UNIVERSIDAD SIMÓN BOLÍVAR

Ingeniería de la Computación

Aplicación de Técnicas de Aprendizaje Evolutivo en Sistemas Multiagentes (Robocup Rescue)

Por

David Ojeda, Luis E. Useche y Ezequiel A. Zamora

Proyecto de Grado

Presentado ante la Ilustre Universidad Simón Bolívar

como Requerimiento Parcial para Optar el Título de

Ingeniero en Computación

Sartenejas, Diciembre de 2005

UNIVERSIDAD SIMÓN BOLÍVAR

DECANATO DE ESTUDIOS PROFESIONALES

COORDINACIÓN DE INGENIERÍA DE COMPUTACIÓN

ACTA FINAL DEL PROYECTO DE GRADO

APLICACIÓN DE TÉCNICAS DE APRENDIZAJE EVOLUTIVO EN SISTEMAS MULTIAGENTES (ROBOCUP RESCUE)

Presentado Por:

DAVID OJEDA, LUIS E. USECHE Y EZEQUIEL A. ZAMORA

Este proyecto de Grado ha sido aprobado por el siguiente jurado examinador:

Prof. Emely Arráiz

Prof. Diego Mosquera Uzcátegui

Prof. Ivette Carolina Martínez (Tutor Académico)

SARTENEJAS, 11 de enero de 2006

Aplicación de Técnicas de Aprendizaje Evolutivo en Sistemas Multiagentes (Robocup Rescue)

Por

David Ojeda, Luis E. Useche y Ezequiel A. Zamora

RESUMEN

RoboCup Rescue Simulation System es un sistema de simulación de desastres utilizado como plataforma de prueba para técnicas y algoritmos en las áreas de sistemas multiagentes, inteligencia artificial y aprendizaje. Una de las principales motivaciones de este proyecto es impulsar el surgimiento de estrategias para coordinar situaciones de rescate en desastre naturales de la vida real. El sistema de simulación plantea una serie de problemas específicos tales como el rescate de víctimas enterradas, desbloqueo de calles y extinción de incendios. Esta investigación propone y desarrolla una solución al rescate de víctimas y extinción de incendios utilizando un sistema de aprendizaje evolutivo denominado clasificadores genéticos de tipo XCS. Este trabajo también utilizó algoritmos de búsqueda informada y de agrupamiento inherentes al área de inteligencia artificial para agilizar tanto la planificación de caminos como el combate de incendios. Además se definió un sistema híbrido de coordinación de agentes, que combina los enfoques centralizado y distribuido.

Los experimentos realizados en esta investigación demostraron que el sistema de aprendizaje resulta apropiado para la toma de decisiones en el sistema. Los agentes producto de este trabajo presentaron resultados satisfactorios que compiten con los participantes del concurso anual **RoboCup Rescue Simulation League** en su edición del 2004 y con un poco más de trabajo pueden estar preparados para presentarlos en la próxima edición de la competencia.

Agradecimientos

En primer lugar, gracias a Dios.

A nuestros padres y familiares: Guaicaipuro Ojeda, Rosario Avellaneda, Verónica Ojeda, Elsy Rivas, Bárbara Espinoza, Elvira Becerra, Ysabel Frontado, Ezequiel Zamora, Luis Rondón, Luisa Frontado e Ysabel Rondón.

Agradecemos especialmente a nuestra tutora Caromar (Ivette Carolina Martínez).

A todos nuestros compañeros cuya ayuda fue invaluable para el desarrollo de este proyecto: Carolina Chang, Eduardo Ruiz (Hawaii), Jorge Guerra, Daniela Ascanio, Simón Ortiz, Mariana Bustamante, Julio Castillo, Carlos Castillo, Miguel Castro y Carlos Figueira.

A los habitantes del Grupo de Inteligencia Artificial, con quienes compartimos largas jornadas de trabajo y esparcimiento: Francelice Sardá, Celso Gorrín, Pedro Piñango, Diego Guerrero, Marynel Vásquez, Kenny Vivas y Sra.

A todos los integrantes de las Aulas Computarizadas, que nos acompañaron durante nuestra carrera: Eduardo Ascanio, Yolifé Arvelo, Glenville Farmer, Edwin y Adriana Santeliz, Jenny Ng Wong. Especialmente los integrantes actuales quienes absorbieron nuestra carga de trabajo: Luis Ovalle (JLO), Nicolás Lara, Gustavo González, Héctor Rodriguez, Andrés Salazar, Vanessa Ferro, Jorge Duque, Jose Mangialomini, Raúl Álvarez, Mauricio Hollando y Pablo Osers.

