CARACTERIZACIÓN DE JUEGOS BASADOS EN AGENTES MÓVILES. DISEÑO E IMPLEMENTACIÓN DE UN CASO PRÁCTICO Y SU INCORPORACIÓN EN EL PROGRAMA DOCENTE DE UNA ASIGNATURA DE REDES DE COMPUTADORES.

Memoria del proyecto de final de carrera correspondiente a los estudios de Ingeniería Superior en Informática presentado por Francisco J. Requena Moreno y dirigido por Mª Carmen de Toro Valdivia.

Bellaterra, Junio de 2007

El firmante, Mª Carmen de Toro , profesora del Departamento de Ingeniería de la Información y de las Comunicaciones de la Universidad Autónoma de Barcelona

CERTIFICA:

Que la presente memoria ha sido realizada bajo su dirección por Francisco J. Requena Moreno

Bellaterra, Junio de 2007

Firmado: Mª Carmen de Toro

A mi familia y a mis amigos.

Agradecimientos

Gracias a mi directora de proyecto, Mª Carmen de Toro, por la paciencia que ha mostrado tener conmigo y ayudarme a resolver aquellas dudas que han ido surgiendo durante la realización del proyecto.

Muchas gracias a todos los miembros del grupo de investigación SeNDA y a los compañeros del laboratorio, Jaume, Abraham, Víctor y Carlos, por la ayuda ofrecida.

Y finalmente gracias a mi familia y amigos por estar a mi lado durante los años que ha durado la carrera.

Índice general

Índice de figuras

Capítulo 1

Introducción

Los paradigmas de programación de juegos van cambiando continuamente y actualmente se está apostando por la que se basa en agentes móviles.

Un agente móvil es una entidad autónoma que reacciona a los estímulos externos y tiene la capacidad de moverse entre diferentes dispositivos conectados entre ellos. Para la utilización de agentes es necesario una plataforma donde se puedan crear, ejecutar, clonar y morir. Existen diferentes plataformas que se distribuyen bajo la licencia de software libre son: Grasshopper, ZEUS, Tracy y JADE.

La plataforma que actualmente está trabajando de manera más rigurosa en su desarrollo, incorporando mejoras de seguridad, movilidad y diferentes servicios es JADE. Esta plataforma cumple totalmente con las especificación de arquitectura de plataformas descritas por la Foundation for Intelligent Physical Agents (FIPA), organización dedicada a la producción de estándares para la interoperabilidad de los agentes.

La apuesta por la programación de ciertos tipos de videojuegos utilizando agentes se debe a las ventajas que estos proporcionan. La principal de ellas es la eficiencia gracias a que el código es lo único que se mueve, de esta manera, el consumo de recursos en la red es inferior. La autonomía que poseen los agentes hacen que no sea necesario una continua conexión entre las máquinas, esto solo es necesario en el momento en que el agente quiere moverse entre las máquinas. De manera que la conexión puede ser limitada o nula y no perder el estado del juego

si el agente se encuentra en una plataforma.

Por este motivo muchas empresas y organizaciones están trabajando en este paradigma de programación, como por ejemplo Telecom Italia Labs. con la que han colaborado diferentes universidades europeas. También están trabajando sobre este tema diferentes empresas de tecnología móvil que han mostrado un interés en incorporar agentes móviles en sus aplicaciones, como es el caso de la empresa Nokia.

Actualmente ciertas empresas de telefonía móvil están trabajando en el uso de agentes móviles para el desarrollo de juegos.

El objetivo del proyecto es la comprobación de los agentes móviles como paradigma de programación para el desarrollo de juegos, donde los usuarios interactuarán con los agentes.

Después de cumplir el objetivo del proyecto se realizará un caso práctico, de manera que se explique, en una asignatura de Redes de Computadores, como el código móvil se puede transmitir por las redes de comunicación.

Para conseguir el objetivo principal del proyecto que se nos ha propuesto, nos hemos propuesto los siguientes objetivos concretos:

- Estudiar la tecnología de agentes móviles
- Analizar el entorno de ejecución de estos agentes, plataforma JADE.
- Caracterizar juegos basados en agentes.
- Desarrollar un *core* que compartan los juegos con tablero.
- Analizar y diseñar un caso práctico.
- Implementar el caso práctico.
- Realizar pruebas.
- Diseñar e implementar una práctica de Redes de Computadores

A continuación haremos una breve descripción de los capítulos que forman la memoria.

- **Capítulo 2. Background:**

  En este capítulo describiremos el entorno de desarrollo y las herramientas que se han utilizado para poder realizar el proyecto. Comentaremos qué son los agentes y la características que tienen. Por último comentaremos la plataforma, donde se ejecutan estos agentes, JADE.

- **Capítulo 3. Caracterización de juegos y análisis de requisitos:**

  En este capítulo inicialmente comentaremos las diferentes tareas en las que hemos dividido el proyecto al hacer la planificación. A continuación explicaremos la gran diversidad de juegos que existen en el mercado clasificándolos en tres grupos y explicaremos cada uno de ellos. Continuaremos analizando los juegos que son aconsejables desarrollarlos mediante la tecnología de agentes y aquellos que no. También explicaremos la perspectiva que ha de tener el sistema y por último los perfiles de los usuarios que utilicen esta aplicación.

- **Capítulo 4. Diseño de un caso práctico:**

  En este capítulo explicaremos la metodología de diseño que hemos utilizado, cómo hemos diseñado el caso práctico del juego del Parchís y la ontología que ha sido necesaria.

- **Capítulo 5. Implementación de un caso práctico:**

  En este capítulo veremos la implementación del caso práctico del juego del Parchís, el cual ha sido planteado en los capítulos anteriores. Comenzaremos implementando un agente, a modo de prueba de concepto, donde comprobaremos las diferentes tareas que los agentes pueden ejecutar. Después de haber realizado esta prueba de concepto implementaremos los agentes que formarán parte del juego del Parchís. Explicaremos como hemos implementado los agentes que forman el juego y la ontología que ha sido necesaria.

- **Capítulo 6. Pruebas:**

Este capítulo se ha dedicado a realizar pruebas sobre el juego del Parchís. Veremos las diferentes situaciones en las que se ha sometido la aplicación para la verificar que el resultado obtenido era el esperado. Hay que comentara que estas pruebas se han ido realizando a medida que se implementaba el juego.

- **Anexo A. Práctica de Redes de Computadores II:** En este anexo veremos el enunciado de una práctica para la asignatura de Redes de Computador II. Para poder realizar este enunciado habrá sido necesario el diseño e implementación de un juego muy sencillo mediante agentes móviles. El objetivo de esta práctica es aprender a desarrollar una aplicación distribuida utilizando la tecnología de agentes móviles.

Finalmente acabaremos la memoria haciendo una valoración de todo el proyecto, donde comentaremos los objetivos conseguidos y no conseguidos. Veremos cuales son las posibles aplicaciones del proyecto y cuales podrían ser las líneas de continuación futuras.

Capítulo 2

Background

En este apartado comentaremos que son los agentes y las características que tienen. También comentaremos la plataforma, donde se ejecutan estos agentes, JADE. Por último describiremos el entorno de desarrollo y las herramientas qué se han utilizado para poder realizar el proyecto.

2.1. Agentes

En este apartado explicaremos el significado del término *agente* y el uso que se ha realizado.

El término *agente* surgió en los años 90 y se ha adaptado a diferentes tecnologías donde ha sido muy utilizado, por ejemplo, en Inteligencia Artificial, Bases de Datos, Sistemas Operativos y Comunicaciones. Existen diferentes definiciones de *agente*, a continuación se expondrán dos de las más relevantes.

- " *Un agente es un sistema capaz de percibir a través de sensores la información que proviene del ambiente donde está insertado y reaccionar a través de efectores, por lo que se le puede definir como una entidad de software que exhibe un comportamiento autónomo, situado en una ambiente en el cual es capaz de realizar acciones para alcanzar sus propios objetivos y a partir del cual percibe los cambios"*, Russell y Norvig (2003)

- *"Un agente inteligente es una entidad software que, basándose en su propio conocimiento, realiza un conjunto de operaciones para satisfacer las necesidades de un usuario o de otro programa, bien por iniciativa propia o porque alguno de éstos se lo requiere"*,Wooldridge y Jennings (1995).

A continuación definiremos los puntos más importantes que tiene este término según el libro [BCG07].

- **Autónomo:** El agente opera sin la intervención directa de personas y tiene el control sobre sus acciones y estados internos.
- **Social:** Un agente debe ser social a causa de que debe cooperar con las personas y otros agentes para realizar diferentes trabajos.
- **Reactivo:** Los agente han de percibir el entorno y responder de forma rápida a los cambios que ocurren en él.
- **Proactivo:** Los agente no simplemente actúan como respuesta a su entorno, sino que pueden coger la iniciativa para llegar a un objetivo en concreto.

Algunos agentes poseen otra característica muy importante.

- **Móvil:** Los agentes tienen la habilidad de transmitirse a través de la red.

Según la característica de movilidad que tengan implementada existen dos grandes variedades de agentes: agentes móviles y agentes estáticos.

- **Agentes estáticos:** Esta variedad de agentes realiza sus funciones en una misma plataforma, esto significa que no tiene la capacidad, de moverse hacia otra plataforma.
- **Agentes móviles:** Los agentes móviles pueden ejecutar su código en diferentes plataformas. Para poder moverse un agente es necesario que el código se mueva, de manera que puedan continuar con su ejecución desde el punto en el que había finalizado en la agencia anterior.

A la acción de moverse un agente a través de la red de una plataforma otra, se le denomina *migración* y es la característica principal de los agentes móviles.

2.2. FIPA

FIPA (*Foundation for Intelligent Physical Agents*), es una organización de estándares para agentes y sistemas multiagentes que fue oficialmente aceptada por IEEE en 2005.

Esta fundación fue creada con el objetivo de producir especificaciones para la interacción de agentes y sistemas de agentes hetereogéneos. Estas especificaciones comprenden principalmente la infraestructura de las plataformas de ejecución de agentes y la arquitectura de éstos.

Según FIPA la definición que adopta de agentes, es: "*los agentes son entidades software que están localizadas en una única plataforma*".

Una plataforma de agentes FIPA se define como el software que implementa un conjunto de especificaciones FIPA. Para que se considere que sigue las normas de FIPA, una plataforma debe implementar al menos la especificación sobre la gestión de agentes y las relativas al lenguaje de comunicaciones de agentes.

FIPA tiene un modelo de referencia para la realización de plataformas que siguen las especificaciones de IEEE que se muestra en la figura 2.1 .

A continuación explicaremos las diferentes entidades que se muestran en la figura.

- **Facilitador de Directorio** (*Directory Facilitator*): Esta entidad ofrece un servicio de páginas amarillas para buscar los servicios que otros agentes ofrecen. Los agentes deben registrarse previamente en este servicio para ser localizados como proveedores de un servicio en concreto, y recurren a él en busca de agentes que ofrezcan un determinado servicio.

- **Sistema de Gestión de Agentes** (*Agent Management System*): Esta entidad controla el acceso y uso de la plataforma de agentes. De modo que ofrece

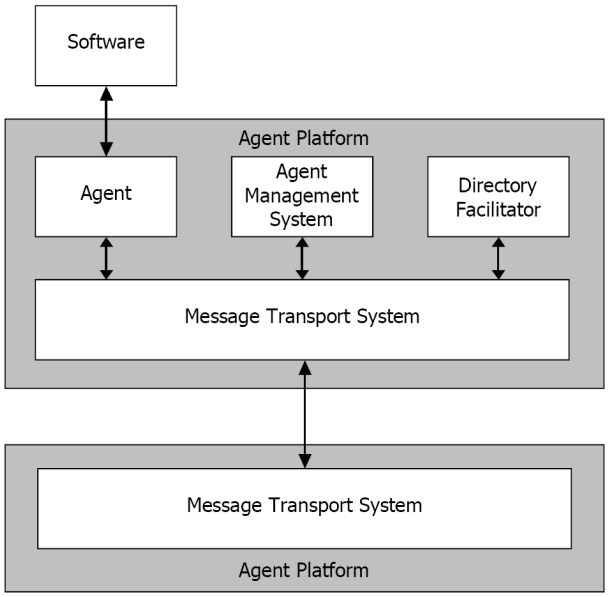

Figura 2.1: Ejemplo de modelo de referencia FIPA

un servicio similar al de las páginas blancas donde se ofrece la posibilidad de buscar agentes. Sólo puede existir un agente AMS por plataforma, y se encarga de generar identificadores válidos a parte de ofrecer un servicio de búsqueda de agentes por nombre.

- **Servicio de Transporte de Mensajes** (Message Transport Service): Esta entidad ofrece un servicio de comunicación entre agentes de diferentes plataformas y se encarga del transporte de mensajes entre agentes. FIPA, para esta entidad, especifica diferentes estándares sobre distintos protocolos y mecanismos de transporte que favorece la interoperabilidad entre distintas plataformas.

Mensajería

En este apartado se explicarán las especificaciones FIPA, sobre el envió de mensajes entre agentes que forman parte de la misma o diferente plataforma.

Mensajes ACL

La comunicación entre agentes es posible gracias a las especificaciones sobre la estructura de mensajes. El nombre que recibe el estándar de este mensaje es ACL (*Agent Communication Language*).

La utilización de este estándar nos asegura que dos agentes puedan comunicarse mediante mensajes que son inteligibles.

El mensaje ACL esta formado por diferentes parámetros. Dentro del conjunto de parámetros que forman un mensaje ACL existe el *performative* de uso obligatorio, que denota el tipo del hecho que se desea comunicar en el mensaje(INFORM, PROPOSE, FAILURE, CONFIRM,etc).

Otro parámetro es el *content* donde incluimos aquello que deseamos comunicar, por defecto el contenido es una cadena de caracteres. Un último parámetro a destacar es el de *ontology* que sirve para dar una mayor comprensión semántica. Este parámetro se explicará a continuación.

2.3. Plataforma JADE

JADE (*Java Agent Development Framework*) es una plataforma de software totalmente implementada en Java y desarrollada por Tilab (Telecom Italia Labs.) en la filosofía de Open Source, para el desarrollo de aplicaciones distribuidas basadas en agentes. Esta plataforma cumple con las especificaciones de FIPA comentadas en el apartado anterior.

La plataforma surgió en 1998 con la primera versión 1.3 que salió en febrero del 2000, bajo la licencia de LGPL(Lesser General Public License). La ejecución de JADE es posible tanto en computadoras como en dispositivos móviles (PDA's y teléfonos móviles).

JADE cumple totalmente con esta especificación de arquitectura descrita por FIPA, y cuando una plataforma JADE se ejecuta, el AMS y el DF se crean inmediatamente y se configura el módulo MTS para poder permitir la comunicación entre agentes mediante mensajes. La plataforma puede estar distribuida en diferentes "*hosts*", de manera que pueden haber diferentes aplicaciones Java y por tan-

to diferentes JVM (Java Virtual Machine) se pueden ejecutar en cada "*host*". Un "*contenedor*" se ejecuta en una JVM, esto significa que proporciona un entorno completo de ejecución de JADE y con el que es posible ejecutar agentes, permitiendo también ejecuciones concurrentes dentro del mismo "*host*". Existe un contenedor principal de la plataforma (*main-container*) que sirve de alojamiento para el AMS y el DF, y al que se deben subscribir el resto de posibles contenedores creados.

La plataforma de agentes contiene una interfaz gráfica GUI (Graphical User Interface) para controlar y monitorizar el estado de los agentes. Esta GUI también se puede utilizar para ejecutar agentes.

Behaviours

Un término importante a destacar en la plataforma JADE, es el *behaviour*. El *behaviour* especifica las tareas o servicios que un agente puede realizar.

Ontologías

Como se ha descrito anteriormente el contenido de un mensaje ACL está formado por diferentes parámetros. En este apartado se explica el parámetro *ontology*.

El parámetro *ontology* se utiliza junto con el *language* para otorgar mayor información dentro de un mismo mensaje ACL. Este parámetro sirve para relacionar el mensaje con una ontología definida anteriormente.

