UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

# DESARROLLO DE UN MOTOR DE JUEGOS DE AVENTURAS

Alejandro Robles Domínguez

Tutor: Daniel Borrajo Millán

Leganés. Septiembre de 2015.

Título: Desarrollo de un motor de juegos de aventuras

Autor: Alejandro Robles Domínguez

Tutor: Daniel Borrajo Millán

EL TRIBUNAL

Presidente:

Vocal:

Secretario:

Realizado el acto de defensa y lectura del Trabajo Fin de Grado el día __ de _____ de 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

# AGRADECIMIENTOS

La realización del trabajo de fin de grado supone el fin de una etapa y el comienzo de otra.

Cuando una etapa acaba es inevitable volver la vista atrás y darse cuenta de todas las experiencias y momentos vividos. Son muchas las personas que han hecho de esta etapa algo especial, algo que nunca podré olvidar y a los cuales agradezco todas las ayudas y aportaciones realizadas.

En especial, debo agradecer a mi familia el apoyo que me han proporcionado durante todos estos años de estudio. Sin ellos no hubiera sido posible llegar hasta aquí.

Gracias también a todos mis compañeros por estos años de carrera ya que juntos hemos aprendido cientos de cosas interesantes y nos hemos enfrentado a largas prácticas que parecían interminables pero que juntos disfrutamos.

Gracias a toda la gente que he conocido durante estos 5 años, ya que en muchas ocasiones me han apoyado y me han dado algún empujón en los momentos que más lo necesitaba.

Gracias a mi gran grupo de amigos y compañeros de la universidad, de no ser por ellos, el grado se me habría puesto mucho más cuesta arriba.

Gracias a mi tutor de trabajo de fin de grado Daniel Borrajo Millán que me ha ayudado en todo lo que he necesitado para que este proyecto saliera adelante. También agradecer a mi tutor por proponer este proyecto tan interesante y por darme la libertad y flexibilidad desde el primer día para cómo realizar el mismo.

Por último y no menos importante, quiero agradecer a todos mis compañeros de la empresa ***Métrica Consulting***. Es en esta empresa donde realmente comencé a formarme profesionalmente y a trabajar como programador en Java. Sin ellos y sin la experiencia adquirida, el desarrollo de este trabajo de fin de grado hubiera resultado mucho más duro.

# Resumen

Si nos remontamos al inicio de los videojuegos, se puede comprobar que en ese momento surge la necesidad de crear algoritmos capaces de tener la suficiente capacidad como para resolver problemas complejos por sí solos en diversas situaciones.

Por otro lado, si trazamos una línea temporal desde hace diez años hasta la actualidad, se podría comprobar cómo la Inteligencia Artificial (IA) se va abriendo paso hasta pasar a formar parte de nuestro día a día.

Uno de los **horizontes** en los que más ha avanzado esta disciplina es el área de los videojuegos.

En la actualidad, todos los días, millones de jugadores de todo el mundo dedican **una cantidad sin fin** de horas a diversas plataformas interactivas para su propio entretenimiento. No obstante, muchos desconocen que la *inteligencia* propia que adquiere un videojuego tiene su raíz en la IA.

Así, de esta manera, existen al igual un gran número de profesionales dedicados a la investigación de nuevos métodos y algoritmos que hagan que esta disciplina evolucione a pasos agigantados.

El objetivo principal de este trabajo de fin de grado es la **recreación** de un motor capaz de *decidir* en cada momento y procesar diversas acciones de un juego de aventuras. Se trata de un motor autosuficiente que acepta a varios jugadores en un mismo escenario.

Actualmente los videojuegos tienen una gran complejidad y se apoyan además en motores gráficos muy potentes de los que se obtiene una gran interfaz gráfica. Por este motivo, en este proyecto no se tendrán en cuenta aspectos relacionados a la parte gráfica y se profundizará en las distintas comprobaciones y el procesamiento interno de las peticiones que realizan los distintos jugadores.

A pesar de que el proyecto no se centra en la parte gráfica, se ha creído conveniente la creación de una interfaz en la que se vayan mostrando los distintos cambios que van ocurriendo en el motor.

De esta forma se logrará no sólo un motor que compruebe peticiones de clientes, sino llevar a cabo un seguimiento visual de las mismas.

# Abstract

From the moment the first video game appeared, the need of creating algorithms able to solution complex problems in different situations became something really necessary.

If we move back in time, we can notice how Artificial Intelligence (AI) became indispensable in the actual lifestyle. One of the more important field in which this technology progressed is the video games.

Everyday millions of players around the world spend most of their time in different interactive platforms just as a way of entertainment for themselves. However just a little of them know about the real origin of those sites, the AI.

To develop these technologies, is necessary a bit group of professionals working in the discovery and innovation of methods and algorithms in order to make the field of technology grow.

The point of this EOG work is recreating a motor able to “decide” in every moment and to process different actions in an adventure game. We are talking about a self-sufficient motor which can accept several players at the same scenario.

Nowadays, video games are complex, as they use potent graphic motors. In this way, it gets a powerful graphic interface. That is why in this project we will not take into account aspects related to the graphic field. We will deepen in the different checks and the internal prosecutions from the requests made be the players.

Despite the fact that the project is not about the graphic field, it was considered convenient the creation of an interface which shows the changes happening in the engine.

After having created that interface, we can make the motor absolutely independent from the scenario. As a result, it will only be attached to the game. Users probably get a bigger level of entertainment, increasing the gameplay notably.

# Índice de contenidos

# Índice de ilustraciones

# Índice de tablas

# Glosario de términos

- **Plugin**: Es un módulo de software que añade una característica o un servicio específico a un sistema más grande.
- **Eclipse**: Programa informático compuesto por un conjunto de herramientas de desarrollo que componen un ambiente de desarrollo integrado (IDE)
- **IOS**: Sistema operativo que utilizan los terminales de Apple.
- **Android**: Es el sistema operativo para dispositivos móviles de Google.
- **Petición**: Término utilizado en el motor para referirse a la acción a realizar que pide un determinado cliente.
- **Inteligencia Artificial (IA):** área multidisciplinaria, que a través de ciencias como las ciencias de la computación, la matemática, la lógica y la filosofía, estudia la creación y diseño de sistemas capaces de resolver problemas cotidianos por sí mismas utilizando como paradigma la inteligencia humana.
- **Base de datos no relacional (NoSQL):** se trata de bases de datos que no utilizan el lenguaje tradicional universal para bases de datos, SQL.
- **Biblioteca gráfica**: conjunto de herramientas que se pueden integrar a un entorno de desarrollo con el fin de crear interfaces gráficas.

# 1 Introducción

El presente documento corresponde a la memoria del trabajo de fin de grado “Desarrollo de un motor para juegos de aventuras”, cuyo principal fin es lograr la creación de un mecanismo capaz de recibir y responder a acciones de distintos jugadores de manera autónoma, de tal manera que éstos logren una experiencia satisfactoria.

Es importante comenzar el planteamiento del trabajo fijando el problema concreto objeto del estudio, así como las causas que han motivado el deseo de solucionar dicho problema.

Por este motivo, será necesario fijar los objetivos que se pretenden cumplir. Finalmente, a modo de ayuda al lector, destacar que existe un glosario de términos con palabras y expresiones que puedan causar cierta incertidumbre.

Este primer capítulo de introducción, está dividido en cinco secciones. En la sección 1.1 se presenta el problema que surge al inicio del trabajo, así como las motivaciones que han llevado a su propia resolución. En la sección 1.2 se enuncian los objetivos que se pretenden alcanzar en el proyecto. A continuación, en la sección 1.3 se explicarán los antecedentes, mientras que en el punto 1.4 se expondrá la metodología de trabajo utilizada.

Para finalizar, en la sección 1.5 se podrá contemplar la estructura que poseerá el presente documento.

## 1.1 Definición del problema

El nuevo modelo de enseñanza universitaria que surgió con la implantación del plan Bolonia ha traído consigo el nacimiento de nuevas necesidades en ámbitos muy diversos.

De este modo, surge la necesidad de aplicar los conocimientos tradicionales de la informática a nuevos mecanismos y funcionalidades que el mercado demanda. Es en este punto en el que nace la industria de los videojuegos y más específicamente, la computación asociada a los mismos.

A medida que pasa el tiempo, cada vez son más los jugadores que exigen mayores especificaciones a los juegos, ya no se conforman con lo típico de un videojuego, sino que piden que se vaya un punto por delante. De igual manera, para atender a todas estas nuevas “*exigencias*” de los usuarios, es necesaria una evolución constante de los algoritmos y de la inteligencia artificial presente en cada uno de los videojuegos.

Debido a la imposibilidad de crear un juego en su totalidad con todas sus funcionalidades y con su aspecto visual, se ha decidido acotar el ámbito de desarrollo únicamente al motor interno de procesamiento de los datos.

Esta solución requiere de mucho menos coste computacional y mucho menos tiempo de desarrollo, ya que gran parte del tiempo de desarrollo de los videojuegos actuales la ocupa el tema gráfico y visual. Al ser una solución que requiere mucho menos tiempo, se ha incorporado una pequeña interfaz de ayuda para comprobar en cada momento el estado del motor.

Como se ha comentado un poco más arriba, este proyecto trata sobre la recreación del comportamiento interno de un motor que puede ser usado para juegos de aventuras, más concretamente, en este caso para probar su funcionamiento, nos centraremos en el juego de "El Señor de los Anillos".

No obstante se ha querido ir un paso más allá y poner en funcionamiento algunos conocimientos adquiridos fuera de la universidad. En la actualidad, todo videojuego cuenta con una base de datos alojada en un servidor que guarda todo lo relativo a la propia aventura.

De esta forma, al final, si juntamos todos los componentes descritos, se obtiene un motor bastante completo, autónomo, portable y económico; con la capacidad de poder gestionar peticiones secuenciales de varios jugadores.

## 1.2 Objetivos

El objetivo principal de este trabajo es que los jugadores dispongan de un motor para multitud de escenarios de un determinado juego de aventuras (El Señor de los Anillos). Además se quiere que simplemente mediante el empleo de una ventana puedan en todo momento ver el estado del mundo, así como las acciones realizadas por otros jugadores. Al no contar con interfaz gráfica potente, el juego puede ser montado en cualquier PC, sin necesitar grandes requerimientos.

Para lograrlo, se realizará desde cero un programa en Java (que será nuestro futuro motor), teniendo en cuenta la existencia de otro trabajo de fin de grado complementario al presente, que se ha encargado de realizar la parte relacionada con la generación del escenario del Señor de los Anillos.

Con la finalidad de delimitar algo más este estudio, es necesario desglosar el objetivo principal en varios objetivos más concretos:

- **Estudio previo de los datos iniciales**. En un inicio, para simplificar el desarrollo posterior, es necesario realizar un análisis de lo que se quiere realizar.
- **Conversión de datos iniciales**. Es importante decidir cómo va a ser nuestro motor, en qué lenguaje se desarrollará, así como los datos de entrada con los que contamos. El uso de unas tecnologías y/o lenguajes **no** condiciona el resultado final ni el funcionamiento del motor.
- **Comprobaciones pertinentes** para cubrir todos los casos de uso posibles del dominio de definición de la aventura.
- Se deben crear **cuatro clientes simulando jugadores** comunes para realizar pruebas de peticiones y para comprobar el motor en su totalidad.
- **Creación de una interfaz de ayuda** para los jugadores en la que se añaden diversas funcionalidades tales como: mostrar el estado del mundo, visualizar las peticiones realizadas, etc.
- **Despliegue y puesta a punto de una base de datos** para persistir los datos una vez finalizada la conexión entre el motor y los jugadores.

## 1.3 Antecedentes

Actualmente, existe en la universidad Carlos III un conjunto de proyectos de fin de grado dedicados a la generación automática de diversos tipos de narrativas.

Para este proyecto se podría haber utilizado cualquiera de las narrativas generadas planificación de tareas, no obstante, era necesario elegir alguna para centrarse en su estudio y se decidió optar por la narrativa fantástica basada en el libro de *"El Señor de los Anillos (ESDLA)"*.

Toda narrativa generada mediante planificación de tareas, se caracteriza por tener un dominio y varios problemas (escenarios). Mientras que el dominio impone las reglas válidas para el motor, los problemas establecen las condiciones iniciales y las metas del juego.

Como resultado final, se obtienen dos proyectos que se complementan, el primero de ellos dedicado a la generación del dominio y problema y el segundo permitiendo a varios jugadores interactuar con la narrativa creada.

## 1.4 Metodología de Trabajo

La realización de este trabajo de fin de grado estuvo expuesta a multitud de cambios durante su desarrollo, debido a este motivo, en las siguientes líneas se expondrán las decisiones tomadas en cuanto a las metodologías de trabajo utilizadas.

Una metodología de desarrollo tradicional como puede ser el modelo en cascada no resultaba práctica, por el contrario, existen las metodologías ágiles de desarrollo [1].

Con el objetivo de responder a los cambios que fueran surgiendo se optó por utilizar este tipo de metodologías, caracterizadas por la realización de entregas parciales del producto final. Por ello, estas metodologías están especialmente indicadas para proyectos en entornos complejos, donde se necesita obtener resultados tempranos, donde los requisitos son cambiantes, escasos o poco definidos, y donde la innovación, la flexibilidad y la productividad son conceptos fundamentales.

Debido a la naturaleza y a la complejidad de un trabajo de fin de grado, las primeras acciones tuvieron como objetivo el estudio de la viabilidad del sistema, la definición del alcance del proyecto, así como las posibles entregas parciales.

Durante la implementación del motor, fue necesario tener en cuenta los cambios que iban surgiendo durante el desarrollo de este proyecto. Es posible que en algunas ocasiones la parte desarrollada no se adapte a lo requerido en el análisis y sea necesario repetirlo o adaptarlo a una nueva funcionalidad requerida a partir de un nuevo requisito.

Para solucionar este problema, el uso de sistemas de control de versiones es de gran utilidad. En este trabajo se ha utilizado BitBucket [2] como herramienta de integración y de control de versiones. BitBucket es un sistema de control de versiones distribuido cuyo objetivo es el de permitir mantener compartido una gran cantidad de código a un equipo de desarrollo de manera en que todos los cambios se queden reflejados en distintas versiones.

Hay dos características de BitBucket que creo que ayudan a entender su gran utilidad. BitBucket guarda el estado de cada archivo en un momento concreto. De esta manera, si uno de los archivos no ha cambiado no crea una nueva copia del mismo, simplemente crea una referencia al archivo original.

Esto además implica que muchas de las operaciones realizadas sobre el código fuente no tienen lugar en la red, permitiendo que la velocidad de proceso dependa únicamente de los recursos locales.

Gracias a estas metodologías ágiles y al repositorio de cambios del proyecto, fue posible el desarrollo de este proyecto de fin de grado.

## 1.5 Estructura de la memoria

El documento está estructurado en los siguientes apartados:

- **Introducción**: Este capítulo ofrece una visión global del trabajo realizado en la totalidad del proyecto. Principalmente, se explica el problema que se quiere resolver con sus objetivos asociados a dicha resolución, así como las causas que han motivado el deseo de darle la solución elegida.
- **Estado del Arte**: En este apartado se van a analizar las diferentes partes que se encuentren vinculadas con el proyecto; así como las tendencias más actuales referentes al entorno de los videojuegos. Finalmente se lleva a cabo un análisis de porqué se ha decidido desarrollar el motor mediante el lenguaje de programación JAVA.
- **Análisis de la solución**: Esta sección contempla un análisis de todas las posibles alternativas a implementar ideadas, seleccionando la más adecuada. A continuación, se obtendrá una especificación de requisitos, posibles restricciones del motor, así como distintos casos de uso del motor.
- **Diseño**: De todas las soluciones posibles establecidas en el análisis, una vez decidida la solución final, se especificará el diseño que deben seguir todos los componentes a implementar en el proyecto.
- **Pruebas del sistema**: Este apartado comprende las pruebas que debe pasar el sistema para verificar la solución elegida.
- **Gestión del proyecto**: Se expondrá la planificación y el presupuesto del proyecto.
- **Conclusión**: Para concluir este documento, se presentarán las conclusiones obtenidas y las líneas futuras de este proyecto.

# 2 Estado del Arte

Una vez se han planteado -a grandes rasgos- los objetivos del trabajo, y además se han aclarado ciertos extremos que se consideran interesantes para la correcta comprensión del mismo, es posible entrar a dar cuerpo al proyecto.

A pesar de tener localizados los grandes objetivos globales, se ha visto oportuno llevar a cabo un pequeño estudio sobre los componentes vinculados al motor. Además en dicho estudio se realizará una visión global sobre las tendencias actuales en entornos de desarrollo de videojuegos.

Con esta labor de estudio lo que se quiere lograr es el enfoque del trabajo a partir de los sistemas actuales, intentando lograr así unos mejores resultados.

Para finalizar, se han indicado los motivos por los que se ha seleccionado realizar este motor en el lenguaje de programación de JAVA.

## 2.1 Tendencias actuales del desarrollo de videojuegos.

A lo largo de los años la finalidad y el uso de los videojuegos ha ido cambiando, si bien en los inicios de la computación o en los primeros motores para videojuegos, los fines eran el aprendizaje y la creación de algo distinto, una realidad virtual en la que una persona pudiera entretenerse y pasar un buen rato. Actualmente muchos de esos principios se han perdido, ya que existen multitud de empresas asociadas al mundo del videojuego cuyo único fin es el beneficio lucrativo.

A pesar de todo esto, en la actualidad también existen desarrolladores dedicados a la investigación gracias a los cuales el mundo de la Inteligencia Artificial sigue avanzando a pasos agigantados. Es por esto que se ha tomado la decisión de implementar un motor para un juego de aventuras desde cero para recrear los inicios de los motores en el mundo del videojuego.

Las tendencias actuales del desarrollo de videojuegos son las siguientes [3]:

1. **Arreglar el problema de la última generación**: Los videojuegos evolucionan a lo largo del tiempo, por lo tanto, en cada generación aparece la necesidad de solventar los problemas surgidos en la anterior. Actualmente los problemas heredados de la generación anterior

(hasta el año 2013) serían la generación de sombras y bordes, así como el aumento de tramas por segundo en cualquier secuencia del juego.

2. **Herramientas de desarrollo fácilmente escalables:** Con esto queremos indicar la facilidad para usar las herramientas para desarrollos actuales en líneas futuras del videojuego. La herramienta ideal sería aquella en la que un juego se puede escalar o portar a distintas plataformas.
3. **Telemetría y análisis social:** Un gran nicho de mercado lo ocupan los videojuegos con fines sociales. Éstos contienen datos de carácter estadístico, analizados por expertos para saber lo que los jugadores quieren o lo que no.
4. **Generación aleatoria del terreno:** Predomina en esta generación la tendencia del uso de algoritmos para la creación aleatoria de terrenos.
5. **Sculpture Modeling:** Nuevas herramientas se abren terreno para crear texturas en 3D.
6. **Tubería de desarrollo web:** Se trata de una técnica novedosa la cual nos permite estandarizar ciertos aspectos de los juegos además de un rápido despliegue del mismo.
7. **Back ends de seguridad y web:** De forma paralela a los videojuegos surge la necesidad de dotar de seguridad a los mismos frente a posibles modificaciones o ataques por parte de hackers que pongan en peligro la experiencia de cualquier otro jugador o la integridad del propio motor del videojuego.
8. **Desarrollo independiente:** Cada vez son más las herramientas disponibles y puestas al alcance de todos los desarrolladores. Por este motivo, cada día es mucho más sencillo poder desarrollar de manera autónoma un videojuego o un motor (como en este proyecto).
9. **Tendencias de juegos sociales y móviles:** Desde hace unos años hasta ahora, la tendencia hacia los juegos con temática social y en plataformas móviles (como IOS o Android) va en aumento. Tanto los desarrolladores como las grandes empresas apuestan por un videojuego en el que el jugador pueda interactuar con otros usuarios además de poder jugarlo en cualquier momento del día en su dispositivo móvil.

