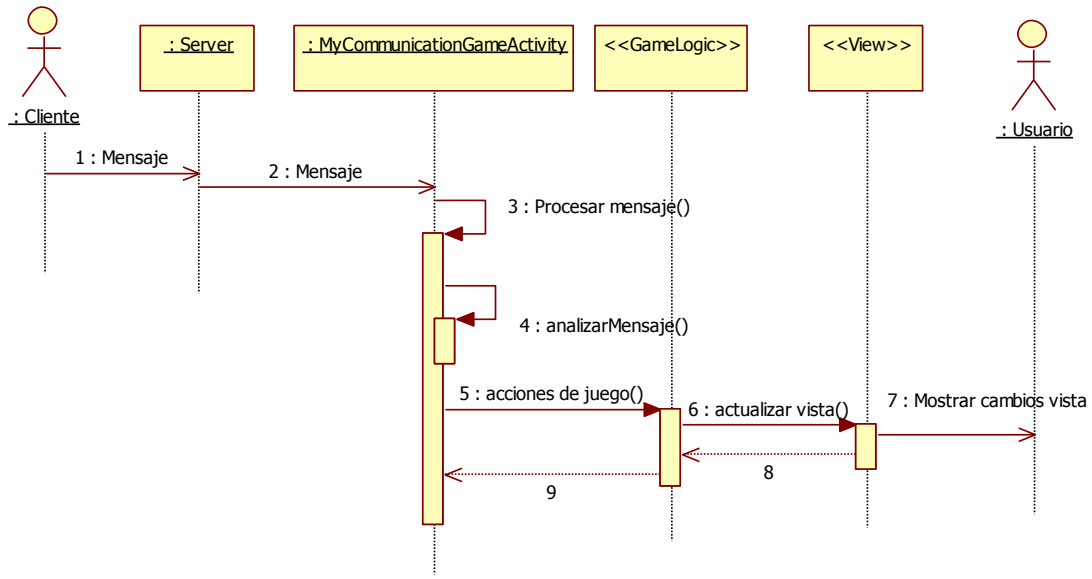


Figura 23. Diagrama Secuencia Procesar mensaje (en servidor)

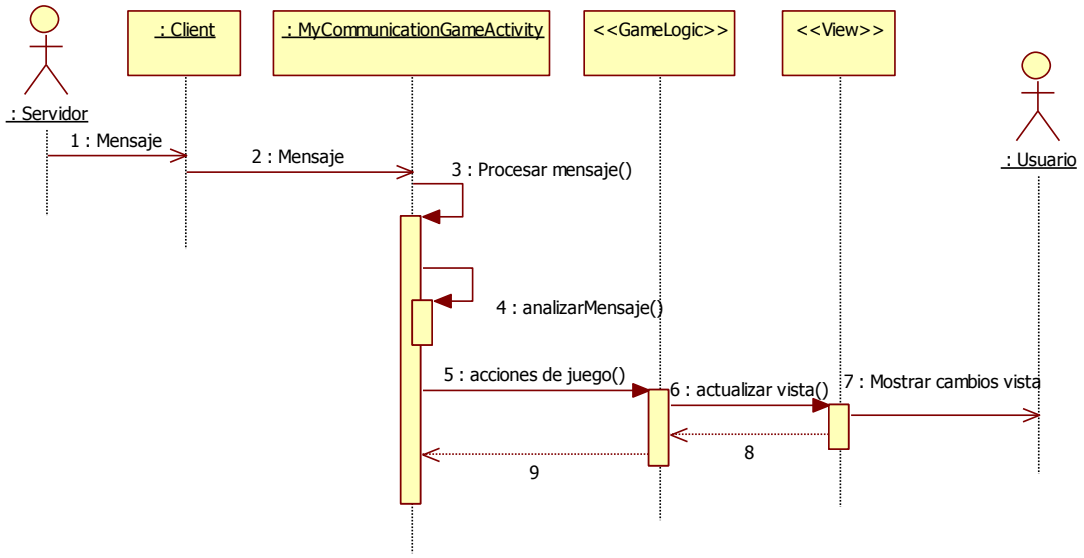


Figura 24. Diagrama Secuencia Procesar mensaje (en cliente)

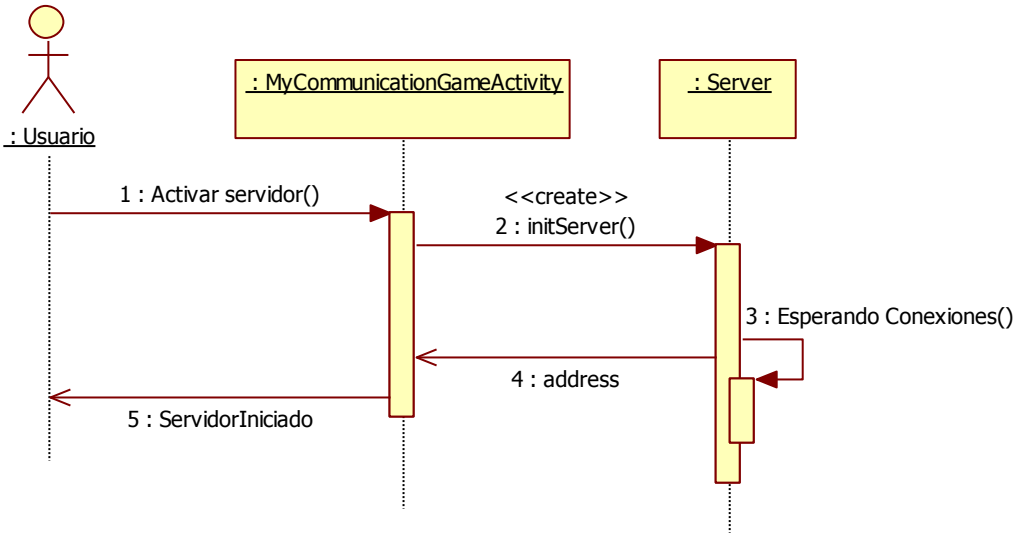


Figura 25. Diagrama Secuencia Iniciar Servidor.

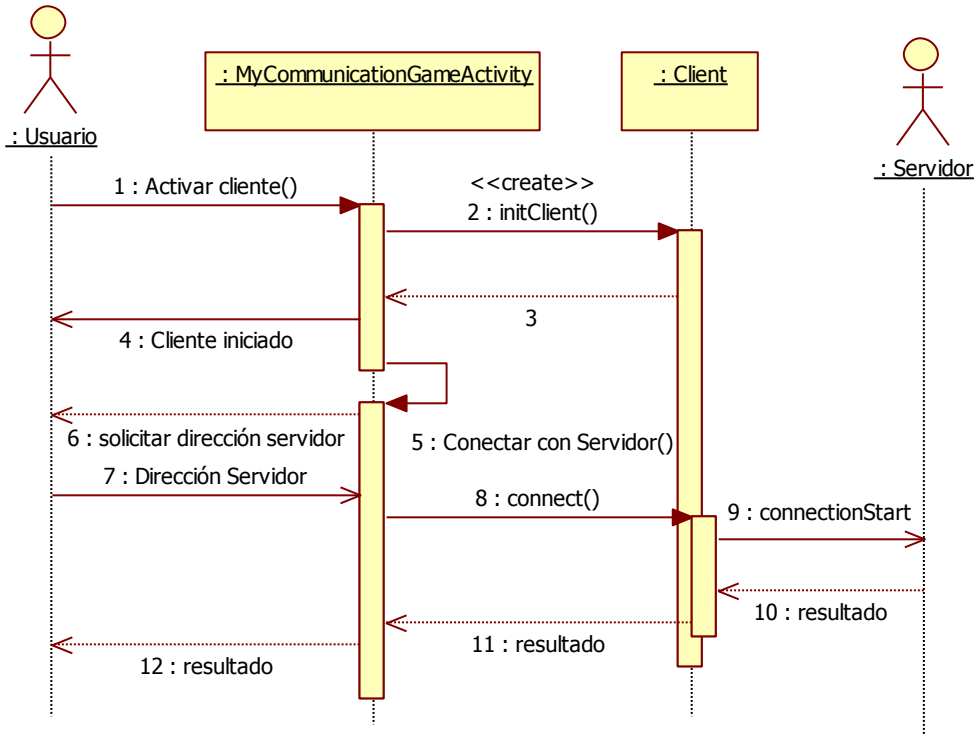


Figura 26. Diagrama Secuencia Iniciar Cliente y conectar con Servidor.

Diagramas de Secuencia subsistema Juegos.

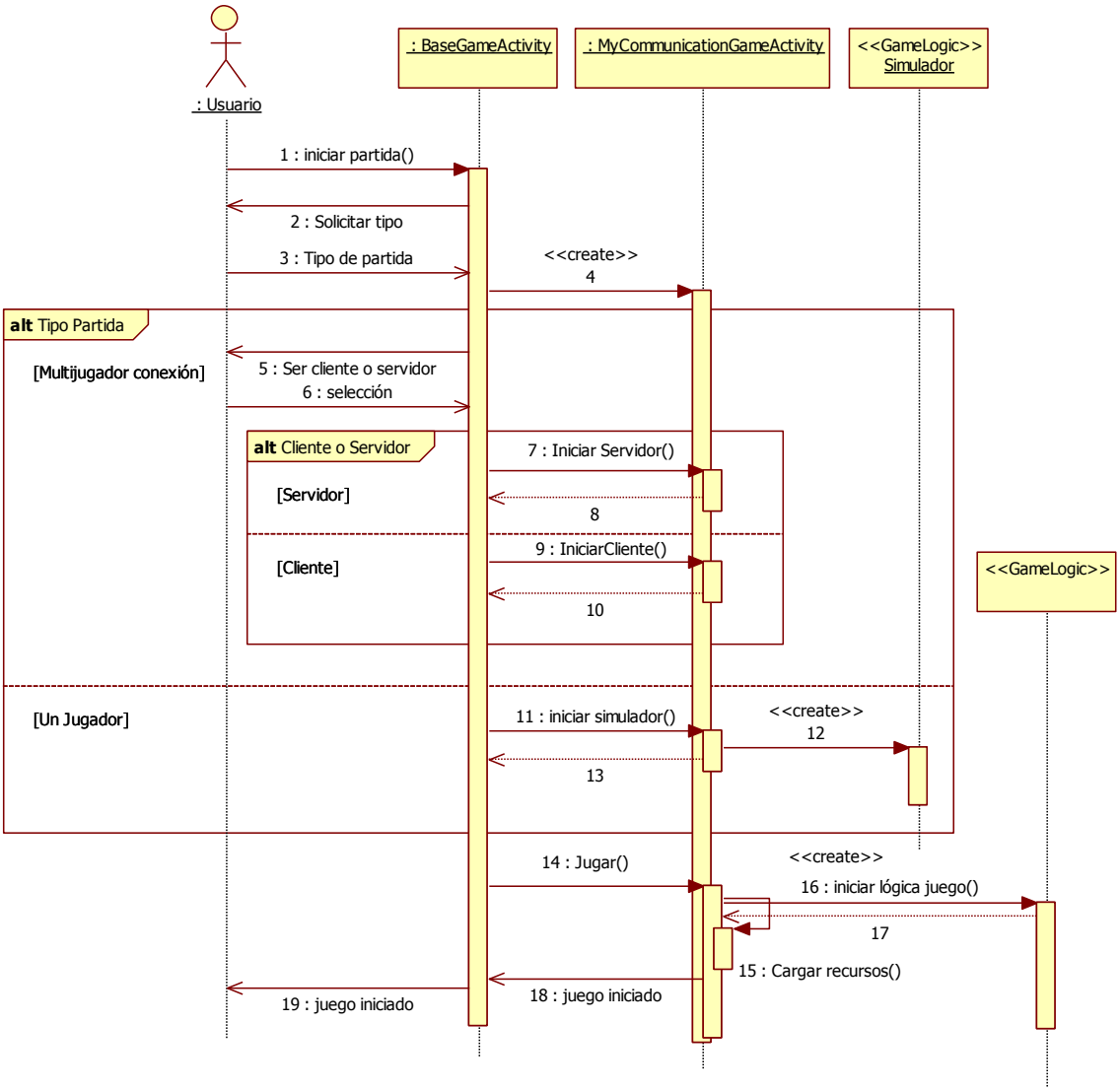


Figura 27. Diagrama Secuencia Iniciar partida.

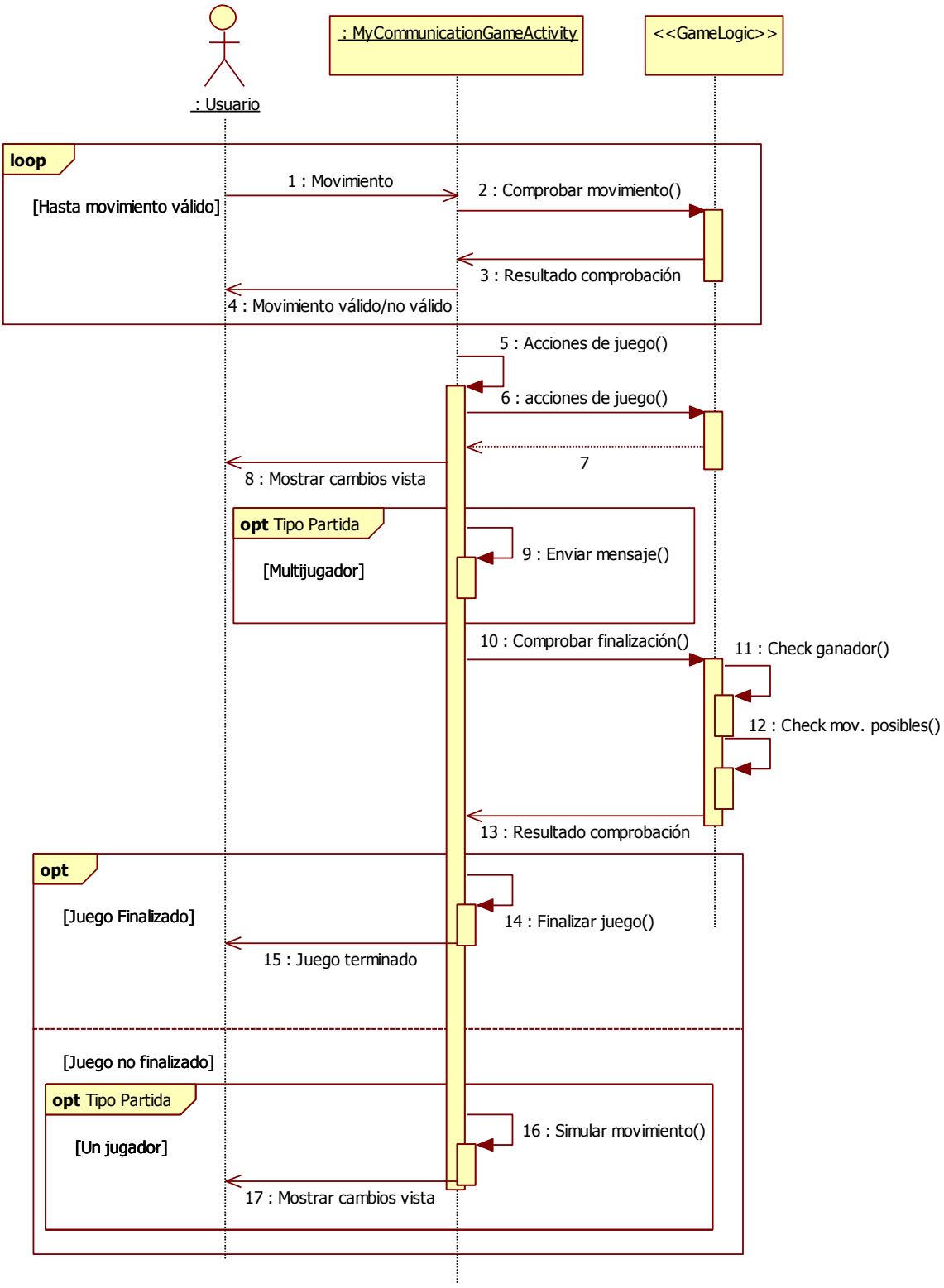


Figura 28. Diagrama Secuencia Realizar movimiento.

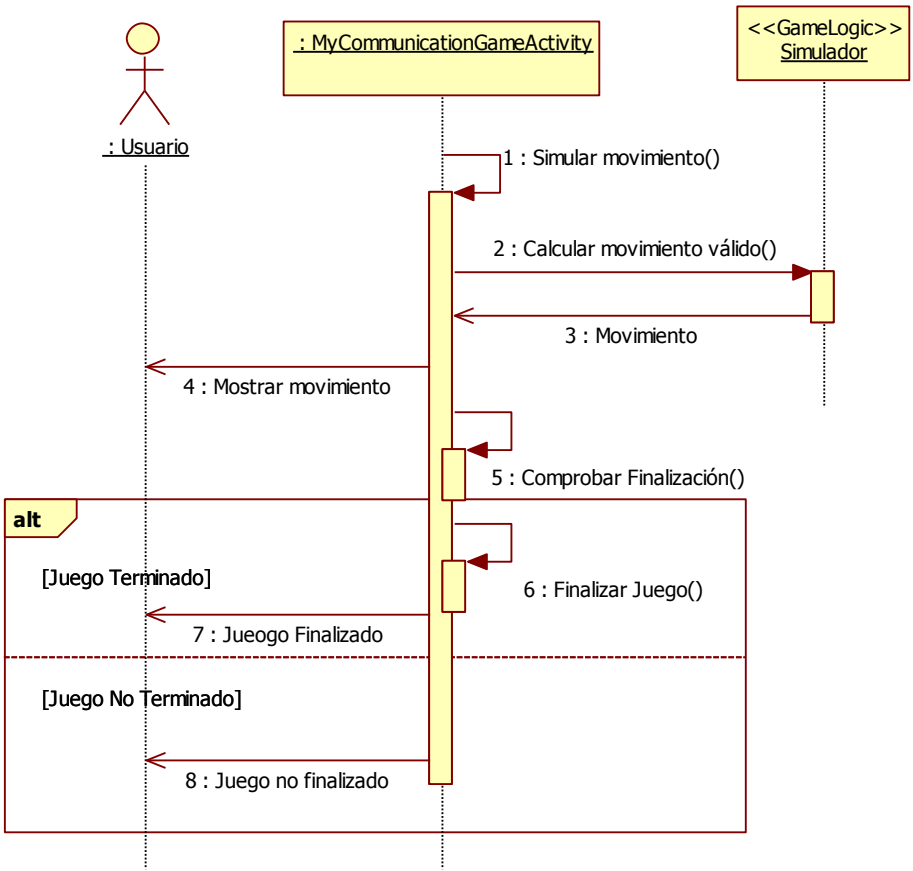


Figura 29. Diagrama Secuencia Simular movimiento.

Diagramas de Secuencia funcionamiento general de juego TicTacToe

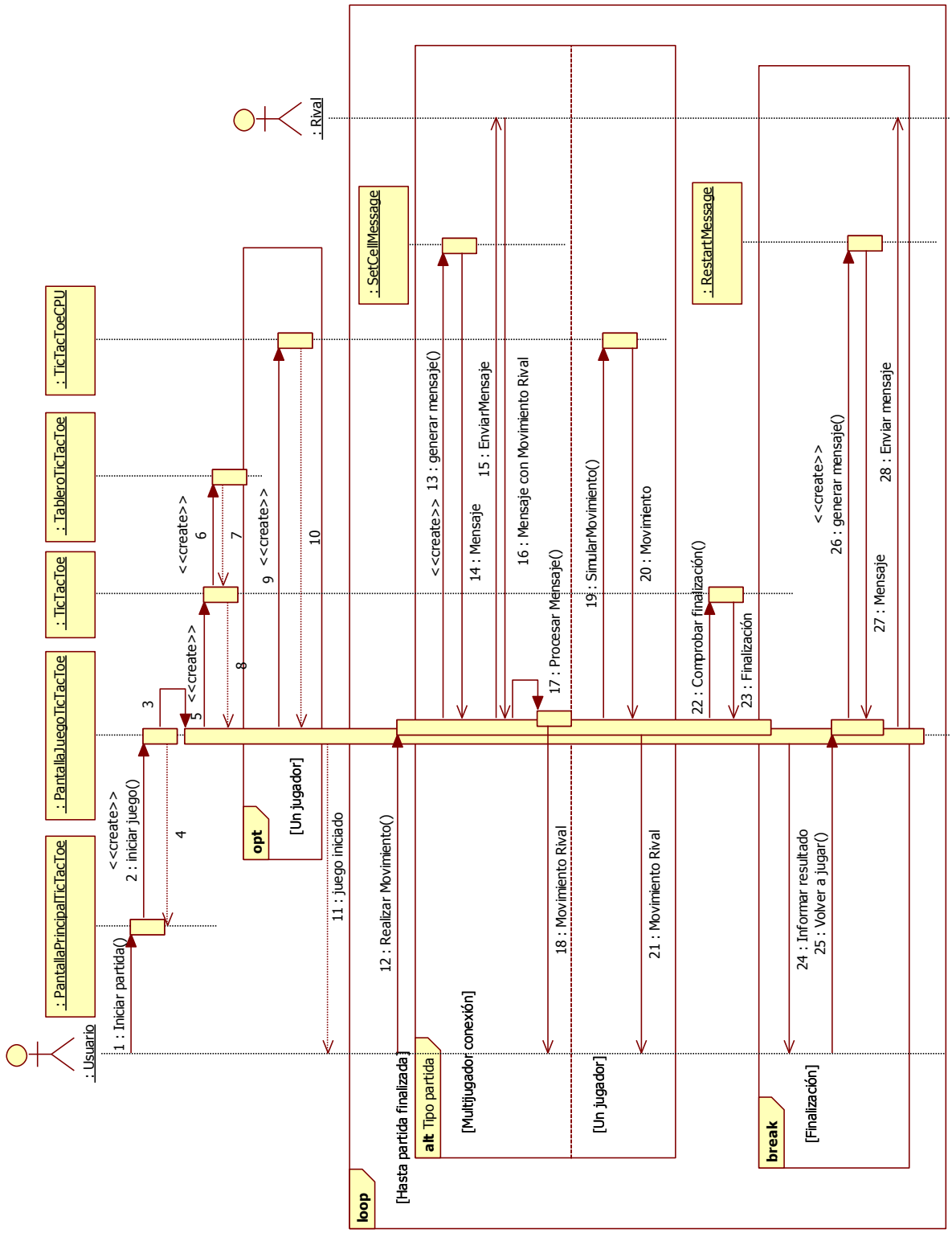


Figura 30. Diagrama Secuencia comportamiento general juego TicTacToe

Diagramas de Secuencia funcionamiento general de juego Battleship

Esta sección cuenta con tres diferentes para que se puedan apreciar de manera correcta. En primer lugar se mostrará el funcionamiento inicial del juego Battleship (los preparativos previos al juego). Seguidamente se mostrará el comportamiento del sistema durante el juego cuando el usuario actual está en posesión del turno. Finalmente, se reflejará con otro diagrama de secuencia el comportamiento del juego cuando es el rival quien está en posesión del turno de juego.