Existen diferentes definiciones de ontología pero a continuación se muestra una muy completa realizada por John F. Sowa [JFS07]:

"*El objetivo de las ontologias es el estudio de las categorías de las cosas que existen o pueden existir en un dominio. El producto de este estudio , llamado ontología, es un catálogo de tipos de cosas de las que se asumen su existencia en un dominio de interés D, desde la perspectiva de una persona que utiliza un lenguaje L para el propósito de hablar sobre D. Los tipos de ontologías representan los predicados, el sentido de las palabras, o conceptos y tipos de relaciones del lenguaje L cuando éste es utilizado en la discursión o temas del dominio D.*"

La ontología debe ser conocida por los agentes que se comunican mediante los mensajes ACL, porque sino los agentes receptores de estos mensajes no entenderán su contenido.

Capítulo 3

Caracterización de juegos y análisis de requisitos

En este capítulo inicialmente comentaremos las diferentes tareas en las que hemos dividido el proyecto al hacer la planificación. A continuación explicaremos la gran diversidad de juegos que existen en el mercado clasificándolos en tres grupos y explicaremos cada uno de ellos. Continuaremos analizando los juegos que son aconsejables desarrollarlos mediante la tecnología de agentes y aquellos que no. También explicaremos la perspectiva que ha de tener el sistema y por último los perfiles de los usuarios que utilicen esta aplicación.

A continuación explicaremos como se ha planificado el proyecto.

3.1. Planificación

En este apartado veremos en el diagrama de Gantt las diferentes tareas en las que hemos dividido el proyecto al hacer la planificación (figura 3.1). Hay seis tareas principales: la planificación, los estudios previos, el análisis de los requisitos necesarios, el diseño y la implementación de la aplicación, el diseño e implementación de un caso práctico para el programa docente y por último la parte de la memoria.

Las fechas que se observan en el diagrama son aproximadas. Inicialmente se

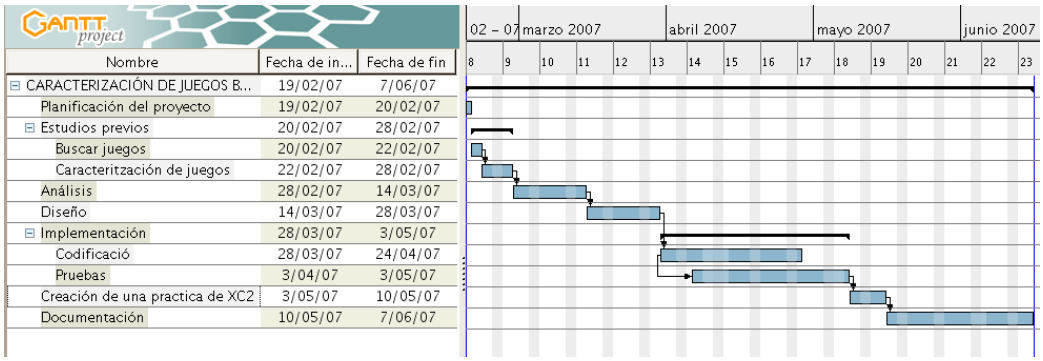

Figura 3.1: Planificación del proyecto

ha previsto que el proyecto se inicie el 19 de Febrero y termine el 7 de Junio.

3.2. Tipos de juegos que existen en el mercado

Existe diferentes tipos de juegos en el mercado, a continuación los explicaremos clasificándolos en tres grupos: Arcade, Estratégicos y Reproducción de juegos de mesa.

3.2.1. Arcade

Son juegos que requieren un ritmo muy rápido, exigiendo tiempos de reacción mínimos y una alta atención. La interacción del usuario y el juego es continua. En este tipo de juegos el factor más relevante es la estrategia.

A continuación explicaremos los tres grupos que forman este tipo de juegos.

De plataforma

Son aquellos juegos donde se utiliza a un jugador que se encuentra en un escenario bidimensional desplazándose de izquierda a derecha y de arriba hacia abajo. El objetivo consiste en pasar de una plataforma a otra, esquivar objetos y eliminar enemigos. Ejemplos de este grupo de jugos Arcade: *Super Mario Bros, Crash Bandicoot, Pandemonium, Sonic, etc*.

DE competiciones deportivas

Son juegos donde se reproducen deportes reales. Esto significa que el usuario se pone en el lugar de un deportista y realiza unas determinadas pruebas. Un ejemplo es el atletismo, donde un usuario puede realizar las diferentes pruebas que se realizan: carreras de 100 metros, salto con pértiga, salto de longitud, salto de altura.

De acción

Son juegos con un gran desarrollo lineal. Esto significa que los escenarios cambiar eliminando un determinado número de enemigos o encontrando un determinado objeto. Ejemplos de este grupo de juegos Arcade son: *Batman Forever, Jurassic Park, etc*.

3.2.2. De simulación

Este tipo de juegos simulan actividades o experiencias que sólo lo pueden realizar un pequeño grupo de la sociedad. Permiten al jugador asumir el control de situaciones o tecnologías específicas. Las características especiales que incorpora son la baja influencia del tiempo de reacción en comparación con los juegos Arcade; la utilización de estrategias complejas y la necesidad de enfrentarse a situaciones donde se requieren conocimientos específicos sobre la simulación. Existen una gran diversidad de simuladores en el mercado, como pueden ser los de dirigir y controlar aviones, coches, motos en situaciones totalmente realistas.

Los juegos de simulación se dividen en dos grupos:

Instrumentales

Este grupo de juegos de simulación surgieron con la necesidad de preparar a futuros pilotos de aviones. Mediante estos simuladores aéreos se pueden entrenar a los pilotos para posibles situaciones de alto riesgo. Ejemplos de este grupo de simulación: *Modern Air Combat y B-17 Fly Fortress*.

Situacionales

En este grupo de juegos de simulación se intenta que el jugador asuma un papel específico. Este papel viene determinado por el tipo de simulación que se realice. A continuación se comentan los diferentes grupos que forman los juegos Situacionales.

- **Deportivos**: El jugador adquiere un papel de entrenador, seleccionando jugadores o planificando una estrategia para llevar a cabo con un equipo. Ejemplos: *PC Futbol.*
- **Simuladores de Dios**: En este tipo de juego el usuario tiene un papel divino. Esto significa que el usuario asume un papel de máximo controlador de la situación, donde puede controlar un grupo de personajes que simulan vivir en la vida real.

3.2.3. Estratégicos

El jugador adopta una papel específico donde sólo conoce el objetivo final. Con frecuencia los personajes son de ficción y provienen del mundo de la literatura y del cine. A diferencia de los juegos de acción, en los de estrategia es necesario pensar los movimientos para ganar u organizar una jugada para ganar. Existen tres grupos que forman parte de los juegos Estratégicos:

Aventura gráfica

La acción se desarrolla a través de las órdenes de un jugador que actúa generalmente en tercera persona. La dinámica de este tipo de juego consiste en avanzar resolviendo rompecabezas, planteados como situaciones que ocurren durante la historia. El usuario interactúa con el personaje y objetos a través de un menú de posibles acciones. Generalmente se utiliza el cursor para mover el personaje por el escenario.

De estrategia militar

Este tipo de juegos es similar a los juegos de mesa militares donde el usuario controla un ejército. El control se realiza en tercera persona y es necesario

esperar tener el turno para poder realizar movimientos. Ejemplo de juegos Estratégicos: *Risk*

De rol

El control se realiza generalmente en tercera persona, y al igual que en los juegos de estrategia militar, es necesario esperar tener el turno para realizar un movimiento. La historia inicialmente no tiene un final esperado.

3.2.4. Reproducción de juegos de mesa

Por último nos encontramos con los juegos que reproducen juegos de mesa. Estos juegos tienen el mismo desarrollo que sus originales de mesa.

Un juego de mesa es un juego que requiere de una mesa o soporte similar para poder jugarse. En este tipo de juegos generalmente juega un grupo de personas alrededor de él. El azar es una parte muy importante en este tipo de juegos, pero también es necesario el uso de estrategias para poder jugar. Ejemplos de reproducciones de juegos de mesa: Domino, Juegos de cartas, Parchís, Tres en ralla, Ajedrez, Monopoly, Puzzles, Conecta 4.

Después de haber comentado los juegos que existen en el mercado, a continuación vamos a caracterizar los juegos que se pueden implementar con la tecnología de agentes.

3.3. Caracterización de juegos

Después de explicar los diferentes tipos de juegos que existen en el mercado actual, hay algunos de ellos donde la posibilidad de implementarlos mediante agentes es más óptima. A continuación se muestra los diferentes tipos de juegos que se puedan implementar con agentes y sin agentes.

3.3.1. Juegos implementados sin agentes móviles

En este apartado comentamos los tres grupos de juegos que no se pueden implementar con la tecnología de agentes móviles.

Simuladores

Como hemos podido ver en el apartado anterior existen diferentes tipos, donde encontramos los simuladores instrumentales y situaciones. Este tipo de juegos no es recomendable realizarlos con agentes móviles debido a que solo participa un usuario, por lo que el agente no tendría la necesidad de migrar.

Arcade

El principal problema en este tipo de juegos es la inexistencia de turnos en el juego porque en el juego siempre están sucediendo acciones. Esta circunstancia es un gran inconveniente, debido a que el comportamiento de los agentes no soporta tal velocidad de reacción.

En este tipo de juegos también hay que tener en cuenta la posición física donde se encuentra el personaje, las variables de entorno (salud, comida, etc) y el tiempo. El tiempo es un punto muy importante debido a que requieren un ritmo muy rápido para poder jugar en condiciones ideales. Este motivo podría ser un problema para poder desarrollar estos juegos con agentes.

A continuación se comentan los dos tipos de juegos Arcade:

- **De plataforma**

  Es necesario el posicionamiento de las plataforma donde saltar, debido a la gran precisión que es necesaria, para poder realizar un salto. La velocidad del juego es otro factor importante a tener en cuenta para comprobar que no es posible la implementación de estos juegos con agentes.

- **De competiciones deportivas**

  Tiene los mismos puntos en común que los de plataforma, añadiendo que ahora no solo es necesario el posicionamiento para poder saltar sino también para poder esquivar objetos.

Algunas reproducciones de juegos de mesa

Las reproducciones de juegos de mesa que forman parte de este grupo son los *Puzzles*, donde los únicos conocimientos necesarios son la posición de las fichas y el total de fichas que forman el puzzle. Existe un contrapunto para un posible desarrollo con agentes móviles y es que solo participa un jugador. Est conlleva que no és necesaria la migración para este juego.

3.3.2. Juegos implementados con agentes móviles

En este apartado comentamos los juegos donde su implementación es apropiada realizarla con la tecnología de agentes móviles.

Este grupo de juegos contiene un estado actual, lo que entendemos como el estado del juego en este momento. A veces no es suficiente conocer el número de de fichas o cartas que se tienen, las que no y las que sí se pueden usar, sino que en ciertos juegos es necesario conocer el estado físico del juego. Este aspecto es necesario para determinar en que posición queremos poner la ficha o el personaje, dependiendo de las que ya hay en ese momento. Para definir este estado físico es necesario crear una matriz para poder situar los objetos que van a intervenir en el juego.

Otro aspecto es el conocimiento de las cartas que se han usado, cuáles no y cuáles poseen los jugadores.

Dependiendo del estado, tenemos diferentes tipos de juegos con agentes móviles:

Tipo matriz

El estado físico del juego se guarda en una matriz para poder situar los objetos que van a intervenir en el juego. Es útil en juegos donde es necesario el uso de un tablero.

Existen dos grupos de juegos que utilizan una matriz para implementar el estado del tablero:

- **Estratégicos** Partimos de la base que aunque un jugador deje la partida, ésta podrá continuar, debido a que habrá más enemigos creados por el propio juego.
  - **De aventura gráficas:** Son tan importantes las variables de entorno como la posición de los elementos físicamente. En el momento que quisiéramos interactuar con algún elemento tendríamos que conocer la posición física donde se encuentra.
  - **De rol y de estrategia militar:** Al igual que en el caso anterior, es necesario conocer el estado físico en el que se encuentra el juego para poder moverte a la posición donde más te interese. También es necesario conocer las variables de entorno, como pudiera ser: el alimento, la vida, la madera para la construcción, minerales, etc.
- **Algunas reproducciones de juegos de mesa:**

  Existe un grupo mayoritario formado por reproducciones de juegos de mesa donde el estado del juego se puede implementar mediante la matriz. Estos juegos son: la Oca, el Tres en ralla, el Ajedrez, el Monopoly, el Risk y el Conecta 4, etc.

Tipo tapete

En este grupo de juegos el estado físico en el que se encuentra la partida no nos interesa, en cambio el estado de las variables de entorno sí que es necesaria. Los juegos que forman parte de este grupo son algunas reproducciones de juegos de mesa:

- Dominó y juegos de cartas: Nos interesa conocer las fichas o cartas que están encima de la mesa, pero en cierta manera no nos interesa la posición que tienen. Es necesario el conocimiento de las cartas o fichas que se han usado, cuáles no y cuáles poseen los jugadores.

A continuación comentaremos el análisis de requisitos de la aplicación, que ha sido necesario para poder diseñar el caso particular del juego del Parchís. Este análisis se ha realizado siguiendo el estándar descrito por IEEE [SRS98].

3.4. Perspectiva de la aplicación

Interfaz del sistema

Los usuario que forman una partida del juego están distribuidos en diferentes plataformas.

Utilizamos la tecnología de agentes móviles para la implementación de esta aplicación distribuida.

Para ejecutar los agentes se necesita un software específico. En nuestro caso se utiliza la plataforma JADE.

El inicio del juego lo llevará a cabo un usuario *master*, encargado de crear la partida. Mediante el conocimiento previo de las direcciones de las plataformas crea un fichero de configuración. Este usuario se encarga de ejecutar un agente móvil.

Este agente contiene el estado del tablero del juego y la lista de turnos definida en el fichero de configuración. Después de cada jugada el agente migra a la siguiente plataforma siguiendo el turno especificado en la lista.

El uso de esta aplicación es posible en terminales móviles. Es necesario tener, JADE-LEAP (Lightweight Extensible Agents Platform), una variante más ligera de la plataforma JADE y el conocimiento de los terminales participantes en la partida o bien utilizar una política de descubrimiento de terminales vecinos.

Interfaz del usuario

El usuario dispone de una ventana gráfica y de la entrada estándar.

La ventana gráfica es utilizada para observar como se encuentra el estado del juego.

Mediante la entrada estándar se realiza una comunicación bidireccional entre el usuario y el Agente Jugador. El usuario introduce la jugada deseada y el Agente Jugador le informa si ha ocurrido algún error al introducir la jugada, del fin del juego, y del resultado obtenido.

Interfaz hardware

Es necesario PC x86 o superiores y dispositivos móviles(PDA's o teléfonos móviles) para hacer posible nuestra aplicación.

Interfaz software

Para la utilización de nuestra aplicación es necesario un sistema operativo que soporte de manera indispensable la maquina virtual de JAVA. Los sistemas operativos que formarían parte de este grupo serían: Windows, Linux, Simbian, MAC OS X, etc.

La aplicación se puede ejecutar mediante una MVJ (máquina virtual de Java) que soporte una versión de JAVA igual o superior a 1.4. Este requisito es necesario porque la plataforma JADE está implementada en este lenguaje.

A continuación comentaremos las aplicaciones software junto con su versión que han sido necesarias para el posterior desarrollo del este proyecto.

- Framework de JADE, versión 3.4.1.
- Servicio de movilidad interplataforma. Este servicio constituye el ADD-ON de movilidad de JADE. Versión 1.96.
- Framework de JDOM, versión 1.0.

Interfaces de comunicaciones

Para la utilización de agentes móviles se necesita una infraestructura de comunicaciones. Esta infraestructura puede estar formada de diferentes formas según

nuestro propósito. En el caso de querer utilizar nuestro proyecto en una red local sería necesario una conexión por cable Ethernet, Wireless o via Bluetooth. En caso de querer utilizarlo en una *WAN* deberemos tener una interfaz de red con conexión a Internet mediante el cable telefónico. Estas interfaces de comunicaciones son necesarias para que los agentes tengan la posibilidad de migrar entre las plataformas ditribuidas.

3.5. Características de los usuarios

En este apartado vamos a comentar las características que deben cumplir los usuarios que utilicen nuestro proyecto. Dependiendo del modo en que se utilice el proyecto existen dos perfiles de usuario.

Jugador

Este usuario es aquel que únicamente va a jugar con el juego Parchís. El único requisito que debe cumplir es el conocimiento de las reglas del Parchís.

Desarrollador

Este usuario podrá utilizar el núcleo de este proyecto para el desarrollo de una nueva aplicación (Monopoly, Risk, Uno, etc). Previamente serán necesarios conocimientos avanzados de los requisitos del software descritos anteriormente.