Las tendencias en los sistemas virtuales, así como en los motores para juegos están en constante evolución. Por lo tanto, a la hora de desarrollar un sistema de este tipo, se debe tener en cuenta que se va a tratar de un sistema que va a necesitar actualizarse a lo largo del tiempo para seguir siendo competitivo con el resto de sistemas del mercado.

En este proyecto, **se va a prestar atención** en que el motor desarrollado pueda ser utilizado para posibles futuras líneas del juego o evolutivos, estando abierto así a posibles incorporaciones de nuevas mejoras o, la creación del entorno de manera visual utilizando herramientas de modelado 3D para la generación de los terrenos y personajes.

## 2.2 Planificación automática de tareas.

En este punto, enfocaremos toda nuestra atención hacia la Inteligencia Artificial (IA), una rama encargada de la resolución de los problemas cotidianos que nos surgen a los seres humanos. Para la resolución de los mismos, existen algoritmos realizados a partir del conocimiento humano que nos ayudan a que, tareas diarias o cotidianas nos sean mucho más amenas o fáciles de llevar a cabo.

Si bien es cierto que existe mucha controversia en este tema, ya que hay quien opina que la IA y el avance de la misma es perjudicial para el ser humano, siempre hay que tener en cuenta que todo avance en la sociedad, y más en esta área, puede ser usado también con buenos fines.

La planificación de tareas es una disciplina perteneciente a la Inteligencia Artificial cuyo fin es llegar a una meta o estado final a través de unas acciones permitidas en dicho escenario. Para llevar a cabo estas acciones, es necesario que se nos provea de un estado inicial. El conjunto de acciones obtenido es el plan. En la planificación de tareas participan tres componentes que interactúan entre ellos:

- **Dominio:** Está formado por un conjunto de predicados que pueden formar parte el estado en cualquier momento. En el dominio se determina el conjunto de predicados válidos para un determinado escenario.
- **Problema:** Está formado por metas que se quieren lograr para ese determinado escenario. También contiene un estado inicial en el que se definen todas las características de los objetos del escenario.
- **Planificador:** Se trata de un software encargado de relacionar el dominio y el problema para generar un conjunto de acciones o plan.

En este proyecto no se usará como tal la planificación de tareas para generar un dominio y un problema, pero el proyecto tiene como punto de partida un dominio y un problema proporcionado por otro Trabajo de Fin de Grado de otro alumno, por lo que es necesario conocer esta disciplina para ubicarse en un punto de vista global desde el que poder comenzar a desarrollar.

## 2.3 Planning Domain Definition Language (PDDL)

Planning Domain Definition Language (PDDL) [5] surge como un intento de estandarización de todos los lenguajes de planificación existentes en la Inteligencia Artificial (IA). Está inspirado en STRIPS y en ADL:

1. **STRIPS [6]** - Stanford Research Institute Problem Solver (STRIPS), en sus inicios estaba referido a un planificador desarrollado por Richard Fikes, aunque posteriormente se usó dicho nombre para referirse al lenguaje formal usado para las entradas de un planificador. Este lenguaje es la base de la mayoría de los lenguajes actuales basados en planificación automática de instancias de un problema.

   Una instancia de STRIPS está compuesta por los siguientes elementos:

   1. Un estado inicial
   2. Metas a alcanzar por el planificador
   3. Un conjunto de acciones, cada una de ellas compuesta por unas precondiciones y unas Postcondiciones.

2. **ADL [7] -** Action Description Language (ADL), es un sistema de planificación y programación automática. Está considerado como un avance de STRIPS. El esquema de ADL comprende el nombre de la acción, así como una lista de parámetros opcionales, dentro de la cual se puede encontrar unas cláusulas también opcionales (Preconditions, Add, Delete, Update). Un ejemplo de dicho esquema sería el siguiente:

```
Action (
  Load (c: Freight, p: Airplane, A: Airport)
  Precondition: At(c, A) ^ At(p, A)
  Effect: ¬At(c, A) ^ In(c, p)
)
```

Ilustración 1 : Esquema ADL.

Desde su creación, PDDL ha ido sufriendo actualizaciones y modificaciones que han hecho que se establezcan distintas versiones del lenguaje siendo las más importantes las siguientes y ordenadas cronológicamente:

- **PDDL1.2 -** La primera de las versiones fue la encargada de separar en dos partes independientes la descripción del dominio y la descripción del problema. De esta manera

se establecen las entradas del planificador aunque las salidas del mismo no quedan especificadas.

- **PDDL2.1 -** En esta nueva revisión se incluye la posibilidad de usar funciones numéricas y métricas en los planes, de forma que éstas sirvan para evaluar la calidad de los planes. De manera adicional, esta revisión incluye acciones durativas.
- **PDDL2.2 -** En este punto se incluyen predicados derivados para representar dependencias existentes entre dos hechos. Por otro lado, también se incluye la posibilidad de utilizar literales iniciales temporales para permitir ejecutar un evento en un instante de tiempo concreto, independientemente del flujo de ejecución del plan global.
- **PDDL3.0 -** se introdujeron restricciones de transición de estados, y preferencias que indican qué objetivos tienen mayor importancia, para maximizar los objetivos principales alcanzados en cualquier tipo de plan y así obtener unos mejores resultados.
- **PDDL3.1 -** Se trata de la última versión y, por tanto, la versión actual de este lenguaje. Las últimas modificaciones del mismo incluyen la adición de funciones-objeto, de manera que el valor que pueden tomar las funciones del plan, puede ser de cualquier tipo/objeto predefinido.

Como se ha comentado en apartados anteriores, la base para comenzar el proyecto reside en un dominio y un problema en lenguaje PDDL, creado a su vez por otro alumno. Por este motivo, la versión usada para comenzar es la inicial, la versión 1.2.

Para llevar a cabo una interpretación de instrucciones descritas en PDDL, es necesario comprender la estructura y los datos contenidos tanto en el dominio como en el problema inicial proporcionado.

La estructura del dominio es la siguiente:

1. Definición del nombre del dominio junto con las características que se van a utilizar.

```
(define (domain narrative)
    (:requirements :strips :typing :equality)
```

Ilustración 2: dominio PDDL.

2. En las siguientes líneas se establecen los objetos que participarán en las acciones posteriores.

```
(:types
    edible dangerous - animal
    character animal - entity
    place
    goodmotive evilmotive neutralmotive - motive
    arm usefulobj - storable
    entity storable - object
    status
)
```

Ilustración 3 : tipos PDDL.

3. Más tarde se añaden los predicados que se utilizarán. Las argumentos que contienen un símbolo *"?"*, seguidamente van acompañados a su derecha del nombre del tipo de variable que son. En este caso en particular, existen multitud de predicados en el dominio por lo que, a modo de ejemplo, se muestran únicamente algunos:

```
(:predicates
        ; PLACES AND TRAVEL
    (at ?obj - object ?place - place)
    (can-be-destroyed-in ?obj - object ?place - place)

        ; CHARACTER STATUS
    (hurt ?char - character)
    (dead ?ent - entity)
    (invisible ?char - character)
    (weak ?char - character)
    (strong ?obj - object)
```

Ilustración 4 : predicados en PDDL.

4. En último lugar, se necesita establecer una lista de acciones posibles a ejecutar en este dominio. Un ejemplo de acción sería:

```
(:action take-obj ; Coger un objeto
        :parameters (?char - character ?obj - storable ?place - place)
        :precondition (and
            (not (dead ?char))
            (at ?char ?place)
            (at ?obj ?place)
        )
        :effect     (and
            (not (at ?obj ?place))
            (have-an-obj ?char ?obj)
        )
)
```

Ilustración 5 : acciones en PDDL.

Cada acción a su vez contiene parámetros que se usarán ("*:parameters*"), precondiciones necesarias que se deben dar para que la acción se pueda llevar a cabo ("*:preconditions*") y efectos de la acción una vez ejecutada ("*:effects*").

## 2.4 Extensible Markup Language (XML)

Otro de los componentes presente en el proyecto es el lenguaje XML [8]. Se trata de un lenguaje de marcas desarrollado por la W3C usado para almacenar datos de una manera legible. Una de las ventajas de XML es que éste permite definir la gramática de un lenguaje en específico para estructurar grandes documentos. XML además, se propone como un estándar para el intercambio de información entre varias plataformas o componentes de un sistema.

En la actualidad permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

En resumen, XML es una tecnología sencilla compatible con otras muchas que pueden llegar a incrementar el potencial y las posibilidades del mismo.

La estructura de un XML puede contener los siguientes elementos:

1. **Prólogo -** Puede o no aparecer y contiene información relativa al tipo de elemento, versión del mismo, codificación del texto usada, etc.
2. **Cuerpo -** Es imprescindible en un XML y únicamente contiene un elemento raíz del que parten todos los demás.
3. **Elementos -** Un documento XML puede contener varios elementos. Pueden aparecen o bien con información o vacíos.

4. **Atributos -** Un elemento a su vez puede tener atributos que indican propiedades del mismo y es necesario que el atributo vaya entrecomillado (“”) dentro del elemento.
5. **Entidades predefinidas -** Encargadas de representar caracteres especiales (*%,&*)
6. **Secciones CDATA -** Se trata de un tipo de construcción para especificar datos sin que el procesador de XML lo interprete como marcado XML.

```
<![CDATA[special code: \n DatosATratar]]>
```

7. **Comentarios -** Texto introducido por el programador a modo informativo que será ignorado por el procesador de XML. Un ejemplo de comentario sería el siguiente:

```
<!-- Esto es un comentario -->
```

Un posible documento XML podría ser el siguiente:

```
<?xml version="1.0" type="XML" encoding="UTF-8" ?>
    <cuerpo>
        <!-- Nombre Propio -->
        <nombre type="persona">Alejandro</nombre>
        <!-- Primer Apellido -->
        <apellido1>Robles</apellido1>
        <!-- Segundo Apellido -->
        <apellido2>Domínguez</apellido2>
    </cuerpo>
```

Ilustración 6 : ejemplo de XML.

## 2.5 Plataforma de desarrollo Java

La plataforma de desarrollo elegida para el desarrollo del motor ha sido Java [9] mediante el IDE Eclipse [10]. Java es un lenguaje de programación orientado a objetos [11] diseñado para permitir a los desarrolladores la escalabilidad del código generado. Esto quiere decir que, dicho código se puede ejecutar en cualquier tipo de sistema o entorno sin que éste pueda estar ligado a compatibilidades propias de un tipo de entorno.

A partir de 2012, Java cobra fuerza y es uno de los lenguajes de programación más usados, debido en gran parte a la facilidad de su aprendizaje a nivel básico. Java deriva en gran

parte de otros dos lenguajes más antiguos como son *C* [12] y *C++* [13], no obstante el lenguaje Java trabaja a mucho más alto nivel que los dos citados anteriormente.

Para la ejecución de un determinado programa en Java, primero es necesaria la compilación mediante la Máquina Virtual de Java (JVM) para lo cual no importa la arquitectura de la máquina en la que se realice la compilación.

A lo largo del tiempo este lenguaje ha ido sufriendo modificaciones y actualizaciones como otros muchos. La primera de las versiones oficiales correspondió a la versión *JDK 1.0* [14] concebida originalmente como un lenguaje de programación para ejecutar aplicaciones específicas en las cajas de televisión digital.

Las revisiones posteriores y nuevas versiones componen una lista de gran tamaño hasta llegar a la versión actual del lenguaje, la versión *Java SE 8*, lanzada en marzo de 2014.

Además de todas las actualizaciones propias de Java, a lo largo del tiempo se han ido creando aplicaciones basadas en Java que incorporan nuevas funcionalidades o mejoran una ya existente.

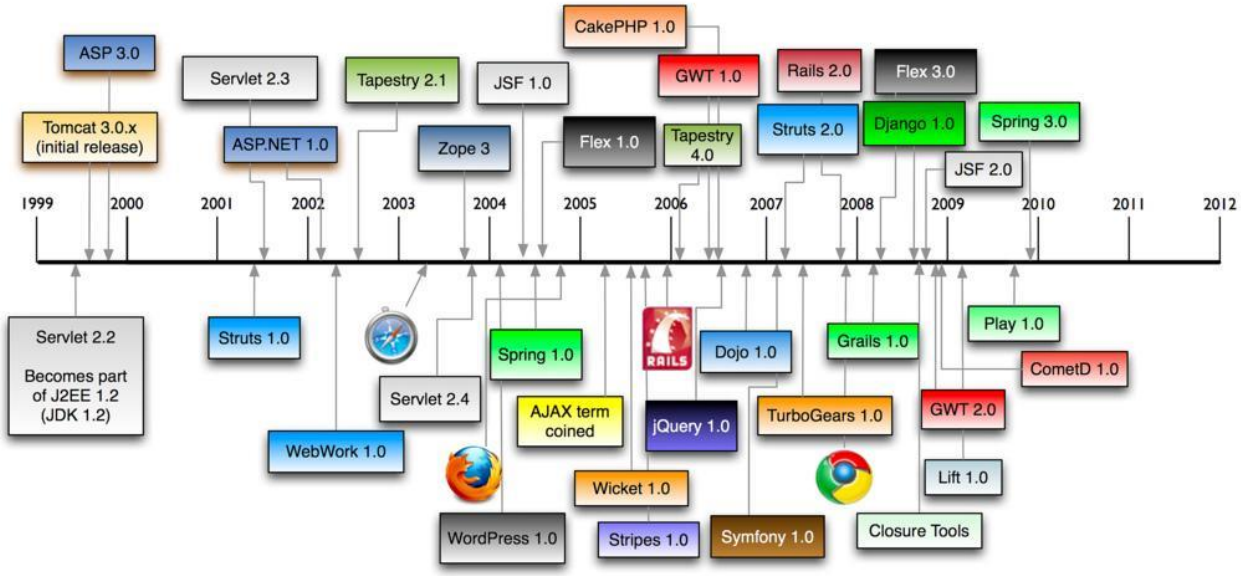


**Ilustración 7 : Frameworks de Java a lo largo del tiempo.**

Otro de los motivos por los que se decidió que la plataforma de desarrollo del motor iba a ser Java es debido a la gran fragmentación que existe en cuanto al gran número de lenguajes de programación existentes en la actualidad. De entre todos los lenguajes, Java predomina en cuanto

a su desarrollo, por este motivo, de cara a un futuro próximo, sería mucho más sencillo encontrar un puesto de trabajo relacionado con Java que con cualquier otro lenguaje.

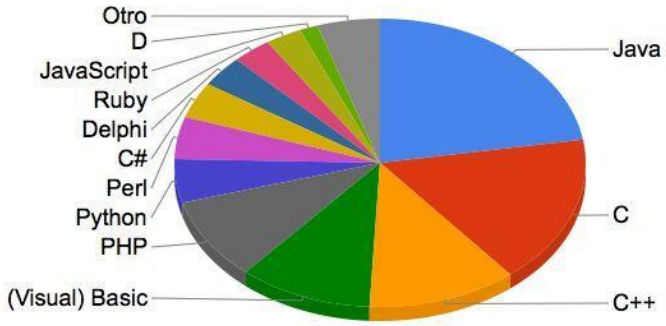


Ilustración 8: Tendencias de uso de los lenguajes de programación.

## 2.6 Swing for Java.

Como medida adicional se optó por incorporar una biblioteca gráfica a Java. Esta biblioteca se llama *Swing* [15] y se decidió añadir para crear una mínima interfaz que facilitase su uso por parte del usuario. Swing sigue un modelo de programación por hilos y posee las siguientes características:

- **Independencia de plataforma.**
- **Extensibilidad -** Los desarrolladores puede extender funcionalidad de clases propias de la biblioteca.
- **Personalizable -** Los usuarios son capaces de dotar a la interfaz de nuevas apariencias sin necesitar tocar el código de la aplicación.

Una interfaz gráfica hecha con Swing tiene el siguiente aspecto:

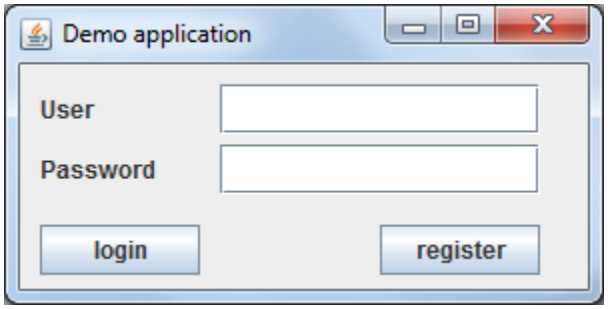


Ilustración 9 : Interfaz gráfica en Swing.

## 2.7 MongoDB.

Para finalizar con la explicación de los componentes que contendrá el motor, es necesario mencionar la base de datos usada para persistir los datos del mismo. En este caso se ha optado por utilizar una base de datos llamada MongoDB [16].

Se trata de un sistema de base de datos no relacional (NoSQL) orientado a documentos y de código abierto. El hecho que sea de código abierto le proporciona ventaja frente a otro tipo de bases de datos en las que es necesaria licencia para su uso.

La característica principal que hace a MongoDB tan atractivas es que guarda los datos en un tipo de estructura llamada JSON [17], haciendo que la integración de esta estructura de datos en las aplicaciones sea mucho más sencilla y rápida.

Otra de las características por la cual se eligió esta base de datos, además de una gran ventaja de MongoDB, es la compatibilidad completa con los distintos tipos de SO y la posibilidad de uso multiplataforma.

A continuación se muestra una ilustración que refleja las ventajas de MongoDB:

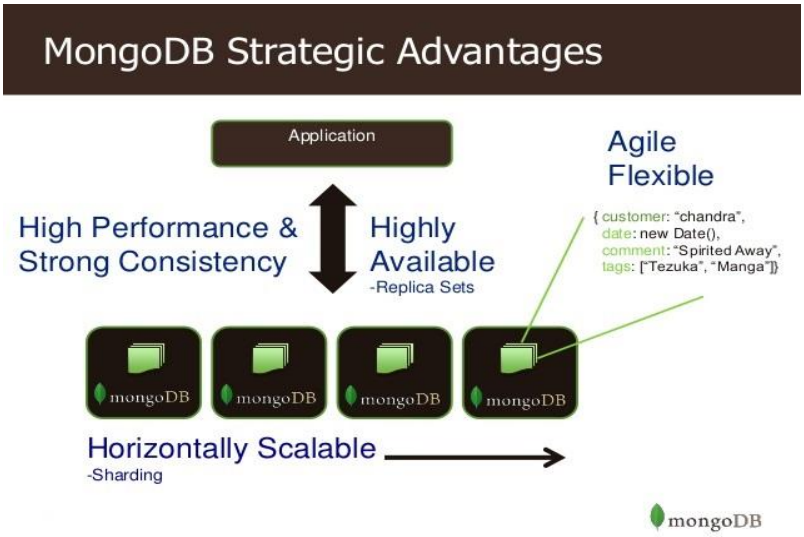


Ilustración 10: Ventajas de MongoDB.

Como se ha comentado antes, el motor se desarrollará en el lenguaje Java, no obstante, tratar con un fichero PDDL, así como con sus datos asociados al dominio y problema es mucho más difícil, debido a esto, se utilizó un *parseador* en Java cuya finalidad es traducir las instrucciones PDDL a XML. Este paso es totalmente necesario porque de no haber usado dicho traductor, el desarrollo del motor hubiese sido complejo y costoso ya que no existe ninguna funcionalidad ni ningún extra de Java que sepa interpretar instrucciones en PDDL.

# 3 Análisis de la solución.

En líneas generales, en el presente capítulo se va a desarrollar una explicación de la solución adoptada, para dar respuesta al problema planteado en los puntos anteriores. Se tomará como punto de partida la descripción de la propuesta inicial, analizando todos los posibles problemas que traería consigo el desarrollo de esta solución. Una vez analizados estos inconvenientes, se verá con detalle la solución final, indicando además todas las restricciones que presenta esta propuesta. A continuación, se va a realizar una especificación de casos de uso y requisitos.

Finalmente, se ha incluido una matriz de trazabilidad entre los requisitos funcionales y los casos de uso.

## 3.1 Descripción de la solución

### 3.1.1 Propuesta inicial

La propuesta inicial consistía en desarrollar un motor para un juego de aventuras capaz de resolver peticiones de distintos clientes en tiempo real. El motor debe realizar un cómputo de las peticiones y decidir si éstas se pueden llevar a cabo según las condiciones impuestas por el escenario.