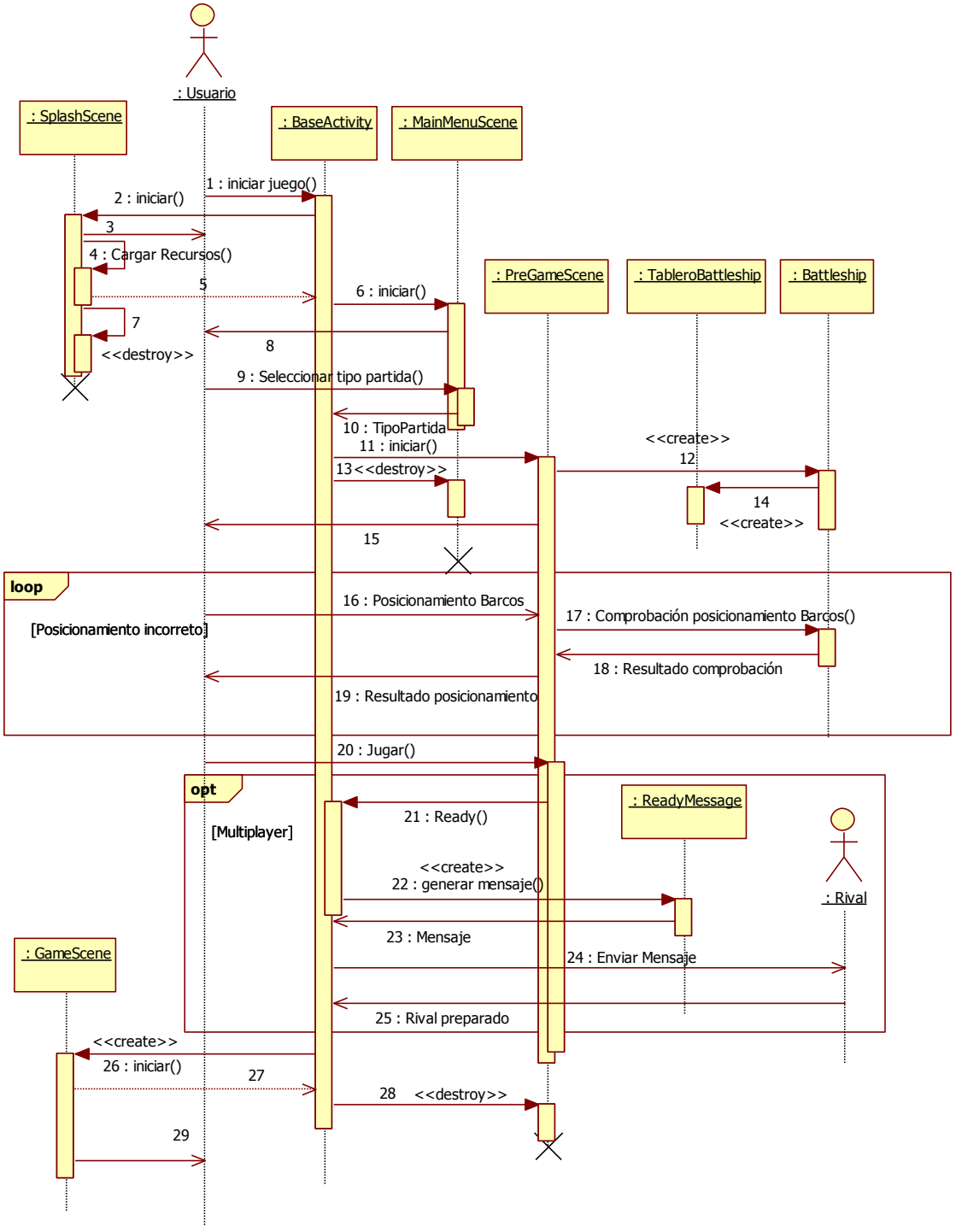


Figura 31. Diagrama Secuencia comportamiento general juego Battleship. Parte de inicio de juego.

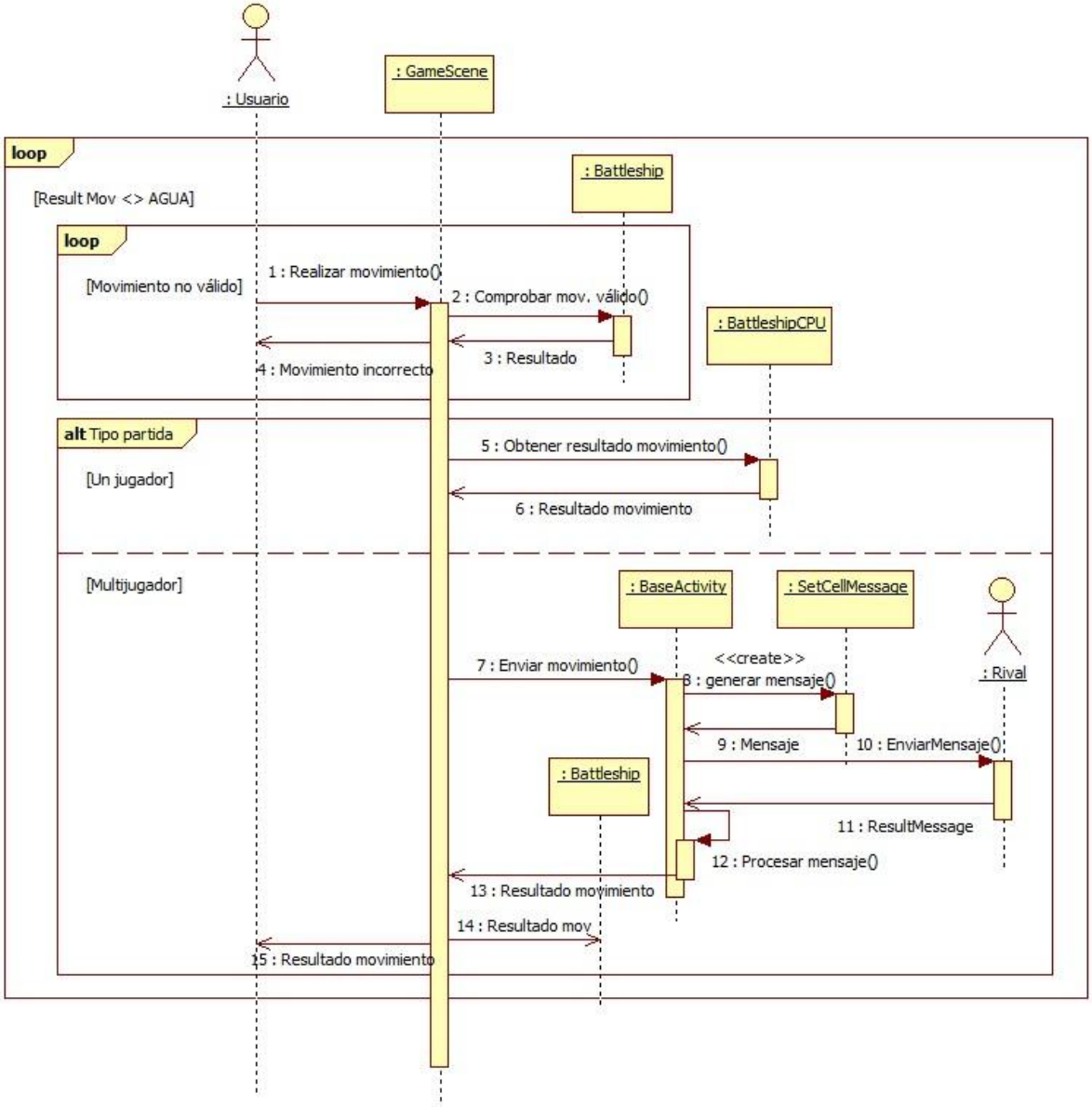


Figura 32. Diagrama Secuencia comportamiento general juego Battleship. Parte juego (Usuario con turno).

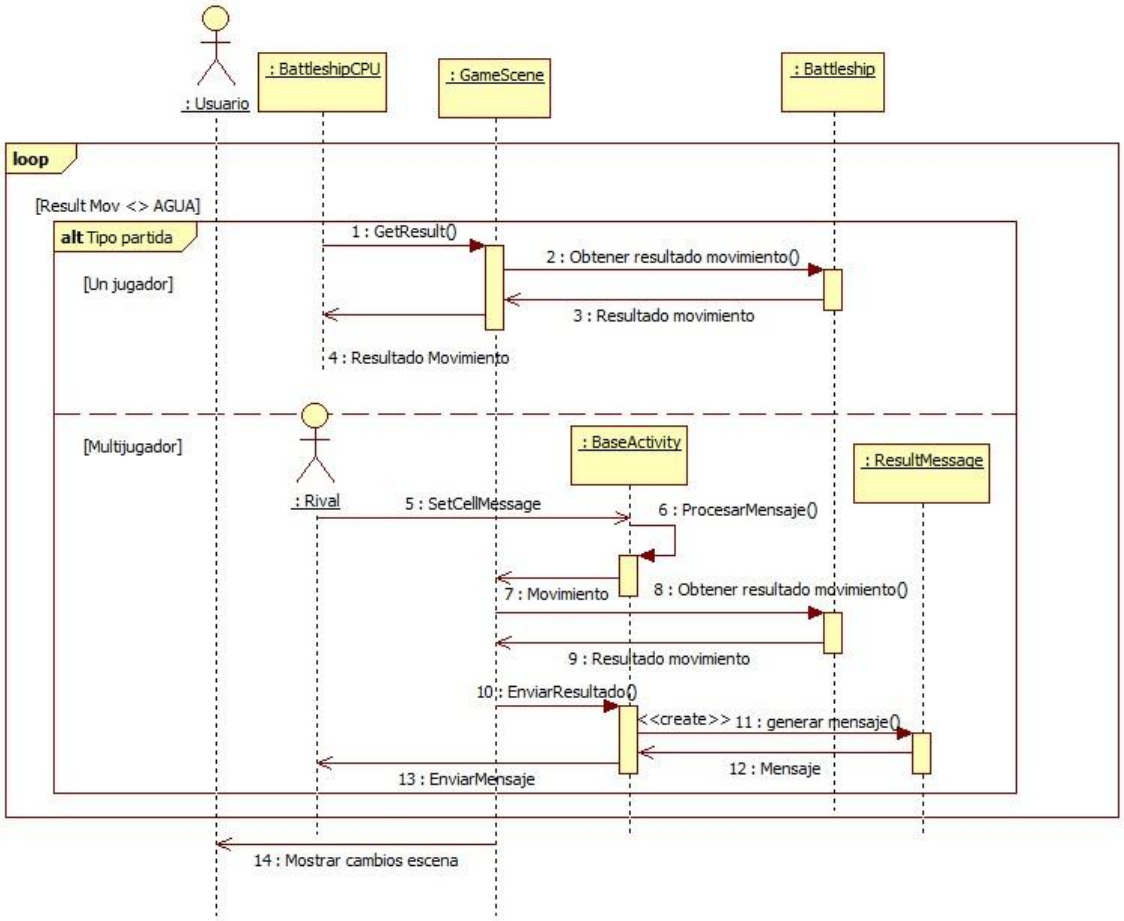


Figura 33. Diagrama Secuencia comportamiento general juego Battleship. Parte juego (Turno del rival).

Diagramas de estados del sistema (DES)

Esta sección contiene los diagramas de estados necesarios que representan estados y cambios de estados representativos del sistema y/o subsistemas y/o casos de uso.
Un diagrama de estado del sistema describe la "secuencia de eventos del sistema externo" que reconoce y maneja un sistema en el contexto de un caso de uso. Es conveniente desarrollar un diagrama de estados para cada caso de uso no trivial.

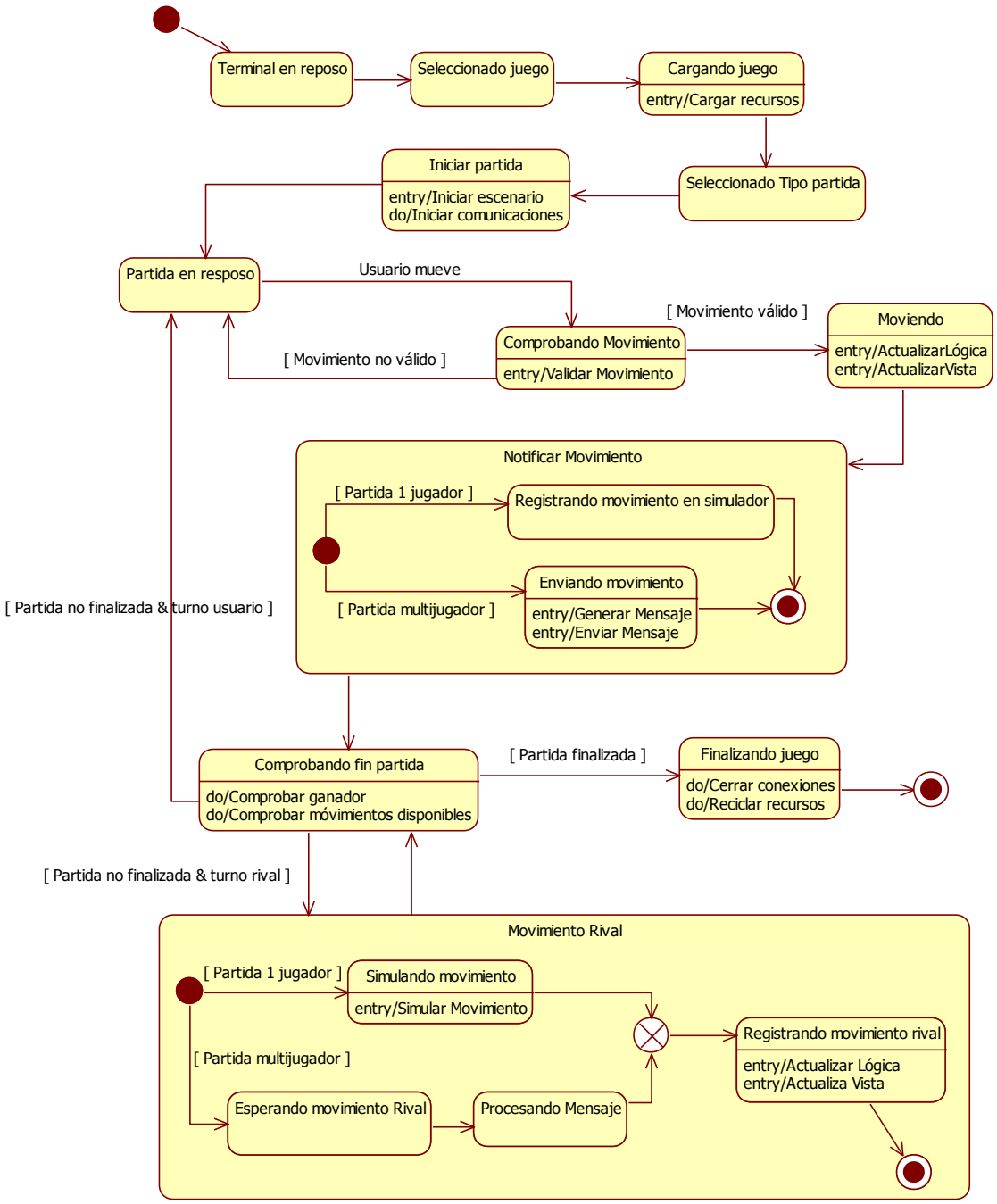


Figura 34. Diagrama de Estados proceso de juego.

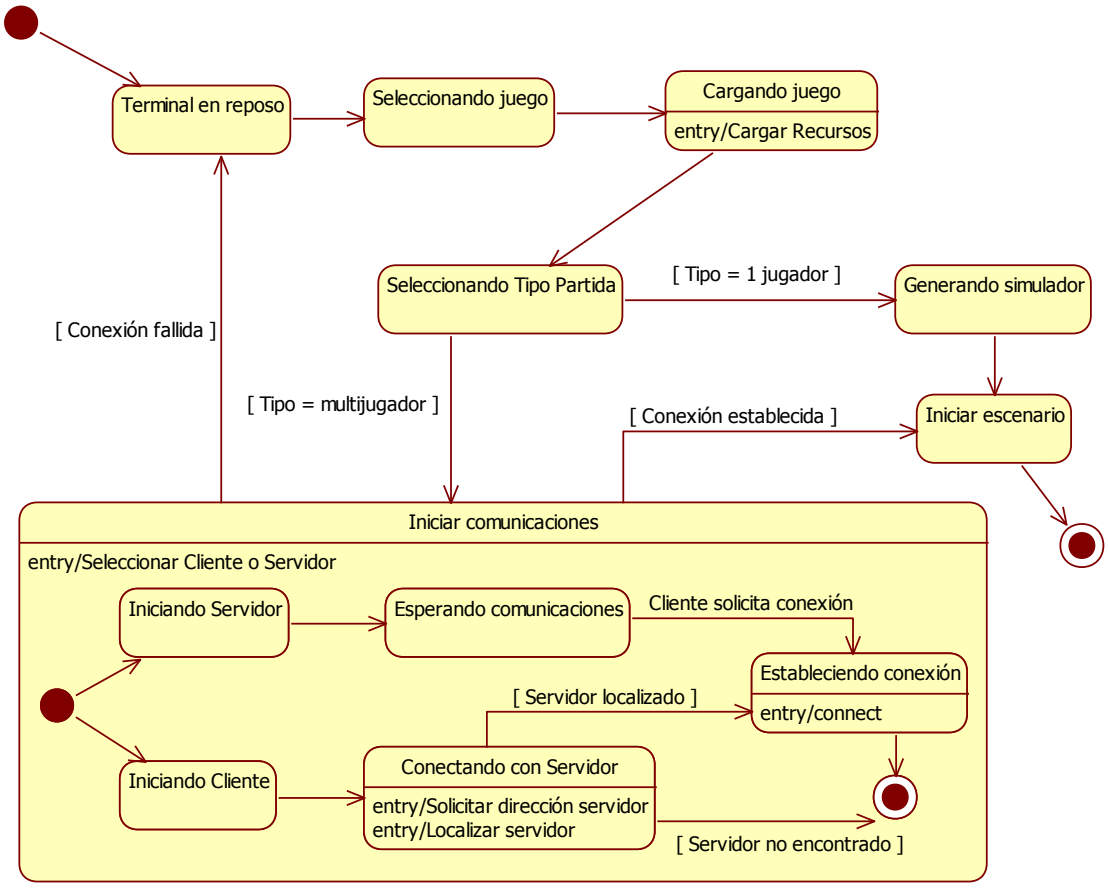


Figura 35. Diagrama de Estados Iniciar Partida

Diagramas de Actividad (DA)

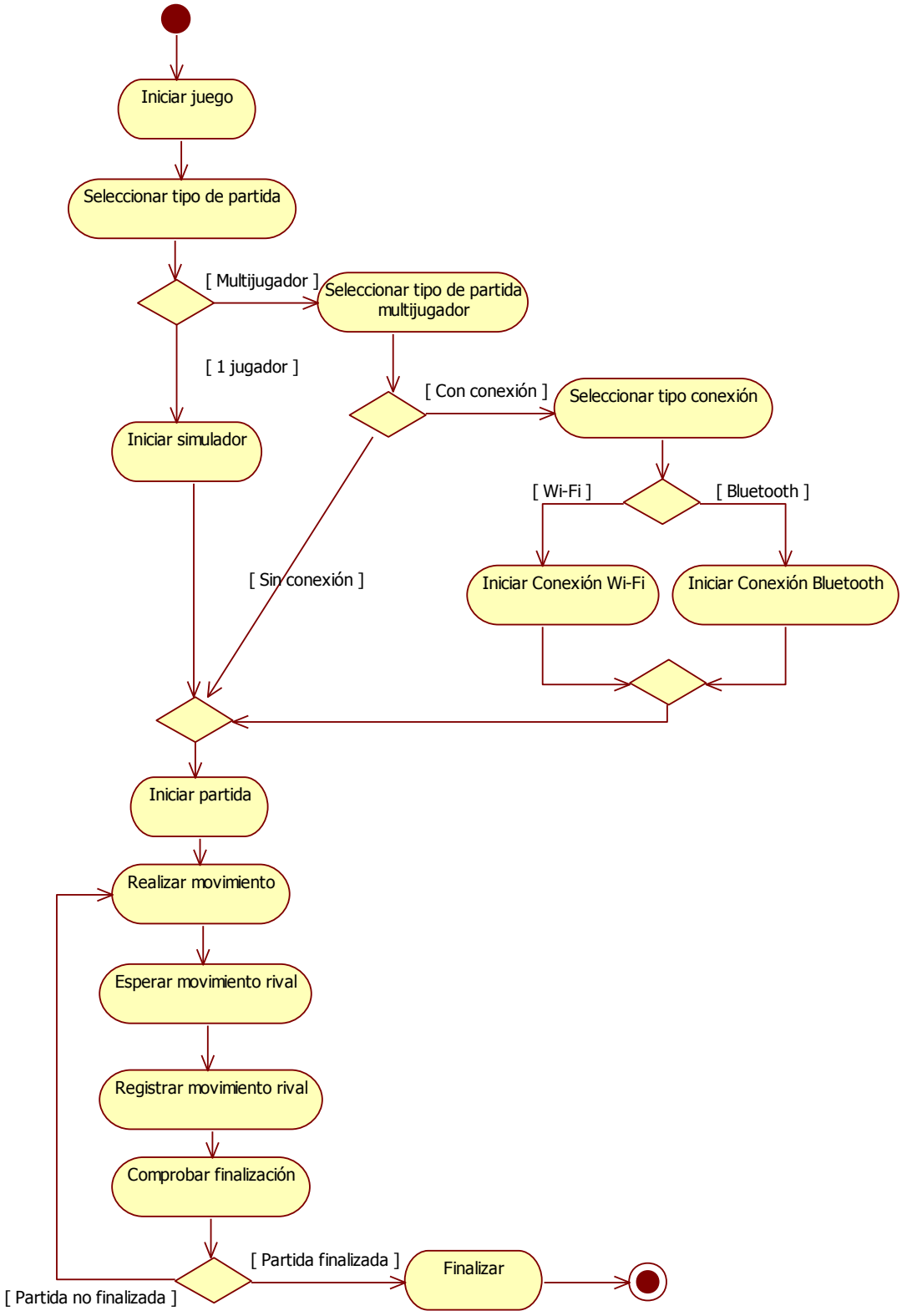


Figura 36. Diagrama de Actividad partida TicTacToe.

