UNIVERSIDAD AUTÓNOMA METROPOLITANA
UNIDAD AZCAPOTZALCO

DIVISIÓN DE CIENCIAS BÁSICAS E INGENIERÍA
LICENCIATURA EN INGENIERÍA EN COMPUTACIÓN

PROYECTO TERMINAL
EN
INGENIERÍA EN COMPUTACIÓN

Software Colaborativo para Dispositivos Móviles con Comunicación Bluetooth

Proyecto que presenta:
José Luis Ortigosa Flores

Para obtener el titulo de:
Ingeniero en Computación

Director del proyecto:
M. en C. Oscar Alvarado Nava

México, D. F. Julio de 2011

RESUMEN

En este documento se presenta el desarrollo del proyecto terminal que lleva por título "*Software Colaborativo para Dispositivos Móviles con Comunicación* ***Bluetooth***[1]***"***; que como se puede inferir del título, se trata básicamente de una aplicación dirigida hacia los *dispositivos móviles*[2] y que además será capaz de comunicarse con otros dispositivos móviles a través de *bluetooth*.

Se mostrará durante todo el documento, el proceso de desarrollo de la aplicación previamente mencionada, desde una breve descripción de la aplicación a desarrollar, pasando por su arquitectura, y todo el proceso de desarrollo, y hasta llegar al proceso de ejecución e instalación.

[1] La comunicación *bluetooth* es un tipo de comunicación estándar por radiofrecuencias. Muy común en los dispositivos móviles de hoy en día.

[2] Los dispositivos móviles que se consideran incluidos, son aquellos que tengan soporte para Java ME y en específico soporte para MIDP 2.0. Entre los que pueden estar: Celulares, PDAs y Smartphones.

AGRADECIMIENTOS

- A la Universidad Autónoma Metropolitana
- A la División de Ciencias Básicas e Ingeniería de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco.
- Al Departamento de Electrónica y de Sistemas.
- A mi asesor el profesor M. en C. Oscar Alvarado Nava, por confiar en mí para realizar este proyecto.
- A mí familia y en especial a mi Madre y Hermano por todo su apoyo.

DEDICATORIA

Dedicado con todo cariño a mis Abuelos la señora Tomasa Santamaría García y el señor Samuel Flores Roja y muy especialmente a mí Madre Ma. Osvelia Flores Santamaría.

LISTA DE FIGURAS

LISTA DE TABLAS

ÍNDICE GENERAL

1 INTRODUCCIÓN

En este apartado se presentan las causas que motivaron la realización del proyecto terminal, así como los objetivos globales que tiene el mismo.

1.1 Motivaciones

En la actualidad los dispositivos móviles están ganando, cada vez más terreno en los mercados, frente a los dispositivos que cuentan con grandes cantidad de recursos hardware/software tales como las *PCs*[3], debido en gran medida a las diferencias de costos; costos que están determinadas por la portabilidad que ofrecen los dispositivos móviles, haciendo que estos cuenten con una menor cantidad de recursos hardware/software, sin embargo, la tendencia de estos últimos es hacia el crecimiento de sus recursos entre ellos el poder de procesamiento, conexión a redes *Wi-Fi*[4], pantallas de mejor resolución, *tecnología touch*[5], entro otros.

Otra característica entre los dispositivos móviles, que esta ayudando en su rápido crecimiento en los mercados, es que estos están permitiendo un acercamiento cada vez más importante hacia la llamada *computación ubicua*[6]. Lo que significa que más usuarios optaran por un dispositivo que les permite acceder a sus datos, documentos y aplicaciones desde cualquier lugar y/o momento en el que se encuentren; estos últimos contra dispositivos grandes y pesados destinados a permanecer en un lugar de forma estática e inamovible.

Todas las características antes mencionadas motivan a los desarrolladores a crear software que aproveche los pocos recursos con los que cuentan los dispositivos. Tendiendo en mente que el mercado de los dispositivos móviles esta en crecimiento.

Muchas de las aplicaciones que existen en el mercado o que son de libre acceso, están orientadas a aplicaciones en las que interviene un solo usuario. La idea de realizar un software colaborativo es la de aprovechar la tecnología *bluetooth,* que esta incluida en

[3] Siglas en inglés para *Personal Computers* que en español significa Computadoras Personales

[4] Tecnología para comunicación entre dispositivos, que permite crear redes WLAN, por sus siglas en inglés para *Wireless Local Area Network* que en español significa Redes Inalámbricas de Área Local; estas redes permiten anchos de banda mayores a las formadas con tecnología de comunicación *bluetooth*.

[5] Tecnología que permite que las pantallas sean sensibles al tacto, y que además permitan una interacción más natural e intuitiva con el usuario.

[6] Es el acceso a datos, información y aplicaciones a través de redes de dispositivos diversos, sin importar, la localización o un momento en específico.

muchos de los dispositivos móviles, con el fin de que varios usuarios colaboren en alguna actividad determinada de tal manera que aprovechen dicho medio de comunicación.

Existen, además de las aplicaciones orientadas a un usuario; aplicaciones que están orientadas a soportar a máximo dos usuarios, un ejemplo seria el clásico juego de ajedrez o juegos similares, sin embargo, aplicaciones en las que intervengan 3 ó 4 ó más usuarios son escazas, o no existentes.

1.2 Objetivos

Los objetivos planteados para la realización del producto final de este proyecto terminal son los siguientes:

- Realizar una aplicación orientada a dispositivos móviles que posibilite un trabajo colaborativo entre máximo cuatro dispositivos.
- Hacer que los dispositivos se comuniquen usando la tecnología bluetooth.
- Hacer que el trabajo colaborativo consista en armar un rompecabezas, es decir, que máximo cuatro usuarios colaboren para armar un único rompecabezas.

1.3 Organización del proyecto

Este documento estará conformado de manera general por las secciones que se muestran en la Tabla 1.

Partes	Descripción
Descripción	Consiste en describir la idea general de lo que será el producto final del proyecto.
Alcances y limites	Describir exactamente los alcances y limitaciones que tendrá la el producto final del proyecto.
Arquitectura	Describir la arquitectura que tendrá el producto final del proyecto.
Desarrollo e implementación de la aplicación	Describir todos los elementos que permitirán el desarrollo del producto final del proyecto.
Producto final	Describir las características del producto final.
Instrucciones de instalación	Describir el proceso de instalación del producto final del proyecto.
Manual de aplicación	Describir el modo de uso del producto final.

Tabla 1: Secciones generales que conforman el proyecto terminal.

2 DESCRIPCIÓN DE LA APLICACIÓN

En este apartado se describe las características del sistema o aplicación, así como sus limitaciones y alcances.

2.1 Especificación técnica

El software colaborativo que se realizará estará enfocado para trabajar a lo más, entre cuatro dispositivos móviles. La comunicación entre los dispositivos móviles será vía bluetooth. Los mensajes que viajarán entre los dispositivos y que permitirán la comunicación, estarán basados en XML[7], (para dispositivos que le den soporte en un protocolo más simple para transferencia de datos). En la Figura 1 se muestra un esquema que representa la idea general de la aplicación.

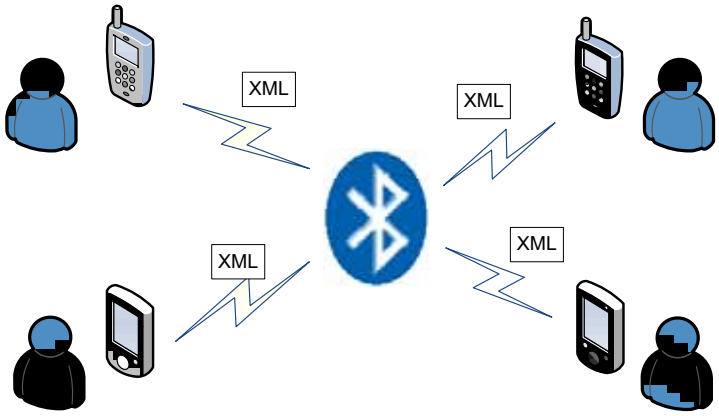

Figura 1: Esquema con la idea general del funcionamiento del sistema.

En general la aplicación permitirá a máximo cuatro usuarios, cada uno desde su propio dispositivo, realizar una actividad de forma colaborativa; dicha actividad consistirá en armar un rompecabezas entre los usuarios que estén participando y donde el medio de comunicación será la vía inalámbrica que proporciona la tecnología *bluetooth*.

[7] Siglas en inglés para *Extensible Markup Language* que en español significa Lenguaje de Marcado Extensible. "Es una tecnología involucrada con la descripción y estructuración de datos (…)".
"(…) Es un lenguaje de marcado que no especifica ni las etiquetas, ni la gramática del lenguaje (…)".

2.2 Descripción técnica

El sistema estará conformado por al menos seis componentes principales; tales componentes o módulos se enlistan y describen en la Tabla 2.

Componente	Descripción
Gestión del Trabajo Colaborativo	Módulo encargado de gestionar los mensajes que se deben de enviar o recibir para hacer posible el trabajo colaborativo, entre los dispositivos móviles. Los mensajes estarán basados en XML para dispositivos que lo soporten. Este módulo determinará a que dispositivo(s) se le(s) enviará cuál(es) mensaje(s).
Gestión del Envío de Mensajes	Es el módulo encargado de asegurarse de enviar los mensajes a los dispositivos correctos.
Gestión de la Recepción de Mensajes	Es el módulo encargado de recibir los mensajes enviados por los dispositivos.
Comunicación Bluetooth	Este módulo es el encargado de la parte física de la comunicación, es decir de hacer posible que los mensajes viajen a través de bluetooth.
Gestión del control del Juego	Es el módulo encargado de gestionar los eventos provenientes del teclado, del dispositivo móvil.
Gestión del dibujado de Gráficos	Módulo encargado de dibujar en pantalla todas y cada una de las vistas de la aplicación.

Tabla 2: Componentes que conforman a la aplicación.

En la Figura 2 se muestra la interacción de todos los componentes que conforman la aplicación y que se enlistaron en la Tabla 2.

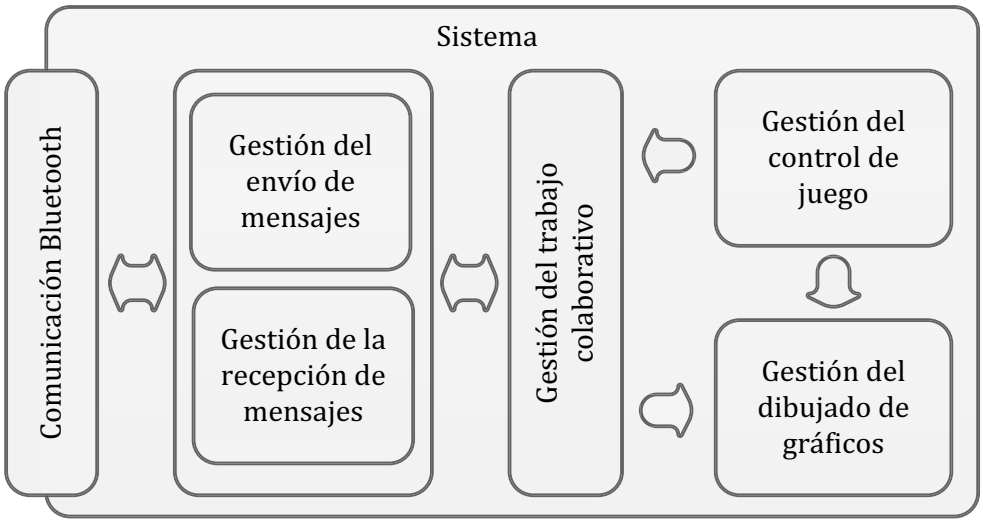

Figura 2: Interacción entre los componentes que conforman la aplicación.

3 ELECCIÓN DE LA TECNOLOGÍA

En este apartado se mencionan algunas tecnologías disponibles para el desarrollo de aplicaciones para dispositivos móviles, así como de los *sistemas operativos*[8] con los que cuentan dichos dispositivos.

Otro punto que se toca en este apartado, son los motivos por los cuales se elige a la tecnología java.

3.1 Tecnologías para el desarrollo

En el mercado existe una enorme variedad de tecnologías disponibles para el desarrollo de aplicaciones para dispositivos móviles, entre las que se encuentran las enlistadas en la Tabla 3.

Tecnologías
Java ME (Oracle, 2011)
C/C++
API[9] Java para Android OS.
API Java para el BlackBerry OS.
API para el sistema operativo de los teléfonos iPhone.

Tabla 3: Tecnologías para el desarrollo de aplicaciones para dispositivos móviles.

Algunos de los sistemas operativos más conocidos y las empresas de celulares que usan esos sistemas operativos son los que se enlistan en la Tabla 4.

[8] Software que se encarga básicamente de administrar los recursos hardware y/o software de un dispositivo con capacidad de procesamiento.

[9] Siglas en ingles para *Application Program Interface*. La API se refiere al conjunto de métodos que permiten a otros programas, aplicación y/o componentes interactuar con otros programas aplicaciones y/o componentes.

Sistema Operativo	Empresas que usan el OS[10]
Symbian OS	Nokia Sony Ericsson Samsung Motorola
iPhone OS (iOS)	iPhone de Apple
BlackBerry OS	BlackBerry
Windows Phone OS	Microsoft
Android	LG Sony Ericsson Samsung
Palm	Nokia

Tabla 4: Sistemas operativos para dispositivos móviles.

Algunas plataformas de desarrollo se muestran en la Tabla 5.

Sistema Operativo	IDE[11]
Symbian OS (SymbianOS.org, 2011)	Eclipse + Plugin[12] +SDK Visual Studio + SDK + Plugin
Windows Phone (Microsoft, 2011)	Visual Studio + SDK
iPhone OS (iPhone OS X, 2011)	Xcode + SDK
Palm	Codewarrior + SDK

Tabla 5: Plataformas de desarrollo para algunos sistemas operativos.

Existen varias empresas que tiene soporte para la tecnología Java. En la Tabla 6 se enlistan las empresas que soportan la tecnología Java ME y los entornos de desarrollo disponibles para desarrollar aplicaciones con la tecnología Java.

Marca	Eclipse (Java ME)	Netbeans (Java ME)
Nokia (Nokia, 2011)	Plugin(SDK[13])	Plugin(SDK)
Sony Ericsson (Sony Ericsson Mobile Communications AB, 2011)	Plugin(SDK)	SDK
Samsung (Samsung, 2010)	SDK	Plugin(SDK)
Motorola (Motorola Mobility, 2011)	SDK	Plugin(SDK)

[10] Siglas en inglés para *Operative System* que en español significa Sistema Operativo.

[11] Siglas en inglés para *Integrated development environment* que en español significa entorno de desarrollo integrado, la mayoría de los IDE integran un editor, un compilador, un depurador entre otros, todos útiles para la creación de aplicaciones.

[12] Es un conjunto de componentes software que agregan habilidades específicas a una aplicación, es decir, amplían o extienden la funcionalidad de una aplicación.

[13] Siglas en inglés para *Software Development Kit* que en español significa juego de herramientas para el desarrollo de aplicaciones software.

Blackberry (BlackBerry, 2011)	Plugin(SDK)	SDK
Android (Google, 2007)	Plugin(SDK)	SDK
Palm (Palm, Hewlett-Packard Company, 2011)	SDK	SDK

Tabla 6: Empresas y entornos de desarrollo que dan soporte a Java ME.

3.2 Tecnología Java

Tal como se puede ver en la Tabla 3 y en la Tabla 4 la tecnología Java resalta entre todas. Esta tecnología fue desarrollada por la desaparecida empresa *Sun Microsystems* que fue comprada por la empresa *Oracle*.

La empresa *Sun Microsystems* desarrollo la Maquina Virtual de Java o JVM[14]. Esta máquina a rasgos generales es un *middleware*[15] que abstrae al programador o al desarrollador de las cuestiones particulares de cada sistema operativo y de la enorme variedad de hardware, permitiéndole al desarrollador solo concentrarse en la lógica del negocio más que en cuestiones meramente técnicas y especificas de cada dispositivo.

Hasta antes de la JVM cada desarrollador tenia qua implementar aplicaciones pensando en un sistema operativo en particular, y en casos extremos también en un hardware en específico.

La *JVM* lo que hace es leer un programa compilado, escrito en un lenguaje robusto llamado Java. Posteriormente la *JVM* se encarga de negociar con el sistema operativo, todas las llamadas al sistema que sean necesarias.

Todo lo anterior es verdad en dispositivos que cuentan con grandes recurso hardware/software, sin embargo la realidad es que en los dispositivos móviles existen varias diferencias.

La empresa *Sun Microsystems* desarrollo una versión reducida de la *JVM* denominada Kilo Virtual Machine o *KVM*[16] por sus siglas en inglés, que en realidad se trata de una versión reducida de la *JVM.* A esta tecnología se le denomino *Java Micro Edition*[17], esta tecnología propone tres especificaciones para el desarrollo de aplicaciones móviles.

La primera denominada CLDC/MIDP por sus siglas en inglés para *Connected Limited Device Configuration* y *Mobile Information Device Profile*, respectivamente. Donde CLDC

[14] Siglas en inglés para *Java Virtual Machine* que en español significa Máquina Virtual de Java.
[15] Capa intermedia de software que permite la abstracción de las particularidades del hardware y/o software como el sistema operativo.
[16] Se le denomina KVM por que la máquina virtual ocupa unos cuantos kilobytes de espacio.
[17] Edición de java enfocada para los dispositivos móviles.

es una especificación del *framework*[18] para Java ME, esta especificación describe a un conjunto de bibliotecas y características de la máquina virtual que deben de estar presentes en una implementación. Y donde MIDP provee una API estándar para desarrollar aplicaciones, estas APIs incluyen el manejo del ciclo de vida de las aplicaciones, conectividad a redes HTTP, interfaces de usuario, almacenamiento persistente, etc.

A los MIDP se les denomina perfil, este perfil esta diseñado para los teléfonos celulares. Existen dos versiones hasta el momento, la MIDP 2.0 y la MIDP 2.1 que están especificadas en el JSR[19] 37 (JSR 37, 2000) y en el JSR 118 (JSR 118, 2000) respectivamente. La mayoría de los celulares están equipados con una implementación de MIDP que es una popular plataforma para juegos.

La segunda es CDC (Oracle) por sus siglas en inglés para *Connected Device Configuration*, descrita en el *JSR 218* (JSR 218, 2000) esta configuración define las capacidades básicas que debe tener un dispositivo móvil y cuyo potencial de procesamiento esta más haya de un teléfono pero de menor capacidad que una computadora de escritorio. CDC esta conformado por un subconjunto de clases de Java SE. Al momento de buscar dispositivos que contarán con este soporte no se lograron encontrar.

La tercera es *Java FX* (Oracle, 2010), *Java FX* es una tecnología cruzada ya que mezcla las tecnologías *Java ME*, *Java EE* y *Java SE* y una nueva plataforma con su script *Java FX*, esta tecnología esta orientada a dispositivos tales como televisores, computadoras, celulares *smartphones*[20], entre otros. El entorno de desarrollo *Netbeans* da soporte a esta tecnología para proporcionarle un entorno de desarrollo.

Java FX es una tecnología que ofrece un modelo de desarrollo y despliegue unificado para la creación de Aplicaciones Enriquecidas de Internet mejor conocidas como RIA por sus siglas en inglés para *Rich Internet Aplication* en computadoras personales y en celulares *smartphones*.

[18] Es el conjunto de clases abstractas y sus objetos que colaboran con ellas, así como las clases concretas. Mientras que la programación orientada a objetos fomenta la reutilización de de software, el framework fomenta la reutilización de los diseños. (Linthicum, 2000)

[19] Siglas en inglés para *Java Specification Request* que es la documentación formal que describe las especificaciones o tecnologías propuestas para ser agregadas a la plataforma java.

[20] Nombre otorgado a los celulares inteligentes, estos celulares por lo regular tienen procesadores con ciclos de reloj mayores a 500 MHz, con RAM superior o igual a los 200 MB, además de características de comunicación tales como Wi-Fi, Bluetooth, GPS y pantallas con tecnología táctil.

3.3 Tecnología para el desarrollo del sistema

Como se mencionó a lo largo de todo el aparado “Elección de la tecnología”, existen varias plataformas que permiten el desarrollo de aplicaciones para los dispositivos móviles, desde las aplicaciones nativas tales como las desarrolladas con el lenguaje C/C++ para los dispositivos con sistema operativo Symbian hasta las tecnologías con capas intermedias que abstraen al programador de particularidades del sistema operativo como lo es la tecnología Java.

Dado que Java es más flexible y dado que esta tecnología es soportada por varias marcas en el mercado incluyendo la más grande que es *Nokia* y tomando en cuenta lo que se menciona en el libro de *Linthicum* (Linthicum, 2000) con respecto a las tecnologías de moda, en donde comenta que no siempre es buena idea usar la tecnología del momento por que son susceptibles a problemas nuevos y poco conocidos, impidiendo dar solución de manera pronta y expedita, como lo son las tecnologías nuevas propuestas por *Oracle* y *Java FX* como la tecnología basada en Java propuesta por *Google*.

Otro factor importante que determino la elección de la tecnología Java y no la tecnología propuesta por *Apple* es que su licencia es muy restrictiva y limitante, a diferencia de Java que es una tecnología abierta.

Por lo anterior la tecnología que se usará para la realización de este proyecto terminal será la tecnología *Java Micro Edition* con las aplicaciones tipo CLDC/MIDP.

El entorno de desarrollo será el IDE Java ME Platform SDK 3.0, que da soporte a las aplicaciones de tipo CLDC/MIDP y que además cuenta con emuladores de celulares.

4 METODOLOGÍA PARA EL DESARROLLO DEL SISTEMA

En este apartado se describen las metodologías y modelos que se tomaron en cuenta para el desarrollo de sistema o aplicación.

4.1 Metodologías y modelos base

Para el desarrollo e implementación del sistema se utilizará la metodología propuesta por Jacobson que de manera breve y resumida consiste en los pasos listados en la Tabla 7.

Etapa	Descripción
Modelo Análisis	Elaboración del Diagrama contextual del Sistema. Elaboración del Diagrama General de Casos de Uso. Diseño de Interfaces, Firmas y Mapas de Navegación. Elaboración del Diagrama General de Clases.
Modelo de Diseño	Elaboración de Diagramas de Casos de Uso particulares. Elaboración de la Descripción de Casos de Uso. Elaboración de Diagramas de Secuencias. Elaboración de Diagramas de Actividad o de Estado. Refinar Diagrama de Clases General.

Tabla 7: Etapas y pasos básicos de la metodología de Jacobson.

Además de la metodología propuesta por Jacobson se usarán ideas de modelos actuales de desarrollo de sistemas, tales como:

- *Dynamic System Development Method* (DSDM).
- *V Model*.

4.1.1 Dynamic System Development Method

El *Dynamic System Development Method* o DSDM, por sus siglas en inglés, que en español significa Método Dinámico de Desarrollo de Sistemas; consiste de manera sintetizada y resumida en las etapas que se describen en la Tabla 8.

Etapa	Descripción
Estudios	Realizar estudios de Factibilidad técnica. Realizar estudios de Factibilidad económica Realizar estudios de Factibilidad operacional. Realizar una planeación global. Administración de riesgos. Identificación de requerimientos prioritarios Definición de la arquitectura.

Etapa	Descripción
Prototipo Funcional	Definición de la funcionalidad de los requerimientos primarios. Calendarización, desarrollo y evaluación.
Desarrollo	Definición de la funcionalidad de los requerimientos no primarios. Calendarización, desarrollo y evaluación.
Implementación	Aprobación de lineamientos del cliente. Entrenamiento. Instalación y arranque. Retroalimentación.

Tabla 8: Etapas básicas del Dynamic System Development Method.

Este método de desarrollo es muy útil para sistemas que son sencillos y/o muy conocidos. Entre las ventajas que ofrece este método están:

- Realizar estudios de factibilidades, estos estudios ayudarán a identificar la tecnología que se empleara para el desarrollo del sistema o aplicación.

- La identificación de los requerimientos primarios y no primarios, es decir, indica que hay que iniciar con los componentes de mayor prioridad y dejar los no prioritarios para después.

- La calendarización de las actividades y la contemplación de evaluaciones.

4.1.2 V Model

El *V model* o en español modelo V, consiste básicamente en las etapas descritas en la Tabla 9.

Etapa	Descripción
Análisis de requerimientos	El análisis se debe estar cotejando con el diseño del sistema para que en el diseño se contemplen los requerimientos.
Diseño del sistema	Diseñar el sistema contemplando el análisis de requerimientos.
Diseño de la arquitectura	Diseñar la arquitectura considerando el diseño del sistema y las pruebas de diseño.
Diseño de los módulos	Diseñar y realizar pruebas al diseño de los módulos de la arquitectura.
Codificación	Codificar en base a las pruebas y diseños de los módulos.
Probar módulos	Probar cada uno de los módulos, cotejándolos con los diseños.
Integrar módulos	Integrar los módulos y cotejar con los diseños.
Probar sistema	Probar el sistema y cotejar con la arquitectura y diseño del mismo.
Pruebas de aceptación	Probar que el sistema cumpla con los requerimientos previamente analizados.

Tabla 9: Etapas básicas del V model.

Este modelo es muy bueno para sistemas en los que se exige un total apego a los requerimientos del sistema, permite, cotejar en todo momento los requerimientos con los diseños y los diseños con la arquitectura y codificación. En la Figura 3, se puede apreciar la interacción que hay entre las diferentes etapas del modelo V.

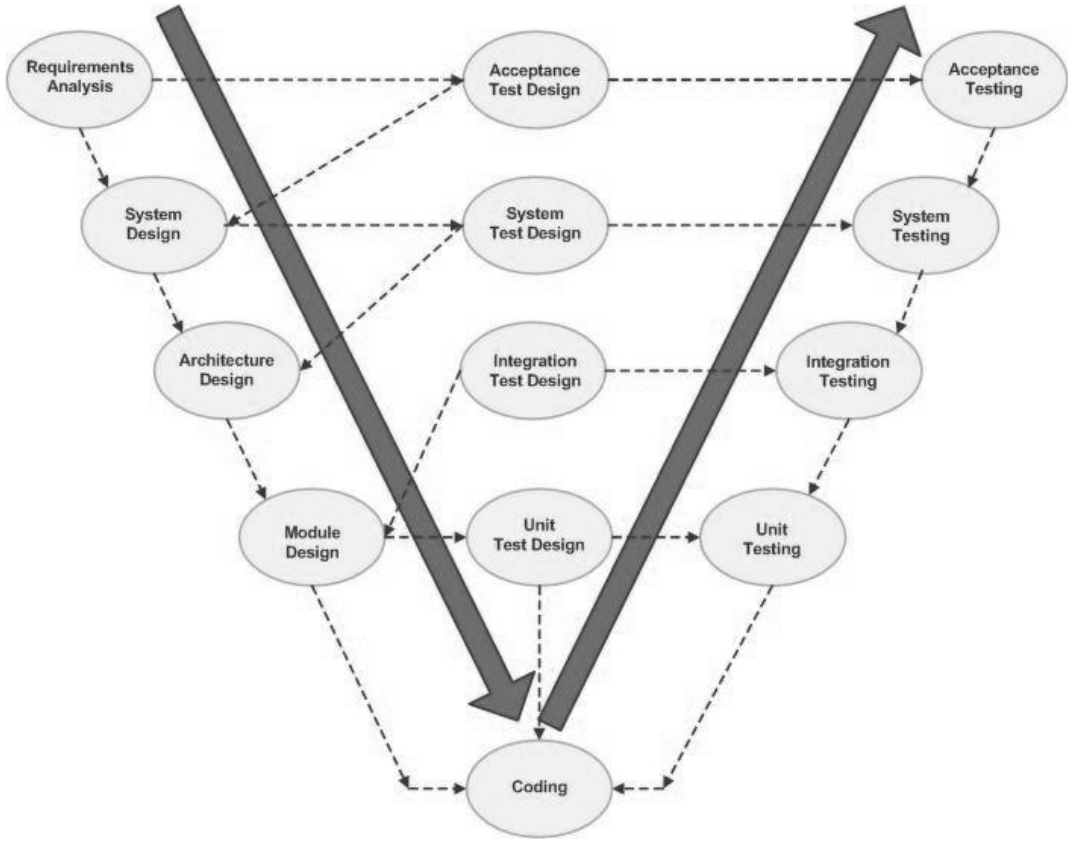

Figura 3: Esquema con la relación entre las etapas que contempla el modelo V.

4.2 Descripcion de la metodología elegida

En el apartado “Metodologías y modelos base” se describieron las metodologías que serán base para el desarrollo del producto final de este proyecto terminal.

Aunque se podría seguir una metodología y/o modelo de desarrollo, también es factible tomar algunas partes de las metodologías y mezclarlas según convenga para obtener mejores resultados.

Para la elaboración del producto final de este proyecto terminal se propone seguir las etapas y los pasos que se enlistan en la Tabla 10.

Etapa	Pasos que incluye
Análisis y Diseño	Elaboración de un esquema contextual del sistema. Elaboración de un diagrama general de casos de uso. Elaboración de los prototipos de las pantallas de mayor prioridad. Elaboración de los prototipos de pantalla no prioritarias. Elaboración del diagrama de navegación entre las pantallas prototipo. Completar y refinar diagrama de casos de uso. Elaborar descripción de casos de uso. Elaborar diagramas de clases de mayor abstracción. Diseñar de la arquitectura del sistema. Diseñar de los módulos y componente primarios. Diseñar de los componentes no primarios. Diseñar el esquema de flujo de datos. Completar y refinar el diagrama de clases general. División de las clases en paquetes por funcionalidad.
Codificación	Implementar las clases más simples y de mayor utilidad durante toda la aplicación. Realizar pruebas a las clases más simples y de mayor utilidad, para asegurar un buen funcionamiento. Implementar módulos auxiliares a los componentes principales, según su prioridad de importancia. Implementar los módulos o componentes de mayor prioridad basándose en los diseños. Implementar las clases por paquete comenzando del más simple y de mayor prioridad pasando por los complejos y de mayor prioridad hasta llegar a los no prioritarios y poco complejos. Mezclar las clases necesarias para generar los modules que constituyen la arquitectura.
Pruebas	Probar de forma independiente los módulos, para asegurar un buen funcionamiento. Cotejar los diseños con la implementación. Probar el funcionamiento de los módulos operando en conjunto. Instalar la aplicación y probar componentes en ambientes reales. De ser necesario realizar pruebas de los módulos en ambientes reales. Refinar módulos que no funcionen durante las pruebas.
Instalación	Instalar versiones estables en ambientes reales para ver el desempeño en diferentes ambientes.
Liberación	Obtención de la versión más estable. Creación de manuales de instalación. Creación de manuales de operación.

Tabla 10: Metodología propuesta para realizar el sistema.

Como se puede ver en Tabla 10 las etapas y pasos que se contemplan seguir para el desarrollo del sistema, están totalmente basados en los modelos y metodologías antes mencionadas.

Los paso se enlistaron de tal manera que se puedan realizar de forma secuencial si embargo, esto no quiere decir que este limitado a no regresar a pasos anteriores.

Durante el seguimiento de la metodología propuesta se tendrá siempre en mente la idea propuesta por el DSDM, de iniciar por las cosas de mayor prioridad y de dejar al final las no prioritarias, también se tendrá en mente la idea fundamental del modelo V, que es la de cotejar en varios momentos los productos con los requerimientos y diseños del sistema.

5 ARQUITECTURA

En este apartado se describirá la arquitectura que se empleara para el diseño e implementación del sistema.

Para elegir la arquitectura adecuada para el sistema, hay que contemplar que se pretende realizar una aplicación que tendrá la habilidad de comunicarse con otras aplicaciones en otros dispositivos, lo que nos remite a que la aplicación deberá de atender a varios clientes de forma simultánea. Y que además la arquitectura deberá de ser flexible o escalable, para que en algún momento determinado, se pueda modificar fácilmente para crecer o adaptarla a dispositivos diferentes.

Teniendo en mente lo anterior las arquitecturas que se usarán para la estructura del sistema serán las siguientes:

- Arquitectura Cliente-Servidor.
- Arquitectura Multicapa.

5.1 Arquitectura Cliente-Servidor

Esta arquitectura es la adecuada para los sistemas que están enfocados en atender a varios clientes. La arquitectura cliente - servidor contempla diferentes configuraciones, aquí se describen algunas.

Clientes Gordos – Servidor Delgado

En esta configuración los clientes realizan la mayor parte de la carga de trabajo y el servidor solo una mínima parte. Esta configuración es útil para aplicaciones en las que los clientes cuentan con recursos suficientes para manejar la mayor parte de la carga de trabajo y cuando el servidor tiene que atender a varios clientes, otorgándole al servidor mejor rendimiento. Ver Figura 4.

Clientes Delgados – Servidor Gordo

En esta configuración se considera que los clientes son dispositivos con pocos recursos por lo que se les delega una mínima carga de trabajo y en la que el servidor es un dispositivo que realiza la mayor parte de la carga. Ver Figura 4.

Para el caso de esta aplicación se considerará usar una configuración de servidor gordo y clientes delgados, por lo que el dispositivo que sea servidor tendrá que tener un conjunto de recursos de procesamiento más abundantes.

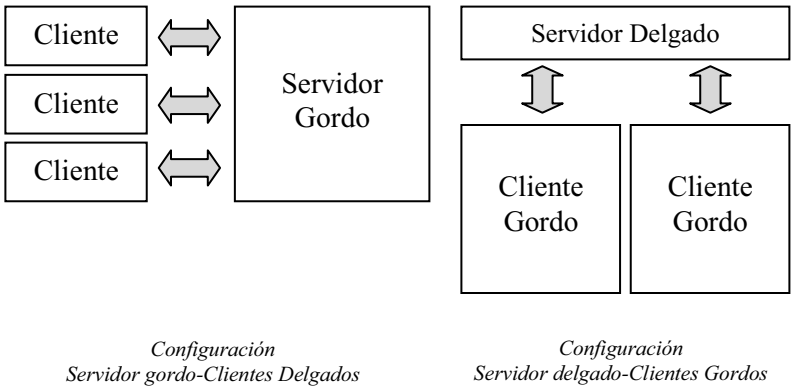

Figura 4: Configuraciones Cliente – Servidor.

5.2 Arquitectura multicapa

Esta arquitectura es la que permite dividir al sistema en varias capas independientes una de la otra siempre que cada capa proporcione una interfaz constante para la intercomunicación entre las mismas.

Se espera que el sistema cuente con al menos tres capas interdependientes que son las que se muestran en la siguiente lista:

- **Capa de vistas.** Es la capa que contendrá todas GUIs[21].
- **Capa de lógica.** Es la capa que estará encargada de toda la lógica de la aplicación.
- **Capa de comunicación.** Es la capa encargada de la transmisión y recepción de la información entre las aplicaciones.

5.3 Arquitectura del sistema

Como ya se reviso en las secciones referentes a la arquitectura, el sistema tendrá internamente una arquitectura cliente - servidor y una arquitectura multicapa.

Cada una de las capas estarán conformadas por distintos componente; componentes que fueron descritos en el aparado “Descripción de la aplicación”.

[21] Siglas en inglés para *Graphic User Interface* que en español significa Interfaz Gráfica de Usuario.

En la Figura 5 se muestra la arquitectura general de la aplicación, en la que se pueden aprecias las capas que conforma al sistema, así como los componentes más importantes que constituyen al sistema.