Índice general

Índice de cuadros

Índice de figuras

Glosario de Términos

ADK Agent Development Kit, API para el desarrollo de agentes para el RCRSS en el lenguaje C++.

AEF Agrupamiento de Edificios en Fuego, grupos de edificios el cual poseen información en conjunto.

AG Genetic Algoritm, técnica usada para la generación de nuevas soluciones en los algoritmos evolutivos.

API Application Program Interface, interfaz de programación para una aplicación.

Charset Nombre usado para referirse a la representación computacional de un conjunto de caracteres.

Ciclo Simulación de un minuto del mundo real en RCRSS.

KIF Knowledge Interchange Format fue creado para servir como sintaxis de lógica de primer orden el cual es procesado por una computadora fácilmente.

RCR Robocup Rescue, proyecto que incentiva el desarrollo en las distintas áreas de Inteligencia Artificial.

RCRSS Robocup Rescue Simulation System, simulador de desastre para pruebas de algoritmos en situaciones de rescate.

Script Un conjunto de comandos escritos en un lenguaje interpretado para automatizar ciertas tareas.

Subsumir Considerar algo como parte de un conjunto más amplio o como caso particular sometido a un principio o norma general.

UDP User Datagram Protocol, protocolo no orientado a conexión que se ubica sobre el protocolo de red IP.

UML Unified Modeling Language, es un lenguaje estándar para la especificación, visualización, construcción y documentación de los artefactos de un sistema.

XCS Tipo sistema de clasificadores genéticos cuya aptitud esta medida según la exactitud de la predicción que tenga el sistema sobre la recompensa.

YabAPI API para el desarrollo de agentes para el RCRSS en el lenguaje Java.

Capítulo 1

Introducción

RoboCup es un proyecto que tiene como finalidad el impulsar la investigación en el área de la robótica, la inteligencia artificial, los sistemas multiagentes y otros campos relacionados [RoboCupFederation, 2004].

El 17 de Enero de 1995 se produjo el Terremoto de *Hanshin-Awaji* en la ciudad de Kobe, Japón, dejando un saldo de alrededor de 6500 muertes y más de un millón de afectados [Tadokoro et al., 2000]. Este desastre natural impulsó la creación, en el año 2000, de una rama de RoboCup llamada *Robocup Rescue* la cual tiene como objeto de estudio un sistema multiagentes cuyo objetivo es salvamento de víctimas civiles en caso de desastres. Este sistema está situado en un ambiente urbano con un alto nivel de incertidumbre, información parcial, comunicación limitada y tiempo de toma de decisiones limitado.

Robocup Rescue es *"un problema de planificación de comportamiento semióptimo con restricciones extremadamente complejas y múltiples objetivos de amplia variación en el tiempo"* [Tadokoro et al., 2000]. La idea de *Robocup Rescue Simulation System* es construir un grupo de agentes que cooperen de manera coherente y que muestren comportamientos inteligentes que minimicen los efectos del desastre natural.

Actualmente muchos investigadores se encuentran dedicados a la creación de equipos de agentes que solucionen el problema de *Robocup Rescue* usando diferentes arquitecturas de agentes, modelos probabilísticos y métodos de aprendizaje que han demostrado ser exitosos en su labor, como por ejemplo, *ResQ Freiburg* [Kleiner et al., 2004], *DAMAS Team* [Paquet et al., 2004], *5Rings* [Silva y Coelho, 2004], *RoboAkut* [Eker y Akin, 2004], *Black Sheep Team* [Skinner et al., 2004] y *Nit Rescue* [Honji et al., 2004].

El presente informe presenta el trabajo de **Tuqueque Team**: una aproximación a la solución al problema de *Robocup Rescue* utilizando sistemas de clasificadores genéticos ([Wilson, 1997]) como soporte para la toma de decisiones, agrupamiento de calles y edifi-

cios, coordinación de agentes usando un esquema híbrido, arquitectura de subsunción como modelo de comportamientos individuales de agentes, un esquema de comunicación que aprovecha al máximo las limitaciones del sistema y algoritmos de búsqueda informada en grafos para la planificación de rutas.

Las medidas de desempeño definidas para la evaluación de los agentes fueron el puntaje, número de agentes vivos y edificios salvados al final de cada simulación. En base a éstas, se realizó una comparación entre varios agentes externos a este estudio con dos versiones de **Tuqueque Team**, una entrenada y otra con reglas aleatorias.