3.6. Restricciones del sistema multiagente

En la elaboración de este proyecto es necesario que se sigan ciertas restricciones. Estas restricciones son:

- La utilización de las especificaciones de FIPA respecto al envío de mensajes entre agentes.
- La identificación única de los agentes mediante el AID.

Capítulo 4

Diseño

Una vez finalizado el análisis de requisitos el siguiente paso es realizar el diseño. En este capítulo explicaremos la metodología de diseño que hemos utilizado, cómo hemos diseñado la aplicación del Parchís y la ontología que ha sido necesaria.

4.1. Metodología de diseño

En esta aplicación se ha seguido un modelo evolutivo. Esto significa que el diseño de la aplicación ha ido variando a medida que ha sido conveniente para el desarrollo de la aplicación de una manera más óptima. Para llevar a cabo el juego se han realizado los diagramas de caso de uso, diagramas de actividades, diagramas de secuencia y por último los diagramas de clase.

A continuación estan comentados loa diferentes decisiones de diseño que se han utilizado en la implementación del juego.

4.2. Decisiones de diseño

En este apartado se explicarán las diferentes decisiones de diseño que se han optado por implementar.

En primer lugar se comentan que tipo de agentes son necesarios.

Agentes de la aplicación

Para la realización del juego son necesario los siguientes agentes.

- **Agente Móvil Controlador(AMC)**:

  El AMC se encarga de implementar y controlar el juego. Esto significa que se encarga de iniciarlo, terminarlo, evaluar la jugada del usuario y comunicarse con el AJ.

  Este agente es el único que contiene el estado del tablero y por lo tanto es el único que lo puede modificar. El comportamiento de este agente lo comentaremos con más profundidad en posteriores apartados.

  Una característica muy importante que diferencia al AMC del resto de agentes que forman el juego es la movilidad. Esta característica es debido al hecho de que el tablero ha de moverse por todas las plataformas jugadoras. Como el AMC es el único agente que contiene el tablero pues ha de ser móvil.

- **Agente Jugador(AJ)**:

  Es el agente encargado de interactuar con el usuario. Recibe los movimientos del juego que desea realizar el usuario y los pasa al AMC. Este agente no es móvil porque siempre permanece en la misma plataforma.

Inicio del juego

Para inicializar el juego existen diferentes posibilidades según el carácter dinámico que se quiera implementar.

- **Plataforma Estática.**

  Existe una plataforma donde se inicia el juego. Dentro de esta plataforma pueden haber implementados tantos AMC cómo juegos posibles. También hay que tener en cuenta que puede haber dos AMC que implementen el mismo juego. Esta plataforma es la encargada de iniciar el juego y ejecutar el AMC que contenga un juego determinado. Para poder ejecutar este agente

es necesario que se le informe de que juego se desea ejecutar y los participantes que formarán parte del juego. En el caso de la *Plataforma Estática* esta información se extrae de un fichero de configuraciones, previamente construido por el propio usuario *master* después de hablar con el resto de participantes. Este usuario *master* es el encargado de ejecutar el AMC que contenga el juego deseado. Este AMC crea la lista de turnos de las plataformas que intervienen en el juego y comienza el juego en esta plataforma.

En nuestro caso para el diseño de nuestra aplicación se ha utilizado una *Plataforma Estática*. El uso de este tipo de plataforma es por el hecho de que se nos ha propuesto la implementación de un juego, por lo que conocemos que AMC es el que se ha de ejecutar.

- **Plataforma Dinámica.**

  Al igual que en el caso anterior existe una plataforma que inicia el juego donde hay implementados diferentes juegos en sus respectivos AMC. En las plataforma dinámica dan la posibilidad de que el usuario se comunique con ella pudiendo asi elegir el tipo de juego al que se desea jugar. Dependiendo de la petición que le llegue ejecuta aquel AMC que tenga implementado ese juego. Para ello siempre tiene ejecutándose un agente que está esperando la petición del juego a ejecutar. Cuando este agente recibe una petición de un AJ, éste ejecuta el AMC correspondiente a ese juego. El AMC tiene que crear la lista de turnos. Para crear esta lista ha de esperar hasta que el número necesario de jugadores para jugar se lo comuniquen. A continuación se comentan las diferentes posibilidades que existen para comunicar el deseo de participar en el juego.

  1. Los AJ se desplazan hasta la plataforma que inicia el juego y esperan hasta que haya el número necesario de jugadores. En ese momento el AMC de la plataforma informa a cada AJ del turno que tiene.
  2. Los AJ envián un mensaje al AMC del juego, que se encuentra en la plataforma que inicia el juego, informando de su participación en

la partida. La plataforma, que inicia el juego, después de recibir los mensajes por parte de los AJ, envía a cada uno de ellos un mensaje informándoles del turno que tienen.

3. Los AJ al igual que en el caso anterior envían un mensaje al AMC, que se encuentra en la plataforma que inicia el juego. Al recibir tantos mensajes como número de jugadores es posible, esta plataforma envía un agente móvil mensajero. Este agente mensajero migra hasta las plataformas jugadoras y modifica su fichero de configuración.

Fin del juego

El AMC, después de evaluar cada jugada proveniente de los AJ, puede determinar si el juego ha finalizado. En este momento dado, el AMC comunica a todos los AJ, mediante mensajes, la puntuación que han obtenido junto al nombre del vencedor de la partida. Éstos a su vez comunican a su usuario esta información por pantalla.

Seguridad

Los requisitos de seguridad son los siguientes:

1. Movimiento: El agente jugador solo puede realizar un movimiento cada vez que el agente controlador le de permiso. De este modo, podemos evitar que el jugador pueda mover dos fichas a la vez. El estado del juego antes de realizar el movimiento no sea modificable por el agente jugador.

2. Puntuación: Para poder evitar una falsificación en la puntuación o en el resultado de una tirada de dados, el agente jugador no puede tener esta información en su poder. De manera que el agente jugador deberá pedir esta información al agente controlador. A esta política se la conoce como *append-only*.

A continuación se muestran los diferentes diagramas UML que se han utilizado en el diseño de nuestro juego.

4.3. Diagramas UML

En este apartado veremos los diferentes diagramas del diseño de la aplicación. Comenzaremos viendo los diagramas de casos de uso, para definir mejor los actores que participan en la aplicación. A continuación veremos los diagramas de actividades para ver las diferentes acciones que se realizan. Después veremos los diagramas de secuencia para conocer el orden de las acciones y por último veremos los diagramas de clases.

A continuación se comentan los diferentes diagramas de casos de uso que se han necesitado para el diseño de la aplicación.

4.3.1. Diagramas de casos de uso

El diagrama de casos de uso (figura 4.1) nos muestra como es la aplicación de manera general. En este diagrama podemos observar los actores que participan y las relaciones que tienen entre ellos. A continuación describiremos las características principales de cada uno de los casos de uso.

- **Inicio la aplicación:**

  Para comenzar la aplicación es necesario que estén en funcionamiento como mínimo dos plataformas. De manera, que estas plataformas hayan lanzado un AJ y pueda interactuar con el AMC. Dependiendo de la plataforma se realizan unas acciones u otras, pero todo esto lo comentaremos posteriormente.

- **Aplicación en juego:**

  Una vez iniciado el juego el AJ muestra a su usuario el tablero. El usuario puede realizar una jugada correcta o no, pero esto lo decide el AMC. En el caso de ser correcta la jugada el AMC modifica el estado del tablero.

- **Evaluación de la jugada.**

  El AMC evalúa si la jugada realizada por el usuario es correcta. Para llevar a cabo este proceso de evaluación es necesario el conocimiento de las reglas

del juego. Dependiendo del resultado de la evaluación de la jugad el AMC realiza diferentes acciones. En el caso de ser correcta el AMC modifica el estado del tablero y migra a la siguiente plataforma. En el caso de ser incorrecta la jugada, el AMC vuelve a enviar el tablero y el número del dado al AJ para que el usuario pueda hacer una nueva jugada.

- **Fin de la aplicación.**

  Existen dos posibilidades para terminar la aplicación.

  La primera posibilidad es la llegada de cuatro fichas de un mismo color lleguen a la casilla final correspondiente a ese color.

  La otra posibilidad es el hecho de que ninguna plataforma esté en ejecución. De este modo, el AMC se elimina y por tanto la aplicación finaliza.

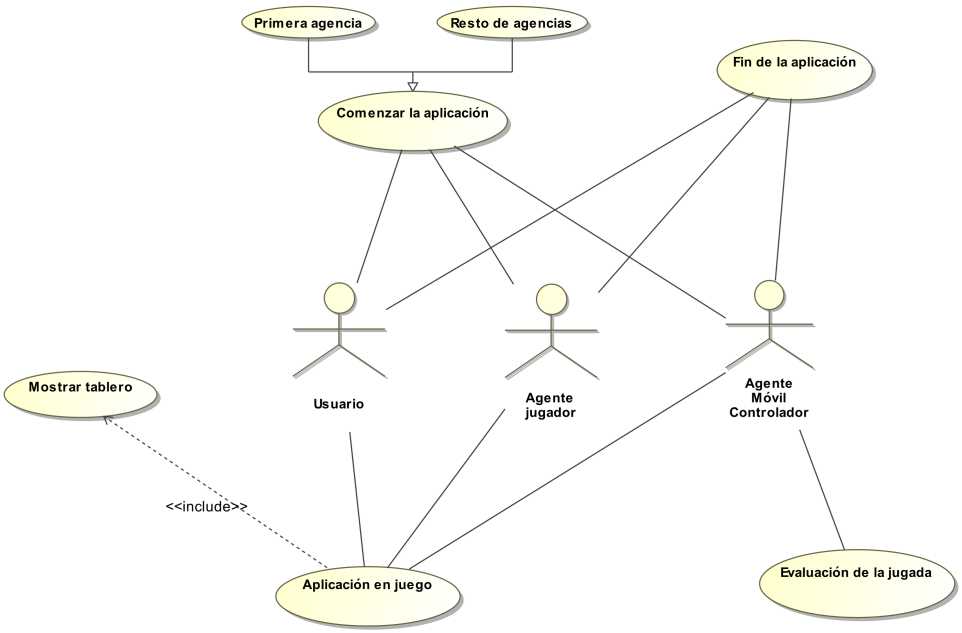

Figura 4.1: Diagrama de casos de uso

4.3.2. Diagrama de actividades

Seguidamente se comentan los diagramas de actividades. Éstos son de gran utilidad para la explicación de las diferentes acciones que se realizan en los casos de uso.

Diagramas de actividades del caso de uso *Inicio la aplicación*.

El caso de uso *Inicio de la aplicación* está formado por dos casos de uso, dependiendo de que plataforma. En la plataforma encargada de iniciar el juego se realizan las acciones reflejadas en el caso de uso *Inicio de la aplicación para la primera plataforma*. En las otra plataformas, que forman parte del juego, se realizan la acción mostrada en el caso de uso *Inicio de la aplicación para el resto de plataformas*. A continuación describiremos las acciones realizadas en cada uno de estos casos de uso.

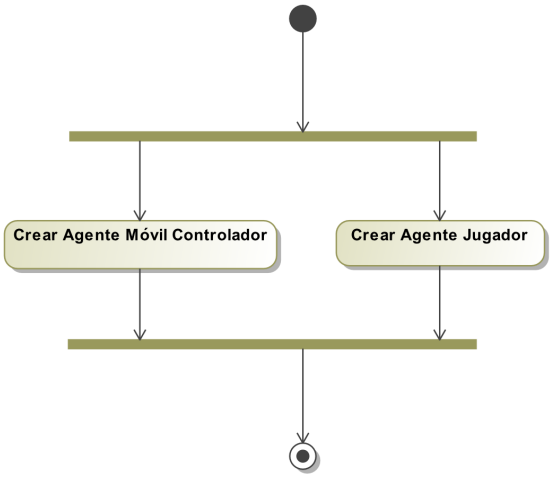

Figura 4.2: Diagrama de actividad correspondiente a Comenzar aplicación en la primera agencia

- **Inicio de la aplicación en la primera plataforma:**

Inicialmente es necesario que en el host, donde se está ejecutando la plataforma encargada de iniciar el juego, estén implementados los AMC y AJ. El usuario participante en el juego ha de realizar las acciones que se muestran en la figura 4.2:

- Lanzar el Agente Móvil Controlador.
- Lanzar el Agente Jugador.

El órden en el que se ejecutan las acciones es importante porque el AMC necesita que esté ejecutándose un AJ. El AMC cuando al ser lanzado envía el tablero al AJ. En el caso de que el AJ no estuviera en ejecución daría una problema al enviar el tablero porque no encuentra al destinatario.

Por el contrario el AJ no necesita que el AMC esté en ejecución. Esto es debido a que el AJ inicialmente espera recibir información por parte del AMC, y hasta que no recibe esta información no realiza ninguna nada.

- **Inicio de la aplicación para el resto de plataforma:**

  A continuación vamos a comentar como es el inicio de una aplicación para las plataformas que forman parte del juego y no son la que lo inicia.

  Los *hosts* donde se están ejecutándose estas plataformas necesitan tener implementado el AJ. Estos hosts son utilizados por los usuario que deben realizar la siguiente acción.

  - Lanzar el Agente Jugador.

  Excepto de la plataforma que inicia el juego, en el resto de las plataformas solo es necesario lanzar al AJ para que éste pueda interactuar con el usuario y el AMC.

Diagrama de actividades del caso de uso *Aplicación en juego*.

El siguiente diagrama es el referente al caso de uso *Aplicación en juego*(figura 4.3). Como se observa en este diagrama interactúan tres actores: el usuario, el

AJ y AMC. Dependiendo de las acciones que realiza cada uno de los actores, el diagrama sigue diferentes flujos de ejecución.

1. El primer flujo de ejecución empieza en con la tirada de dados por parte del AMC y la posterior acción de mostrar el tablero por parte del AJ. Dependiendo de las posiciones de las fichas del usuario, éste no puede hacer una jugada. En este caso el AMC ha de migrar a la siguiente plataforma según la lista de turnos. Si la migración ha sido posible el AMC vuelve ha realizar las acciones iniciales (tirar dados y mostrar tablero).

2. El siguiente flujo empieza con la posibilidad de que el usuario puede hacer una jugada. El usuario realiza un movimiento y lo comunica al AJ. El AJ le envía el movimiento deseado al AMC. El AMC al recibir un movimiento lo evalúa para conocer la veracidad de éste. En el caso de ser correcta la jugada y no se haya llegado el final del juego, el AMC migra.

3. El último posible flujo de ejecución continúa cuando el AMC conoce que el movimiento, por parte del usuario, es incorrecto. En este caso el AMC informa al AJ sobre la jugada incorrecta. El AJ al recibir esta información le comunica al usuario el error del movimiento, el número de jugadas que le restan y vuelve a mostrarle el tablero.

Diagrama de actividades del caso de uso *Fin de la aplicación*.

El diagrama que se muestra en la figura 4.4 es el que muestra el comportamiento del caso de uso *Fin de la aplicación*. En este diagrama podemos observar que forman parte de él dos actores: el Agente Jugador(AJ) y el Agente Móvil Controlador(AMC). En este diagrama, al igual que en el apartado anterior, se pueden observar diferentes flujos de ejecución. El AMC, después de evaluar el último movimiento realizado por parte del usuario, conoce el fin del juego. Para que el usuario conozca el fin del juego, el AMC le comunica al AJ quien ha sido el jugador ganador y la puntuación que ha conseguido. A continuación se explican los diferentes flujos que existen.