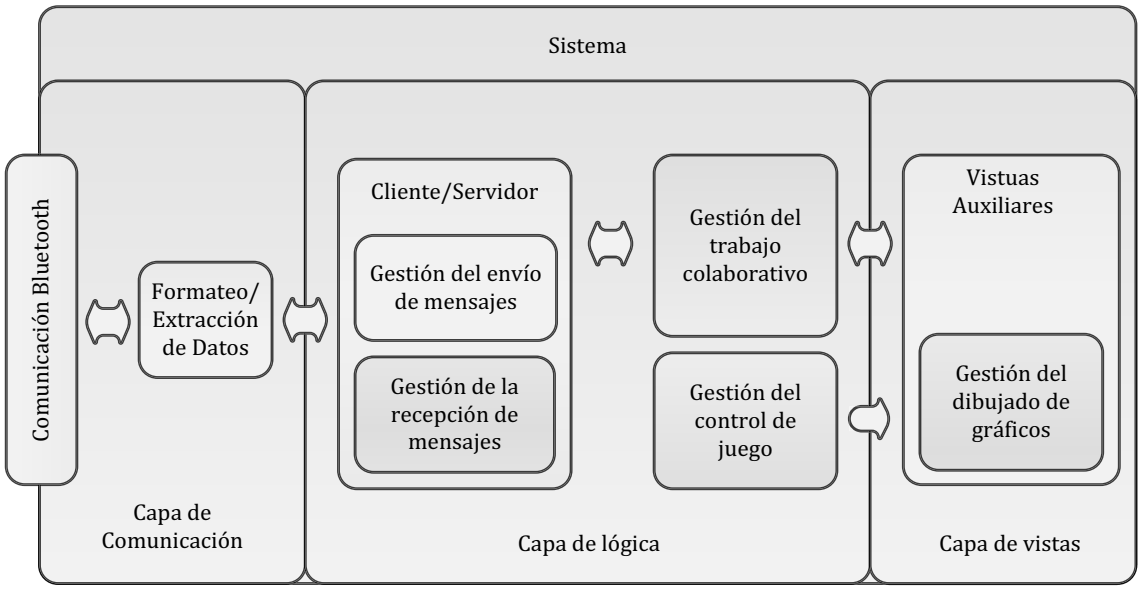

Figura 5: Arquitectura general de la aplicación.

6 DISEÑO DEL SISTEMA

En este apartado se muestran y describen todos los artefactos de diseño software que fueron utilizados durante el proceso de creación del sistema.

6.1 Pantallas prototipo

Las pantallas prototipo permiten pre visualizar a un sistema antes de que siquiera exista o antes de que siquiera se comience.

El realizar pantallas prototipo no interfiere en nada con el diseño del sistema o con su arquitectura, sino que más bien contribuye a tener un rumbo y a delimitar la funcionalidad final de lo que será el sistema.

6.1.1 Navegación entre pantallas

Para la realización de la aplicación se prevé que el sistema cuente con las pantallas que se describen en la Tabla 11.

Nombre de Pantalla	Descripción
Presentación	Pantalla que mostrará información general.
Selección de Modo de Juego	Pantalla que le permitirá al usuario decidir entre jugar solo o iniciar un juego colaborativo y/o participar en un juego.
Cantidad de Colaboradores	Pantalla que le permitirá al usuario decidir cuantos colaboradores ayudarán y jugarán, para el caso en el que haya decidido, iniciar un juego colaborativo.
Espera de Colaboradores	Pantalla que le permitirá enlistar los colaboradores que se vayan uniendo al juego.
Búsqueda de Dispositivos y Servicios	Pantalla que le permitirá al usuario buscar y enlistar a los dispositivos que se encuentren a su alcance.
Conexión	Pantalla que le permitirá conectarse con cualquiera de los dispositivos que cuente con el juego y estén en un modo tal que acepte conexiones entrantes para los participantes.
Imagen Objetivo	Pantalla que le permitirá observar la imagen que deberá ser armada durante el juego de rompecabezas.
Juego	Pantalla que le permitirá jugar un rompecabezas, tanto solo, como de forma colaborativa.

Tabla 11: Pantallas prototipo del sistema.

Las pantallas que se muestran en la Tabla 11 se describen de manera breve y de forma tal que permite imaginar la funcionalidad que tendrán, sin embargo, no permiten visualizar

la navegación entra las mismas por lo que en la Figura 6 se muestra un diagrama con el flujo la navegación entre las diferentes pantallas prototipo.

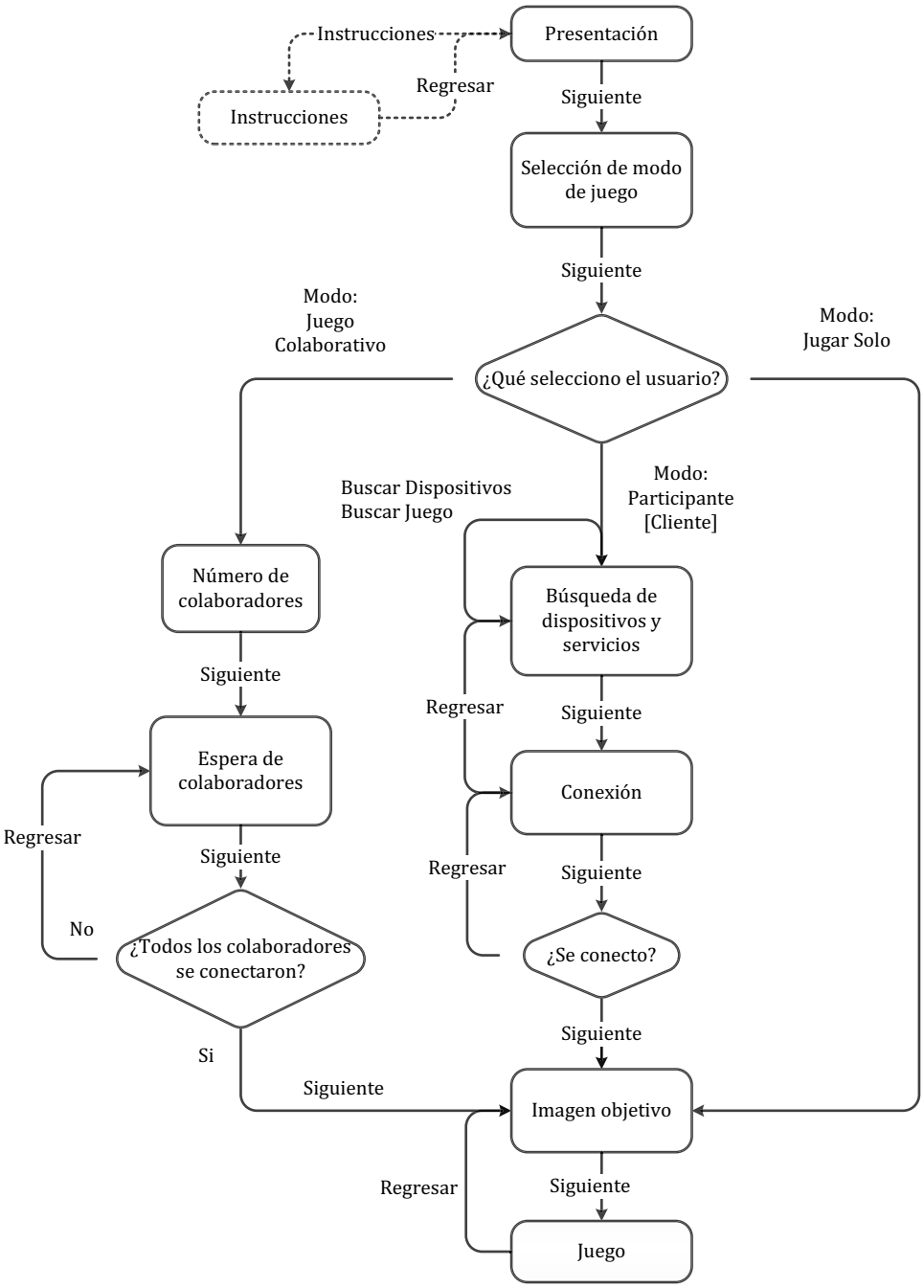

Figura 6: Flujo de navegación entre pantallas prototipo.

Cada conexión o flecha que una los elementos del diagrama de flujo de navegación que se muestra en la Figura 6 corresponderá eventualmente a un botón para que lo accione el usuario o por el sistema, en caso de ser necesario.

6.2 Prototipos

Para la aplicación móvil se diseñaron las siguientes pantallas prototipo.

6.2.1 Pantalla Presentación

Esta pantalla será la primera que se le mostrara al usuario. Su función principal será la de mostrar información general sobre la aplicación. En la Figura 7 se muestra una imagen de cómo se vera la pantalla.

Figura 7: Prototipo de la pantalla: Presentación.

6.2.2 Pantalla Modo de Juego

La función de esta pantalla le permitirá al usuario que seleccione un modo de juego. El sistema le mostrara tres opciones de modo de juego, que son las siguientes:

- Jugar solo. Esta opción remite al usuario a la pantalla que le muestra la imagen objetivo que deberá armar en el rompecabezas.

- Juego Colaborativo. Esta opción le permite al usuario configurar un juego colaborativo. La configuración consiste en indicar cuantos colaboradores se espera que ayuden en el armado o construcción del rompecabezas.

- Participante. Esta opción le permite al usuario participar en un juego colaborativo existente en un dispositivo remoto.

En la Figura 8 se muestra un prototipo de lo que será la pantalla.

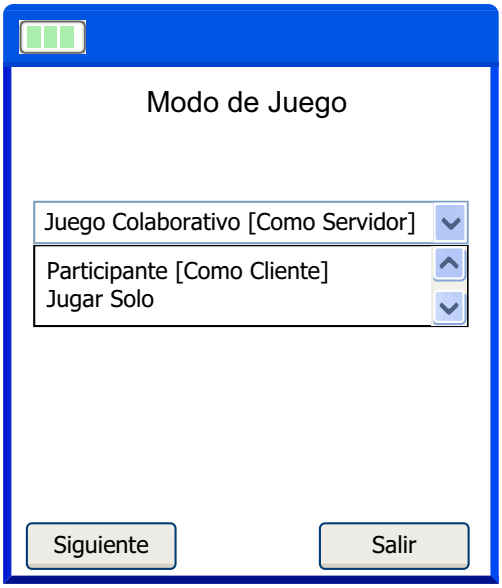

Figura 8: Prototipo de la pantalla: Modo de juego.

6.2.3 Pantalla Cantidad de Colaboradores

La función de esta pantalla es la de permitirle al usuario que seleccione la cantidad de colaboradores que estarán participando de forma remota. Esta pantalla se le mostrará al usuario siempre que el usuario haya seleccionado el modo de juego colaborativo, en la pantalla de la Figura 8.

Como se menciono en el apartado "Descripción de la aplicación" la aplicación estará diseñada para soportar a cuatro jugadores participando en una misma actividad; por lo que esta pantalla le permitirá al usuario seleccionar a máximo tres colaboradores remotos.

En la Figura 9 se muestra un prototipo de lo que será la pantalla.

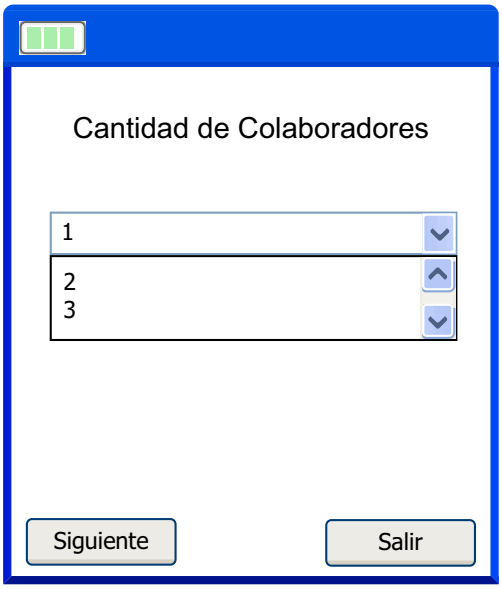

Figura 9: Prototipo de la pantalla: Cantidad de Colaboradores.

6.2.4 Pantalla Esperar a los Colaboradores

La función de esta pantalla, es estar a la espera de que los colaboradores se conecten para participar en un nuevo juego colaborativo; cuando un colaborador este conectado y listo, esta pantalla mostrará el nombre del dispositivo móvil, indicando que un colaborador esta listo para participar. La pantalla no permitirá avanzar a la siguiente, solo hasta que todos los colaboradores que se esperan lleguen, estén conectados y listos.

En la Figura 10 se muestra un prototipo de lo que será la pantalla.

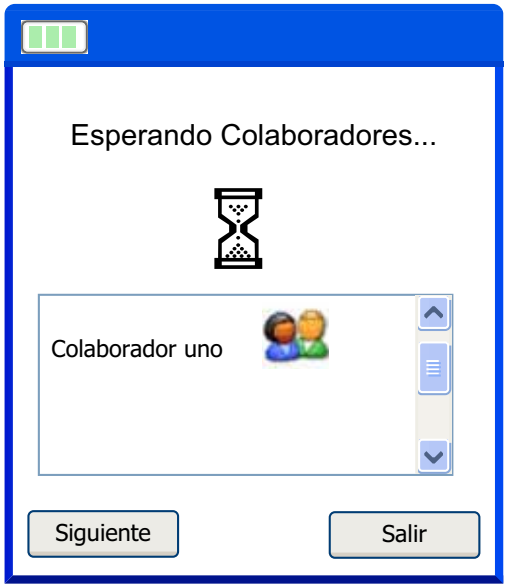

Figura 10: Prototipo de la pantalla: Esperar a los colaboradores.

6.2.5 Pantalla Buscar Dispositivos y Juego

Esta pantalla es la que se le muestra al usuario cuando ha seleccionado el modo de juego “participante”.

La función de esta pantalla es la de permitirle al usuario buscar los dispositivos que están a su alcance, para que posteriormente seleccione, entre los dispositivos cercanos, cual es el que esta en el modo “juego colaborativo”.

Un la Figura 11 se muestra un prototipo de lo que será la pantalla.

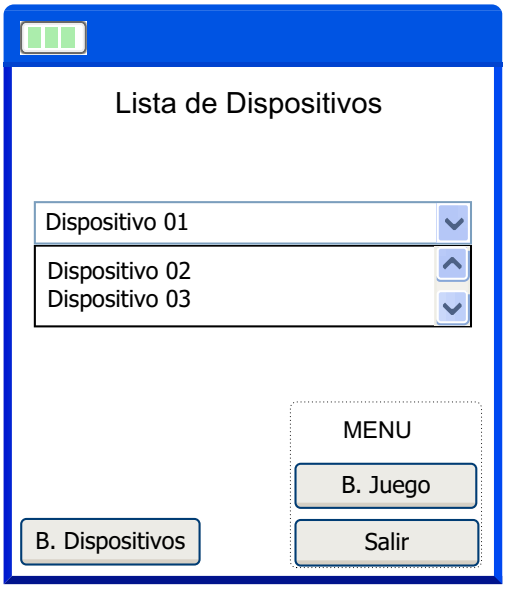

Figura 11: Prototipo de la pantalla: Buscar dispositivos y juego.

6.2.6 Pantalla Conexión con Dispositivo

La función de esta pantalla es la de avisar al usuario sobre si el sistema podrá o no conectarse.

En la Figura 12 se muestra un prototipo de lo que será la pantalla.

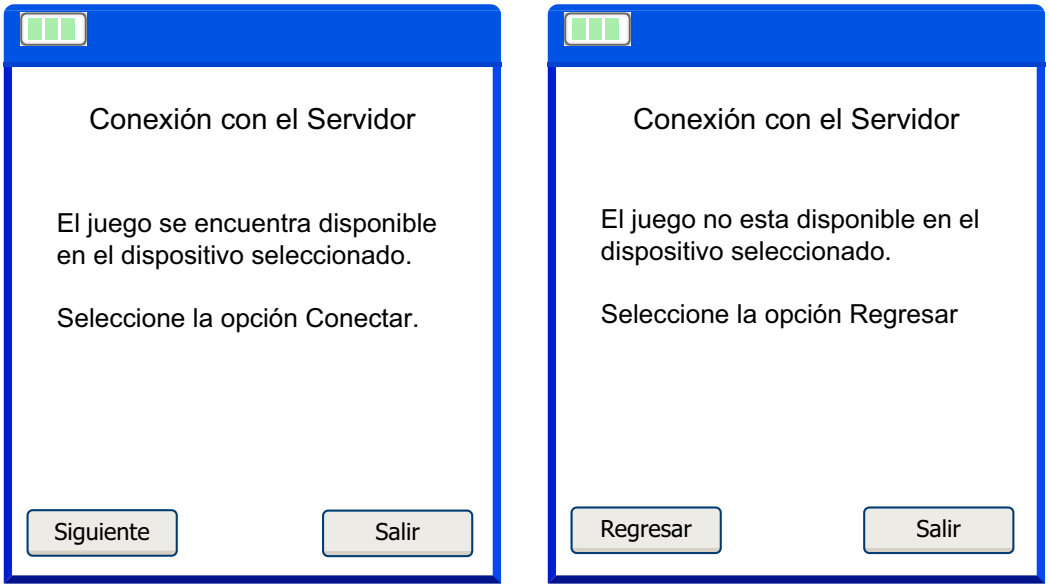

Figura 12: Prototipo de la pantalla: Conexión con dispositivo.

6.2.7 Pantalla Imagen Objetivo

La función de esta pantalla es la de mostrar al usuario la imagen que deberá de armar durante el juego de rompecabezas.

Durante el juego, el usuario podrá regresar a esta pantalla tantas veces como lo desee.

En la Figura 13 se muestra un prototipo de lo que será la pantalla.

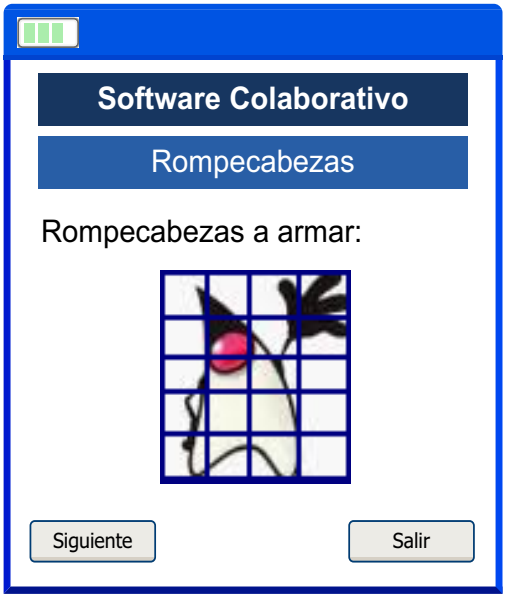

Figura 13: Prototipo de la pantalla: Imagen Objetivo.

6.2.8 Pantalla Juego de Rompecabezas

Esta pantalla es la principal de toda la aplicación. En esta pantalla se desarrollará el juego colaborativo, es en donde se podrán ver los movimientos realizados por todos los jugadores.

La pantalla tendrá tres áreas principales que son las siguientes:

- **Área principal del juego.** En esta área se podrá ver el avance de la completes del rompecabezas. Esta área contará con un cursor para seleccionar la posición en la que se colocará una pieza.

- **Área de piezas del rompecabezas.** En esta área se mostrarán todas las piezas que conforman al rompecabezas, las piezas estarán distribuidas de forma aleatoria, para cada jugador. Esta área contará con un cursor para seleccionar una pieza que se colocará en la posición que indique el cursor del área principal.

- **Área de notificaciones.** En esta área se mostraran todas las notificaciones referentes al juego, por ejemplo: quien movió determinada pieza, sí el movimiento fue o no correcto y en determinado momento el estado de completes del rompecabezas.

En la Figura 14 se muestra un prototipo.

Figura 14: Prototipo de la pantalla: Juego de rompecabezas.

6.3 Descripción y diagramas de casos de uso

En este apartado se formalizarán las pantallas prototipo, el diseño de los diagramas de casos de uso y se incluirán descripciones de los mismos.

6.3.1 Diagrama general de casos de uso

El diagrama general de casos de uso del sistema o aplicación se muestra en la Figura 15. En este diagrama se puede ver la interacción entre el usuario y los diferentes casos de uso del sistema, así como, las relaciones y/o dependencias de casos de uso.

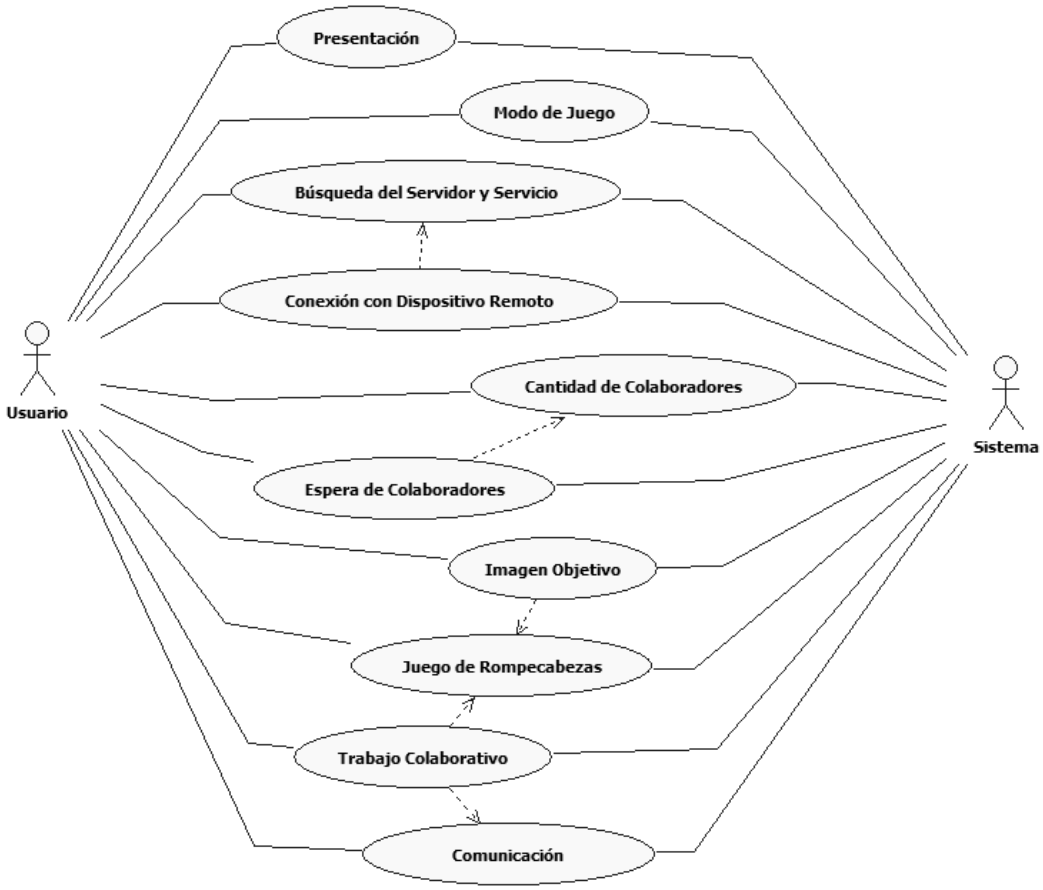

Figura 15: Diagrama general de casos de uso

6.3.2 Caso de Uso Presentación

Figura 16: Diagrama de caso de uso presentación.

Nombre:		**Presentación.**
Descripción:		El sistema le muestra al usuario información general sistema.
Casos de Uso Implicados:		Ninguno.
Precondiciones:		Ninguna.
Actores:		Usuario. Sistema.
Secuencia Principal		
Paso	**Actor**	**Descripción**
1	Sistema	Muestra el titulo “Rompecabezas”.
2	Sistema	Muestra el nombre de la universidad: “Universidad Autónoma Metropolitana Unidad Azcapotzalco”.
3	Sistema	Muestra el texto: “Proyecto Terminal”.
4	Sistema	Muestra el nombre de la carrera: “Ingeniería en Computación”.
5	Sistema	Muestra el nombre del autor: “Ortigosa Flores José Luis”.
6	Sistema	Muestra las opciones ‘Salir’ y ‘Siguiente’.
7	Usuario	Selecciona la opción ‘Siguiente’.
7.a	Usuario	Selecciona la opción ‘Salir’. Ir a **Secuencia Alterna 1 paso 7.a.1.**
8	Sistema	Va al caso de uso Modo de Juego.
9	Sistema	Fin del caso de uso.
Secuencia Alterna 1		
Paso	**Actor**	**Descripción**
7.a.1	Sistema	Finalizar la aplicación.
7.a.2	Sistema	Fin del caso de uso.
Validaciones:	Ninguna.	
Mensajes:	Ninguno.	
Comentarios:	Ninguno.	
Pantalla:	Figura 7: Prototipo de la pantalla: Presentación.	
Diagrama:	Figura 16: Diagrama de caso de uso presentación.	
Fecha:	26 de mayo de 2011	

6.3.3 Caso de Uso Modo de Juego

Figura 17: Diagrama de caso de uso modo de juego.

Nombre:		**Modo de Juego.**
Descripción:		El sistema le muestra al usuario los modos de juego disponibles en esta aplicación.
Casos de Uso Implicados:		Cantidad de Colaboradores. Búsqueda de Dispositivos. Juego de Rompecabezas.
Precondiciones:		Ninguna.
Actores:		Usuario. Sistema.
Secuencia Principal		
Paso	**Actor**	**Descripción**
1	Sistema	Muestra el titulo: “Modo de Juego”.
2	Sistema	Muestra una lista con las siguientes opciones: Opción ‘Juego Colaborativo [Como Servidor]’. Opción ‘Participante [Como Cliente]’. Opción ‘Jugar Solo’.
3	Sistema	Muestra las opciones ‘Siguiente’ y ‘Salir’.
4	Usuario	Selecciona la opción ‘Juego Colaborativo [Como Servidor]’ de la lista.
4.a	Usuario	Selecciona la opción ‘Participante [Como Cliente]’ de la lista. Ir a **Secuencia Alterna 1 paso 4.a.1.**
4.b	Usuario	Selecciona la opción ‘Jugar Solo’ de la lista. Ir a **Secuencia Alterna 2 paso 4.b.1.**
5	Usuario	Selecciona la opción ‘Siguiente’.
5.a	Usuario	Selecciona la opción ‘Salir’. Ir a **Secuencia Alterna 3 paso 5.a.1.**
6	Sistema	Va al caso de uso Cantidad de Colaboradores.
7	Sistema	Fin del Caso de Uso.
Secuencia Alterna 1		
Paso	**Actor**	**Descripción**
4.a.1	Usuario	Selecciona la opción ‘Siguiente’.
4.a.2	Sistema	Va al caso de uso Búsqueda de Dispositivos.
4.a.3	Sistema	Fin del Caso de Uso.
Secuencia Alterna 2		
Paso	**Actor**	**Descripción**
4.b.1	Usuario	Selecciona la opción ‘Siguiente’.

4.b.2	Sistema	Va al caso de uso Juego de Rompecabezas.
4.b.3	Sistema	Fin del Caso de Uso.
Secuencia Alterna 3		
Paso	**Actor**	**Descripción**
5.a.1	Sistema	Finaliza la aplicación.
5.a.2	Sistema	Fin del Caso de Uso.
Validaciones:	Ninguna.	
Mensajes:	Ninguno.	
Comentarios:	Ninguno.	
Pantalla:	Figura 8: Prototipo de la pantalla: Modo de juego.	
Diagrama:	Figura 17: Diagrama de caso de uso modo de juego.	
Fecha:	26 de mayo de 2011	

6.3.4 Caso de Uso Cantidad de Colaboradores

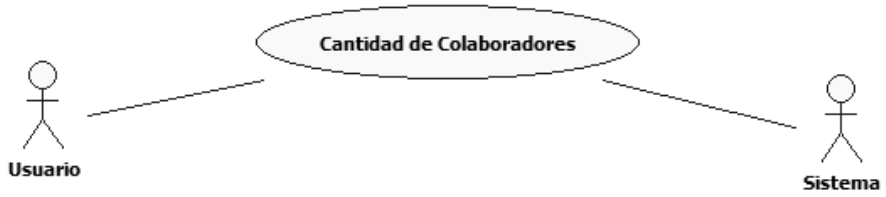

Figura 18: Diagrama de caso de uso cantidad de colaboradores.

Nombre:	Cantidad de Colaboradores.
Descripción:	El sistema le permite al usuario indicar la cantidad de colaboradores remotos que se espera participen en un juego colaborativo.
Casos de Uso Implicados:	Espera de Dispositivos.
Precondiciones:	El usuario selecciono en el caso de uso Modo de Juego la opción 'Juego Colaborativo [Como Servidor]'.
Actores:	Usuario. Sistema.

Secuencia Principal		
Paso	**Actor**	**Descripción**
1	Sistema	Muestra el titulo: "Cantidad de Colaboradores".
2	Sistema	Muestra una lista con las siguientes opciones: Opción '1'. Opción '2'. Opción '3'.
3	Sistema	Muestra las opciones 'Siguiente' y 'Salir'.
4	Usuario	Selecciona cualquiera de las tres opciones de la lista.
5	Usuario	Selecciona la opción 'Siguiente'.
5.a	Usuario	Selecciona la opción 'Salir'. Ir a secuencia alterna 1, paso 5.a.1.
6	Sistema	Almacena temporalmente la opción, de la lista, seleccionada por el usuario.

7	Sistema	Va al caso de uso Espera de Dispositivos.
8	Sistema	Fin del Caso de Uso.
Secuencia Alterna 1		
Paso	**Actor**	**Descripción**
5.a.1	Sistema	Finaliza la aplicación
5.a.2	Sistema	Fin del Caso de Uso.
Validaciones:	Ninguna.	
Mensajes:	Ninguno.	
Comentarios:	Ninguno.	
Pantalla:	Figura 9: Prototipo de la pantalla: Cantidad de Colaboradores.	
Diagrama:	Figura 18: Diagrama de caso de uso cantidad de colaboradores.	
Fecha:	26 de mayo de 2011	

6.3.5 Caso de uso Espera de Colaboradores

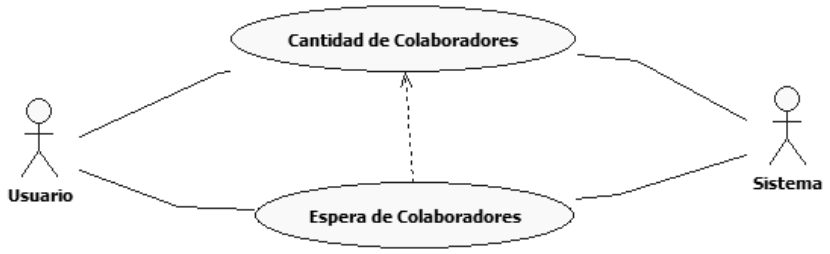

Figura 19: Diagrama de caso de uso espera de colaboradores.

Nombre:	**Espera de Colaboradores.**
Descripción:	El sistema espera a que los colaboradores se conecten, en el momento en el que llegue un colaborador el sistema lo registra.
Casos de Uso Implicados:	Imagen Objetivo. Cantidad de Colaboradores.
Precondiciones:	El usuario selecciono, en el caso de uso Cantidad de Colaboradores, un número de colaboradores a esperar a que se conecten.
Actores:	Usuario. Sistema.

Secuencia Principal		
Paso	**Actor**	**Descripción**
1	Sistema	Muestra el título: “Esperando Colaboradores”.
2	Sistema	Pública el servicio.
3	Sistema	Espera por peticiones del servicio.
4	Sistema	Muestra una relación con los dispositivos conectados. Por cada dispositivo se muestra: -Nombre del dispositivo.
5	Sistema	Muestra las opciones ‘Siguiente’ y ‘Salir’.
6	Sistema	Atiende una petición de un cliente.
7	Sistema	Almacena la conexión entrante.
8	Sistema	Sí hay más peticiones de cliente, ir a **Secuencia Principal, paso 3**.
9	Usuario	Selecciona la opción ‘Siguiente’.

9.a	Usuario	Selecciona la opción 'Salir'. Ir a **Secuencia Alterna 1 paso 9.a.1.**
10	Sistema	Sistema coteja sí la cantidad de colaboradores esperados es igual a la cantidad de elementos de la relación con los dispositivos conectados. Sí son iguales, ir a **Secuencia Principal, paso 11**. Sí no, ir a **Secuencia principal, paso 3.**
11	Sistema	Fin del caso de uso. Ir a caso de uso Imagen Objetivo.
Secuencia Alterna 1		
Paso	**Actor**	**Descripción**
9.a.1	Sistema	Finalizar la aplicación.
9.a.2	Sistema	Fin del caso de uso.
Validaciones:	Ninguna.	
Mensajes:	Ninguno.	
Comentarios:	El sistema estará en espera, siempre que el usuario no salga de la aplicación o a menos que el hardware no lo permita.	
Pantalla:	Figura 10: Prototipo de la pantalla: Esperar a los colaboradores.	
Diagrama:	Figura 19: Diagrama de caso de uso espera de colaboradores.	
Fecha:	26 de mayo de 2011	

6.3.6 Caso de Uso Búsqueda del Servidor o Servicio

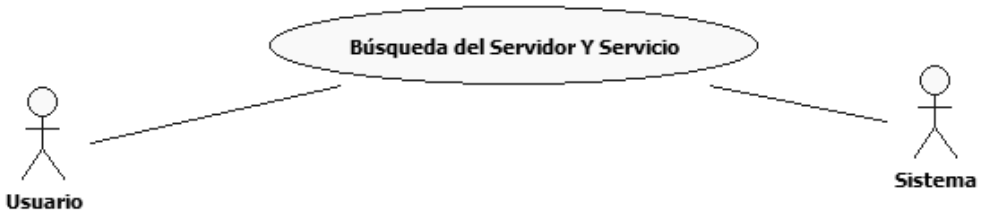

Figura 20: Diagrama de caso de uso búsqueda del servidor o servicio.

