UNIVERSITAT DE BARCELONA

Trabajo fin de grado

GRADO DE INGENIERÍA INFORMÁTICA

Facultad de Matemáticas e Informática
Universitat de Barcelona

Normas éticas para la coordinación de agentes en el Robocup Rescue Simulator

Autor: Jorge Tomé Blanco

Director: Dr. Maite López Sánchez
Realitzat a: Departament de
(Matemàtiques i Informàtica)

Barcelona, June 27, 2019

Contents

List of Figures

1 Resumen

El siguiente trabajo estudia los efectos provocados por la aplicación de un subconjunto de normas dentro de una simulación. Las normas han sido seleccionadas usando el método propuesto en el paper [3] y sobre el conjunto descrito en este otro [2]. En ambos se exponen las normas como elementos ordenadores y con propiedades coordinativas de una comunidad compuesta ya sea por agentes humanos, ya sea por robots, la problemática abierta que supone la elección del mejor subconjunto de normas y una aproximación teórica y experimental que formaliza una metodología programática capaz de resolverlo considerando la relación existente entre moral y normas y cuán bien se adecúan las últimas a la primera. El simulador elegido ha sido como propuso [2]: Robocup Rescue Agent Simulator.

2 Introducción

Hay en la actualidad una gran variedad de problemas que abre el desarrollo de la inteligencia artificial y su aplicación, los sistemas multi-agente y su interacción es uno de ellos y también lo es y presumiblemente de mayor trascendencia por lo que refiere a los círculos no solo académicos el hecho de que se requiera integrar una ética, moral o normativa sobre robots así como a los seres humanos aplica.

Las normas jurídicas dibujan los límites de un area de acción a la que llama libertad la persona jurídica. Esta forma de moldear el espacio de acción que tan común nos es está siendo estudiado y extrapolado e implementado sobre sociedadedes no humanas sean estas artificialmente simuladas o bien sobre robots. Tanto si se considera su aplicación sobre sociedades humanas como no, varios problemas quedan planteados y abiertos. Entre ellos la problemática de seleccionar el mejor subconjunto atendiendo a fundamentos de lógica deóntica y en algunos casos más factores. Una investigació en activo desarrollada mediante la cooperación de la Universidad de Barcelona, Lleida y Oxford junto a IIIA ("Artificial Intelligence Research Institute") ha propuesto como solución un método de selección formal que se vale de imponer principios lógicos como el de no contradicción/exclusividad, las relaciones de sustituibilidad y generealización (Véase [3]) junto a una optimización en base a un sistema moral donde cada regla obtiene una relación con uno o más valores. De este modo el problema queda definido como un problema de carácter matemático de optimización con restricciones.

Posteriormente se publicó [2] donde se propuso usar la plataforma de Robocup Rescue Agent Simulator para poner en práctica una serie de normas elegidas mediante la metodología planteada. La elección de la plataforma cumple con los requisitios dado que los agentes que interactuan en la simulación representan seres humanos y el enfoque de la investigación se plantea sobre la elección de normas en sociedades humanas. Además el simulador ofrece un contexto que permite generar un problema donde la coordinación entre agentes es una pieza clave para el "rescate" y la

"reducción de daños" así como las normas representan herramientas de coordinación en la investigación y resulta evidente que el buen o mal desarrollo de la simulación puede permitir discernir si la optimización basada en valores morales converge con mejores o peores resultados.

3 Objetivos

Los objetivos son: implementar e integrar las normas seleccionadas de entre el conjunto definido por el artículo [2] mediante el software seleccionador desarrollado que aplica la metodología propuesta por la investigación usando la plataforma de simulación Robocup Rescue Agent Simulator y analizar los resultados.

4 Robocup Rescue Simulation

Robocup es tanto un proyecto de carácter educativo como de investigación que engloba diversas competiciones que se celebran anualmente. Algunas de ellas aplican sobre plataformas de simulación como Robocup Rescue Simulation o Robocup Soccer Simulation y también las hay en las que participan robots. Respecto a las ligas estas pueden ser clasificadas según las iniciativas que las promueven y son: RoboCupSoccer, RoboCupRescue, RoboCupJunior, RoboCupHome, RoboCupWork y RoboCupLogistic.

Originalmente Robocup se funda en 1996 y la iniciativa RoboCupRescue nació en respuesta al desastre natural que denominado ya por entonces el Gran terromoto de Hanshin-Awaji afectó a la prefectura de Hyõgo y muy especialmente la ciudad de Kobe en enero de 1995 donde dejó más de seiscientos fallecidos y numerosos daños materiales.

Tanto la liga de rescate simulado como real comparten el objetivo de hallar la mejor forma de reaccionar ante un desastre natural sobrevenido y fomentar investigaciones en este campo. El proyecto RoboCupRescue se compromete en ofrecer una plataforma de simulación capaz de representar de form realística escenarios de desastres naturales y es esta misma la que ha sido usada para el desarrollo de este trabajo.

4.1 RAMBench

La interacción con la plataforma de simulación se ha realizado mediante el proyecto de master [1] por motivos de simplicidad a la hora de interactuar e intervenir sobre el comportamiento de los agentes y configuración del entorno. El proyecto de master base usado denominado RMASBench cuenta con la siguiente estructura:

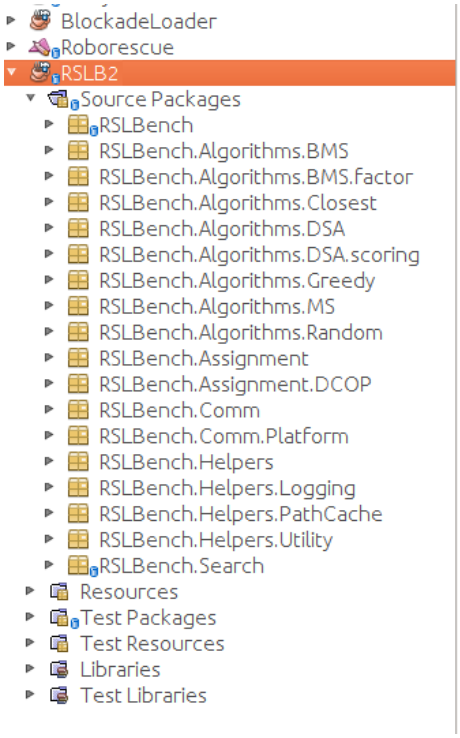


Figure 1: Estructura proyecto RMASBench

En la figura 1 se obervan un total de tres proyectos. Los tres están escritos en java 1.7, y tanto BlockadeLoader como RSLB2 contraen dependencias con el proyecto de Roborescue, este último contiene la plataforma. El contenido del proyecto RSLB2 ha sido extendido pero únicamente en el package RMASBench y RSLBench.Search el resto del proyecto base no ha sido editado y como ya ha sido mencionado forma parte del proyecto de master [1] que consistió en la aplicación de diversos algoritmos destinados a proporcionar soluciones a un planteamiento del problema de extinción de incencios y desbloqueo de colapsos en carreteras formalizando este en uno de tipo DCOP (siglas de Distributed Constraint Optimization Problem). Una clase denominada **CenterAgent** implementada ya en el proyecto base se vale del algoritmo **BinaryMaxSum** para realizar la asignación de objetivos tanto a agentes de bomberos como policías. Los algoritmos de resolución y que realizan esta asignación básica serán mencionados en este trabajo como Solvers en adelante.

Roborescue define un set de entidades que componen e interactuan en la simulación y no es sino mediante la comunicación entre el proceso Kernel y el proceso RSLB2 que las acciones tomadas desde este último terminan por ser recibidas por el proceso Kernel y después de otros procesos comunicativos termina reflejándose aquella en la simulación.

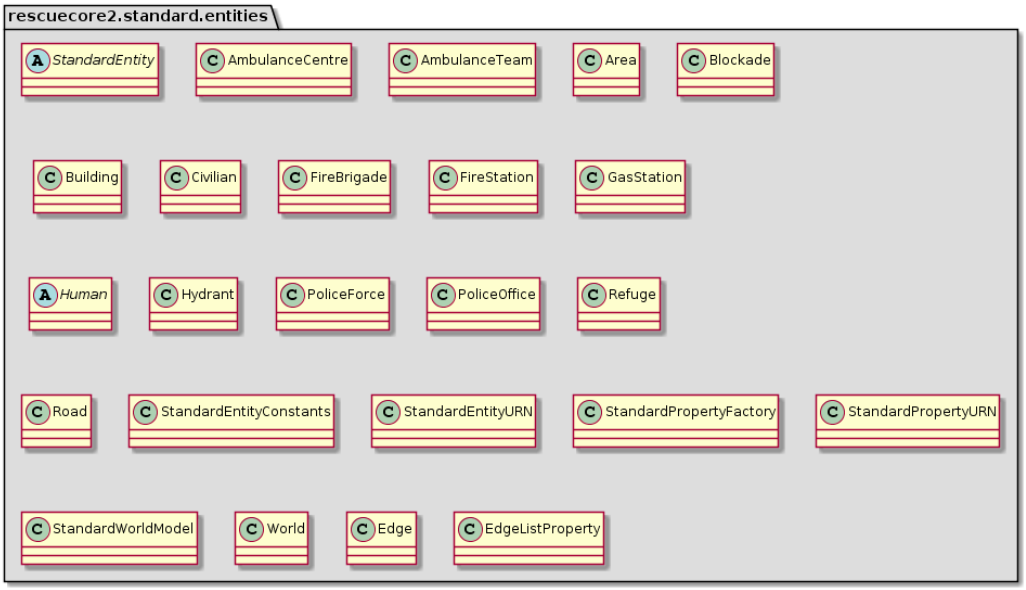


Figure 2: Contenido del package rescuecore2.entities

En la figura 2 se presenta el set de entidades que componen e interaccionan en el simulacro. Las entidades pueden clasificarse en dos grupos bien diferenciados: por un lado las entidades que representan seres humanos y por otro aquellas que representan estructuras. En las siguientes secciones se mostrarán las relaciones de herencia que hay entre ellas y de modo concreto se introducirán las más relevantes por lo que a este trabajo se refiere.

4.1.1 Entidades Humanas

Las entidades que representan seres humanos tienen una serie de características comunes y reflejan estados similares. Obsérvese pues la siguiente figura:

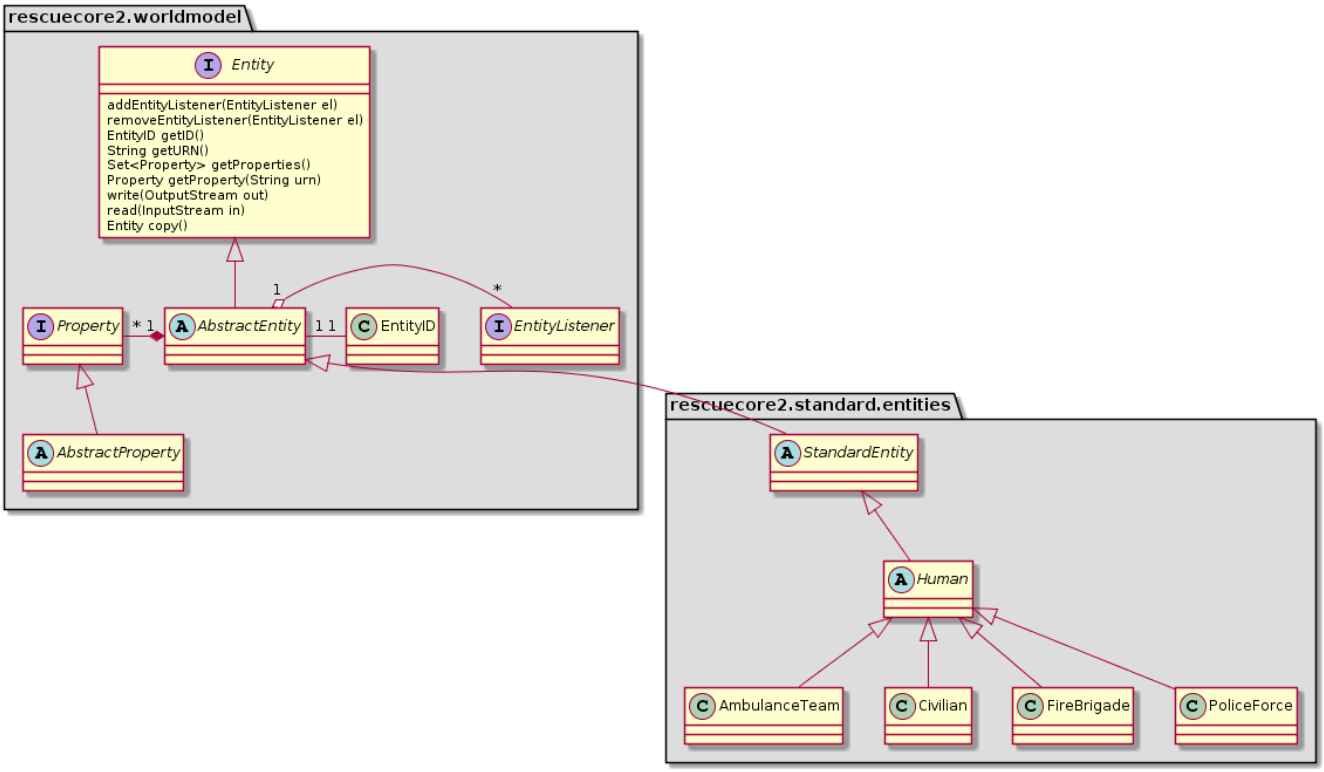


Figure 3: Diagrama de Clases, Agentes y sus principales relaciones

En el diagrama de clases de la figura 3 únicamente se han mostrado las clases de relieve en esta sección. Se observa que los agentes humanos son todos aquellos que heredan de la clase Human definida en el package de entidades y por lo tanto se distinguen cuatro: **AmbulanceTeam**, **Civilian**, **FireBrigade** y **PoliceForce**. Tal y como se observa en el diagrama todo humano hereda de la clase abstracta StandardEntity y su ancestro tal y como se obervará con la categoría de entidades que podemos catalogar como estructuras. Es por ello que común a todas las entidades de la simulación es el hecho de que: todas y cada una de ellas esté asociada a un identificador encapsulado mediante la entidad **EntityID** único, que esté compuesta por múltiples propiedades (las hay de distintos tipos pero todas ellas tienen en común ser partícipes de la firma de la interfaz **Property**) y además se les posibilita la capacidad de enviar notificaciones a listeners implementando en este caso un patrón Observer para desarrollar un diseño reactivo basado en eventos.

A continuación se expondrá exactamente que representa conceptualmente cada tipo de ser humano:

- **AmbulanceTeam**: representa un equipo sanitario que equipado con una ambulancia sería de este modo capaz de desplazarse de forma veloz al mismo tiempo que estaría capacitado para realizar una primera asistencia y transportar heridos. Aunque su nombre incluya un equipo (del inglés "team") de cara a su análisis ha sido tratado como un único humano y no como a un con-

junto, esta es tan bien su forma según la implementación base del proyecto Roborescue.

- **FireBrigade**: representa una brigada de bomberos abordo de un camión cisterna y equipado con mangueras que les capacitaría para desarrollar acciones contra incendios. Tal y como sucedía con AmbulanceTeam todo y que conceptualmente represetaría a más de un ser humano han sido tratados tal y como especifica la lógica de la plataforma de simulación en su estructura de clases, esto es, como un único ser humano.
- **PoliceForce**: representa una patrulla de policías. Contarían con un vehículo el cual les permitiría moverse velozmente a través del escenario y estarían capacitados para despejar caminos si es que estos estuviesen invadidos por bloqueos.
- **Civilian**: tal y como el nombre indica representa un civil. La lógica de la simulación lo trata como a un ser humano menos capaz de resistir daños y por lo tanto algo más vulnerable a las distintas catástrofes capaces de ser simuladas.

Las propiedades que componen a los distintos agentes son comunes prácticamente en su totalidad exceptuando el agente de bomberos como se observa en el siguiente diagrama:

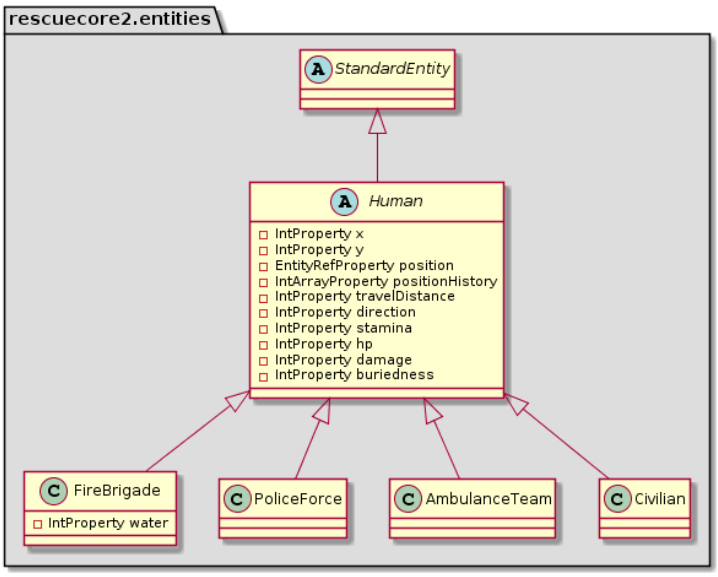


Figure 4: Propiedades comunes e individuales de los agentes

Observando la figura 4 se percibe cómo la composición de los agentes es fuertemente compartida y se explica debido a la común reacción de estos con los distintos eventos y estados que se producen a su vez en el simulador. El naming de las propiedades es bastante esclarecedor aunque los eventos que interactúan sobre ellas

surge de una interacción que requiere un nivel más de entendimiento y será introducido más adelante.

Tenemos propiedades de localización que permiten almacenar la posición del ente en el escenario como son x e y que en conjunción definen una coordenada en el mapa, la propiedad *position* que almacena en cada instante una instancia del tipo EntityID identificando a que entidad del tipo **Area** pertence la coordenada (x,y), otras almacenan información sobre la distancia total recorrida como el caso de *travelDistance* y otras un historial de posiciones visitadas como *positionHistory*. La propiedad *hp* (abreviatura de "health points", en castellano "puntos de vida") permite almacenar el estado de salud, se sabe que son 10.000 los puntos base para todos los tipos de seres humanos. La propiedad *damage* es tan solo incremental y se actualiza cada vez que se recibe una lesión, se entiende que la información que almacena no sería deducible únicamente con el nivel base de vida y los puntos restantes de salud si algún mecanismo de recuperación entrase en juego. En el caso la entidad **FireBrigade** su propiedad *water* es usada para almacenar la cantidad de agua consumida.

4.1.2 Entidades Estructura

Las entidades que conforman el mapa pertenecen a alguna de las siguientes:

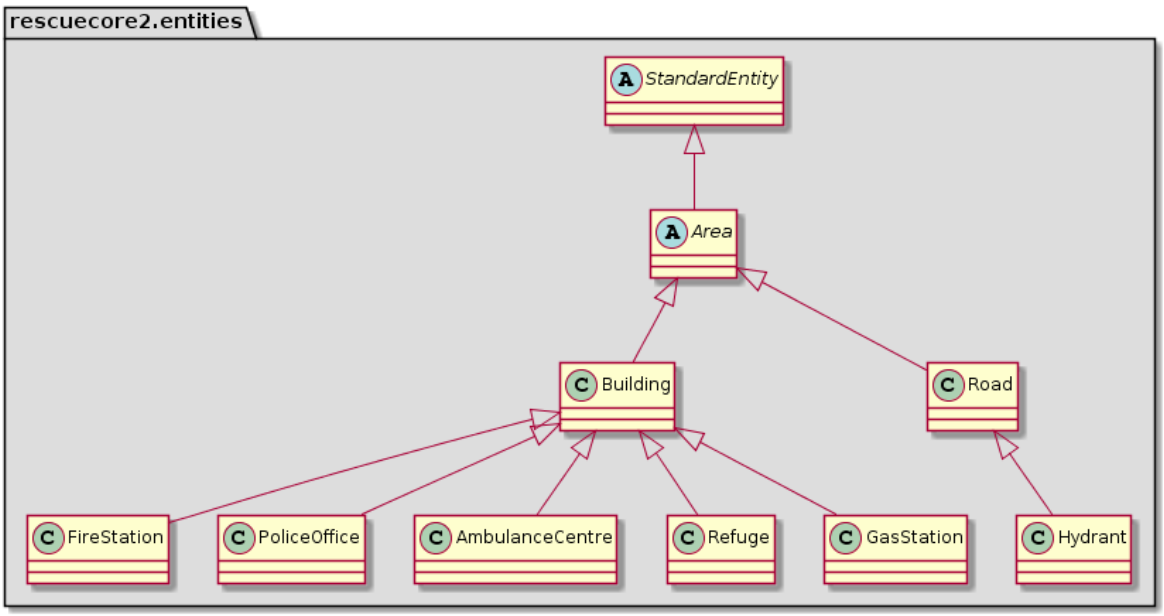


Figure 5: Relaciones de herencia, entidades estructura

La Figura 5 muestra a modo de diagrama de clases cuáles son las relaciones de herencia entre este conjunto de entidades. Al heredar todas ellas de **StandardEntity** (y esta a su vez de **AbstractEntity** manteniendo las relaciones anterioemente mostradas) guardan atributos y funcionalidades comunes a los agentes como ejemplo

estar todas ellas relacionadas con un identificador (**EntityID**). Las entidades más relevantes para este trabajo han sido: **Building**, **Road** y **Refuge** que representan un edificio, una calle y un refugio respectivamente.