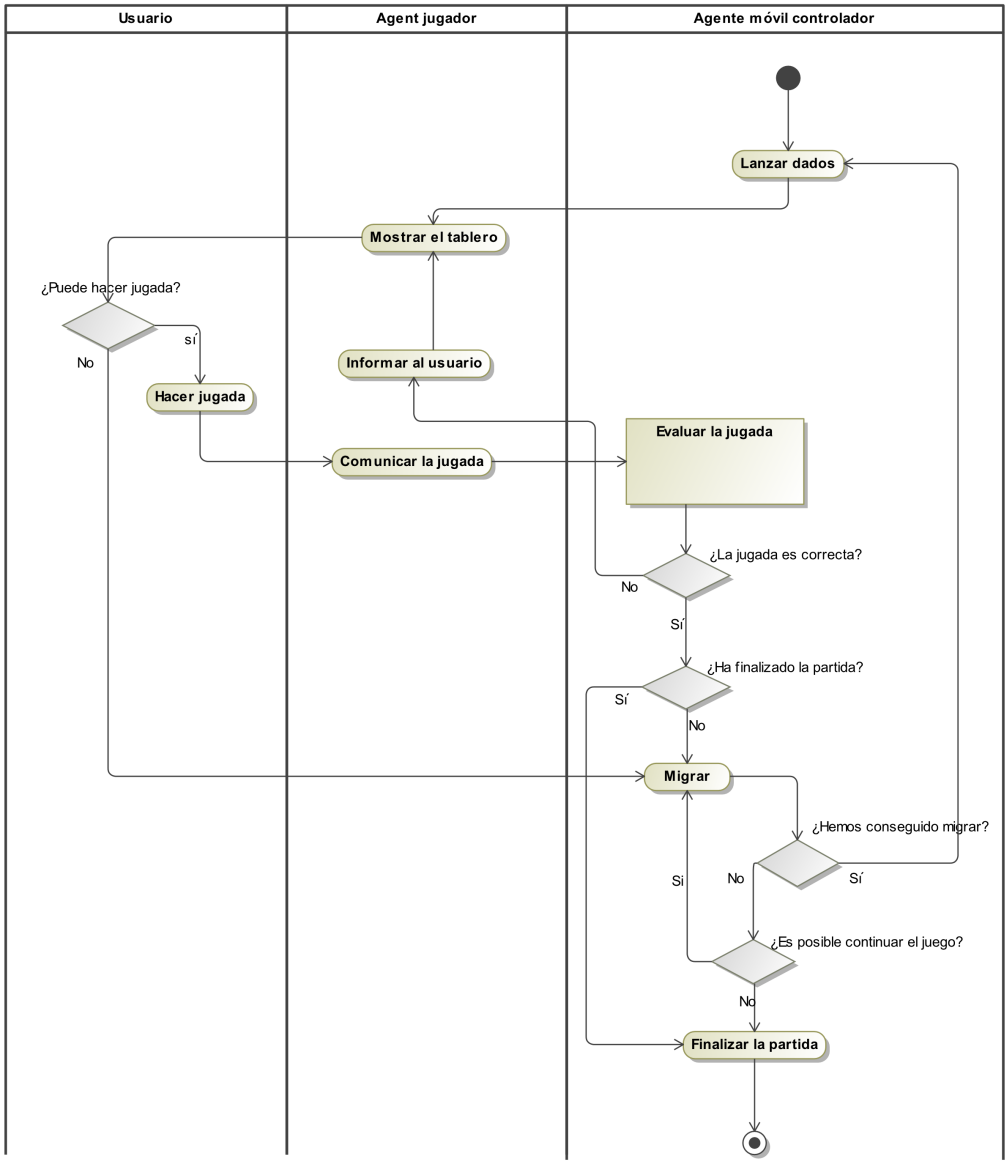

Figura 4.3: Diagrama de actividad correspondiente a Aplicación en juego

1. El primer flujo continúa cuando el AMC conoce que la plataforma que está informando, es la última en conocer el fin del juego. El AMC informa al AJ que se ha de morir. Después de realizar esta acción, el AMC se muere, finalizando así el juego.

2. Otro posible flujo continúa después de que el AMC conozca que no está en la última plataforma. El AMC migra a la siguiente plataforma, según la lista de turnos. Si la migración ha sido posible se vuelve ha informar del fin del juego a otro usuario.

3. El último posible flujo es en el momento en el que es posible informar al resto de plataformas jugadoras. En este caso el AMC migra a otra plataforma para poder continuar informando a todas las plataformas jugadoras del fin del juego.

4.3.3. Diagramas de estado

En este apartado se muestran los diferentes estados que tienen los agentes implicados en el juego. Primero explicaremos los estados por los que pasa el Agente Móvil Controlador y posteriormente los del Agente Jugador.

Estados del Agente Móvil Controlador

El AMC, de la misma manera que el AJ, pasa por ciertos estados a lo largo de su vida, realizando diferentes acciones. En la figura 4.5 se observan los estados por los que pasa. A continuación explicaremos más a fondo estos estados.

- **Estado de inicializar el tablero.** El primer estado por el que pasa el AMC es el que realiza la acción de recoger información de un fichero de configuración e inicializar el estado del tablero del juego.

- **Estado de enviar el tablero y la tirada.** El AMC después de haber inicializado el tablero ha de enviar el tablero y el número de la tirada al usuario. Para poder enviar esta información al usuario se necesita crear un mensaje

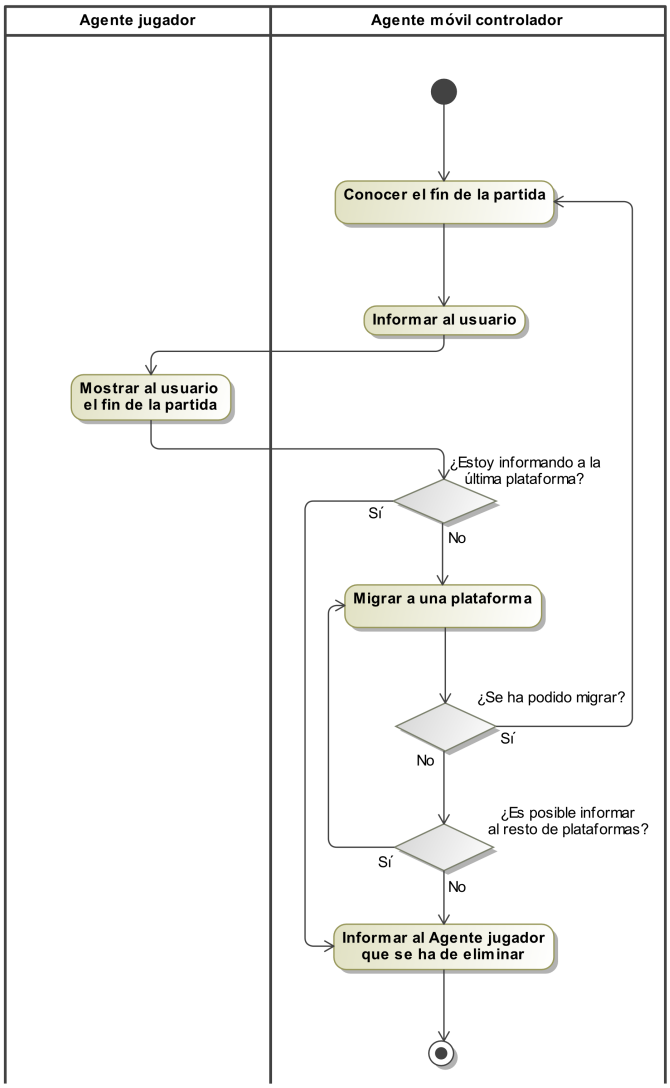

Figura 4.4: Diagrama de actividad correspondiente al caso de uso Fin de la aplicación

que contenga esta información. La introducción de la información en este mensaje no es un proceso sencillo y se explica en el capítulo de implementación.

- **Estado de esperar respuesta del Agente Jugador.** Posterior al envío del tablero y del número del dado, el AMC queda esperando recibir un movimiento por parte del jugador. Mientras el AMC espera recibir un movimiento, queda en un estado de total bloqueo. Esto significa que el agente no realiza ninguna acción.

- **Estado de evaluar la jugada recibida.** El AMC entra en nuevo estado después de recibir un movimiento por parte del jugador. En este estado el AMC evalúa el movimiento recibido para comprobar si el movimiento es lícito. En el caso de que el movimiento sea correcto el AMC ha de hacer las modificaciones necesarias sobre el estado del tablero.

- **Estado de migrar.** El agente después de evaluar la jugada y modificar el tablero, ha de pasar a un estado de migración. El AMC migra a la siguiente plataforma según la lista de turnos. En el caso de no poderse realizar la migración, el Add-On de JADE proporciona un servicio con el que intenta la migración durante un tiempo determinado. Si pasado este tiempo el AMC no ha conseguido migrar a la plataforma deseada, migra a la siguiente plataforma. El tiempo de migración del AMC entre diferentes plataformas puede variar dependiendo del tráfico que tenga la red o de la versión del Add-On de migración de la plataforma JADE.

- **Estado de enviar resultado del juego.** El AMC, después de pasar por el estado de *Evaluar la jugada recibida*, sabe que ha llegado el fin del juego. En este momento el agente comunica a todos los AJ quien es el ganador y la puntuación de cada jugador.

- **Estado de eliminación.** El último estado por el que pasa el AMC es por el estado de eliminación. El AMC en este estado se muere.

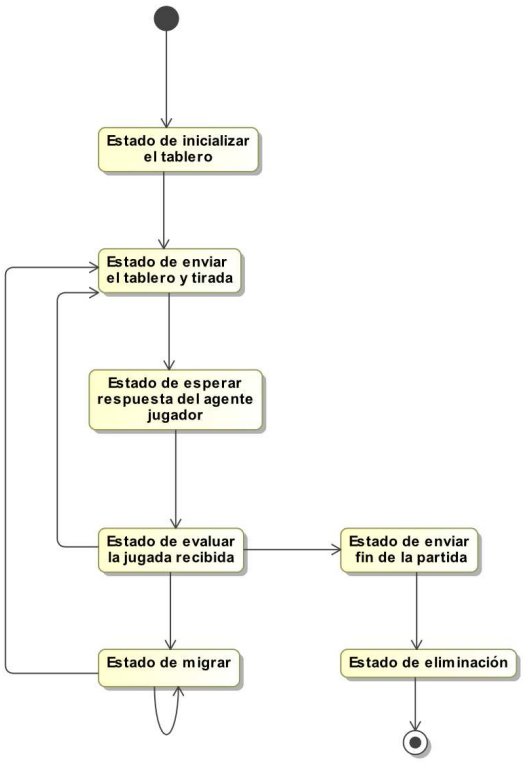

Figura 4.5: Diagrama de estados correspondiente al Agente Móvil Controlador

Estados del Agente Jugador

El Agente Jugador durante su vida pasa por diferentes estados, esto lo podemos observar en la figura 4.6. A continuación describiremos estos estados.

- **Estado de esperar información.**

  El AJ después de ser lanzado queda esperando recibir información proveniente del AMC. Mientras el AJ espera recibir esta información, queda bloqueado. Esto significa que el agente no realiza ninguna acción.

- **Estado de mostrar información al usuario.**

  El AJ al recibir información por parte del AMC entra en un nuevo estado. En este estado el AJ comunica al usuario la información obtenida. Para el Aj pueda mostrar esta información al usuario se utiliza un formato de texto o una ventana gráfica. De manera, que el usuario pueda ver el tablero de una de las dos formas.

- **Estado de esperar respuesta del usuario.**

  Posterior a mostrar la información al usuario el AJ se queda esperando una respuesta por parte del usuario. Esta espera se prolonga durante un tiempo determinado. Si pasa este tiempo el AJ continúa con su ejecución.

- **Estado de informar al Agente Móvil Controlador.** El AJ después de recibir el movimiento deseado por parte del usuario, lo comunica al AMC. El AMC se encarga de procesar este movimiento.

- **Estado de eliminación.** El AJ recibe el resultado del juego por parte del AMC. Al recibir esta información el AJ entiende que ha de morirse.

4.3.4. Diagramas de secuencia

A continuación se muestran los diferentes diagramas de secuencia según el estado del juego en el que nos encontremos. El primer diagrama de secuencia

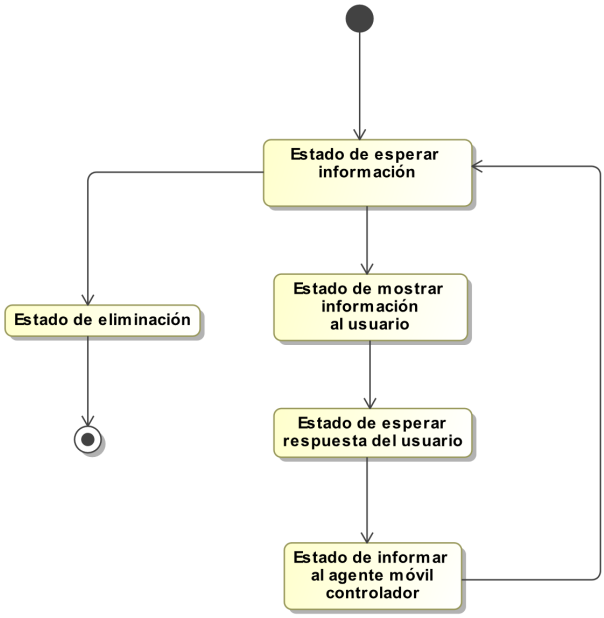

Figura 4.6: Diagrama de estados correspondiente al Agente Jugador

pertenece a las acciones que se realizan para comenzar el juego (figura 4.7). En un segundo diagrama (figura **??**) se muestra las acciones que se realizan para terminar el juego.

Diagrama de secuencia del inicio partida

Las acciones que se realizan para iniciar el juego son las siguientes:

1. El usuario modifica el fichero de configuraciones situado en el *host* donde se encuentra. Esta modificación consiste en cambiar el atributo del color de las fichas con el que desea jugar o el modo de vista del tablero.

2. El AMC para poder inicializar el tablero del juego ha de obtener el número de jugadores, el color de cada uno y la dirección de los *hosts*, donde están ejecutándose las plataforma que forman parte del juego. Esta información la extrae del fichero de configuraciones.

3. El AMC envía el estado del tablero y el número del dado al AJ.

4. El AJ muestra al usuario el estado del tablero y el número del dado.

5. El usuario comunica el movimiento que desea realizar al AJ.

6. El AJ envía el movimiento al AMC para que este último lo procese.

7. El AMC recibe un mensaje por parte del AJ con el movimiento que desea realizar el usuario. El movimiento que recibe el AMC lo evalúa para saber si es lícito. En caso de ser correcto el movimiento, el AMC modifica el estado del tablero.

8. El AMC después de haber realizado la modificación en el estado del tablero migrar a la siguiente plataforma. Si ha sido posible la migración se vuelven ha realizar las acciones desde la 2 hasta la 7.

Diagrama de secuencia del fin de partida

A continuación se comenta con mayor claridad las acciones necesarias para el terminar el juego. Este diagrama es en el caso de que dos jugadores formen el juego:

1. El Agente Móvil Controlador(AMC) envía el resultado final de la partida a todos los jugadores. El resultado final está compuesto por la puntuación del jugador en cuestión y del nombre del jugador ganador.

2. El Agente Jugador(AJ) de cada plataforma después de recibir el resultado del juego, lo muestra al usuario para que tenga conocimiento del mismo.

