Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior
(Jaén)

Trabajo Fin de Grado

DESARROLLO DE PROTOTIPO 3D PARA EL JUEGO DE DIPLOMACIA

Alumno: Marín Soto, Juan José

Tutor: Prof. D. Jiménez Pérez, Juan Roberto
Prof. D. Sánchez Sánchez, Pedro José
Dpto: Informática

Septiembre, 2015

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don JUAN ROBERTO JIMÉNEZ PÉREZ y Don PEDRO JOSÉ SÁNCHEZ SÁNCHEZ, tutores del Trabajo Fin de Grado titulado: Desarrollo de prototipo 3D para el juego de diplomacia, que presenta JUAN JOSÉ MARÍN SOTO, autorizan su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Septiembre de 2015

El alumno:

JUAN JOSÉ MARÍN SOTO

Los tutores:

JUAN ROBERTO JIMÉNEZ PÉREZ

PEDRO JOSÉ SÁNCHEZ SÁNCHEZ

Índice

1. Introducción

La interacción de usuario se encuentra en una continua evolución a través de los años suscitando el debilitamiento de la frontera límite que separa el espacio de la interacción humana de la de las máquinas. La investigación en el campo de la psicología cognitiva nos ha dado modelos de procesamiento de información referente a agentes humanos y el acercamiento a la ciencia cognitiva nos provee de un marco útil para razonar sobre el pensamiento y razonamiento a la hora de realizar tareas (Hutchins, 1995). El principal objetivo de esta interacción es permitir una correcta efectividad respecto al control de la contraparte inanimada desde el punto de vista humano, al mismo tiempo que esta ofrece una retroalimentación en forma de estímulos que ayuden al operador a tomar decisiones en el proceso. Años atrás queda la era de la computación por lotes o línea de comandos y desde la década de los ochenta se explota a través de la manipulación directa los receptores sensoriales humanos por medio de elementos gráficos o sonoros reconocibles con el objetivo de reducir la carga cognitiva asociada a la interacción (E. Hutchins, 1985).

A la hora de analizar flujos de información y procesos de toma de decisiones sobre más de una unidad de análisis, el marco cognitivo tradicional se extiende hacia la llamada cognición distribuida, la cual abstrae el individuo para centrarse en la estructura distribuida (Halverson, 2002). Un claro ejemplo de cognición distribuida es un juego de mesa típico, donde son varios los actores que se comunican entre sí plasmando sus decisiones en un tablero, el cual representa el entorno o mundo, resultado de las acciones de los actores (a continuación definidos como agentes) a través de los artefactos a su alcance. Esta interacción no es unilateral y el resultado de las acciones de uno de los agentes tendrá su efecto directo o indirecto en el resto de los agentes implicados en el proceso. Cada agente a la hora de tomar sus decisiones deberá estudiar el estado del entorno y tomar en cuenta las posibles acciones de los demás agentes con la finalidad de alcanzar su objetivo.

En las siguientes páginas, se plasma el proceso de adaptación del juego de mesa Diplomacia a un entorno de agentes inteligentes, definiendo los elementos necesarios en materia de plataforma y organización estructural de los distintos agentes necesarios para ello. El auge de la computación gráfica y la mejora que supone su uso en el proceso cognitivo, es la principal motivación para buscar abordar

la inclusión de una solución capaz de plasmar de forma visual la interacción de este sistema y la huella que dejan dichos agentes en el entorno virtual. Para llevar esto a cabo, será necesario definir una forma de comunicación entre ambos subsistemas, gráfico y lógico, los cuales no necesariamente estarán próximos en el espacio o estarán basados en un lenguaje de programación común.

Este trabajo tiene como propósito específico el desarrollo de un prototipo 3D del juego de mesa llamado ‘Diplomacia’. Dicho prototipo estará compuesto por el diseño de una arquitectura lógica sustentada sobre un sistema de agentes software además de una visualización del entorno virtual y diseño de experiencia de interacción sobre un motor de juegos. El trabajo analiza tanto las opciones existentes actualmente para elaborar la interfaz de usuario y el mundo virtual en forma de un motor de juego que ofrezcan una retroalimentación al usuario como las diferentes plataformas de gestión de agentes y su idoneidad, motivadas por la elección previa del motor. Incluida en esta primera etapa de investigación y estudio, se ofrecen diferentes soluciones de integración de ambos elementos, motor de juego y plataforma de agentes. Esta primera etapa, se lleva a cabo valorando la viabilidad e idoneidad de cada una de las soluciones existentes y argumentando de forma crítica las decisiones tomadas.

Diplomacia es un juego de astucia y negociación en el que el azar no forma parte del juego. Cada jugador rige los destinos de una de las grandes potencias europeas de principios de siglo a través de los vericuetos de la política internacional. Por medio de las negociaciones con el resto de jugadores cada jugador busca lograr el control de Europa (Las reglas de la diplomacia, 1987). A través de una serie de rondas que representan épocas desde principios del siglo pasado, los jugadores por medio de movimientos de una serie de unidades otorgadas al inicio del juego, son capaces de conquistar distintos territorios que pueden contener suministros para la creación de nuevas unidades. Estas unidades pueden ser de tipo marítimo o terrestre, limitando de este modo los movimientos que podrán efectuar por el tablero. A su vez, las unidades podrán efectuar una serie de movimientos como son la ocupación de territorios o defensa, apoyados o no por otras unidades, resultando en un balance de fuerzas al final de cada turno que definirá el control del mismo.

1.1. Motivación

El motivo de la elección del presente tema como trabajo fin de grado es muy variado. Por un lado permite aplicar los distintos conocimientos adquiridos por el alumno en múltiples de las asignaturas cursadas durante el periodo de estudio, desde materias específicas y fuertemente ligadas como Desarrollo de Videojuegos o Sistemas Multiagente que forman parte del itinerario de especialización autor en Tecnologías de la Información con mención en sistemas gráficos. A su vez, se aplican conocimientos más generales como Estructura de Datos para mejorar el rendimiento de la solución aportada, Interacción Persona-Ordenador para evaluar la usabilidad de la interfaz o Algebra Lineal aplicada a teoría de grafos para representar la codificación interna.

Por otro lado, se intenta comprender mejor a través del proceso de elaboración de este trabajo la integración de subsistemas, materia que fundamenta una de las bases de las nuevas Tecnologías de la Información. Para ello, se busca una forma de comunicación de los dos componentes principales que forman el prototipo, motor de juegos y plataforma de agentes, para obtener una solución unificada.

1.2. Objetivos

El objetivo principal de este trabajo es realizar un prototipo para el juego de mesa diplomacia y a su vez definir las bases necesarias para el proceso de diseño y elaboración de otros trabajos que compartan cierto grado de similitud con el proyecto a tratar. A su vez, la memoria se redactará intentando no presuponer más allá de los conocimientos mínimos del área, facilitado así la accesibilidad a todo aquel interesado.

Los puntos principales a tratar en este trabajo serán:

- Revisión bibliográfica de las tecnologías existentes.
- Diseño de la arquitectura multiagente.
- Diseño del entorno virtual para el desarrollo del juego, donde se representará las elecciones tomadas por el sistema multiagente.
- Diseño de la interfaz de usuario y definición de interacciones.
- Elaboración de la memoria de trabajo.

1.3. Organización de la memoria

La memoria de este trabajo fin de grado se organiza en los siguientes capítulos:

En el capítulo 1 se introduce el tema a tratar así como la motivación que ha llevado a la elección de dicho tema y los objetivos previstos además de este resumen.

En el capítulo 2 se ofrece una visión global de los principales puntos a tratar por el proyecto, así como un estudio de las soluciones existentes en los tres ámbitos de actuación y un razonamiento objetivo respecto a las soluciones elegidas.

En el capítulo 3 se describe detalladamente la solución propuesta, describiendo el flujo de trabajo dentro de la aplicación y la lista de requisitos que cumplirá la aplicación.

En el capítulo 4 se expone la metodología de desarrollo usada, junto a una descripción precisa de los módulos y clases principales. Todo ello acompañado de la descripción y pruebas de la interfaz de usuario.

En el capítulo 5 se realiza una evaluación del sistema completo, así como la definición de datos de entrada y resultados obtenidos.

En el capítulo 6 se razona sobre los objetivos alcanzados y las conclusiones fruto del trabajo realizado, además de analizar los problemas surgidos y sugerir trabajos futuros.

2. Estudio previo de soluciones tecnológicas y su integración

El proyecto, como se ha mencionado en la introducción, trata sobre la integración de una plataforma de agentes software sobre un motor de juegos, con la finalidad de aportar entornos visuales a la plataforma. Al basar la solución sobre agentes, el control de los mismos puede recaer sobre agentes humanos o una inteligencia artificial programada dependiendo si los agentes son o no inteligentes. Aumentar el nivel de autonomía de un agente dentro del sistema es transparente al resto, siendo necesario cambiar únicamente el comportamiento del agente implicado, en cuanto al nivel de asistencia al usuario o autonomía de decisiones. Gracias a esto es posible definir dentro del juego resultante, el nivel de asistencia en la toma de decisiones de los agentes jugadores implicados, pudiendo incluso servir únicamente de consejero, dejando al usuario que los maneja tomar la decisión final basándose o no en las indicaciones aportadas por estos.

Durante el proceso de juego, los agentes jugadores tomarán diferentes roles motivados por las características puntuales del juego, representadas como el entorno de actuación, que harán que su actitud vaya cambiando de un estado competitivo con la finalidad de ganar el juego a la generación de alianzas entre ellos tornando a un estado colaborativo y negociador, en caso que deseen batir a una fuerza mayor, aunque no se verán obligados a cumplir lo pactado durante dichas negociaciones, lo cual puede ser un elemento extra de juego.

Una plataforma de agentes, se basa en un entorno software sobre el que construir sistemas de agentes para la gestión de recursos de información interconectados (Bellifemine, Poggi, & Rimassa, 2001). Esta plataforma se encargará de gestionar el estado de vida de cada agente contenido en ella, además de servir como una vía de comunicación entre estos.

Dichos agentes software son entidades autónomas, que interaccionan con un entorno a través de cierto nivel de reactividad frente a estímulos y autonomía en la toma de decisiones, y que al actuar de manera distribuida con un objetivo común constituyen un sistema multi-agentes, definido como la agregación de distintos elementos (Ferber, 1999):

1. Un entorno
2. Un conjunto de objetos
3. Un conjunto de agentes
4. Un conjunto de relaciones que unen objetos y agentes
5. Un conjunto de operaciones
6. Operadores que representan la aplicación de operaciones sobre el mundo y la reacción de éste al ser alterado.

Además, es necesario definir formalmente el entorno virtual, el cual se trata de la representación mundo o entorno artificial inspirado o no en la realidad, en el cual los usuarios pueden interactuar entre sí a través de su simulación en un soporte software. Este entorno estará sustentado sobre un motor de juego, que aportará al usuario la capacidad de visualización de una forma más cómoda que la interpretación de texto y que esta descrito a continuación.

2.1. Motor de juegos

Un motor de juegos se trata de un entorno de desarrollo software diseñado para la creación y desarrollo de videojuegos. Sus funcionalidades principales suelen estar compuestas por un motor de renderizado, ya sea en dos o tres dimensiones, un motor de físicas o detección de colisiones, además de sonido, animación, inteligencia artificial, capacidades de red o concurrencia entre otros.

El término de motor de videojuegos surgió a mediados de los 90 con la aparición del videojuego en primera persona “Doom”, desarrollado por la compañía id Software. Doom fue el primero de los videojuegos diseñado con una arquitectura orientada a la reutilización, mediante una separación adecuada en distintos módulos de los componentes fundamentales, como por ejemplo el sistema de renderizado gráfico, el sistema de detección de colisiones o el sistema de audio (Gregory, 2009).

Esta definición, aboga por la reutilización del software, la cual facilita enormemente la labor de los programadores, permitiendo diseñar videojuegos sin apenas modificar el núcleo del mismo, permitiendo dirigir esfuerzos hacia la parte artística o la experiencia de juego. Este proceso de reutilización reduce tiempos de desarrollo y costes, a la vez que incrementa la fiabilidad y la productividad.

Pese a ello, la separación de un motor de videojuegos por el que es implementado nunca es total, y siempre se mantienen ciertas dependencias que no permiten la reusabilidad completa de los componentes internos. Esta dependencia se hace más visible al estar el motor basado en un género concreto, ya que un motor diseñado para un videojuego de acción en primera persona será difícilmente reutilizable para un videojuego de carreras o conducción.

Para la elaboración de este trabajo, se ha optado por el motor de Unity Technologies, que a pesar de poseer una licencia propietaria, ofrece a su vez la opción de usar sus servicios siempre que sean proyectos no comerciales. El principal motivo de elección de dicha plataforma, frente a otras con licencias menos restrictivas como Ogre3D o CryEngine, ha venido liderado por la gran comunidad de usuarios que este posee que aportan gran cantidad de documentación, además de distintos Assets o elementos integrables al videojuego libres o de coste reducido.

