Universidad de las Ciencias Informáticas
Facultad 2

Título: Framework para desarrollar juegos Multi-jugador sobre J2ME para móviles con conexión Bluetooth.

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor(es): Yoandy del Toro González
Raida Zaldívar Picasso

Tutor(es): DrcT. Abel Alba Alfonso

Ciudad de la Habana, 29 de Junio del 2007

"Año 49 de la Revolución Cubana"

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Entidad Procyon Soluciones de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ____ días del mes de junio del 2007.

_____	_____	_____
Firma del Autor	Firma del Autor	Firma del Tutor
Yoandy del Toro González	Raida Zaldivar Picasso	Dr. Abel Alba Alfonso

OPINIÓN DEL USUARIO DEL TRABAJO DE DIPLOMA

El Trabajo de Diploma, titulado “Framework para desarrollar juegos multi-jugador sobre J2ME para móviles con conexión Bluetooth.”, fue realizado en la Universidad de las Ciencias Informáticas (UCI) de la provincia de Ciudad Habana. Esta entidad considera que, en correspondencia con los objetivos trazados, el trabajo realizado le satisface

1. Totalmente
2. Parcialmente en un ____ %

Los resultados de este Trabajo de Diploma le reportan a esta entidad los beneficios siguientes:

Y para que así conste, se firma la presente a los ___ días del mes de junio del 2007

_____ _____
Representante de la entidad Cargo

_____ _____
Firma Cuño

OPINIÓN DEL TUTOR DEL TRABAJO DE DIPLOMA

Título: Framework para desarrollar juegos multi-jugador sobre J2ME para móviles con conexión Bluetooth.

Autores: Yoandy del Toro González y Raida Zaldívar Picasso.

El tutor del presente Trabajo de Diploma considera que durante su ejecución el estudiante mostró las cualidades que a continuación se detallan.

Por todo lo anteriormente expresado considero que el estudiante está apto para ejercer como Ingeniero de las Ciencias Informáticas; y propongo que se le otorgue al Trabajo de Diploma la calificación de ____ .

_____ ____ de junio del 2007

Firma

Dedicatoria

A quien le debo mi ser, para quien no tengo las palabras capaces de expresar el sentimiento que despierta en mí y por quien daría mi vida por verla feliz
Mi madre, a ella una y mil veces...
A quienes supieron darme todo el amor y educación del mundo,
Mi padre, mi abuela, mis tías, mis primas, Carmen Lima, Beatriz Roca.
A quienes me han apoyado incondicionalmente, en este largo trecho……..
Roberto Cardascia, Pablo Olmeda Casado, mis amigos y profesores
A todos aquellos que con buena o mala intención me han hecho fortalecer ante cada obstáculo

Raida Zaldívar Picasso

A quien me forjó, me educó, siempre me guía y alumbra el camino, quien será mi orgullo donde quiera que este, Mi madre.
A mi padre por darme todo su amor, comprensión y todo lo posible.
A quienes supieron darme todo el amor y educación del mundo,
Mis abuelos, mis tíos.
A quien me han apoyado incondicionalmente,
Mi padrastro Luís y Ernesto Febles (el palestino)
A mi hermano Aniel por siempre tenerme un apretón y una sonrisa.
A mi esposa Elvia por estar siempre para mí en estos 5 años de Universidad y quererme tanto…
A todos los que confiaron en mí para que llegara a ser ingeniero…

Yoandy Del Toro González

Agradecimientos

A la máxima dirección de la Revolución Cubana, nuestro comandante Fidel Castro y a la UCI, por permitirnos formar parte de este proyecto futuro y contribuir a nuestra formación como profesionales revolucionarios. A Procyon Soluciones por contribuir en gran medida en nuestro desarrollo como profesionales. Agradecemos a nuestro tutor Abel, a York, y a todos los trabajadores de Procyon que de una forma u otra nos ayudaron.

A todos ellos, Muchas Gracias

Raida y Yoandy

Agradezco a todos los que me han acompañado en este largo camino de la vida y la educación dándome fuerzas para salir adelante y ver este sueño hecho realidad, mis amigos y mi esposa Elvia. A mi madre por preocuparse y darme siempre todo su apoyo y amor porque sin ella nada de esto hubiera sido posible. A mi padre, familiares y amigos que confiaron tanto en mí.

Muchas gracias, Yoandy del Toro González

Le agradezco a todas aquellas personas que siempre han confiado en mi y que me han apoyado aún cuando sea muy larga la distancia que nos separe. A Noel y a Alejandro por siempre estar pendientes de mis preocupaciones, a Roberto Cardascia por todo su apoyo y paciencia y a todos los amigos y amigas que con su inmenso cariño me han brindado seguridad. A mi familia, por no abandonarme nunca y a mi MADRE que la adoro.

Muchas gracias, Raida Zaldívar Picasso

RESUMEN

La telefonía móvil es uno de los negocios más productivos del mercado mundial. Uno de los contenidos más descargados son los juegos, con una mayor preferencia por los multi-jugador. La empresa cubana Procyon Soluciones se dedica al desarrollo de aplicaciones para móviles, con un mayor interés por los juegos sobre la plataforma J2ME, incursionándose tempranamente en el desarrollo de juegos multi-jugador con conexión Bluetooth.

Esta empresa posee un pequeño personal capacitado con una alta demanda de estos contenidos y aunque reutilizan código de juegos realizados con anterioridad, **no cuenta con una herramienta para el desarrollo de estos productos a mayor escala en un corto plazo de tiempo**. Luego de realizar un amplio estudio de las modernas técnicas de reutilización para alcanzar mayores beneficios, se decidió desarrollar JBGames: un framework que permite realizar juegos sobre J2ME para dispositivos móviles que se comuniquen vía Bluetooth; permitiéndole a los desarrolladores configurar el diseño de las pantallas; agregar elementos al menú; incorporar interfaces de usuario y abstraerse de la comunicación (capa en la que se establece la misma, detección de dispositivos y servicios, entre otros); centrándose así, sólo en el desarrollo de la lógica del juego.

Como aplicaciones de prueba sobre el framework fue implementado el juego por turno de Dominó con alta calidad y un menor esfuerzo y tiempo de desarrollo; lográndose así, satisfactorios resultados con la instanciación de JBGames.

INDICE

INTRODUCCIÓN

El vertiginoso desarrollo de la informática, la electrónica y las comunicaciones ha propiciado un mayor uso de las tecnologías de comunicación inalámbrica debido a su bajo nivel de inversión inicial y escalabilidad, su despliegue técnico relativamente simple, sus bajos costos y estándares abiertos, y su adaptabilidad a requisitos de voz y datos. Las mismas han permitido el surgimiento de nuevos dispositivos y de aplicaciones que ofrecen todo tipo de funciones.

En la actualidad, una de las tendencias en el software móvil es el desarrollo de contenidos. Entre las empresas de software mundial que ven en las aplicaciones del software móvil una nueva oportunidad de negocio se encuentran Handango, MobiHand, Mobile & Wireless Group, Motricity, entre muchas otras. Solamente la empresa Handango (www.handango.com), la principal plataforma de software para equipos móviles, tiene en el mercado más de 25 mil aplicaciones que distribuye a dispositivos móviles de Palm, Handspring, Sony, Nokia, Motorola, Spring, Orange SA, Microsoft, Sony Ericcson, HP, Sharp Electronics y más de otras 100 compañías[1].

A medida que los teléfonos celulares se van sofisticando, las aplicaciones que van relacionadas a estos dispositivos aumentan. A sus funciones básicas de recepción y emisión de llamadas, mensajes de voz, agenda y organizador personal, se han sumado las de MP3, capacidad de grabación, navegación por Internet, envío y recibo de correo electrónico, mensajería de texto, discado por comando de voz, radio FM, mensajería multimedia, cámara integrada, etc. En los últimos tiempos se han sumado capacidad de recibir, reproducir y filmar video; además de permitir la conectividad entre dos móviles que se encuentren a corta distancia para transferir archivos o ejecutar juegos: entre las tecnologías de comunicación inalámbrica que proveen este tipo de conexión de radiofrecuencia de bajo coste, baja potencia y corto alcance se encuentran Bluetooth e infrarroja (Infrared Data Association à IrDA).

Los propietarios de celulares en todo el mundo realizan descargas de contenido para sus equipos con el fin de personalizarlos. Una investigación realizada por la consultora LogicaCMG recientemente, que incluyó a Europa, el Pacífico Asiático, Norte y Sur América, reveló que el promedio mensual de descargas por usuario es actualmente de US$ 8.2 [1]. Los ringtones, la música y los juegos son los contenidos con mayor porcentaje de descarga; éstos últimos han sido clasificados, según el tipo de adversario con que el usuario interactúa, en modo simple o

multi-jugador. Los juegos en modo multi-jugador son muy solicitados en la actualidad por promover el crecimiento del aspecto competitivo, además de ser mucho más atractivos y emocionantes que los de modo simple; motivo por el cual múltiples empresas se dedican al desarrollo de éstas aplicaciones con altos ingresos monetarios.

La empresa cubana Procyon Soluciones presta una especial atención al empleo de nuevas tecnologías de acceso a contenidos y nuevos modelos de dispositivos móviles así como nuevos formatos multimedia y tipos de contenido. Dentro de las aplicaciones desarrolladas por esta entidad ocupan un lugar importante los juegos sobre la plataforma J2ME; ya se ha trabajado en los de modo simple pero por la elevada solicitud de los de modo multi-jugador con vía de comunicación Bluetooth en el mercado internacional se pretende alcanzar un mayor número de productos terminados en el menor tiempo posible.

Los juegos multi-jugador poseen funcionalidades comunes tales como el diseño de pantallas (presentación, estilos de los menú, etc.), la comunicación (capa en la que se establece la misma, detección de dispositivos y servicios, etc.), entre otros; lo cual hace de su diseño e implementación una tarea repetitiva. Los desarrolladores reutilizan códigos realizados con anterioridad, pero esta solución no es suficiente para cumplir con la demanda de productos terminados. A ello se le suma el reducido personal capacitado, la poca experiencia en esta esfera y el corto plazo de tiempo, motivo por el cual se plantea el siguiente Problema: **No se cuenta con una tecnología para desarrollar rápida, eficiente y a mayor escala los juegos multi-jugador sobre J2ME para dispositivos móviles que utilicen como vía de comunicación Bluetooth.**

Como respuesta al mismo, Procyon Soluciones ha decidido realizar un estudio de las soluciones implantadas a situaciones similares. La reutilización solamente de código es la que produce menores ganancias de productividad y fiabilidad, se alcanzan mayores beneficios al reutilizar diseños, modelos, componentes, frameworks[2]. Éstos últimos son el esqueleto de un conjunto de aplicaciones que pertenecen a un dominio concreto[3] y permiten la reutilización tanto del código del diseño como del código fuente; determinándose la realización de uno que integre todas aquellas funcionalidades que los juegos tengan en común.

Se ha trazado como Objetivo General desarrollar un framework que permita realizar juegos sobre J2ME para dispositivos móviles que se comuniquen vía Bluetooth; y como Objetivos Específicos el análisis de la estructura de este protocolo de comunicación, determinar el estado

del arte en el desarrollo de aplicaciones para móviles utilizando el mismo y realizar la implementación y documentación de un framework que permita desarrollar juegos multi-jugador.

Por ello se plantea como Objeto de Estudio un framework o marco de infraestructura para desarrollar juegos multi-jugador para móviles que utilicen como vía de comunicación Bluetooth; enmarcándose en el campo de acción de desarrollo de juegos para móviles con este tipo de tecnología inalámbrica.

La presente tesis contribuye a minimizar esfuerzos y a aumentar la producción de juegos para móviles en la empresa Procyon Soluciones, con el fin de satisfacer la alta demanda de estos productos y las expectativas de los clientes. Garantizando así, un mínimo de tiempo en la producción que propicie competir en el mercado con calidad y rapidez.

ESTRUCTURA DE LA TESIS

El presente trabajo está estructurado en cinco capítulos. En el primero se sintetizan la descripción de la situación problémica y las vías existentes para darle solución, así como el análisis crítico de las técnicas de reuso Orientadas a Objetos con el propósito de justificar la decisión de realizar un framework y el estudio de las tecnologías involucradas en el desarrollo del mismo. En el capítulo dos se analizan las particularidades de los frameworks y la manera en que se va a desarrollar el propuesto. Se realiza además, el análisis del dominio y la captura de los principales requerimientos. La definición del diseño del framework, acompañado de la descripción de cada uno de sus componentes, se resume en el tercer capítulo y en el cuarto se mencionan las aplicaciones construidas sobre el framework y se realiza un juego sobre el mismo como prueba de su funcionamiento. Por último, en el capítulo cinco, se realiza un exhaustivo análisis de la factibilidad del sistema.

1 CAPÍTULO
FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

El presente capítulo se refiere al estado actual del desarrollo de juegos multi-jugador para celulares internacional y nacionalmente. Se detallan las condiciones que provocaron el problema y la necesidad de darle solución a la mayor brevedad posible con el análisis de las técnicas modernas de reutilización Orientadas a Objetos (OO), así como las características de los frameworks que hacen apropiado su uso para solucionar el problema planteado.

1.2 Situación Problémica

El teléfono celular fue inventado en 1947 por la empresa norteamericana AT&T[1]. Su Primera Generación hizo su aparición en 1979 y se caracterizó por ser analógica y estrictamente para voz pero no se hizo portátil de manera práctica hasta 1983 cuando Motorola culmina el proyecto DynaTAC 8000X, el que es presentado oficialmente en 1984[4, 5].

En los últimos diez años, la evolución de la tecnología ha dado un salto impresionante. Con el surgimiento de Internet se ha revolucionado la rama de las comunicaciones y los teléfonos celulares o móviles han crecido en cuanto a capacidad y funcionalidad, de manera que han llegado a convertirse - algunos modelos - en computadoras de bolsillo, con las limitantes que sus características implican.

Como se planteó en la introducción, el negocio de la telefonía móvil es uno de los más importantes a nivel global, por lo que la gestión de los contenidos y aplicaciones desarrolladas o accesibles desde estos dispositivos es hoy en día uno de los porcentajes más importantes del negocio de las comunicaciones. A las funciones elementales de recepción y emisión de llamadas se le han integrado las de mensajes de voz, agenda y organizador personal, MP3, capacidad de grabación, navegación por Internet, envío y recibo de correo electrónico, mensajería de texto (Short Message Service à SMS), discado por comando de voz, radio FM, mensajería multimedia (Multimedia Message Service à MMS) y con ésta la opción de cámara

[1] American Telegraph & Telephone.

integrada; capacidad de recibir, reproducir y filmar video, televisión integrada; además de permitir la conectividad entre dispositivos para la transferencia de archivos o ejecución de juegos.

Los juegos son uno de los contenidos estrellas y la demanda de ellos es muy grande. El desarrollo de estas aplicaciones está en plena revolución y es un mercado con enorme potencial de crecimiento en el futuro. Un estudio de la firma In-Star/MDR calcula que sólo en EE.UU. los ingresos anuales procedentes de esta industria alcanzarán los 1.800 millones de dólares en el 2009[6]. En pocos años se ha pasado de los juegos en blanco y negro a los juegos a color, de los embebidos a los descargables y de los de un jugador a los multi-jugador. Se entiende por juego multi-jugador aquel que permite participar simultáneamente a dos o más jugadores. En estos juegos, los participantes compiten en demostrar su habilidad, bien conjuntamente para alcanzar un objetivo común o enfrentándose directamente en diferentes grupos o individualmente. Los juegos multi-jugador a su vez, pueden ser de dos categorías: por turno, en los que un jugador espera por la acción de los otros para poder realizar la suya y de acción, en los que no hay turnos sino que el tiempo transcurre de forma continua. El deseo de competir contra otros y lo atractivo de que las reacciones de los contrincantes sean mucho más imprevisibles que las de una máquina, ha provocado que múltiples empresas se dediquen al desarrollo de éstas aplicaciones obteniendo altos ingresos monetarios por la alta demanda de éstos productos en el mercado.

En Cuba, es muy incipiente el negocio de contenidos para dispositivos móviles. Una de las empresas pioneras que labora en este sentido es Procyon Soluciones, la cual presta una especial atención al empleo de nuevas tecnologías de acceso a contenidos y nuevos modelos de dispositivos móviles así como nuevos formatos multimedia y tipos de contenido.

Su mayor interés recae en la realización de juegos en modo simple y multi-jugador sobre la tecnología más extendida en el desarrollo de juegos para móviles: Java 2 Micro Edition (J2ME); la cual necesita la máquina virtual (KVM à Kilobyte Virtual Machine) destinada a cargar y ejecutar las aplicaciones con el fin de convertir el código de byte en código de máquina ejecutable. La KVM de los celulares tiene la responsabilidad de administrar las aplicaciones a medida que se ejecutan y estará formada por dos componentes: el CLDC (Connect Limited Device Configuration), configuración para dispositivos de bajo nivel con recursos y conectividad limitados y el MIDP (Mobile Information Device Profile), perfil en el que se almacena la interfaz de usuario, tipo de conexión de red, tipo de almacenamiento de datos y control de ejecución.

La conexión entre los dispositivos móviles inmersos en un juego multi-jugador puede establecerse de diversas formas. Dentro de las más usuales se encuentran el envío y recepción - con su respectivo costo - de SMS, pero puede tardar el recibo de mensajes en caso de saturaciones del servidor. Al mismo tiempo están GPRS, donde el mayor inconveniente es la obligatoria existencia de un servidor - lo cual incluye un precio - para poder efectuar una partida del juego, pero permite la comunicación a largas distancias; infrarrojo, con poca velocidad de transmisión, un rango visual de alcance máximo de 1 a 2 metros con probabilidades de pérdida de la conexión y Bluetooth, que a diferencia de GPRS y SMS no acarrea costos para el cliente pero está diseñado para conexiones a corta distancia, además no requiere del enfrentamiento visual de los dispositivos por ser omnidireccional y tiene un rango de alcance un poco mayor que el infrarrojo. Procyon Soluciones ha incursionado tempranamente en el desarrollo de los juegos multi-jugador mediante este último tipo de conexión, por lo que se profundizará en el estudio del mismo más adelante en el epígrafe 1.2.

Esta empresa consta de un pequeño personal calificado, con poca experiencia en esta esfera y con una alta demanda de juegos en un tiempo reducido. Para desarrollar uno nuevo, los programadores reutilizan métodos y clases implementados en otros, pero aún así aparecen trabas durante el proceso de desarrollo de la aplicación. Cada juego multi-jugador tiene su versión en modo simple de la cual es iniciada su implementación. Muchas de las funcionalidades comunes son realizadas una y otra vez redundantemente, incluso, la agrupación de componentes no es la adecuada para desarrollar un juego multi-jugador a partir de su versión de un jugador. Con la presencia de una herramienta que proporcione estos componentes ya implementados y su correcta estructura, los desarrolladores se dedicarían sólo a la particularidad del juego. Es por ello que se ve la necesidad de aumentar el nivel de reutilización del código con el fin de minimizar tiempo y energía en inventar los conceptos y eventos presentes en este dominio de aplicaciones.

A raíz de las mencionadas dificultades para la implementación de este tipo de contenido se decide realizar un estudio de los recursos y herramientas existentes, notándose que **no se cuenta con una tecnología para el desarrollo rápido, eficiente y a gran escala de juegos multi-jugador sobre J2ME para dispositivos móviles mediante una conexión Bluetooth**. Para solucionar este problema se hizo necesario realizar un estudio más detallado de este protocolo de comunicación.

1.3 Protocolo de comunicación Bluetooth. Especificación JSR- 82

Bluetooth es un estándar de comunicaciones que utiliza ondas de radio en la banda de frecuencia ISM[2] de 2,4 Ghz, permitiendo la comunicación entre dispositivos móviles que tengan transceptor Bluetooth. Su máxima velocidad de transmisión de datos en la versión 1.0 es de 1 Mbps y está diseñado pensando básicamente en tres objetivos: pequeño tamaño, mínimo consumo y bajo precio.

El hecho de utilizar una banda de uso libre que puede ser compartida por otras redes como por ejemplo, algunos teléfonos inalámbricos que trabajan en esta misma banda de frecuencias, puede provocar una gran cantidad de interferencias. Para ello, en los sistemas de radio Bluetooth se suele utilizar el método de Salto de Frecuencia (Frecuency Hopping): principal característica de la implementación del canal físico real de la capa de comunicación más baja de la pila de protocolos Bluetooth llamada Banda Base[7].

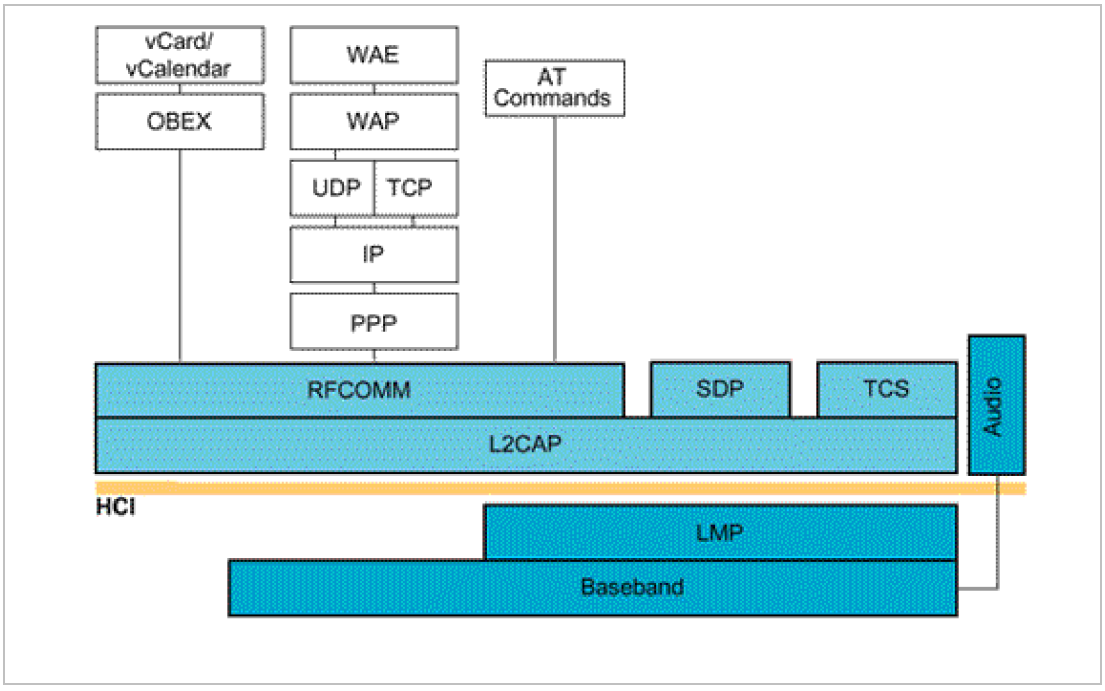


Figura 1.1 Pila de protocolos Bluetooth.

Los protocolos de alto nivel como el SDP (Protocolo utilizado para encontrar otros dispositivos Bluetooth dentro del rango de comunicación, encargado, también, de detectar la función de los

[2] Industrial, Scientific and Medical: Banda médico – científica internacional que no requiere de licencia del operador y está disponible en casi todo el mundo.

dispositivos en rango), RFCOMM (Protocolo utilizado para emular conexiones de puerto serial) y TCS (Protocolo de control de telefonía) interactúan con el controlador de banda base a través del Protocolo L2CAP (Logical Link Control and Adaptation Protocol). El protocolo L2CAP se encarga de la segmentación y reensamblaje de los paquetes para poder enviar paquetes de mayor tamaño a través de la conexión Bluetooth[7]. Más adelante se abordarán los protocolos RFCOMM y L2CAP.

Por la complejidad de este protocolo de comunicación, se hizo necesario un recurso que les permitiera a los programadores centrarse en el desarrollo de las aplicaciones y no en los detalles de bajo nivel del mismo. Como consecuencia, surgió "JSR-82 Java APIs para Bluetooth" como una iniciativa del fabricante de teléfonos móviles Motorola con el fin de estandarizar un conjunto de clases para que los dispositivos con capacidades Java se puedan integrar en entornos Bluetooth[7]. Su objetivo era definir un paquete de clases estándar abierto, no propietario que pudiera ser usado en todos los dispositivos que implementen J2ME. Por consiguiente fue diseñado usando los APIs J2ME y el entorno de trabajo CLDC/MIDP.