3. El AJ de cada agencia, después de mostra el resultado final al usuario, muere.

4. Por último el AMC del juego después de que se hayan eliminado todos los AJ, muere. Al morir el AMC finaliza el juego.

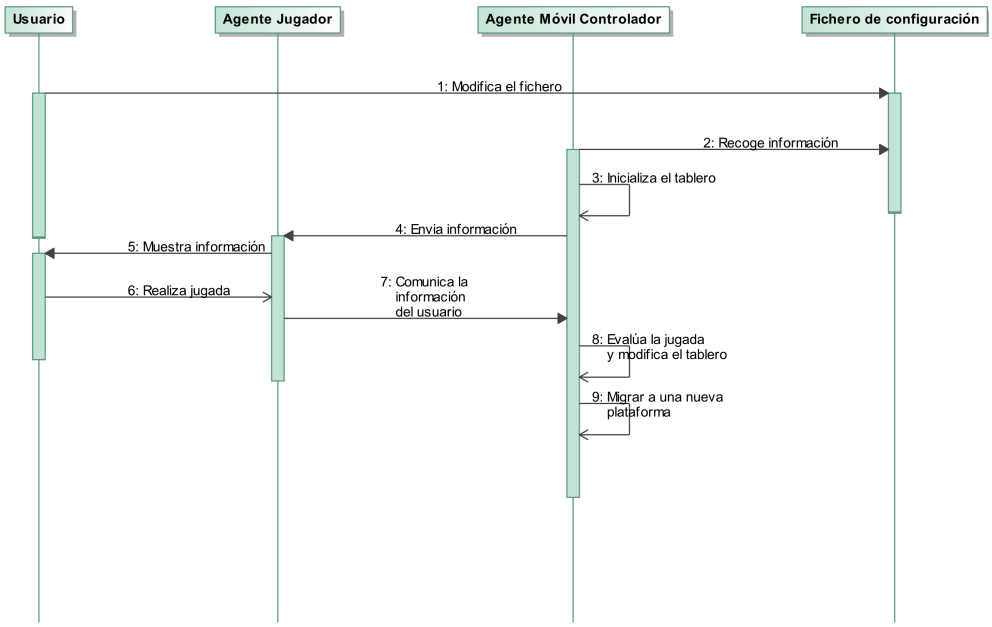

Figura 4.7: Diagrama de secuencia del inicio de una partida

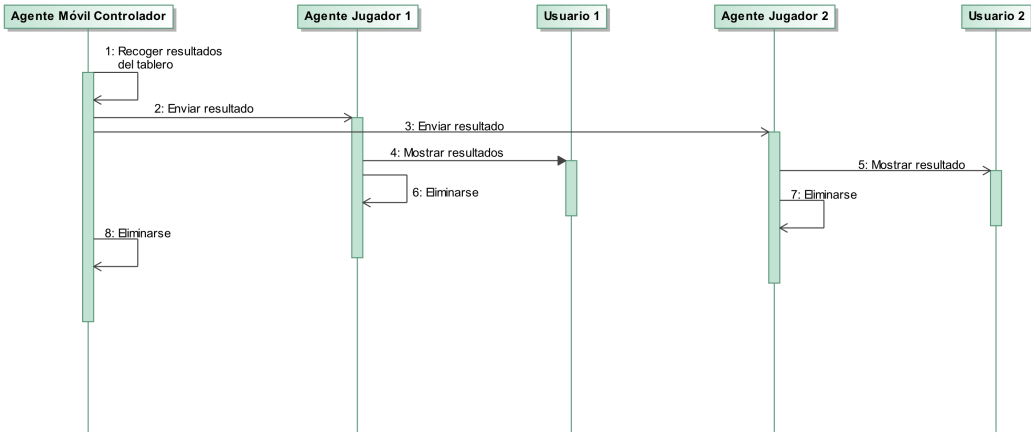

Figura 4.8: Diagrama de secuencia correspondiente al fin de una partida

4.3.5. Diagramas de clases

Diagrama de clases de BoardAgent

En la figura 4.9 se pueden ver todas las clases necesarias para la realización del AMC, recordamos que es el único agente que contiene el estado del tablero. Este diagrama se divide en dos grupos.

El primer grupo está formado por las aquellas clases que forman el esqueleto(*core*) de un juego. Estas clases son de gran utilidad para aquellos usuarios que quieren desarrollar un nuevo juego.

El segundo grupo está formado por las clases que implementan un juego en particular. En este caso, estas clases implementan el juego del Parchís.

A continuación explicaremos las clases que forman los dos grupos.

Clases principales del Agente Móvil Controlador de un juego.

- **BoardAgent:** Esta clase es abstracta y hereda los métodos de la clase Agent. La clase BoardAgent necesita una clase para que implemente los métodos abstractos y sea posible la implementación de un agente que contenga un juego. Depende del juego, la clase que implementa estos métodos varia, en nuestro caso se utiliza la clase *ParchisPlayerAgent*.

- **PlayAndSendBoardBehaviour:** En una clase interfaz donde están declaradas las acciones que se han de realizar antes de enviar el tablero al AJ. En el caso del Parchís es necesario lanzar un dado.

- **EvaluationAndModificationBoardBehaviour:** Esta es la clase que implementa el comportamiento del agente para realizar las acciones de evaluar un movimiento y modificar el tablero en caso necesario. Esta clase es abstracta y es interna a la clase *ParchisBoardAgent*.

- **Board:** Esta clase es la principal encargada de crear la estructura de datos que contiene el estado del tablero. Dependiendo del tipo de juego que se

implemente la estructura de datos varía. La clase *Board* contiene los tableros de los juegos que hayan sido implementados.

- **TakePlayers:** Esta clase es interna a la clase *Board*. Es la encargada de extraer la información necesaria del fichero de configuración.

Clases del Agente Móvil Controlador del Parchís.

- **ParchisBoardAgent:** Esta clase hereda de la clase *BoardAgent*. De manera que se pueden utilizar los métodos implementados en la clase *BoardAgent*. La clase *ParchisBoardAgent* tiene implementados los métodos necesarios para crear el AMC.

- **PlayAndSendParchisBoardBehaviour:** Esta clase implementa los métodos declarados en la interfaz *PlayAndSendBoardBehaviour*. La clase *PlayAndSendParchisBoardBehavioures* es interna a la clase *ParchisBoardAgent*, de manera, que solo la puede utilizar esta última clase. Esta clase es la encargada de implementar las acciones que realiza el AMC antes de enviar el tablero. En el caso en particular del Parchis, se realiza la tirada de un dado.

- **InformBehaviour:** Esta clase implementa la acción de comunicar al Agente Jugador del Parchís mediante mensajes ACL. De esta manera se tiene al Agente Jugador informado del estado del tablero en ese momento.

- **EvalAndModifParchisBehaviour:** Esta clase hereda de la clase *EvaluationAndModificationBoardBehaviour* e implementa el comportamiento del AMC encargado de evaluar el movimiento del usuario y modificar el estado del tablero. En el caso en particular del Parchís, a la hora de evaluar el movimiento, comprueba si la posición que informa el usuario contiene alguna ficha de su color. Otro aspecto a tener en cuenta en la evaluación del movimiento, es la posibilidad de poder mover la ficha seleccionada un número determinado de posiciones.

  En el caso de que el movimiento sea correcto, se ha de modificar el estado del tablero utilizando la clase *ParchisEvalEngine*. Para que la clase *Eva-*

lAndModifParchisBehaviour pueda modificar el tablero necesita una relación de utilidad con la clase *ParchisBoard*.

En caso de que el movimiento es incorrecto y se ha de informar al usuario es necesario la utilización de la clase *InformBehaviour*.

- **EvalEngine:** Esta un clase interfaz que declara los métodos necesarios para evaluar el movimiento del usuario.
- **ParchisEvalEngine:** Esta clase deriva hereda de la clase *EvalEngine*. La clase *ParchisEvalEngine* se encarga de implementar los métodos necesarios para la evaluación de los movimientos del usuario. Para hacer posible esta evaluación es necesario la implementación de las reglas del juego. En este caso en particular evalúa la veracidad del movimiento con las reglas del Parchís.

  La clase *ParchisEvalEngine* en caso de ser correcto el movimiento del usuario, modifica el estado del tablero.
- **Movement:** Esta clase contiene el movimiento que realiza el usuario. Para que el usuario realice un movimiento, debe introducir el identificado de la casilla donde se encuentra la ficha que desea mover.
- **ParchisBoard:** Esta clase tiene una relación de herencia con la clase *Board-Agent*. Como se observa en la figura 4.9, esta clase es utilizada por la clase *PlayAndSendParchisBoardBehaviour*, la clase *InformBehaviour* y la clase *EvalAndModifParchisBehaviour*.

  La clase *ParchisBoard* implementa la estructura de datos que forma el estado del tablero. Esta clase contiene los métodos necesarios para la modificación del estado del tablero.

Clases principales del Agente Jugador de un juego.

- **PlayerAgent:** Esta clase es abstracta y hereda de la clase Agent. La clase *PlayerAgent* necesita una clase para que implemente los métodos abstractos

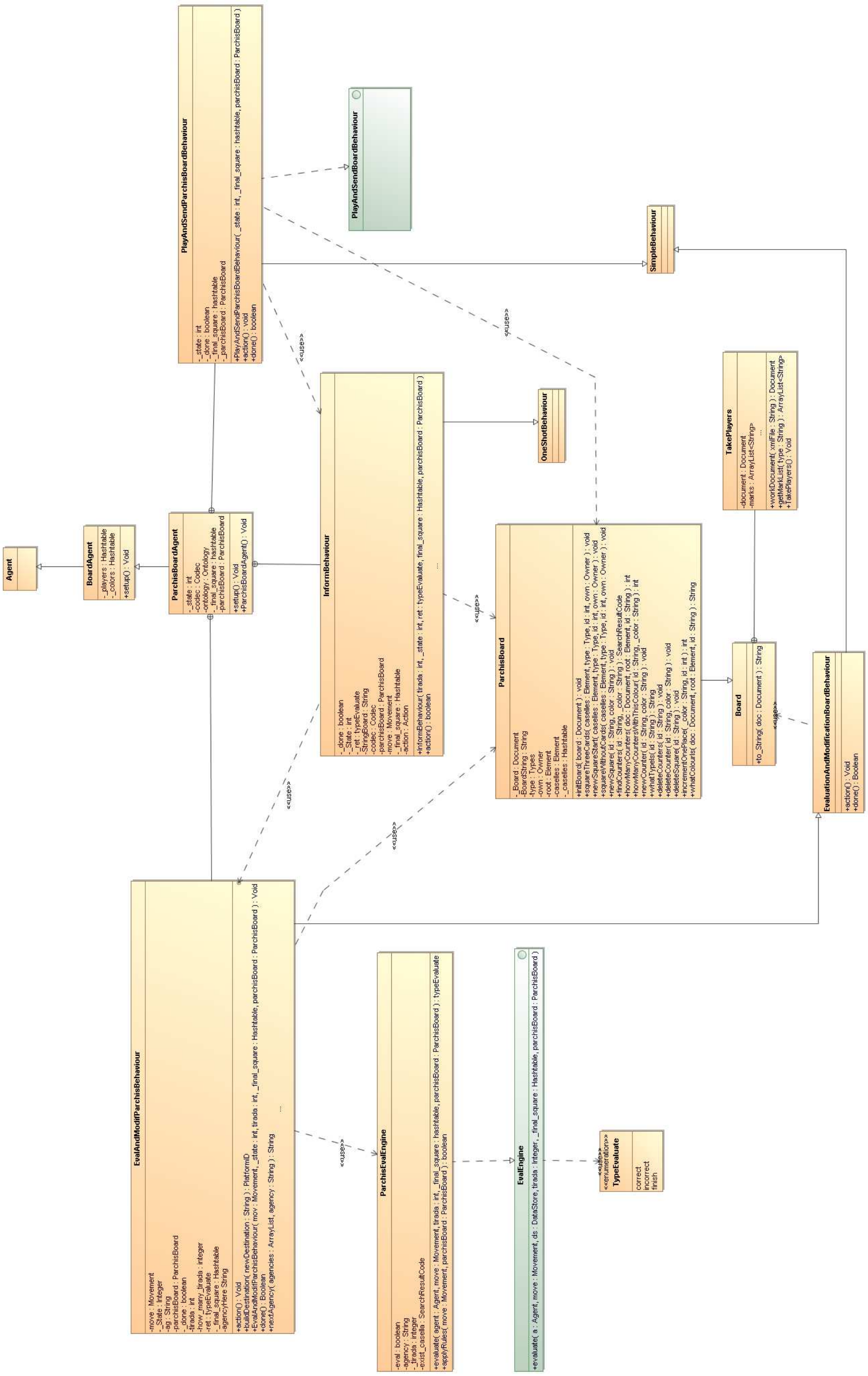

Figura 4.9: Diagrama de clase del agente móvil controlador

y se implemente un agente. La clase que implementa estos métodos depende del juego que queremos desarrollar, en este caso se utiliza la clase *ParchisPlayerAgent*.

- **BoardFormatter:** Esta es la clase encargada de mostrar el estado del tablero de la partida. El estado del tablero se puede mostrar de modo texto o gráfico, por ello implementa el método para mostrar el tablero. Este método es necesario para que el jugador pueda ver como se encuentra el tablero y elegir el movimiento que desea realizar.

- **WaitForBoardBehaviour:** Esta clase es abstracta y a su vez es interna a la clase *ParchisPlayerAgent*. La clase *WaitForBoardBehaviour* es se encarga de declarar los métodos necesarios para la implementación del AJ.

Clases del Agente Jugador del Parchís.

- **ParchisPlayerAgent:** Esta clase hereda de la clase *PlayerAgent* y es la encargada de implementar el AJ del Parchís. Como se observa en la figura 4.10 esta clase contiene las clases *WaitForBoardBehaviour* y *WaitForUserActionBehaviour*. Ésto se debe al hecho de que, estas clases, implementan tareas que realiza el AJ.

- **WaitForParchisBoardBehaviour:** Esta clase hereda de la clase *WaitForBoardBehaviour* e implementa una tarea del AJ. La tarea que implementa es la esperar información por parte del AMC del Parchís.

- **WaitForUserActionBehaviour:** Esta clase es interna a la clase *ParchisPlayerAgent* y se encarga de la implementación de un comportamiento del AJ. En este comportamiento, se muestra al usuario el estado del talbero y se espera un movimiento del usuario. Para poder mostrar al usuario el estado del tablero, se necesita la clase *BoardFormatter*.

- **TextBoardFormatter:** Esta clase hereda de la clase *BoardFormatter* e implementa los métodos necesario para la visualización del estado del tablero de modo de texto.

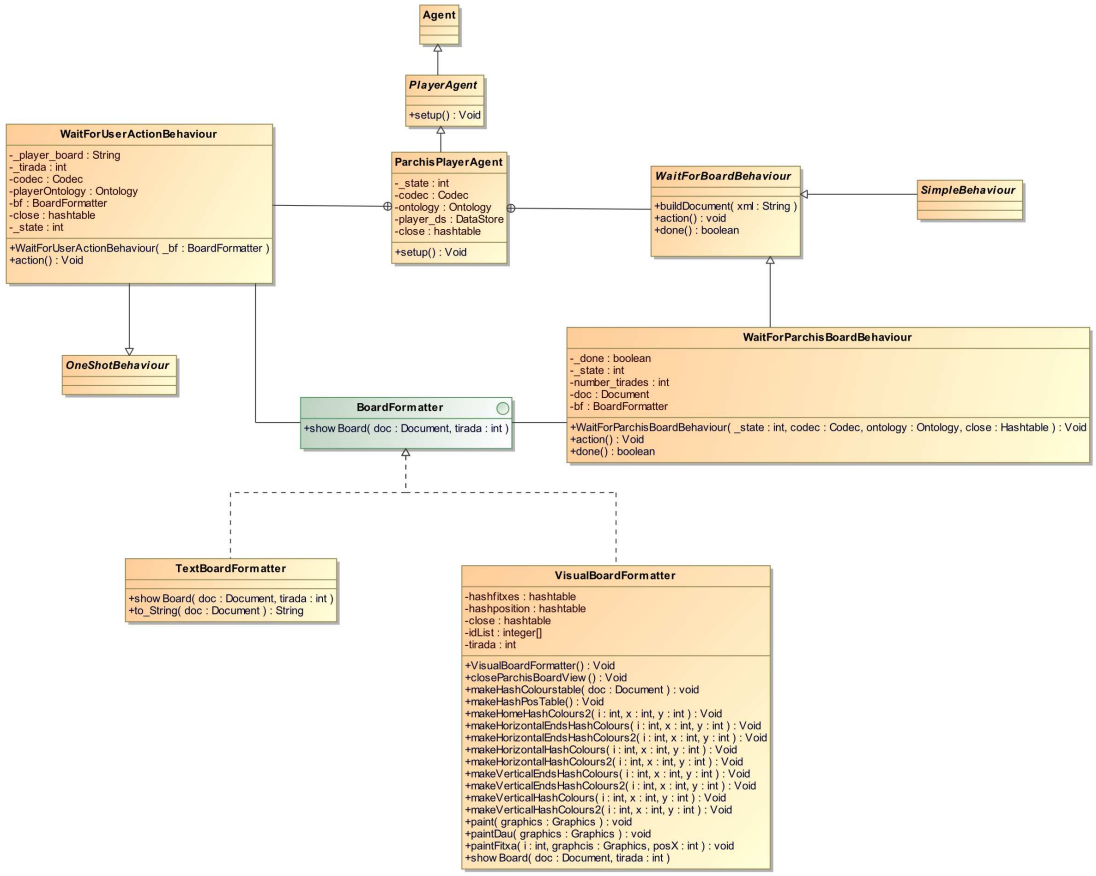

Figura 4.10: Diagrama de clase del agente jugador

- **VisualBoardFormatter:** Esta clase al igual que la anterior hereda de la clase *BoardFormatter* e implementa los métodos necesarios para la visualización del estado del tablero de modo gráfico. El contenido de la clase VisualBoardFormatter se explica en el apartado de *Interfaz* en el capítulo de *Implementación*.

