SIMULACIÓN DE MISIONES DE RESCATE CON ROBOTS

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

CURSO: 2015-2016

Luis García Terriza

Sergio Moreno de Pradas

Director: Juan Pavón Mestras

Codirector: Francisco J. Garijo

Departamento de Ingeniería del Software e Inteligencia Artificial

Facultad de Informática

Universidad de Complutense de Madrid

En primer lugar, queremos agradecer a nuestro tutor ***Juan Pavón Mestras*** *el ofrecernos la posibilidad de trabajar en este proyecto y por su apoyo durante todo el año.*

A nuestro cotutor, ***Francisco J. Garijo*** *por su paciencia y dedicación constante a nuestras dudas.*

A todas las personas que hayamos conocido en la carrera que de una forma u otra nos han ayudado y sobre todo a nuestra familia por apoyarnos día a día.

A todos, ***muchas gracias.***

ÍNDICE

1. INTRODUCCIÓN 5

1.1 Antecedentes 5

1.2 Resumen 5

1.3 Summary 6

1.4 Propósito general y definición de objetivos 7

1.5 Bases tecnológicas 7

1.6 Metodología 8

1.7 Palabras clave (Key words) 9

2. DESARROLLO 10

2.1 Requisitos 10

2.2 Riesgos 17

2.3 Diseño de organización del sistema 18

2.4 Modelo de información 27

2.5 Diseño de los objetivos 32

2.5.1 DefinirMiEquipo 32

2.5.2 AyudarVictima 32

2.5.3 ReconocerTerreno 34

2.5.4 DecidirQuienVa 35

2.5.5 TerminarSimulacion 35

2.6 Diseño de la interfaz 36

2.7 Diseño del algoritmo de rutas 39

2.8 Arquitectura 41

3. IMPLEMENTACIÓN 45

3.1 Estructura de paquetes. Implementación de agentes y recursos 45

3.1.1 Agentes 45

3.1.2 Recursos 49

3.2 Uso del sistema 52

3.2.1 Guía de instalación 52

3.2.2 Guía de uso 52

3.3 Pautas de codificación 58

3.4 Organización del código 58

3.5 Interfaz 59

3.6 Contribución personal al proyecto 62

3.6.1 Luis García Terriza 62

3.6.2 Sergio Moreno de Pradas 64

3.7 Problemas encontrados y solución adoptada 66
4. EXPERIMENTACIÓN 69
4.1 Escenarios a usar 69
4.2 Tabla de pruebas 71
4.3 Datos a obtener 72
4.4 Métricas de resultados 73
5. RESULTADOS Y CONCLUSIONES 75
5.1 Análisis de resultados 75
5.1.1 Escenarios tipo Llanura 75
5.1.2 Escenarios tipo Casa 77
5.1.3 Escenarios tipo Estadio 81
5.2 Conclusiones 87
5.3 Métricas del proyecto 91
5.4 Versiones futuras 92
6. BIBLIOGRAFÍA 95
7. ANEXOS 96
7.1 Anexo I: Casos de uso 96
8. AUTORIZACIÓN DE DIFUSIÓN 99

1. INTRODUCCIÓN

1.1 ANTECEDENTES

El proyecto surge con la idea de desarrollar un sistema donde un equipo de robots pueda organizarse para realizar misiones de rescate de posibles víctimas en un escenario donde puedan ocurrir imprevistos y el equipo de robots tenga que responder y adaptar el proceso colectivo de realización de la misión en tiempo real.

El objetivo del sistema es estudiar la relación entre las características del escenario donde se realizará la simulación del rescate, la organización del equipo que realiza la misión, la comunicación y coordinación entre sus miembros de forma que puedan decidir en tiempo de ejecución cómo reaccionar ante distintos eventos de la manera más óptima. El comportamiento de los agentes se ha modelado usando objetivos (por ejemplo, ayudar a una víctima o el de reconocer el terreno) que se consiguen realizando una serie de tareas, el proceso de consecución de los objetivos de un agente/robot se explica con más detalle en secciones posteriores.

En esencia lo que se quería conseguir es una plataforma que sirviera para simular misiones de rescate usando equipos de robots y que pudiera utilizarse para realizar estudios sobre cómo actúa cada organización de equipo en las misiones.

1.2 RESUMEN

Este proyecto consiste en el desarrollo de un sistema para simular misiones de rescate usando equipos de robots donde cada robot tiene sus propios objetivos y debe coordinarse con el resto de sus compañeros para realizar con existo la misión de rescate en escenarios dinámicos.

El escenario se caracteriza por contener:

- Agentes Robot: son las entidades del sistema encargado de tareas relacionadas con el rescate, como por ejemplo, explorar el terreno o rescatar a una víctima. Se organizan de forma jerárquica, esto es, hay un jefe encargado de asignar tareas a los demás robots, que serán subordinados.
- Víctimas: son los objetivos a rescatar en la misión. Tienen una identificación, una localización y una esperanza de vida.
- Obstáculos: delimitan una zona por la que el robot no puede pasar. Simulan la existencia de paredes, rocas, árboles…, es decir, cualquier tipo de estructura existente en un escenario real.
- Zona segura: marca un punto del mapa adonde los robots moverán a las víctimas en el rescate. Representa lo que en un rescate real sería un campamento u hospital.

El sistema permite:

- Crear y gestionar escenarios de simulación

- Definir equipos de robots con diferentes miembros, diferentes objetivos y comportamientos.
- Definir modelos organizativos en los equipos y estrategias de coordinación.
- Realizar los objetivos individuales y de grupo para salvar a las víctimas llevándolas al sitio seguro esquivando los obstáculos.
- Realizar experimentos de simulación: probar distintas configuraciones de equipo con un número variable de robots, varias víctimas en lugares diferentes y escenarios independientes.

Se ha partido del proyecto ROSACE(Robots et Systèmes AutoCommunicants Embarqués / *Robots y sistemas embebidos autocomunicantes*), que está construido sobre la herramienta ICARO, que es una Infraestructura Ligera de Componentes Software Java basada en Agentes y Recursos y Organizaciones para el desarrollo de aplicaciones distribuidas. El punto de partida ya implementaba una versión preliminar del proyecto capaz de organizar objetivos entre los robots y que consigan ir a la localización objetivo. El presente proyecto utiliza el patrón arquitectónico de ROSACE y parte de su infraestructura pero desarrolla un sistema original con nuevas herramientas para definir y gestionar escenarios, disponer de un modelo más realista del comportamiento de los robots y controlar el proceso de simulación para incluir posibles fallos de los robots y para el estudio individual y colectivo de los miembros de los equipos.

1.3 SUMMARY

This project consists on developing a MultiAgent System simulator to rescue victims in dynamic environments. Agents are goal oriented autonomous robots organized in teams.

The scenario is made up of the following:

- Robots: they are the entities of the system in charge of tasks related to the rescue, like for example, exploring the terrain or rescuing a victim. They are organized hierarchically, that means that there is a boss in charge of assign tasks to the rest of the robots, which are subordinates.
- Victims: they are the objectives to rescue in the mission. They have an id, a location and a life expectancy.
- Obstacles: they delimit a zone where robots cannot go through. They simulate the existence of walls, rocks, trees…, that is to say, any kind of existing structure in a real scenario (real life).
- Safe place: it is a mark of the map where the robots will move the victims to in the rescue. It represents what in a real rescue would be a camp or a temporary hospital.

The simulator provides the

- Creation & management of scenarios
- Definition of robot teams with different members, objectives and behaviours.

- Organization models and goal resolution strategies: role in the team, individual goals, goal resolution process.
- Simulation experiments by selecting a scenario and a team to achieve a mission consisting on rescuing the victims in the scenario. Metrics considered for evaluation are:
    - Simulation time
    - Rescue victims mean
    - Percentage of rescued victims

The project is based on the architecture of the ROSACE project incorporating a significant set of new features regarding the creation and management of scenarios, agent models, team organization and simulation.

1.4 PROPÓSITO GENERAL Y DEFINICIÓN DE OBJETIVOS

El principal objetivo de este proyecto es la creación de un Sistema Multi-Agente donde se pueda simular el salvamento de unas víctimas en un escenario definido por un usuario, en el escenario pueden ocurrir imprevistos como que un robot se estropee o una víctima muera y el sistema debe saber reconocer estos eventos y adaptarse a ellos para conseguir realizar la misión.

El proyecto pretende proporcionar una herramienta para investigar la forma en que la organización de un equipo, que trabaja en una misión de rescate de víctimas en diferentes entornos, influye en la eficiencia del equipo. Lo que se pretende es obtener una organización óptima según el número de víctimas y las características del entorno.

1.5 BASES TECNOLÓGICAS

Se ha concebido el sistema como un SMA (Sistema Multi-Agente) por considerarlo más realista y por disponer de una infraestructura ICARO para implementar los agentes, el entorno y otros elementos del sistema como componentes. Proporciona una infraestructura sólida, fácilmente integrable y configurable sobre la que construir un Sistema Multi-Agente como el que se ha desarrollado en este proyecto. Una de las razones principales es que ofrece un modelo de componentes, con interfaces bien definidas, y patrones arquitecturales para dos tipos de agentes (dirigido por objetivos y reactivo) y para integrar recursos en el sistema.

Se han diseñado diversos componentes de la aplicación, como Recursos y Agentes. Los agentes de la aplicación son robots con distinto rol en el equipo de robots, dotado cada uno de ellos con una cierta inteligencia artificial para decidir y moverse, mientras que los Recursos son componentes importantes que aportan funcionalidades al proyecto.

Aparte de ICARO se han usado otras tecnologías para facilitar el desarrollo de la aplicación.

- **Lenguaje de programación Java**: La principal razón de utilizar Java es que es un lenguaje de fácil comprensión y desarrollo sobre el que ya se tenía conocimientos medianamente avanzados.

- **Interfaz gráfica**: SWT. Es una librería de Java muy sencilla para crear interfaces de usuario completas e intuitivas.

- **Sistema de Reglas**: Drools 5.5. Es un sistema de reglas moderno que sustituye la funcionalidad de un autómata. La razón por la que elegimos un sistema de reglas y en concreto Drools es porque, una vez obtienes una interfaz de uso sobre él motor, es muy fácil de programar las reglas y de manejar los hechos.

- **Control de versiones**: Github. Es una Web de almacenamiento de código que ofrece almacenamiento del código y ventajas para control de versiones, entre otras funcionalidades. La razón por la que escogimos Github es porque es una herramienta muy utilizada en el mundo del software.

- **Diseño UML**: Enterprise Architect. Es una herramienta software de modelado UML que, a diferencia de otras, funciona eficientemente y tiene ingeniería inversa de código.

1.6 METODOLOGÍA

El primer mes se dedicó a conocer todo el proyecto en profundidad (estructura de clases, funcionamiento, etc…). Una vez terminada esta fase, se plantearon una serie de objetivos más simples a corto plazo para continuar con el aprendizaje sobre el proyecto. Algunos ejemplos de estos objetivos eran la resolución de errores ya existentes o la introducción de obstáculos.

Una vez se consiguieron los objetivos a corto plazo se fueron planteando nuevas funcionalidades o mejoras en el actual sistema para más adelante. Algunos ejemplos fueron la creación de nuevos algoritmos de ruta, mejora en la "inteligencia" de los agentes(robots), implementación de funcionalidades para una simulación más realista…

En nuestro caso, se realizó el trabajo juntos siempre que se podía, repartiéndose las tareas de forma puntual. Para conseguir una buena dinámica de trabajo y para obtener resultados de forma óptima, se asignaron unas horas de dedicación exclusiva al proyecto a diario, con lo que se asegura la consecución de los objetivos.

Cada vez que se completaba una nueva funcionalidad, se realizaban varias ejecuciones de prueba para comprobar que el resultado obtenido es el deseado y que la nueva funcionalidad no entrara en conflicto con alguna de las funcionalidades ya implantadas en el proyecto.

El tipo de metodología que se usa en el desarrollo del proyecto fue la **metodología ágil**, teniendo reuniones con los tutores de forma semanal mostrando los objetivos que se han conseguido durante la semana y comentando las posibles dudas que surgían para buscar una

solución entre todos. Además, al final de cada reunión se marcaban unos objetivos para la siguiente reunión. De esta forma, conseguimos que el desarrollo fuera más dinámico y siempre se tuviera algo que hacer en cada momento. En cada iteración exitosa (normalmente cada una o dos semanas) se obtenía una nueva funcionalidad o un nuevo avance sobre el sistema.

1.7 PALABRAS CLAVE (KEY WORDS)

Español	Inglés
Misión de rescate	Rescue mission
Robots	Robots
Multiagente	Multiagent
Cálculo de rutas	Route calculation
Inteligencia artificial	Artificial Inteligence
Victima	Victim
Salvamento	Saving
ICARO	ICARO
ROSACE	ROSACE
Organización de equipos	Team organization

Tabla 1: Palabras Clave

2. DESARROLLO

Para poder definir con exactitud la funcionalidad del proyecto y su alcance, se definieron los requisitos funcionales y no funcionales que debía cumplir la aplicación. A raíz de aquí todo lo que se diseñó a continuación se basó en estas directrices iniciales.

2.1 REQUISITOS

- Funcionales:

Por cada conjunto de funcionalidades del sistema se mostrará un diagrama de casos de uso junto con 2 tablas del caso de uso, quedando en el ***Anexo I – Casos de Uso*** el resto de las tablas del caso de uso.

- Inicio:
  - Iniciar sistema
  - Gestionar Escenarios
  - Gestionar Simulación
  - Finalizar sistema

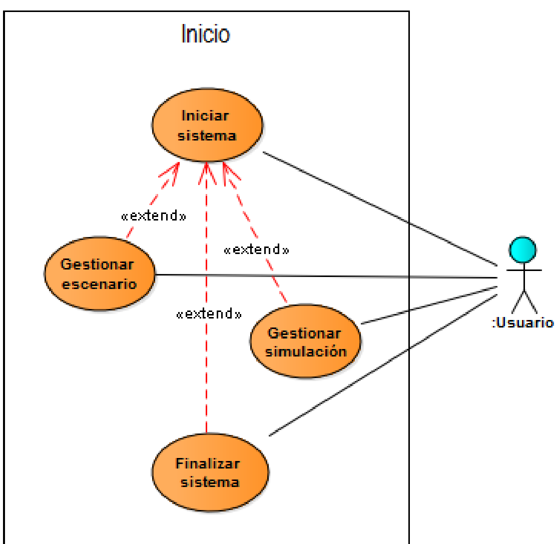

Figura 1: Diagrama de casos de uso:

CU–Inicio-01	Iniciar Sistema
Objetivo en contexto	Arrancar el sistema
Precondiciones	Tener descargado e instalado el proyecto.
Postcond. si éxito	Arranca el sistema
Postcond. si fallo	No inicia el sistema
Actores	Usuario
Secuencia normal	1 – Seleccionar el descriptor de organización 2 – Iniciar el script de arranque del sistema.

CU–Inicio-02	Gestionar Escenarios
Objetivo en contexto	Obtener las capacidades de gestionar escenarios.
Precondiciones	El sistema debe estar iniciado.
Postcond. si éxito	Carga la interfaz de editor de escenarios.
Actores	Usuario
Secuencia normal	1 – Elegir la vista de la interfaz de control. 2 – Se obtienen la capacidad de realizar los casos de uso de **gestionar escenarios**.

- Gestión de escenarios:
  - Creación escenario
  - Modificación escenario
  - Eliminación escenario
  - Abrir escenario

En la gestión de los escenarios solo se encuentra al agente de **Usuario** ya que es el único con capacidad de crear, modifica o borrar los escenarios además de poder abrirlos.

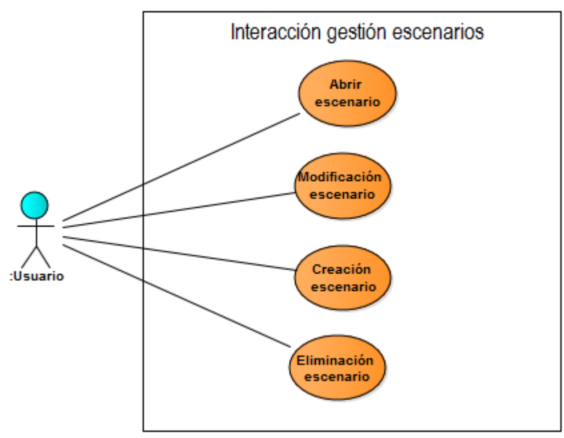

Figura 2: Diagrama de casos de uso: gestión de escenarios

CU–escenarios-01	Abrir escenario
Objetivo en contexto	Abre un escenario para usarlo en simulación.
Precondiciones	El fichero debe tener buen formato.
Postcond. si éxito	Se carga un escenario para usarlo en simulación.
Postcond. si fallo	Mensaje de error y vuelve al menú principal.
Actores	Usuario.
Secuencia normal	1 – Ir al menú de Editor de escenarios 2 – Seleccionar *Abrir escenario.* 3 – Buscar el escenario a cargar. 4 – Abrir el escenario.

CU–escenarios-02	Creación de un escenario
Objetivo en contexto	Crear un nuevo escenario.
Precondiciones	El sistema debe estar cargado correctamente.
Postcond. si éxito	Se crea un nuevo escenario para usarlo en simulación
Postcond. si fallo	Mensaje de error y vuelve al menú de edición.
Actores	Usuario.
Secuencia normal	1 – Ir al menú de Editor de escenarios 2 – Seleccionar *Crear escenario.* 3 – Crear el escenario. 4 – Almacenar el nuevo escenario

- Gestionar Simulación:
    - Usuario
        - Seleccionar escenario
        - Salvar a una víctima.
        - Salvar todas las víctimas
        - Parar robot.
        - Parar la simulación.

En la simulación el usuario puede interactuar con varias tareas diferentes, por ejemplo puede seleccionar el escenario que se va a utilizar en la simulación, puede elegir el tipo de salvamento (si va a salvar a una víctima o a varias) además de que puede parar la simulación en cualquier momento con el botón Terminar del recurso de trazas. Por último una vez inicia la simulación puede parar cualquier robot de ésta pulsando el botón del robot al que corresponda en la interfaz de misión

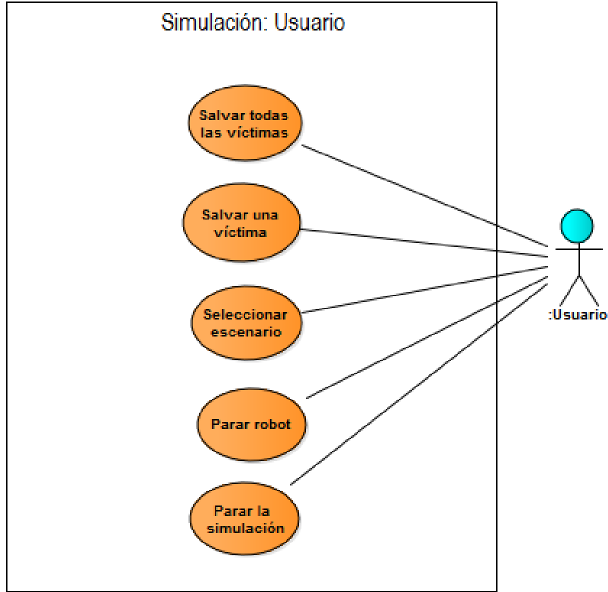

Figura 3: diagrama de caso de uso de simulación usuario.

CU-simulación–01	Salvar todas las víctimas
Objetivo en contexto	Salvar todas las víctimas de la simulación.
Precondiciones	El sistema debe estar cargado correctamente.
Postcond. si éxito	Envía el mensaje de comenzar a salvar a todas las víctimas.
Actores	Usuario. ArranqueScript.
Secuencia normal	1 – Seleccionar función de evaluación. 2 – Seleccionar *Salvar victimas del escenario.* 3 – Elegir el tiempo de sleep. 4 – Aceptar tiempo de sleep. 5 – Envío del mensaje.

CU–simulación-02	Salvar una víctima
Objetivo en contexto	Salvar a una víctima del escenario.
Precondiciones	El sistema debe estar cargado y funcionando.
Postcond. si éxito	Se salva a una víctima del escenario.
Actores	Usuario.
Secuencia normal	1 – Seleccionar la víctima que se quiere salvar del menú con doble clic. 2 – Pulsar en Salvar Victima

- Agentes
    - Iniciar simulación
    - Guardar resultados ejecución
    - Asignar objetivo
    - Reconocer el terreno
    - Parar robot
    - Salvar una víctima
    - Finalizar simulación

En la simulación no solo actúa el usuario, invisible a este están los agentes comunicándose entre ellos realizando las tareas necesarias para completar la misión de rescate. El agente controlador es el que se encarga de iniciar y finalizar la simulación, es decir, iniciar la misión cuando el usuario realiza el CU de salvar a una víctima / salvar a todas las víctimas y finalizar esta cuando el asignador le avisa de que han terminado. El asignador se encarga no solo de asignar los objetivos (ayudar una víctima y reconocer el terreno) sino también de almacenar los datos que se obtienen de la simulación. Por su parte el agente subordinado se encarga de las tareas que le vaya enviando el asignador, este puede reconocer el terreno o salvar a una víctima dada si se le encomienda esta tarea o parar un robot cuando este subordinado se queda sin energía.

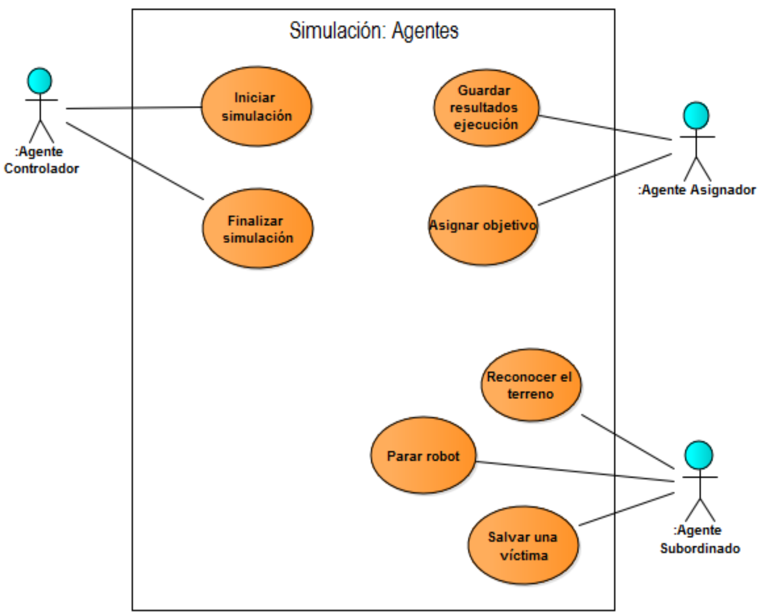

Figura 4: Diagrama de casos de uso de simulación usuario.

CU-agentes–01	Iniciar simulación
Objetivo en contexto	Inicia la simulación de la misión.
Precondiciones	El usuario debe haber iniciado la simulación.
Postcond. si éxito	Comienza la simulación de la misión.
Actores	AgenteControlador
Secuencia normal	1 – Obtiene los datos de configuración de simulación. 2 – Empieza la simulación.

CU–agentes-02	Asignar objetivo
Objetivo en contexto	Asigna un objetivo a un subordinado.
Precondiciones	La simulación debe estar iniciada y el asignador tiene que tener un objetivo para asignar.
Postcond. si éxito	Se asigna un objetivo a un subordinado para que lo realice.
Actores	AgenteAsignador
Secuencia normal	1 – Realiza una petición del objetivo. 2 – Se obtienen las evaluaciones de los subordinados. 3 – El asignador selecciona al subordinado óptimo. 4 – El asignador envía la propuesta al subordinado.

- Gestión de los resultados
    - Cargar datos de ejecución.
    - Creación de métricas.
    - Visualización de resultados.

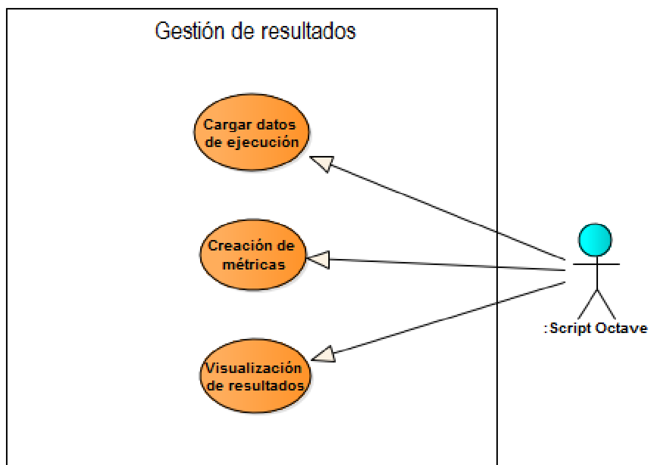

Figura 5: diagrama de casos de uso de gestión de los resultados.

CU–resultados-01	Cargar datos de ejecución
Objetivo en contexto	Obtiene los datos de la simulación.
Precondiciones	Los datos deben estar en el formato esperado.
Postcond. si éxito	Carga los datos de la ejecución.
Postcond. si fallo	Mensaje de error y el programa termina.
Actores	ScriptOctave
Secuencia normal	1 – Busca la carpeta del sistema donde se almacenan los datos. 2 – Carga todos los ficheros de las simulaciones.

CU–resultados-02	Creación de métricas
Objetivo en contexto	Crear las métricas de los datos obtenidos.
Precondiciones	Los datos deben haberse cargado correctamente.
Postcond. si éxito	Se crean las métricas con los datos.
Postcond. si fallo	Mensaje de error y el programa termina.
Actores	ScriptOctave
Secuencia normal	1 – Selecciona los escenarios concretos. 2 – Realiza cálculos con los datos. 3 – Crea las gráficas y las almacena.

En la gestión de los resultados se comprobó la necesidad de añadir un nuevo actor al sistema que sería el encargado de obtener los datos de las simulaciones y crear las métricas y mostrarlas, este actor es un **script programado en octave** que realiza dichas tareas

- No funcionales:
    - Sistema operativo: Compatibilidad con Windows.
    - Procesador: Mínimo: Intel i5 o equivalente. Recomendado: i7-4770K
    - Memoria RAM: Mínimo 2GB
    - Espacio en disco: Inferior a 150MB.
    - Ancho de banda: no necesario.
    - Máquina virtual Java Versión 7.
    - Confiabilidad: Se toleran fallos ocasionales si no hay peligro en la integridad de los datos.
    - Escalabilidad y flexibilidad: Debe ser perfectamente adaptable a posibles cambios futuros

2.2 RIESGOS

Identificación	Descripción	Probabilidad de riesgo	Consecuencias
1	El simulador falla cuando aumenta el número de los robots	5	2
2	Falta de estabilidad del motor de reglas	2	4
3	Versión incompatible de la máquina virtual Java	1	5
4	Que el usuario realice acciones inesperadas	3	2

Tabla 2: Riesgos.

LEYENDA

Probabilidad:

- 1: Muy improbable
- 2: Poco probable
- 3: Moderadamente probable
- 4: Probable
- 5: Muy probable

Consecuencias:

- 1: No supone ningún problema
- 2: Supone un pequeño problema
- 3: Supone un problema moderado
- 4: Supone un gran problema
- 5: Catastrófico

Una vez expuestos los posibles riesgos, se procede a explicar el plan de contingencia para cada uno:

Identificador	Plan de contingencia
1	El problema de este riesgo es que, con la cantidad de cómputo que se tiene que realizar por agente, cuanta más cantidad haya de robots más aumenta el cómputo. Además, llegado a cierto punto puede colgar el programa entero. El plan de contingencia fue la creación de modelos de organización con un número bajo de agentes.
2	Este problema ocurre cuando se ha diseñado mal las reglas del sistema. Para combatir este problema, antes de implementarlas, se realizó un minucioso diseño de éstas para el correcto funcionamiento.
3	El problema que subyace aquí es que las librerías utilizadas en el proyecto están diseñadas para unas ciertas versiones de java, y al ejecutar el proyecto con Java 8 o mayor, no funciona correctamente. Para mitigar este problema se ha informado de este requisito en la documentación del proyecto.
4	Este problema existe en todos los sistemas donde el usuario necesita realizar una interacción, puede pasar que el usuario no realice las cosas que se espera por el sistema y por consiguiente pueda colgarlo. Por esta razón se ha tenido un plan de contingencia donde se tiene una gestión de errores para todas esas interacciones que no se hayan implementado.

Tabla 3: Tabla de planes de contingencia.

2.3 DISEÑO DE ORGANIZACIÓN DEL SISTEMA

El diseño de organización del sistema se utiliza para explicar la estructura interna del sistema mostrando los componentes de este, cómo están conectados entre sí y las interfaces que se usan para acceder a cada uno. De esta forma en un único diagrama podemos estudiar la estructuración completa del sistema y analizar el funcionamiento del mismo.

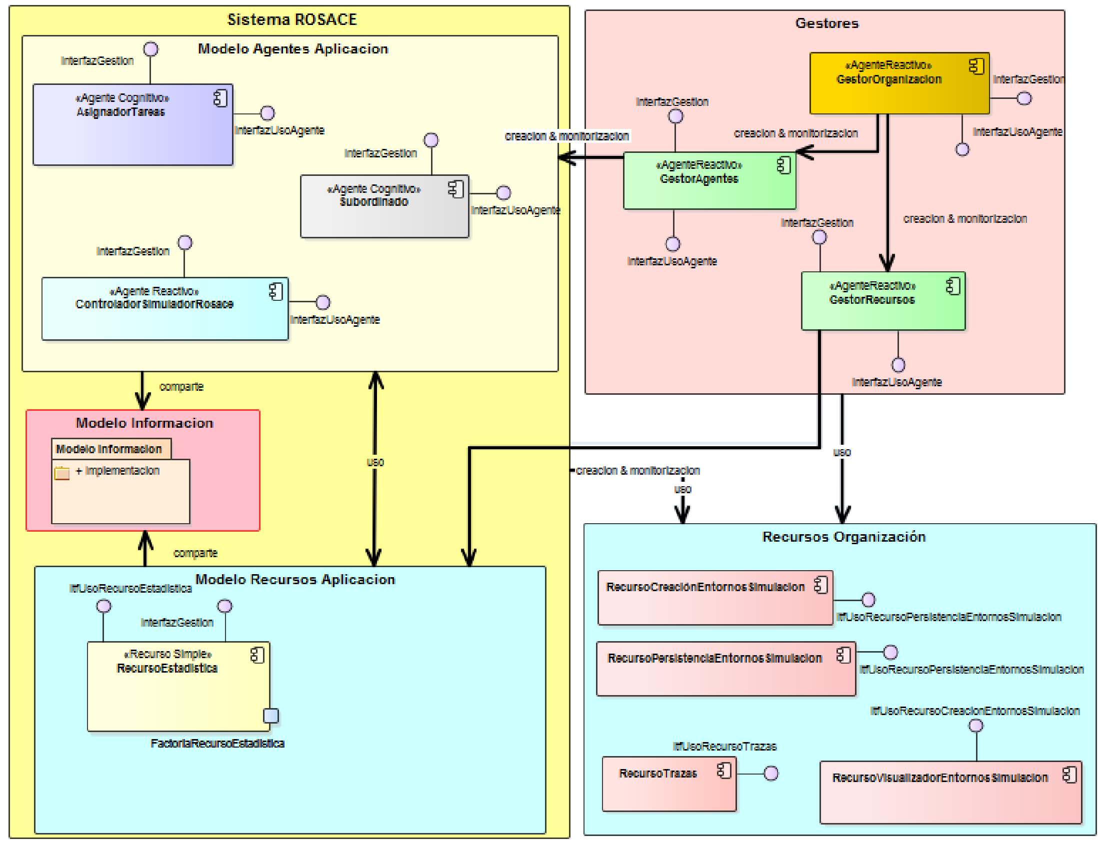

Figura 6: Diagrama de la organización del sistema.