Nombre:	**Búsqueda del Servidor y Servicio.**
Descripción:	El sistema le permite al usuario buscar dispositivos que estén dentro del alcance del sistema, para así el usuario pueda seleccionar cual de ellos es el que esta en modo servidor. Es decir que seleccione al dispositivo cuyo usuario selecciono en el caso de uso **Modo de Juego**, el modo 'Juego Colaborativo [Como Servidor]'
Casos de Uso Implicados:	Conexión con el Dispositivo Remoto.
Precondiciones:	En el caso de uso **Modo de Juego** el usuario ha seleccionado el modo de juego 'Participante [Como Cliente]'
Actores:	Usuario. Sistema.
Secuencia Principal	
Paso	**Actor**	**Descripción**
1	Sistema	Muestra el titulo: "Listado de Dispositivos".
2	Sistema	Muestra una **relación con los dispositivos localizados**. Por cada registro

		se muestra el nombre del dispositivo.
3	Sistema	Muestra las opciones ‘Buscar Dispositivos’, ‘Buscar Juego’ y ‘Salir’.
4	Usuario	Selecciona la opción ‘Buscar Dispositivo’.
4.a	Usuario	Selecciona la opción ‘Buscar Juego’. Ir a **secuencia alterna 1, paso 4.a.1.**
4.b	Usuario	Selecciona la opción ‘Salir’. Ir a **secuencia alterna 2, paso 4.b.1.**
5	Sistema	Inicia la búsqueda de dispositivos que estén a su alcance.
6	Sistema	Llena la **relación con los dispositivos localizados.**
7	Sistema	Agrega un registro a la **relación con los dispositivos conectados** y lo muestra.
7.a	Usuario	Sí la lista esta vacía. Va a **secuencia principal, paso 4.**
8	Usuario	Selecciona a un dispositivo de la lista.
9	Usuario	Selecciona la opción ‘Buscar Juego’.
9.a	Usuario	Selecciona la opción ‘Buscar Dispositivos’. Ir a **secuencia principal, paso 4.**
9.b	Usuario	Selecciona la opción ‘Salir’.
10	Sistema	Busca en el dispositivo seleccionado el juego, es decir si en el dispositivo se localiza el servicio, para el juego de rompecabezas.
11	Sistema	Va al caso de uso **Conexión con Dispositivo Remoto.**
12	Sistema	Fin del Caso de Uso.
Secuencia Alterna 1		
Paso	**Actor**	**Descripción**
4.a.1	Sistema	Si no hay un dispositivo seleccionado, no hace nada.
4.a.2	Sistema	Ir a **secuencia principal, paso 4.**
Secuencia Alterna 2		
Paso	**Actor**	**Descripción**
4.b.1	Sistema	Finaliza la aplicación.
4.b.2	Sistema	Fin del Caso de Uso.
Secuencia Alterna 3		
Paso	**Actor**	**Descripción**
9.b.1	Sistema	Finalizar la aplicación.
9.b.2	Sistema	Fin del Caso de Uso.
Validaciones:	Ninguna.	
Mensajes:	Ninguno.	
Comentarios:	Ninguno.	
Pantalla:	Figura 11: Prototipo de la pantalla: Buscar dispositivos y juego.	
Diagrama:	Figura 20: Diagrama de caso de uso búsqueda del servidor o servicio.	
Fecha:	27 de mayo de 2011	

6.3.7 Caso de Uso Conexión con Dispositivo Remoto

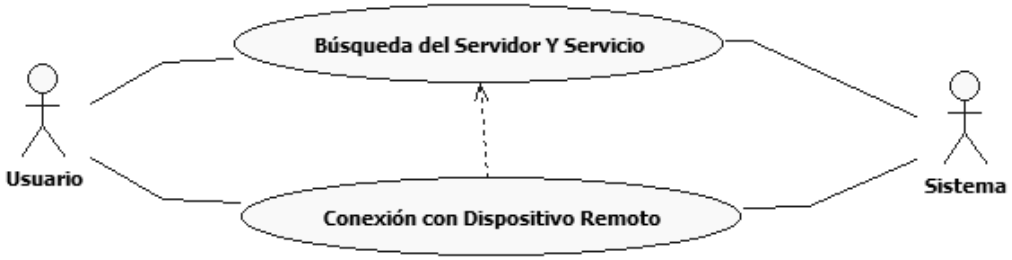

Figura 21: Diagrama de caso de uso conexión con dispositivos remoto.

Nombre:	**Conexión con Dispositivo Remoto.**
Descripción:	El sistema le permite al usuario establecer una conexión con el servidor, siempre que el dispositivo seleccionado en el caso de uso Búsqueda del Servidor y Servicio haya encontrado el servicio en el dispositivo seleccionado.
Casos de Uso Implicados:	Búsqueda del Servidor y Servicio. Imagen Objetivo.
Precondiciones:	El caso de uso Búsqueda del Servidor y Servicio se llevo a cabo.
Actores:	Usuario. Sistema.

Secuencia Principal		
Paso	**Actor**	**Descripción**
1	Sistema	Muestra el titulo: “Conexión con el Servidor”.
2	Sistema	Sí en el caso de uso Búsqueda del Servidor y Servicio, se logro encontrar el juego, muestra el mensaje: “MSG-01”. Sí no entonces, muestra el mensaje: “MSG-02”.
3	Sistema	Obtiene la dirección del dispositivo que cuenta con el servicio.
4	Sistema	Si se logro obtener la dirección entonces, muestra la opción ‘Conectar’ Sí no entonces, muestra la opción ‘Regresar’.
5	Sistema	Muestra la opción ‘Salir’.
6	Usuario	Sí esta la opción ‘Conectar’. Selecciona la opción ‘Conectar’.
6.a	Usuario	Sí esta la opción ‘Regresar’. Selecciona la opción ‘Regresar’. Ir a caso de uso Búsqueda del Servidor y Servicio.
6.b	Usuario	Selecciona la opción ‘Salir’. Ir a **secuencia alterna 1, paso 6.b.1.**
7	Sistema	Va al caso de uso Imagen Objetivo.
8	Sistema	Fin del Caso de Uso.

Secuencia Alterna 1		
Paso	**Actor**	**Descripción**
6.b.1	Sistema	Finaliza la aplicación.
6.b.2	Sistema	Fin del Caso de Uso.

Validaciones:	Ninguna.
Mensajes:	MSG-01: ‘El juego se encuentra disponible en el dispositivo seleccionado. Seleccione la opción Conectar’. MSG-02: ‘El juego no esta disponible en el dispositivo seleccionado. Seleccione la opción Regresar’.
Comentarios:	Ninguno.

Pantalla:	Figura 12: Prototipo de la pantalla: Conexión con dispositivo.
Diagrama:	Figura 21: Diagrama de caso de uso conexión con dispositivos remoto.
Fecha:	27 de mayo de 2011

6.3.8 Imagen Objetivo

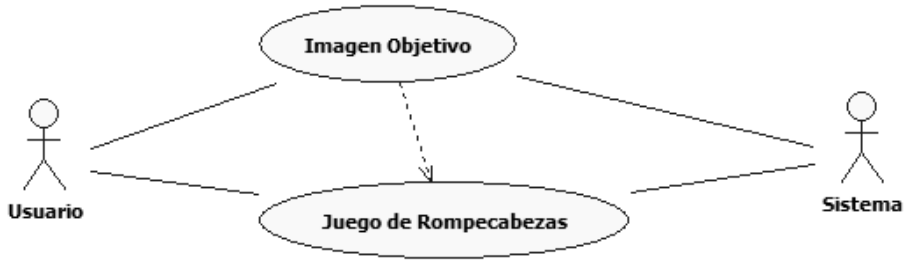

Figura 22: Diagrama de caso de uso imagen objetivo.

Nombre:		**Imagen Objetivo.**
Descripción:		El sistema le muestra al usuario la imagen que debe ser armada en el rompecabezas.
Casos de Uso Implicados:		Juego de Rompecabezas.
Precondiciones:		Ninguna.
Actores:		Usuario. Sistema.
Secuencia Principal		
Paso	**Actor**	**Descripción**
1	Sistema	Muestra el titulo: “Software Colaborativo”.
2	Sistema	Muestra el subtitulo: “Rompecabezas”.
3	Sistema	Muestra el texto: “Imagen a Formar en el Rompecabezas”.
4	Sistema	Muestra la imagen que deberá ser armada durante el juego.
5	Sistema	Muestra las opciones ‘Siguiente’ y ‘Salir’.
6	Usuario	Selecciona la opción ‘Siguiente’.
6.a	Usuario	Selecciona la opción ‘Salir’. Ir a **secuencia alterna 1 paso 6.a.1.**
7	Sistema	Va al caso de uso Juego de Rompecabezas.
8	Sistema	Fin del Caso de Uso.
Secuencia Alterna 1		
Paso	**Actor**	**Descripción**
6.a.1	Sistema	Finaliza la aplicación.
6.a.2	Sistema	Fin del Caso de Uso.
Validaciones:	Ninguna.	
Mensajes:	Ninguno.	
Comentarios:	Ninguno.	
Pantalla:	Figura 13: Prototipo de la pantalla: Imagen Objetivo.	
Diagrama:	Figura 22: Diagrama de caso de uso imagen objetivo.	

Fecha:	27 de mayo de 2011

6.3.9 Caso de Uso Juego de Rompecabezas

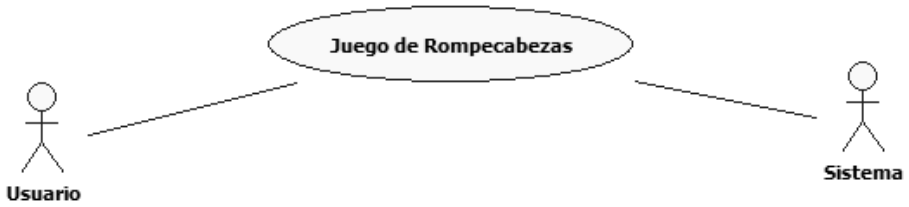

Figura 23: Diagrama de caso de uso juego de rompecabezas.

Nombre:	**Juego de Rompecabezas.**
Descripción:	Caso de Uso principal, le permite al usuario jugar en los tres modos de juego disponibles de esta aplicación.
Casos de Uso Implicados:	Imagen Objetivo. Trabajo Colaborativo. Comunicación Bluetooth.
Precondiciones:	Ninguna.
Actores:	Usuario. Sistema.

Secuencia Principal		
Paso	**Actor**	**Descripción**
1	Sistema	Muestra el titulo: “Software Colaborativo”.
2	Sistema	Muestra un **área principal** en la que el usuario podrá ver el avance del rompecabezas. Esta área muestra el avance del armado del rompecabezas. Dentro de esta área muestra un cursor que le permite al usuario seleccionar un área específica, dentro del área.
3	Sistema	Muestra una **barra con todas las piezas** que conforman al rompecabezas, distribuidas cuasi-aleatoriamente. Dentro de esta área muestra un cursor que le permite al usuario seleccionar una pieza para ser colocada en el área principal.
4	Sistema	Muestra una **barra de estado** en la que se visualizarán todos los mensajes referentes al estado del juego.
5	Sistema	Muestra las opciones ‘Match’, ‘Regresar’ y ‘Salir’.
6	Usuario	Selecciona la opción ‘Regresar’. Ir al caso de uso **Imagen Objetivo.**
6.a	Usuario	Selecciona la opción ‘Salir’. Ir a **secuencia alterna 1 paso 6.a.1.**
7	Usuario	Selecciona una pieza de la **barra de piezas.**
8	Usuario	Selecciona un área específica en el área principal, lugar donde se desea colocar la pieza seleccionada en la **barra de piezas**.
9	Usuario	Selecciona la opción ‘Match’.
10	Sistema	Realiza validación: VAL-01.
11	Sistema	Sí validación VAL-01 es verdadera, muestra mensaje: MSG-01, en la

		barra de estado. Sí validación VAL-01 es falsa, muestra mensaje: MSG-02, en la **barra de estado.**
12	Sistema	Sí el usuario selecciono en el caso de uso **Modo de Juego** la opción 'Participante [Como Cliente]' entonces, envía el movimiento al servidor (caso de uso **Trabajo Colaborativo**) a través del caso de uso **Comunicación**. Sí el usuario selecciono en el caso de uso **Modo de Juego** la opción 'Juego Colaborativo [Como Servidor]' entonces, el movimiento lo envía al caso de uso **Trabajo Colaborativo**.
13	Sistema	Sí el usuario selecciono en el caso de uso **Modo de Juego** 'Participante [Como Cliente]' o 'Juego Colaborativo [Como Servidor]'. El sistema recupera los movimientos enviados por el servidor a través del caso de uso **Comunicación** y los ejecuta, para posteriormente mostrarlos en el área principal.
14	Sistema	Muestra en la barra de estado el mensaje: MSG-03.
15	Sistema	Fin del Caso de Uso.
Secuencia Alterna 1		
Paso	**Actor**	**Descripción**
6.a.1	Sistema	Finaliza la aplicación.
6.a.2	Sistema	Fin del Caso de Uso.

Validaciones:	VAL-01: El sistema verifica que la pieza seleccionada en la barra de piezas vaya en el lugar especificado con el cursor del área principal.
Mensajes:	MSG-01: 'Bien!!'. MSG-02: 'Incorrecto!!'. MSG-03: '[Nombre del Jugador] pieza [Número de pieza]'
Comentarios:	[Nombre del Jugador]: Cambia dependiendo del jugador que haya realizado el movimiento. [Número de Pieza]: Cambia dependiendo del número de pieza que haya colocado correctamente un jugador. Servidor se refiere al caso de uso Trabajo Colaborativo.
Pantalla:	Figura 14: Prototipo de la pantalla: Juego de rompecabezas.
Diagrama:	Figura 23: Diagrama de caso de uso juego de rompecabezas.
Fecha:	27 de mayo de 2011

6.3.10 Comunicación

Figura 24: Diagrama de caso de uso comunicación.

Nombre:		**Comunicación.**
Descripción:		Caso de uso encargado de la comunicación.
Casos de Uso Implicados:		Juego de Rompecabezas. Trabajo Colaborativo.
Precondiciones:		Ninguna.
Actores:		Sistema.
Secuencia Principal		
Paso	**Actor**	**Descripción**
1	Sistema	Inicia una cola de movimientos.
2	Sistema	Esta a la escucha de los movimientos que el servidor envíe, de forma indefinida.
3	Sistema	Cada movimiento recibido por el servidor es almacenado en la cola de movimientos.
4	Sistema	Toma los movimientos enviados por los casos de uso **Juego de Rompecabezas y Trabajo Colaborativo** y los envía.
5	Sistema	No finaliza este caso de uso, sino hasta que el caso de uso **Juego de Rompecabezas** lo haga.
6	Sistema	Fin del Caso de Uso.
Secuencia Alterna		
Paso	**Actor**	**Descripción**
Validaciones:	Ninguna.	
Mensajes:	Ninguno.	
Comentarios:	Lo aconsejable es que se lance este caso de uso en un hilo independiente.	
Pantalla:	Ninguna.	
Diagrama:	Figura 24: Diagrama de caso de uso comunicación.	
Fecha:	27 de mayo de 2011.	

6.3.11 Caso de uso Trabajo Colaborativo

Figura 25: Diagrama de caso de uso trabajo colaborativo.

Nombre:		**Trabajo Colaborativo.**
Descripción:		Caso de uso encargado de gestionar todo lo referente al envío y recepción de movimientos de todos los jugadores.
Casos de Uso Implicados:		Juego de Rompecabezas. Comunicación.
Precondiciones:		El modo de juego seleccionado en el caso de uso **Modo de Juego** es 'Juego Colaborativo [Como Servidor]'
Actores:		Usuario. Sistema.
Secuencia Principal		
Paso	**Actor**	**Descripción**
1	Sistema	Inicia una cola de movimientos, cola que almacenará todos y cada uno de los movimientos enviados por los jugadores.
2	Sistema	Esta a la escucha de los movimientos enviados por los jugadores, cada movimiento es almacenado en la cola de movimientos.
3	Sistema	Esta distribuyendo todos y cada uno de los movimientos que estén encolados.
4	Sistema	No finaliza este caso de uso, sino hasta que el caso de uso **Juego de Rompecabezas** lo haga.
5	Sistema	Fin del Caso de Uso.
Secuencia Alterna		
Paso	**Actor**	**Descripción**
Validaciones:	Ninguna.	
Mensajes:	Ninguno.	
Comentario:	Lo aconsejable es que la recepción y la distribución estén en hilos separados.	
Pantalla:	Ninguna.	
Diagrama:	Figura 25: Diagrama de caso de uso trabajo colaborativo.	
Fecha:	27 de mayo de 2011	

6.4 Diseño esquemático del sistema

El principal componente de la aplicación es el módulo encargado de la gestión del trabajo colaborativo así como algunos módulos auxiliares al mismo.

La gestión del trabajo colaborativo constará de los siguientes subcomponentes:

- El subcomponente encargado de procesar los movimientos realizados por todo los jugadores. El procesamiento consistirá en recibir los movimientos de todos los jugadores y de repartirlos.
- El subcomponente que representará a un colaborador remoto.
- El subcomponente que represente a un colaborador o jugador local.

Otro aspecto importante es el diseño del flujo de datos que tendrá la aplicación.

6.4.1 Diseño del colaborador remoto

El Colaborador Remoto será el componente que represente a un jugador que se encuentra remotamente y que participará eventualmente en un juego.

Componentes

El Colaborador Remoto será la entidad que estará del lado de los jugadores remotos que están participando en un juego colaborativo, esta entidad les permitirá a los colaboradores remotos almacenar en una cola todos los movimientos distribuidos por el servidor y que corresponden a movimientos realizados por otros jugadores.

Básicamente este componente constará de lo siguiente:

- Una cola en la cual serán almacenados todos los movimientos provenientes del servidor.
- Un hilo que estará a la escucha de movimientos entrantes.
- Un hilo que estará aplicando todos los movimientos que estén en la cola.

Descripción del Funcionamiento

Habrá un hilo que estará a la escucha de movimientos entrantes. Cada uno de los movimientos entrantes será almacenado en una cola de movimientos.

Habrá otro hilo encargado de sacar los movimientos que estén en la cola, de extraer la información que contiene y de ejecutar el movimiento para que posteriormente el hilo encargado del dibujado del juego los refleje en pantalla.

Los movimientos de los colaboradores remotos serán enviados al servidor accionado por un evento del teclado.

En la Figura 26 se muestra un esquema con los hilos y componentes que intervienen en el colaborador remoto.

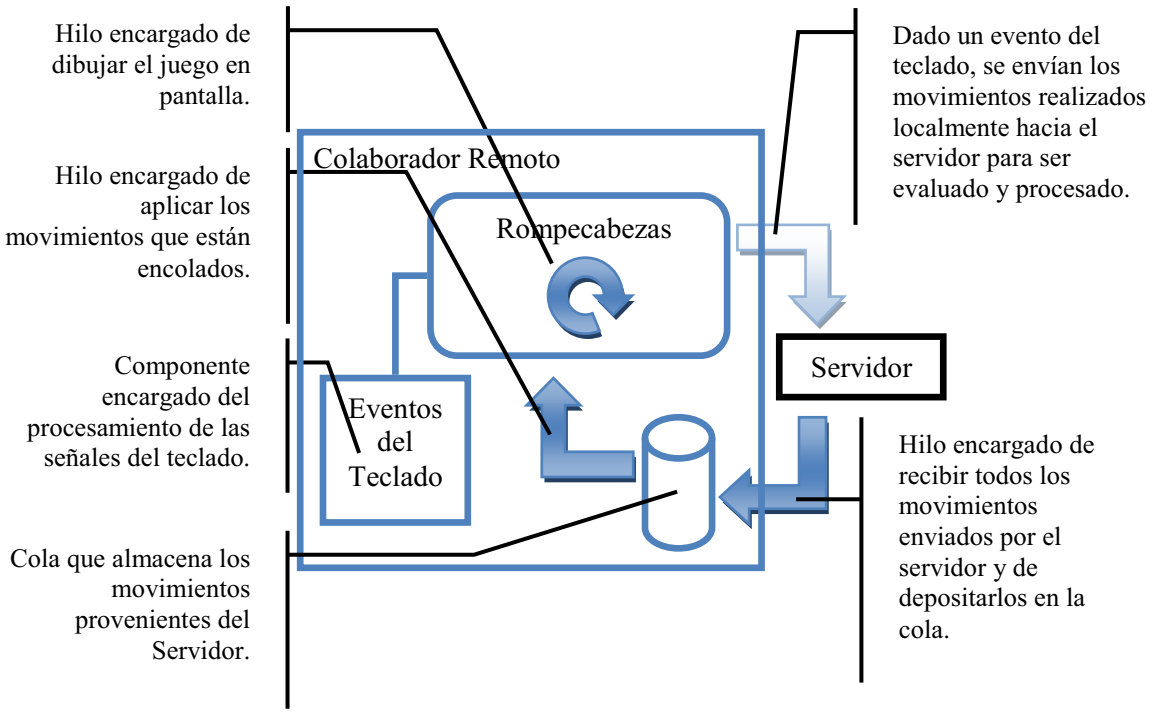

Figura 26: Hilos y componentes del colaborador remoto.

6.4.2 Diseño del colaborador o jugador local

El colaborador Local será el componente que representa al jugador local, dicho de otra manera es el jugador que se ejecuta en el mismo dispositivo donde se ejecuta el servidor.

Componentes y Descripción del Funcionamiento

A diferencia del Colaborador Remoto este componente no tiene que enviar la información a través del canal bluetooth, por lo que este componente estará constituido por dos colas de movimientos una con los movimientos salientes y otra con los movimientos entrantes, en la primera el jugador local colocará sus movimientos realizados, la segunda es la cola en la que el servidor colocará los movimientos que reparta. En la Figura 27 se puede ver de forma esquemática como interactúan los componentes.

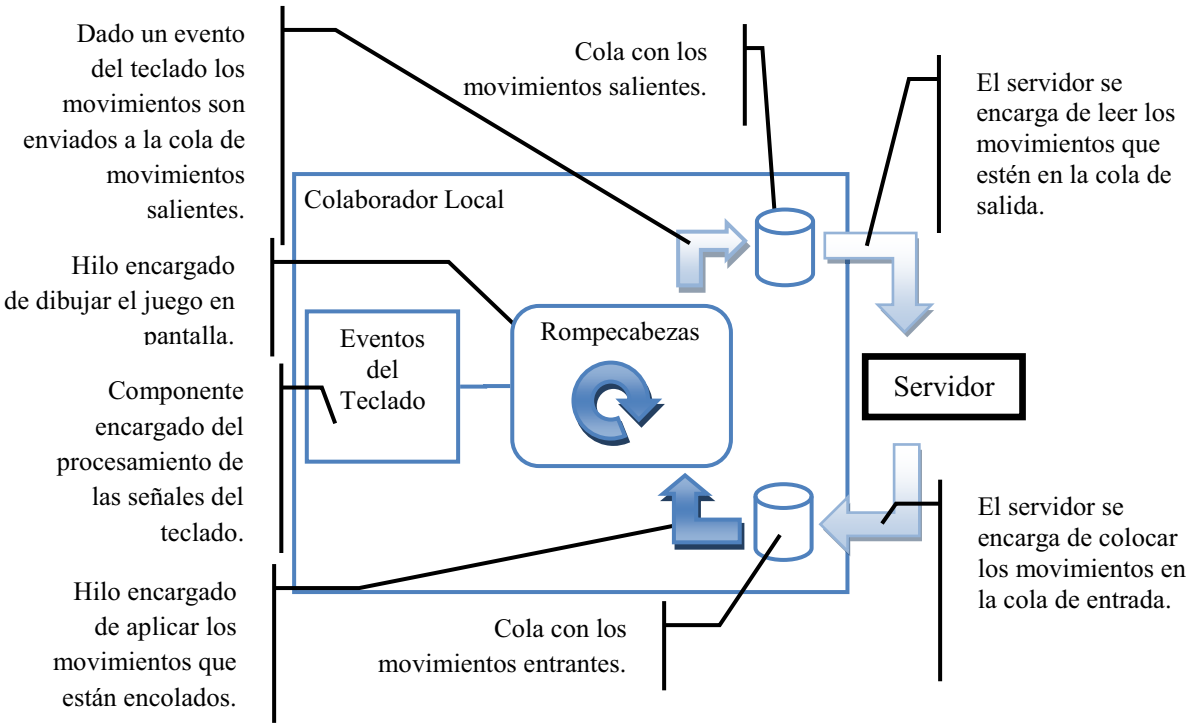

Figura 27: Hilos y componentes que intervienen en el colaborador local.

6.4.3 Diseño del trabajo colaborativo

Componente encarga de la recepción y distribución de movimientos desde y hacia todos los colaboradores respectivamente.

Componentes

La gestión del trabajo colaborativo consta principalmente de los siguientes elementos:

- Una cola con los movimientos entrantes.
- Un hilo que se encarga de distribuir a todos los colaboradores los movimientos.
- Varios hilos que atienden a cada colaborador o jugador a excepción del colaborador local.

Descripción del Funcionamiento

Los hilos que atienden a los colaboradores recibiendo los mensajes que envían, toman el mensaje y lo colocan en la cola de mensajes.

Un hilo estará encargado de sacar los movimientos de la cola y cada uno de esos movimientos será distribuido y enviado a todos los colaboradores.

La cola de movimientos va ser accedida por varios hilos, por lo que esta cola será un objeto mutuamente excluyente.

En la Figura 28 se muestra de manera esquemática de los hilos y componentes que interviene en trabajo colaborativo.

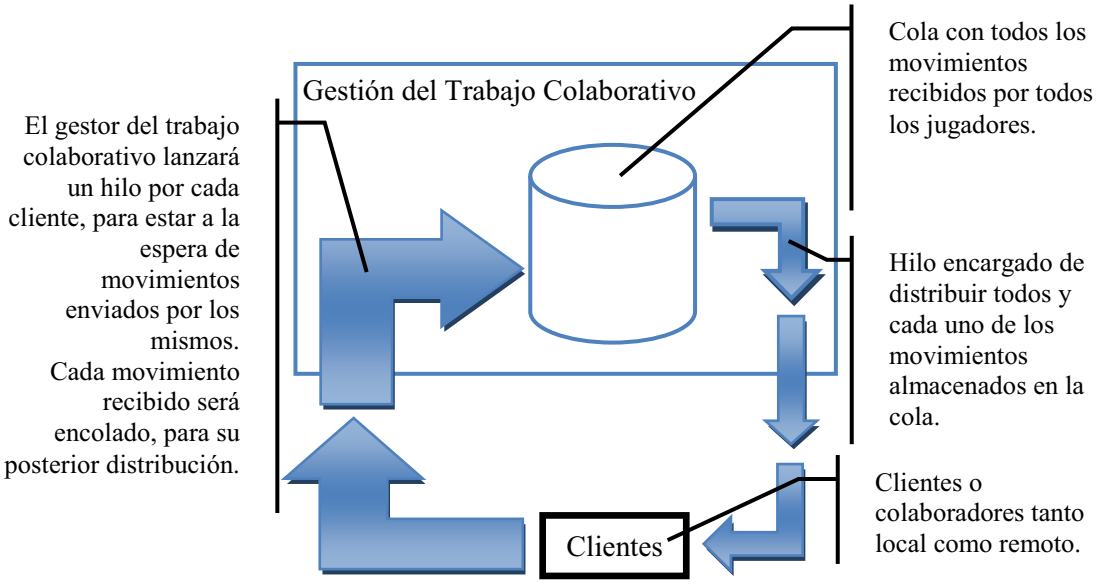

Figura 28: Hilos y componentes del trabajo colaborativo.

6.4.4 Diseño del protocolo de comunicación

Sobre los mensajes que permitirán la comunicación entre los dispositivos móviles, se considerará a la pantalla prototipo de la Figura 14.

Como se puede ver en la Figura 14 la pantalla costará de tres áreas principales. Una es el área principal, la segunda es la barra de piezas y la barra de estado.

Se considera que lo más relevante e importante que deberá transmitirse es:

- Un ID (identificador) del jugador o participante.
- El nombre del dispositivo o jugador que realizo el movimiento.
- El ID de la pieza que se intenta colocar.
- Las coordenadas de la pieza que se intenta colocar.

Se considerará que habrá dispositivos que no soporten el procesamiento de XML y dispositivos que si lo soporten.

Para los dispositivos que soporten XML los mensajes tendrán el siguiente formato:

```
<movimiento>
     <jugador id=”0” nombre=”ortigosa”/>
     <pieza id=”10”>
          <coordenadas i=”0” j=”0”/>
     </pieza>
</movimiento>
```

El script anterior representa un mensaje en formato XML que será útil para transmitir la información requerida por la aplicación.

Ahora para el caso de los dispositivos que no soporten el procesamiento de XML se propone utilizar el siguiente formato simple:

```
0,Ortigosa,10,0,0
```

El texto anterior contiene la misma información que la del mensaje en formato XML, donde el primer elemento separado por comas (,) se refiere al ID del jugador, el segundo elemento se refiere al nombre del usuario o jugador, el tercero es el ID de la pieza que será movida y los últimos dos elementos son las coordenadas.

Los mensajes anteriores serán los que permitan la comunicación entre las aplicaciones instaladas en los diferentes dispositivos. Estos mensajes serán los que enviarán y recibirán los dispositivos.

6.4.5 Flujo de datos en un ambiente colaborativo

En esta sección se muestra gráficamente el flujo de datos en un ambiente colaborativo, en donde intervienen cuatro jugadores. Ver Figura 29.

En la Figura 29 se puede visualizar claramente que todos los movimientos son enviados a el gestor del trabajo colaborativo y este los procesa y reparte los mensajes con los movimientos adecuados.

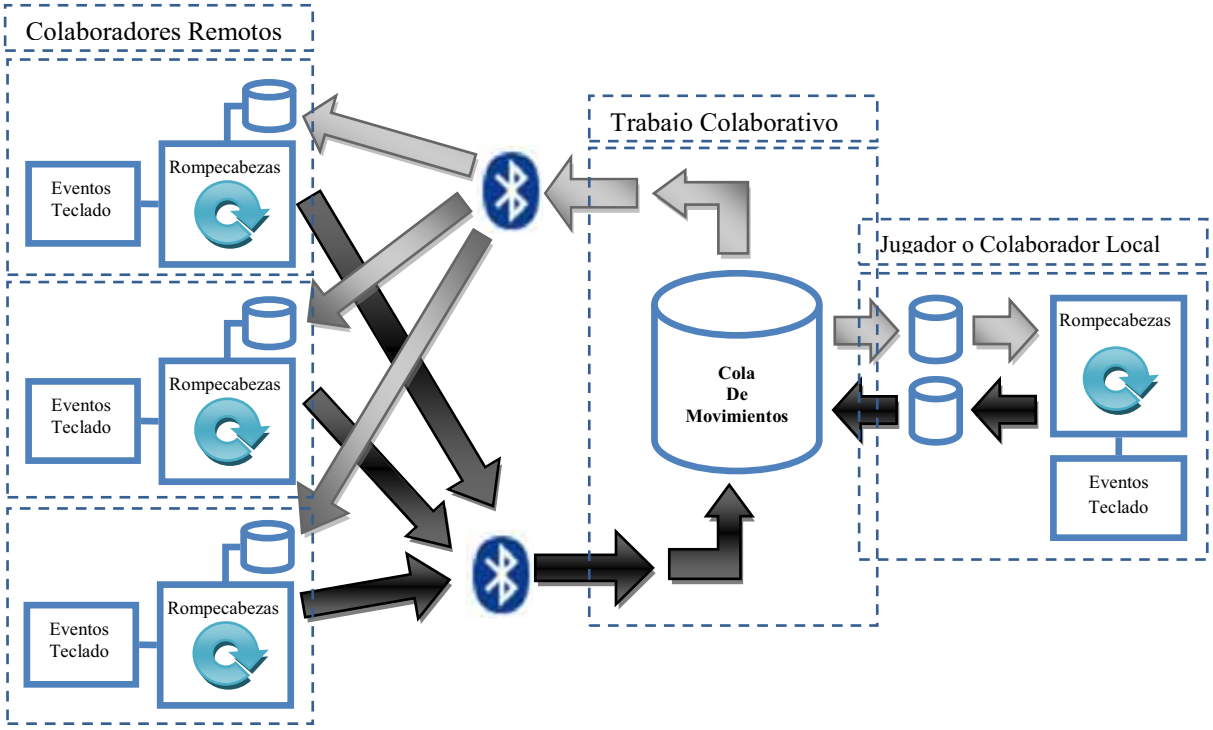

Figura 29: Flujo de datos del sistema en una ambiente colaborativo.

6.5 Diagramas de clases

El diagrama de clases evolucionó de un escueto esquema a uno dividido en varios paquetes. Los paquetes que conforman al sistema son los siguientes (ver Figura 30):

- uam.pt.ortigosa.bluetooth
- uam.pt.ortigosa.colaborativo
- uam.pt.ortigosa.graphics
- uam.pt.ortigosa.midlet
- uam.pt.ortigosa.util
- uam.pt.ortigosa.util.xml
- uam.pt.ortigosa.vistas

Figura 30: Paquetes de clases que conforman el sistema

En la Figura 31 se muestra el diagrama de clases del sistema con todas las clases más importantes.

En el apéndice “Código fuente” se presenta todo el código que interviene en cada clase de cada diagrama que se mostrara durante esta sección.

Algunos Diagramas no se visualizan correctamente debido a su tamaño, sin embargo, adjunto a este documento se proporcionan las imágenes en formato PDF para una mayor claridad.

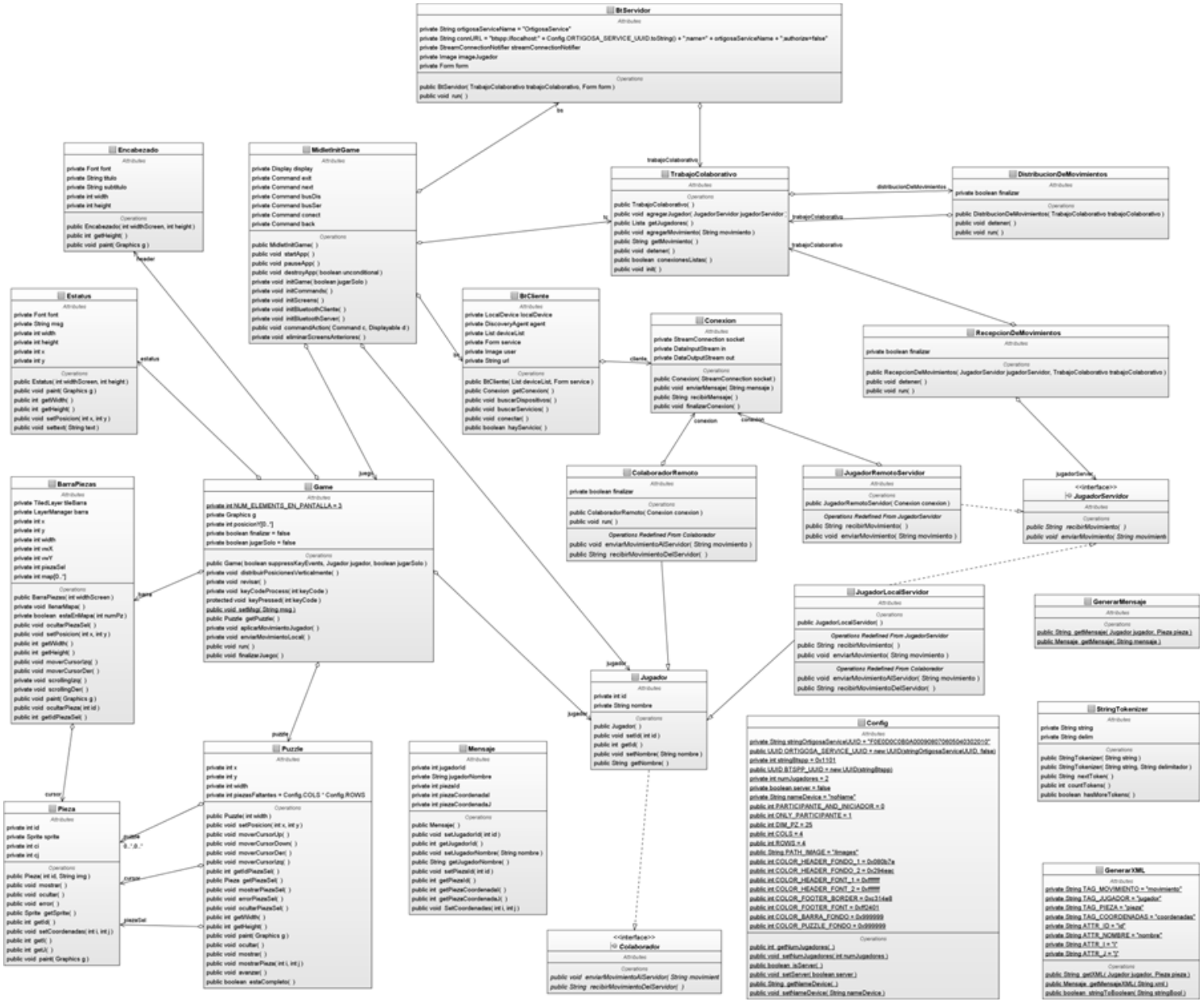

Figura 31: Diagrama de clases general del sistema.