Unity está escrito en la familia de lenguajes C y permite su desarrollo de scripts tanto en un lenguaje propio llamado UnityScripts como C# o JavaScript. Esto junto a la polivalencia para ejecutarse en distintas plataformas como PC, consolas, dispositivos móviles o web le han hecho ganarse el apoyo de la comunidad de usuarios. Su motor de scripts está construido sobre Mono, la implementación open source de .NET Framework, basada en los estándares ECMA para C#. (MonoProject, 2015)

2.2. Sistema multiagente

En los últimos años, la comunidad de modelado basado en agentes ha desarrollado varios conjuntos de herramientas de modelado práctico que permiten a los individuos desarrollar aplicaciones basadas en agentes. Cada vez más este tipo de herramientas están viniendo a la existencia, y cada toolkit tiene una variedad de características diferentes entre sí. Los desarrolladores de aplicaciones normalmente usan estos conjuntos de herramientas en lugar de desarrollar su propio software compatible con estándares ya creados. Esto facilita el desarrollo y reduce el periodo de pruebas, asumiendo que el toolkit elegido ya ha pasado por dicho periodo. Existe una larga lista de opciones disponibles y se han hecho muchos intentos para comparar conjuntos de herramientas entre sí, pero esta se reduce drásticamente al comparar soluciones compatibles con el estándar de mensajes propuesto por la fundación para los agentes físico-inteligentes (FIPA).

Dicha fundación fue creada con el objetivo de definir un conjunto de normas aplicables al mundo de los agentes inteligentes, especificando como se comunican e interactúan los agentes entre sí. Los dos estándares más ampliamente adoptados son las especificaciones de gestión de agentes y el lenguaje de comunicación de agentes (FIPA-ACL). Durante su periodo de vida se han creado decenas de toolkits que incluyen dichas especificaciones. Para este proyecto se han estudiado y evaluado tres de las múltiples opciones disponibles, valorando la calidad que pueden aportar como solución: AgentService, FIPA-OS y JADE. Todas ellas, comparten el fundamento de ser opciones de código libre y ser compatibles con el estándar de comunicación ACL. A continuación se exponen los detalles concretos de cada plataforma así como las razones de elección de la solución candidata para la implementación.

2.2.1. *AgentService*

El proyecto AgentService abarca el desarrollo y mantenimiento de agentes orientados al marco de la portabilidad sobre todos los sistemas operativos de hosting de aplicaciones diseñadas sobre la infraestructura de lenguaje común. Sus creadores lo definen como un entorno seguro, eficiente, extensible y modular, para implementar sistemas multiagente siguiendo los estándares FIPA (L.I.D.O., 2005). Ha sido probado sobre lenguajes como Mono, .Net, y SSCLI, siendo los dos primeros el principal atractivo de estudio de esta plataforma, al ser los lenguajes empleados por el motor de videojuegos elegidos en el apartado anterior.

Dicha plataforma es elogiada en diferentes fuentes bibliográficas como una plataforma que simplifica el proceso de desarrollo y proporciona un buen apoyo para la producción de software de calidad. Esto queda reflejado en las opciones de diseño de los componentes esenciales del sistema: el modelo de agente y la infraestructura de tiempo de ejecución de apoyo al sistema. Estos dos componentes están integrados en un tiempo de ejecución robusto que permite la distribución y evolución de los agentes en el tiempo. Comparado con otros frameworks, AgentService no admite todas las fases del ciclo de vida de software con herramientas de alto nivel (Vecchiola, Grosso, & Boccalatte, 2008) además de tratarse de una plataforma con diez años de antigüedad, aun en estado beta, que se encuentra abandonada desde hace seis años.

A la hora de intentar integrarlo sobre el motor de videojuegos descrito, han surgido multitud de problemas pese a configurarlo en modo de compatibilidad con compilaciones anteriores. Esta razón, junto a tratarse de un proyecto en fase beta y abandonado han sido las principales razones para desestimarlo como plataforma de agentes.

2.2.2. *FIPA-OS*

Es una plataforma de agentes de código abierto, procedente de Nortel Networks la cual soporta la comunicación de agentes que usan mensajes bajo el estándar FIPA. Se ha demostrado su versatilidad para operar con otras plataformas heterogéneas bajo el mismo estándar, y su uso está distribuido por todo el mundo (Poslad, Buckle, & Hadingham, 2000). Al ser la plataforma de código abierto, todos sus componentes internos están al descubierto y la comunidad de desarrolladores evoluciona en múltiples vías, aunque a pesar de ello, necesita la aprobación de los arquitectos originales quienes pueden limitar su potencial.

La falta de documentación, junto a ser una plataforma no desarrollada activamente desde hace años, han sido razones suficientes para desestimarla como candidata a acompañar el motor de videojuegos.

2.2.3. *JADE*

El framework de desarrollo de agentes Java o JADE, es un middleware que como su nombre indica esta implementado sobre java para la creación de agentes software. El sistema permite la coordinación entre distintos agentes FIPA y provee estándares de comunicación ACL que facilitan dicha comunicación. JADE provee un entorno donde los agentes son ejecutados junto a librerías de clases para crear agentes usando herencia y redefinición de comportamiento. Su interfaz gráfica para monitorizar y gestionar la plataforma de agentes inteligentes es su principal reclamo, al ofrecer un gran potencial a la hora de depurar la aplicación resultante.

Existe una gran cantidad de documentación relacionada con el proyecto, junto a multitud de addons de la plataforma que permiten ampliar sus posibilidades haciéndola compatible con otros servicios o lenguajes, aunque al estar implementado sobre una máquina virtual java esto ralentiza su ejecución y eleva los tiempos de respuesta en situaciones de alta ocupación de agentes. Pese a ello, se ha considerado la opción más viable a implementar por ser una de las plataformas más populares en el ámbito de los agentes inteligentes y el soporte en forma de documentación prestado.

2.3. Integración de subsistemas

Una vez elegidos el motor de videojuegos sobre el que realizar la implementación del mundo virtual y controles de usuario, junto a la plataforma de agentes que albergara a estos, es necesario encontrar un nexo de unión entre ambos al estar basados ambos en dos lenguajes distintos (C# y Java) e incompatibles entre sí a priori. Ambos lenguajes son considerados de alto nivel y ejecutados sobre máquinas virtuales, lo que significa que el código escrito es compilado en lenguajes intermedios y después ejecutados por .NET Framework (CLR) o Java Runtime (JVM) los cuales traducen el código a código nativo.

Para ello, se han estudiado distintas soluciones expuestas a continuación, destacando virtudes y defectos de cada una de ellas, además de la idoneidad en cuanto a tiempo de desarrollo necesario para su implementación.

2.3.1. *Interoperabilidad con librerías*

Desde estos lenguajes, se puede acceder a niveles inferiores con características específicas implementadas en los compiladores y máquinas virtuales. Código nativo como C, C++,… puede ser llamado declarando métodos nativos e indicando a que librería implementa dicho método. Al realizar dicha llamada desde C# o Java la máquina virtual enruta la llamada hacia una librería nativa particular, invocando el método nativo desde otro lenguaje. En Java, este mecanismo es llamado JNI (Java Native Interface) y en .NET es gestionado por DllImport. En ambos casos es usado internamente por métodos del framework que interactúan con capas bajas del sistema.

Unity, permite a través de Mono, la inclusión de distintas librerías a través de la infraestructura de lenguaje común (CLI), la cual está diseñada para facilitar la interoperabilidad de código existente. Estas librerías están definidas como plugins en el entorno de desarrollo y se diferencian en dos categorías: gestionadas o nativas. Para la transformación de librerías Java a C#, existen diferentes soluciones entre la que cabe destacar IKVM, la cual es definida por sus creadores como una implementación de Java para Mono y .NET incluyendo la máquina virtual java, implementación de librerías de clase y herramientas que permiten la interoperabilidad (Frijters, 2015).

Como solución alternativa a IKVM, existe un addon de Jade llamado Leap, el cual es un proyecto orientado a reducir los recursos necesarios de la máquina virtual java, motivado por la orfandad de dispositivos móviles de una plataforma de agentes idónea para su potencia y cubrir las necesidades asociadas a enlaces Wireless que no son cubiertos por Jade. El problema de dicha solución es que basa su dependencia en la versión de .NET 2.0, desfasada y con la que el motor de videojuegos ofrece problemas de compatibilidad. Esto sumado a que el desarrollo anda parado desde hace años hace que no sea una solución viable.

El proyecto IKVM en cambio ofrece un buen ritmo de actualizaciones y una gran facilidad de portabilidad, pero la solución en conjunto queda descartada como medio de integración al restringir Unity la inclusión de librerías a usuarios Pro de su plataforma.

2.3.2. *Integración entre máquinas virtuales*

Cabe la posibilidad también de integrar máquinas virtuales como Java Runtime y .NET Framework con la creación de un bridge nativo. Dicho bridge es cargado por ejemplo por la aplicación Java, la cual integra el tiempo de ejecución .NET en el mismo proceso en memoria, junto a la librería .NET dll en el mismo motor. Cada vez que se llama cualquier método .NET desde Java, la información es enrutada nativamente en memoria hacia la librería e invocada dentro del tiempo de ejecución .NET cuyo resultado es pasado a Java. Esto resulta en un método muy eficiente y rápido a la vez que seguro. Una clase .NET puede heredar de otro Java simplemente heredando del puente entre ambas.

Este método tiene la ventaja añadida de que, desde que Java es ejecutado en una máquina virtual o un servidor de aplicación Java EE, no es necesario tener código fuente, la solución aportada funcionará también únicamente con los binarios. Al mantenerse las clases Java compiladas bajo códigos de bits, se siguen manteniendo compatibles con diferentes plataformas.

Existen soluciones como Javonet, las cuales permiten insertar código .NET en aplicaciones java (SdNcenter Company, 2015) pero que no son viables al tener como centro la plataforma de agentes y no el motor de videojuego, el cual por su mayor complejidad necesita ser quien gestione al resto. La integración en la otra dirección, viene ofrecida por soluciones como jnbridge, la cual a través de diferentes adaptadores y bridges permite solucionar la integración en ambas direcciones (JNBridge LLC, 2015) pero pese a ello, es descartada también como solución al ser un producto comercial con una licencia costosa.

2.3.3. *Integración a través de la comunicación*

Existen otras soluciones como el uso de objetos COM y DCOM los cuales exponen interfaces comunes que pueden ser consumidos por otras aplicaciones. Dichas librerías son construidas como objetos COM y registrados en el sistema para ser accesibles desde un lugar o código. Otra opción es el uso de memoria compartida entre procesos, la cual provee un medio de comunicación que incluye mecanismos para evitar redundancia. Este mecanismo es ampliamente usado en la programación multihilo, para comunicar distintos hilos de un mismo proceso entre sí.

Estas dos soluciones han sido descartadas al ofrecer un acoplamiento demasiado dependiente de la plataforma y sistema operativo sobre el que se lance el videojuego, existiendo otros mecanismos de mensajes más portables y con una mayor facilidad de uso.

Para la elaboración del proyecto se ha optado por la inclusión de un bróker de mensajería ApacheMQ el cual actuara de intermediario entre el motor de videojuegos y la plataforma multiagente, haciendo de intérprete y gestionando los mensajes entre ambos. Al ofrecer variedad de lenguajes de programación compatibles, entre ellos Java y C#, cumple los requisitos necesarios, además de poder ubicarse el servidor de mensajes fuera del ámbito local y ofrecer servicio a múltiples instancias del videojuego o jugadores.

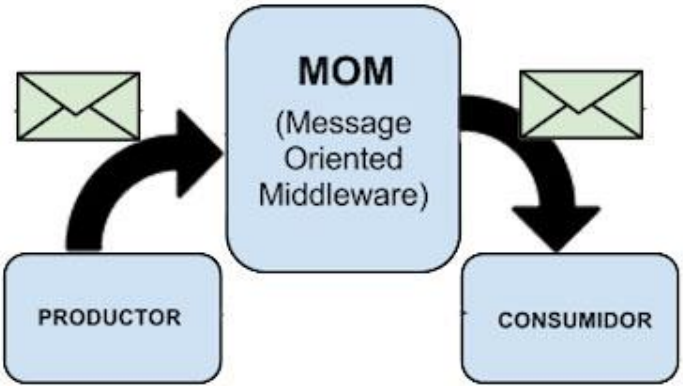

Ilustración 2.1: Flujo de mensajes

3. Análisis

En este apartado se describen las opciones elegidas para desarrollar el prototipo del juego, como componentes de una solución final y el proceso de desarrollo en el tiempo de dichos componentes, planificados siguiendo los requisitos temporales impuestos por la escuela para su realización en materia de horas dedicadas que quedan definidas al detalle en el apartado de planificación, el cual incluye el tiempo dedicado a la planificación en sí.