El sistema está basado en tres **componentes:**

1. **Agentes**

   Al comienzo del proyecto el sistema tenía dos tipos de organización de equipos de agentes, el **igualitario** y el **jerárquico**; en el igualitario todos los agentes actuaban al mismo nivel, es decir, solo había un tipo de agente y era el denominado *Igualitario*; en el jerárquico por el contrario siempre hay un agente que tiene el rango de jefe y los demás el de subordinado teniendo claro que el jefe toma las decisiones y el subordinado las acata.

   Se centró en el jerárquico dejando de lado el igualitario ya que este primero es un modelo más parecido a la realidad y donde todo se podía parametrizar y organizar mejor donde un tipo de agente tuviera ciertas tareas y otro tipo otras, y se conozca en todo momento quien es el que realiza cada una.

   La razón de usar la organización jerárquica en vez de igualitaria es porque en la organización igualitaria hay una carga muy grande de envío de mensajes, lo que sumado a la carga de cómputo que se iba a introducir podía ralentizar demasiado el sistema y hacer muy difícil la ejecución.

   Otra razón es que el modelo jerárquico es una forma más organizada de modelo de equipo y permite una sencilla distribución de trabajos respecto al igualitario. Un

ejemplo es que en el igualitario, como cualquier robot puede realizar cualquier trabajo, hay que mantener una discusión previa para cualquier objetivo mientras que en el jerárquico esto se minimiza: cada robot tiene unos trabajos asignados y en algunos casos no es necesaria la comunicación entre ellos ya que el jerárquico asignador tiene todo centralizado.

De esta forma lo que conseguimos es un sistema con 3 tipos diferentes de agentes, primero un controlador que es propio del sistema, invisible para la simulación, el subordinado y el jerárquico ya explicados:

- Agente controlador

  Es el agente del sistema encargado de gestionar la simulación, indica tanto el inicio como el final de la misma. Además, durante la ejecución, es el encargado de pintar en la interfaz de misión tanto robots como víctimas además del estado de estas (se tiene un código de colores: No_descubierta Asignada Desasignada Muerta).

  Este agente se encarga también de parar un robot y desasignar las víctimas que este tenga o guardar los resultados de la simulación.

- Agente subordinado

  Este agente es al que el agente asignador enviará las tareas que tiene que hacer y controlará su ejecución. Puede desempeñar dos roles dependiendo de la función de evaluación que se elija:

  - Reconocedor: este rol es el encargado de comenzar a reconocer el terreno y avisar al asignador cada vez que encuentre una víctima para que este la asigne. Cuando este rol se complete, es decir, reconozca todo el terreno, podrá cambiar de rol para salvar alguna víctima.

  - Sanador: este rol lo tiene el robot al que se le haya encargado salvar una víctima por el asignador, el sanador encargado es el agente que tarda menos en salvar a la víctima desde su punto de origen.

- Agente asignador

  Este tipo de agente es el que se va a encargar de estudiar el estado de la simulación y actuar en consecuencia gestionando a los agentes subordinados. Por ejemplo, si le llegara un mensaje para que comience a explorar el terreno, el asignador tendría que gestionar este mensaje convirtiéndolo en un objetivo

a cumplir y asignarle este objetivo a algún subordinado, además debería estar gestionándolo a la espera de que este lo finalice.

2. **Recursos**

   En el sistema existen una serie de recursos. Inicialmente había muchos recursos, pero puesto que no se utilizaban se decidió eliminarlos. Un ejemplo de recurso eliminado es el recursoMorse. Actualmente, los recursos que hay y se utilizan son:

   - **Recurso Creación Entornos Simulación**: se encarga de la creación del entorno de simulación.
   - **Recurso Persistencia Entornos Simulación**: se encarga de la persistencia de escenarios.
   - **Recurso Visualizador Entornos Simulación**: se encarga de la parte visual del escenario y del centro de control.
   - **Recurso Estadística**: sirve para crear estadísticas relacionadas con la simulación.
   - **Recurso Trazas**: se utiliza para mostrar mensajes generados por los distintos componentes del sistema.

3. **Gestores**

   Finalmente, hay 3 gestores crean y gestionan los distintos componentes declarados en el descriptor de la organización:

   - **Gestor Organización**: se encarga de la creación y configuración de la organización y del gestor de Agentes y de Recursos.
   - **Gestor Agentes**: se encarga de la creación y gestión de los agentes de la aplicación.
   - **Gestor Recursos**: se encarga de la creación y gestión de los recursos.

Para explicar tanto la estructura como la comunicación de los componentes explicados arriba se han utilizado **diagramas UML**.

Se han implementado varios tipos de diagrama, tanto diagramas estáticos para mostrar la estructura de los componentes como dinámicos para explicar alguna comunicación entre componentes o una funcionalidad programada para facilitar el entendimiento.

- **Diagramas estáticos**

  Este tipo de diagramas se han usado para representar cada componente que deba contener el sistema además de explicar la estructura de los mismos. Aunque existen varios tipos del mismo en el proyecto se ha utilizado un único tipo de diagrama, el **diagrama de clases**.

Estos diagramas se usan para describir la estructura de algún componente mostrando sus clases internas, atributos, métodos y relaciones entre los objetos. Usado sobre todo para **entender como están relacionados los objetos y poder comprobar dependencias que quizás no queremos que existan**, un ejemplo que tuvimos importante en el proyecto fue la dependencia de componentes con la vista.

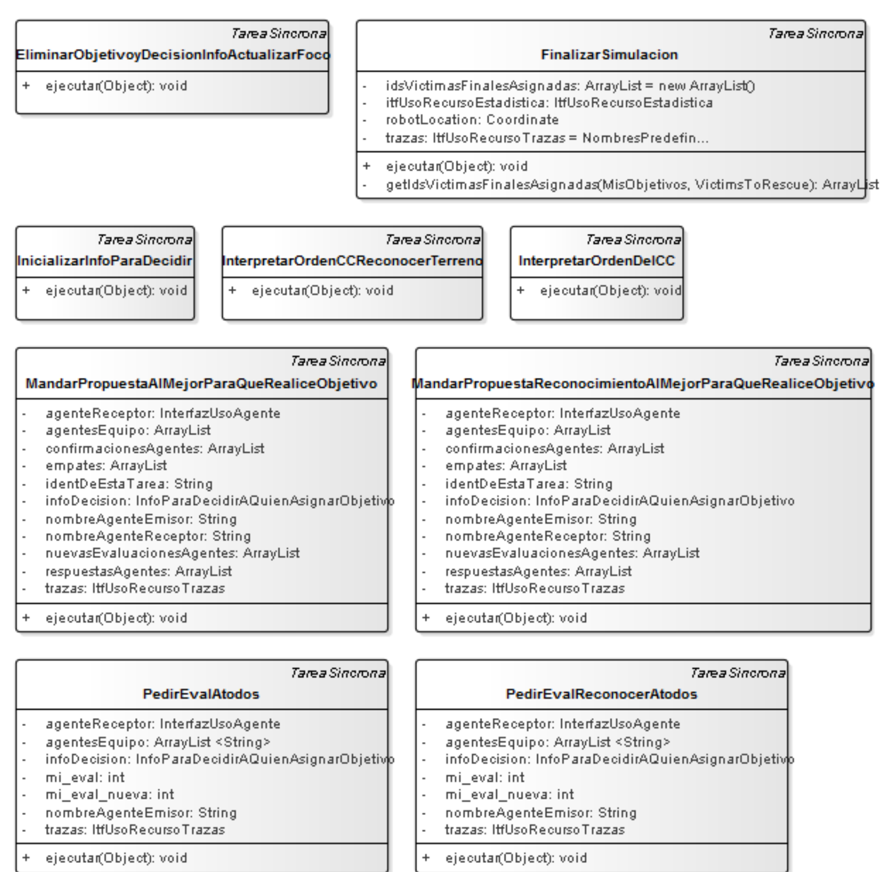

Figura 7: Parte I: Tareas del agente asignador.

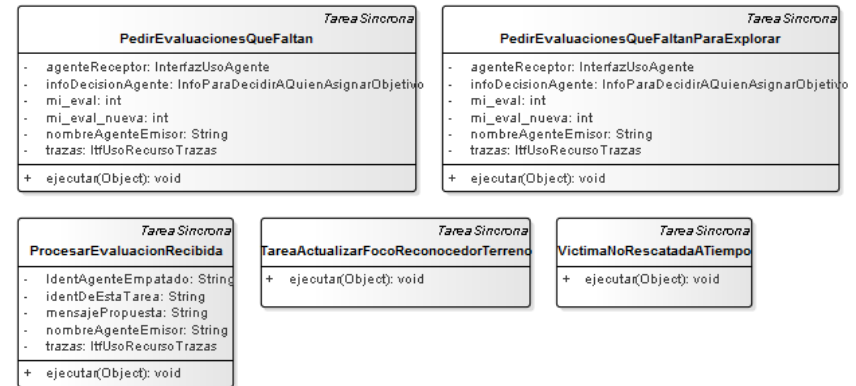

Figura 8: Parte II: Tareas del agente asignador.

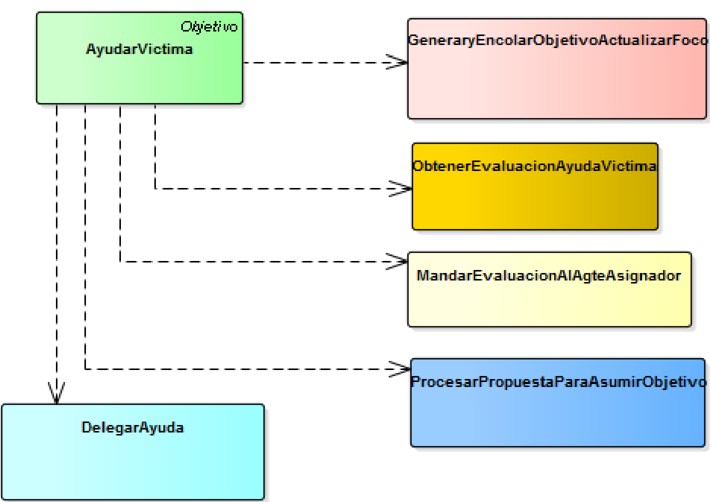

Figura 9: Dependencias entre objetivo AyudarVictima y tareas.

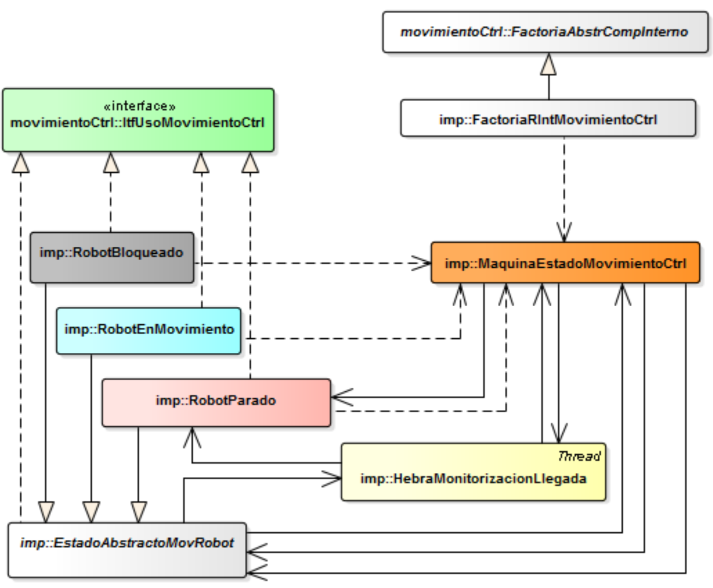

Figura 10: Dependencias de la componente de movimiento del agente.

- **Diagramas dinámicos**

  Los diagramas dinámicos expresan el comportamiento de los elementos computacionales en ejecución.

  Se detallan los siguientes:

  1. <u>Diagramas de actividad</u>

     Este tipo de diagramas se usa para **describir gráficamente un algoritmo o proceso** para facilitar su entendimiento por gente externa a su desarrollo. Se representa como un **flujo de trabajo** paso a paso hasta que el proceso esté completo.
     En el diseño del sistema se han utilizado para la representación de la concisión de los objetivos del mismo, es decir, como los objetivos interactúan desde su creación hasta la conclusión de los mismos, mostrando porque estados pasan, cuando cambian de estado y porque, con quien se comunican y con qué propósito, etc.

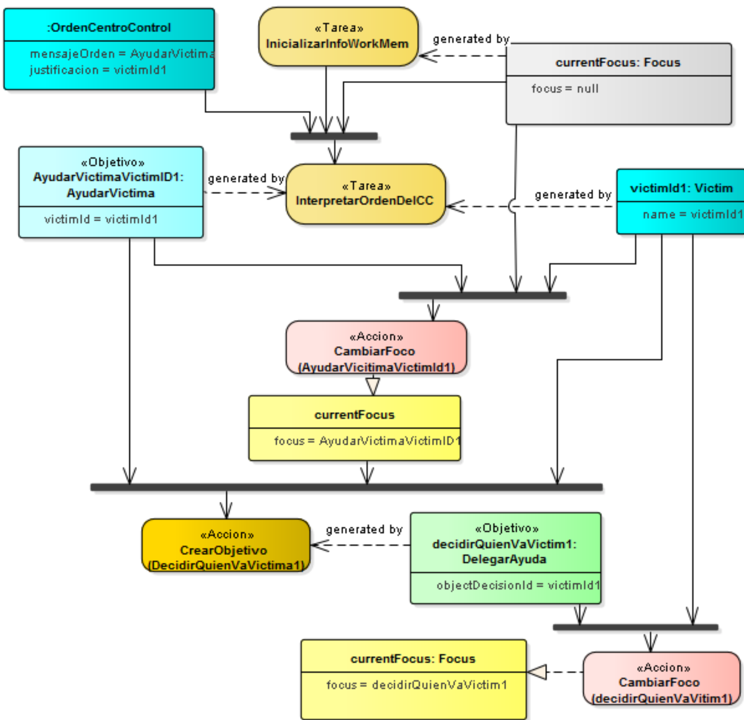

Figura 11: Diagrama de interpretar orden del CC cuando es AyudarVictima.

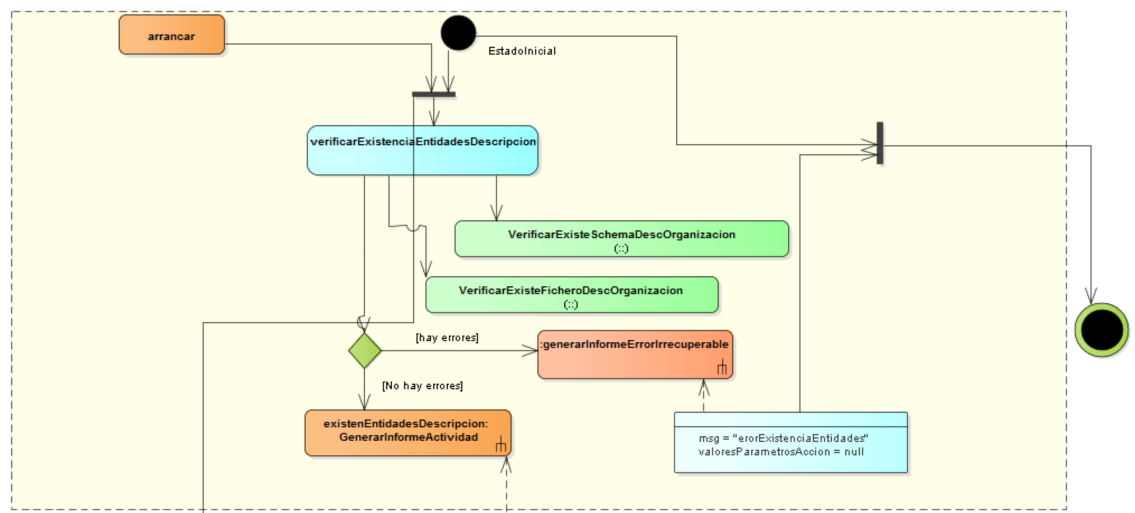

Figura 12 Parte I: diagrama de inicio.

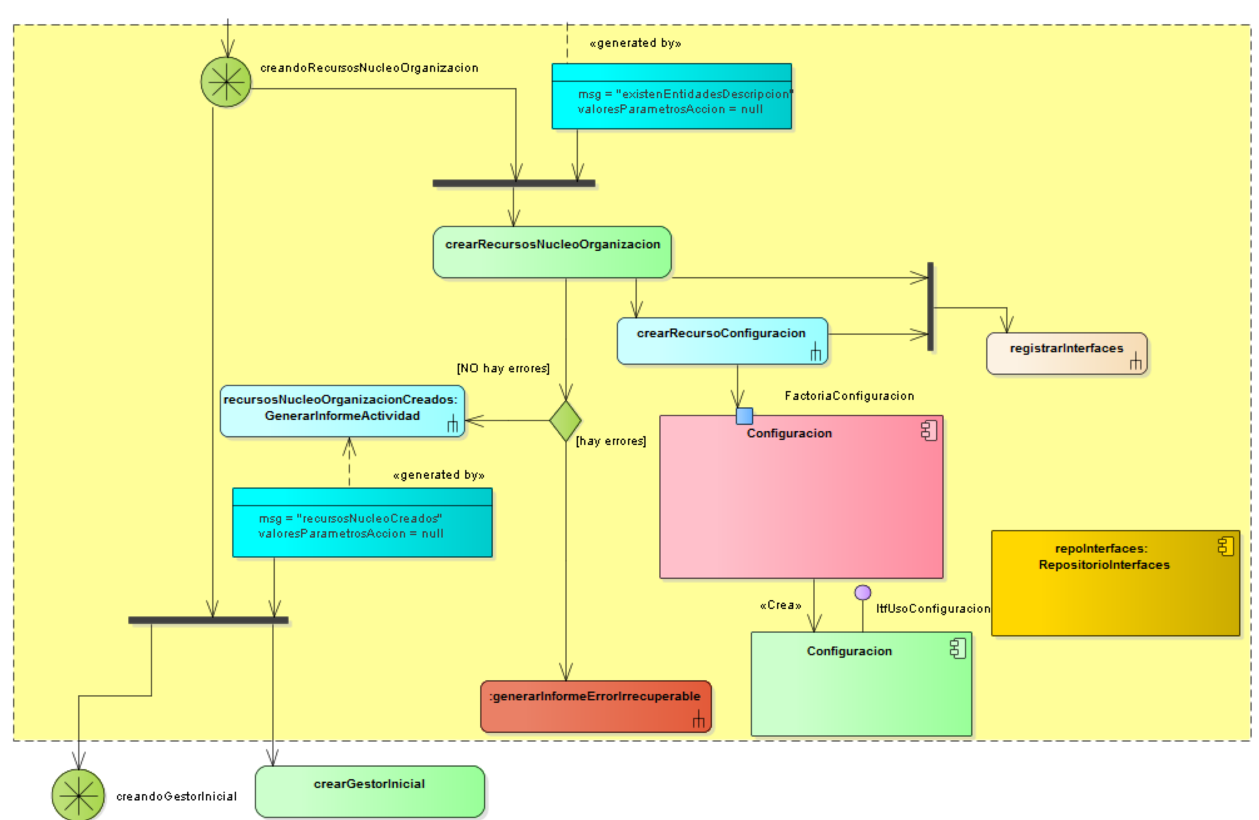

Figura 12 Parte II: diagrama de inicio

2. <u>Diagramas de secuencia</u>

   Este tipo de diagramas se usan para mostrar la interacción entre objetos, esta interacción viene dada por llamadas a métodos de un objeto o el paso de mensajes entre ellos.

Con estos diagramas se explica cómo se comporta el sistema y como se trabaja entre varios objetos para completar una tarea pudiendo así comprobar paso a paso que es lo que se quiere hacer y cómo sería la comunicación entre agentes para la completitud de este. La parte importante de estos diagramas es que permiten mostrar una funcionalidad de forma sencilla y rápida de entender para alguien ajeno al desarrollo además de afianzar la idea del cómo hacer la funcionalidad y analizar primero el impacto que tendrá esta y poder decidir si realizarla o no.

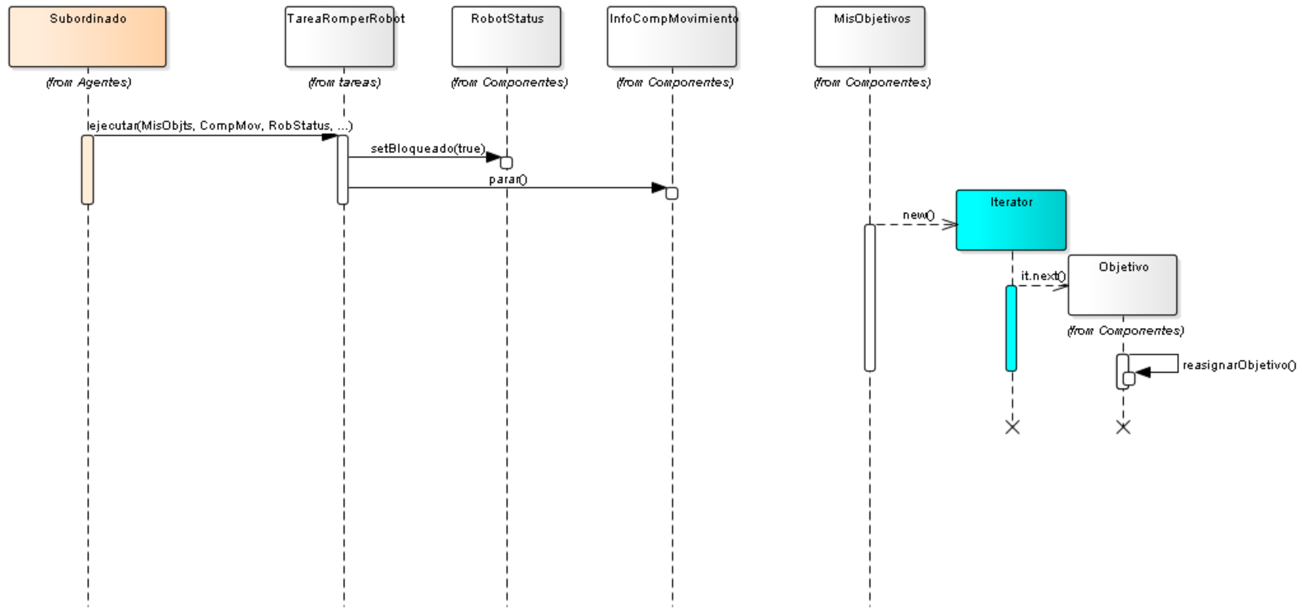

Figura 13: Diagrama de tarea romper robot.

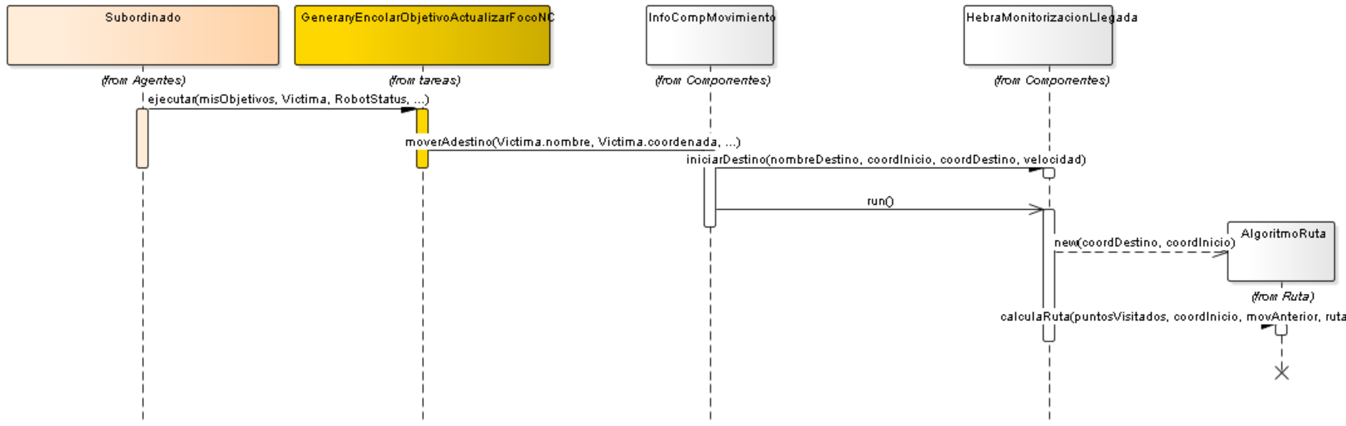

Figura 14: Diagrama de cálculo de ruta.

Figura 15: Diagrama de asignación de tareas.

2.4 MODELO DE LA INFORMACIÓN

El modelo de información define las entidades utilizadas por los componentes del sistema. Se encuentran en la carpeta **Rosace > información**, en esta podemos comprobar que se encuentra todos los tipos de mensajes que se envían entre agentes. Además, también se usa como “repositorio” para almacenar información que es común entre varios elementos. A continuación, se explicará algunos de las clases de este modelo:

- **InfoEquipo**

  Esta es una clase muy importante en el modelo, contiene información del equipo de agentes que se están usando en la misión como puede ser el identificador de equipo, identificador del jefe de equipo o número de robots en el equipo. Además, como cada agente tiene su propio elemento InfoEquipo contiene también información de los otros agentes como los identificadores de cada uno de ellos y quienes tienen el mismo rol que el agente propietario del InfoEquipo o el estado completo de cada robot definido en el sistema como RobotStatus.

- **RobotStatus**

  Este elemento del modelo almacena el estado del agente que lo tiene como propietario, para representar el estado se almacenan varios atributos:

  1. Identificador del agente.
  2. Rol que tiene en el equipo.
  3. Energía disponible.
  4. Componente de movimiento.
  5. Estado de movimiento (bloqueado o no).

Todos estos atributos se usarán en varias partes del sistema para o bien actualizarlo ante algún evento o bien comprobar el valor que tiene actualmente para analizar cómo actuará el agente.

- **OrdenCentroControl**

  Este mensaje es el más priorizado en la comunicación del sistema, se utiliza cuando el propio sistema quiere comunicar algo a los agentes, normalmente el inicio de un objetivo que se tiene que cumplir. Actualmente en el proyecto este tipo de mensaje pueden ser enviados también por el agente que actuará como explorador para avisar de que ha encontrado una víctima.

- **Victim**

  Esta clase refleja la representación de la víctima en el sistema, contiene toda la información de esta que se necesita en el rescate:

  1. Nombre de la víctima.
  2. Coordenadas donde esta se encuentra.
  3. Tiempo de vida que tiene la víctima.
  4. Estado vital de la víctima (si está viva o no).
  5. Si la víctima se ha encontrado o no.
  6. Identificador del robot que está encargado de la víctima.
  7. Si la víctima se ha rescatado o no.

Igual que en RobotStatus estos atributos se actualizarán durante la ejecución dependiendo de los eventos que ocurran y se utilizarán para comprobar el valor actual y actuar en consecuencia a estos valores.

- **VictimsToRescue**

  Elemento del modelo de información que contiene la información de todas las víctimas a rescatar en la misión. La información de cada víctima la tiene contenida en elementos de tipo Victim que se han explicado antes, además contiene una

lista de las víctimas no asignadas, la cual se usará para comprobar en medio de ejecución cuales son estas víctimas y poder controlar que siempre se asignen todas las víctimas ya que podría surgir el problema de que un robot que tenga asignada varias víctimas se rompa y se necesite saber que víctimas son las que habría que reasignar, o que por problemas de reglas falte alguna víctima por tratar.

- **VocabularioRosace**

  Esta clase del modelo se crea por la necesidad de tener una zona donde se pueda acceder desde cualquier sitio. Aquí se almacenan los elementos comunes a todo el sistema, se almacenan sobre todo valores de texto que se usan en mensajes ara indicar el tipo o el contenido de los mismos, además de nombres de agentes y recursos con sus interfaces. Gracias a esto se facilita que todos los elementos que se usen en varias zonas del sistema sean siempre el mismo para facilitar que sea común en todos.

- **PeticionAgente**

  Este elemento se usa por el agente asignador cuando quiere tiene algún objetivo que aún se necesita asignar a un agente para su cumplimiento, lo que hace es crear una petición para el objetivo y se la envía a todos los agentes subordinados normalmente para que estos contesten con una evaluación de lo que les costará cumplir dicho objetivo. La petición contiene varios atributos como puede ser el identificador del agente que realiza la petición o el mensaje que indica sobre que se realiza la petición.

- **EvaluacionAgente**

  Esta clase de información se usa por el agente al que le han enviado la petición que se ha explicado arriba para la concesión de un objetivo. El agente que la ha recibido realizará los cálculos necesarios para analizar el coste que supondría si el objetivo se le asignara a él para luego encapsular esta información en la EvaluacionAgente. La información que encapsula esta clase es el identificador del agente que está calculando el coste y el propio coste de completar el objetivo.

- **AceptacionPropuesta**

  Este mensaje se envía al asignador de parte de un subordinado que acaba de aceptar un objetivo que el asignador ha decidido que realizaría este. Contiene cuatro atributos que se definen en la creación del mensaje por parte del subordinado:

  1. Identificador del agente que envía el mensaje, es decir, la id del subordinado.

2. Mensaje que determina que el agente que ha recibido la propuesta la acepta. Este mensaje está definido en otra clase del modelo de información que se explicará más adelante definido como VocabularioRosace.
3. La propuesta que se le ha enviado por el asignador para que se sepa a qué propuesta es al que hace referencia el mensaje de aceptarla.
4. El identificador del objeto al que hace referencia esta propuesta, normalmente se tratara del nombre de una víctima.

- **RechazarPropuesta**

  El elemento de RechazarPropuesta se utiliza cuando un agente que recibe el mensaje de que el asignador le ha indicado que realice una tarea no puede hacerla en este momento, así que en vez de crear una clase de AceptacionPropuesta, el que crea es de este tipo para indicar que no la puede realizar y así que el asignador pueda asignarle la tarea a otro agente que si pueda. Esta clase tiene el identificador de agente que rechaza la tarea, el mensaje que hace referencia a que rechaza la tarea y la petición que rechaza para que se pueda vincular.

- **Coordinate**

  Es la implementación propia de una coordenada de un mapa, en este caso tendremos un *x* y un *y* que serán los puntos que usemos. Se ha redefinido el método equals para comprobar que dos coordenadas son iguales.

- **Coste**

  Esta clase se usa como una forma de encapsular el cálculo y gestión del coste que supone realizar una tarea, de esta forma se puede modificarlo tanto como se desee sin que esto afecte a la parte externa. Además, permite una ventaja más y es que podemos tener varias formas de calcular el coste que se controlen dependiendo de un mensaje que se adjunte a cuando se llama al método para calcularlo que decida cual usar. Además de tener métodos para calcular el coste dependiendo de unos parámetros, también permite obtener ese cálculo y poder trabajar únicamente con el dato que se busca.