Este componente está dividido en dos partes que dependen del paquete CLDC javax.microedition.io: javax.bluetooth y javax.obex. Los dos paquetes son totalmente independientes. El primero de ellos define clases e interfaces básicas para el descubrimiento de dispositivos, descubrimiento de servicios, conexión y comunicación. La comunicación a través de javax.bluetooth es a bajo nivel: mediante flujos de datos o mediante la transmisión de arrays de bytes. Por el contrario, el paquete javax.obex permite manejar el protocolo de alto nivel OBEX (OBject EXchange). El protocolo OBEX es un estándar desarrollado por IrDA y es utilizado también sobre otras tecnologías inalámbricas distintas de Bluetooth[8]. Es muy similar a HTTP y es utilizado sobre todo para el intercambio de archivos. Este paquete no será utilizado en el desarrollo de las aplicaciones que le conciernen al presente trabajo, por lo cual no se profundizará en su estudio.

JSR-82 no depende de las APIs de MIDP, y en términos generales puede funcionar también sobre J2SE aunque su enfoque primordial son los dispositivos con capacidad de procesamiento limitada, poca memoria y que operan con batería[7].

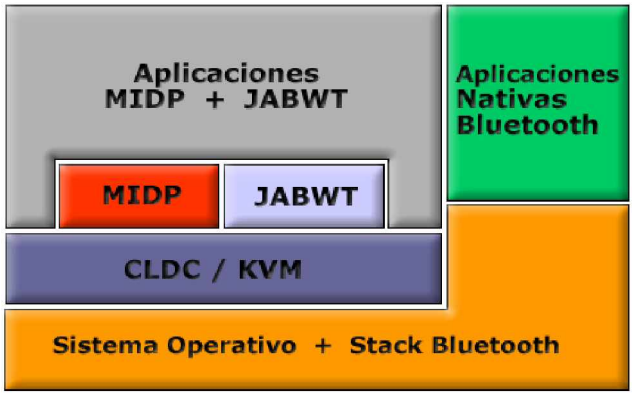


Figura 1.2 Arquitectura JSR-82 y MIDP.

El bloque del primer nivel en la figura 1.2 es el software del sistema o el sistema operativo del servidor. El sistema operativo del servidor tiene la parte del servidor de la pila de protocolos de Bluetooth y otras bibliotecas que son utilizadas internamente y por las aplicaciones nativas del sistema. Las aplicaciones nativas de Bluetooth interactúan con el sistema operativo directamente[7].

De manera general, en una comunicación Bluetooth existe un dispositivo que ofrece un servicio (servidor) y otros que acceden a él (clientes). Dependiendo de qué parte de la comunicación va a ser programada se deberá realizar una serie de acciones que se enumeran a continuación.

Un servidor Bluetooth deberá:

- Crear una conexión servidora.
- Especificar los atributos de servicio.
- Establecer la comunicación con los clientes.

Por otro lado, un cliente Bluetooth deberá realizar las siguientes:

- Búsqueda de dispositivos: La aplicación realizará una búsqueda de los dispositivos Bluetooth a su alcance que estén en modo conectable.
- Búsqueda de servicios: La aplicación realizará una búsqueda de servicios por cada dispositivo.
- Establecimiento de la conexión: Una vez encontrado un dispositivo que ofrece el servicio deseado el cliente se conectará a él.

Comunicación: Para establecer la conexión con el dispositivo servidor se pueden considerar dos tipos de protocolos L2CAP y RFCOMM de la pila de protocolos Bluetooth.

L2CAP es el protocolo de la capa inferior que está orientado al envío de paquetes que tienen un tamaño máximo variable según el dispositivo utilizado, por lo que hay que dividir la información a enviar en paquetes y luego volver a unirla en el destino[9].

RFCOMM (Radio Frequency Communication) es el protocolo de la capa superior y usa el perfil de puerto serie (SPP à Serial Port Profile) para la comunicación. La principal característica de este protocolo es que no está orientado al envío de paquetes. Se considera que hay un flujo (stream) entre los dos extremos de la conexión, en el que cada uno puede escribir toda la información que quiera y el otro la lee cuando la necesite, garantizando que el orden en que se leen los datos es el mismo en el que se escribieron. Este protocolo de "sustitución de cable serie" emula las señales de control y datos RS-232, proporcionando capacidades de transporte a los servicios de niveles superiores que utilizan el cable serie como mecanismo de transporte. Soporta hasta 9 puertos serie RS-232 y permite hasta 60 conexiones simultáneas (canales RFCOMM) entre dos dispositivos Bluetooth [9, 10].

Para usar un servicio en un dispositivo Bluetooth remoto, el dispositivo local debe comunicarse usando el mismo protocolo que el servicio remoto. Uno de los puntos fuertes de la tecnología Bluetooth es el alto grado de interoperabilidad que permite entre aplicaciones alojadas en diferentes dispositivos la cual se consigue gracias a la pila de protocolos Bluetooth [11]. Para profundizar el estudio de los protocolos de bajo y alto nivel de la pila Bluetooth puede consultarse [12],[13],[14].

Durante el estudio realizado acerca de Bluetooth y su vinculación con J2ME a través del API JSR-82, se analizaron una serie de documentos con el propósito de reutilizar código fuente de manera eficiente, pero con esto solamente no se satisfacen los requerimientos necesarios para un mayor porcentaje de productos terminados.

1.4 Técnicas de reutilización

Entre los artefactos reutilizables, además del código fuente, se encuentran los diseños, modelos, componentes y frameworks. La reutilización de un artefacto ocurre cuando es utilizado en un contexto diferente para el que originalmente fue diseñado conduciendo a un

desarrollo más rápido y programas de mejor calidad[2]. El reutilizar software significa incrementar, más que automatizar, los procesos mismos del negocio[15].

Las modernas técnicas de reutilización OO – componentes de software, patrones de diseño y frameworks – han demostrado satisfacer las necesidades de los programadores de los últimos tiempos.

Un componente de software es una porción de código a ser reusada. Según Clemens Szyperski, *"... Un componente es una unidad de composición de aplicaciones software, que posee un conjunto de interfaces y un conjunto de requisitos que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes de forma independiente, en tiempo y espacio ..."[16]*. Las interfaces determinan tanto las operaciones que el componente implementa como las que precisa utilizar de otros componentes durante su ejecución y los requisitos determinan las necesidades del componente en cuanto a recursos, como por ejemplo las plataformas de ejecución que necesita para funcionar.

Los programadores de Procyon Soluciones utilizaron componentes del proyecto Benhui[3] para el establecimiento de la comunicación en aquellos juegos para un jugador que permiten la implementación de su versión multi-jugador. Dicho proyecto proporciona un conjunto de recursos y herramientas para el desarrollo de aplicaciones Bluetooth, pero al relacionar los componentes del juego y los del proyecto se presentan problemas de compatibilidad entre ellos.

Esta falta de convergencia se manifiesta por diferentes factores. La diversidad en los diseños de clases de los juegos es uno de los mayores inconvenientes, debido a que cada programador crea su propio diseño siéndole más fácil o no incluir la comunicación. Aún cuando se logran integrar los componentes de Benhui, estos están sujetos a cambios pues la metodología y diseño de implementación propuesto están orientados a aplicaciones de otra índole, por lo que no tiene mucha relación con la teoría de comunicación en los juegos por turno. Este factor influye principalmente en el tiempo de desarrollo de los juegos, pues los programadores están obligados a estudiar el protocolo de comunicación antes de realizar cualquier cambio en el componente de Benhui. Es por ello que la reutilización de componentes de software no fue

[3] http://www.benhui.net

suficiente para el desempeño de lo que se pretendía realizar, motivo por el cual fue necesaria la búsqueda de otras vías alternativas.

Se analizó la posibilidad de esbozar un diseño capaz de integrar la comunicación al resto de la aplicación. Los patrones son una técnica de reutilización utilizada en la fase de diseño. Una de las definiciones más aceptadas por la comunidad de software es que: *Un patrón de diseño es una información que captura la estructura esencial y la perspicacia de una familia de soluciones probadas con éxito para un problema repetitivo que surge en un cierto contexto y sistema[17]*. Resumiendo, es la descripción de un problema que ocurre repetidas veces con su respectiva solución y las consecuencias de su aplicación, si algo no se repite, no es posible que sea un patrón.

A partir del análisis a un conjunto de trabajos relacionados con el tema, se intentaron extraer aquellas características comunes que lograran sentar las bases para la descripción de un problema específico que pudiera solucionarse mediante un patrón de diseño. Se consultaron porciones de código y documentos aportados por el proyecto "Vega Solaris"[18], en el que se desarrolla un juego para móviles con conexión Bluetooth; la tesis de diplomado realizada por Arantza Sánchez Fernández[11], que realiza un minucioso estudio de las tecnologías abordadas en este trabajo y las especificaciones o recomendaciones - para el desarrollo de juegos con las mencionadas tecnologías - propuestas por las corporaciones Nokia[19] y Motorola[20]. Como resultado, se adquirieron ideas para el diseño de clases en la implementación de un juego multi-jugador a partir de su versión simple sin afectar el diseño de este último, pero esto no garantizaba el correcto intercambio de información durante una partida de juego al integrar los componentes de Benhui, principalmente por el último de los factores mencionados.

Ésta técnica permite sólo el reuso de ideas de diseño por lo que no es capaz de resolver completamente el problema que impulsa el desarrollo de la presente investigación. Sólo con el diseño de clases para la creación de un juego no basta, pues el componente principal que diferencia los juegos de un jugador de los multi-jugador es el de la comunicación y los patrones de diseño no garantizan la implementación del mismo. Sin embargo, relacionados con los patrones de diseño están los frameworks: un framework se puede ver como la implementación de un sistema de patrones de diseño[17]. Los patrones de diseño fueron descubiertos al examinar varios frameworks y fueron seleccionados como representativos de software reusable, por lo que son elementos microarquitectónicos de ellos[21].

Tanto componentes como patrones, aportan los beneficios necesarios para solventar las dificultades de los programadores de juegos multi-jugador de Procyon Soluciones, sólo que no de manera independiente. Sin embargo, ambos en conjunto proporcionan una herramienta básica para la construcción de múltiples aplicaciones que presentan características comunes. *Un framework se define como una colección de componentes software que se organizan y colaboran según el modelo descrito por una arquitectura y proporciona el más alto nivel de reutilización en el desarrollo de sistemas complejos[22]*. Pueden incluir soporte de programas, librerías y un lenguaje de scripting[4] entre otros softwares para ayudar a desarrollar y unir los diferentes componentes de un proyecto[22]. Los más evolucionados incorporan herramientas y un conjunto de componentes base que conforman el esqueleto de la arquitectura y diseño del producto.

En el mercado en general, se observan tendencias comunes en los principales fabricantes de software. Dichas tendencias pasan por la existencia de framework / marcos de infraestructura que abstraen a los desarrolladores de la complejidad tecnológica del software de base[23]. Los frameworks son más abstractos y flexibles que los componentes de software debido a que no resuelven una funcionalidad concreta, sino que con la colaboración de otros componentes y la modificación de requisitos particulares del dominio en el que se desenvuelve, es capaz de ajustarse a cada aplicación. Por otro lado, son más concretos y fáciles de reutilizar que el diseño puro de los patrones porque están orientados a las particularidades de un dominio de aplicaciones concreto, mientras los patrones se pueden utilizar en prácticamente cualquier tipo de aplicación.

A pesar de que los frameworks OO no son capaces de sintetizar todas las características de su dominio y de que los desarrolladores deben realizar un profundo estudio del mismo antes de utilizarlos, se ha determinado el uso de esta técnica OO como solución al problema planteado. Su capacidad de reutilización del código de diseño y del código fuente los hace mucho más abarcadores al resolver las necesidades arquitectónicas y funcionales de las aplicaciones permitiendo una productividad mayor y un tiempo de mercado breve.

[4] Los lenguajes script presentan un estilo de programación diferente, basado en la existencia de un conjunto de componentes que deben ser combinados de un modo fácil y rápido, por medio de sentencias que proveen mayor funcionalidad y carencia de restricciones de tipo. Los componentes base son comúnmente desarrollados en un lenguaje de sistema que es más eficiente, y son unidas o combinadas por el código script.

En 1997, Mohamed Ebrahim Fayad y Douglas C. Schmith, plantearon que "…en contraste con las técnicas de reuso OO iniciales basadas en librerías, los framework están orientados a unidades del negocio particulares y a dominios de aplicación…"[21]. Los mismos son clasificados según su ámbito en: framework de dominio específico que describe algún tema o problema, como por ejemplo, CORBA que define un enfoque de sistemas distribuidos y .NET Framework que denota la infraestructura sobre la cual se reúnen un conjunto de lenguajes, herramientas y servicios que simplifican el desarrollo de aplicaciones en entorno de ejecución distribuido. Su otra categorización es framework de aplicación, que es un conjunto de componentes de implementación que proporcionan una base sobre la cual construir una aplicación, como el que se pretende realizar. Los frameworks de aplicaciones OO son una tecnología promisoria para rectificar diseños e implementaciones de software[24].

En el estudio realizado de frameworks de aplicación destinados al desarrollo de juegos multi-jugador, los que se han encontrado, están orientados a las videoconsolas principalmente, como es el caso del framework para videojuegos de acción bi-dimensionales[15]; el programa INXENA[25] que nace con el propósito de crear un framework para desarrollar juegos de estrategia por turnos multi-jugador con la participación de agentes inteligentes; TRINX[25], para juegos de aventura gráfica y Project Darkstar, para la construcción de juegos on-line multi-jugador, accesibles desde varios clientes distintos (entre ellos un móvil). Hasta el momento, no se conoce aún de la existencia de uno destinado a los juegos multi-jugador para móviles sobre J2ME con conexión Bluetooth.

El principal propósito de esta tesis, es la realización de este framework con su respectiva documentación. Para la elaboración del mismo, – que en el próximo capítulo se abordará con profundidad –, se ha de ahondar en el estudio del campo de acción en el que se desenvuelve, siendo imprescindible la intervención de los programadores de este tipo de juegos, pues a partir de sus experiencias se implementarán los componentes base necesarios; además de efectuarse estudios de la estructura de clases que utilizan las grandes empresas de desarrollo de este tipo de contenidos basado en los frameworks mencionados.

1.5 Proceso de Desarrollo

El diseño de un framework es un proceso iterativo, principalmente porque los desarrolladores no conocen cómo hacerlo en una sola iteración, para lo que es necesario un buen entendimiento del dominio del problema[3]. En la actualidad, se destacan las metodologías

Extreme Programming (XP) para proyectos de corto plazo con un pequeño equipo; y Rational Unified Process (RUP à Proceso Unificado Rational) que se caracteriza por ser iterativo e incremental, dirigido por casos de uso y centrado en la arquitectura. Aquellos equipos que adoptan RUP no están obligados a realizar todas las actividades propuestas sino que deben considerarlo como la base para sus propios procesos[26]; por lo que será la metodología a utilizar en el desarrollo del framework.

Esta metodología de IBM Rational para el desarrollo y construcción de software está basada íntegramente en el Lenguaje Unificado de Modelado UML (Unified Modeling Language). UML es un lenguaje de modelado visual que se utiliza para especificar, visualizar, construir y documentar artefactos (modelos) de un sistema de software, desde una perspectiva OO[27]. Describe la notación para clases, componentes, nodos, actividades, flujos de trabajo, casos de uso, objetos, estados y cómo modelar la relación entre esos elementos. El UML provee beneficios significativos para los ingenieros de software y las organizaciones al ayudarles a construir modelos rigurosos, trazables y sostenidos, que soporten el ciclo de vida de desarrollo de software completo.

Las herramientas CASE[5] de modelado con UML permiten aplicar la metodología de análisis y diseño orientados a objetos y abstraerse del código fuente, en un nivel donde la arquitectura y el diseño se tornan más obvios y más fáciles de entender y modificar[28]. En la tesis se emplea la herramienta Rational Rose Enterprise Edition 2003 con este propósito.

1.6 Conclusiones

En el presente capítulo se han abordado las causas que dan origen al problema, la necesidad de su solución, así como la profundización en el estudio del protocolo de comunicación Bluetooth y de las modernas técnicas OO existentes para solventar dicho inconveniente. Se definió además, la metodología y herramienta a utilizar y las tecnologías involucradas en el desarrollo del futuro sistema. A continuación se ampliará el estudio de los frameworks, se determinará su dominio y sus principales requerimientos.

[5] Computer Aided Software Engineering: Ingeniería de Software Asistida por Ordenador

2 CAPÍTULO
CARACTERÍSTICAS DEL SISTEMA

2.1 Introducción

En este capítulo se profundiza en el concepto de framework, se enuncian sus principales características, se fundamenta qué tipo de framework se ha de aplicar y cómo se desarrollará el mismo, comenzando por el análisis del dominio y la captura de sus principales requerimientos.

2.2 Características de los Frameworks

La decisión de desarrollar en base a un framework se sustenta en la identificación de un conjunto de proyectos de aplicaciones, actuales o futuros, que comparten un dominio específico y una lógica sobre éste. En el epígrafe 1.3 se hizo referencia a la clasificación según su ámbito. El propuesto es de aplicación que, atendiendo a la forma en la que permiten ser extendidos se les divide en[16]:

- Caja de Cristal, que admiten la inspección de su estructura e implementación, pero no su modificación ni reescritura, salvo para sus puntos de entrada (término que será explicado posteriormente).
- Caja Blanca, que requiere del completamiento o redefinición de algunos métodos mediante herencia, por lo cual es necesario el conocimiento de mecanismos internos del sistema.
- Caja Negra, que es un sistema completo, requiere solo seleccionar y configurar componentes ya existentes, para lo cual basta con conocer la interfaz y comportamiento general del sistema.
- Caja Gris, que define una forma intermedia de reutilización, en donde sólo parte del interior de un framework (o componente) se ofrece de forma pública, mientras que el resto se oculta. Es una solución intermedia entre las cajas blancas y negras, y de ahí su nombre. Aunque el nombre de caja gris es menos conocido que los otros, es el que describe la forma de extensión más utilizada.

El framework a realizar será de código abierto para su posible inspección y modificación, por lo cual es clasificado de Aplicación por Caja Blanca debido a que los desarrolladores tienen que redefinir algunos métodos mediante herencia en algunas clases y podrán incorporar o transformar funcionalidades internas del mismo.

El término framework es usado en forma corriente en la Ingeniería del Software para describir una diversidad de modelos y ambientes que tienen por común una característica mínima: establecen un marco de trabajo para desarrollar un producto de un género determinado. Su desarrollo tiene tres etapas principales: análisis del dominio, diseño e "instanciación"[29], tal como lo muestra la siguiente figura:

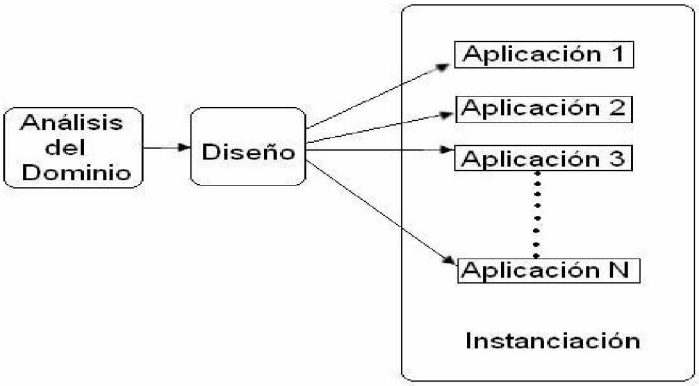


Figura 2.1 Proceso de desarrollo del Framework.

En el análisis del dominio se seleccionan y abstraen las funciones y objetos específicos, se identifican las características comunes y las relaciones fundamentales. Durante esta etapa, los puntos calientes o de entrada (hot-spots)[6] y los puntos congelados o de salida (frozen-spots)[7] se destapan parcialmente.

[6] Los puntos calientes o hot-spots son las clases o los métodos abstractos que deben ser implementados o puestos en ejecución.

[7] Algunas de las características del framework no son mutables ni tampoco pueden ser alteradas fácilmente. Estos puntos inmutables constituyen el núcleo o kernel de un framework, también llamados como los puntos congelados o frozen-spots del framework.

La etapa del diseño define las abstracciones de éste. Se modelan los puntos calientes y los puntos congelados y la extensión y la flexibilidad propuesta en el análisis del dominio se esboza en líneas generales. Es aquí donde son utilizados los modelos del diseño.

Finalmente, en la etapa de "instanciación", los puntos calientes son implementados, generando un software del sistema. Es importante observar que cada una de estas aplicaciones tendrá los puntos congelados en común.

Para la construcción de un framework existen dos enfoques: desde una arquitectura o desde las aplicaciones. El primero parte desde una arquitectura de software que representa un modelo de las partes del sistema y sus interacciones. El segundo, identifica los componentes comunes y sus relaciones de las aplicaciones pertenecientes a un dominio.

El framework a desarrollar será enfocado desde las aplicaciones, basados principalmente en el conjunto de componentes comunes para la implementación de los juegos multi-jugador con conexión Bluetooth. Anteriormente, se mencionó sobre qué metodología se fundamenta el desarrollo del mismo. RUP debe ser ajustado a las necesidades y realidades del equipo de desarrollo[26] y se basará en el modelo de desarrollo orientado a la reutilización; este modelo reduce el tiempo de entrega y disminuye el esfuerzo, el costo y los riesgos durante el desarrollo. Consta de cuatro fases[30]:

1. Análisis de componentes: Se determina qué componentes pueden ser utilizados para el sistema en cuestión. Casi siempre hay que hacer ajustes para adecuarlos.
2. Modificación de requisitos: Se adaptan (en lo posible) los requisitos para concordar con los componentes de la etapa anterior. Si no se puede realizar modificaciones en los requisitos, hay que seguir buscando componentes más adecuados (fase 1).

   Estas dos fases entrarían dentro de la etapa de Análisis del Dominio del desarrollo del framework. Las siguientes se relacionan con las otras dos etapas respectivamente.

3. Diseño del sistema con reutilización: Se diseña o reutiliza el marco de trabajo para el sistema. Se debe tener en cuenta los componentes localizados en la fase 2 para diseñar o determinar este marco.
4. Desarrollo e integración: Se integran los componentes y subsistemas. La integración es parte del desarrollo en lugar de una actividad separada.

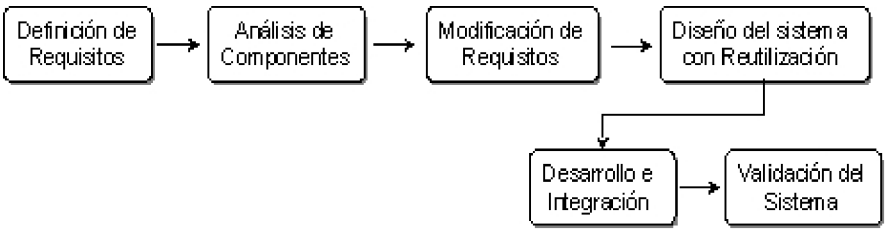


Figura 2.2 Desarrollo basado en reutilización de componentes

La Figura 2.2 muestra el flujo de acciones a realizar en el modelo de desarrollo orientado a la reutilización. En la presente tesis, se reflejará la utilización de cada una de estas fases durante el desarrollo del framework, que se determinó nombrarlo JBGames.

La estructura de comportamiento abstracta de los frameworks tiene la característica de poder ser utilizada como base para implementar aplicaciones del dominio en el que se desenvuelve. La implementación es realizada mediante la subclasificación de sus clases genéricas, seguida por la composición de clases concretas, utilizando el concepto de colaboración. De esta manera, componentes de software pueden ser reutilizados a través de herencia y composición[3].

La estructura de clases de ellos la componen cuatro tipos de métodos: *abstractos*, definen únicamente una interfaz sin implementación y deben ser implementados en cada subclase; *template*, definen un flujo de control común entre objetos del sistema; *hook*, definen una implementación por defecto que puede ser re-implementada en las subclases; y *base*, que representan un comportamiento completo y genérico de los sistemas pertenecientes al dominio del framework, y no puede ser reimplementado. El objetivo del desarrollo de un framework es construir una arquitectura de software que pueda ser personalizada en una aplicación concreta a través de los “hot spots” (clases abstractas, métodos *template* y *hook*) definidos[3, 31].

Una aplicación construida a partir de un framework reutiliza tanto la estructura de composición como la estructura de control. Solo las subclases necesitan ser definidas para una aplicación particular, implementando métodos abstractos heredados y reimplementando métodos hook heredados, que requieran cambios de comportamiento.

Los frameworks permiten la *modularidad* porque encapsulan los detalles de implementación detrás de una interfaz estable. Estas interfaces permiten la reutilización al definir componentes generales que pueden ser aplicados para crear nuevas aplicaciones. Análogamente, permiten

la extensibilidad a través de los métodos hook, que brindan a las aplicaciones la posibilidad de cambiar el comportamiento por defecto definido en él[3].