En base a la comparación de las métricas antes mencionadas, se pudo concluir que la solución desarrollada en este estudio es exitosa para la resolución del problema. Sin embargo, se identificaron elementos de diseño que pueden ser modificados para mejorar la calidad de los resultados obtenidos por los agentes de **Tuqueque Team**.

El trabajo consta de seis capítulos. El capítulo 2 contiene el basamento teórico necesario para el desarrollo de la aproximación a la solución de *Robocup Rescue*. En el capítulo 3 se describen los sub-problemas identificados del problema *Robocup Rescue* y el diseño de sus soluciones. El capítulo 4 describe los detalles de implementación de cada una de las soluciones y los problemas encontrados durante el proceso. El capítulo 5 detalla los esquemas para el entrenamiento de los sistemas de clasificadores y se presentan los resultados de este proceso. Además se diseñan y analizan experimentos para observar y comparar el funcionamiento de la solución planteada para el problema de *Robocup Rescue*. Y por último, el capítulo 6 expone una serie de conclusiones y recomendaciones para trabajos futuros.

Capítulo 2

Marco Teórico

Este capítulo contiene el basamento teórico usado para el desarrollo de la aproximación a la solución de *Robocup Rescue* implementada por el equipo **Tuqueque Team**.

2.1. El proyecto RoboCup

RoboCup es un proyecto que tiene como finalidad el impulso de la investigación en el área de la robótica, la inteligencia artificial, los sistemas multiagentes y otros campos relacionados. En 1993 se lanzó por primera vez con el nombre tentativo de Robot J-League por un grupo de investigadores dentro de los cuales estaban: Minoru Asada, Yasuo Kuniyoshi y Hiroaki Kitano. Luego para la internacionalización del proyecto se cambió el nombre a *Robot World Cup Initiative* y como abreviación se obtuvo *RoboCup* [RoboCupFederation, 2004].

De este proyecto se derivan una serie de ramas entre las cuales están:

- **RoboCupSoccer:** Esta es la principal rama del proyecto, tiene como tema de investigación de sistemas multiagentes en el juego del fútbol. Se traza la meta de que a mediados del siglo XXI un grupo de robots sean capaces de ganar un juego completo a los campeones mundiales humanos.

- **RoboCupJunior:** Tiene como principal objetivo impulsar la investigación en el área de robótica para estudiantes de primaria y secundaria.

- **RoboCupRescue:** Tiene como principal motivación el rescate de víctimas civiles en caso de desastres naturales [RCRCommitee, 2005].

2.1.1. RoboCupRescue

Esta rama del proyecto RoboCup nace motivado al terremoto ocurrido el 17 de enero de 1995 en Hanshin-Awaji. Este desastre natural tuvo como resultado más de 6400 muertos y 530000 edificios dañados [Morimoto, 2005b].

RoboCupRescue ofrece una serie de proyectos que incentivan el desarrollo en los campos de coordinación de multiagentes, robótica para la búsqueda y rescate y soporte de decisiones, entre otros. Su principal objetivo es canalizar los esfuerzos y aprovechar la tecnología en cada una de estas áreas para salvar vidas en desastres naturales.

Entre los proyectos más importantes de *RoboCup* se encuentra el *RoboCup-Rescue Simulation Project* [RCRCommitee, 2005]. Una de las principales tareas del proyecto ha sido la de coordinar el desarrollo de los componentes del simulador de desastre desde su creación en el año de 1999. La idea es simular una ciudad completa por un tiempo prolongado luego de haber sufrido las consecuencias de un terremoto. Al principio de la simulación el escenario refleja las consecuencias directas del desastre natural, como los incendios de edificios, los bloqueos de vías de comunicación y los derrumbes de estructuras. A medida que avanza la simulación, se presentan consecuencias indirectas del desastre, como propagación de incendios, réplicas del terremoto original e incapacitación de civiles y agentes por el derrumbe de edificios.

Robocup Rescue Simulation System es un sistema que reune todas las condiciones para ser utilizado como plataforma de pruebas para nuevos algoritmos y estrategias de coordinación de sistemas multiagentes y algoritmos distribuidos [Tadokoro et al., 2000].

Para motivar el desarrollo de agentes para el RCRSS se creó el proyecto *RoboCup-Rescue Simulation Competition* [RCRCommitee, 2005]. Aquí los desarrolladores de todo el mundo pueden evaluar sus resultados comparándolos con los demás competidores. La idea es minimizar los efectos posteriores causados por el desastre en la ciudad. Cada equipo desarrolla sus agentes para cumplir las tres tareas principales que se les plantea: rescatar víctimas en edificios derrumbados, extinguir edificios y desbloquear los caminos lo más rápido posible. Este evento se lleva a cabo anualmente desde el 2002 y tiene alta participación con alrededor de 20 competidores provenientes de Alemania, Irán, India, Italia, Canadá, entre otros.