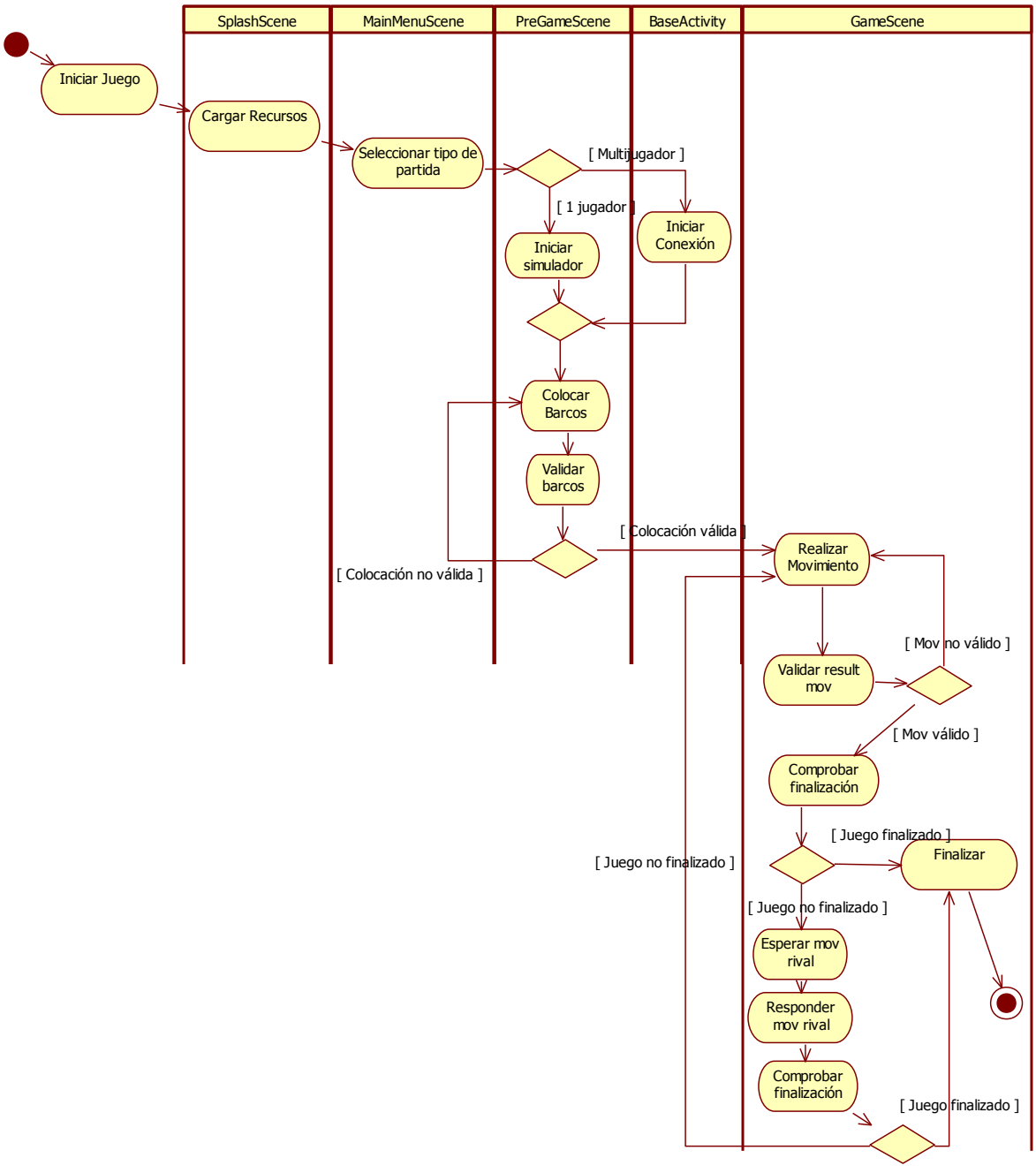


Figura 37. Diagrama de Actividad juego Battleship.

Matriz de Rastreabilidad Requisitos/Análisis (elementos de modelado).

Esta sección contiene una matriz requisitos/elementos de modelado, de forma que para cada elemento de modelado se puede identificar el requisito asociado, y ascendentemente el objetivo asociado. El formato de la matriz de rastreabilidad es el siguiente:

Tipo de objeto	**IRQ** asociados	**UC** asociados
Server	IRQ-2	UC-1, UC-4, UC-5
WiFiServer	IRQ-2	UC-1, UC-3, UC-4, UC-5
BluetoothServer	IRQ-2	UC-1, UC-2, UC-4, UC-5
Client	IRQ-3	UC-1, UC-4, UC-6, UC-7
WiFiClient	IRQ-2.2	UC-1, UC-3, UC-4, UC-6, UC-7
BluetoothClient	IRQ-1.2	UC-1, UC-2, UC-4, UC-6, UC-7
SimpleMessage	IRQ-1	UC-1, UC-2, UC-3, UC-4
MyCommunicationGameActivity	IRQ-2, IRQ-3	UC-1, UC-2, UC-3, UC-4, UC-5, UC-6, UC-7, UC-8
PantallaJuegoTicTacToe	IRQ-1, IRQ-2, IRQ-3, IRQ-4	UC-1, UC-4, UC-5, UC-6, UC-7, UC-9, UC-10, UC-11, UC-12
PantallaPrincipalTicTacToe	IRQ-4	UC-8
TicTacToe	IRQ-4	UC-9, UC-10
TicTacToeCPU	IRQ-4	UC-10, UC-11
Tablero	IRQ-4	UC-9, UC-10, UC-11, UC-12
TableroTicTacToe	IRQ-4	UC-9, UC-10, UC-11, UC-12
TableroBattleship	IRQ-4	UC-9, UC-10, UC-11, UC-12
BaseActivity	IRQ-1, IRQ-2, IRQ-3, IRQ-4	UC-9, UC-10, UC-11, UC-12
SplashScene	IRQ-4	UC-8
MainMenuScene	IRQ-4	UC-8
PreGameScene	IRQ-4	UC-1, UC-4, UC-5, UC-6, UC-7, UC-8
GameScene	IRQ-4	UC-9, UC-10, UC-11, UC-12
Battleship	IRQ-4	UC-9, UC-10, UC-12
BattleshipCPU	IRQ-4	UC-10, UC-11, UC-12

Resumen

TIPO	Descripción
Tipos de Objetos	Server
	WiFiServer
	BluetoothServer
	Client
	WiFiClient
	BluetoothClient
	SimpleMessage
	MyCommunicationGameActivity
	PantallaJuegoTicTacToe
	PantallaPrincipalTicTacToe
	TicTacToe
	TicTacToeCPU
	Tablero
	TableroTicTacToe
	TableroBattleship
	BaseActivity
	SplashScene
	MainMenuScene
	PreGameScene
	GameScene
	Battleship
	BattleshipCPU
Asociaciones	Se comunica con (Server, Client)
	Tiene (MyCommunicationGameActivity, WiFiServer)
	Tiene (MyCommunicationGameActivity, BluetoothServer)
	Tiene (MyCommunicationGameActivity, WiFiClient).
	Tiene (MyCommunicationGameActivity, BluetoothClient)
	Utiliza (PantallaJuegoTicTacToe, TicTacToe).
	Usa (PantallaJuegoTicTacToe, TicTacToeCPU)
	Inicia (PantallaPrincipalTicTacToe, PantallaJuegoTicTacToe)
	Tiene (TicTacToe, TableroTicTacToe).
	Muestra (BaseActivity, MyScene).
	inicia (PreGameScene, GameScene)
	inicia (PreGameScene, Battleship)
	Utiliza (GameScene, Battleship
	Usa (GameScene, BattleshipCPU)
	Posee (Battleship, TableroBattleship)

TIPO	Descripción
Diagramas de Secuencia	Enviar mensaje (se servidor a cliente)
	Enviar mensaje (de cliente a servidor)
	Procesar mensaje (en servidor)
	Procesar mensaje (en cliente)
	Iniciar Servidor
	Iniciar Cliente y conectar con Servidor
	Iniciar partida
	Realizar movimiento
	Simular movimiento
	Comportamiento general juego TicTacToe
	Comportamiento general juego Battleship. Parte de inicio de juego
	Comportamiento general juego Battleship. Parte juego (Usuario con turno)
	Comportamiento general juego Battleship. Parte juego (Turno del rival)
	Enviar mensaje (se servidor a cliente)
	Enviar mensaje (de cliente a servidor)
	Procesar mensaje (en servidor)
	Procesar mensaje (en cliente)
	Iniciar Servidor
	Iniciar Cliente y conectar con Servidor
	Iniciar partida
Diagramas de Actividad	Partida TicTacToe
	Juego Battleship
Diagramas de Estados	Proceso de juego
	Iniciar Partida

Documento de Diseño del Sistema

Índice DDS

Introducción

Después de haber desgranado los requisitos del proyecto y realizado el análisis, vamos a pasar a detallar el diseño del sistema.

El diseño describe cómo se desarrollará el dominio de la solución. Es decir, se describe la organización del sistema a nivel lógico y físico.

Inicialmente se explicará la arquitectura del sistema a desarrollar. Después pasaremos a diseñar el sistema propiamente dicho. En este proceso de diseño, caminaremos desde el diseño de subsistemas y componentes (definiendo cada uno de ellos), pasando por el diseño de clases, las interfaces y perfiles de usuario, y las comunicaciones. Por último, se expondrá y analizará la matriz de rastreabilidad.

Arquitectura del Sistema

La arquitectura es la organización fundamental de un sistema descrita en sus componentes, la relación entre esos componentes y el ambiente, y en los principios que guían su diseño y evolución.

A alto nivel se tendría una arquitectura lógica cliente-servidor. Sin embargo, como ya se ha comentado previamente, se ha planteado un sistema en el que no existen apenas diferencias funcionales entre cliente y servidor. Por tanto, será de más utilidad centrarse en el diseño de la arquitectura lógica específica de la aplicación.

En lo referente a la arquitectura lógica de la aplicación individual, se tiene una separación entre la lógica del juego y la parte de la presentación. Esto implica contar con diferentes capas, que son: capa de presentación, capa de lógica de negocio y capa de acceso a datos.

Inicialmente, la **capa de acceso a datos** para nuestro sistema es prácticamente inexistente, puesto que no se almacena información que deba perdurar en el tiempo, más allá de las preferencias básica de todos los juegos (por ejemplo la activación de sonido). Sin embargo, previendo una mejora o ampliación del sistema se podría especificar un componente lógico de acceso a datos para reflejar la posibilidad de almacenar partidas para ser retomadas en otro momento. De igual manera, podría hacerse constar un agente de servicios atendiendo a la posibilidad de que, en un futuro, el sistema tenga conectividad con servicios externos (jugar a través sitios web en Internet).

En cuanto a la **capa de lógica de negocio**, en ella principalmente se define la lógica de cada uno de los juegos diseñados (Componentes de Negocio: TicTacToe y Battleship). Además se tienen entidades de negocio, que son estructuras de datos para el intercambio de información entre capas.

La **capa de presentación** representa dónde se muestran los juegos y se divide en Componentes de interfaz de usuario (pantallas, escenas, etc.) y Componentes de Procesos de interfaz de usuario (principalmente encargados de controlar el flujo de operaciones con el usuario).

Además de todo esto, hay una serie de **componentes de servicios comunes**, cuyo elemento más importante es el referido a las comunicaciones.

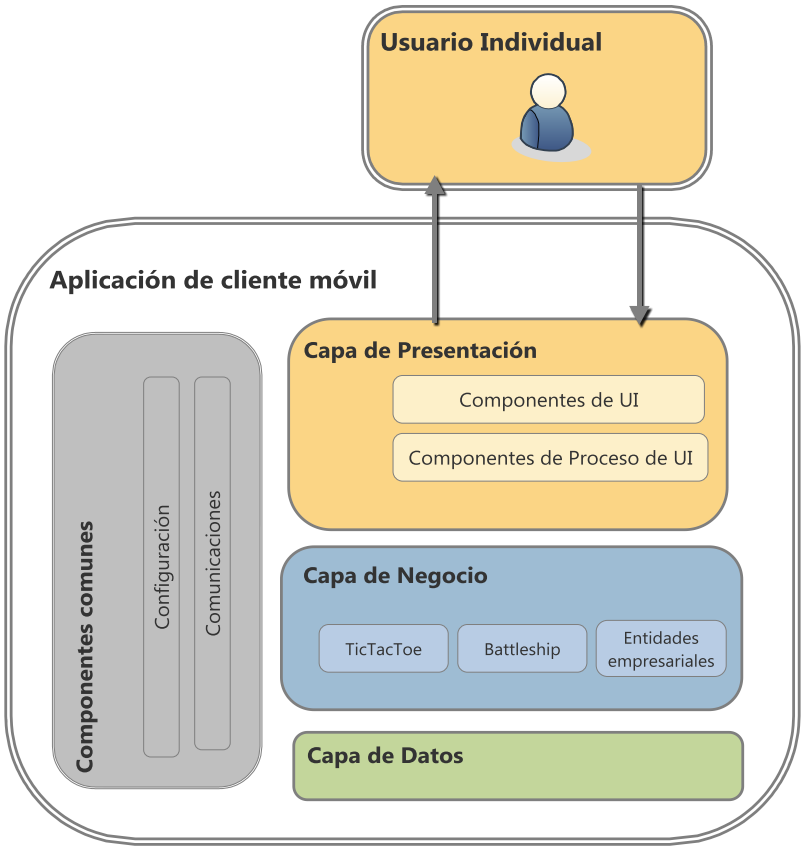


Figura 38. Arquitectura Lógica del sistema

En la figura 40 se muestra cómo quedaría la arquitectura lógica ideal incluyendo los componentes de acceso a datos antes mencionados y algún elemento más.

En cuanto a la arquitectura física del sistema (Figura 39), simplemente comentar que tendremos dos dispositivos que se pueden comunicar (enviar y recibir mensajes) mediante conexión Bluetooth o a través de una red local Wi-Fi.

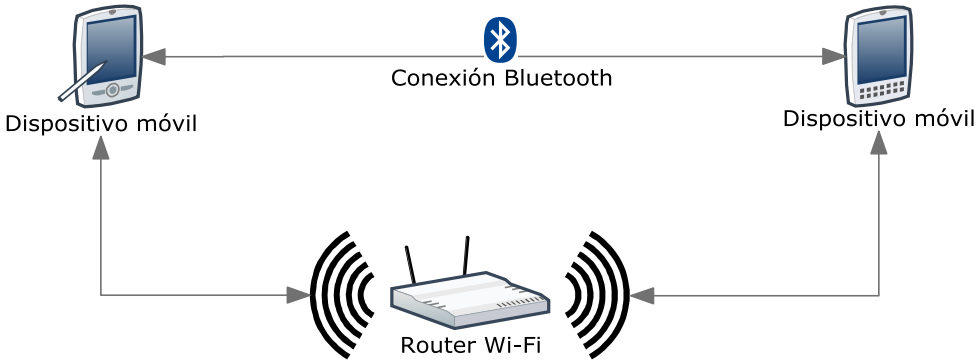


Figura 39. Arquitectura física.

Esto mismo puede verse un poco más detallado (sólo la parte interna al sistema) en el diagrama de despliegue (Figura 41).

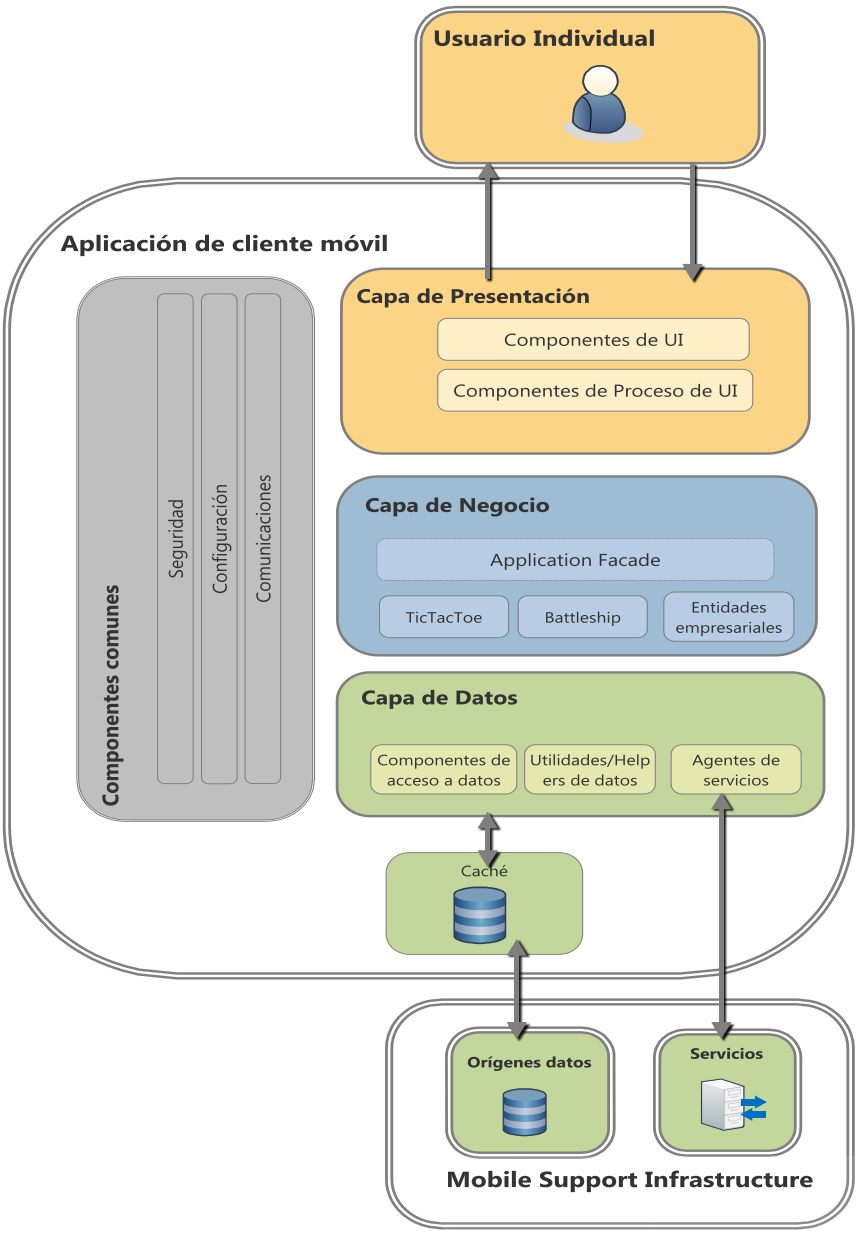


Figura 40. Arquitectura lógica ideal.

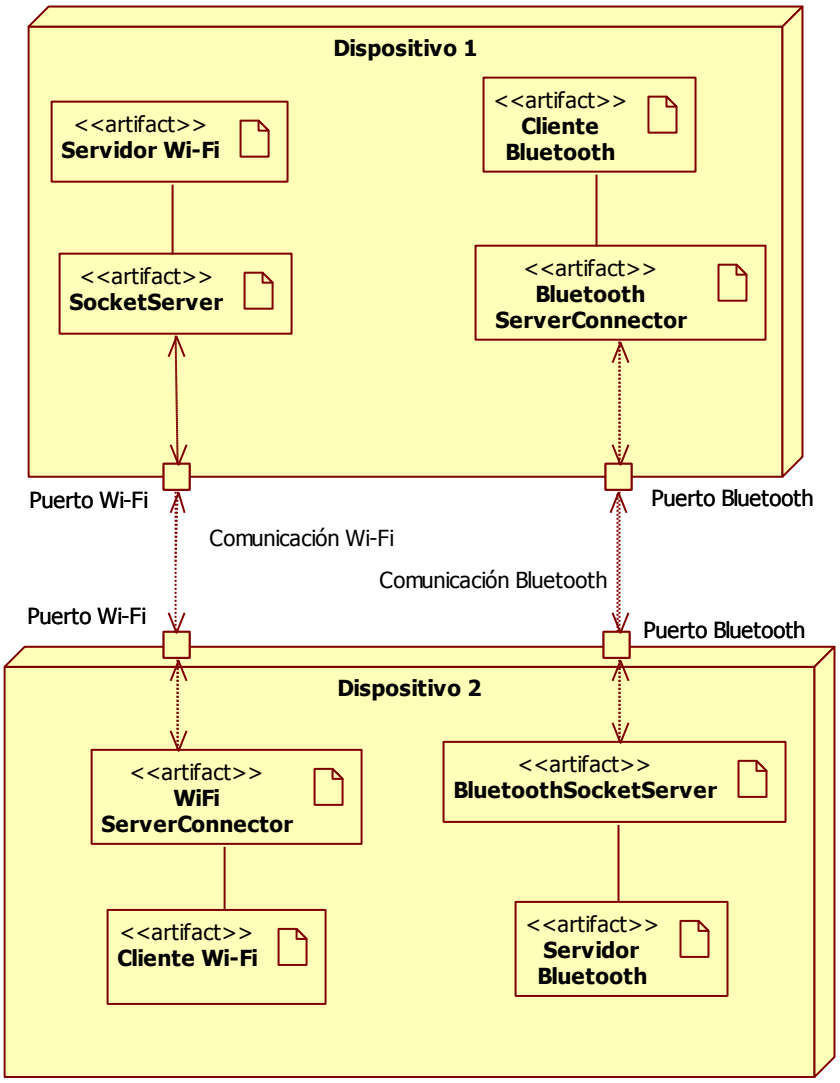


Figura 41. Diagrama de despliegue.

Diseño de Subsistemas y Componentes

División de requisitos software en conjuntos coherentes de elementos reutilizables.