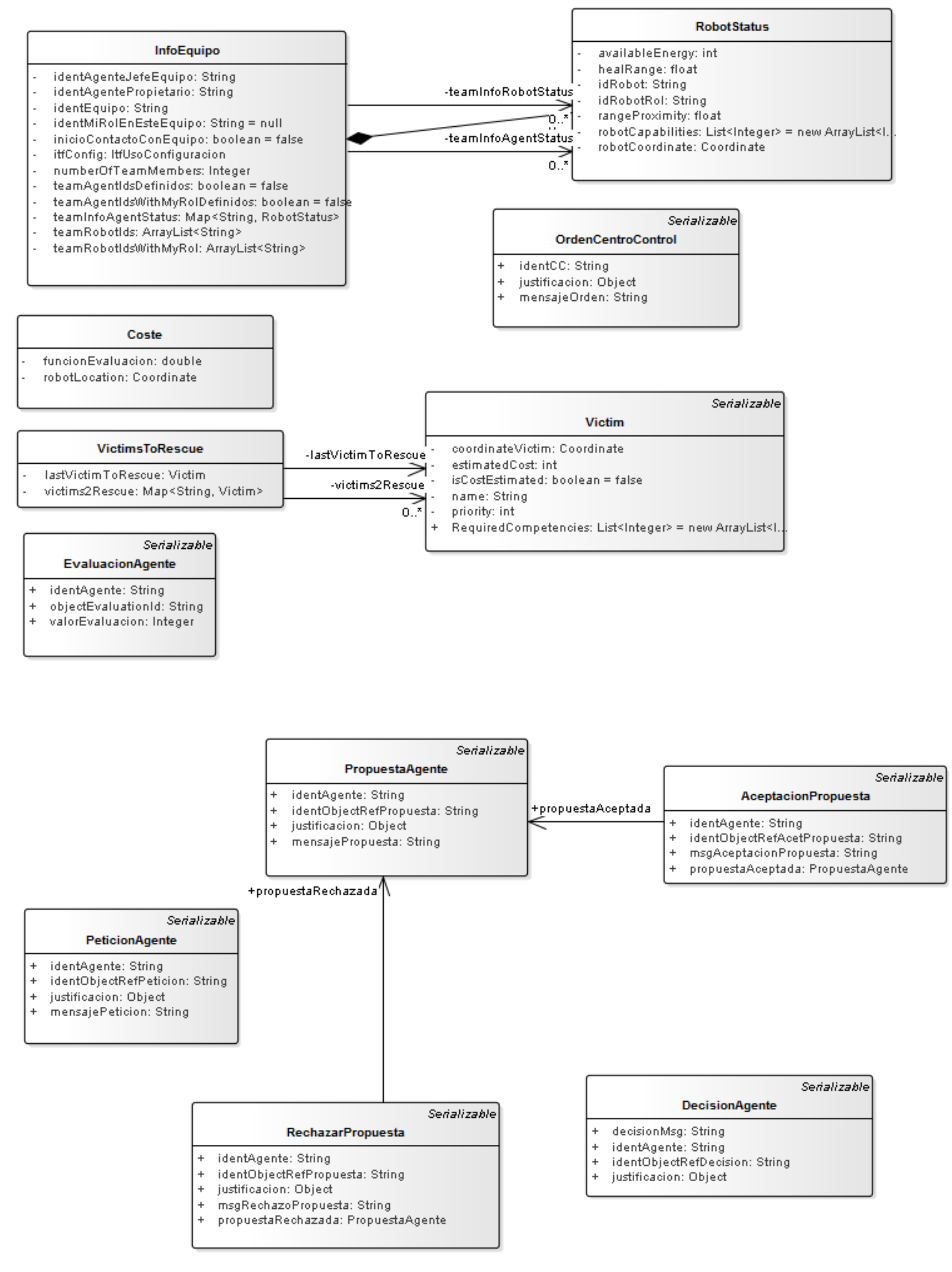

Figura 16: Diagrama de clases del modelo de información.

2.5 DISEÑO DE OBJETIVOS

Una parte del sistema muy importante son los objetivos, marcan lo que el agente debe cumplir y a lo que debe estar focalizado en cada momento. Cada objetivo tiene un tiempo de vida y una gestión diferente (aunque en algunos casos tienen partes muy semejantes) y esta gestión normalmente la lleva el agente jerárquico, es el que se encarga de crear los objetivos (al recibir un evento ya sea de un agente subordinado o del propio sistema), de asignarlos si tuviera que hacerlo y del cierre cuando se haya cumplido por el agente asignado. Hay que recordar que los agentes tienen una serie de tareas cada uno que se realizarán dependiendo de los hechos que estos tengan, estas tareas son las que el agente usará para cumplir los objetivos marcados.

En el sistema hay 5 objetivos:

- DefinirMiEquipo
- AyudarVictima
- ReconocerTerreno
- DecidirQuienVa
- TerminarSimulacion

Todos estos objetivos siguen un ciclo de vida muy parecido, creación, gestión y finalización. Algunos de ellos tienen una gestión más compleja como es el caso de AyudarVictima o ReconocerTerreno que necesitan crear objetivos secundarios para decidir a quién se le asignará. A continuación, pasaremos a explicar uno a uno cada objetivo.

2.5.1 DEFINIRMIEQUIPO

El objetivo DefinirMiEquipo es el primer objetivo que tiene el sistema, se crea cuando el sistema comienza a ejecutar y se utiliza por cada agente disponible en el sistema para obtener información de su equipo, de esta forma cada agente tendrá toda la información del equipo al que está asignado con las identidades de los demás integrantes, la identidad del jefe o que agentes tienen el mismo rol que el suyo. La utilidad de este objetivo reside en que en una misión de rescate no tendría por qué usarse únicamente un equipo de agentes, podrían ser varios, y estos necesitarían conocer a que grupo concreto es al que pertenecen para enviar su información de ejecución únicamente a este y así no corromper los demás equipos con información no útil de agentes pertenecientes a grupos ajenos; de esta forma lo que se consigue es que cada grupo de agentes sea independiente y funcione totalmente ajeno a los demás de esta forma se mejora el intercambio de información ya que se enviará únicamente a quien necesite enviarse y no se hará un mensaje *"broadcast"* recibiéndolo cualquier agente aunque no le sirva.

2.5.2 AYUDARVICTIMA

El objetivo de AyudarVictima es un objetivo prioritario en el sistema, es el objetivo que se relaciona con el salvamento de las víctimas. Este objetivo lo crea el agente asignador cuando recibe un evento de tipo *mensaje de centro de control*, este tipo de mensajes los puede enviar el agente que se ha definido como explorador o el propio sistema si es una ejecución donde no hay exploradores. En este objetivo tenemos toda la información necesaria para el salvamente

de la víctima asociada al objetivo, tenemos las coordenadas donde se encuentra, el identificador de la víctima, así como su prioridad.

Cuando el objetivo se crea se le asigna un objetivo secundario de tipo DecidirQuienVa que explicaremos más adelante, el sistema primero procederá a cumplir este objetivo para obtener que agente es el idóneo para el objetivo de ayudar a la víctima y una vez se tiene esta información se le asigna mediante una propuesta el salvamento de esta víctima. El agente asignado procederá a dirigirse a las coordenadas de la víctima y posteriormente a llevarla al lugar seguro definido en el escenario.

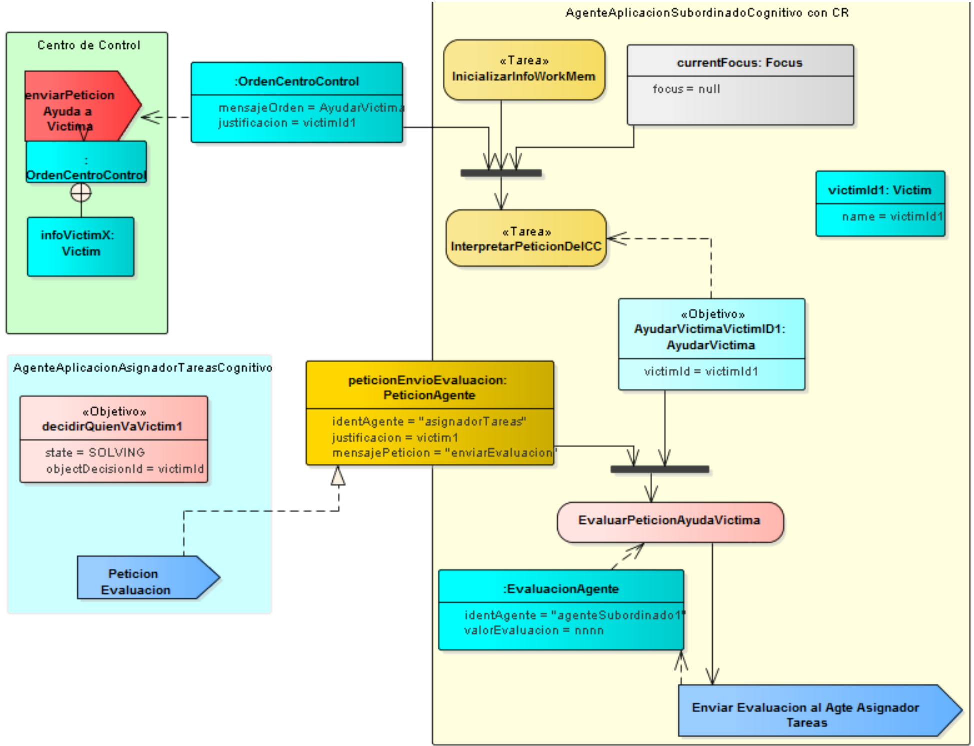

Figura 17: Creación del objetivo AyudarVictima y DecidirQuienVa relacionado.

El agente que ha sido elegido para salvar a la víctima enviará un mensaje de la finalización del objetivo independientemente de si la víctima se ha salvado o ha fallecido (esto es para que el sistema priorice las víctimas vivas y no siga intentando salvar a una víctima que ha fallecido, si no que proceda al salvamento de otra víctima que aún siga viva) y el asignador que es el que recibe este mensaje dará por cumplido el objetivo y pasará a focalizar otro. A continuación, se muestra un pequeño esquema de los pasos que sigue este objetivo desde que se crea hasta que finaliza:

1. Crear objetivos.
2. Focalizar el DecidirQuienVa asociado.
3. Se crea la información necesaria para decidir quién va.
4. Se pide las evaluaciones de los agentes para cumplir el objetivo AyudarVictima.

5. Se procesan las evoluciones y se elige un agente.
6. Se envía al agente una propuesta para realizar el objetivo.
7. Se da el objetivo por resuelto.

El último paso es cuando se elimina de los hechos el objetivo para permitir que el RETE de Drools únicamente tenga los hechos que le sirven y que no tenga nada más que pueda ocasionar fallos en la ejecución del sistema por variables que ya se han dado por finalizadas.

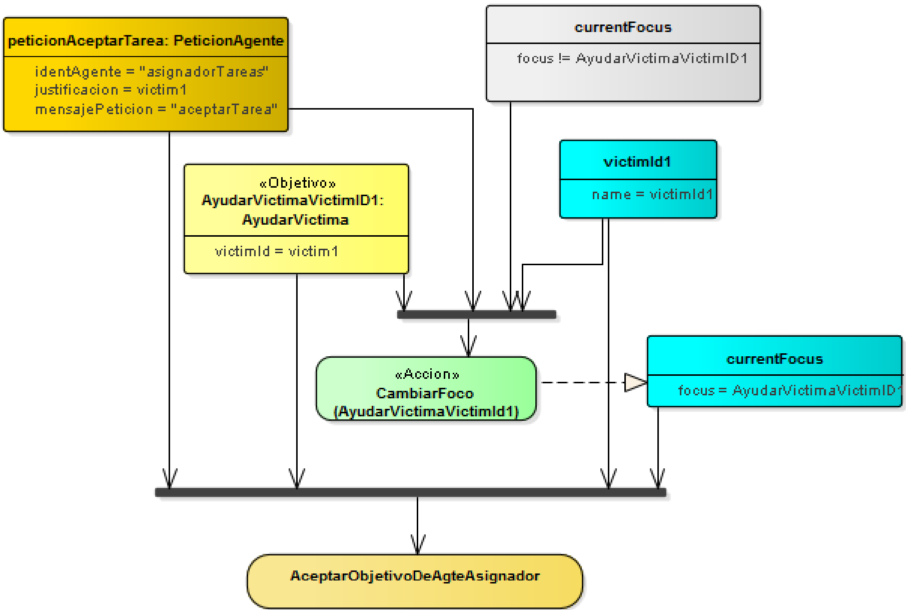

Figura 18: Diagrama de la aceptación del agente elegido para el objetivo AyudarVictima.

2.5.3 RECONOCERTERRENO

El objetivo de ReconocerTerreno comparte muchas similitudes con AyudarVictima ya que en esencia hacen lo mismo, decidir que agente es el idóneo para moverse a un sitio concreto y realizar una tarea. Este objetivo se crea cuando el sistema envía una orden al asignador indicándole que seleccione un robot explorador (esto se cumple únicamente si la simulación del escenario está hecha de forma que se vayan a utilizar los exploradores, además si se han seleccionado más de un explorador se enviara un mensaje del centro de control por cada uno).

Como en el objetivo de AyudarVictima se creará un objetivo asociado de tipo DecidirQuienVa para elegir que agente es el que mejor puede cumplir el objetivo, una vez resuelto se enviará por parte del asignador un mensaje al agente elegido para que comience a explorar el terreno.

En el caso de que hubiera más de un explorador el procedimiento es prácticamente el mismo, en un principio se divide el terreno en partes iguales y se envía a cada explorador a la esquina superior izquierda de su división de terreno que se le ha asignado, de esta forma podemos explorar un terreno por varios agentes de forma paralela aumentando la velocidad de exploración.

Algo a tener en cuenta del agente explorador es que, aunque esté realizando esa tarea se le pueden asignar víctimas para su salvamento una vez acabe de explorar el terreno que le han asignado, de esta forma no se limita al sistema a que asigne si está explorando y se consigue que trabajen todos los agentes que tengan la energía para ello y que sea siempre el que tardará menos en realizar el objetivo.

2.5.4 DECIDIRQUIENVA

El objetivo DecidirQuienVa lo usa el sistema para asociarlo a otros objetivos como puede ser AyudarVictima o ReconocerTerreno para seleccionar de todos los agentes que tenemos disponibles en el equipo cual es el que realizará el objetivo que tiene asociado el DecidirQuienVa, es decir, cada vez que se crea un objetivo que se podría llamar primario (AyudarVictima o ReconocerTerreno) se necesita saber que agente es el idóneo para realizar esta tarea siempre que esté disponible para ello (por ejemplo que no esté roto o que tenga la energía para realizarla). Para obtener esta información se utiliza este objetivo que se califica como secundario o asociado.

El sistema focalizará primero este objetivo asociado para gestionarlo antes del primario y poder obtener el identificador del agente que se usará, para saber el agente lo que hace el sistema es enviar un mensaje a todos los agentes indicando que objetivo tendrían que realizar, los agentes calculan el coste que tendrían si realizarán el objetivo lo que se traduce en el tiempo empleado. Los agentes envían su evaluación al asignador que usa esta información para seleccionar al que tenga menor coste.

Este objetivo tiene únicamente dos atributos, el identificador propio de este DecidirQuienVa y el identificador al que hace referencia para poder mantener una relación entre estos y evitar que se relacionen objetivos secundarios a algún primario al que no hacen referencia.

2.5.5 TERMINARSIMULACION

El objetivo de TerminarSimulacion se crea al inicio del sistema, este objetivo tiene asociado varios atributos ya que no únicamente comprueba el estado de la misión, es decir, si ha llegado a su fin, sino que también almacena la información de la misión necesaria para crear las gráficas que usaremos para analizar el comportamiento del sistema. Concretamente tiene los siguientes atributos:

- El número de víctimas que están resueltas, es decir, que o se han salvado o han muerto.
- El número de víctimas que ya se han asignado, útil cuando hay una víctima que por problemas del RETE del Drools no ha forzado la ejecución de la regla para interpretar el mensaje donde viene y de esta forma se puede comprobar y solucionar.
- La información asociada a cada víctima del momento en que la víctima se asignó a algún agente.

- La información asociada a cada víctima que indica el momento exacto en que la víctima se resolvió, ya sea salvándola o muriendo.

La gestión de este objetivo se hace añadiendo la información necesaria al objetivo cuando una víctima es asignada por el jefe y cuando llega un mensaje de que dicha víctima se ha resuelto. Una vez todas las víctimas están resueltas, el objetivo llama a la tarea asociada que tiene la cual almacena todos los datos en un fichero de texto para su posterior tratado y finaliza la simulación.

2.6 DISEÑO DE LA INTERFAZ

El diseño de la interfaz de usuario, se dividió en **cuatro** interfaces:

1. Interfaz de trazas

En esta interfaz se muestran las trazas de ejecución donde se puede ver información de la ejecución del sistema, además se puede acceder a cada uno de los agentes para comprobar su estado, reglas que se han ejecutado (si hubiera), etc.

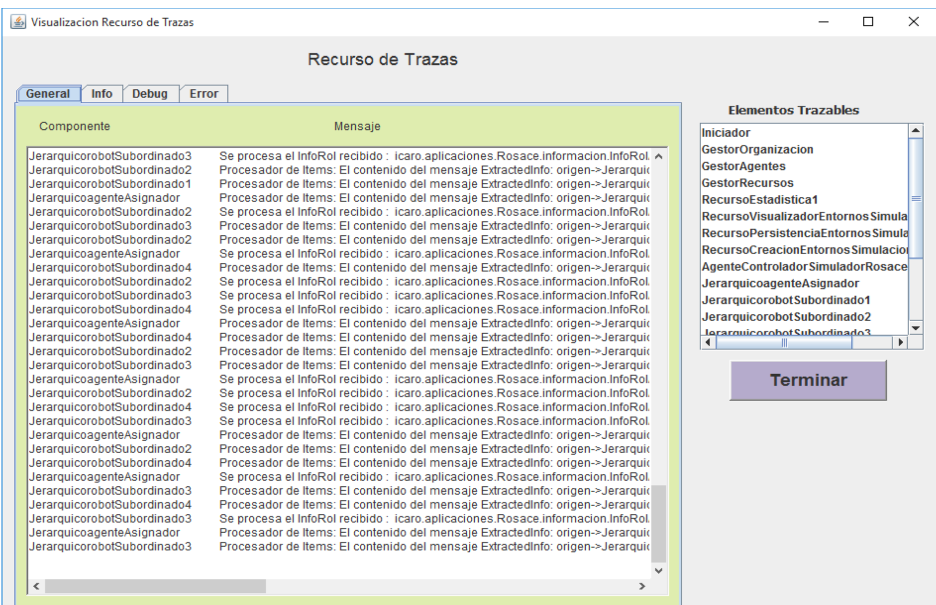

Figura 19: Interfaz de usuario: interfaz de trazas.

2. Interfaz de control

Esta interfaz controla las variables relacionadas con la ejecución del sistema, se puede marcar que función de evaluación se quiere usar en la ejecución, si se va a salvar una única víctima o todas las del escenario además pudiendo controlar cada cuanto se va a enviar la víctima al robot jefe para que trabaje en asignarla. Además, se va a mostrar información del escenario actual como son los robots que se van a usar o las víctimas que hay que salvar.

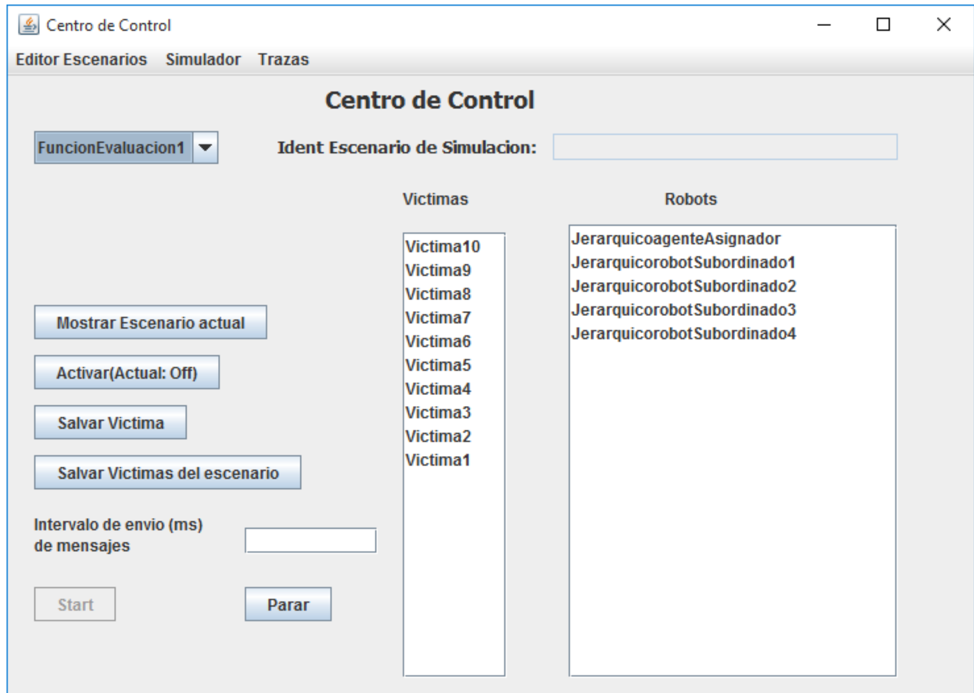

Figura 20: Interfaz de usuario: interfaz de control.

3. Interfaz de gestión de escenarios

Con esta interfaz se permite el crear escenarios nuevos y editarlos, sea cual sea la opción al pulsar clic derecho sobre cualquier punto de la pantalla se despliega un menú de acciones que tomar como:

- *Añadir un robot/víctima*: añade un robot/víctima en la posición marcada donde se desplego el menú.
- *Añadir un obstáculo*: permite añadir un obstáculo al escenario, el primer punto del obstáculo será donde se desplegó el menú mientras que el segundo punto se seleccionará cuando pulsemos clic izquierdo en cualquier parte del mapa. Cabe destacar que no se admiten obstáculos inclinados por motivos de desarrollo, por lo que al crear el obstáculo se comprobará si está más cerca de ser obstáculo horizontal o vertical y se creará así.

- *Definir un punto seguro*: esta opción permite seleccionar una coordenada del mapa como punto seguro, con lo que los robots llevarán a las víctimas a este punto.

Además, al realizar clic derecho sobre víctimas o robots tendremos opciones adicionales:

- *Modificar el tiempo de vida de las víctimas.*
- *Fijar la energía de un robot* (Teniendo en cuenta que cada pixel recorrido equivale a 1 de energía)
- *Eliminar tanto víctimas como robots.*

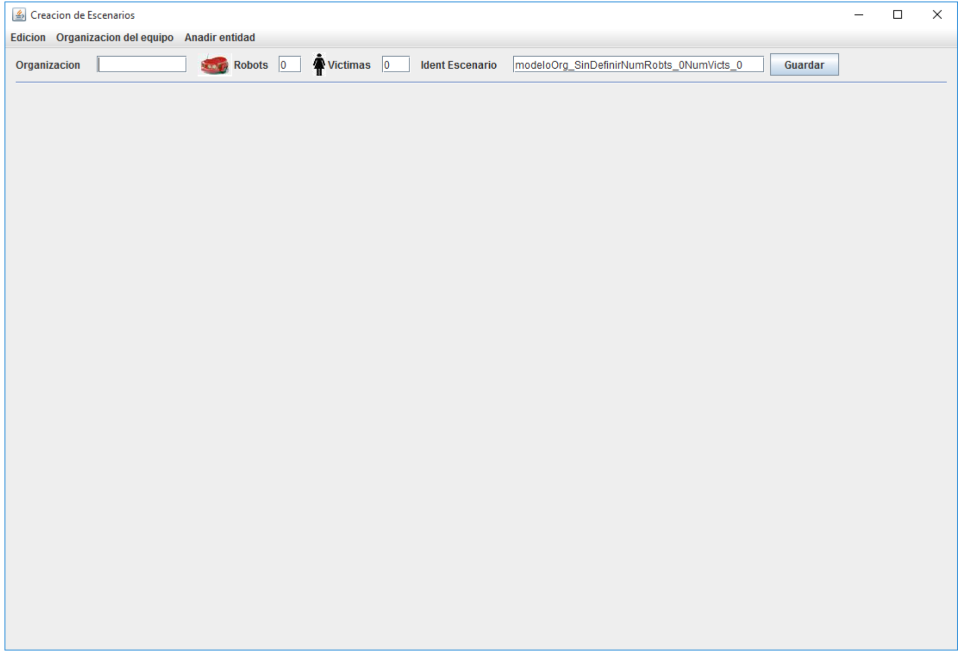

Figura 21: Interfaz de usuario: interfaz de gestión de escenarios.

4. Interfaz de misión

En esta interfaz se muestra el escenario que se ha cargado de forma visual, en este podemos ver todas las entidades que contuviera este; ya sea robots, víctimas u obstáculos. Además, esta interfaz ofrece la posibilidad de romper cualquier robot.

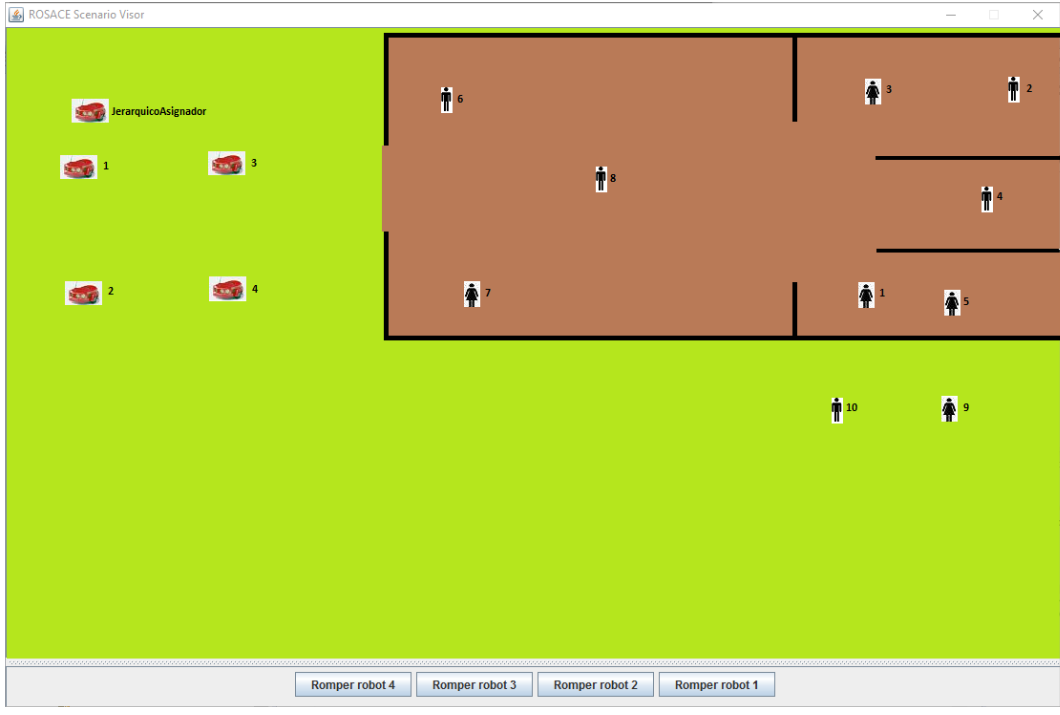

Figura 22: Interfaz de usuario: interfaz de misión.

2.7 DISEÑO DEL ALGORITMO DE CALCULO DE RUTAS

Una parte muy importante del proyecto ha sido la creación de un algoritmo para permitir a los agentes calcular la ruta que debían seguir para llegar hasta el punto del escenario donde se encontraba el objetivo a resolver.

Al empezar con el proyecto el agente calculaba la ruta usando la pendiente que había entre el punto inicial y final y seguía dicho recorrido, al implementar la posibilidad de tener obstáculos en el escenario se comprobó que esta forma de calcularla no servía puesto que los agentes no tenían una heurística relacionada con las posiciones de los obstáculos con lo que chocaban contra estos y no tenían ninguna forma de esquivarlos o de recalcular la ruta una vez chocado. Con este problema se implementó un algoritmo nuevo basado en vuelta atrás y usando una heurística que comprobaba no solo la posición de los obstáculos si no también si se estaba moviendo hacia la víctima y todo esto para las ocho posiciones donde podía ir el agente desde el punto en el que estaba (norte, sur, este, oeste, noreste, noroeste, sureste, suroeste) moviéndose solo un punto, todos estos cálculos se introducían en una cola de prioridad ordenándose por la distancia que les faltaba hasta llegar al punto final. El cálculo se hacía cada vez que el agente se movía un punto lo cual se comprobó que consumía demasiado cómputo y no solo ralentizaba en gran medida al sistema si no también consumía por entero la pila de llamadas de Java. La heurística del algoritmo era de la siguiente forma:

Si es robot estaba en la posición X se comprobarían 8 posiciones

1	2	3
4	X	5
6	7	8

Primero se analizaría a que posiciones se podría mover, por poner un ejemplo si el punto final estuviera a la derecha del agente (da igual si es al este, noreste o sureste) el agente priorizaría los movimientos que le acercaran a la víctima, estos serían el 2, 3, 5, 7 y 8 aunque estos tuvieran más distancia al punto, se introducirían y consumirían primero en la cola de prioridad que el punto 1, 4 o 6 aunque este teóricamente tuviera menos distancia al punto final.

Como se ha explicado esta solución sería una temporal ya que, aunque resolvía el problema de la ruta una vez añadidos los obstáculos, traía más problemas como el cómputo necesario (capaz de poner un i7 al 100% en una ejecución) por lo que después de implementar funcionalidades que se consideraron prioritarias se llegó a la decisión de implementar un nuevo algoritmo de rutas, denominado **Algoritmo de Lee.**

Este nuevo algoritmo de rutas es específico para la resolución de laberintos y permitir a un individuo encontrar siempre la ruta más corta si existe en un coste *nxm* siendo n el ancho y m el alto del mapa. El algoritmo funciona de la siguiente manera: es una versión de una exploración de nodos en anchura, numerando cada punto explorado con un valor numérico. Este valor numérico se calcula de la siempre manera: es el valor del punto desde el que fuiste generado más uno. De esta forma, cuando se ha llegado al punto de destino, simplemente hay que recorrer los números descendentemente para crear la ruta como una sucesión de puntos a recorrer.

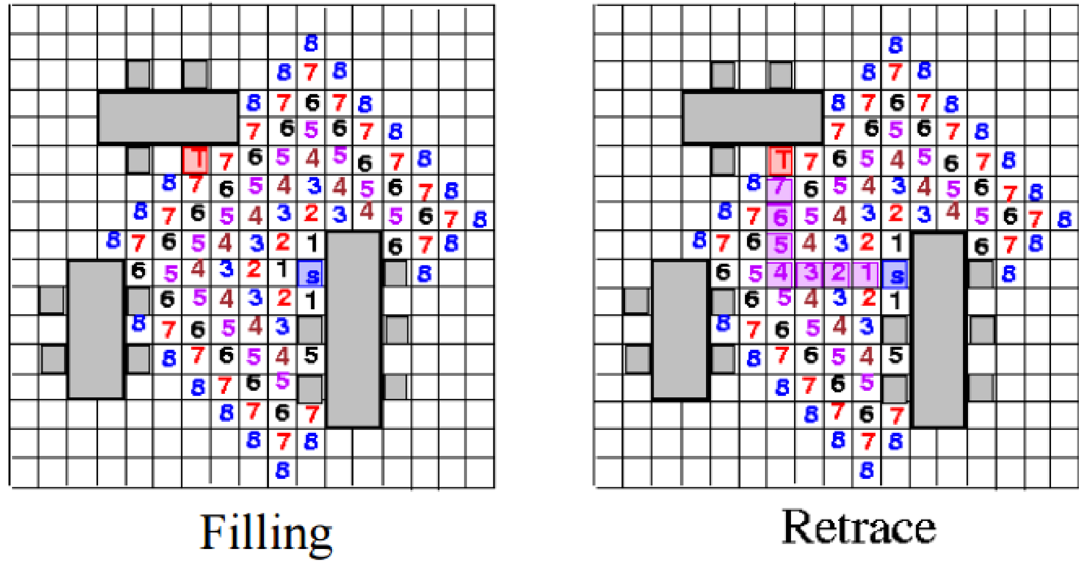

Figura 23: Ejemplo algoritmo de Lee.