Para poder enviar peticiones, los clientes deberían estar conectados al propio motor además de elegir la acción a realizar. El motor, a su vez, permanecerá constantemente activo escuchando posibles peticiones de los clientes. Si la petición recibida es correcta y se puede realizar, se ejecutaría la acción o acciones asociadas a dicha petición, así como los efectos que ésta implica en el escenario.

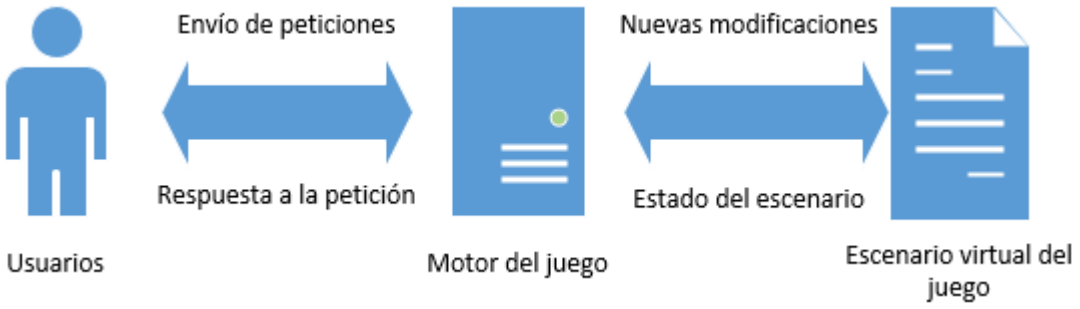


Ilustración 11: Diagrama general de la propuesta inicial.

### 3.1.2 Problemas y cambios en la propuesta inicial

Al llevar a cabo un primer análisis de la propuesta inicial y estudiar más a fondo el funcionamiento del motor a desarrollar, se hicieron evidentes algunos problemas así como ciertas mejoras necesarias para alcanzar un motor  lo más completo y robusto posible. Los nuevos cambios y problemas encontrados fueron los siguientes:

- En primer lugar, es necesario establecer el número de clientes que va a poder soportar nuestro motor. Actualmente los motores permiten conexiones simultáneas a una gran cantidad de usuarios, no obstante, al tratarse de un proyecto con alcance y tiempo limitado, se llegó a la conclusión que los clientes soportados por dicho motor serían cuatro.
- En segundo lugar, además de establecer los clientes permitidos, era necesario establecer el orden en el que éstos enviarán los datos al igual que es importante decidir qué datos se enviarán. Para solventar este contratiempo se optó porque los clientes realizasen peticiones secuenciales.
- En tercer lugar, la solución inicial trata sobre la creación de un motor a nivel funcional, es decir, el alcance de esta propuesta únicamente contempla el comportamiento interno del servidor y todo el código que conlleva dicho comportamiento. Por este motivo y como extra a la propuesta inicial, se ideó una manera con la que los usuarios pudieran realizar las peticiones de una manera más fácil e intuitiva. Los usuarios se conectarán al motor a través de una interfaz de escritorio en la que podrán realizar varias tareas, entre ellas el envío de peticiones.
- Por último, surge el problema de la persistencia de datos. Estos datos serán guardados en formato XML pero, además, se realizarán inserciones de las peticiones realizadas en una base de datos ajena al motor para proporcionar robustez al mismo. Si el día de mañana el

motor sufre cualquier tipo de fallo y se desconecta, el registro de peticiones queda intacto y fuera de cualquier tipo de peligro quedando almacenado en la base de datos.

### 3.1.3 Solución final

Teniendo en cuenta todas estas consideraciones, la solución final deberá contar con varios componentes que permitan, tanto a usuarios como al motor, interactuar entre ellos; y al mismo tiempo, integrarse perfectamente con el escenario sobre el cual se lleva a cabo dicho juego de aventuras.

Tras este análisis, se llegó a la conclusión que el producto final obtenido tendría todos los componentes y aplicaciones mencionados en epígrafes anteriores. De esta manera, el diagrama en conjunto de la aplicación sería el siguiente:

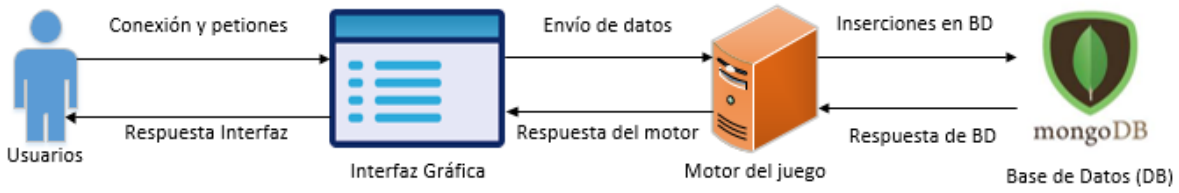


Ilustración 12: Diagrama general de la solución final.

Las nuevas funcionalidades necesarias para que el motor se adapte al nuevo objetivo de este proyecto son las siguientes:

1. Conexión de los clientes a la interfaz gráfica para poder así comenzar a usar todas las opciones disponibles en dicha interfaz. Para llevar a cabo dicha conexión se integrará la propia interfaz en el motor a modo de prueba, esto quiere decir que, cuando el motor se inicie, automáticamente se levantará la propia interfaz. De esta forma nos podemos asegurar que los clientes sólo se podrán conectar al juego si éste está funcionando, evitando fallos de seguridad o posibles errores en las peticiones. Normalmente, la interfaz debe de estar separada del motor, pero al tratarse de una aproximación a un motor real, se ha optado por esta integración.

2. El flujo de datos entre la interfaz gráfica y motor interno se realizará de manera transparente para el usuario, siendo imposible así llevar a cabo modificaciones sobre el entorno.
3. Conexión del motor a la base de datos externa para realizar las inserciones de las peticiones de los usuarios. Esta conexión se realizará a través del driver, distribuido por la propia entidad de la base de datos.

## 3.2 Restricciones del Sistema

La solución adoptada deberá adaptarse a los datos que gestiona internamente el motor, a la parte de la interfaz gráfica y a la propia base de datos:

- Datos usados por el motor:
    - Los ficheros de entrada de datos proporcionados para este proyecto están implementados en PDDL. Para este motor se ha decidido convertir esos datos a XML y posteriormente se serializan estos datos para convertirlos de nuevo en objetos en Java mediante un plugin de eclipse llamado JAXB [18].
    - El archivo de datos generado una vez se pare el motor debe seguir la estructura XML.

- Interfaz gráfica y motor interno:
    - Tanto el motor como la interfaz están desarrolladas en su totalidad en el lenguaje de programación de Java. Este dato supone una restricción de cara a futuras ampliaciones, ya que sería necesario realizarlas en Java.
    - La interfaz gráfica está diseñada e implementada con una biblioteca particular de Java, denominada Swing. Esto se hizo así para evitar posibles conflictos en el flujo de datos entre motor e interfaz.
- Base de datos MongoDB: la base de datos trae consigo una serie de restricciones asociadas a su uso:
    - En primer lugar, al tratarse de una base de datos no relacional (NoSQL), no existe ningún tipo de relación entre los datos alojados en la misma.
    - MongoDB utiliza una estructura de datos llamada JSON. Por este motivo, para el uso de la base de datos, la comunicación entre la aplicación y la base de datos se realizará utilizando esta estructura de datos.

Para finalizar es necesario aclarar que, para el correcto funcionamiento de la aplicación, es necesario que el driver de MongoDB[1] esté instalado en el equipo de despliegue. Con motivo de facilitar la instalación del driver, al final de este documento, se incluirá un anexo en el que se explicarán los pasos a seguir para la correcta instalación de MongoDB.

## 3.3 Especificación de casos de uso

En este apartado se describen los casos de uso del sistema. En primer lugar se presentarán los diagramas de casos de uso en UML. Finalmente se realizará una descripción detallada de cada caso de uso.

### 3.3.1 Diagramas de casos de uso.

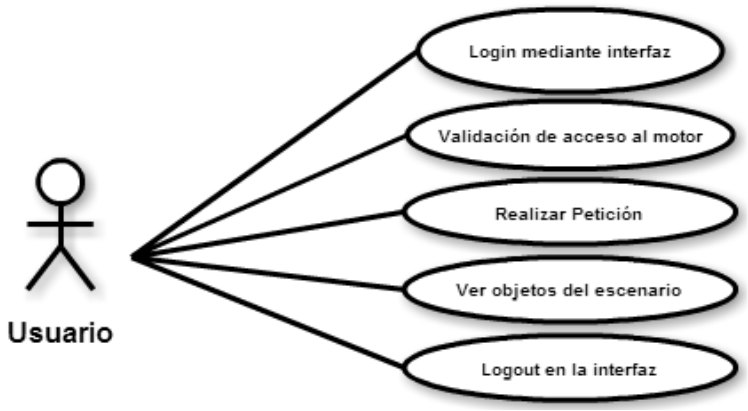


Ilustración 13: Diagrama de casos de uso del usuario.

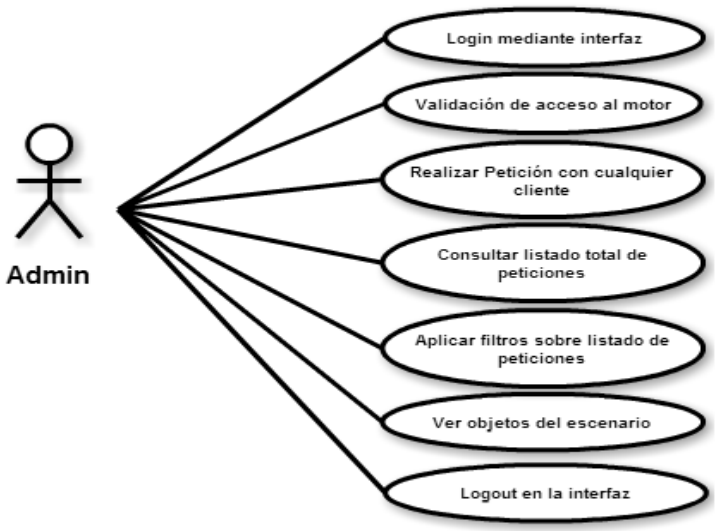


Ilustración 14: Diagrama de casos de uso de Administrador.

### 3.3.2 Descripción de los atributos de los casos de uso.

Para la descripción de los casos de uso se utilizará una tabla con los siguientes campos:

- Identificador: Identifica cada caso de uso. El formato utilizado es el siguiente "CU-XX", donde XX se corresponde con un número.
- Título: El título asignado al caso de uso.
- Actores: Los posibles roles de usuario que pueden intervenir en el caso de uso.
- Descripción: Se realiza una descripción básica de cada caso de uso.
- Precondiciones: Condiciones previas que se deben cumplir para poder ejecutar el caso de uso.
- Postcondiciones: condiciones que se deben cumplir cuando termine el caso de uso.
- Escenario: Descripción de las fases que componen el caso de uso.
- Casos de uso relacionados: Se indicarán los casos de uso que tienen relación con el caso de uso descrito.

| **Identificador** | |
|---|---|
| **Título** | |
| **Actores** | |
| **Descripción** | |
| **Precondiciones** | |
| **Postcondiciones** | |
| **Escenario** | |
| **Casos de uso relacionados** | |

**Tabla 1: Tabla de ejemplo de especificación de los casos de uso.**

### 3.3.3 Descripción textual de los casos de uso.

A continuación, se describen textualmente los casos de uso:

| **Identificador** | **CU-01** |
|---|---|
| **Título** | Realizar login en el motor mediante la interfaz gráfica. |
| **Actores** | Usuarios/Administrador. |
| **Descripción** | Cualquier usuario que desee comenzar o retomar una partida debe realizar login en el motor del jugo mediante el panel de autenticación de la interfaz gráfica. |
| **Precondiciones** | 1. El motor debe estar iniciado.<br>2. La interfaz debe de estar activa.<br>3. La ventana de autenticación debe aparecer correctamente con los campos de usuario y password. |
| **Postcondiciones** | El usuario queda logueado en la interfaz y a su vez en el motor. |
| **Escenario** | 1. El usuario accede a la interfaz del motor.<br>2. El usuario rellena su usuario y su contraseña.<br>3. El usuario realiza clic sobre el botón de *"Login"*<br>4. El usuario accede a la interfaz. |
| **Casos de uso relacionados** | - |

**Tabla 2: Caso de uso CU-01. Realizar login en el motor mediante interfaz gráfica.**

| **Identificador** | **CU-02** |
|---|---|
| **Título** | Validación de acceso al motor. |
| **Actores** | Usuarios/Administrador. |
| **Descripción** | Se realizan las autenticaciones y validaciones para cada usuario según el usuario y contraseña introducidos. |
| **Precondiciones** | El usuario/administrador debe haber realizado login correctamente.<br><br>El usuario/administrador debe elegir si desea continuar o comenzar una nueva partida. |
| **Postcondiciones** | El usuario queda validado en el motor y se carga la configuración de la partida anterior o se carga la configuración inicial de la partida. |
| **Escenario** | 1. Se realiza en caso de uso CU-01<br><br>2. Se elige si desea nueva partida o continuar la anterior.<br><br>3. Una vez elegida la opción deseada se carga la interfaz general del motor. |
| **Casos de uso relacionados** | CU-01 |

**Tabla 3 : Caso de uso CU-02. Validación de acceso al motor.**

| **Identificador** | **CU-03** |
|---|---|
| **Título** | Realizar petición de Usuario. |
| **Actores** | Usuario. |
| **Descripción** | Se realiza una petición al motor con una serie de parámetros. |
| **Precondiciones** | El usuario debe haber realizado login correctamente.<br>El usuario debe encontrarse en la pestaña de realización de peticiones. |
| **Postcondiciones** | Se envía la petición al motor del juego. |
| **Escenario** | 1. Se realiza en caso de uso CU-02<br>2. Se accede a la pestaña de realización de peticiones.<br>3. Se elige la petición a realizar.<br>4. Se realiza clic sobre el botón "*Realizar petición*". |
| **Casos de uso relacionados** | CU-01, CU-02 |

**Tabla 4: Caso de uso CU-03. Realizar petición de Usuario.**

| **Identificador** | **CU-04** |
|---|---|
| **Título** | Consultar los objetos existentes en el mundo. |
| **Actores** | Usuario/Administrador. |
| **Descripción** | Se pueden visualizar en pestañas los distintos objetos que existen en el escenario del juego de aventuras (personajes, objetos, etc.) |
| **Precondiciones** | El usuario debe haber realizado login correctamente.<br>El usuario debe encontrarse en la pestaña de visualización de objetos del escenario. |
| **Postcondiciones** | Se observan los distintos objetos existentes para el escenario actual. |
| **Escenario** | 1. Se realiza en caso de uso CU-02<br>2. Se accede a la pestaña de "Ver objetos del mundo".<br>3. Se navega entre las distintas pestañas observando los distintos objetos presentes en el escenario. |
| **Casos de uso relacionados** | CU-01, CU-02 |

**Tabla 5: Caso de uso CU-04. Consultar los objetos existentes en el mundo.**

| **Identificador** | **CU-05** |
|---|---|
| **Título** | Logout del motor mediante la interfaz. |
| **Actores** | Usuario/Administrador. |
| **Descripción** | Se realiza un logout o desconexión del usuario/administrador del motor a través de la interfaz. |
| **Precondiciones** | El usuario debe haber realizado login correctamente. |
| **Postcondiciones** | El usuario/administrador queda desconectado de la aplicación. |
| **Escenario** | 1. Se realiza en caso de uso CU-02<br>2. Se realiza logout de la aplicación haciendo clic en el botón "*Logout*". |
| **Casos de uso relacionados** | CU-01, CU-02 |

**Tabla 6: Caso de uso CU-05. Logout del motor mediante la interfaz.**

| **Identificador** | **CU-06** |
|---|---|
| **Título** | Realizar petición con cualquier cliente disponible en el motor. |
| **Actores** | Administrador. |
| **Descripción** | El administrador puede realizar peticiones suplantando a cualquiera de los usuarios registrados en la interfaz. |
| **Precondiciones** | El administrador debe haber realizado login correctamente.<br>El administrador debe encontrarse en la pestaña de realización de peticiones.<br>El administrador debe elegir qué cliente suplantar para la petición. |
| **Postcondiciones** | El administrador realiza la petición suplantando al usuario elegido. |
| **Escenario** | 1. Se realiza en caso de uso CU-02<br>2. Se accede a la pestaña de realización de peticiones.<br>3. Se elige el usuario con el que se va a llevar a cabo la petición.<br>4. Se realiza la petición haciendo clic en el botón "*Realizar Petición*". |
| **Casos de uso relacionados** | CU-01, CU-02 |

**Tabla 7: Caso de uso CU-06. Realizar petición con cualquier cliente disponible en el motor.**

| Identificador | CU-07 |
|---|---|
| **Título** | Consultar listado total de peticiones realizadas. |
| **Actores** | Administrador. |
| **Descripción** | El administrador puede consultar el listado total de las peticiones que se han llevado a cabo en el motor en la partida actual. |
| **Precondiciones** | El administrador debe haber realizado login correctamente.<br>El administrador debe encontrarse en la pestaña de consulta de peticiones. |
| **Postcondiciones** | El administrador observa el listado total de peticiones realizadas. |
| **Escenario** | 1. Se realiza en caso de uso CU-02<br>2. Se accede a la pestaña de consulta de peticiones. |
| **Casos de uso relacionados** | CU-01, CU-02 |

**Tabla 8: Caso de uso CU-07. Consultar listado total de peticiones realizadas.**

| Identificador | CU-08 |
|---|---|
| **Título** | Aplicación de filtros sobre el listado de peticiones. |
| **Actores** | Administrador. |
| **Descripción** | El administrador puede aplicar distintos filtros sobre el listado total de peticiones realizadas en el motor y partida actual. |
| **Precondiciones** | El administrador debe haber realizado login correctamente.<br>El administrador debe encontrarse en la pestaña de consulta de peticiones. |
| **Postcondiciones** | El administrador observa según el filtro establecido el listado total de peticiones realizadas. |
| **Escenario** | 1. Se realiza en caso de uso CU-02<br>2. Se realiza el caso de uso CU-07<br>3. Se realiza clic sobre el icono de los prismáticos (Filtrar).<br>4. Se elige el filtro que se quiere aplicar sobre el listado.<br>5. Se configuran las distintas opciones que da el filtro.<br>6. Se filtra la lista total de peticiones. |
| **Casos de uso relacionados** | CU-01, CU-02, CU-07 |

**Tabla 9: Caso de uso CU-08. Aplicación de filtros sobre el listado de peticiones.**

| Identificador | CU-09 |
|---|---|
| **Título** | Eliminar filtros previamente aplicados sobre el listado de peticiones. |
| **Actores** | Administrador. |
| **Descripción** | El administrador puede eliminar filtros que haya aplicado previamente sobre el listado total de peticiones realizadas en el motor y partida actual. |
| **Precondiciones** | El administrador debe haber realizado login correctamente.<br>El administrador debe encontrarse en la pestaña de consulta de peticiones. |
| **Postcondiciones** | El administrador observa sin filtros el listado total de peticiones realizadas. |
| **Escenario** | 1. Se realiza en caso de uso CU-02<br>2. Se realiza el caso de uso CU-07<br>3. Se realiza el caso de uso CU-08<br>4. Se realiza clic sobre el icono de los prismáticos tachados (Eliminar filtro).<br>5. Se observa la lista de peticiones sin ningún filtro activo. |
| **Casos de uso relacionados** | CU-01, CU-02, CU-07, CU-08 |

**Tabla 10: Caso de uso CU-09. Eliminar filtros previamente aplicados sobre el listado de peticiones.**

## 3.4 Especificación de Requisitos

En este apartado se van a describir todos los requisitos que deberá cumplir el sistema desarrollado, para, a partir de los ellos, diseñar la solución al problema planteado en el presente trabajo de fin de grado.