Para una primera aproximación a solucionar el problema, ha sido necesario entender y analizar el juego en cuestión, para lo que se ha usado distintos manuales de juego y la experiencia generada del proceso de juego a una partida tradicional en tablero físico. A través de ello ha sido posible definir la cantidad de actores implicados en el proceso de juego o la metodología de una partida entre otros factores, lo que ha sido necesario para identificar los componentes internos de ambos subsistemas y su modo de representación, las vías y métodos de comunicación o las restricciones impuestas por las reglas del juego.

3.1. Componentes del proyecto

En lo referente al motor de juegos, la elección tomada ha sido el framework de desarrollo Unity3d con scripting basado en C#, que ofrece la peculiaridad de poder desplegar el software resultante en múltiples plataformas sin grandes restricciones de arquitecturas o sistemas operativos. Esto ofrece la posibilidad de ejecutar en teoría la solución resultante en más de diez plataformas, como dispositivos móviles con los sistemas operativos más populares que acumulan en conjunto una cuota de mercado cercana al 100% (IDC Corporate USA, 2015) en la fecha de realización de este trabajo, lo que en conjunto al soporte por parte de los tres principales sistemas operativos de escritorio es posible generar soluciones con acceso casi universal. A su vez, Unity ofrece la posibilidad de adaptar el código resultante para consolas de última generación, visores web o plataformas emergentes de realidad virtual. Es por ello que se puede resumir unity como un software de creación de contenido multidisciplinario y ampliamente apoyado por la comunidad.

En cuanto a la arquitectura multiagente, JADE ha sido la opción elegida al ser un sistema asentado en cuanto a desarrollo y estar basado en java, el cual cuenta con una naturaleza multiplataforma per se. Este lenguaje fue diseñado para tener el mínimo de dependencias de implementación posibles, lo cual permite ejecutar el código escrito en diferentes plataformas sin la necesidad de ser reescrito o recompilado. La arquitectura de comunicación ofrece mensajería flexible y eficiente, donde JADE crea y gestiona una cola de mensajes entrantes ACL privados para cada agente. Los agentes pueden tener acceso a su cola a través de una combinación de varios modos: el bloqueo, la votación, el tiempo de espera y la coincidencia de patrones base. El modelo de comunicación completo FIPA es implementado por JADE y sus componentes integrados: protocolos de interacción sobre ACL, idiomas de contenido, esquemas de codificación, ontologías y por último los protocolos de transporte. El mecanismo de transporte empleado por JADE en particular, es como un camaleón ya que se adapta a cada situación, eligiendo de forma transparente el mejor protocolo disponible (Telecom Italia SpA, 2015). La mayoría de los protocolos de interacción definidos por FIPA están disponibles y se pueden crear instancias después de definir el comportamiento de las aplicaciones que dependen de cada estado del protocolo y ontología, así como el apoyo a las lenguas de contenido definidos por el usuario y ontologías que pueden ser implementadas, registrados con los agentes, y automáticamente utilizadas por el framework.

La elección de dos subsistemas con lenguajes diferentes y no compatibles a priori viene motivada por la idea de buscar un nexo de unión aplicable a una combinación similar de propuestas. Para realizar dicha sinergia, se ha optado por usar un bróker de mensajería basado en ActiveMQ el cual se encargará de hacer de mediador entre ambos. Esta solución propuesta sería capaz de ofrecer tanto una multiplexación de servicio a más de un cliente como ser agnóstica del sistema huésped de estos, al poder ubicarse tanto en un entorno local como remoto, accesible a través de internet. Para ello, ha sido necesario la instalación de un servidor de mensajes ActiveMQ y la inclusión de librerías de comunicación en Unity como en Java.

3.2. Diagramas de caso de uso

Los métodos de interacción con el sistema, quedan definidos a través de los siguientes diagramas de casos de uso, que representan el proceso de interactuación de los actores humanos con el sistema y ofrecen una mejor idea de los requisitos necesarios para su implantación.

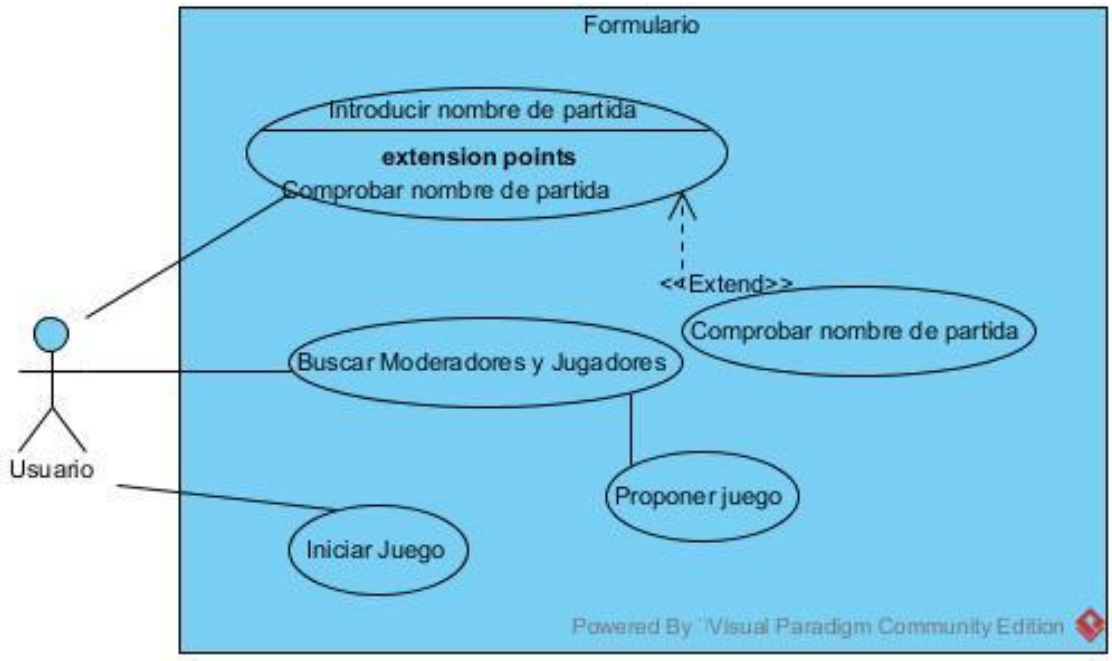

Ilustración 3.1: Diagrama de caso de uso del formulario

El formulario, será la forma de iniciar nuevos juegos, interactuando con él un usuario que no tiene por qué ser el que tome después el rol de jugador o moderador asigna un nombre a la partida, el cual es comprobado para ver si ya existe en el sistema. Habiendo validado el nombre de la partida, procede a la búsqueda de agentes de tipo moderador y jugador inscritos en las páginas amarillas de la plataforma de agentes y les propone iniciar un juego. Una vez recibido el mensaje de aceptación o rechazo de estos agentes, se ofrece la posibilidad de iniciar el juego y finalizar la vida del agente formulario.

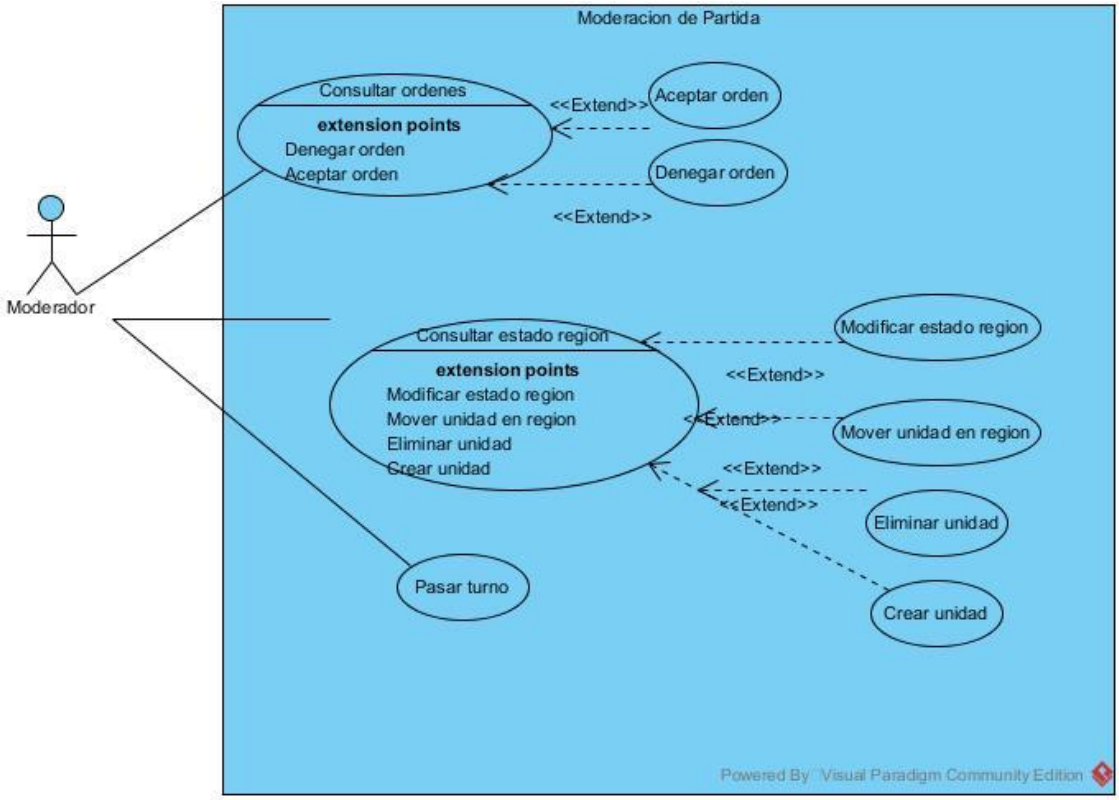

Ilustración 3.2: Diagrama de caso de uso moderar partida

El agente moderador dispondrá de información en pantalla referente al estado del juego además de la posibilidad de consultar las órdenes que los agentes jugadores le envíen. Podrá aceptar o denegar la validez de dichas órdenes accediendo al estado de cada región y modificando la nación que estará al control o las unidades que contenga. Por último, podrá pasar turno cambiando la etapa en la que se encuentre el juego, notificando con ello a los agentes jugadores implicados y enviando el estado actual del mapa al motor de juegos Unity a través del bróker de mensajería. Unity se encargará de forma periódica de consultar el estado de la cola relacionada con la partida actual y realizará los cambios oportunos en caso de que hayan mensajes con cambios.

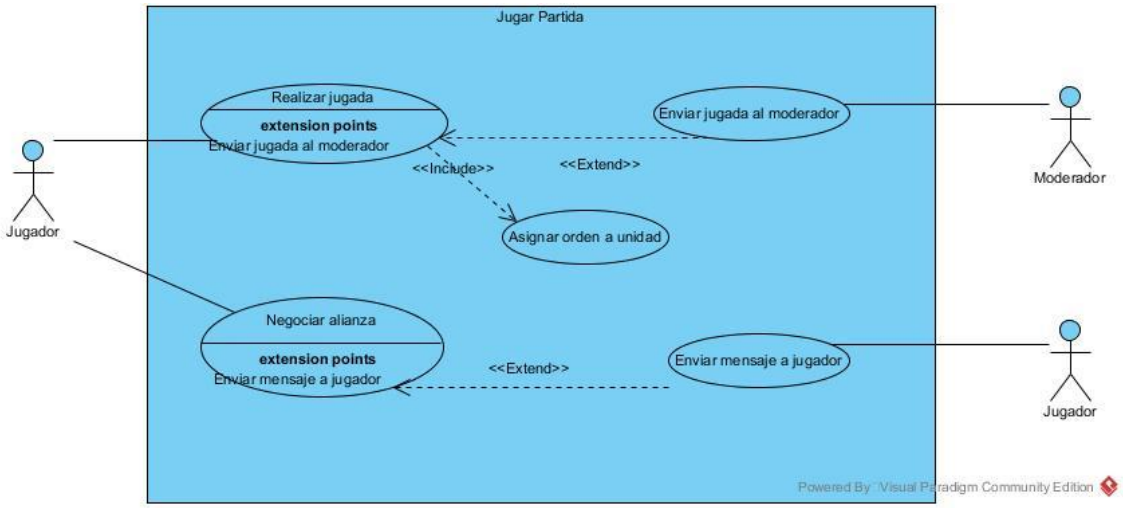

Ilustración 3.3: Diagrama de caso de uso jugar partida

El agente jugador, puede a través de su interfaz de usuario informar al moderador de las acciones que desea ejecutar sobre el conjunto de unidades en su poder. También podrá comunicarse con otros jugadores para realizar alianzas a través del envío de mensajes dentro de la plataforma de agentes. Estas comunicaciones seguirán un lenguaje de formato texto plano legible para el caso de la negociación de alianzas entre jugadores y para comunicarse con el moderador habrá que regirse por el formato de lenguaje típico del juego de mesa que contempla el reglamento de juego y que queda especificado dentro del anexo del manual de juego.