Veamos, pues, cuáles son las propiedades que componen estas clases con el siguiente diagrama:

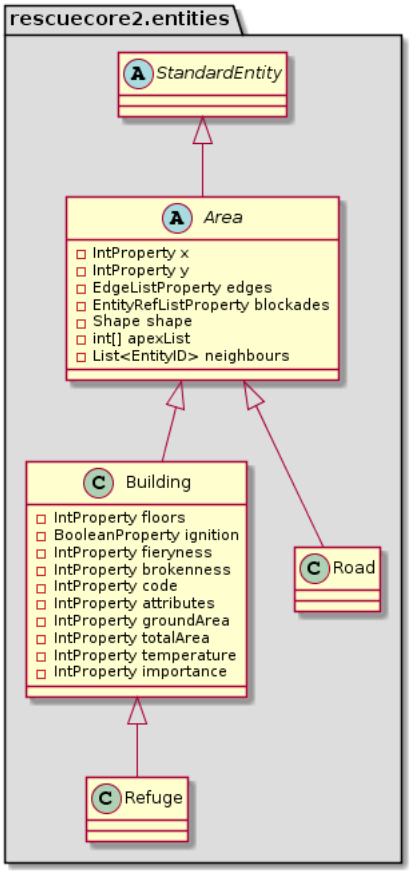


Figure 6: Estructuras y sus propiedades

La clase abstracta **Area** define tanto la localización en el mapa (véase los atributos coordenada x e y) como su forma mediante el atributo *shape*, también sus conexiones en relación a otras áreas mediante el listado de entidades que almacena el atributo *neighbours* y un listado de bloqueos mediante *blockades*. A pesar de que la implementación permite que los edificios contengan bloqueos esto experimentalmente no sucede y los bloqueos siempre ocupan total o parcialmente una entidad tipo **Road** cuando los hay, y aunque la lógica permita almacenar una lista de ellas mediante el uso de la propiedad tipo **EntityRefListProperty** experimentalmente cuando una calle tiene un bloqueo siempre es uno solo.

4.1.3 Componentes Simulador

Los simuladores son componentes clave en el proyecto Roborescue dado que permiten moldear los eventos y las situaciones, implementan la lógica necesaria para emular variados contextos y son los encargados de procesar los comandos de acción de los agentes dado que son éstos y no estos últimos los conocederos de cuáles son las consecuencias de los actos que deciden llevar a cabo. Esto y mucho más podrá comprenderse mejor con el siguiente diagrama de clases:

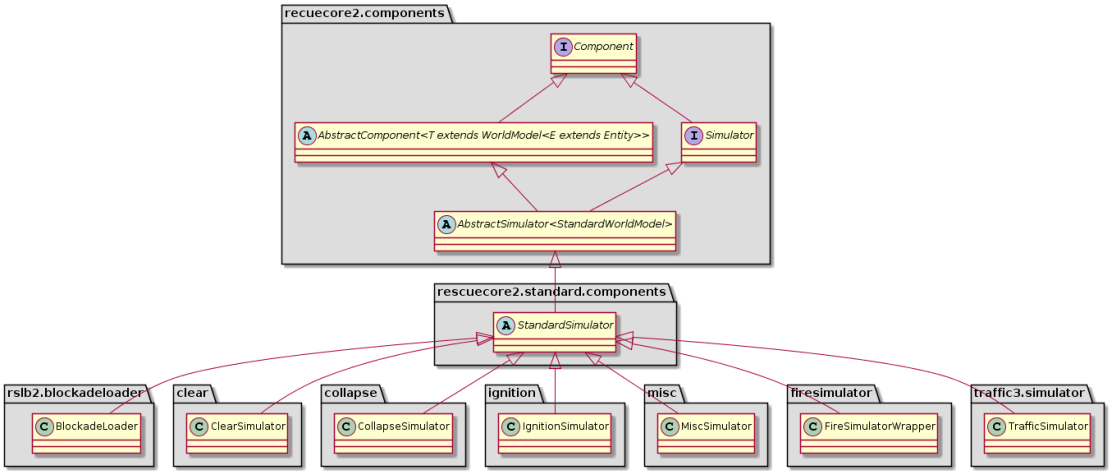


Figure 7: Familia simuladores estandard

En la figura 7 se muestra la jerarquía completa de lo que se puede catalogar como familia estandard de simuladores, siendo todos ellos herederos de la clase abstracta **StandardSimulator**. Tal y como se percibe en el diagrama los simuladores estandard implementan tanto la interfaz **Component** como **Simulator** lo cual les obliga a implementar por un lado un método de conexión y un método post conexión respectivamente. Estas firmas se producen porque todo componente está ideado para ser ejecutado como proceso externo al proceso **Kernel**, proceso que posibilita la coordinación no solo de simuladores sino también de agentes y componentes gráficos como se verá en secciones posteriores y cumple un papel esencial para el correcto funcionamiento de la plataforma, los componentes están obligados a implementar un método por lo tanto que les permita conectarse y mantener una comunicación con este último. El método post conexión firmado por la interfaz **Simulator** es requerido para desarrollar aquel conjunto de acciones que sea necesario antes de empezar a procesar las requests que le sean enviadas desde el kernel, son por lo tanto acciones de carácter configurativo o inicializador, de una única ejecución y que requieren de una conexión aprobada por el proceso central o kernel.

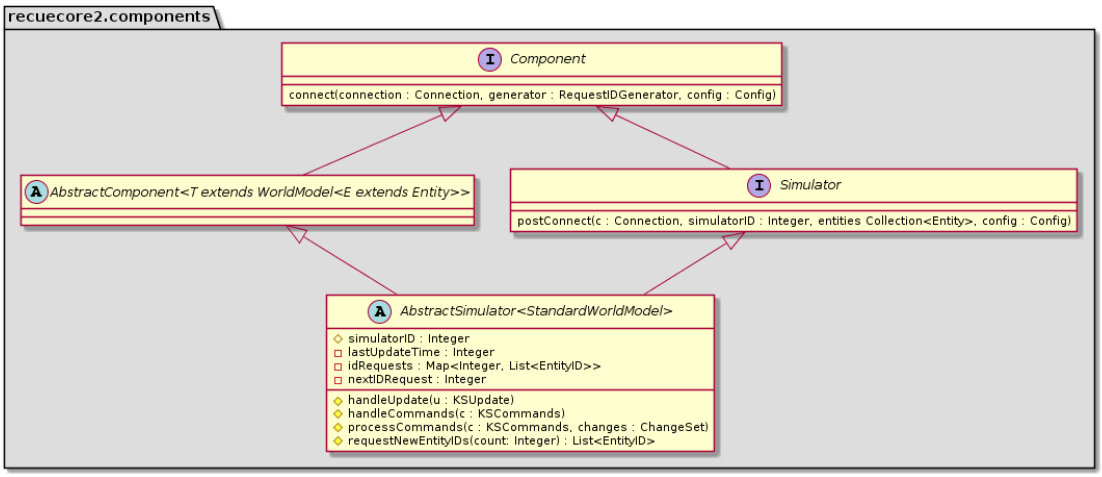


Figure 8: AbstractSimulator, detalle

En la figura 8 se muestra el detalle de la clase **AbstractSimulator**, también han sido incluidas las firmas de los métodos *connect* por parte de la interfaz **Component** y *postConnect* por parte de la interfaz **Simulator**, el resto de detalles han sido excluidos para una mejor legibilidad. El detalle de la clase abstracta **AbstractSimulator** muestra únicamente los métodos declarados por vez primera excluyendo por lo tanto aquellos heredados y/o sobreescritos.

Los métodos *handleUpdate*, *handleCommands* y *processCommands* junto a *connect* y *postConnect* permiten describir el comportamiendo básico de todos los simuladores estandard. Como se introducirá más adelante Simuladores y Agentes son Componentes y por ello comparten el comportamiento que denominaremos a partir de ahora ”proceso de conexión”, proceso mediante el cual se establece la comunicación con el Kernel y posibilitará el segundo proceso que denominaremos ”proceso específico” dado que es dependiente del tipo de componente y es sobreescrito según las clases herederas.

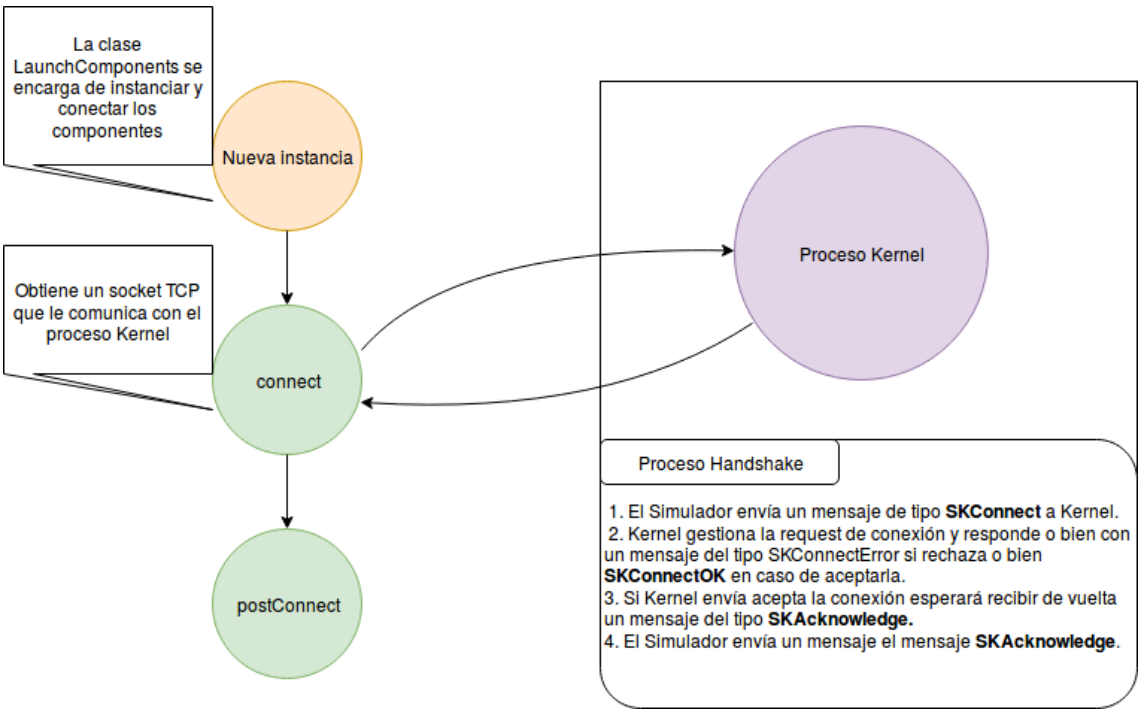


Figure 9: Proceso de conexión, Simuladores

La figura 9 muestra a modo a diagrama la relación comunicativa inicial entre ambos procesos, Simulador y Kernel, la esfera anaranjada representa la etapa de instanciación del Simulador, las esferas verdes las etapas de conexión ya desarrolladas dentro de la clase Simulador y de mayor tamaño y color violaceo se ha representado el proceso externo kernel. Primeramente cabe saber que todos los Simuladores son ejecutados mediante la clase main **LaunchComponents**, entre sus argumentos se introduce el nombre de la clase a instanciar, una vez instanciada se llama al método *connect* primero de la clase **LaunchComponent** luego dentro de éste se llama al método homónimo implementado por la clase Simulador instanciada. Una vez allí la adición de un listener de tipo **SimulatorConnectionListener** gestiona el *handshake* producido entre Simulador y Kernel. El diagrama especifica cuáles son los mensajes, el orden de los mismos y tal y como se introdujo este proceso de conexión es común al resto de componentes con la salvedad de que los mensajes intercambiados con el proceso kernel son de un tipo distinto según el tipo del componente, así como los simuladores envían un mensaje **SKConnect** los agentes hacen uso del mensaje **AKConnect** pero la función de ellos permanece análoga. En caso de ser componente Agente los mensajes intercambiados son: AKConnect y AKAknowledge emitidos por parte del Agente, AKConnectOK o AKConnectError por parte del Kernel. En caso de ser componentes GUI los mensajes son: VKConnect y VKAknowledge, en este caso emitidos por el componente gráfico y Kernel emitiría uno de tipo VKConnectOK o VKConnectError. El listener usado en los distintos casos mantiene esta afinidad por lo que respecta al comportamiento pero en caso de ser Agente el componente se vale del **AgentConnectionListener** y en caso de ser componente gráfico **ViewConnectionListener**.

Ahora, por lo que respecta al ”proceso específico” de los componentes de tipo Simulador véase el siguiente diagrama:

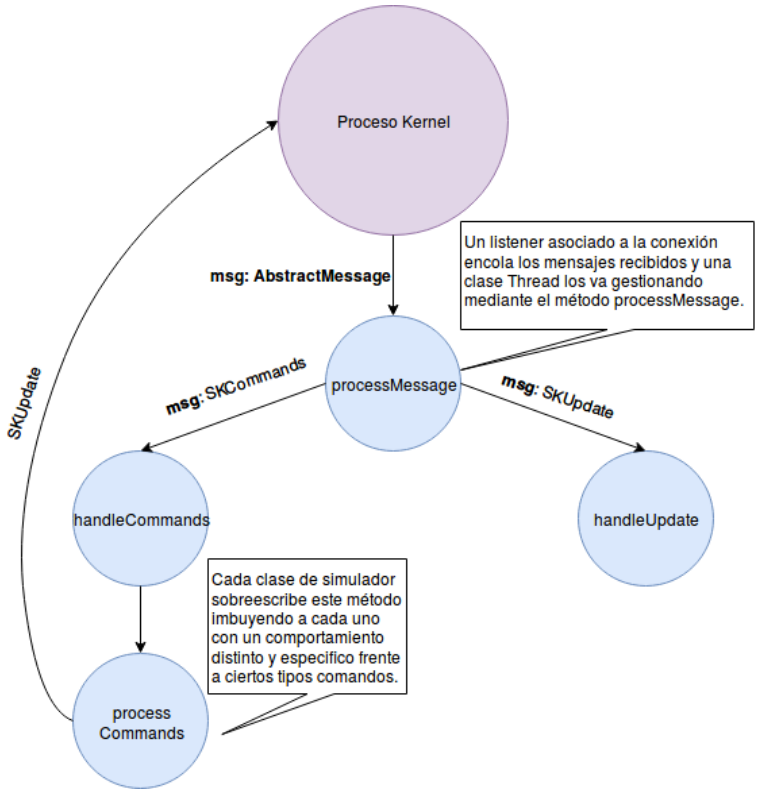


Figure 10: Proceso específico, Simuladores

Esquemáticamente la figura 10 muestra cuál es la rutina que lleva a cabo todo simulador estandard una vez que queda conectado con kernel. La clase **AbstractComponent** define dos clases internas que se ocupan de gestionar los mensajes que van llegando por parte del kernel. Una de ellas se denomina **MessageListener** y queda asociada a la comunicación TCP establecida y ya aceptada, se encarga de encolar los mensajes en una colección de ”mensajes por procesar” a medida que estos son recibidos. La otra clase denominada **MessageProcessor** hereda de Thread en última instancia y queda a cargo de procesar los mensajes mediante la llamada al método *processMessage*, éste es a su vez sobreescrito por AbstractSimulator para de este modo especificar sobre qué tipo de mensajes actuará y de qué modo (en la siguiente sección se percibirá una estrategia análoga por lo que refiere a los componentes Agentes). En el caso que atañe, los Simuladores, implementan una gestión de los comandos **SKCommands** y **SKUpdate** mediante los métodos *handleCommands* y *handleUpdate* respectivamente. El comando de tipo **SKCommands** recoje una colección de comandos enviados por los componentes agentes. Cada simulador en específico gestiona uno o varios de estos comandos. El comando **SKUpdate** contiene una serie de cambios encapsulados en una entidad denominada **ChangeSet** que a modo de diccionario mapea las entidades y los cambios producidos en sus propiedades, el receptor de este tipo de mensaje realiza un ”merge” de los cambios con la instancia del ”mundo” sobre la que opera, dicha instancia es de la clase **Stan-**

dardWorldModel que a modo de colección recoge dentro de sí todas las entidades que componen el mundo de la simulación, y ello permite una sincronización ente el mundo del Simulador y el mundo del Kernel. Dado que los simuladores al gestionar las comandas de los agentes mediante el método *processCommands* generan cambios sobre el "mundo" estos a su vez crean una instancia **ChangeSet** con los cambios producidos y envían esta a Kernel para que los tenga en consideración.

Los simuladores estandard usados durante el desarrollo de este trabajo serán a continuación brevemente introducidos y se especificará qué tipo de comandos de agente son capaces de gestionar.

- **BlockadeLoader**: desarrollado en un proyecto aparte se encarga de generar entidades de tipo **Blockade** bloqueantes que siempre ocupan un area igual a la calle (entidad **Road**) en la que se ubica. Permite configurar un escenario con bloqueos y cercionarse de que éstos serán siempre bloqueantes. No gestiona ningún comando de componente agente, su papel entra en acción cuando la especificación del escenario es procesada.
- **ClearSimulator**: gestiona los comandos del tipo **AKClear**, estos describen la acción de limpiar un bloqueo, el simulador se encarga en este caso de chequear si los requisitos permiten la realización de la acción y si es así ejecuta la eliminación del bloqueo.
- **CollapseSimulator**: permite simular daños en estructuras tanto por causa de las llamas provocando rotura en edificios incendiados como por efecto de seísmo. Además emula, mediante la generación de bloqueos de formas irregulares y a menudo bloqueantes en la entrada de edificios, la caída de cascotes. No gestiona el comando de ningún componente agente.
- **MiscSimulator**: se encarga de simular las diversas lesiones que pueden sufrir los agentes gestionando bajo qué contexto puede un agente lesionarse, cuál debe ser el tipo de la lesión, así como el progreso de la misma a lo largo del tiempo. Para ello interactúa con las propiedades de *hp*, *damage* y *buriedness* de las entidades humanas controladas por los agentes. Respecto a los comandos de agente procesados únicamente se encarga del de tipo **AKRescue** y lo hace reduciendo los niveles de *buriedness* del objetivo especificado en el comando.
- **FireSimulatorWrapper**: simula los efectos que las acciones de extinción tienen sobre los edificios y para ello implementa la gestión del comando **AKExtinguish**.
- **TrafficSimulator**: se encarga de simular el movimiento de los agentes. Define las velocidades para los distintos tipos, otorgándoles una velocidad muy superior a los agentes de rescate (bomberos, policías y equipo sanitario) que al civil. Gestiona los comandos de tipo: **AKMove**, **AKLoad**, **AKUnload**, **AKRescue**, **AKClear** y **AKExtinguish**.

- **IgnitionSimulator**: simula el efecto de los incendios sobre las estructuras, además es capaz de generarlos y regular la intensidad emulando la propagación de incendios y una graduación de estados según va evolucionando el incendio y la gravedad de las llamas se intensifica.

A modo de síntesis es importante recordar los siguientes puntos: las clases componente en el proyecto Roborescue son todas aquellas que implementan la firma de la interfaz **Component**. Todo componente está ideado para ser ejecutado en un proceso aparte del proceso Kernel pero que es capaz de mantener una comunicación con él por lo que requiere de un proceso de conexión que consiste en un handshake. Todo componente opera sobre una instancia propia de **StandardWorldModel** que viene a ser una representación del mundo simulado. La sincronización de la instancia propia del mundo con la instancia que mantiene el proceso kernel se realiza mediante el envío, de kernel a los simuladores, del comando **SKUpdate** en el caso de los simuladores (y **KASense** en el caso de los agentes) y a su vez los simuladores registran cambios al operar sobre su instancia, los encapsulan dentro de una clase denominada **ChangeSet** y son enviados al kernel mediante un comando **SKUpdate**. En este caso los emisores son los simualdores y el kernel el receptor.

Llegados a este punto se concluye que la plataforma de simulación Roborescue es la composición de múltiples componentes donde tenemos: una serie de componentes simuladores que moldean distintos aspectos y eventos, componentes agentes y componentes gráficos. Todos ellos regulados y sincronizados mediante un proceso central o kernel.

Para finalizar esta sección serán introducidos brevemente los simuladores no estandard usados: **ChannelComunicationModel** que permite simular canales de comunicación de radio (**RadioChannel**) o voz (**VoiceChannel**). Los agentes pueden subscribirse mediante el comando **AKSubscribe** y pueden usarlos mediante el comando **AKSpeak** y **StandardPerception** que permite simular qué les es visible a los agentes y qué no, actualizando únicamente las entidades que ven basándose en la distancia que hay entre las entidades y el agente.

4.1.4 Componentes Agentes

Los agentes implementan la interfaz **Component** por lo tanto, y tal y como ya fue comentado en el apartado anterior, comparten el proceso de conexión ya visto.