### 3.4.1 Formato de los requisitos

A continuación, se van a puntualizar los atributos que presentan los requisitos, de manera que cada uno de ellos será descrito en una tabla con los siguientes atributos:

- Código: Código que describe de manera única a cada uno de los requisitos. El formato utilizado será "RF-XX" para los requisitos funcionales y "RNF-XX" para los requisitos no funcionales donde XX se corresponde con un número.
- Nombre: Nombre descriptivo del requisito.
- Descripción: Descripción completa del requisito.

| | |
|---|---|
| **Código** | |
| **Nombre** | |
| **Descripción** | |

**Tabla 11: Tabla de ejemplo de especificación de requisitos.**

### 3.4.2 Requisitos Funcionales.

En este apartado se van a describir los requisitos relacionados con las funcionalidades que debe cumplir el sistema. Para una mejor comprensión de estos, se han dividido en dos categorías, requisitos del administrador y requisitos del usuario:

#### *3.4.2.1 Requisitos del Administrador.*

| | |
|---|---|
| **Código** | **RF-01** |
| **Nombre** | Login en la aplicación con usuario administrador. |
| **Descripción** | El administrador introducirá su usuario y contraseña correspondientes, obteniendo acceso total a la interfaz. |

**Tabla 12: RF-01. Login en la aplicación con usuario administrador.**

| | |
|---|---|
| **Código** | **RF-02** |
| **Nombre** | Validación de credenciales de administrador. |
| **Descripción** | El motor internamente realizará una validación dando acceso al usuario a la interfaz como administrador. |

**Tabla 13: RF-02. Validación de credenciales de administrador.**

| | |
|---|---|
| **Código** | **RF-03** |
| **Nombre** | Suplantación de usuarios para realizar peticiones. |
| **Descripción** | El administrador podrá suplantar usuarios para llevar a cabo peticiones en el motor. |

**Tabla 14: RF-03. Suplantación de usuarios para realizar peticiones.**

| | |
|---|---|
| **Código** | **RF-04** |
| **Nombre** | Acceso total por parte del administrador. |
| **Descripción** | El administrador tendrá acceso a todas las pestañas de la interfaz. |

**Tabla 15: RF-04. Acceso total por parte del administrador.**

| | |
|---|---|
| **Código** | **RF-05** |
| **Nombre** | Consulta de peticiones. |
| **Descripción** | El administrador podrá consultar el listado total de peticiones realizadas en el motor. |

**Tabla 16: RF-05. Consulta de peticiones.**

| | |
|---|---|
| **Código** | **RF-06** |
| **Nombre** | Filtrado de peticiones. |
| **Descripción** | El administrador podrá aplicar y configurar filtros sobre el listado total de peticiones realizadas en el juego. |

**Tabla 17: RF-06. Filtrado de peticiones.**

| | |
|---|---|
| **Código** | **RF-07** |
| **Nombre** | Eliminar filtros aplicados sobre peticiones. |
| **Descripción** | El administrador podrá eliminar filtros previamente aplicados sobre el listado total de peticiones realizadas. |

**Tabla 18: RF-07. Eliminar filtros aplicados sobre peticiones.**

| | |
|---|---|
| **Código** | **RF-08** |
| **Nombre** | Consultar objetos presentes en el escenario. |
| **Descripción** | El administrador podrá consultar los objetos presentes en el escenario del juego actual. |

**Tabla 19: RF-08. Consultar objetos presentes en el escenario.**

| | |
|---|---|
| **Código** | **RF-09** |
| **Nombre** | Realizar logout en la interfaz. |
| **Descripción** | El administrador podrá realizar logout de la aplicación, siendo así desconectado del motor y a su vez de la interfaz. |

**Tabla 20: RF-09. Realizar logout en la interfaz.**

#### *3.4.2.1 Requisitos del Usuario.*

A continuación, se definen los requisitos funcionales que debe cumplir la aplicación por parte del usuario:

| | |
|---|---|
| **Código** | **RF-10** |
| **Nombre** | Login en la aplicación con usuario común. |
| **Descripción** | El usuario común introducirá su usuario y contraseña correspondientes, obteniendo acceso a algunas funcionalidades de la interfaz (no obtiene acceso total). |

**Tabla 21: RF-10. Login en la aplicación con usuario común.**

| | |
|---|---|
| **Código** | **RF-11** |
| **Nombre** | Validación de credenciales de usuario común. |
| **Descripción** | El motor internamente realizará una validación dando acceso al usuario a la interfaz como usuario común sin conceder ciertos permisos. |

**Tabla 22: RF-11. Validación de credenciales de usuario común.**

| | |
|---|---|
| **Código** | **RF-12** |
| **Nombre** | No suplantación de usuarios con usuario común. |
| **Descripción** | Cualquier usuario común no podrá realizar la suplantación de otros usuarios para realizar peticiones, únicamente podrá realizar peticiones con el usuario con el que se encuentra logueado. |

**Tabla 23: RF-12. No suplantación de usuarios con usuario común.**

| | |
|---|---|
| **Código** | **RF-13** |
| **Nombre** | Acceso parcial por parte de usuarios comunes. |
| **Descripción** | Los usuarios comunes tendrán acceso a ciertas pestañas de la interfaz (Ver mundo, Realizar Peticiones). |

**Tabla 24: RF-13. Acceso parcial por parte de usuarios comunes.**

| | |
|---|---|
| **Código** | **RF-14** |
| **Nombre** | No consulta de peticiones por parte de usuario común. |
| **Descripción** | Un usuario común no podrá visualizar el listado total de aplicaciones hechas para esa partida en el motor. |

**Tabla 25: RF-14. No consulta de peticiones por parte de usuario común.**

| Código | RF-15 |
|---|---|
| Nombre | Consultar objetos presentes en el escenario. |
| Descripción | El usuario común podrá acceder a consultar los objetos presentes en el escenario del juego actual. |

Tabla 26: RF-15. Consultar objetos presentes en el escenario.

| Código | RF-16 |
|---|---|
| Nombre | Realizar logout en la interfaz. |
| Descripción | Cualquier usuario común podrá realizar logout de la aplicación, siendo así desconectado del motor y a su vez de la interfaz. |

Tabla 27: RF-16. Realizar logout en la interfaz.

### 3.4.3 Requisitos No Funcionales.

En este apartado se describen los requisitos relacionados con las restricciones a las que el sistema debe ajustarse:

| Código | RNF-01 |
|---|---|
| Nombre | Uso de función hash para validar acceso. |
| Descripción | Para la validación de los datos se utilizarán funciones hash. |

Tabla 28: RNF-01. Uso de función hash para validar acceso.

| Código | RNF-02 |
|---|---|
| Nombre | Compatibilidad del motor con cualquier Sistema Operativo (SO). |
| Descripción | El motor será compatible con cualquier SO así como todos los componentes que forman parte del mismo. |

Tabla 29: RNF-02. Compatibilidad del motor con cualquier Sistema Operativo (SO).

| Código | RNF-03 |
|---|---|
| Nombre | Desarrollo del motor interno en Java. |
| Descripción | El motor estará desarrollado con el lenguaje de programación Java, siendo este compatible con todo tipo de arquitecturas. |

Tabla 30: RNF-03. Desarrollo del motor interno en Java.

| Código | RNF-04 |
|---|---|
| Nombre | Desarrollo de la interfaz mediante librería Swing. |
| Descripción | La interfaz gráfica del motor se desarrollará utilizando la librería de Java llamada Swing. Esta permitirá aumentar la funcionalidad del proyecto así como su alcance final. |

Tabla 31: RNF-04. Desarrollo de la interfaz mediante librería Swing.

| Código | RNF-05 |
|---|---|
| Nombre | Datos de entrada al motor en formato XML. |
| Descripción | Los ficheros desde donde se cargan tanto el escenario como las posibles acciones a realizar por parte de los usuarios (peticiones), se cargan de unos ficheros en formato XML. |

Tabla 32: RNF-05. Datos de entrada al motor en formato XML.

| Código | RNF-06 |
|---|---|
| Nombre | Base de datos local para guardar peticiones. |
| Descripción | Se requiere de una base de datos alojada en el entorno local, desde el cual se despliega el motor, que sirva de almacén para todas las peticiones realizadas en la partida actual. |

Tabla 33: RNF-06. Base de datos local para guardar peticiones.

| Código | RNF-07 |
|---|---|
| Nombre | Base de datos no relacional MongoDB. |
| Descripción | La base de datos será no relacional (NoSQL), eliminando así todo tipo de restricciones y haciendo el acceso y consumo de los datos mucho más simple y eficiente. |

Tabla 34: RNF-07. Base de datos no relacional MongoDB.

| Código | RNF-08 |
|---|---|
| Nombre | Peticiones diferenciadas según cliente. |
| Descripción | En la lista de peticiones, las peticiones aparecerán diferenciadas con distinto color según el cliente (usuario) que realice la petición. |

Tabla 35: RNF-08. Peticiones diferenciadas según cliente.

| Código | RNF-09 |
|---|---|
| Nombre | Eliminar datos al finalizar partida. |
| Descripción | Se deberán eliminar los datos de peticiones de la partida actual si se termina la partida o si se decide comenzar una nueva. |

Tabla 36: RNF-09. Eliminar datos al finalizar partida.

| Código | RNF-10 |
|---|---|
| Nombre | Acceso total |
| Descripción | Solo tendrán acceso total a la aplicación aquellos usuarios que tengan rol de administrador. |

Tabla 37: RNF-10. Acceso total.

## 3.5 Trazabilidad entre requisitos funcionales y casos de uso

Con el objetivo de comprobar que se cumplen todos los requisitos funcionales, se ha realizado una matriz de trazabilidad entre los casos de uso y los requisitos funcionales.

| | CU-01 | CU-02 | CU-03 | CU-04 | CU-05 | CU-06 | CU-07 | CU-08 | CU-09 |
|---|---|---|---|---|---|---|---|---|---|
| **RF-01** | X | | | | | | | | |
| **RF -02** | | X | | | | | | | |
| **RF -03** | | X | | | | X | | | |
| **RF -04** | | X | | | | | | | |
| **RF -05** | | | | | | | X | | |
| **RF -06** | | | | | | | | X | |
| **RF -07** | | | | | | | | | X |
| **RF -08** | | | | X | | | | | |
| **RF -09** | | | | | X | | | | |
| **RF -10** | X | | | | | | | | |
| **RF -11** | | X | | | | | | | |
| **RF -12** | | X | | | | X | | | |
| **RF -13** | | | X | | | | | | |
| **RF -14** | | | | | | | X | | |
| **RF -15** | | | | X | | | | | |
| **RF-16** | | | | | X | | | | |

Tabla 38: Matriz de trazabilidad: Requisitos funcionales – Casos de uso.

Se puede comprobar que todos los requisitos están cubiertos por algún caso de uso.

# 4 Diseño del Sistema

Como se ha comentado anteriormente, el presente proyecto tiene como objetivo final crear un motor para un juego de aventuras. El juego de aventuras para el cual se ha desarrollado el motor está basado en *"El Señor de los Anillos"*. Para comenzar con su implementación, se parte en un inicio de dos ficheros en lenguaje PDDL desarrollados por otro alumno de la universidad, por este motivo, el diseño del sistema gira entorno tanto al dominio como al problema suministrado, así como a sus correspondientes predicados.

A continuación se detalla la arquitectura del sistema y se describen las acciones principales a realizar en el sistema mediante diagramas de secuencia. Finalmente se muestra el diseño final de la interfaz gráfica.

## 4.1 Descripción general de la Arquitectura.

La arquitectura general del sistema engloba un conjunto de componentes necesarios para el correcto funcionamiento del motor en su totalidad. En el siguiente diagrama se puede observar la arquitectura del sistema:

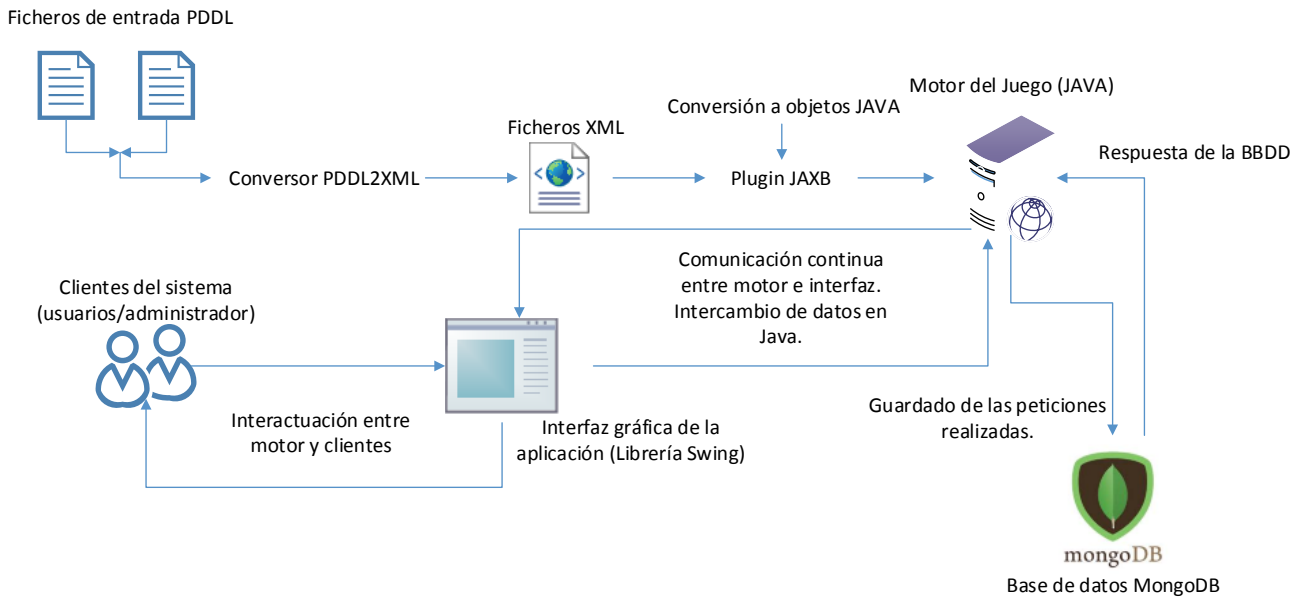


**Ilustración 15: Diagrama general de la arquitectura final.**

## 4.2 Descripción de los predicados válidos.

Como se ha comentado en el apartado anterior, es necesario ajustarse a unos ficheros de entrada en concreto facilitados por el tutor del proyecto. Por este motivo, en el fichero de entrada "*Domain.pddl*" se establecen todos los predicados válidos para nuestro escenario en concreto.

A continuación se explicará con más detalle cuáles son los predicados y las acciones válidas contenidas en el documento.

| Predicado | Parámetros recibidos | Descripción |
|---|---|---|
| **At** | ?obj–object , ?place–place | Un personaje se encuentra en un determinado lugar. |
| **Can-be-destroyed-in** | ?obj-object, ?place-place | Un objeto puede ser destruido en un determinado lugar. |
| **Hurt** | ?char-character | Indica que un personaje está herido. |
| **Dead** | ?char-character | Indica que un personaje está muerto. |
| **Invisible** | ?char-character | Indica que un personaje es invisible. |
| **Weak** | ?char-character | Indica que un personaje es débil. |
| **Strong** | ?obj-objeto | Indica que un objeto es fuerte. |
| **isNPC** | ?ent-entity | Indica que una entidad del juego es NPC (personaje no jugador). |
| **Killed-by** | ?ent1-entity, ?ent2-entity | Indica que la entidad 1 ha sido matada por la entidad 2. |
| **Is** | ?obj-object, ?st-status | Indica un determinado status o propiedad de un objeto. |
| **Friends** | ?char-character, ?char2-character | Indica que el personaje 1 es amigo del personaje 2. |
| **Enemies** | ?char-character, ?char2-character | Indica que el personaje 1 es enemigo del personaje 2. |
| **Have-an-obj** | ?char-character, ?obj-object | Indica que el personaje tiene el objeto señalado. |
| **Wish-an-obj** | ?char-character, ?obj-object | Indica que el personaje desea o quiere el objeto seleccionado. |
| **Can-use-magic-kill** | ?char-character | Indica que el personaje puede usar magia para matar. |
| **Can-use-magic-heal** | ?char-character | Indica que el personaje puede usar magia para curar. |
| **Magic-invisible** | ?obj-usefulobj | Indica que un objeto puede volverse invisible. |

| | | |
|---|---|---|
| **Magic-kill** | ?obj-usefulobj | Indica que un objeto puede usar magia que mata. |
| **Magic-heal** | ?obj-usefulobj | Indica que un objeto puede usar magia que cura. |
| **Have-motive** | ?char-character, ?m-motive | Indica que un personaje tiene un determinado motivo. |
| **Implies-that** | ?m-motive, ?m2-motive | Indica que el motivo 1 implica el motivo 2. |
| **Want-motive** | ?m-motive, ?obj-object, ?st-status | Indica que se quiere un motivo sobre un objeto con un determinado status. |
| **Motive-complete** | ?char-character, ?m-motive | Indica que el personaje ha terminado el motivo. |

**Tabla 39: Predicados incluidos en el escenario.**

## 4.3 Descripción de las acciones válidas.

El conjunto de acciones constituye el apartado más importante del motor interno. Para que una determinada historia sea interesante, es necesario que su lista de acciones válidas e incluidas sea numerosa a la vez que variada. Cada una de las acciones consta de unos parámetros de entrada y unas precondiciones que se deben cumplir para ejecutar los efectos de dicha acción.