El framework propuesto, JBGames, está compuesto por una serie de clases y relaciones que abstrae las características de los juegos multi-jugador para móviles con conexión Bluetooth; permitiéndole a los desarrolladores configurar el diseño de las pantallas; agregar elementos al menú; incorporar, en caso necesario, interfases de usuario. En el caso de los juegos multi-jugador el programador podrá abstraerse de la comunicación (capa en la que se establece la misma, detección de dispositivos y servicios, entre otros), centrándose así, sólo en el desarrollo de la lógica del juego.

2.3 Análisis del Dominio

El análisis de dominio es el estudio del campo de acción en el que se desarrolla JBGames, utiliza las mismas técnicas de abstracción, composición, generalización y especialización que se emplea en el análisis OO, pero no trata aplicaciones en particular. Un modelo de dominio sirve como base para diseñar y construir objetos del ámbito en el que se enmarca.

Dado el tipo de aplicaciones seleccionado ─ juegos multi-jugador mediante conexión Bluetooth ─ las cuales se encuentran en un campo de acción amplio y versátil, con lógica de negocio compleja y muchas reglas implantadas por el framework para su desarrollo, las cuales varían con el tiempo, y van modificando a las actuales, y nutriéndose con otras nuevas, la idea central es modelar el dominio utilizando programación orientada a objetos, obteniendo así, un modelo del dominio, formado por objetos muy similares a la realidad, que se rigen según las características de las aplicaciones. El modelo de dominio describe con claridad el problema a los desarrolladores y sirve como base para diseñar y construir objetos del dominio[2].

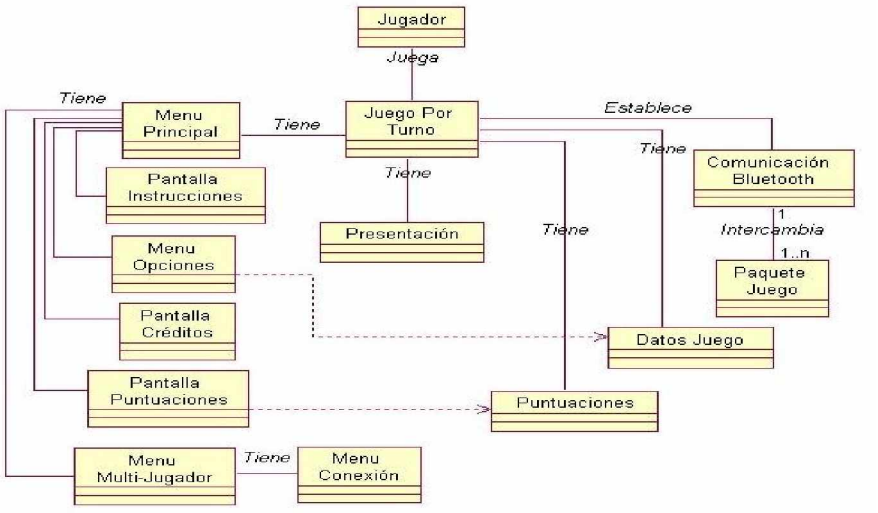


Figura 2.3 Modelo del Dominio

En la Figura 2.3 se reflejan todas las funcionalidades que poseen los juegos por turno que han de implementar los programadores, entre las que se encuentran las interfaces de usuario con las que se relaciona un jugador, los objetos encargados del intercambio de información posterior al establecimiento de la conexión y aquellos relacionados con los datos persistentes del juego. A continuación, un conjunto de requisitos a cumplir por cualquier juego del dominio

2.4 Requerimientos

En este segmento se enumera un conjunto de requerimientos del campo de acción de juegos para móviles con tecnología Bluetooth, a partir de la experiencia de los autores como desarrolladores de juegos en el dominio y de las entrevistas al grupo de desarrolladores de la empresa Procyon. Este conjunto de requisitos no representa a un juego en particular, sino que se incorporan los posibles requisitos generales de un juego del dominio.

2.4.1 Requerimientos funcionales

Una vez analizadas las características del sistema y las principales necesidades de los clientes se definió un listado de las capacidades o condiciones que el mismo debe cumplir, las mismas son:

1. **R1. Iniciar un Juego:** Iniciar una partida del juego seleccionado.
2. **R2. Mostrar Opciones:** Mostrarle la pantalla de menú Opciones al jugador.

3. **R3. Mostrar Instrucciones:** Mostrarle la pantalla de menú Instrucciones al jugador.
4. **R4. Mostrar Acerca de:** Mostrarle la pantalla de menú Acerca de al jugador.
5. **R5. Mostrar Puntuaciones:** Mostrarle la pantalla de menú Puntuaciones al jugador.
6. **R6. Gestionar Persistencia:** Guardar, modificar y actualizar todos los datos persistentes de un juego.
7. **R7. Crear Juego Servidor:** Iniciar una conexión servidora del juego seleccionado.
8. **R8. Iniciar Juego Servidor:** Iniciar una partida del juego seleccionado, actuando el jugador como Servidor del juego.
9. **R9. Unir a Juego Creado:** Unir al jugador a un juego servidor que está prestando servicios.
10. **R10. Chequear Estado de Conexión:** Chequea si existe o no conexión entre los dispositivos.
11. **R11. Intercambiar Información:** Envía y recibe los datos propios del juego.

2.4.2 Requerimientos no funcionales

Con el propósito de satisfacer al máximo las exigencias de los clientes así como la calidad de JBGames, se definió un listado de las propiedades o cualidades que el producto debe tener, las mismas se citan a continuación.

- **Restricciones del Diseño:** Conjunto de condiciones que deben cumplir los programadores al implementar JBGames. Las clases construidas deben tener el mayor nivel de abstracción posible, de manera que sea posible modificar, en cualquier momento, la forma de representación de los distintos elementos sin afectar al resto del diseño. Se implementará el framework utilizando J2ME con configuración CLDC 1.0, perfil MIDP 2.0, el API Bluetooth JSR-82 y la especificación JSR-185 para el almacenamiento de datos persistentes en el RMS.
- **Ayuda y documentación en línea:** JBGames debe estar acompañado de una documentación que especifique claramente las reglas para su uso. Para ello existe un Manual de Usuario en el que se define cada uno de los pasos a seguir para la implementación de las aplicaciones usando el framework. Además, en el código están comentadas las clases con el propósito de guiar a los programadores y posibilitarles una familiarización con el ambiente de desarrollo. Se realizará la documentación del código mediante el javadoc, según lo propuesto por el Estándares de Código de Java[32].

- **Requisito de Usabilidad:** Es la cualidad que tiene el producto de ser utilizado con facilidad para el fin al que ha sido destinado. JBGames debe estar orientado a programadores con conocimiento de la plataforma J2ME, y para su correcto uso debe poseer un Manual de Usuario y una documentación del código (Javadoc). Además, debe estar dividido en paquetes que agrupen las diferentes funcionalidades y ser nombrados, al igual que las clases, con calificativos que indiquen claramente su funcionalidad.
- **Requisito de Software:** JBGames debe seguir los Estándares de Código (Coding Guidelines) de la empresa Procyon Soluciones, se debe desarrollar utilizando el IDE (entorno de desarrollo integrado) Eclipse 3.1. Además, debe utilizar el API Bluetooth JSR-82 y el protocolo RFCOMM para la conexión de puerto serie.
- **Requisito de Confiabilidad:** JBGames debe garantizar el pausado de la aplicación cuando esta sea interrumpida guardando la configuración necesaria. Debe garantizar además, el correcto almacenamiento en el RMS y la liberación de recursos al interrumpirse la conexión para no afectar la disponibilidad de memoria.
- **Requisito de Portabilidad:** JBGames debe ser portable a los modelos de teléfonos móviles que soporten la especificación JSR-82, por lo que el diseño y la implementación no deben ser dependientes del dispositivo hardware en el que se implante. Esto es especialmente importante en cuanto a las dimensiones de la pantalla, muy variables de unos modelos a otros.

Tras haber capturado los principales requerimientos, se expondrá en el siguiente epígrafe el Modelo de Casos de Uso del Sistema, que comprende el diagrama de casos de uso y la descripción textual de cada uno de ellos. Su propósito primario es comunicar la funcionalidad al cliente o al usuario final; asegurando así una comprensión mutua de los requisitos y verificar la captura de los mismos.

2.5 Modelo de Casos de Uso del Sistema

2.5.1 Definición de los actores del sistema a automatizar

Los actores del sistema pueden representar el rol de una o varias personas, un equipo o un sistema automatizado. Éstos son externos al sistema y pueden intercambiar información con él o ser un recipiente pasivo de información. A continuación, en la tabla 2.1, se definen los actores con su respectiva descripción.

Tabla 2.1 Descripción textual de los actores del sistema

Actores	Justificación
Jugador	Este actor se encarga de teclear y modificar la información referente al juego. Hereda el rol del Actor Comunicación.
Reloj	Este actor ficticio inicia cada cierto tiempo el chequeo del estado de la conexión durante una partida del juego. Hereda el rol del Actor Comunicación.
Actor Comunicación	Este actor realiza todas aquellas acciones que se ejecutan durante una partida del juego.

Cada uno de estos actores inicializa un conjunto de casos de uso que muestran lo que sucede cuando él interactúa con el sistema para conseguir un objetivo determinado. El siguiente epígrafe refleja dicha interacción.

2.5.2 Diagrama de Casos de Uso del Sistema

El diagrama de casos de uso del sistema captura los requisitos funcionales que debe cumplir JBGames. Los casos de uso son descripciones de la funcionalidad del sistema ─ independientes de la implementación ─ que aportan un resultado de valor a los actores. La intención de este diagrama es proporcionar los requisitos funcionales que especifican qué debe hacer el producto. A continuación la representación del mismo.

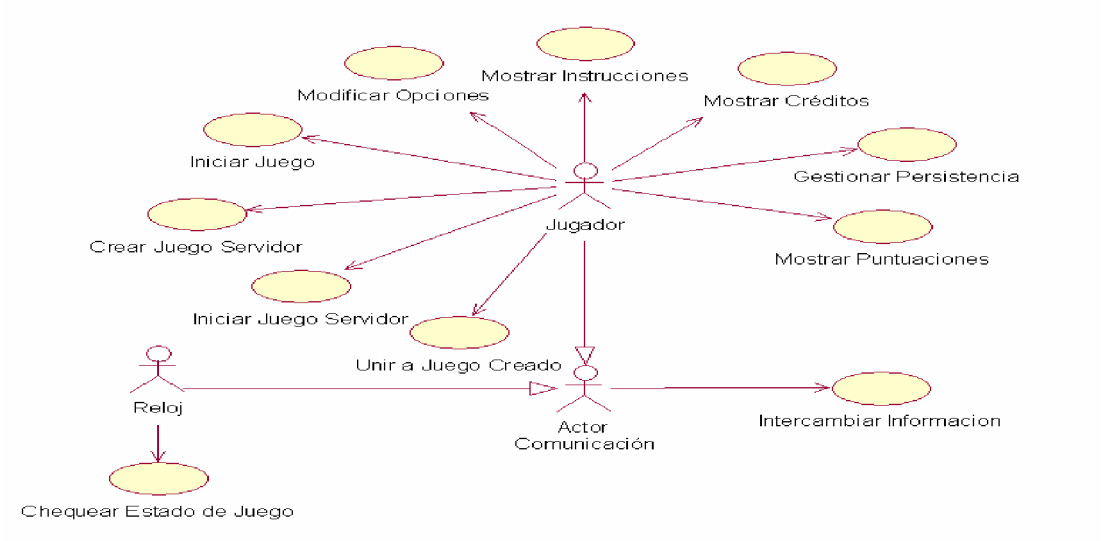


Figura 2.4 Diagrama de Casos de Uso del Sistema.

El Diagrama de Casos de Uso del Sistema de la Figura 2.4 muestra al Actor Comunicación como el que resume todas las funcionalidades de JBGames, especializado por los actores Reloj ─ que se encarga de verificar la conexión entre los partícipes del juego ─ y Jugador ─ que inicializa las interfaces gráficas de usuario y todas aquellas funcionalidades relacionadas con los datos propios de la partida del juego. La descripción de cada uno de los casos de uso reflejados en la anterior figura se encuentra en el Anexo 1.

2.5 Conclusiones

En el presente capítulo se abordaron las principales características de los frameworks, determinándose que clasificación era la idónea para el que se va a desarrollar y cómo se realizaría el mismo. Se ahondó en el análisis del dominio de las aplicaciones a implementar sobre él y se definieron los requisitos funcionales que influyeron en la elaboración del Modelo de Casos de Uso del Sistema. Se abordaron además, las restricciones fundamentales que debe cumplir JBGames para su correcto funcionamiento. En el próximo capítulo se detallarán las particularidades del sistema con la explicación de cada uno de los paquetes que lo integran.

3 CAPÍTULO
DISEÑO DE JBGames

3.1 Introducción

En este apartado se esboza el diseño de JBGames, con la intención de especificar cómo solucionar los requisitos funcionales. Se mencionan además, las principales funcionalidades de cada una de las clases que conforman los paquetes para una mejor comprensión del framework.

3.2 Características del perfil MIDP de J2ME

En el epígrafe 1.1 se mencionó que los juegos se desarrollarían sobre J2ME. MIDP es el perfil sobre la configuración CLDC, que le permite al desarrollador acceder a funcionalidades específicas de un dispositivo como interfaz gráfica de usuario, comunicaciones y almacenamiento persistente. Tanto la configuración CLDC, como el perfil MIDP aportan un conjunto de clases necesarias para desarrollar las funciones propias de control sobre un determinado ambiente de ejecución.

Tabla 3.1 Librerías del perfil MIDP

Paquete del MIDP	Descripción
javax.microedition.midlet	Clases de definición de la aplicación
javax.microedition.lcdui	Clases e interfaces que soportan componentes de interfaz gráfica de usuario (GUIs), específicos para las pantallas de los dispositivos móviles.
javax.microedition.rms	*Record Management Storage*. Soporte para el almacenamiento persistente del dispositivo
javax.microedition.io	Clases e interfaces de conexión genérica
java.io	Clases e interfaces de E/S básica
java.lang	Clases e interfaces de la *Máquina Virtual*
java.util	Clases e interfaces de utilidades estándar

Las aplicaciones realizadas utilizando MIDP reciben el nombre de MIDlet, el cual se define como una aplicación Java realizada con este perfil sobre la configuración CLDC. El paquete javax.microedition.midlet.* es el encargado de especificar el comportamiento de una aplicación ante el entorno de ejecución. Este paquete incluye dos clases principales, *MIDlet* y *MIDletstateChangeException*, la primera es la aplicación que define la estructura básica de la ejecución, mientras que la segunda recoge todas las excepciones o fallos en el cambio de estado. La clase *MIDlet* obliga a implementar tres métodos públicos, necesarios para la ejecución de la aplicación:

startApp(): es el primer método al que se llama cuando se inicia la ejecución de la aplicación o cuando ésta se reanuda.

pauseApp(): es el método que se invoca cuando la aplicación es detenida, normalmente por la recepción de una llamada entrante en el dispositivo móvil y debe definir las acciones que se deben llevar a cabo antes de pausar la ejecución.

destroyApp(): método invocado al finalizar la aplicación, se encarga de liberar y destruir todos los recursos y componentes utilizados durante la ejecución del MIDlet.

La aplicación debe heredar de la clase *MIDlet* para que le permita gestionar el control del software.

3.3 Diagrama de Clases del Diseño

El diagrama de casos de uso de JBGames (presentado en el capítulo anterior) muestra los casos de uso arquitectónicamente significativos para el posterior diseño del sistema. A partir de ellos se obtuvo el diagrama de clases del diseño que se proyectará a continuación.

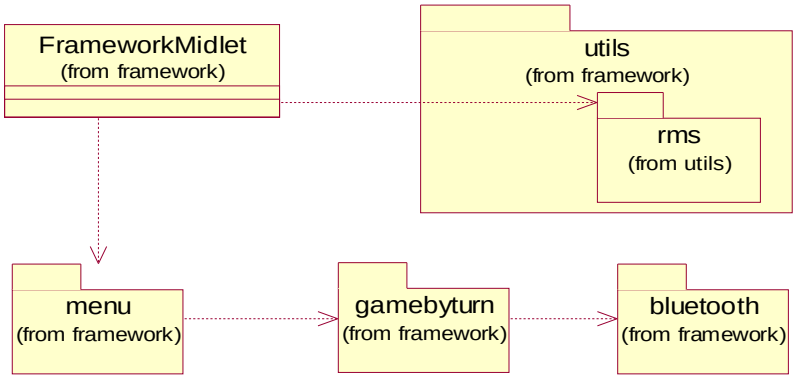


Figura 3.1 Diagrama de clases del diseño.

JBGames está conformado por cinco módulos o paquetes agrupados por su funcionalidad y una clase: *FrameworkMidlet*, que representa al MIDlet de la aplicación.

3.3.1 Clase *FrameworkMidlet*

Es la clase que se ejecuta al inicio. Controla por tanto la creación y destrucción de la aplicación, desde el momento de la presentación (clase *SplashScreen*), hasta el cierre completo, ya sea debido al final de la partida, o a que el usuario pulse el botón para abandonar el juego. También posee una instancia del objeto display, que es el encargado de mostrar información en la pantalla.

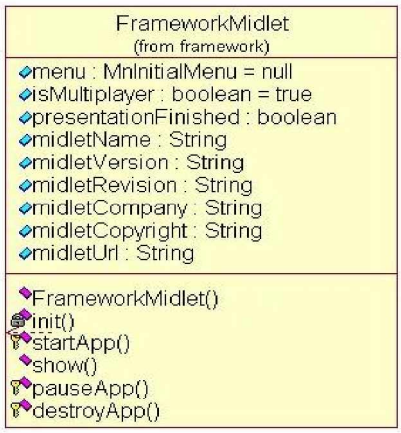


Figura 3.2 - Clase FrameworkMidlet

3.3.2 Framework.menu

Framework.menu: Conjunto de clases que conforman las posibles interfaces con el usuario accesibles desde el menú principal.

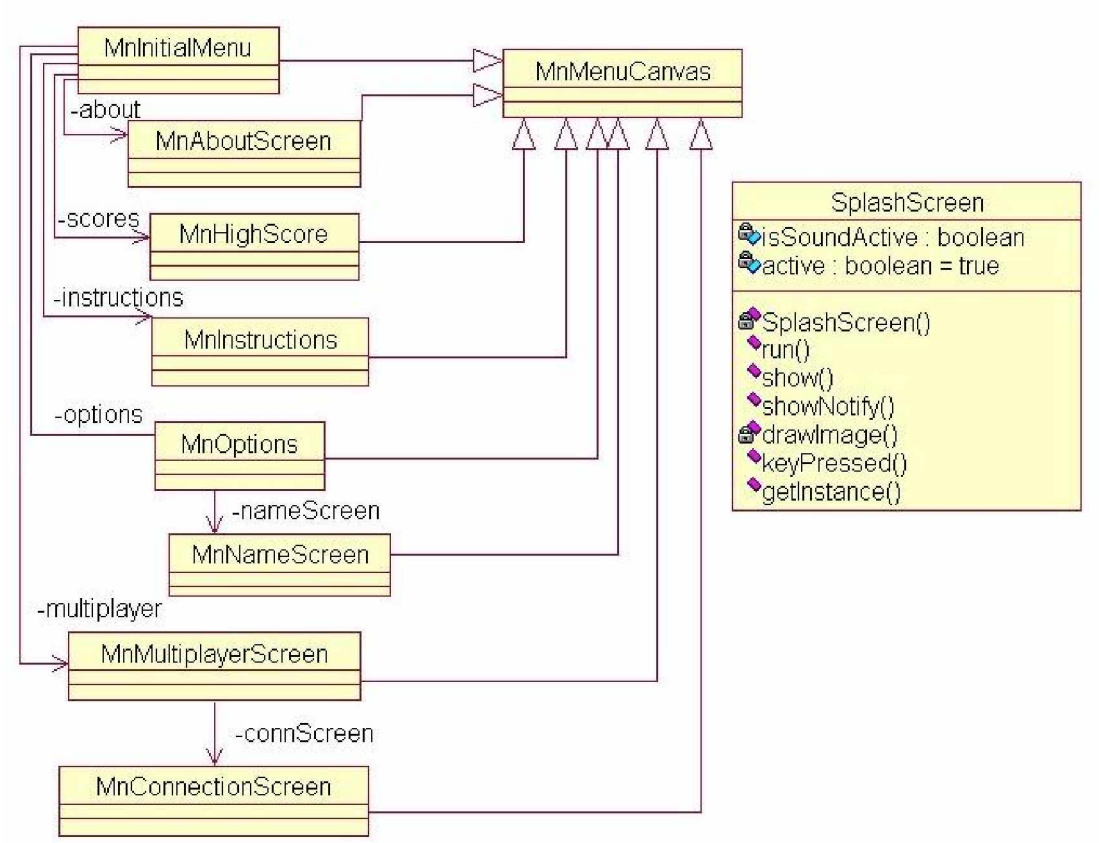


Figura 3.3 -. Diagrama de clases del paquete Framework.menu

En este paquete se encuentra la clase abstracta *MnMenuCanvas*, que extiende de *GameCanvas*, y de la cual heredan todas las otras interfaces de usuario. Es en ella donde se encuentran las principales funcionalidades de un menú, tales como el de añadir elementos a la pantalla mediante el método addItem(String text), al que únicamente hay que pasarle por parámetro el texto que representa el nombre de la opción a adicionar en caso de ser un menú de selección; ajustar el texto a las dimensiones de la pantalla en un menú informativo a través del método formatText(String txtChain); dibujar las imágenes de fondo, botones, barras de desplazamiento; operar el control de eventos de teclado donde se controla la lógica de selección a través de los métodos keyPressed(int KeyCode) y keyRepeated(int KeyCode) y otras relacionadas con la configuración de los elementos de un menú.

MnInitialMenu es la encargada de controlar el menú principal, en ella se definen, como en el resto de las pantallas que heredan de *MnMenuCanvas*, las funcionalidades básicas de un menú. Se encarga de manejar el evento - mostrar una pantalla o el juego en si -, cuando un

elemento del menú es seleccionado. Además, contiene todas las posibles pantallas asociadas al juego, correspondientes a las clases: *MnAboutScreen*, para mostrar información del juego; *MnInstructions*, para mostrar las instrucciones del juego; *MnHighScore*, para mostrar niveles y puntuaciones alcanzadas por un determinado jugador; *MnOptions*, para modificar las opciones del juego, quien a su vez, contiene una instancia de la clase *MnNameScreen* para cambiarle el nombre asociado al jugador y *MnMultiplayerScreen*.

La pantalla *MnMultiplayerScreen* permite crear un juego servidor o unirse a uno ya iniciado. En dependencia de la opción seleccionada, la clase *MnConnectionScreen* muestra un determinado escenario de rol cliente o servidor. Si la opción seleccionada en la pantalla *MnMultiplayerScreen* es Crear Juego Servidor, la pantalla *MnConnectionScreen* muestra los clientes conectados hasta que el servidor decida comenzar la partida. Si por el contrario, es Unir a Juego Creado, muestra los servidores que se han encontrado hasta que se seleccione alguno y a partir de ese instante, debe mostrarse el estado del servidor hasta que éste último comience la partida del juego. Se profundizará en el estudio de cómo se establece la conexión cuando se explique el paquete **Framework.bluetooth** y **Framework.gamebyturn**.

La clase *SplashScreen* es la pantalla de presentación del juego inicializada por la clase *FrameworkMidlet* al levantar la aplicación.

3.3.3 Framework.utils

Framework.utils: Conjunto de clases que facilitan determinadas utilidades al juego.

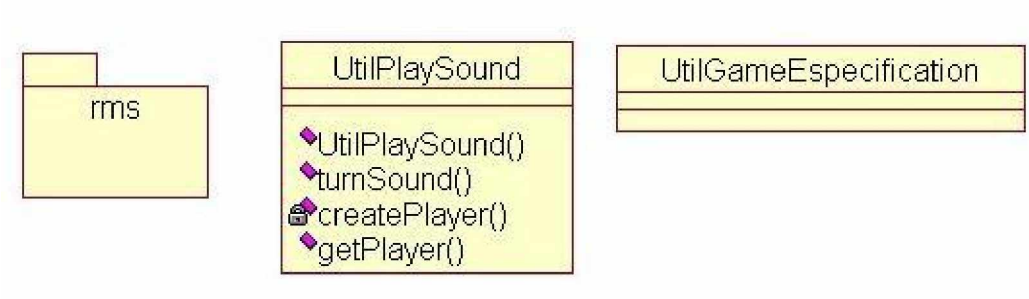


Figura 3.4 - Diagrama de clases del paquete Framework.utils

Incluye las clases *UtilGameEspecification* encargada de la configuración de los márgenes, sonidos, imágenes y comandos o textos a utilizar en cada una de las pantallas, incluyendo al propio juego; y *UtilPlaySound* destinada a todo lo referente al sonido en la aplicación. Además,

se encuentra el paquete **Framework.utils.rms**, en el que se encuentran un conjunto de clases encargadas de salvar y actualizar los datos.