Como último punto se pensó en introducir un algoritmo basado en programación genética usando una gramática para mejorarlo, pero más adelante se comprobó que la aplicación en el proyecto no era recomendable ya que se necesitaría ejecutar el algoritmo cada vez que el agente necesite moverse a una posición y se consumiría demasiado cómputo volviendo al problema de la primera implementación, además se necesita tener la certeza de que el agente llegará, y esto no se puede tener con un modelo genético donde el azar influye en gran medida por lo que se optó por mantener este nuevo algoritmo.

2.8 ARQUITECTURA

Tal y como se ha comentado en la Introducción, este proyecto está basado en un Sistema MultiAgente (SMA), por lo que antes de continuar, se va a introducir este concepto:

Un **Sistema Multiagente** es un sistema compuesto por **varios agentes inteligentes** que interactúan entre ellos. Está compuesto por:

1. Un entorno
2. Un conjunto de objetos.
   Se encuentran relacionados con el entorno.
   - Son pasivos.
   - Pueden ser percibidos, creados, destruidos y modificados por agentes.
3. Un conjunto de agentes.
   - Son objetos especiales que representan las entidades activas del sistema.
   - Tienen habilidad social (Interacción, delegación, coordinación, cooperación y negociación).
4. Un conjunto de relaciones que unen objetos y agentes.
5. Un conjunto de operaciones.
   - Hacen posible que los agentes perciban, produzcan, consuman, transformen y manipulen objetos.
6. Operadores que representan la aplicación de operaciones sobre el mundo y la reacción de éste al ser alterado.
   - Estos operadores se pueden entender como las leyes del universo.

Ahora un ejemplo de cómo funciona un SMA ambientado en nuestra aplicación.
Caso: El centro de control manda salvar a una víctima.

1. El agente asignador(jefe) está a la espera.
   - "Lee/escucha" al centro de control
   - Reconoce las "ordenes" del centro de control.

   Mientras el usuario está usando la interfaz, el agente está a la espera de recibir información que le indique que debe realizar alguna acción.

2. El agente asignador persigue satisfacer sus objetivos
    - Toma decisiones
    - Puede descomponer objetivos en subobjetivos.
    - Ejecuta tareas

   Cuando el agente recibe una petición (en este caso, salvar a una víctima), la analiza y descompone, ejecutando las acciones pertinentes para satisfacerla de forma transparente al usuario.

3. Para cumplir objetivos necesita la colaboración con otros agentes (mediante mensajería)
    - Negociación
    - Delegación
    - Coordinación

   Para realizar algunas tareas, el agente puede necesitar comunicarse con otros agentes, pedirles información o hacer que trabajen juntos. En este caso, el agente asignador (jefe) se da cuenta de que esta tarea le concierne en su mayor parte al agente mensajería.

   Como resultado, el asignador pide a todos los demás agentes de tipo "subordinado" evaluaciones para ir a salvar a la víctima en cuestión. Los agentes subordinados contestan al jefe con su evaluación, y una vez el asignador tiene todas las evaluaciones, entonces asigna la víctima a uno mediante otro mensaje a través de mensajería. Es posible que los mensajes no lleguen, por lo que existen temporizadores(timeouts) por si se pierden mensajes, que no se bloquee la aplicación.

4. Los agentes se comunican con sus propios recursos.
   Cada agente tiene sus propios recursos en los que no interfieren otros agentes, esto permite una implementación muy modulable y separada.

5. El resultado de la acción salvar a víctima se ve a través de la interfaz visual (el escenario) pintando en verde la víctima en cuestión y viendo al robot designado como encargado del salvamento en camino de la víctima en cuestión.

   En general la metodología que se ha seguido es:

    - La de asignar un agente a cada rol de la aplicación definiendo para él sus propios recursos.
    - Asignar agentes a tareas específicas que comparten recursos de manera que, se divida la funcionalidad en pequeñas clases fácilmente tratables de manera individual.

La arquitectura de la aplicación depende del modelo de equipo que se quería llevar a cabo, esto es por ejemplo, si es un equipo de robots con organización Igualitaria o Jerárquica basada en roles. En nuestro proyecto se ha desarrollado únicamente el modelo de equipo basado en roles, es decir, el modelo Jerárquico.

Una vez aclarado esto, la aplicación se creó con un agente por cada rol, y en nuestro caso, hay agente asignador (jefe) y agente subordinado. Aparte de estos dos agentes principales, se necesita otro agente llamado AgteControladorSimuladorRosace con tareas diversas como la comunicación inicial con los agentes para inicializaciones.

A continuación se adjunta el estudio que se aplicó para determinar que agentes debía tener la aplicación:

Estudio de agentes:

Las operaciones que aparecen asignadas a cada agente son las que realizan en cada simulación:

AgenteControladorSimuladorRosace: acciones propias del controlador del sistema.

Operaciones:

- Realizar las acciones iniciales (inicialización de agentes, GUI's, demás controladores…)
- Dar la orden de iniciar simulación: tanto con exploración como sin exploración.
- Dar la orden de salvar a una víctima en concreto.
- Dar la orden de visualización de víctima asignada, desasignada o muerta.
- Almacenamiento de las estadísticas.
- Mandar orden de parada a un robot.
- Actualización del escenario
- Comunica con los agentes Agente Jerárquico Asignador y Agente Robot Subordinado.

AgenteJerarquicoAsignador: Acciones propias de un jefe de equipo.

Operaciones:

- Creación de objetivos: DecidirQuienVa, AyudarVictima, ReconocerTerreno, TerminarSimulacion.
- Interpretar órdenes del Centro de Control (representado por una GUI que comunica con AgenteControladorSimuladorRosace).
- Pedir evaluaciones a los robots subordinados.
- Asignar víctimas a robots basándose en las evaluaciones.
- Salvar víctimas.

- Comunicar al AgenteControladorSimuladorRosace la muerte de una víctima.
- Comprobar el fin de la simulación.
- Comunicación con los agentes AgenteControladorSimuladorRosace, AgenteRobotSubordinado.

AgenteRobotSubordinado: Acciones propias de un subordinado en un equipo.

Operaciones:

- Salvar víctima(s).
- Reconocer terreno.
- Notificar de la existencia de víctimas en un área.
- Calcular rutas hacia objetivos.
- Moverse hacia un destino.

  Comunicación con los agentes AgenteControladorSimuladorRosace, AgenteJerarquicoAsignador.

3. IMPLEMENTACIÓN

3.1 ESTRUCTURA DE PAQUETES. IMPLEMENTACIÓN DE AGENTES Y RECURSOS

3.1.1 AGENTES

La implementación de un agente se debe realizar basándose en una plantilla ya definida en ICARO. Una vez se tiene una idea de cuáles son las funcionalidades del agente, se debe decidir si es más conveniente utilizar un funcionamiento mediante autómata o mediante un sistema de reglas. En este proyecto se han implementado agentes de los dos tipos: tanto con reglas como con autómatas.

Para que sirva de ejemplo, el AgenteControladorSimuladorRosace es un agente basado en un autómata con transiciones a distintos estados.

Estructura de un agente basado en un autómata:

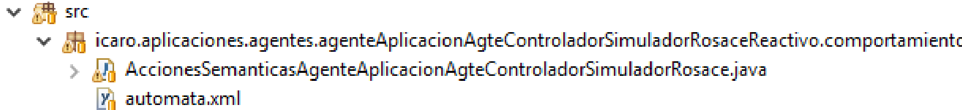

Figura 24: Paquete del agente y su autómata

- **automata.xml**. Es un fichero XML que representa el autómata y su comportamiento

```xml
<tablaEstados descripcionTabla="Tabla de estados para el agente AgenteAplicacionAgteControladorSimuladorRosace">
<!--**********************************************************************************
    ******************     Declaracion del estado inicial y su transicion          ******************
    **********************************************************************************-->
  <estadoInicial idInicial="estadoInicial">
    <transicion input="comenzar" accion="AccionComenzar" estadoSiguiente="esperandoIniciarSimulacion" modoDeTransicion="bloqueante"/>
  </estadoInicial>

<!--**********************************************************************************
    ******************     Declaracion de estados intermedios y sus transiciones      ******************
    **********************************************************************************-->

  <estado idIntermedio="esperandoIniciarSimulacion">
    <transicion input="actualizarEscenarioActual" accion="updateEscenario" estadoSiguiente="esperandoIniciarSimulacion" modoDeTransicion="bloqueante"/>
    <transicion input="iniciarBusqueda" accion="iniciarBusqueda" estadoSiguiente="simulando" modoDeTransicion="bloqueante"/>
    <transicion input="sendSequenceOfSimulatedVictimsToRobotTeam" accion="SendSequenceOfSimulatedVictimsToRobotTeam" estadoSiguiente="simulando" modoDeTransicion="bloqueante"/>
    <transicion input="peticionSimulacionVictima" accion="SendSimulatedVictimToRobotTeam" estadoSiguiente="simulando" modoDeTransicion="bloqueante"/>
    <transicion input="mostrarEscenarioActualSimulado" accion="MostrarEscenarioActualSimulado" estadoSiguiente="esperandoIniciarSimulacion" modoDeTransicion="bloqueante"/>

  </estado>


  <estado idIntermedio="simulando">
    <transicion input="sendSequenceOfSimulatedVictimsToRobotTeam" accion="SendSequenceOfSimulatedVictimsToRobotTeam" estadoSiguiente="simulando" modoDeTransicion="bloqueante"/>
    <transicion input="peticionSimulacionVictima" accion="SendSimulatedVictimToRobotTeam" estadoSiguiente="simulando" modoDeTransicion="bloqueante"/>
    <transicion input="victimaAsignadaARobot" accion="VictimaAsignadaARobot" estadoSiguiente="simulando" modoDeTransicion="bloqueante"/>
    <transicion input="desasignarVictima" accion="desasignarVictima" estadoSiguiente="simulando" modoDeTransicion="bloqueante"/>
    <transicion input="victimaMuerta" accion="victimaMuerta" estadoSiguiente="simulando" modoDeTransicion="bloqueante"/>
    <transicion input="finSimulacion" accion="FinSimulacion" estadoSiguiente="finalizandoSimulacion" modoDeTransicion="bloqueante"/>
    <transicion input="mostrarEscenarioActualSimulado" accion="MostrarEscenarioActualSimulado" estadoSiguiente="simulando" modoDeTransicion="bloqueante"/>
    <transicion input="enviarIdentsEquipo" accion="MostrarRobotsActivos" estadoSiguiente="simulando" modoDeTransicion="bloqueante"/>
    <transicion input="pararRobot" accion="PararRobot" estadoSiguiente="simulando" modoDeTransicion="bloqueante"/>
    <transicion input="iniciarBusqueda" accion="iniciarBusqueda" estadoSiguiente="simulando" modoDeTransicion="bloqueante"/>
  </estado>


  <estado idIntermedio="finalizandoSimulacion">

    <transicion input="finSimulacion" accion="FinSimulacion" estadoSiguiente="finalizandoSimulacion" modoDeTransicion="bloqueante"/>

    <transicion input="mostrarEscenarioActualSimulado" accion="MostrarEscenarioActualSimulado" estadoSiguiente="finalizandoSimulacion" modoDeTransicion="bloqueante"/>

  </estado>

<!--**********************************************************************************
    ******************     Declaracion de estados finales                          ******************
    **********************************************************************************-->

  <estadoFinal idFinal="estadoFinal"/>

<!--**********************************************************************************
    ******************     Declaracion de transiciones universales                 ******************
    **********************************************************************************-->

</tablaEstados>
```

Figura 25: Ejemplo de fichero automata.xml

Esta figura es una extracción del archivo "automata.xml" del AgenteControladorSimuladorRosace.

Hay tres tipos de estados: inicial, intermedio y final.

- **AccionesSemanticasAgenteAplicacionAgteControladorSimuladorRosace.** Se trata de la clase que se encarga de implementar su comportamiento. Es decir, cada una de las acciones que puede realizar el agente según refleja su autómata (iniciarBusqueda, sendSequenceOfSimulatedVictimsToRobotTeam, FinSimulacion o MostrarEscenarioActualSimulado)

```
//Esta accion semantica se ejecuta cuando se envia el input "finSimulacion" en la
//tarea sincrona FinalizarSimulacion del agente Subordinado y el igualitario
//Nos permite generar un fichero EstadisticaFinalSimulacionAsignacionMisionV2.xml que resume que victimas han sido asignadas a cada robot.

public void FinSimulacion(String robot, ArrayList idsVictimasFinalesAsignadas, Double tiempoTotalCompletarMisionAtenderVictimasFinalesAsignadas) {

    trazas.aceptaNuevaTraza(new InfoTraza(this.nombreAgente, "Accion FinSimulacion  .... "
            + "robot->" + robot + " ; idsVictimasFinalesAsignadas->" + idsVictimasFinalesAsignadas
            + " ; tiempoTotalCompletarMisionAtenderVictimasFinalesAsignadas->" + tiempoTotalCompletarMisionAtenderVictimasFinalesAsignadas, InfoTraza.NivelTraza.debug));
    try {
        ArrayList<InfoAsignacionVictima> infoAsignVictms = new ArrayList();
        infoAsignVictms = itfUsoRecursoPersistenciaEntornosSimulacion.obtenerInfoAsignacionVictimas();
        contadorRobotsQueContestanFinsimulacion++;
        if (contadorRobotsQueContestanFinsimulacion == identsAgtesEquipo.size())
            this.itfUsoRecursoVisualizadorEntornosSimulacion.mostrarResultadosFinSimulacion();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public void MostrarEscenarioActualSimulado() {

    trazas.aceptaNuevaTraza(new InfoTraza(this.nombreAgente, "Accion MostrarEscenarioActualSimulado  ....", InfoTraza.NivelTraza.debug));
    try {
        //            itfUsoRecursoCreacionEntornosSimulacion.MostrarEscenarioActualSimulado();
        itfUsoRecursoVisualizadorEntornosSimulacion.mostrarEscenario();

    } catch (Exception e) {
        e.printStackTrace();
    }
}
public void  MostrarRobotsActivos (){
    try {
        itfUsoRecursoVisualizadorEntornosSimulacion.mostrarIdentsEquipoRobots(this.identsAgtesEquipo);
    } catch (Exception ex) {
        Logger.getLogger(AccionesSemanticasAgenteAplicacionAgteControladorSimuladorRosace.class.getName()).log(Level.SEVERE, null, ex);
    }
}
public void  PararRobot (String idRobot){
    /*OrdenParada orden = new OrdenParada(nombreAgente);
    //        orden.setMensajePeticion(VocabularioRosace.MsgePeticionParar);
    this.comunicator.enviarInfoAotroAgente(orden, idRobot);*/

    Informacion x=new Informacion(VocabularioRosace.MsgRomperRobot);
    this.comunicator.enviarInfoAotroAgente(x, idRobot);
}
```

Figura 26: Acciones semánticas del agente

Y, por último, está el agente basado en reglas, que, por poner un ejemplo, mostraremos reglas y ejemplo de funcionamiento del AgenteJerarquicoAsignador.

Los agentes basados en reglas que hay en el proyecto son los robots, que funcionan con un fichero de reglas concreto para cada agente. El sistema de reglas está basado en Drools 5.4, y el funcionamiento es idéntico a cualquier sistema de reglas: se establecen una serie de premisas para que salte una regla (las premisas, o antecedente, serán en nuestro caso objetos creados o condiciones en objetos ya existentes en el RETE o “base de hechos”) y una vez se cumplen las premisas, se ejecuta la parte derecha, es decir, el consecuente, que en nuestro caso será comúnmente la modificación(o eliminación) sobre objetos desde el propio sistema de reglas y la ejecución de Tareas concretas.

agenteAplicacionAsignadorTareasCognitivo
- informacion
- objetivos
- procesoResolucionObjetivos
  - reglas.drl
  - reglasJerarqJefeCambioRol.drl
- tareas
  - ComprobarSimulacionFinalizada.java
  - EliminarObjetivoyDecisionInfoActualizarFoco.java
  - FinalizarSimulacion.java
  - InicializarInfoParaDecidir.java
  - InterpretarOrdenCCReconocerTerreno.java
  - InterpretarOrdenDelCC.java
  - MandarPropuestaAlMejorParaQueRealiceObjetivo.java
  - MandarPropuestaReconocimientoAlMejorParaQueRealiceObjetivo.java
  - PedirEvalAtodos.java
  - PedirEvalReconocerAtodos.java
  - PedirEvaluacionesQueFaltan.java
  - PedirEvaluacionesQueFaltanParaExplorar.java
  - ProcesarEvaluacionRecibida.java
  - TareaActualizarFocoReconocedorTerreno.java
  - VictimaNoRescatadaATiempo.java

Figura 27: Estructura de reglas y tareas agente

Un ejemplo de cómo se define una regla en Drools es el siguiente:

```
rule “Nombre de la regla”

when

exists(ObjetoEjemplo)

objeto: ObjetoEjemplo (condicionX == true)

then

Tarea ejemplotarea = new
TareaSincrona(EjemploTarea.class);

tarea. ejecutar(objeto);//Ejecutar la tarea
EjemploTarea mandando “objeto”, instancia de la
clase ObjetoEjemplo

end
```

Para ver un ejemplo concreto, se muestra cómo es una parte de un fichero de reglas del AgenteJerarquicoAsignador:

```
//=========================================================================
//   Reglas del ciclo de vida de la consecucion del objetivo DecidirQuienVa
//
//=========================================================================
//

rule "Comenzar la consecucion del objetivo DecidirQuienVa primera vez"
  when
    miEquipo: InfoEquipo(idTeam:teamId,inicioContactoConEquipo== true)
    victimaCC:Victim(idVict:name)
    AyudarVictima(state == Objetivo.PENDING,victimId == idVict)
    obj:DecidirQuienVa (state == Objetivo.SOLVING,objectDecisionId == idVict)
    focoActual:Focus(foco == obj )
    not (exists ( InfoParaDecidirAQuienAsignarObjetivo(idElementoDecision == idVict)))
  then
     TareaSincrona tarea = gestorTareas.crearTareaSincrona(InicializarInfoParaDecidir.class); //mete en el motor un InfoParaDecidirQuienVa inicializado
     tarea.ejecutar(obj, idVict,miEquipo );
     System.out.println( "\n"+agentId +" EJECUTO LA REGLA: "+drools.getRule().getName()+" \n\n" );
     recursoTrazas.aceptaNuevaTrazaEjecReglas(agentId,"( ID Victima: "+idVict + " ) EJECUTO LA REGLA " + drools.getRule().getName() + "\n");
end

//=================================================================================================================================

rule "Pido la evaluacion para realizar el objetivo al equipo de robots "
no-loop true
  when
    victimaCC:Victim(idVict:name)
    obj:DecidirQuienVa (state == Objetivo.SOLVING,objectDecisionId == idVict )
    focoActual:Focus(foco == obj )
    infoDecision: InfoParaDecidirAQuienAsignarObjetivo(idElementoDecision == idVict,hanLlegadoTodasLasEvaluaciones== false, peticionEvaluacionEnviadaAtodos == false )
    then
    TareaSincrona tarea = gestorTareas.crearTareaSincrona(PedirEvalAtodos.class);
    tarea.ejecutar(obj,infoDecision,victimaCC );
    recursoTrazas.aceptaNuevaTrazaEjecReglas(agentId,"( ID Victima: "+idVict + " ) EJECUTO LA REGLA " + drools.getRule().getName() + "\n");
    System.out.println( "\n"+agentId +"EJECUTO LA REGLA: "+drools.getRule().getName()+" \n\n" );

 end
```

Figura 28: Ejemplo reglas Drools

Como se puede observar en la regla *Pido la evaluación para realizar el objetivo al equipo de robots*, se va a ejecutar la tarea *PedirEvalATodos,* que principalmente se encarga de pedir evaluaciones a todos los robots del equipo para rescatar a un objetivo (ya sea una víctima o una exploración), y entonces se ejecutará el método *ejecutar* de la tarea *PedirEvalATodos*. Como ejemplo, se muestra el método *ejecutar:*

```
public class PedirEvalAtodos extends TareaSincrona {
    private InterfazUsoAgente agenteReceptor;
    private ArrayList <String> agentesEquipo;//resto de agentes que forman mi equipo
    private int mi_eval, mi_eval_nueva;
    private String nombreAgenteEmisor;
    private ItfUsoRecursoTrazas trazas;
    private InfoParaDecidirAQuienAsignarObjetivo infoDecision;

    // private TimeOutRespuestas tiempoSinRecibirRespuesta;  //no usado

    @Override
    public void ejecutar(Object... params) {
        try {
            trazas = NombresPredefinidos.RECURSO_TRAZAS_OBJ;
            Objetivo objetivoEjecutantedeTarea = (Objetivo)params[0];
            infoDecision = (InfoParaDecidirAQuienAsignarObjetivo)params[1];
            Victim victima = (Victim)params[2];

            nombreAgenteEmisor = this.getAgente().getIdentAgente();
            String identTarea = this.getIdentTarea();
            trazas.aceptaNuevaTraza(new InfoTraza(nombreAgenteEmisor, "Se Ejecuta la Tarea :"+ identTarea , InfoTraza.NivelTraza.debug));
            if(!infoDecision.peticionEvaluacionEnviadaAtodos){
                agentesEquipo = infoDecision.getIdentsAgentesEquipo();
                int numAgtesEnEquipo = agentesEquipo.size();
                if(ControladorVisualizacionSimulRosace.asignadorSeMueve){
                    infoDecision.anadirAgenteAInformacion(this.identAgente);
                    numAgtesEnEquipo++;
                    agentesEquipo.add(this.identAgente);
                }

                if(numAgtesEnEquipo ==0) trazas.aceptaNuevaTraza(new InfoTraza(nombreAgenteEmisor,
                        "  En la tarea : " +identTarea + " No se puede enviar la peticion porque el grupo de agentes a los que hay que enviar la informacion esta vacio ",
                else{
                    PeticionAgente peticionEval = new PeticionAgente(nombreAgenteEmisor);
                    peticionEval.setidentObjectRefPeticion(objetivoEjecutantedeTarea.getobjectReferenceId());
                    peticionEval.setMensajePeticion(VocabularioRosace.MsgPeticionEnvioEvaluaciones);
                    peticionEval.setJustificacion(victima); // para que se sepa qué evaluacion le pedimos
                    this.getComunicator().informaraGrupoAgentes(peticionEval, agentesEquipo);
                    infoDecision.setRespuestasEsperadas(agentesEquipo.size());
                    infoDecision.setPeticionEvaluacionEnviadaAtodos(Boolean.TRUE);
                    this.getEnvioHechos().actualizarHecho(infoDecision);
```

Figura 29: Ejemplo tarea PedirEvalATodos

Esta tarea, al igual que las demás, son clases que extienden a TareaSincrona e implementan el método *ejecutar* con parámetros variables. Desde cualquier clase que extienda a

TareaSincrona se puede acceder al procesador de objetivos (el sistema de reglas) y al comunicador, para comunicarse con los demás robots.

```
public abstract class TareaSincrona {

    protected ItfProcesadorObjetivos itfProcObjetivos;
    protected AgenteCognitivo agente;
    protected String  identTarea;
    protected String  identAgente;
    protected Object[] params;
    protected boolean terminada;
    protected ItfUsoRecursoTrazas trazas;
    protected ItfUsoRepositorioInterfaces repoInterfaces;
    protected ItfUsoConfiguracion itfConfig;
    protected GestorTareas gestorTareas;
    protected ComunicacionAgentes comunicator;

    public TareaSincrona(){

        this.trazas = NombresPredefinidos.RECURSO_TRAZAS_OBJ;
        this.repoInterfaces = NombresPredefinidos.REPOSITORIO_INTERFACES_OBJ;
    }

    public TareaSincrona(ItfProcesadorObjetivos envioHechos, AgenteCognitivo agente) {

        this.itfProcObjetivos = envioHechos;
        this.agente = agente;
        this.trazas = NombresPredefinidos.RECURSO_TRAZAS_OBJ;
        this.identAgente = agente.getIdentAgente();
        this.repoInterfaces = NombresPredefinidos.REPOSITORIO_INTERFACES_OBJ;
    }

    public abstract void ejecutar(Object... params);

    public void generarInformeConCausaTerminacion (String idTarea,Objetivo contxtGoal,String idAgenteOrdenante,Object contenido, CausaTerminacionTarea causaTerminacion ){

        String identGoal = null;
        if (contxtGoal!= null) identGoal = contxtGoal.getgoalId();
        InformeDeTarea resultadoTarea = new InformeDeTarea (idTarea,identGoal,idAgenteOrdenante,contenido, causaTerminacion );
        itfProcObjetivos.insertarHecho(resultadoTarea);
        //    envioHechos.insertarHecho(contenido);
    }
```

Figura 30: Ejemplo clase TareaSincrona

Básicamente, se utiliza el componente de comunicación entre robots para enviar un mensaje a todos los robots del equipo definido y éste se encarga de insertar el mensaje en el procesador de objetivos de los receptores. Cabe mencionar que un robot receptor puede ser también el mismo Asignador, ya que tiene la capacidad de participar en el rescate de forma más activa, es decir, realizando tareas de rescate y de organización simultáneamente. Algoritmo de cálculo de rutas

3.1.2 RECURSOS

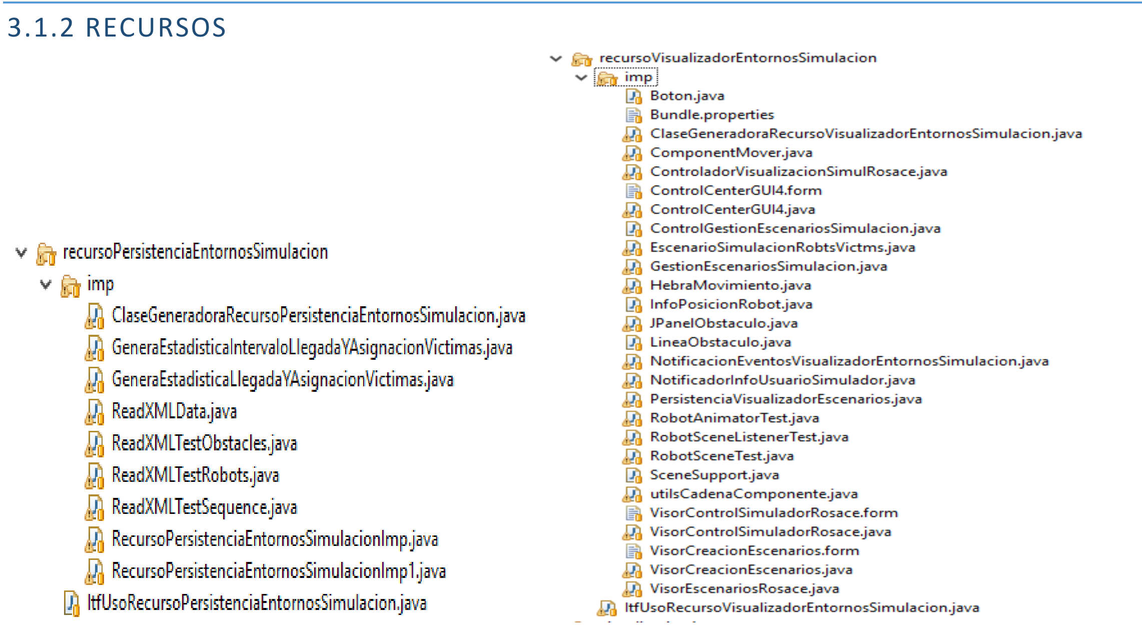

Figura 31: Representación estructura de recursos persistencia y visualizador

El contenido de las carpetas es similar para ambos tipos de recurso:

- Una clase que define la interfaz de uso del recurso.
- Visitando el interior de la carpeta *"imp"* se encuentra una clase generadora que se encarga de la creación del modelo computacional del recurso.

Para ambos tipos de recursos siempre se tienen que definir todos los paquetes que aparecen en la imagen superior.

a) Recurso de Visualización

Es el encargado de mostrar las ventanas al usuario.

En el paquete *"imp"* se mete el código de las ventanas (en este caso usando la librería grafica SWT) que se han diseñado para comunicarse con el usuario y que competen al agente de este recurso.

En el interior de la carpeta *"imp"* se definen las operaciones del recurso que requieren comunicación con el agente y por tanto el envío de eventos, además de posibles pantallas de error.

La clase generadora se encarga de implementar las operaciones que se refieren a las ventanas o paneles que tiene este recurso.

b) Recurso de persistencia

El recurso de persistencia es el encargado almacenar los elementos computacionales de la aplicación. Concretamente, es el encargado de la lectura y escritura de diversos archivos XML que representan robots, victimas y obstáculos. Además de esto, también se encarga de almacenar la información relativa a escenarios, robots, victimas y obstáculos.

Descripción de la organización.