En este caso la lista de acciones perteneciente a nuestro escenario es la siguiente:

| Acción | Parámetros de entrada | Efecto |
|---|---|---|
| **Take-obj** | ?char-character, ?obj-storable, ?place- place | El personaje coge el objeto que está en un determinado lugar. |
| **Give-obj** | ?char-character, ?char2-character, ?obj-storable, ?place- place | El personaje 1 da el objeto al personaje 2. |
| **Trade-obj** | char-character, ?char2-character, ?obj-storable, ?obj2-storable,?place- place | El personaje 1 da el objeto 1 al personaje 2 mientras que el personaje 2 da el objeto 2 al personaje 1. |
| **Loot-obj** | ?char-character, ?char2-character, | El personaje 1 tiene el objeto que tenía el |

| | | |
|---|---|---|
| | ?obj-storable, ?place- place | personaje 2. |
| **Hurt-punch** | ?char-character, ?char2-character,?place- place | El personaje 1 hiere al personaje 2. |
| **Hurt-arm** | ?char-character, ?char2-character,?place- place, ?arm-arm | El personaje 1 hiere con el arma al personaje 2. |
| **Kill-arm** | ?char-character, ?char2-character,?place- place, ?arm-arm | El personaje 1 mata con el arma al personaje 2. |
| **Kill-and-eat-animal** | ?char-character, ?animal-edible,?place- place, ?arm-arm | El personaje 1 ya no está herido mientras que el animal muere. |
| **Hurt-by-animal** | ?char-character, ?animal-edible,?place- place | El animal hiere al personaje 1. |
| **Kill-by-animal** | ?char-character, ?animal-edible,?place- place | El animal mata al personaje 1. |
| **Kill-animal** | ?char-character, ?animal-edible,?place- place, ?arm-arm | El personaje 1 mata al animal. |
| **Kill-magic** | ?char-character, ?char2-character,?place- place, ?obj-usefulobj | El personaje 1 mata al personaje 2 (con magia). |
| **Heal-magic** | ?char-character, ?char2-character,?place- place, ?obj-usefulobj | El personaje 1 cura al personaje 2 (con magia). |
| **Heal-magic-himself** | ?char-character, ?obj-usefulobj | El personaje ya no está herido (se ha curado). |
| **Magic-become-invisible** | ?char-character, ?obj-usefulobj | El personaje se vuelve invisible. |
| **Magic-become-visible** | ?char-character, ?obj-usefulobj | El personaje se vuelve visible. |
| **Same-motive-friends** | ?char-character, ?char2-character,?place- place, ?m-motive | El personaje 2 tiene el motivo. |
| **Make-friends-** | ?char-character, ?char2- | El personaje 1 es amigo del 2 y viceversa. |

| | | |
|---|---|---|
| **motive** | character,?place- place, ?m-motive | El personaje 1 no es enemigo del 2 y viceversa. |
| **Make-friends-want-kill** | ?char-character, ?char2-character,?place- place, ?enemy-character | El personaje 1 es amigo del 2 y viceversa. El personaje 1 no es enemigo del 2 y viceversa. |
| **Make-friends-friends** | ?char-character, ?char2-character,?place- place, ?char3-character | El personaje 1 es amigo del 2 y viceversa. El personaje 1 no es enemigo del 2 y viceversa. |
| **Make-enemy-motive** | ?char-character, ?char2-character,?place- place, ?gm-goodmotive, ?em-evilmotive | El personaje 1 es enemigo del 2 y viceversa. El personaje 1 no es amigo del 2 y viceversa. |
| **Make-enemy-friends** | ?char-character, ?char2-character,?place- place, ?char3-character | El personaje 1 es enemigo del 3 y viceversa. El personaje 1 no es amigo del 3 y viceversa. |
| **Make-enemy-kill-friend** | ?char-character, ?char2-character,?place- place, ?char3-character | El personaje 1 es enemigo del 3 y viceversa. El personaje 1 no es amigo del 3 y viceversa. |
| **Learn-motive** | ?char-character, ?m-motive,?m2-motive | El personaje 1 tiene el motivo 2. |
| **Learn-motive-wish-obj** | ?char-character, ?m-motive,?obj-object, ?st-status | El personaje 1 desea el objeto. |
| **Learn-place-obj** | ?char-character, ?place-place,?obj-object, ?place2-place | El personaje ya no está en place 1, ahora está en place 2. |
| **Learn-place-obj-char** | ?char-character, ?char2-character,?place-place,?obj-object, ?place2-place | El personaje ya no está en place 1, ahora está en place 2. |
| **Learn-place-give-obj-char** | ?char-character, ?char2-character,?place-place,?obj-object, ?place2-place | El personaje ya no está en place 1, ahora está en place 2. |
| **Learn-place-obj-be-** | ?char-character, ?m-motive,?place-place,?obj-object, | El personaje ya no está en place 1, ahora está en place 2. |

| | | |
|---|---|---|
| **destroyed** | ?place2-place | |
| **Learn-place-enemy** | ?char-character, ?char2-character,?place-place, ?place2-place | El personaje ya no está en place 1, ahora está en place 2. |
| **Learn-place-food** | ?char-character, ?animal-edible,?place-place, ?place2-place | El personaje ya no está en place 1, ahora está en place 2. |
| **Learn-place-heal** | ?char-character, ?animal-edible,?place-place, ?place2-place, ?obj-usefulobj | El personaje ya no está en place 1, ahora está en place 2. |
| **Destroy-obj** | ?char-character, ?m-motive,?place-place, ?obj-object | El personaje 1 completa el motivo. El personaje 1 ya no tiene el objeto. El objeto se ha destruido. |
| **Acquire-obj** | ?char-character, ?m-motive, ?obj-object | El personaje 1 completa el motivo. El objeto es adquirido. El personaje ya no tiene el motivo m, ahora su motivo es neutral. |
| **Kill-strong** | ?char-character, ?char2-character,?place-place, ?arm-arm, ?m-motive | El personaje 2 muere y es matado por el personaje 1. |

**Tabla 40: Acciones incluidas en el escenario.**

## 4.4 Estructura y diseño de los datos internos

Uno de los puntos más importantes que define el comportamiento interno del motor es la decisión de cómo estructurar y diseñar los datos internos que necesitaba tratar el motor desarrollado. Al tratarse de una arquitectura basada en objetos y desarrollada en su totalidad en Java, lo más obvio fue el uso en todo momento del paradigma de orientación a objetos.

No obstante, dividiremos este apartado en varias secciones en las que detallaremos cada uno de los puntos necesarios en los que se llevaron a cabo decisiones de diseño.

### 4.4.1 Generación de objetos en Java

El primero de los puntos trata sobre la creación y generación de objetos en Java, en un principio vacíos, que nos servirán para almacenar los datos necesarios relativos al todo el entorno del motor.

Mediante el plugin para Eclipse JAXB se generan los objetos vacíos a partir de los ficheros XML de entrada. En este caso, es muy importante que los objetos se generen una única vez al inicio de la partida.

Cada uno de los objetos generados tiene la estructura que se indica en los ficheros XML y a su vez en los ficheros PDDL. Aunque los datos internos que puede almacenar un objeto son variados, todos ellos comparten dos métodos:

- Método SetXXX – Cuando se llama o se usa este método, le estamos indicamos el nuevo valor que contendrá la variable *XXX*, siendo *XXX* el nombre de una variable contenida en un objeto. Un ejemplo de un método Set generado sería el siguiente:

```
/**
 * Sets the value of the precondition property.
 *
 * @param value
 *     allowed object is
 *     {@link Precondition }
 *
 */
public void setPrecondition(Precondition value) {
    this.precondition = value;
}
```

Ilustración 16: Método setPrecondition.

- Método GetXXX: Cuando se llama o se usa este método, le estamos indicando que queremos recuperar el valor de la variable *XXX* siendo *XXX* una variable contenida en un objeto. Un ejemplo de un método Get generado sería el siguiente:

```
/**
 * Gets the value of the precondition property.
 *
 * @return
 *     possible object is
 *     {@link Precondition }
 *
 */
public Precondition getPrecondition() {
    return precondition;
}
```

Ilustración 17: Método getPrecondition.

Por otro lado, otra gran ventaja que nos brinda JAXB es la estructura que nos genera en clases de los objetos generados. Las clases generadas para nuestro dominio y problema en particular corresponderían a las siguientes:

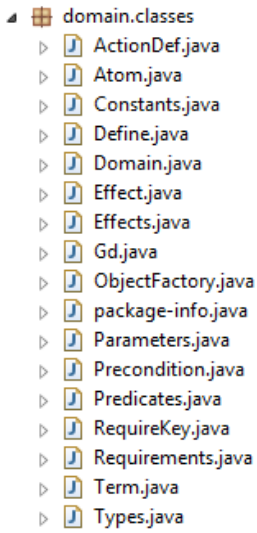


Ilustración 18 : Clases generadas del dominio.

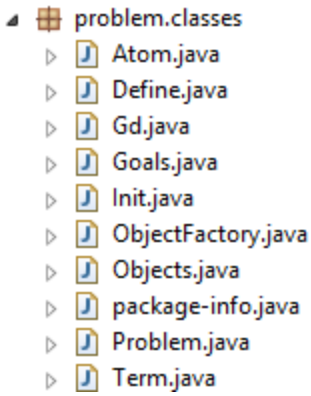


Ilustración 19: Clases generadas del problema.

### 4.4.2 Estructura de las peticiones.

Otro de los puntos importantes a tener en cuenta fue cómo se iban a recibir las peticiones, ya que no se impuso ningún tipo de restricción sobre este punto.

Una vez estudiadas varias alternativas, se decidió que las peticiones tendrían la siguiente estructura:

i) Nombre de la acción a ejecutar.
ii) Parámetros necesarios para ejecutar la acción. Todos ellos separados por comas.

Así, un modelo de petición sería el siguiente:

```
"(take-obj,frodo,stone,mordor)"
```

Ilustración 20: Estructura de la petición.

Como puede comprobarse, la acción que se desea realizar es la llamada "*Take-obj*" que requiere tres parámetros para su correcta ejecución. El primero de ellos corresponde a un *char (tipo character)*, el segundo corresponde a un *obj (tipo storable)*, mientras que el último corresponde a un *place (tipo place).*

Si las peticiones no se realizan con el formato descrito anteriormente, el motor no será capaz de interpretarlas y responderá con un mensaje indicando que se está formando mal la petición.

Parámetros de entrada incorrectos

Ilustración 21: Mensaje de parámetros incorrectos.

## 4.5 Funcionamiento del sistema.

La descripción del funcionamiento del sistema se ha realizado a través de diagramas de secuencia, con este tipo de diagramas es posible detallar las interacciones entre los distintos componentes del sistema que se producen al realizar una acción. A continuación, se describen las acciones que se han desarrollado.

### 4.5.1 Identificación mediante la interfaz gráfica.

En el siguiente diagrama se describe el proceso en el que un usuario se identifica a través de la interfaz gráfica.

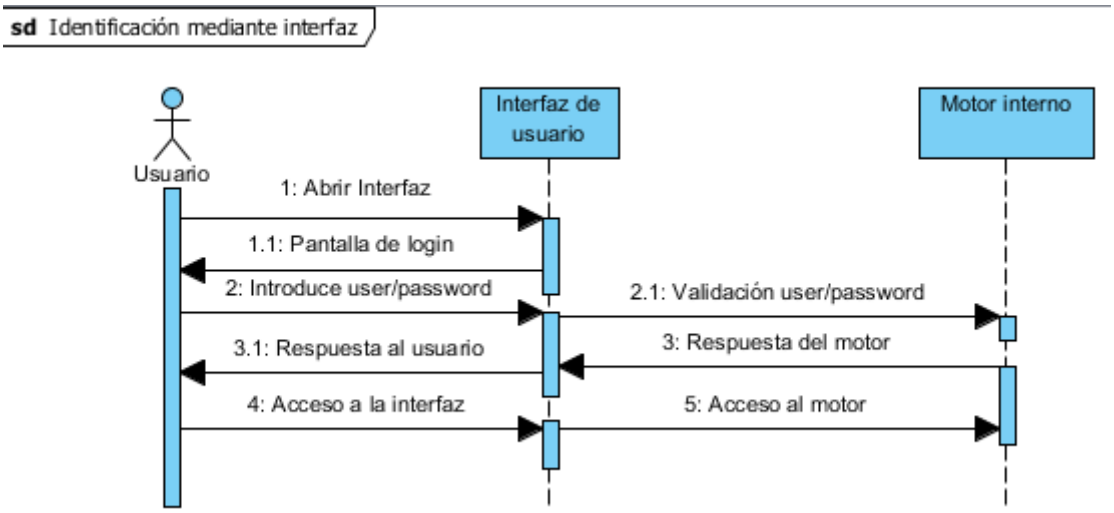


Ilustración 22: Diagrama de secuencia- identificación mediante interfaz.

A continuación se describe la secuencia detalladamente:

1 **Abrir interfaz**: El usuario abre la interfaz del juego para comenzar a interactuar con ella.

1.1 **Pantalla de login**: Una vez iniciada la interfaz, se devuelve al usuario la pantalla principal de login, pidiendo *user* y *password.*

2 **Introduce user/password**: el usuario rellena los campos de *user* y *password* con sus datos de acceso.

2.1 **Validación de datos**: Se envían los datos al motor y se validan para corroborar que el usuario existe y puede acceder a la interfaz.

3 **Respuesta del motor**: se envía la respuesta desde el motor a la interfaz.

3.1 **Respuesta al usuario**: se muestra un mensaje al usuario indicando si todo ha ido bien o bien un mensaje de error para informar que los datos que ha introducido no son válidos.

4 **Acceso a la interfaz** : Si los datos de login son correctos, se accede a la interfaz.

5 **Acceso al motor**: Una vez que accedemos a la interfaz, seguidamente accedemos al motor y comienza la partida.

### 4.5.2 Realizar peticiones a través de la interfaz.

En el siguiente diagrama de secuencia se describe realización de peticiones por parte de los usuarios a través de la interfaz gráfica.

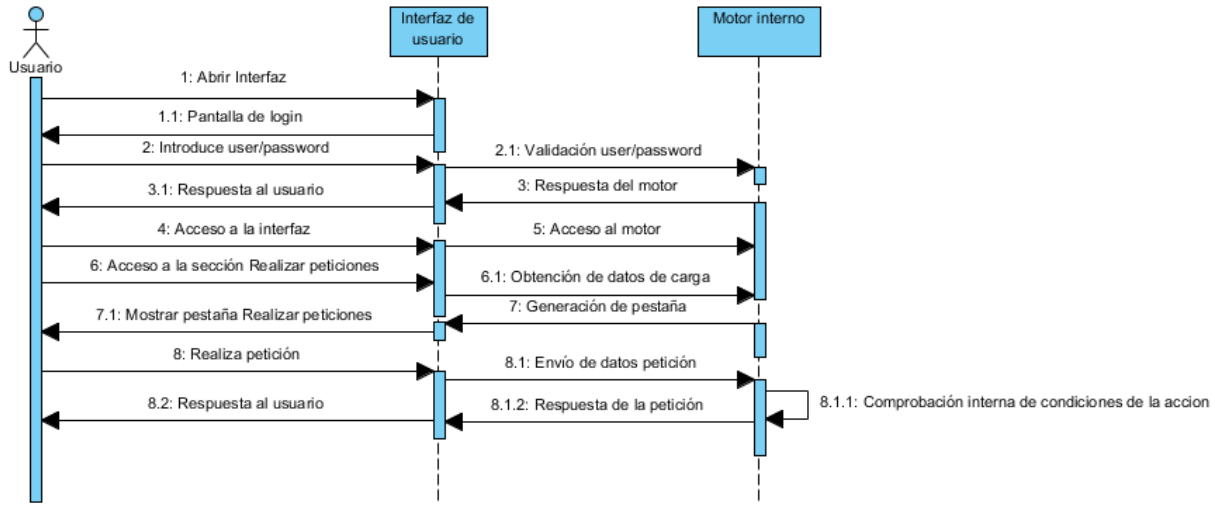


Ilustración 23: Diagrama de secuencia- realización de peticiones mediante interfaz.

Los 5 primeros pasos de esta secuencia son idénticos al diagrama anterior, por lo tanto, no es necesario describirlos de nuevo. A continuación, se describen los pasos a seguir para realizar peticiones una vez obtenido acceso al motor:

6 **Acceso a la sección "Realizar Peticiones"**: el usuario identificado accede a la sección de realización de peticiones.
   6.1 **Obtención de datos de carga**: la interfaz pide al motor los datos necesarios para poder generar la esta pestaña.
7 **Generación de datos de carga**: el motor devuelve los datos requeridos a la interfaz para que se puede visualizar correctamente.
   7.1 **Mostrar pestaña**: el usuario puede visualizar ahora la pestaña de realización de peticiones con todas sus opciones disponibles.
8 **Realiza petición**: el usuario realiza una determinada petición en la interfaz.
   8.1 **Envío de datos de la petición**: la interfaz recoge los datos de la petición que acaba de realizar el usuario y se los envía al motor para que los procese.
      8.1.1 **Comprobaciones internas**: El motor debe realiza las comprobaciones internas correspondientes para saber si la acción se puede llevar a cabo o no.
      8.1.2 **Respuesta de la petición**: El servidor responde a la interfaz si ha podido llevar a cabo la petición o no.
   8.2 **Respuesta al usuario**: La interfaz muestra los datos obtenidos al usuario.

### 4.5.3 Consultar objetos del escenario a través de la interfaz.

En el siguiente diagrama de secuencia se describe la consulta de los objetos existentes en el escenario por parte de los usuarios a través de la interfaz gráfica.

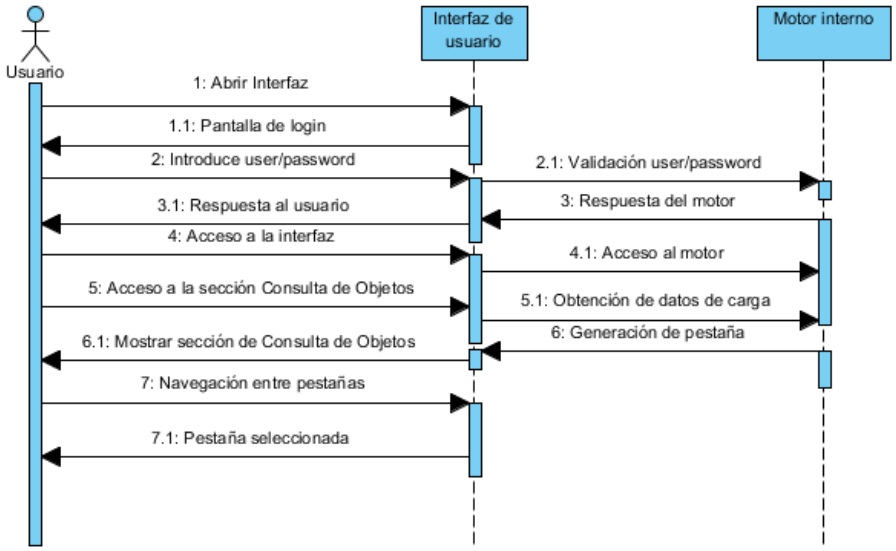


**Ilustración 24: Diagrama de secuencia- consulta de objetos mediante interfaz.**

Hasta el paso 4.1 incluido, corresponden a los pasos explicados anteriormente necesarios para realizar login en la aplicación, por lo tanto, no es necesario describirlos de nuevo. A continuación, se describen los pasos a seguir para consultar los objetos del escenario, una vez obtenido acceso al motor:

5 **Acceso a la sección "Consulta de Objetos"**: el usuario identificado accede a la sección de consulta de objetos existentes en el escenario.

5.1 **Obtención de datos de carga**: la interfaz pide al motor los datos necesarios para poder generar la esta pestaña.

6 **Generación de pestaña**: el motor devuelve los datos requeridos a la interfaz para que se pueda visualizar correctamente la sección accedida.

6.1 **Mostrar sección**: el usuario puede visualizar ahora la pestaña de consulta de objetos disponibles, así como todas sus pestañas internas disponibles.

7 **Navegación entre pestañas**: el usuario puede navegar entre las distintas pestañas que contiene la sección de consulta de objetos.

7.1 **Pestaña seleccionada**: la interfaz proporciona en cada caso la vista correspondiente a la pestaña seleccionada.

### 4.5.4 Consultar listado de peticiones a través de la interfaz.

En el siguiente diagrama de secuencia se describe la consulta de la lista de las peticiones ya realizadas en el motor a través de la interfaz gráfica.

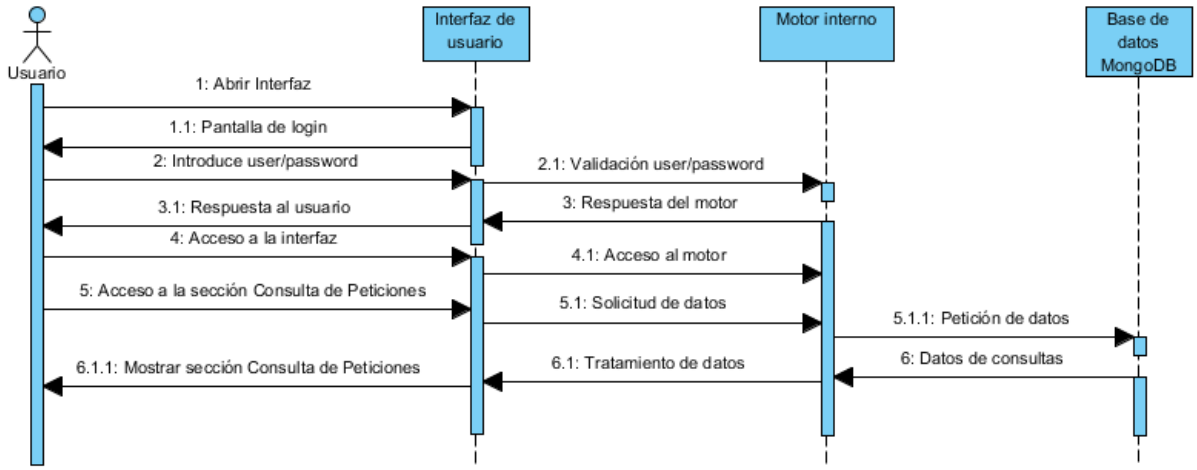


Ilustración 25: Diagrama de secuencia- consulta de peticiones.

En el diagrama anterior los pasos comprendidos entre el 1 y el 4.1 incluido, corresponden a los pasos explicados anteriormente necesarios para realizar login en la aplicación, por lo tanto, no es necesario describirlos de nuevo. A continuación, se describen los pasos a seguir para consultar la lista de peticiones realizadas. Se ha de recordar nuevamente en este punto, que únicamente un usuario con perfil de **administrador** puede acceder a la visualización de esta sección.