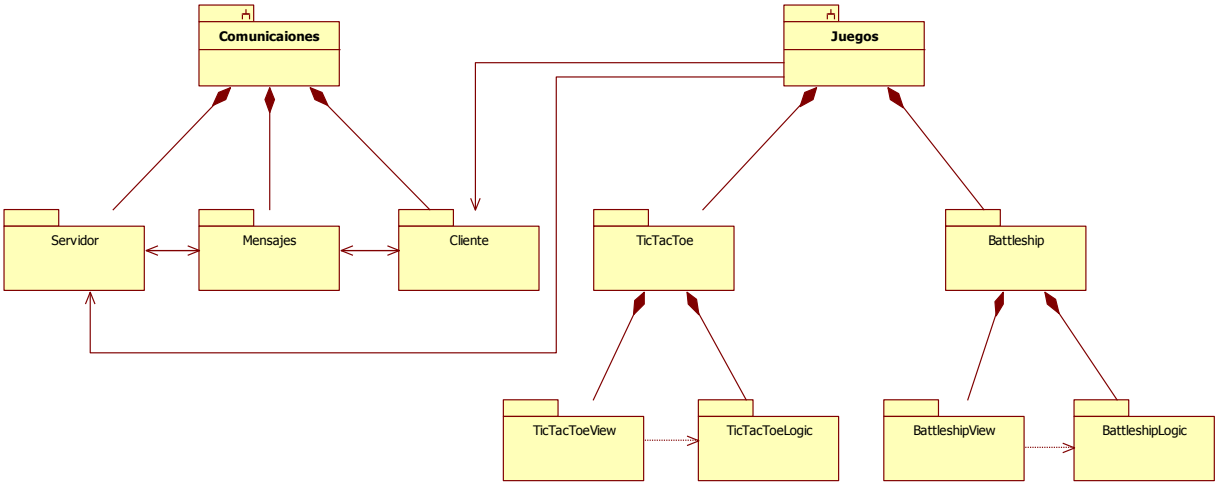


Figura 42. Subsistemas / Componentes.

Como se ha visto anteriormente, el sistema se divide en dos grandes subsistemas de los que derivan el resto de componentes.

El subsistema comunicaciones está formado por los paquetes Servidor, Cliente y Mensajes. Tanto servidor como cliente pueden enviar y recibir mensajes a/desde el otro extremo de la comunicación.

Por otra parte, se tiene el subsistema Juegos que está formado por todos los juegos que hay en nuestro sistema. Estos juegos son TicTacToe y Battleship, y ambos cuentan con su propia lógica de juego y su parte de presentación o vista.

Además, todos los juegos hacen uso de funciones de comunicación a través de clientes o servidores.

Diseño de Clases

A continuación, se explica el funcionamiento general del proyecto, resaltando las clases deducidas del mismo, para después mostrar todas ellas en el diagrama de clases general y también en diagramas de clases separados por subsistemas.

Comencemos por la parte de comunicaciones. Ya se ha comentado en varias ocasiones que la comunicación se realiza entre un cliente y un servidor, es por ello por lo que se tienen una clase para cada uno de dichos elementos. Hay que decir que se trata de clases abstractas y genéricas. Son abstractas porque se quiere tener varios tipos de servidores (y clientes) y porque todos ellos definen un comportamiento común que se puede materializar de forma distinta para cada uno. Por la misma razón las clases son también genéricas, puesto que se quiere definir el comportamiento común haciendo uso de atributos de distinto tipo. En este caso, viene en parte determinado por el motor de juegos **AndEngine** que define la funcionalidad a bajo nivel de servidor y cliente con diferentes tipos de *Servers*, *ServersConnector* y *Sockets*. De este modo será posible el uso de varios tipos de comunicaciones (y de servidores y clientes) sin que suponga un gran esfuerzo de implementación. Además, evidentemente se mejora la reutilización de componentes.

Cada servidor tiene dos subtipos (*WiFiServer* y *BluetoothServer*) que diferencian el tipo de comunicación a realizar. En caso de incluir otro tipo de comunicación "bastaría" con crear una clase que herede la funcionalidad de la clase *Server* (análogamente para el cliente). Además, estas clases son también abstractas y genéricas, en este caso son genéricas para determinar el tipo de interfaz de ejecución de acciones particular a cada juego. Esto lo veremos un poco más adelante.

WiFiServer (y las clases análogas) son abstractas en contra de lo que cabría esperar porque tal y como está diseñado el motor de juegos AndEngine no es posible recibir un mensaje de cualquier tipo y "pasarlo" a niveles lógicos superiores para que sea gestionado. En lugar de eso, a la hora de definir el propio servidor específico hay que indicar qué acciones se deben ejecutar al recibir cada tipo de mensaje. Evidentemente, esto no es algo muy práctico y es un problema de diseño que hay que solucionar. Por esta razón, estas clases son abstractas y se debe crear una clase específica para cada juego. Por ejemplo, para el juego Battleship existen las clases concretas *BattleshipWiFiServer*, *BattleshipWiFiClient*, *BattleshipBluetoothServer* y *BattleshipBluettothClient*, aunque en el diagrama no se especifican por razones de espacio.

Llegados a este punto podríamos pensar: "Si por cada juego tengo que crear las cuatro clases diferentes para realizar las conexiones, no veo dónde está la ventaja de este sistema". Sin embargo, esto no es del todo cierto, puesto que las clases abstractas nos quitan mucho trabajo definiendo el comportamiento general. Es aquí donde volvemos a la generalidad de las clases.

Como ya se ha mencionado de pasada, *Server* ejecuta *ServerActions* y *Client* ejecuta *ClientActions*. Esto no es otra cosa que la creación de unas interfaces básicas que establecen unos métodos que deben ser implementados para manejar las comunicaciones. De esta manera, para nuestro juego TicTacToe, se tienen las interfaces *IClientTicTacToeActions* (subtipo de *ClientActions*, que define el comportamiento común de conexión de cliente a servidor) y *IServerTicTacToeActions* (subtipo de *ServerActions*, que define el comportamiento común de conexión de servidor a cliente), que tienen un método por cada tipo de mensaje que debe manejar el juego. Estas interfaces serán implementadas por la

clase base del juego (en este caso *PantallaJuegoTicTacToe*) aportando la funcionalidad asociada al juego que se debe realizar al recibir el mensaje.

En definitiva, esas cuatro clases que hay que implementar por cada juego para manejar las comunicaciones, siguen siempre la misma mecánica y no requieren esfuerzo de diseño (y apenas de implementación). Una posible vía de mejora sería la creación de un framework que generalizara este comportamiento para cada conector y mensaje.

En cuanto a los mensajes que se intercambian cliente y servidor, existe una clase base abstracta (*SimpleMessage*) que es supertipo de cada uno de esos mensajes que intercambiará el sistema. Aquí también hay restricciones causadas por AndEngine, ya que hay que implementar cada mensaje dependiendo de si lo va a enviar el servidor o el cliente. Pero esto sería entrar demasiado en detalles de implementación.

Los tipos de mensajes de los que hace uso el sistema son: SetCellMessage, RestartMessage, ResadyMessage y ResultMessage. En la siguiente figura se puede ver la jerarquía de mensaje.

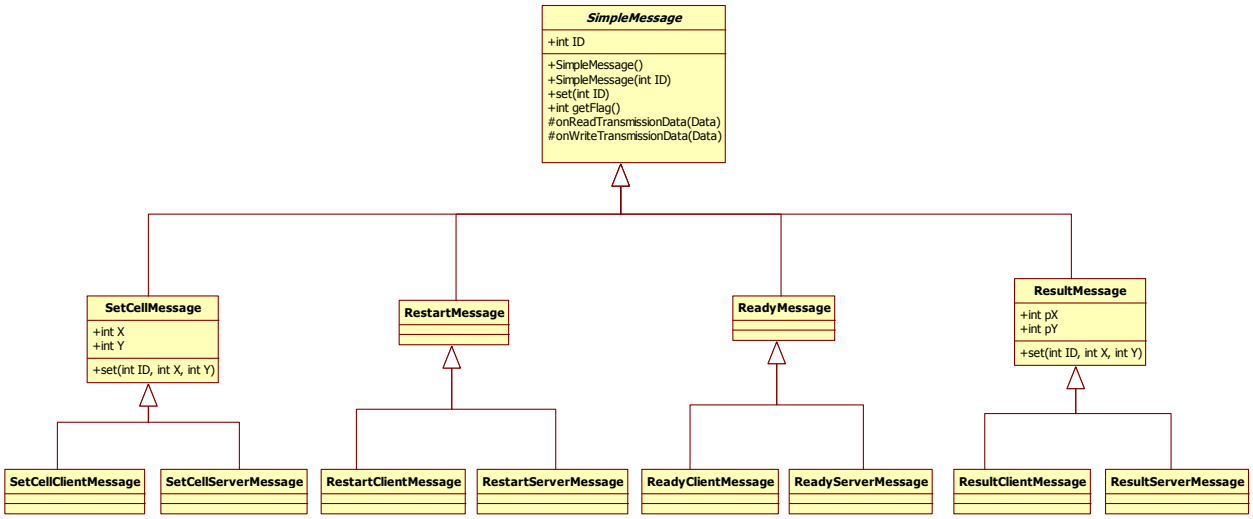


Figura 43. Jerarquía de mensajes

Por otra parte, se tiene la clase *MyCommunicationGameActivity* que es la actividad base sobre la que se ejecutará cada juego y que facilita el intercambio de mensajes. Esta clase también es genérica y abstracta, pues se debe especificar qué tipo de servidor/es y cliente/s se quieren utilizar (WiFi, Bluetooth y de qué juego). Además, presenta una seríe de métodos que facilitan la inicialización y finalización de los componentes de comunicación y el envío de mensajes. Esta clase es supertipo de cada juego específico, que no tiene por qué tener una estructura fija en su parte de presentación (vista). Por ejemplo, para este proyecto se han diseñado dos juegos, uno de ellos (TicTacToe) se compone de dos *BaseGameActivity* (la clase base para las "ventanas" en AndEngine), la primera de ellas sirve de inicio y se selección previa al juego y la segunda consta de toda la funcionalidad de comunicaciones y de la lógica de juego y la simulación de contrincante. El otro juego se ha diseñado con una sola *BaseGameActivity* (que más concretamente es del tipo *MyCommunicationActivity*) sobre la que se va cambiando la escena que se muestra. Es decir, es como si la clase base fuera un marco sobre el que van pasando distintas fotos (escenas). Las escenas que se pueden mostrar para el juego Battleship son *SplashScene* (escena de introducción para cargar los recursos del juego), *MainMenuScene* (para seleccionar el tipo de partida e iniciar

el juego), *PreGameScene* (es la escena previa al juego, que sirve para iniciar comunicaciones si fuera necesario y colocar los barcos correctamente en el tablero) y *GameScene* (que es la escena de juego propiamente dicha y contiene la lógica de juego y el simulador de jugador contrincante). La clase base del juego sólo puede mostrar una única escena al mismo tiempo.

Para finalizar se tiene la lógica específica de cada juego que, puesto que se trata en ambos casos de juegos de tablero por turnos, tienen un tablero bidimensional asociado (dos en el caso de Battleship, uno para "apuntar" los movimientos propios y otro para los movimientos del jugador rival).

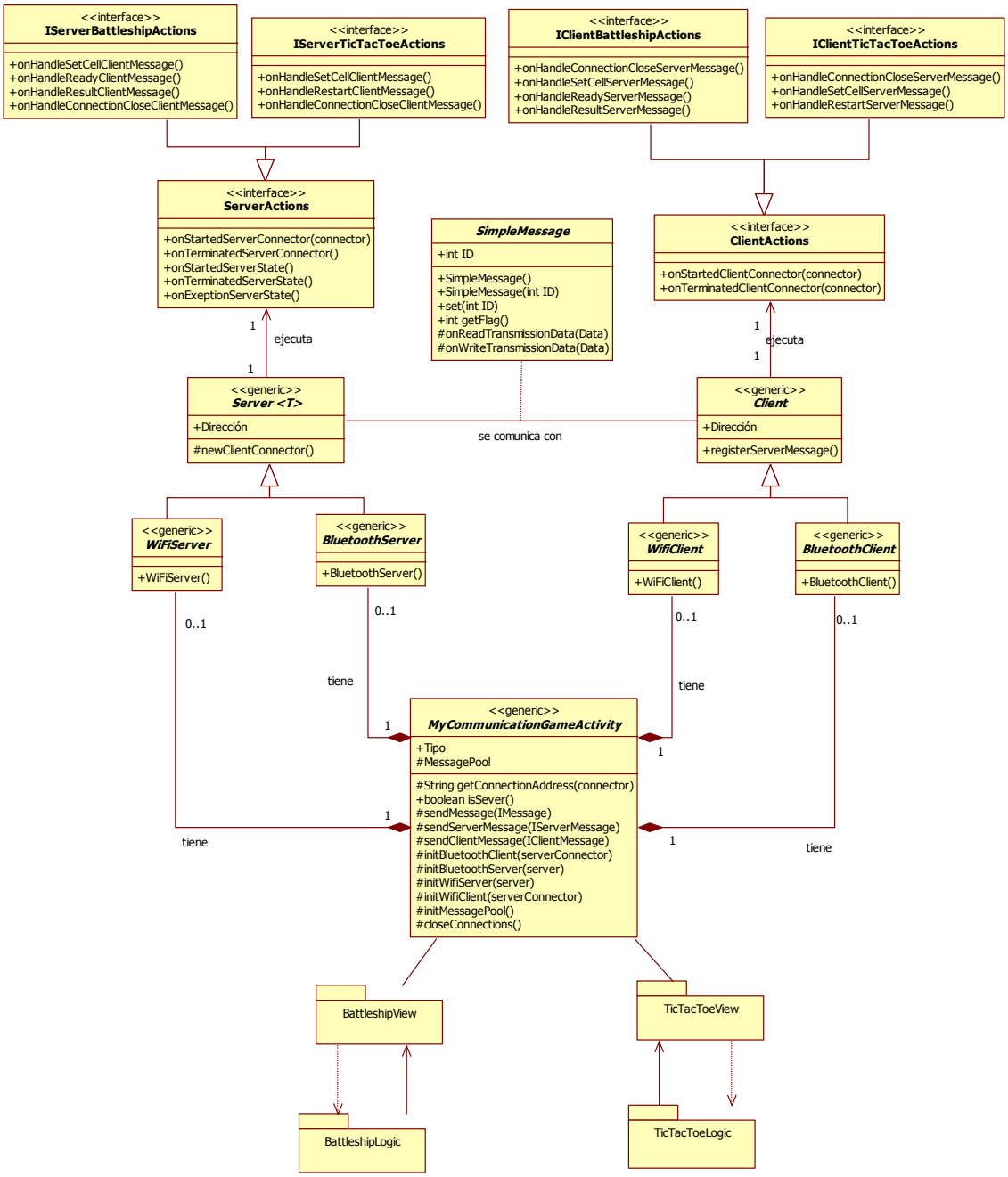


Figura 44. Diagrama de clases de Diseño Parte comunicaciones.

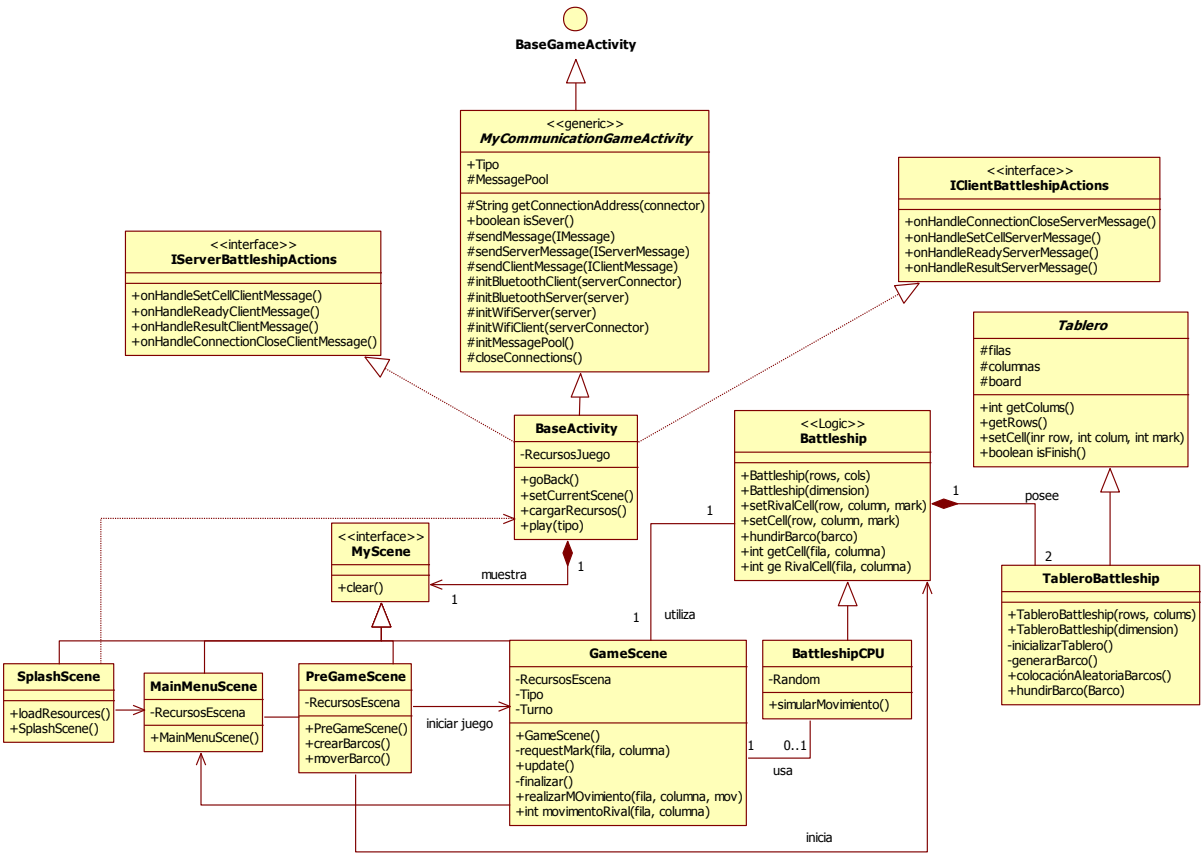


Figura 45. Diagrama de Clases de diseño. Battleship

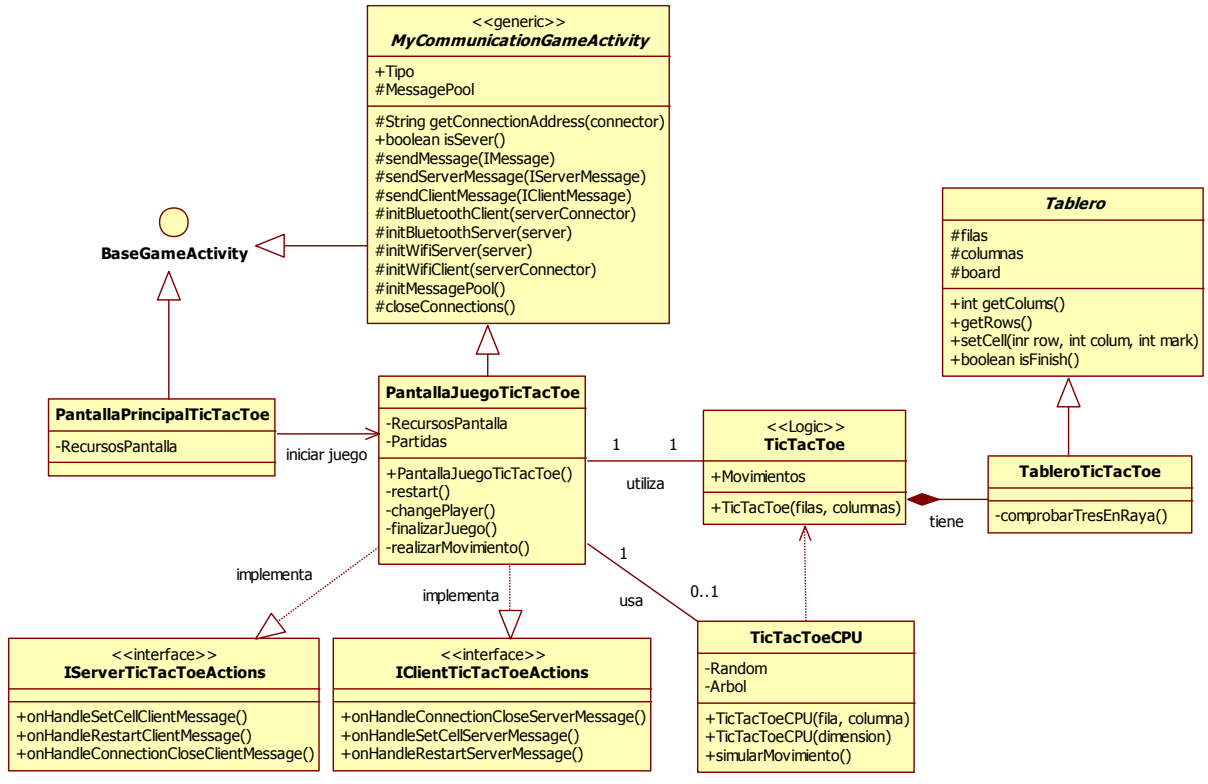


Figura 46. Diagrama de Clases de diseño. TicTacToe.

Diseño de Interfaces y Perfiles de Usuario

El diseño se realizará según estas dos características: Accesibilidad y Usabilidad. La primera característica es importante, ya que se trata de que cualquier tipo de usuario pueda acceder a la aplicación, sin que éste requiera un alto nivel informático. La segunda característica es la facilidad de uso de la aplicación. Se diseñará una interfaz lo más cómoda e intuitiva (amigable) para que el usuario no tenga dudas y puede acceder a cualquier función de la aplicación rápidamente.

Puesto que la aplicación trata de juegos simples muy reconocibles, el diseño no tratará de esconder al usuario esa simpleza.

Respecto a los perfiles de usuario, solo hay uno, el usuario normal que maneja y juega con la aplicación.