3.3. Planificación y costes del proyecto

Esta sección pretende ofrecer un resumen del proyecto desde el punto de vista temporal y económico, exponiendo las bases de los recursos empleados, en este caso el estudiante a cargo de la realización del trabajo.

3.3.1. *Planificación temporal*

Las funcionalidades necesarias para la elaboración del trabajo han sido plasmadas a través de una planificación realista y precisa. La normativa expone la necesidad de dedicar un mínimo de 300 horas ajustadas a una jornada laboral estándar. La disponibilidad de tiempo ha sido de dos meses, para desarrollar la totalidad del sistema y cumplir con los siguientes objetivos iniciales del proyecto:

1. Revisión bibliográfica de las tecnologías existentes.
2. Diseño de la arquitectura multiagente.
3. Diseño del entorno virtual para el desarrollo del juego, donde se representará las elecciones tomadas por el sistema multiagente.
4. Diseño de la interfaz de usuario y definición de interacciones.
5. Elaboración de la memoria de trabajo.

Dichos objetivos quedan analizados a través de una metodología a desarrollar, que definen una serie de tareas necesarias para la correcta realización del trabajo y que deben llevarse a cabo en tu totalidad para completar la funcionalidad:

1. Revisión bibliográfica de las tecnologías disponibles.
    a. Para el sistema multiagente se deberá justificar la elección tomada por el alumno una vez revisada las soluciones disponibles en la bibliografía.
    b. Para el motor de juegos se determinará de manera justificada la elección teniendo en cuenta la necesidad de integrar toda la solución.
2. Análisis y diseño de la arquitectura multiagente.
    a. Tipos de agentes disponibles.
    b. Conversaciones entre los agentes.
    c. Tareas de cada agente.

3. Análisis y diseño del entorno virtual.
4. Análisis y diseño de la experiencia de juego.
    a. Estudiar la representación de las jugadas de los agentes y la interconexión entre ambas arquitecturas.
5. Documentación y pruebas para la experiencia de usuario de nuestro prototipo.
6. Pruebas de estabilidad y seguridad del prototipo.
7. Manuales asociados al prototipo.
8. Generación de la memoria del trabajo realizado.

Una vez definidas las tareas, se especifica la estimación en tiempo necesario para su realización. Dicha distribución en tiempo se hace teniendo en cuenta los recursos de los que se dispone para la ejecución, en este caso un solo estudiante, por lo que la ejecución de actividades al mismo tiempo queda descartada, dando lugar a una planificación en serie o por hitos, en la que el estudiante realizará una jornada laboral dividida de 7 horas diarias con un primer turno 8:00 horas hasta 13:00 horas y un segundo turno 15:00 horas hasta 17:00 horas. Dicha jornada se repetirá cada día de la semana, obteniendo un total de 49 horas semanales trabajadas. La distribución temporal de trabajo queda detallada en la tabla 3.1, indicando la fecha de inicio y fin de cada tarea específica.

#	Tarea	Duración	Trabajo	Inicio	Fin	Precede
1	**Trabajo Fin Grado**	**74 días**	**427 hrs**	**Mie 01/07/15**	**Sab 12/09/15**	
2	Inicio	0 días	0 hrs	Mie 01/07/15	Mie 01/07/15	
3	**Planificación**	**1 días**	**7 hrs**	**Mie 01/07/15**	**Mie 01/07/15**	**2**
4	Diagrama Gantt	1 días	7 hrs	Mie 01/07/15	Mie 01/07/15	
5	**Revisión bibliográfica**	**8 días**	**56 hrs**	**Jue 02/07/15**	**Jue 09/07/15**	**3**
6	Reglas del juego	2 días	14 hrs	Jue 02/07/15	Vie 03/07/15	
7	Sistema multiagente	3 días	21 hrs	Sab 04/07/15	Lun 06/07/15	6
8	Motor de juegos	3 días	21 hrs	Mar 07/07/15	Jue 09/07/15	7
9	**Aprendizaje**	**9 días**	**63 hrs**	**Vie 10/07/15**	**Sab 18/07/15**	**5**
10	Sistema multiagente	4 días	28 hrs	Vie 10/107/15	Lun 13/07/15	
11	Motor de juegos	4 días	28 hrs	Mar 14/07/15	Vie 17/07/15	10
12	Plataforma de mensajes	1 días	7 hrs	Sab 18/07/15	Sab 18/07/15	11
13	**Análisis y diseño**	**25 días**	**175 hrs**	**Mie 22/07/15**	**Sab 15/08/15**	**26**
14	**Arquitectura multiagente**	**9 días**	**63 hrs**	**Mie 22/07/15**	**Jue 30/07/15**	
15	Tipos de agentes disponibles	2 días	14 hrs	Mie 22/07/15	Jue 23/07/15	
16	Tareas individuales	2 días	14 hrs	Vie 24/07/15	Sab 25/07/15	15
17	Conversaciones	5 días	35 hrs	Dom 26/07/15	Jue 30/07/15	16
18	**Plataforma de juego**	**11 días**	**77 hrs**	**Vie 31/07/15**	**Lun 10/08/15**	**14**
19	Entorno virtual	4 días	28 hrs	Vie 31/07/15	Lun 03/08/15	
20	Experiencia de juego	3 días	21 hrs	Mar 04/08/15	Jue 06/08/15	19
21	Interfaz de usuario	2 días	14 hrs	Vie 07/08/15	Sab 08/08/15	20
22	Interacción	2 días	14 hrs	Dom 09/08/15	Lun 10/08/15	21
23	Integración	5 días	35 hrs	Mar 11/08/15	Sab 15/08/15	18
24	Depuración	5 días	35 hrs	Dom 16/08/15	Jue 20/08/15	13
25	**Documentación**	**43 días**	**91 hrs**	**Dom 19/07/15**	**Dom 30/08/15**	
26	Memoria de seguimiento	3 días	21 hrs	Dom 19/07/15	Mar 21/07/15	9
27	Memoria final	10 días	70 hrs	Vie 21/08/15	Dom 30/08/15	26;24
28	Depósito de memoria final	0 días	0 hrs	Mie 02/09/15	Mie 02/09/15	27FF
29	Periodo de defensa	6 días	0 hrs	Lun 07/09/15	Sab 12/09/15	28

Tabla 3.1: Distribución de trabajo en tiempo

A continuación se muestra gráficamente la distribución de las tareas mediante un diagrama de Gantt que engloba la sucesión en el tiempo de forma visual del trabajo a realizar:

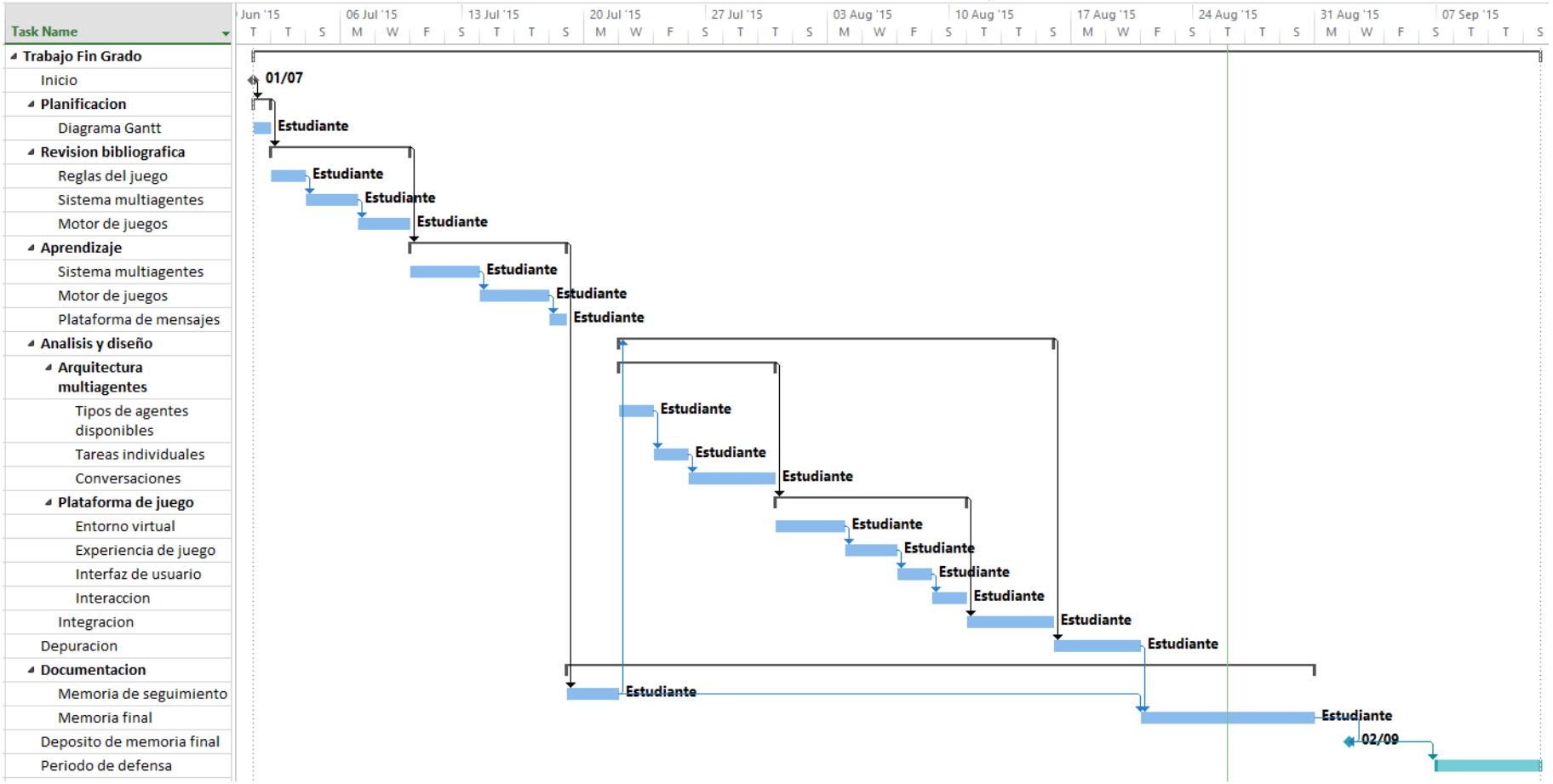

Ilustración 3.4: Diagrama de Gantt

3.3.2. *Presupuesto del proyecto*

Para el presupuesto del trabajo, se ha partido de la base de tiempo empleado expuesta en la planificación temporal del proyecto. De ello se extrae que la carga de trabajo asociada es de un total de 427 horas, las cuales representan los costes de mano de obra por parte del alumno. Para definir el coste de la hora trabajada, se ha estudiado el mercado laboral y extraído un precio medio de 14€ por hora trabajada, que aplicado al caso actual son 5.978€ de coste de mano de obra. Dicho coste queda explicado más detallado en la siguiente tabla.

Tarea	Duración	Trabajo	Recurso	Coste
Trabajo Fin Grado	**74 días**	**427 hrs**		**5.978,00 €**
Inicio	0 días	0 hrs	Estudiante	0,00 €
Planificación	**1 días**	**7 hrs**		**98,00 €**
Diagrama Gantt	1 días	7 hrs	Estudiante	98,00 €
Revisión bibliográfica	**8 días**	**56 hrs**		**784,00 €**
Reglas del juego	2 días	14 hrs	Estudiante	196,00 €
Sistema multiagente	3 días	21 hrs	Estudiante	294,00 €
Motor de juegos	3 días	21 hrs	Estudiante	294,00 €
Aprendizaje	**9 días**	**63 hrs**		**882,00 €**
Sistema multiagente	4 días	28 hrs	Estudiante	392,00 €
Motor de juegos	4 días	28 hrs	Estudiante	392,00 €
Plataforma de mensajes	1 días	7 hrs	Estudiante	98,00 €
Análisis y diseño	**25 días**	**175 hrs**		**2.450,00 €**
Arquitectura multiagente	**9 días**	**63 hrs**		**882,00 €**
Tipos de agentes disponibles	2 días	14 hrs	Estudiante	196,00 €
Tareas individuales	2 días	14 hrs	Estudiante	196,00 €
Conversaciones	5 días	35 hrs	Estudiante	490,00 €
Plataforma de juego	**11 días**	**77 hrs**		**1.078,00 €**
Entorno virtual	4 días	28 hrs	Estudiante	392,00 €
Experiencia de juego	3 días	21 hrs	Estudiante	294,00 €
Interfaz de usuario	2 días	14 hrs	Estudiante	196,00 €
Interacción	2 días	14 hrs	Estudiante	196,00 €
Integración	5 días	35 hrs	Estudiante	490,00 €
Depuración	5 días	35 hrs	Estudiante	490,00 €
Documentación	**43 días**	**91 hrs**		**1.274,00 €**
Memoria de seguimiento	3 días	21 hrs	Estudiante	294,00 €
Memoria final	10 días	70 hrs	Estudiante	980,00 €
Depósito de memoria final	0 días	0 hrs	Estudiante	0,00 €
Periodo de defensa	6 días	0 hrs		0,00 €

Tabla 3.2: Distribución de costes en etapas

A los honorarios del proyecto en materia de mano de obra, hay que añadir los costes de ejecución material para la realización de dicho proyecto, que incluyen el arrendamiento de una máquina virtual en un servidor que albergue el sistema de mensajería por un periodo de un año a razón de 200€ el primer año. Material de consulta como manuales de juego o libros especializados para las etapas de aprendizaje y revisión bibliográfica ascienden a un total de 300€.