6.5.1 Paquete de clases uam.pt.ortigosa.bluetooth

En este paquete se agrupan las clases necesarias para la creación de los objetos cliente y servidor para establecer comunicación vía bluetooth.

En la Figura 32 se muestra el diagrama de clases correspondiente.

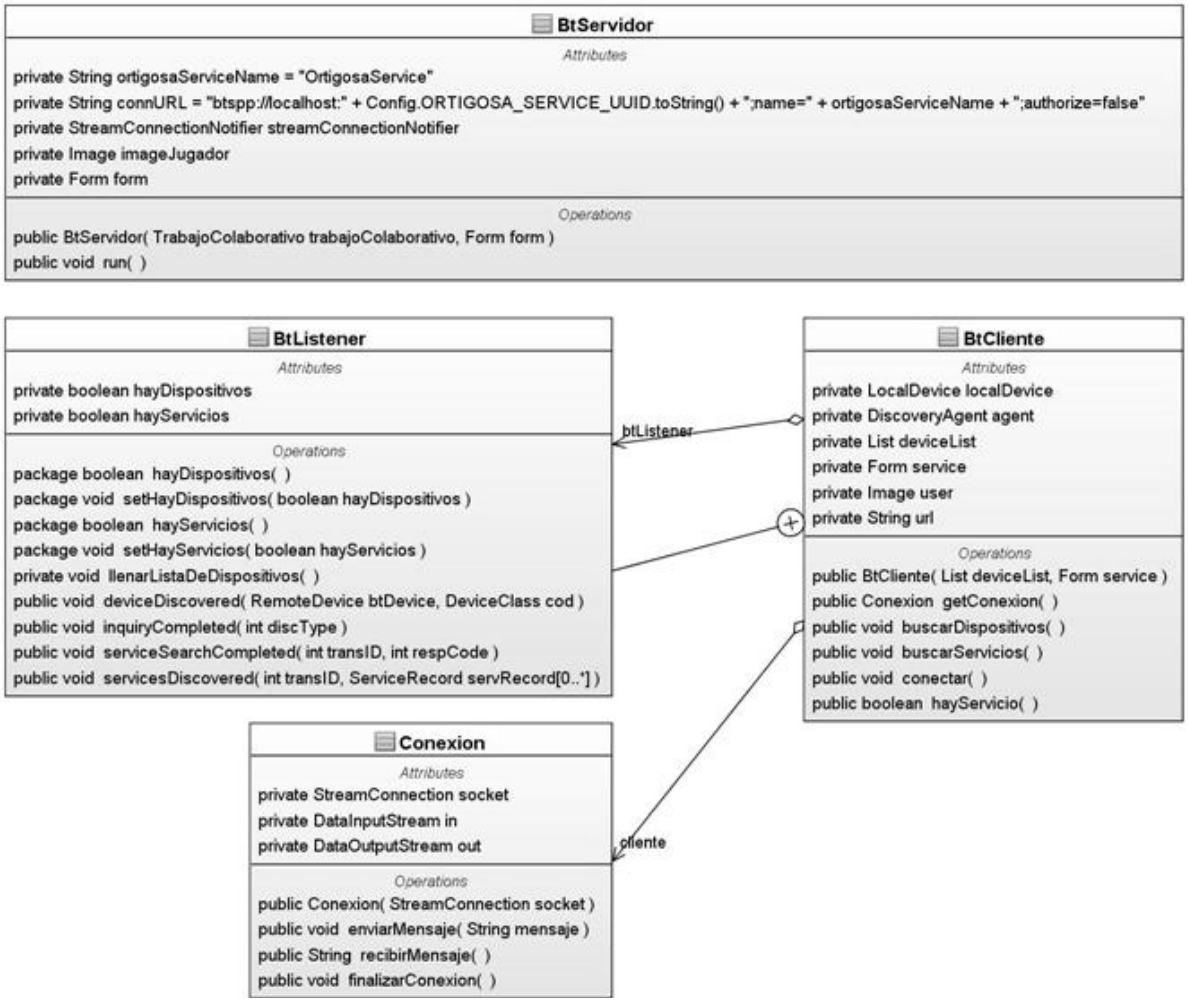

Figura 32: Diagrama de clases del paquete uam.pt.ortigosa.bluetooth.

6.5.2 Paquete de clases uam.pt.ortigosa.colaborativo

Paquete de clases que intervienen con el trabajo colaborativo y que permite describir a los siguientes componentes:

- Colaborador Remoto.
- Colaborador o Jugador Local.
- Gestión del Trabajo Colaborativo.

La descripción del funcionamiento de los componentes enlistados se describe en la sección “Diseño esquemático del sistema” del apartado “Diseño del sistema”.

En la Figura 33 se muestra el diagrama de clases de este paquete.

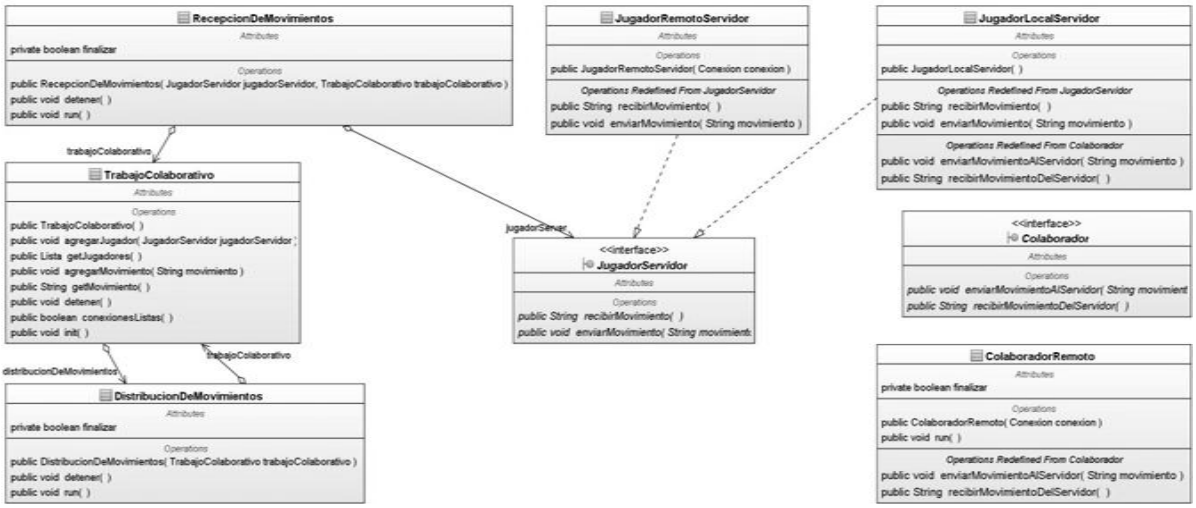

Figura 33: Diagrama de clases del paquete uam.pt.ortigosa.colaborativo.

6.5.3 Paquete de clases uam.pt.ortigosa.graphics

Este paquete contiene todas las clases necesarias que permiten la creación y manejo del juego de rompecabezas.

Esta paquete contiene las clases que permiten dibujar en pantalla los principales componentes del prototipo de la

Pantalla Juego de Rompecabezas descrita en el apartado

Prototipos".

En la Figura 34 se muestra un diagrama de clases del paquete.

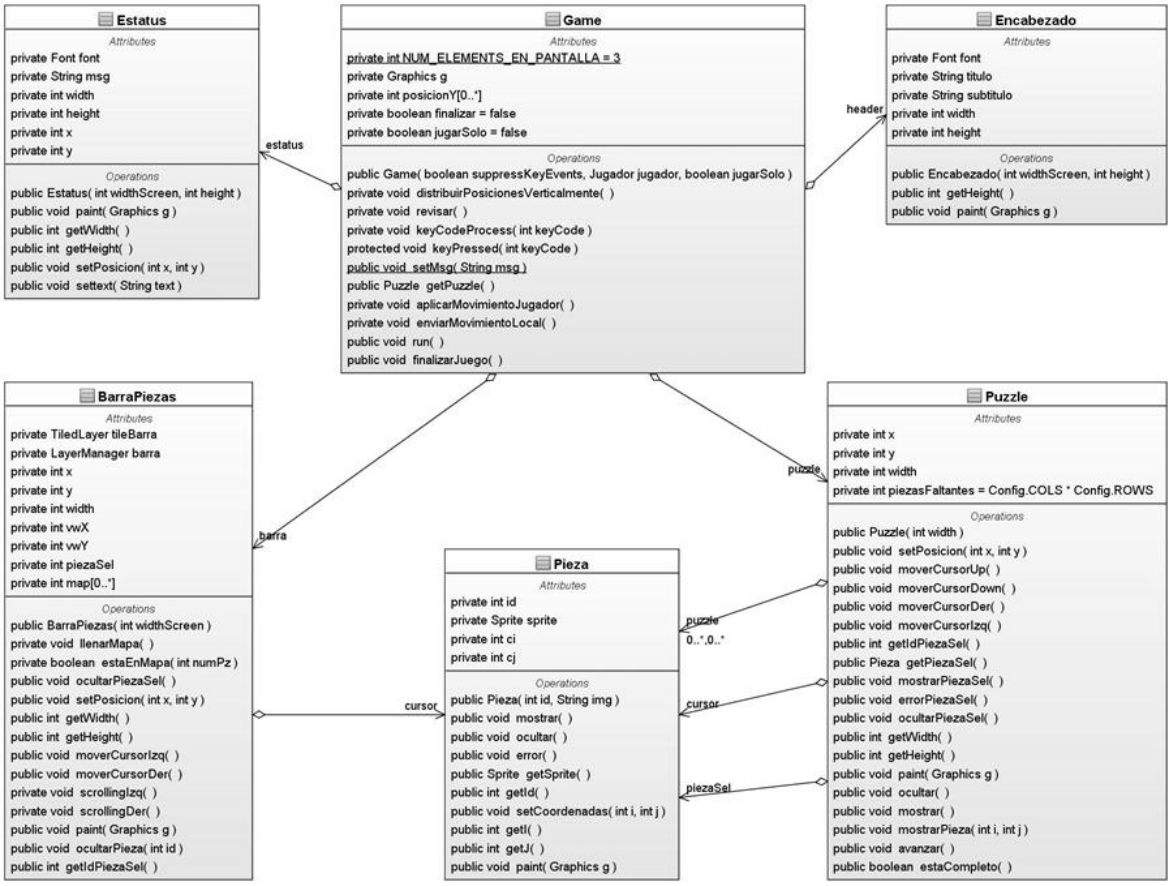

Figura 34: Diagrama de clases del paquete uam.pt.ortigosa.graphics.

6.5.4 Paquete de clases uam.pt.ortigosa.midlet

Este paquete contiene únicamente una clase llamada MidletInitGame que es la clase que se encargará de inicializar todo lo necesario para visualizar, gestionar y planificar la aplicación o sistema.

En la Figura 35 se muestra el diagrama de la única clase que compone al paquete.

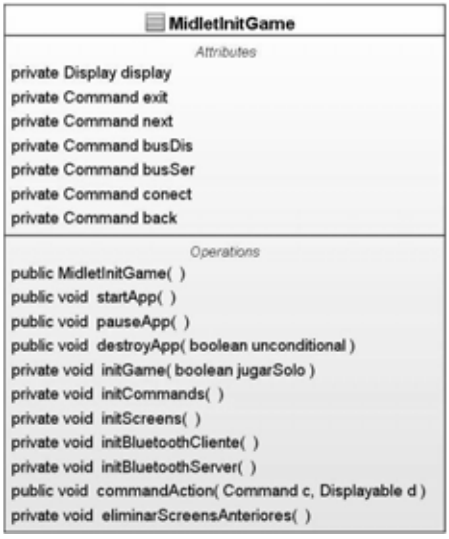

Figura 35: Diagrama de la única clase del paquete uam.pt.ortigosa.midlet.

6.5.5 Paquete de clases uam.pt.ortigosa.util

Este paquete contiene varias clases que son útiles en la lógica de los otros paquetes. Entre las clases que se encuentran en el paquete esta la clase Config que es la encargada de tener varios parámetros globales del juego; entre los que se encuentran: color, tamaño, estado, entre otros.

Otras de las clases importantes que tiene este paquete son la implementación de una cola. Esta clase permitirá instanciar objetos de tipo cola que tienen la particularidad de ser objetos mutuamente excluyentes, para el caso en el que se requiera acceso a un objeto de este tipo de forma simultánea por varios hilos.

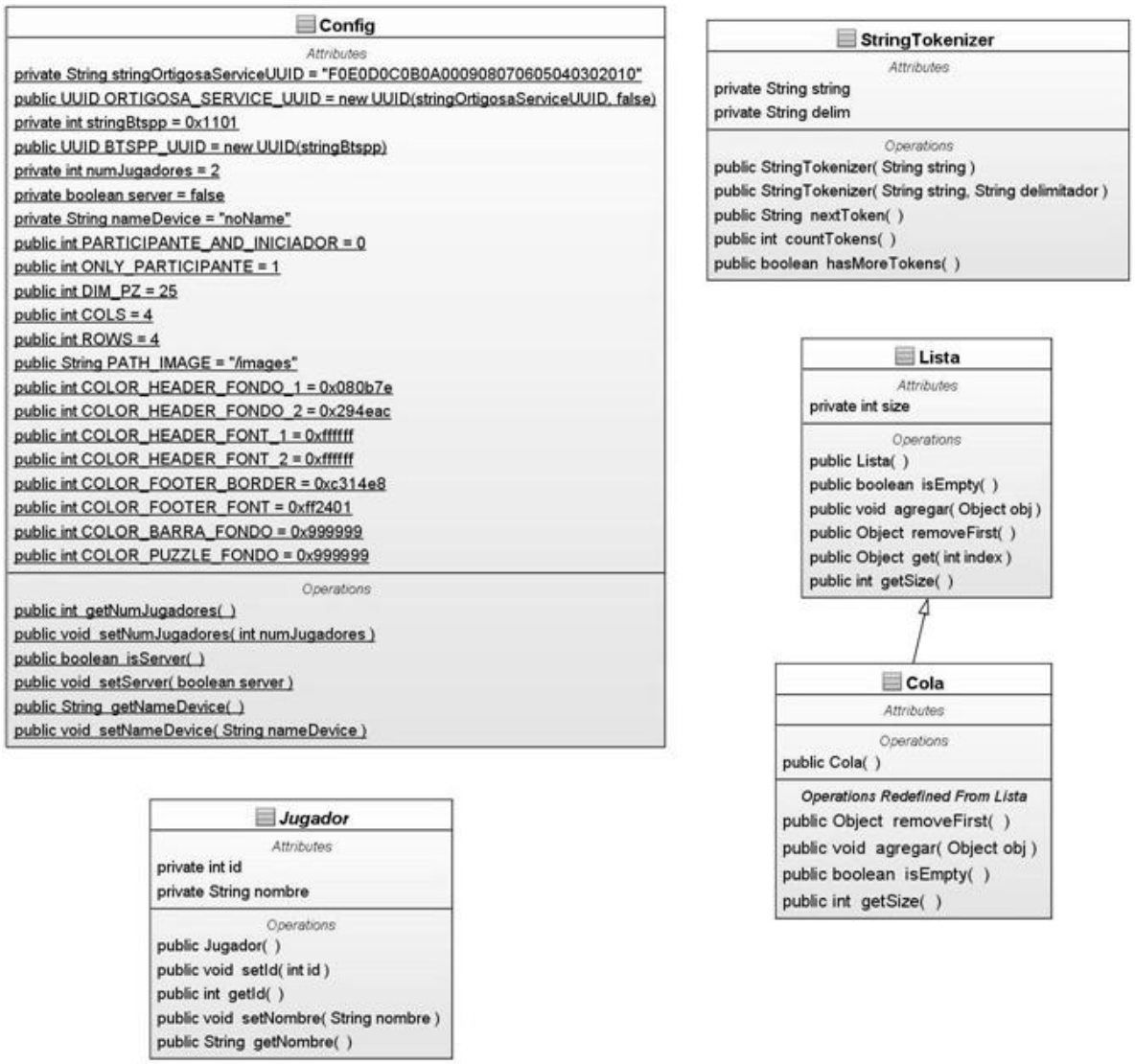

Figura 36: Diagrama de clases del paquete uam.pt.ortigosa.util.

6.5.6 Paquete de clases uam.pt.ortigosa.util.xml

Este paquete contiene las clases necesarias para el formateo y extracción de los datos contenidos en los mensajes que viajan de un dispositivo a otro.

Tal como se mencionó en el apartado llamado “Descripción de la aplicación”, los mensajes que viajarán entre los dispositivos estarán en un formato XML, con el fin de extender el protocolo de la aplicación y además de reforzar la escalabilidad, ya que al contar con los mensajes en XML, cualquier otra aplicación que se desee comunicar con el juego solo necesitará conocer el protocolo y podrá extraer la información de los mensajes en XML.

Sin embargo; a la fecha existen muchos dispositivos que carecen de la habilidad para manejar XML, es decir, no cuentan con las bibliotecas necesarias para realizar esta operación. Por tal motivo se implemento otro protocolo más simple para la comunicación entre dispositivos más austeros.

En la Figura 37 se muestra el diagrama de clases del paquete, en el se puede observar la clase GenerarXML y Generar Mensaje ambas clases formatean y extraen información contenida en los mensajes encapsulados en un protocolo.

La clase GenerarXML, permite generar mensajes en un formato de XML además de esta función también cuenta con el proceso inverso, para extraer la información y convertirla en un objeto simple llamado Mensaje.

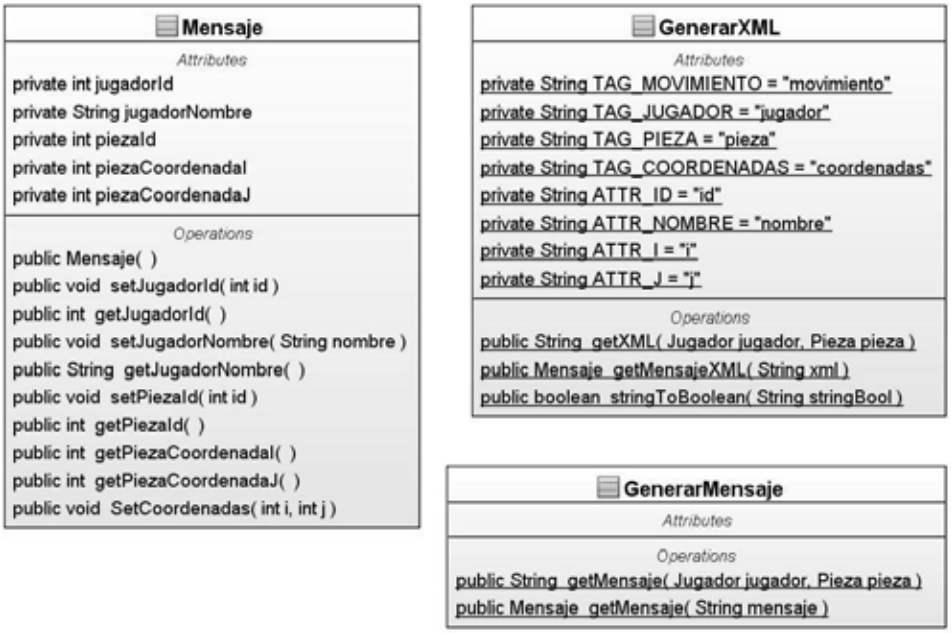

Figura 37: Diagrama de clases del paquete uam.pt.ortigosa.util.xml.

6.5.7 Paquete de clases uam.pt.ortigosa.vistas

Este paquete contiene las clases que permiten la interacción directa con el usuario. Contiene las vistas o pantallas con la funcionalidad descrita en la sección “

Prototipos”.

En la Figura 38 se muestran las clases que conforman al paquete uam.pt.ortigosa.vistas.

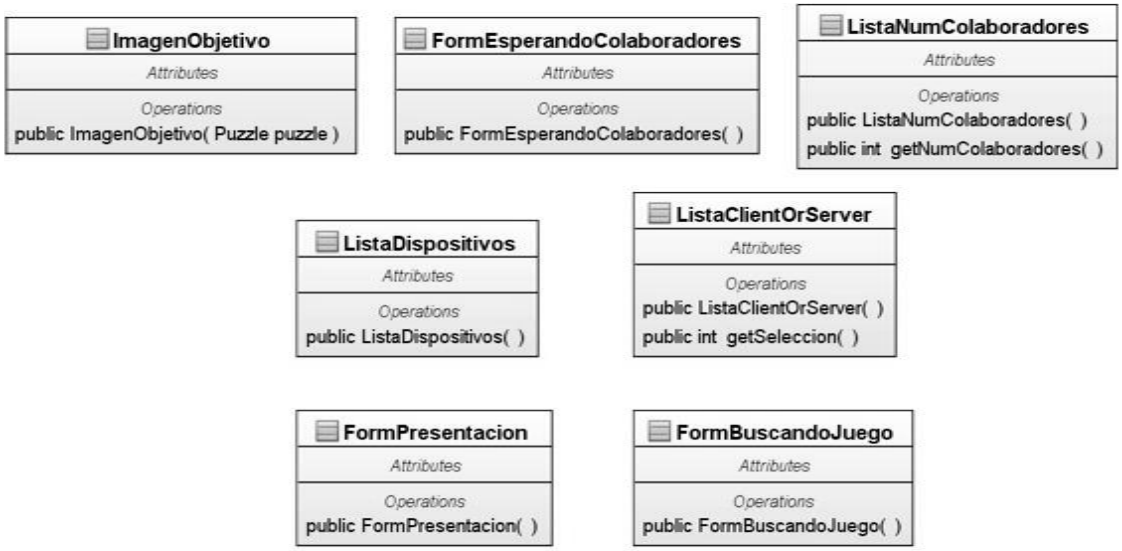

Figura 38: Diagrama de clases del paquete uam.pt.ortigosa.vistas.

7 PRODUCTO FINAL

Durante todo el proceso de creación del sistema se realizaron pruebas a cada uno de los componentes y objetos que conformarían al sistema o aplicación final tal y como se había previsto en la metodología propuesta en el apartado "Metodología para el desarrollo del sistema", en este apartado solo se presenta el producto final obtenido después de haber seguido la metodología y de haber codificado todo lo necesario.

Las imágenes que se muestran en este apartado pertenecen a la simulación de teléfonos celulares proporcionada por el entorno de desarrollo Java ME platform SDK 3.0.

Aún cuando las pantallas cambiarán dependiendo del dispositivo en el que se instale la aplicación, la variación será mínima o solo se presentará en la apariencia, debido quizá por la enorme variedad de temas para cambiar la apariencia de las pantallas de los dispositivos móviles, sin embargo la funcionalidad será la misma.

7.1 Pantallas finales

En este apartado se presenta y se hace la asociación entre el prototipo de pantalla y la pantalla obtenida después de la codificación de la aplicación.

7.1.1 Pantalla Presentación

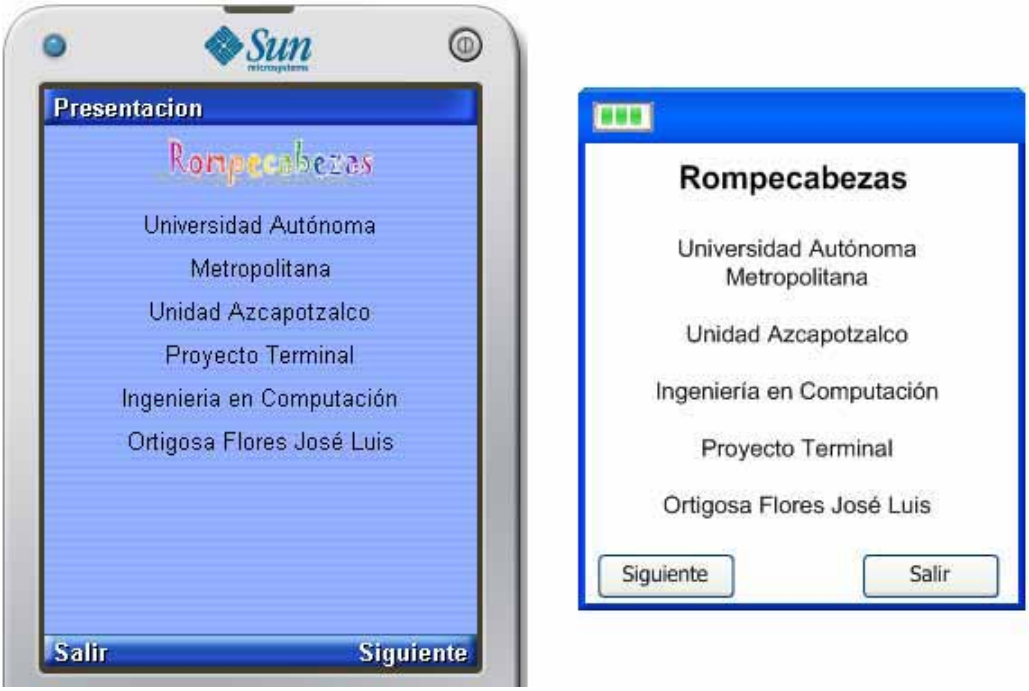

Figura 39: Pantalla Final: Presentación.

7.1.2 Pantalla Modo de Juego

Figura 40: Pantalla Final: Modo de Juego.

7.1.3 Pantalla Cantidad de Colaboradores

Figura 41: Pantalla Final: Cantidad de Colaboradores.

7.1.4 Pantalla Espera a los Colaboradores

Figura 42: Pantalla Final: Espera a colaboradores.

7.1.5 Pantalla Buscar Dispositivos y Juego

Figura 43: Pantalla Final: Buscar Dispositivo y Juego.

7.1.6 Pantalla Conexión con Dispositivo

Figura 44: Pantalla Final: Conexión con Dispositivo.

7.1.7 Pantalla imagen objetivo

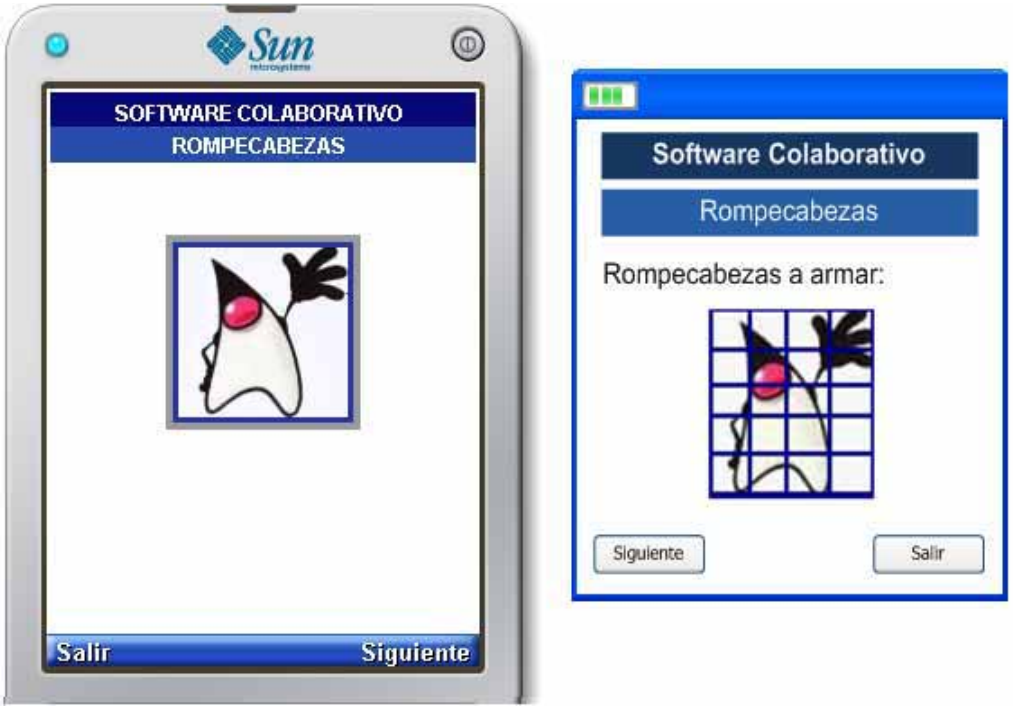

Figura 45: Pantalla Final: Imagen Objetivo.

7.1.8 Pantalla juego de rompecabezas

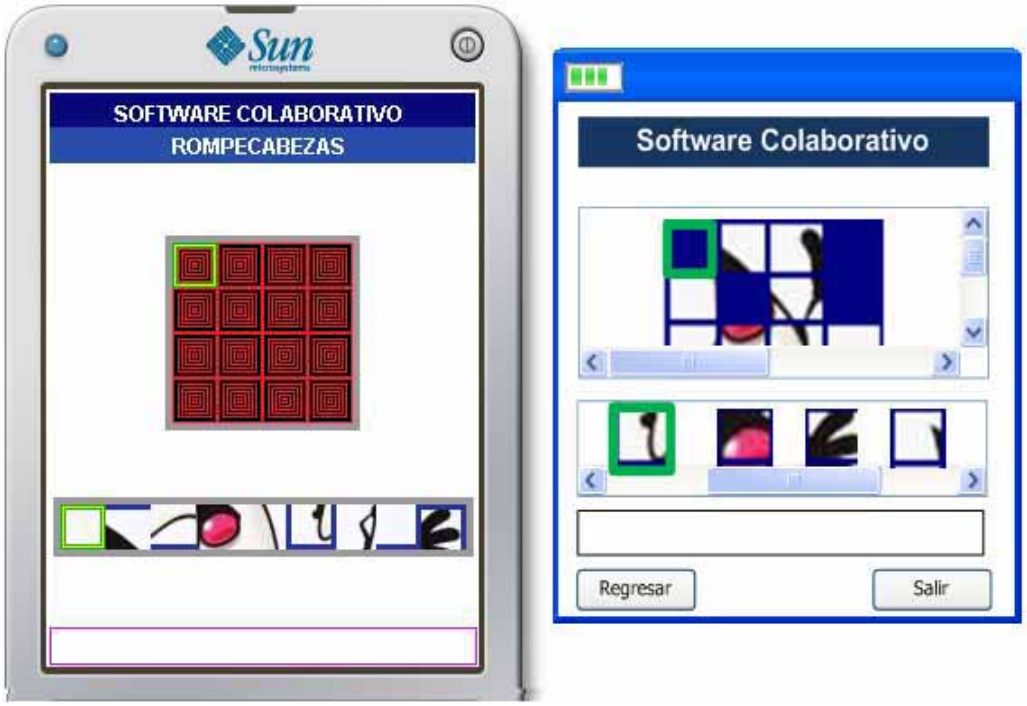

Figura 46: Pantalla Final: Juego de Rompecabezas.

7.2 Especificaciones finales

La aplicación esta diseñada para ser ejecutada en dispositivos móviles que cumplan con las características que se muestra en la

Característica	Especificación mínima
Pantalla	La aplicación fue diseñada para que los gráficos tuvieran cierta adaptabilidad a las dimensiones de la pantalla de cada dispositivo, es decir, la distribución de los gráficos estará en función de las dimensiones de la pantalla del dispositivo. Sin embargo, dada la tecnología usada, la aplicación no puede adaptarse al 100% de dimensiones de pantalla. La aplicación tiene un límite en cuanto a las dimensiones de la pantalla. Las dimensiones que se recomiendan para la pantalla son 176x220 (pixel). Los gráficos de la aplicación se adaptaran a pantallas con dimensiones superiores, pero con dimensiones inferiores la aplicación no se adaptará de una manera aceptable.
Bluetooth	Puede ser cualquier tipo de versión de bluetooth, pero se obtiene mejor rendimiento en la versión 2.0 o superior.
Poder de procesamiento	Se recomienda que el procesador corra a 600 Mhz con 512 MB de memoria RAM, sin embargo, fue probado en celulares con un nivel de procesamiento, muy por debajo del recomendado, tal es el caso de los celulares Sony W800i y S500i.
Juego de Rompecabezas	El sistema (el software) esta diseñado para soportar tres jugadores de forma simultanea, siempre que el hardware así lo permita, en pruebas con un celular Sony Ericsson S500i con un nivel de poder de procesamiento inferior, al recomendado, la aplicación es muy lenta. En celulares con la capacidad de procesamiento señalada el rendimiento es óptimo. En la aplicación solo se puede armar una figura, para fines prácticos.
Tipo de aplicación	La aplicación esta basada en la tecnología *Java ME*. Es una aplicación tipo CLDC/MIDP la aplicación esta orientada a la versión 1.1 de CLDC y a las versiones 2.0 y 2.1 de MIDP. El dispositivo en el que se ejecutará la aplicación deberá contar con el soporte adecuado para esta tecnología.

Tabla 12: Características requeridas y proporcionadas por el sistema.

8 PRUEBAS

Al finalizar la codificación; la aplicación fue probada en un celular Sony Ericsson w800i. La aplicación se ejecuto en el modo de juego: “jugar solo”, el resultado se muestra en la foto de la Figura 47. En dicha foto se muestra el juego de rompecabezas.

Figura 47: Aplicación instalada y en ejecución en un celular Sony Ericsson W800i.

La siguiente prueba que se realizó fue entre dos celulares uno Sony Ericsson modelo W800i y un celular LG modelo GM360i.

La prueba consistió en que el celular LG iniciara un nuevo juego colaborativo, es decir, que iniciara en modo servidor y que esperará a solo un colaborador y finalmente por otro lado el celular Sony ejecutará la aplicación en un modo participante, es decir, en el modo cliente para que el sistema le permitiera conectarse con el celular LG.

En la Figura 48 se muestra a los dos celulares al Sony y LG ejecutando la aplicación, En dicha foto se muestra a los dos celulares ejecutando la aplicación, donde se puede

percibir que los componentes gráficos se adaptan y distribuyen en la pantalla dependiendo de las dimensiones de cada dispositivo.

Otro punto importante que se observa es que al momento de tomar la foto el celular LG había realizado un movimiento tal como se ve en la barra de estado de la aplicación donde se muestra el mensaje: “LG GM360i coloca pieza 8”; mensaje que es mostrado tanto en el celular LG como en el Sony.

El rendimiento de la aplicación en ambos celulares fue fluido durante toda la aplicación, no hubo momento en los que se considerara que se trabo la aplicación o que la aplicación dejo de funcionar.

En la imagen el celular LG esta en modo servidor y el Sony en modo cliente.