Las relaciones jerárquicas son las siguientes:

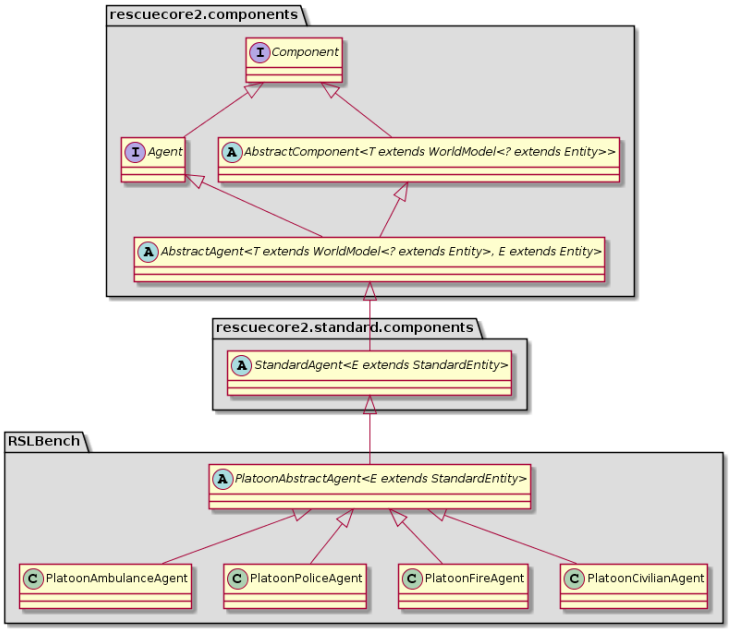


Figure 11: Relaciones jerárquicas, Agentes

Como se obeserva las clases agente son todas aquellas que implementan la firma de la interfaz **Agent**, que participa a su vez de la interfaz **Component**. El árbol genealógico de los agentes muestra la dependecia del proyecto RSLB2 respecto al proyecto Roborescue dado que el package RSLBench se halla en el primero y los otros dos que aparecen en la figura 11 al segundo. También se muestra cómo la clase abstracta **PlatoonAbstractAgent** y sus ancestros hacen uso de un poliformismo basado en subtipado siendo este tipo heredero en primer lugar de la clase abstracta **StandardEntity**, y las cases agente específicas especifican la entidad humana que manipulan en cada caso. Así, la extensión particular queda del siguiente modo:

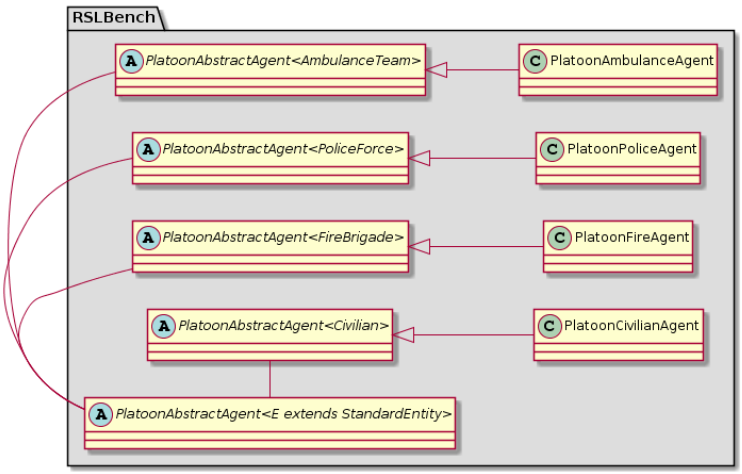


Figure 12: Extensión particular de cada Agent específico

La figura 12 muestra cada subtipado en particular acorde con la entidad que

cada agente controla. Todo agente específico acompaña su request de conexión (mensaje **AKConnect**) al kernel con el tipo/s de entidad/es que controla mediante la sobreescritura del método *getRequestedEntityURNsEnum()* devolviendo en cada caso un enum con el nombre de la entidad humana, "FireBrigade" en el caso de **PlatoonFireAgent**, "PoliceForce" en el caso de **PlatoonPoliceAgent** y de modo acorde el resto de agentes. En el proyecto base RMASBench únicamente estaban implementadas las clases **PlatoonFireAgent** y **PlatoonPoliceAgent** con un comportamiento base que en el caso de los agentes de bombero los hacía moverse hacia el refugio si es que se quedaban sin agua, aplicar la acción de extinguir (uso del comando **AKExtinguish**) cuando el objetivo seleccionado por el Solver DCOP se hallaba dentro de su alcance y en caso contrario acercarlos al objetivo mediante la acción moverse (uso del comando **AKMove**) . El comportamiento del agente de policía es análogo. Más adelante fueron implementados los agentes **PlatoonCivilianAgent** y **PlatoonAmbulanceAgent** porque ciertas normas seleccionadas les condieraban y por lo tanto fue necesario incluirlos en la simulación.

Dado que el proceso de conexión ya ha sido explicado en apartado anterior y presenta un comportamiento semejante para todos los componentes, a continuación se expondrá el "proceso específico" de los componentes Agentes pero antes es necesario exponer el detalle de las clases responsables de esa rutina.

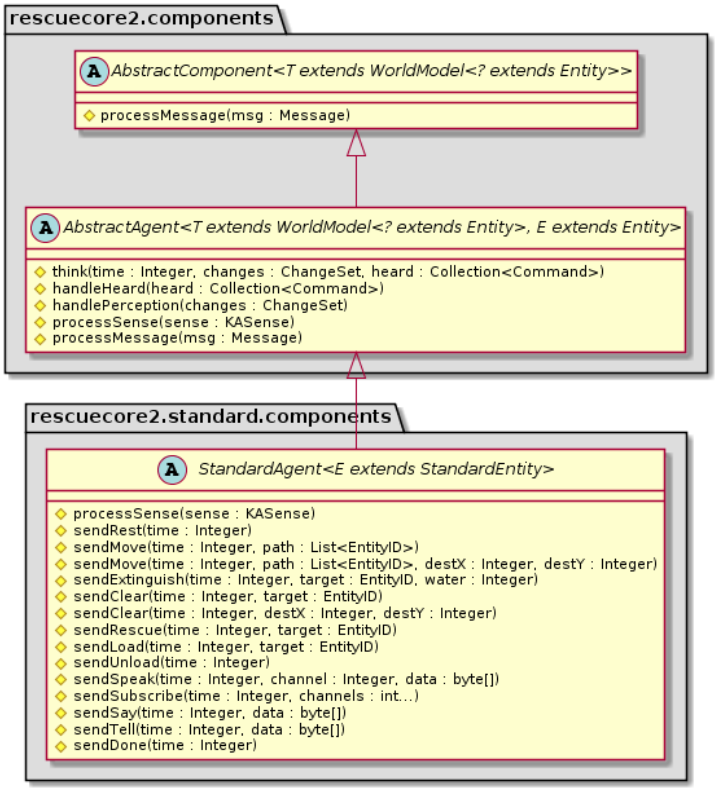


Figure 13: Detalle 1, Componentes Agentes

La figura 13 muestra los métodos más relevantes sin incluir atributos de clase para mejorar la legibilidad. Tal y como se mostraba en la sección anterior, el método

processMessage a cargo de la clase **MessageProcesor** es heredado desde la clase abstracta **AbstractComponent** y en este caso sobreescrito por la clase igualmente abstracta **AbstractAgent** quien realiza una reeimplementación para gestionar los comandos recibidos de tipo **AKSense**. El comando **AKSense**, enviado por el proceso Kernel y recibido por los componentes agentes, encapsula lo que se denominará a partir de ahora como "información sensible"o "información sensible del agente" puesto que dentro del comando se hallan datos sobre lo que es percibido sensiblemente y éstos se dividen en: "visible" (lo visible) y "heard" (lo escuchado), que representan respectivamente aquello que visiblemente percibe como distinto (y es encapsulado mediante un objeto **ChangeSet**) y aquello que ha podido escuchar mediante un objeto Collection de tipo Command que debería contener comandos tipo **AKSpeak** o bien **AKTell** o **AKSay** según el modelo de comunicación usado. La gestión del comando **AKSense** queda a cargo del método *processSense* quien se encarga de realizar el "merge" de los cambios visibles en la representación del mundo sobre la que opera el agente (semejante al caso de uso **SKUpdate** de la sección anterior), de ejecutar los métodos *handleHeard* y *handlePerception* que han sido incluidos en la realización de este trabajo, de ejecutar el método think que es sobreescrito por cada agente específico dotándole de un comportamiento y finalmente envía el comando **AKDone** mediante el método *sendDone* como extensión de la lógica realizada por la clase **StandardAgent** sobreescribiendo el método *processSense*.

Esquemáticamente puede visualizarse del siguiente modo:

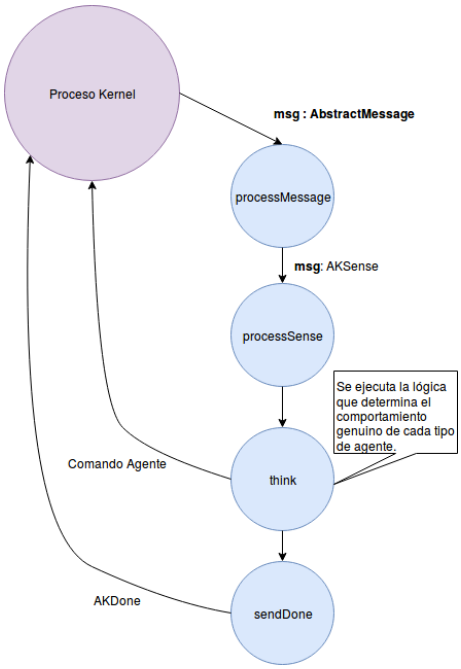


Figure 14: Proceso específico 1, Agentes

El comando enviado durante la ejecución del método *think* por parte de la clase agente específica hace uso de los múltiles métodos de envío implementados por la

clase abstracta **StandardAgent** (Véase la figura 13). Aunque la implementación permite envío indiscriminado de comandos sin importar qué tipo de entidad controla el agente específico la realidad es que cada agente tiene un abanico de acciones y no otro, la lógica de los simuladores interceptaría el intento de realizar una acción no disponible para un cierto tipo de entidad humana, si es que esta tratase de llevarla a cabo, impidiéndolo.

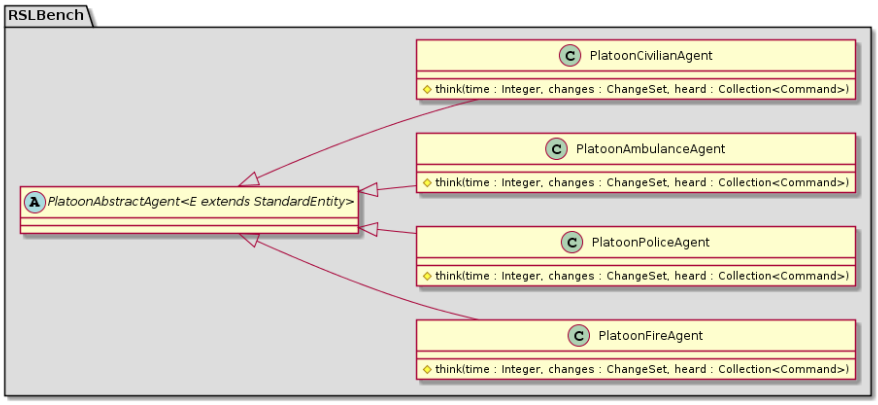


Figure 15: Comportamiento espcífico 2, Agentes

La figura 15 muestra únicamente la sobreescritura del método *think* por parte de cada tipo de agente. La lógica incluida en su interior define el comportamiento de cada uno de ellos y como se comentó con anterioridad el proyecto base RMASBench ya contaba con las clases agente de bomberos y agente policía con una lógica por defecto que basaba los objetivos según el algoritmo resolvedor configurado.

Ahora sigue el abanico de acciones del que dispone cada agente según su tipología.

Agente sanitario:

- Rescue: permite liberar a otro agente humano inmovilizado por los efectos de colapso de una estructura de tipo edificio (entidad **Building**). La acción de rescate se puede aplicar sobre todo tipo de agente humano y las principales limitaciones son: no es aplicable sobre uno mismo y no es aplicable sobre un objetivo que no se halle adjacente a la posición del agente sanitario. El comando asociado es **AKRescue** y el envío de dicho comando es encapsulado bajo un método denominado *sendRescue* (véase la figura 13).

- Load: permite cargar a otro agente humano a la ambulancia, la posición del agente pasajero será aquella que tome el agente sanitario desde el momento en que el primero haya sido cargado. La principal limitación de esta acción es que el agente humano sobre el que es aplicable esta acción solo puede ser de tipo civil (entidad **Civilian**), solo es posible portar un único pasajero y el objetivo de esta acción debe hallarse adjacente al agente sanitario. El comando asociado es **AKLoad** y el envío de dicho comando es encapsulado bajo un método denominado *sendLoad* (véase la figura 13).

- Unload: permite desencadenar la posición del pasajero con la del agente sanitario de modo tal que el agente sanitario y el agente que portaba pueden moverse libre e independientemente. La posición del agente pasajero despuúes de serle aplicada la acción Unload será aquella en la que se halle el agente sanitario. El requisito para ejecutar esta acción es tener un pasajero. El comando asociado es **AKUnload** y el envío de dicho comando es encapsulado bajo un método denominado *sendUnload* (véase la figura 13).

Agente de bomberos:

- Refill: permite al agente rellenar el tanque de agua. Su principal limitación es que esta tan solo es practicable en determinadas estructura: el refugio (entidad **Refuge**) y en una calle con hidratantes mediante el uso de la entidad **Hydrant**. El tanque tiene una capacidad máxima por lo que cualquier acción de este tipo en este caso no tendrá efecto alguno. No tiene asociado comando alguno pero **FireSimulatorWrapper** determina que este evento se realice.
- Extinguish: permite al agente disparar agua contra una estructura y de este modo reducir la intensidad de las llamas o extinguirlas completamente. Las principales limitaciones para su realización son: tener una cantidad de agua positiva y mayor que cero en el tanque, el objetivo debe ser una estructura de tipo edificio (solo los edificios son candidatos a incendiarse) y la distancia entre agente e estructura objetivo no debe rebasar un cierto valor. El comando asociado es **AKExtinguish** y el envío de dicho comando es encapsulado bajo un método denominado *sendExtinguish* (véase la figura 13).

Agente de policía:

- Clear: permite al agente remover un bloqueo. Los bloqueos en la simulación son representados mediante la entidad **Blockade**. Para que el agente pueda realizar la acción este debe estar en una posici^{o}'on adjacente a la entidad bloqueo. El comando asociado es **AKClear** y el envío de dicho comando es encapsulado bajo un método denominado *sendClear* (véase la figura 13).

Los agentes además son capaces de obtener estados alterados que en el lenguaje de la simulación son considerados como lesiones (en inglés injuries). Estos estados alterados son provocados por distintos eventos y sus efectos también varían aunque todos ellos tienen en común una reducción de la salud de la entidad contralada por el agente afectado. Los tipos de lesiones son:

- Fire Injury: simula el perjuicio que procarían las quemaduras sobre un ser humano. Aplica un daño que va progresando con el tiempo y tan solo incrementa en intensidad en función de la temperatura del edificio que la ha provocado. Estos efectos son capaces de provocar el fallecimiento de entidades humanas que ya no se encuentran cerca de los incendios como pudiese ser en edificios donde las llamas ya fueron extinguidas u otra estructura sin quemar.

- Collapse Injury: simula el perjuicio que tendría quedar atrapado por fragmentos generados por fracturas en un edifico. Aplica un daño únicamente cuando un evento de rotura se produce sobre un edifico en el que se está ubicado, la cantidad de daño es variable aunque guarda relación con la gravedad en que haya quedado al edificio en cuestión. Además de inflinjir esta reducción de la salud puntal provoca un estado de inmovilización por lo que el agente en cuestión no es capaz de realizar acción hasta que la entidad controlada sea rescatada.

- Buried Injury: simula el perjuicio de ser enterrado en su totalidad. Puede producirse cuando un edifcio alcanza niveles de colapso críticos y se está en su interior. El daño vuelve a ser puntual, hay muchas probabilidades de perder todos los puntos de salud y tal y como en el caso anterior se produce una inmovilicación.

4.1.5 Kernel

El proceso Kernel se inicia mediante la clase main **StartKernel**. Su importancia reside en el hecho de que opera como mediador, sincronizador y coordinador. Mediador porque se encarga de recibir los comandos de acción que formulan los agentes y acaba listándolos y encapsulándolos en un único comando del tipo **SKCommands** ya visto en la figura 10 que acaba siéndoles enviado a los componentes simuladores. Sincronizador y coordinador, en primer lugar porque ordena las operaciones asíncronas entre procesos de tal modo que no se generan incongruencias dentro de la simulación, y en segundo lugar porque procura la consistencia de la instancia del mundo de la simulación individual de cada componente mediante los comandos de actualización ya sea a través del comando **SKUpdate** en el caso de los simuladores o **AKSense** en los agentes. Esto último es importante tenerlo en cuenta, la instancia del mundo sobre la que recaen todas las modificaciones provocadas por los simuladores es aquella que tiene kernel, una vez kernel "mergea" todos los cambios envía a todos los componentes el conjunto de cambios a actualizar porque únicamente mantienen los componentes comunicación con kernel y de otro modo no serían indiferentes a los cambios no producidos por ellos mismos.

Kernel realiza múltiples operaciones pero por lo que respecta a este trabajo y la significación respecto a este será únicamente introducido su core; que consiste una serie de operaciones que se repiten una y otra vez generando iteraciones en la simulación que más adelante en el trabajo serán denominadas con el nombre de TIMES, siendo un TIME una sola de estas iteraciones.

El Core del Kernel es implementado por la clase **Kernel**, la lógica se localiza en su método *timestep* y a continuación se enumerarán los pasos que contiene en orden cronológico de ejecución:

1. Incrementa en una unidad el atributo time (valor por defecto 0).

2. La percepción de todos los agentes es actualizada mediante la construcción particular de mensajes del tipo **AKSense** valiéndose de **StandardPerception** para obtener los cambios visibles, y el modelo de comunicación para recojer los mensajes escuchados ya sea a través del canal de radio o voz. Una vez construido se les es enviado.

3. Se espera a recibir todos los comandos enviados por los agentes y van siendo añadidos en una colección a medida que llegan. Recuérdese en este punto que el comando **AKDone** enviada por el componente agente permite a kernel reconocer que no espera más comandos suyos hasta la próxima iteración.

4. Una vez ha recibido el mensaje **AKDone** por parte de todos los agentes construye un mensaje del tipo **SKCommands** con todos ellos e iterando por cada uno de los simuladores les es enviado.

5. Se espera recibir un mensaje del tipo **SKUpdate** por parte de todos los simuladores (véase la figura 10). Todos los cambios particulares son unificados en un único objeto **ChangeSet** que pasa a contener por lo tanto todos los cambios producidos por los simuladores ese timestep.

6. La instancia **ChangeSet** es "mergeada" en la instancia de **StandardWorldModel** o "mundo" de la simulación obteniendo así la versión actualizada.

7. Se se envía un comando **SKUpdate** a cada simulador para darles a conocer cuales son todos los cambios producidos y así coordinar la instancia actualizada de kernel con la individual de cada componente simulador.

8. Los cambios acontecidos son encapsulados y enviados a los componentes gráficos para que de este modo puedan renderizar el nuevo estado de la simulación. El mensaje que encapsula los cambios es de tipo **KVTimestep**.

4.1.6 Configuración

El proyecto RMASBench permite arrancar la plataforma, el proyecto Blockade-Loader y RSLB2 desde este último con el uso de un script denominado *start.sh* ya definido y mencionado en el archivo README del proyecto base. El hecho es que la plataforma Roborescue permite la configuración de múltiples características que aplican tanto a los distintos componentes simuladores como agentes, también sobre kernel. El script accede a la carpeta *config* que se halla en la ruta */RMAS-Bench/RSLB2/boot*. Para la realización de este trabajo se han usado las configuraciones por defecto del proyecto RMASBench y el script *functions.sh* del que hace uso *start.sh* ha sido extendido pero no se ha alterado el contenido por defecto. Lo anterior mencionado implica el uso del archivo de configuración *misc.cfg* para ejecutar **MiscSimulator**, *traffic3.cfg* para **TrafficSimulator**, *resq-fire.cfg* para **FireSimulatorWrapper**, *clear.cfg* para **ClearSimulator** y *collapse.cfg* cuando **CollapseSimulator** fue incluido.

Toda simulación se realiza sobre un mapa. El mapa usado puede intercambiarse por otros, sin embargo, todas las pruebas que serán presentadas en este trabajo han sido realizadas usando siempre el mismo: París. Los archivos que describen los mapas son de extensión *.gml*, todos ellos pueden encontrarse en la ruta */RMASBench/roborescue/maps/gml*. Los mapas que se hallan en el interior del directorio son: *berlin*, *kobe*, *Kobe2013*, *legacy*, *meijo*, *paris*, *shouthampton* y finalmente uno denominado *test*. Es importante mencionar tal y como advierte el archivo README del proyecto base RMASBench que el uso de un nuevo mapa puede demorar la ejecución una cantidad considerable de tiempo. La tardanza únicamente afecta la primera simulación con el susodicho. El motivo de que el tiempo de ejecución se dilate en este caso se debe a un proceso de optimización que resulta en la generación de dos archivos denominados *<map_name>-BreadthFirstSearch.paths* y *<map_name>-BreadthFirstSearch.paths.p*, pueden encontrarse en la ruta *RMASBench/RSLB2/boot/cache*. Codifican información relativa a las múltiples conexiones dentro del grafo que es el mapa y las siguientes operaciones resultan menos costosas. Es importante remover los archivos caché si es que la ejecución primera fuese interrumpida.

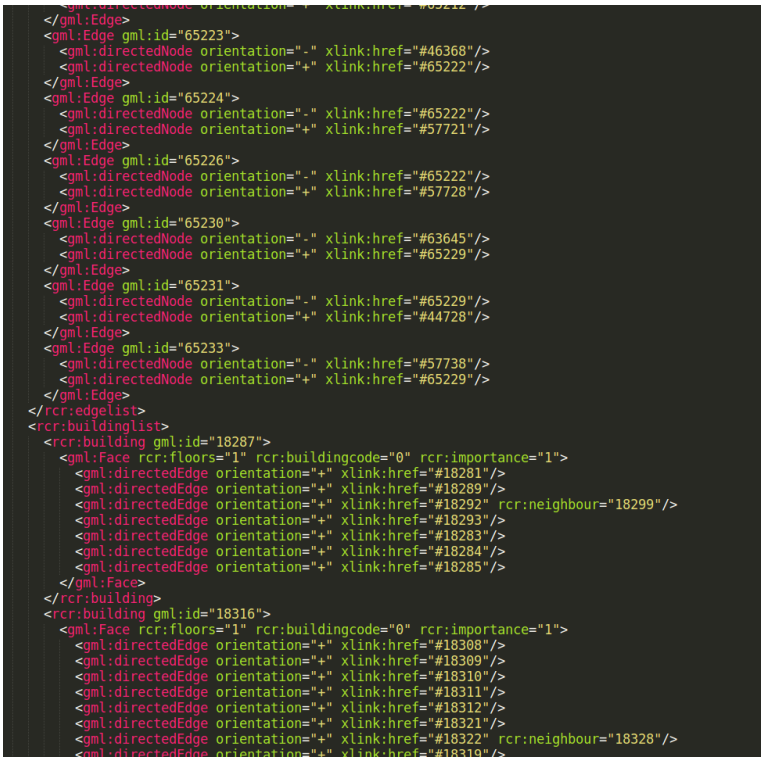


Figure 16: Archivo gml, mapa París

La figura 16 es una imagen extraída del archivo *map.gml* de la ciudad de París. Analizando la defnición del archivo completo se oberva como son definidos elementos del tipo **Node** o nodo y arista o **Edge** en primer lugar y posteriormente elementos **Road** y **Building**. El mapa se compone de la pareja primera de elementos primitivos conformando un grafo dirigido fíjese, pues, como la imagen especifica la orientación de la entidad **Edge** en cada caso. Toda endentidad como muestra en la

figura cuenta con un valor asociado a la propiedad *gml:id* que permite identificarlas únivocamente. Las aristas definen mediante la propiedad *xlink:href* su conexión con otras entidades primitivas. Despueés se proveen la definición de edificios y calles, relacionando cada una de estas entidades con múltiples elementos del tipo arista, esto último puede observarse en los subtags detallados dentro del tag superior *rcr:building* en cada caso. Estas aristas pueden tener asociado o no un valor a la propiedad *rcr:neighbour*, si lo tiene este será un idetificador de otro edificio o calle igualmente definido en el mapa del tal modo que quedarán conectados.