3.3.3.1 Framework.utils.rms

MIDP proporciona un mecanismo a los MIDlets que les permite almacenar datos de forma persistente para su futura recuperación a través del paquete javax.microedition.rms. Dicho mecanismo es llamado Record Management System o RMS (Sistema de gestión de registros) y se compone de una serie de clases e interfaces que proporcionan soporte a un sistema simple de base de datos.

El objetivo del RMS es almacenar datos de tal forma que estén disponibles una vez que el MIDlet pare su ejecución. La unidad básica de almacenamiento es el registro (record) que será almacenado en una base de datos especial, denominada almacén de registros (record store). La información a guardar se almacenará en una zona de memoria del dispositivo dedicada para este propósito. La cantidad de memoria y la zona asignada para ello dependerán de cada dispositivo.

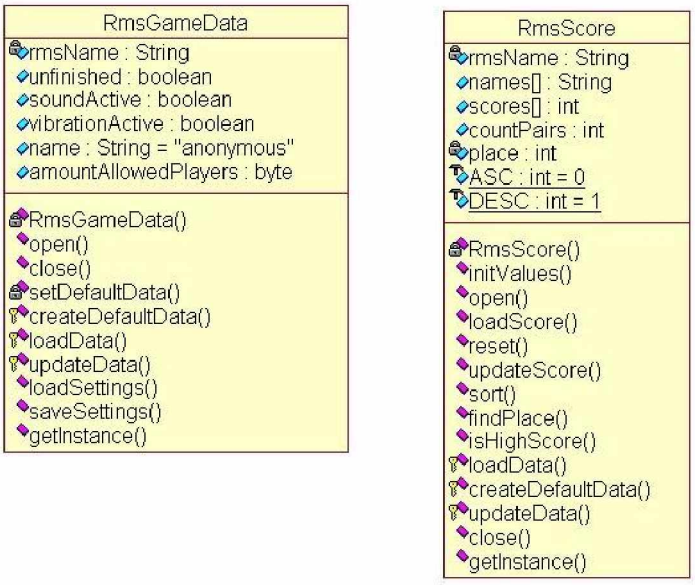


Figura 3.5- Diagrama de clases del paquete Framework.utils.rms

En **Framework.utils.rms** se encuentran las clases encargadas de actualizar los datos obtenidos del RMS, así como de guardar los datos del juego (*RmsGameData*) y las puntuaciones (*RmsScore*).

3.3.4 Framework.bluetooth

Framework.bluetooth: Conjunto de clases encargadas de establecer la comunicación.

En el epígrafe 1.2 se hizo alusión a los protocolos L2CAP y RFCOMM. Se ha determinado utilizar éste último porque a diferencia de L2CAP no es necesario mantener un tamaño fijo de paquetes a transferir pues los juegos por turno no intercambian grandes cantidades de información, ni es necesaria una alta velocidad de transmisión. Además, en Procyon Soluciones se había implementado con anterioridad - sobre la base de este protocolo -, un paquete de clases con el mismo propósito de **Framework.bluetooth** que permitía la conexión punto a punto; sin embargo, el actual conjunto de clases garantiza la conexión multipunto.

Cuando la tecnología inalámbrica Bluetooth se utiliza para sustituir al cable, se emplea el Perfil de Puerto Serie (SPP, Serial Port Profile) para el canal resultante orientado a conexión. Este perfil define cómo deben configurarse los dispositivos Bluetooth para emular una conexión a través de un cable serie utilizando RFCOMM. Este protocolo provee múltiples emulaciones de los puertos serie RS-232 entre dos dispositivos Bluetooth. Las direcciones Bluetooth de los dos puntos terminales definen una sesión RFCOMM. Una sesión puede tener más de una conexión, el número de conexiones dependerá de la implementación. Un dispositivo podrá tener más de una sesión RFCOMM en tanto que esté conectado a más de un dispositivo.

Una aplicación que ofrezca un servicio basado en el perfil de puerto serie es un servidor SPP. Una aplicación que inicie una conexión a un servicio SPP es un cliente SPP. Cliente y Servidor residen en los extremos de una sesión RFCOMM. El Servidor SPP registra su servicio en la base de datos de descubrimiento del servicio SDDB (Service Discovery Data Base), y como parte del proceso de registro, se añade un identificador de canal (channel identifier) al ServiceRecord por la implementación.

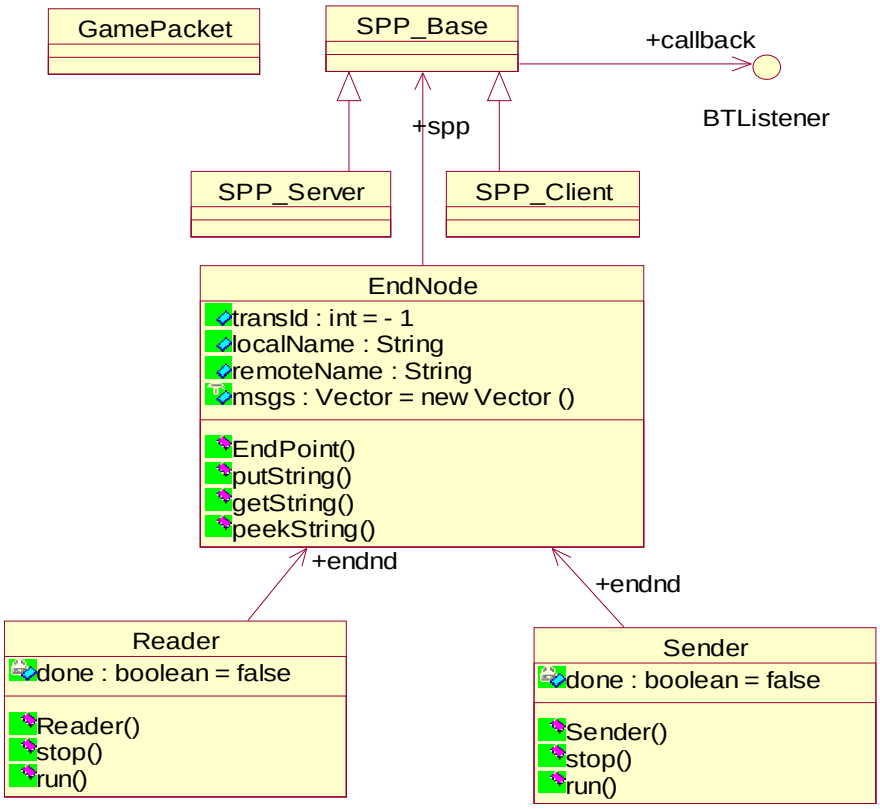


Figura 3.6 – Diagrama de clases del paquete Framework.bluetooth

En la Figura 3.6, *BTListener* es la interfaz de la cual tienen que implementar las clases controladoras e intermediarias entre la comunicación y el juego. En ella se definen los distintos eventos que pueden ser lanzados durante un juego y el método handleAction (String action, Object param2) que se ejecuta en la clase que implementa dicha interfaz cuando ocurre una determinada acción durante la comunicación.

En la clase abstracta *SPP_Base* se establece el estado de conectividad ilimitado en su método init(), se define el identificador único universal (UUID[8]) del servicio de juego, sus atributos, la instancia de la interfaz *BTListener*, los distintos tipos posibles de señales a intercambiar (pausa, reanudar, abandono, pérdida de conexión, unir a juego, mensajes específicos del juego, entre otros) y los métodos encargados de intercambiar la información correspondiente a cada una de las señales: sendPause(), para pausar el juego; sendResume(), para reanudar la partida después de haber sido interrumpida; sendTerminate(), para abandonar la partida del juego; sendLost(),

[8] Identificadores Únicos Universales: Son enteros de 128 bits que identifican protocolos y servicios.

para informar de la pérdida de conexión; sendString(String s), para el envío de mensajes propios del juego; entre otros. Estos métodos son los que proporcionan las funcionalidades de la comunicación en un juego sin la necesidad de que el programador conozca cómo ocurren estos procesos internamente. De la clase *SPP_Base* heredan *SPP_Client* y *SPP_Server* que son las encargadas de realizar las particularidades de cada rol.

En el epígrafe 1.2 se enumeraron las acciones a realizar por clientes y servidores para el establecimiento de una conexión. A continuación se describen las clases *SPP_Client* y *SPP_Server* que realizan dichas acciones.

La clase *SPP_Server* en su método init() crea una conexión servidora con los atributos especificados en su clase padre, se inicializan los servicios que se van a ofrecer definidos en *SPP_Base* y se registra en el SDDB; esto lo hace a través de la llamada al método Connector.open(String) el cual devuelve un objeto de la clase *StreamConnectorNotifier* que escucha las conexiones clientes que demanden este servicio.

Posteriormente, se obtiene un objeto de tipo ServiceRecord a través de la llamada del método getRecord(Connection) de la clase *LocalDevice*[9]; este objeto describe el servicio y cómo puede ser accedido por dispositivos remotos. En *SPP_Server* además, se inicia el hilo que espera por el establecimiento de la comunicación de un cliente en el método run() donde se llama al método acceptAndOpen() de la clase *StreamConnection* que posibilita múltiples conexiones de diferentes clientes. Una vez establecida esta comunicación, se crea un nuevo nodo (instancia de la clase *EndNode* que será explicada más adelante) y se incorpora a la lista de nodos conectados a este servidor. Esta lista facilita el intercambio de información entre los clientes y el servidor usando como teoría que un cliente no tiene comunicación directa con el resto de ellos sino a través del servidor.

También, se redefinen los métodos proporcionados por su clase padre encargados del intercambio de información: sendPause(EndNode endnd), sendResume(EndNode endnd) sendTerminate(EndNode endnd), sendLost(EndNode endnd), sendString(String s, EndNode endnd) para la difusión de la información entre clientes; un ejemplo de ello es si un cliente pausa el juego

[9] LocalDevice: representa al dispositivo local. Este objeto es el punto de partida de prácticamente cualquier operación que se realice sobre el API JSR-82.

por una llamada entrante, el servidor debe informar al resto de los clientes conectados (lista de nodos) del evento ocurrido.

En el método init() de la clase *SPP_Client* se obtiene una instancia de la clase *DiscoveryAgent* que es proporcionada por el API Bluetooth para descubrir servicios y dispositivos. A esta instancia se le ejecuta el método startInquiry() para la búsqueda de dispositivos, el cual requiere que la aplicación tenga especificado una función de escucha, por esta razón, se crea una clase *listener* que implementa la interfaz *DiscoveryListener* permitiéndole obtener las respuesta a los eventos del tipo *ServiceDiscovery* o *DeviceDiscovery*. Estas respuestas a dichos eventos son:

- deviceDiscovered(RemoteDevice remoteDevice, DeviceClass deviceClass): el cual se lanza cada vez que se encuentra un dispositivo. A la lista de dispositivos pendientes (pendingEndNodes) se le incorpora cada nuevo dispositivo encontrado, para realizar luego la búsqueda del servicio.

- inquiryCompleted(int transId): que se lanza cuando el proceso de búsqueda de dispositivos se ha completado o cancelado. Una vez lanzado este método se procede a realizar la búsqueda de servicios en las listas de dispositivos pendientes.

- servicesDiscovered(int transId, ServiceRecord[] svcRec): el cual se lanza cada vez que se encuentra un servicio con las características propias buscadas; luego de ser encontrado se incorpora a la lista de servidores (serviceRecordToEndNode) que será mostrada a los usuarios para la selección del juego servidor deseado.

- serviceSearchCompleted(int transID, int respCode): se lanza cuando el proceso de búsqueda de servicio en un dispositivo se ha completado o cancelado.

Una vez finalizada la búsqueda de dispositivos y servicios, se cuenta con la lista de servidores (*serviceRecordToEndPoint*) que será tratada por las clases controladoras entre la comunicación y el juego a través del método getDetectedServers().

La clase *SPP_Cliente* da la posibilidad de establecer la conexión con el servidor seleccionado por el jugador a través del método connect(int serverSeleted), e intercambia información con el servidor seleccionado a través de los métodos heredados.

EndNode es la clase que representa un nodo final en la comunicación. En el caso de la clase *SPP_Client* sólo va a tener un nodo final - el servidor - y en el caso de *SPP_Server*, tantos

nodos como jugadores conectados al juego. Esta clase tiene asociado un vector de los mensajes a intercambiar, además, tiene instancias a las clases *Reader* y *Sender*, encargadas del envío y recepción de señales y eventos, respectivamente. A su vez, tiene una instancia de la clase abstracta *SPP_Base* mediante la cual lanza el evento handleAction(String, Object, Object) de la clase *callback* (instancia de la interfaz *BTListener*) cuando su hilo servidor (*Reader*) reciba un evento determinado.

Dentro de esta librería se encuentra también la clase *GamePacket* que es la contenedora de los datos a enviar durante el juego, éstos son el tipo de evento, el emisor y el mensaje.

3.3.5 Framework.gamebyturn

Framework.gamebyturn: Conjunto de clases encargadas de controlar la secuencia lógica de los juegos por turno.

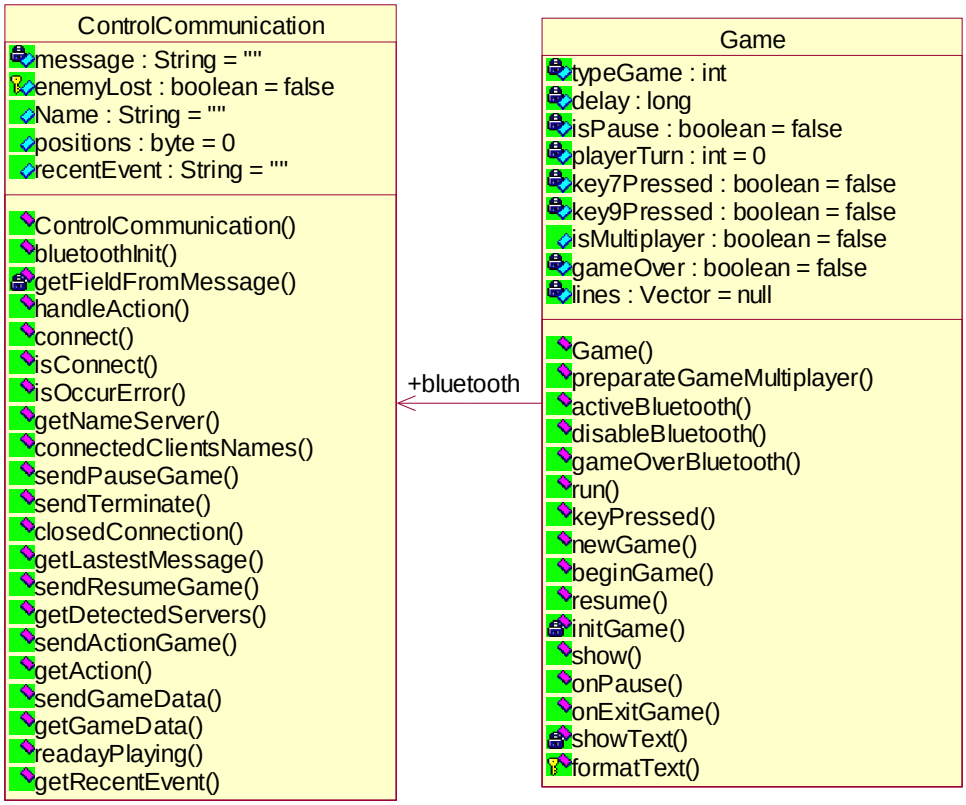


Figura 3.7 – Diagrama de clases del paquete Framework.gamebyturn

En este paquete solo se encuentran dos clases .La clase *Game* que es la destinada para la implementación de la lógica propia del juego. En ella se definen métodos y reglas que garantizan el establecimiento de la comunicación e intercambio de información en juegos multi-jugador, así como la relación entre el juego y las interfaces de usuario (*MnConnectionSreen*, *MnInitialMenu*) básicas para el inicio de un juego, pausado, reanudación y finalización de este. Estos métodos son:

- activeBluetooth(int type) : en dependencia del parámetro se le comunica a la clase controladora entre el juego y el paquete de comunicación (clase *ControlCommunication* que más adelante será explicada) que establezca un rol en la comunicación (servidor o cliente).

- preparateGameMultiplayer(): encargado de enviar los principales datos para comenzar un juego multi-jugador en caso de que el rol establecido en el juego sea el de servidor o esperar por los datos establecidos por el servidor para comenzar un nuevo juego en caso de que el rol fuese cliente.

- disableBluetooth(): encargado de liberar la memoria y poner la instancia de la clase controladora en vacío.

- gameOverBluetooth(): encargado de enviar el abandono de una partida cuando se está jugando un juego multi-jugador. Además, hace una llamada al método disableBluetooth().

- run(): método principal dentro del ciclo de vida de un juego. En él se proponen reglas a tener en cuenta para la detección de determinado evento en un juego multi-jugador, además se define donde se debe implementar y chequear las acciones propias de un juego de un solo jugador; esto garantiza el desarrollo de un juego de un solo jugador sin ningún inconveniente.

- keyPressed(int code): encargado de capturar los eventos de entrada de teclado del teléfono. En este método solo se establece como regla que al presionar la tecla número 0 esta indique pausa en el juego. El resto de las reglas son propias del juego realizado sobre el framework.

- newGame(): realiza una llamada al método de inicio del hilo principal o de control en un juego. Además, como regla se propone inicializar aquí las variables propias del juego desarrollado sobre el framework.

- beginGame(): encargado de inicializar el hilo principal del juego.

- resume(): encargado de enviar, si se está ejecutando un juego multi-jugador, la acción de reanudación del juego. Además, establece valor de falso a la variable local de chequeo de si el juego está en pausa o no para la reanudación del juego.

- initGame(): encargado de inicializar los valores para un nuevo juego o recuperar datos de un juego salvado (acción que se realiza a través del método LoadRMSData() propuesto como regla en JBGames para recuperar datos salvados)

- show(): encargado de mostrar el juego.

- onPause(): encargado de que cuando ocurra un evento que deba pausar el juego, establezca el valor de verdadero en la variable local de chequeo de si el juego está en pausa o no para su reanudación, además, si el juego es multi-jugador se informa que ha ocurrido un evento de pausa. Se muestra por último el menú principal (*MnInitialMenu*).

- onExitGame(): encargado de que cuando ocurra un evento de abandono del juego, se salven los datos de persistencia - acción realizada a través del método SaveRMSData() propuesto como regla en JBGames para salvar datos de juego no terminado -, esto solo ocurre si es un juego para un jugador. Si por el contrario el juego es multi-jugador, se envía el evento de abandono de juego.

- showText(String text,Graphics g, int sleepTime) y el método formatText(Graphics g, String txtChain) son los propuestos para facilitar la muestra de información de tipo mensajes para el usuario durante un tiempo y dentro de un formato determinado en dependencia de las dimensiones de la pantalla en la que se está mostrando.

Además, este paquete cuenta con la clase *ControlCommunication* para el procesamiento de eventos y acciones durante una partida del juego desempeñando su papel de clase intermediaria entre la clase *Game* y el paquete encargado de la comunicación (**Framework.bluetooth**). Estos eventos están muy relacionados con las facilidades

garantizadas por el paquete **Framework.bluetooth** en cuanto al envío de información - propia del juego o de eventos del mismo -, así como a la detección de servidores o listado de clientes conectados a un servidor, entre otros. A continuación, algunos de sus principales métodos:

- bluetoothInit(): encargado de invocar a los métodos de la clase que desempeña el rol seleccionado para ese juego (servidor o cliente).

- handleAction(String action, Object param2): evento que es lanzado cuando ocurre una determinada acción durante la comunicación. En él se propone como regla que se guarde el último evento recibido en la variable recentEvent. Si el evento es de tipo EVENT_RECEIVED se guarda en la variable message el valor del mensaje de juego recibido, pero si se recibe el resto de los posibles eventos se guarda en dicha variable el ejecutor de la acción. La variable message podrá ser accedida por la clase *Game* desde el método getLastestMessage().

- sendGameData(Object obj): encargado de enviar los datos de un juego multi-jugador del servidor a los clientes. Como regla, es aquí donde el programador debe conformar el formato o estructura de los datos a intercambiar.

- getGameData(): encargado de chequear la recepción de los datos de un juego enviados por el servidor a los clientes conectados a él para el comienzo del juego. Como regla, los datos deben recibirse en el mismo formato o estructura en que han sido enviados. Mientras el cliente no reciba los datos correctos del servidor retornará falso.

El paquete **Framework.gamebyturn** tiene relación directa con el paquete **Framework.bluetooth** a través de la última clase analizada para la implementación de aquellos juegos de un jugador que permitan desarrollar su versión multi-jugador. De esta manera, se cumple con la principal problemática que impulsa el desarrollo del presente trabajo: no se cuenta con una herramienta que permita implementar juegos multi-jugador a partir de su versión simple con el fin de lograr una mayor producción de éstos contenidos en el menor tiempo posible.

Estos paquetes proporcionan y proponen reglas y condiciones a tener en cuenta para desarrollar un juego multi-jugador con un riesgo mínimo de posibles errores a la hora de

establecer o mantener una correcta interrelación entre la lógica del juego, las interfaces de usuario y las comunicaciones.

3.6 Conclusiones

Como se había planteado anteriormente, el conjunto de clases presentado en el diseño de JBGames le permiten al desarrollador abstraerse totalmente del establecimiento de la conexión entre los dispositivos mediante los paquetes **Framework.gamebyturn** y **Framework.bluetooth**, le facilita la implementación de un grupo de funcionalidades -entre ellas las interfaces de usuarios- que solo debe transformar en dependencia de sus intereses, logrando así minimizar costos y esfuerzo. A continuación, un capítulo que describe cómo fue instanciado el framework JBGames con la realización de un juego de Dominó, lo cual comprueba la utilidad del mismo como primera solución al problema planteado.

4 CAPÍTULO
INSTANCIACIÓN

4.1 Introducción

Este capítulo muestra los diagramas de componentes y despliegue del sistema. Describe también, en líneas generales, los pasos a seguir en la instanciación del framework, brindando luego un ejemplo de cómo se ha implementado el juego de Dominó usando JBGames.

4.2 Modelo de Implementación de JBGames

El modelo de implementación describe cómo los elementos del modelo del diseño se implementan en términos de componentes y cómo estos se organizan de acuerdo a los nodos específicos en el modelo de despliegue.

4.2.1 Diagrama de Despliegue

El diagrama de despliegue muestra la distribución de los componentes de software desarrollados en el entorno donde será aplicada la solución. La siguiente figura muestra como será desplegado el framework en un teléfono celular para que entre ellos se pueda establecer una comunicación bluetooth para los distintos roles que puedan jugar los usuario.

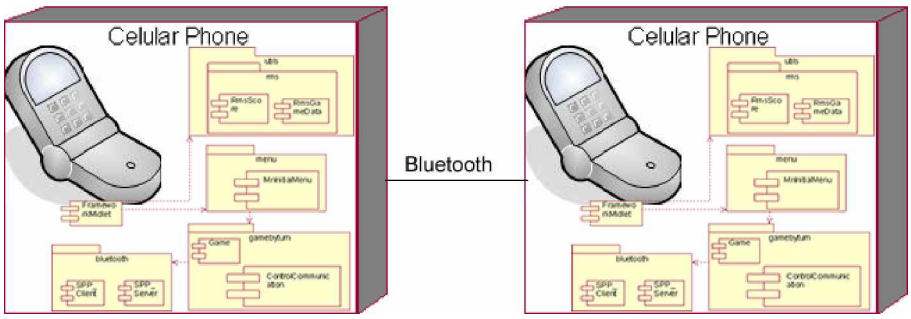


Figura 4.1 Diagrama de Despliegue

4.2.2. Diagrama de Componentes

El diagrama de componentes muestra las relaciones de dependencia entre las partes modulares del sistema desarrollado y encapsula la implementación. A continuación se muestran los componentes identificados y sus relaciones.