Junto a los costes de ejecución material, el proyecto conlleva unos costes indirectos asociados de dicha actividad que ascienden al 12% presupuestado en materia a horas de horas de trabajo, en este caso:

$$5.978\ € \times 0,12 = 717,36\ €$$

El material fungible relacionados a los gastos de entrega del proyecto ascienden a 100€ para gastos de impresión y grabación en medios.

Con todo ello, el presupuesto queda representado a continuación.

Presupuesto del proyecto

1. Honorarios del proyecto
    - 427 horas a 14€ / hora……..5.978€
2. Ejecución material
    - Servidor de mensajes……...200€
    - Material de consulta……..300€
3. Gastos generales
    - 12% sobre honorarios…….717,36€
4. Material fungible
    - Gastos de grabación e impresión……..100€
5. Subtotal del presupuesto
    - Subtotal……..7.295,36€
6. IVA Aplicable
    - 21% subtotal del presupuesto……1.532,03€
7. Total del presupuesto
    - Total……**8.827,39€**

4. Diseño

El diseño del sistema viene dividido en tres subsistemas que interactúan entre sí a través del uso de mensajería. El cliente gráfico representado por Unity3D será el encargado de representar visualmente el estado del tablero y ofrecer una retroalimentación al usuario del estado del juego. Para representar dicho estado, se hace uso de mensajes que representan el estado total del tablero en un instante dado, definido al inicio del mensaje. Estos mensajes son gestionados por el bróker de mensajería ActiveMQ el cual los gestiona a modo de colas, almacenando los mensajes que entran a una cola por parte de un productor hasta el momento que un consumidor, en este caso Unity decide acceder al mensaje. De este modo no es crítico mantener un enlace de comunicación permanente, pudiendo reducir los accesos al servidor solo cuando se detecte el cambio de fase o de forma esporádica. Los mensajes son generados por parte de la plataforma de agentes, más concretamente por el agente moderador, el cual será el encargado de gestionar en todo momento la representación binaria del tablero. El proceso de comunicación queda explicado a través de la siguiente ilustración.

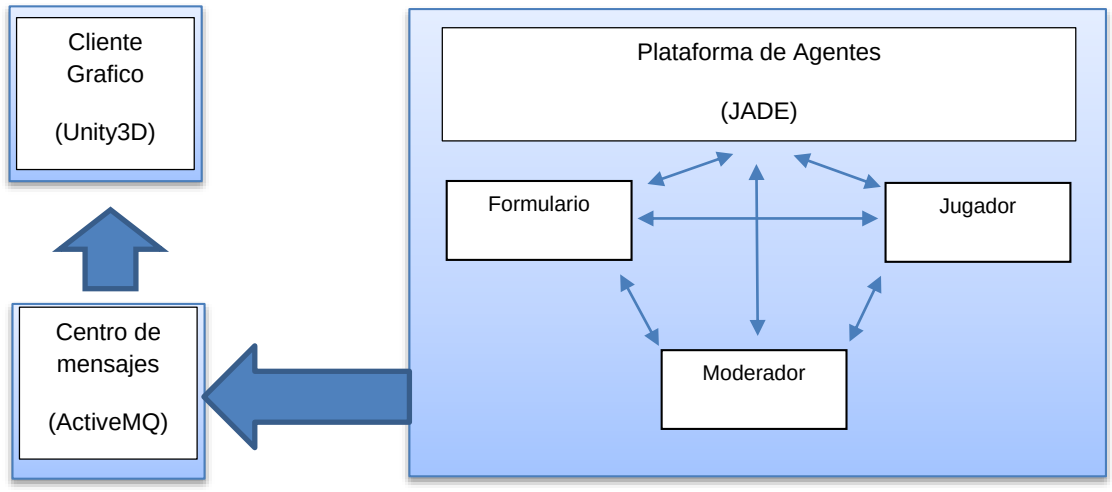

Ilustración 4.1: Arquitectura del sistema

Los componentes quedan comunicados siguiendo el flujo de comunicación que fija el sentido de las flechas. Para el caso de la comunicación JADE-Unity, se ha optado por un sentido simplex de la comunicación, al ser necesario unity únicamente para representar el estado del mapa y no necesitar responder al agente que envía dicha información. Dentro de la plataforma de agentes, es posible comunicar a todos los agentes entre ellos de forma sencilla al usar la inscripción en páginas amarillas y pertenecer a un mismo contenedor.

La representación de las regiones que conforman los distintos territorios a ocupar queda definida como una estructura de grafo simple no dirigido, donde cada una de las regiones representa un nodo del grafo. Esta forma de representación simplifica el proceso de codificación interno, más aun si se hace uso de librerías de terceros que aporten algoritmos de gestión de nodos o cálculo de caminos entre nodos.

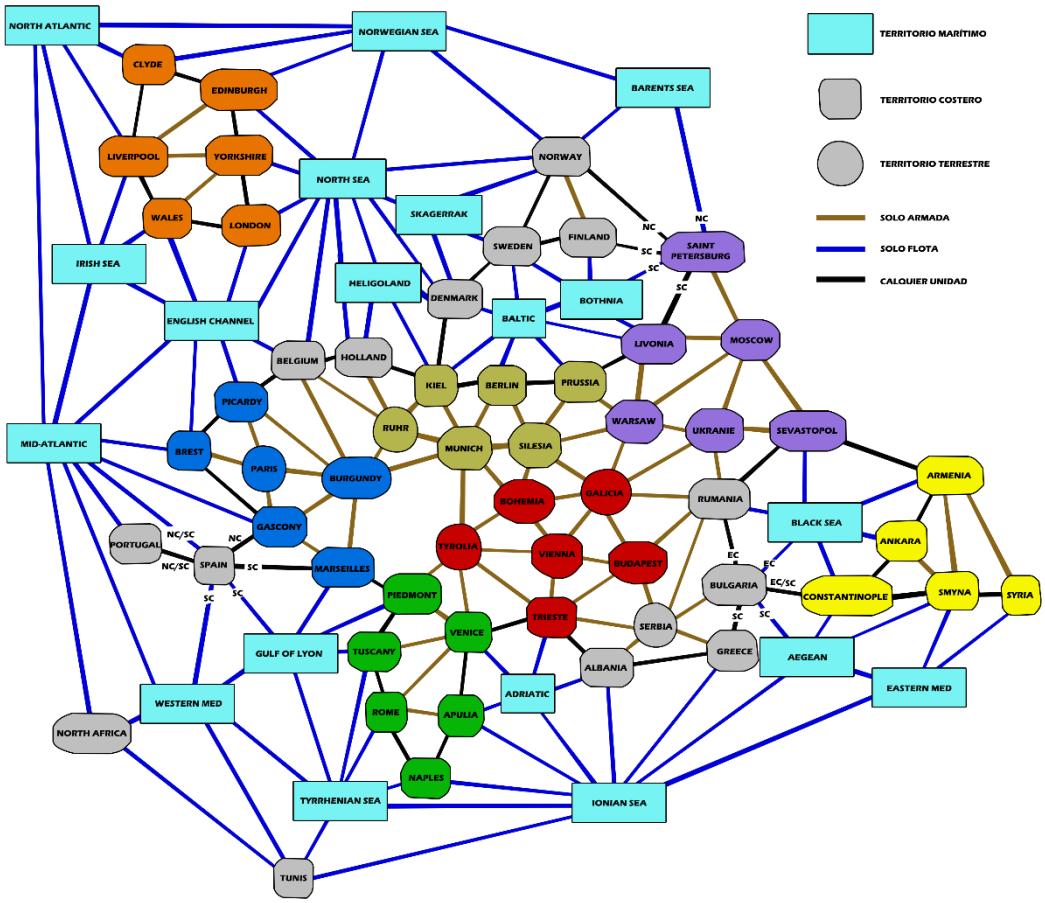

Ilustración 4.2: Grafo de regiones

Estas regiones pueden ser transitables por las unidades, aunque las rutas quedan restringidas a un tipo de unidad específica según el tipo de camino que interconecte cada uno de los nodos. Es por ello que se han creado dos subgrafos específicos para cada una de las unidades, representando los caminos que pueden tomar y la unidad que puede transitarlos.

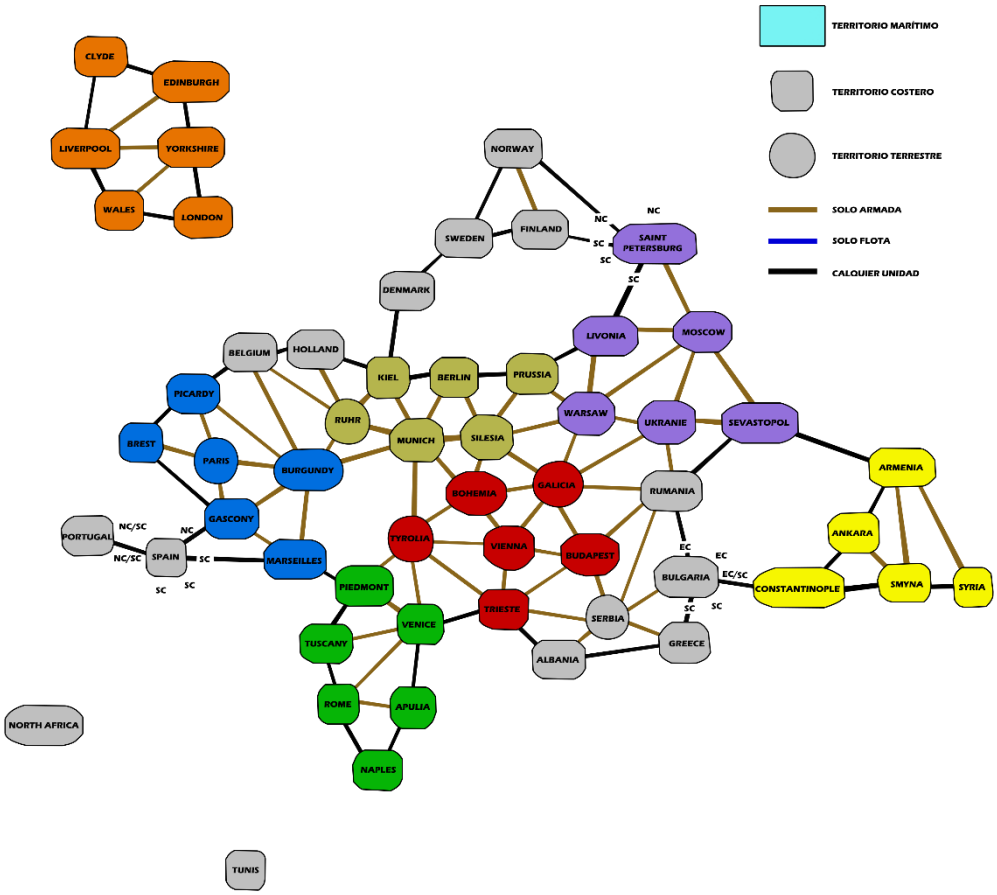

Ilustración 4.3: Grafo transitable por ejército

Este subgrafo, representa los nodos por los que las unidades de tipo ejército de los jugadores podrán desplazarse. Existen nodos aislados, a los que no se puede mover directamente dependiendo del origen, pero pese a ello se podrá acceder a través de la orden transporte de las unidades flota si están disponibles en los nodos anexos.