A continuación se muestra el diseño de pantalla (sólo de las pantallas más significativas) de los diferentes elementos del sistema a partir de unas imágenes de referencia:

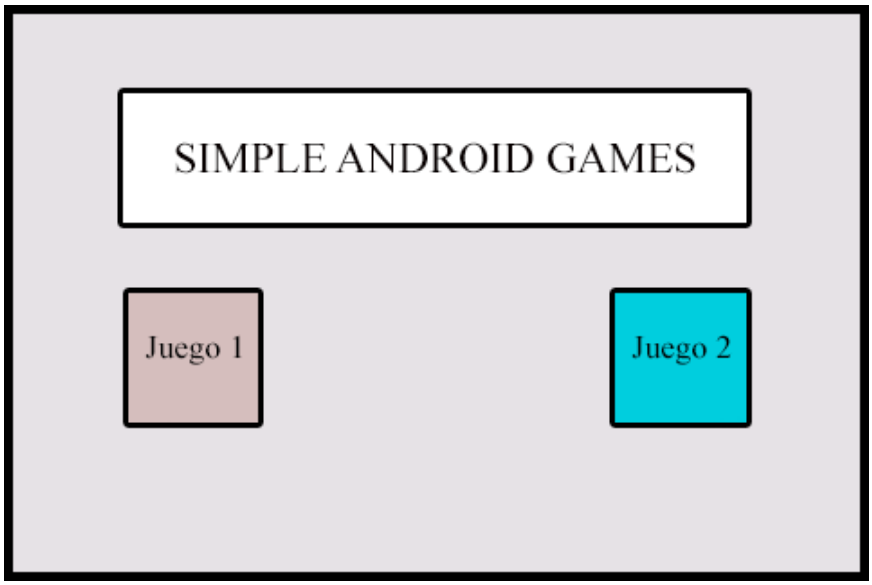


Figura 47. Diseño Pantalla principal

La aplicación nos permite ejecutar dos juegos diferentes, por esta razón debe existir una pantalla de previa de selección de juego (Figura 47).

Figura 48. Pantallas principales TicTacToe

Centrándonos en el juego TicTacToe (Tres en raya), tenemos una primera pantalla (Figura 48) donde se da al usuario la posibilidad de seleccionar el tipo de partida. Y la pantalla principal de juego (Figura 49) en la que se ejecuta el juego propiamente dicho y que cuenta con una representación del tablero de juego que tiene varias celdas o casillas sobre las que se pueden poner fichas (realizar movimientos). Cada celda debe responder al ser pulsada (entradas de usuario).

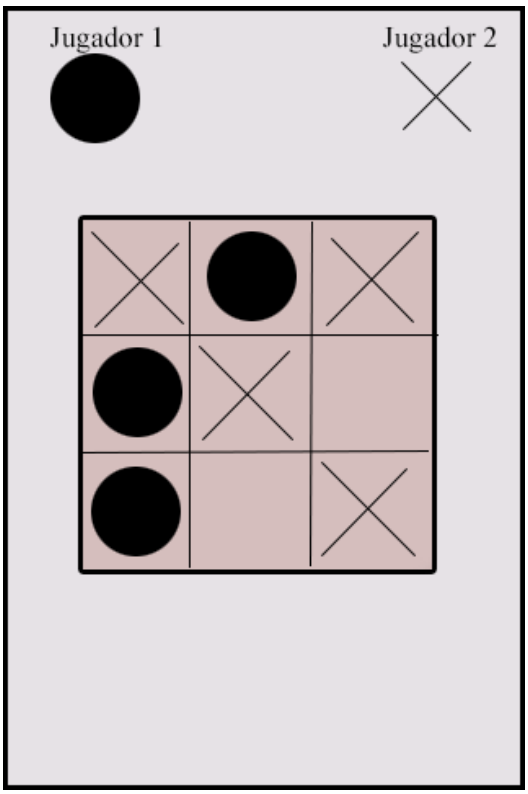


Figura 49. Pantalla de juego TicTacToe

En cuanto al juego Battleship, como se ha mencionado a lo largo del proyecto, cuenta con una "ventana" principal sobre la que se van mostrando escenas. A efectos de diseño de interfaces de usuario sólo nos interesa ver un boceto de lo que las diferentes escenas deben mostrar.

Figura 50. Diseño MainMenuScene.

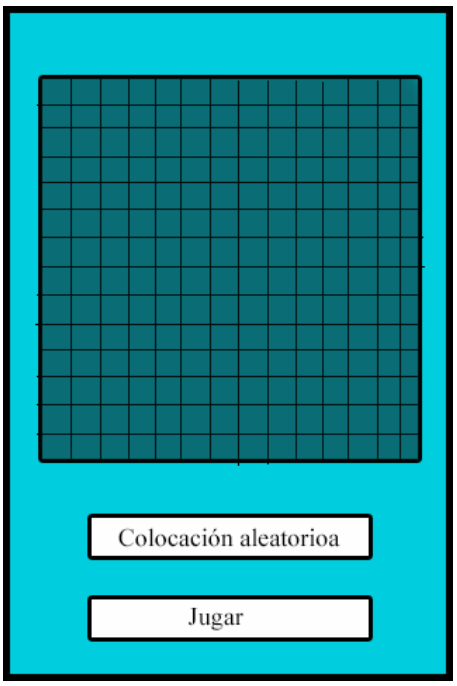


Figura 51. Diseño PreGameScene.

La figura 50 representa la escena de menú del juego Battleship en la que se puede seleccionar el tipo de partida que se puede jugar. Para esto, la escena presenta tres botones, además cuenta con un botón más para salir.

La siguiente imagen (figura 51) muestra el diseño de la escena previa al juego en la que se selecciona la colocación de los barcos del usuario. En esta escena hay tres elementos principales, dos botones y el tablero en el que colocar los barcos.

Finalmente, la escena de juego (Figura 52) tiene un primer tablero en la parte superior izquierda que muestra los barcos del propio usuario y los movimientos del contrincante; un segundo tablero en el que realizar los movimientos del usuario (barcos del rival); y un botón para confirmar los movimientos del usuario.

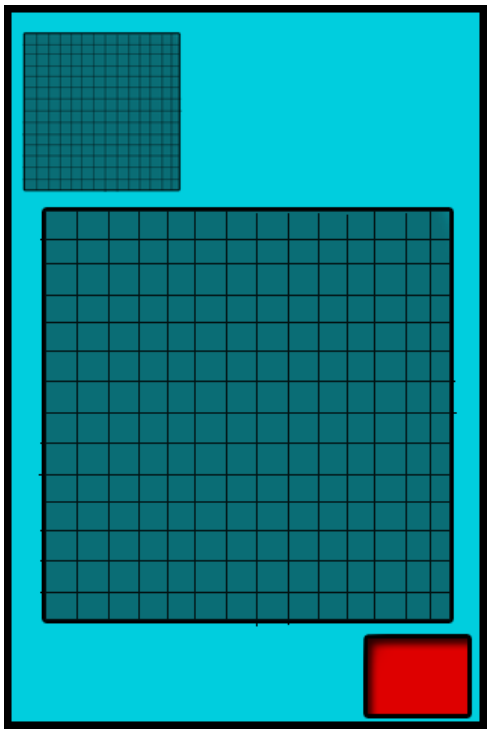


Figura 52. Interfaz usuario GameScene.

No se muestra el diseño de la *SplashScene* puesto que consiste tan sólo en una pantalla que se muestra apenas unos segundos para cargar los recursos.

Diseño de comunicaciones

Una parte importante del proyecto es la comunicación entre los diferentes dispositivos que intervienen en el proceso fundamental de intercambio de mensajes. Por tanto, habría que definir un protocolo de comunicaciones entre cliente y servidor.

"Un protocolo de comunicaciones es un conjunto de reglas y normas que permiten que dos o más entidades de un sistema de comunicación se comuniquen entre ellos para transmitir información por medio de cualquier tipo de variación de una magnitud física. Se trata de las reglas o el estándar que define la sintaxis, semántica y sincronización de la comunicación, así como posibles métodos de recuperación de errores. Los protocolos pueden ser implementados por hardware, software, o una combinación de ambos." (Wikipedia)

Pues bien, el protocolo de comunicaciones del sistema será muy simple. A bajo nivel es el motor AndEngine (su extensión multiplayer concretamente) el que define cómo se intercambian los mensajes y se realizan las conexiones (aquí se determinan cuestiones como *Handshaking*, detección de la conexión física, formateo de mensajes, manejo de mensajes corruptos, etc.). A este respecto, dicho motor de juegos es un tanto cerrado, así que el protocolo de comunicaciones se adaptará a ello. En cualquier caso, no profundizaremos a nivel de intercambio de tramas de bits, datagramas o paquetes.

Básicamente, el protocolo de comunicaciones establece que se pueden intercambiar mensajes (de forma síncrona) siempre y cuando exista una conexión previa entre cliente y servidor, es decir, primeramente se establecerá la conexión (que permanecerá abierta) y sobre esa conexión se intercambiarán mensajes. La conexión finalizará a petición de cualquiera de los elementos de la comunicación.

Se podría haber iniciado la conexión por cada mensaje que se quiera enviar, pero eso sería una pérdida de recursos y de tiempo. Supongamos que queremos desarrollar un juego que requiera de más interacción con el usuario que TicTacToe o Battleship, como por ejemplo un juego de Air Hockey. Para dicho juego el intercambio de mensajes es mucho mayor, cada movimiento del usuario o de la ficha implica intercambio de mensajes, así, si se tuviera que abrir y cerrar la conexión con cada mensaje, en el juego se podría producir un retardo (o *lag*) que se iría acumulando, además habría un descenso del rendimiento del juego.

Otros aspectos que el protocolo define son el reenvío de mensajes, la confirmación de llegada y la recuperación ante errores. Sobre esto, no hay una confirmación de llegada propiamente dicha, pero en el emisor se informa (mediante manejo de excepciones) de si ha ocurrido algún error durante el envío del mensaje (conexión cerrada, conexión del tipo incorrecto, error de entrada/salir, etc). A partir de ahí, es responsabilidad de cada juego particular decidir si se reenvían los mensajes perdidos o si no es necesario. Por ejemplo, en juegos como TicTacToe o Battleship es necesario que cada mensaje llegue a su destino, pero en juegos como Air Hockey es preferible omitir un mensaje erróneo y enviar otro posterior que esté pendiente, teniendo en cuenta siempre que si se producen muchos errores de envío podría ser necesario finalizar el juego por incongruencias entre el cliente y el servidor. De hecho, en la situación de Air Hockey seguramente se envíen mensajes con escasas milésimas de segundos de diferencia, si un mensaje se pierde y se quiere reenvía llegaría

desordenado respecto a la secuencia normal y provocaría comportamientos extraños en la actualización de elementos del dispositivo receptor.

En cuanto a otras normas o reglas, el protocolo no establece ningún tipo de cifrado de los mensajes ya que se considera innecesario teniendo en cuenta la esencia del proyecto (no hay necesidad de cifrar comunicaciones en las que se intercambian posiciones de un tablero de Tres en Raya).

A continuación se muestran, mediante figuras, cómo se produce el intercambio de mensajes entre cliente y servidor específicamente en cada juego.

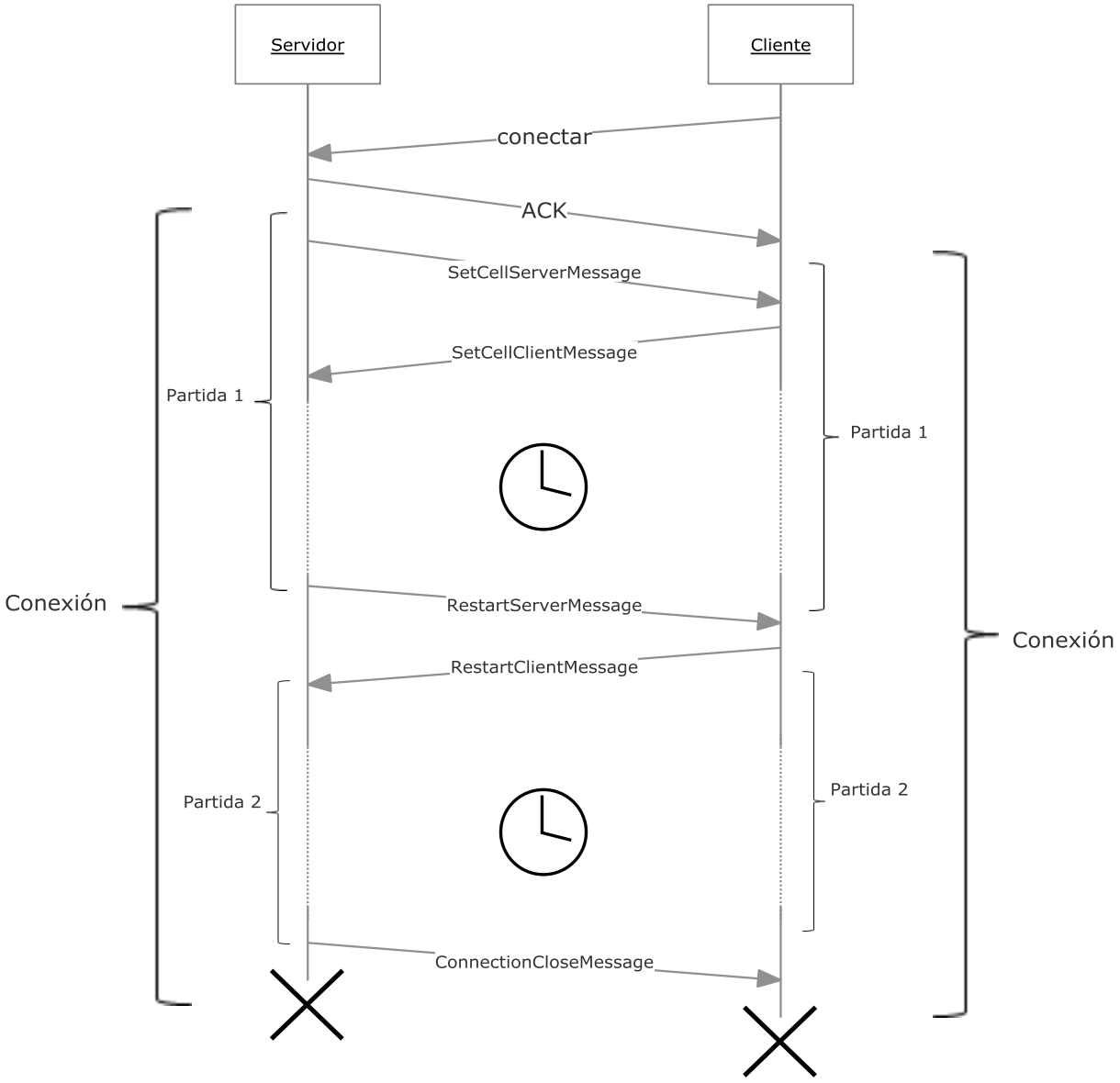


Figura 53. Comunicaciones TicTacToe.

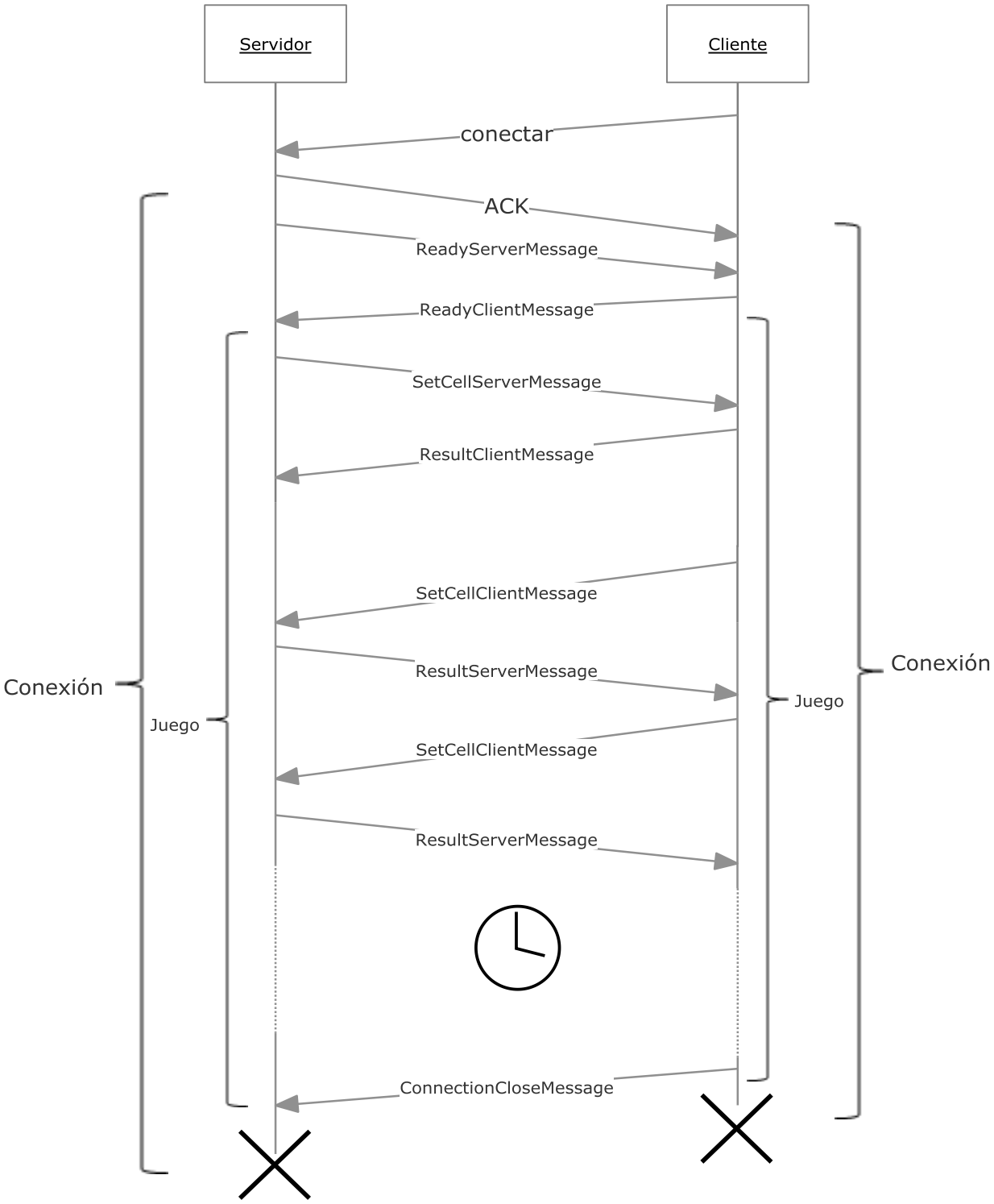


Figura 54. Comunicaciones Battleship.

La figura 53 representa las comunicaciones que se producen en el juego TicTacToe. Inicialmente el cliente se conecta al servidor (que previamente ya estaba iniciado). El servidor acepta la comunicación y se lo notifica al cliente (si la conexión se rechaza, NAK, también se comunica al cliente y no se inicia el juego). A partir de entonces la conexión permanece abierta hasta que uno de los extremos de la comunicación la finalice.

Con la comunicación ya abierta, se inicia el juego y el servidor (que es quien inicialmente tiene el turno de juego) envía un mensaje al cliente indicando la casilla que ha seleccionado. El cliente procesa el mensaje y envía un mensaje semejante cuando seleccione otra casilla. Este proceso se repite hasta que la partida finalice, momento en el que cual se ofrece a los usuarios la posibilidad de iniciar otra partida. En caso de que los usuarios decidan jugar otra partida se envían mensajes de tipo *Restart* entre ellos y se inicia la nueva partida siguiendo el mismo proceso descrito anteriormente.

La conexión finaliza en ambos extremos cuando alguno de los dos jugadores decida no continuar y cierre el juego, para ello se envía un mensaje de tipo *ConnectionClose* desde el dispositivo en el que se ha cerrado el juego. Al recibir este mensaje el otro extremo de la comunicación finalizará automáticamente el juego (y la conexión).

En cuando a las comunicaciones del juego Battleship (Figura 54), comienza realizando de igual manera la conexión entre cliente y servidor. Una vez conectados, los usuarios pueden colocar los barcos en el tablero y cuando lo hayan hecho se enviará el mensaje de tipo *Ready* al otro dispositivo. Cuando ambos jugadores estén preparados (barcos colocados correctamente) comienza la partida.

Nuevamente es el servidor quién comienza la partida. El proceso de movimiento se refleja en el intercambio de mensajes *SetCell* y *Result*. Cuando el usuario del dispositivo servidor selecciona una casilla se envía un mensaje al cliente, que responde indicando qué hay en la casilla en cuestión. Lo mismo desde el cliente al servidor. Además, si el resultado del movimiento es distinto de *AGUA*, es decir hay un barco (o parte de él) en la casilla (celda) seleccionada, el jugador (cliente en este caso) seguirá en posesión del turno, enviando por tanto mensaje *SetCell* y recibiendo *Result*.

El proceso se repite hasta que finalice la partida o alguno de los componentes finalice la conexión (partida) de igual manera que sucedía en el juego TicTacToe.

Documento de Implementación del Sistema

Índice DIS

Introducción

Llegados a este punto se procede a comentar las decisiones y medidas tomadas a la hora de implementar el sistema anteriormente analizado y diseñado, así como las peculiaridades y problemas surgidos en él.

Se ha hablado a lo largo de todo el proyecto (y especialmente en el proceso de diseño) de dos partes diferenciadas (la referida a las Comunicaciones y los Juegos) que conforman el conjunto del proyecto. También se ha comentado que la comunicación se realiza mediante un sistema cliente-servidor de intercambio de mensajes. A alto nivel (en lo tocante a los juegos) esto en cierta medida irrelevante, por lo que se debía implementar el sistema de manera que los mensajes se enviaran independientemente del tipo que fueran. Profundizaremos un poco más en ello en la parte de implementación de comunicaciones.

La implementación de la aplicación se ha llevado a cabo mediante el lenguaje de programación orientado a objetos Java aplicado a Android, que es el lenguaje propio con el que se desarrollan la gran mayoría de aplicaciones para este sistema operativo, conjugándolo con archivos XML que conforman la parte visual de la aplicación (en realidad, en la aplicación tan sólo la pantalla de preferencias se apoya en archivos XML).

Además, para la implementación de los componentes software del proyecto, se hará uso de **AndEngine,** que es un motor Java de código abierto para desarrollar juegos en 2D mediante OpenGL para Android y con terminología propia. AndEngine además cuenta con una serie de extensiones que permiten separar funcionalidades. Para este proyecto la extensión que nos interesa es la llamada *Multiplayer*, que como su nombre indica sirve para dotar a los juegos desarrollados usando AndEngine de funcionalidad multijugador (Wi-Fi y Bluetooth). Básicamente, consiste en una serie de bibliotecas de código que nos aportan la funcionalidad. Estas bibliotecas son completamente libres y se puede obtener el código completo sin ningún problema, pudiendo además realizar modificaciones si fuera necesario.

Implementación de juegos

Cada juego tiene su propia lógica específica y su parte de presentación. La lógica de cada juego se encuentra especificada en clases independientes y separadas de la vista. La vista se actualizar con los cambios que se genera en la lógica de juego a partir de las entradas del usuario (*captados* en la parte de la vista). Además, cumpliendo con los requisitos establecidos al comienzo del proyecto, se han creado *simuladores* para generar movimientos y poder jugar contra la *máquina*.