Agentes, Recursos y Modelo de información constituyen los elementos básicos para construir aplicaciones, pero para que la aplicación funcione utilizando ICARO es necesario definir explícitamente una Organización. La organización que implementa el sistema se define con un documento formal –expresado en XML- donde se especifican las propiedades y los componentes de la organización, los roles, atributos, características y dependencias de los componentes. Este documento se utiliza para crear la organización, es decir para crear los componentes computacionales que deben llevar a cabo la funcionalidad del sistema.

```xml
<!--*****************************************************************************************
******************     Propiedades globales de la organizacion                   ******************
*****************************************************************************************-->
<icaro:PropiedadesGlobales>
    <icaro:intervaloMonitorizacionGestores>800000</icaro:intervaloMonitorizacionGestores>
    <icaro:activarPanelTrazasDebug>true</icaro:activarPanelTrazasDebug>
      <icaro:listaPropiedades>
         <icaro:propiedad atributo="identificadorEquipo" valor = "Jerarquico" />
         <icaro:propiedad atributo="rutaFicheroEscenarioSimulacion" valor = "PersistenciaEscenariosSimulacion/modeloOrg_JerarquicoNumRobts_4NumVicts_9" />
         <icaro:propiedad atributo="timeMsTarea:TimeOutContactarMiembrosEquipo" valor = "1000" />
         <icaro:propiedad atributo="timeMsTarea:TimeOutEsquivarObstaculo" valor="2000" />
         <icaro:propiedad atributo="intervaloMilisegundosEnvioMensajesDesdeCC" valor = "10000" />   <!-- valor expresado en milisegundos -->
         <icaro:propiedad atributo="timeMsTarea:TimeOutRecibirEvaluaciones1" valor = "4000" />   <!-- se usa en el igualitario y en el jerarquico -->
         <icaro:propiedad atributo="timeMsTarea:TimeOutRecibirEvaluaciones2" valor = "1000" />
         <icaro:propiedad atributo="timeMsTarea:TimeOutRecibirConfirmacionesRealizacionObjetivo1" valor = "8000" />
         <icaro:propiedad atributo="timeMsTarea:TimeOutRecibirEstatusComunicacionConJefe" valor = "3000" />
         <icaro:propiedad atributo="timeMsTarea:TimeOutReaccionJefeAmsgeCC" valor = "3000" />
         <icaro:propiedad atributo="timeMsTarea:TimeOutRespuestaJefe" valor = "3000" />
         <icaro:propiedad atributo="timeMsTarea:TimeOutRecibirRespuestasEquipo" valor = "10000" />
         <icaro:propiedad atributo="timeTimeoutPorDefecto" valor = "10000" />
      </icaro:listaPropiedades>
</icaro:PropiedadesGlobales>

<icaro:DescripcionComponentes>

<icaro:DescComportamientoAgentes>
   <!--*****************************************************************************************
   ******************  Descripcion del comportamiento de los gestores                 ******************
   *****************************************************************************************-->
<icaro:DescComportamientoGestores>
      <icaro:DescComportamientoAgente
         nombreComportamiento="GestorOrganizacion" rol="Gestor" tipo="Reactivo" />
      <icaro:DescComportamientoAgente
         nombreComportamiento="GestorAgentes" rol="Gestor" tipo="Reactivo" />
      <icaro:DescComportamientoAgente
         nombreComportamiento="GestorRecursos" rol="Gestor" tipo="Reactivo" />
</icaro:DescComportamientoGestores>
   <!--*****************************************************************************************
   ******************   Descripcion del comportamiento de los agentes de aplicacion       ******************
   *****************************************************************************************-->
<icaro:DescComportamientoAgentesAplicacion>
   <icaro:DescComportamientoAgente
     nombreComportamiento="AgenteAplicacionAgteControladorSimuladorRosace" rol="AgenteAplicacion" localizacionComportamiento="icaro.aplicaciones.agentes.agenteAplicacionAgteControladorSimuladorRosaceReactivo.comportamiento"
                    tipo="Reactivo" />
   <icaro:DescComportamientoAgente
     nombreComportamiento="AgenteAplicacionAsignadorTareas" rol="AgenteAplicacion" localizacionComportamiento="icaro.aplicaciones.agentes.agenteAplicacionAsignadorTareasCognitivo" tipo="Cognitivo"
     localizacionFicheroReglas="icaro.aplicaciones.agentes.agenteAplicacionAsignadorTareasCognitivo.procesoResolucionObjetivos.reglasJerarqJefeCambioRol.drl"/>
    <icaro:DescComportamientoAgente
     nombreComportamiento="AgenteAplicacionSubordinado" rol="AgenteAplicacion" localizacionComportamiento="icaro.aplicaciones.agentes.agenteAplicacionSubordinadoConCambioRolCognitivo" tipo="Cognitivo"
     localizacionFicheroReglas="icaro.aplicaciones.agentes.agenteAplicacionSubordinadoConCambioRolCognitivo.procesoResolucionObjetivos.reglasSubordinadoReconocedor.drl"/>
</icaro:DescComportamientoAgentesAplicacion>
</icaro:DescComportamientoAgentes>


   <!--*****************************************************************************************
   ******************   Descripcion de los recursos de aplicacion                    ******************
   *****************************************************************************************-->
<icaro:DescRecursosAplicacion>
   <icaro:DescRecursoAplicacion nombre="RecursoEstadistica" localizacionClaseGeneradora="icaro.aplicaciones.recursos.recursoEstadistica.imp.ClaseGeneradoraRecursoEstadistica"/>
      <icaro:DescRecursoAplicacion nombre="RecursoVisualizadorEntornosSimulacion" localizacionClaseGeneradora="icaro.aplicaciones.recursos.recursoVisualizadorEntornosSimulacion.imp.ClaseGeneradoraRecursoVisualizadorEntornosSimulacion"/>
   <icaro:DescRecursoAplicacion nombre="RecursoPersistenciaEntornosSimulacion" localizacionClaseGeneradora="icaro.aplicaciones.recursos.recursoPersistenciaEntornosSimulacion.imp.ClaseGeneradoraRecursoPersistenciaEntornosSimulacion"/>
   <icaro:DescRecursoAplicacion nombre="RecursoCreacionEntornosSimulacion" localizacionClaseGeneradora="icaro.aplicaciones.recursos.recursoCreacionEntornosSimulacion.imp.ClaseGeneradoraRecursoCreacionEntornosSimulacion"/>
</icaro:DescRecursosAplicacion>

</icaro:DescripcionComponentes>

<icaro:DescInstancias>
   <!--*****************************************************************************************
   ******************   Instancias de gestores                                       ******************
   *****************************************************************************************-->
         <icaro:Gestores>
            <icaro:InstanciaGestor id="GestorOrganizacion" refDescripcion="GestorOrganizacion">
               <icaro:componentesGestionados>
                     <icaro:componenteGestionado refId="GestorAgentes" tipoComponente="Gestor"/>
                     <icaro:componenteGestionado refId="GestorRecursos" tipoComponente="Gestor"/>
               </icaro:componentesGestionados>
            </icaro:InstanciaGestor>
            <icaro:InstanciaGestor id="GestorAgentes" refDescripcion="GestorAgentes">
               <icaro:componentesGestionados>
                     <icaro:componenteGestionado refId="AgenteControladorSimuladorRosaceReactivo1" tipoComponente="AgenteAplicacion"/>
                     <icaro:componenteGestionado refId="JerarquicoagenteAsignador" tipoComponente="AgenteAplicacion"/>
                     <icaro:componenteGestionado refId="JerarquicorobotSubordinado1" tipoComponente="AgenteAplicacion"/>
                      <icaro:componenteGestionado refId="JerarquicorobotSubordinado2" tipoComponente="AgenteAplicacion"/>
                     <icaro:componenteGestionado refId="JerarquicorobotSubordinado3" tipoComponente="AgenteAplicacion"/>
                     <icaro:componenteGestionado refId="JerarquicorobotSubordinado4" tipoComponente="AgenteAplicacion"/>
               </icaro:componentesGestionados>
            </icaro:InstanciaGestor>
            <icaro:InstanciaGestor id="GestorRecursos" refDescripcion="GestorRecursos">
               <icaro:componentesGestionados>
                        <icaro:componenteGestionado refId="RecursoEstadistica1" tipoComponente="RecursoAplicacion"/>
                        <icaro:componenteGestionado refId="RecursoVisualizadorEntornosSimulacion1" tipoComponente="RecursoAplicacion"/>
                        <icaro:componenteGestionado refId="RecursoPersistenciaEntornosSimulacion1" tipoComponente="RecursoAplicacion"/>
                        <icaro:componenteGestionado refId="RecursoCreacionEntornosSimulacion1" tipoComponente="RecursoAplicacion"/>
               </icaro:componentesGestionados>
            </icaro:InstanciaGestor>
         </icaro:Gestores>


   <!--*****************************************************************************************
   ******************   Instancias de Agentes de aplicacion                          ******************
   *****************************************************************************************-->
      <icaro:AgentesAplicacion>
            <icaro:Instancia id="AgenteControladorSimuladorRosaceReactivo1" refDescripcion="AgenteAplicacionAgteControladorSimuladorRosace">
            </icaro:Instancia>
            <icaro:Instancia id="JerarquicoagenteAsignador" refDescripcion="AgenteAplicacionAsignadorTareas">
            </icaro:Instancia>
           <icaro:Instancia id="JerarquicorobotSubordinado1" refDescripcion="AgenteAplicacionSubordinado">
            </icaro:Instancia>
            <icaro:Instancia id="JerarquicorobotSubordinado2" refDescripcion="AgenteAplicacionSubordinado">
            </icaro:Instancia>
            <icaro:Instancia id="JerarquicorobotSubordinado3" refDescripcion="AgenteAplicacionSubordinado">
            </icaro:Instancia>
            <icaro:Instancia id="JerarquicorobotSubordinado4" refDescripcion="AgenteAplicacionSubordinado">
            </icaro:Instancia>
      </icaro:AgentesAplicacion>
   <!--*****************************************************************************************
   ******************   Instancias de Recursos de aplicacion                         ******************
   *****************************************************************************************-->
      <icaro:RecursosAplicacion>
            <icaro:Instancia id="RecursoEstadistica1" refDescripcion="RecursoEstadistica" xsi:type="icaro:Instancia"/>
            <icaro:Instancia id="RecursoVisualizadorEntornosSimulacion1" refDescripcion="RecursoVisualizadorEntornosSimulacion" xsi:type="icaro:Instancia"/>
            <icaro:Instancia id="RecursoPersistenciaEntornosSimulacion1" refDescripcion="RecursoPersistenciaEntornosSimulacion" xsi:type="icaro:Instancia"/>
            <icaro:Instancia id="RecursoCreacionEntornosSimulacion1" refDescripcion="RecursoCreacionEntornosSimulacion" xsi:type="icaro:Instancia"/>
      </icaro:RecursosAplicacion>

   <!--*****************************************************************************************
   ******************   Descripcion comun del nodo en el que se despliegan las instancias  ******************
   *****************************************************************************************-->
<icaro:nodoComun>
    <icaro:nombreUso>NodoPrincipal</icaro:nombreUso>
    <icaro:nombreCompletoHost>localhost</icaro:nombreCompletoHost>
</icaro:nodoComun>

</icaro:DescInstancias>
</icaro:DescOrganizacion>
```

Figura 32: Ejemplo de descripción de organización completa

La organización tiene una serie de propiedades computacionales globales aplicables al conjunto de sus componentes. Las que se especifican en el documento se refieren al intervalo de monitorización de los gestores, a la activación o no de las trazas para comprobar el comportamiento de los componentes. Pueden añadirse nuevos propiedades siguiendo el formato atributo valor.

A continuación se describen los Componentes de la organización. Se distinguen tres tipos de componente: Gestores de la Organización. Agentes de aplicación y recursos de aplicación.

La organización tiene tres gestores: El Gestor de la Organización, el Gestor de Agentes y el Gestor de Recursos. Los gestores son agentes cuyo rol en la organización esta predefinido, es decir deben dedicarse a gestionar otras entidades.

Para cada agente es necesario definir su comportamiento, es decir la información que va utilizarse para crear su modelo computacional. Los agentes de aplicación se describen de forma similar a los agentes gestores.

Los recursos se describen especificando el identificador del recurso y la localización de la clase que permite la creación del recurso. Si no se especifica la localización se buscará en la ruta por defecto.

Una vez definidos los componentes se especifican los - ejemplares de agentes y recursos – que implementarán la funcionalidad de la aplicación. Ya que puede haber varias instancias de cada agente o recurso.

3.2 USO DEL SISTEMA

3.2.1 GUÍA DE INSTALACIÓN

En primer lugar, es necesario descargar el proyecto, el cual está alojado en GitHub. El repositorio es el siguiente: Repositorio Github.

En el mismo repositorio se encuentra un README.me en el cual vienen las instrucciones para la correcta ejecución. A modo de resumen, es necesario importar el proyecto ya existente a Eclipse (que es el IDE utilizado), elegir como clase Main de ejecución “*ArranqueSistemaConCrtlGestorO”*, y poner como argumentos de ejecución alguna de las descripciones de organización existentes en la carpeta “*descripcionesOrganizacion*”. Si todo esto se realizó de forma correcta, ya sólo quedaría lanzar la simulación desde la ventana “*Centro de Control*”.

3.2.2 GUÍA DE USO

En cómo ejecutar el proyecto tanto para salvar una víctima, para salvar todas ellas o incluso para crear un escenario nuevo.

Lo primero es tener la interfaz de control, una vez se tiene delante podemos conseguir cualquiera cosa de las mencionadas arriba:

1. Salvar una víctima
   Para salvar una única víctima una vez se tenga la interfaz es necesario hacer doble clic en el nombre de la víctima del panel que se quiere salvar.

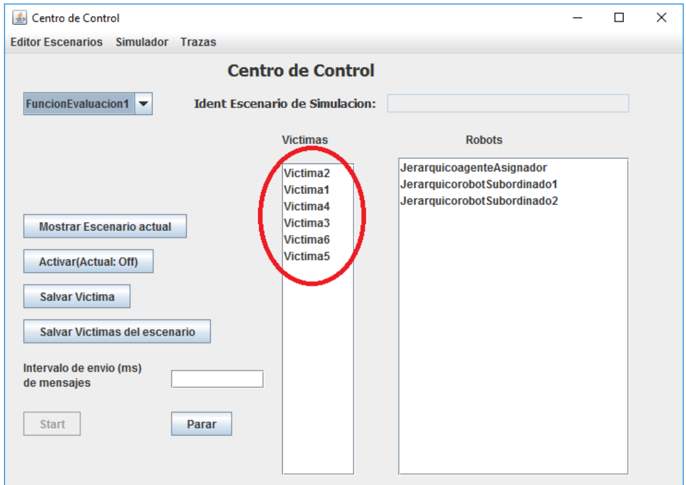

Figura 33: Paso I Salvar una víctima

   Una vez elegida, se debe pulsar en el botón de *salvar víctima*, acto seguido se mostrará el escenario con la simulación del salvamento de dicha víctima.

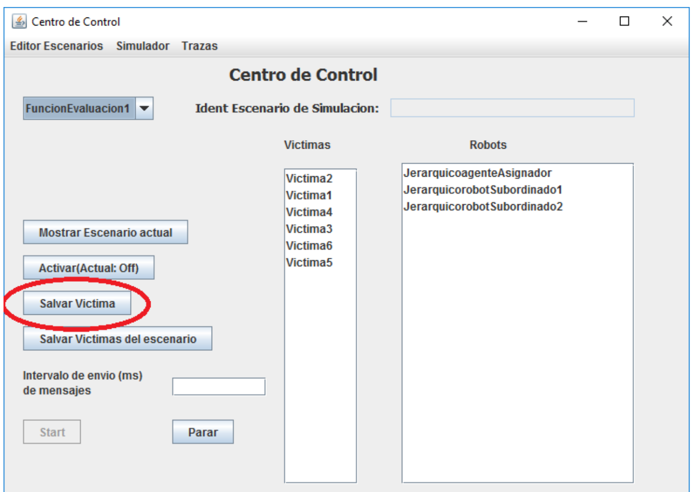

Figura 34: Paso II Salvar una víctima

2. Salvar todas las víctimas
   Para salvar a todas las víctimas del escenario primero se necesita pulsar en *salvar víctimas del escenario*.

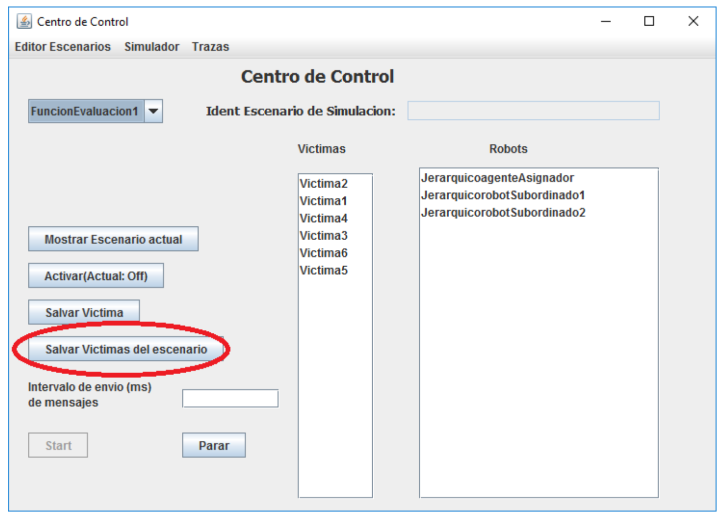

Figura 35: Paso I Salvar todas las víctimas

Esto provocará que aparezca un menú extra donde debemos decidir el tiempo que esperará el sistema entre el envío de víctimas. Este valor únicamente tiene efecto con la función de evaluación 3, que es la que no tiene robots exploradores y las víctimas las envía el centro de control.

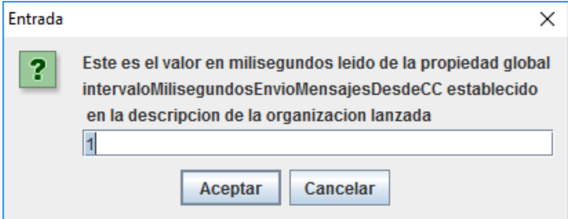

Figura 36: Paso II Salvar todas víctima

Una vez seleccionado el tiempo que queremos que espere y haber pulsado aceptar, volveremos a la interfaz de control donde debemos pulsar *Start* para que la simulación de comienzo.

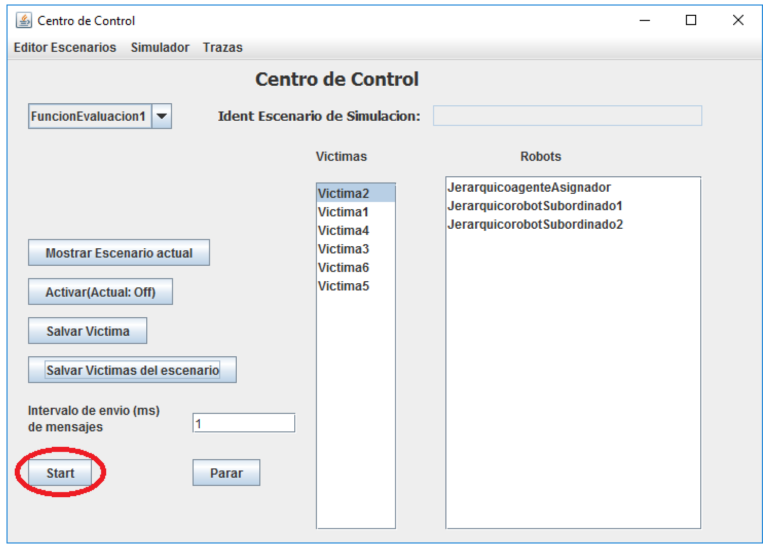

Figura 37: Paso III Salvar todas las víctimas

3. Creación/Edición de un escenario

   Para llegar a la interfaz de gestión de escenarios, hay que ir al menú superior llamado *Editor Escenario* y pulsar en *crear escenario*.

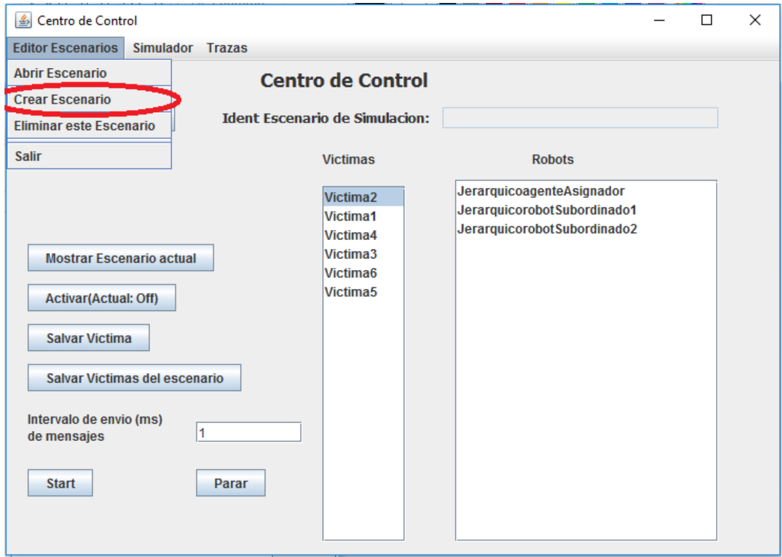

Figura 38: Paso I Crear/editar escenario

Una vez se haya pulsado, llegaremos a la interfaz de gestión de escenarios, aquí tendremos la opción de empezar un nuevo escenario o abrir un escenario ya hecho y editarlo. Para abrir un escenario ya cargado debemos ir al menú superior de nuevo llamado *Edición* y pulsar en *Abrir.*

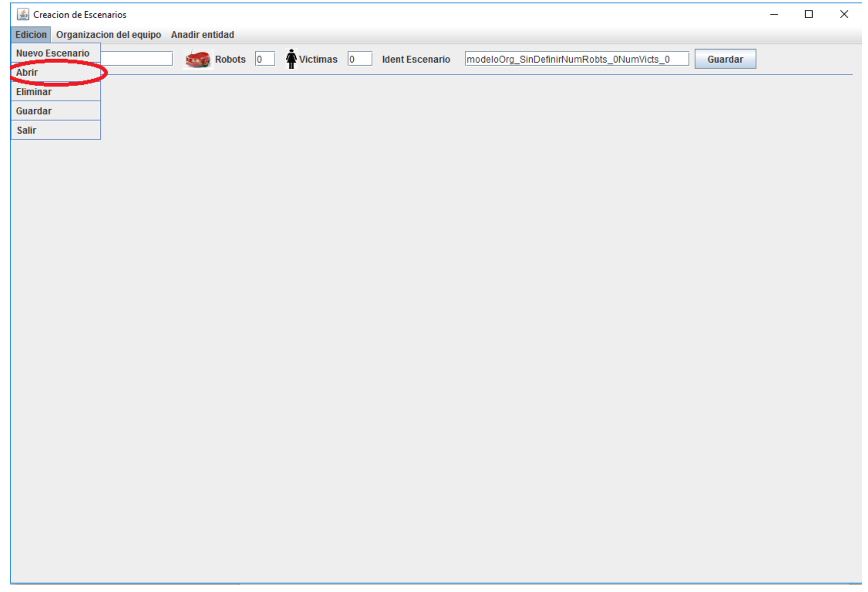

Figura 39: Paso II Crear/editar escenario

Se haya cargado uno o no, los pasos para añadir elementos al escenario son los mismos. Si se pulsa clic derecho sobre el escenario obtenemos un menú donde podemos elegir que añadir.

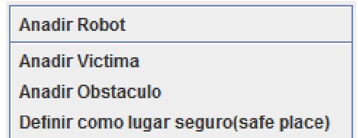

Figura 40: Paso III Crear/editar escenario

Si se decide *añadir un robot o víctima*, en la posición donde se diera ese clic derecho aparecerá el elemento seleccionado. Si después se desea añadir otro elemento del mismo tipo se podría hacerlo realizando doble clic en el punto donde se quiere que aparezca.
Si se pulsa en *añadir obstáculo* lo que el sistema guarda es el punto donde se realiza clic derecho y este será una esquina del obstáculo, después se necesita realizar clic izquierdo en alguna parte del mapa para delimitar la otra esquina del obstáculo y de esta forma se creará la línea (el sistema no acepta obstáculos inclinados por lo que automáticamente lo convertirá en obstáculo vertical u horizontal dependiendo de que esté más cerca).
Por último, si se pulsa en *Definir como lugar seguro* el sistema cogería el punto donde se realiza clic derecho y lo almacenaría como el lugar seguro, siendo este punto donde los agentes llevarán a las víctimas.

Una vez se haya hecho todos los cambios que se quieran, se necesita guardar el escenario, para ello se pulsa en el botón *Guardar* en la parte superior derecha.

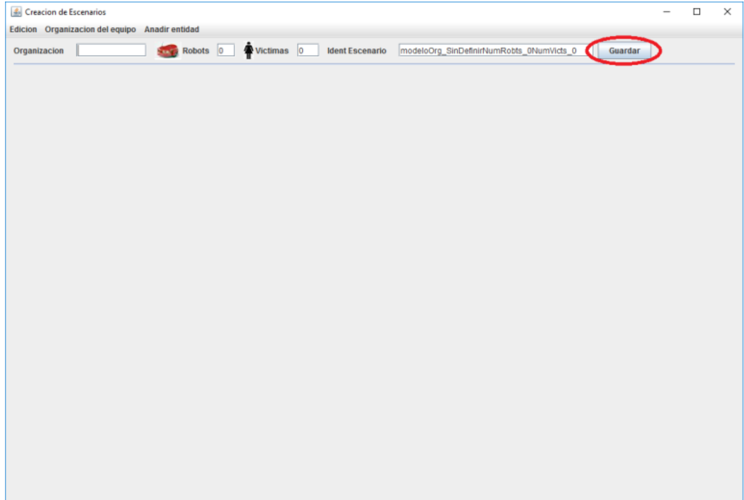

Figura 41: Paso IV Crear/editar escenario

4. Abrir un escenario

   Si se quiere abrir otro escenario distinto para la ejecución de una simulación, se necesita ir al menú superior izquierdo llamado *Editor escenario* desde la interfaz de control, una vez ahí se pulsa en *Abrir escenario* y mostrará un menú para la búsqueda del archivo.

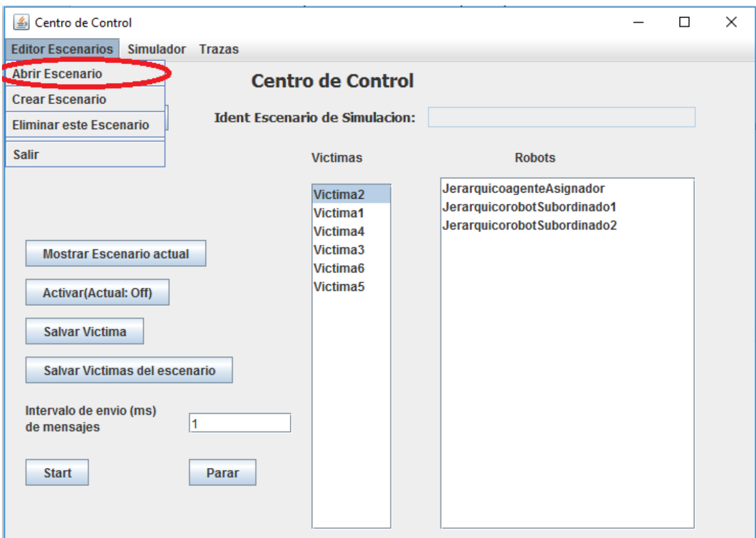

Figura 42: Paso I Abrir escenario

Estas son las opciones más usadas en el sistema y con las que más es necesario familiarizarse si se quiere utilizar.

3.4 PAUTAS DE CODIFICACIÓN

Para mantener un orden y permitir una comprensión clara del código se han seguido unas pautas de codificación. Este aspecto es muy importante ya que facilita el trabajo en equipo. No basta con dividir las tareas entre los integrantes del grupo, sino que hay que mantener un estándar en los nombramientos de clases, tareas, métodos y atributos.

Varias de las pautas establecidas se han heredado de ICARO con el fin de hacer el código lo más parecido al estile de la infraestructura. De esta manera cualquiera que conozca ICARO, podrá entender rápidamente lo que hace un agente y sus recursos. Otras son pautas ya generalizadas entre los programadores de Java que, aunque no son oficiales, se podría calificar como "buenas practicas".

A continuación, están listadas las pautas de codificación establecidas para desarrollar este proyecto, divididas en categorías:

- Documentación:
    - Documentación: al crear nuevas clases, por ejemplo, se ha hecho una breve descripción de su uso y sus métodos para que sea más sencillo de entender a cualquiera que las vea. Además de todo esto, puesto que el proyecto es visible en el repositorio de *Github*, en cada *commit* realizado se ha hecho una breve descripción de los cambios llevados a cabo de forma que, de forma rápida, se puede echar un vistazo en el *commit* y entender qué se hizo.
    - Estructura: Se ha estructurado cada agente en paquetes para separar las distintas partes del agente. Esta estructura se ha heredado de los ejemplos de aplicaciones que vienen en ICARO y a partir de ahí todos los agentes realizados se han implementado siguiendo la misma estructura para conseguir la homogeneidad. Más abajo se explica en detalle la estructura del código de los agentes.
- Nombrado:
    - Cada una de las ventanas de la interfaz está separada en una clase llamada PanelX.java. Por ejemplo: PanelVisor.java
    - Las Clases De Dominio usadas para enviar la información entre agentes se llaman InfoX.java. Por ejemplo: InfoDecision.java
    - La clase con la que se comunica la interfaz para enviar eventos a los agentes se llama UsoAgenteX.java. Por ejemplo: UsoAgenteAgenteControladorSimuladorRosace.java

3.5 ORGANIZACIÓN DEL CÓDIGO

El código de la aplicación es básicamente un conjunto de agentes que se comunican entre sí. Cada agente es independiente, por lo que se ha definido una estructura basada en paquetes que debe tener cada uno de los agentes y sus subclases y ficheros. Esto hace que la creación de nuevos agentes sea más rápida y que cualquier programador pueda entender fácilmente el funcionamiento de un agente que ha programado otra persona. Por un lado, están los

recursos, por otro están las clases de dominio y aparte están las clases del agente en sí. A continuación, se puede ver más gráficamente la estructura:

```
icaro.aplicaciones.agentes.agenteAplicacionAgteControladorSimuladorRosaceReactivo.comportamiento
    AccionesSemanticasAgenteAplicacionAgteControladorSimuladorRosace.java
    automata.xml
icaro.aplicaciones.agentes.agenteAplicacionAsignadorTareasCognitivo.informacion
    InfoParaDecidirAQuienAsignarObjetivo.java
icaro.aplicaciones.agentes.agenteAplicacionAsignadorTareasCognitivo.objetivos
    DecidirQuienVa.java
    ReconocerTerreno.java
    TerminarSimulacion.java
icaro.aplicaciones.agentes.agenteAplicacionAsignadorTareasCognitivo.procesoResolucionObjetivos
    reglas.drl
    reglasJerarqJefeCambioRol.drl
icaro.aplicaciones.agentes.agenteAplicacionAsignadorTareasCognitivo.tareas
    ComprobarSimulacionFinalizada.java
    EliminarObjetivoyDecisionInfoActualizarFoco.java
    FinalizarSimulacion.java
    InicializarInfoParaDecidir.java
    InterpretarOrdenCCReconocerTerreno.java
    InterpretarOrdenDelCC.java
    MandarPropuestaAlMejorParaQueRealiceObjetivo.java
    MandarPropuestaReconocimientoAlMejorParaQueRealiceObjetivo.java
    PedirEvalAtodos.java
    PedirEvalReconocerAtodos.java
    PedirEvaluacionesQueFaltan.java
    PedirEvaluacionesQueFaltanParaExplorar.java
    ProcesarEvaluacionRecibida.java
    TareaActualizarFocoReconocedorTerreno.java
```

Figura 43: Estructura de las clases del AgenteJerarquicoAsignador.

3.6 INTERFAZ

La interfaz se ha implementado usando una librería llamada SWT creada por los desarrolladores de Eclipse. Es una librería grafica basada en widgets, muy versátil, fácil de usar y multiplataforma. Las interfaces gráficas se han implementado basándonos en el Modelo Vista-Controlador(MVC), en el que el controlador notifica a la vista implementada por SWT los cambios que han ocurrido.

Concretamente, hay dos interfaces: escenario y centro de control.

Por una parte, la interfaz del escenario es el principal componente visual de la aplicación puesto que es ahí donde se comprueba cómo ha ido el rescate.

Inicialmente, la interfaz del escenario era simplemente una ventana, con un componente JPanel incrustado en el cual se pintaban iconos, que representaban a robots y víctimas. Los iconos que representaban a los robots se movían por el panel simulando el movimiento del robot (obviamente, en concordancia con la lógica interna del robot).

Ahora bien, cuando se planteó el objetivo de implementar obstáculos, la primera duda fue cómo representarlos. Este hecho planteó un problema que se solucionó de la siguiente forma: se creó una nueva clase llamada JPanelObstaculo, que extendía a JPanel y en la cual se contenían los obstáculos, definidos como líneas, y se sobrescribió el método interno de la clase JPanel llamado “Paint”, donde a partir de ahora se pintarían líneas de color negro en el fondo del escenario.

```
package icaro.aplicaciones.recursos.recursoVisualizadorEntornosSimulacion.imp;

import java.awt.Graphics;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import java.util.Map;

import javax.swing.JPanel;

import icaro.aplicaciones.Rosace.informacion.Coordinate;

public class JPanelObstaculo extends JPanel{

    private ArrayList<LineaObstaculo> obstaculos;
    private ArrayList<Coordinate> rastroExploracion;
    public JPanelObstaculo(ArrayList<LineaObstaculo> obs){
        this.obstaculos = obs;
        rastroExploracion = new ArrayList<Coordinate>();
    }
    public synchronized void paint(Graphics g){
        super.paint(g);
        for(LineaObstaculo obs : obstaculos){
            g.drawLine((int)obs.getCoordenadaIni().getX(), (int)obs.getCoordenadaIni().getY()
            , (int)obs.getCoordenadaFin().getX(), (int)obs.getCoordenadaFin().getY());
        }
        for(Coordinate coor : rastroExploracion){
            g.drawRect((int)coor.getX(), (int)coor.getY(), 1, 1);
        }

    }

    public synchronized void addCoordRastro(Coordinate coord){
        this.rastroExploracion.add(coord);
    }

}
```

Figura 44: Implementación del nuevo panel del escenario.