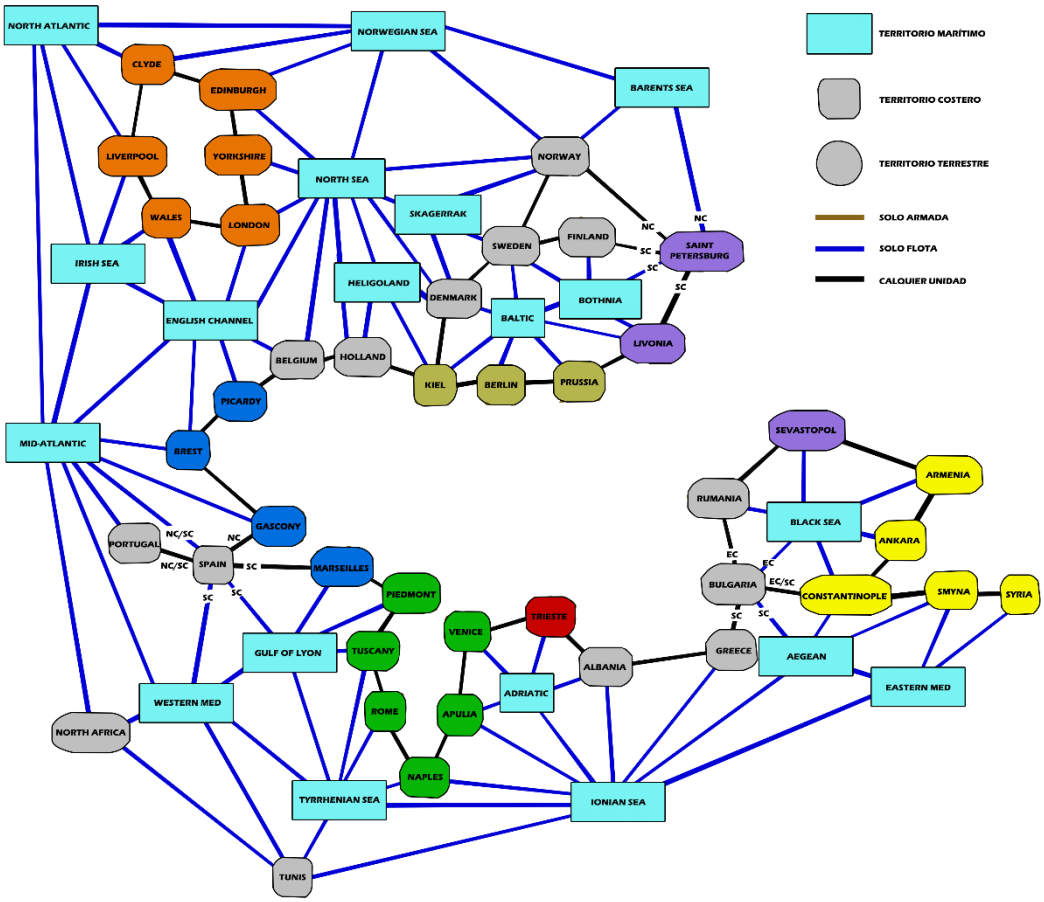

Ilustración 4.4: Grafo transitable por flota

Este subgrafo representa la capacidad de movimientos de las unidades de tipo flota. La combinación de ambos subgrafos representa la estructura de movimientos posibles a realizar por el usuario, teniendo en cuenta que cada unidad puede desplazarse con un límite de longitud de camino igual a uno. Para la codificación de esta representación por medio de grafos, se ha optado por usar la librería externa JGraphT, la cual aporta teoría matemática de grafos y algoritmos a la vez de ser gratuita (Naveh, 2015). El uso de grafos aporta una ayuda para el actor que interactúe con el agente moderador, de forma que los movimientos quedan restringidos a los accesibles desde un desplazamiento de longitud de lado uno, desde el origen de la unidad. De este modo se incrementa la asistencia al usuario y se reducen posibles errores generados por una lectura errónea del mapa.

4.1. Diagramas de secuencia

En este apartado se muestran los diagramas de secuencia de la interacción relacionada con los distintos agentes, que representan el flujo de la comunicación y las acciones efectuadas por cada agente durante su vida. Seguidamente se irán describiendo cada uno de ellos.

- Creación de una partida.

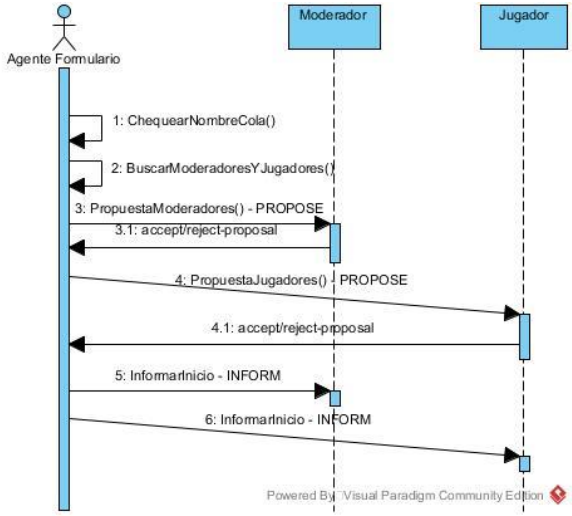

Ilustración 4.5: Diagrama de secuencia de creación de una partida

En este diagrama se muestra el proceso de creación de una nueva partida y el flujo de comunicación de la información al resto de agentes. El proceso inicia con la petición de un nombre de partida y la comprobación de existencia de dicho nombre entre las colas del servidor de mensajes. A continuación se busca en las páginas amarillas los agentes disponibles de cada tipo y a través de mensajes de tipo PROPOSE, el formulario envía una propuesta de juego a cada uno de ellos. En caso de existir un mínimo de 7 respuestas positivas por parte de los jugadores y un moderador, se dispone a iniciar el juego, informando de ello a cada uno de los agentes implicados.

- Jugar una partida.

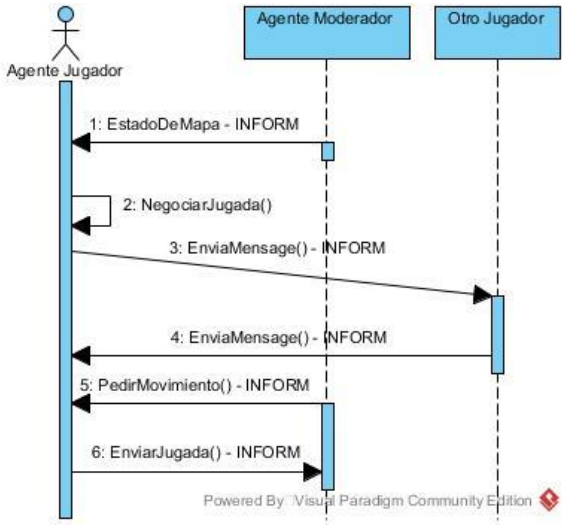

Ilustración 4.6: Diagrama de secuencia de juego de una partida

Una vez iniciado el juego por parte del formulario, el agente moderador será el encargado de gestionar el proceso de la misma. Para ello informará a los jugadores del estado del mapa y estos a través de mensajes a otros jugadores podrán negociar alianzas para ciertas órdenes concretas. Una vez finalizada la ronda de negociación, el agente moderador procede a pedir las órdenes relativas a la etapa de movimiento concreta.

- Moderar una partida.

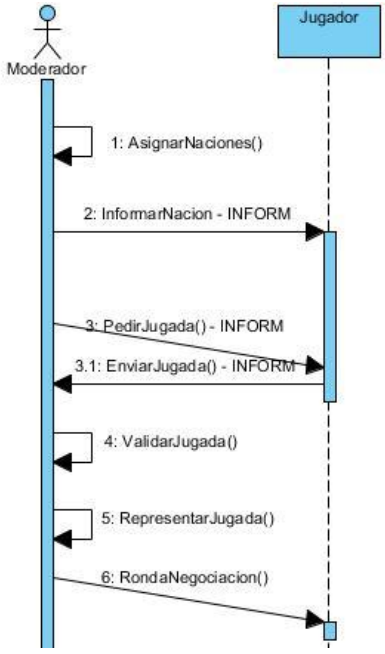

Ilustración 4.7: Diagrama de secuencia de moderación de una partida

El agente moderador a través de una simulación de tirada de dados asigna las naciones correspondientes a los jugadores implicados. Tras ello, informa de los resultados a estos para que puedan empezar a crear alianzas. Tras un periodo inicial de negociación, se suceden etapas de petición de jugada y negociación, entre las cuales han de validarse las jugadas enviadas por los jugadores y representar el estado del mapa tras los movimientos efectuados.

4.2. Descripción de interfaz de usuario

La interfaz de usuario del prototipo queda dividida en los tres módulos componentes: la plataforma de agentes, el motor de juego y el bróker de mensajería. Para el caso de ActiveMQ, la interfaz de uso es inapreciable ya que es gestionada por código a través del envío y recepción de mensajes por parte de los agentes y motor de juegos.

Para el caso de la plataforma de agentes, podemos ver desde el gestor visual de agentes de JADE, la existencia y la posibilidad de crear nuevos agentes a través de ella, que serán capaces de comunicarse libremente con el resto de agentes de su contenedor.

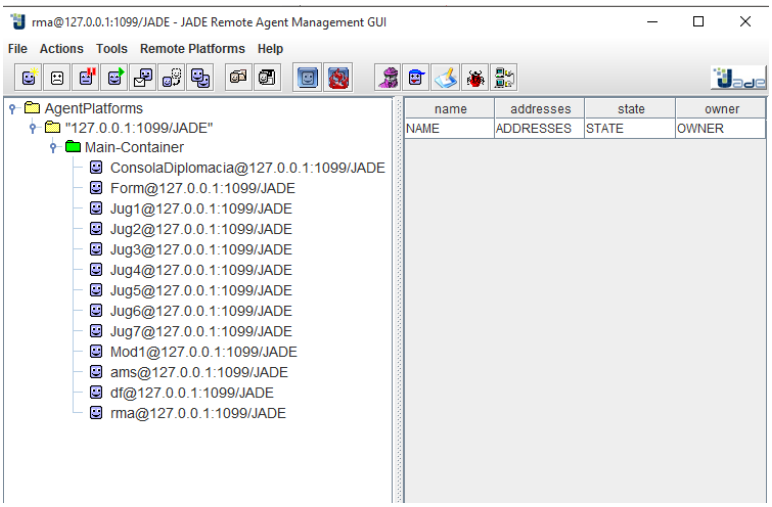

Ilustración 4.8: Plataforma de agentes JADE

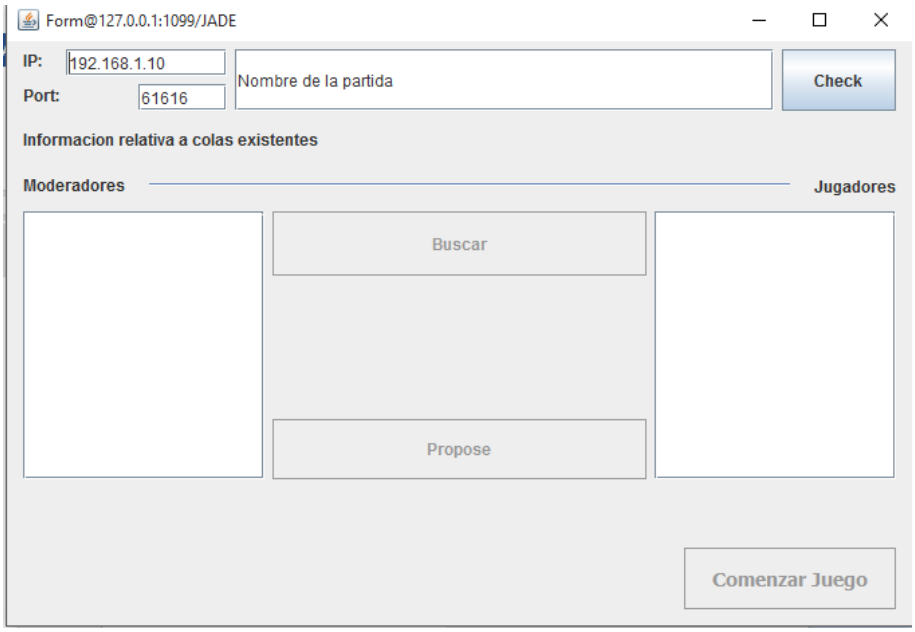

Ilustración 4.9: Formulario de inicio de partida

Cuando iniciamos un agente formulario, vemos que lo primero que pide al usuario es la asignación de un nombre para poder crear una partida junto a la dirección IP y el puerto en el que está instalado el bróker de mensajería. El resto de campos permanecerán inaccesibles hasta que se haya comprobado con el servidor de mensajería la idoneidad del nombre aportado y se reserve la cola para la partida. A continuación a través del botón de búsqueda, se hace un listado de todos los agentes disponibles e inscritos en las páginas amarillas de la plataforma de agentes, pudiendo a continuación proponerles el inicio del juego. En caso de recibir respuesta positiva del número necesario de agentes para iniciar una partida, se habilita la opción de iniciar juego, con la que se informará a todos los agentes implicados del proceso a seguir. Con esto, acaba la vida del agente formulario y la gestión de la partida pasa a ser del agente moderador.