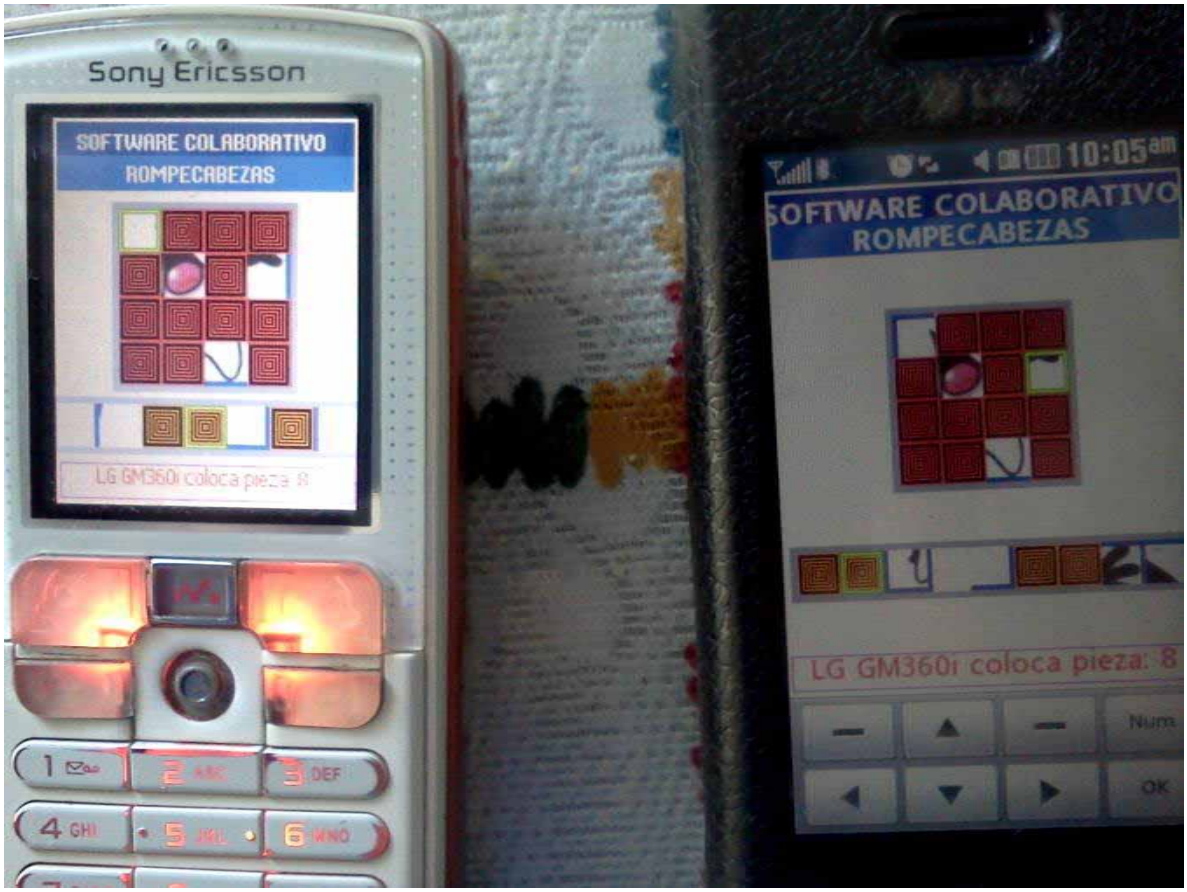

Figura 48: Sony y LG ejecutando la aplicación y jugando colaborativamente.

También la aplicación fue probada entre un celular Sony Ericsson w800i y un celular HP modelo iPaq GListen. En la Figura 49 se muestra a los dos celulares ejecutando la aplicación y trabajando de forma colaborativa. En la imagen también se puede apreciar que aparece en la barra de estado el mensaje: “Ortigosa coloca pieza 4” indicando que el último

movimiento que se realizo fue hecho por el celular con nombre “Ortigosa” que pertenece al celular Sony Ericsson w800i.

Otro aspecto importante de destacar es que la parte gráfica de la aplicación se logro adaptar satisfactoriamente.

El rendimiento de la aplicación en los dos celulares fue excelente durante la realización de esta prueba.

El celular HP esta ejecutando la aplicación en modo servidor y en el Sony esta en modo cliente.

Durante la prueba no hubo errores o fallas de la aplicación.

El celular HP tiene un sistema operativo Windows Mobile 6.5 y el Sony tiene un Symbian.

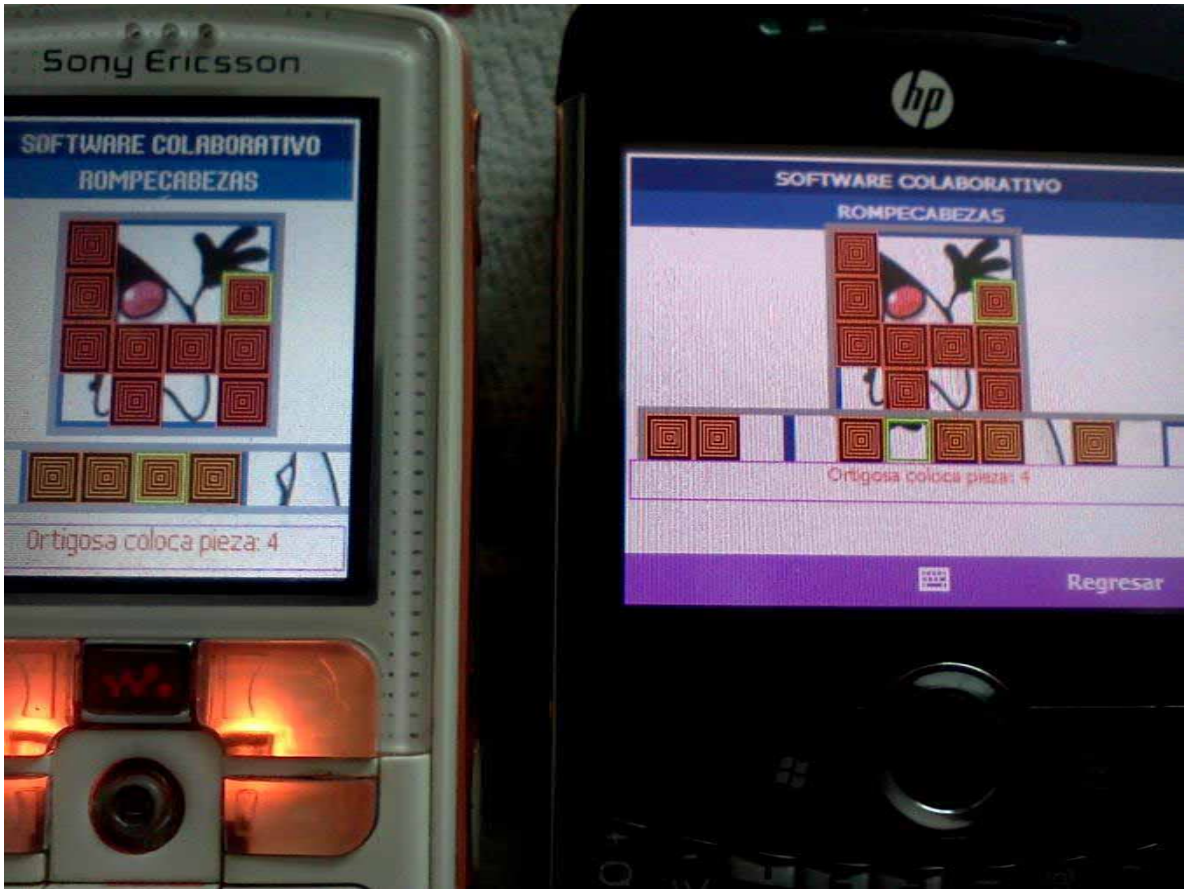

Figura 49: Sony y HP ejecutando la aplicación y jugando colaborativamente.

También se realizaron pruebas entre un celular Sony Ericsson modelo S500i y uno de la misma marca pero de modelo W800i.

En la Figura 50 se muestran a los dos celulares armando el rompecabezas de forma colaborativa.

Aquí el celular modelo S500i esta en modo “Juego Colaborativo [Como Servidor]” y el celular modelo W800i esta en modo “Participante [Como Cliente]”.

El rendimiento de la aplicación entre estos dos celulares es muy pobre. Las características de poder de procesamiento del celular modelo S500i son escasas, motivo por el cual los movimientos tardan en reflejarse en ambos celulares, sin embargo, la aplicación funciona, los gráficos se adecuaron perfectamente a las dimensiones de la pantalla.

Figura 50: W800i y S500i ejecutando la aplicación y jugando colaborativamente.

La siguiente prueba que se realizó fue hacer una simulación con ayuda del Java ME Platform 3.0 y sus emuladores de celulares.

La prueba consistió en ejecutar tres emuladores de celulares donde cada uno ejecutará la aplicación. En la Figura 51, se muestra la ejecución de tres emuladores en donde cada uno ejecuta la aplicación.

Figura 51: Ejecución de la aplicación en tres emuladores de celular #1.

En la Figura 52 se muestra a los tres emuladores de celular, en donde el celular que esta en la extrema izquierda de la imagen, esta ejecutando la aplicación en modo servidor, en donde además se puede apreciar que ya se han conectado dos celulares, que justo son los dos celulares restantes de la misma figura; celulares que muestran la pantalla “Imagen Objetivo”, señal de que están listos para comenzar a colaborar.

Figura 52: Ejecución de la aplicación en tres emuladores de celular #2.

En la Figura 53 se muestran a los mismos tres emuladores, trabajando de forma colaborativa.

La imagen de la Figura 53 es muestra de que la prueba resulto satisfactoriamente, dado que los tres celulares están trabajando de forma colaborativa, lo que significa que, en el momento en el que en un emulador de celular se realice un movimiento, este movimiento se vera reflejado de forma simultanea (solo limitado por la velocidad de transferencia de la conexión bluetooth, que por cierto también se encarga de emular el SDK) en los otros emuladores.

Figura 53: Ejecución de la aplicación en tres emuladores de celular #3.

9 INSTALACIÓN E INSTRUCCIONES DE LA APLICACIÓN

Todos los detalles de instalación y de instrucciones de juego están expresados en el "Manual de Instalación e Instrucciones de la aplicación", Documento que esta adjunto en el CD.

Es importante mencionar que el juego cuenta con una versión simplificada de las instrucciones para operar la pantalla principal.

Para acceder a las instrucciones instaladas dentro de la aplicación es necesario iniciar la aplicación, ubicarse en la pantalla de "Presentación" y seleccionar la opción 'Instrucciones'. En la Figura 54 se muestra la pantalla en la que se pueden encontrar las instrucciones correspondientes a como usar la aplicación.

***Nota**: La posición de las opciones, varía de dispositivo a dispositivo.*

Figura 54: Instrucciones #1

Después de seleccionar la opción 'Instrucciones' dentro de la pantalla "Presentación", el sistema mostrará la pantalla de la Figura 55.

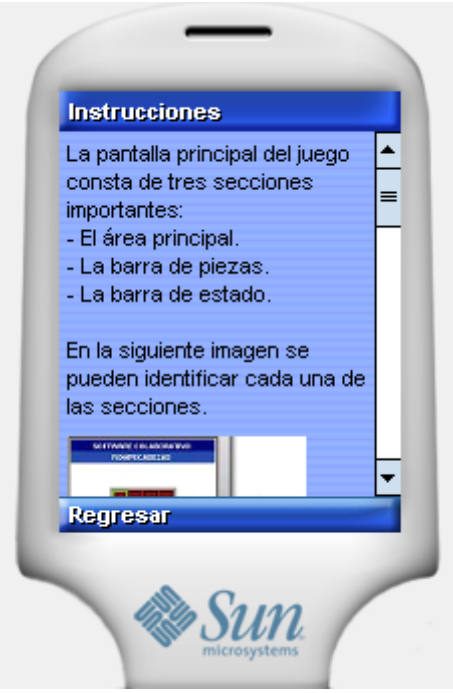

Figura 55: Instrucciones #2.

En la Figura 55 se mostrara una breve descripción de los componentes principales de la pantalla “Juego de Rompecabezas”, así como las teclas que estarán activas durante la aplicación.

La pantalla tiene la opción ‘Regresar’ que es la manera en la que el usuario podrá retornar a la pantalla “Presentación”.

10 TRABAJO A FUTURO

El principal objetivo del proyecto terminal es crear un software colaborativo para dispositivos móviles con comunicación bluetooth, tal como se menciona en el titulo del presente documento; objetivo que se cumple tal como se ve en el apartado "Pruebas y producto final", y que además como se menciono en el apartado "Elección de la tecnología", *Java* y su edición micro (*Java ME*) es una tecnología que va de salida pero que a demostrado su enorme potencial al permitir el desarrollo de aplicación en dispositivos con muy pocos recursos hardware/software y que esta siendo sustituida por otras tecnologías tales como *JavaFX*, perteneciente a la misma compañía de *Java ME*, así como otras pertenecientes a otras empresas tales como la *API* java desarrollada por la empresa *Google* para sus sistema operativo *Android*, que ahora están siendo implantadas en los nuevos dispositivos llamados *Smartphone* y que además están ganando terreno rápidamente tal como se menciona en el apartado "Elección de la tecnología".

Estas tecnologías mencionadas son nuevas pero no necesariamente son mejores y libres de errores, son tecnologías que están en pleno desarrollo y que tienen un enorme potencial pero que no están totalmente probadas y que además aún no tienen el alcance y estabilidad que proporciona la tecnología *Java ME*, esto último a mi parecer opinión y basado en los comentarios del libro de *Linthicum*[22] en el que se menciona que no siempre es bueno usar lo que el llama la *cool-technology*[23], y que es más preferible usar las aplicaciones más estables y más probadas que enfrentarse con una tecnología nueva y poco probada, como es el caso de la nueva *API* de java para *Android*, sin embargo, estas nuevas tecnologías están demostrando un rápido crecimiento y por tanto no deben de ser menospreciadas ya que en pocos años quizá sean la tecnología que predomine para los dispositivos móviles del futuro cercano. Por tal motivo se propone que a futuro el producto de este proyecto terminal sea migrado a una de las nuevas tecnologías tales como la propuesta por *Oracle* y *JavaFX* así como la *API* desarrollada por *Google* para *Android* y usando el mismo medio de comunicación *bluetooth*, que por cierto también esta evolucionando para mejorar la velocidad o quizás usar otros medios, que ya están disponibles en los nuevos dispositivos móviles tales como *Wi-Fi*, que permitirían una comunicación más rápida.

El núcleo central del producto de este proyecto terminal es el conjunto de clases que permiten el trabajo colaborativo; conjunto de clases que dada la arquitectura de la aplicación están diseñados con un bajo acoplamiento, por lo que no dependen de las clases propias proporcionadas por la tecnología *Java ME*, por tal motivo esta capa principal no será reescrita si se seleccionan las tecnologías de *JavaFX* y la *API* de java de *Google*, sin

[22] (Linthicum, 2000), en el capitulo 1.
[23] Se puede traducir como: la tecnología del momento, la última tecnología o la tecnología de moda.

embargo lo que si es probable y susceptibles de cambios son las capas que están muy acopladas con la tecnología *Java ME* tal es el caso de la capa de comunicación y la que permite el dibujado de los gráficos.

Aún cuando las capas antes mencionadas tendrán que ser modificadas y adecuadas a la tecnología destino, el diseño que se propuso en el apartado "Diseño del sistema" así como la Arquitectura propuesta serán de mucha utilidad para su migración.

Resumiendo, a futuro se planea migrar el producto de este proyecto terminal a las nuevas tecnologías desarrolladas para los dispositivos inteligentes que poseen un mayor poder de procesamiento, para tener un mejor desempeño de la aplicación y aprovechar que estas nuevas tecnologías permiten y soportan la multitarea[24].

[24] Multitarea o multiproceso se refiere a varios tareas o procesos que comparte un recurso de procesamiento tal como un CPU. Cuando en una computadora solo se cuenta con un solo CPU, solo una tarea o proceso puede usar el procesador a la vez. La multitarea se encarga de resolver el problema de planificar las tareas o proceso, es decir, decide que programa se ejecuta en que momento.

BIBLIOGRAFÍA

BlackBerry. (2011). *BlackBerry*. Recuperado el 15 de junio de 2011, de http://us.blackberry.com/

Google. (2007). *Android*. Recuperado el 15 de junio de 2011, de http://www.android.com/

iPhone OS X. (2011). *iPhone OS X, Aplicaciones, Cydia, Jailbreak, Noticias*. Recuperado el 15 de junio de 2011, de http://iphoneosx.com/

JSR 118. (2000). Recuperado el 4 de abril de 2011, de MIDP 2.0: http://www.jcp.org/en/jsr/detail?id=118

JSR 218. (2000). Recuperado el 31 de enero de 2011, de CDC 1.1.2: http://www.jcp.org/en/jsr/detail?id=218

JSR 37. (2000). Recuperado el 4 de abril de 2011, de MIDP 1.0: http://www.jcp.org/6.en/jsr/detail?id=37

Linthicum, D. S. (2000). *Enterprise Application Integration.* USA: Addison Wesley.

Microsoft. (2011). *Windows Phone*. Recuperado el 15 de junio de 2011, de http://www.microsoft.com/windowsphone/es-Es/default.aspx

Motorola Mobility. (2011). *Motorola Mobility*. Recuperado el 15 de junio de 2011, de http://www.motorola.com/Consumers/MX-ES/Home

Nokia. (15 de junio de 2011). *Nokia, Connecting people*. Recuperado el 15 de junio de 2011, de http://www.nokia.com/

Oracle. (diciembre de 2010). *Java FX*. Recuperado el 10 de junio de 2011, de http://www.javafx.com

Oracle. (s.f.). *Oracle*. Recuperado el 9 de junio de 2011, de Java ME Technology - CDC: http://www.oracle.com/technetwork/java/javame/tech/index-jsp-139293.html

Oracle. (2011). *Oracle*. Recuperado el 15 de junio de 2011, de Java ME and Java Card Technology: http://www.oracle.com/technetwork/java/javame/index.html

Palm, Hewlett-Packard Company. (2011). *HP Palm*. Recuperado el 15 de junio de 2011, de http://www.hpwebos.com/es/es/

Samsung. (2010). *Samsung*. Recuperado el 15 de junio de 2011, de http://www.samsung.com/mx/

Sony Ericsson Mobile Communications AB. (2011). *Sony Ericsson, Make belleve*. Recuperado el 15 de junio de 2011, de http://www.sonyericsson.com

SymbianOS.org. (2011). *Symbian OS*. Recuperado el 15 de junio de 2011, de http://www.symbianos.org/intro

A APLICACIÓN HOLA MUNDO

Como ya se mencionó en el apartado “Elección de la tecnología” para la realización de este proyecto se selecciono la **tecnología Java ME** y el entorno de desarrollo **Java ME platform SDK 3.0** (ver Figura 56), para la realización de una aplicación tipo CLDC/MIDP.

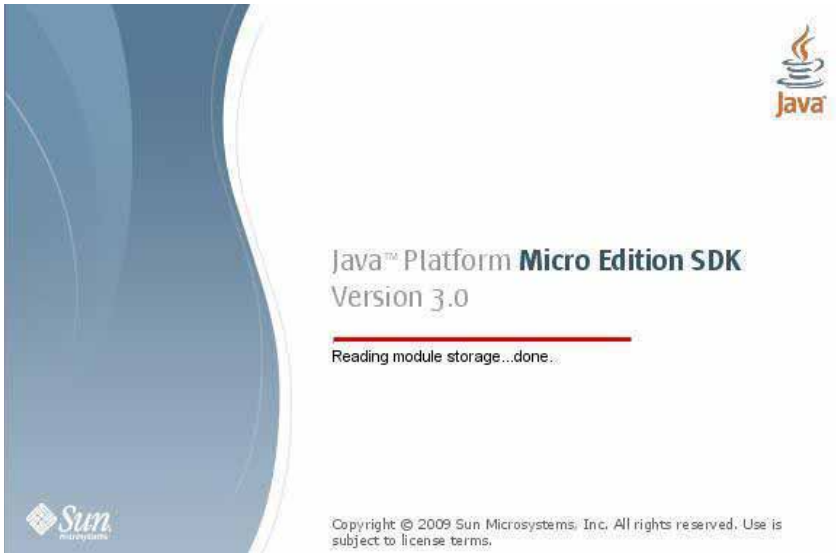

Figura 56: Carga del IDE java ME platform SDK 3.0.

Para crear un nuevo proyecto en la plataforma de desarrollo, hay que ir al menú “File” y seleccionar la opción “New Proyect…” (Ver Figura 57)

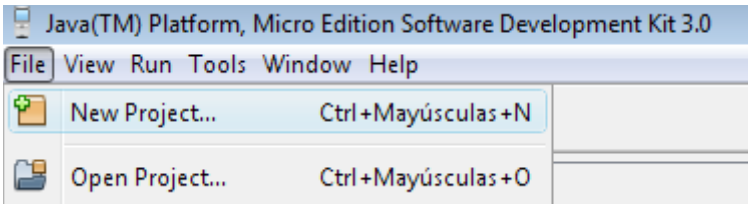

Figura 57: Crear un proyecto en la plataforma de desarrollo #1.

Al seleccionar la opción “New Proyect…” aparecerá el cuadro de diálogo de la Figura 58, en el cuadro de diálogo hay dos secciones una denominada “Categories” y otra denominada “Proyects”; dado que esta plataforma de desarrollo esta basada en NetBeans ambas plataformas cuentan con un cuadro de diálogo similar sin embargo en este caso como se ve en la Figura 58, solo existe una opción en la sección “Categories” dado que la plataforma Java ME SDK esta orientada solo al desarrollo de aplicaciones para dispositivos móviles.

Dado que se va ha hacer un aplicación de tipo CLDC/MIDP se selecciona la opción “MIDP Application” y se selecciona la opción “Mext >”.

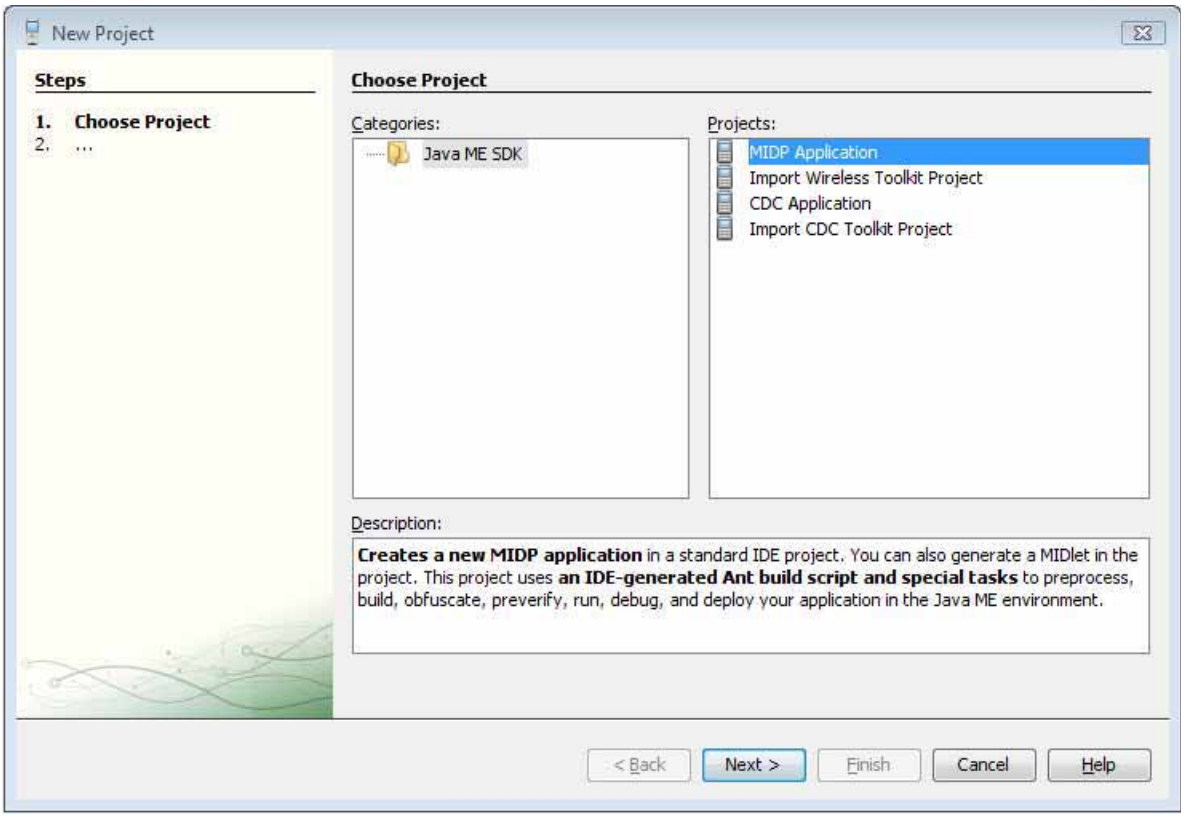

Figura 58: Crear un proyecto en la plataforma de desarrollo #2.

Después de haber seleccionado la opción “Next >” aparecerá el cuadro de diálogo que se muestra en la Figura 59, en el que el entorno de desarrollo solicita que se le proporcione un nombre de proyecto. Además del nombre el sistema pregunta si el proyecto nuevo será el principal (opción “Set as Main Project”) y si se desea crear un MIDlet Hello (opción “Create Hello MIDlet”).

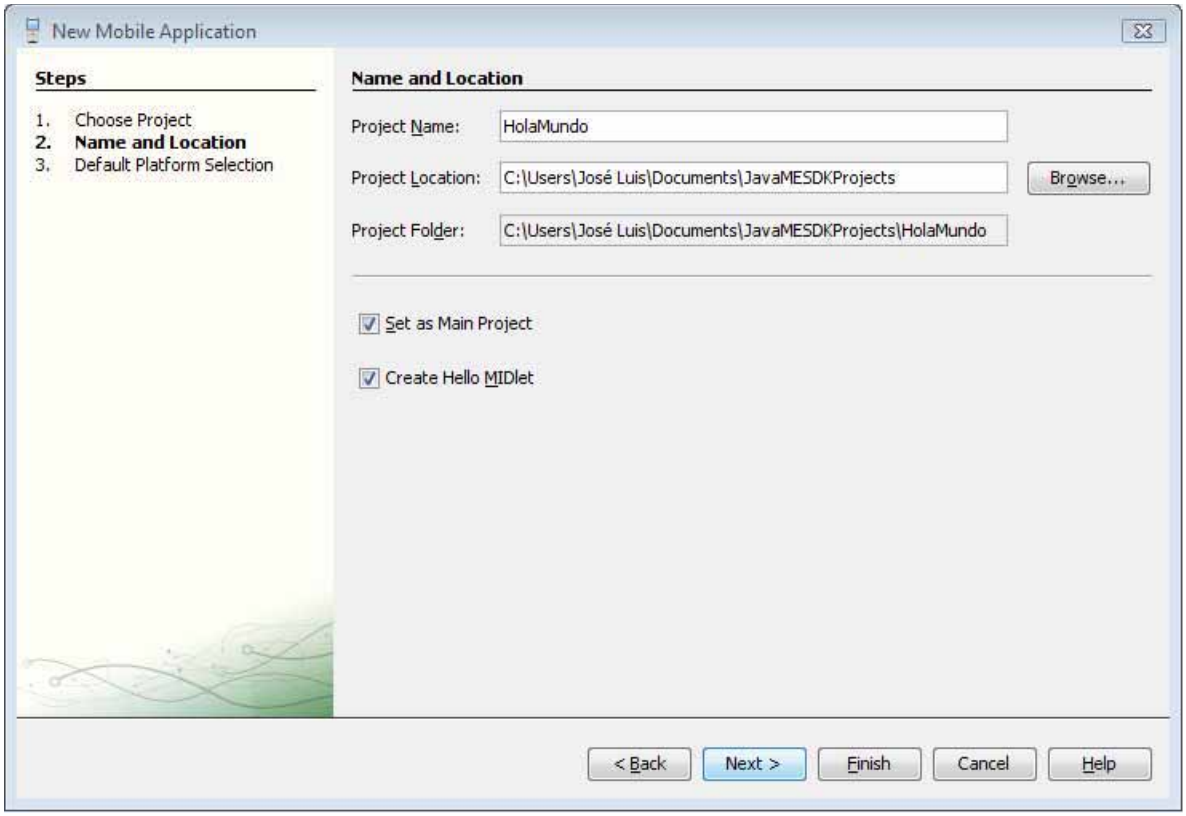

Figura 59: Crear un proyecto en la plataforma de desarrollo #3.

Después en este mismo cuadro de diálogo se selecciona la opción “Next >”. Al seleccionar esta opción se mostrará el cuadro de diálogo de la Figura 60, en donde se solicita que el usuario seleccione un emulador y un dispositivo, así como seleccionar la versión de CLDC-1.0 o CLDC 1.1 y una versión de MIDP-2.0 o MIDP-2.1.

Y finalmente se selecciona la opción “Finish”.

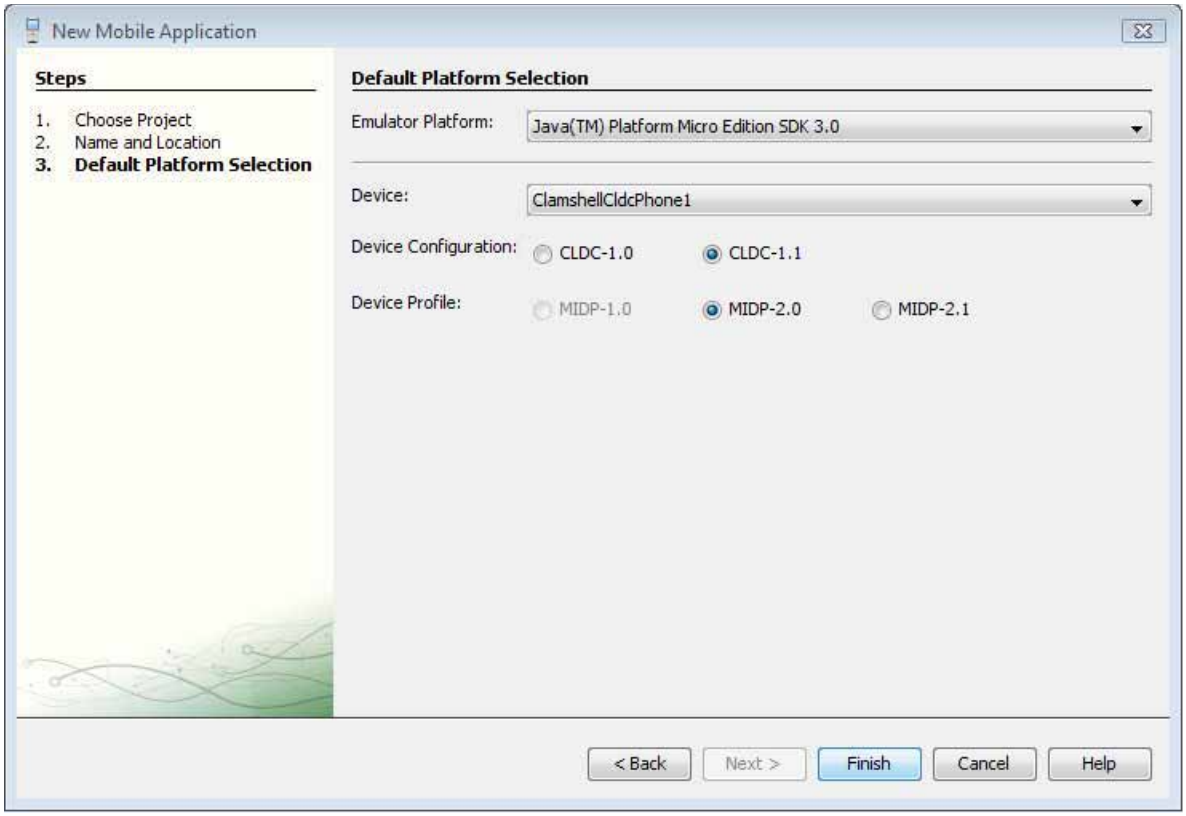

Figura 60: Crear un proyecto en la plataforma de desarrollo #4.

Al finalizar la configuración el entorno de desarrollo genera el código siguiente:

```
1   package hello;
2
3   import javax.microedition.midlet.*;
4   import javax.microedition.lcdui.*;
5
6   public class HelloMIDlet extends MIDlet implements CommandListener {
7
8       private Command exitCommand; // The exit command
9       private Display display;     // The display for this MIDlet
10
11      public HelloMIDlet() {
12          display = Display.getDisplay(this);
13          exitCommand = new Command("Exit", Command.EXIT, 0);
14      }
15
16      public void startApp() {
17          TextBox t = new TextBox("Hello", "Hello, World!", 256, 0);
18
19          t.addCommand(exitCommand);
20          t.setCommandListener(this);
21
22          display.setCurrent(t);
23      }
24
25      public void pauseApp() {
26      }
27
28      public void destroyApp(boolean unconditional) {
29      }
30
31      public void commandAction(Command c, Displayable s) {
32          if (c == exitCommand) {
33              destroyApp(false);
34              notifyDestroyed();
```

```
35                  }
36              }
37
38      }
```

En el código se muestra a una clase llamada HelloMidlet que debe de heredar de la clase *MIDlet* para crear un midlet.

La clase tiene dos datos miembro uno de nombre exitCommand de tipo *Command* y otro de nombre display de tipo *Display*. El primero es un objeto que modela a un botón en pantalla. El segundo es un objeto que representa a la pantalla y que es fundamental para desplegar información en pantalla.

Además, la clase HelloMIDlet implementa la interfaz *CommandListener*. Esto permite definir la funcionalidad del objeto *Command*, tal como se ve en las líneas que están en el intervalo de líneas 31 a 36 del código anterior, en el método *commandAction.*

La clase abstracta MIDlet exige que se implementen tres métodos: *startApp (intervalo de líneas [16 a 23])*, *pauseApp* y *destroyApp*.

El ciclo de vida del MIDlet define el momento en el que los métodos serán llamados.

En la Figura 61 se muestra de manera esquemática los estados por los que pasa un MIDlet, así como los métodos que realizan la transición.

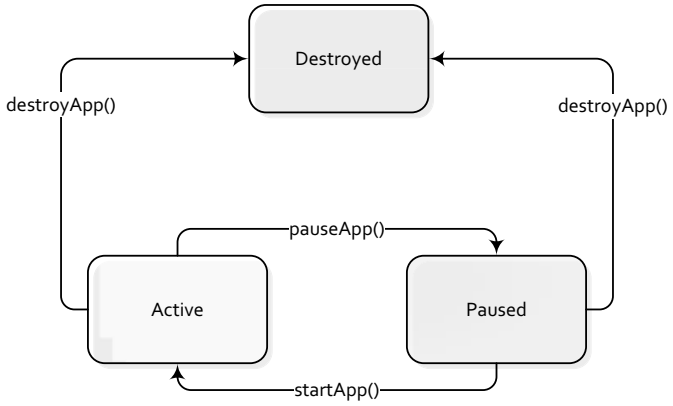

Figura 61: Ciclo de vida del MIDlet.

Al iniciar una aplicación CLDC/MIDP, y como todo objeto, se inicia por el constructor que es el que se llama al momento de iniciar la aplicación, seguido de la llamada al método *startApp*.

En el constructor se inician los datos miembro display y exitCommand.

Para iniciar el dato miembro display se llama al método estático *getDisplay* pasándole a esté la referencia de un objeto de tipo *MIDlet*, por lo que se le pasa a "*this*".