4.4. Ontologías

Las ontologías, como hemos visto en el capítulo 2, son un mecanismo para estructurar la información. En este proyecto las ontologías nos permiten dar una de-

Ontología	parchis-ontology		
Frame	movement-description		
Parámetros	**Descripción**	**Presencia**	**Tipo**
oldPosition	La posición donde se encuentra la ficha	OBLIGATORIO	Integer

Cuadro 4.1: Concepto. Características de un movimiento

terminada forma y orden a la información que queremos transmitir entre agentes. Para poder transmitir cierta información, como hemos visto anteriormente, se hace a través del envío de mensajes ACL. Hay que resaltar que el uso de ontologías no está ligado al uso de mensajes de tipo ACL.

En el juego del Parchís se necesita definir una ontología de permita especificar las estructuras de datos y las determinadas acciones. Las estructuras de datos son utilizadas durante el juego para incluir en un mensaje ACL todas los datos necesarios para que los agentes hagan diferentes tareas. Las acciones se utilizan para indicar cual es la tarea asociada al mensaje recibido.

A continuación se explica la ontología que ha de crearse para el intercambio de mensajes entre el Agente Móvil Controlador del Parchís y el Agente Jugador del Parchís.

La ontología está formada por tres conceptos de datos. Cada concepto se utiliza para el envío de diferente información entre los agentes.

Se ha de crear un concepto que contenga los campos necesarios para transmitir el movimiento del usuario. Este campo (ver tabla 4.1) en nuestro caso es oldPosition.

- oldPosition. Contiene el identificador de la casilla donde se encuentra la ficha que el usuario desea mover.

Hemos de crear un segundo concepto que ha de servir para que el AMC transmita al AJ, sobre la información necesario para que el usuario realice un movimiento. Los campos que forman este concepto son los que se muestran en la tabla 4.2. Es necesario el envío del estado del tablero y del número del dado para

Ontología	parchis-ontology		
Frame	parchis-description		
Parámetros	**Descripción**	**Presencia**	**Tipo**
dice	El número que ha salido en la tirada del dado	OBLIGATORIO	Integer
board	El tablero	OBLIGATORIO	String
throws	El número de tiradas que lleva realizadas	OBLIGATORIO	Integer
playerColour	El color del jugador	OBLIGATORIO	String

Cuadro 4.2: Concepto. Características de la información que recibe el usuario

Ontología	parchis-ontology		
Frame	end-of-game-description		
Parámetros	**Descripción**	**Presencia**	**Tipo**
winner	La dirección de la plataforma ganadora	OBLIGATORIO	String
points	La puntuación del jugador	OBLIGATORIO	Integer

Cuadro 4.3: Concepto. Características del resultado final

que el usuario pueda realizar un movimiento. El envío del color del jugador sirve para que el jugador no tenga que recordar el color que tiene. Por último se ha de indicar el número de movimientos que lleva incorrectos porque hay un número limitado de tres.

La ontología por último ha de tener un último concepto creado que sirve para que el AMC comunique al usuario el resultado final del juego. En este caso son necesarios dos campos: *winner* y *points*(ver tabla 4.3). Para informar del resultado final de juego se ha de comunicar el jugador ganador y la puntuación que ha conseguido cada usuario.

Con el uso de esta ontología se fija la información que ha de enviarse entre el agente AMC y AJ.

Capítulo 5

Implementación del caso práctico

Una vez analizados los requisitos y haber creado el diseño del juego del Parchís, el siguiente paso que realizaremos será la implementación de éste. En este capítulo veremos la implementación del caso práctico del juego del Parhís, el cual ha sido planteado en los capítulos anteriores. Comenzaremos implementando un agente sencillo de manera que podamos introducir métodos complejos, que utilizaremos en la implementación de los agentes que formarán parte del juego. Explicaremos como hemos implementado los agentes que forman el juego y la ontología que ha sido necesaria.

5.1. Desarrollo de una agente prueba

En este apartado se implementa un agente para una plataforma JADE y se explican las diferentes tareas que puede realizar.

Un agente JADE es una instancia de una clase Java que extiende de la clase `jade.core.Agent`. De manera que se consigue las interacciones básicas con la plataforma de agentes y un conjunto básico de métodos que implementa las tareas del agente.

Recordamos que un *behaviour* es el comportamiento de un agente y está formado por diferentes tareas. Un agente contiene una lista de *behaviours* que se han ido añadiendo. Para poder ejecutar un *behaviour* es necesario que un planificador

seleccione aquel *behaviour* que lo puede hacer.

A continuación se comentan los métodos básicos de la clase jade.core.Agent que se utilizan en el proyecto. Los métodos que no se citan a continuación se pueden observar en [JADE07].

- `setup()`: En este método se le especifica al agente las acciones que debe realizar y se utiliza para añadir *behaviours* a la lista de *behaviours*, al menos uno por agente. Una vez finalizada la ejecución de este método, el agente ejecuta el primer *behaviour* de la cola de *behaviours* activos.

- `doMove()`: Este método permite al agente migrar a una plataforma remota.

- `addBehaviour()`: Este método añade a la cola de *behaviours*, que contiene el agente, un nuevo *behaviour*.

- `blockingReceive()`: Este método se utiliza para que el agente bloquee su hilo de ejecución hasta que reciba un mensaje ACL.

- `doDelete()`: Este método sirve para destruir el agente.

- `send()`: Este método sirve para el envío de mensajes ACL. Se debe comunicar previo al uso de este método, el destinatario del mensaje.

- `receive()`: Este método recibe un mensaje y lo introduce en una cola de mensajes. En el caso de estar vacía la cola, devuelve el valor de null.

Después de haber comentado los métodos que son necesarios en la implementación de agentes, vamos a pasar a comentar los diferentes *behaviours* que puede realizar un agente. JADE proporciona una serie de clases base para implementar *behaviours* para facilitar la implementación de métodos básicos de los *behaviours*.

Todos los *behaviours* implementan el método `action()` que se encarga de la ejecución de un *behaviour*. Este método es el primero que se ejecuta cuando el planificador de tareas da permiso.

Existe el método `done()` encargado de ir comprobando si el *behaviour* se ha completado con éxito. Si es así, el *behaviour* se elimina de la cola, sino se mantendrá a la espera hasta que el planificador le vuelva a dar permiso para continuar su ejecución.

Existen dos tipos de *behaviours* principales:

- `SimpleBehaviour`: Esta clase se encarga de manipular behaviours formado por una sola tarea.
  - `Clase OneShotBehaviour` : Esta clase manipula behaviours que estan formados por una única tarea y debe ser ejecutada solamente una vez. Este behaviour no puede detener la tarea que contiene. De esta manera su método `done()` devuelve siempre true.
  - `Clase CyclicBehaviour` : Manipula *behaviours* que están formado por tareas que se ejecutan ciclicamente mientras el agente exista. Su método `done()` devolverá siempre false.
  - `Clase WakerBehaviour` : Esta clase planifica la ejecución de la clase `OneShotBehaviour`.
  - `Clase TickerBehaviour`: Estas clase manipula behaviours derivados de la clase `OneShotBehaviour`, de manera que se ejecuten de forma periódica.
- `Clase CompositeBehaviour` : Manipula *behaviours* que están compuestas por otros *behaviours*. De esta manera las operaciones que ejecuta el *behaviour* padre no están definidas dentro de el, sino dentro de sus *behavours* hijos. El *behaviour* padre solo tiene en cuenta que los *behaviours* se ejecuten mediante una política dada.
  - `Clase SequentialBehaviour` : Es una clase que hereda de la clase `CompositeBehaviour` que ejecuta sus behavious secuencialmente y termina cuando todos han terminado. Se utiliza cuando un *behaviour* complejo se puede expresar como una secuencia de tares a realizar.

- `Clase ParallelBehaviour` : Es una que hereda de la clase `clase CompositeBehaviour` que ejecuta sus tareas concurrentemente y termina cuando se completan todas estas tareas, o cuando se encuentra una condición particular de terminación en alguna de ellas.

A continuación explicaremos la ontología que ha sido necesaria para el intercambio de mensajes entre el Agente Móvil Controlador y el Agente Jugador

5.2. Ontología del Parchís

La ontología del Parchís se ha implementado siguiendo la propuesta del apartado 4.4. Está formada por ocho clases que implementan las interfaces que proporciona JADE para la creación de ontologías.

En JADE las ontologías se realizan a partir de clases que implementan las interfaces: `jade.content.AgentAction`, `jade.content.Concept`, `jade.domain.JA` y `jade.content.onto.ontology` ([GCDC04]).

La ontología que se ha desarrollado está formada por cuatro partes: el vocabulario, los conceptos, las acciones y la clase que une todas las partes. A continuación se muestran las partes implementadas (figura):

- **Vocabulario**: El vocabulario está implementado mediante la clase *ParchisVocabulary* en forma de variables estáticas. Esta clase implementa la interfaz `jade.domain.JADEAgentManagement.JADEAgentManagementVocabulary`.
- **Conceptos**: Se han implementado los conceptos propuestos en las tablas del apartado 4.4, donde se define como se estructura la información que contiene la ontología. Los conceptos se representan con clases que implementan la interfaz `jade.content.Concept` del mensaje ACL. En nuestro caso las clases que implementan los diferentes conceptos son: `EndofGameDescription`, `MovementDescription` y `ParchisDescription`
- **Acciones**: Indican las tareas que debe realizar el agente que recibe el mensaje. En nuestro caso la acción que se realiza es simplemente la de extraer

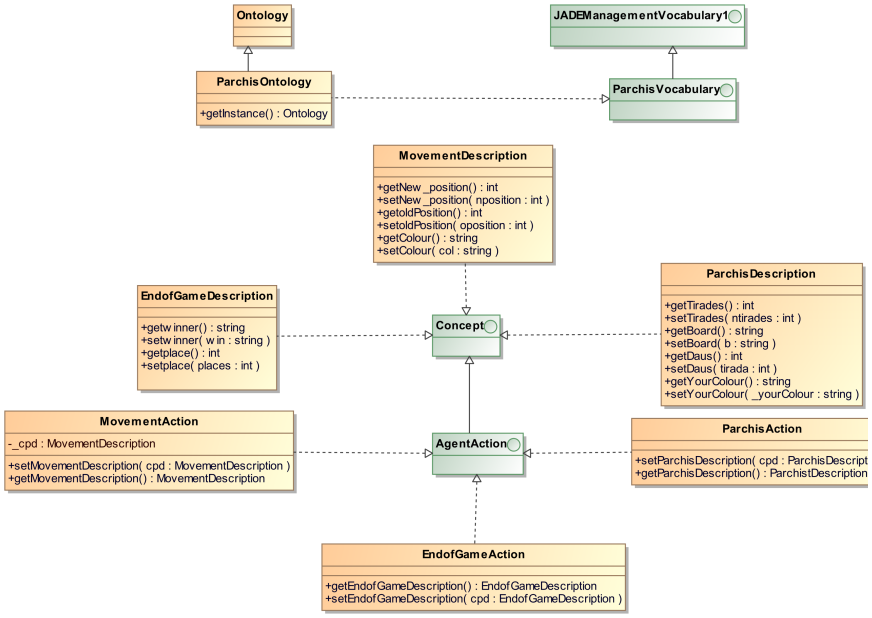

Figura 5.1: Ontologia del Parchís

la información de un concepto determinado. En nuestro caso se utilizan: `EndofGameAction`, `MovementAction` y `ParchisAction`

- **Ontología**: La ontología, propiamente dicha, se encuentra implementada en la clase `ParchisOntology`. Esta clase se encarga de definir la ontología utilizando las clases que implementan el vocabulario, los conceptos y las acciones.

.

Con esta implementación nos podemos asegurar que los mensajes siguen esta ontología y no incluyen ninguna información que no este relacionada con la ontología. Esto nos asegura que los agentes podrán intercambiar información semánticamente conocida por ellos.

Después de comentar los diferentes tipos de *behaviours*, pasaremos a comentar como hemos implementado las diferentes tareas del Agente Móvil Controlador.

5.3. Tareas del Agente Móvil Controlador

La funcionalidad del Agente Móvil Controlador(AMC) está formado por diferentes tareas como hemos visto en el capítulo de diseño. Estas tareas se implementan mediante diferentes *behaviours*.

La primera tarea que realiza el AMC es la recogida de información para poder comenzar el juego. Esta información la obtiene del fichero de configuraciones en XML. A partir de este fichero crea una lista con los turnos de la partida, recoge los nombres de los usuarios y las direcciones de los *hosts* donde están situados. A continuación comentaremos la construcción del tablero del Parchís.

5.3.1. Generación del tablero de juego

El tablero, como se ha comentado en capítulos anteriores es el estado físico del juego en un momento dado. El estado del tablero se representa en formato XML. El XML del tablero ha ido evolucionando dependiendo de la información necesaria para el desarrollo de nuestra aplicación.

Para la implementación del tablero se ha utilizado la clase `org.jdom.Document` del *framework de JDOM*.

La utilización de este framework y no de otros como es DOM, SAX, o a la extensión JAVAX.XML se debe a que el uso de JDOM es más sencillo y ofrece grandes ventajas para el manejo de fichero XML para JAVA ([JDOM]). La representación de XML del tablero se carga en memoria.

A continuación se explica detalladamente el contenido del XML del tablero formado por cinco niveles de elementos.

- `<board>`: Es el elemento raíz del XML y no contiene ningún atributo.

- `<casillas>`: Este elemento es el contenido del elemento `<board>` y al igual que éste no contiene atributos. Este elemento se crea para estructurar mejor el documento XML de manera que contenga dentro todas las casillas posibles del tablero del Parchís.

- `<casilla>`: El elemento `<casillas>` está formado por elementos elemento `<casilla>`. Como explicaremos más adelante el tablero está formado por tantos elementos `<casilla>`, como casillas que contengan fichas. De esta manera ahorramos espacio en memoria y por tanto tiempo en la migración del tablero. El elemento `<casilla>` contiene diferentes atributos.
  - id: Para poder diferenciar las diferentes casillas se utiliza este argumento.
  - type: Este argumento es necesario para conocer de qué clase es la casilla. Existen cuatro tipos de casillas:
    - *home*: Es la casilla donde están situadas inicialmente las fichas dependiendo del color. Existen cuatro casillas con de esta clase.
    - *end*: Es la casilla final. Existen cuatro casillas, es decir, una casilla para cada color.
    - *safe*: Las casillas *safe* son las que forman parte de esta clase. En este tipo casilla no se pueden eliminar fichas y hay ocho casillas en el tablero.
    - *normal*: De este tipo de casillas forman parte el resto de casillas que acaban formando el total del tablero.
  - owner: Este atributo sirve para conocer el propietario de la casilla de la clase *start*.
- `<fichas>`: Este elemento se utiliza del mismo modo que el elemento `<casillas>`, pero en este caso para estructurar el conjunto de fichas que puede contener una casilla.
- `<ficha>`: El elemento `<fichas>` está formado por elementos `<ficha>`. Este elemento tiene el atributo *color* que se utiliza para mencionar el propietario al que pertenece.

Como se ha podido ver la representación XML que implementa el tablero puede llegar a tener un tamaño enorme debido al gran número de casillas que existen en un tablero físico. La solución a este problema fué incluir tan solo aquellas

casillas que fueran de una clase diferente a la *normal*. A medida que transcurre el juego se van añadiendo o eliminando las casillas de tipo *normal* que contiene fichas.

El objeto `org.jdom.Document` que representa el tablero está formado por un conjunto de objetos `org.jdom.Element` que representan cada uno de los elementos de los que está compuesto la representación XML del tablero.

A posteriori de la creación del tablero se ha de añadir a la cola de tareas del AMC, el `PlayAndSendParchisBoardBeahviour` que se explica a continuación.

5.3.2. Realizar la tirada

En este apartado se comenta las acciones que realiza el AMC cada vez que el agente se ejecuta sobre las plataformas donde migra. La clase que implementa esta tarea es `PlayAndSendParchisBoardBeahviour`.

En el caso en particular del Parchís el AMC extrae un número aleatorio del uno al seis de manera que se simula la tirada del dado. Por último se pasa está información a la tarea `InformBehaviour` para que la envíe al Agente Jugador (AJ).

5.3.3. Evaluar, modificar el tablero y migrar

En este apartado se comenta la implementación de aquellas acciones que realiza el AMC después de recibir la jugada por parte del AJ.

La tarea que implementa las acciones de evaluar, modificar el tablero y migrar a otra plataforma es *EvalAndModifParchisBehaviour*.