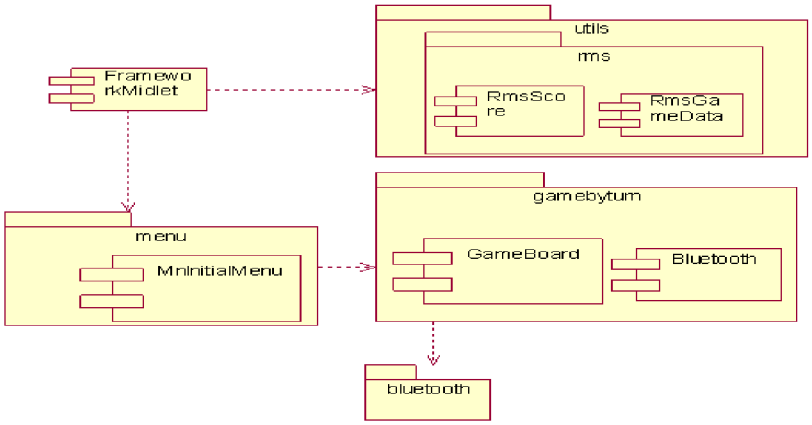


Figura 4.1 Diagrama de Componentes

4.3 Instanciación del framework

A la hora de instanciarse el framework, deben cumplirse ciertas reglas para su correcto uso. Seguidamente, se explicarán aquellas que son fundamentales y se concluirá con la descripción de la implementación del juego de Dominó en modo multi-jugador realizado sobre el framework JBGames.

A partir de que el programador incorpora JBGames a su proyecto J2ME, debe determinar qué pantallas va a utilizar: si va a agregar alguna nueva tiene que crear una clase que herede de *MnMenuCanvas* debido a que en ella se agrupan las funcionalidades básicas de un menú (estilo de letra, imagen de fondo, color de los textos, etc.). Si por el contrario, desea excluir alguna de las proporcionadas por defecto (*MnMenuInitial, MnOptions, MnIntructions, MnAboutScreen, MnMultiplayerScreen, MnConectionScreen*), deberá eliminarla del paquete y en la clase *MnMenuInitial* retirar el fragmento de código que hace referencia a ella en el método onSelect(int pos) y el asociado a esa pantalla. Luego, deberá definir en la clase *UtilGameSpecification* la configuración de textos, teclado, imágenes e icono que acompaña a la opción de menú seleccionada en una pantalla de tipo menú selección.

La lógica del juego podrá implementarla en la clase *Game* con el cumplimiento de las reglas propuestas que fueron explicadas en el subepígrafe 3.3.5. En *RmsGameData* incorporará las variables determinadas como persistentes del juego y agregará en *MnOptions* aquellas variables que pueden ser modificadas por el usuario (ejemplo de estas variables son la dificultad, calidad visual, etc.). Por último, se define en la clase *ControlCommunication* los datos del juego a enviar y recibir al iniciar una nueva partida multi-jugador; así como los métodos para recibir y enviar las jugadas (movimientos, acciones, situaciones y respuestas a éstas).

Tras haber abordado las principales normas para la correcta instanciación de JBGames, se describirá el primer ejemplo: el Juego de Dominó.

4.3.1 Descripción del juego de Dominó

El dominó es un juego de mesa en el que se emplean unas fichas rectangulares divididas en dos cuadrados que indican valores entre 0 y 6. El juego completo de fichas de dominó consta de 28 piezas, en cada una de ellas se representa un par de valores posibles. El objetivo del juego es lograr colocar todas las fichas en primer lugar, con la única restricción de que dos piezas sólo pueden colocarse juntas cuando los cuadrados adyacentes sean del mismo valor.

Antes de empezar, las fichas se colocan boca abajo sobre la mesa y se revuelven para que los jugadores las recojan en igual número cada uno (normalmente 7). Empieza la partida el jugador designado al azar, y continúa el jugador situado a su derecha. En las siguientes rondas, empezará el ganador de la ronda anterior, que podrá ubicar cualquier ficha, no tiene por qué ser doble. En caso de cierre, es decir, cuando a pesar de quedar fichas en juego ninguna pueda colocarse, ganará el jugador cuyas fichas sumen menos puntos.

Normalmente se juega a varias rondas. El jugador que gana una ronda, suma los puntos de las fichas de sus adversarios y/o pareja. Finalmente, gana el primer jugador o pareja que alcanza una puntuación fijada al principio de la partida.

4.3.2 Instanciación de JBGames: Juego de Dominó

El Dominó es un juego clásico por turno. Para comenzar su implementación se crea un nuevo proyecto J2ME llamado domino_MP_Bluetooth_v0.x al cual se le incorpora el paquete de JBGames. En él se hará uso de todas las pantallas facilitadas por defecto y de una que se

adicionará encargada de cambiar los nombres de los jugadores artificiales que participan en una partida de un solo jugador.

En la clase *UtilGameSpecification* se incorporan los textos en inglés o en el idioma deseado para los distintos elementos de los menús, instrucciones y créditos. Se definen igualmente en esta clase las imágenes de la presentación, fondo en las pantallas, así como el icono que acompaña al menú; además de los distintos sonidos que se reproducen en el juego ya sea en la presentación o durante el mismo. Se define también, el teclado por el cual responde el juego, en este caso el SonyEricsson K-750 que es el emulador de pruebas.

Ahora sólo queda realizar el desarrollo de la lógica del juego, para ello se implementan un conjunto de clases que a continuación se describen:

- *Cursor*: Encargada de seleccionar, cambiar y mover las fichas en el tablero.
- *Ficha*: Contiene los datos de una ficha (valores de ambos lados de una ficha, posición en el tablero, etc.)
- *Player*: Contiene el arreglo de fichas de un jugador, el nombre del jugador, posición en la pantalla, cantidad de puntos alcanzados entre otros datos propios de un jugador.

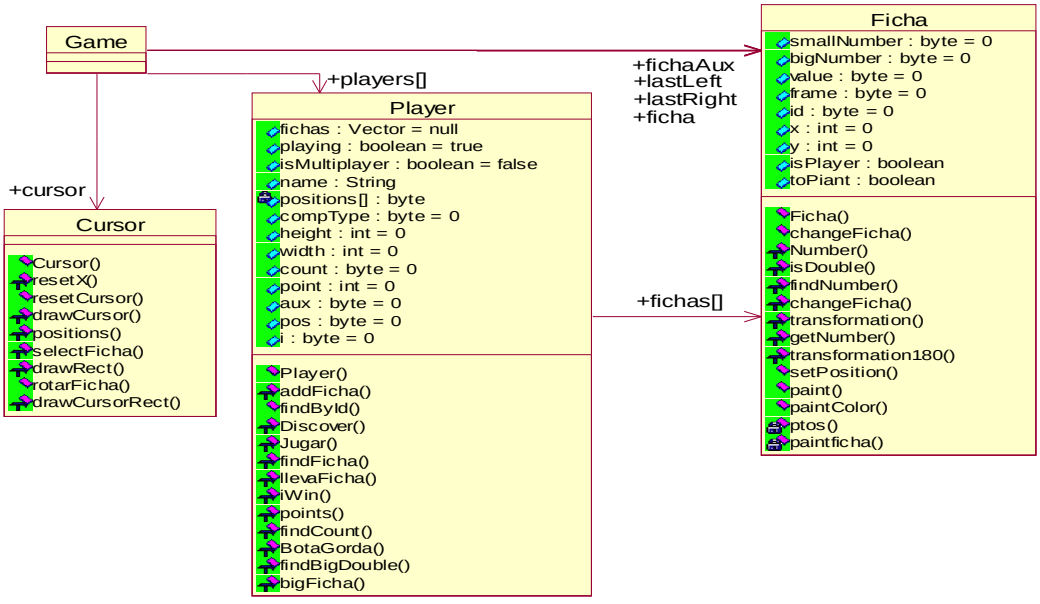


Figura 4.3 Diagrama de clases del juego de Dominó.

La lógica principal del juego se implementa en el método run() de la clase Game siempre cumpliendo con las reglas propuestas por el framework.

```
public void run() {
    long time1 = 0, time2 = 0;
    currenTime = System.currentTimeMillis();
    System.out.println("Inicio de hilo juego**********");
    while(gameOver == false){
      while(isPause)waitGame(50);
      time1 = System.currentTimeMillis();
      if(isMultiplayer){
        if(bluetooth.recentEvent == BTListener.EVENT_PAUSE ){
          showText(bluetooth.getUltimoMensaje()+" Paused Game", 1000);
        }
        else if(bluetooth.recentEvent == BTListener.EVENT_LEAVE ){
          gameOver =true;
          showText(bluetooth.getUltimoMensaje()+" Abort Game", 3000);
          midlet.show();
        }
        else if(bluetooth.recentEvent == BTListener.EVENT_LOST ){
          gameOver =true;
          showText(bluetooth.getUltimoMensaje()+" Lost Connection", 1000);
          midlet.show();
        }else{
          if(drawing)
            drawScreen();
          if(System.currentTimeMillis() - currenTime > speedGame)
            advanceGame();
          if(key != -1000)
            checkKeyPressed();
        }
      }else{
        if(drawing)
          drawScreen();
        if(System.currentTimeMillis() - currenTime > speedGame)
          advanceGame();
        if(key != -1000)
          checkKeyPressed();
      }
      time2 = delay -(System.currentTimeMillis() - time1);
      if(time2<0) time2 = 1;
      waitGame(time2);
    }
    System.gc();
    gameOverBluetooth();
    current = null;
  }
```

Figura 4.3 Método run() de la clase *Game*

En la Figura 4.3 se aprecia que se cumplió con la regla de chequeo de estado de los jugadores en el ciclo del juego. Posteriormente, se incorpora a la clase persistente *RmsGameData* un grupo de variables que son fundamentales a la hora de reanudar un juego no terminado para que permanezca su configuración; conjuntamente con aquellas variables que comprueban el estado de las puntuaciones hasta ese momento del jugador. Este último grupo de variables no están definidas en la clase *RmsScore* la cual es eliminada del proyecto porque en la estructura

del juego es necesario conocer las puntuaciones de los jugadores siempre que se termina una ronda (esto está dado por las reglas propias del juego). Además, permiten ver desde el menú principal dichas puntuaciones. A continuación las variables incorporadas a la clase *RmsGameData*:

```
public static Ficha ficha    = null;   // Fichas
public Player[]players = null;   // Player
/* Game */
public int turn = 0;
public int yvieWindow = 0;
public int fFicha = 0;
public int fLastRight = 0;
public int fLastLeft = 0;
public int fCFicha = 0;
public int fCSelec= 0;
public int nRight  = 0;
public int nLeft    = 0;
public int xRight  = 0;
public int yRight  = 0;
public int xLeft = 0;
public int yLeft = 0;
public int xCursor  = 0;
public int yCursor  = 1;
public int xSelect = 1;
public int posIndex = 0;
public int compCount= 3;
public static int countPtos  = 50;
public String [] names = {"Player1","Player2","Player3","Player4"};

public boolean playing;
public boolean adGame;
public boolean right;
public boolean left;
public boolean rightPos;
public boolean leftPos;
public boolean changeRight;
public boolean changeLeft;
public boolean select;
public boolean start;
public boolean selc;
```

Figura 4.4 Declaración de variables del juego

Estas variables son incorporadas a la clase *RmsGameData* encargada de la persistencia del juego.

Luego, se determinan los métodos para el intercambio de información del juego en la clase *ControlCommunication* y se incorporan los datos a intercambiar y recibir en el inicio de uno nuevo.

```
public void sendGameData(Object obj) {
    String msg = "10,"+RmsGameData.countPtos+","+RmsGameData.amountAllowedPlayers+",";
    Vector v = (Vector) obj;
    for(int i = 0; i<v.size(); i++)
        msg +=((Ficha)v.elementAt(i)).id+ ",";
    try {
        bluetooth.sendString(msg);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
public boolean getGameData() {
    if(message=="")
        return false;
    else{
        if (message.substring(0,2).equals("10"))
        {
            int ch = 0,pos=0;
            byte a = 0;
            for (int index = 3; index < message.length(); index++)
            {
                ch = message.indexOf(",",index);
                if (ch != -1){
                    if(index == 3){
                        Game.multiplayerCountPtos   = Integer.parseInt(message.substring(index,ch));
                        index = ch+1;
                        ch = message.indexOf(",",index);
                        Game.allowedPlayers     = Byte.parseByte(message.substring(index,ch));
                    }
                    else{
                        pos = Integer.parseInt(message.substring(index,ch));
                        ((Ficha)Game.fichas.elementAt(a++)).changeFicha((byte) pos);
                    }
                    index = ch;
                }
                else index = message.length();
            }
            return true;
        }
```

Figura 4.5 Fragmento de código

En la Figura 4.5 se observan las modificaciones y la incorporación de datos a intercambiar en el inicio de un juego multi-jugador.

4.3.3 Instanciación de JBGames: Resultados del Juego de Dominó

Después de haber implementado el juego de Dominó sobre JBGames se realizaron pruebas con varios emuladores SonyEricsson K-750 proporcionados por J2ME Wireless Toolkit 2.2 para chequear el correcto funcionamiento del framework. A continuación, se describirán algunos resultados:

- Se pudo observar que se mostraron y configuraron con calidad las distintas pantallas del juego, ya sean de selección (Domino) o de solo texto explicativo (Instructions)

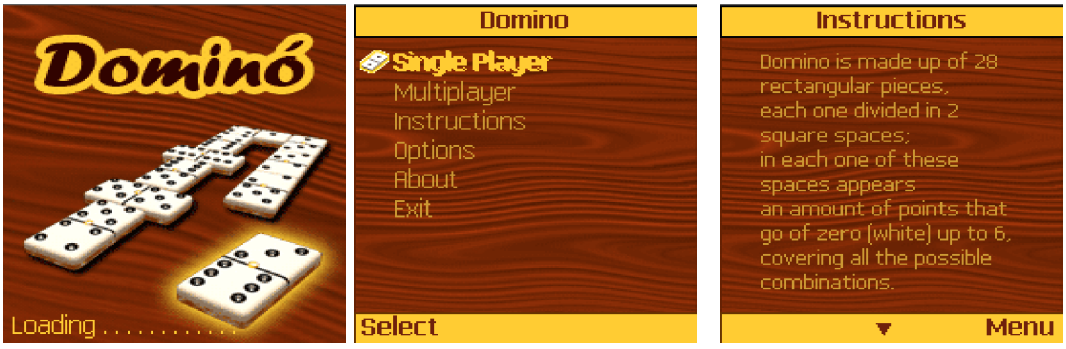


Figura 4.6 Pantallas de los emuladores (1).

La Figura 4.6 muestra la pantalla de presentación del Juego, el menú principal y la pantalla de instrucciones.

- Se apreció el resultado de la creación de un juego servidor. Proceso que se muestra en las pantallas de espera del servidor por sus clientes y como éstos se incorporan al juego.

Figura 4.7 Pantallas de los emuladores (2).

En la figura anterior se reflejan la pantalla de selección de juego multi-jugador, que una vez seleccionado la creación de un juego servidor muestra la pantalla de espera por conexión de clientes en el dispositivo servidor.

- Se apreció cómo un cliente se unió a un juego servidor. Proceso que se muestra en la pantalla de selección del servidor y la pantalla de espera del servidor para comenzar un juego.

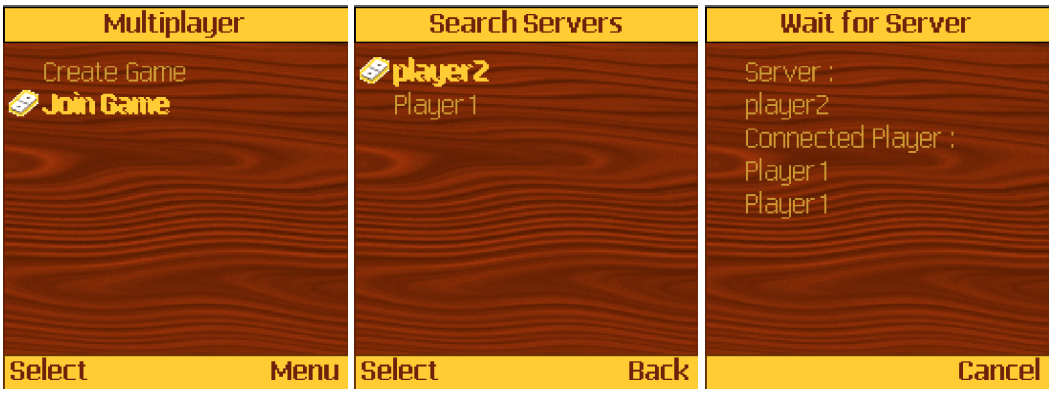


Figura 4.8 Pantallas de los emuladores (3)

La imagen anterior refleja cómo a un cliente que se une a un juego se le muestra la pantalla de espera por el comienzo del juego con el estado del servidor y de los clientes asociados a él.

- Se apreció cómo se inicia un juego servidor. Proceso mostrado en las pantallas de inicio del servidor y en el juego ya comenzado después de haber intercambiado los datos de inicio de un juego.

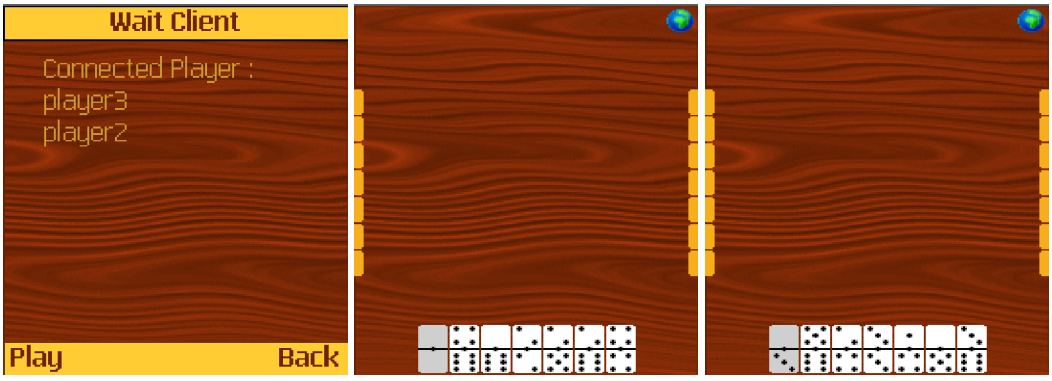


Figura 4.9 Pantallas de los emuladores (4)

La Figura 4.9 refleja la pantalla de selección del servidor para iniciar un juego a través del botón Play y a continuación, las pantallas del Juego tanto en el servidor como en el cliente.

- Se apreció cómo se muestran los mensajes de aviso de ocurrencia de un determinado evento en un juego: pausado del juego, una jugada o el abandono de algún jugador.

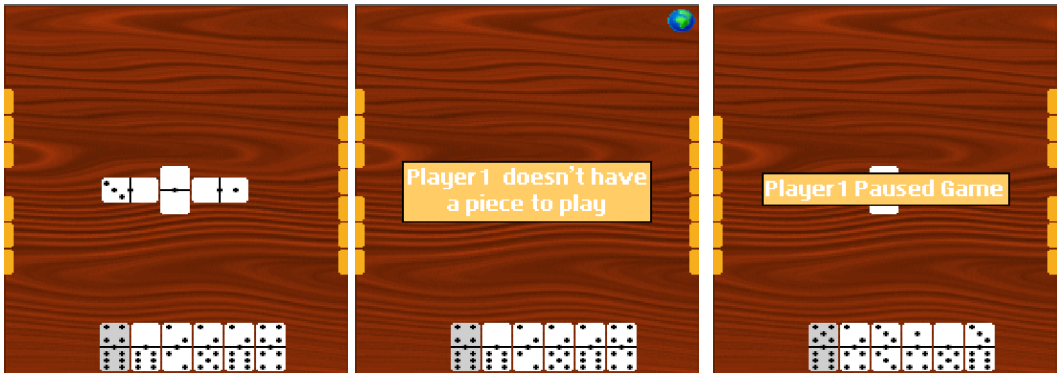


Figura 4.10 Pantallas de los emuladores (5)

La Figura 4.10 muestra tres pantallas en diferentes instantes del juego: intercambio de jugadas, mensaje indicador de que un jugador no puede jugar por no tener pieza y mensaje de recepción de un pausado de un jugador.

4.4 Conclusiones

En este capítulo se presentó el Juego de Dominó como prueba de la instanciación de JBGames. Se explicaron las reglas a seguir para el correcto uso del framework, se mostraron fragmentos del código del juego con el fin de que exista una mejor comprensión de las facilidades que ofrece el framework y los resultados obtenidos en el juego.

5 CAPÍTULO
ESTUDIO DE FACTIBILIDAD

5.1 Introducción.

El estudio de la factibilidad es la herramienta imprescindible para conocer la totalidad de los gastos en que incurrirá la empresa al incorporar el nuevo sistema, así como el incremento de los costos que demandará su funcionamiento luego de la implementación. En este capítulo se valora la estimación de esfuerzos y costos en la realización del framework, utilizándose el Análisis de Puntos de Casos de Uso para un estudio más exhaustivo de los beneficios tangibles e intangibles que aporta el desarrollo de la aplicación con respecto al costo de la realización del mismo.

5.2 Análisis de Puntos de Casos de Uso

La estimación mediante el análisis de Puntos de Casos de Uso es un método propuesto originalmente por Gustav Karner de Objectory AB, y posteriormente refinado por muchos otros autores. Se trata de un método de estimación del tiempo de desarrollo de un proyecto mediante la asignación de "pesos" a un cierto número de factores que lo afectan, para finalmente, contabilizar el tiempo total estimado para el proyecto a partir de esos factores[33].

5.3 Cálculo de Puntos de Casos de Uso sin ajustar

Éste cálculo se realiza a partir de la siguiente fórmula: UUCP = UAW + UUCW donde, UUCP son los puntos de Casos de Uso sin ajustar, UAW es el factor de Peso de los Actores sin ajustar y UUCW es el factor de Peso de los Casos de Uso sin ajustar. Las tablas que seguidamente se muestran están relacionadas con éstos valores.

Tabla 5.1 Factor de Peso de los Actores sin ajustar (UAW)

Tipo de actor	Descripción	Factor de peso	Actores	Total
Simple	Sistema con sistema a través de interfaz de programación.	1	0	0

Medio	Sistema con sistema mediante protocolo de interfaz basada en texto.	2	0	0
Complejo	Persona que interactúa con el sistema mediante interfaz gráfica.	3	1	3

UAW = Σ (Factor * Actores) = 3

Tabla 5.2 Factor de Peso de los Casos de Uso sin ajustar (UUCW)

Tipo de CU	Descripción	Peso	Cantidad de CU	Total
Simple	El caso de uso tiene de 1 a 3 transacciones.	5	3	15
Medio	El caso de uso tiene de 4 a 7 transacciones.	10	2	20
Complejo	El caso de uso tiene más de 8 transacciones.	15	0	0

UUCW = Σ (Factor * Actores) = 35

UUCP = UAW + UUCW = 38

5.4 Cálculo de Puntos de Casos de Uso ajustados

Éste cálculo se realiza a partir de la fórmula: UCP = UUCP * TCF * EF donde, UCP son los puntos de Casos de Uso ajustados, UUCP son los puntos de Casos de Uso sin ajustar, TCF es el factor de complejidad técnica y EF es el factor de ambiente. A continuación, las tablas relacionadas con éstos valores.

Tabla 5.3 Factor de complejidad técnica (TCF)

Factor	**Descripción**	**Peso**	**Valor asignado**	**Total**
T1	Sistema distribuido	2	2	4
T2	Tiempo de respuesta	1	2	2
T3	Eficiencia del usuario final	1	4	4
T4	Funcionamiento Interno complejo	1	1	1
T5	El código debe ser reutilizable	1	5	5
T6	Facilidad de instalación	0,5	5	2,5
T7	Facilidad de uso	0,5	3	1,5
T8	Portabilidad	2	5	10

T9	Facilidad de cambio	1	4	4
T10	Concurrencia	1	0	0
T11	Incluye objetivos especiales de seguridad	1	3	3
T12	Provee acceso directo a terceras partes	1	0	0
T13	Se requieren facilidades especiales de entrenamiento de usuarios	1	3	3

TCF=0.6+0.01* Σ (Peso * Valor)= 1

Tabla 5.4 Factor de ambiente (EF)

Factor	Descripción	Peso	Valor asignado	Total
E1	Familiaridad con el modelo de proyecto utilizado	1,5	2	3
E2	Experiencia en la aplicación	0,5	4	2
E3	Experiencia en la orientación a objetivos.	1	4	4
E4	Capacidad del analista líder.	0,5	2	1
E5	Motivación.	1	5	5
E6	Estabilidad de requerimientos	2	3	6
E7	Personal Part–Time	-1	0	0
E8	Dificultad del lenguaje de programación	-1	2	-2

EF=1.4-0.03* Σ (Peso * Valor)= 0.83

UCP = UUCP * TCF * EF = 31.54

5.5 De los Puntos de Casos de Uso a la estimación del esfuerzo

Para determinar el valor del Factor de Conversión se contabiliza cuántos factores de los que afectan al Factor de ambiente están por encima del valor medio, es decir 3, para los factores E7 y E8. Si los factores involucrados no sobrepasan el valor 2, como es el caso del presente análisis, se toma como Factor de Conversión 20 horas-hombre/Punto de Casos de Uso, lo cual indica que un Punto de Caso de Uso toma 20 horas-hombre.