La lógica de juego no mece mucho comentario, puesto que los juegos desarrollados son conocidos por todo el mundo, TicTacToe (Tres en Raya), que consiste en poner tres fichas iguales en línea (o diagonal) y Battleship (Hundir la flota), que consiste en *destruir* (hundir) todos los barcos del contrincante antes de que él hunda los nuestros. Simplemente indicar que en el juego Battleship se establece que cada jugador tenga seis barcos: dos de longitud 2, dos de longitud 3, uno de longitud 4 y otro de longitud 5.

La simulación se ha implementado de forma diferente para cada juego. Para el juego TicTacToe se crea una clase que "simplemente" genera movimientos a partir de la lógica de juego. Para esta simulación/generación de movimientos se aplican técnicas de lógica de teoría de juegos, estos es, haciendo uso de un árbol se van generando todos los posibles movimientos y hacia donde conduce cada uno. De esta forma se le asigna mayor peso a los movimientos más *prometedores* y se obtiene finalmente el mejor movimiento posible. El algoritmo genera movimientos tan *fiables*, que es imposible ganar a la máquina. Como curiosidad, cuando el inicio de juego le correspondía a la máquina siempre se iniciaba de la misma manera, por esta razón se ha introducido un cierto grado de incertidumbre haciendo que el primer movimiento de la máquina sea aleatorio (sólo cuando la CPU inicia el juego).

En cuanto a la simulación de movimientos para el juego Battleship, se ha optado por que la clase de simulación sea un subtipo de la lógica de juego. De esta menara el *simulador* tendría toda la funcionalidad de la lógica más la funcionalidad para generar movimientos. Aquí lo que se simula es un jugador rival completo, que tiene dos tableros (uno en el que coloca sus barcos y otro para *apuntar* los movimientos que realizar *contra* el rival). En esta ocasión la implementación de la clase de simulación es menos ambiciosa y funciona de manera más aleatoria. Así, inicialmente se generarán movimientos sobre casillas completamente aleatorias hasta encontrar algún barco. Cuando algún barco rival esté *tocado* se generarán movimientos buscando el resto de partes del barco en posiciones adyacente a la encontrada. Se ha tenido cuidado en evitar generar movimientos en los que seguro no habrá barcos rivales, esto es, cuando hay una casilla rodeada por *agua* (puesto que no hay barcos de longitud 1).

La parte de la vista de cada juego se implementa siguiendo lo establecido por el diseño; esto es, para TicTacToe se crean dos clases que representan *ventanas* diferentes y para Battleship se implementa una *ventana* básica sobre la que se van mostrando distintas escenas posibles. Evidentemente, en ambas situaciones se hereda de clases propias del motor de juegos AndEngine, siguiendo las pautas establecidas por él. Una de estas pautas se refiere a la carga previa de recursos (tanto imágenes como sonidos) antes de ser utilizados, es decir todos los recursos deben ser cargados al inicio del juego, o en su defecto al inicio de cada escena de pantalla. Por esta razón, se incluye la escena *Splash* en el juego Battleship, que no es otra cosa que una animación muy simple que se ejecuta en el momento de iniciar el juego para realizar la carga de todos los recursos que se van a usar a lo largo del juego (y de las partidas) y que finaliza cuando esta carga se completado. Además, se ha prestado especial atención en finalizar todos los recursos y cerrar todas las comunicaciones que puedan haber utilizado, con el fin de cuidar el rendimiento de la aplicación.

Por otra parte, teníamos un requisito no funcional referente a la **internacionalización.** Respecto a esto, aunque la aplicación no está disponible en varios idiomas, la estructura de la misma se ha diseñado como marcan las buenas prácticas; esto es, todo el texto que se muestra en la aplicación está almacenado en un archivo .xml independiente, por lo que para tener más idiomas disponibles bastaría con crear un nuevo archivo .xml con la traducción del original y cambiar entre uno u otro en función del idioma que tenga fijado el dispositivo.

Implementación de comunicaciones

En cuanto a la parte de comunicaciones, realmente no es necesario hablar mucho de ella a nivel de implementación, puesto que los conceptos generales han sido explicados en la parte de diseño (comunicaciones y diagrama de clases). En esencia, se basa en adaptar la extensión multiplayer de AndEngine a nuestras necesidades para la aplicación. Teniendo en cuenta que es la propia extensión la que se ocupa de detalles de implementación a bajo nivel, no hay mucha necesidad de profundizar más el desarrollo.

No obstante, a alto nivel cliente y servidor deben ser prácticamente iguales. A la hora de implementar un juego es mucho más cómodo decir que se envíe un mensaje y el proceso de envío sea siempre igual (*sendMessage(mensaje)*), que tener que estar discriminando en función del tipo de conector (cliente o servidor) y del tipo de conexión (Wi-Fi o Bluetooth). Y esto es algo que el motor de juegos AndEnginie no hace por sí mismo, es decir, AndEngine no permite enviar el mismo mensaje desde cliente y servidor, sino que cada uno debe implementar una interfaz diferente (*IClientMessage* e *IServerMessage*). Se ha tratado de solventar esta deficiencia, aplicando la jerarquía de clases vista en la parte de diseño, adaptándola a los requisitos de AndEngine a nivel funcional.

Problemas surgidos en la fase de implementación

Los principales problemas encontrados a la hora de implementar las distintas partes del proyecto han sido debidos, en su mayoría, a problemas de inexperiencia; y es que antes de sumergirme en este proyecto no tenía ningún tipo de conocimiento previo en desarrollo de juegos para cualquier tipo de plataforma, por lo que fue necesario un esfuerzo previo de formación, que afortunadamente ya había realizado antes de definir las bases concretas de lo que quería que fuera este proyecto. Evidentemente, esa formación también ha estado presente a lo largo de todo el proceso de desarrollo, de hecho la mejor formación es la práctica.

Por otra parte, aunque ya conocía un poco cómo funciona el desarrollo de aplicaciones para Android, aplicar estos conocimientos al desarrollo de juegos para esta plataforma no es inmediato y también requiere de cierto esfuerzo de documentación y *ensayo/error*. A esto hay que unir el uso del motor de juegos AndEngine, que aunque ciertamente ahorra mucho trabajo inicial, define un comportamiento un tanto cerrado y hay que ceñirse a lo que él establece. Tampoco ayuda la escasa documentación encontrada acerca de él más allá del foro *oficial* en el que distintos usuarios intercambian opiniones y consejos (siempre en inglés).

Afortunadamente antes de iniciar el proyecto también realicé cierto estudio del motor de juegos AndEngine, pues de otro modo habría sido inviable cumplir los plazos inicialmente establecidos.

En cuanto a problemas concretos, más allá de los habituales de tener que *pegarse* con diferentes tipos de imágenes, escenas, cámaras, animaciones, gestión eficiente de recursos, excepciones impensables y otro tipo de cosas variopintas; destacaría problemas debidos a incorrecciones en la biblioteca de clases del motor de juegos AndEngine.

Estos problemas causaron significativos dolores de cabeza por tener que revisar y modificar bastante código que en principio se daba por fiable. Recuerdo con especial *nostalgia* días enteros perdidos intentando solucionar un problema que causaba que los elementos mostrados en pantalla presentaran bordes cuadrados entorno a ellos aunque fuera elementos, por ejemplo, circulares. No era muy agradable hacer una aplicación de juegos en la se mostrarán los elementos de forma incorrecta. También pasaron varios días hasta que pude solucionar errores de conectividad (especialmente Bluetooth) entre cliente y servidor. Básicamente el sistema informaba de conexión correcta con el servidor aunque se hubiera solicitado conectarse con un dispositivo apagado.

Por fortuna, estos errores pudieron ser solucionados tras realizar muchas consultas en Internet y revisar el código fuente de AndEngine; aunque, como es lógico, hicieron que la fase de implementación se alargara más de lo que inicialmente se había estimado.

Documento de Pruebas del Sistema

Índice DPS

Introducción

Este documento se ha realizado con un objetivo claro, la validación o comprobación de la aplicación. En él se documentan las distintas pruebas realizadas con ese fin.

Para empezar, se han realizado pruebas de cada uno de los requisitos del sistema, detallando los pasos realizados de la prueba que se ha hecho a ese requisito específico y el resultado de dicha prueba. Con estas pruebas se comprueba si la aplicación realiza correctamente lo que se estipuló en un principio, obteniendo el resultado esperado.

Por último, se han realizado unas pruebas de esfuerzo o estrés utilizando programas externos pensados para tal fin.

Otra cosa que conviene comprobar cuando hablamos de juegos es la cantidad de frames por segundo (FPS) a la que *se mueve* el juego. Se ha realizado esta comprobación en varias ocasiones a través del depurador y de las indicaciones que hace AndEngine para este tipo de chequeos y en ninguno de los casos el número de FPS era inferior a 60, lo cual es una buena señal y algo que era de esperar puesto que los juegos desarrollados no requieren de una gran carga gráfica.

Pruebas de cumplimiento de requisitos

Es necesario realizar diversas pruebas para garantizar y asegurar el buen funcionamiento de todos los requisitos establecidos en el proyecto. Por cada requisito se diseña y ejecuta al menos una prueba que garantice su cumplimiento, pudiendo ser necesarias más de una prueba para garantizar totalmente el correcto desempeño del requisito.

Requisito	UC- 1 Enviar mensaje
Descripción prueba	Se deben realizar pruebas que garanticen y verifiquen el envío de mensajes desde el cliente y el servidor
Resultado esperado	Mensaje enviado por el emisor y recibido por el receptor.

Se debe comprobar que los mensajes se envían correctamente desde el servidor y desde el cliente. Además, se debe verificar que el los mensajes se envían a través de Wi-Fi y de Bluetooth.

Prueba	PCR-1A Envío de mensajes desde servidor a cliente por Bluetooth
Requisito asociado	UC- 1 Enviar mensaje.
Prerrequisito	Ninguno.
Procedimiento	1. Entrar en el juego TicTacToe, seleccionar partida multijugador por Bluetooth y actuar como Servidor. 2. Iniciar el juego TicTacToe en otro dispositivo, seleccionar partida multijugador por Bluetooth, actuar como cliente y conectarse al servidor. 3. Desde el dispositivo servidor, seleccionar una casilla del tablero. 4. Comprobar en el dispositivo cliente que la misma casilla seleccionada en el servidor se ha *marcado* en el cliente.
Resultado	Mensaje enviado por Bluetooth desde el servidor y recibido en el cliente.
Comentario	Con esta prueba también se garantiza el funcionamiento de los requisitos UC-2 Enviar mensaje Bluetooth y UC-4 Procesar Mensaje.

* Esta prueba es análoga a la que comprobaría el envío de mensajes desde cliente a servidor por Bluetooth. La prueba se ha realizado obteniendo los resultados esperados.

Prueba	PCR-1B Envío de mensajes desde cliente a servidor por Wi-Fi.
Requisitos asociados	UC- 1 Enviar mensaje.
Prerrequisito	Ninguno.
Procedimiento	1. Entrar en el juego Battleship, seleccionar partida Wi-Fi y actuar como Servidor. 2. Iniciar el juego Battleship en otro dispositivo, seleccionar partida Wi-Fi, actuar como cliente y conectarse al servidor. 3. Pulsar jugar en ambos dispositivos. 4. Realizar un movimiento desde el servidor. 5. Realizar un movimiento desde el cliente. 6. Comprobar en el dispositivo servidor que el movimiento realizado en el cliente se ha *marcado* en el servidor (tablero superior de Battleship).
Resultado	Mensaje enviado por Wi-Fi desde el cliente y recibido en el servidor.
Comentario	Con esta prueba también se garantiza el funcionamiento de los requisitos UC-3 Enviar mensaje Wi-Fi y UC-4 Procesar Mensaje.

* Esta prueba es análoga a la que comprobaría el envío de mensajes desde servidor a cliente por Wi-Fi. La prueba se ha realizado obteniendo los resultados esperados.

Requisito	UC- 5 Iniciar Servidor
Descripción prueba	Se deben realizar pruebas que garanticen y verifiquen el arranque del servidor.
Resultado esperado	Servidor iniciado.

Se debe comprobar que se inician tanto servidores Wi-Fi y como Bluetooth.

Prueba	PCR-2A Iniciación de Servidor Wi-Fi.
Requisito asociado	UC- 5 Iniciar Servidor
Prerrequisito	Ninguno.
Procedimiento	1. Entrar en el juego Battleship, seleccionar partida Wi-Fi (activar Wi-Fi si estuviera desconectado y esperar a que haya señal válida) y actuar como Servidor. 2. Comprobar que se muestra por pantalla que el servidor se ha iniciado y su dirección IP. 3. Iniciar el juego Battleship en otro dispositivo, seleccionar partida Wi-Fi, actuar como cliente. 4. Comprobar que se muestra por pantalla que se ha iniciado el cliente y se solicita la dirección IP del servidor. 5. En el cliente, introducir la dirección IP del servidor y conectar. 6. Comprobar tanto en cliente como en servidor que se ha realizado la conexión y se puede realizar una partida.
Resultado	Servidor iniciado, cliente iniciado y cliente conectado con servidor.
Comentario	Esta prueba depende la calidad de la señal entre los dispositivos y el router Wi-Fi que crea la red local inalámbrica. Con esta prueba también se garantiza el funcionamiento de los requisitos UC-6 iniciar cliente y UC-7 Conectar con servidor.

Prueba	PCR-2B Iniciación de Servidor Bluetooth.
Requisito asociado	UC- 5 Iniciar Servidor
Prerrequisito	Ninguno.
Procedimiento	1. Entrar en el juego TicTacToe, seleccionar partida multijugador por Bluetooth (activar Bluetooth si estuviera desconectado) y actuar como Servidor. 2. Comprobar que se muestra por pantalla que el servidor se ha iniciado y su dirección MAC. 3. Iniciar el juego TicTacToe en otro dispositivo, seleccionar partida multijugador por Bluetooth y actuar como cliente. 4. Comprobar que se muestra por pantalla que se ha iniciado el cliente y se solicita conectar con servidor. 5. En el cliente, seleccionar el servidor y conectar. 6. Comprobar tanto en cliente como en servidor que se ha realizado la conexión y se puede realizar una partida.
Resultado	Servidor iniciado, cliente iniciado y cliente conectado con servidor.
Comentario	Con esta prueba también se garantiza el funcionamiento de los requisitos UC-6 iniciar cliente y UC-7 Conectar con servidor.

Requisito	UC-8 Iniciar partida
Descripción prueba	Se deben realizar pruebas que garanticen y verifiquen el inicio de una partida.
Resultado esperado	Partida iniciada.

Existen varios tipos de partidas para cada juego. Con pruebas anteriores se ha verificado indirectamente el inicio de partidas multijugador por Wi-Fi y por Bluetooth, así que falta por comprobar que funciona en partidas de un jugador y de multijugador sin conexión.

Prueba	PCR-3A Inicio de partida Multijugador sin conexión TicTacToe.
Requisito asociado	UC- 8 Iniciar partida
Prerrequisito	Ninguno.
Procedimiento	1. Entrar en el juego TicTacToe, seleccionar partida multijugador sin conexión 2. Comprobar que se muestran correctamente por pantalla los recursos de la escena. 3. Comprobar que se puede empezar a jugar (pulsar una casilla del tablero).
Resultado	Partida iniciada.
Comentario	Con esta prueba también se garantiza el funcionamiento del requisito UC-9 Realizar movimiento.

Prueba	PCR-3B Inicio de partida Un jugador Battleship.
Requisito asociado	UC- 8 Iniciar partida
Prerrequisito	Ninguno.
Procedimiento	1. Entrar en el juego Battleship, seleccionar partida de un jugador. 2. Comprobar que se muestran correctamente por pantalla los recursos de la escena. 3. Comprobar que se puede realizar la colocación de barcos en el tablero y jugar (pulsar botón jugar).
Resultado	Partida iniciada.
Comentario	Con esta prueba también se garantiza el funcionamiento del requisito UC-9 Realizar movimiento.

Requisito	UC-11 Simular movimiento
Descripción prueba	Se deben realizar pruebas que garanticen y verifiquen la simulación de un movimiento por parte de *la máquina*.
Resultado esperado	Movimiento lógico simulado.

Se debe comprobar la simulación de movimientos en ambos juegos.

Prueba	PCR-4A Simulación de movimiento TicTacToe.
Requisito asociado	UC-11 Simular movimiento
Prerrequisito	Ninguno.
Procedimiento	1. Entrar en el juego TicTacToe, seleccionar partida de un jugador. 2. Realizar movimiento. 3. Comprobar que se muestra un movimiento generado por la aplicación y que este movimiento *tiene sentido*.
Resultado	Movimiento simulado y mostrado por pantalla.
Comentario	Con esta prueba también se garantiza el funcionamiento del requisito UC-9 Realizar movimiento.

Prueba	PCR-4B Simulación de movimiento Battleship.
Requisito asociado	UC-11 Simular movimiento
Prerrequisito	Ninguno.
Procedimiento	1. Entrar en el juego Battleship seleccionar partida de un jugador. 2. Colocar los barcos en el tablero e iniciar partida. 3. Realizar movimientos hasta perder el turno de juego. 4. Comprobar que se muestra un movimiento generado por la aplicación y que este movimiento *tiene sentido*.
Resultado	Partida iniciada.
Comentario	Con esta prueba también se garantiza el funcionamiento del requisito UC-9 Realizar movimiento.

Requisito	UC-12 Finalizar juego.
Descripción prueba	Se deben realizar pruebas que garanticen y verifiquen la finalización de la partida de juego.
Resultado esperado	Partida finalizada.

Se debe comprobar la finalización de partidas en ambos juegos.

Prueba	PCR-5A Finalización de partida TicTacToe.
Requisito asociado	UC-12 Finalizar juego.
Prerrequisito	Ninguno.
Procedimiento	1. Entrar en el juego TicTacToe e iniciar una partida. 2. Realizar movimientos hasta que alguien haga *Tres en raya* o no haya más movimientos posibles. 3. Comprobar que se muestra el resultado de la partida por pantalla y se ofrece la posibilidad de volver a jugar.
Resultado	Partida finalizada
Comentario	Con esta prueba también se garantiza el funcionamiento del requisito UC-10 Comprobar finalización.

Prueba	PCR-5B Finalización de partida Battleship.
Requisito asociado	UC-12 Finalizar juego.
Prerrequisito	Ninguno.
Procedimiento	1. Entrar en el juego Battleship e iniciar una partida. 2. Colocar los barcos en el tablero y comenzar a jugar. 3. Realizar movimientos hasta que alguno de los jugadores *hunda* todos los barcos del otro. 4. Comprobar que se muestra por pantalla el resultado de la partida (Victoria o Derrota).
Resultado	Partida finalizada
Comentario	Con esta prueba también se garantiza el funcionamiento del requisito UC-10 Comprobar finalización.

Pruebas de estrés

Además de pruebas de cumplimiento de los requisitos, se realizan pruebas de esfuerzo o estrés para asegurar que el sistema se comparte de manera correcta ante situaciones de mucha carga de datos o de procesamiento.

Para esto, se hace uso de la herramienta "Monkey" que se incluye con el paquete de desarrollo para Android. Esta herramienta genera distintos eventos de usuario como pulsaciones u otros gestos sobre la pantalla y escritura en caso de encontrar campos de texto, así como una serie de eventos a nivel de sistema. Se puede programar el número de eventos de la prueba.

La herramienta controla la aparición de errores, excepciones no controladas o bloqueo de la aplicación de manera que si alguna de estas cosas sucede se detiene la prueba y se informa del error. Además, como la aplicación tiene funcionalidades muljugador con base en conexiones entre dispositivos, las pruebas realizadas con Monkey serán con una partida de tipo Bluetooth iniciada.

Bajo estas condiciones, se han realizado pruebas estableciendo 500, 5.000 y 50.000 eventos o pulsaciones en la herramienta Monkey en dos dispositivos diferentes (una tablet Nexus 7 relativamente actual con Android 4.3 y una Smartphone Samsung Galaxy Ace S5830 con más de dos años de vida funcionando con Android 2.3), sin observarse ralentización o errores durante las mismas.

Resultados Nexus 7:

Número de eventos	Tiempo (ms)	Tiempo
500	2681	2,7 segundos
5000	28943	28,943 segundos
50000	317662	5,29 minutos ≈ 160 eventos/seg

Resultados Samsung Galaxy Ace S5830:

Número de eventos	Tiempo (ms)	Tiempo
500	12900	12,9 segundos
5000	130564	2,18 minutos
50000	1117536	18,63 minutos ≈ 45 eventos/seg

Manual de usuario

Índice de Contenidos

1. Introducción

Este manual pretende transmitir los conceptos y estructura de la aplicación SimpleAndroidGames para que cualquier usuario pueda sacar el máximo partido de la misma.

El manual comienza explicando los objetivos del proyecto y construcción de la aplicación para que se pueda entender la solución adoptada. Posteriormente, se comenta dicha solución. Una vez establecida esta base, se pasa a desgranar toda la aplicación, explicando su estructura, así como las funciones principales de ésta. Finalmente, se incluye un glosario con la terminología utilizada en este manual.

2. Objetivos del proyecto

El objetivo principal de esta aplicación para dispositivos móviles Android (smartphones y tablets) es dar soporte a juegos simples que se ejecutarán en ella, proporcionando una base común para jugar partidas multijugador.

Para ello, se tienen distintos subobjetivos alcanzados, que son:

- Definición de protocolo general de comunicación para juegos: encargado de administrar y gestionar los mensajes que se envían entre los dispositivos.
- Desarrollo de la lógica de dos juegos clásicos: encargado de llevar a cabo la lógica específica de cada juego.
- Diseño de elementos de juego: encargado de la gestión de la parte visual de los juegos desarrollados.
- Aplicación específica del protocolo sobre los juegos desarrollados: se encargará de administrar y gestionar cada tipo de comunicación soportada por la aplicación (Comunicaciones Bluetooth y Wi-Fi).

3. Plataforma de la aplicación

SimpleAndroidGame es una aplicación cliente-servidor. Por tanto, la plataforma de la aplicación adoptada se divide en estas dos unidades organizativas: terminal cliente y terminal servidor.

En caso de que se quiera jugar una partida multijugador mediante conexión Wi-Fi o Bluetooth, antes de iniciar un juego se debe elegir entre actuar como servidor o como cliente. Hay que tener en cuenta que el contrincante deber tomar el otro rol.

A efectos prácticos, como usuarios de la aplicación, sólo nos debe preocupar actuar como servidor si queremos ser el jugador que tiene inicialmente el turno de juego, no apreciaremos mayores diferencias entre actuar como cliente o como servidor.