Para la evaluación de la jugada recibida del AJ se utiliza una instancia de la clase *ParchisEvalEngine*. Esta clase contiene todos aquellos métodos que hacen posible la evaluación de la jugada.

En la modificación del tablero inteviene el objeto org.jdom.Document que representa el tablero como una tabla *Hash*. Esta tabla contiene todas aquellas casillas que contiene fichas. A la vez esta tabla hace referencia al documento de manera

que toda aquella información que insertemos en esta tabla queda reflejada en el documento que implementa el tablero. Todas las acciones que se necesitan a la hora de modificar el tablero (inserción de una ficha, creación de una nueva casilla o eliminación en el caso de ser del tipo *normal* y no contener fichas), se realizan directamente sobre esta tabla de manera que se consiga mayor velocidad.

Después de realizar la modificación del tablero el agente ha de moverse a la siguiente plataforma dependiendo de la lista de turnos. Para realizar esta migración es necesario la llamada al método doMove() comentado en el apartado *Desarrollo de un agente base*.

Para la implementación del *behaviour* que contiene las tareas comentadas anteriormente es necesario la utilización del `SimpleBehaviour`. Se utiliza este comportamiento base porque es el que mayor se adapta a la implementación de una máquina de estados para controlar todas las acciones que realiza.

5.3.4. Informar al Agente Jugador

El último comportamiento que tiene el AMC es el que implementa la acción de comunicar al AJ cierta información. Este comportamiento se implementa en la clase `InformBehaviour` y hereda de la clase `OneShotBehaviour`, comentada en el apartado *Desarrollo de un agente base*. La utilización de este *behaviour* base es por el hecho de que solo tiene una tarea y el *behaviour* no la puede bloquear. Esto nos es de gran utilidad porque la acción de *Informar al AJ* sólo la queremos realizar una vez, de manera que al acabar la acción se elimine este comportamiento de la lista de *behaviours* que contiene el AMC.

Para poder comunicar información del AMC al AJ es necesario el intercambio de mensajes ACL. A continuación se comenta los diferentes mensajes que se intercambian el AMC y AJ.

Mensaje con el tablero y el número del dado

En el envío de esta información se utiliza la ontología *parchis-ontology*. Dentro de esta ontología es necesario crear una clase que implemente la interfaz

`jade.content.concept`, de manera, que se implementen aquellos métodos necesarios para la modificación de las variables del concepto. La clase `ParchisDescrition` implementa esta interfaz donde se especifican el vocabulario y los métodos accesores al vocabulario. El vocabulario para este mensaje está formado por: el tablero, el color del usuario, el número de tiradas y el número del dado.

Para poder especificar el vocabulario dentro de una ontología hay que tener en cuenta de que tipos son. Los únicos tipos que pueden contener el vocabulario de la ontología son los básicos: *string, integer, float, arrayList*, etc. Para especificar el valor de elemento tablero del vocabulario debemos convertir el objeto `org.jdom.Document` a una cadena de caracteres. Para llevar a cabo este proceso se realiza una instancia de la clase `XMLOutputter` que forma parte del framework de JDOM y contiene el método de transformar un objeto `org.jdom.Document` en el formato XML a cadena de caracteres.

El tipo del mensaje ACL que contiene esta ontología es *INFORM*.

La jugada es incorrecta

Después de que el AMC evalúe la jugada y compruebe que es incorrecta ha de comunicarlo al AJ.

En el momento en que el AMC comprueba que la jugada es incorrecta vuelve a enviar el tablero y en número del dado, de manera que el usuario pueda volver hacer un nueva jugada. Esta acción de volver a enviar el tablero y el número del dado, como vimos anteriormente, se realiza un máximo de tres veces, de forma que si no se ha realizado una jugada correcta en estas oportunidades el agente migra a la siguiente plataforma.

La construcción del mensaje para informar de que la jugada es la misma que en el caso de enviar tablero y número de dados. En este caso el tipo del mensaje es *FAILURE*. Esta información ayuda al AJ a conocer el motivo por el cual se ha vuelto a enviar el tablero.

Resultado final del juego

Para el envío de esta información es necesario la utilización de la ontología *parchis-ontology*. Dentro de esta ontología es necesario que se implemente la interfaz `jade.content.concept`. La clase `EndofgameDescription` implementa esta interfaza donde se especifican el vocabulario y los métodos accesores al vocabulario. El vocabulario para este mensaje está formado por: el nombre del usuario ganador y la puntuación que ha conseguido cada jugador.

El tipo del mensaje que comunica el resultado final de juegos es *CONFIRM* . Esta información sirve para que el AJ conozca que ha llegado el fin de la partida y que se ha de eliminar. Como hemos podido ver en el capitulo de diseño, estos mensajes los reciben todos los participantes en el juego.

5.4. Tareas del Agente Jugador

La funcionalidad del AJ está dividida en diferentes tareas, como hemos visto en el capítulo de diseño, implementados mediante diferentes *behaviours* del agente. A continuación se comenta las distintas tareas que realiza el AJ.

5.4.1. Esperar información del Agente Móvil Controlador

El agente AJ inicialmente queda esperando algún mensaje proveniente del AMC. La clase que implementa este *behaviour* es `WaitForParchisBoardBehaviour`.

La primera acción que realiza el AJ es conocer el tipo del mensaje que ha recibido. Para conocer de que clase es el mensajes se utiliza el método `getPerformative()`. En este caso existen tres clases de mensajes, dependiendo de la información se quiere comunicar: *INFORM*, *FAILURE* Y *CONFIRM*.

- *INFORM*: El AJ al recibir este tipo de mensaje ha de conocer la debida ontología para poder extraer la información contenida en el mensaje. En este caso es necesario conocer la ontología *parchis-ontology* . De esta ontología obtendremos el objeto `ParchisDescription` de tipo

`jade.content.concept` donde extraeremos la información del tablero, el número de tiradas, el color y el número del dado.

Después de haber recibido el tablero y el número del dado se le ha de comunicar al usuario. El tablero inicialmente está en formato cadena de caracteres y es necesario crear un objeto `org.jdom.Document`. De esta manera es más fácil extraer la información que contiene.

- *FAILURE*: Al igual que en el caso anterior el AJ ha de conocer la ontología *parchis-ontology* . De esta ontología obtendremos el objeto `ParchisDescription` de tipo `jade.content.concept` donde extraeremos la información del tablero, el número de tiradas, el color y el número del dado.

  El AJ al recibir un mensaje de esta clase sabe que ya ha proporcionado una jugada anteriormente y ha sido incorrecta. Este hecho hace que comunique al usuario mediante la salida estándar, las oportunidades que le restan. Al igual que en caso anterior el AJ recibe el tablero en formato cadena de caracteres y es necesario crear un objeto `org.jdom.Document`. De esta manera es más fácil extraer la información que contiene.

- *CONFIRM*: Del mismo modo que en los casos anteriores, el AJ al conocer de qué tipo es el mensaje recibido ha de conocer tanto la ontología como el concepto. De esta manera extrae la información del mensaje. En este caso la ontología es *parchis-ontology* con la clase *EndofgameDescription* que implementa la interfaz `jade.content.concept`. Donde se especifican el vocabulario y los métodos accesores al vocabulario. El vocabulario para este mensaje está formado por: el nombre del usuario ganador y la puntuación que ha conseguido cada jugador. El AJ comunica esta información por la salida estándar al usuario y realiza la operación de eliminarse.

5.4.2. Mostrar el tablero

Como se ha podido ver en el apartado anterior cuando el mensaje recibido era de la clase *FAILURE* o *INFORM* era necesario mostrar el tablero al usuario. Para mostrar el tablero al usuario es necesario recoger del fichero de configuración el modo en el que lo desea visualizar. A continuación se comentan los dos modos que existen.

- Texto: El tablero y el número de tiradas se muestra mediante la clase . `TextBoardFormatter`. Esta clase muestra por la salida estándar el tablero en formato de cadena de caracteres y el número del dado. Este modo de observar el tablero es muy confuso debido a la gran información que contiene, pero es más rápido de mostrar al usuario.

- Visual: En este caso el tablero y el número de tiradas se muestra con la clase `VisualBoardFormatter`. Esta clase muestra el tablero y el número del dado en una ventana gráfica. Ésto sirve al usuario para poder ver con mayor facilidad el estado del juego y decidir que jugada desea realizar. En el apartado *Interfaz gráfica* se explica como se ha implementado este modo de visualizar el tablero.

5.4.3. Recoger y comunicar la jugada del usuario

El AJ después mostrar el tablero y el número de la tirada al usuario ha de recibe la jugada por parte del usuario y comunicarla al AMC. Estas acciones están implementadas en la clase *WaitForUserActionBehaviour* que deriva del comportamiento base `OneShotBehaviour`.

El AJ, mediante la entrada estándar, espera la jugada del usuario y una vez realizada, éste le comunica el movimiento al AMC mediante un mensaje ACL. Este mensaje ACL contiene la ontología *parchis-ontology* con la clase *Movement-Description* que implementa la interfaz `jade.content.concept`. Donde se especifican el vocabulario y los métodos accesores al vocabulario. El vocabulario para este mensaje el identificador de la casilla donde está situada la ficha a mover.

5.5. Interfaz gráfica

Como se ha comentado anteriormente el usuario puede elegir entre dos formatos de visualizar el tablero. En este apartado se comenta como se ha implementado el formato *visual*. La interfaz gráfica se ha desarrollado mediante el paquete `java.awt`.

La clase que hemos utilizado ha sido `java.awt.Frame` para la colocación de todos los elementos que forman el tablero.

Inicialmente se ha creado una tabla *Hash* que contiene todas las posiciones de las fichas en cada casilla. Utilizando esta tabla de posiciones junto con la tabla *Hash* formada por los campos: identificador de la casilla y número de fichas que contiene, es posible dibujar las fichas en la posición deseada.

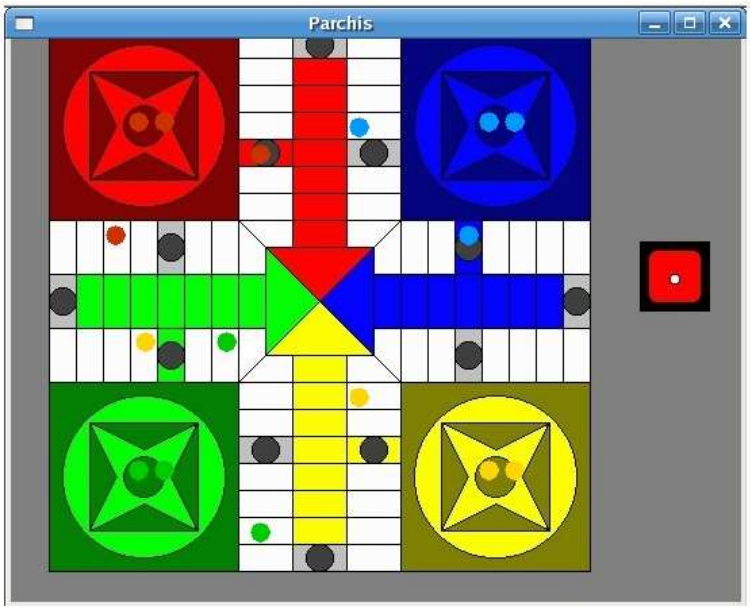

Figura 5.2: Interfaz del tablero del Parchís

La interfaz, como se muestra en la figura la figura 5.2, está formado por las imágenes de un tablero y un dado. Para dibujar las fichas y el número en el dado se utilizan objetos de tipo `java.awt.geom.Ellipse2D`.

Capítulo 6

Pruebas

En este capítulo veremos las diferentes situaciones en las que se ha sometido la aplicación para la verificar que el resultado obtenido era el esperado. Hay que comentar que estas pruebas se han ido realizando a medida que se implementaba el juego.

A continuación se muestran las posibles situaciones donde se ha probado el juego: *Testbed*

6.1. Testbed

El primer Testbed que hemos realizado verifica que nuestra aplicación cuando hay problemas de migración es correcta.

6.1.1. Testbed de migración

- Una posible situación puede ser cuando el Agente Móvil Controlador (AMC) intenta migrar a una plataforma que no está ejecutándose. Esta plataforma puede haber estado ejecutándose hasta que por razones externas, esta ejecución haya terminado. La plataforma JADE contiene un protocolo para verificar si la migración es posible. Los agentes AMS de las dos plataformas han de intercambiar información. Si el intercambio no ha sido posible, vuelve a intentar este intercambio de información durante un tiempo determinado. El

AMC al conocer que no es posible la migración a esta plataforma, intenta migrar a la siguiente plataforma.

- Otra posible situación es cuando la migración entre las diferentes plataformas es correcta. En este caso se ha de comprobar el cumplimiento de todas las reglas del Parchís. Se ha conseguido probar todas las acciones posibles entre fichas, como pueden ser: matar ficha de otro color, hacer puente entre fichas del mismo color, que no se pueda saltar un puente, que se sitúen dos fichas de distinto color haciendo puente en una casilla de tipo *safe*, que no se pueda salir de la casa si no sale un cinco en el dado, etc.

El resultado de estas pruebas ha sido satisfactorio.

A continuación vamos a comentar los Tesbed que hemos realizado para verificar el correcto funcionamiento del juego durante la partida.

6.1.2. Testbed relacionados con el juego

Los Testbed que se comentan posteriormente parten de la misma base, donde el usuario que juega ha decidido visualizar el tablero mediante una ventana gráfica. Esta manera ayuda a comprender mejor sobre la transmisión de información entre actores del sistema.

- **Movimiento correcto**

  El AMC llega a una plataforma y el usuario introduce un movimiento. En este caso el usuario desea move la ficha que hay en la casilla 19. Hay que tener en cuenta que el usuario está viendo el estado del tablero.

```
INFO:----------------------------------------------------

Agent container Main-Container@JADE-IMTP://ccd-pr3 is ready.
------------------------------------------------------------
Recuerda que tu color es blue

Posibilidades de hacer una jugada correcta: 3
```

```
Introduce el identificador de una casilla: 19

BoardAgent go to ccd-pr4
```

- **Movimiento incorrecto**

  El usuario ha introducido un movimiento incorrecto. Cuando el AMC evalúa el movimiento y conoce que es incorrecto se lo comunica el usuario. El usuario tiene tres posibilidades de introducir un movimiento correcto.

```
INFO:----------------------------------------------------
Agent container Main-Container@JADE-IMTP://ccd-pr3 is ready.
------------------------------------------------------------
Recuerda que tu color es blue
Posibilidades de hacer una jugada correcta: 3

Introduce el identificador de una casilla: 19
El movimiento introducido es incorrecto
Posibilidades de hacer una jugada correcta: 2
Introduce el identificador de la casilla: 22
```

- **Finalización de la partida.**

  Un usuario consigue llegar con sus cuatro fichas a la casilla final. El AMC comunica a cada jugador, el ganador y la puntuación que ha conseguido. A continuación se muestra la información que recibe un usuario que no es ganador y tiene tres fichas en su casilla final. Después de comunicar el resultado final, los agentes mueren.

```
Introduce el identificador de una casilla: 206
 El jugador ganador es: ccd-pr3
 La puntuación obtenida es: 3

AJ muere en paz

AMC muere en paz.
```

Para acabar este capítulo de pruebas hemos simulado una última situación donde una plataforma que no contiene el AMC deja de ejecutarse. El usuario ha de volver a ejecutar la plataforma JADE y lanzar de nuevo el agente AJ. Si durante este proceso el AMC no ha intentado migrar ha esta plataforma, porque se estaba ejecutando en otra plataforma jugadora, el juego continúa. El juego puede continuar porque cuando el AMC migra, lleva con él su código y el estado de sus variables. En cambio, si la plataforma en la que se encuentra el AMC deja de ejecutarse el juego termina repentinamente.

Capítulo 7

Conclusiones

En este proyecto se ha diseñado y desarrollado una aplicación basada en el paradigma de los agentes móviles.

Este proyecto ha demostrado que es posible la utilización de esta tecnología en aplicaciones con finalidad lúdica.

Al principio del proyecto nos hemos propuesto una serie de objetivos, los cuales han sido logrados. A continuación comentaremos cada uno de ellos.

Un objetivo propuesto fue el estudiar la tecnología de agentes móviles y la plataforma donde estos se pueden ejecutar. Ha sido necesario adquirir conocimientos avanzados sobre esta tecnología, y sobre la plataforma JADE. A su vez hemos tenido que consultar los estándares para agentes y sistemas multiagente desarrollados por la organización FIPA.