Según se fue avanzando en el proyecto añadiendo nuevas funcionalidades, la interfaz del escenario adquirió algunos nuevos componentes como botones para parar/romper robots o la funcionalidad de pintar el rastro seguido por un robot explorador.

Por último, está la interfaz del centro de control, que está implementada metódicamente con botones colocados de forma ordenada. Desde ésta interfaz se puede elegir el tipo de simulación, elegir salvar a una víctima concreta o cambiar de escenario. Algunas de las clases utilizadas para esta interfaz son:

- **JButton y JTextField**: implementa un botón y un campo de escritura, respectivamente.

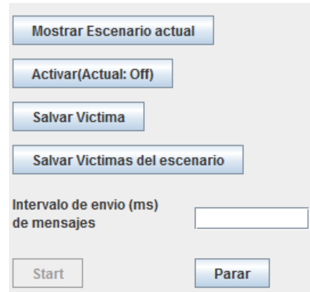

Figura 45: Ejemplo de botones.

- **JComboBox**: despliega una lista de elecciones.

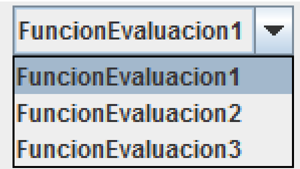

Figura 46: Elección de tipo de rescate

- **JList**: utilizada para la lista de robots y víctimas.

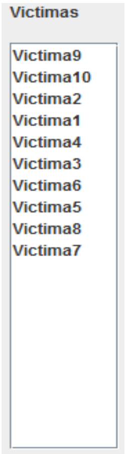

Figura 47: Ejemplo de lista con las víctimas

- **JTabbedPane**: cuando se quieren mostrar varios menús simultáneamente

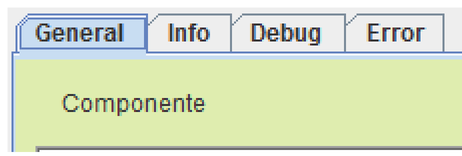

Figura 48: Ejemplo de "TabbedMenu" en la interfaz de trazas.

- **JMenu**: usado para mostrar menús superiores

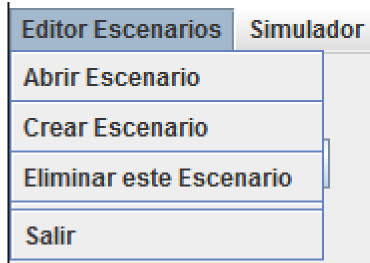

Figura 49: Ejemplo de "JMenu" en la interfaz de control.

- **JFileChooser**: Este componente se usa para buscar ficheros en el ordenador para cargarlos o directamente almacenar un fichero en un sitio.

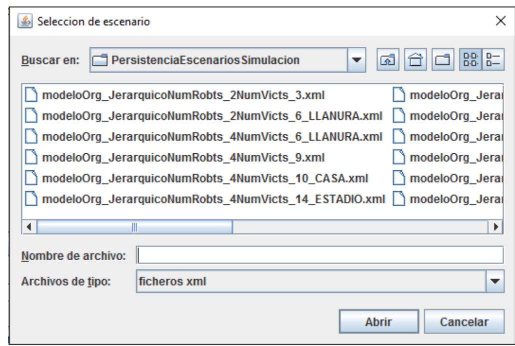

Figura 50: Ejemplo de "FileChooser" al cargar un escenario.

3.7 CONTRIBUCIÓN PERSONAL AL PROYECTO

3.7.1 LUIS GARCÍA TERRIZA

Aquí cito algunas de las partes más importantes de las que yo, personalmente, me he encargado de realizar son las siguientes:

CÁLCULO DE RUTAS

Al comenzar el proyecto, uno de los primeros objetivos que se planteó fue introducir obstáculos. Al introducir esta nueva característica a los escenarios resultó que los robots no respetaban los obstáculos porque la componente de movimiento que existía simplemente creaba coordenadas en función de la pendiente hacía el objetivo con una simple fórmula matemática sin tener nada más en cuenta. Por esto, se decidió crear un nuevo componente de

movimiento que tenga en cuenta tanto la energía disponible del robot como los lugares por los que se podía mover o no en función de la existencia de un obstáculo.

Primero, se creó un algoritmo de cálculo de rutas basado en un algoritmo típico de exploración de nodos en profundidad con heurística. El algoritmo calculaba la mejor solución pero no en un tiempo razonable para este proyecto, por lo que se decidió limitar el algoritmo a que devuelva la primera solución que encuentre y que lo haga en un número limitado de pasos. El algoritmo funcionaba relativamente bien porque la mayoría de las veces encontraba en un tiempo razonable una solución buena, pero cuando se introducía un escenario tipo laberinto el algoritmo no devolvía solución por el límite de pasos.

Es por esto mismo por lo que mucho más adelante se planteó la posibilidad de implementar otro algoritmo de rutas.

La segunda versión y final del algoritmo de rutas utilizado está basado en el algoritmo de Lee, que básicamente es una exploración de nodos en anchura numerando las coordenadas con un número. Esta nueva versión del algoritmo de rutas garantiza la mejor solución si existe y además lo realiza en un tiempo razonablemente bajo, por lo que finalmente se utilizó esta versión.

SOLUCIÓN A LA PARALELIZACIÓN

Durante las simulaciones había un problema detectado: cuando un robot se estaba moviendo hacia una víctima para rescatarla y tenía que saltar una nueva regla, no funcionaban las reglas. Este problema dio muchos problemas ya que si no funcionan los robots mientras se mueven no se obtienen buenos resultados. Tras muchas horas investigando y ejecutando el proyecto en modo debug, se descubrió que la forma en que se lanzaba el movimiento no era adecuada en lo que a la concurrencia se refiere. Resulta que la hebra del motor de reglas ejecutaba la tarea de moverse hacia la víctima y cuando se llamaba a la hebra de movimiento para moverse hacia la víctima, no se llamaba de forma concurrente lanzando la hebra sino que se ejecutaba la hebra desde la hebra del motor de reglas, llevando a que mientras se mueve, el motor de reglas está ocupado realizando los cálculos del movimiento.

RESCATE REAL

Esta es una de las nuevas funcionalidades del rescate. Se decidió que en vez de acercarse a la víctima en cuestión y, al llegar, dejarla ahí como rescatada, era mejor cargarla y moverla hasta un punto de rescate, acercándose ligeramente a lo que en un rescate real se haría.

Para llevar a cabo esta funcionalidad se tenía que añadir un nuevo proceso: al llegar a la víctima, se tenía que cambiar el destino de retorno hacia el lugar seguro modificando la posición de la víctima a su vez. Esto era algo trivial, pero introducía un nuevo componente al escenario: el lugar seguro. Básicamente es un punto en el mapa adonde habría que llevar a las

víctimas para salvarlas. Simula ser lo que en un rescate real es un campamento o un hospital temporal establecido en una zona.

El lugar seguro es, en cuanto a implementación se refiere, una coordenada. Esa coordenada está definida en función del escenario, por lo que había que cambiar la forma de cargar escenarios y también la forma de creación de escenarios. Así pues, se realizaron cambios en la interfaz de creación de escenarios para poder especificar un lugar seguro, además de arreglar los escenarios ya existentes, que sin la definición del lugar seguro causaban error al cargarse.

RESULTADOS SIMULACIONES

Para obtener los resultados de las simulaciones había que conseguir que el sistema inicie sólo, empiece la simulación y devuelva de alguna forma los resultados de la simulación. Para esto se creó un nuevo main que arrancase el sistema a modo script, sin mostrar ventanas de trazas y que escriba los resultados de la simulación en un fichero. Esto se realizó creando un nuevo objetivo, llamado simulación, en el que se escribía toda la información relativa al rescate de víctimas. Esta información era el tipo de rescate, el tipo de escenario, el número de víctimas, el número de robots, la identificación de cada víctima, el momento (tiempo relativo) de asignación, el momento (tiempo relativo) de resolución, resolución(viva o muerta) y qué robot fue el encargado de salvar a la víctima.

Una vez funcionaba la parte del sistema como un script, se creó una demo en formato .jar era fácilmente ejecutable desde un script en batch. Así se pudieron realizar más de cien simulaciones distintas de forma automática.

Finalmente, cuando se tenían los resultados de las simulaciones, se creó un script en lenguaje MatLab que leyese todos los datos de las simulaciones, los almacenase y realizase cálculos con ellos para crear gráficas. Este script en MatLab se programó de forma que, independientemente del número de simulaciones, crease los mismos tipos de gráficas y las guardase automáticamente en formato JPG.

3.7.2 SERGIO MORENO DE PRADAS

Personalmente me he encargado de cuatro tareas:

- Robots exploradores
- Obstáculos del escenario
- Muerte de víctimas
- Los robots se paran

A continuación procederé a explicarlas detenidamente.

ROBOTS EXPLORADORES

Al comenzar el proyecto las víctimas se enviaban por el centro de control, de forma que ya se conocían donde estaban las víctimas y únicamente se le enviaban al agente asignador para que decidiera a quien asignar el objetivo de salvarla. La idea de tener ya información sobre todas

las víctimas del escenario con su posición en el mismo no parecía que fuera muy realista y que se pudiera aplicar a una misión real con robots de verdad ya que normalmente en una misión de rescate únicamente se tiene información sobre el terreno o el mapa, pero se desconoce la cantidad de víctimas o la situación de estas. Por esta razón decidimos crear un objetivo que fuera reconocer el terreno y que se le asignara a un agente el cumplimiento del mismo, de esta forma habría un agente que decidiría en tiempo de ejecución el agente asignador que se pondría a explorar el mapa punto a punto y con cada víctima encontrada se lo haría saber al agente asignador para que este realizara la decisión de a que agente enviar a salvarla. Esta forma parece la más indicada de actuar ante un rescate, habrá un agente que será el que explore todo el mapa cerciorándose de que ha encontrado todas las víctimas y avisando de todas las que encuentre.

OBSTÁCULOS DEL ESCENARIO

Los obstáculos en el escenario se crearon para simular la posibilidad de paredes o escombros en un escenario de rescate, de esta forma no se tiene únicamente un escenario llano donde hay X robots e Y víctimas, se tiene también unos obstáculos que limitan y dificultan el movimiento al agente y por consiguiente el salvamento de las víctimas. Es una manera muy interesante de ampliar el estudio de las organizaciones de equipos, además se puede crear escenarios más complejos y más cercanos a la realidad.

Un ejemplo de estos obstáculos lo vemos en el tipo de escenarios Casa donde con obstáculos se ha simulado una casa con víctimas tanto dentro como fuera de esta. Los obstáculos están creados usando un componente de Java llamado Line, lo que se hizo fue crear un JPanel propio con la información de los obstáculos y se redefinió el método Paint añadiendo el dibujo de estas líneas que representarían los obstáculos, luego se añadió este tipo de panel al visor de escenarios; de esta forma tenemos dibujado el escenario antiguo sumado al dibujo de los obstáculos.

MUERTE DE VÍCTIMAS

Otra de las decisiones que se tomaron para aumentar los datos del análisis de la misión de salvamento fue añadir un tiempo de vida a las víctimas y de esta forma dar la posibilidad de que si una víctima no era salvada en un tiempo X, moriría.

Esta idea se decidió porque una parte importante del proyecto era el querer añadirle realismo, ya sea por obstáculos en el terreno o por la posibilidad de que un robot se rompa se quería que la simulación del rescate fuera lo más realista posible y alejándose de la “perfección” de una simulación donde las víctimas no mueren, el terreno es llano y los robots tienen energía ilimitada. Así se definió la posibilidad de dar a cada víctima un tiempo donde esta estaría viva, el tiempo se define en el XML del escenario y cuando la víctima se descubre o por el sistema (si no hay exploradores) o por el robot explorador se lanza una hebra que contará tantos milisegundos como tiempo de vida tenga la víctima, tras ese tiempo la víctima enviará un mensaje al agente asignador de que ha muerto y este la actualizará en el mapa y en la información que se disponga de esta.

Un aspecto importante que se tiene en el proyecto y que creemos que es interesante es que si hay un agente que se dirige a una víctima para salvarla o la está salvando y muere en ese momento, el agente irá automáticamente a otra víctima si tuviera alguna asignada, de esta forma se consigue priorizar a las víctimas vivas y no se realizará el salvamente de víctimas que ya han fallecido.

LOS ROBOTS SE PARAN

De la misma manera que con la muerte de las víctimas, lo que se buscaba con la posibilidad de que los robots se pararan era aumentar el realismo y no tener una simulación utópica donde no ocurriera nada imprevisto y todo fuera según lo planeado.

Los robots se pueden parar por dos razones:

1 El robot se rompe
2 El robot se queda sin energía

La primera razón se tiene cuando un robot por una razón u otra se estropea en el curso de la misión, ya sea porque se ha chocado contra algo, se ha caído en un agujero o ha tenido un problema interno. Esto se simula con un botón situado en la interfaz de misión donde se puede parar a cualquier robot del equipo en cualquier momento de forma que el sistema tiene que responder ante este evento. Algo muy interesante es ver que si un robot se está dirigiendo a una víctima A teniendo más víctimas asignadas para su posterior salvamento, si se para dicho robot el sistema las reasigna automáticamente entre los robots que queden disponibles en el sistema además siempre asignándose al óptimo.

La otra razón para parase es que el robot se quede sin energía, esto puede ocurrir porque el robot tiene una energía limitada para la misión y con cada movimiento gasta una parte de esta, cuando el robot se queda sin energía para moverse automáticamente se para y le envía un mensaje al asignador indicándole que se ha quedado sin energía y se ha parado. Cuando esto ocurre, el asignador de forma automática vuelva a asignar las víctimas de este robot entre los demás de forma que el sistema no se atasca ante estos imprevistos y siempre sabe reaccionar acorde a la situación.

3.8 PROBLEMAS ENCONTRADOS Y SOLUCIÓN ADOPTADA

Al tratarse de la implementación sobre un software ya empezado gran parte de los problemas encontrados radicaron en entender la funcionalidad ya creada para poder modificarla para nuestras necesidades, además, al añadir estas nuevas funcionalidades se necesitaba cambiar partes ya existentes lo que no siempre funcionaba, algunos de los mayores fallos fueron:

- **La ejecución de las reglas de los agentes parece bloquearse** y tarda bastante en continuar con la ejecución normal lo que no se puede tener si se quiere un sistema de respuesta rápida ante escenarios. El error se producía cuando a un agente se le asignaban más de una víctima, lo que parecía bloquear al jefe hasta que dicho agente

terminase y no se continuaban asignando nuevas víctimas lo que retrasaba en gran medida la ejecución.

El error que se encontró fue que no se estaba realizando correctamente la concurrencia del sistema con lo que las reglas se bloqueaban. **La solución** que se encontró fue la implementación de un semáforo que, aunque primitivo controlaba el acceso al método *moverADestino* de la clase *RobotParado*.

- **Las víctimas muertas se salvaban**, lo cual no debería pasar ya que se tendría que priorizar las víctimas que están vivas para maximizar las que se han salvado.

  El error se encontró en la comprobación para enviar una víctima para su salvamento ya sea por el sistema o por explorador cuando este se la encuentra. **La solución** que se encontró fue añadir una comprobación antes de enviarla donde solo se envíe siempre que la víctima aún siga viva, de esta forma se soluciona el error y se elimina computación inservible de intentar salvar una víctima que ya ha fallecido.

- **Las víctimas se salvaban varias veces** por diferentes agentes, primero un agente salvaba a la víctima y la llevaba al lugar seguro para posteriormente otro agente ir al lugar donde se encontraba la víctima (ahora vacío) y salvarla, esto podía pasar como más de un agente.
  El error estaba relacionado a la solución que se dio a otro error que explicaremos a continuación que fue que algunas víctimas no se salvaban, la solución que se pensó fue enviar el mensaje de salvamento varias veces. Lo que ocasionó esto fue que Drools tuviera como hechos residuales varios mensajes de la misma víctima y tratara todos ellos, salvándola varias veces. **La solución** para quitarnos tanto este como el error relacionado fue la implementación de una cola de prioridad donde se almacenarán las víctimas cuando se envían al jefe, de esta forma se mantendría un control a la hora de consumir la víctima y además se aseguraría que la víctima se consume siempre.

- **Algunas víctimas no se salvaban**, se quedaban en el estado no descubierta, aunque se hubiera enviado el mensaje para salvarla ya sea por el sistema o por el robot explorador. La primera solución a este error fue la causante del error de arriba, por lo que este y el superior están relacionados.
  El error se encontró en que no se forzaba a Drools a ejecutar la regla asociada al salvamento de la víctima con lo que dejaba al hecho de ayudar a la víctima sin tratar y, en consecuencia, sin salvar. **La solución** fue la misma que el error de arriba, el implementar la cola de prioridad por lo que se aseguraría que todas y cada una de las víctimas se salvarían.

- **El sistema no podía ejecutar**, un problema que se tardó bastante tiempo en entender y solucionar. Esto ocasionó bastantes quebraderos de cabeza y sobretodo parón en

implementar funcionalidad en el sistema ya que no se podía ejecutar correctamente y se necesita solucionar este fallo.
Tras bastante tiempo analizando el problema se encontró **el error** que se resumía en conflicto de librerías con Java. La librería de Drools era la 5.5 teniendo la última versión de Drools en 6.4.0 en estos momentos, y Java estaba en la versión 1.8 (la final al tiempo de escribir esto). Se comprobó que al no estar Drools actualizado producía ese error en tiempo de ejecución con la versión de Java lo que imposibilitaba su ejecución.
Tras varios intentos se llegó a la conclusión de que **la solución** era usar Java 7 en JRE y JDK para solventarlo ya que actualizar la versión de Drools traía muchos problemas y ninguna garantía de que solventara todo.

- **La pila de java se consume.** Este error surgió con la implementación del primer algoritmo para el cálculo de rutas, con tantas llamadas recursivas lo que ocurría es que agotaba la pila de llamadas (recordemos que en cada pixel el agente llamaba a este algoritmo que comprobaba las 8 posiciones a su alrededor, siendo el mapa de 720 x 480 esto era un cómputo demasiado grande).
  El error ya se conocía, el algoritmo usaba demasiado cómputo al ejecutar únicamente para un agente, teniendo más de 4 por media en el proyecto se convertía en un problema bastante difícil de solventar manteniendo el algoritmo actual. **La solución** temporal que se pensó consistió en aumentar el tamaño de pila en eclipse para posibilitar la ejecución, más adelante se llegó a la conclusión de que el algoritmo no era lo suficientemente inteligente ni eficiente por lo que se cambió por el Algoritmo de Lee.

- **Los robots atraviesan los obstáculos** al pasar por ellos, al principio se pensó que era un problema al leer los datos de los obstáculos por los agentes, pero se comprobó usando la herramienta de "*debug*" de Eclipse que esto no era así.
  Posteriormente se llegó a la conclusión de que **el error** era que, al poder los agentes moverse en diagonal, eran capaces de saltar el obstáculo ya que la representación que Java daba a la línea que es el obstáculo no era la pensada. Cuando el agente llega al obstáculo y prueba a mover en diagonal descubre que si puede ya que el obstáculo tiene un píxel encima de otro marcando la línea pero es de grosor 1, lo que hace que si está totalmente pegado al obstáculo lo pueda saltar verticalmente ya que solo comprobaban que no sea obstáculo la coordenada donde se moverían.
  La solución que se encontró fue que el agente comprobara no solo comprobara la coordenada donde se movería si no que en cada movimiento comprobara cada coordenada que le rodea, de esta forma únicamente se moverá en una dirección si no tiene un obstáculo en dicha dirección.

4. EXPERIMENTACIÓN

En la parte de pruebas del proyecto se ha decidido realizar varios tipos de escenarios donde pueda variar el número de robots y víctimas, pero manteniendo la estructura del escenario intacta. Esto ha permitido definir una plantilla de escenario que simulen un caso donde habría que enviar a los agentes en una misión de rescate y así obtener datos qué se puedan usar para investigación y análisis de que estrategia de salvamento es mejor dependiendo del escenario usado. Por ejemplo, se podría comprobar qué compensa más, si tener más agentes exploradores y reducir el tiempo de exploración de terreno a costa de tener un robot de salvamento menos (al menos hasta que este termine las exploraciones) o tener menos exploradores y aumentar el tiempo de exploración, pero teniendo más robots de salvamento disponibles; este caso dependerá de cuantas víctimas haya en el escenario y de cuantos agentes se dispongan.

De este modo, cuando se han obtenido los datos y mostrados por gráficas, lo que se quiere estudiar es la forma en la que influyen en el sistema las variables descritas, como puede ser el número de víctimas, número de agentes, número de exploradores, obstáculos en el escenario y todo esto siempre teniendo en cuenta los imprevistos que pueden pasar en la misión que son, por ejemplo, que una víctima muera o que un robot se estropee. Habiendo realizado todas estas relaciones lo que se busca es obtener la configuración óptima de robots para salvar N víctimas dependiendo del escenario y cada configuración, además de averiguar cuál es el tipo de rescate más óptimo en cuanto a salvamento.

4.1 ESCENARIOS A USAR

En las plantillas de escenarios a usar se han decidido tres:

1. **Llanura**
   Este escenario simula un área de misión donde no hay obstáculos en el camino, por lo que los agentes se pueden mover de una forma más libre por el terreno y podrán ir de una forma más directa a las víctimas.

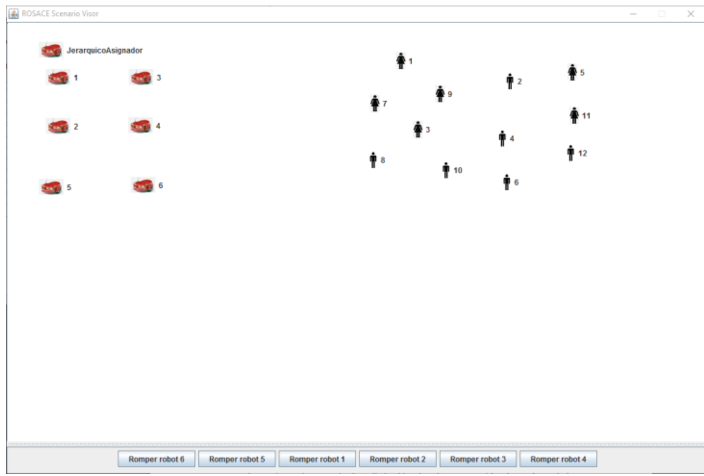

Figura 51: Tipo de escenario: Llanura.

2. **Casa**

Este escenario representa una misión para los agentes donde las víctimas están en una casa, es decir, hay obstáculos que tendrán que solventar para entrar en la casa y en las habitaciones de esta para comprobar si hay víctimas y si las hay salvarlas.

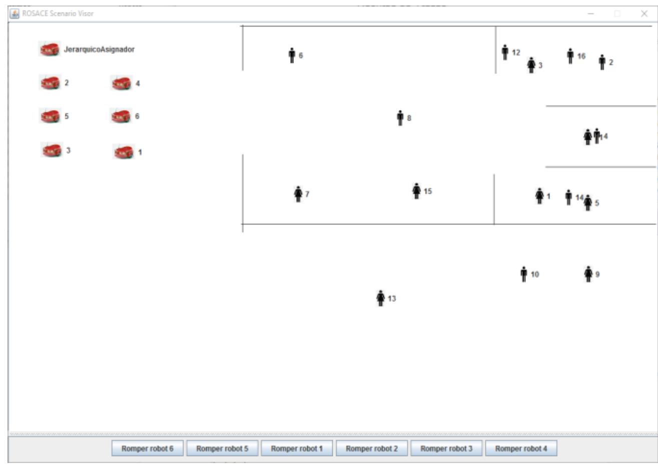

Figura 52: Tipo de escenario: Casa.

3. **Estadio**

Por último, se representa un área de misión de un estadio de fútbol, simulando las gradas donde estarían las víctimas dificultando al agente moverse por el escenario junto con una mayor aglomeración de víctimas para ver cómo se comporta el sistema. En este mapa será donde estén el mayor número de víctimas y agentes.

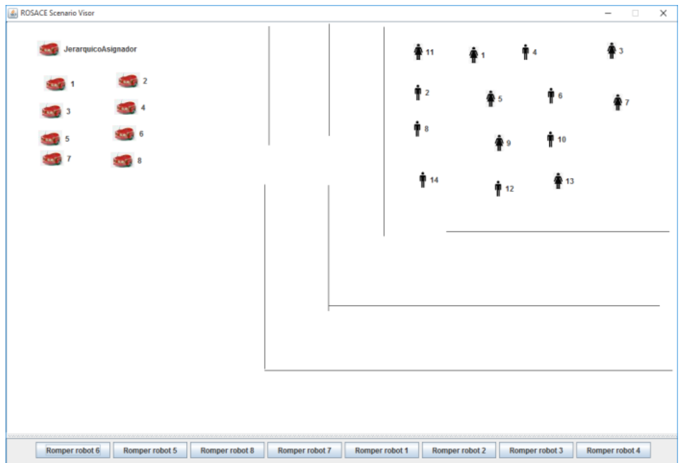

Figura 53: Tipo de escenario: Estadio.

4.2 TABLA DE PRUEBAS

Para los escenarios ya descritos se han realizado unas pruebas donde se modifica el número de agentes y el número de víctimas y además el número de exploradores. Para la definición de pruebas se ha creado una tabla donde se muestra cada prueba a realizar con el valor que tiene para cada variable.

Escenario	Nº robots	Nº Sanadores	Nº Exploradores	Nº Víctimas
Llanura	4	4	0	6
Llanura	4	3	1	6
Llanura	4	2	2	6
Llanura	6	6	0	12
Llanura	6	5	1	12
Llanura	6	4	2	12
Casa	4	4	0	10
Casa	4	3	1	10
Casa	4	2	2	10
Casa	6	6	0	16
Casa	6	5	1	16
Casa	6	4	2	16
Estadio	8	8	0	14
Estadio	8	7	1	14
Estadio	8	6	2	14
Estadio	8	8	0	20
Estadio	8	7	1	20
Estadio	8	6	2	20

Tabla 3: Plantilla de Pruebas

4.3 DATOS A OBTENER

Los datos recopilados de cada simulación son los siguientes:

- Tiempo que ha tardado desde que el programa se inició hasta que se asignó la víctima para comprobar la velocidad que existe en el envío del mensaje por parte del agente explorador (si hubiera) o centro de control y la recepción y gestión del mismo por parte del agente asignador además de la gestión del objetivo de ayudar a la víctima hasta que esta se asigna a un robot para su salvamento.

$$tAsigVictima = tAceptAyudarVictima + (tNotificacionVictima - tInicioSimulacion)$$

- Tiempo que ha tardado en salvar a la víctima (se considera que es salvarla cuando el agente lleva a la víctima al lugar seguro que se ha definido en el escenario) desde que la misma se le asigno por el agente asignador. Con esto se quiere conseguir el tiempo que tardaría un robot en salvar a una víctima y si fue la opción acertada.

$$tRescateVictima = tResolucion - tNotificacion$$

- Si la víctima se consiguió salvar o no para obtener las víctimas medias salvadas en el escenario y poder estudiar el impacto de los obstáculos en el mismo (se supone que el agente consumirá más tiempo en el salvamento si tiene que esquivar obstáculos que si puede ir directo a la víctima). El tiempo de vida de las víctimas será el mismo en todos los escenarios para poder mantener una relación entre los datos de los diferentes escenarios y poder enfrentarlos.

- El agente que se le asignó a la víctima para su salvamento. Con esto se quiere conseguir estudiar si la media de víctimas por robot es estable o no se distribuirían de forma uniforme entre los agentes y hubiera algunos que trabajan más que otros dependiendo del escenario en que se encuentren.

Todos estos datos se recogerán para cada prueba de la tabla de configuraciones mostrada arriba y serán los que se usen para mostrarlos mediante gráficas y estudiar el impacto que tiene cada variable en la misión.
Una vez obtenidos todos los resultados de simulaciones, mediante un script programado en *"MatLab"* se juntarán los resultados y se generarán las gráficas correspondientes. Las gráficas que se van a obtener son las siguientes:

- Porcentaje de víctimas salvadas
- Duración del salvamento
- Tiempo de salvamento por víctima de media

- Víctimas asignadas a robot de media

Todas estas gráficas se mostrarán por plantilla de escenario, diferenciando las distintas configuraciones de cada una (número de víctimas y número de agentes) y dependiendo del número de exploradores usados.

4.4 MÉTRICAS DE RESULTADOS

Con los resultados de las simulaciones se pretende realizar un análisis para obtener las conclusiones. Para esto se utilizan, entre otras, las siguientes métricas, que se describen de la siguiente forma:

- Porcentaje de víctimas salvadas.
- Duración del salvamento: tiempo total de salvamento.
- Tiempo de salvamento por víctima de media.
- Víctimas asignadas a robot de media.

Una vez explicado esto, mediante un script programado en **MatLab** se juntarán los resultados y se generarán las gráficas correspondientes.

```
for i=1:length(escenariosSeleccionados)
  if escenariosSeleccionados{i}{3} == 1
    if escenariosSeleccionados{i}{1} == nrobots1
      p14 = [p14 sum(escenariosSeleccionados{i}{5}(:,4))/escenariosSeleccionados{i}{2}];
    elseif escenariosSeleccionados{i}{1} == nrobots2
      p16 = [p16 sum(escenariosSeleccionados{i}{5}(:,4))/escenariosSeleccionados{i}{2}];
    elseif escenariosSeleccionados{i}{1} == nrobots3
      p18 = [p18 sum(escenariosSeleccionados{i}{5}(:,4))/escenariosSeleccionados{i}{2}];
    endif
  elseif escenariosSeleccionados{i}{3} == 2
   if escenariosSeleccionados{i}{1} == nrobots1
      p24 = [p24 sum(escenariosSeleccionados{i}{5}(:,4))/escenariosSeleccionados{i}{2}];
    elseif escenariosSeleccionados{i}{1} == nrobots2
      p26 = [p26 sum(escenariosSeleccionados{i}{5}(:,4))/escenariosSeleccionados{i}{2}];
    elseif escenariosSeleccionados{i}{1} == nrobots3
      p28 = [p28 sum(escenariosSeleccionados{i}{5}(:,4))/escenariosSeleccionados{i}{2}];
    endif
  elseif escenariosSeleccionados{i}{3} == 3
   if escenariosSeleccionados{i}{1} == nrobots1
      p34 = [p34 sum(escenariosSeleccionados{i}{5}(:,4))/escenariosSeleccionados{i}{2}];
    elseif escenariosSeleccionados{i}{1} == nrobots2
      p36 = [p36 sum(escenariosSeleccionados{i}{5}(:,4))/escenariosSeleccionados{i}{2}];
    elseif escenariosSeleccionados{i}{1} == nrobots3
      p38 = [p38 sum(escenariosSeleccionados{i}{5}(:,4))/escenariosSeleccionados{i}{2}];
    endif
  endif
endfor

pglobal1 = mean([mean(p14),mean(p24),mean(p34)]);
pglobal2 = mean([mean(p16),mean(p26),mean(p36)]);
pglobal3 = mean([mean(p18),mean(p28),mean(p38)]);
figure(numFigure);
plot(indices,[mean(p14) mean(p24) mean(p34) pglobal1], "b",indices,[mean(p16) mean(p26) mean(p36) pglobal2], "r",indices,[mean(p18) mean(p28) mean(p38) pglobal3], "g");
if tipoEscenario == 1
 nombreEscenario = 'CASA';
elseif tipoEscenario == 2
 nombreEscenario = 'LLANURA';
elseif tipoEscenario == 3
 nombreEscenario = 'ESTADIO';
endif
title(strcat('Porcentaje de salvamento de victimas ',nombreEscenario,'-',num2str(numVictimas),' Victimas'));
xlabel('Modos de rescate');
ylabel('% VS');
legend(strcat(num2str(nrobots1),' Robots'),strcat(num2str(nrobots2),' Robots'),strcat(num2str(nrobots3),' Robots'));
axis([1,5,0.5,1.01]);
set(gca,'xtick',indices);
set(gca,'xticklabel',{'1Explorador','2Exploradores','SinExploradores','Global'});
%clf();
%surf(peaks);
saveas(numFigure,strcat('PorcentajeSalvamento',nombreEscenario,num2str(numVictimas),'.jpg'));
```

Figura 54: Ejemplo creación gráfica porcentaje de salvamento

Todas estas gráficas se mostrarán por plantilla de escenario, diferenciando las distintas configuraciones de cada una (número de víctimas y número de agentes) y dependiendo del número de exploradores usados.