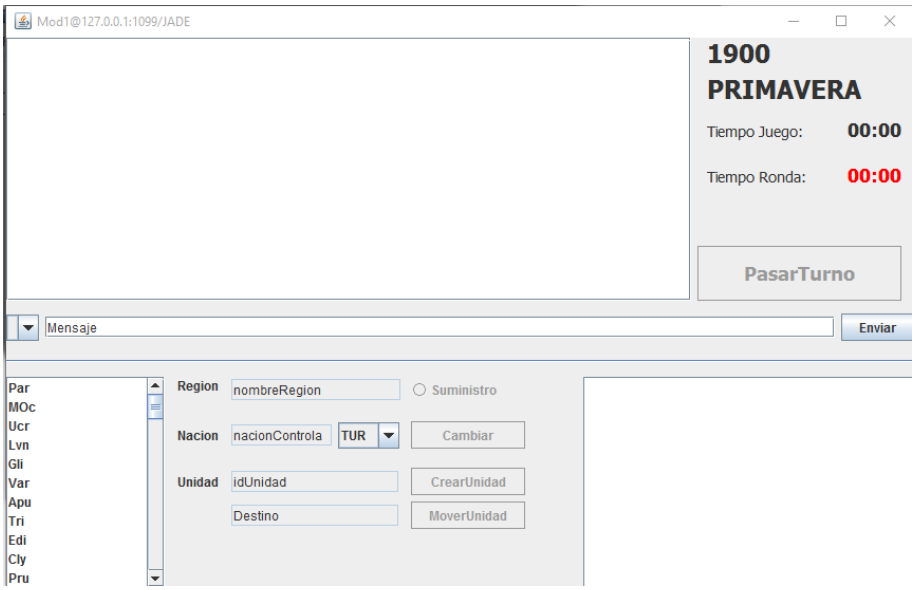

Ilustración 4.10: Ventana de moderación de partida

El moderador dispone de una ventana superior en la que irá recibiendo las ordenes de cada jugador, además de la posibilidad de comunicarse con estos para dar respuesta a un mensaje concreto como lo haría en el transcurso de una partida real. Incluye un botón para pasar el turno actual de forma secuencial por etapas e información del año y estación actual en la que se encuentra el juego. A través de un listado, el moderador puede seleccionar una región de entre todas las existentes para cambiar el control de un territorio o gestionar unidades dentro de este. Los mensajes generados por el agente ayudante a la hora de realizar dichas acciones quedan reflejados a la derecha de la ventana, separando órdenes de usuario de mensajes del sistema.

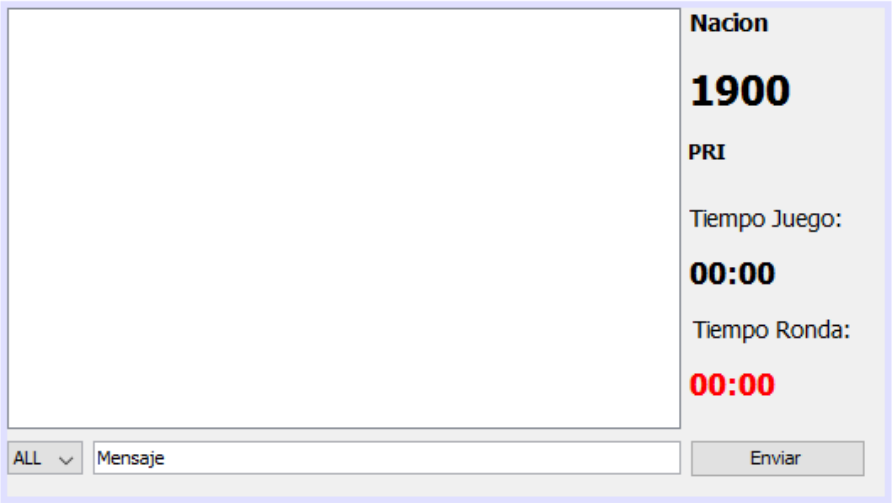

Ilustración 4.11: Ventana de mensajes de Jugador

Los jugadores disponen de una ventana simple para comunicarse como lo harían en una partida física a través de órdenes escritas. Al poder seleccionar como destino de su mensaje otros jugadores, estarían capacitados de negociar con ellos alianzas o movimientos relacionados con la partida. Los mensajes provenientes del sistema, moderador, u otros jugadores se mostrarán en esta ventana acompañados de una marca de tiempo y el origen concreto. Además, aparece presente dos contadores, uno para el tiempo de juego que almacenará y mostrara el tiempo que lleva la partida jugada y uno regresivo que se encargará de mostrar el tiempo restante hasta la próxima ronda. Estos contadores de tiempo son sincronizados cada vez que llega un mensaje del moderador respecto a la gestión de la partida, el cual lleva incrustado los valores específicos que deberán tomar.

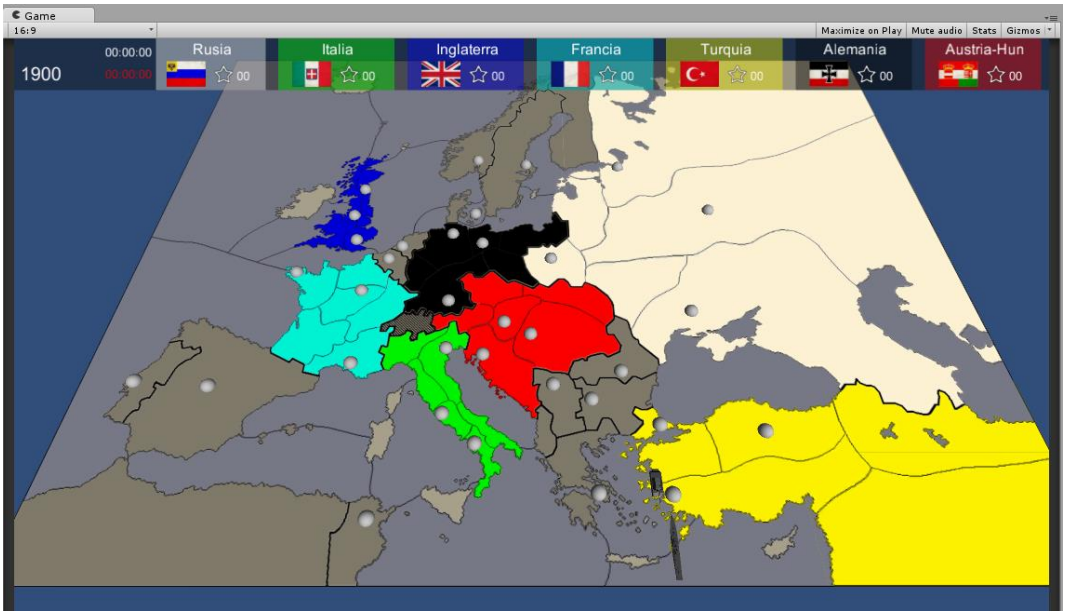

Ilustración 4.12: Representación de Mapa en Unity

El tablero de juego queda representado en unity como un mapa en el que cada región se abstrae del resto y se colorea de forma independiente para representar la nación que la controla. Los centros de suministro quedan representados como esferas que se iluminan con el color de la nación que controla el territorio y las unidades a través de modelos 3d de tanque para representar la unidad ejército y de barco para la unidad flota.

En la barra superior se ofrece un HUD con información relacionada al estado de la partida, como el año y la estación en la que se encuentra el juego y los centros de suministro controlados por cada una de las naciones controladas por los jugadores.

5. Resultados

En el desarrollo de cualquier sistema software, la realización de pruebas juega un papel fundamental a la hora de evaluar la calidad de la solución generada y evaluar su correcto funcionamiento y solventar los errores encontrados en el proceso. Para ello, se han realizado pruebas referentes a la experiencia de usuario a la hora de interactuar con el sistema junto a pruebas de estabilidad y seguridad del prototipo. Estas pruebas han desembocado en una etapa de depuración de errores encontrados.

Las pruebas de experiencia de usuario se han realizado con sujetos cercanos a través de un test de usabilidad para el cual se han expuesto una serie de órdenes a ejecutar y se ha cuantificado el tiempo empleado en encontrar la solución. A través de los comentarios por parte de los usuarios, se ha modificado la interfaz convenientemente para satisfacer las sugerencias. De esta forma se pueden detectar fallos de accesibilidad al contar con la experiencia de uso de personas ajenas al proceso de programación y que definen a un usuario general. Algunos de los comentarios surgidos han dado como resultado la inclusión de distintos elementos en la interfaz de usuario como es el caso de contadores de tiempo regresivo y una segunda consola para el moderador para separar los mensajes internos.

Las pruebas de estabilidad y seguridad tenían como objetivo buscar y corregir posibles fallos generados por pérdida de conectividad y acceso desde múltiples orígenes múltiple a datos compartidos. Para ello se busca comprobar si la comunicación entre las distintas partes del sistema es la esperada y que el orden de ejecución fuese el correcto. En cada fase relacionada con el desarrollo se ha tenido especial cuidado en implementar una solución que cuide todos estos aspectos, pero ha sido especialmente durante el periodo de pruebas donde gracias a tener una versión completa del sistema que ha sido posible detectar errores de forma más fácil.

Tras finalizar el periodo de pruebas se puede concluir que la integración de ambos subsistemas ha sido llevada a cabo de forma correcta y la solución aportada satisface las necesidades expuestas en los requisitos del sistema para considerarse un prototipo.

6. Conclusiones

A continuación se exponen las distintas conclusiones surgidas tras la elaboración del presente trabajo, como se ha llevado a cabo y una serie de ideas futuras a implementar.

6.1. Descripción del desarrollo del proyecto

La elaboración del presente trabajo ha constituido el objetivo principal por parte del estudiante para culminar sus estudios en ingeniería informática referente a la asignatura trabajo fin de grado. Para ello, se ha seguido una metodología de trabajo secuencial sin solapamiento, al disponer únicamente de una persona para soportar la carga de trabajo generada y con una disponibilidad y dedicación plena. Las fases de desarrollo del proyecto y su ubicación en el tiempo quedan expuestas en el apartado de planificación, las cuales han sido seguidas rigurosamente. El trabajo realizado ha sido posible gracias a la supervisión del estudiante por parte de los coordinadores implicados, quienes han guiado y aconsejado al estudiante para la elaboración del mismo, llevándole al punto que define esta memoria. El software usado para ello ha sido amplio y variado, optando por licencias libres o gratuitas.

El entorno virtual ha sido realizado como se ha indicado anteriormente con Unity3D, más concretamente la versión 5.1, última disponible al comienzo del desarrollo de este trabajo. Para la implementación de los scripts de juego, se ha utilizado el lenguaje C# y los IDEs MonoDevelop integrado con la instalación de Unity y VisualStudio 2013 que facilitan el proceso de desarrollo y programación de los scripts. La gestión de mensajes se ha llevado a cabo del lado de unity gracias a la librería específica de ActiveMQ con soporte para .NET ApacheNMSActiveMQ en la versión 1.7.0 la librería correspondiente al soporte de lenguaje Mono2.0, usado como intérprete de lenguaje por Unity.

Los agentes virtuales son el sistema principal del proyecto, que representa la base lógica que sustenta las acciones de los jugadores. Para conseguir implementar los agentes implicados, se ha usado el IDE Netbeans en su versión 8.0.2, sobre el que se ha creado un proyecto JAVA basado en dependencias MAVEN, gracias a las cuales ha sido posible incluir las librerías externas de codificación de grafos JGraphT en su

versión 0.9.1 y librerías de comunicación ActiveMQ para JAVA con versión 5.9.0. Para la plataforma gestora de agentes, se ha usado JADE en su versión 4.3.3 incluida también como una dependencia del repositorio MAVEN.

El servidor de mensajes ActiveMQ se ejecuta en su versión 5.9.0 sobre un equipo que tiene el tiempo de ejecución JAVA versión 1.8.0_51, necesario para el funcionamiento de ActiveMQ.

6.2. Conclusiones del trabajo realizado

Los agentes virtuales son ampliamente usados en el día a día en todas sus variantes pero mayoritariamente los agentes inteligentes en el ámbito de la robótica o en transacciones virtuales gracias al razonamiento que la inteligencia artificial les brindan. El uso de agentes software se prevé que siga incrementándose en un futuro próximo como una forma de solventar tareas modular y con capacidades de simulación de comportamiento humano gracias a los avances que no paran de llegar por parte de la comunidad científica en donde disciplinas como ciencias de la computación dejan de comportarse como una ciencia aislada y se complementan con otras para por ejemplo integrar modos de aprendizaje adaptados de estudios biológicos o una mejora en la comunicación al usuario fruto de la psicología cognitiva.