Además es de relieve conocer que es posible configurar las simulaciones para disponer múltiples entidades por todo el mapa, localizándolas donde sea de mayor interés. Para harcelo se hace uso de los archivos escenario que son de extensión *.xml* y si la ejecución arraca desde el proyecto RSLB2 tal y como será explicado en el apartado *Up Running proyecto RMASBench* la localización de este tipo de archivos se encuentra en la ruta */RMASBench/RSLB2/scenarios*.

```
:scenario:scenario xmlns:scenario="urn:roborescue:map:scenario">

  <scenario:aftershock scenario:time="15" scenario:intensity="0.2"/>

  <!-- The fire station (do *not* remove this) -->
  <scenario:firestation scenario:location="64173"/>

  <!-- The refuge -->
  <scenario:refuge scenario:location="18421"/>

  <!-- ambulance centre -->
  <!--<scenario:ambulancecentre scenario:location="18421"/>-->

  <!-- The initial fires -->
  <scenario:fire scenario:location="35014"/>
  <!--<scenario:fire scenario:location="64580"/>-->
  <scenario:fire scenario:location="39441"/>

  <!-- The fire brigades -->
  <scenario:firebrigade scenario:location="35014"/>

  <scenario:firebrigade scenario:location="10345"/>
  <scenario:firebrigade scenario:location="10345"/>
  <scenario:firebrigade scenario:location="10345"/>

  <scenario:firebrigade scenario:location="14737"/>
  <scenario:firebrigade scenario:location="14737"/>
  <scenario:firebrigade scenario:location="14737"/>

  <scenario:firebrigade scenario:location="18220"/>
  <scenario:firebrigade scenario:location="18220"/>
  <scenario:firebrigade scenario:location="18220"/>
```

Figure 17: Ejemplo archivo escenario

La figura 17 muestra un archivo escenario para ejemplificar en que consisten estos. Mediante tags resulta posible detallar entidades humanas y estructuras siempre y cuando estas hereden de o bien la clase **Building** o bien **Road** (véase 5). Además es necesario localizar las entidades detalladas mediante la propiedad *location*, el valor asociado es siempre el identificador de una entidad edificio o calle especificada en el archivo mapa *map.gml* en el que se piense usar el escenario. Cuando se tratan estructuras su posición debe ser congruente con su tipo o de lo contrario un error del tipo *CastClassException* provocará la fallida de la simulación en su ejecución temprana cuando el mundo simulado está siendo construido. Se es congruente cuando la entidad estructura definida en el escenario puede ser casteada a la entidad estructura asociada en su posición, si la localización es de tipo *Building* solo sus subclases serán válidas y de igual modo con la calles.

4.1.7 Robocup Rescue UI

La plataforma Robocup cuenta con una interfaz gráfica que permite visualizar la evolución de las simulaciones. En apartados anteriores fue mencionado la existencia de componentes gráficos que como el resto de componentes interactuan con el proceso kernel y estos en particular permiten graficar el cotexto emulado TIME a TIME.

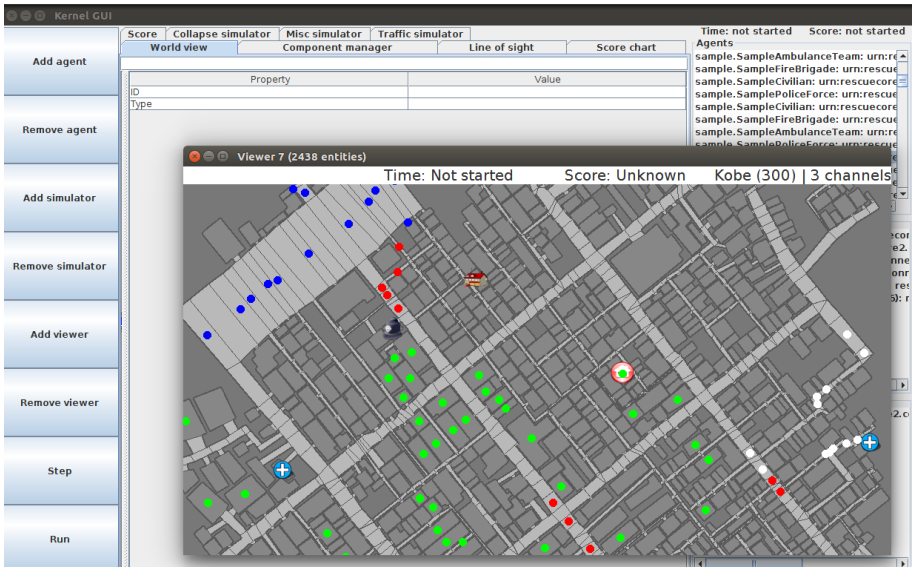


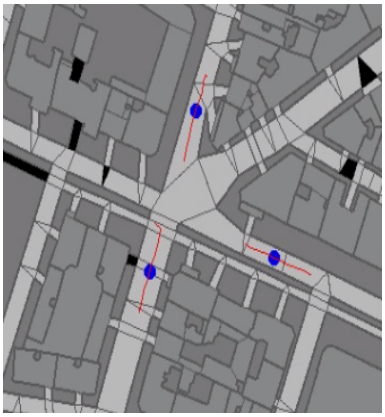
Figure 18: Interfaz gráfica de configuración, plataforma Robocup Rescue

La figura 18 muestra la interfaz gráfica de configuración de la plataforma Robocup Rescue Simulator. Esta interfaz no ha sido utilizada en este trabajo porque se ha trabajado arrancando la simulación desde el proyecto RSLB2 y desde allí se realizan las tareas configurativas. Más adelante en este mismo trabajo se mostrarán capturas relativas a la evolución de las simulaciones con las que se han realizado varios experimentos por lo que es necesario introducir cómo visualmente representa la interfaz cada entidad:

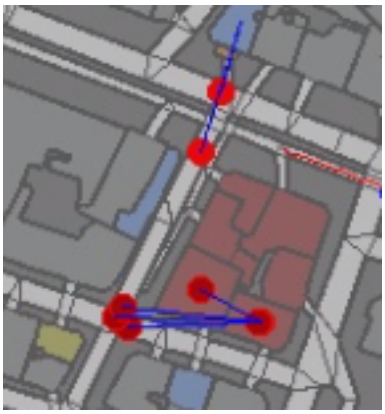
Figure 19: Representación gráfica, Entidades

La figura 19 muestra a modo de leyenda de la figura 18 cual es la representación de cada entidad según su tipología.

Además las acciones tomadas por los agentes y los estados alterados de las entidades humanas y estructurales también obtiene una representación visual. Véanse las siguientes:



(a) Tres policías en movimiento



(b) Múltiples agentes extinguiendo incendios

Figure 20: Representación gráfica, acciones y estados

La figura 20 permite ilustrar como son representadas gráficamente tanto la acción de moverse como los bloqueos en el caso la imagen *(a)* y los estados alterados de las estructuras provocados por la acción del fuego y la acción de extinción en el caso de la imagen *(b)*.

En la figura *a* concretamente se muestra un total de tres agentes de policía en movimiento. La acción de moverse tal y como se observa se representa mediante líneas rojas que parten de la posición del agente y llegan hasta el edificio o calle que ha logrado alcanzar durante la última iteración. Cada TIME los agentes de rescate (bomberos, policías y sanitarios) en movimiento pueden desplazarse varias calles en el mapa. Los bloqueos representados mediante la ya conocida entidad **Blockade** aparecen como zonas totalmente negras.

En la figura *(b)* aparecen múltiples agentes de bomberos extinguiendo simultáneamente varios edificios incendiados. Tal y como muestra esta figura la acción de extinción es representada mediante un línea azul que originándose en el agente actuador termina en el incendio objetivo. Por otra banda también aquí pueden distinguirse varios colores con los que se representan los distintos estados de los edifcios afectados por las llamas. Los edificios que aparecen en la figura de un color rojo intenso son aquellos gravemente afectados, en una situación crítica pues to que el el siguiente estado si la situación no se remedia es el de calcinado. Un edificio calcinado se mostrará totalmente oscuro similar a lo que sería un bloqueo. Antes de alzancanzar una situación crítica el edificio en llamas ha pasado por dos estados anteriores: uno de gravedad moderada en el que el edificio manifiesta un color anaranjado y antes de este un estado inicial de color amarillo que implica daños menores. Esta figura muestra un único edificio con un nivel de daño inicial justo hacia la esquina inferior

izquierda de la imagen. Además los edificios extinguidos toman colores azulados, claro cuando el incendio apagado era leve, algo más oscuro cuando era moderado y por último más bien oscuro cuando el edifico sufría de una situación límite. En la figura aparecen un total de tres edifcios levemente afectados y extinguidos.

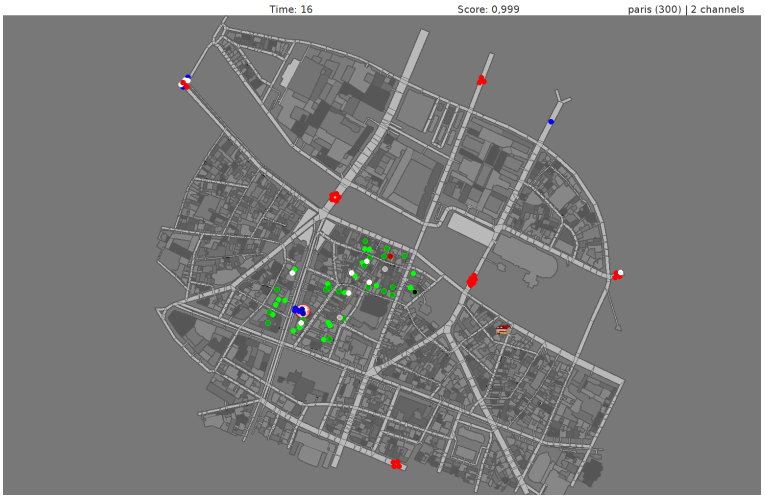


Figure 21: Ejemplo de seísmo y dañados estructurales por colapso

Otro estado alterado del que son susceptibles los edificios es el de rotura. En la figura 21 aparece un escenario de catástrofe por seísmo usando **CollapseSimulator**. El gris suave por defecto de los edificios se torna más oscuro en la medida en que es mayor la gravedad de sus fracturas, es decir, la propiedad *brokenness* tiene un valor mayor. La plataforma de simulación cataloga los estados de colapso como: destruido si la rotura supera el 75%, si es inferior pero superior al 50% estaría gravemente danñado, moderadamente si es inferior al 50% y superior a 25%, en último lugar se condiera levemente perjudicado un edificio con un nivel de rotura inferior al 25%.

4.1.8 Up & Running proyecto RMASBench

El proyecto se orquesta del siguiente modo: en primer lugar el comando ./start.sh [-v] [-b] -c [CONFIG FILE] -m [MAP] -s [SCENARIO] ejecuta el script start.sh que se halla en el directorio RLSB2/boot. El opcional -v permite habilitar la interfaz gráfica. El opcional -b permite habilitar bloqueos si es que el escenario que tratamos de usar los ha definido. Al modificador -c le sigue el archivo de configuración.cfg, en el README es usado un archivo llamado example.cfg y que ha sido usado en este trabajo sin editarse. Seguido del modificador -m el nombre de un directorio que contie- ne un archivo .gml que especifica los nodos en coordenadas y los vértices que les unen a modo de grafo que configura una estructura que termina modelando el mapa del escenario. Finalmente seguido del modificador -s se espera un archivo escenario con extensión .xml donde es posible usar etiquetas para describir la creación de una gran variedad de entidades además de ser posible determinárseles ciertas

propiedades. Por defecto en el proyecto RSLB2 vemos dos escenarios de Paris: example-nopolice y example-police, el primero establece únicamente una estación de bomberos, fuegos y bomberos, el segundo incluye además bloqueos y policías.

5 Normativa

En esta sección van a introducirse las normas seleccionadas por el software selector, recuérdese que el conjunto de normas está desclarado en el paper [2], exáctamente entre las páginas 64 y 70. Cada normativa está formulada como proposición y cada una genera tres reglas porque se contemplan la versión obligatoria, de facultad y prohibitiva de la misma. En el paper puede obervarse que las reglas están presentadas en bloque según al agente al que aplican, en esta sección se mantendrá esta estructura para presentarlas y mostrar la interpretación que se ha hecho de ellas.

Durante la realización de este trabajo dos conjuntos lijeramente distintos fueron seleccionados. Inicialmente el selector tomó un subconjunto del set original pero una vez se avanzó y las primeras pruebas con el conjunto de normativa de bomberos fueron realizadas se tomó la decisión de incluir en el set original una proposición más contemplando como hasta entonces su versión obligatoria, facultativa y prohibitiva. Las normas incluidas fueron n_{283}, n_{284} y n_{285} y la n_{283} fue seleccionada posteriormente por el método para ser aplicada por los agentes. La norma en cuestión obliga a los agentes a priorizar el incendio de los edificios habitados ante los edificios en llamas no ocupados y fue considerada suficientemente relevante y elemental como para incluirla. Las normas elegidas en el primer conjunto y perdidas después de la segunda selección serán mencionadas en los siguientes apartados.

5.1 Reglas de Agentes de bomberos

La normativa de los agentes de bomberos versa especialmente sobre la reducción en lo posible de pérdidas humanas. Las normas de bomberos seleccionadas en el conjunto original y que finalmente fueron excluidas en la segunda selección fueron: n_{32} y n_{40}.

- n_{37}: *If an unburnt building with people is next to one with fire, the fire brigade is OBLIGUED to water it.*
- n_{41}: *If the refuge is in fire, the fire brigade is PERMITTED to extinguish it.*
- n_{70}: *If the fire brigade has no water, the fire brigade is OBLIGUED to move to the refuge.*
- n_{79}: *If the fire brigade reaches a blockaded road in its path, it is OBLIGUED to change the path.*

- n_{82}: *If a police, ambulance or fire brigade discovers a new fire and the communication channel is overused, it is OBLIGUED to tell the fire fighters and ambulances and police about it.*

- n_{85}: *If a police, ambulance or fire brigade discovers a new fire and the communication channel is not overused, it is OBLIGUED to tell the fire fighters and ambulances about it.*

- n_{90}: *If a fire brigade extinguishes a fire and the communication channel is overused, it is FORBIDDEN to tell the fire fighters and ambulances about it.*

- n_{91}: *If a fire brigade extinguishes a fire and the communication channel is not overused, it is OBLIGUED to tell the fire fighters and ambulances about it.*

5.2 Reglas de Agentes de ambulancias

La normativa que aplica a los agentes sanitarios prioriza los rescates donde la situación resulta más crítica y algunas fomentan la velocidad de rescate. Únicamente la norma n_{115} fue seleccionada en el conjunto original y desechada en el final.

- n_{94}: *If the street with highest number of trapped people is unattended, the ambulance is OBLIGUED to attend it first.*

- n_{97}: *If the block with highest number of trapped people is unattended the ambulance is OBLIGUED to attend it first.*

- n_{110}: *If the most damaged building with people inside is unattended, the ambulance is PERMITTED to attend it first.*

- n_{116}: *If people are trapped in a heavily broken building (brokenness $>$ 0.5), the ambulance is PERMITTED to attend them.*

- n_{121} *If the ambulance has a patient, the ambulance is OBLIGUED to move to the refuge.*

- n_{142}: *If an ambulance agent is buried, another ambulance is OBLIGUED to attend him.*

- n_{151}: *If an ambulance agent is damaged, another ambulance is OBLIGUED to attend him.*

- n_{162}: *If a civilian is being attended, the ambulance is FORBIDDEN to attend him.*

- n_{169}: *If the ambulance reaches a blockaded road in its path, it is OBLIGUED to change the path.*

- n_{174}: *= If a police, ambulance or fire brigade discovers a new injured person and the communication channel is overused, it is FORBIDDEN to tell the ambulance about it.*

- n_{175}: *If a police, ambulance or fire brigade discovers a new injured person and the communication channel is not overused, it is OBLIGUED to tell the ambulance about it.*

- n_{180}: *If an ambulance attends an injured person and the communication channel is overused, it is FORBIDDEN to tell other ambulances about it.*

- n_{181}: *If an ambulance attends an injured person and the communication channel is not overused, it is OBLIGUED to tell other ambulances about it.*

5.3 Reglas de Agentes de policías

La normativa de los agentes de policía promueve la limpieza y desbloqueo de caminos con motivo de permitir y facilitar el acceso a edificios colapsados permitiendo un mejor desarrollo de las tareas de rescate por parte de los agentes sanitarios, y por otro lado el desbloqueo y por tanto una mejora de movilidad que permitiría a los bomberos reunirse y alcanzar los fuegos prontamente. Lo último provocaría un aumento de la efectividad de las acciones de extinción y control. La normas de policía que fueron excluidas en conjunto final y sí fueron seleccionadas en el primero fueron: n_{185} y n_{244}.

- n_{191}: *If the block with highest number of injured people is unreachable, the police is PERMITTED to clear a path to it.*

- n_{194}: *If the street with highest number of blockades is unattended, the police is PERMITTED to attend it first.*

- n_{197}: *If the block with highest number of blockades is unattended, the police is PERMITTED to attend it first.*

- n_{206}: *If the most damaged building with people inside is unreachable, the police is PERMITTED to clear a path to it.*

- n_{218}: *If a not heavily broken building (brokenness* 0.5*) is unreachable, the police car is PERMITTED to clean a path to it.*

- n_{221}: *If a heavily broken building (brokenness* > 0.5*) is unreachable, the police is PERMITTED to clear a path to it.*

- n_{230}: *If a blockade in a crossing is unattended the police is PERMITTED to clear it.*

- n_{233}: *If the blockade blocking most streets is unattended, the police is PERMITTED to clear it.*

- n_{242}: *If a blockade in the area of the police is unattended, the police is PERMITTED to clear it.*

- n_{245}: *If a police finds a blockaded road, it is PERMITTED to clear it.*

- n_{261}: *If a police, ambulance or fire brigade discovers a new blockade and the communication channel is overused, it is FORBIDDEN to tell all agents about it.*

- n_{262}: *If a police, ambulance or fire brigade discovers a new blockade and the communication channel is not overused, it is OBLIGUED to tell all agents about it.*

- n_{267}: *If a police clears a blockade and the communication channel is overused, it is FORBIDDEN to tell other agents about it.*

- n_{268}: *If a police clears a blockade and the communication channel is not overused, it is OBLIGUED to tell other agents about it.*

5.4 Reglas aplicables a todos Agentes

El siguiente tipo de normas no fueron incluidas en la realización de este trabajo. Sin embargo dada la normativa y el comportamiento básico que definía el proyecto base para bomberos y policías la regla n_{273} se cumple siempre.

- n_{273}: *If no norm applies to an agent, it is FORBIDDEN to return to the refuge.*

- n_{275}: If no norm applies to an agent, it is PERMITTED to patrol around the city randomly.

- n_{277}: *If no norm applies to an agent, it is OBLIGUED to patrol randomly near other agents.*

6 Implementación

A nivel de implementación es importante primero analizar que tipo de problema debía enfrentarse en la realización de este trabajo. El problema consistía en implementar un conjunto de reglas de tal modo que permitiese una sencilla escalabilidad y por lo tanto la inclusión de una nueva regla no implicase grandes cambios de código, tampoco si alguna ya implementada se decidía no incluir. Además debía permitir modelar sobre qué tipo de agente iba a aplicar y bajo qué situación o condición. La solución ideal tenía que ser compatible con la concurrencia permitiendo así que la aplicación de las normas puediese actuar simultaneamente. Es importante comentar respecto a esto el hecho de que cada comunicación entre componentes está a cargo de un thread por lo que como mínimo tantos threads como dos veces el número de componentes (un thread está del lado de un proceso externo al kernel, el otro thread está del lado del proceso kernel) conviven durante una simulación. Otro punto importante que debía cumplir la solución era tener el mínimo de acoplamiento de cara analizar posteriormente el impacto provocado por la aplicación o no de las reglas.

La solución elegida ha pasado por usar un motor de reglas, concretamente Drools en su séptima versión. Drools o JBoss Rules es un sistema de gestión de reglas de negocio que permite declarar reglas del tipo WHEN→THEN que además está desarrollado en Java y era compatible con Java 7 por lo que podía ser integrado en el proyecto. La normas seleccionadas se ajustan perfectamente a la forma en que se definen en el motor y cumplen con los requisitios a los que debía ceñirse una buena solución. Hay varias formas de escribir las normas de modo que después puedan ser interpretadas por Drools, es posible usar documentos excel, leerlas de un archivo de texto aunque se ha optado por la opción que entre la comunidad de usuarios resulta más popular y es escribirlas en lenguaje DRL de misma extensión. En este trabajo se generó un único documento de este tipo y en él han sido incluidas todas y cada una de las reglas implementadas. La organización sin embargo puede realizarse de múltiples formas y si es conveniente pueden usarse múltiples documentos DRL, por ejemplo generar tres de ellos e incluir en cada uno la normativa de cada tipo de agentes podría haber sido una opción. Otra opción por lo general siempre será hacer uso de una interfaz gráfica puesto que la mayoria de tools de este tipo contemplan entre sus productos una. Esto suele ser así debido a que los motores de reglas a menudo son usados para centralizar la lógica del negocio y desacoplar esta de la plataforma tratando de este modo relegar a los recursos humanos técnicos del conocimiento y control de las reglas de negocio ya que a menudo no son las mismas personas. Los conceptos que usa son equivalentes a los motores de reglas de negocio alternativos como puedan ser BizTalk Server de Microsoft para .Net o InRules, entre otros. Para integrarlo todas sus dependencias fueron incluidas en el proyecto RSLB2.

```
Logger.info("RSLB2 Started!");

//START DROOLS INITILIZATION, test.drl

KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();
Resource resource = ResourceFactory.newClassPathResource("RSLBench" + "/" + "test.drl");
kbuilder.add(resource, ResourceType.DRL);

if (kbuilder.hasErrors()) {
    System.out.println(kbuilder.getErrors().toString());
}else {
    System.out.println("KBuilder initialized SUCCESSFULLY");
}

InternalKnowledgeBase kbase = KnowledgeBaseFactory.newKnowledgeBase();
kbase.addPackages(kbuilder.getKnowledgePackages());

StatefulKnowledgeSession ksession = (StatefulKnowledgeSession) kbase.newKieSession();

// END DROOLS INITIALIZATION
```

Figure 22: Inicialización de Drools en el proyecto RSLB2

La figura 22 muestra cuál es el proceso inicial en el que el *conocimiento del negocio*, en este caso concreto la normativa seleccionada en este trabajo, contenido en el archivo *test.drl* es incluido en primer lugar a la entidad *KnowledgeBuilder* que

lee y procesa las normas para que sean comprendidas por el motor de reglas. Ese conocimiento se vuelve útil para el motor una vez procesado y es posible extraerlo de la entidad anteriormente mencionada haciendo uso de su método *getKnowledgePackages*, ello permite construir una instancia de *InternalKnowledgeBase* e incluirle el conocimiento, esta entidad permite generar la denominada **KieSession** que cuenta con la capacidad para interactuar con el motor. El objeto sesión cuenta con lo que se denomina *Working memory* o en español "Memoria de trabajo", todas las instancias candidatas a ser manipuladas por las reglas deben incluirse. La instancia **StatefulKnowledgeSession** es subclase de **KieSession** y hereda el método *FireAllRules*, una vez se llama a este método todas las instancias en el espacio de memoria son chequeadas contra las condiciones *When* y si alguna cumple con la condición definida en el bloque *Then* se ejecutará.