5. **Acceso a la sección "Consulta de Peticiones"**: si el usuario identificado posee rol de administrador, accede a la sección de consulta de peticiones realizadas en la partida actual.
    5.1. **Solicitud de datos**: la interfaz pide al motor que le devuelva la lista de todas las peticiones realizadas para poder mostrarlas al usuario.
        5.1.1. **Petición de datos**: el motor realiza una consulta a la base de datos de MongoDB para recuperar todas las peticiones que se hayan realizado en esa partida.
6. **Datos de consultas**: la base de datos responde y devuelve los datos (si los hay) de todas las peticiones que se hayan realizado.
    6.1. **Tratamiento de datos**: el motor trata los datos en bruto obtenido de la base de datos y posteriormente los envía a la interfaz.

6.2. **Mostrar sección "Consulta de Peticiones"**: la interfaz recibe los datos tratados y los aplica sobre la vista de la sección, mostrando al usuario una visualización general del total de la lista de peticiones.

### 4.5.5 Aplicar filtro a listado de peticiones a través de la interfaz

En el siguiente diagrama de secuencia se describe la aplicación de un filtro una vez obtenida la lista de peticiones totales.

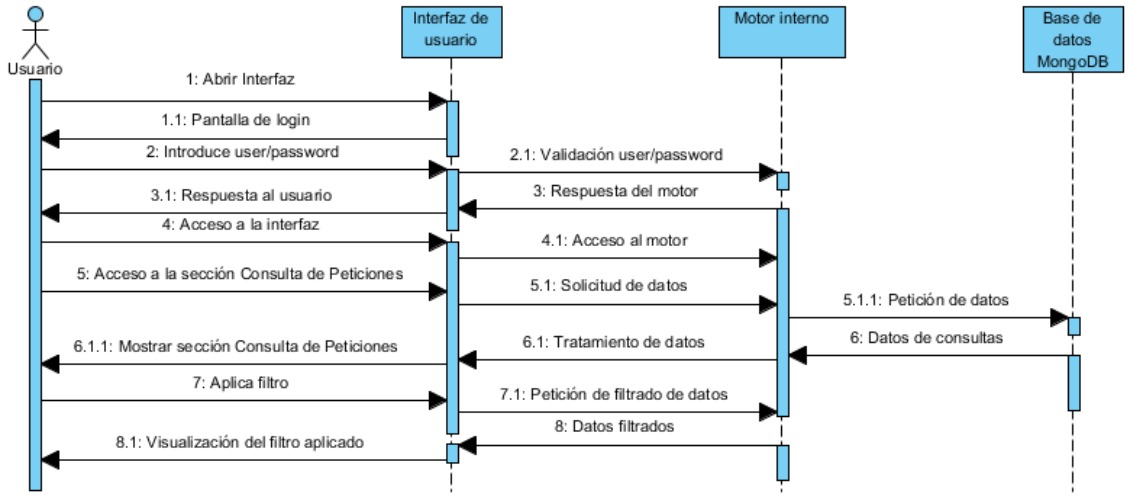


Ilustración 26: Diagrama de secuencia- Aplicar filtro a lista de peticiones.

Los 6 primeros pasos son idénticos al diagrama anterior, por lo que no es necesario volverlos a explicar. A partir del paso 7 se explica cómo se aplica un filtro y los procesos intermedios relacionados.

7. **Aplicar filtro:** una vez obtenida la lista total de peticiones realizadas, el usuario configura y establece uno de los filtros disponibles.
    7.1. **Petición de filtrado de datos**: la interfaz envía al motor la petición de filtro con la configuración y el filtro seleccionados.
8. **Datos filtrados**: el motor aplica de manera lógica el filtro a los datos y devuelve a la interfaz unos nuevos datos obtenidos con el filtro ya aplicado para que ésta se encargue de pintarlos en pantalla.

8.1. **Visualización del filtro aplicado**: la interfaz recibe los nuevos datos y sobrescribe en pantalla la lista ya existente con los nuevos datos filtrados.

### 4.5.6 Eliminar filtro a listado de peticiones filtradas a través de la interfaz

En el siguiente diagrama de secuencia se describe la eliminación de un filtro previamente añadido en la lista de peticiones realizadas.

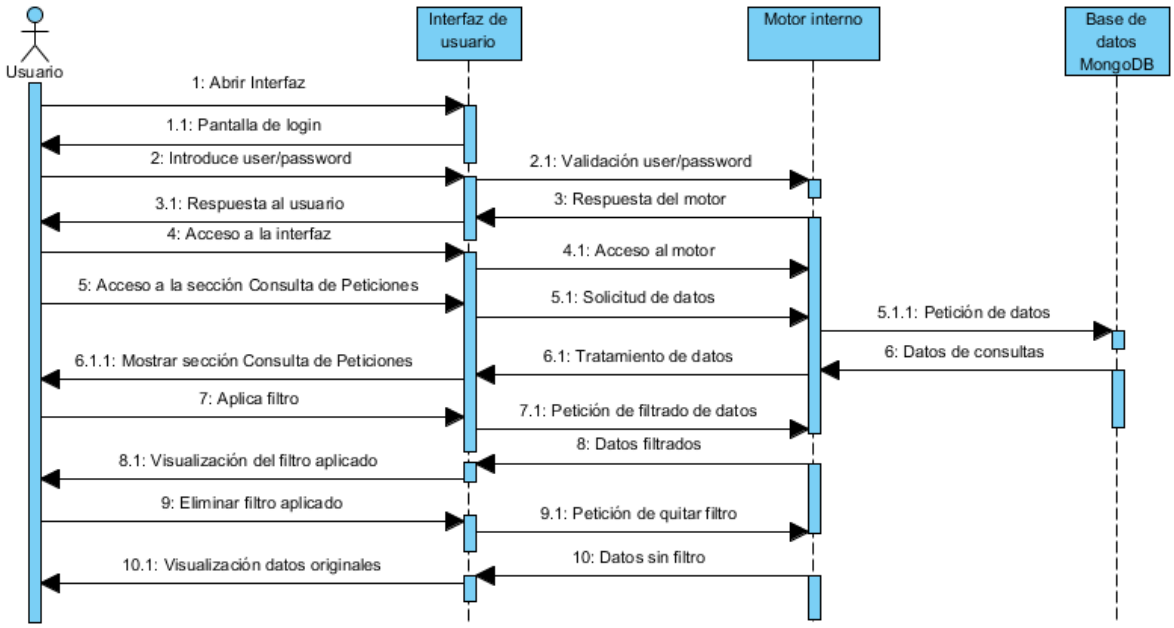


**Ilustración 27: Diagrama de secuencia- Eliminar filtro de lista de peticiones.**

Los 8 primeros pasos son idénticos al diagrama anterior, por lo que no es necesario volverlos a explicar. A partir del paso 9 se explica cómo se elimina un filtro previamente añadido en la lista de peticiones realizadas, así como sus pasos intermedios.

9. **Eliminar filtro aplicado:** el usuario elimina el filtro activo sobre la lista de peticiones.
   9.1. **Petición de quitar filtro**: la interfaz pide al motor interno que le elimine el filtro que está activo actualmente sobre la lista de las peticiones.

10. **Datos sin filtro**: el motor realiza un tratamiento de los datos y elimina el filtro activo, obteniendo de nuevo la lista original son ningún tipo de configuración ni filtros. Seguidamente envía estos datos originales a la interfaz para que se visualicen en pantalla.
    10.1. **Visualización de datos originales**: la interfaz recibe la información con la nueva lista y la muestra por pantalla al usuario.

### 4.5.7 Desconectarse a través de la interfaz

En el siguiente diagrama de secuencia se describe la desconexión del sistema, tanto de la interfaz como del motor.

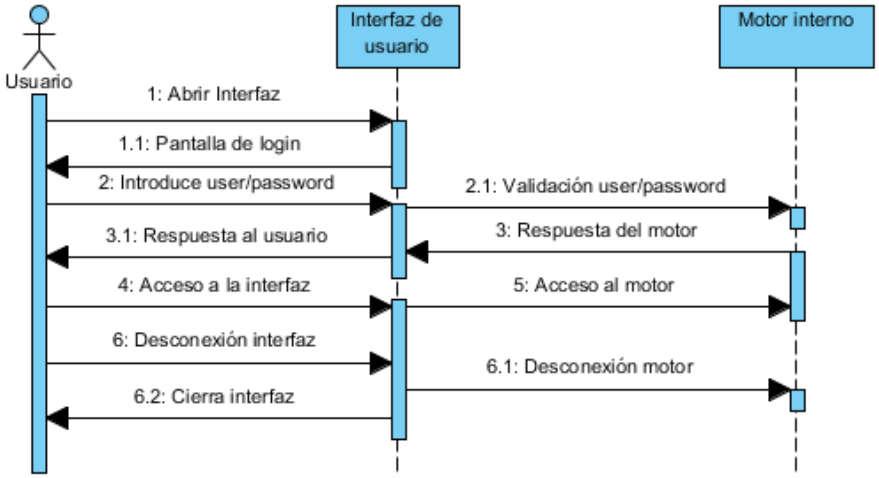


Ilustración 28: Diagrama de secuencia- Desconexión del sistema.

Como puede observarse, en el diagrama anterior los 5 primeros pasos corresponden al proceso de identificación y acceso del usuario al sistema ya explicado, por este motivo, a continuación se describen el resto de los pasos.

6. **Desconexión interfaz:** el usuario realiza clic en el icono para salir de la interfaz. La interfaz recibe una señal que le indica que el usuario desea cerrar la conexión tanto con la parte visual (interfaz) como con el motor
    6.1. **Desconexión motor**: la interfaz envía al motor una segunda señal para que pare su servicio y se desconecte.
    6.2. **Cierra interfaz**: una vez terminada la conexión con el motor, la interfaz se cierra

## 4.6 Diseño de la interfaz gráfica

Las nuevas funcionalidades añadidas en este trabajo de fin de grado han requerido de nuevas vistas dentro del sistema. Con el fin de crear una interfaz gráfica lo más adecuada al proyecto posible, desde el punto de vista estético, se ha seguido la temática de **El Señor de los Anillos**.

A continuación, se muestran las vistas creadas para la interfaz:


Ilustración 29: Vista de autenticación de la interfaz.

En la figura X se puede ver la ventana de autenticación inicial que nos proporciona la interfaz gráfica. Para simplificar la tarea al usuario, únicamente dispone de dos cuadros de texto, el primero de ellos para introducir el usuario y el segundo para la contraseña. Para brindar mayor seguridad, no se muestra ni el usuario ni la contraseña escrita en estos cuadros de texto.

Una vez rellenos tanto usuario como contraseña, sólo es necesario realizar clic sobre el botón ACCEDER. Si la autenticación del usuario ha ido correctamente, antes de acceder al menú principal del juego, nos aparecerá un cuadro de diálogo preguntándonos si deseamos crear una nueva partida o continuar la anterior.

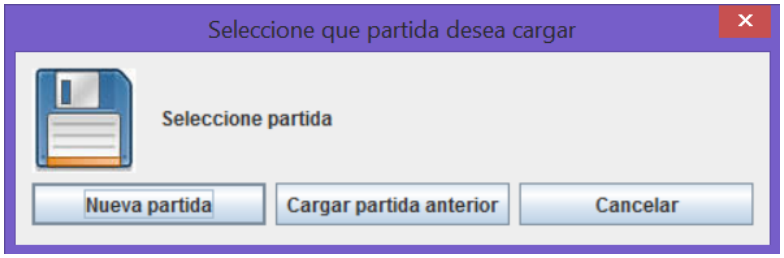


Ilustración 30: Cuadro de diálogo partida a cargar.

Si se elige *Nueva Partida* los datos de la partida anterior (peticiones) se borrarán, por lo que la interfaz nos avisa con el siguiente mensaje:

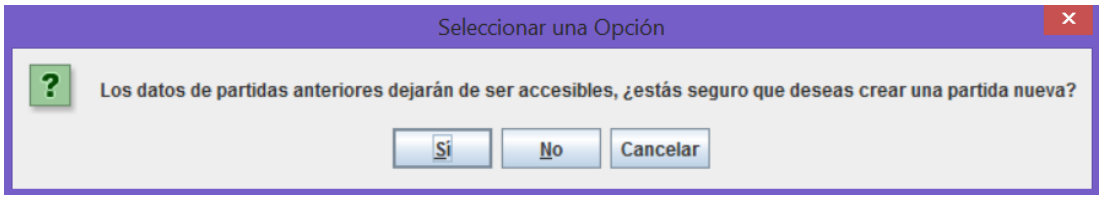


Ilustración 31: Cuadro de diálogo confirmación nueva partida.

En caso contrario, si se desea proseguir la partida anterior donde se dejó, se accederá sin ningún tipo de problema al menú principal de la interfaz.

### 4.6.1 Menú principal de la interfaz

La vista correspondiente al menú principal es la siguiente:


Ilustración 32: Menú principal interfaz.

Como se puede observar en la ilustración anterior, se ha querido simplificar y facilitar el uso al usuario final, por lo que diferencian tres grandes iconos en el centro de la interfaz. Cada uno de ellos corresponde a una determinada sección y para saber cuál es el nombre de dicha sección, únicamente hay que pasar por encima el cursor:


Ilustración 33: Secciones con texto descriptivo.

### 4.6.2 Sección de realización de peticiones

La vista correspondiente a la sección de realización de peticiones es la siguiente

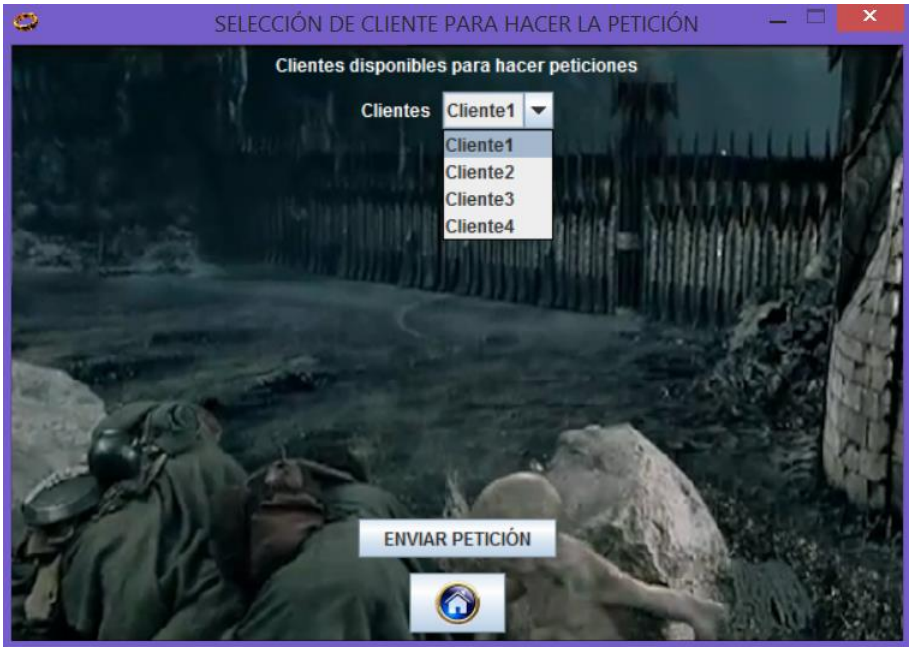


Ilustración 34: Sección para realizar peticiones.

Para realizar una petición, únicamente es necesario elegir el cliente con el cual vamos a realizar la petición (en caso de estar con usuario administrador) y realizar clic sobre el botón "*Enviar Petición*".

### 4.6.3 Sección de consulta de peticiones

La sección de consulta de peticiones corresponde a otra nueva vista en la que se pueden visualizar todas aquellas peticiones realizadas en la partida actual.

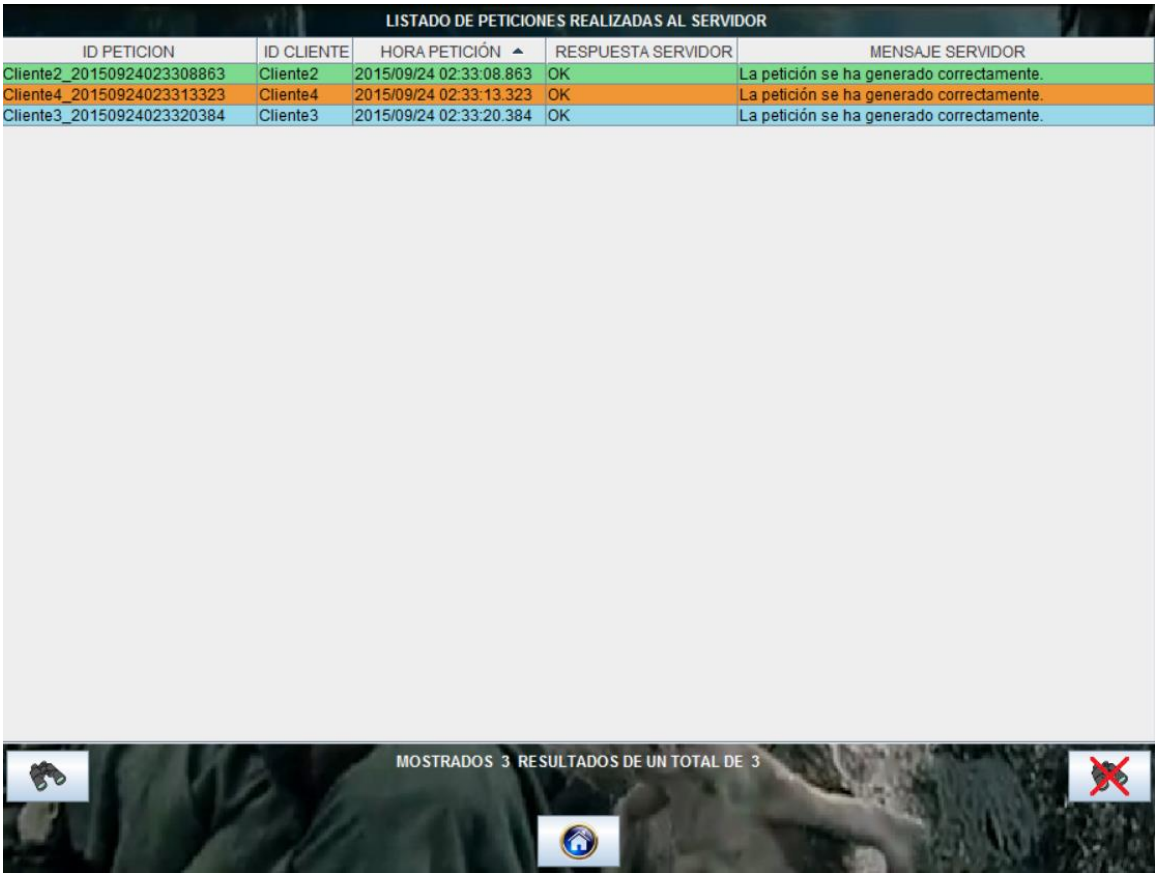


**Ilustración 35: Sección para listar peticiones.**

Además de mostrar las peticiones realizadas, en esta sección podemos crear y configurar nuestro propios filtros y eliminar un filtro previamente creado.