Este trabajo ha analizado diferentes opciones respecto a plataformas existentes tanto en sistemas multiagente como motores gráficos, y la utilidad que surge de la unión de ambas plataformas. Cada vez son más los juegos clásicos portados del típico tablero físico a una plataforma digital y son estos juegos clásicos, un referente entre los más demandados por los usuarios para emplear su tiempo libre. Aunque se haya analizado un caso de juego especifico en este trabajo, al igual que los motores de videojuegos son capaces de representar juegos de un mismo tipo cambiando la forma en la que están programados, otro juego que comparta las bases con Diplomacia, seria fácilmente adaptable con solo cambiar el comportamiento interno de los agentes y la representación del tablero. Esta es la elasticidad que se obtiene a través de la integración de un sistema multiagente a la solución, que junto a la portabilidad y mutabilidad de los agentes, los hace ser una de las soluciones tecnológicas en auge hoy en día.

Al elegir dos plataformas de desarrollo diferentes para solucionar el problema, se pone de manifiesto la necesidad que existe para comunicar distintos sistemas entre sí, la cual es llevada un paso más allá al estar ambas plataformas basadas en lenguajes distintos. La especialización y reutilización que marca las pautas de las tecnologías de la información queda plasmada a la hora de estudiar las soluciones existentes, donde podemos ver como las restricciones aparentemente impuestas por la elección de una opción en una de las partes afecta a la idoneidad de la otra. La gran mayoría de las veces esas limitaciones vienen lastradas por la falta de flexibilidad de los sistemas existentes, los cuales quedan reservados para ciertas plataformas o lenguajes concretos. Estas limitaciones se acentúan aún más a la hora de operar con subsistemas con distintas limitaciones, pues el resultado tiende a contener las restricciones de ambos.

Como conclusión final, este trabajo ha abordado con éxito la creación de un prototipo para la adaptación de un juego de mesa a una plataforma multiagente dotándolo de una interfaz gráfica basada en un motor de juegos y la integración de ambos a través de una solución multiplataforma. Este trabajo a su vez aporta diferentes ideas y futuras líneas de investigación detalladas en el siguiente apartado que facilitarán la tarea de conseguir una versión funcional final continuando el trabajo referente al prototipo realizado y expuesto en esta memoria.

6.3. Trabajos futuros

En este apartado se plasman las posibles líneas de trabajo futuro en cuanto a mejora de la solución aportada en este trabajo. Al tratarse de un prototipo la funcionalidad final se ve limitada, con lo que las siguientes ideas aportan un punto de partida para continuar el trabajo actual realizado y añadir funcionalidades que ayuden a alcanzar una etapa madura de desarrollo y la posibilidad de distribuir finalmente el sistema.

Aportar de autonomía al agente moderador. En lugar de basar el comportamiento del agente en un ayudante de la parte humana se puede programar para que realice las comprobaciones de las jugadas de forma autónoma e independiente. Esto sería posible definiendo con anterioridad el lenguaje de comunicación entre ambos como parte de la ontología o limitando las opciones disponibles por parte del jugador a la hora de comunicarse con el moderador.

Integrar la interfaz del agente jugador dentro del motor de juegos. A través del centro de mensajería ActiveMQ, se puede establecer un doble sentido (dúplex) de la comunicación permitiendo a Unity comunicarse con JADE e integrar las acciones de usuario como parte de la interfaz de juego que representa el mapa. De esta forma a la vez que se ofrece información del estado del mapa al jugador, las opciones disponibles quedan fácilmente identificables y su interacción resulta más natural al usar elementos visuales enriquecidos u otros métodos de entrada, lo que reduce la carga cognitiva y el proceso de aprendizaje para el uso de la solución.

Dotar de inteligencia artificial al agente jugador. Programando al agente jugador para que se comporte de forma autónoma e inteligente reaccionando a los estímulos generados por otros jugadores, podríamos integrar agentes no humanos en la partida de forma transparente al resto de jugadores, ya sea para crear un juego en el que no se alcanzan el número mínimo de jugadores al no haber personas disponibles o para simular el comportamiento de estos en caso de que abandonen la partida. La mayor dificultad vendría definida a la hora de realizar negociaciones entre jugadores no humanos y humanos, a ser el procesamiento de lenguaje natural un campo de la inteligencia artificial aun por mejorar en cuanto a eficacia computacional.

Añadir variaciones del juego. Existen multitud de variantes del juego diplomacia comercializadas: Machiavelli, Kamakura, Colonial Diplomacy, Hundred, Ard-Rí… Se distinguen por incluir distintas regiones y naciones, en definitiva, una variación del tablero y la inclusión de alguna otra variable como el dinero, pero manteniendo suficientes similitudes para ser considerados variantes de Diplomacia.

Posibilitar un juego de menos jugadores. Aplicar las reglas de juego para permitir iniciar un juego con menos de 7 jugadores. Las regiones correspondientes a naciones no controladas por ningún jugadores quedan pues repartidas entre el resto de los presentes conforme a las reglas expuestas en el manual de juego.

Seguimiento de varias partidas al mismo tiempo. Ofrecer la posibilidad a los agentes jugadores y moderador de participar en más de un juego simultáneo, de forma que a través de un identificador de partida puedan diferenciar el juego en el que se encuentran en un momento puntual.

Recreación de partidas ya acabadas. A través de un histórico de movimientos almacenados en una estructura de datos apropiada, sería posible recrear los movimientos de juego una vez acabado para su estudio. Esta recreación podría ir acompañada de estadísticas de la partida o datos de interés de cara a un análisis a posteriori.

Inclusión de animaciones y elementos multimedia. Aumentar la carga cognitiva asociada al entorno modelado con unity a través de animaciones y transiciones que doten a la escena de un mayor realismo. Simular las batallas realizadas entre naciones y aportar elementos sonoros acordes a los acontecimientos para aumentar el nivel de inmersión.

Apéndice A: Descripción de la forma de usar la aplicación

Para el correcto funcionamiento de la aplicación, es necesario con anterioridad instalar las dependencias java necesarias y expuestas. Una vez cumplido con el requisito de motor de ejecución, ha de configurarse un servidor ActiveMQ en la máquina destinada para ese efecto y gestionar la apertura de puertos en caso de que sea un dominio distinto al que usarán la plataforma de agentes y el motor de videojuegos.

Los agentes se gestionarán a través de la plataforma JADE incluida dentro del ejecutable, la cual será la primera ventana que aparecerá al abrirlo. Para el correcto funcionamiento del prototipo, ha de poblarse el sistema con al menos un agente formulario encargado de iniciar las partidas, uno o más agentes moderador que gestionarán el correcto funcionamiento del juego cambiando etapas y modificando el mapa, y por ultimo siete o más agentes jugadores para poder iniciar una partida al juego Diplomacia. En caso de existir más agentes de los necesarios, el formulario elegirá entre estos por orden de respuesta.

Una vez lanzados los agentes jugadores y moderador tras la inicialización de la partida por parte del formulario, puede procederse al juego de una partida, siendo el formulario el encargado de la inicialización de la misma. Para la correcta sucesión de una partida, el moderador se encargará de gestionar el estado del mapa a través de la interfaz y alternar las etapas de negociación y movimiento.

Durante las etapas de negociación, los jugadores son libres de comunicarse entre ellos para pactar alianzas o definir estrategias. Estos mensajes son comunicados a través de la plataforma de agentes y con un destino único. Al alcanzarse una etapa de movimiento fijada por el moderador, los jugadores disponen de un límite de cinco minutos para hacer llegar al moderador el mensaje con la orden deseada, de lo contrario se entenderá como desistido el movimiento. Estas órdenes de movimiento mantendrán un formato fijo que empezará por la letra del tipo de unidad que se desee mover Ejercito “E” o Flota “F” seguido por la abreviatura de la región de origen y destino.

Unity actuará como un visor de las partidas en juego consultando las colas activas en el bróker y mostrando el estado de cada una de ellas tras seleccionarla. El mapa europeo representa el tablero y las distintas regiones que pueden conquistar los jugadores así como un HUD que aporta información relativa al estado de la partida: tiempo de juego, estación, estado de los jugadores,…

Se puede cambiar de partida con solo seleccionar la cola correspondiente a dicha partida, pero cada vez que se consume un mensaje relacionado con el estado del mapa, éste no puede ser consultado de nuevo, dado el carácter efímero de los mensajes empleados por ActiveMQ en sus colas.

Apéndice B: Abreviatura de cada región

- TURQUIA (Amarillo)

- Sir Siria
- Arm Armenia
- Smi Smirna
- Ank Ankara
- Con Constantinopla

- RUSIA (Blanco)

- Seb Sebastopol
- Ucr Ucrania
- Var Varsovia
- Mos Moscú
- Lvn Livonia
- SPt San Petersburgo

- ITALIA (Verde)

- Nap Nápoles
- Apu Apulia
- Rom Roma
- Tos Toscana
- Ven Venecia
- Pia Piamonte

- ALEMANIA (Negro)

- Mun Munich
- Sil Silesia
- Pru Prusia
- Ber Berlin
- Ruh Ruhr
- Kie Kiel

- FRANCIA (Azul claro)

- Mar Marsella
- Gas Gascuña
- Bor Borgoña
- Par París
- Bre Brest
- Pic Picardía

- INGLATERRA (Azul oscuro)

- Lon Londres
- Gal Gales
- Yor Yorkshire
- Liv Liverpool
- Edi Edimburgo
- Cly Clyde

- AUSTRIA-HUNGRÍA (Rojo)

- Bud Budapest
- Gli Galicia
- Tri Trieste
- Vie Viena
- Boh Bohemia
- Tir Tirol

- NEUTRALES

- Gre Grecia
- Bul Bulgaria
- Ser Serbia
- Alb Albania
- Rum Rumania

- Fin Finlandia
- Sue Suecia
- Nor Noruega
- Din Dinamarca
- Hol Holanda
- Bel Bélgica
- Esp España
- Por Portugal
- Afr África del Norte
- Tun Tunez

- MARITIMOS:

- MBa Mar de Barents
- MNo Mar de Noruega
- GBo Golfo de Botnia
- Bal Mar Baltico
- Ska Skagerrak
- MNt Mar del Norte
- Hel Helgoland
- CMa Canal de la Mancha
- AtN Atlantico Norte
- MIr Mar de Irlanda
- AtC Atlantico Central
- MOc Mediterraneo occidental
- GLe Golfo de Leon
- MTi Mar de Tireno
- MJo Mar Jonico
- MAd Mar Adriatico
- MOr Mediterraneo oriental
- MEg Mar Egeo
- MNe Mar Negro

Bibliografía

Telecom Italia SpA. (30 de Agosto de 2015). *JADE*. Obtenido de JADE: http://www.jade.tilab.com/

Allan, R. (2009). *Survey of Agent Based Modelling and Simulation Tools.* Daresbury, Warrington: STFC.

Bellifemine, F., Poggi, A., & Rimassa, G. (2001). JADE: a FIPA2000 compliant agent development environment. *Proceeding AGENTS '01 Proceedings of the fifth international conference on Autonomous agents* (págs. 216-217). New York: AMC.

E. Hutchins, J. H. (1985). Direct manipulation interfaces. *Human-Computer Interaction*, 311-338.

Ferber, J. (1999). *Multi-Agent System: An Introduction to Distributed Artificial Intelligence.* Harlow: Addison Wesley.

Frijters, J. (30 de Agosto de 2015). *IKVM*. Obtenido de IKVM: http://www.ikvm.net/

Gregory, J. (2009). *Game Engine Architecture.* CRC Press.

Halverson, C. (2002). Activity Theory and Distributed Cognition:. *Computer Supported Cooperative Work*, 243-267.

Hutchins, E. (1995). How a Cockpit Remembers Its Speeds. *Cognitive Science*, 265-288.

IDC Corporate USA. (30 de Agosto de 2015). *IDC*. Obtenido de IDC: http://www.idc.com/getdoc.jsp?containerId=prUS25037214

JNBridge LLC. (30 de Agosto de 2015). *jnbridge*. Obtenido de jnbridge: http://jnbridge.com/

L.I.D.O. (2005). *AgentService*. Obtenido de AgentService: http://www.agentservice.it/agentservice.aspx

Las reglas de la diplomacia. (1987). JOC Internacional.

MonoProject. (30 de Agosto de 2015). *Mono*. Obtenido de Mono: http://www.mono-project.com/

Naveh, B. (30 de Agosto de 2015). *JGraphT*. Obtenido de http://www.jgrapht.org/

Poslad, S., Buckle, P., & Hadingham, R. (2000). The FIPA-OS agent platform: Open Source for Open. *Proceedings of the 5th international conference and exhibition on the practical application of intelligent agents and multi-agents*, (pág. 368). London.

SdNcenter Company. (30 de Agosto de 2015). *javoNEt*. Obtenido de javoNet: https://www.javonet.com/#tutorials

Vecchiola, C., Grosso, A., & Boccalatte, A. (2008). AgentService: a framework to develop distributed multiagent systems. *International Journal of Agent-Oriented Software Engineering*, 290-323.