5. RESULTADOS Y CONCLUSIONES

5.1 ANÁLISIS DE RESULTADOS

Antes de comenzar, hay que mencionar que las gráficas representan varios datos, pero tienen todas en común que diferencian el tipo de rescate realizado en la simulación, siendo tres tipos de rescate los desarrollados y simulados:

- Rescate con 1 explorador
- Rescate con 2 exploradores
- Rescate inmediato sin exploradores: se conoce previamente la información relativa a las victimas (Tiempo de vida, localización…).

Para crear las gráficas que se van a visualizar a continuación se han realizado un total de 145 simulaciones, que han llevado un total de 3 horas de simulación.

5.1.1 ESCENARIOS TIPO LLANURA

Este escenario, tal y como se ha comentado ligeramente antes, se caracteriza por no tener obstáculos, facilitando el movimiento de los robots, y por tener menor cantidad de víctimas.

Para comenzar, se muestra una gráfica en la que se muestra el tiempo medio de salvamento de una víctima, **variando el número de robots** participantes en el rescate y, como en todas las demás, el tipo de rescate.

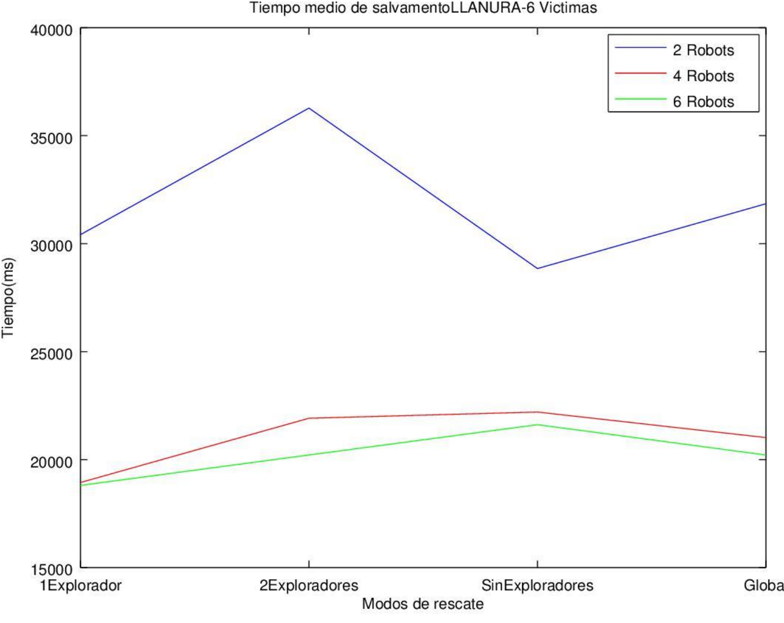

Figura 55: Tiempo medio de salvamento

Tal y como se puede ver, el tiempo medio de rescate de una víctima disminuye a medida que aumenta el número de robots, aunque es cierto que la diferencia entre 4 y 6 robots es mínima, lo que indica que la proporción de robots por víctima empieza a ser buena a partir de 1,5 víctimas por robot. De esta gráfica también se puede obtener otra conclusión evidente: el tipo de rescate influye en cómo de eficiente se es al rescatar a las víctimas, y se demuestra que, independientemente de cuántos robots haya, el mejor modo de rescate para este escenario sin obstáculos es el de exploración.

Con el fin de facilitar la comprensión de la simulación, se adjuntan más gráficas que muestran datos importantes como la relación victimas/robots o tiempo total de rescate.

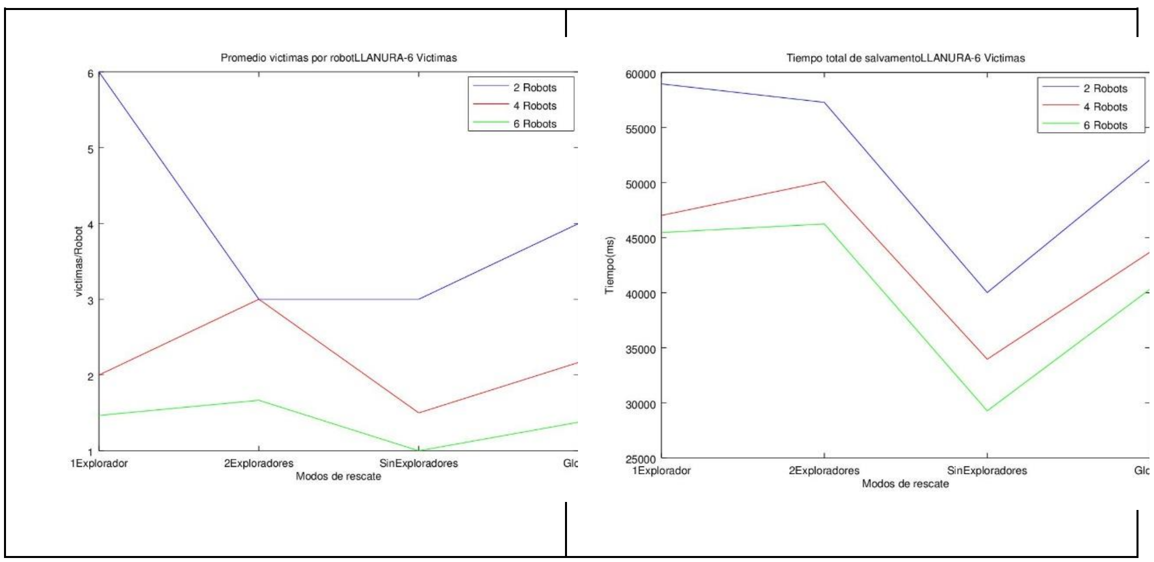

Figura 56: Promedio víctimas por robot. *Figura 57: Tiempo total de salvamento*

Estas gráficas confirman lo que parece lógico: cuanto mayor es la relación víctimas/robot, mayor es el tiempo de salvamento.

Por último, se va a comentar el aspecto que es quizás el más importante en un rescate: el porcentaje de **víctimas salvadas con vida**. Para ello se aporta otra gráfica:

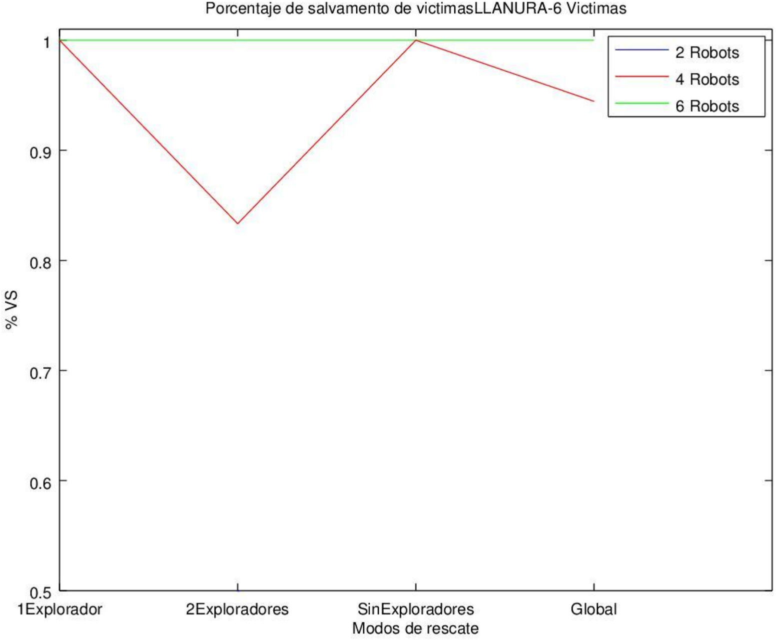

Figura 58: Porcentaje de salvamento

Como se puede ver en la gráfica, el porcentaje de víctimas salvadas con éxito es siempre del 100% excepto cuando se rescata con 2 exploradores. La razón encontrada a este suceso es simple: en los equipos de 2 y 4 robots, al tener que dedicarse 2 robots a la exploración, las víctimas comienzan a morir mientras los robots están explorando, lo que conlleva a una muerta prematura al rescate. Sin embargo, de las 6 víctimas "sólo" muere 1 víctima, lo cual no es fracaso total. Finalmente, la explicación al alto porcentaje de víctimas salvadas comparado con los porcentajes que se mostrarán en las gráficas de otros escenarios es simple: al no existir obstáculos, no hay que realizar grandes cálculos de ruta ni recorrer grandes distancias esquivando obstáculos, por lo que se facilita el rescate.

5.1.2 ESCENARIOS TIPO CASA

En este escenario se complica un poco el salvamento, ya que se introduce un elemento existente en cualquier rescate ligeramente real: los obstáculos. Los robots tendrán que realizar cálculos de ruta muy complejos para evitar los obstáculos y conseguir llegar hasta la víctima objetivo. Al haber elementos en el terreno, el movimiento a realizar por los robots será mayor y, por tanto, la simulación debe tender a ser peor en cuanto a resultados.

Se han hecho dos tipos de pruebas: primero variando el número de robots y después variando víctimas, con el fin de obtener resultados complementarios.

- **Variando Robots**

Se ha variado el número de robots de 4 a 6 y a 8 robots, teniendo un total de 10 víctimas a rescatar. Al igual que en el anterior apartado, se comienza la exposición de resultados con una gráfica que muestra el tiempo medio de salvamento de una víctima.

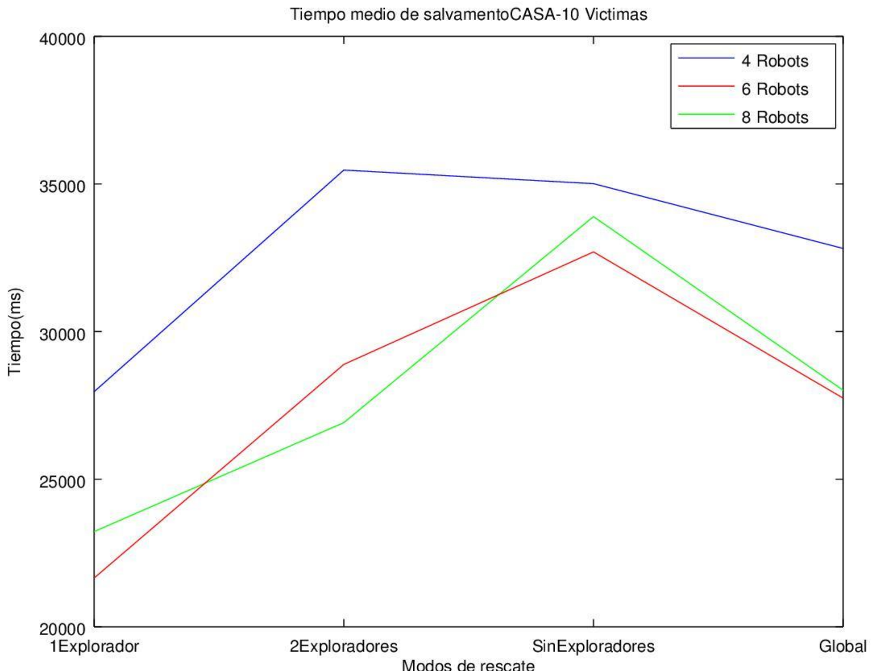

Figura 59: Tiempo medio de salvamento

Se observa, una vez más, que con menos robots se obtienen peores resultados pero que los resultados entre 6 y 8 robots son similares, lo que lleva a pensar que aumentar el número de robots no mejora los resultados siempre. También se observa que el tipo de rescate en el que menos se tarda en rescatar a las víctimas es el de 1 explorador, y el que más, el de sin exploradores.

Las siguientes gráficas expresan un poco lo ya comentado:

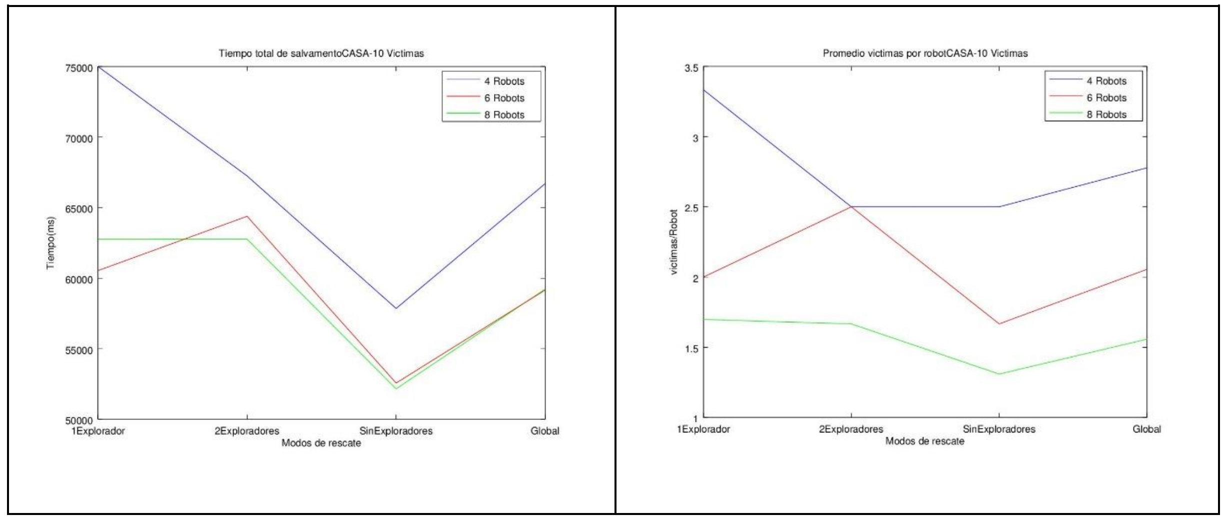

Figura 60: Tiempo total de salvamento. *Figura 61: Promedio víctimas por robot*

Como se puede observar, los resultados vienen a decir lo mismo que en la anterior gráfica: los tiempos de rescate son similares entre 6 y 8 robots, siendo peores con 4 robots. También se observa que el número de robots promedio por robot es mayor cuando se explora, lo cual es lógico ya que mientras se explora no se rescata.

Para acabar con este análisis, se va a ver cuántas víctimas han sido rescatadas con vida:

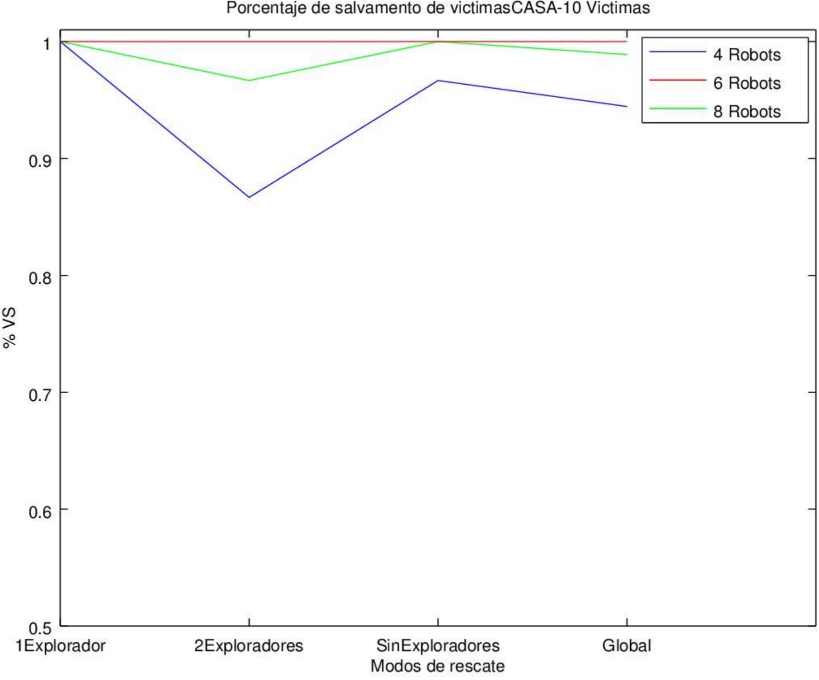

Figura 62: Porcentaje de salvamento

Aquí se puede confirmar lo que ya se sospechaba: si el número de robots es pequeño los resultados no son buenos, pero si el número de robots es demasiado grande tampoco se

garantiza obtener los mejores resultados. La razón por la que se puede explicar esto es que al aumentar el número de robots aumenta también el número de cálculos que se deben de realizar, lo que empeora la rapidez de respuesta de los mismos y ,por tanto, los resultados. Este hecho es muy importante ya que más adelante es posible que ocurra este suceso. Lo positivo que se puede contestar a esto es que si en vez de ser un único sistema el que realiza los cálculos de la simulación fuesen N sistemas para N robots, entonces seguramente no importaría tanto que el tamaño del equipo sea grande.

Por último, mencionar que los mejores resultados se obtienen con un explorador, mientras que con 2 exploradores se obtienen peores resultados.

- **Variando Víctimas**

Para cambiar un poco la dinámica de pruebas y añadir un nuevo fenómeno, se ha variado el número de víctimas en vez del número de robots. Concretamente, se ha variado el número de víctimas en este escenario de 10 a 15 y a 20 víctimas, siendo un total de 4 robots los encargados de salvarlas. A priori, parece una tarea complicada y nada parecida a los ratios víctimas/robot de anteriores simulaciones, por lo que las expectativas no son altas.

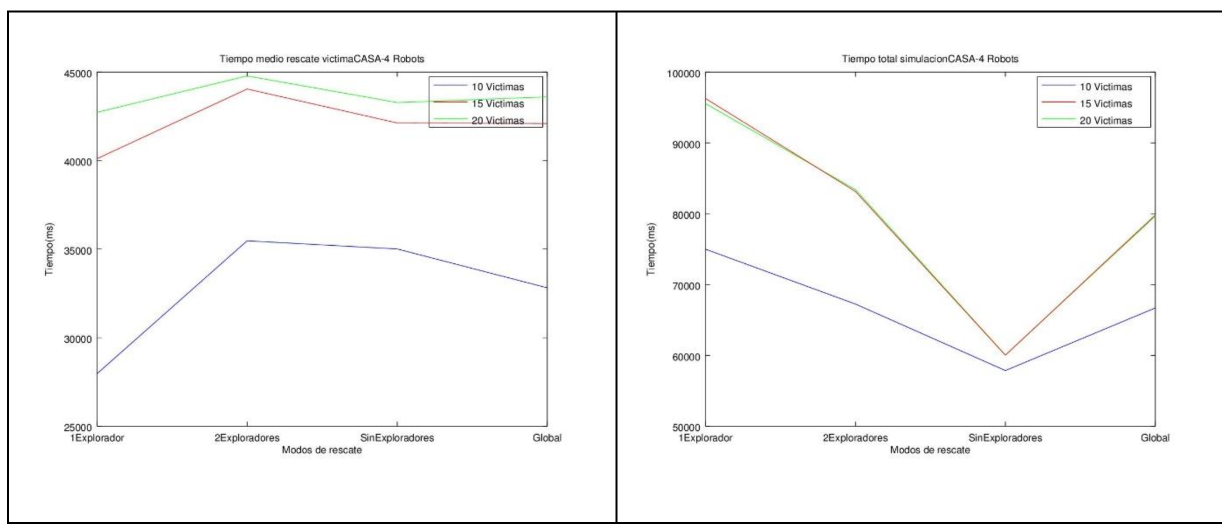

Figura 63: Tiempo medio de salvamento por víctima. *Figura 64: Tiempo total simulación*

Tal y como se observa, los resultados de 10 víctimas son mejores, lo cual es lógico, no es número muy alto de víctimas a rescatar por 4 robots, pero una vez se pasa a 15 o 20 víctimas, los resultados son similares y bastante peores: el tiempo de rescate medio de una víctima es bastante mayor sin importar realmente el tipo de rescate, y el tiempo total de simulación es mayor también. Sin embargo, también se puede extraer una buena conclusión: si el tipo de rescate es con 1 explorador, los resultados son casi siempre ligeramente mejores. Para finalizar este análisis, se expone la gráfica de porcentajes de salvamento y los ratios víctima/robot:

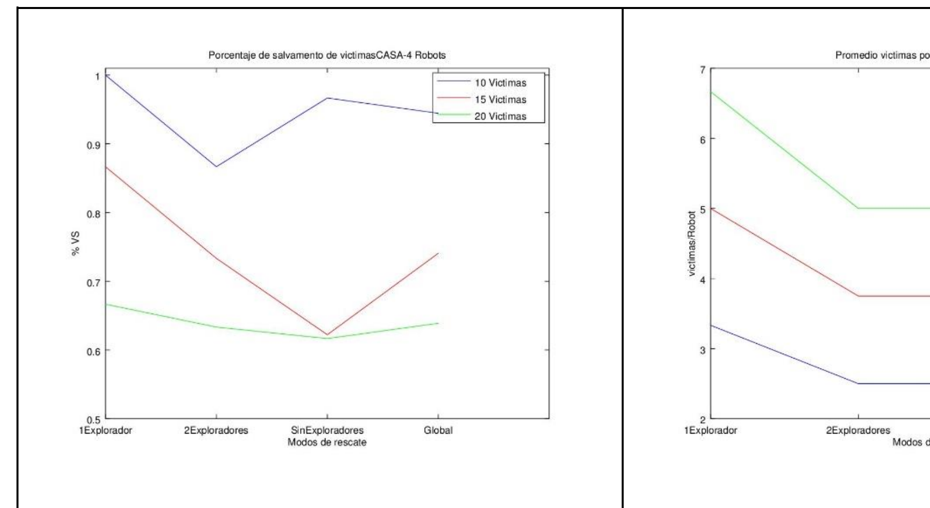

Figura 65: Porcentaje de salvamento. *Figura 66: Promedio víctimas por robot*

En primer lugar, la gráfica de la izquierda representa el porcentaje de salvamento de víctimas, el cual se puede observar con amplia diferencia que con a mayor número de víctimas a rescatar, peor porcentaje de salvamento. Esto es algo claro, como se puede observar en la gráfica de la derecha, se ven los ratios, siendo mayores o iguales a 4 cuando hay 15 o 20 víctimas, que coincide con porcentajes de salvamento mucho peores que con ratios inferiores. De igual forma, al igual que se ha comentado previamente, el tipo de rescate que mejores porcentajes de salvamento obtiene, aun sin ser lo suficientemente buenos, es con 1 explorador.

5.1.3 ESCENARIOS TIPO ESTADIO

Por último, tenemos el escenario tipo Estadio, que se caracteriza por tener bastantes obstáculos y una sola entrada al recinto de las víctimas, lo que dificulta el cálculo de rutas y el rescate en sí. Al igual que en el anterior escenario de pruebas, las expectativas no son altas.

De igual forma se han hecho pruebas variando robots de 4 a 6 y a 8 robots y variando víctimas de 14 a 17 y 20 víctimas.

- Variando Robots

  Se han realizado simulaciones con 4, 6 y 8 robots para rescatar a un total de 14 víctimas. Se comienza viendo los tiempos medios de rescate de una víctima:

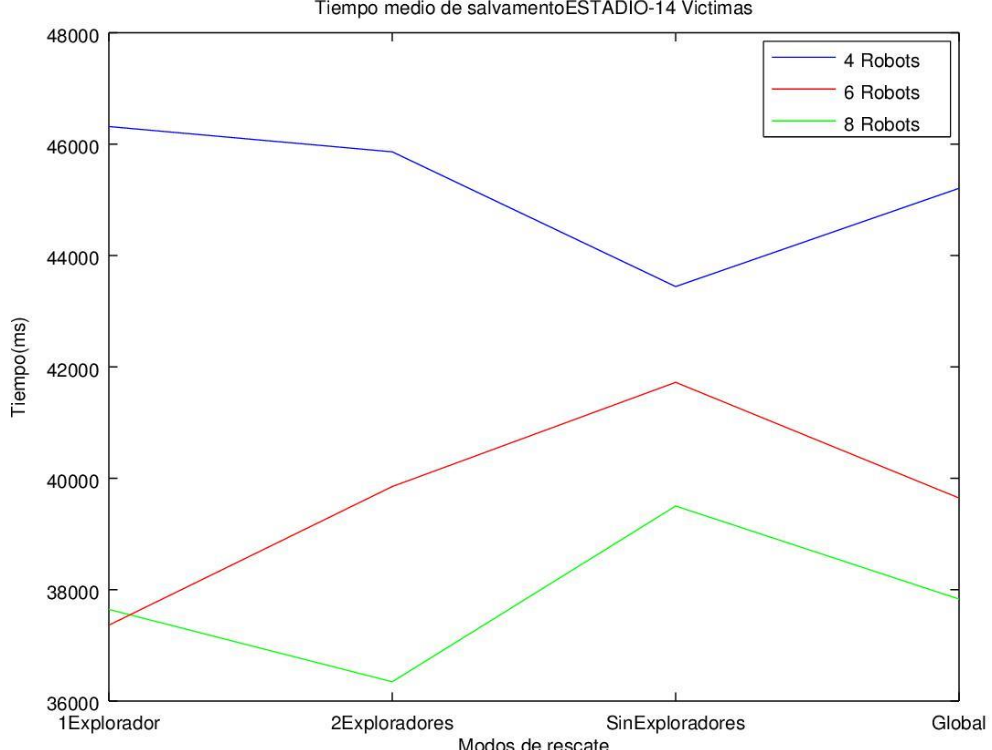

Figura 67: Tiempo medio de salvamento

Como se puede observar, los tiempos medios de rescate de víctima son menores a medida que aumenta el número de robots, y al contrario que en las demás pruebas, no existe un tipo de rescate que sea mejor en este caso.

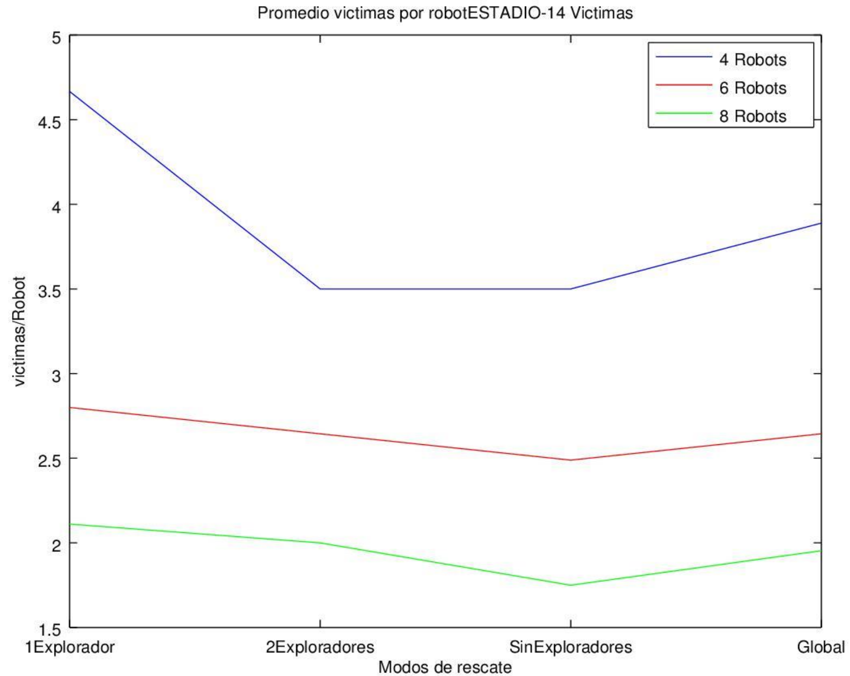

Figura 68: Promedio víctimas por robot

Tal y como se observa en la gráfica de arriba que representa la ratio víctimas/robot, como es lógico, a menor número de robots mayor es la ratio, y en consecuencia con lo comentado antes, el tiempo medio de salvamento de una víctima en mayor, lo que repercute directamente en el resultado de la misión de rescate.

Para finalizar este análisis, se va a observar cuál ha sido el resultado de la simulación en lo que a las víctimas se refiere:

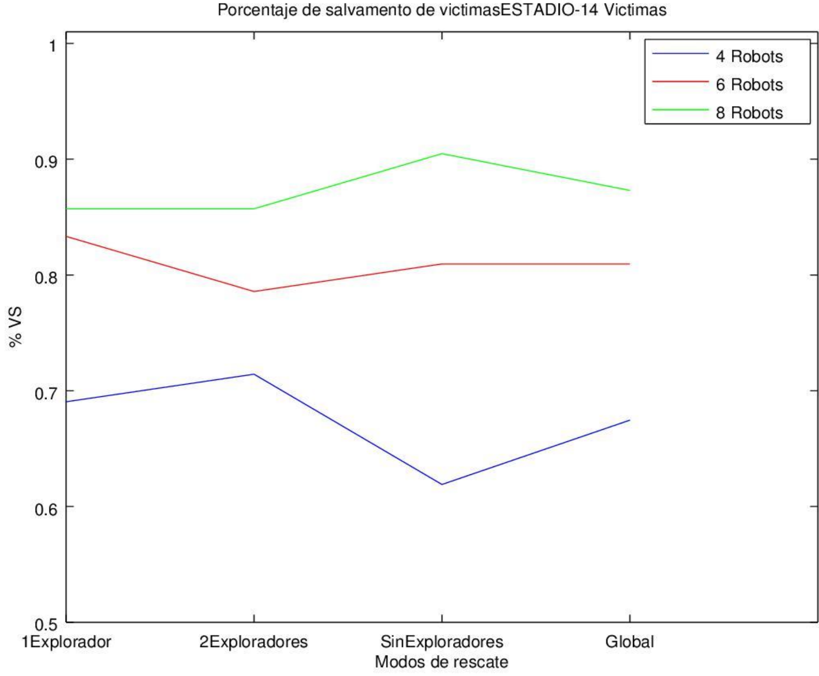

Figura 69: Porcentaje de salvamento de víctimas

Un simple vistazo sobre la gráfica denota algo claro: en concordancia con las anteriores gráficas, al ser menor el tiempo medio de rescate a mayor número de robots y siendo menor el ratio víctimas/robot, el porcentaje de salvamento es mayor, por lo que se puede concluir que el tamaño del equipo de robots es un factor determinante y el hecho de tener un ratio de victimas/robot mayor a 3 implica normalmente un porcentaje de salvamento inferior al 80% , resultado nada despreciable pero sí mejorable.

Por último, cabe mencionar que el mayor porcentaje con en torno al 92% se obtiene con el modo de rescate sin exploradores, hecho que es la primera vez que se repite, siendo en los demás escenarios determinante el tipo de rescate con 1 explorador.