Hemos necesitado realizar una planificación de tareas para organizar el desarrollo de nuestro proyecto. Esta planificación ha sido bastante precisa. Simplemente ha habido la eliminación de la tarea *Semana integradora*. Esta tarea consistía en realizar, en una semana, la integración del este proyecto con otros basado en la misma temática para dar mayor estabilidad o mejoras en el paradigma de agentes.

Inicialmente la planificación que hemos realizado daba como fecha de terminación del proyecto una semana antes a la fecha límite. Finalmente debido a la etapa de implementación y documentación se ha alargado unos día más. La etapa

de implementación se ha alargado para dar mejor rendimiento al juego.

Una vez adquiridos los conocimiento necesarios sobre la tecnología de agentes, hemos realizado una clasificación de los juegos que existen en el mercado. Hemos podido comprobar la dificultad que conlleva realizar una clasificación óptima de juegos, donde la principal dificultad ha sido encontrar una clasificación que los abarque a todos. Mediante esta clasificación hemos realizado una caracterización de juegos donde su desarrollo mediante la tecnología de agentes móviles es el más óptimo. También hemos explicado la perspectiva que ha de tener el sistema y los perfiles de los usuarios del mismo.

Después de haber realizado la caracterización de juegos, hemos decidido diseñar un *core* común para la tipología de juegos definida. Durante el proceso de diseño ha sido necesario establecer como reacciona el sistema delante de los acontecimientos que se pueden producir durante su ejecución. Todo este proceso se ha realizado mediante el lenguaje de modelado UML(Unified Modeling Language) ya que nos ha proporcionado la estructura básica del sistema y como los diferentes componentes actúan e interaccionan entre ellos. Después de haber diseñado el core hemos diseñado e implementado un caso particular, el Parchís.

El último objetivo que nos hemos propuesto consiste en el diseño e implementación de una práctica para la asignatura de Redes de Computadores II. La práctica ha consistido en la implementación de un juego muy sencillo donde hemos utilizado el *core*, implementado en el proyecto, para su diseño e implementación. Finalmente hemos redactado el enunciado de esta práctica.

Los objetivos que nos hemos propuesto han sido logradas pero han surgido posibles mejoras a realizar que se comentarán a continuación.

Mejoras del sistema

En este apartado se explicarán las mejoras que se pueden realizar del proyecto.

- Una posible mejora consiste en realizar el inicio del juego de manera más flexible que en este proyecto. Existe una plataforma central donde ser realizan las peticiones de participar en un juego y hasta que no se llene el

cupo de participantes no se puede jugar. Otra característica puede ser el hecho de que puede tener implementados diferentes juegos de manera que al recibir una petición del usuario, la plataforma lanza el agente que contiene implementado el juego.

- Otra posible mejora de este proyecto es la implementación de una interfaz gráfica más interactiva con el usuario. En este proyecto el usuario puede ver el estado del tablero en una ventana gráfica pero para que éste pueda comunicar un movimiento lo tiene que comunicar mediante la entrada estándar. Se puede utilizar la ventana gráfica implementada en el proyecto y que el usuario solo tenga que seleccionar la ficha y arrastrarla hasta la posición deseada.

Líneas futuras

En este apartado se explicarán las líneas futuras que pueden surgir de este proyecto.

- El diseño e implementación de un *core*, compartido por una gran cantidad de juegos, hace posible que se pueda utilizar para el desarrollo de éstos. Los usuarios pueden desarrollar nuevos juegos utilizando como base nuestro *core*, de manera que éstos puedan modificar el *core* básico desarrollado e implementado en nuestro proyecto incorporando mejoras funcionales. Este proceso ayudaría al desarrollo de juegos mediante la tecnología de agentes.

- Este proyecto ha sido realizado sobre un entorno estático, pero utilizando una JVM(Java Virtual Machine) y una versión simplificada de JADE(JADE-LEAP) puede ser utilizado en terminales móviles.

Apéndice A

Práctica para la asignatura Redes de Computadores II

Redes de Computadores II **Práctica 3**
Curso 2007-2008
Agentes Móviles **Fecha de entrega X/X/XX**

Objectivo

El objetivo de esta práctica es aprender a desarrollar una aplicación distribuida utilizando la tecnología de agentes móviles. En concreto, se desarrollará una aplicación sobre un juego similar al Nim utilizando la plataforma JADE.

JADE (*Java Agent DEvelopment Framework*) es una plataforma desarrollada en *Java* que permite la ejecución y interacción de múltiples agentes según las especificaciones de *FIPA*. JADE también incorpora un conjunto de herramientas que facilitan la depuración i el desarrollo de estos agentes.

JADE incorpora un add-on de movilidad interplataforma desarrollado en el DEIC.

La movilidad y la seguridad en entornos de agentes móviles es una de la ramas actuales de investigación del *Departament d'Enginyeria de la Informació i de les Comunicacions*, formando parte de uno de los proyectos más avanzados en esta area a nivel mundial.

Enunciado

La práctica consistirá en el desarrollo de diferentes agente móviles.

El juego del Nim consiste en una bolsa llena de palitos de donde un jugador puede quitar uno, dos o tres en cada turno. Es necesario que hayan un mínimo de tres jugadores. El juego termina cuando el último jugador quita el último palito de la bolsa.

Para la implementación del juego del Nim serán necesarios dos agentes.

En todas las máquinas de los laboratorios hay instaladas una plataforma JADE para que vuestros agentes puedan migrar.

A continuación se comentan los agentes necesarios en la implementación de la práctica.

Agente *ping* (`GX_PingAgent`)

Crear un agente móvil que implemente una funcionalidad similar a la de la aplicación "ping". El agente será invocado indicándole un nombre de agencia. Después de obtener la hora actual en la plataforma de partida, el agente migrará a la plataforma indicada. Inmediatamente después el agente volverá a la plataforma origen, obtendrá de nuevo la hora y mostrará la diferencia con el primer valor obtenido.

El nombre de la clase que implementa este agente ha de ser: `Grupo_PingAgent`. Por ejemplo: `B12_PingAgent`

Agentes del juego Nim (`GX_NimAgent` y `GX_PlayerAgent`)

Es necesario la creación de dos agentes para la implementación del juego del Nim.

GX_NimAgent

Crear un agente móvil que implemente el juego del Nim. Este agente será el encargado de:

- Informar a los usuarios
- Evaluar la jugada realizada por parte del usuario

- Modificar el contenido de la bolsa de palitos.

A continuación se explicar las acciones que debe realizar al agente NimAgent:

Informar a los usuarios.

El agente NimAgent informará al usuario de diferentes hechos que ocurren en la partida. Para hacer esto es necesario que el NimAgent envie al PlayerAgent la información. De manera, que el PlayerAgent le muestre la información al usuario.

La manera de enviar información entre agentes se realiza mediante mensajes ACL. Este mensaje es un objeto de la clase `jade.lang.acl.ACLMessage`. A continuación se muestra un ejemplo de creación de un mensaje.

A la hora de crear un mensaje ACL se ha de pasar como argumento el tipo del mensaje (INFORM, PROPOSE, FAILURE, etc). Con el método `setContent()` introducimos el contenido del mensaje.

```
ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
msg.setContent("Hola agente");
msg.addReceiver(agente_destinatario)
```

Muy Importante: El contenido de un mensaje ACL es siempre un *string*.

Por último se ha de añadir el receptor del mensaje mediante el método *addReceiver()*.

En el caso de querer extraer el contenido del menaje se utiliza el método *getContent()*.

Para que el PlayerAgent pueda diferenciar la información que le envía el agente NimAgent se necesitan unos códigos que incluimos como prefijo en el contenido del mensaje. En nuestro caso serán necesarios.

0	La jugada es correcta
-1	La jugada es incorrecta
-2	El resultado del juego
-3	Información inicial

Evaluar la jugada realizada por parte del usuario

El NimAgent deberá comprobar si la jugada que desea hacer el usuario es correcta o no.

Recordamos, que el número de palitos para extraer ha de ser entre 1 y 3. También hay que recordar que no puede quedar un número negativo de palitos en la bolsa, no se puede retirar más palitos que los que quedan en la bolsa.

Modificar el contenido de la bolsa de palitos.

El NimAgent, después de comprobar que el número de palitos a extraer por parte del usuario es correcto, restará esta cantidad al número total de palitos que hay en la bolsa.

GX_PlayerAgent

El agente PlayerAgent será un agente estático que hará de conector entre el usuario y el NimAgent. El PlayerAgent se encargará de realizar tres acciones.

- Informar al usuario, mediante la salida estándar, de la cantidad de palitos que hay en la bolsa.
- Recibir la jugada por parte del usuario.
- Enviar mediante un mensaje ACL, la jugada al NimAgent. En este caso no es necesario incorporar ningún código al cuerpo del mensaje, porque el PlayerAgent solo envía esta información.

Finalmente, cuando el juego ha terminado, el NimAgent ha de volver a la agencia inicial y reconstruir un fichero con el historial de la partida:

`/tmp/Grupo_NimAgent.log`

El nombre de la clase que implementa el agente jugador tiene que ser : `Grupo_PlayerAgent`. Por ejemplo: `B10_PlayerAgent`

La clase `agents/src/agents/MobileAgent.java` es un ejemplo de agente que migra hacia una plataforma inicial.

Observaciones

- Quando lancéis un agente, este estará en la plataforma de la máquina actual.
- Para que un agente pueda migrar, todas las clases que utiliza han de ser serializables, es decir, han de implementar la interfaz:

  `java.io.Serializable`.

  Las classes: `Agent` y las básicas de la API de Java que hayáis utilizado, ya implementan esta interfaz. Si vosotros necesitáis alguna otra clase, tendrá que implementar esta interfaz.
- Todos los parámetros que necesita vuestros agente NimAgent, se le pasarán a través de un fichero de propiedades. El nombre de este archivo se le especificará al agente como parámetro.

  El contenido de un fichero de propiedades es una lista de entradas del tipo:

  ```
  propiedad=valor
  ```

  Por ejemplo:

```
destination=ccd-dc1, ccd-dc2
id = partida1
players = 3
bag = 10
```

En el caso del agente `PingAgent`, se utilizará otro fichero de propiedades donde será necesario la propiedad: `destination`, indicando la única plataforma a la que queremos migrar, des de la plataforma actual.

Respecto al `NimAgent`, a más, necesitará:

- `id`, indicando el identificador de la partida.
- `players`, indicando el número de jugadores que forman la partida.
- `bag`, indicado en número total de palitos que contiene la bolsa de palitos.
- `destination`, indicando las plataformas que forman el juego.

Muy Importante: Por lo referente a la propiedad `destination`, es neceario que especifiquéis el nombre de las máquinas donde están las plataformas participantes en el juego, sin incluir el nombre del dominio, por ejemplo: `ccd-dc2, ccd-dc3`.

Como ejemplo, tenéis el fichero `MobileAgent.properties`, que corresponde al fichero de propiedades que utiliza la clase:

```
src/agents/MobileAgent.java
```

- Cuando el agente NimAgent conozca que ha llegado el fin del juego, es necesario que muera. Tenéis que ejecutar el método `doDelete()`. En caso contrario, el agente quedará en estado *zombie* en la plataforma. **Pensar que una plataforma no puede ejecutar dos instancias del mismo agente a la vez**.

- Relacionado con lo anterior: **Un agente no puede migrar a la plataforma donde se encuentra actualmente**.
- Si en algún momento os aparece un mensaje de error indicando que el agente que habéis hecho migrar ya está ejecutándose en la plataforma de destino, quiere decir que se os ha quedado el agente colgado en esta plataforma. Lo que tenéis que hacer es cambiar el nombre de la clase del agente, por ejemplo: `G12_NimAgent_1` o `G12_PlayerAgent_1` y lanzarlo de nuevo.
- Los nombres que debéis utilizar para el fichero de propiedades y para los agentes son los siguientes:

Agente	Grupo_NombreAgente
Código fuente	Grupo_NombreAgente.java
Clase	Grupo_NombreAgent.class
Registro	Grupo_NombreAgent.log

- Para compilar y ejecutar vuestros agentes utilizaréis la herramienta `ANT` de *Apache*.

  Antes falta que tengáis definida la variable JAVA_HOME. Para comprobarlo podéis ejecutar:

  ```
  echo $JAVA_HOME
  ```

  En el caso de que no la tengáis definida, necesitaréis hacerlo:

  ```
  export JAVA_HOME=/usr/java/jdk1.5.0_03
  ```

  Podéis incluir la línia anterior en el vuestro fichero `.bashrc`. De este modo cada vez que abráis una consola se cargará este valor.

  Para compilar, desde el directorio `agents`, necesitáis que ejecutéis:

  ```
  ant compile-agents
  ```

Para lanzar el agente se necesita que ejecutéis, desde el directorio `agents`:

```
ant launch -Dagent=MobileAgent -Dparams=MobileAgent.properties
```

Con el parámetro `agent` le indicamos el nombre de la clase que implementa nuestro agente y que queremos ejecutar. Con el parámetro `params` indicamos el nombre del fichero de propiedades que necesita el agente.

- Durante la primera sesión de prácticas recibiréis instrucciones detalladas de los procedimientos básicos para el lanzamiento y programación de agentes móviles.
- Hay plataformas ejecutándose desde la máquina 1 a la 13, las dos incluidas.
- Hemos replicado todo el entorno de ejecución del laboratorio QC/2008 en el laboratorio QC/2009. Es decir, en las máquinas de la 14 a la 26, las dos incluidas, hay una plataforma JADE ejecutándose.

Bibliografía

[FIPA07] FIPA. *The Foundation for Intelligent Physical Agents*. <http://www.fipa.org/>

[AUML07] *The FIPA Agent UML Web Site*. <http://www.auml.org/>

[SRS98] IEEE SRC.*IEEE recomended practice for software requirements specifications*, octubre 1998. <http://ieeexplore.ieee.org/>

[BCG07] BELLIFEMINE, Fabio; CAIRE, Giovanni; GREENWOOD, Dominic. **Developing multi-agent systems with JADE**. WILEY, 2007. 286 pág.

[JADE07] JADE. *Java Agent DEvelopment Framework*. <http://jade.tilab.com/>

[FOSS07] *FIPA Ontology Service Specification*. <http://fipa.org/specs/fipa00086/index.html/>

[GCDC04] G.Caire and D. Cabanillas. *Jade Tutorial: Application-defined content languages and ontologies.* TILab S.p.A., November 2004. acc. Feb. 2006. <http://sharon.cselt.it/projects/jade/doc/CLOntoSupport.pdf>

[JFS07] John F. Sowa. *Ontology* .
http://www.jfsowa.com/ontology/index.htm>

[JDOM] JDOM. *JDOM Project*.
<http://www.jdom.org/>

Firmado: Francisco J. Requena Moreno
Bellaterra, Junio de 2007

Resum

En aquest projecte s'han classificat i posteriorment caracteritzat els jocs on la tecnologia d'agents mòbils és més òptima. Posteriorment s'ha dissenyat un *core* per al desenvolupament de jocs d'aquesta tipologia. Mitjançant aquest *core* s'han implementat el joc del Parchís i el joc del Nim. Aquest últim cas pràctic ha estat realitzat per a l'elaboració d'una pràctica per a l'assignatura de Xarxes de Computadors II. Gràcies als avantatges que ens oferix la tecnologia d'agents mòbils, com la tolerància a fallades i flexibilitat, hem demostrat amb aquest projecte que és òptim utilitzar aquesta tecnologia per a la implementació de jocs.

Resumen

En este proyecto se han clasificado y posteriormente caracterizado los juegos donde la tecnología de agentes móviles es más óptima. Posteriormente se ha diseñado un *core* para el desarrollo de juegos de esta tipología. Mediante este *core* se han implementado el juego del Parchís y el juego del Nim. Este último caso práctico ha sido realizado para la elaboración de una práctica para la asignatura de Redes de Computadores II. Mediante las ventajas que nos ofrece la tecnología de agentes móviles, como la tolerancia a fallos, flexibilidad y seguridad estructural, hemos demostrado con este proyecto que es óptimo utilizar esta tecnología para la implementación de juegos.

Abstract

In this project we have classified and characterized games applications that can be optimally implemented using mobile agente technology. Later we have been designed a *core* for the development of this games typology. With this *core* we have been implemente Ludo game and Nim game. This last practical case has been made for a subject Networks of Computer II practice. With the advantages that mobile agent technology offers like tolerance to failures, flexibility, we have demonstrated with this project that this technology is ideal to for games implementation.