Para iniciar al objeto exitCommand se usa uno de los constructores de la clase *Command*, en este ejemplo al constructor se le pasa un cadena de caracteres con la etiqueta que tendrá el botón, un tipo de botón, estos tipos están definidos como miembros estáticos de la clase *Command*, tales como: EXIT, OK, BACK, etcétera, y finalmente se le pasa un entero con la prioridad del botón. Esto de la prioridad es útil cuando se desean colocar varios botones en pantalla.

En el método *startApp* se crea un nuevo objeto de tipo *TextBox* que permite crear un área para insertar texto.

Una observación importante que hay que hacer es que la mayoría de los objetos similares a *TextBox* ocupan toda la pantalla debido a las dimensiones de las mismas.

Por lo anterior *TextBox* es un objeto que ocupará toda la pantalla y por tal motivo se le deben de asociar a esta pantalla los botones que se mostrarán cuando esta pantalla este visible.

Algo más que hay que hacer notar es que todos los objetos que hereden de *Screen* ocuparán toda la pantalla y serán suceptibles de asociar botones o mejor dicho objetos de tipo *Command*.

Por ejemplo en el método startApp se muestra el siguiente fragmento de código:

```
39        public void startApp() {
40            TextBox t = new TextBox("Hello", "Hello, World!", 256, 0);
41
42            t.addCommand(exitCommand);
43            t.setCommandListener(this);
44
45            display.setCurrent(t);
46        }
```

Al tener la referencia “t” (que es de tipo *TextBox*) a esta hay que asociarle según se necesite un botón que en este caso es el botón exitCommand tal como se ve en la línea 42 del código.

También hay que asociar a la pantalla la clase que se encargará de manejar los eventos provenientes de presionar los botones; la clase encargada de manejar los eventos debe de implementar la interfaz *CommandListener* y como la clase HelloMIDlet la implementa, se pasa a “*this*”, tal como se ve en la línea 43.

Y finalmente hay que hacer visible el objeto *TextBox* para lo cual tenemos que llamar al método setCurrent del objeto display y pasarle como parámetro la referencia a un objeto de tipo *Displayable* que en este caso es TextBox, que hereda de *Screen* que a su vez hereda de *Displayable*.

En la Figura 62 se muestra a la aplicación “HolaMundo” que consta del código analizado en esta sección.

Figura 62: Ejecución de la aplicación HolaMundo.

B OBSERVACIONES DURANTE LA CODIFICACIÓN

Durante la codificación se presentaron varios eventos que es importante que se mencionen.

Hilos

Durante la codificación del componente gestión del trabajo colaborativo que es donde intervienen varios hilos de ejecución, se observó durante las pruebas a los subcomponentes, que conforman este mismo componente, que el comportamiento de los hilos en los emuladores de celular y en los celulares reales es muy diferente.

En los emuladores de celular se pudo observar que la ejecución de los hilos es de forma aleatoria y probablemente manejada por el planificador de tareas del sistema operativo sobre el que esta instalado el *Java ME Platform SDK 3.0*, que en este caso fue un sistema operativo *Windows Vista Home Edition*, en donde se soporta la multitarea. Mientras que cuando se realizó la prueba en el celular se noto que la ejecución de los hilos es secuencial, es decir que si existen tres hilos y sí se ejecuta en el orden siguiente: hilo 1, hilo 2 e hilo 3, en ese mismo orden se ejecutarán hasta la muerte de los hilos, mientras que en el emulador nuca se sabe en que orden se van a ejecutar los hilos.

Implementación de interfaz en una clase abstracta

Durante la codificación de la aplicación llego un momento en el que se debió definir una clase abstracta llamada Jugador, esta clase, según la lógica, debe de heredar de *Thread* e implementar la interfaz Colaborador. En la Figura 63 se muestra el diagrama de clases.

Como se ve en la Figura 63 la clase abstracta Jugador implementa la interfaz Colaborador y las clases ColaboradorRemoto y JugadorLocalServidor heredan de Jugador y por lo tanto deben de implementar todos los métodos abstractos de la clase Jugador.

Aquí el detalle esta en ¿qué pasaría si una clase maneja solo objetos de tipo Jugador (que en este caso es útil para manejar tanto objetos de tipo ColaboradorRemoto y JugadorLocalServidor) y quiere acceder a los métodos de la interfaz Colaborador?

La respuesta sería que tendría acceso dado que Jugador implementa la interfaz Colaborador, sin embargo, **el compilador no obliga al programado a que la clase abstracta implemente los métodos de la interfaz Colaborador,** esto es importante de hacer notar por que en el emulador de celulares funciona bien, es decir, una clase solo

maneja objetos de tipo Jugador y se le pasaron objetos de tipo ColaboradorRemoto y de tipo JugadorLocalServidor, y estos implementaron los métodos de la interfaz colaborador. La clase que solo maneja objetos de tipo Jugador puede acceder a los métodos de la interfaz Colaborador sin ningún problema.

Sin embargo, **cuando este mismo código que funcionó en el emulador se pasa a los dispositivos reales, estos no pueden acceder a los métodos de la interfaz colaborador**.

Este detalle es importante de mencionar por que cuando se probó en dos celulares, uno de marca *Sony Ericsson* modelos *W800i* y *S500i* no arroja ningún error ni ningún mensaje, pero cuando se probó la aplicación en un celular de marca *Nokia* modelo 5310, la aplicación arrojo un error indicando que el método no se encuentro.

La solución a este problema fue hacer que la clase abstracta Jugador sobre escribiera los métodos de la interfaz Colaborador y listo.

Aquí el problema estuvo en que **ni el compilador ni en el emulador arrojo error alguno** de hecho todo funcionó bien, pero a la hora de hacer pruebas con los celulares, antes mencionados, no funcionó la aplicación y el error era desconocido hasta que se probó en un celular *Nokia* en el que si se mostró un mensaje de error.

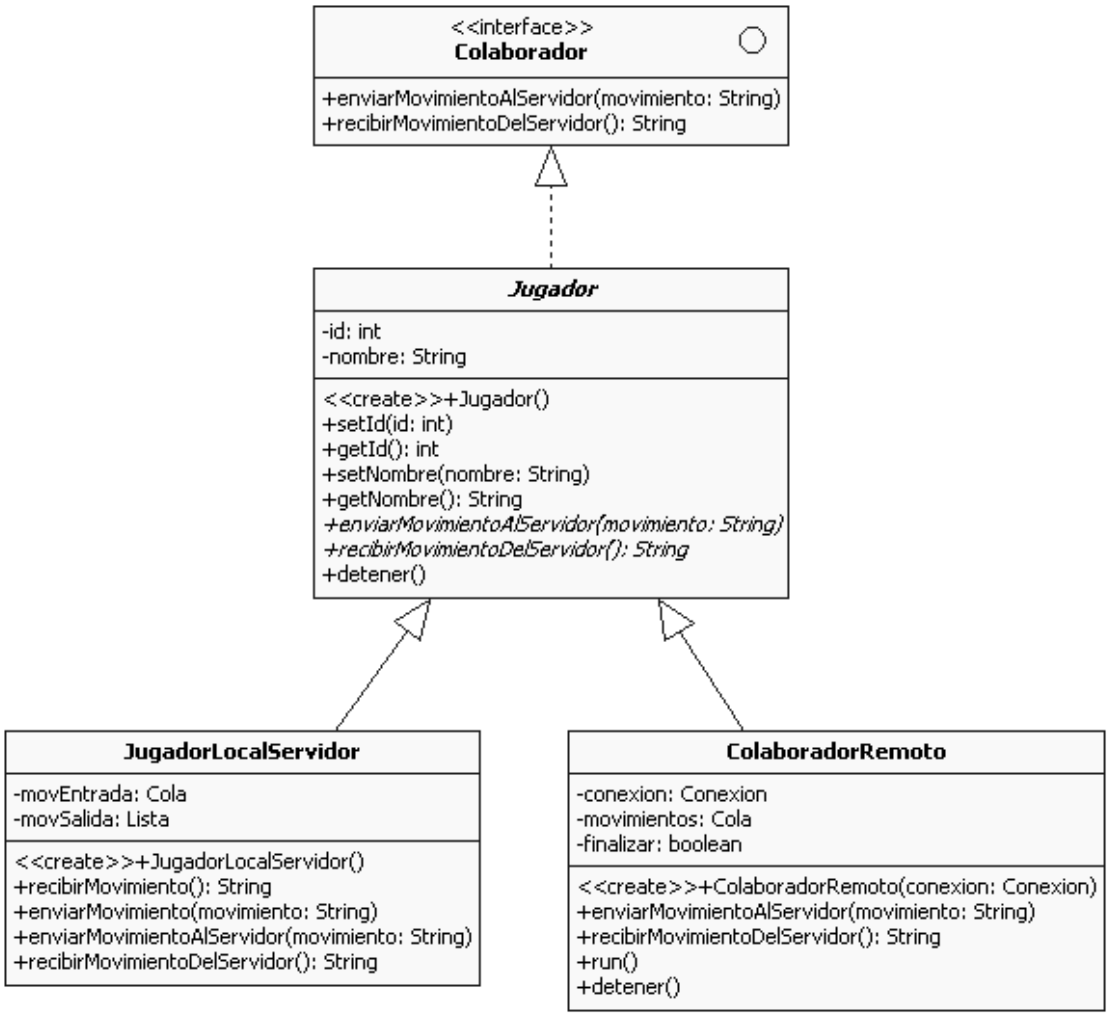

Figura 63: Diagrama de clase para: Clase abstracta implementa interfaz.

C CÓDIGO FUENTE

En este apartado se presenta todo el código fuente que comprende a la aplicación o sistema.

El código fuente estará organizado en los paquetes que se enlistan a continuación:

- uam.pt.ortigosa.bluetooth.
- uam.pt.ortigosa.colaborativo.
- uam.pt.ortigosa.graphics.
- uam.pt.ortigosa.midlet.
- uam.pt.ortigosa.util.
- uam.pt.ortigosa.util.xml.
- uam.pt.ortigosa.vistas.

Clases del paquete uam.pt.ortigosa.bluetooth

Clase BtServidor.java

```
1  package uam.pt.ortigosa.bluetooth;
2
3  import java.io.IOException;
4  import javax.bluetooth.BluetoothStateException;
5  import javax.bluetooth.LocalDevice;
6  import javax.bluetooth.RemoteDevice;
7  import javax.microedition.io.Connector;
8  import javax.microedition.io.StreamConnection;
9  import javax.microedition.io.StreamConnectionNotifier;
10 import javax.microedition.lcdui.Form;
11 import javax.microedition.lcdui.Gauge;
12 import javax.microedition.lcdui.Image;
13 import javax.microedition.lcdui.ImageItem;
14 import javax.microedition.lcdui.Item;
15 import uam.pt.ortigosa.colaborativo.JugadorRemotoServidor;
16 import uam.pt.ortigosa.colaborativo.TrabajoColaborativo;
17 import uam.pt.ortigosa.util.Config;
18
19 /**
20  * Clase que modela al servidor y que permite poner un servicio a disposición de los clientes
   que
21  * se conectarán al juego, es el encargado de determinar que se hará cuando llegue un cliente.
22  *
23  * @author José Luis Ortigosa Flores
24  */
25 public class BtServidor extends Thread {
26
27     private String ortigosaServiceName = "OrtigosaService";
28     private String connURL = "btspp://localhost:" + Config.ORTIGOSA_SERVICE_UUID.toString() +
   ";name=" + ortigosaServiceName + ";authorize=false";
29     private StreamConnectionNotifier streamConnectionNotifier;
```

```
30      private Image imageJugador;
31      private Form form;
32      private TrabajoColaborativo trabajoColaborativo;
33
34      /**
35       * Crea un objeto servidor, para lo cual requiere de que se le pasen las referencias a un
    objeto de tipo
36       * TrabajoColaborativo, y de un Form para enviar mensajes al usuario.
37       *
38       * @param trabajoColaborativo objeto que se encargará de gestionar a los jugadores.
39       * @param form en el que se desplegará información del servidor.
40       */
41      public BtServidor(TrabajoColaborativo trabajoColaborativo, Form form) {
42          try {
43              System.out.println(LocalDevice.getLocalDevice().getFriendlyName());
44          } catch (BluetoothStateException ex) {
45              ex.printStackTrace();
46          }
47          this.trabajoColaborativo = trabajoColaborativo;
48          try {
49              imageJugador = Image.createImage(Config.PATH_IMAGE + "/jugador.png");
50          } catch (IOException ex) {
51              System.err.println("Error: " + ex.getMessage());
52          }
53
54          this.form = form;
55          Gauge g = new Gauge("Esperando colaboradores...", false, Gauge.INDEFINITE,
    Gauge.CONTINUOUS_RUNNING);
56          form.append(g);
57      }
58
59      /**
60       * Inicia el servidor.
61       */
62      public void run() {
63          try {
64              streamConnectionNotifier = (StreamConnectionNotifier) Connector.open(connURL);
65              while (!trabajoColaborativo.conexionesListas()) {
66                  StreamConnection cliente = streamConnectionNotifier.acceptAndOpen();
67                  Conexion con = new Conexion(cliente);
68                  JugadorRemotoServidor jrs = new JugadorRemotoServidor(con);
69                  trabajoColaborativo.agregarJugador(jrs);
70                  String nombre = RemoteDevice.getRemoteDevice(cliente).getFriendlyName(true);
71                  form.append(new ImageItem(nombre, imageJugador, Item.LAYOUT_CENTER, "Jugador
    Image"));
72              }
73              form.delete(0);
74              trabajoColaborativo.init();
75
76          } catch (IOException ex) {
77              System.err.println("Error: " + ex.getMessage());
78          }
79      }
80  }
```

Clase BtCliente.java

```
1   package uam.pt.ortigosa.bluetooth;
2
3   import java.io.IOException;
4   import javax.bluetooth.BluetoothStateException;
5   import javax.bluetooth.DeviceClass;
6   import javax.bluetooth.DiscoveryAgent;
7   import javax.bluetooth.DiscoveryListener;
8   import javax.bluetooth.LocalDevice;
9   import javax.bluetooth.RemoteDevice;
10  import javax.bluetooth.ServiceRecord;
11  import javax.bluetooth.UUID;
12  import javax.microedition.io.Connector;
13  import javax.microedition.io.StreamConnection;
14  import javax.microedition.lcdui.Form;
15  import javax.microedition.lcdui.Image;
```

```
16  import javax.microedition.lcdui.List;
17  import uam.pt.ortigosa.util.Config;
18  import uam.pt.ortigosa.util.Lista;
19
20  /**
21   * Clase que permite modelar a un cliente bluetooth para comunicarse con un servidor a través de
    este medio.
22   * Su función es la de buscar y establecer conexión con un servidor que presente el servicio del
    juego.
23   *
24   * @author José Luis Ortigosa Flores.
25   */
26  public class BtCliente {
27
28      private LocalDevice localDevice;
29      private DiscoveryAgent agent;
30      private List deviceList;
31      private Form service;
32      private Lista devices;
33      private Lista services;
34      private BtListener btListener;
35      private Image user;
36      private String url;
37      private Conexion cliente;
38
39      /**
40       * Clase que permite crear a un objeto cliente bluetooth.
41       *
42       * @param deviceList es un objeto List que permite desplegar una lista seleccionable de
    dispositivos localizados.
43       * @param service es un objeto de tipo Form que permite informar al usuario de la existencia
    o no del juego en un
44       * dispositivo seleccionado.
45       */
46      public BtCliente(List deviceList, Form service) {
47          try {
48              localDevice = LocalDevice.getLocalDevice();
49              System.out.println(localDevice.getFriendlyName());
50              localDevice.setDiscoverable(DiscoveryAgent.GIAC);
51              agent = localDevice.getDiscoveryAgent();
52          } catch (BluetoothStateException ex) {
53              System.err.println("Error: " + ex.getMessage());
54          }
55
56          try {
57              user = Image.createImage(Config.PATH_IMAGE + "/usuario.png");
58          } catch (IOException ex) {
59              System.err.println("Error: " + ex.getMessage());
60          }
61
62          devices = new Lista();
63          services = new Lista();
64          this.deviceList = deviceList;
65          this.service = service;
66          btListener = new BtListener();
67          url = new String();
68      }
69
70      /**
71       * Método que permite obtener un objeto tipo Conexión que contiene la conexión establecida
    con el servidor.
72       * @return un objeto de tipo Conexion.
73       */
74      public Conexion getConexion() {
75          return this.cliente;
76      }
77
78      /**
79       * Método que hace que el objeto comience la busqueda de dispositivos.
80       */
81      public void buscarDispositivos() {
82          try {
83              agent.startInquiry(DiscoveryAgent.GIAC, btListener);
84          } catch (BluetoothStateException ex) {
85              System.err.println("Error: " + ex.getMessage());
```

```
86              }
87          }
88
89          /**
90           * Método que hace que el objeto comience la busqueda de los servicios disponibles en los
    dispositivos localizados.
91           */
92          public synchronized void buscarServicios() {
93              try {
94                  UUID[] uuid = new UUID[]{Config.ORTIGOSA_SERVICE_UUID, Config.BTSPP_UUID};
95                  if (!btListener.hayDispositivos()) { // si no hay dispositivos disponibles no inicia
    la busqueda de servicios
96                      try {
97                          wait();
98                      } catch (InterruptedException ex) {
99                          System.err.println("Error: " + ex.getMessage());
100                     }
101                 }
102                 int dispSel = deviceList.getSelectedIndex();
103                 agent.searchServices(null, uuid, (RemoteDevice) devices.get(dispSel), btListener);
104             } catch (BluetoothStateException e) {
105                 System.err.println("Error: " + e.getMessage());
106             }
107         }
108
109         /**
110          * Método que permite establecer la conexión con el servicio encontrado en algún
    dispositivo.
111          */
112         public void conectar() {
113             try {
114                 StreamConnection sc = (StreamConnection) Connector.open(url);
115                 cliente = new Conexion(sc);
116             } catch (IOException ex) {
117                 System.err.println("Error: " + ex.getMessage());
118             }
119         }
120
121         /**
122          * Método que permite saber si el servicio se encuentra disponible en el dispositivo
    deseado.
123          *
124          * @return verdadero o falso dependiendo de la situacion.
125          */
126         public boolean hayServicio() {
127             return btListener.hayServicios();
128         }
129
130         private class BtListener implements DiscoveryListener {
131
132             private boolean hayDispositivos;
133             private boolean hayServicios;
134
135             synchronized boolean hayDispositivos() {
136                 return hayDispositivos;
137             }
138
139             synchronized void setHayDispositivos(boolean hayDispositivos) {
140                 this.hayDispositivos = hayDispositivos;
141             }
142
143             synchronized boolean hayServicios() {
144                 return hayServicios;
145             }
146
147             synchronized void setHayServicios(boolean hayServicios) {
148                 this.hayServicios = hayServicios;
149             }
150
151             private void llenarListaDeDispositivos() {
152                 deviceList.deleteAll();
153                 for (int i = 0; i < devices.getSize(); i++) {
154                     RemoteDevice rd = (RemoteDevice) devices.get(i);
155                     try {
156                         deviceList.append(rd.getFriendlyName(false), user);
```

```
157                     } catch (IOException ex) {
158                         System.err.println("Error: " + ex.getMessage());
159                     }
160                 }
161             }
162 
163             public void deviceDiscovered(RemoteDevice btDevice, DeviceClass cod) {
164                 synchronized (this) {
165                     devices.agregar(btDevice);
166                     setHayDispositivos(true);
167                     notify();
168                 }
169             }
170 
171             public void inquiryCompleted(int discType) {
172                 switch (discType) {
173                     case INQUIRY_COMPLETED: {
174                         llenarListaDeDispositivos();
175                     }
176                     break;
177                     case INQUIRY_ERROR: {
178                         System.err.println("Hubo un error.");
179                     }
180                     break;
181                     case INQUIRY_TERMINATED: {
182                         System.err.println("El inquiry fue cancelado sin finalizar.");
183                     }
184                     break;
185                 }
186             }
187 
188             public void serviceSearchCompleted(int transID, int respCode) {
189                 if (hayServicios()) {
190                     service.append("El juego esta disponible.");
191                     url += ((ServiceRecord)
    services.removeFirst()).getConnectionURL(ServiceRecord.NOAUTHENTICATE_NOENCRYPT, false);
192                 } else {
193                     service.append("El juego no esta.");
194                 }
195             }
196 
197             public void servicesDiscovered(int transID, ServiceRecord[] servRecord) {
198                 for (int i = 0; i < servRecord.length; i++) {
199                     services.agregar(servRecord[i]);
200                 }
201                 if (servRecord.length > 0) {
202                     setHayServicios(true);
203                 }
204 
205             }
206         }
207     }
```

Clase Conexión.java

```
1   /*
2    * To change this template, choose Tools | Templates
3    * and open the template in the editor.
4    */
5   package uam.pt.ortigosa.bluetooth;
6 
7   import java.io.DataInputStream;
8   import java.io.DataOutputStream;
9   import java.io.IOException;
10  import javax.microedition.io.StreamConnection;
11  import uam.pt.ortigosa.util.Config;
12 
13  /**
14   * Clase que representa la conexión establecida entre el servidor y un cliente o viceversa.
15   * Proporciona los métodos necesarios para enviar y recibir mensajes de la conexión.
16   *
17   * @author José Luis
18   */
```

```
19  public class Conexion {
20
21      private StreamConnection socket;
22      private DataInputStream in;
23      private DataOutputStream out;
24
25      /**
26       * Crea un objeto en base a un StreamConnection
27       *
28       * @param socket es el socket con la conexión establecida entre el cliente y el servidor.
29       */
30      public Conexion(StreamConnection socket) {
31          this.socket = socket;
32          try {
33              in = this.socket.openDataInputStream();
34              out = this.socket.openDataOutputStream();
35          } catch (IOException ex) {
36              System.err.println("Error: " + ex.getMessage());
37          }
38      }
39
40      /**
41       * Método que envia strings a través del socket, los caracteres estan en UTF
42       *
43       * @param mensaje que será enviado a través de la conexión.
44       */
45      public void enviarMensaje(String mensaje) {
46          if (mensaje != null) {
47              try {
48                  out.writeUTF(mensaje);
49              } catch (IOException ex) {
50                  Config.error();
51                  System.err.println("Error: " + ex.getMessage());
52              }
53          }
54      }
55
56      /**
57       * Método encargado de leer las cadenas de caracteres de la conexión.
58       *
59       * @return un string leido de la conexión.
60       */
61      public String recibirMensaje() {
62          String mensaje = null;
63          try {
64              mensaje = in.readUTF();
65          } catch (IOException ex) {
66              Config.error();
67              System.err.println("Error: " + ex.getMessage());
68          }
69          return mensaje;
70      }
71
72      /**
73       * Método encargado de terminar la comunicación o de cerrar el socket.
74       */
75      public void finalizarConexion() {
76          try {
77              this.socket.close();
78          } catch (IOException ex) {
79              System.err.println("Error: " + ex.getMessage());
80          }
81      }
82  }
```

Clases del paquete uam.pt.ortigosa.colaborativo

Clase Colaborador.java

```
1   package uam.pt.ortigosa.colaborativo;
2
3   /**
4    * Interfaz que debén implementar los clientes que se comunicarán con el juego y este con el
    servidor.
5    *
6    * @author José Luis Ortigosa Flores Jose Luis
7    */
8   public interface Colaborador {
9
10      /**
11       * Método que permite enviar un movimiento al servidor.
12       * @param movimiento es la cadena de caracteres que sera enviada al servidor.
13       */
14      public void enviarMovimientoAlServidor(String movimiento);
15
16      /**
17       * Método que permite recibir un movimiento del servidor.
18       * @return el movimiento recibido por el servidor.
19       */
20      public String recibirMovimientoDelServidor();
21  }
```

Clase ColaboradorRemoto.java

```
1   package uam.pt.ortigosa.colaborativo;
2
3   import uam.pt.ortigosa.bluetooth.Conexion;
4   import uam.pt.ortigosa.util.Cola;
5   import uam.pt.ortigosa.util.Jugador;
6
7   /**
8    * Esta clase modela a los clientes remotos que estarán jugado remotamente y que participan en
    algún juego.
9    *
10   * @author José Luis Ortigosa Flores
11   */
12  public class ColaboradorRemoto extends Jugador {
13
14      private Conexion conexion;
15      private Cola movimientos;
16      private boolean finalizar;
17
18      /**
19       * Para crear un Objeto de este tipo se requiere de una conexión establecida con algún
    servidor.
20       *
21       * @param conexion establecida con algún servidor.
22       */
23      public ColaboradorRemoto(Conexion conexion) {
24          super();
25          this.conexion = conexion;
26          movimientos = new Cola();
27          finalizar = false;
28      }
29
30      public void enviarMovimientoAlServidor(String movimiento) {
31          if (movimiento != null) {
32              conexion.enviarMensaje(movimiento);
33          }
34      }
35
36      public String recibirMovimientoDelServidor() {
37          return (String) movimientos.removeFirst();
```

```
38          }
39
40          /**
41           * inicia la recepción de mensajes provenientes del servidor. Cada mensaje contiene
42           * movimientos reralizados por otros jugadores, los cueles se encolarán y se aplicarán en la
    forma y manera en
43           * la que lleguen.
44           */
45          public void run() {
46              while (!finalizar) {
47                  String mov = conexion.recibirMensaje();
48                  if (mov != null) {
49                      movimientos.agregar(mov);
50                  }
51              }
52          }
53
54          public void detener() {
55              finalizar = true;
56          }
57      }
```

Clase DistribucionDeMovimientos.java

```
1       package uam.pt.ortigosa.colaborativo;
2
3       import uam.pt.ortigosa.util.Lista;
4
5       /**
6        * Clase encargada de la distribución de los movimientos a todos los jugadores.
7        *
8        * @author José Luis Ortigosa Flores
9        */
10      public class DistribucionDeMovimientos extends Thread {
11
12          private TrabajoColaborativo trabajoColaborativo;
13          private Lista jugadores;
14          private boolean finalizar;
15
16          /**
17           * Permite crear al objeto que se encargará de distribuir los movimientos que esten siendo
    recopilados por
18           * el objeto Trabajo colaborativo.
19           *
20           * @param trabajoColaborativo que se esta encargando de la administración de los
    movimientos.
21           */
22          public DistribucionDeMovimientos(TrabajoColaborativo trabajoColaborativo) {
23              this.trabajoColaborativo = trabajoColaborativo;
24              jugadores = trabajoColaborativo.getJugadores();
25          }
26
27          /**
28           * Método que permite detener en cualquier momento la distribución de los movimientos hacia
    todos los jugadores.
29           */
30          public void detener() {
31              finalizar = true;
32          }
33
34          /**
35           * Inicia la distribución de los movimientos.
36           */
37          public void run() {
38              while (!finalizar) {
39                  String mov = trabajoColaborativo.getMovimiento();
40                  if (mov != null) {
41                      for (int i = 0; i < jugadores.getSize(); i++) {
42                          ((JugadorServidor) jugadores.get(i)).enviarMovimiento(mov);
43                      }
44                  }
45              }
46          }
```

```
47  }
48
```

Clase JugadorLocalServidor.java

```
1   package uam.pt.ortigosa.colaborativo;
2
3   import uam.pt.ortigosa.util.Cola;
4   import uam.pt.ortigosa.util.Lista;
5   import uam.pt.ortigosa.util.Jugador;
6
7   /**
8    * Clase que permite modelar a un cliente local. Por lo que este objeto tendrá que comportarse
    como
9    * un cliente remoto y como uno local de modo que tiene ambos comportamientos.
10   *
11   * @author José Luis Ortigosa Flores
12   */
13  public class JugadorLocalServidor extends Jugador implements JugadorServidor {
14
15      private Cola movEntrada; // el servidor colocara movimientos en esta cola
16      private Lista movSalida; // el servidor leera movimientos de esta cola
17
18      /**
19       * Permite crear a un jugador local que solo le será útil al servidor.
20       */
21      public JugadorLocalServidor() {
22          super();
23          movEntrada = new Cola();
24          movSalida = new Lista();
25      }
26
27      // para que el servidor le envie movimeientos al cliente local
28      /**
29       * Método utilizado por el objeto trabajoColaborativo que permite obtener los movimientos
    realizados por el cliente.
30       *
31       * @return el movimiento enviado por el cliente.
32       */
33      public String recibirMovimiento() {
34          return (String) movSalida.removeFirst();
35      }
36
37      // para que el servidor envie movimientos al cliente
38      /**
39       * Método que permite al servidor enviar movimientos al cliente local.
40       *
41       * @param movimiento que se enviará al jugador local.
42       */
43      public void enviarMovimiento(String movimiento) {
44          movEntrada.agregar(movimiento);
45      }
46
47      // para que el cliente envie movimientos al servidor
48      public void enviarMovimientoAlServidor(String movimiento) {
49          movSalida.agregar(movimiento);
50      }
51
52      // para que el cliente reciba los movimientos enviados por el servidor.
53      public String recibirMovimientoDelServidor() {
54          return (String) movEntrada.removeFirst();
55      }
56  }
57
```

Clase JugadorRemotoServidor.java

```
1   package uam.pt.ortigosa.colaborativo;
2
3   import uam.pt.ortigosa.bluetooth.Conexion;
4
5   /**
6    * Clase que permite manejar a los jugadores remotos desde la perspectiva del servidor.
7    *
8    * @author José Luis Ortigosa Flores
9    */
10  public class JugadorRemotoServidor implements JugadorServidor {
11
12      private Conexion conexion;
13
14      /**
15       * Crea un objeto para gestionar a un jugador remoto.
16       *
17       * @param conexion que debera ser atendida.
18       */
19      public JugadorRemotoServidor(Conexion conexion) {
20          this.conexion = conexion;
21      }
22
23      /**
24       * Método que permite recibir los movimientos enviados por el jugador.
25       *
26       * @return una cadena de caracteres con el mensaje enviado por el cliente remoto.
27       */
28      public String recibirMovimiento() {
29          return conexion.recibirMensaje();
30      }
31
32      /**
33       * Método que permite enviar movimientos hacia los clientes remotos.
34       *
35       * @param movimiento que será enviado.
36       */
37      public void enviarMovimiento(String movimiento) {
38          if (movimiento != null) {
39              conexion.enviarMensaje(movimiento);
40          }
41      }
42  }
43
```

Clase JugadorServidor.java

```
1   package uam.pt.ortigosa.colaborativo;
2
3   /**
4    * Interfaz que maneja el objeto TrabajoColaborativo para gestión de los jugadores, cualquier
    jugador que
5    * deba ser gestionado por el TrabajoColaborativo debe implementar esta interfaz.
6    *
7    * @author José Luis Ortigosa Flores
8    */
9   public interface JugadorServidor {
10
11      /**
12       * Método que permite recibir los movimientos realizados por un jugador que esta remoto
13       *
14       * @return el mensaje enviado por el jugador remoto.
15       */
16      public String recibirMovimiento();
17
18      /**
19       * Método que permite enviar movimientos a los jugadores remotos.
20       *
21       * @param movimiento
22       */
23      public void enviarMovimiento(String movimiento);
```

```
24      }
25
```

Clase RecepcionDeMovimientos.java

```
1   package uam.pt.ortigosa.colaborativo;
2
3   /**
4    * Clase encargada de recibir los movimientos de un jugador en particular. Cada movimiento
    recibido será enviado
5    * a un objeto de tipo TrabajoColaborativo para gestionar el movimiento.
6    *
7    * @author José Luis Ortigosa Flores
8    */
9   public class RecepcionDeMovimientos extends Thread {
10
11      private TrabajoColaborativo trabajoColaborativo;
12      private JugadorServidor jugadorServer;
13      private boolean finalizar;
14
15      /**
16       * Crea un objeto capáz de recibir todos los movimientos provenientes de los clientes
    remotos para
17       * gestionar cada uno de los movimientos de los jugadores.
18       *
19       * @param jugadorServidor jugador el cual se estará atendiendo para recibir sus movimientos
20       * @param trabajoColaborativo es la referencia del objeto encargado de la gestión del
    trabajo colaborativo.
21       */
22      public RecepcionDeMovimientos(JugadorServidor jugadorServidor, TrabajoColaborativo
    trabajoColaborativo) {
23          this.trabajoColaborativo = trabajoColaborativo;
24          this.jugadorServer = jugadorServidor;
25          this.finalizar = false;
26      }
27
28      /**
29       * Interrumpe la recepción de movimientos del jugador dado.
30       */
31      public void detener() {
32          finalizar = true;
33      }
34
35      /**
36       * Inicia la recepción de movimientos.
37       */
38      public void run() {
39          while (!finalizar) {
40              String movimiento = jugadorServer.recibirMovimiento();
41              if (movimiento != null) {
42                  trabajoColaborativo.agregarMovimiento(movimiento);
43              }
44          }
45      }
46  }
47
```

Clase TrabajoColaborativo.java

```
1   package uam.pt.ortigosa.colaborativo;
2
3   import uam.pt.ortigosa.util.Cola;
4   import uam.pt.ortigosa.util.Config;
5   import uam.pt.ortigosa.util.Lista;
6
7   /**
8    * Clase encargada de gestionar el trabajo colaborativo en otras palabras es la encargada de
    recibir y distribuir mensajes
9    * a todos los jugadores que se le haya indicado.
10   *
11   * @author José Luis Ortigosa Flores
12   */
```

```
13  public class TrabajoColaborativo {
14
15      private Cola movimientos;  // es el conjunto de movimientos acumulados que deberán ser
    repartidos.
16      private Lista jugadores; // Lista con los jugadores que estan participando.
17      private Lista hilosAtendiendo; // hilos que estan recibiendo los movimientos de los
    jugadores y los ingresan en la cola de movimientos
18      private DistribucionDeMovimientos distribucionDeMovimientos;
19
20      /**
21       * Constructor que permite crear un objeto para gestionar el trabajo colaborativo.
22       */
23      public TrabajoColaborativo() {
24          movimientos = new Cola();
25          jugadores = new Lista();
26          hilosAtendiendo = new Lista();
27          distribucionDeMovimientos = new DistribucionDeMovimientos(this);
28      }
29
30      /**
31       * Método que permite agregar a un jugador para ser gestionado.
32       *
33       * @param jugadorServidor al que deberá gestionarse.
34       */
35      public void agregarJugador(JugadorServidor jugadorServidor) {
36          jugadores.agregar(jugadorServidor);
37          RecepcionDeMovimientos rdm = new RecepcionDeMovimientos(jugadorServidor, this);
38          hilosAtendiendo.agregar(rdm);
39      }
40
41      /**
42       * Método que permite obtener una lista con todos los jugadores gestionados por este objeto.
43       * @return Una lista con los jugadores gestionados por este objeto.
44       */
45      public Lista getJugadores() {
46          return jugadores;
47      }
48
49      /**
50       * Método que permite agregar un movimiento que deberá gestionarse.
51       *
52       * @param movimiento que será gestionado.
53       */
54      public void agregarMovimiento(String movimiento) {
55          movimientos.agregar(movimiento);
56      }
57
58      /**
59       * Permite obtener el primer movimiento disponible.
60       *
61       * @return una cadena de caracteres con el primer mensaje encolado.
62       */
63      public String getMovimiento() {
64          return (String) movimientos.removeFirst();
65      }
66
67      /**
68       * Se encarga de detener el envío y recepción de mensajes y de la gestión colaborativa.
69       */
70      public void detener() {
71          for (int i = 0; i < hilosAtendiendo.getSize(); i++) {
72              ((RecepcionDeMovimientos) hilosAtendiendo.get(i)).detener();
73          }
74
75          distribucionDeMovimientos.detener();
76      }
77
78      /**
79       * Método que permite saber si los usuarios están o no listos.
80       * @return verdadero o falso dependiendo de si ya estan conectados todos los jugadores
    esperados.
81       */
82      public boolean conexionesListas() {
83          return Config.getNumJugadores() == jugadores.getSize();
84      }
```

```
85
86          /**
87           * Inicia la gestión colaborativa, así como el envío y recepción de movimientos.
88           */
89          public void init() {
90              for (int i = 0; i < hilosAtendiendo.getSize(); i++) {
91                  ((RecepcionDeMovimientos) hilosAtendiendo.get(i)).start();
92              }
93
94              distribucionDeMovimientos.start();
95          }
96      }
97
```