La implementación realizada es compatible con la concurrencia. Todos los agentes toman una referencia de la misma instancia **KieSession** y cada uno cuando alcanza el momento en que debe aplicársele las normas se introduce a sí mismo dentro del *WorkingMemory* y dispara todas las normas con el uso del método ya mencionado. Una vez hecho esto se elimina a sí mismo de la memoria, esto último es necesario porque la sesión almacena los estados de las instancias y si a una instancia ya le fueron aplicadas las normas no vuelven estas a aplicársele. La lógica expuesta ha sido encapsulada en un método denominado FireAllRules en la clase **PlatoonAbstractAgent**.

Respecto a las reglas comunicativas, es decir, todas aquellas que consideran la obligación, facultad u obligatoriedad de los agentes de comunicarse entre ellos ha sido integrada en la definición del problema que gestiona el Solver y define asignaciones para agentes de bomberos y policías. Para ello ha sido necesario suscribir a la entidad **CenterAgent** al canal de comunicación de los agentes actuando como oyente únicamente. Durante la simulación **CenterAgent** va almacenando los distintos datos que se comunican entre sí los agentes y con ellos construye la definción del problema que son capaces de interpretar los Solvers. Ello implica que las asignaciones por defecto realizadas por el algoritmo **BinaryMaxSum** en la configuración base y que se ha mantenido siempre activa en este trabajo para permitir a los agentes con reglas no activas seguir funcionando ahora basa la estrategia de asignación en base a conocimientos obtenidos y compartidos por los agentes.

7 Experimentación y Resultados

7.1 Settings de los experimentos

En esta sección se mostrarán los resultados obtenidos mediante la aplicación de las reglas implementadas y ya mencionadas en el apartado anterior. Los experimentos se han basado en una comparativa por bloques así como las reglas fueron definidas según el tipo de agente al que aplicaban distinguiendo de este modo normativa de

agentes de bomberos, de policía y equipos sanitarios. La comparación se ha realizado entre agentes del mismo tipo los unos con sus normas activas (además de todas las reglas comunicativas) y los otros únicamente con las normas comunicativa. Únicamente las reglas comunicativas en su conjunto (esto es, las normas de comunicación de bomberos, policías y equipo sanitario) han sido usadas para modelar lo que ha supuesto para este trabajo el entorno base sobre el que realizar comparaciones. Excluir las normas comunicativas después de la integración de éstas en la definición del problema gestionado por el Solver **BinaryMaxSum** usado por defecto implicaba la inoperabilidad de los agentes de policías así como los agentes de bomberos, que no contarían con una asignación de objetivos por defecto ni estrategia básica de acción. Por el contrario, mantener en exclusivo todas las normas comunicativas activadas simula el comportamiento que tendrían los agentes de policías y bomberos si requiriesen de una estrategia para organizarse y actuar y dicha estrategia fuese fruto del intercambio de mensajes entre agentes. Recuérdese en este punto que las normas comunicativas implementadas han sido las sguientes: n_{82}, n_{85}, n_{90}, n_{91}, n_{174}, n_{175}, n_{180}, n_{181}, n_{261}, ,n_{262}, n_{267} y n_{268}.

Dado que las simulaciones cuentan con factores aleatorios como puedan ser por ejemplo el seguimiento (o no) de las reglas de facultad por parte de los agentes, la propagación de los incendios modelada por **IgnitionSimulator** o la elección azarosa de un agente entre un abanico de posibles objetivos todos ellos igualmente prioritarios; han sido realizadas un total de veinte simulaciones en cada experimento, diez en los que las reglas del tipo comparado estaban activadas y otras diez donde no lo estaban para de este modo extraer la media de los resultados en cada caso y por tanto reducir moderadamente la variabilidad de los datos expuestos y analizados a continuación.

En cada comparación ha sido modelado un escenario que permitiese recrear un contexto en el que todas las normas pudiesen ser aplicables. El escenario usado en cada comparativa ha sido exactamente el mismo en todas y cada una de las veinte simulaciones por lo que tanto los agentes del tipo comparado con reglas activas y sin ellas han desarrollado su actividad en una simulación que ha contado con un mismo estado inicial y únicamente la variabilidad consustancial de la simulación y la aplicación o no de la normativa ha desencadenado estados posteriores distintos.

En los experimentos donde ha sido necesario simular daños estructurales **CollapseSimulator** ha sido incluido en la simulación, cuando se ha requerido de la especificación de bloqueos en el archivo escenario y se ha querido que estos fuesen completamente bloqueantes **BlockadeLoader** ha sido usado. Idealmente el uso de ambos hubiese permitido la generación de entornos más interesantes pero existen fallas de sincronización y congruencia de datos entre el simulador **BlockadeLoader**, proceso Kernel y el proceso RSLB2 cuando ambos simuladores son arrancados.

Los experimentos irań acompanñados de gráficos basados en los datos extraídos

de los propios experimentos. Respecto a los datos extraídos, todos aquellos que guardan alguna relación con la propiedad TIME han sido recogidos siempre partiendo de que el TIME 23 era el TIME inicial o 0. Por lo tanto a modo de ejemplo: los datos relativos a la velocidad de extinción de los agentes de bomberos han comenzado a cuantificarse no en el TIME 0 sino en el 23 así que aunque en la gráfica pueda partir del valor 0 en las coordenadas la simulación en ese entonces se hallaba en el TIME 23. La decisión de realizar la extracción de datos de esta forma se debe a que los agentes no son operativos hasta el TIME 23 por motivos de configuración: sólo una vez se alcanza dicho TIME comienzan a actuar los agentes.

7.2 Experimentación y Resultados: Agentes de bomberos

En el siguiente experimento las reglas especifícas activas han sido la siguientes: n_{37}, n_{41}, n_{79} y n_{283}. Y fielmente al comentario introductorio de esta sección las reglas comunicativas han sido incluidas en ambos casos. Los bloqueos generados han sido definidos a nivel escenario por lo que **BlockadeLoader** ha sido usado y excluido **CollapseSimulator**.

Todo y que la norma n_{70} está contenida en el conjunto seleccionado su activación en la simulación produjo un desarrollo destacablemente negativo en los agentes de bomberos debido a la gran pérdida de tiempo que demoraban en ir y volver del refugio con sus tanques de agua de nuevo llenos y de vuelta preparados para la acción. Tomando en consideración que todas las simulaciones de este trabajo se han producido sobre el mapa de París se optó por excluirla del set de pruebas debido a que una ciudad cosmopolita, densamente poblada y rica en infraestructuras lo normal es que cuente con un gran número de recursos hidratantes. Este tipo de recursos deberían permitir al agente de bomberos el acceso a un cabal constante y suficientemente elevado como para mantener el poder de extinción configurado en el archivo *resq-fire* (500l por acción de extinción) que por otro lado es estandard si se tiene en cuenta que una manguera de incendios consume entre 400l-600l/minuto. En este sentido los agentes de bomberos en esta y en el resto de experimentos han sido configurados para obtener una cantidad inicial de agua a efectos prácticos inagotable.

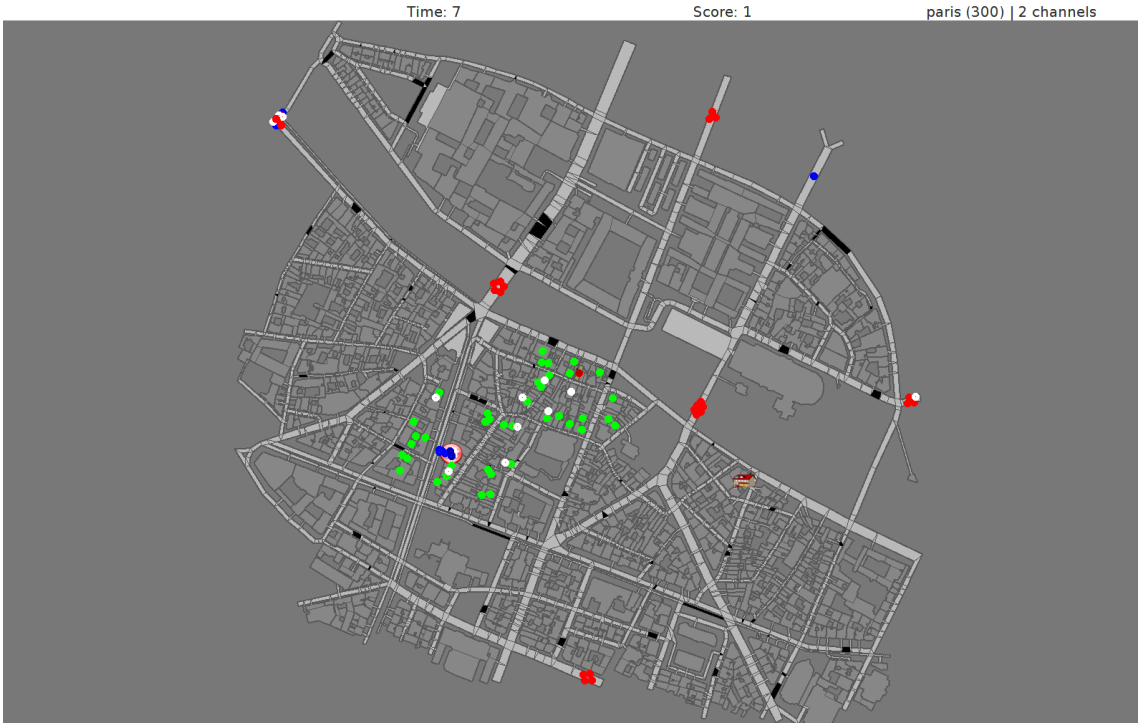


Figure 23: Estado inicial del escenario utilizado en el experimento de agentes de bomberos

La figura 23 muestra el estado inicial del escenario usado en este experimento. Cuenta con dos focos de incendio, ambos en una posición central del mapa cerca de donde pueden observarse los civiles. Uno de ellos concretamente situado sobre la posición asociada a la estructura Refugio, el otro se localiza exáctamente en el edificio en el que ha sido situado un agente de bomberos, puede verse de color rojo oscuro un punto situado exactamete en el centro del mapa y rodeado por civiles. Los incendios que no resultan de la propagación sino que son establecidos en el escenario dañan gravemente el edificio en el que se originan. Esto se debe a que los agentes comienzan a actuar a partir del TIME 23 por lo que tal y como se observa en la siguiente figura, el estado del edificio foco resulta ya dañado por el fuego moderadamente. El incendio en el Refugio por lo tanto supone un reto difícil de lograr para la regla n_{41} que ha sido dispuesta con un seguimiento de 0.5 de probabilidad.

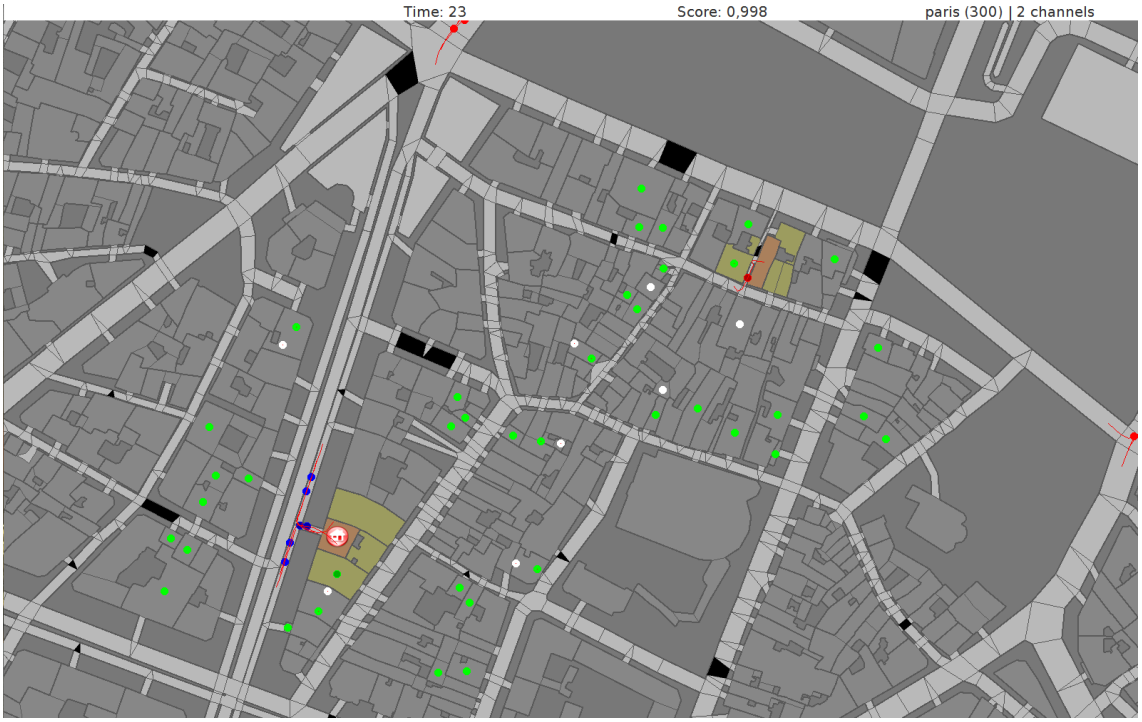


Figure 24: TIME 23, Escenario usado en la experimentación con agentes de bomberos

La figura 24 muestra el instante en el que los agentes comienzan a actuar, esto es, en el TIME número 23. Se ha realizado un zoom para de este modo mostrar cómo el estado de los incendios ya ha evolucionado lo suficiente como para dañar moderadamente los edificios foco y además propagar las llamas a los edificios vecinos.

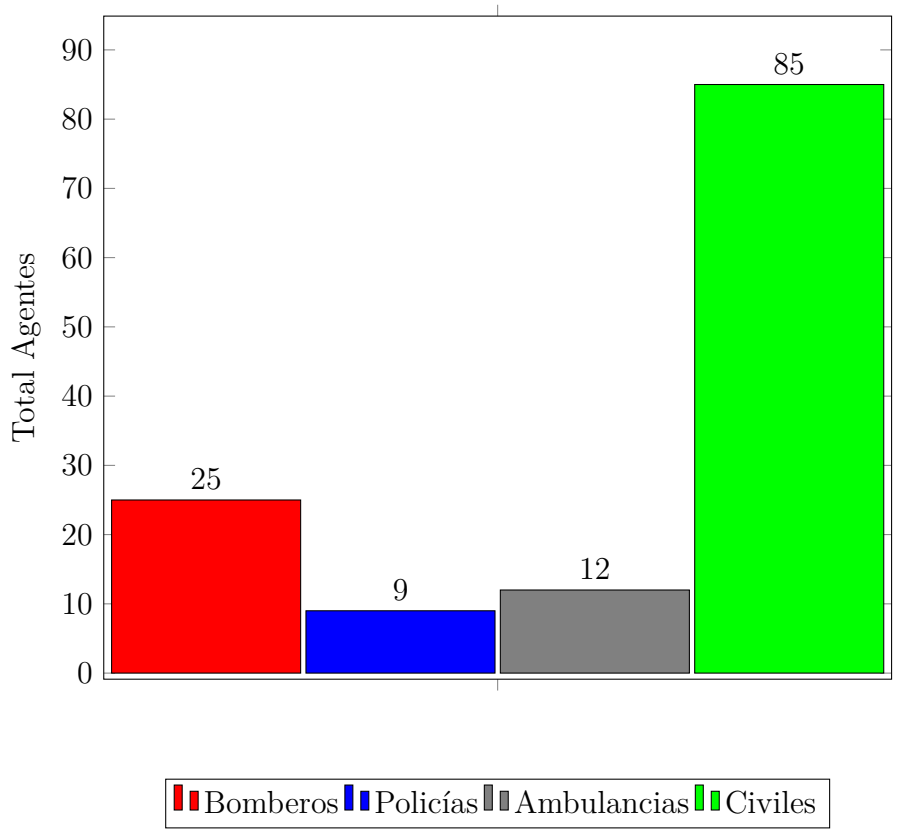


Figure 25: Total Agentes según su tipología, Experimento Agentes de Bomberos

El gráfico de barras de la figura 25 muestra las cantidades de los distintos tipos de agentes con los que se han contado para realizar este experimento. El número de agentes de bomberos ha sido seleccionado en función a pruebas anteriormente realizadas tratando de generar un escenario que convergiese a una solución, esto es, que los agentes de bomberos no quedasen sobrepasados por el poder de los incendios y les fuese imposible controlarlo. Tampoco un número elevado de ellos que no permitiese comprobar una evolución suficientemente prolongada de las acciones y efectos de los agentes en la simulación.

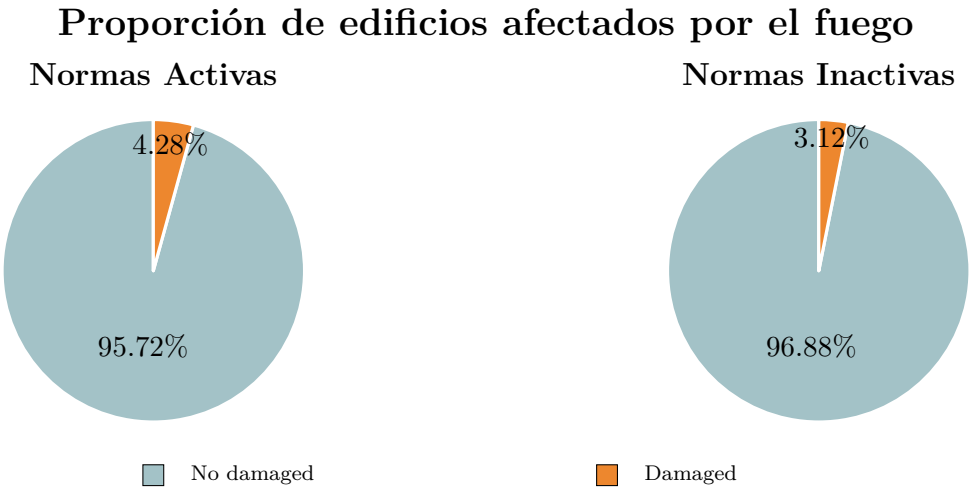


Figure 26: Experimento Agentes de Bomberos: Proporción de edifios dañados por el efecto del fuego sobre el total.

En la figura 26 se muestran dos gráficos circulares, ambos permiten visualizar cuál ha sido la proporción de edificios danñados por efecto del fuego sobre el total de edificios en cada caso. Así pues, el resultado en este aspecto ha sido negativo cuando las normas han estado activas puesto que un 4.28% frente a un 3.12% de edificios quemados equivale aproximadamente a 19 edificios quemados de más (teniendo en cuenta que en total hay 1618 edificios en el mapa de París) en la simulación con reglas activas. Resulta importante considerar la igualdad existente entre número de incendios y números de edificios dañados por las llamas debido a que todo incendio ocurre en un edificio por lo que existe una igualdad ambos.

Entendiendo que el número promedio de TIMES de una simulación es la cantidad de iteraciones del Core de kernel necesarias para finalizar y ello ocurre una vez todos los incendios son extinguidos. En este sentido el número de TIMES promedio de la simulación con reglas activas ha sido levemente inferior en promedio obteniendo un valor de 110.6 mientras que en la simulación sin las reglas activadas los bomberos han necesitado 127.8 TIMES. Esto implica han sido más veloces en el desarrollo de las acciones de extinción los agentes de bomberos con reglas pero que en el control de incendios han sido mejores los agentes sin ellas porque de este modo han evitado antes la propagación.