2.2. Caracterización del dominio

Robocup Rescue es *"un problema de planificación de comportamiento semióptimo con restricciones extremadamente complejas y múltiples objetivos de amplia variación en el tiempo"* [Tadokoro et al., 2000]. La idea es crear un grupo de agentes que cooperen de manera coherente y que posean comportamientos inteligentes que minimicen los efectos del desastre natural.

El problema de *Robocup Rescue* consiste en que un conjunto de agentes maximicen su desempeño en un ambiente simulado, el cual será evaluado utilizando la función de evaluación descrita por la ecuación (1) (que da la puntuación a los equipos), donde P es el número de agentes vivos, S es la suma de la cantidad de vida restante de todos los agentes, $S_{inicial}$ es la cantidad de vida inicial de las víctimas B es el área de los edificios no quemados y $B_{inicial}$ el área total de todos edificios [Llona, 2004], [RCRCommitee, 2004b] y [Tadokoro et al., 2000].

$$\text{Puntaje} = \left(P + \frac{S}{S_{inicial}}\right) \cdot \sqrt{\frac{B}{B_{inicial}}} \tag{1}$$

Cada simulación de *Robocup Rescue Simulation System* se ejecuta en una escala temporal de 1:60 la cual representa las 5 horas subsiguientes al terremoto. La plataforma de simulación de *Robocup Rescue* impone una serie de restricciones y problemas computacionales inherentes al dominio, los cuales enumeramos a continuación:

- Comunicación limitada de los agentes.
- Incertidumbre elevada proveniente de las variaciones aleatorias del ambiente. Además restricciones diversas que varían en el tiempo como disminución de la cantidad de agentes de rescate vivos, salud de las víctimas, expansión de incendios.
- Fuerza de cómputo limitada.
- Insuficiente capacidad de los agentes ante la magnitud del problema.
- Información incompleta o parcial.

- Tiempo de toma de decisiones muy limitado.
- Comportamiento coordinado y cooperativo de agentes heterogéneos.
- Transferencia del conocimiento entre los distintos agentes de la simulación.

2.3. Plataforma de Simulación

El *Robocup Rescue Simulation System* (RCRSS) [RCRCommitee, 2005], como su nombre lo indica, es un conjunto de simuladores que proveen la información necesaria de cada uno de los componentes ambientales que integran el espacio de desastre. **Tuqueque Team** utiliza la versión 0.45 del simulador como plataforma para el desarrollo de sus agentes.

2.3.1. Simuladores

El sistema de simuladores está compuesto por una serie de módulos que se conectan a un componente principal o kernel como se muestra en la figura 1. Estos módulos comparten información detallada de todos los aspectos de la simulación y se comunican utilizando un protocolo creado por sus desarrolladores el cual se encuentra explicado en detalle en el *Manual de Desarrollo de Agentes de Robocup Rescue* [Morimoto, 2002].

Esta arquitectura modular otorga una serie de ventajas deseables en este tipo de proyectos como lo son la diversidad y flexibilidad en la elección de lenguajes de programación para la codificación de cualquiera de los simuladores y la ejecución distribuida de cada uno de los módulos.

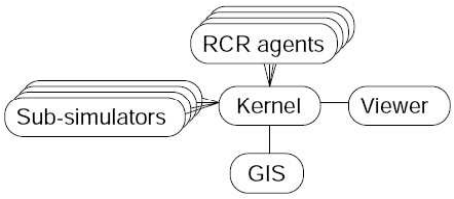


Figura 1: Arquitectura del RCRSS

Cada uno de estos componentes cumple una función muy específica en el *RCRSS*. A continuación se explican brevemente cada uno de ellos [Morimoto, 2005b]:

- **Sistema de Información Geográfica:** también conocida como GIS por sus siglas en inglés, es el componente que provee toda la información inicial del espacio de desastre, información geográfica y topológica de la ciudad e información sobre el terremoto ocurrido.

- **Kernel:** es el punto de encuentro de cada una de las partes del simulador. Se encarga de recibir y enviar información, de manera selectiva, proveniente de todos los módulos del simulador y los agentes.