Clases del paquete uam.pt.ortigosa.graphics

Clase BarraPiezas.java

```
1   package uam.pt.ortigosa.graphics;
2
3   import java.io.IOException;
4   import java.util.Random;
5   import javax.microedition.lcdui.Graphics;
6   import javax.microedition.lcdui.Image;
7   import javax.microedition.lcdui.game.LayerManager;
8   import javax.microedition.lcdui.game.Sprite;
9   import javax.microedition.lcdui.game.TiledLayer;
10  import uam.pt.ortigosa.util.Config;
11
12  /**
13   * Clase que contiene todas las piezas que forman la imagen del rompecabezas,
14   * las piezas son mostradas de manera aleatoria.
15   *
16   * La barra cuenta con un cursor para que el usuario pueda seleccionar una pieza.
17   *
18   * @author José Luis Ortigosa Flores
19   */
20  public class BarraPiezas {
21
22      private TiledLayer tileBarra;
23      private LayerManager barra;
24      private Pieza cursor;
25      private int x;
26      private int y;
27      private int width;
28      private int vwX;
29      private int vwY;
30      private int piezaSel;
31      private int[] map;
32
33      /**
34       * @param widthScreen es la cantidad de pixeles que mide la pantalla en su parte horizontal.
35       * Esto permite que la barra se adapte a la pantalla.
36       */
37      public BarraPiezas(int widthScreen) {
38          map = new int[Config.COLS * Config.ROWS];
39          llenarMapa();
40          barra = new LayerManager();
41          this.vwX = 0;
42          this.vwY = 0;
43          cursor = new Pieza(0, "/cursor.png");
44          try {
45              tileBarra = new TiledLayer((Config.COLS * Config.ROWS), 1,
    Image.createImage(Config.PATH_IMAGE + "/objetivo.png"), Config.DIM_PZ, Config.DIM_PZ);
46          } catch (IOException ex) {
47              System.err.println("Error construyendo barra de piezas.");
48              System.err.println("Es posible que la imagen objetivo.png no este o no este en la
    ruta.");
49          }
```

```
50              for (int i = 0; i < map.length; i++) {
51                  tileBarra.setCell(i, 0, map[i]);
52              }
53              barra.append(tileBarra);
54              width = 0;
55
56              do {
57                  width += Config.DIM_PZ;
58              } while ((width + Config.DIM_PZ) <= widthScreen);
59
60              this.x = (widthScreen / 2) - (width / 2);
61              this.y = 0;
62
63              barra.setViewWindow(vwX, vwY, width, Config.DIM_PZ);
64              cursor.getSprite().setPosition(x, y);
65              piezaSel = 0;
66          }
67
68          private void llenarMapa() {
69              Random rnd = new Random();
70              for (int i = 0; i < map.length; i++) {
71                  while (true) {
72                      int ale = (Math.abs(rnd.nextInt()) % ((Config.COLS * Config.ROWS))) + 1;
73                      if (!estaEnMapa(ale)) {
74                          map[i] = ale;
75                          break;
76                      }
77                  }
78              }
79          }
80
81          private boolean estaEnMapa(int numPz) {
82              for (int i = 0; i < map.length; i++) {
83                  if (map[i] == numPz) {
84                      return true;
85                  }
86              }
87              return false;
88          }
89
90          /**
91           * Método que muestra la parte posterior de una pieza.
92           */
93          public void ocultarPiezaSel() {
94              tileBarra.setCell(piezaSel, 0, 17);
95          }
96
97          /**
98           * Método que sirve para situar la barra de piezas en la pantalla.
99           *
100          * @param x es la posición en la parte horizontal.
101          * @param y es la posicion en la parte vertical.
102          */
103         public void setPosicion(int x, int y) {
104             if (x > 0) {
105                 this.x = x;
106             }
107             this.y = y;
108             cursor.getSprite().setPosition(this.x, this.y);
109         }
110
111         /**
112          * @return la cantidad de pixeles que mide la barra de piezas en su parte horizontal.
113          */
114         public int getWidth() {
115             return this.width;
116         }
117
118         /**
119          * @return la cantidad de pixeles que mide la barra de piezas en su parte vertical.
120          */
121         public int getHeight() {
122             return tileBarra.getHeight();
123         }
124
```

```
125         /**
126          * Método que permite mover el cursor a la izquierda de la barra de piezas.
127          */
128         public void moverCursorIzq() {
129             Sprite s = cursor.getSprite();
130             if ((s.getX() - Config.DIM_PZ) >= x) {
131                 s.setPosition(s.getX() - Config.DIM_PZ, s.getY());
132                 piezaSel--;
133             } else {
134                 scrollingIzq();
135             }
136         }
137 
138         /**
139          * Método que permite mover el cursor a la derecha de la barra de piezas.
140          */
141         public void moverCursorDer() {
142             Sprite s = cursor.getSprite();
143             if ((s.getX() + Config.DIM_PZ) < (x + width)) {
144                 s.setPosition(s.getX() + Config.DIM_PZ, s.getY());
145                 piezaSel++;
146             } else {
147                 scrollingDer();
148             }
149         }
150 
151         private void scrollingIzq() {
152             if ((vwX - Config.DIM_PZ) >= 0) {
153                 barra.setViewWindow(vwX -= Config.DIM_PZ, vwY, width, Config.DIM_PZ);
154                 piezaSel--;
155             }
156         }
157 
158         private void scrollingDer() {
159             if ((vwX + width) < tileBarra.getWidth()) {
160                 barra.setViewWindow(vwX += Config.DIM_PZ, vwY, width, Config.DIM_PZ);
161                 piezaSel++;
162             }
163         }
164 
165         /**
166          * Pinta la barra de piezas en la pantalla del dispositivo.
167          * @param g
168          */
169         public void paint(Graphics g) {
170             g.setColor(Config.COLOR_BARRA_FONDO);
171             g.fillRect(x - 4, y - 4, this.getWidth() + 8, this.getHeight() + 8);
172             barra.paint(g, x, y);
173             cursor.paint(g);
174         }
175 
176         /**
177          * Método que permite ocultar la pieza con el id señalado.
178          *
179          * @param id de la pieza que deberá ser ocultado.
180          */
181         public void ocultarPieza(int id) {
182             for (int i = 0; i < map.length; i++) {
183                 if (map[i] == id) {
184                     tileBarra.setCell(i, 0, 17);
185                     return;
186                 }
187             }
188         }
189 
190         /**
191          * @return el id de la pieza seleccionada con el cursor.
192          */
193         public int getIdPiezaSel() {
194             return map[this.piezaSel];
195         }
196     }
197 
```

Clase Encabezado.java

```
1   package uam.pt.ortigosa.graphics;
2
3   import javax.microedition.lcdui.Font;
4   import javax.microedition.lcdui.Graphics;
5   import uam.pt.ortigosa.util.Config;
6
7   /**
8    * Clase que permite dibujar el encabezado, que contiene los títulos del juego.
9    *
10   * @author José Luis Ortigosa Flores
11   */
12  public class Encabezado {
13
14      private Font font;
15      private String titulo;
16      private String subtitulo;
17      private int width;
18      private int height;
19
20      /**
21       * @param widthScreen es la cantidad de pixeles de la pantalla en la parte horizontal.
22       * @param height es la cantidad de pixeles que tendrá la altura del encabezado.
23       */
24      public Encabezado(int widthScreen, int height) {
25          this.width = widthScreen - 4;
26          this.height = height;
27          titulo = new String("SOFTWARE COLABORATIVO");
28          subtitulo = new String("ROMPECABEZAS");
29          font = Font.getFont(Font.FACE_PROPORTIONAL, Font.STYLE_BOLD, Font.SIZE_SMALL);
30      }
31
32      /**
33       * @return la cantidad de pixeles que mide la altura del encabezado.
34       */
35      public int getHeight() {
36          return this.height;
37      }
38
39      /**
40       * Método que se encarga de pintar el encabezado en la pantalla del juego.
41       * @param g
42       */
43      public void paint(Graphics g) {
44          g.setFont(font);
45
46          g.setColor(Config.COLOR_HEADER_FONDO_1);
47          g.fillRect(2, 2, width, (height / 2));
48
49          g.setColor(Config.COLOR_HEADER_FONDO_2);
50          g.fillRect(2, (height / 2) + 1, width, height / 2);
51
52          g.setColor(Config.COLOR_HEADER_FONT_1);
53          g.drawString(titulo, (int) (width / 2), (int) ((height / 6)), Graphics.TOP |
    Graphics.HCENTER);
54
55          g.setColor(Config.COLOR_HEADER_FONT_2);
56          g.drawString(subtitulo, (int) (width / 2), (int) ((height / 6) * 4), Graphics.TOP |
    Graphics.HCENTER);
57      }
58  }
59
```

Clase estatus.java

```
1   package uam.pt.ortigosa.graphics;
2
3   import javax.microedition.lcdui.Font;
4   import javax.microedition.lcdui.Graphics;
5   import uam.pt.ortigosa.util.Config;
6
```

```
7   /**
8    * Clase que permite mostrar en pantalla una área para mostrar mensajes en pantalla y que haya
    interacción con el usuario.
9    *
10   * @author José Luis Ortigosa Flores
11   */
12  public class Estatus {
13
14      private Font font;
15      private String msg;
16      private int width;
17      private int height;
18      private int x;
19      private int y;
20
21      /**
22       * @param height es la altura en pixeles que deberá tener la barra de estatus.
23       * @param widthScreen es la cantidad de pixeles que la pantalla.
24       */
25      public Estatus(int widthScreen, int height) {
26          this.width = widthScreen - 4;
27          this.height = height;
28          this.x = 2;
29          this.y = 0;
30          msg = new String();
31          font = Font.getFont(Font.FACE_PROPORTIONAL, Font.STYLE_PLAIN, Font.SIZE_SMALL);
32      }
33
34      /**
35       * Pinta la barra de estatus en la pantalla del dispositivo.
36       * @param g
37       */
38      public void paint(Graphics g) {
39          g.setFont(font);
40          g.setColor(0xffffff);
41          g.fillRect(x, y, width, height);
42          g.setColor(Config.COLOR_FOOTER_BORDER);
43          g.drawRect(x, y, width, height);
44          g.setColor(Config.COLOR_FOOTER_FONT);
45          g.drawString(msg, (int) width / 2, y, Graphics.TOP | Graphics.HCENTER);
46      }
47
48      /**
49       * @return la cantidad de pixles en su parte horizontal.
50       */
51      public int getWidth() {
52          return this.width;
53      }
54
55      /**
56       * @return la cantidad de pixleles en su parte vertical.
57       */
58      public int getHeight() {
59          return this.height;
60      }
61
62      /**
63       * Método que permite establecer la posición de este objeto.
64       *
65       * @param x posición horizontal.
66       * @param y posición vertical.
67       */
68      public void setPosicion(int x, int y) {
69          if (x > 0) {
70              this.x = x;
71          }
72          this.y = y;
73
74      }
75
76      /**
77       * Método que permite establecer el texto que se mostrara en el mensaje.
78       *
79       * @param text es el mensaje que se deberá visualizar en la barra de estatus.
80       */
```

```
81          public void settext(String text) {
82              msg = text;
83          }
84      }
85
```

Clase Game.java

```
1       package uam.pt.ortigosa.graphics;
2
3       import javax.microedition.lcdui.Graphics;
4       import javax.microedition.lcdui.game.GameCanvas;
5       import uam.pt.ortigosa.util.Config;
6       import uam.pt.ortigosa.util.Jugador;
7       import uam.pt.ortigosa.util.xml.GenerarMensaje;
8       import uam.pt.ortigosa.util.xml.Mensaje;
9
10      /**
11       * Clase que se encarga de todo el control del juego de rompecabezas.
12       *
13       * @author José Luis Ortigosa Flores
14       */
15      public class Game extends GameCanvas implements Runnable {
16
17          private static final int NUM_ELEMENTS_EN_PANTALLA = 3;
18          private static Estatus estatus;
19          private Graphics g;
20          private Encabezado header;
21          private Puzzle puzzle;
22          private BarraPiezas barra;
23          private int[] posicionY;
24          private boolean finalizar = false;
25          private boolean jugarSolo = false;
26          private Jugador jugador;
27          private AplicarMovimientosRemotos aplicarMovimientosRemotos;
28
29          /**
30           * Constructor que permite crear una instancia del juego.
31           *
32           * @param suppressKeyEvents true to suppress the regular key event mechanism for game keys,
        otherwise false
33           * @param jugador es la referencia la jugador local puede ser un Colaborador remoto u otro
        que herede de la clase
34           * abstracta jugador.
35           * @param jugarSolo método que le permite decidir al juego si se debe estar en modo
        colaborativo o en modo solitario.
36           */
37          public Game(boolean suppressKeyEvents, Jugador jugador, boolean jugarSolo) {
38              super(suppressKeyEvents);
39              this.jugador = jugador;
40
41              this.jugarSolo = jugarSolo;
42              g = getGraphics();
43              setFullScreenMode(true);
44              posicionY = new int[NUM_ELEMENTS_EN_PANTALLA];
45
46              header = new Encabezado(getWidth(), 40);
47              puzzle = new Puzzle(getWidth());
48              barra = new BarraPiezas(getWidth());
49              estatus = new Estatus(getWidth(), 20);
50
51              distribuirPosicionesVerticalmente();
52
53              if (!jugarSolo) {
54                  aplicarMovimientosRemotos = new AplicarMovimientosRemotos();
55              }
56          }
57
58          private void distribuirPosicionesVerticalmente() {
59              int suma = header.getHeight() + puzzle.getHeight() + barra.getHeight() +
        estatus.getHeight();
60              int pixelRestantes = getHeight() - suma;
61              int espacio = pixelRestantes / 3;
62
```

```
63            posicionY[0] = header.getHeight() + espacio;
64            posicionY[1] = posicionY[0] + puzzle.getHeight() + espacio;
65            posicionY[2] = posicionY[1] + barra.getHeight() + espacio - 2;
66
67            puzzle.setPosicion(-1, posicionY[0]);
68            barra.setPosicion(-1, posicionY[1]);
69            estatus.setPosicion(-1, posicionY[2]);
70        }
71
72        private void revisar() {
73            if (puzzle.getIdPiezaSel() == barra.getIdPiezaSel()) {
74                puzzle.mostrarPiezaSel();
75                barra.ocultarPiezaSel();
76                setMsg("Bien!!");
77            } else {
78                setMsg("Mala elección!!");
79            }
80        }
81
82        private void keyCodeProcess(int keyCode) {
83            switch (getGameAction(keyCode)) {
84                case FIRE: {
85                    if (jugarSolo) {
86                        revisar();
87                    } else if (Config.hayErrorCom()) {
88                        setMsg("Error Com.");
89                        revisar();
90                    } else {
91                        enviarMovimientoLocal();
92                    }
93                }
94                break;
95                case UP: {
96                    puzzle.moverCursorUp();
97                }
98                break;
99                case DOWN: {
100                   puzzle.moverCursorDown();
101               }
102               break;
103               case RIGHT: {
104                   puzzle.moverCursorDer();
105               }
106               break;
107               case LEFT: {
108                   puzzle.moverCursorIzq();
109               }
110               break;
111               default:
112                   break;
113           }
114
115           switch (keyCode) {
116               case KEY_NUM0: {
117                   if (jugarSolo) {
118                       revisar();
119                   } else if (Config.hayErrorCom()) {
120                       setMsg("Error Com.");
121                       revisar();
122                   } else {
123                       enviarMovimientoLocal();
124                   }
125               }
126               break;
127               case KEY_NUM1: {
128                   barra.moverCursorIzq();
129               }
130               break;
131               case KEY_STAR: {
132                   barra.moverCursorIzq();
133               }
134               break;
135               case KEY_NUM3: {
136                   barra.moverCursorDer();
137               }
```

```
138                 break;
139                 case KEY_POUND: {
140                     barra.moverCursorDer();
141                 }
142                 break;
143                 default:
144                     break;
145             }
146         }
147 
148         /**
149          * Método que permite procesar las acciones que se ejecutarán al presionar alguna de las
150          * teclas que son útiles para el juego.
151          * @param keyCode
152          */
153         protected void keyPressed(int keyCode) {
154             super.keyPressed(keyCode);
155             keyCodeProcess(keyCode);
156         }
157 
158         /**
159          * Método que permite poner un mensaje en la barra de estatus del juego.
160          *
161          * @param msg es el mensaje que se imprimirá en pantalla.
162          */
163         public static void setMsg(String msg) {
164             estatus.settext(msg);
165         }
166 
167         /**
168          * Método que permite obtener el puzzle del juego.
169          *
170          * @return el puzzle del juego.
171          */
172         public Puzzle getPuzzle() {
173             return this.puzzle;
174         }
175 
176         private void aplicarMovimientoJugador() {
177             String mov = jugador.recibirMovimientoDelServidor();
178 
179             if (mov != null) {
180     //            Mensaje m = GenerarXML.getMensajeXML(mov);
181                 Mensaje m = GenerarMensaje.getMensaje(mov);
182                 puzzle.mostrarPieza(m.getPiezaCoordenadaI(), m.getPiezaCoordenadaJ());
183                 barra.ocultarPieza(m.getPiezaId());
184                 puzzle.avanzar();
185                 setMsg(m.getJugadorNombre() + " coloca pieza: " + m.getPiezaId());
186             }
187         }
188 
189         private void enviarMovimientoLocal() {
190             if (puzzle.getIdPiezaSel() == barra.getIdPiezaSel()) {
191     //            String mov = GenerarXML.getXML(jugador, puzzle.getPiezaSel());
192                 String mov = GenerarMensaje.getMensaje(jugador, puzzle.getPiezaSel());
193                 jugador.enviarMovimientoAlServidor(mov);
194                 setMsg("Bien!!");
195             } else {
196                 setMsg("Mala elección!!");
197             }
198         }
199 
200         /**
201          * Método que pone en marcha el juego, para que todos los participantes puedan iniciar el
    juego.
202          */
203         public void run() {
204             header.paint(g);
205 
206             if (!jugarSolo) {
207                 aplicarMovimientosRemotos.start();
208             }
209 
210             while (!finalizar) {
211                 puzzle.paint(g);
```

```
212                 barra.paint(g);
213                 estatus.paint(g);
214                 if (puzzle.estaCompleto()) {
215                     setMsg("Juego Completado!!");
216                 }
217                 flushGraphics();
218             }
219         }
220 
221         /**
222          * Método que permite finalizar el juego.
223          */
224         public void finalizarJuego() {
225             this.finalizar = true;
226         }
227 
228         private class AplicarMovimientosRemotos extends Thread {
229 
230             public void run() {
231                 while (!finalizar && !Config.hayErrorCom()) {
232                     aplicarMovimientoJugador();
233                 }
234             }
235         }
236     }
237 
```

Clase Pieza.java

```
1   package uam.pt.ortigosa.graphics;
2   
3   import java.io.IOException;
4   import javax.microedition.lcdui.Graphics;
5   import javax.microedition.lcdui.Image;
6   import javax.microedition.lcdui.game.Sprite;
7   import uam.pt.ortigosa.util.Config;
8   
9   /**
10   * Clase que modela a una pieza del rompecabezas, para su control se le puede indicar un id de
    pieza.
11   * Basicamente se trata de una envoltura de la clase Sprite, donde esta deberá tener al menos
    tres frames diferentes
12   * uno para error uno para la parte posterior de la pieza y otro para la parte frontal de la
    pieza.
13   *
14   * @author Java
15   */
16  public class Pieza {
17  
18      private int id;
19      private Sprite sprite;
20      private int ci;
21      private int cj;
22  
23      /**
24       * Para crear una pieza del rompecabezas se requiere de un id y una imagen que contenga las
    dimensiones
25       * que establece la clase Config en su variable final y static DIM_PZ.
26       * @param id
27       * @param img
28       */
29      public Pieza(int id, String img) {
30          this.id = id;
31          try {
32              sprite = new Sprite(Image.createImage(Config.PATH_IMAGE + img), Config.DIM_PZ,
    Config.DIM_PZ);
33          } catch (IOException ex) {
34              System.err.println("Error cargando imagen en un pieza del juego.");
35              System.err.println("Error con el nombre, o la ruta, o no esta la imagen.");
36          }
37      }
38  
39      /**
40       * Método que permite indicar que la pieza debe ser visible en pantalla.
```

```
41          * Cambia al frame 0.
42          */
43         public void mostrar() {
44             sprite.setFrame(0);
45         }
46 
47         /**
48          * Método que permite indicar que la pieza debe de ser ocultada al jugador.
49          * Cambia al frame 1.
50          */
51         public void ocultar() {
52             sprite.setFrame(1);
53         }
54 
55         /**
56          * Método que permite mostrar en pantalla un frame de error.
57          */
58         public void error() {
59             sprite.setFrame(2);
60         }
61 
62         /**
63          * @return el Sprite envuelto en esta clase.
64          */
65         public Sprite getSprite() {
66             return this.sprite;
67         }
68 
69         /**
70          * @return el id de la pieza.
71          */
72         public int getId() {
73             return this.id;
74         }
75 
76         /**
77          * Método que permite establecer la posición de la pieza dentro de la regilla rectangular
    del rompecabezas.
78          *
79          * @param i
80          * @param j
81          */
82         public void setCoordenadas(int i, int j) {
83             this.ci = i;
84             this.cj = j;
85         }
86 
87         /**
88          * @return i-esima posición de la pieza dentro del puzzle.
89          */
90         public int getI() {
91             return ci;
92         }
93 
94         /**
95          * @return j-esima posición de la pieza dentro del puzzle.
96          */
97         public int getJ() {
98             return cj;
99         }
100 
101        /**
102         * Método que permite pintar la pieza en la pantalla.
103         *
104         * @param g
105         */
106        public void paint(Graphics g) {
107            sprite.paint(g);
108        }
109    }
110 
```

Clase Puzzle.java

```
1   package uam.pt.ortigosa.graphics;
2
3   import javax.microedition.lcdui.Graphics;
4   import javax.microedition.lcdui.game.Sprite;
5   import uam.pt.ortigosa.util.Config;
6
7   /**
8    * Clase que permite dibujar un rompecabezas en la pantalla del dispositivo, las imagenes deben
    de estar en el path de
9    * imagenes indicado en la clase Config, y deberán de estar enumeradas del 1 hasta el número que
    resulte de la multiplicación
10   * de las columnas por las filas que tendrá el rompecabezas, estas variables también deberán de
    establecerse en la clase Config.
11   * En el caso de que se desee la generación de un nuevo rompecabezas, se deberán cambiar las
    imagenes pero conservar los nombres
12   * así como los cambios pertinentes en las variables necesarias.
13   *
14   * @author José Luis Ortigosa Flores
15   */
16  public class Puzzle {
17
18      private Pieza[][] puzzle;
19      private Pieza cursor;
20      private int x;
21      private int y;
22      private Pieza piezaSel;
23      private int width;
24      private int piezasFaltantes = Config.COLS * Config.ROWS;
25
26      /**
27       * Para la creación el puzzle se requiere de las cantidad de pixeles de la pantalla en su
    parte horizontal.
28       * @param width es la cantidad de pixeles en la parte horizontal de la pantalla.
29       */
30      public Puzzle(int width) {
31          this.width = width;
32
33          puzzle = new Pieza[Config.COLS][Config.ROWS];
34          int countImg = 1;
35          for (int i = 0; i < Config.COLS; i++) {
36              for (int j = 0; j < Config.ROWS; j++) {
37                  puzzle[i][j] = new Pieza(countImg, "/" + (countImg++) + ".png");
38                  puzzle[i][j].setCoordenadas(i, j);
39                  puzzle[i][j].ocultar();
40              }
41          }
42          cursor = new Pieza(0, "/cursor.png");
43          this.x = (width / 2) - (this.getWidth() / 2);
44          this.y = 0;
45          cursor.getSprite().setPosition(x, y);
46      }
47
48      /**
49       * Método que permite establecer la posición dentro de la pantalla de todo el objeto puzzle.
50       *
51       * @param x posicion a lo horizontal de la pantalla.
52       * @param y posicion a lo vertical en la pantalla.
53       */
54      public void setPosicion(int x, int y) {
55          if (x > 0) {
56              this.x = x;
57          }
58          this.y = y;
59          for (int i = 0; i < Config.COLS; i++) {
60              for (int j = 0; j < Config.ROWS; j++) {
61                  puzzle[i][j].getSprite().setPosition(this.x + (j * Config.DIM_PZ), this.y + (i *
    Config.DIM_PZ));
62              }
63          }
64          cursor.getSprite().setPosition(this.x, this.y);
65      }
66
```

```
67          /**
68           * Método que permite mover el cursor una posición hacia arriba dentro del tablero.
69           * Siempre que no se salga del tablero.
70           */
71          public void moverCursorUp() {
72              Sprite s = cursor.getSprite();
73              if ((s.getY() - Config.DIM_PZ) >= y) {
74                  s.setPosition(s.getX(), s.getY() - Config.DIM_PZ);
75              }
76          }
77
78          /**
79           * Método que permite mover el cursor una posición hacia abajo dentro del tablero.
80           * Siempre que no se salga del tablero.
81           */
82          public void moverCursorDown() {
83              Sprite s = cursor.getSprite();
84              if ((s.getY() + Config.DIM_PZ) < y + (Config.ROWS * Config.DIM_PZ)) {
85                  s.setPosition(s.getX(), s.getY() + Config.DIM_PZ);
86              }
87          }
88
89          /**
90           * Método que permite mover el cursor una posición hacia la derecha dentro del tablero.
91           * Siempre que no se salga del tablero.
92           */
93          public void moverCursorDer() {
94              Sprite s = cursor.getSprite();
95              if ((s.getX() + Config.DIM_PZ) < x + (Config.COLS * Config.DIM_PZ)) {
96                  s.setPosition(s.getX() + Config.DIM_PZ, s.getY());
97              }
98          }
99
100         /**
101          * Método que permite mover el cursor una posición hacia la izquierda dentro del tablero.
102          * Siempre que no se salga del tablero.
103          */
104         public void moverCursorIzq() {
105             Sprite s = cursor.getSprite();
106             if ((s.getX() - Config.DIM_PZ) >= x) {
107                 s.setPosition(s.getX() - Config.DIM_PZ, s.getY());
108             }
109         }
110
111         /**
112          * Método que permite obtener el id de la pieza que esta seleccionada.
113          * @return el id de la pieza seleccionada con el cursor.
114          */
115         public int getIdPiezaSel() {
116             Sprite c = cursor.getSprite();
117             for (int i = 0; i < Config.COLS; i++) {
118                 for (int j = 0; j < Config.ROWS; j++) {
119                     if (c.collidesWith(puzzle[i][j].getSprite(), true)) {
120                         piezaSel = puzzle[i][j];
121                         return puzzle[i][j].getId();
122                     }
123                 }
124             }
125             return -1;
126         }
127
128         /**
129          * Método que permite obtener la pieza seleccionada.
130          *
131          * @return la pieza seleccioanda.
132          */
133         public Pieza getPiezaSel() {
134             return piezaSel;
135         }
136
137         /**
138          * Método que hace visible en pantalla la pieza seleccionada con el cursor.
139          * Cada que este método es llamado, se tiene avance en el juego.
140          */
141         public void mostrarPiezaSel() {
```

```
142             piezaSel.mostrar();
143             piezasFaltantes--;
144         }
145 
146         /**
147          * Método que hace que la pieza muestre el frame de error de la pieza seleccionada, se
    recomienda
148          * en caso de que se quiera mostrar en pantalla un error con un frame.
149          */
150         public void errorPiezaSel() {
151             piezaSel.error();
152         }
153 
154         /**
155          * Método que hace que la pieza muestre el frame que se supone es la parte posterior de una
    pieza.
156          */
157         public void ocultarPiezaSel() {
158             piezaSel.ocultar();
159             piezasFaltantes++;
160         }
161 
162     //    public boolean estaCompleto() {
163     //        Sprite p;
164     //        for (int i = 0; i < Config.COLS; i++) {
165     //            for (int j = 0; j < Config.ROWS; j++) {
166     //                p = puzzle[i][j].getSprite();
167     //                if (!(p.getFrame() == 0)) {
168     //                    return false;
169     //                }
170     //            }
171     //        }
172     //        return true;
173     //    }
174 
175         /**
176          * @return la cantidad de pixeles del rompecabezas en su parte horizontal.
177          */
178         public int getWidth() {
179             return Config.COLS * Config.DIM_PZ;
180         }
181 
182         /**
183          * @return la cantidad de pixeles del rompecabezas en su parte vertical.
184          */
185         public int getHeight() {
186             return Config.COLS * Config.DIM_PZ;
187         }
188 
189         /**
190          * Pinta el puzzle y su cursor.
191          *
192          * @param g
193          */
194         public void paint(Graphics g) {
195 
196             g.setColor(Config.COLOR_PUZZLE_FONDO);
197             g.fillRect(x - 4, y - 4, this.getWidth() + 8, this.getHeight() + 8);
198 
199             for (int i = 0; i < Config.COLS; i++) {
200                 for (int j = 0; j < Config.ROWS; j++) {
201                     puzzle[i][j].paint(g);
202                 }
203             }
204             cursor.paint(g);
205         }
206 
207         /**
208          * Método que permite mostrar la parte posterior de todas las piezas del rompecabezas.
209          */
210         public void ocultar() {
211             for (int i = 0; i < Config.COLS; i++) {
212                 for (int j = 0; j < Config.ROWS; j++) {
213                     puzzle[i][j].ocultar();
214                 }
```

```
215             }
216             cursor.getSprite().setVisible(true);
217         }
218 
219         /**
220          * Método que permite mostrar la parte frontal de todas las piezas del rompecabezas.
221          */
222         public void mostrar() {
223             cursor.getSprite().setVisible(false);
224             for (int i = 0; i < Config.COLS; i++) {
225                 for (int j = 0; j < Config.ROWS; j++) {
226                     puzzle[i][j].mostrar();
227                 }
228             }
229         }
230 
231         /**
232          * Método que permite mostrar la partefrontal de una pieza en especifico, pasando las
    coordenadas de la pieza dentro del puzzle o rompecabezas.
233          * @param i
234          * @param j
235          */
236         public void mostrarPieza(int i, int j) {
237             puzzle[i][j].mostrar();
238         }
239 
240         /**
241          * Método que permite indicarle al juego que se a avanzado en la completez del rompecabezas.
242          */
243         public void avanzar() {
244             piezasFaltantes--;
245         }
246 
247         /**
248          * Método que verifica si el rompecabezas esta o no completo.
249          * @return verdadero o falso dependiendo de sí el rompecabezas esta o no completo.
250          */
251         public boolean estaCompleto() {
252             return (piezasFaltantes == 0);
253         }
254     }
255 
```

Clases del paquete uam.pt.ortigosa.midlet

Clase MidletInitGame.java

```
1   package uam.pt.ortigosa.midlet;
2   
3   import javax.microedition.lcdui.Command;
4   import javax.microedition.lcdui.CommandListener;
5   import javax.microedition.lcdui.Display;
6   import javax.microedition.lcdui.Displayable;
7   import javax.microedition.midlet.*;
8   import uam.pt.ortigosa.bluetooth.BtCliente;
9   import uam.pt.ortigosa.bluetooth.BtServidor;
10  import uam.pt.ortigosa.colaborativo.ColaboradorRemoto;
11  import uam.pt.ortigosa.colaborativo.JugadorLocalServidor;
12  import uam.pt.ortigosa.colaborativo.JugadorServidor;
13  import uam.pt.ortigosa.colaborativo.TrabajoColaborativo;
14  import uam.pt.ortigosa.graphics.Game;
15  import uam.pt.ortigosa.vistas.*;
16  import uam.pt.ortigosa.util.Config;
17  import uam.pt.ortigosa.util.Jugador;
18  
19  /**
20   * Midlet principal que se encarga de toda la administración, inicialización y comunicación del
    juego.
21   *
```

```
22      * @author José Luis Ortigosa Flores
23      */
24     public class MidletInitGame extends MIDlet implements CommandListener {
25
26         private Display display;
27         private FormPresentacion fp;
28         private ListaClientOrServer lcos;
29         private FormEsperandoColaboradores fec;
30         private ListaNumColaboradores lnc;
31         private ListaDispositivos ld;
32         private FormBuscandoJuego fbj;
33         private ImagenObjetivo io;
34         private Game juego;
35         private Command exit;
36         private Command next;
37         private Command busDis;
38         private Command busSer;
39         private Command conect;
40         private Command back;
41         private BtServidor bs;
42         private BtCliente bc;
43         private TrabajoColaborativo tc;
44         private Jugador jugador;
45
46         /**
47          * Se encarga de inicializar todo el juego, incluyendo commands y pantallas.
48          */
49         public MidletInitGame() {
50             display = Display.getDisplay(this);
51
52             initCommands();
53             initScreens();
54         }
55
56         /**
57          *
58          */
59         public void startApp() {
60             display.setCurrent(fp);
61         }
62
63         /**
64          *
65          */
66         public void pauseApp() {
67             notifyPaused();
68         }
69
70         /**
71          * Para destruir la aplicacion
72          *
73          * @param unconditional
74          */
75         public void destroyApp(boolean unconditional) {
76             notifyDestroyed();
77         }
78
79         /**
80          * Método que inicia todo lo necesario para los graficos del juego.
81          */
82         private void initGame(boolean jugarSolo) {
83             if (!Config.isServer() && !jugarSolo) {
84                 jugador = new ColaboradorRemoto(bc.getConexion());
85                 jugador.start();
86             }
87
88             juego = new Game(false, jugador, jugarSolo);
89             io = new ImagenObjetivo(juego.getPuzzle());
90             new Thread(juego).start();
91
92             io.addCommand(next);
93             io.addCommand(exit);
94             io.setCommandListener(this);
95
96             juego.addCommand(back);
```

```
97  //        juego.addCommand(exit);
98              juego.setCommandListener(this);
99          }
100
101         private void initCommands() {
102             exit = new Command("Salir", Command.EXIT, 7);
103             next = new Command("Siguiente", Command.OK, 0);
104             busDis = new Command("B.Dispositivos", Command.SCREEN, 0);
105             busSer = new Command("B.Juego", Command.SCREEN, 0);
106             conect = new Command("Conectar", Command.OK, 0);
107             back = new Command("Regresar", Command.BACK, 6);
108         }
109
110         private void initScreens() {
111             fp = new FormPresentacion();
112             lcos = new ListaClientOrServer();
113             fec = new FormEsperandoColaboradores();
114             lnc = new ListaNumColaboradores();
115             ld = new ListaDispositivos();
116             fbj = new FormBuscandoJuego();
117
118             fp.addCommand(exit);
119             fp.addCommand(next);
120             fp.setCommandListener(this);
121
122             lcos.addCommand(exit);
123             lcos.setSelectCommand(next);
124             lcos.setCommandListener(this);
125
126             fec.addCommand(exit);
127             fec.addCommand(next);
128             fec.setCommandListener(this);
129
130             lnc.addCommand(exit);
131             lnc.setSelectCommand(next);
132             lnc.setCommandListener(this);
133
134             ld.addCommand(exit);
135             ld.addCommand(busDis);
136             ld.setSelectCommand(busSer);
137             ld.setCommandListener(this);
138
139             fbj.addCommand(conect);
140             fbj.addCommand(exit);
141             fbj.addCommand(back);
142             fbj.setCommandListener(this);
143         }
144
145         private void initBluetoothCliente() {
146             bc = new BtCliente(ld, fbj);
147         }
148
149         private void initBluetoothServer() {
150             tc = new TrabajoColaborativo();
151             bs = new BtServidor(tc, fec);
152             jugador = new JugadorLocalServidor();
153             tc.agregarJugador((JugadorServidor) jugador);
154         }
155
156         /**
157          * En este método se encuentra toda la lógica necesaria para la navegación entre pantallas.
158          * @param c comando de la pantalla d.
159          * @param d pantalla actual.
160          */
161         public void commandAction(Command c, Displayable d) {
162             if (c == exit) {
163                 detener();
164                 destroyApp(true);
165             } else if (c == next) {
166                 if (d.equals(fp)) {
167                     display.setCurrent(lcos);
168                 } else if (d.equals(lcos)) {
169                     switch (lcos.getSeleccion()) {
170                         case Config.PARTICIPANTE_AND_INICIADOR: {
171                             Config.setServer(true);
```

```
172                         initBluetoothServer();
173                         display.setCurrent(lnc);
174                     }
175                     break;
176                     case Config.ONLY_PARTICIPANTE: {
177                         Config.setServer(false);
178                         initBluetoothCliente();
179                         display.setCurrent(ld);
180                     }
181                     break;
182                     default: {
183                         initGame(true);
184                         display.setCurrent(io);
185                     }
186                     break;
187                 }
188             } else if (d.equals(lnc)) {
189                 Config.setNumJugadores(lnc.getNumColaboradores() + 1);
190                 bs.start();
191                 display.setCurrent(fec);
192             } else if (d.equals(io)) {
193                 display.setCurrent(juego);
194             } else if (d.equals(fec)) {
195                 if (tc.conexionesListas()) {
196                     initGame(false);
197                     display.setCurrent(io);
198                     eliminarScreensAnteriores();
199                 }
200             }
201         } else if (c == busDis) { // para el cliente
202             bc.buscarDispositivos();
203         } else if (c == busSer) {
204             bc.buscarServicios();
205             display.setCurrent(fbj);
206         } else if (c == conect) {
207             bc.conectar();
208             initGame(false);
209             display.setCurrent(io);
210             eliminarScreensAnteriores();
211         } else if (c == back) {
212             if (d.equals(fbj)) {
213                 display.setCurrent(ld);
214             } else if (d.equals(juego)) {
215                 display.setCurrent(io);
216             }
217         }
218     }
219 
220     private void eliminarScreensAnteriores() {
221         fp = null;
222         lcos = null;
223         fec = null;
224         lnc = null;
225         ld = null;
226         fbj = null;
227         System.gc();
228     }
229 
230     private void detener() {
231         if (tc != null) {
232             tc.detener();
233         }
234         if (bc != null) {
235             bc.getConexion().finalizarConexion();
236         }
237         jugador.detener();
238         System.gc();
239     }
240 }
241 
```