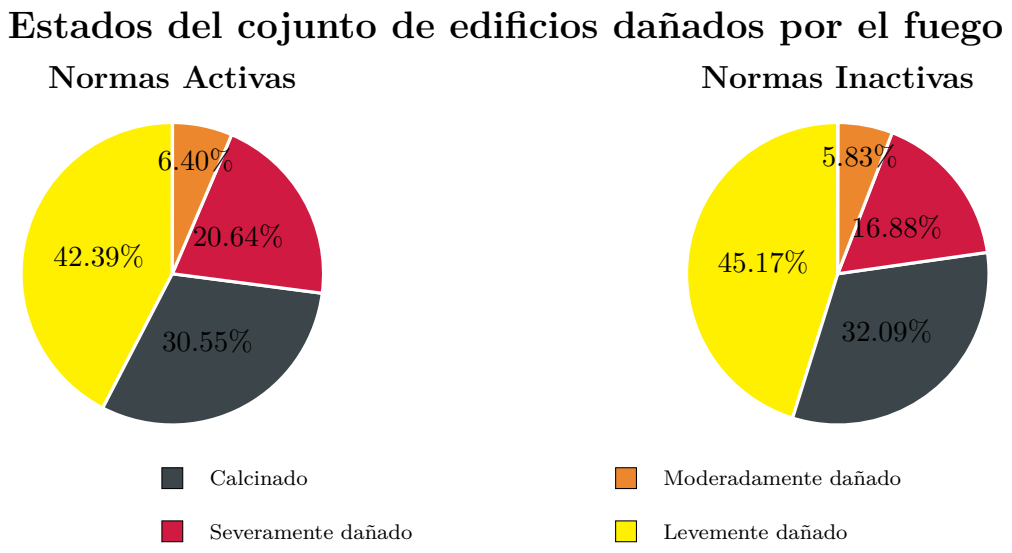


Figure 27: Proporción de edifios dañados por el efecto del fuego, Experimento Agentes de Bomberos

En la figura 27 se muestran de nuevo dos gráficos circulares que representan visualmente en qué proporción han sido dañados los edificios afectados por las llamas, esto es, tal y como se acaba de mostrar en el la figura 26 4.28% o 69.3 edificios en el caso de la simulación con reglas activas y 3.12% o 50.6 edificios con reglas inactivas. Se observa una alta semenjanza entre ambos resultados, quizá es detacable el hecho

de que enfrentándose a un mayor número de incendios los agentes con reglas han logrado evitar en mayor medida la calcinación de los edificios manteniendo un mayor número de ellos en la línea roja que implica estar severamente dañado. Donde se observa un incremento de cerca de 4 puntos respecto al 16.88% del porporcionado por el gráfico de los agentes sin reglas activas.

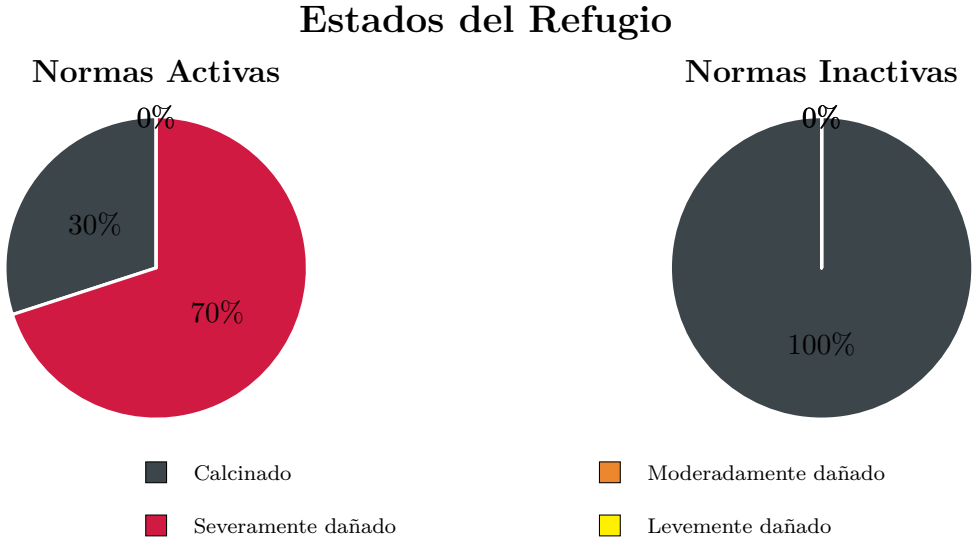


Figure 28: Probabilidades de estados finales de la estructura Refugio, expresadas porcentualmente

Asímismo, se han extraido datos específicos relativos al estado del Refugio en cada simulación para construir los gráficos circulares mostrados en esta figura 28. Merece una especial atención por la relevancia que tiene la estructura Refugio en este trabajo debido a que es el centro de atención donde son llevados los civiles rescatados y en teoría con la aplicación de la regla n_{70} la única fuente de recursos hidratantes de los agentes de bomberos. Aunque la plataforma de simulación Roborescue simula que los edificios calcinados ya no queman más y permite el acceso de los agentes al interior de estos sin problemas, la realidad es que en este trabajo se ha considerado como "perdido" o "destruido" todo aquel edificio que resulte calcinado. Perder por lo tanto un edificio como el refugio implica potencialmente la pérdida de múltples vidas humanas. De acuerdo con esto los resultados obtenidos muestran una probabilidad de éxito del 0.7 a la hora de salvar el refugio cuando las reglas han estado activadas y una probabilidad nula cuando no las ha habido. A pesar de que se ha salvado un 70% de ocasiones el refugio siempre ha quedado severamente dañado aunque como se mostró con anterioridad el estado en el que ya estaba el refugio al comenzar a actuar los agentes era ya moderadamente dañado. La regla n_{41} en este caso ha tenido un papel realmente relevante.

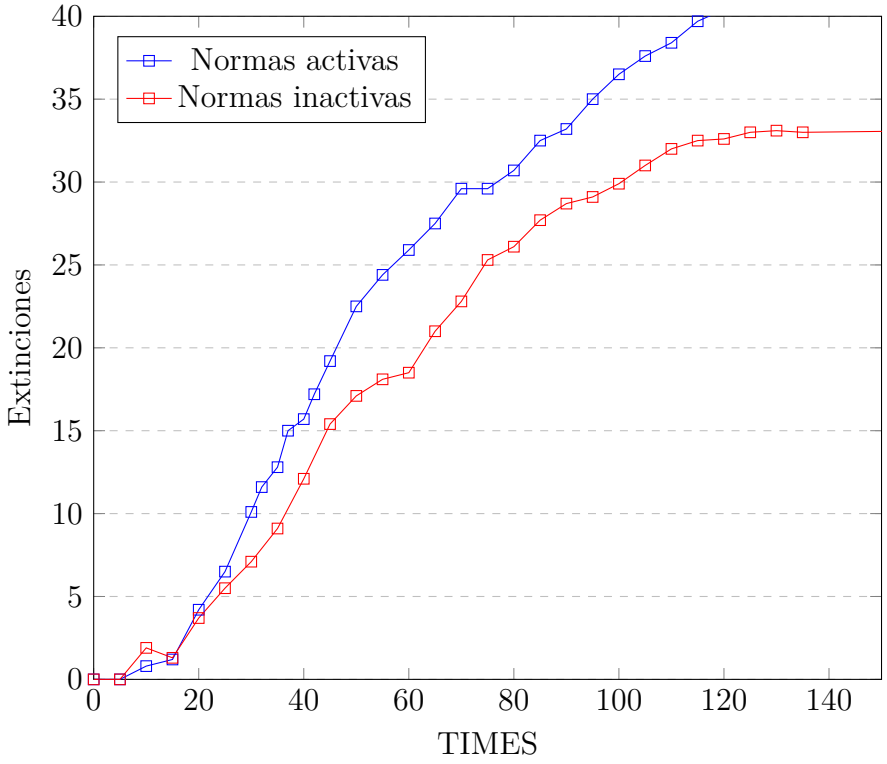


Figure 29: Evolución de la cantidad de edificios extintos a lo largo de los TIMES

La figura 29 muestra a modo de función discreta un par de ellas, la de color azul tal y como señala la leyenda, ha resultado de usar los datos extraidos en las simulaciones con reglas y en rojo el resultado de la experimentación sin reglas. Las funciones representan el número de fuegos extintos totales que había en cada TIME en la simulación de modo que en el último TIME hallariamos el total de edificios extintos promedio que resultaría de sustraer al total de edificios dañados por las llamas los edificios calcinados puesto que toda simulación finaliza con la extinción de todos los incendios y no se consideran los edificios calcinados como fuegos activos. Se sabe llegados a este gráfico y vistos anteriormente los de la figura 26 que los agentes de bomberos con reglas han tenido que enfrentarse a un mayor número de incendios y por lo tanto es lógico la función azul acabe superando a la de color rojo hacia el final, una vez los agentes con reglas inactivas han controlado los incendios y todavía queda para los otros como una tarea pendiente. Pero todo y que la divergencia que muestran las funciones cuando toman los valores hacia el final de las coordenadas en las abscisas era de esperar el hecho de que la función azul se mantenga con valores algo más elevados desde prácticamente el TIME 22, 23 implica un mayor número de extinciones acumuladas en cada momento de la simulación desde entonces por partede los agentes con reglas activas. El pico producido por la línea roja durante los primeros instantes entre el TIME 0 y el TIME 17 aproximadamente donde esta invade la azul superándola significativamente muestra esa primera reacción más rápida frente a los incendios por parte de los bomberos con reglas inactivas.

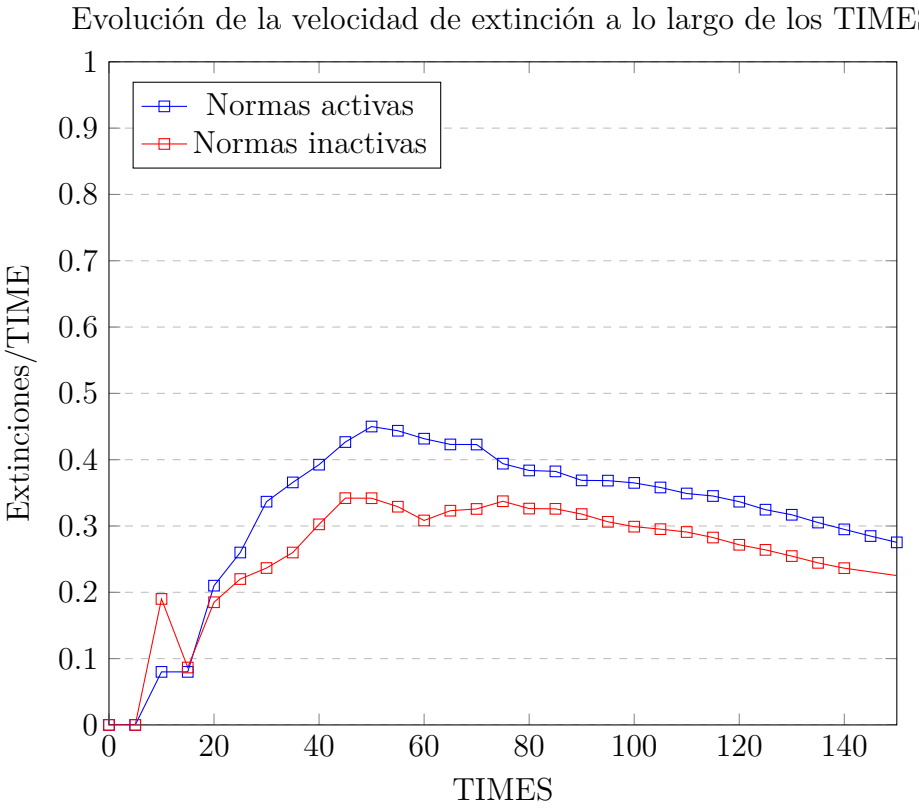


Figure 30: Evolución de la cantidad de la velocidad de extinción a lo largo de los TIMES

La figura 30 muestra mediante funciones discretas la evolución de la velocidad de extinción de los agentes de bomberos. La función azul ha sido construida mediante los resultdos obtenidos con reglas activas y la línea de color rojo la función obtenida sin ellas. Aquí se muestra más claramenre el pico de velocidad que demuestran los agentes sin reglas en los primeros TIMES pero luego los agentes con reglas les superan generando una mayor diferencia respecto a los otros, especialmente en el intervalo contenido entre el TIME 20 y TIME 65. Un comportamiento que se ha mostrado empíricamente es el hecho de una reducción en la velocidad de extinción cuando los agentes tienen ya controlado el incendio. Esto se debe principalmente a que los incendios más graves e intensos se hallan dentro de la zona todavía activa mientras que ésta queda circundada a menudo por edificios dañados levemente y ya extintos. Dicho lo anterior, la reducción paulatina pero incesante de ambas veocidades coincidiría con esta etapa de control donde no están extintos todos los fuegos pero los que quedan están ya restringidos y son costosos de apagar dada su intensidad.

Empíricamente se ha observado que la atención especial sobre el refugio procura unos esfuerzos extra al principio, al menos hasta que está salvado o en el peor de los casos ya se ha perdido. Esto explicaría la falta de velocidad reactiva inicial por parte de los agentes de bomberos con reglas. La superioridad en la velocidad

posterior podría ser el resultado de la acción de la regla n_{79}, ya que el que hecho de poder tomar rutas alternativas se ha demostrado en pruebas anteriores, que no serán presentadas en este trabajo, que tiene la capacidad de reducir la cantidad de TIMES significativamente así como de permitir a los agentes alcanzar sus objetivos en un menor tiempo.

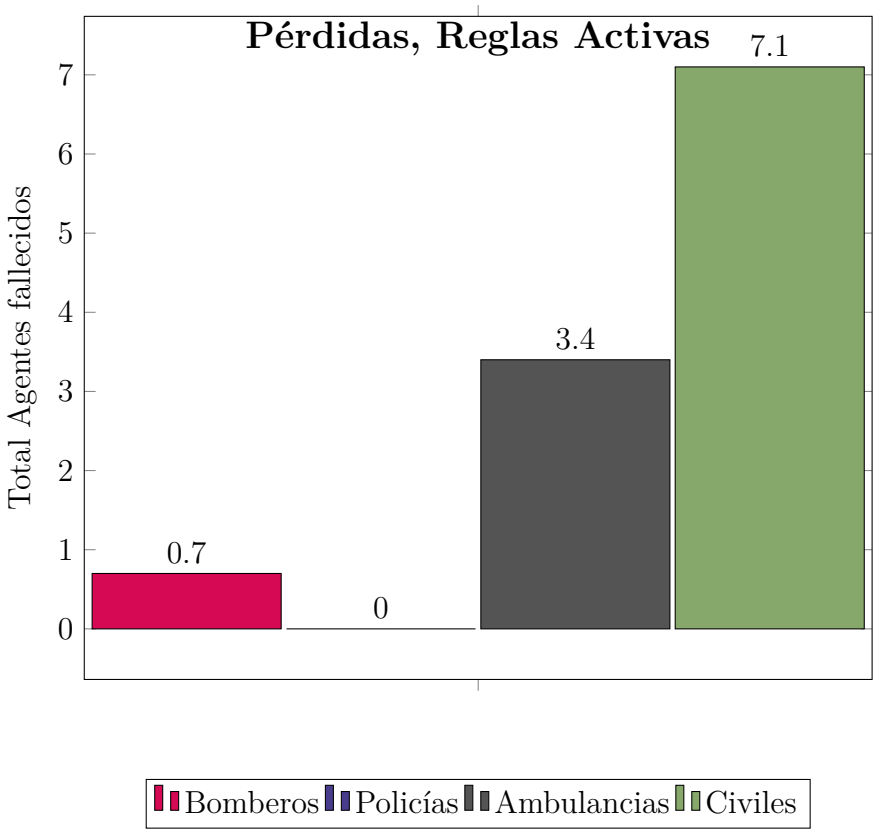


Figure 31: Agentes fallecidos por tipología con reglas activadas

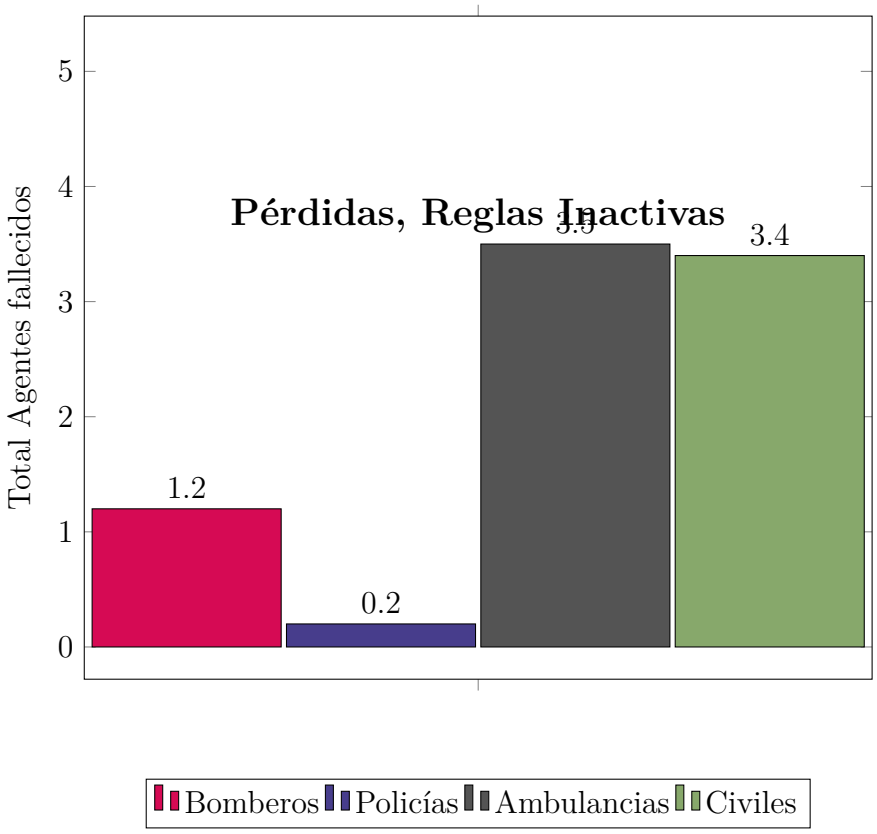


Figure 32: Agentes fallecidos por tipología con reglas inactivadas

Las figuras 31 y 32 muestran la cantidad de agentes fallecidos diferenciados por su tipología en las simulaciones con reglas activas e inactivas respectivamente. Los resultados entre ambas no son demasiado esclarecedores aunque pareciese que con las reglas activas un número mayor de civiles mueren. La distancia que separa la cuantía de civiles fallecidos en la primera gráfica respecto la segunda es 3.7 puntos lo cual podría estar relacionado con el mayor número de edificios afectados por las llamas en las simulaciones con reglas activas. Igualmente pareciese necesario realizar un mayor número de simulaciones para excluir la posibilidad de que se deba a la variablidad propia de la simulación. También se asemejan el total de civiles dañados y no fallecidos, que en el caso de la simulación con normas ha obtenido un valor de 3.5 y 4 en caso de normas inactivas. Las reglas n_{37} y n_{283} no parecen haber desarrollado un gran papel, de media un único edificio ha sido mojado y dado que a esta acción solo la fuerza la regla n_{37}, como era de esperar, los agentes sin regla no han mojado ningún edificio. Aunque no haya sido graficado ha sido extraido el número de fallecidos que se encontraban en edificios extintos en el instante en que estos han perdido la totalidad de sus puntos de vida. Este valor guarda relación con el comportamiento modelado por la regla n_{283} que prioriza la extinción de incendios en edificios ocupados pero que a menudos el progreso del daño de las quemaduras termina con el agente que ha tratado de protegerse. Aunque los resultados han mostrado solo 1.7 fallecidos en edificios extintos en la simulación con reglas frente 1.1. La poca aplicación de las reglas n_{37} y n_{283} es fruto o bien del escenario elegido que genera pocos contextos de aplicación, la interpretación e implementación o una

mezcla de ambas.

7.3 Experimentación y Resultados: Agentes sanitarios

En el siguiente experimento se han puesto a prueba la normativa que aplica a los agentes sanitarios. Por ello las reglas activadas han sido las propias de este colectivo que son: n_{94}, n_{97}, n_{110}, n_{116}, n_{121}, n_{142}, n_{162} y n_{169}. La regla n_{151} todo y ser incluida en el conjunto seleccionado no ha sido activada debido a que la estructura **Refuge** no produce una recuperación de los puntos de vida y los agentes sanitarios tampoco producen un efecto en este sentido. Por ello la regla resulta insatisfactoria en su aplicacón puesto que un agente sanitario no puede mejorar la salud de otro agente sanitario dañado y en esto mismo reside el sentido de la misma.
El escenario utilizado ha sido el siguiente:

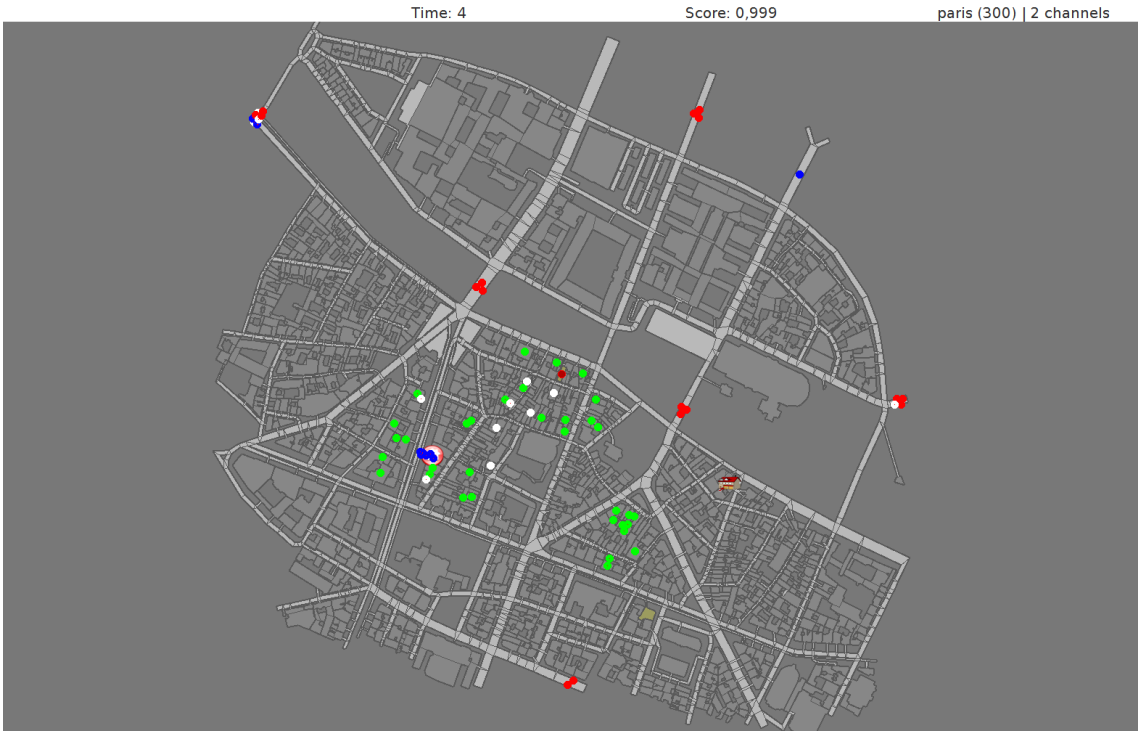


Figure 33: Estado inicial del escenario de pruebas, Agentes sanitarios

La figura 33 muestra la visualización del escenario usado para esta serie de pruebas. La imagen ha sido tomada en el TIME 4 por lo que los agentes todavía no han tomado acción. Han sido introducidos dos focos de incendios, uno de ellos se halla en el centro del mapa. Se han dispuesto también las entidades civiles en el centro donde tienen mayor posibilidad de ser afectadas por el incendio, especialmente si los agentes de bomberos no consiguen controlarlo a tiempo. Dado que la normativa de los agentes sanitarios propone múltiples reglas que consideran a humanos atrapados ha sido necesario provocar daños en estructuras para poder aplicar a los afectados el estado de "enterramiento" (**CollapseInjury** o **BuriedInjury**, recuédese en este punto el atributo *buriedness*). Para provocar los colapsos en

los edificios ha sido incluido en la especificación un terremoto en el TIME 15 de intensidad 0.2 haciendo uso en este caso de **CollapseSimulator**.