- Variando víctimas

La última batería de pruebas ha sido variando víctimas de 14 a 17 y a 20 víctimas, para que un total de 8 robots las rescaten. El hecho de variar víctimas es parecido a variar robots, pero añade un factor determinante: las víctimas están en posiciones nuevas, no necesariamente cerca del resto de víctimas, por lo que los resultados no tienen por qué ser iguales a los expuestos anteriormente.

Se comienza con el tiempo medio de salvamento de una víctima, factor determinante:

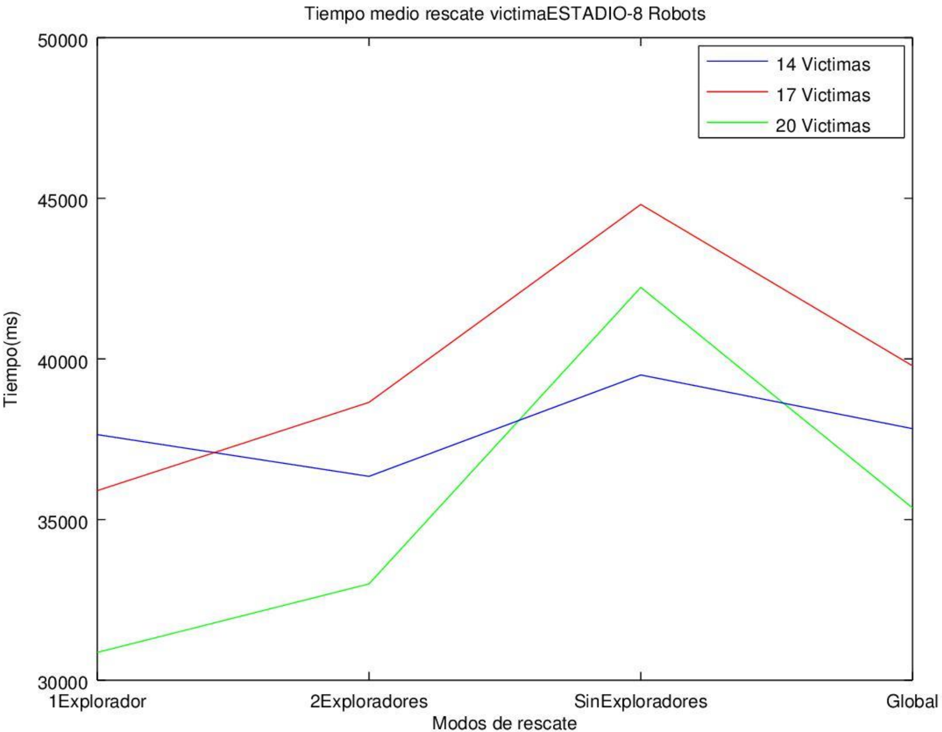

Figura 70: Tiempo medio de salvamento

Observando la gráfica a primera vista, no está claro determinar cómo ha variado el tiempo medio en función del número de víctimas, sin embargo, en función del tipo de rescate sí que se saca algo en claro: si el rescate es de 1 explorador, el tiempo de rescate es menor, y a mayor número de víctimas el tiempo medio de rescate es menor, lo cual indica que el de 1 explorador es buen tipo de rescate. También se puede ver que si el tipo de rescate es sin exploradores los tiempos medios aumentan, lo cual no es una buena señal.

Se continúa viendo las ratios víctimas/robot:

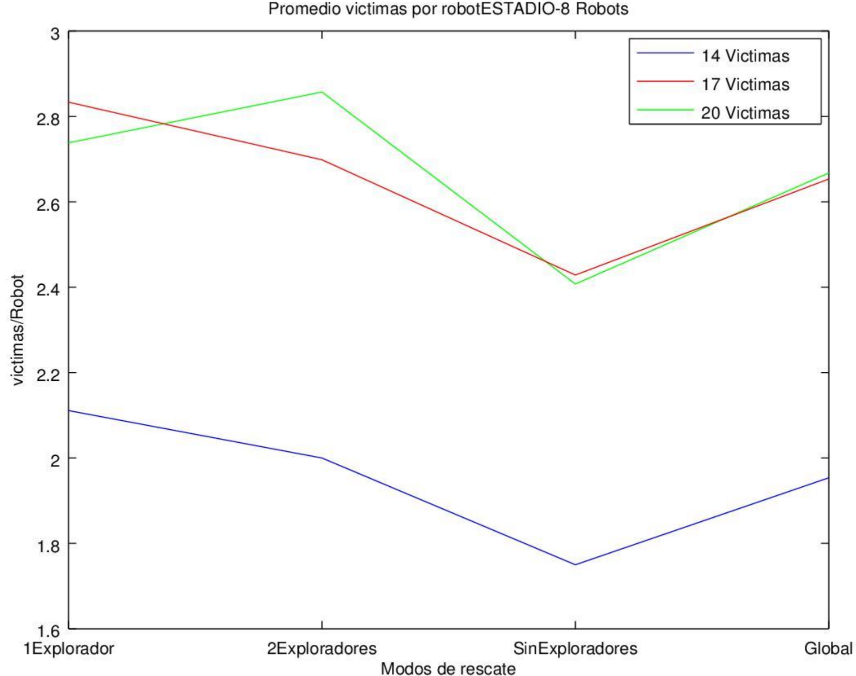

Figura 71: Promedio víctimas por robot

Tal y como se puede observar, la ratio es mucho menor para 14 víctimas, como es lógico, pero con 17 y 20 víctimas las ratios son prácticamente idénticos, lo que puede indicar que probablemente no habrá mucha diferencia entre el rescate de 17 y 20 víctimas.

Se sigue con los resultados clave de la simulación:

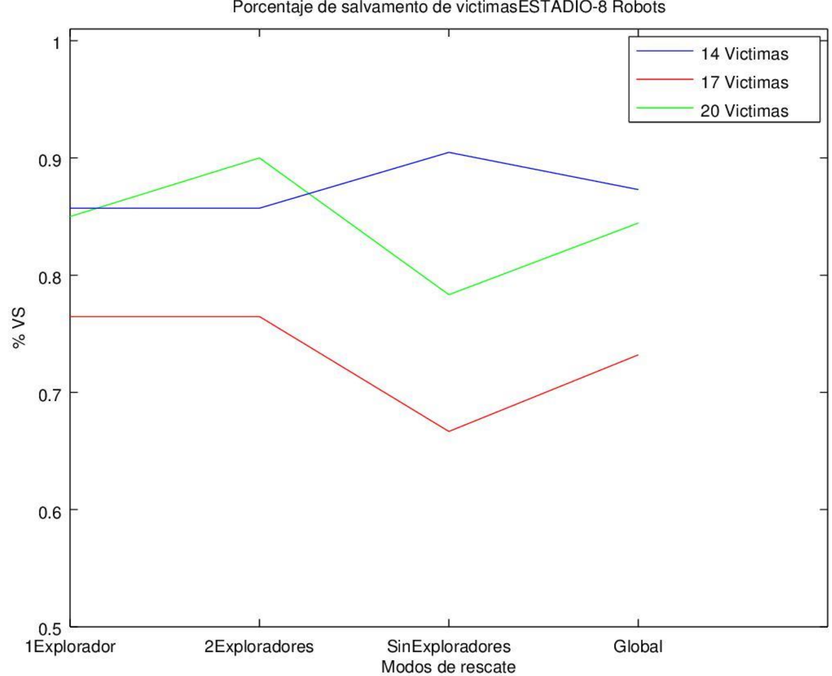

Figura 72: Porcentaje de salvamento

Para acabar, podemos ver en la gráfica de arriba el porcentaje de víctimas salvadas dependiendo del número de ellas en la simulación. Se puede ver que no hay un factor determinante, puesto que los peores resultados se obtienen con 17 víctimas, mientras que con 14 y 20 víctimas se van alternando el mejor resultado dependiendo del tipo de rescate. La explicación que podemos atribuir a este fenómeno es que la distribución de las víctimas sobre el terreno es determinante, si hay más víctimas, pero están más cerca del *lugar seguro* es normal que tarde menos de lo esperado y, al contrario, si hay pocas víctimas que están muy lejos del *lugar seguro* es lógico que también se tarde bastante.

5.2 CONCLUSIONES

RELACIONADAS CON LA SIMULACIÓN: EXPERIMENTACIÓN CON ROBOTS

- El número de robots es determinante: el número de robots en un rescate es determinante y para que los resultados sean óptimos es necesario que el número de robots sea en función del número de víctimas a rescatar, sin embargo, llevar un número excesivo de robots puede influir negativamente en la simulación por las siguientes razones:
  - A mayor número de robots aumenta la probabilidad de perder mensajes de comunicación entre ellos, lo que hace que se queden esperando a que salten los timeouts, perdiendo tiempo.
  - La simulación se realiza en un único host (un PC) y, a mayor número de robots, más cálculos de ruta hay que realizar, lo que lleva a un retardo de dichos

cálculos ya que no todos los cálculos se pueden realizar a la vez (en la simulación hay más de 20 hilos de ejecución activos simultáneamente). Este factor en estas simulaciones es importante, pero en teoría, si este proyecto se implementase en robots reales, donde cada uno realiza su cálculo de ruta en su chip, este problema desaparecería y, en principio, un alto número de robots debería dar buenos resultados.

- El tipo de rescate que mejores resultados da es de 1 explorador.
  El sistema está diseñado de forma que a partir de los N robots existentes, se destinan a exploración X robots a exploración, de forma que, mientras están explorando, no rescatan víctimas.
  En el grueso general de las pruebas el rescate con 1 explorador ha obtenido casi siempre los mejores resultados dentro de las distintas circunstancias, lo que lo sitúa como un método de rescate viable. Sin embargo, aumentar el número de exploradores a 2 no mejora los resultados y se encuentra una causa principal a este hecho:
  - A mayor número de robots destinados a exploración, menos robots hay para el rescate real de víctimas, por lo que influye negativamente en el rescate. Es cierto que las víctimas se localizan antes determinando sus tiempos de vida, pero al haber menos robots para el salvamento el resultado es peor. Obviamente, si se mantuviesen el número de robots “sanadores” y se declarase otro tipo de robot únicamente como explorador (un “dron”), este problema desaparecería y los resultados tenderían a ser mejores.
- El rescate sin exploradores no ha dado resultados suficientemente buenos: salvo en pocas ocasiones, el modo de rescate sin exploradores, en el que se conoce previamente la localización y el estado vital de las víctimas de antemano, no ha dado el mejor resultado. En contraposición a los resultados, creemos que este debería ser el mejor método de rescate, puesto que saber la información previamente es primordial para priorizar en función de las necesidades y creemos saber también la razón por la que no ha funcionado todo lo bien que debería:
  - Este modo de rescate implica una notificación masiva de las localizaciones de las víctimas al agenteJerarquicoAsignador, lo cual conlleva a una sobrecarga de procesamiento en su motor de objetivos. Es por esto mismo por lo que creemos que, al igual que otros problemas, si se llevase este proyecto a robots reales con su propio chip de procesamiento, entonces probablemente mejorarían resultados.

- Respuesta a la pregunta *“¿Qué número de robots es necesario para salvar N víctimas?”*: esta pregunta es la más compleja. El número de robots depende principalmente del número de víctimas, pero además de este factor también es determinante si las víctimas están en zonas complicadas de acceder o no (es decir, si hay obstáculos o no) y de la estrategia adoptada (modo de rescate).

  Por tanto, no se puede asegurar al 100% un número de robots óptimos, pero sí que se puede aconsejar que, si no hay obstáculos, con una ratio víctimas/robot menor que 3

se obtienen buenos resultados (%Salvamento > 90%) y que, si hay obstáculos en el escenario, lo cual es casi siempre en situaciones reales, es aconsejable no pasar de una ratio de 2 para asegurar buenos resultados.

$$Numero\ de\ robots = \begin{cases} \frac{n}{2}, & si\ hay\ obstáculos \\ \frac{n}{3}, & si\ no\ hay\ obstáculos \end{cases}$$

Siendo n el número de víctimas.

Como se ha explicado en primer apartado de las conclusiones, aumentar el número de robots no implica una mejoría en los tiempos de rescate por distintos retardos y mayores capacidades de cómputo.
Por lo que configurar la misión con ratios robot/víctimas cercanos a 1 deja de ser productivo, incluso llevando más tiempo de espera debido al mayor número de robots que tienen que hacer cálculos y de los que se tiene que esperar una respuesta.

RELACIONADAS CON EL PROYECTO

- **La elección de ICARO**: tal y como se explicó en otros apartados, ICARO es la infraestructura sobre la que se soporta toda la aplicación del proyecto ROSACE, y habiendo experimentado durante todo este año ICARO, se han sacado algunas conclusiones:

    - **Ventajas de ICARO**: es una plataforma ya desarrollada, que aporta interfaces y directivas para, entre otras cosas y de una forma relativamente sencilla, crear agentes, recursos y gestores y permitir la administración de los mismos junto con la comunicación entre agentes.

    - **Desventajas de ICARO**: se necesita previamente aprender a utilizar esta plataforma, lo cual no es trivial. A veces se echan en falta algunas funcionalidades y surgen problemas a la hora de implementarlas. Por último, un inconveniente que hemos encontrado a lo largo del proyecto es la alta cantidad de hebras que se crean y están activamente en funcionamiento, lo que en ordenadores personales de baja capacidad de computación supone un problema en ejecución.

- **Metodología utilizada**: en apartados anteriores se comentó acerca de la metodología de desarrollo de software utilizada, la cual ha sido **metodología ágil** con reuniones semanales. Con respecto a la metodología se encuentran ventajas e inconvenientes:

    - **Ventajas**: la metodología ágil permite una mejora constante del proyecto, pudiendo comprobar semanalmente cómo hemos avanzado y comentar las dudas que surgen. Utilizar esta metodología ágil ha ayudado a avanzar continuamente en el proyecto sin estar parados en ningún momento.

- **Desventajas**: la única desventaja que se encuentra a la metodología ágil es que a veces puede llegar a ser demasiado trabajo constante y provocar que disminuya el rendimiento por provocar situaciones de estrés.

- **Aspectos a mejorar**: en la creación del proyecto se cometieron fallos, por ejemplo, el familiarizarse con el proyecto con el sistema debería haber sido más rápido para poder empezar a trabajar en este, además se perdió tiempo en conseguir que funcionara en el IDE *Eclipse* y quizás hubiera sido mejor empezar a utilizarlo en el IDE en el que se encontraba, *NetBeans*. En resumen, el fallo más importante en el desarrollo fue la pérdida de tiempo en tareas que se podrían haber obviado o reducido.

ÚLTIMAS CONCLUSIONES

En este apartado se va a resumir las conclusiones del apartado anterior relacionadas con la simulación:

- El tipo de rescate que ha dado mejores resultados es el de 1 explorador. El de sin exploradores obtiene peores resultados debido a la alta computación del asignador.
- Los obstáculos, como es de esperar, dificultan el trabajo a los robots haciendo que haya peor porcentaje de salvamento y que el cálculo de la ruta tarde lleve más tiempo.
- No se puede asegurar al 100% un número de robots para N víctimas, pero generalmente para todo modo de rescate, **si hay obstáculos** es conveniente que no haya más de 2 víctimas por robot y si **no hay obstáculos** es conveniente que no haya más de 3 víctimas por robot.
- Este proyecto, portado a robots reales con su propio procesador cada uno, debería obtener mejores resultados porque el principal problema existente es que la simulación sobrecarga mucho al procesador ralentizando los cálculos de la simulación, por lo que habiendo N procesadores para N robots eliminaría retardos de cómputo.

LAST CONCLUSIONS

Here we are going to sum up the conclusions of the previous head related to the simulation:

- The rescue mode that has given the best results is the one-explorer mode. The rescue mode without explorers (this one has the information about the victims beforehand) gets worse results due to the high computing on the head robot.
- The obstacles, as expected, hinder the work to the robots causing the percentage of rescued victims decrease and increasing the calculation time of routes.
- It is not possible to ensure at 100% a number of robots for N victims but, in general, if there are obstacles is convenient not to set less than N/2 robots and if there aren’t obstacles is convenient not to set less than N/3 victims because it would cause a worse rescue.

- This project, ported to real robots with each own processor, should get better simulation results because the main existing problem is the overload of the processor which slows the computing of the simulation, so having N processors for N robots would remove the computing delays.

5.3 MÉTRICAS DEL PROYECTO

A lo largo del curso se ha ido trabajando con la herramienta de alojamiento de código GitHub, donde ha quedado todo el trabajo registrado mediante commits y pull-request. Observando lo que hemos ido haciendo por semanas y por meses, hemos creado un gráfico circular en el que se puede ver a qué aspectos del proyecto hemos dedicado más o menos tiempo indicado en modo porcentual:

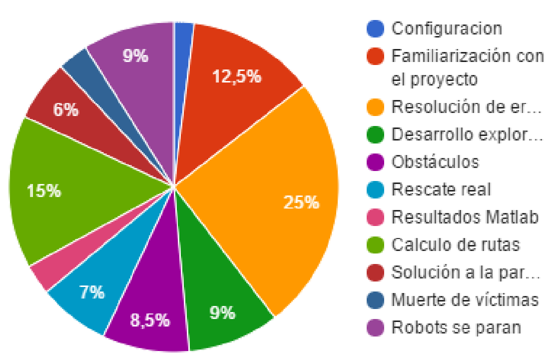

Figura 73: Gráfico del tiempo empleado en el proyecto.

Como se puede observar en el gráfico, a lo que más tiempo se dedicó fue a familiarización del proyecto, resolución de errores y cálculo de rutas. La explicación de por qué llevó más tiempo es la siguiente:

- El conocimiento del proyecto llevó un mes entero prácticamente, y esto fue porque el proyecto desde el que se partió era muy grande (>150.000 líneas de código) y quizás nos sobrepasó al principio.
- La resolución de errores ya existentes o que empezaron a saltar a lo largo del proyecto llevó una gran cantidad de tiempo repartido a lo largo de todo el año. Los errores más grandes que tuvimos fue en la ejecución del proyecto debido a

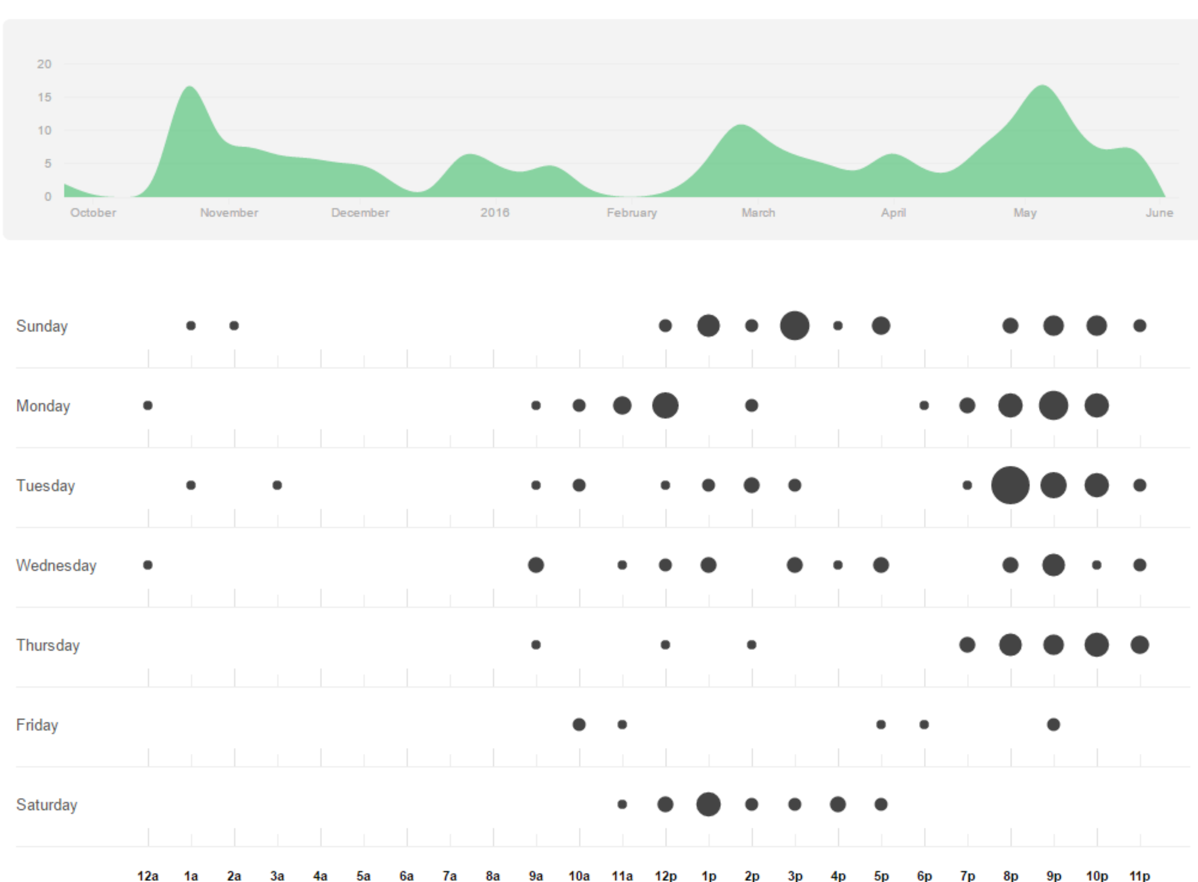

Figura 74: Gráficos de GitHub.

Como se pueden comprobar en las imágenes de encima, se ha trabajado en el proyecto de forma constante y a diario, normalmente entre las 20:00 y 22:00, dedicándole dos horas mínimo cada día. El único momento en que se ha bajado el ritmo ha sido en el mes de Febrero a causa de los exámenes, aunque nada más terminarlos se comprueba en la primera gráfica un pico considerable mostrando que se volvió a la forma usual de trabajar.

5.4 VERSIONES FUTURAS

Por falta de tiempo se han dejado varias cosas en el proyecto sin hacer que creemos que mejoraría bastante el sistema, aquí ponemos todas las posibles implementaciones que incrementarían la riqueza del mismo:

- **Actualizar librerías Drools**
  Esta mejora no se ha realizado por falta de tiempo, pero sería bastante interesante de realizar, la razón es que actualmente la versión de *Drools* únicamente ejecuta de forma correcta con *Java 1.7* lo que disminuye la portabilidad del proyecto. Además, Drools ha dado problemas con las reglas en más de una ocasión, por nombrar algún problema había ocasiones donde no se ejecutaban reglas seguramente por el RETE; se sabe que la nueva versión de Drools ha cambiado esto por lo que puede ser bueno para el proyecto el actualizar.

- **Mantener independientes las rutas del proyecto**
  Este problema surge porque las rutas de los archivos que se utilizan en el proyecto (como pueden ser los de los autómatas) tienen una ruta que depende de *Eclipse*, por esto no se puede crear una demo en formato *JAR* del proyecto para ejecutar en cualquier ordenador, se necesita poner este archivo *JAR* en el directorio con los archivos del proyecto para que esto funcione. El coste de solucionar este error es bastante grande por la gran cantidad de rutas que hay en el proyecto, aunque sin duda es necesario si se quiere convertir en un sistema totalmente portable.

- **Implementar diferentes formas de organización de equipos**
  Cuando se comenzó con el proyecto existían dos formas de organizar el equipo, **Igualitario** y **Jerárquico**, se centró toda la atención en el jerárquico ya que pareció una forma más interesante de organizarlo, además permitió centrarlos completamente en este y mejorarlo todo lo posible. Por esto es interesante crear nuevas formas de organizarlos, ya sea implementar todo lo hecho en **Igualitario** o crear uno nuevo como puede ser un tipo Igualitario donde cada agente sea jefe de un tipo de tareas, por ejemplo, uno puede ser el jefe de seleccionar quien salva a quien, otro quien reconoce el terreno, otro como se reasignan las víctimas, etc.

- **Implementar la posibilidad de que exista más de un equipo en la misión**
  Esta idea se planteó en un inicio para realizar en este proyecto, lamentablemente por falta de tiempo al final no se pudo realizar, pero sin duda es una de las ideas más interesantes que se barajan para mejorarlo. Actualmente en un escenario puede haber un único equipo de robots, pero si se pudiera utilizar varios equipos se podría conseguir una gran capacidad de paralelismo y aumentar la cantidad de coordinación que tendría que haber. Además, gracias a esto se podría aumentar el tamaño del escenario de salvamento ya que aumentaría la cantidad de agentes disponibles y la velocidad a la que se tomarían decisiones porque habría más de un agente tomándolas.
  Un problema de esta mejora es que aumentaría la cantidad de cómputo exigida por el proyecto que ya de por si es bastante grande.

- **Permitir los obstáculos diagonales**
  En el proyecto al usar los antiguos algoritmos de rutas se deshabilito la posibilidad de crear obstáculos diagonales ya que no funcionaban correctamente además aumentaban bastante el cómputo (con el antiguo algoritmo el proyecto llegaba a poner un procesador i7 al 100%) por lo que se llegó a la conclusión de que lo mejor era no permitirlos. Actualmente con el nuevo algoritmo tampoco funciona, esto es porque únicamente comprueba la siguiente posición y como se explicó antes por la forma en que las líneas se dibujan no es posible.
- **Mejorar la vista**
  Quizás la parte más descuidada del proyecto es la vista, al inicio del mismo se planteó usar MASSIS, un proyecto desarrollado para proporcionarle una vista 3D, lamentablemente al final no pudo ser posible y la vista ha quedado de esta forma. Las mejoras posibles en este aspecto son dos, o bien se mejora la imagen 2D añadiendo color al escenario para darle un toque más real además de poner obstáculos más gruesos para que se vean bien; o se implementa una vista en 3D ya sea con MASSIS o con otro sistema para mejorar aún más la realidad de las misiones.
- **Exportación a robots reales**
  Quizás la idea que más interesante y más emocionante, actualmente es solo un software que permite estudiar las organizaciones de los equipos de agentes y analizarlas, pero si este software se pudiera cargar en un robot real y darle información del mapa en modo plano aéreo simulando los escenarios de misión podría funcionar en misiones reales lo que sería el objetivo final de este proyecto.

6. BIBLIOGRAFÍA

- Proyecto ROSACE en GitHub
  - https://github.com/fgarijo/rosaceSIM
- Proyecto ICARO en GitHub
  - https://github.com/fgarijo/ICARO
- Información sobre Drools
  - https://es.wikipedia.org/wiki/Drools
  - http://www.tutorialspoint.com/drools/
- Algoritmo de Lee
  - http://users.eecs.northwestern.edu/~haizhou/357/lec6.pdf
- Información sobre el error que no permitía ejecutar
  - http://stackoverflow.com/questions/7970622/java-7-jvm-verifyerror-in-eclipse
  - http://stackoverflow.com/questions/10404929/java-lang-verifyerror-when-running-main-class-in-eclipse
  - https://www.eclipse.org/forums/index.php/t/58517/

ANEXO I: CASOS DE USO

INICIO

0CU–Inicio-03	**Gestionar Simulación**
Objetivo en contexto	Obtener las capacidades de gestionar la simulación.
Precondiciones	El sistema debe estar iniciado.
Postcond. si éxito	Tener la interfaz de control..
Actores	Usuario
Secuencia normal	1 – Elegir la vista de la interfaz de control. 2 – Se obtienen la capacidad de realizar los casos de uso de **gestionar simulación.**

CU–Inicio-04	**Termina Sistema**
Objetivo en contexto	Finalizar el sistema
Precondiciones	Tener iniciado el sistema
Postcond. si éxito	Cierra el sistema
Actores	Usuario
Secuencia normal	1 – Ir a la ventana de recursos de traza 2 – Pulsar el botón "Terminar Simulación" 2 – Confirmar el cierre pulsando si

GESTIÓN ESCENARIO

CU–escenarios-03	**Modificación de un escenario**
Objetivo en contexto	Crear un nuevo escenario.
Precondiciones	El sistema debe estar cargado correctamente.
Postcond. si éxito	Se crea un nuevo escenario para usarlo en simulación
Postcond. si fallo	Mensaje de error y vuelve al menú de edición.
Actores	Usuario.
Secuencia normal	1 – Ir al menú de Editor de escenarios 2 – Seleccionar *Abrir escenario.* 3 – Modificar el escenario. 4 – Sobrescribir el escenario.

CU–escenarios-04	Eliminación de un escenario
Objetivo en contexto	Eliminar un escenario ya creado.
Precondiciones	El sistema debe tener cargado un escenario.
Postcond. si éxito	Se elimina el escenario que está cargado.
Postcond. si fallo	
Actores	Usuario.
Secuencia normal	1 – Ir al menú de Editor de escenarios 2 – Seleccionar *Eliminar este escenario.*

GESTIÓN DE SIMULACIÓN

CU–simulación-03	Salvar una víctima
Objetivo en contexto	Salvar a una víctima del escenario.
Precondiciones	El sistema debe estar cargado y funcionando.
Postcond. si éxito	Se salva a una víctima del escenario.
Postcond. si fallo	
Actores	Usuario.
Secuencia normal	1 – Seleccionar la víctima que se quiere salvar del menú con doble clic. 2 – Pulsar en Salvar Victima

CU–simulación-04	Seleccionar escenario
Objetivo en contexto	Selecciona un escenario donde realizar la simulación.
Precondiciones	El fichero del escenario debe tener bien el formato.
Postcond. si éxito	Se cambia el escenario para simular.
Postcond. si fallo	Mensaje de error y vuelta al menú principal.
Actores	Usuario.
Secuencia normal	1 – Seleccionar el menú superior Editor Escenarios 2 – Seleccionar Abrir Escenario en el submenú 3 – Elegir el escenario que se quiere usar. 4 – Aceptar.

CU–simulación-05	Parar robot
Objetivo en contexto	Parar un robot del equipo
Precondiciones	La simulación debe estar iniciada.
Postcond. si éxito	El robot elegido se para.
Postcond. si fallo	
Actores	Usuario, RobotSubordinado.
Secuencia normal	1 – Elegir un agente a parar 2 – En la interfaz de simulación pulsar el botón de dicho agente
Secuencia alternativa	2a – Si el actor es el subordinado no pulsa nada, directamente envía el mensaje al asignador y se para.

GESTIÓN DE RESULTADOS

CU–resultados-03	Visualización de resultados
Objetivo en contexto	Visualiza las métricas que ha creado.
Precondiciones	Los métricas deben estar cargadas.
Postcond. si éxito	Se visualizan las métricas con los datos.
Postcond. si fallo	Mensaje de error y el programa termina.
Actores	ScriptOctave
Secuencia normal	1 – Selecciona los datos a mostrar. 2 – Muestra los datos.

AUTORIZACIÓN PARA LA DIFUSIÓN DEL TRABAJO FIN DE GRADO Y SU DEPÓSITO EN EL REPOSITORIO INSTITUCIONAL E-PRINTS COMPLUTENSE

Los abajo firmantes, alumno/s y tutor/es del Trabajo Fin de Grado (TFG) en el Grado en Ingeniería Informática de la Facultad de Informática, autorizan a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el Trabajo Fin de Grado (TF) cuyos datos se detallan a continuación. Así mismo autorizan a la Universidad Complutense de Madrid a que sea depositado en acceso abierto en el repositorio institucional con el objeto de incrementar la difusión, uso e impacto del TFG en Internet y garantizar su preservación y acceso a largo plazo.

Periodo de embargo (opcional):

- ☐ 6 meses
- ☐ 12meses

TÍTULO del TFG: SIMULACIÓN DE MISIONES DE RESCATE CON ROBOTS

Curso académico: 2015/ 2016

Nombre del Alumno/s:

Luis García Terriza

Sergio Moreno de Pradas

Tutor/es del TFG y departamento al que pertenece:

Juan Pavón Mestras

Francisco J. Garijo

Firma del alumno/s

Firma del tutor/es