Clases del paquete uam.pt.ortigosa.util

Clase Cola.java

```
1   package uam.pt.ortigosa.util;
2
3   /**
4    * Clase que permite modelar a una cola o una lista de tipo FIFO,
5    * que esta adaptada para el acceso simultanea por varios hilos a la infomración contenida en la
    misma
6    *
7    * @author José Luis
8    */
9   public class Cola extends Lista {
10
11      /**
12       * Permite crear un objeto de tipo cola. que soporta acceso simultáneo
13       */
14      public Cola() {
15          super();
16      }
17
18      /**
19       * Método que permite obtener el primer objeto encolado.
20       *
21       * @return el primero objeto almacenado en la cola.
22       */
23      public synchronized Object removeFirst() {
24          while (this.isEmpty()) {
25              try {
26                  wait();
27              } catch (InterruptedException ex) {
28                  System.err.println("Error: " + ex.getMessage());
29              }
30          }
31          return super.removeFirst();
32      }
33
34      /**
35       * Método que permite agregar elmentos en la cola, cada objeto se agrega en la última
    posicion
36       *
37       * @param obj se agregará al final de la cola.
38       */
39      public synchronized void agregar(Object obj) {
40          super.agregar(obj);
41          notify();
42      }
43
44      /**
45       * Método que permite verificar si la cola esta o no vacia.
46       *
47       * @return verdadero o falso dependiendo de si esta o no vacia.
48       */
49      public synchronized boolean isEmpty() {
50          return super.isEmpty();
51      }
52
53      /**
54       * Método que permite obtener la cantidad de elementos en la cola.
55       *
56       * @return la cantidad de elementos en la cola.
57       */
58      public synchronized int getSize() {
59          return super.getSize();
60      }
61  }
62
```

Clase Config.java

```
1   package uam.pt.ortigosa.util;
2
3   import javax.bluetooth.UUID;
4
5   /**
6    * Clase que contiene todas las variables y constantes de la aplicación, incluidas:
7    * colores, medidas, etc.
8    *
9    * @author José Luis Ortigosa Flores
10   */
11  public class Config {
12
13      private static String stringOrtigosaServiceUUID = "F0E0D0C0B0A000908070605040302010";
14      /**
15       * Es el UUID de la publicación del servicio.
16       */
17      public static final UUID ORTIGOSA_SERVICE_UUID = new UUID(stringOrtigosaServiceUUID, false);
18      private static int stringBtspp = 0x1101;
19      /**
20       * Es el UUID del BTSPP
21       */
22      public static final UUID BTSPP_UUID = new UUID(stringBtspp);
23      private static int numJugadores = 2;
24      private static boolean server = false;
25      private static String nameDevice = "noName";
26      private static boolean errorCom = false;
27      /**
28       * Constante que indica si se trata de un servidor.
29       */
30      public static final int PARTICIPANTE_AND_INICIADOR = 0;
31      /**
32       * Constante que indica si se trata de un cliente.
33       */
34      public static final int ONLY_PARTICIPANTE = 1;
35      /**
36       * Constante que indica la cantidad de pixeles horizontales y verticales de las piezas.
37       */
38      public static final int DIM_PZ = 25;
39      /**
40       * Constante que contiene la cantidad de columnas del rompecabezas.
41       */
42      public static final int COLS = 4;
43      /**
44       * Constante que contiene la cantidad de filas del rompecabezas.
45       */
46      public static final int ROWS = 4;
47      /**
48       * Constante que contiene el directorio con las imagenes.
49       */
50      public static final String PATH_IMAGE = "/images";
51      /**
52       * Constante con un valor hexadecimal para representar el color del header.
53       */
54      public static final int COLOR_HEADER_FONDO_1 = 0x080b7e;
55      /**
56       * Constante con un valor hexadecimal para representar el segundo color del header.
57       */
58      public static final int COLOR_HEADER_FONDO_2 = 0x294eac;
59      /**
60       * Constante con un valor hexadecimal para representar el color de la fuente.
61       */
62      public static final int COLOR_HEADER_FONT_1 = 0xffffff;
63      /**
64       * Constante con un valor hexadecimal para representar el color de la segunda fuente.
65       */
66      public static final int COLOR_HEADER_FONT_2 = 0xffffff;
67      /**
68       * Constante con un valor hexadecimal para representar el color del borde de footer.
69       */
70      public static final int COLOR_FOOTER_BORDER = 0xc314e8;
71      /**
72       * Constante con un valor hexadecimal para representar el color de la fuente del footer.
```

```
73          */
74         public static final int COLOR_FOOTER_FONT = 0xff2401;
75         /**
76          * Constante con un valor hexadecimal para representar el color del fondo del rompecabezas.
77          */
78         public static final int COLOR_BARRA_FONDO = 0x999999;
79         /**
80          * Constante con un valor hexadecimal para representar el color del fondo de la barra de
    piezas.
81          */
82         public static final int COLOR_PUZZLE_FONDO = 0x999999;
83
84         /**
85          * Método que permite obtener la cantidad de colaboradores que están participando en el
    juego.
86          * @return el número total de jugadores que estan jugando.
87          */
88         public static int getNumJugadores() {
89             return numJugadores;
90         }
91
92         /**
93          * Método que permite establecer la cantidad de colaboradores que estarán participando en el
    juego.
94          * @param numJugadores
95          */
96         public static void setNumJugadores(int numJugadores) {
97             Config.numJugadores = numJugadores;
98         }
99
100        /**
101         * Método que permite ver si un dipositivo es servidor o cliente.
102         *
103         * @return verdadero o falso según sea trate de un servidor o de un cliente.
104         */
105        public static boolean isServer() {
106            return server;
107        }
108
109        /**
110         * Método que permite establecer si un dispositivo es servidor o cliente, por default se
    supone es cliente.
111         *
112         * @param server es un variable boolena para indicar si se trata de un cliente o un
    servidor.
113         */
114        public static void setServer(boolean server) {
115            Config.server = server;
116        }
117
118        /**
119         * Método encargado de retornar el nombre del dispositivo local.
120         *
121         * @return una cadena con el nombre del dispositivo local
122         */
123        public static String getNameDevice() {
124            return Config.nameDevice;
125        }
126
127        /**
128         * Método que permite establecer el nombre del dispositivo local
129         *
130         * @param nameDevice nombre del dispositivo local.
131         */
132        public static void setNameDevice(String nameDevice) {
133            Config.nameDevice = nameDevice;
134        }
135
136        public static boolean hayErrorCom() {
137            return Config.errorCom;
138        }
139
140        public static void error() {
141            Config.errorCom = true;
142        }
```

```
143     }
144
```

Clase Jugador.java

```
1     package uam.pt.ortigosa.util;
2
3     import javax.bluetooth.BluetoothStateException;
4     import javax.bluetooth.LocalDevice;
5     import uam.pt.ortigosa.colaborativo.Colaborador;
6
7     /**
8      * Clase abstracta que se encarga de modelar a un jugador tanto remoto como local.
9      *
10     * Por si sola no es útil para esta aplicación por lo que es abstracta.
11     *
12     * @author José Luis Ortigosa Flores
13     */
14    public abstract class Jugador extends Thread implements Colaborador {
15
16        private int id;
17        private String nombre;
18
19        /**
20         * Constructor que permite crear a un jugador estandard
21         */
22        public Jugador() {
23            id = 0;
24            nombre = "sinNombre";
25            try {
26                LocalDevice local = LocalDevice.getLocalDevice();
27                if (!local.getFriendlyName().equals("")) {
28                    nombre = local.getFriendlyName();
29                }
30            } catch (BluetoothStateException ex) {
31                System.err.println("Error: " + ex.getMessage());
32            }
33        }
34
35        /**
36         * Método que permite establer un nuevo Id al jugador.
37         *
38         * @param id que se establecera a un jugador.
39         */
40        public void setId(int id) {
41            this.id = id;
42        }
43
44        /**
45         * Método que permite obtener el Id del jugador.
46         *
47         * @return el Id del jugador.
48         */
49        public int getId() {
50            return id;
51        }
52
53        /**
54         * Método que permite establecer el nombre de un jugador.
55         *
56         * @param nombre que se establecera al jugador.
57         */
58        public void setNombre(String nombre) {
59            this.nombre = nombre;
60        }
61
62        /**
63         * Método que permite obtener el nombre de un jugador.
64         *
65         * @return el nombre del jugador que invoque al método.
66         */
```

```
67          public String getNombre() {
68              return this.nombre;
69          }
70
71          public abstract void enviarMovimientoAlServidor(String movimiento);
72
73          public abstract String recibirMovimientoDelServidor();
74
75          public void detener() {
76          }
77      }
78
```

Clase Lista.java

```
1   package uam.pt.ortigosa.util;
2
3   /**
4    * Clase que modela a una lista muy simple con los métodos más comunes.
5    * Esta lista acepta cualquier tipo de obejeto que herede de Object.
6    *
7    * @author José Luis Ortigosa Flores
8    */
9   public class Lista {
10
11      private Nodo cabeza;
12      private int size;
13
14      /**
15       * Permite crear un Objeto de tipo lista.
16       */
17      public Lista() {
18          cabeza = null;
19          size = 0;
20      }
21
22      /**
23       * Método que permite ver si la lista esta o no vacia.
24       *
25       * @return verdadero o falso dependiendo de si esta o no vacia.
26       */
27      public boolean isEmpty() {
28          return ((cabeza == null) || (size == 0));
29      }
30
31      /**
32       * Método que permite agregar un objeto a la lista.
33       *
34       * @param obj que será agregado a la lista.
35       */
36      public void agregar(Object obj) {
37          if (isEmpty()) {
38              cabeza = new Nodo();
39              cabeza.setObject(obj);
40          } else {
41              Nodo aux = cabeza;
42              while (aux.getSig() != null) {
43                  aux = aux.getSig();
44              }
45              Nodo nuevo = new Nodo();
46              nuevo.setObject(obj);
47              aux.setSig(nuevo);
48          }
49          size++;
50      }
51
52      /**
53       * Método que permite obtener el elemento que se encuentre en la primera posición de la
    lista,
54       * a la vés que lo remueve de la misma.
55       *
```

```
56          * @return El primer objeto en la lista removiendolo de la misma.
57          */
58         public Object removeFirst() {
59             Nodo aux = cabeza;
60             if (aux != null) {
61                 cabeza = aux.getSig();
62                 size--;
63                 return aux.getObject();
64             }
65             return null;
66         }
67 
68         /**
69          * Método que permite obtener un objeto de la lista proporcionando un índice de la posicioón
    en la que esta
70          * el objeto requerido
71          *
72          * @param index con la posición del objeto requerido.
73          * @return el objeto que esta en la posición index.
74          */
75         public Object get(int index) {
76             Nodo aux = cabeza;
77             int i = 0;
78             if (index >= 0 && !isEmpty()) {
79                 while (i < getSize()) {
80                     if (i == index) {
81                         return aux.getObject();
82                     }
83                     aux = aux.getSig();
84                     i++;
85                 }
86             }
87             return null;
88         }
89 
90         /**
91          *Método que permite obtener la cantidad de elementos en la lista.
92          *
93          * @return La cantidad de elementos en la lista.
94          */
95         public int getSize() {
96             return this.size;
97         }
98 
99 
100        private class Nodo {
101 
102            private Object obj;
103            private Nodo sig;
104 
105            Nodo() {
106                obj = null;
107                this.sig = null;
108            }
109 
110            public synchronized Object getObject() {
111                return this.obj;
112            }
113 
114            public synchronized void setObject(Object obj) {
115                this.obj = obj;
116            }
117 
118            public Nodo getSig() {
119                return this.sig;
120            }
121 
122            public void setSig(Nodo sig) {
123                this.sig = sig;
124            }
125        }
126    }
127 
```

Clase StringTokenizer.java

```
1   package uam.pt.ortigosa.util;
2
3   /**
4    * Clase que permite obtener el comportamiento de la clase original provista por Java, pero que
    no esta disponible
5    * en su edición micro.
6    *
7    * Esta clase solo esta diseñanda para buscar toquens delimitados por espacios. En algún momento
    dado podria toquenizar bajo
8    * otro delimitador diferente.
9    *
10   * @author José Luis Ortigosa Flores
11   */
12  public class StringTokenizer {
13
14      private String string;
15      private String delim;
16
17      /**
18       * Permite crear un objeto tokenizador.
19       *
20       * @param string es la cadena que será dividida en tokens.
21       */
22      public StringTokenizer(String string) {
23          this.string = string;
24          delim = " ";
25          this.string += delim;
26
27      }
28
29      /**
30       * Permite crear un objeto tokenizador.
31       *
32       * @param string es la cadena que sera dividida en tokens.
33       * @param delimitador es el string que se requiere para tokenizar.
34       */
35      public StringTokenizer(String string, String delimitador) {
36          this.string = string;
37          this.delim = delimitador;
38          this.string += delim;
39
40      }
41
42      /**
43       * Método que permite obtener los tokens en la manera en la que van apareciendo de izquierda
    a derecha.
44       *
45       * @return un string con el token encontrado.
46       */
47      public String nextToken() {
48          String token = null;
49          if (string.length() > 0) {
50              int endIndex = string.indexOf(delim);
51              token = string.substring(0, endIndex);
52              string = string.substring(endIndex + 1, string.length());
53          }
54          return token;
55      }
56
57      /**
58       * Método que permite obtener la cantidad de tokens en el string dado al momento de su
    llamada.
59       * @return la cantidad de tokens en este objeto.
60       */
61      public int countTokens() {
62          String copy = new String();
63          System.arraycopy(string, 0, copy, 0, string.length());
64          int count = 0;
65          while (copy.length() > 0) {
66              int endIndex = copy.indexOf(delim);
67              copy = copy.substring(endIndex + 1, copy.length());
68              count++;
```

```
69            }
70            return count;
71        }
72
73        /**
74         * Método que retorna verdadero o falso dependiendo de si hay o no tokens disponibles al
      momento de su llamada.
75         *
76         * @return verdadero o falso dependiendo de si hay o no más tokens.
77         */
78        public boolean hasMoreTokens() {
79            return (string.length() > 0);
80        }
81    }
82
```

Clases del paquete uam.pt.ortigosa.util.xml

Clase GenerarMensaje.java

```
1     package uam.pt.ortigosa.util.xml;
2
3     import uam.pt.ortigosa.graphics.Pieza;
4     import uam.pt.ortigosa.util.Jugador;
5     import uam.pt.ortigosa.util.StringTokenizer;
6
7     /**
8      * Clase que permite generar el XML adecuado para el intercambio de datos o infomrmación
      importante para la aplicación
9      * de juego de rompecabezas.
10     * Esta clase se encarga tanto de generar cadenas de caracteres con el xml adecuado, o de
      extraer la información contenida en
11     * las cadenas con xml.
12     *
13     * @author José Luis Ortigosa Flores
14     */
15    public class GenerarMensaje {
16
17        private static final String delim = ",";
18
19        /**
20         * Método que permite generar el xml pertinente adecuado a la construcción del objeto.
21         *
22         * @param pieza a la cual se le debe de hacer la acción señalada por las variables booleanas
23         * @param jugador que es el responsable de mover o modificar la pieza indicada.
24         * @return un string con la información necesaria que represente a un movimiento de un
      jugador.
25         */
26        public static String getMensaje(Jugador jugador, Pieza pieza) {
27            String cad = new String();
28
29            if (jugador != null) {
30                cad += jugador.getId() + delim;
31                if(jugador.getNombre().equals(""))
32                    cad += "noName" + delim;
33                else
34                    cad += jugador.getNombre() + delim;
35            } else {
36                cad += "0" + delim;
37                cad += "noName" + delim;
38            }
39            cad += pieza.getId() + delim;
40            cad += pieza.getI() + delim;
41            cad += pieza.getJ();
42            System.out.println(cad);
43            return cad;
44        }
45
46        /**
```

```
47          * Este método estático permite convertir una cadena de caracteres que contiene xml adecuado
    a un objeto de tipo mensaje
48          * @param mensaje al cual se extraera infomración.
49          * @return un objeto de tipo Mensaje para manejar la información contenida en el string
    mensaje
50          */
51         public static Mensaje getMensaje(String mensaje) {
52             Mensaje men = new Mensaje();
53             StringTokenizer token = new StringTokenizer(mensaje, delim);
54 
55             men.setJugadorId(Integer.parseInt(token.nextToken()));
56             men.setJugadorNombre(token.nextToken());
57             men.setPiezaId(Integer.parseInt(token.nextToken()));
58             int i = Integer.parseInt(token.nextToken());
59             int j = Integer.parseInt(token.nextToken());
60             men.SetCoordenadas(i, j);
61             return men;
62         }
63     }
64 
```

Clase GenerarXML.java

```
1     package uam.pt.ortigosa.util.xml;
2 
3     import java.io.ByteArrayInputStream;
4     import javax.xml.parsers.DocumentBuilder;
5     import javax.xml.parsers.DocumentBuilderFactory;
6     import org.w3c.dom.DOMImplementation;
7     import org.w3c.dom.Document;
8     import org.w3c.dom.Element;
9     import org.w3c.dom.NamedNodeMap;
10    import org.w3c.dom.Node;
11    import org.w3c.dom.NodeList;
12    import uam.pt.ortigosa.graphics.Pieza;
13    import uam.pt.ortigosa.util.Jugador;
14 
15    /**
16     * Clase que permite generar el XML adecuado para el intercambio de datos o infomrmación
    importante para la aplicación
17     * de juego de rompecabezas.
18     * Esta clase se encarga tanto de generar cadenas de caracteres con el xml adecuado, o de
    extraer la información contenida en
19     * las cadenas con xml.
20     *
21     * @author José Luis Ortigosa Flores
22     */
23    public class GenerarXML {
24 
25 
26        private static final String TAG_MOVIMIENTO = "movimiento";
27        private static final String TAG_JUGADOR = "jugador";
28        private static final String TAG_PIEZA = "pieza";
29        private static final String TAG_COORDENADAS = "coordenadas";
30        private static final String ATTR_ID = "id";
31        private static final String ATTR_NOMBRE = "nombre";
32        private static final String ATTR_I = "i";
33        private static final String ATTR_J = "j";
34 
35        /**
36         * Método que permite generar el xml pertinente adecuado a la construcción del objeto.
37         *
38         * @param pieza a la cual se le debe de hacer la acción señalada por las variables booleanas
39         * @param jugador que es el responsable de mover o modificar la pieza indicada.
40         * @return Una cadena de caracteres con xml con la información referente a la pieza y al
    jugador.
41         */
42        public static String getXML(Jugador jugador, Pieza pieza) {
43            Document documento;
44            try {
45                // Obtengo un constructor de documentos XML
46                DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
```

```
47              DocumentBuilder builder = factory.newDocumentBuilder();
48
49              // Creamos un documento nuevo
50              DOMImplementation impl = builder.getDOMImplementation();
51              documento = impl.createDocument(null, TAG_MOVIMIENTO, null);
52
53              Element eJugador = documento.createElement(TAG_JUGADOR);
54              eJugador.setAttribute(ATTR_ID, "" + jugador.getId());
55              eJugador.setAttribute(ATTR_NOMBRE, jugador.getNombre());
56
57              documento.getDocumentElement().appendChild(eJugador);
58
59              Element ePieza = documento.createElement(TAG_PIEZA);
60              ePieza.setAttribute(ATTR_ID, "" + pieza.getId());
61
62              Element eCoordenadas = documento.createElement(TAG_COORDENADAS);
63              eCoordenadas.setAttribute(ATTR_I, "" + pieza.getI());
64              eCoordenadas.setAttribute(ATTR_J, "" + pieza.getJ());
65
66              ePieza.appendChild(eCoordenadas);
67
68              documento.getDocumentElement().appendChild(ePieza);
69
70              return documento.toString();
71          } catch (Exception e) {
72              System.err.println("Error creando XML: " + e.getMessage());
73              return null;
74          }
75      }
76
77      /**
78       * Este método estático permite convertir una cadena de caracteres que contiene xml adecuado
    a un objeto de tipo mensaje
79       * @param xml es la cadena con el xml que se desea extraer información
80       * @return Un objeto de tipo Mensaje el cual permite acceder a la infomación que esta en
    xml.
81       */
82      public static Mensaje getMensajeXML(String xml) {
83          Mensaje mensaje = new Mensaje();
84          try {
85              ByteArrayInputStream in = new ByteArrayInputStream(xml.getBytes());
86              // obtain a document builder
87              DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
88              DocumentBuilder builder = factory.newDocumentBuilder();
89
90              // Creo un documento para parsear la cadena
91              Document document = builder.parse(in);
92              // Obtengo todos los movimientos (solo hay uno)
93              NodeList list = document.getElementsByTagName(TAG_MOVIMIENTO);
94
95              Node movimiento = list.item(0);
96
97              NodeList hijosMov = movimiento.getChildNodes();
98
99              Node nJugador = hijosMov.item(0);
100             NamedNodeMap attrs = nJugador.getAttributes();
101             mensaje.setJugadorId(Integer.parseInt(attrs.getNamedItem(ATTR_ID).getNodeValue()));
102             mensaje.setJugadorNombre(attrs.getNamedItem(ATTR_NOMBRE).getNodeValue());
103
104             Node nPieza = hijosMov.item(1);
105             attrs = nPieza.getAttributes();
106             mensaje.setPiezaId(Integer.parseInt(attrs.getNamedItem(ATTR_ID).getNodeValue()));
107
108             NodeList hijosPz = nPieza.getChildNodes();
109             Node coordenadas = hijosPz.item(0);
110             attrs = coordenadas.getAttributes();
111             int i = Integer.parseInt(attrs.getNamedItem(ATTR_I).getNodeValue());
112             int j = Integer.parseInt(attrs.getNamedItem(ATTR_J).getNodeValue());
113             mensaje.SetCoordenadas(i, j);
114
115         } catch (Exception e) {
116             System.err.println("Error obteniendo mensaje: " + e.getMessage());
117         }
118         return mensaje;
119     }
```

```
120
121         /**
122          * Método que permite convertir un string con las letras “false” o “true” en una variable
    booleana
123          * @param stringBool debe ser "true" o "false"
124          * @return un primitivo true o false
125          */
126         public static boolean stringToBoolean(String stringBool) {
127             String cad = stringBool.toLowerCase();
128             if (cad.equals("true")) {
129                 return true;
130             }
131             return false;
132         }
133     }
134
```

Clase Mensaje.java

```
1   package uam.pt.ortigosa.util.xml;
2
3   /**
4    * Clase que se encarga de modelar los mensajes en XML que estarán viajando entre distintos
    dispositivos.
5    *
6    * @author José Luis Ortigosa Flores
7    */
8   public class Mensaje {
9
10      private int jugadorId;
11      private String jugadorNombre;
12      private int piezaId;
13      private int piezaCoordenadaI;
14      private int piezaCoordenadaJ;
15
16      /**
17       * Objeto que puede contener la infomración que esta en un string de mensaje
18       */
19      public Mensaje() {
20      }
21
22      /**
23       * @param id del jugador que esta enviando el mensaje.
24       */
25      public void setJugadorId(int id) {
26          this.jugadorId = id;
27      }
28
29      /**
30       * @return el id del jugador que esta enviando un mensaje.
31       */
32      public int getJugadorId() {
33          return this.jugadorId;
34      }
35
36      /**
37       * @param nombre del jugador que esta realizando el movimiento.
38       */
39      public void setJugadorNombre(String nombre) {
40          this.jugadorNombre = nombre;
41      }
42
43      /**
44       * @return el nombre del jugador que esta haciendo el movimiento.
45       */
46      public String getJugadorNombre() {
47          return this.jugadorNombre;
48      }
49
50      /**
51       * @param id de la pieza que será afectada.
52       */
```

```
53          public void setPiezaId(int id) {
54              this.piezaId = id;
55          }
56
57          /**
58           * @return el id de la pieza que será modificada.
59           */
60          public int getPiezaId() {
61              return this.piezaId;
62          }
63
64          /**
65           * @return la posición i de la pieza dentro del puzzle o rompecabezas.
66           */
67          public int getPiezaCoordenadaI() {
68              return this.piezaCoordenadaI;
69          }
70
71          /**
72           * Obtiene la coordenada J de la pieza almacenada en este mensaje.
73           *
74           * @return la posición j de la pieza dentro del puzzle o rompecabezas.
75           */
76          public int getPiezaCoordenadaJ() {
77              return this.piezaCoordenadaJ;
78          }
79
80          /**
81           * Método que permite almacenar las coordenadas de una pieza en este mensaje
82           *
83           * @param i posicion i
84           * @param j posicion j
85           */
86          public void SetCoordenadas(int i, int j) {
87              this.piezaCoordenadaI = i;
88              this.piezaCoordenadaJ = j;
89          }
90      }
91
```

Clases del paquete uam.pt.ortigosa.vistas

Clase FormBuscandoJuego.java

```
1   package uam.pt.ortigosa.vistas;
2
3   import javax.microedition.lcdui.Form;
4
5   /**
6    * Pantalla que solo se le muestra al cliente, que permite mostrar mensajes con respecto
7    * a la busqueda de servicios en el dispositivos señalado.
8    *
9    * @author José Luis Ortigosa Flores
10   */
11  public class FormBuscandoJuego extends Form {
12
13      /**
14       * Pantalla que le permite al usuario saber si el juego se localizó en un dispositivo dado.
15       */
16      public FormBuscandoJuego() {
17          super("Buscando juego...");
18      }
19  }
20
```

Clase FormEsperandoColaboradores.java

```
1   package uam.pt.ortigosa.vistas;
2
3   import javax.microedition.lcdui.Form;
4
5   /**
6    * Clase para mostrar al usuario una pantalla que le indique al dispositivo
7    * que este como servidor, una animación de espera.
8    *
9    * @author José Luis Ortigosa Flores
10   */
11  public class FormEsperandoColaboradores extends Form {
12
13      /**
14       * Crea una pantalla para mostrar la lista de colaboradores que ya estan conectados.
15       * y no permite avanzar sino hasta que esten todos los colaboradores conectados.
16       */
17      public FormEsperandoColaboradores() {
18          super("Esperando colaboradores");
19      }
20  }
21
```

Clase FormPresentacion.java

```
1   package uam.pt.ortigosa.vistas;
2
3   import java.io.IOException;
4   import javax.microedition.lcdui.Form;
5   import javax.microedition.lcdui.Image;
6   import javax.microedition.lcdui.ImageItem;
7   import javax.microedition.lcdui.Item;
8   import uam.pt.ortigosa.util.Config;
9
10  /**
11   * Pantalla que permite mostrar al usuario una pantalla de presentación del juego.
12   *
13   * @author José Luis Ortigosa Flores
14   */
15  public class FormPresentacion extends Form {
16
17      /**
18       * Constructor que permite crear una pantalla de presentación.
19       */
20      public FormPresentacion() {
21          super("Presentacion");
22          Image logo = null;
23          try {
24              logo = Image.createImage(Config.PATH_IMAGE + "/rompecabezas.png");
25          } catch (IOException ex) {
26              System.out.println("Error: " + ex.getMessage());
27          }
28          append(new ImageItem(null, logo, Item.LAYOUT_CENTER, null));
29          append(new ImageItem("Universidad Autónoma", null, Item.LAYOUT_CENTER, null,
    Item.PLAIN));
30          append(new ImageItem("Metropolitana", null, Item.LAYOUT_CENTER, null, Item.PLAIN));
31          append(new ImageItem("Unidad Azcapotzalco", null, Item.LAYOUT_CENTER, null));
32          append(new ImageItem("Proyecto Terminal", null, Item.LAYOUT_CENTER, null));
33          append(new ImageItem("Ingenieria en Computación", null, Item.LAYOUT_CENTER, null));
34          append(new ImageItem("Ortigosa Flores José Luis", null, Item.LAYOUT_CENTER, null));
35      }
36  }
37
```

Clase ImagenObjetivo.java

```
1   package uam.pt.ortigosa.vistas;
2
3   import javax.microedition.lcdui.Graphics;
4   import javax.microedition.lcdui.game.GameCanvas;
5   import uam.pt.ortigosa.graphics.Encabezado;
6   import uam.pt.ortigosa.graphics.Puzzle;
7
8   /**
9    * Pantalla que le muestra al usuario la imagen que deberá de armarse en el
10   * rompecabezas.
11   *
12   * @author José Luis Ortigosa Flores
13   */
14  public class ImagenObjetivo extends GameCanvas {
15
16      private Encabezado header;
17      private Puzzle puzzle;
18
19      /**
20       * Para crear esta pantalla se necesita de un objeto puzzle ya iniciado.
21       *
22       * @param puzzle al cual se le debe de extraer la imagen objetivo.
23       */
24      public ImagenObjetivo(Puzzle puzzle) {
25          super(false);
26          header = new Encabezado(getWidth(), 40);
27          this.puzzle = puzzle;
28
29          Graphics g = getGraphics();
30          header.paint(g);
31          this.puzzle.mostrar();
32          this.puzzle.paint(g);
33          this.puzzle.ocultar();
34      }
35  }
36
```

Clase ListaClientOrServer.java

```
1   package uam.pt.ortigosa.vistas;
2
3   import javax.microedition.lcdui.List;
4
5   /**
6    * Pantalla que le permite al usuario seleccionar si va ser cliente o servidor.
7    *
8    * @author José Luis Ortigosa Flores
9    */
10  public class ListaClientOrServer extends List {
11
12      /**
13       * Permite crear una pantalla para que el usuario pueda decidir entre crear un nuevo juego
    colaborativo y
14       * o participar como colaborador en uno ya existente. Además de mostrarle la opción de jugar
    solo.
15       */
16      public ListaClientOrServer() {
17          super("Decidir", List.EXCLUSIVE | List.IMPLICIT);
18          append("Iniciar nuevo juego [como Servidor]", null);
19          append("Participar en un juego [como Cliente]", null);
20          append("Jugar solo", null);
21      }
22
23      /**
24       * Método que permite obtener la opción seleccionada por el usuario.
25       *
26       * @return el id de la opción seleccionada.
27       */
28      public int getSeleccion() {
```

```
29              return super.getSelectedIndex();
30          }
31      }
32
```

Clase ListaDispositivos.java

```
1   package uam.pt.ortigosa.vistas;
2
3   import javax.microedition.lcdui.List;
4
5   /**
6    * Pantalla que es mostrada a los cliente para enlistar los dispositivos que se
7    * encuentren en el alcance del dispositivo servidor.
8    *
9    * @author José Luis Ortigosa Flores
10   */
11  public class ListaDispositivos extends List {
12
13      /**
14       * Pantalla que permite enlistar a los dispositivos localizados en las proximidades.
15       */
16      public ListaDispositivos() {
17          super("Dispositivos localizados", List.EXCLUSIVE | List.IMPLICIT);
18      }
19  }
20
```

Clase ListaNumColaboradores.java

```
1   package uam.pt.ortigosa.vistas;
2
3   import javax.microedition.lcdui.List;
4
5   /**
6    * Pantalla que se muestra a los usuarios que estan actuando como servidor y que le
7    * permite decidir cuantos jugadores van a colaborar.
8    *
9    * @author José Luis Ortigosa Flores
10   */
11  public class ListaNumColaboradores extends List {
12
13      /**
14       * Constructor que crea una pantalla que le permite al usuario seleccionar la cantidad
15       * de colaboradores que participarán el juego colaborativo.
16       */
17      public ListaNumColaboradores() {
18          super("Número de colaboradores:", List.IMPLICIT | List.EXCLUSIVE);
19          append("1", null);
20          append("2", null);
21          append("3", null);
22      }
23
24      /**
25       * Método que permite obtener la elección del usuario.
26       *
27       * @return la cantidad de colaboradores que seleccionó el usuario.
28       */
29      public int getNumColaboradores() {
30          int nc = Integer.parseInt(getString(getSelectedIndex()));
31          return nc;
32      }
33  }
34
```