Después del terremoto, que conceptualmente Robocuprescue encapsula mediante un comando denominado **Aftershock**, el escenario ha quedado del siguiente modo:

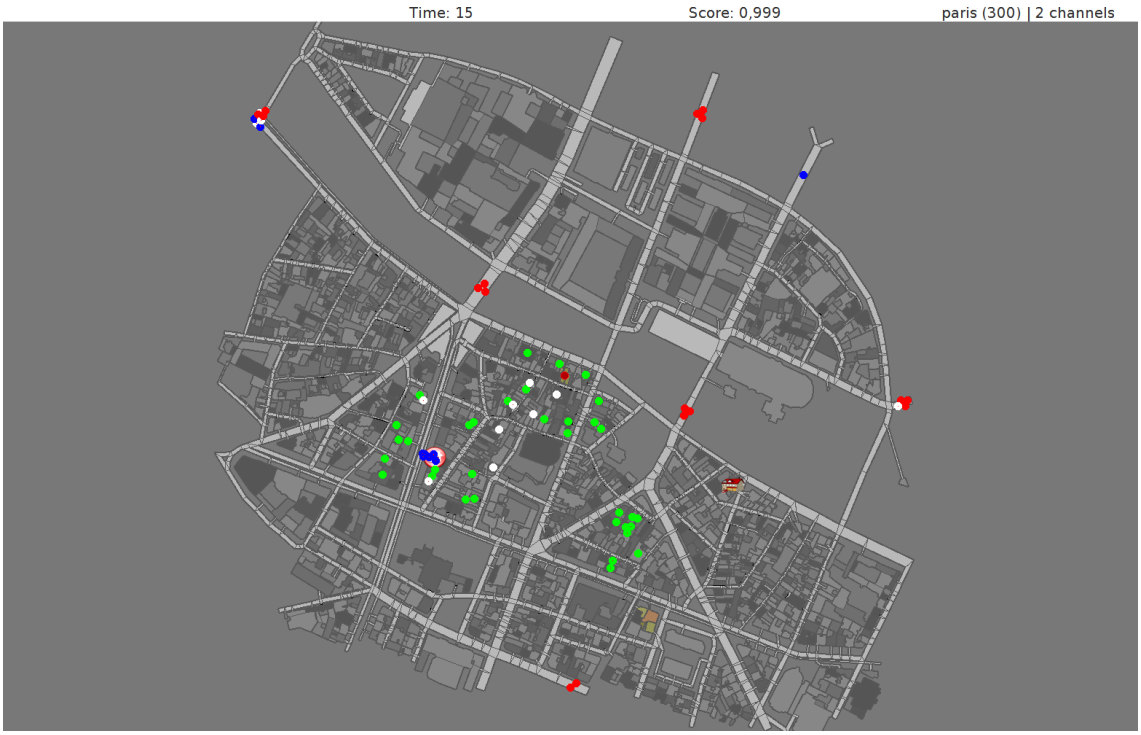


Figure 34: Estado post colapso del escenario de prueba, Agentes sanitarios

En la figura 34 se muestran visiblemente afectados múltiples edificios por todo el mapa. Más adelante se enumerarán los afectados en promdio por el terremoto y cuántos seres humanos han quedado inmovilizados. Varios agentes sanitarios han sido colocados por defecto en el interior de edicios con la finalidad de que algunos queden atrapados y la regla n_{142} encuentre así algún contexto de aplicación.

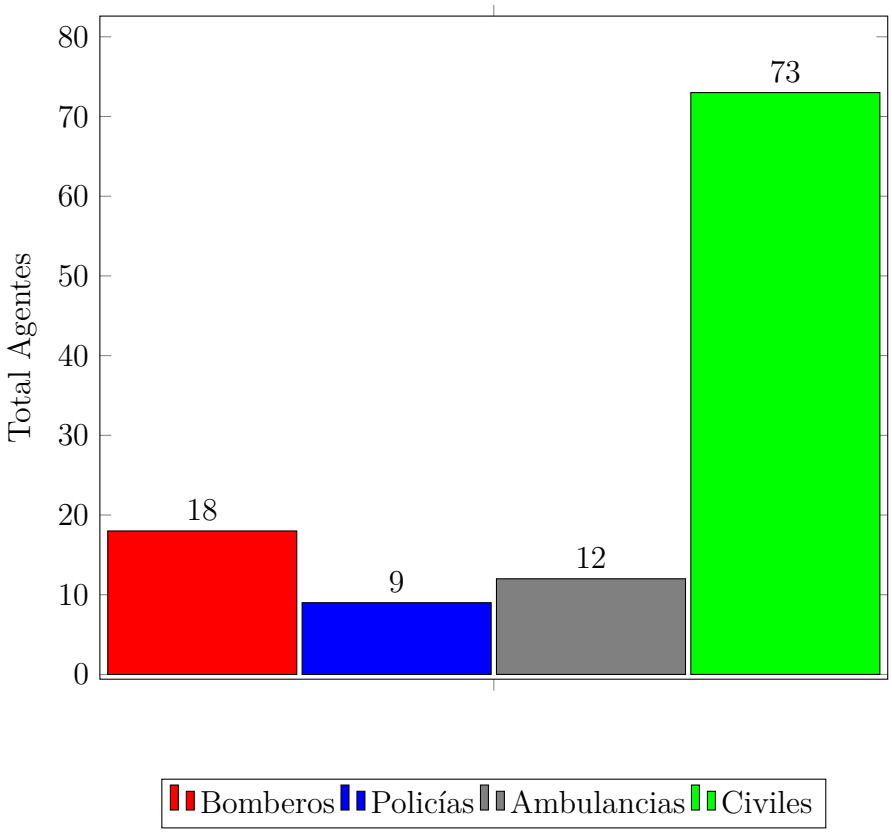


Figure 35: Total Agentes según su tipología, Agentes Sanitarios

En la figura 35 se muestran a modo de gráfico de barras la cantidad de agentes según su tipología con los que cuenta el escenario usado en este experimento.

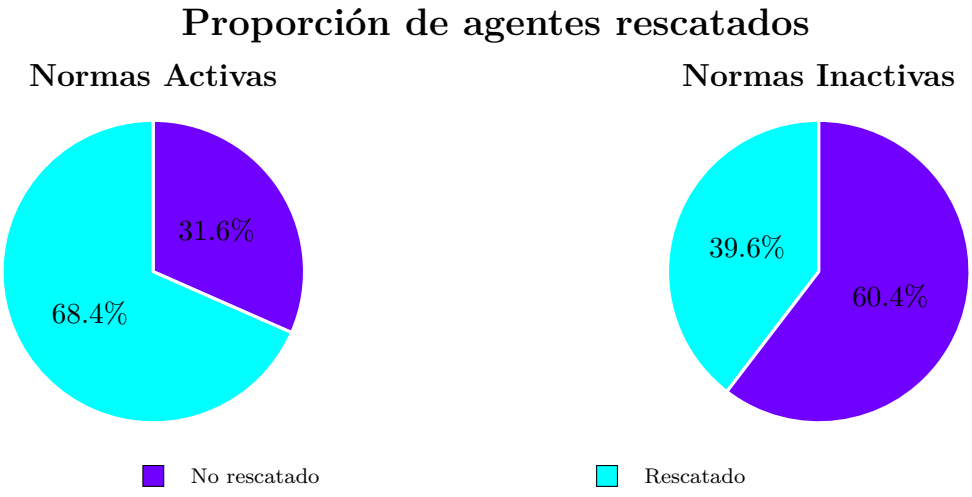


Figure 36: Gráfico circular que representa la proporción de rescates en función del nivel de rotura de los edificios desde los que fueron rescatados, Experimento Sanitarios

La figura 36 muestra dos gráficos circulares que representan la proporción de gente rescatada sobre el total de agentes enterrados víctimas del seísmo programado

en el escenario. Es importante mencionar que el simulador **CollapseSimulator** provoca siempre los mismo daños y en los mismos edificios por lo que es posible contar con la seguridad de que su inclusión en la simulación es controlable y su ejecución siempre mostrará los mismos resultados. Por ello el número de agentes enterrados a lo largo de todas las simulaciones ha sido el mismo tanto en aquellas donde las reglas estaban activas como en las que no, y la cantidad de personas enterradas ha sido siempre 25. Ahora, por lo que refiere a los gráficos circulares mostrados en esta figura 36 la diferencia entre ellos es evidente y muy notable. Los agentes de ambulancias con reglas activas han rescatado un 28.8% más de agentes, en total 17.1 agentes rescatados en promedio frente al 9.9 de los agentes con reglas inactivas. Cabe también destacar la diferencia obtenida en términos de la cantidad de TIMES promedio que un tipo de simulación y otra alcanzaron, obteniendo 93.2 TIMES en la simulación con reglas activas y 142.1 TIMES con reglas inactivas. Dado que toda simulación finaliza cuando los incendios en su totalidad han sido extintos y el comportamiento los agentes sanitarios no interfiere aparentemente en el desarrollo de los agentes de bomberos, pareciera ser fruto de la variabilidad de la propia plataforma de simulación. Lo anteriormente comentado implica además que los agentes con reglas han actuado con mayor velocidad de lo que lo han hecho los otros agentes.

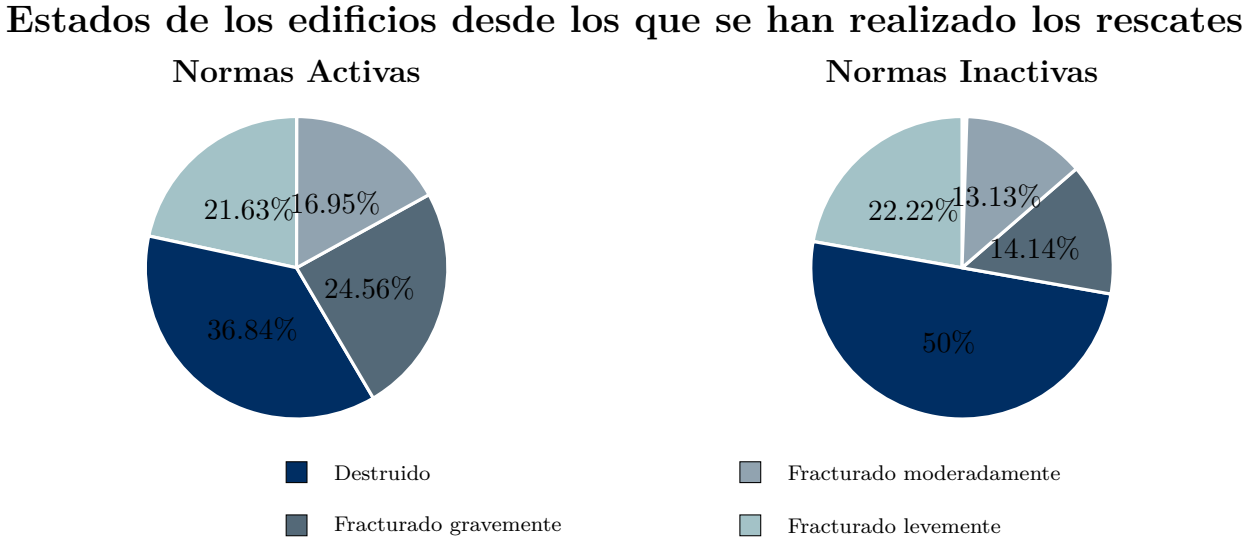


Figure 37: Gráfico circular que representa la proporción de rescates en función del nivel de rotura de los edificios desde los que fueron rescatados, Experimento Sanitarios

La figura 37 contiene dos gráficos circulares que muestran las proporciones de los distintos estados de rotura en los que se hallaban los edificios desde los que fueron rescatados los agentes. Vemos por un lado que en el gráfico circular obtenido de la simulación con normas activas existe una preponderancia de los edificios con mayor rotura ya sean aquellos que están destruidos ($brokenness \geq 75\%$) o gravemente fracturados ($50\% \leq brokenness < 75\%$). En el gráfico resultado de la simulación con normas inactivas hay una mayor proporción de agentes rescatados de edificios

destruidos pero por lo que respecta a los edificios moderada y gravemente fracturados la proporción se reduce respecto al primero. Es importante recordar llegados a este punto que los agentes santarios por defecto usan una estrategia *Closest* para seleccionar sus objetivos por lo que la disposición de los agentes en el mapa tendrá una repercusión total. Da por lo tanto la casualidad que las disposición inicial de los agentes sanitarios se halla cerca de múltiples edificios destruidos accesibles y con agentes en su interior que rescatar. Respecto al primer gráfico habría que considerar las reglas que puedan estar proponiendo priorizar este tipo de objetivos. Las reglas que priorizan objetivos que se hallen en edificios altamente danãdos son: n_{110} y n_{116}.

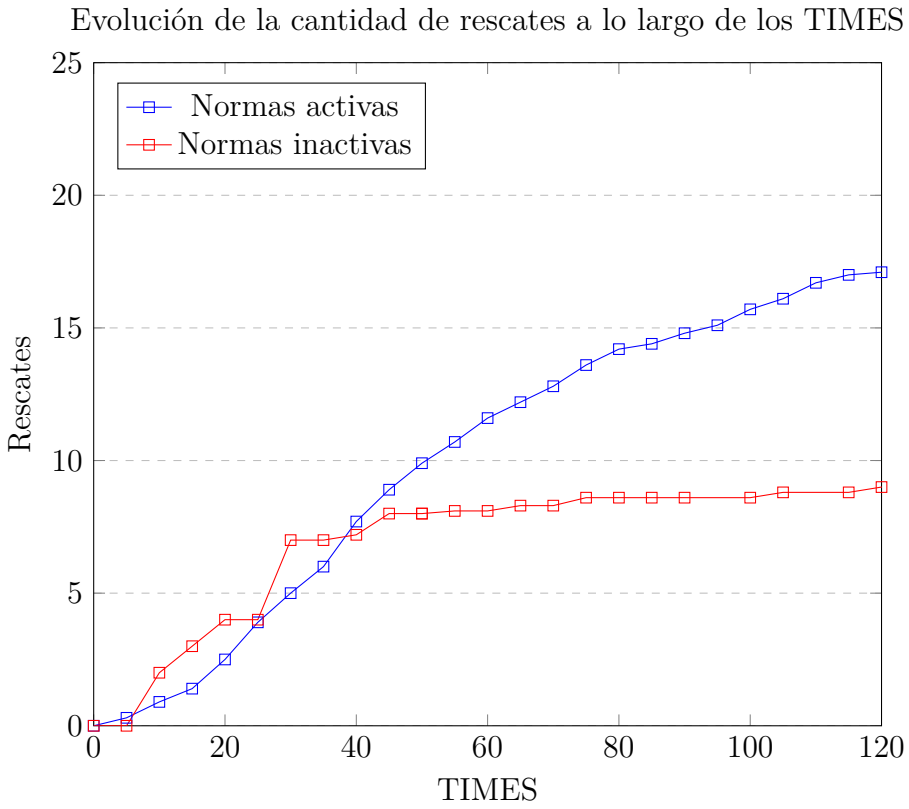


Figure 38: Evolución de la cantidad de rescates a lo largo de los TIMES, Experimento Sanitarios

La figura 38 muestra mediante el uso de funciones discretas la evolución de la cantidad total de personas rescatadas TIME a TIME en la simulación, obteniendo de este modo el último punto el total promedio de personas rescatadas, 17.1 en caso de la simulación con reglas y 9,9 en el otro. El comportamiento de la función roja se estanca especialmente a partir del TIME 43 aproximadamente, aunque ya parece comennzar esta en el TIME 34. Dicha etapa se define por su escaso crecimiento aunque viene precedida de una etapa inicial con importantes crecidas includo superando en esos instantes la función azul. En cambio, la azul, pese a ser inicialmente superada por la función roja, crece progresivamente con una pendiente lijeramente cambiante pero que hace pensar en la posibilidad de que el número de rescates obtenidos por los agentes con reglas haya sido cortado por la brevedad

de sus simulaciones, ahí se ve pues que posterior al TIME promedio 93.2 que han durado las simulaciones con reglas activas la función azul sigue creciendo, por lo que aquellas que se han demorado más del promedio han continuado realizando rescates.

Los agentes sanitarios sin reglas acceden antes a los agentes afectados por el colapso de los edificios puesto que recorren menos distancia eligiendo siempre dirigirse al objetivo más cercano. Pero hay que recordar que **CollapseSimulator** genera mútiples bloqueos y especialmente bloqueos que tornan inaccesible la entrada en los edificios. Esto último provoca que una disposición aparentemente buena de los agentes de ambulancias termine afectando negativamente porque una vez que realizan los primeros rescates convergen muchos de ellos a un mismo objetivo, que a menudo se encuentra inaccesible, lo cual les mantiene inmóviles a la espera de una agente de policía que venga a socorrerles.

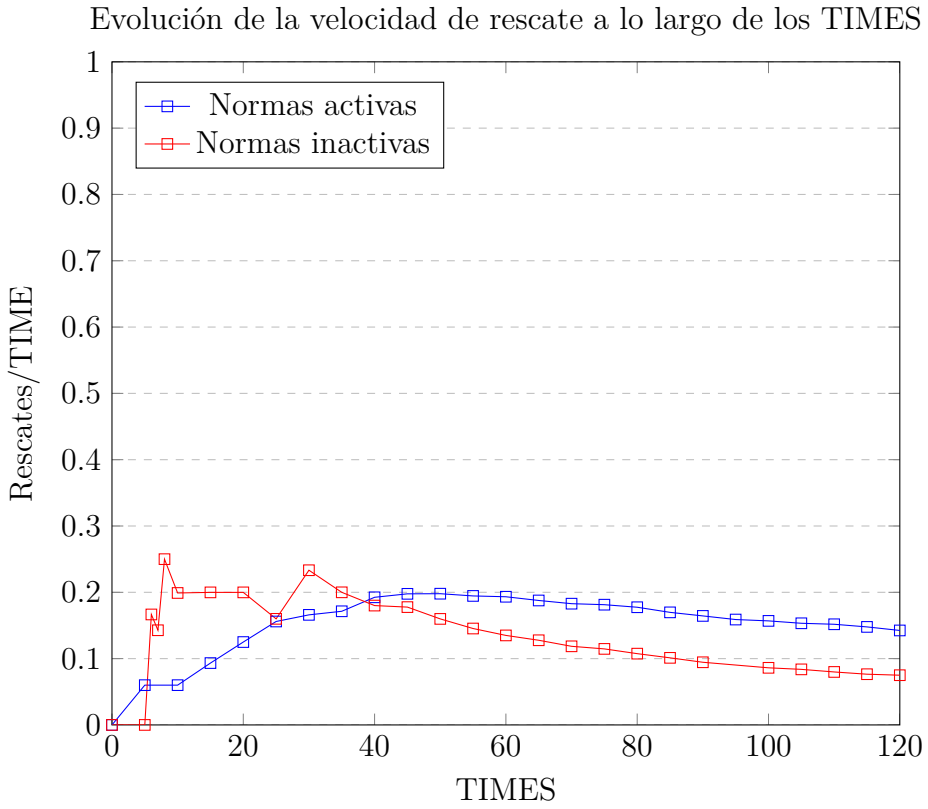


Figure 39: Evolución de la velocidad de rescate a lo largo de los TIMES, Experimento Sanitarios

En la figura 39 se muestra mediante dos funciones discretas la evolución de la velocidad de rescate a lo largo de los TIMES. Tal y como destacaba la acción de los agentes sanitarios sin reglas los instantes primeros aquí se observa como sobrepasa notablemente la velocidad de los agentes sanitarios con reglas, desde el TIME 5 hasta el 32 aproximadamente obteniendo su máximo valor cerca del TIME 8 con alrededor de 0.25 rescates/TIME. Posteriormente la velocidad de los agentes se reduce cuantiosamente y la función azul intersecta en el TIME 40 y la supera en

adelante.
Los agentes sanitarios con reglas han resultado en este experimento ser más veloces y la regla n_{142} tiene un papel muy importante que ha podido ser comprobado en experimentos anteriores a éste. Múltiples agentes sanitarios eran inmovilizados por el seísmo y terminaban siendo rescatados en fases tempranas de la simulación debido a esta regla. También los agentes con reglas activas quedan atorados debido a que su objetivo se halla en un edificio inaccesible con un bloqueo en su entrada y en este aspecto tanto los agentes con reglas activas como sin ellas no han sido capacitados para tomar otro objetivo que sí sea accesible. La regla n_{169} no resulta satisfactoria cuando únicamente hay una ruta posible y esa está bloqueada, lo que no sucede normalmente es que múltiples agentes convengan en rescatar al mismo objetivo siendo este inaccesible algo que sí resulta mucho más común en los agentes sin reglas.

7.4 Experimentación y Resultados: Agentes de policía

El experimento considerará el escenario anterior, usado en el experimento de agentes sanitarios. En este experimento excepcionalmente se han mantenido activas tanto las normas de los agentes sanitarios como la de los policías, esto se debe a la propiedad sinérgica de la acción de desbloqueo característica de los policías. Por ello la comparación se realizará entre las simulaciones realizadas con normas tanto de agentes sanitarios como policías activadas frente a simulaciones donde únicamente las reglas relativas a los agentes sanitarios estaban activas. Todas las normas de facultad han tenido seguimiento con probabilidad 0.5.