4. Estructura de navegación de la aplicación

Teniendo en cuenta existen dos unidades organizativas (juegos) bien diferenciadas en la aplicación, se explica la navegabilidad de la misma.

Se ha intentado mejorar todo lo posible la organización de contenidos para lograr un acceso más eficiente y sencillo a la información por parte de los usuarios con independencia de sus conocimientos técnicos.

Con tal fin se deben tener en cuenta los siguientes aspectos:

1.- Existe un menú principal para dirigirse rápidamente a cada unidad organizativa (a cada juego).

2.- Existe un menú por cada unidad organizativa (TicTacToe y Battleship).

3.- Cada unidad organizativa tiene unas funciones específicas definidas, fácilmente accesibles para los usuario y que, una vez seleccionadas, se completan de forma intuitiva y sencilla, siendo guiado el usuario en todo momento por la aplicación.

5. Estructura del menú principal

Cuando se inicia la aplicación lo primero que se muestra es el menú principal. En este menú principal es posible seleccionar entre los diferentes juegos de la aplicación pulsando sobre la imagen que lo representa. Además, en esta pantalla se puede acceder a las preferencias de la aplicación donde es posible activar/desactivar el sonido y la vibración de los juegos. Para acceder a dichas preferencias hay que pulsar en la tecla menú del dispositivo seleccionar *Settings*.

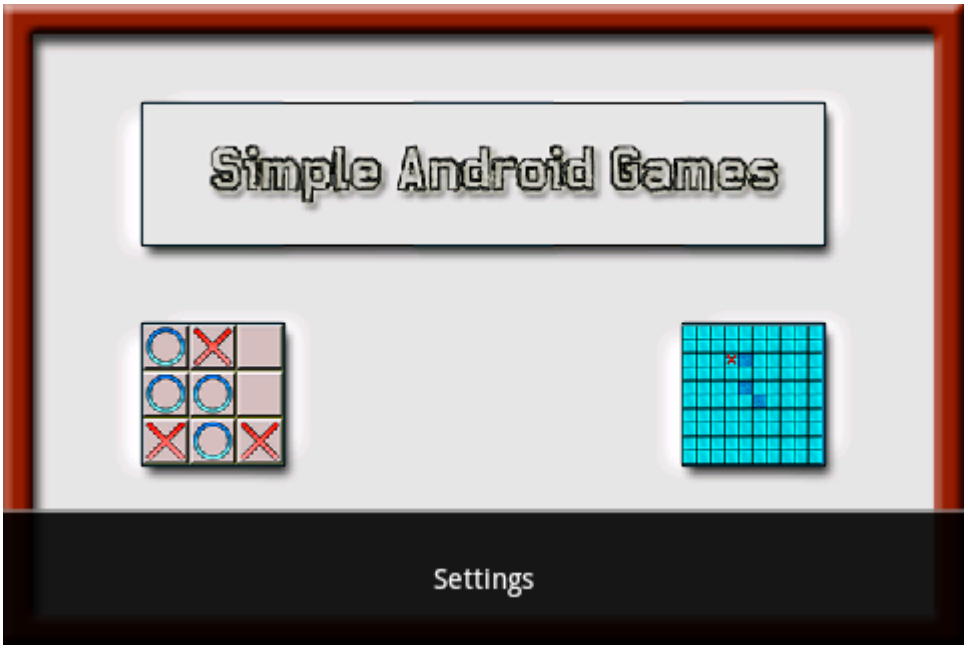


Figura 55. Pantalla principal.

6. TicTacToe

Para iniciar el juego TicTacToe hay que seleccionar el cuadro izquierdo en la pantalla de principal de la aplicación. Tras ello se cargará la estructura propia del juego, que en este caso se compone de dos pantallas, una principal para seleccionar el tipo de juego y una segunda pantalla donde se realiza el juego como tal.

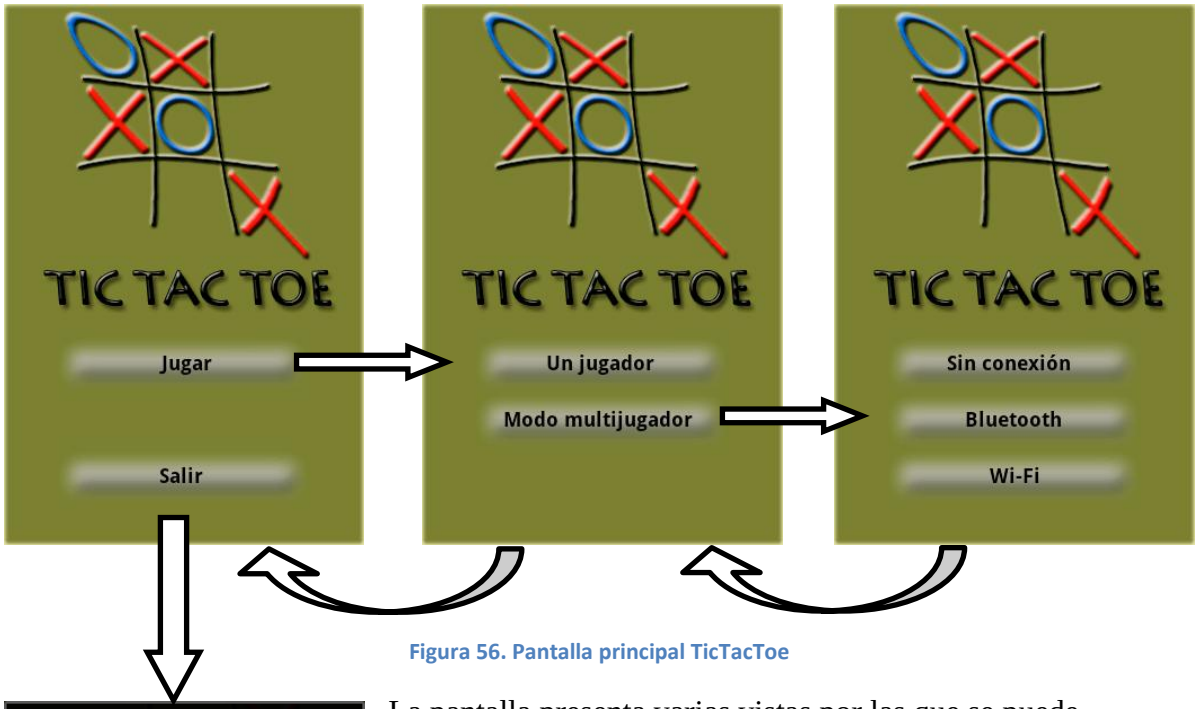
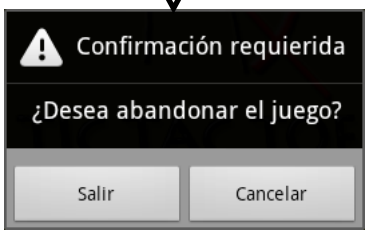


Figura 56. Pantalla principal TicTacToe



La pantalla presenta varias vistas por las que se puede *transitar* pulsado en los diferentes botones que contienen. En la primera vista hay dos botones, uno para salir del juego, que al ser pulsado hará que la aplicación solicite confirmación de salida; y otro para jugar, que al ser pulsado cargará la segunda vista.

En esta segunda vista se puede seleccionar el tipo de juego. Si se selecciona "Un jugador" la partida comenzará y se cargará la pantalla de juego. Si se selecciona "Modo multijugador" se mostrará la tercera vista para seleccionar el tipo de partida multijugador. Cuando se seleccione el tipo de juego multijugador se iniciará la pantalla de juego.

Desde cada vista es posible volver a la vista anterior pulsado el botón *Back* (atrás) del dispositivo.

Por su parte, la pantalla de juego consta de 2 elementos. En la parte superior se muestran los símbolos de cada jugador y se encontrará resaltado el símbolo del jugador que esté en posesión del turno de juego, de modo que el otro jugador no podrá realizar ningún movimiento hasta que tenga el turno. En la parte central de la pantalla se puede encontrar el tablero de juegos, que está formado por nueve casillas que se pueden seleccionar. Al seleccionar una de las casillas vacías se realizará el movimiento, poniéndose la ficha correspondiente en esa casilla y pasando el turno de juego al contrincante. Si se pulsa una casilla que ya contenga una ficha, no ocurrirá nada.

El juego finaliza cuando un jugador consiga colocar tres de sus fichas en línea horizontal, vertical u horizontal, o cuando no queden más movimientos posibles. Cuando la partida finaliza se muestra quién ha sido el ganador (o si ha habido empate) y se ofrecerá la posibilidad de jugar otra partida (*Volver a jugar*).

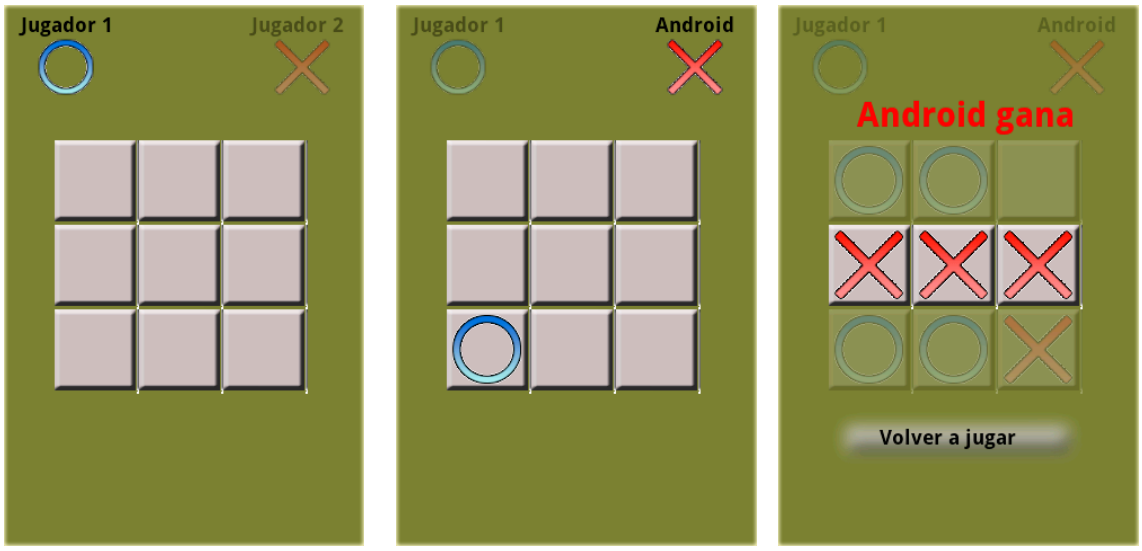


Figura 57. Pantalla de juego TicTacToe

7. Battleship.

Para iniciar el juego Battleship hay que seleccionar el cuadro derecho en la pantalla de principal de la aplicación. Tras ello se cargará la estructura propia del juego, que en este caso se compone de varias escenas, una inicial de carga, una principal para seleccionar el tipo de juego, otra escena para colocar los barcos en las posiciones del tablero que el usuario desee y una última escena donde se realiza el juego como tal.

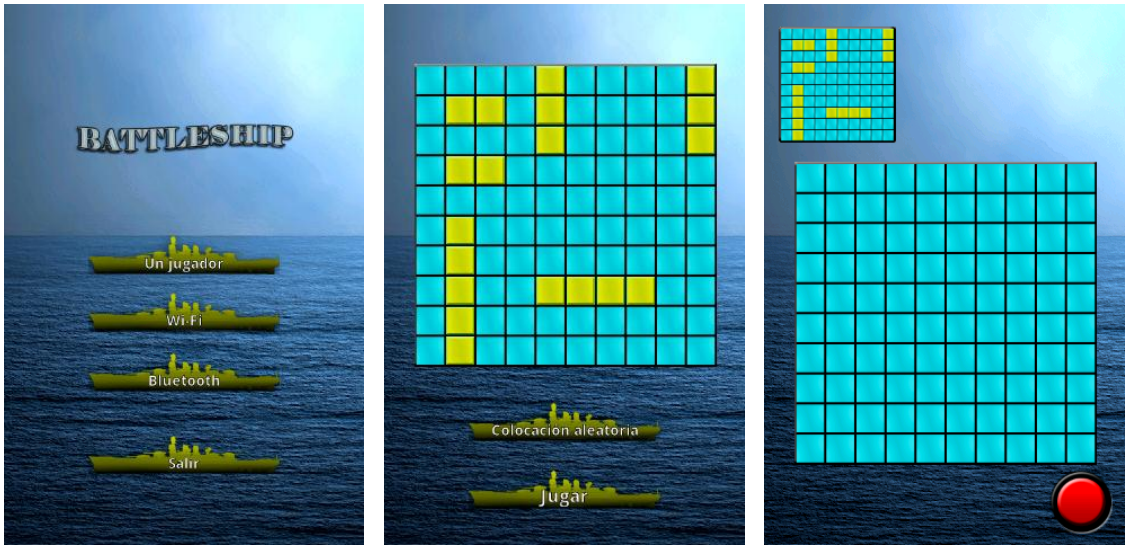


Figura 58. Pantallas Battleship.

En la pantalla principal se puede seleccionar el tipo de juego, que puede ser *Un jugador*, multijugador por *Wi-Fi* o multijugador por *Bluetooth*, y también se peude salir del juego (para lo que se requirirá confirmación).

Cuando se pulsa sobre uno de los botones de juego de esta pantalla, se pasa a la pantalla para colocar los barcos (requiriendo antes conexión cliente-servidor en caso de ser una partida por Wi-Fi o Bluetooth). En esta pantalla hay un tablero con diferentes barcos colocados que se pueden mover para que ocupen las posiciones desadas. Para mover un barco basta con pulsar en él y arrastrar (sin soltar por la pantalla) y para cambiar su horientación hay que pulsar dos veces sobré él. Esta pantalla nos da la opción de colocar los barocs automáticamente (*Colocación aleatoria*) y finalmente, nos posibilita iniciar la partida.

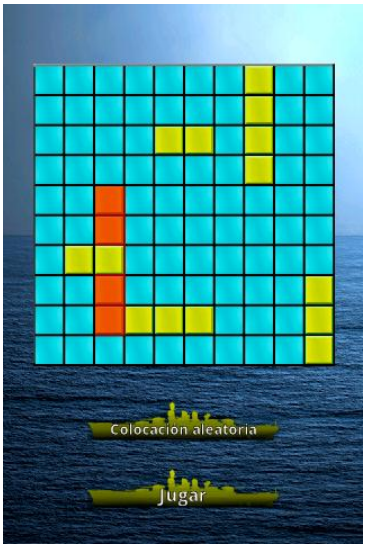


Figura 59. Colocación incorrecta

Existen restricciones sobre la oclocación de los barcos, estas son principalmente que no se pueden solapar unos con otros y que hay que dejar al menos una casilla vacía entre cada posición de un barco y otro adyacente. Cuando un barco no está colocado correctamente, se pondrá de color rojo y no será posible mover otro barco hasta que el restaldo sea colocado respetando las restricciones. Además, sólo se iniciará la partida de todos los barcos están correctamente colocados.

Tras pulsar el botón *Jugar*, se pasará a la pantalla de juego. Dicha pantalla consta de tres elementos principales. Estos elementos son un tablero en la parte superior izquierda que muestra los barocs propios y los movimientos que a realizado el contrincante, otro tablero que sirve para realizar movimientos e ir viendo el resultado de los mismos y finalmente un botón situado en la parte inferior derecha se sirve para confirmar los moviminetos.

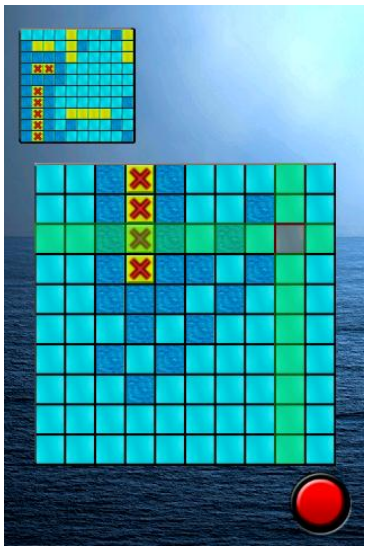


Figura 60. Selector de casilla.

Para realizar un movimiento estando en posesión del turno de juego hay que pulsar sobre una de las casillas del tablero central. Al pulsar se mostrará un selector que resalta en rojo la casilla seleccionada, si dicha casilla es válida (es decir, no se ha realizado un movineto previo en esa casilla) basta con pulsar sobre el botón para confirmar el movimiento. Una vez realizado el movimiento se mostrará el resultado del mismo que puede ser *agua*, *barco tocado* o *barco hundido*. En caso de que el resultado no se *agua*, el jugado seguirá con el turno de juego, es decir podrá realizar otro movimiento.

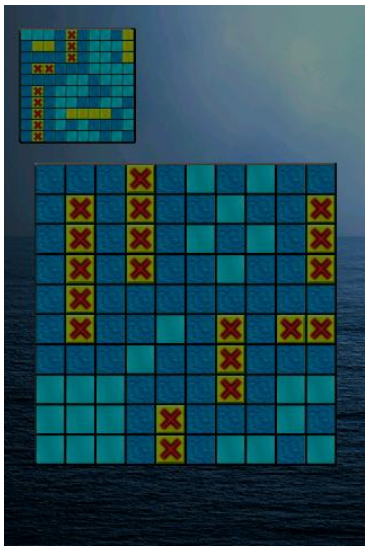


Figura 61. Partida finalizada.

La partida continuará con una sucesión de movimientos del usuario y del rival hasta que uno de los dos hunda todos los barcos del otro, momento en el cual se indicará el resultado de la partida (victoria o derrota) y finalizará el juego.

8. Partidas multijugador por Bluetooth y Wi-Fi.

El funcionamiento de las comunicaciones en partidas multijugador es igual en ambos juegos así que se engloban las instrucciones en un punto común.

Al seleccionar una partida multijugador por Wi-Fi, se comprobará si el conector de este tipo del dispositivo está activado. En caso de no estar activado, la aplicación abre los ajustes de conexión para que el usuario lo active. Una vez activado hay que conectarse a una red Wi-Fi. Una vez conectados el router asignará una dirección IP al dispositivo y ya estaremos en disposición de jugar la partida.

Cuando el dispositivo esté conectado a una red Wi-Fi podremos seleccionar entre actuar como cliente o como servidor. En caso se seleccionar *Servidor*, éste se iniciará, se mostrará la dirección IP para conectarse a él y se mantendrá a espera de que algún cliente se conecte.

La partida se iniciará cuando un cliente se conecte correctamente y se informará al usuario de ello.

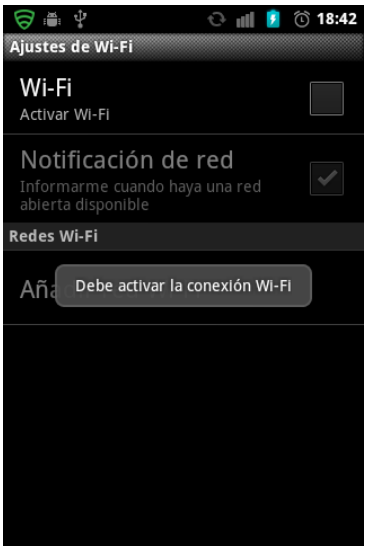


Figura 62. Ajustes de Wi-Fi

Por otra parte, si se selecciona actuar como cliente, se mostrará un campo de texto para introducir la dirección IP en la que localizar al servidor con el que se quiere conectar. Al pulsar *conectar*, se intentará realizar la conexión y en caso se realizarse se notificará al usuario y comenzará el juego.

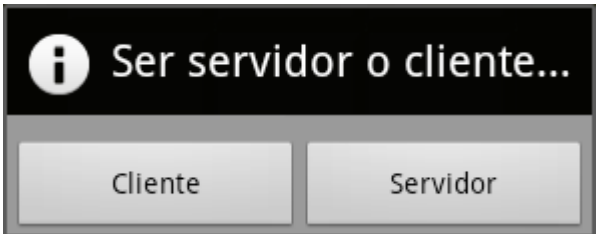


Figura 63. Actuar como cliente o servidor

Figura 65. Dirección IP del servidor.

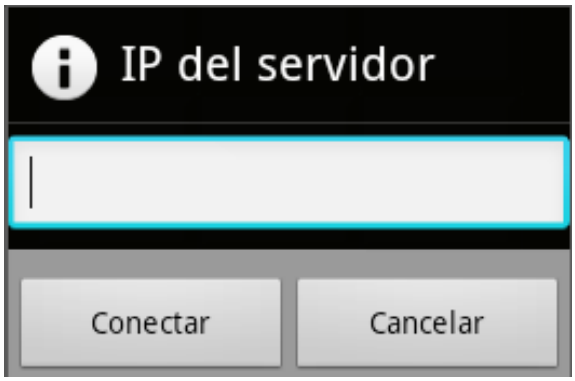


Figura 64. Conectar con servidor.

Al seleccionar una partida multijugador por Bluetooth, se comprobará si el conector de este tipo del dispositivo está activado. En caso de no estar activado, la aplicación solicitará permiso para activarlo. En caso de responder positivamente, se activará el Bluetooth y ya estaremos en disposición de jugar la partida.

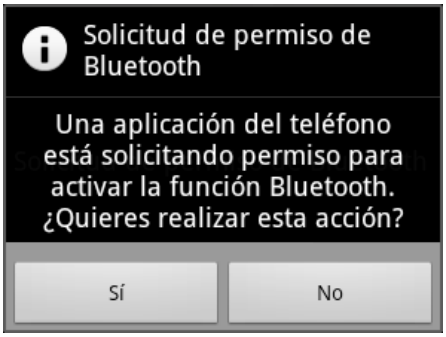


Figura 66. Activar Bluetooth.

Figura 67. Activando bluetooth.

Cuando el terminal tenga el conector Bluetooth activado podremos seleccionar entre actuar como cliente o como servidor de la misma manera que se hace en la conexión Wi-Fi. En caso se seleccionar *Servidor*, éste se iniciará, se mostrará el nombre del dispositivo y su dirección MAC para conectarse a él; además se mantendrá a espera de que algún cliente se conecte.

La partida se iniciará cuando un cliente se conecte correctamente y se informará al usuario de ello.

Figura 68. Datos servidor Bluetooth.

Por otra parte, si se selecciona actuar como cliente, se mostrarán en pantalla los dispositivos Bluetooth de los que nuestro terminal tiene constancia y se ofrece la posibilidad de escanear para buscar otros dispositivos. Cuando veamos el dispositivo al que queremos conectarnos para jugar, lo podremos seleccionar y conectarse a él, Al seleccionarlo, se intentará realizar la conexión y en caso se realizarse se notificará al usuario y comenzará el juego.