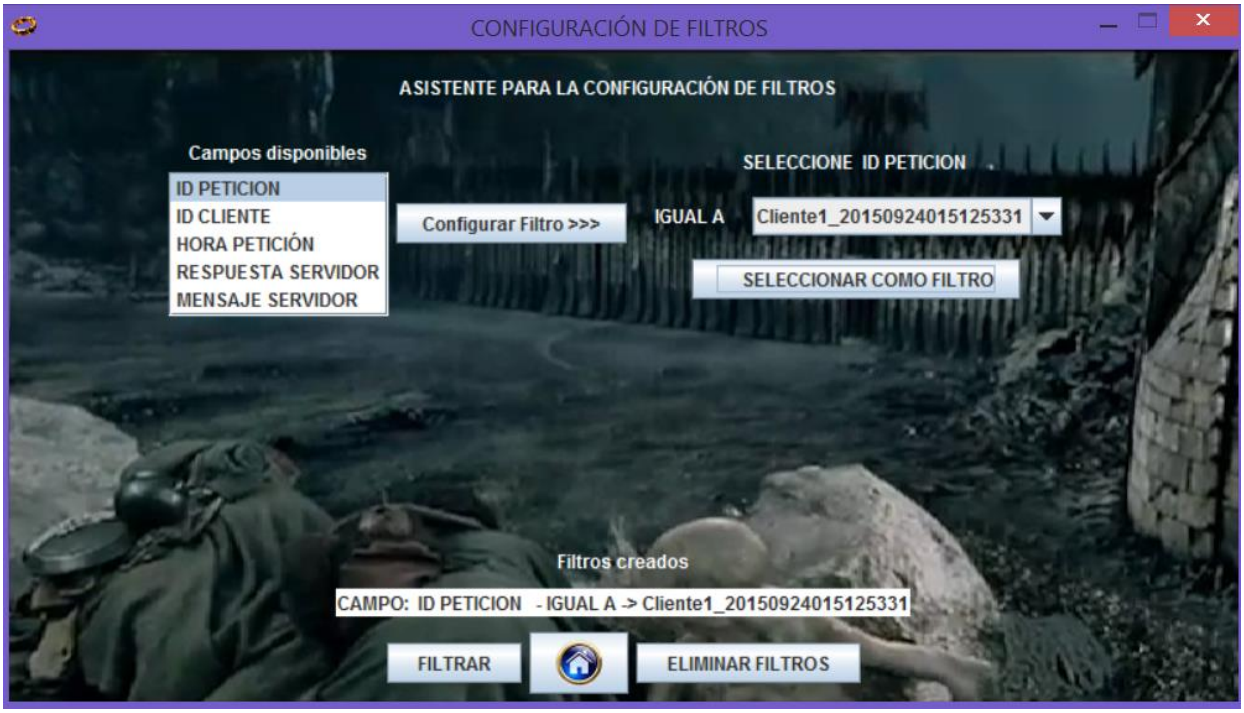


Ilustración 36: Configuración de filtro.

### 4.6.4 Sección de consulta de objetos del escenario

Para terminar con las secciones de la interfaz, la última de ellas corresponde a un conjunto de pestañas que contienen los objetos, personajes, motivos, etc. Que existen en nuestro escenario en particular.

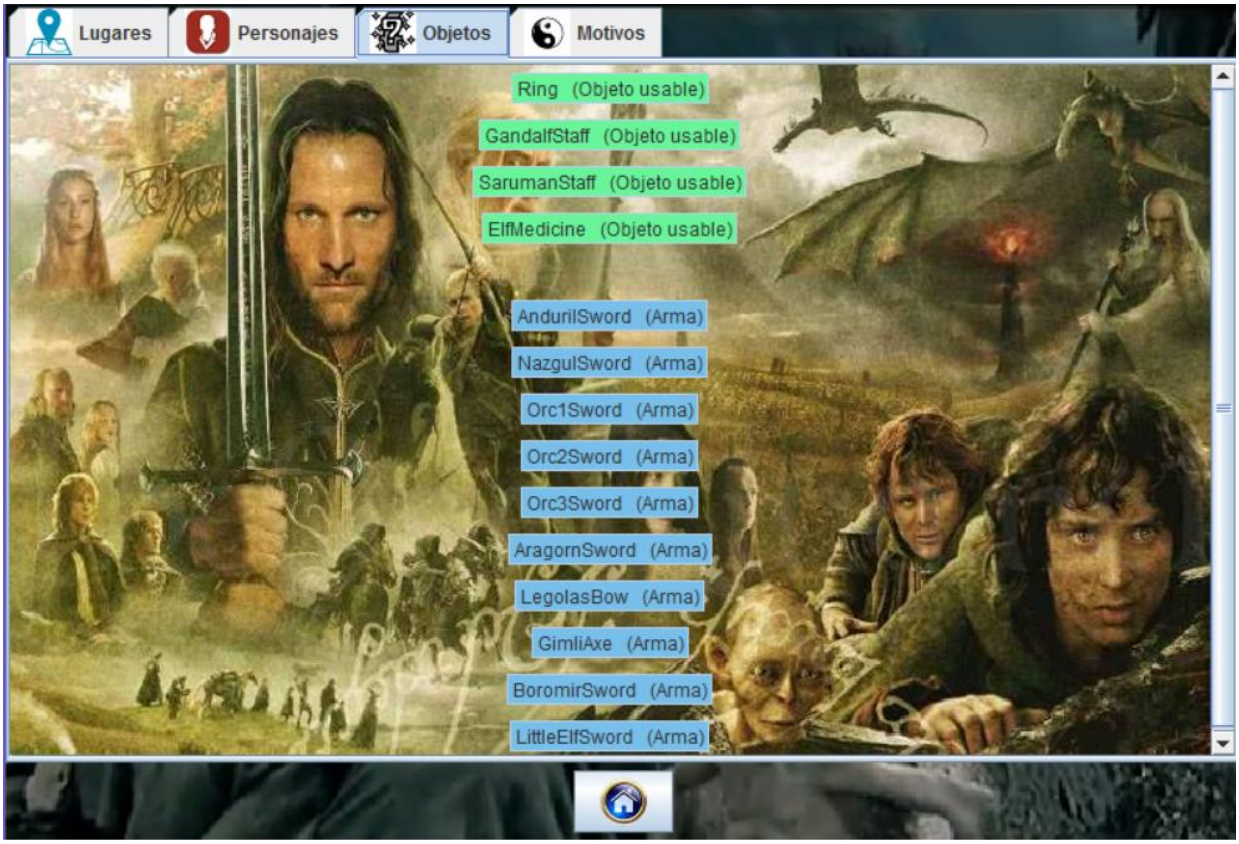


Ilustración 37: Entidades presentes en el escenario.

Para finalizar con el diseño de la interfaz, como en toda pantalla, era necesaria la creación de un botón que realizase una desconexión del sistema. Por este motivo, en el menú principal existe un botón de "apagado" para dar la opción a un usuario de salir de la aplicación.

Si realizamos clic sobre este botón, nos aparecerá un cuadro de diálogo como el siguiente:

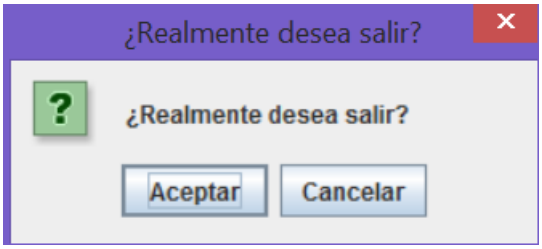


Ilustración 38: Cuadro de diálogo salir del sistema.

# 5 Pruebas del Sistema

En este apartado se definen las pruebas que debe superar la solución desarrollada para considerarse válida.

## 5.1 Especificación de pruebas del sistema

Con el objetivo de comprobar que el sistema desarrollado funciona correctamente se ha desarrollado un plan de pruebas del sistema que servirá para validar que el sistema cumple todos los requisitos planteados en este trabajo.

### 5.1.1 Formato de las pruebas

A continuación se describen los atributos que definen cada una de las pruebas.

- Identificador: Identifica cada prueba del sistema. El formato utilizado es el siguiente "PS-XX", donde XX se corresponde con un número.
- Descripción: Descripción de la prueba.
- Pasos: Secuencia de pasos que debe seguirse para realizar la prueba.
- Requisitos: Se indican que requisitos funcionales se están comprobando con la prueba.
- Resultado esperado: Resultado que se espera de la realización de la prueba.

| Parámetro | Descripción |
|---|---|
| **Identificador** | |
| **Descripción** | |
| **Pasos** | |
| **Requisitos** | |
| **Resultado esperado** | |

**Tabla 41: Ejemplo de tabla de pruebas de sistema.**

### 5.1.2 Formato de las pruebas

En este apartado se exponen todas las pruebas del sistema:

| Parámetro | Descripción |
|---|---|
| **Identificador** | PS-01 |
| **Descripción** | Acceso al motor con user y password de administrador correctos a través de la interfaz gráfica. |
| **Pasos** | 1. Se abre la interfaz.<br>2. Se introduce el usuario y contraseña correctos de administrador.<br>3. Se realiza clic sobre el botón “Acceder”. |
| **Requisitos** | RF-01 |
| **Resultado esperado** | Se accede al menú principal de la interfaz. |

**Tabla 42: PS-01. Acceso al motor con user y password de administrador correctos a través de la interfaz gráfica.**

| Parámetro | Descripción |
|---|---|
| **Identificador** | PS-02 |
| **Descripción** | Acceso al motor con user y password de administrador erróneos a través de la interfaz gráfica. |
| **Pasos** | 1. Se abre la interfaz.<br>2. Se introduce el usuario y contraseña erróneos de administrador.<br>3. Se realiza clic sobre el botón “Acceder”.<br>4. Se muestra un mensaje de datos erróneos. |
| **Requisitos** | RF-01 |
| **Resultado esperado** | Se accede al menú principal de la interfaz. |

**Tabla 43: PS-02. Acceso al motor con user y password de administrador erróneos a través de la interfaz gráfica.**

| Parámetro | Descripción |
|---|---|
| **Identificador** | PS-03 |
| **Descripción** | Validación de los credenciales de administrador. |
| **Pasos** | 1. Se abre la interfaz.<br>2. Se introduce el usuario y contraseña de administrador.<br>3. Se realiza clic sobre el botón “Acceder”.<br>4. Se muestra el mensaje de elegir partida. |
| **Requisitos** | RF-02 |
| **Resultado esperado** | Se muestra el mensaje de elegir partida. |

**Tabla 44: PS-03. Validación de los credenciales de administrador.**

| Parámetro | Descripción |
|---|---|
| **Identificador** | PS-04 |
| **Descripción** | Administrador puede suplantar clientes |
| **Pasos** | 1. Se abre la interfaz.<br>2. Se introduce el usuario y contraseña de administrador.<br>3. Se realiza clic sobre el botón “Acceder”.<br>4. Se muestra el menú principal con todas las secciones disponibles.<br>5. Se accede a la sección “Realizar Petición”<br>6. Se observa que se pueden realizar peticiones con cualquiera de los clientes. |
| **Requisitos** | RF-03 |
| **Resultado esperado** | Se suplanta un cliente para realizar una petición. |

**Tabla 45: PS-04. Administrador puede suplantar clientes**

| Parámetro | Descripción |
|---|---|
| **Identificador** | PS-05 |
| **Descripción** | Administrador accede a consultas. |
| **Pasos** | 1. Se abre la interfaz.<br>2. Se introduce el usuario y contraseña de administrador.<br>3. Se realiza clic sobre el botón “Acceder”.<br>4. Se muestra el menú principal con todas las secciones disponibles, incluida las consultas (sólo admin). |
| **Requisitos** | RF-04,RF-05 |
| **Resultado esperado** | Se visualizan todas las secciones de la interfaz. |

**Tabla 46: PS-05. Administrador accede a consultas.**

| Parámetro | Descripción |
|---|---|
| **Identificador** | PS-06 |
| **Descripción** | Administrador puede filtrar peticiones. |
| **Pasos** | 1. Se abre la interfaz.<br>2. Se introduce el usuario y contraseña de administrador.<br>3. Se realiza clic sobre el botón “Acceder”.<br>4. Se muestra el menú principal con todas las secciones disponibles, incluida las consultas (sólo admin).<br>5. Se accede a la sección de consulta de peticiones.<br>6. Se hace clic en el icono de los prismáticos.<br>7. Se crea y configura un filtro<br>8. Aparece la lista de peticiones filtrada. |
| **Requisitos** | RF-06 |
| **Resultado esperado** | Aparece la lista filtrada por el filtro establecido. |

**Tabla 47: PS-06. Administrador puede filtrar peticiones.**

| Parámetro | Descripción |
|---|---|
| **Identificador** | PS-07 |
| **Descripción** | Administrador puede eliminar filtros establecidos. |
| **Pasos** | 1. Se abre la interfaz.<br>2. Se introduce el usuario y contraseña de administrador.<br>3. Se realiza clic sobre el botón “Acceder”.<br>4. Se muestra el menú principal con todas las secciones disponibles, incluida las consultas (sólo admin).<br>5. Se accede a la sección de consulta de peticiones.<br>6. Se hace clic en el icono de los prismáticos.<br>7. Se crea y configura un filtro<br>8. Aparece la lista de peticiones filtrada.<br>9. Se hace clic en el icono de los prismáticos con un aspa.<br>10. Aparece la lista original sin filtro. |
| **Requisitos** | RF-07 |
| **Resultado esperado** | Aparece la lista original sin filtro. |

**Tabla 48: PS-07. Administrador puede eliminar filtros establecidos.**

| Parámetro | Descripción |
|---|---|
| **Identificador** | PS-08 |
| **Descripción** | Administrador puede ver objetos del escenario. |
| **Pasos** | 1. Se abre la interfaz.<br>2. Se introduce el usuario y contraseña de administrador.<br>3. Se realiza clic sobre el botón “Acceder”.<br>4. Se muestra el menú principal con todas las secciones disponibles, incluida las consultas (sólo admin).<br>5. Se accede a la sección de consulta de objetos del escenario.<br>6. Se observan todos los objetos existentes en el escenario. |
| **Requisitos** | RF-08 |
| **Resultado esperado** | Se observan todos los objetos existentes en el escenario. |

**Tabla 49: PS-08. Administrador puede ver objetos del escenario.**

| Parámetro | Descripción |
|---|---|
| **Identificador** | PS-09 |
| **Descripción** | Administrador puede realizar logout. |
| **Pasos** | 1. Se abre la interfaz.<br>2. Se introduce el usuario y contraseña de administrador.<br>3. Se realiza clic sobre el botón “Acceder”.<br>4. Se muestra el menú principal con todas las secciones disponibles, incluida las consultas (sólo admin).<br>5. Se hace clic sobre el botón de desconexión.<br>6. Se desconecta al administrador del sistema y la interfaz desaparece. |
| **Requisitos** | RF-09 |
| **Resultado esperado** | Se desconecta al administrador del sistema y la interfaz desaparece. |

**Tabla 50: PS-09. Administrador puede realizar logout.**

| Parámetro | Descripción |
|---|---|
| **Identificador** | PS-10 |
| **Descripción** | Acceso al motor con user y password (no administrador) correctos a través de la interfaz gráfica. |
| **Pasos** | 1. Se abre la interfaz.<br>2. Se introduce el usuario y contraseña correctos (no administrador).<br>3. Se realiza clic sobre el botón “Acceder”. |
| **Requisitos** | RF-10 |
| **Resultado esperado** | Se accede al menú principal de la interfaz. |

**Tabla 51: PS-10. Acceso al motor con user y password no administrador correctos a través de la interfaz gráfica.**

| Parámetro | Descripción |
|---|---|
| **Identificador** | PS-11 |
| **Descripción** | Acceso al motor con user y password (no administrador) erróneos a través de la interfaz gráfica. |
| **Pasos** | 1. Se abre la interfaz.<br>2. Se introduce el usuario y contraseña erróneos (no administrador).<br>3. Se realiza clic sobre el botón “Acceder”.<br>4. Se muestra un mensaje de datos erróneos. |
| **Requisitos** | RF-10 |
| **Resultado esperado** | Se accede al menú principal de la interfaz. |

**Tabla 52: PS-11. Acceso al motor con user y password (no administrador) erróneos a través de la interfaz gráfica.**

| Parámetro | Descripción |
|---|---|
| **Identificador** | PS-12 |
| **Descripción** | Validación de los credenciales (no administrador). |
| **Pasos** | 1. Se abre la interfaz.<br>2. Se introduce el usuario y contraseña (no administrador).<br>3. Se realiza clic sobre el botón “Acceder”.<br>4. Se muestra el mensaje de elegir partida. |
| **Requisitos** | RF-11 |
| **Resultado esperado** | Se muestra el mensaje de elegir partida. |

**Tabla 53: PS-12. Validación de los credenciales (no administrador).**

| Parámetro | Descripción |
|---|---|
| **Identificador** | PS-13 |
| **Descripción** | Usuario común no pueden suplantar clientes. |
| **Pasos** | 1. Se abre la interfaz.<br>2. Se introduce el usuario y contraseña (no administrador).<br>3. Se realiza clic sobre el botón “Acceder”.<br>4. Se muestra el menú principal con todas las secciones disponibles excepto consultar peticiones.<br>5. Se accede a la sección “Realizar Petición”<br>6. Se observa que sólo se puede realizar petición con su usuario. |
| **Requisitos** | RF-12 |
| **Resultado esperado** | No se puede suplantar a otro cliente. |

**Tabla 54: PS-13. Usuario común no pueden suplantar clientes.**

| Parámetro | Descripción |
|---|---|
| **Identificador** | PS-14 |
| **Descripción** | Usuario común no accede a consultas. |
| **Pasos** | 1. Se abre la interfaz.<br>2. Se introduce el usuario y contraseña (no administrador).<br>3. Se realiza clic sobre el botón “Acceder”.<br>4. Se muestra el menú principal con todas las secciones disponibles excepto consultar peticiones. |
| **Requisitos** | RF-13,RF-14 |
| **Resultado esperado** | Se muestra el menú principal con todas las secciones disponibles excepto consultar peticiones. |

**Tabla 55: PS-14. Usuario común no accede a consultas.**

| Parámetro | Descripción |
|---|---|
| **Identificador** | PS-15 |
| **Descripción** | Usuario común puede ver objetos del escenario. |
| **Pasos** | 1. Se abre la interfaz.<br>2. Se introduce el usuario y contraseña (no administrador).<br>3. Se realiza clic sobre el botón “Acceder”.<br>4. Se muestra el menú principal con todas las secciones disponibles excepto consultar peticiones.<br>5. Se accede a la sección de consulta de objetos del escenario.<br>6. Se observan todos los objetos existentes en el escenario. |
| **Requisitos** | RF-15 |
| **Resultado esperado** | Se observan todos los objetos existentes en el escenario. |

**Tabla 56: PS-15. Usuario común puede ver objetos del escenario.**

| Parámetro | Descripción |
|---|---|
| **Identificador** | PS-16 |
| **Descripción** | Usuario común puede realizar logout. |
| **Pasos** | 1. Se abre la interfaz.<br>2. Se introduce el usuario y contraseña (no administrador).<br>3. Se realiza clic sobre el botón “Acceder”.<br>4. Se muestra el menú principal con todas las secciones disponibles excepto consultar peticiones.<br>5. Se hace clic sobre el botón de desconexión.<br>6. Se desconecta al usuario común del sistema y la interfaz desaparece. |
| **Requisitos** | RF-16 |
| **Resultado esperado** | Se desconecta al usuario común del sistema y la interfaz desaparece. |

**Tabla 57: PS-16. Usuario común puede realizar logout.**

### 5.1.3 Trazabilidad entre pruebas de sistema y requisitos funcionales

Con el objetivo de comprobar que el sistema cumple todos los requisitos funcionales, se ha realizado una matriz de trazabilidad que muestra la correspondencia entre las pruebas del sistema y los requisitos funcionales:

| | RF-01 | RF-02 | RF-03 | RF-04 | RF-05 | RF-06 | RF-07 | RF-08 | RF-09 | RF-10 | RF-11 | RF-12 | RF-13 | RF-14 | RF-15 | RF-16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PS-01 | X | | | | | | | | | | | | | | | |
| PS-02 | X | | | | | | | | | | | | | | | |
| PS-03 | | X | | | | | | | | | | | | | | |
| PS-04 | | | X | | | | | | | | | | | | | |
| PS-05 | | | | X | X | | | | | | | | | | | |
| PS-06 | | | | | | X | | | | | | | | | | |
| PS-07 | | | | | | | X | | | | | | | | | |
| PS-08 | | | | | | | | X | | | | | | | | |
| PS-09 | | | | | | | | | X | | | | | | | |
| PS-10 | | | | | | | | | | X | | | | | | |
| PS-11 | | | | | | | | | | X | | | | | | |
| PS-12 | | | | | | | | | | | X | | | | | |
| PS-13 | | | | | | | | | | | | X | | | | |
| PS-14 | | | | | | | | | | | | | X | X | | |
| PS-15 | | | | | | | | | | | | | | | X | |
| PS-16 | | | | | | | | | | | | | | | | X |

Tabla 58: Matriz de trazabilidad. Pruebas del sistema- Requisitos funcionales.