El esfuerzo en horas-hombre se estima por la ecuación E = UCP * CF con un valor de E = 630.8. A este valor del esfuerzo se le adicionan las estimaciones de esfuerzo del resto de las actividades relacionadas con el desarrollo del framework cuantificadas en la siguiente tabla.

Tabla 5.5 Esfuerzo de las actividades

Actividad	**Porcentaje %**	**Horas-Hombres**
Análisis	20%	315,4
Diseño	20%	315,4
Implementación	40%	630,8
Pruebas	10%	157,7
Sobrecarga (otras actividades)	10%	157,7
Total	100%	1577

Resultados alcanzados

Esfuerzo Total (Horas-Hombre)	ET1 = 1577
Esfuerzo Total (Meses-Hombre)	ET2 = 8,21354167
Tiempo de Desarrollo	TD ≈ 4 meses
Salario Promedio	SM = $100
Cantidad de Hombres	CH = 2
Costo Hombre-Mes	CHM = $200
Costo Total	CT = 1642,70833

Teniendo en cuenta que un mes tiene aproximadamente 240 horas, un hombre puede realizar el proyecto en 8 meses (ET2). El framework es desarrollado por 2 personas con un salario promedio de $100 cada uno, por lo que el costo del proyecto es de casi $1643 en un plazo de 4 meses.

5.6 Análisis de los costos y beneficios tangibles e intangibles

JBGames es un producto de y para la entidad Procyon Soluciones. Este framework está destinado a facilitar la implementación de juegos multi-jugador con conexión bluetooth con el fin de minimizar esfuerzo y costo. Al realizarse por estudiantes, con la infraestructura tecnológica de la Universidad de las Ciencias Informáticas, no acarreó gastos adicionales por concepto de salario para la empresa. Además, todo el software utilizado en su implementación ha sido libre, motivo por el cual no requieren del pago de licencias.

El tiempo de desarrollo de estos contenidos no depende de la utilización de JBGames sino de su complejidad, pues cada uno tiene características propias que complican la programación de la lógica del juego. Aunque no es menos cierto que por la diversidad de diseños que interferían en la integración del componente de comunicación y la adaptación de este último los juegos podían tardar mucho más que los que hoy se implementan siguiendo las normas propuestas en el Manual de Usuario de JBGames. Esto era debido a que los juegos no estaban diseñados tanto para un jugador como para multi-jugador y además, los programadores estaban obligados a realizar un estudio del protocolo de comunicación antes transformar dicho componente. Indiscutiblemente, desarrollar un juego sobre una herramienta que garantiza las interfaces de usuario más comunes, la comunicación en caso de que el juego sea multi-jugador, el almacenamiento de datos propios del juego – si es para un jugador –, entre otras funciones, disminuye el tiempo de desarrollo de estos contenidos.

5.7 Conclusiones

En este capítulo se estimaron los costos en el desarrollo del framework y el esfuerzo de las personas que lo implementaron. Se realizó un análisis de los beneficios que reporta JBGames y se compararon los aportes de este framework en el desarrollo de las aplicaciones que contribuyen a mejorar el esfuerzo y tiempo de desarrollo de los programadores de Procyon Soluciones.

CONCLUSIONES

En la presente tesis, se profundizó en el estudio del protocolo de comunicación Bluetooth y en el diseño de una arquitectura básica para la implementación de juegos para móviles. JBGames les proporciona a los programadores de Procyon Soluciones una herramienta capaz de ofrecer un marco de trabajo sobre el cual realizar juegos multi-jugador a partir de aquellos que son ejecutados por un solo jugador. Los componentes del framework responden a la necesidad de obtener un mayor nivel de reuso por medio de un determinado diseño orientado a ese fin.

Los resultados obtenidos con la instanciación del mismo muestran la posibilidad de mejorar la productividad de juegos multi-jugador sobre J2ME con conexión Bluetooth. Estos resultados son ratificados al observarse que:

- Se implementaron un conjunto de interfaces de usuario mediante las clases *MnMenuInitial, MnOptions, MnIntructions, MnAboutScreen, MnMultiplayerScreen, MnConectionScreen* que pueden ser modificadas o eliminadas; incluso se puede agregar alguna otra en dependencia de lo deseado por el programador.
- Fue implementado un componente encargado de la comunicación Bluetooth. Dicho componente realiza la búsqueda de dispositivos y servicios, crea un dispositivo servidor que ofrece un servicio de juego, establece la comunicación entre clientes y servidores garantizando también, el envío de mensajes a partir de la ocurrencia de determinados eventos.
- Se desarrolló la clase *ControlComunication* que controla el flujo de información entre el paquete de comunicación y la lógica del juego.
- Se implementó un paquete para el almacenamiento de datos persistentes en aquellos juegos que deban guardar partidas que puedan ser reanudadas.
- Garantiza la conexión multipunto, por lo que pueden coexistir más de dos jugadores.
- Se creó una documentación del código (Javadoc) y un Manual de Usuario que explican detalladamente cada funcionalidad y los pasos a seguir para su correcto uso.

Con el desarrollo de esta tesis se profundizó en el conocimiento de estas tecnologías, aportando una gran experiencia en la teoría de implementación de la comunicación para juegos.

RECOMENDACIONES

Una posible ampliación de este trabajo consiste en la posibilidad de implementarle a JBGames un conjunto de funcionalidades que permitan desarrollar los juegos de acción para los cuales ha de tenerse en cuenta la latencia, la cantidad de información a intercambiar, entre otros aspectos de importancia.

BIBLIOGRAFÍA

1. Peláez, A., *Tendencias Software Móvil.* Noviembre 2006.
2. Rosanigo, I.Z.B., *Maximizando reuso en software para Ingeniería Estructural. Modelos y Patrones*, in *Facultad de Informática*. 2000, Universidad Nacional de La Plata: La Plata, Argentina.
3. Avancini, H.H., *FraMaS: Un Framework para Sistemas Multi-agente basado en Composición*. Facultad de Ciencias Exactas. Junio 2000: Universidad Nacional del Centro de la provincia de Buenos Aires.
4. España, U.d.c.d. (2007) *Telefonía Móvil: Servicios de atención al cliente*.
5. Martínez, E., *La evolución de la telefonía móvil (La guerra de los celulares).* in *Revista RED*. Mayo 2001.
6. Aubareda, D., *Juegos multijugador, el deseo de competir contra otros.* 2004.
7. Mary Luz Chicangana, J.F.B.P., Javier Alexander Hurtado Guaca, *Desarrollo de aplicaciones Bluetooth utilizando el API Java JSR-82.* 2004.
8. Brieba, A.G., *JSR-82: Bluetooth desde Java.* 2004.
9. Juan P. Pece, C.F.a., Carlos J. Escudero, *Bluesic: Sistema de información contextual para terminales móviles basado en tecnología Bluetooth*, D.d.E.y.S.U.d.L. Coruña, Editor. 2005, Junta de Galicia.
10. Tablado, A.M. (2007) *Seguridad Mobile*.
11. Fernández, A.S., *Desarrollo de juego multijugador wireless para dispositivos móviles con J2ME*. 2004, Universidad de Deusto.
12. Juzgado, P.D.B., *Java 2 Micro Edition Soporte Bluetooth*

2004, Universidad de Carlos III: Madrid, España.

13. Romero. A, T.J., Baños.J, *Diseño e implementación de aplicaciones bluetooth*, D.d.I.d.C.U.d. Málaga, Editor. 2003, Centro de Tecnología de las Comunicaciones (CETECOM), S.A.
14. Tablado, A.M., *Especificación Bluetooth*, in *Seminario de Tecnologías Wireless*. 2007, Seguridad Mobile: Universidad de Nebrija, Madrid.
15. Ahues, R.A.S., *Un sistema para el reuso de componentes en videojuegos de acción bi-dimensionales*, in *Departamento de Ciencia de la Computación*. 2000, Pontificia Universidad Católica de Chile Santiago de Chile.

16. Lidia Fuentes, J.e.M.T.y.A.V., *Desarrollo de Software Basado en Componentes*. 2005, Dept. Lenguajes y Ciencias de la Computación. Universidad de Málaga.
17. Juan, F.J.M., *Guía de construcción de software en JAVA con Patrones de Diseño*, in *Departamento de Ingeniería Técnica en Informática*. 2000, Universidad de Oviedo: Oviedo.
18. Sergio Diaz Rubera, J.G.A., José María Sobrinos García, *Vega Solaris: The Remake*, in *Dpto. Sistemas Informáticos y Programación*. 2005-2006, Facultad de Informática. Universidad Complutense de Madrid: España.
19. Nokia, F., *Games over Bluetooth: Recommendations to Game Developers*. 2003, Nokia.
20. Motorola, I., *Freescale Semiconductor, Inc.* 2002.
21. María A. Pérez de Ovalles, L.E.M.M., *Sistemas de información II Teoría*. 2004, Departamento de procesos y sistemas. Universidad Simón Bolívar.
22. Pedro Álvarez, J.Á.B., *Conceptos y estándares de arquitecturas orientadas a servicios Web.* 2006/2007, Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza.
23. Ángel I. Rodríguez Alcalde, C.C.r., *Arquitectura de desarrollo: Framework del CSIC v.2.0*, C.T.d. Informática, Editor. Febrero 2006.
24. Inclan, S.D.R., *Frameworks De Aplicaciones Orientadas a Objetos.* 2006.
25. Juan Francisco Valdés Gayo, M.R.F., *TRINX, Entorno para el Desarrollo de Juegos Persistentes Multijugador.* The Human Communication and Interaction Research Group, Department of Computer, 2003.
26. Andrés Vignaga, D.P., *Enfoque Metodológico para el Desarrollo Basado en Componentes*, in *Instituto de Computación de la Facultad de Ingeniería*. 2007, Universidad de la República: Uruguay.
27. Molina, J.G., *El Lenguaje Unificado de Modelado, UML*. 2004-2005, Departamento de Informática y Sistemas, Universidad de Murcia.
28. Apexnet (2005) *Comparación de Herramientas de modelado UML: Enterprise Architect y Rational Rose*. Apexnet software factory
29. Lucena, M.E.M.y.C.J.P.d., *El Desarrollo del Framework Orientado al Objeto.* 2001.
30. Letelier, P., *Proceso de Desarrollo de Software*. 2004, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica: Valencia, España.
31. Codenie, W. (1997) *Communication of the ACM* Custom Applications to Domain-Specific Frameworks

32. Ambler, S.W., *Writing Robust Java Code.The AmbySoft Inc. Coding Standards for Java v17.01d*. 2000: Copyright 1998-1999 AmbySoft Inc.

33. Peralta, M., *Estimación del esfuerzo basada en Casos de Uso.*, C.d.I.d.S.e.I.d.C. (CAPIS), Editor. 2004, Escuela de Postgrado. Instituto Tecnológico de Buenos Aires.

ANEXOS

Anexo 1: Descripción detallada de los casos de uso

Tabla 1.1. Descripción detallada del CUS: Iniciar un Juego

Nombre del Caso de Uso	Iniciar un Juego	
Actores	Jugador	
Propósito	Iniciar el juego seleccionado	
Resumen	El caso de uso se inicia cuando un Jugador selecciona en su móvil el juego para su ejecución.	
Referencias	R1.	
Precondiciones		
Poscondiciones	Es mostrada la pantalla de presentación al usuario y enviado hacia el menú principal del juego.	
Curso Normal de los Eventos		
Acciones del Actor		**Respuesta del Sistema**
1- El Jugador selecciona el juego a ejecutar.		1.1. El sistema deberá cargar los datos para comenzar un juego.
		1.2 El sistema deberá mostrar la pantalla de presentación.
		1.3 El sistema deberá mostrar el menú principal.
Puntos de Extensión:		
Prioridad:	Primaria	
Prototipo de Interfaz:	 	

Tabla 1.2. Descripción detallada del CUS: Mostrar Opciones

Nombre del Caso de Uso	Mostrar Opciones
Actores	Jugador
Propósito	Mostrar i modificar las opciones
Resumen	El caso de uso se inicia cuando un Jugador selecciona en su menú principal la opción de modificar las opciones del juego. Posteriormente deberá poder modificar las opciones del juego
Referencias	R2.

Precondiciones	
Poscondiciones	Es mostrada la pantalla de modificación de las opciones tantas veces desee modificar las opciones el jugador y para finalizar se muestra el menú principal.

Curso Normal de los Eventos	
Acciones del Actor	**Respuesta del Sistema**
1- El Jugador selecciona la opción de modificar opciones de juego.	1.1. El sistema deberá cargar los datos para mostrar en la pantalla de modificar opciones.
	1.2 El sistema deberá mostrar la pantalla de modificar opciones.
2- El jugador decide que opción va a modificar	2.1 El sistema deberá modificar la opción de juego seleccionada.
	1.2 El sistema deberá mostrar la pantalla de modificar opciones con los nuevos datos.

Puntos de Extensión:	
Prioridad:	Secundaria
Prototipo de Interfaz:	

Tabla 1.3. Descripción detallada del CUS: Mostrar Instrucciones

Nombre del Caso de Uso	Mostrar Instrucciones
Actores	Jugador
Propósito	Mostrar las instrucciones de un juego
Resumen	El caso de uso se inicia cuando un Jugador selecciona en su menú principal la opción de mostrar las instrucciones del juego y estas son mostradas.
Referencias	R3.
Precondiciones	
Poscondiciones	Es mostrada la pantalla de Instrucciones.

Curso Normal de los Eventos	
Acciones del Actor	**Respuesta del Sistema**
1- El Jugador selecciona la opción de mostrar las Instrucciones de juego.	1.1. El sistema deberá cargar las instrucciones para mostrar en dicha pantalla de instrucciones.
	1.2 El sistema deberá mostrar la pantalla de instrucciones.

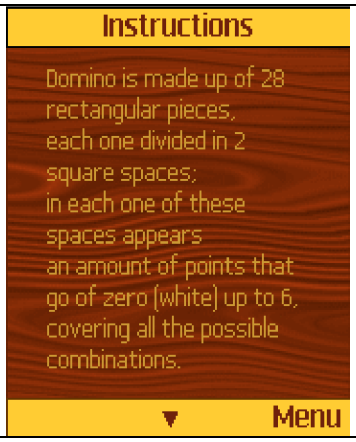
Puntos de Extensión:	
Prioridad:	Secundaria
Prototipo de Interfaz:	

Tabla 1.4. Descripción detallada del CUS: Mostrar Créditos

Nombre del Caso de Uso	Mostrar Créditos	
Actores	Jugador	
Propósito	Mostrar los créditos de un juego	
Resumen	El caso de uso se inicia cuando un Jugador selecciona en su menú principal la opción de mostrar los créditos del juego y estas son mostradas.	
Referencias	R4.	
Precondiciones		
Poscondiciones	Es mostrada la pantalla de Créditos.	
Curso Normal de los Eventos		
Acciones del Actor		**Respuesta del Sistema**
1- El Jugador selecciona la opción de mostrar los créditos del juego.		1.1. El sistema deberá cargar los créditos para mostrar.
		1.2 El sistema deberá mostrar la pantalla de Créditos.
Puntos de Extensión:		
Prioridad:	Secundaria	
Prototipo de Interfaz:		

Tabla 1.5. Descripción detallada del CUS: Mostrar Puntuaciones

Nombre del Caso de Uso	Mostrar Puntuaciones	
Actores	Jugador	
Propósito	Mostrar las puntuaciones de un juego	
Resumen	El caso de uso se inicia cuando un Jugador selecciona en su menú principal la opción de mostrar las puntuaciones de un juego y estas son mostradas.	
Referencias	R5.	
Precondiciones		
Poscondiciones	Es mostrada la pantalla de Puntuaciones.	
Curso Normal de los Eventos		
Acciones del Actor		**Respuesta del Sistema**
1- El Jugador selecciona la opción de mostrar las puntuaciones de un juego.		1.1. El sistema deberá cargar las puntuaciones obtenidas hasta el momento y mostrarlas.
		1.2 El sistema deberá mostrar la pantalla de Puntuaciones.
Puntos de Extensión:		
Prioridad:	Secundaria	
Prototipo de Interfaz:		

Tabla 1.6. Descripción detallada del CUS: Gestionar Persistencia

Nombre del Caso de Uso	Gestionar Persistencia
Actores	Jugador
Propósito	Salvar y cargar los datos de un juego.
Resumen	El caso de uso se inicia cuando un Jugador inicia un juego o finaliza alguno, entonces se debería cargar o salvar los datos del juego respectivamente.
Referencias	R6.
Precondiciones	
Poscondiciones	

Curso Normal de los Eventos	
Acciones del Actor	**Respuesta del Sistema**
1- El Jugador selecciona la opción de inicio de un juego.	1.1. El sistema deberá cargar los datos salvados para poder reanudar el juego no terminado.
2- El Jugador selecciona la opción de salida del juego.	2.1 El sistema deberá salvar los datos del juego si esto no esta concluido.
Puntos de Extensión:	
Prioridad:	Secundaria
Prototipo de Interfaz:	--------

Tabla 1.7. Descripción detallada del CUS: Crear Juego Servidor

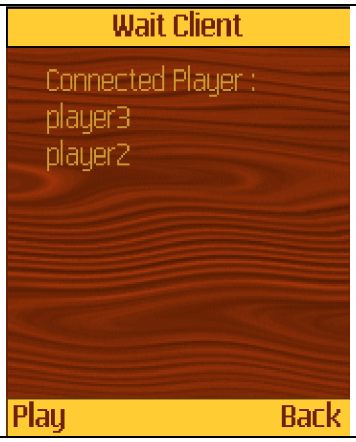
Nombre del Caso de Uso	Crear Juego Servidor	
Actores	Jugador	
Propósito	Permite al jugador hacer el papel de servidor de un determinado juego.	
Resumen	El caso de uso se inicia cuando un jugador decide jugar el modo multi-jugador. A continuación se crea un nuevo juego, este esperara por la conexión de otros jugadores y muestra los conectados. El caso de uso termina con la selección del jugador de Iniciar Juego.	
Referencias	R7.	
Precondiciones		
Poscondiciones	Inicialización de juego de modo multi-jugador servidor. Esperar por conexión de jugadores al juego.	
Curso Normal de los Eventos		
Acciones del Actor		**Respuesta del Sistema**
1- El Jugador selecciona la opción de multi-jugador servidor.		1.1. El sistema deberá inicializar y crear un servidor para permitir las conexiones de muchos clientes.
		1.2 El sistema debe de mostrar la pantalla de clientes conectados al servidor y actualizarla.
Puntos de Extensión:		
Prioridad:	Primaria	
Prototipo de Interfaz:		

Tabla 1.8. Descripción detallada del CUS: Iniciar Juego Servidor

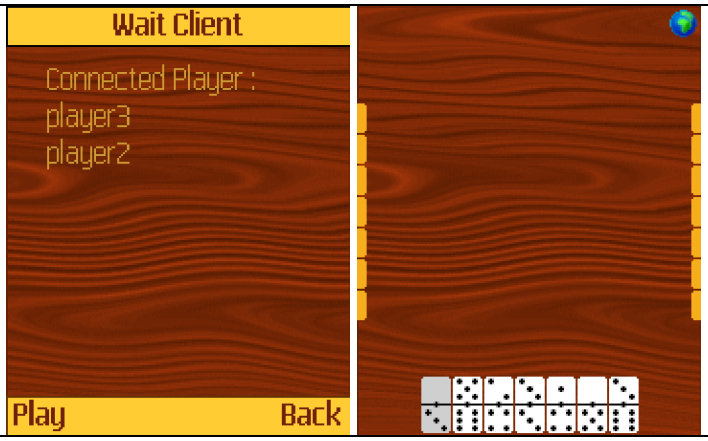
Nombre del Caso de Uso	Iniciar Juego Servidor	
Actores	Jugador	
Propósito	Un jugador comienza una nueva partida multi-jugador del juego	
Resumen	El Caso de uso se inicia cuando el jugador selecciona la opción de Iniciar Juego, se establece la conexión y se intercambian los datos para comenzar un nuevo juego. El caso de uso concluye con el inicio de la partida del juego.	
Referencias	R8.	
Precondiciones		
Poscondiciones	Comenzar un nuevo juego	
Curso Normal de los Eventos		
Acciones del Actor		**Respuesta del Sistema**
1- El Jugador selecciona la opción de Iniciar Juego		1.1. El sistema deberá intercambiar con el resto de los jugadores los datos del juego para poder comenzar uno nuevo.
		1.2 El sistema deberá indicar al resto de los jugadores que va a comenzar la partida.
		1.3 El sistema debe de mostrar la pantalla de Juego.
Puntos de Extensión:		
Prioridad:	Primaria	
Prototipo de Interfaz:		

Tabla 1.9. Descripción detallada del CUS: Unir a Juego Creado

Nombre del Caso de Uso	Unir a Juego Creado
Actores	Jugador
Propósito	Permite al jugador unirse a un juego servidor.
Resumen	El caso de uso se inicia cuando el jugador selecciona la opción de Unir a Juego, luego el sistema realiza una búsqueda de dispositivos y de los servicios que éstos prestan, mostrando un listado de juegos creados, el jugador escoge, el sistema lo agrega al juego y le envía un mensaje de confirmación, concluyendo el CUS.
Referencias	R9.