Figura 69. Conectar con servidor.

9. Información para desarrolladores

Si como desarrollador se quiere usar la base diseñada para añadir funcionalidad multijugador Wi-Fi y Bluetooth a juegos, habrá que tener en cuenta una serie de cuestiones que se comentan a continuación.

Se debe analizar qué información deben intercambiar los dispositivos durante la partida (y para configuraciones previas) y crear los mensajes específicos que se necesiten. Estos mensajes deben crearse por duplicado, uno implementando la interface *IMessageClient* e *IMessageServer* de AndEngine. Además, los mensajes puede extender la funcionalidad del tipo de mensaje base creado para esta aplicación *SimpleMessage*.

Habrá que crear, además, las interfaces para manejar esos mensajes implementando tanto *ClientActions* para los mensajes que llega al cliente, como *ServerActions* para los mensajes que llegan al servidor.

Sería conveniente crear clases específicas que sean subtipos de las clases genéricas WiFiServer, WiFiConnectorSever, BluetoothServer y BluetoothConnectorServer registrando los mensajes creados previamente para ello.

Como ejemplo los siguientes fragmentos de código:

```
public class BattleshipBluetoothServerConnector extends
BluetoothServerConnector<IClientBattleshipActions> implements BattleshipConstants
{

	public BattleshipBluetoothServerConnector(IClientBattleshipActions,
BluetoothSocketConnection, IServerConnectorListener<BluetoothSocketConnection>)
throws IOException {
	super(pClientActions, pConnection, pServerConnectorListener);


	this.registerServerMessage(FLAG_MESSAGE_SERVER_SET_CELL,
SetCellServerMessage.class, new
IServerMessageHandler<BluetoothSocketConnection>() {

	@Override
	public void onHandleMessage(final
ServerConnector<BluetoothSocketConnection> pServerConnector, final IServerMessage
pServerMessage) throws IOException {
		clientActions.onHandleSetCellServerMessage(pServerMessage);
		}
	});
	}

}
```

Listado 1. Estructura de cliente Bluetooth particular

```
public class BattleshipWiFiServer extends WifiServer<IServerBattleshipActions>
implements BattleshipConstants {

	public BattleshipWiFiServer(IServerBattleshipActions,
IClientConnectorListener<SocketConnection>,
ISocketServerListener<SocketConnectionClientConnector) {
		super();
	}

	@Override
	protected SocketConnectionClientConnector
newClientConnector(SocketConnection pSocketConnection) throws IOException {

		final SocketConnectionClientConnector clientConnector = new
SocketConnectionClientConnector(pSocketConnection);

	clientConnector.registerClientMessage(FLAG_MESSAGE_CLIENT_RESULT,
ResultClientMessage.class, new IClientMessageHandler<SocketConnection>() {

		@Override
		public void onHandleMessage(final IClientMessage pClientMessage)
throws IOException {
	serverActions.onHandleResultClientMessage(pClientMessage);
			}
		});
		return clientConnector;
	}
}
```

Listado 2. Estructura de Clase Servidor Wi-Fi particular

La Activity principal del juego debe extender de MyCommunicationGameActivity. En ella se deben implementar al menos el método *initMessagePool()* en el que habrá que registrar todos los mensajes que serán usados en el juego.

Además, dicha Activity principal debe implementar las interfaces creadas previamente para el juego. Las acciones a realizar cuando se reciba cada mensaje se deben especificar en el método *onHandleMessage* correspondiente, esa será la función que se ejecute cuando llegue un mensaje del tipo específico.

En cuanto a la creación de un juego desde cero, se aconseja que si se hace uso del motor de juego AndEngine se cree una única Activity base con varias escenas posibles para mostrar. De esta forma el rendimiento será mejor y el tránsito entre *vistas* más fluido. En caso de hacerse así habrá que tener especial atención a finalizar correctamente todos los recursos de cada escena antes de ser cerrada y gestionar correctamente el orden de transición entre escenas (especialmente cuando hay que seguir un orden inverso).

Consultar la documentación técnica del proyecto SimpleAndroiGames y la documentación propia de AndEngine para más información.

10. Glosario de términos.

SimpleAndroidGames: juegos simples para Android, nombre de la aplicación.

TicTacToe: juego Tres en Raya.

Battleship: juego Hundir la Flota.

Wi-Fi: mecanismo de conexión de dispositivos electrónicos de forma inalámbrica a través de un punto de acceso.

Bluetooth: especificación industrial para Redes Inalámbricas de Área Personal que posibilita la transmisión de datos entre diferentes dispositivos mediante un enlace por radiofrecuencia.

Cliente: dispositivo que se conecta al servidor.

Servidor: dispositivo que espera y recibe conexiones de clientes.

Manual de instalación

1. Introducción

En este documento se recoge el procedimiento a seguir para realizar la instalación de la aplicación SimpleAndroidGame en un dispositivo Android.

Para empezar, se explicarán los requisitos que debe cumplir todo dispositivo móvil en el que se quiera instalar la aplicación para su correcto funcionamiento. Seguidamente, se explicará el proceso completo de instalación de la aplicación.

Aunque no sea necesario, en ocasiones puede resultar útil tener instalado el conjunto de aplicaciones y complementos de desarrollo de aplicaciones para Android, para realizar diversas tareas así como labores de mantenimiento y control de la aplicación (incluso es posible crear y utilizar distintos emuladores de terminales Android). Como no se trata de algo necesario para el funcionamiento de la aplicación, simplemente se hace referencia a ello y se proporcionan los siguientes enlaces para poder realizar la instalación de estas aplicaciones y complementos:

- http://developer.android.com/sdk/installing.html
- http://www.contenidoandroid.com/android/como-usar-y-descargar-sdk-android/
- http://developeando.net/instalar-android-sdk-windows/

Se puede encontrar mucha más información al respecto (incluso videotutoriales que muestran detalladamente los pasos a seguir) realizando una rápida búsqueda en Google sobre la instalación de SDK Android, plugin ADT, desarrollo Android, etc.

2. Requisitos de instalación

La aplicación ha sido desarrollada para funcionar únicamente en dispositivos con sistema operativo Android, es por ello que el primer requisito y más importante es que el dispositivo en el que se quiera instalar la aplicación tenga dicho sistema operativo.

Como en todos los sistemas operativos, Android dispone de varias versiones. Cada versión contiene nuevas funcionalidades respecto a la anterior y necesitará un hardware más potente. En el caso concerniente, SimpleAndroidgame ha sido diseñada para funcionar a partir de la versión 2.2 de Android (también conocida como Froyo).

Además de lo anterior, para realizar la instalación se necesita espacio interno libre o una tarjeta de memoria compatible con el terminal (generalmente microSD), donde copiar el archivo de instalación de la aplicación.

3. Instalación

El proceso de instalación se realiza en tres pasos: Copiar el archivo de instalación en el dispositivo móvil, ejecutar el archivo de instalación desde el dispositivo móvil; y, finalmente, instalación de la aplicación.

A continuación se explican esos pasos:

1. Copiar el archivo de instalación en el dispositivo móvil.

Este paso consiste en "ubicar" el archivo de instalación de la aplicación en la tarjeta de memoria del dispositivo para poder ejecutarlo desde ahí y que se realice la instalación. El archivo en cuestión es *SimpleAndroidGames.apk* que se encuentra situado en la parte software del disco de instalación.

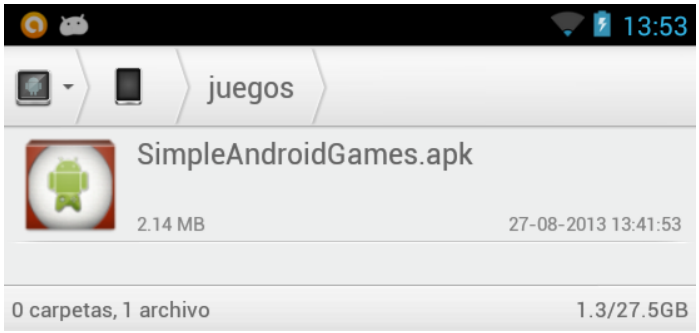
La manera normal de *pasar* el archivo de instalación al dispositivo Android conectándolo al ordenador y activar la depuración USB y el almacenamiento USB para poder acceder al dispositivo como si fuera una memoria externa.

En caso de esta opción no fuera posible, existen diversas formas de copiar el archivo en la tarjeta de memoria del dispositivo, entre las que se encuentran copiar el archivo directamente desde un ordenador utilizando un lector de tarjetas de memoria, enviarlo por bluetooth, correo electrónico o cualquier otra vía accesible desde el dispositivo o utilizar algún programa de intercambio de archivos entre ordenador y dispositivo (generalmente todos los dispositivos tienen uno propio que se instala al conectarlo con un ordenador).

Se recomienda copiar el archivo en un lugar conocido que sea rápidamente accesible, por ejemplo creando una nueva carpeta en el directorio raíz de la tarjeta de memoria.

2. Ejecutar el archivo de instalación desde el dispositivo móvil.

Este paso consiste en localizar y ejecutar desde el dispositivo el archivo que anteriormente se ha copiado. Para ello, el dispositivo debe tener algún programa u opción que permita explorar los directorios y archivos del dispositivo (Solid Explorer, FileManager, ES File Explorer, etc.); con este explorador de archivos hay que localizar el fichero que se ha copiado anteriormente para poder ejecutarlo



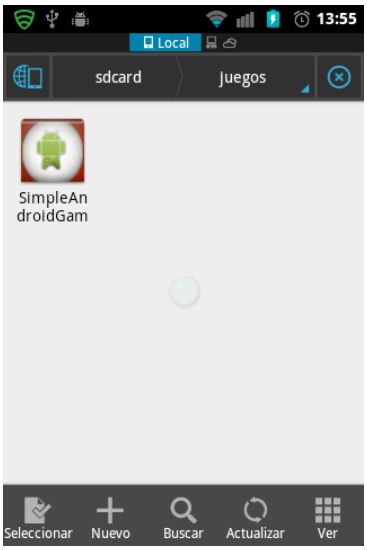


Figura 70. Archivo SimpleAndroidGames.apk

Antes de realizar la instalación será necesario activar la opción de "orígenes desconocidos" en el dispositivo Android. Para ello hay que ir a Ajustes→ Seguridad (o Aplicaciones dependiendo de la versión de Android) → Orígenes desconocidos (ver imagenes siguientes).

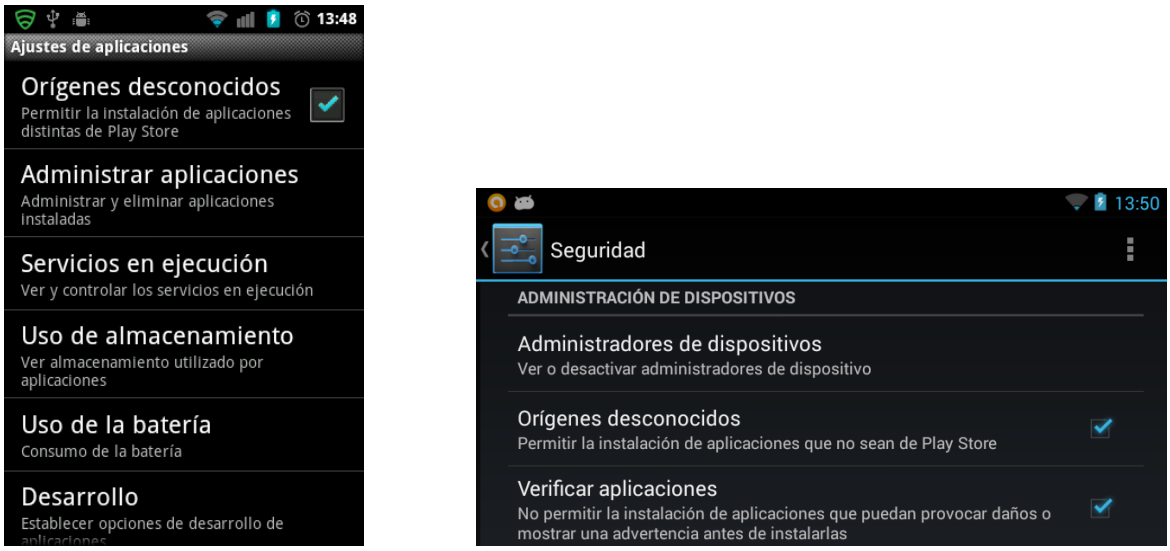


Figura 71. Activar orignes desconocidos.

3. Instalar la aplicación.

Al ejecutar el archivo *SimpleAndroidGame*.apk, tal y como se explica en el paso anterior, aparecerá pantalla en la que informa de los permisos que requiere la aplicación para funcionar y se solicita confirmación. Para confirmar basta con pulsar sobre el botón *Instalar*.

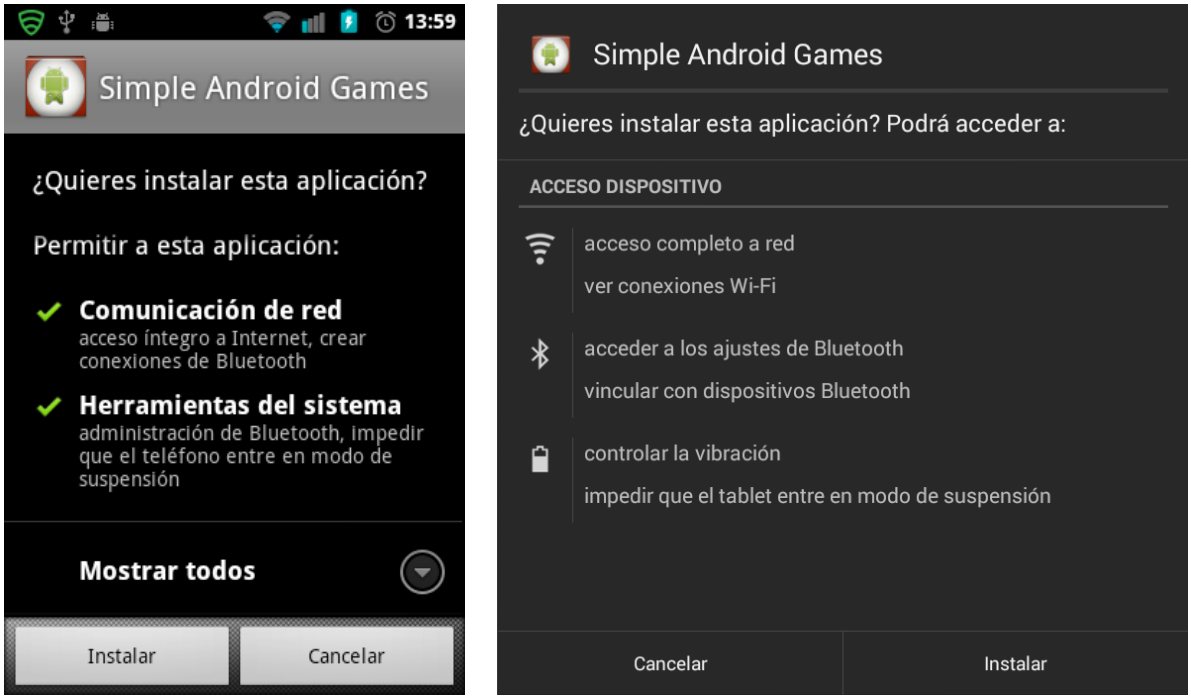


Figura 72. Instalación de aplicación.

Tras pulsar el botón *Instalar* comenzará el proceso de instalación de la aplicación.

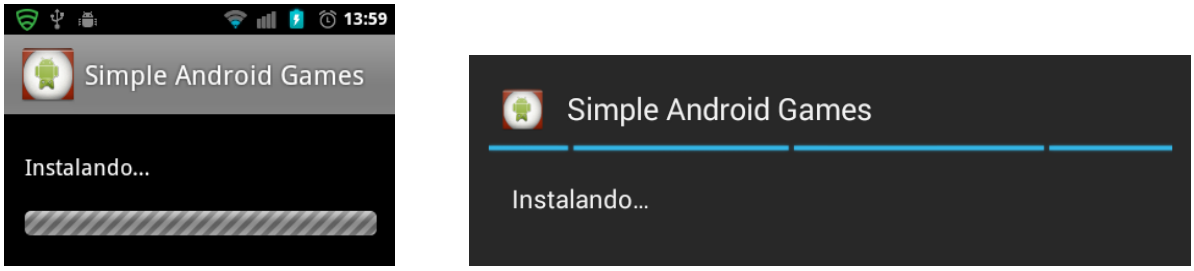


Figura 73. Progreso de instalación

Cuando finalice la instalación, se informa al usuario y se da la posibilidad de abrir la aplicación. Tras esto nuestra aplicación ya estará disponible para ser jugada en el dispositivo.

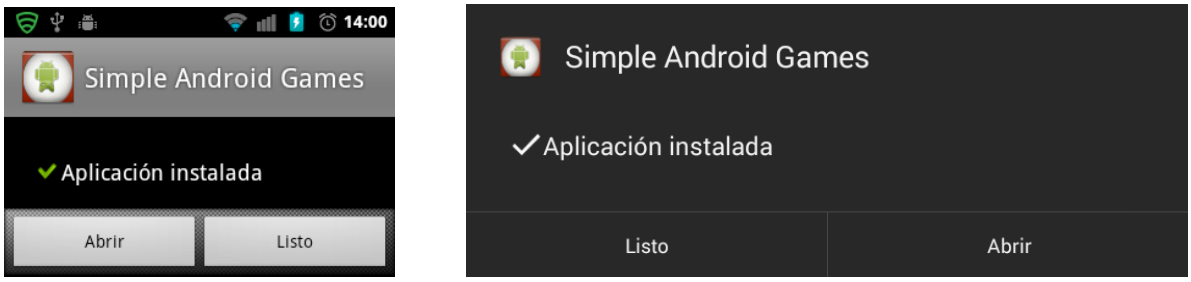


Figura 74. Instalación finalizada.

8. Conclusiones y posibles aplicaciones.

El proyecto se ha desarrollado con el objetivo principal de terminar la carrera, dada la necesidad de realización de Trabajo Fin de Grado para ello. Sin embargo, quería aprovechar la situación para trabajar sobre un aspecto desconocido para mí hasta ese momento, y de esta manera me serviría de excusa para ampliar mis conocimientos. Por esta razón me aventuré a explorar el mundo del desarrollo de juegos para Android.

Aunque claramente este mundo tiene mucho más que ofrecer que desarrollar un simple Tres en Raya o Hundir la flota, considero que ha sido un acercamiento interesante que sentará las bases para futuros desarrollos relacionados que seguro surgirán cuando me vuelva a *picar el gusanillo*.

Debido a varios inconvenientes y a la falta de práctica en desarrollos de este tipo, el proyecto ha requerido de más trabajo del inicialmente previsto y se han quedado sin llevar a cabo algunos aspectos que me habría gustado desarrollar. Entre ellos tenía especial interés en hacer algún juego que se saliera de la mecánica *por turnos* que siguen TicTacToe y Battleship, con más interacción por parte de los usuarios. De hecho, antes de sentar las bases de este proyecto, en el tiempo que estuve indagando sobre cómo hacer juegos para Android y sobre AndEngine, comencé el desarrollo de un juego de AirHockey, que me he propuesto terminar algún día.

Otro aspecto interesante que se ha quedado fuera del proyecto es la posibilidad de jugar con otro jugador que no esté necesariamente cerca de nosotros. Esto es, a través de un servidor central funcionando en Internet. Juegos como los conocidos Apalabrados o Triviados son un ejemplo de esto y es una vía de ampliación muy interesante para los juegos desarrollados.

Respecto a otras posibles mejoras o ampliaciones, se podrían añadir más juegos y más formas de comunicación a la aplicación con relativamente poco esfuerzo siguiendo el diseño de clases establecido en el proyecto. También se podían mejorar los juegos desarrollados en el aspecto gráfico y dando la posibilidad de almacenar partidas para ser retomadas en un momento dado.

Finalmente, aunque no sea una mejora propiamente dicha, se podría registrar y *subir* la aplicación a **Google Play**, que es el lugar oficial de Android donde se almacenan la gran mayoría de las aplicaciones de este sistema, lo cual facilitaría la difusión de la aplicación, siendo posible además realizar la instalación de la aplicación desde el propio Google Play.

9. Bibliografía.

Para la realización de este proyecto se ha utilizado los siguientes elementos bibliográficos:

- Libro **Android. Guía para desarrolladores**. De Frank Ableson, Charlie Collins y Robi Sen. Editorial Anaya Multimedia.

- Libro **Desarrollo de juegos ANDROID** (Beginning Android Games). De Mario Zechner. Editorial Anaya Multimedia.

- Libro **Java a fondo.** De Pablo Sznajdleder. Ra-Ma editorial.

- Apuntes de asignaturas cursadas a lo largo de la carrera de Ingeniero Técnico en Informática de Gestión de la Universidad de Valladolid, Escuela Universitaria de Informática de Segovia:
  - **Programación III** de Jesús Álvarez Gómez.
  - **Ingeniería del Software I** de Jesús Álvarez Gómez y Fco. José González Cabrera.
  - **Ingeniería del Software II** de Montserrat Serrano Montero.
  - **Redes** de Nicolás Sacristán Machín.
  - **Transmisión de Datos** de Nicolás Sacristán Machín.
  - **Calidad del Software** de Luís Ezequiel Hernanz Albertos.

- Apuntes de asignaturas cursadas a lo largo de la carrera de Grado de Ingeniería Informática de Servicios y Aplicaciones de la universidad de Valladolid, Escuela Universitaria de Informática de Segovia:
  - **Gestión de proyectos basados en las TI** de Fco. José González Cabrera.
  - **Plataformas de software empresariales** de Aníbal Bregón Bregón.
  - **Informática gráfica** de Luís María Fuentes García.

- Sitios web de consulta:
  - **Android developers.** Página oficial de Google para desarrolladores Android: http://developer.android.com
  - **Stack Overflow.** Foro de resolución de dudas referidas a todo tipo de lenguajes de programación: http://stackoverflow.com/
  - **Sitio Web AndEngine:** http://www.andengine.org/
  - **Jimmar's blog thingie**: https://jimmaru.wordpress.com/
  - **Wikipedia:** http://www.wikipedia.org/

- Sitios web de funcionalidades externas utilizadas:
  - **AndEngine**: http://www.andengine.org/

Índice de Figuras

Índice de Tablas