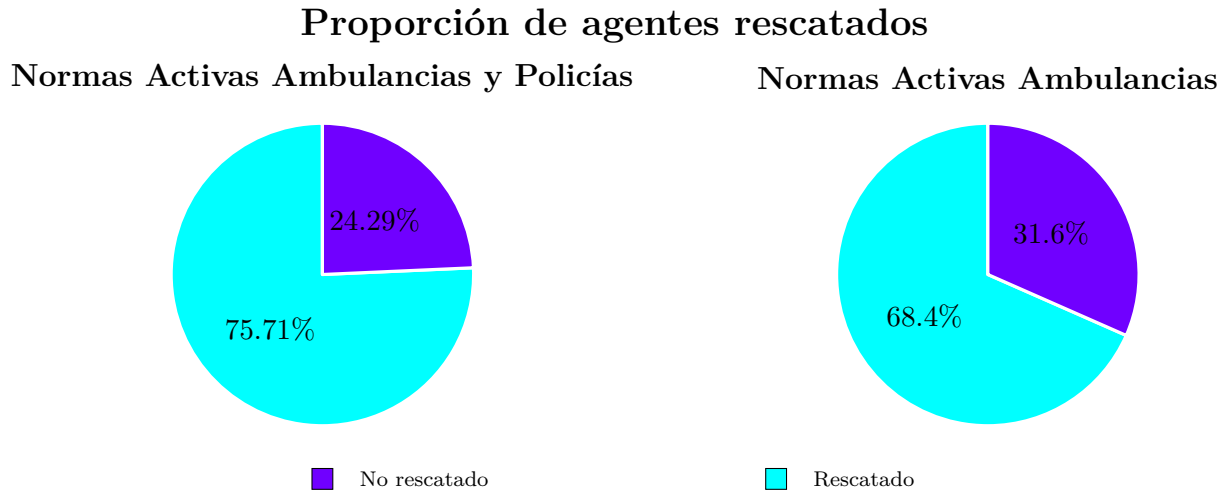


Figure 40: Gráfico circular que representa la proporción de rescates en función del nivel de rotura de los edificios desde los que fueron rescatados, Experimento Policías

La figura 40 muestra dos gráficos circulares (tal y como se presentaban en el experimento anterior) que representan la proporción de agentes rescatados sobre el total de agentes susceptibles de serlo. El gráfico circular de la izquierda ha sido construido con la simulación de policías y agentes sanitarios con normas activas. A la derecha se muestra un gráfico obtenido de simular exclusivamente con la activación

de las reglas de los agentes sanitarios. Se observa una leve mejora de algo más de 6 puntos por lo que respecta a la gente rescatada cuando los policías están con reglas activas frente a cuando no lo están. Dado que ha sido usado el mismo escenario, el número de agentes atrapados por culpa del seísmo programado es siempre de 25 agentes inmovilizados. Por ello el 75.71% equivale a cerca de 19 agentes rescatados frente aproximadamente a los 17 que equivale el 68.4%. Es interesante observar que los agentes con reglas han limpiado cerca de 2 caminos y las reglas que consideran esta tareas son: n_{206}, n_{218}, n_{221} y una de ellas, la regla n_{206} limpia un camino que alcanza hasta un edificio altamente dañado y ocupado por lo que dado que las tres normas son de facultad y todas ellas han tenido la misma probabilidad de seguimiento por parte de los agentes de policía, lo más probable es que uno de cada tres caminos limpiados haya provocado una via de acceso a un agente con necesidad de ser rescatado.

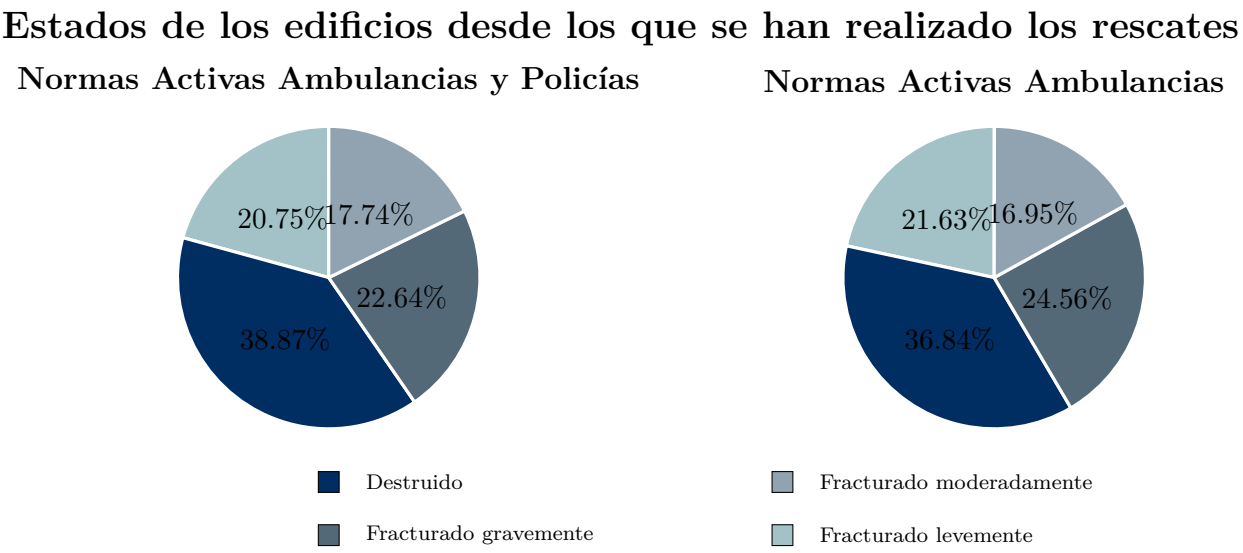


Figure 41: Gráfico circular que representa la proporción de rescates en función del nivel de rotura de los edificios desde los que fueron rescatados, Experimento Policías

La figura 41 muestra dos gráficos circulares que representan la proporción de los estados de los edificios desde los que se sucedieron los rescates. Los resultados obtenidos no presentan diferencias notables en sí, se observa un incremento de 2 puntos en la proporción de edificios destruidos en el gráfico de la derecha respecto al de la izquierda, pero sin embargo, el izquierdo pierde aproximadamente el mismo número de puntos en la proporción de edificios gravemente fracturados. Pareciera que los criterios de selección de objetivos han sido semejantes en ambas situaciones, sin embargo el leve incremento en los rescates ocurridos en edificios destruidos podría relacionarse otra vez con la norma n_{206}.

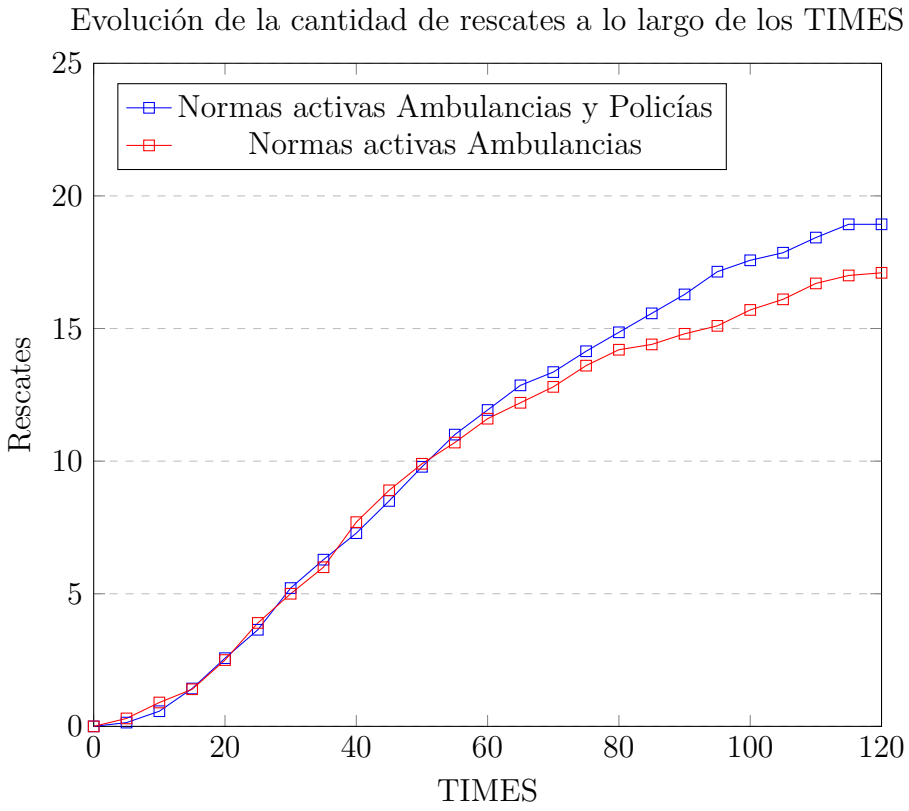


Figure 42: Evolución de la cantidad de rescates a lo largo de los TIMES, Experimento Policías

De forma análoga al experimento anterior, la figura 43 muestra la evolución de la cantidad de rescates a lo largo de los TIMES. En este caso la función azul resulta de la simulación con reglas de policías y agentes sanitarios, mientras que la función roja únicamente considera las normas de los sanitarios. Se observa una un comportamiento prácticamente igual por parte de ambas funciones desde el origen de coordenadas hasta el TIME 60. A partir de entonces se genera una distancia entre ellas, elevándose la azul por encima de la roja. Podría concluirse que la acción de los agentes de policía marcó la diferencia a partir del TIME 60. Si los policías con normas han posibilitado un número mayor de rescates esto se debe a la limpieza de bloqueos que mantenian de otro modo a agentes sanitarios sin capacidad de alcanzar sus objetivos.

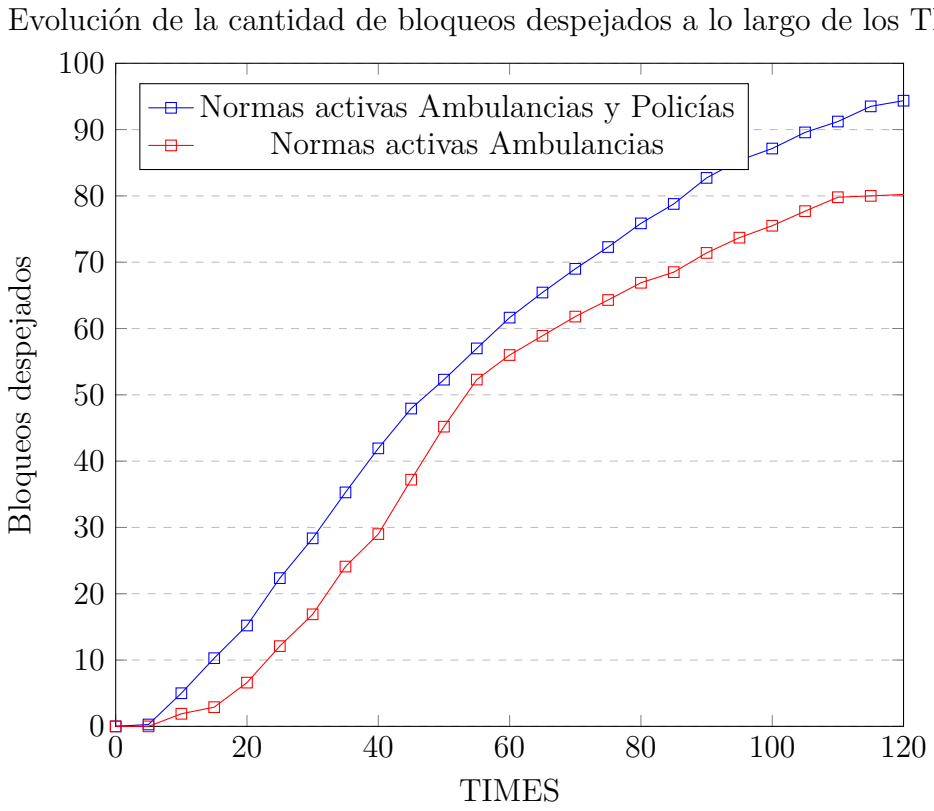


Figure 43: Evolución de la cantidad de bloqueos despejados a lo largo de los TIMES, Experimento Policías

La figura 43 muestra, mediante funciones discretas, la cantidad total de bloqueos despejados por los agentes de policía. Como anteriormente, la función azul ha sido obtenida de la simulación de agentes tanto de policía como sanitarios con normas activas. La roja es producto de los datos obtenidos considenrando exclusivamente agentes sanitarios con normas activas. Las funciones en este caso sí muestran una diferencia significativa. La función azul se mantiene por encima de la roja desde el centro de coordenadas hasta el final por lo que se deduce que los agentes de policía con reglas han sido significativamente más veloces que aquellos que no tenían. Limpiar un mayor número de bloqueos también incrementa la probabilidad de dar con uno que esté incapacitando a un agente sanitario y por lo tanto estos despejes pueden provocar nuevos rescates.

El incremento de la velocidad debe resultar de la aplicación de una o más reglas que potencian esta característica. Se deduce que los objetivos señalados por el seguimiento de las reglas que estén en ello implicadas de media se hallan más cerca del agente de policía de modo que alcanzarlo le supone un número menor de TIMES. Como ya ha sido mencionado en ocasiones anteriores la asignación base de los policías, al igual que la de los bomberos, viene determinada por el Solver **BinaryMaxSum** y precisamente estas se hallarían, por lo ya comentado, de media algo más lejos. La regla n_{242} forma parte de las reglas que puedan estar implicadas en este aumento puesto que su seguimiento fuerza a los agentes a la limpieza de

los bloqueos, no atendidos por otro policía, que se hallen en su radio acción, son por lo tanto siempre objetivos muy cercanos los que propone esta regla en concreto. Tambiés las reglas n_{206}, n_{218} y n_{221} son susceptibles de incrementar la velocidad en determinadas ocasiones. Todas ellas fuerzan al policía que las sigue a despejar un camino que se sabe bloqueado, el camino viene determinado por la ruta más corta que une al agente con el objetivo, en el caso de la regla n_{206} será el edificio más dañado, en caso de n_{218} un edificio con rotura inferior al 50% y finalmente si se sigue n_{221} será un edificio con rotura mayor al 50% (Véase la sección *Normativa*). Despejar la ruta implica la limpieza de todos los bloques que se hallen en las distintas calles que la componen, en un escenario en el que los bloqueos son generados con cierta uniformidad por todo el mapa como sucede con el uso de **CollapseSimulator** y la simulación del seísmo, provoca que los bloqueos objetivo a menudo se hallen cerca del policía, porque la ruta más corta también implica las rutas más cortas entre las posiciones de las que se compone esta por lo que el policía se moverá entre los bloqueos que halla también haciendo uso del camino más corto entre ellos. Igualmente habría que analizar detenidamente el comportamiento desarrollado por la activación de una o algunas reglas y no todas con el proposito de aislar cuales pueden ser el resto de reglas partícipes.

8 Planificación

La realización de este trabajo comenzó el *14 de enero de 2019*. Las tareas implicadas se han enumerado según su orden cronológico y se adjunta el tiempo estimado en cada tarea:

1. Introducción a la plataforma Robocup Rescue Simulator. 1 mes y medio, *28 de febrero de 2019*
2. Introducción a la tool BRMS Drools. 1 semana, *7 de marzo de 2019*
3. Integración de Drools. 2 semanas, *21 de marzo de 2019*
4. Implementación de la normativa de bomberos. 3 semanas, *11 de abril de 2019*
5. Implementación de la normativa de policías. 3 semanas, *2 de mayo de 2019*
6. Implementación de la normativa de sanitarios. 3 semanas, *23 de mayo de 2019*
7. Realización de la memoria. 1 mes, *20 de junio de 2019*

Aunque posteriormente a la reselección de normas con la inclusión de la regla n_{283} ciertos cambios sobrevinieron y un tiempo añadido fue requerido para implementar las normas no incluidas en el conjunto original. Después del reajuste la planificación quedó como:

1. Introducción a la plataforma Robocup Rescue Simulator. 1 mes y medio, *28 de febrero de 2019*

2. Introducción a la tool BRMS Drools. 1 semana, *7 de marzo de 2019*
3. Integración de Drools. 2 semanas, *21 de marzo de 2019*
4. Implementación de la normativa de bomberos. 3 semanas, *11 de abril de 2019*
5. Implementación de la normativa de policías. 3 semanas, *2 de mayo de 2019*
6. Reselección de normas
7. Implementación de nuevas normas de bomberos y policías. 1 semana, *9 de mayo de 2019*
8. Implementación de normativa de sanitarios. 2 semanas, *23 de mayo de 2019*
9. Realización de la memoria. 1 mes, *20 de junio de 2019*

9 Conclusiones

La coordinación de agentes de bomberos, policías y sanitarios mediante la normativa, experimentalmente ha obtenido resultados positivos y acordes con los valores éticos implicados en la selección, preponderando la reducción de las pérdidas humanas. Además las normas han permitido dotar de un mayor sentido las acciones de los agentes implicándoles en objetivos más complejos de lo que han estado mantuviéndose atados a la inmediatez de un algoritmo que selecciona objetivos que base las decisiones en el contexto actual sin previsión a un futuro cercano y además no considera en su función de costos la pérdida de vidas humanas tal y como son movidos por defecto los agentes. La inmediatez se pierde cuando los agentes implicados deben cumplir un objetivo que es capaz de demorarse en el tiempo como la limpiezas de rutas y no simplemente el despeje de bloqueos puntuales.

En este trabajo ha sido usado un nivel de percepción muy elevado en todos los agentes de modo que eran capaces de conocer los cambios tales como la aparición de un nuevo foco o la situación crítica de una gente atrapado varias calles de distancia. La realidad, sin embargo, propone retos más difíciles e incrementar el rango de percepción de los agentes de rescate es clave para una rápida reacción por lo que la apuestar por las ciudades inteligentes sería un gran paso en esta dirección.

Respecto a los agentes de bomberos, y también sanitarios el nivel elevado de percepción junto a los mecanismos de comunicación les han permitido desenvolverse mejor en el mapa obteniendo rutas alternativas cuando les era posible y alcanzar los incendios antes. Salvaguardar una estructura de importancia como pueda ser un hospital conlleva un gato excepcional hasta que termina extinguiéndose y en un contexto con múltiples focos la propagación de esto puede agravarse, por ello parece necesario contemplar estas consecuencias y estudiar si los indencios que están dejándose de controlar podrían poner en riesgo una estructura de misma o mayor importancia que aquella que se está tratando de proteger. Los rescursos hidratantes

también han demostrado ser imprescindibles en el desarrollo de los bomberos, sin ellos las posibilidades de éxito se reduce drásticamente y apenas serían capaces de extinguir fuegos estériles, incapaces de propagarse debido a su localización.

Respecto a las bajas entre agentes los civiles obtiene aquí siempre el primer lugar mientras que el segundo parece depender del escenario en el que rivalizan los agentes de bomberos y sanitarios. Los bomberos se mantienen cerca de los edificios y los sanitarios acceden a ellos para realizar tareas de rescate, las llamas y los cascotes son dos perjuicios a los que se exponen. Resulta de ello que medidas de protección para ambos colectivos deberían maximizarse, si los agentes de rescate sufren heridas que los incapacitan un menor número de ellos en condiciones estar'a dispuesto a para socorrer a civiles.

Finalmente respecto a los agentes de policía resulta bastante positivo que puedan coordinarse con el resto de agentes y si el camino a despejar puede converger con el foco de incendios sobre que el se quieren dirigir los bomberos o alcanza el desbloqueo de accesos a edificios colapsados con personas atrapadas y objetivo de los agentes sanitarios resulta en una mejora en el desarrollo de la acción de los últimos dos tipos de agentes. Tal y como por ejemplo la regla n_{206} y n_{110} o la preja de normas n_{116} y n_{221} son capaces de coordinar bajo ciertas circunstancias la acción de policías y sanitarios para posibilitar el acceso de estos últimos al edificio.

10 Trabajo futuro

- Resultaría importante realizar un mayor número de simulaciones para extraer resultados de mayor fiabilidad.
- Coordinar los simuladores **BlockadeLoader** y **CollapseSimulator** con motivo de generar escenarios más interesantes en los que tanto se generasen bloqueos que impidiesen el acceso a otra calles como bloqueos que impiediesen el acceso a edificios.
- Hallar mejoras en la eficiencia computacional para habilitar el desarrollo de escenarios de mayor complejidad sin extender en damasía el tiempo total de la simulaci'on. Otra opción podría ser incrementar el hardware rentando quizás un servicio Cloud y usar un clúster de ordeadores para pruebas de mayor complejidad.
- Realizar los cambios necesarios para hacer compatible unos niveles de percepción humanos con el desarrollo de los agentes. Y realizar una integración más profunda de la comunicación entre agentes para que tome esta un papel más importante por lo que refiere al análisis que sobre su propia instancia del mundo hace cada agente. Dependiendo más de la información compartida que de su propia percepción una vez que esta se configure a niveles normales.

- Implementar las normas n_{275} y n_{277} que no fueron incluidas en este trabajo y probar sus efectos.
- Reimplementar las normas n_{37} y n_{283} para que considere a los edificios vecinos por cercanía y no como lo hacen ahora, por conexión en el grafo. Las conexiones a veces no resultan intuitivas y dos edificios contiguos pueden no ser vecinos al no estar conectados.

Bibliografia

[1] Marc Pujol. Trabajo final de master, rmasbench. Universidad de Barcelona, 2015.

[2] Marc Serramia, Maite Lopez-Sanchez, Juan A. Rodriguez-Aguilar, Filippo Bistaffa, Paula Boddington, Michael Wooldridge, Carlos Ansotegui, Javier Morales, and Manel Rodriguez. Computing ethical norm systems based on a formalization of moral values. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1294–1302. International Foundation for Autonomous Agents and Multiagent Systems, 2019.

[3] Marc Serramia, Maite Lopez-Sanchez, Juan A Rodriguez-Aguilar, Manel Rodriguez, Michael Wooldridge, Javier Morales, and Carlos Ansotegui. Moral values in norm decision making. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1294–1302. International Foundation for Autonomous Agents and Multiagent Systems, 2018.