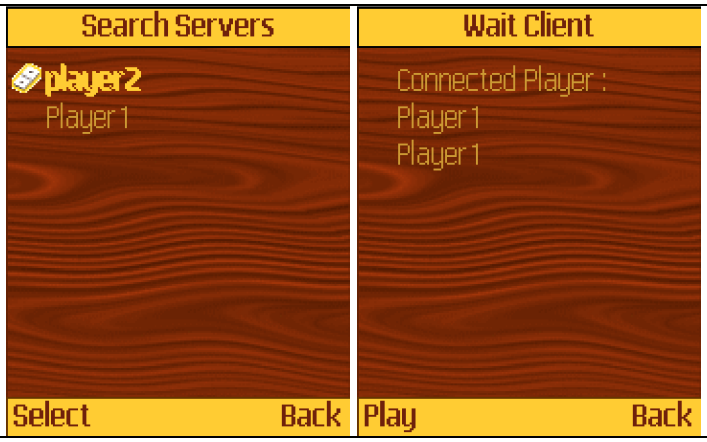
Precondiciones		
Poscondiciones	Muestra el listado de servidores Establece comunicaron con el servidor y espera porque este de inicio de jugo	
Curso Normal de los Eventos		
Acciones del Actor		**Respuesta del Sistema**
1- El Jugador selecciona la opción de Unirse a Juego		1.1 El sistema realiza una búsqueda de dispositivos y de servicios para determinar los servidores en el área.
		1.2 El sistema muestra un listado de juegos disponibles.
2-El jugador selecciona el juego al cual desea unirse		2.1El sistema establece la comunicación con el servidor y espera hasta que el servidor indique inicio de juego ,mostrara el resto de los clientes conectados a el servidor
Puntos de Extensión:		
Prioridad:	Primaria	
Prototipo de Interfaz:		

Tabla 1.10. Descripción detallada del CUS: Chequear Estado de Conexión

Nombre del Caso de Uso	Chequear Estado de Conexión	
Actores	Reloj	
Propósito	Permite es chequeo de el intercambio de información en un juego multi-jugador.	
Resumen	El Caso de uso se inicia cuando los ciclos de reloj hacen llamadas a la fase de chequeo de estado de la comunicación para si ocurrió algún determinado evento (pausa de un juego, perdida de comunicación, juego reanudado) darle solución.	
Referencias	R10.	
Precondiciones		
Poscondiciones	En caso de suceder algún evento darle respuesta.	
Curso Normal de los Eventos		
Acciones del Actor		**Respuesta del Sistema**
1- El reloj hace la llamada a la fase de chequeo de estado de la conexión.		1.1 El sistema determina el estado de la conexión ,y chequea por la ultima información

	recibida para saber si ocurrió algún evento de juego
	1.2 El sistema muestra un letrero si el evento detectado es pausado, conexión perdida o abandono de un juego.
	1.3 El sistema reanuda el juego si el evento detectado reanudación.
Puntos de Extensión:	
Prioridad:	Primaria
Prototipo de Interfaz:	

Tabla 1.11. Descripción detallada del CUS: Intercambiar Información

Nombre del Caso de Uso	Intercambiar Información	
Actores	Jugador, Reloj, Actor Comunicación	
Propósito	Permite el envió de información del juego	
Resumen	El Caso de uso se inicia cuando un Actor Comunicación (sea Reloj o Jugador) desea enviar una información del juego a través de la comunicación establecida.	
Referencias	R11.	
Precondiciones		
Poscondiciones	Se envía una determinada información al otro extremo de la comunicación.	
Curso Normal de los Eventos		
Acciones del Actor		**Respuesta del Sistema**
1- El Actor Comunicación hace referencia a enviar un determinado datos de juego		1.1 El sistema deberá enviar el determinado dato en el formato establecido para el intercambio de información por el juego
		1.2 El sistema espera por la confirmación de llegada o respuesta de a esa información.
Puntos de Extensión:		
Prioridad:	Primaria	
Prototipo de Interfaz:		

Anexo 2: Diagramas secuencias del diseño

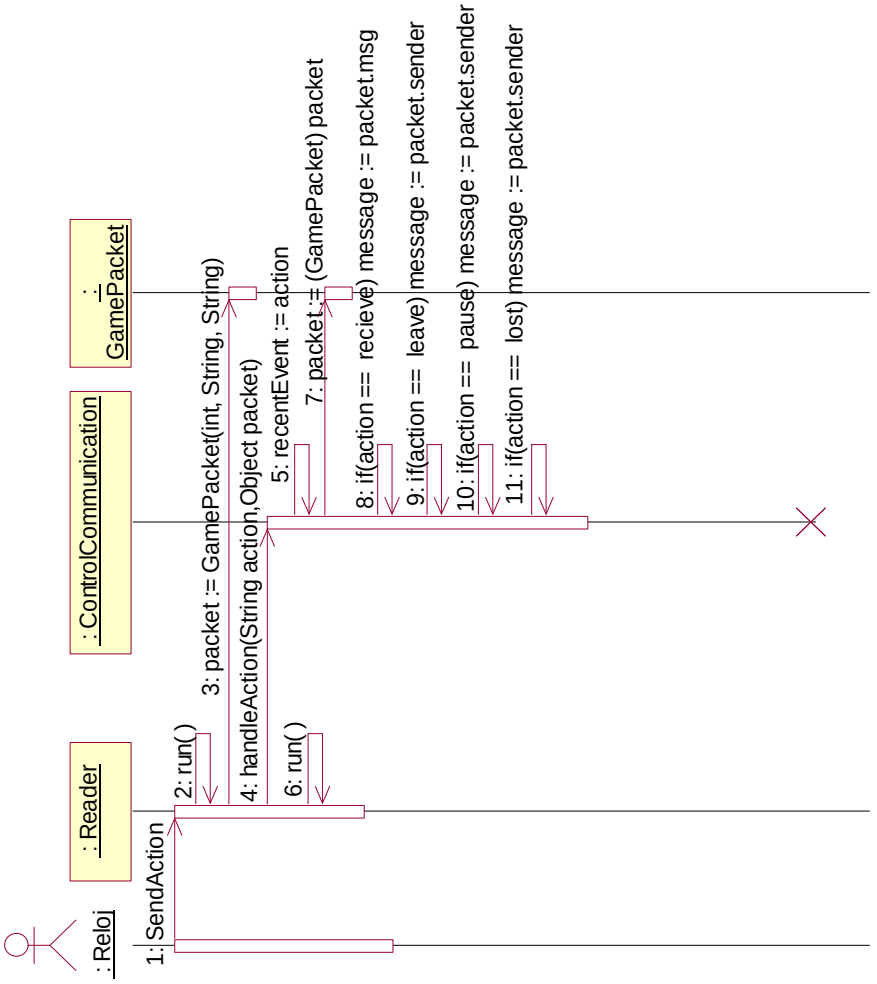


Figura 2.1 CUS Chequear Estado de Juego (Escenario Recepción de Mensaje)

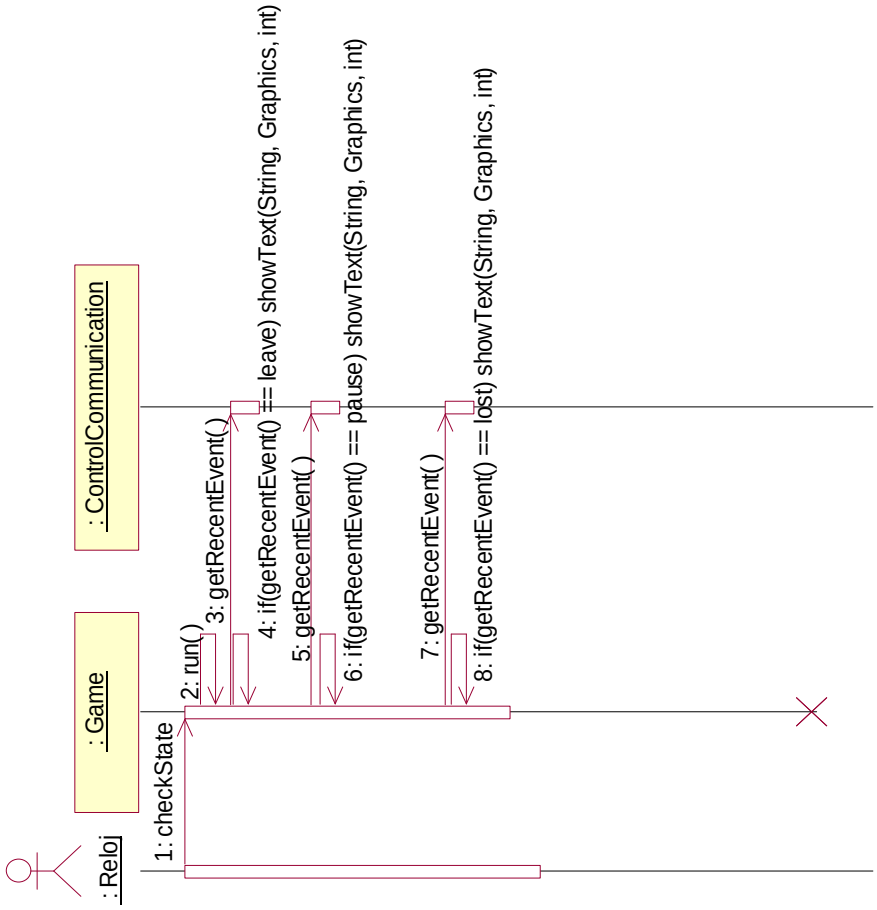


Figura 2.2 CUS Chequear Estado de Juego (Escenario Chequear Ultimo Mensaje)

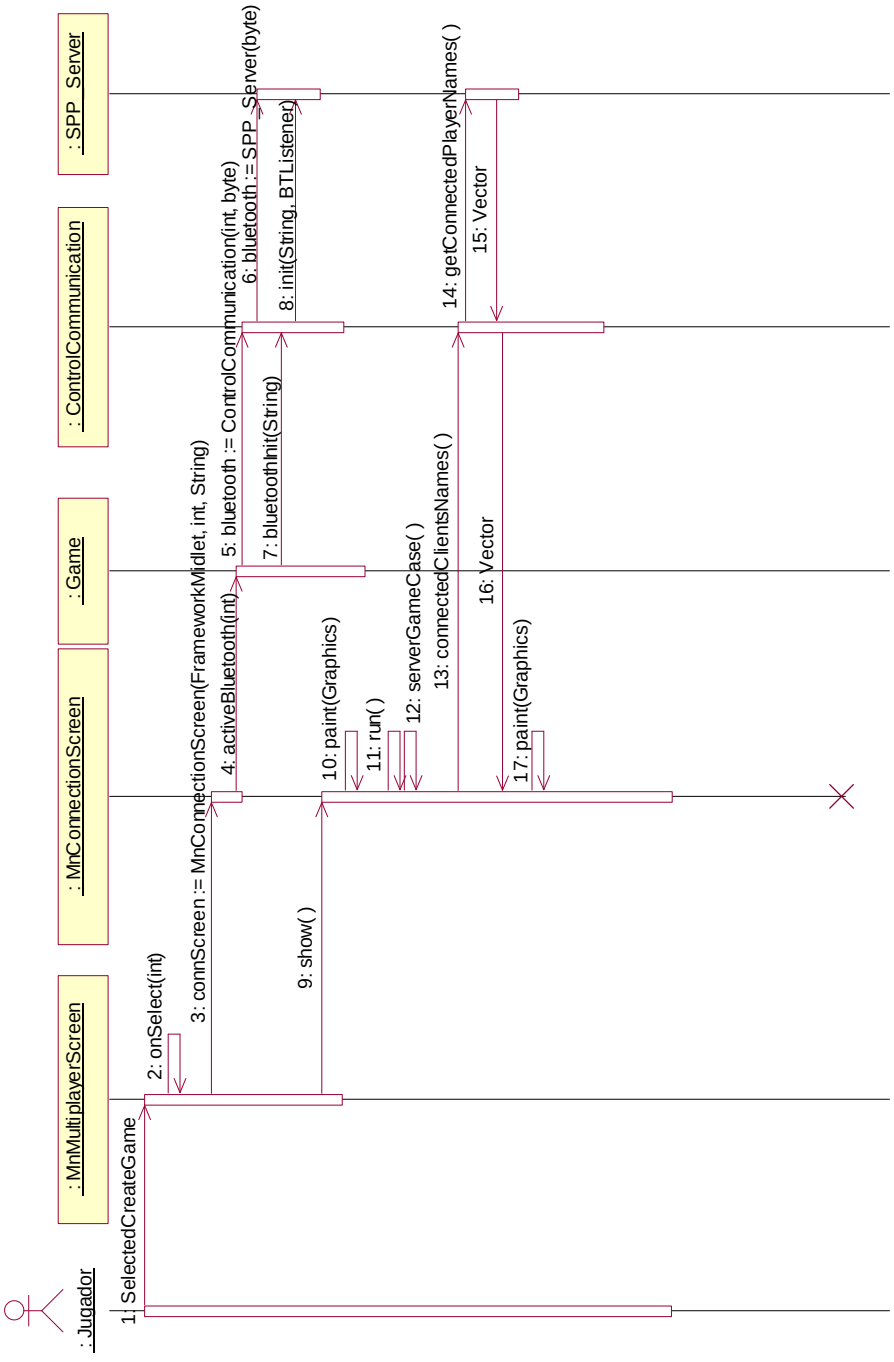


Figura 2.3 CUS Crear Juego Servidor

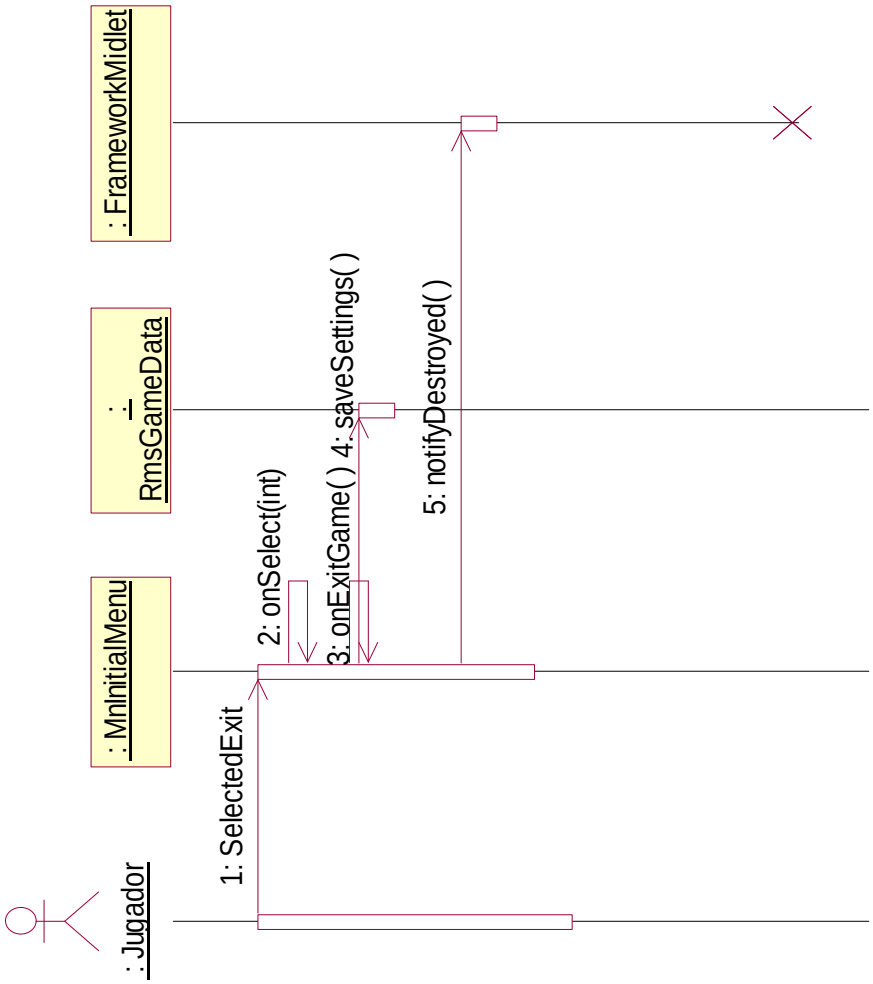


Figura 2.4 CUS Gestionar Persistencia (Escenario Guardar Datos)

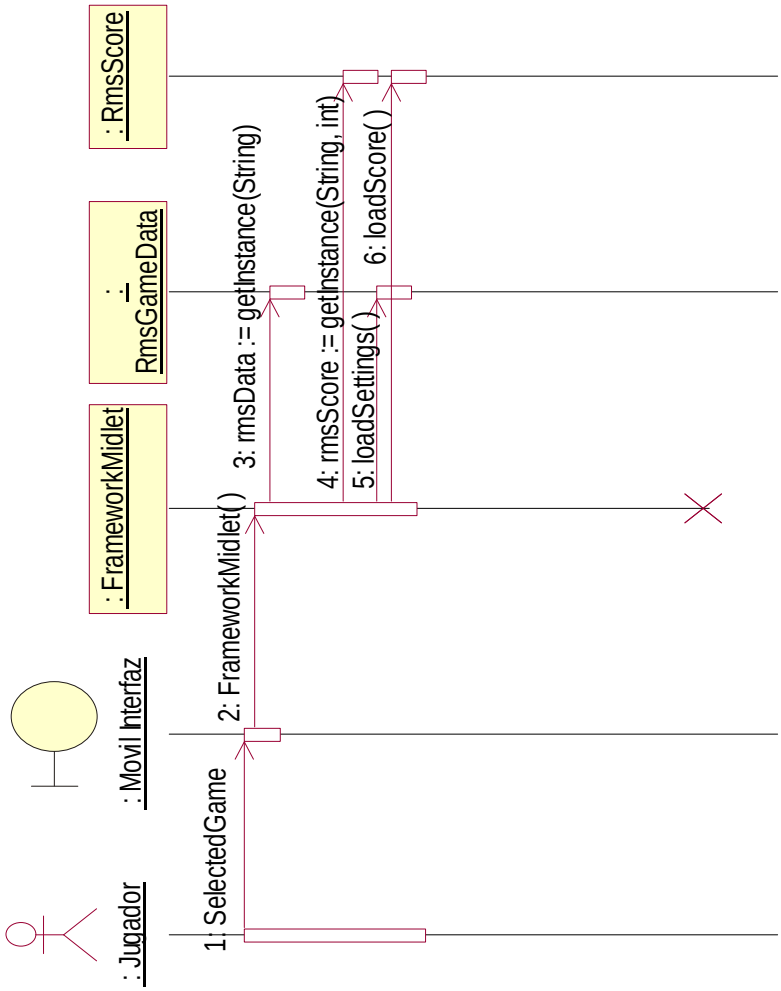


Figura 2.5 CUS Gestionar Persistencia (Escenario Leer Datos)

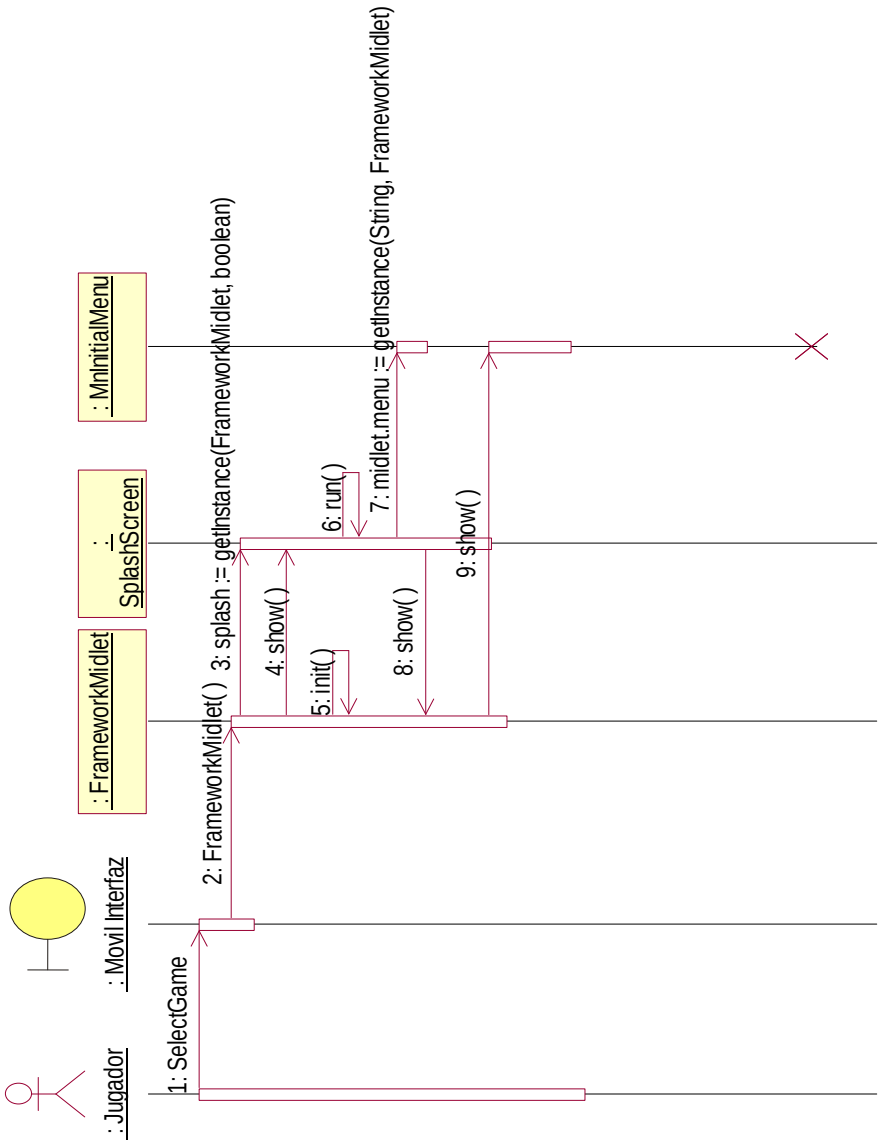


Figura 2.6 CUS Iniciar Juego

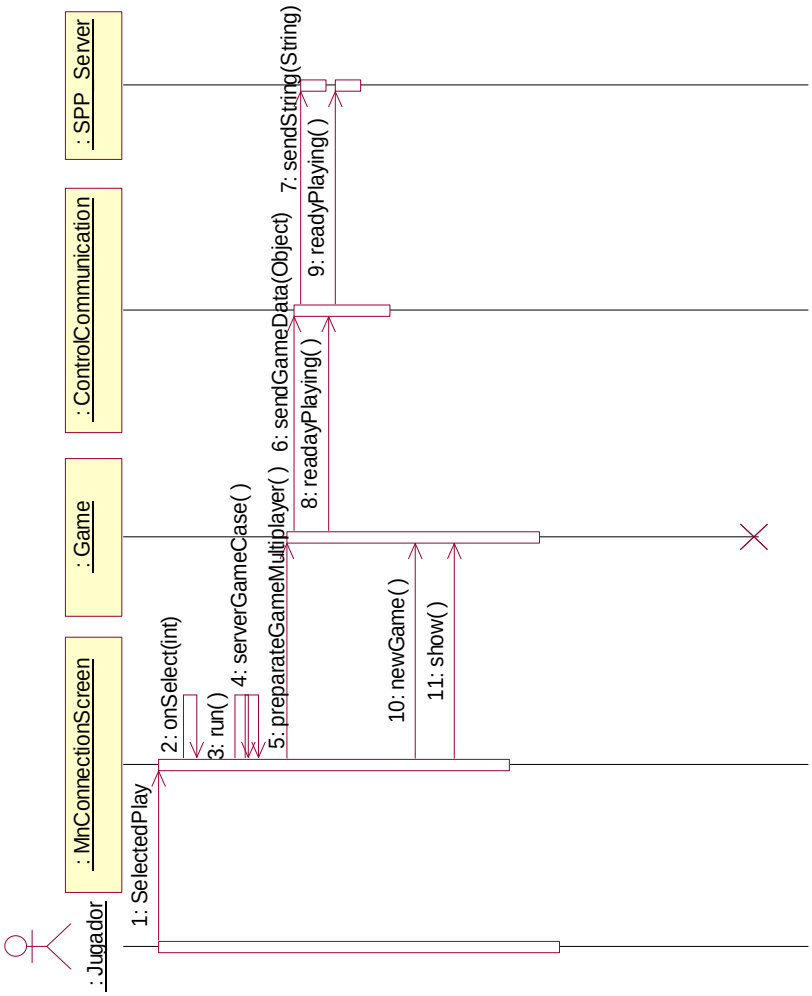


Figura 2.7 CUS Iniciar Juego Servidor

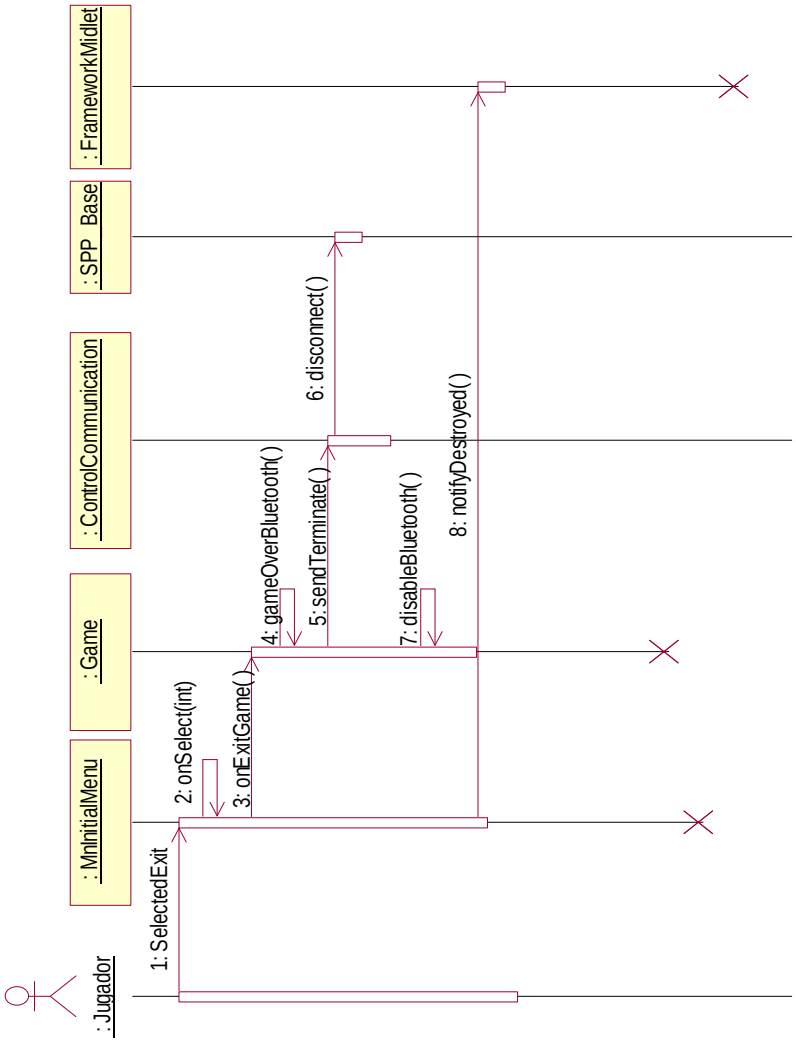


Figura 2.8 CUS Intercambiar Información (Escenario Enviar Abandono)