Como puede observarse en la tabla anterior, todas las pruebas del sistema pasadas cubren, al menos, un requisito funcional. También cabe destacar que todas las pruebas se han pasado satisfactoriamente con el debido resultado esperado.

# 6 Gestión del Proyecto.

En este apartado, se especifica la planificación seguida para el desarrollo de este trabajo de fin de grado. A partir de esta planificación, se establece un presupuesto para el proyecto.

## 6.1 Planificación

La realización de este proyecto comenzó el 16 de enero de 2015 y finalizo el 27 de septiembre de 2014; por lo tanto la duración de este proyecto ha sido de 8 meses aproximadamente. Este proyecto se ha compatibilizado tanto con el curso académico como con trabajo a jornada completa fuera de la universidad, por lo tanto la dedicación de horas ha sido diferente según la etapa. A continuación, se describen cada una de las fases realizadas:

- **Fase 1**: En esta primera iteración se estudió la viabilidad de este sistema, para ello fue necesario recopilar información y aprender sobre la inteligencia artificial, ya que la especialidad de la cual provengo (Ingeniería de Computadores) no realiza tanto hincapié como la especialidad de Computación, sobre la inteligencia artificial. Como el proyecto parte de unos ficheros generados por otro alumno en su trabajo de fin de grado, la tarea de aprendizaje es un punto importante para entender con precisión el trabajo del otro alumno.

- **Fase 2**: Una vez asimilados los conceptos pertenecientes al mundo de las narrativas y al lenguaje de generación de planes PDDL, se pasó a disponer el alcance del proyecto y los objetos a cumplir. Para fijar este alcance y objetivos, fue necesario establecer una serie de tareas para fragmentar el trabajo y que se pudiese llevar a cabo con mayor facilidad.

- **Fase 3**: Una vez definidos los objetivos, se obtuvieron los primeros casos de uso y requisitos mínimos que debería tener el proyecto para cumplir estos objetivos. En esta etapa surgieron dudas tales como el lenguaje en el cual se desarrollaría el motor. En un principio se iba a optar por desarrollarlo en C++ pero, prácticamente me habría tocado aprender ese lenguaje desde 0 mientras que Java lo dominaba bastante bien. Si a esto le sumamos el trabajo fuera de la universidad junto con las clases y exámenes propios de asignaturas pendientes, obtenemos la conclusión de que el desarrollo en C++ iba a llevar tres o cuatro veces más tiempo que el desarrollo en Java, por tanto se decidió desarrollar en Java.

- **Fase 4**: Una vez decidido el lenguaje en que se iba a desarrollar el motor, se comenzaron a traducir los ficheros PDDL de inicio con el fin de que se pudiesen usar de alguna manera en

Java. En este punto entró en juego un *parseador* que convertía los ficheros PDDL a formato XML. El resto de la fase consistió en entender y lograr hacer que el parseador funcionase en Java y crease los ficheros con estructura XML.

- **Fase 5:** En esta fase ya se tenían los XML por tanto se pudo usar el plugin de Java **JAXB** para generar la estructura de clases y objetos final que seguiría el proyecto.

- **Fase 6**: En este punto, se había avanzado en la programación y se tenía estructurado el entorno, pero es aquí cuando surgen una serie de ideas y modificaciones que harán del sistema algo mucho más visual y accesible. En un principio los objetivos establecidos pasaban por la recreación del comportamiento interno del motor. Lo que se ideó entonces fue, la creación de una interfaz gráfica que sirviese de puerta entre el usuario y el motor, a través de la cual se pudieran realizar distintas funcionalidades tales como la realización de peticiones, la visualización de las mismas, etc.

- **Fase 7:** Esta fue una de las fases más extensas. Durante esta fase, se creó la interfaz gráfica utilizando las librerías de Swing, hechas también en Java.

- **Fase 8:** Esta fase correspondió a la creación de métodos que comprobasen cada una de las acciones permitidas y de los predicados contenidos en el dominio de la narrativa.

- **Fase 9:** Llegado este punto, se optó por añadir una pequeña base de datos ligera localmente al sistema para guardar datos de peticiones. Como se necesitaba algo multiplataforma y ligero se decidió poner la base de datos de MongoDB.

- **Fase 10:** Se realizaron pruebas para cada uno de los componentes. También se comenzó con la redacción de la memoria según mucha documentación generada durante el desarrollo.

- **Fase 11:** Esta fase, casi en su totalidad, fue dedicada a la redacción de la memoria. Se realizaron borradores y se fue componiendo el documento final.

En la siguiente tabla se muestra de manera visual cómo se fueron distribuyendo las tareas a lo largo de las fases:

| Fase | Fecha | Distribución de tareas | Horas | Total de horas |
|---|---|---|---|---|
| 1 | 16/01/15<br>10/02/15 | Selección del trabajo de fin de grado.<br>Toma de contacto con el trabajo de fin de grado.<br>Análisis inicial de la solución inicial.<br>Aprendizaje inicial PDDL.<br>Aprendizaje narrativas.<br>Análisis del entorno. | 6<br>8<br>3<br>5<br>5<br>8 | 31 |
| 2 | 12/02/15<br>21/02/15 | Alcance del proyecto.<br>Objetivos del proyecto.<br>Fragmentación de tareas iniciales. | 3<br>3<br>4 | 10 |
| 3 | 23/02/15<br>04/03/15 | Análisis de Requisitos iniciales.<br>Análisis de Casos de Uso.<br>Plataforma de desarrollo. | 5<br>5<br>3 | 13 |
| 4 | 07/03/15<br>31/03/15 | Aprendizaje de uso parseador PDDL2XML.<br>Aprendizaje con ficheros PDDL. | 30<br>6 | 36 |
| 5 | 01/04/15<br>12/04/15 | Aprendizaje de uso y manejo de ficheros XML. | 4 | 18 |

| | | | | |
|---|---|---|---|---|
| | | Aprendizaje y uso de plugin JAXB. | 12 | |
| 6 | 15/04/15<br>22/04/15 | Nuevas ideas y modificaciones.<br>Análisis de la interfaz gráfica.<br>Análisis de requisitos de la interfaz. | 6<br>8<br>8 | 22 |
| 7 | 30/04/15<br>17/07/15 | Implementación y desarrollo de la interfaz.<br>Aprendizaje de la librería Swing. | 60<br>15 | 75 |
| 8 | 19/07/15<br>01/08/15 | Generación de métodos de comprobación de acciones y predicados. | 35 | 35 |
| 9 | 03/08/15<br>11/08/15 | Creación de MongoDB.<br>Integración de MongoDB.<br>Pruebas MongoDB. | 6<br>10<br>3 | 19 |
| 10 | 16/08/15<br>26/08/15 | Pruebas en distintos componentes.<br>Redacción de memoria | 4<br>12 | 16 |
| 11 | 16/08/15<br>27/09/15 | Redacción y revisión de la memoria. | 45 | 45 |
| **Total** | | | | **320 horas** |

Tabla 59: Distribución de horas.

El contenido de un trabajo de fin de grado debe estar ajustado para una dedicación de 300 horas aproximadamente (haciendo el cálculo de 1 crédito → 25 horas). En este trabajo se han dedicado 320 horas por lo que se ha producido una desviación de 20 horas. Esto es debido a las dificultades encontradas en un principio por la necesidad de aprender información relevante para el proyecto, y a la serie de modificaciones surgidas al inicio del desarrollo. También se consumió mucho tiempo en un inicio en comprender cómo funcionaba correctamente el parseador *PDDL2XML*.

## 6.2 Presupuesto.

En este apartado se ha realizado el cálculo del coste total del proyecto desarrollado, especificando los costes reales de personal y materiales utilizados durante el desarrollo.

### 6.2.1 Costes de personal

En esta sección se detallan los costes del personal que ha participado en este proyecto. En la siguiente tabla se indica el nombre de la persona que participa, el cargo que ha desempeñado, su coste por hora, las horas totales de dedicación en este proyecto y, por último, el coste total que ha generado esta persona para el proyecto.

| Nombre | Cargo desempeñado | Coste / hora | Total de horas | Coste total |
|---|---|---|---|---|
| Alejandro Robles Domínguez | Programador Junior | 18 €/hora | 320 | 5760 € |
| Daniel Borrajo Millán | Catedrático de la Universidad | 45€/hora | 25 | 1125 € |
| **Total** | | | | **6885 €** |

Tabla 60: Costes de personal.

Como puede observarse en la tabla anterior, el coste de personal asciende a 6885€.

### 6.2.2 Costes de materiales

En esta sección se calculan los costes totales de los materiales utilizados durante la realización del proyecto, teniendo en cuenta tanto software como hardware. Para obtener el coste total de cada uno de los materiales utilizados dentro del proyecto se han tenido en cuenta los meses de uso, el periodo de amortización, el precio, el porcentaje de dedicación al proyecto y el coste que ha generado para el proyecto.

| Material | Meses de uso | Amortización (meses) | Precio (€) | Dedicación al proyecto (%) | Coste generado al proyecto |
|---|---|---|---|---|---|
| **Sony VAIO i5** | 8 | 24 | 780€ | 70% | 182€ |
| **HP Pavillion Core 2 Duo** | 3 | 20 | 450€ | 30% | 20.25€ |
| **Microsoft Office Professional** | 8 | 24 | 120€ | 100% | 40€ |
| **Certificado de desarrollador de MongoDB en Java** | 2 | 6 | 65€ | 25% | 5.42€ |
| **Total** | | | | | 247.67€ |

Tabla 61: Costes de materiales.

El coste total de los materiales para este proyecto asciende a 247.67€.

### 6.2.3 Coste total del proyecto

Por último, para calcular el coste total al que asciende el proyecto, es necesario realizar la suma de los costes de personal y los costes materiales (incluyendo un 15% de costes indirectos del proyecto como son el agua, luz, gas, etc.):

| Componentes | Coste Total |
|---|---|
| **Costes de personal** | 6885€ |
| **Costes materiales** | 247.67€ |
| **Costes indirectos** | 1258.70€ |
| **Coste total del proyecto** | 8391.37€ |

Tabla 62: Costes total del proyecto.

# 7 Conclusiones y líneas futuras

Para finalizar con este proyecto, es interesante destacar las conclusiones extraídas durante la realización del mismo, así como una serie de posibles líneas futuras abiertas a nuevas modificaciones y cambios.

## 7.1 Conclusiones

Tal y como se ha comentado anteriormente, este trabajo se inició con la idea de desarrollar un motor desde cero que fuese capaz de realizar los cálculos y operaciones pertinentes de manera autónoma y eficiente. Sin embargo, la decisión final difiere en muchos aspectos de lo deseado inicialmente.

Gracias al análisis realizado antes y durante las primeras semanas de desarrollo, se pudo idea y diseñar un sistema cuyo resultado superaba con creces a la solución inicial planteada. Este hecho, me ha permitido darme cuenta de la gran importancia que tiene realizar un buen análisis antes de comenzar a diseñar y desarrollar una aplicación, así como tener una lista finita de requisitos prácticamente cerrada, no expuesta a posibles modificaciones constantes.

Uno de los primeros inconvenientes encontrados en el inicio del proyecto fue la incompatibilidad de los ficheros PDDL con las estructuras de clases en las que se apoya Java. Además de existir muy poca documentación relativa a la gestión o conversión de ficheros en lenguaje PDDL a otro tipo de lenguaje o estructura.

Otro de los principales inconvenientes iniciales fue la incapacidad para crear un motor en su totalidad y limitarse únicamente a la parte interna, despreciando la parte visual. La realización de un escenario con sus texturas y renderizado requerían mucho tiempo.

El evolutivo realizado en el propio proyecto, correspondiente a la interfaz gráfica sencilla, ahorrará mucho tiempo en las pruebas y mostrará de una manera más clara y concisa los datos del sistema.

Aunque han sido muchos los problemas que han surgido durante el desarrollo del trabajo, el problema que más tiempo retrasó el proyecto fue encontrar la manera de un usuario realizase peticiones suplantando a varios clientes y luego pudiera ver los resultados obtenidos.

No obstante, la experiencia adquirida en las diferentes asignaturas cursadas en la universidad y, especialmente, la realización de prácticas en empresa y posterior contrato de trabajo, me han servido para ir resolviendo todos los problemas que han surgido durante la realización de este proyecto.

Para finalizar, creo que el haber analizado y desarrollado todos los componentes del sistema, y no sólo haber limitado este proyecto al motor interno, me ha proporcionado nuevos conocimientos y habilidades que me serán de gran utilidad en mi futuro profesional, como graduado en ingeniería informática.

## 7.2 Líneas para trabajo futuros

Puesto que ha sido imposible realizar un proyecto cuyo alcance cubriera todos y cada uno de los componentes de un motor de un juego de aventuras actual, el abanico de opciones que esto supone es enorme. Además, no sólo podrían realizarse evolutivos de nuevos componentes para la aplicación, la interfaz en Swing está abierta a multitud de nuevas funcionalidades que pueden ser desarrolladas. A continuación se exponen algunas líneas futuras hacia las que puede evolucionar este sistema:

- **Extensión de la base de datos**. Actualmente la base de datos es una base de datos ligera almacenada localmente en la máquina. Una posibilidad sería adaptar el sistema a una base de datos relacional alojada en algún servidor en internet. La nueva base de datos proporcionaría mayor seguridad frente a posibles pérdidas y las peticiones no desaparecerían con nuevas partidas.
- **Mejora de la interfaz gráfica**. Actualmente la librería Swing está abierta a un montón de nuevas funcionalidades, así como a la adicción de nuevas librerías o plugin que enriquezcan a la propia interfaz actual.
- **Generación de la parte visual**. Actualmente se tienen las comprobaciones internas que debe realizar el motor para nuestro escenario particular. Uno de los posibles grandes evolutivos reside en la creación de texturas que representen a todas las entidades presentes en el escenario. Este puede ser un evolutivo demasiado grande para un único proyecto, por lo que se podría dividir entre la generación del entorno y la generación de las entidades.
- **Migración del proyecto a otro lenguaje de programación**. El motor está diseñado y desarrollado para funcionar con Java, pero se podría migrar a cualquier otro lenguaje de programación (como por ejemplo C ++).
- **Aplicación del motor para dispositivos móviles**. Otro posible evolutivo podría residir en realizar aplicaciones para los distintos Sistemas Operativos que existen en dispositivos móviles. Los más importantes serían Android e IOS.

- **Creación de aplicación web**. Se podría crear una aplicación web que estuviera alojada en un servidor con disponibilidad completa. De esta manera los usuarios se conectarían a través de la web de la aplicación.
- **Mejorar la seguridad de todo el sistema**. En este proyecto se ha puesto especial hincapié en el desarrollo y comprobaciones internas hechas por el motor, no obstante, en la actualidad uno de los pilares más importantes en la creación de un videojuego es la seguridad que brinda tanto a los usuarios como interna frente a posibles "*hackers*".

# 8 Bibliografía

[1] Wikipedia. Metodologías ágiles de desarrollo. Disponible en:

https://es.wikipedia.org/wiki/Desarrollo_%C3%A1gil_de_software

[2] Control de versiones en BitBucket. Disponible en:

http://www.sospedia.net/bitbucket/

[3] Tendencias actuales del desarrollo de los videojuegos. Disponible en:

https://mariusagm.wordpress.com/2013/02/13/tendencias-actuales-del-desarrollo-de-videojuegos/

[4] El País. Artículo sobre la IA. Publicado el 31/07/2015. Disponible en:

http://elpais.com/elpais/2015/07/28/icon/1438099629_170249.html

[5] Wikipedia. PDDL. Disponible en:

https://en.wikipedia.org/wiki/Planning_Domain_Definition_Language

[6] Wikipedia. STRIPS. Disponible en:

ttps://en.wikipedia.org/wiki/STRIPS

[7] Wikipedia. ADL. Disponible en:

https://en.wikipedia.org/wiki/Action_description_language

[8] Wikipedia. XML. Disponible en:

https://es.wikipedia.org/wiki/Extensible_Markup_Language

[9] Wikipedia. Lenguaje de programación Java. Disponible en:

https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)

[10] Wikipedia. IDE Eclipse. Disponible en:

http://www.genbetadev.com/herramientas/eclipse-ide

[11] Wikipedia. Programación orientada a objetos. Disponible en:

https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos

[12] Wikipedia. Lenguaje de programación C. Disponible en:

https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n)ç

[13] Wikipedia. Lenguaje de programación C++. Disponible en:

https://es.wikipedia.org/wiki/C%2B%2B

[14] Artículo sobre JDK 1.0. Disponible en:

http://ordenador.wingwit.com/software/software-development-companies/167421.html

[15] Wikipedia. Biblioteca gráfica Swing. Disponible en:

https://es.wikipedia.org/wiki/Swing_(biblioteca_gr%C3%A1fica)

[16] Wikipedia. Base de datos MongoDB. Disponible en:

https://es.wikipedia.org/wiki/MongoDB

[17] Web de JSON. Disponible en:

https://es.wikipedia.org/wiki/MongoDB

[18] Wikipedia. Plugin JAXB. Disponible en:

https://es.wikipedia.org/wiki/JAXB

[19] Instalación de MongoDB en el sistema operativo ***Mac OS.*** Disponible en:

http://docs.mongodb.org/manual/tutorial/install-mongodb-on-os-x/

[20] Acceso a las variables de entorno del sistema en Windows 7. Disponible en:

http://www.edu4java.com/es/conceptos/variables-de-entorno-windows-ms-dos.html

# Anexo I. Instalación de MongoDB en Windows.

En el siguiente anexo se describirá en líneas generales cómo realizar la instalación del driver de MongoDB en Windows. Los pasos a seguir son:

1.- Descargar el MongoDB:

URL:https://www.mongodb.org/downloads

Versión: Windows 64-bit legacy

Descargar MSI

2.- Instalamos. Recordar la ruta donde se instala.

(Por defecto: "*C:\Program Files\MongoDB\Server\3.0\bin*")

**MUY IMPORTANTE**: crear una carpeta llamada "data" colgando del directorio raíz *C* y dentro de *"data"* otra que se llamada "*db"* (quedando así "*C:\data\db*").

3.- Poner en las variables de entorno del sistema [20] la ruta del punto 2 ("*C:\data\db*").

Editamos la variable ***Path*** ponemos al final: *";C:\Program Files\MongoDB\Server\3.0\bin;"*

4.- Reiniciamos para que el Sistema Operativo coja los cambios (Dependiendo del pc las actualizaciones sobre las variables de entorno las coge en el momento o no)

5.- Tras reiniciar ya tenemos la variable de entorno bien cargada.

Tenemos que arrancar el servicio de MongoDB:

5.1.- Esto se puede automatizar para que Windows lo arranque por defecto (y no estar una y otra vez cada que arrancas el pc haciéndolo a mano)

```
md "C:\Program Files\MongoDB\Server\3.0\log"

echo logpath=C:\Program
Files\MongoDB\Server\3.0\log\mongo.log > "C:\Program
Files\MongoDB\Server\3.0\mongod.cfg"

"C:\Program Files\MongoDB\Server\3.0\bin\mongod.exe" -v --config "C:\Program
Files\MongoDB\Server\3.0\mongod.cfg" --install
```

5.2.- Otra opción es arrancar el servicio mongod:

5.2.1.- En un cmd escribimos mongod.

5.2.2.- Otra opción es ejecutar el .exe que está en la ruta por defecto

*"C:\Program Files\MongoDB\Server\3.0\bin\mongod.exe"*

**NOTA: NO CERRAR LA CONSOLA PORQUE BAJARÍAMOS EL SERVICIO DE MONGO.**

6.- Una vez que el servicio de Mongo está arrancado, ya se puede ir a un nuevo terminal (cmd) y escribir *"mongo",* con lo que estaríamos dentro de la base de datos de mongo.

**NOTA: PARA INSTALAR EL DRIVER EN MAC OS, REALIZAR ESTA GUÍA [19].**