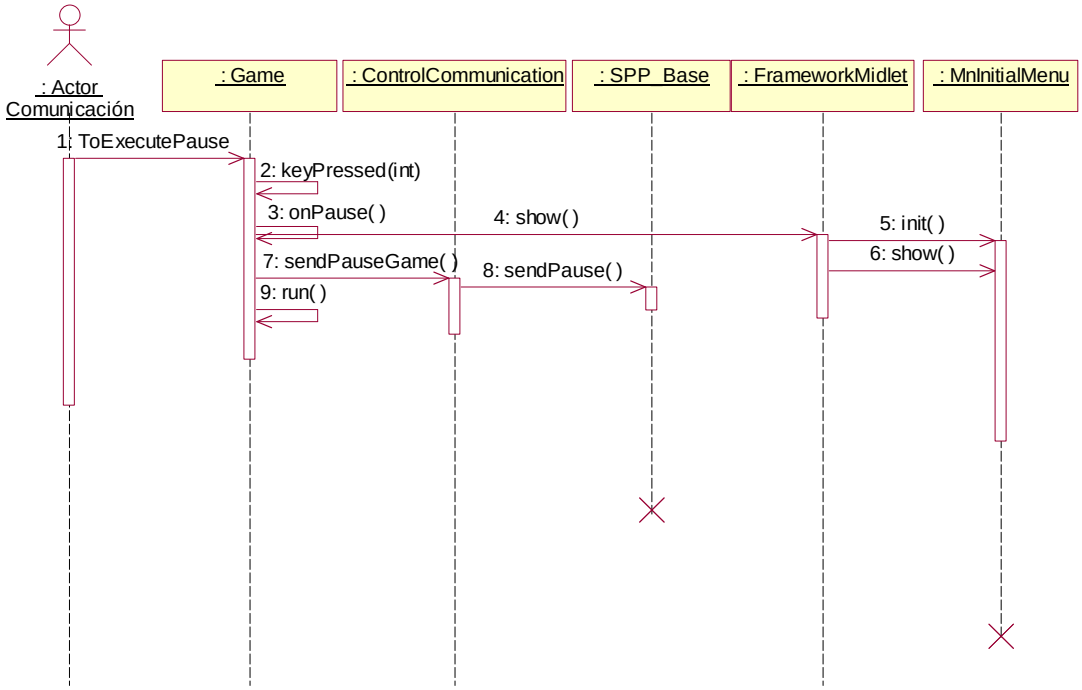


Figura 2.9 CUS Intercambiar Información (Escenario Enviar Pause)

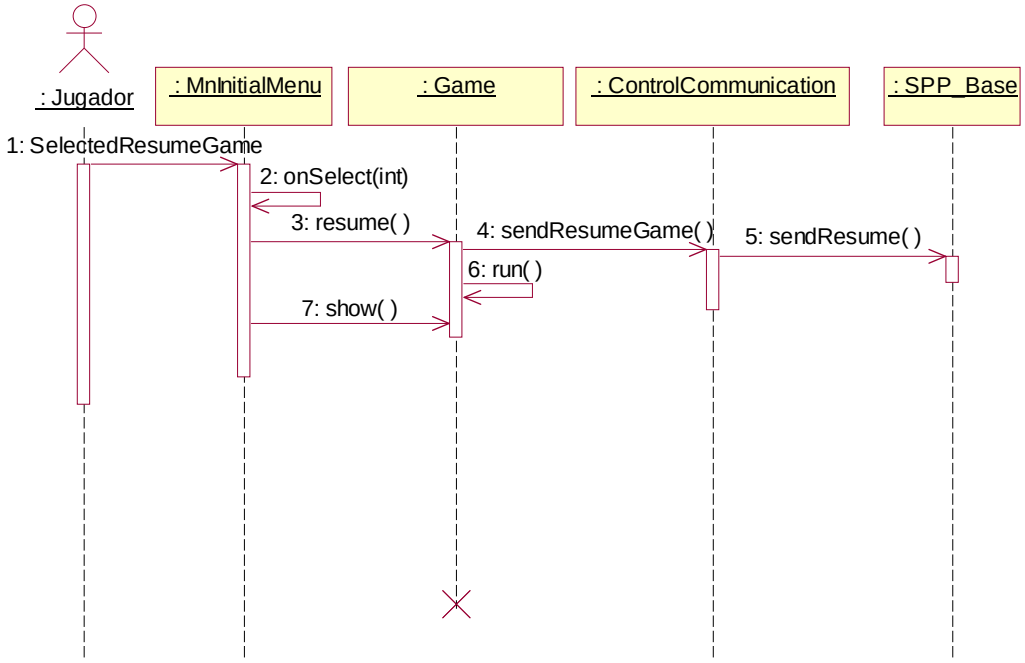


Figura 2.10 CUS Intercambiar Información (Escenario Enviar Reanudar Juego)

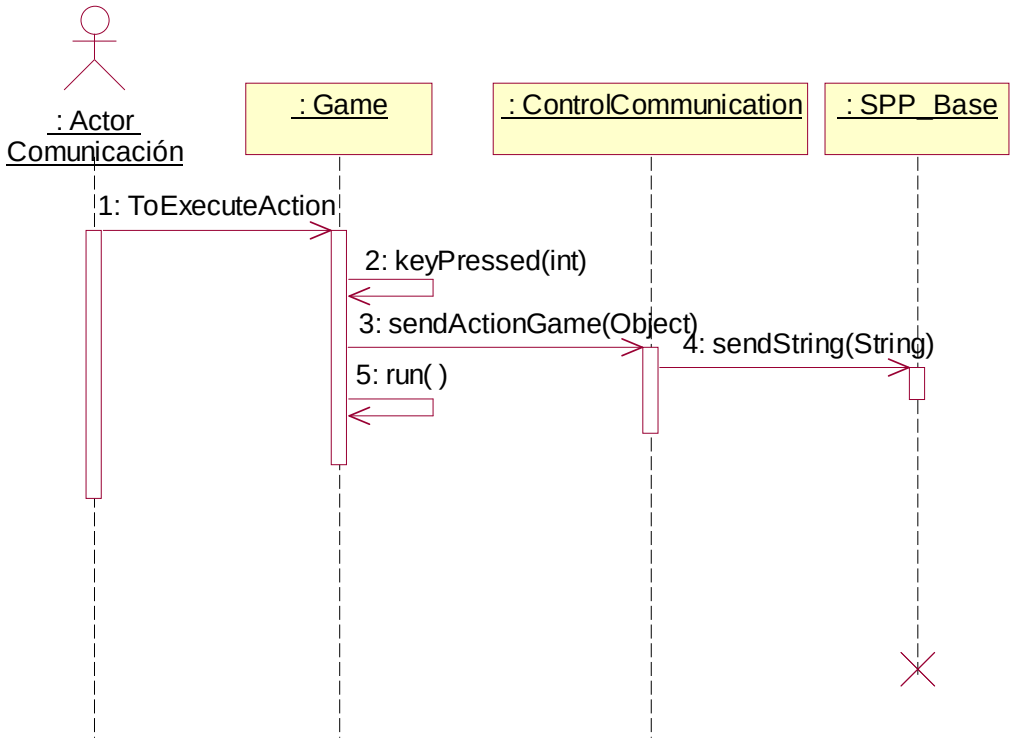


Figura 2.11 CUS Intercambiar Información (Escenario Enviar Jugada)

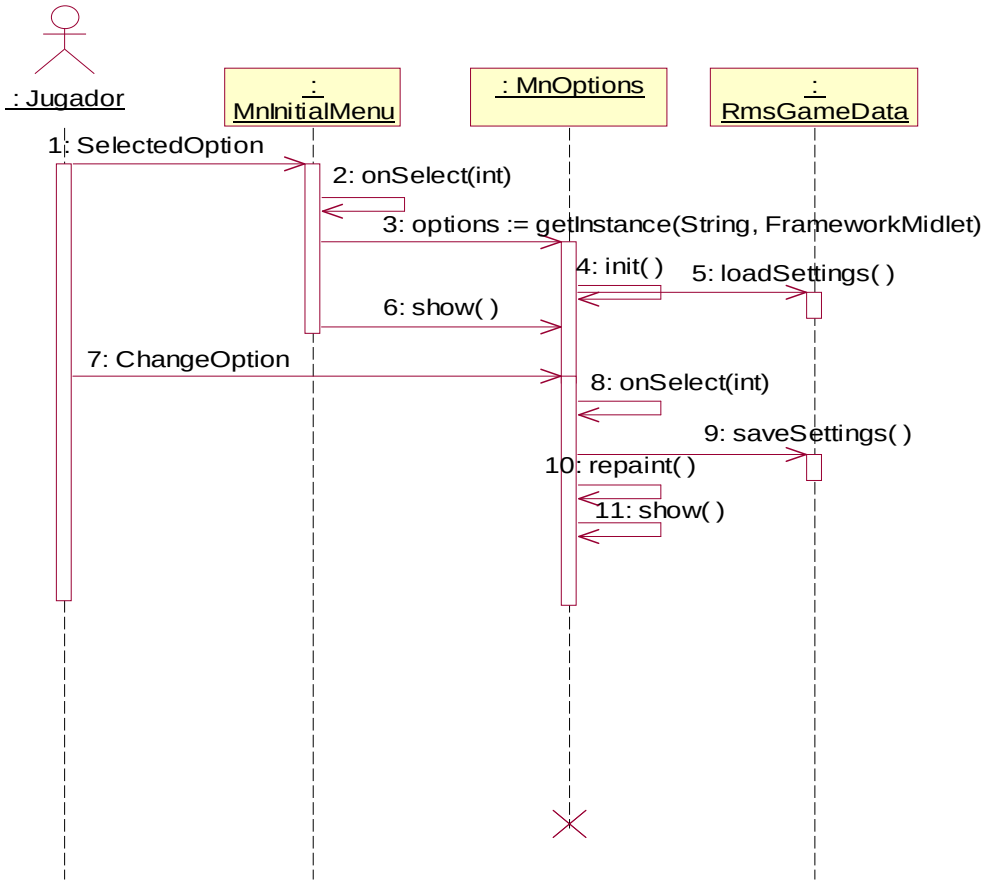


Figura 2.12 CUS Modificar Opciones

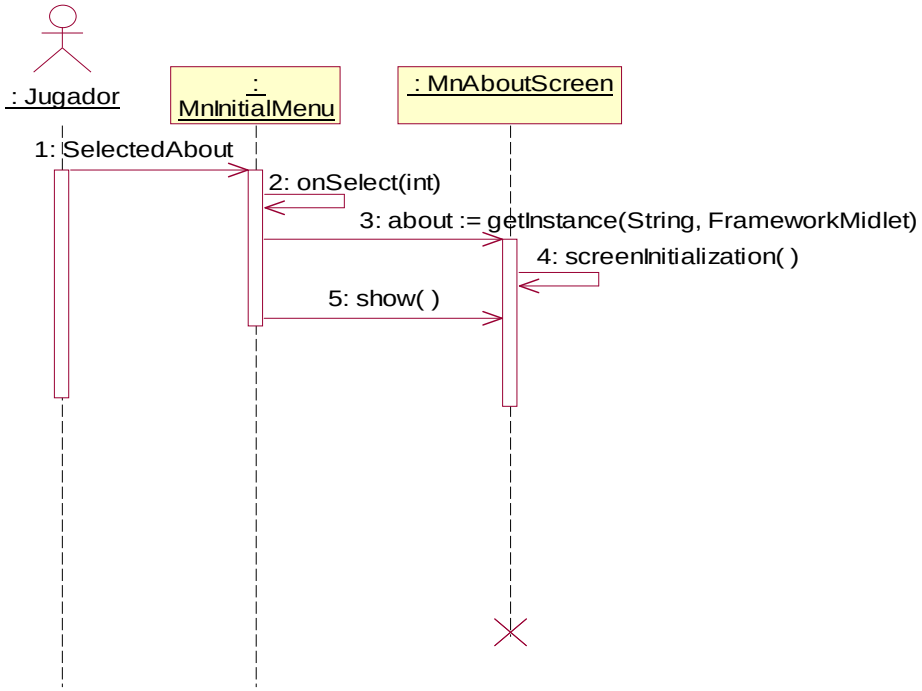


Figura 2.13 CUS Mostrar Créditos

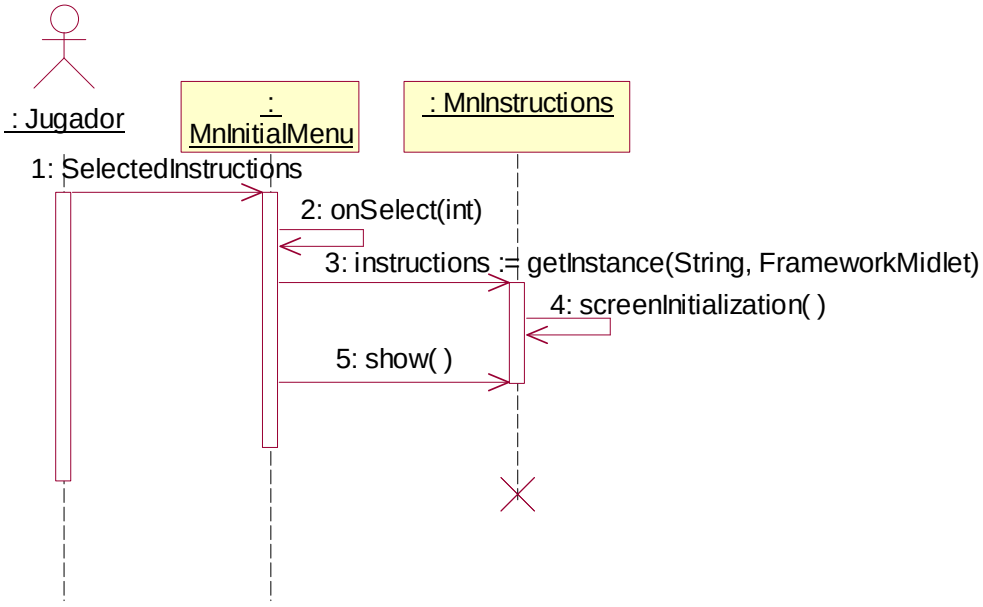


Figura 2.14 CUS Mostrar Instrucciones

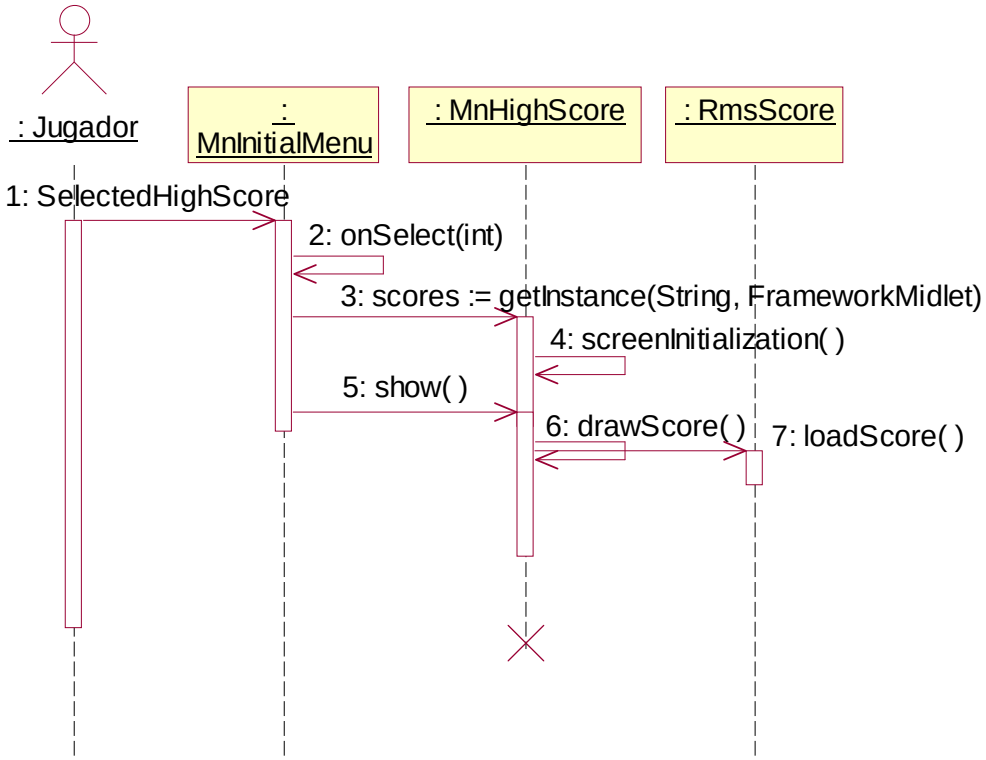


Figura 2.15 CUS Mostrar Puntuaciones

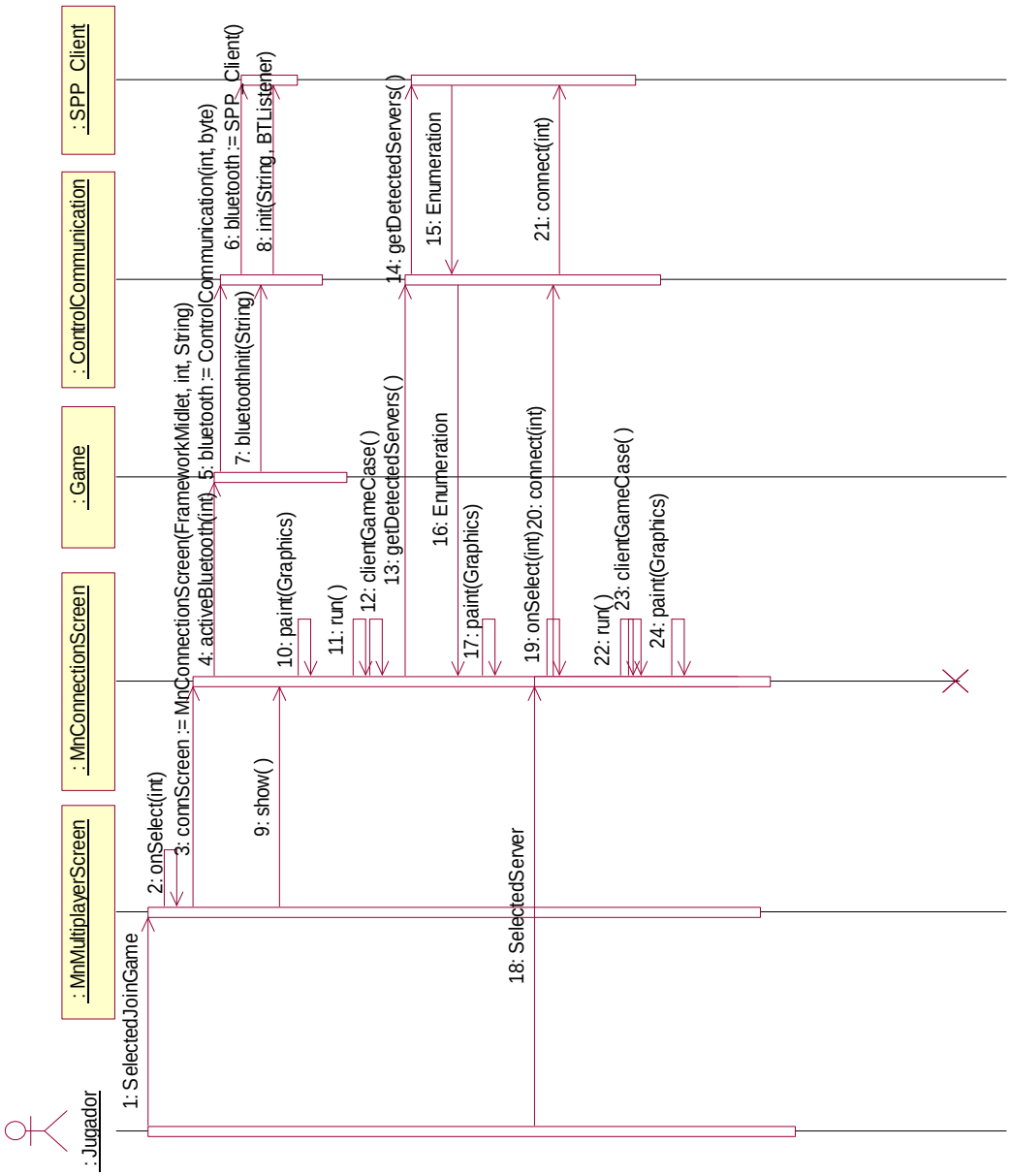


Figura 2.16 CUS Unir a Juego Creado

Anexo 3: Manual de Usuario para Instanciar JBGames

Pasos a seguir para instanciar JBGames:

1. Incorporar JBGames a su proyecto J2ME

2. Determinar qué pantallas va a utilizar: si va a agregar alguna nueva tiene que crear una clase que herede de *MnMenuCanvas* debido a que en ella se agrupan las funcionalidades básicas de un menú (estilo de letra, imagen de fondo, color de los textos, etc.). Si por el contrario, desea excluir alguna de las proporcionadas por defecto (*MnMenuInitial, MnOptions, MnIntructions, MnAboutScreen, MnMultiplayerScreen, MnConectionScreen*), deberá eliminarla del paquete y en la clase *MnMenuInitial* retirar el fragmento de código que hace referencia a ella en el método onSelect(int pos) y el asociado a esa pantalla.

3. Definir en la clase *UtilGameSpecification* la configuración de textos, teclado, imágenes e icono que acompaña a la opción de menú seleccionada en una pantalla de tipo menú selección.

4. Implementar la lógica de juego en la clase *Game* con el cumplimiento de las reglas propuestas.

5. En *RmsGameData* incorporar las variables determinadas como persistentes del juego y agregar en *MnOptions* aquellas variables que pueden ser modificadas por el usuario (ejemplo de estas variables son la dificultad, calidad visual, etc.).

6. Definir en la clase *ControlCommunication* los datos del juego a enviar y recibir al iniciar un nuevo juego multi-jugador; así como los métodos para recibir y enviar las jugadas (movimientos, acciones, situaciones y respuestas a éstas).

GLOSARIO DE TÉRMINOS Y SIGLAS

1. Bluetooth: Sistema de comunicación inalámbrica que permite la interconexión de diferentes dispositivos electrónicos (PCs, teléfonos fijos o móviles, agendas electrónicas, auriculares, etc.).
2. Framework: Estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado.
3. IDE: (del inglés: Integrated Development Environment) Entorno Integrado de Desarrollo)
4. J2ME (Java 2 Micro Edition): Versión desarrollada por la Sun Microsystems de Java, destinada a dispositivos de recursos limitados como PDAs, teléfonos móviles.
5. MIDP (Mobile Information Device Profile): Es el perfil para dispositivos de información móviles que combina con la configuración CLDC para proporcionar un entorno de ejecución para dispositivos móviles.
6. CLDC (Connected, Limited Device Configuration): Es una configuración diseñada para dispositivos con conexiones de red intermitentes, procesadores lentos y memoria limitada como teléfonos móviles y asistentes personales.
7. CASE: Computer Aided Software Engineering.
8. Herramientas CASE: Herramientas utilizadas para el desarrollo de proyectos de Ingeniería de Software.
9. Hardware: Componentes electrónicos, tarjetas, periféricos y equipo que conforman un sistema de computación; se distinguen de los programas (software) porque son tangibles.
10. RUP: Rational Unified Process (Proceso Unificado de desarrollo). Metodología para el desarrollo de Software.
11. Software: Programas de sistema, utilerías o aplicaciones expresados en un lenguaje de máquina.
12. UCI: Universidad de las Ciencias Informáticas.
13. UML: Unified Modeling Language. Es una notación estándar para modelar objetos del mundo real como primer paso en el desarrollo de programas orientados a objetos. Es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema de software.

14. IrDA(Infrared Data Association): Organización con el fin de crear normas internacionales para el hardware y el software empleados en comunicaciones por infrarrojo, muy importante en comunicaciones inalámbricas.
15. SDDB(Service Discovery Data Base): Base de Datos de Descubrimiento del Servicio.
16. L2CAP, (Logical Link Control and Adaptation Protocol): Protocolo de control y adaptación del enlace lógico es utilizado dentro de la pila de protocolos de Bluetooth.
17. RFCOMM (Radio Frequency Communication) :Comunicación por radio frecuencia.