- **Visor:** se encarga de mostrar todos los componentes de la ciudad y su estado actual. Provee información adicional como el puntaje logrado hasta el momento, el estado de los agentes, las acciones que realizan y el ciclo actual de la simulación. Existen varias propuestas de desarrollo de este componente, las más conocidas son la de Takeshi Morimoto [Morimoto, 2005a], la de Yoshitaka Kuwata [Kuwata, 2005] y un visor tridimensional desarrollado en la Universidad de Freiburg en Alemania [Kleiner y Göbelbecker, 2005].

- **Simulador de Misceláneos:** se encarga de simular el comportamiento de algunas propiedades de los agentes, como sus niveles de daño, vida e incapacidad por encontrarse tapiados.

- **Simulador de Tráfico:** se encarga de simular el tráfico de los agentes civiles y de rescate tomando en cuenta, semáforos, canales disponibles y libres de bloqueos en cada calle, congestión por uso simultáneo de vías, entre otros.

- **Simulador de incendios:** se encarga de simular la propagación de los incendios en edificios considerando factores como dirección y velocidad del viento, propiedades de los edificios como su área, número de pisos, material, entre otros. Realiza una micro-simulación utilizando modelos de los procesos de combustión, ignición, propagación y extinción de materiales.

- **Simulador de colapsos:** simula el colapso de edificios basándose en la información del terremoto de *Hanshin-Awaji* sobre la relación de la aceleración de superficie, estructura

y edad de los edificios con su nivel de destrucción.

- **Simulador de bloqueos:** simula el bloqueo de cada una de las calles en la ciudad usando información sobre el terremoto de *Hanshin-Awaji* sobre la relación del ancho de las calles y la escala sísmica con su nivel de bloqueo.

- **Civiles:** simula el comportamiendo estándar de los civiles. Los civiles que no han sido dañados por los efectos del terremoto se trasladan hacia el refugio más cercano y aquellos que han sido inhabilitados por encontrarse tapiados solicitan ayuda por medio de mensajes de voz.

2.3.2. Espacio de desastre

El espacio de desastre que conforma una ciudad de *Robocup Rescue* se compone de un conjunto de edificios, calles, nodos y humanoides distribuidos en un ambiente bidimensional. A continuación se presenta una descripción de los atributos y cualidades de estos elementos:

Edificios

En *Robocup Rescue Simulation System* existen tres tipos de edificios, los civiles, los centros de comando y los refugios. Los atributos mencionados en el cuadro 9 del apéndice B aplican para todos los edificios de naturaleza civil presentes en el espacio de desastre de la simulación. Existen edificios que son invulnerables al fuego y a los colapsos, eston son, el *Ambulance Center*, *Police Office*, *Fire Station* lo cual les facilita su labor de coordinar las acciones. También los refugios están exentos de sufrir los estragos del terremoto con el fin de facilitar las tareas de rescate.

Los refugios son edificios especializados que cumplen con dos funciones primordiales. En primer lugar son el sitio en donde todos los civiles pueden guarecerse. En el momento en que un civil llegua al refugio deja de perder vida automáticamente. La segunda es permitir a los agentes de tipo *Fire Brigade* rellenar su tanque de agua.

Nodos

Los nodos son estructuras artificiales creadas con la finalidad de modelar la estructura del mundo en una simulación. Son utilizados para indicar el inicio y final de cada calle así como las entradas a un edificio. Sus atributos están específicados en el cuadro 10 del apéndice B.

Calles

Las calles permiten la circulación de los diferentes agentes por el espacio de desastre. Pueden estar obstruidas como consecuencia de los efectos de los derrumbes de los edificios adyacentes a ellas. Están modeladas como un arco entre un par de nodos. Sus atributos están específicados en el cuadro 11 del apéndice B.

Agentes

Los agentes que viven en el ambiente simulado pueden clasificarse de dos maneras, por su utilidad en la simulación en equipos de ambulancias de bomberos, de policias y civiles; y por su estructura en agentes base y agentes humanoides.

Los agentes base poseen las propiedades expuestas en el cuadro 9 y los agentes humanoides comparten los atributos explicados en el cuadro 12. Los agentes de rescate, que son humanoides, también son llamados agentes pelotón.

Los bomberos tienen un atributo adicional llamado *WaterQuantity* que representa la cantidad de agua disponible en el tanque, medida en litros. Inicialmente tiene un valor de 15.000. y puede disminuir su valor hasta en 1.000. unidades por turno cuando se realiza la acción *extinguish*.

Agentes pelotón

Poseen movilidad y cualidades especiales para cumplir con algunas tareas específicas en la ciudad. A continuación se describen cada uno de los tres tipos de agentes humanoides en el RCRSS:

- ***Ambulance Team*:** Se encargan de rescatar a los agentes cubiertos por los escombros de un edificio y transportarlos al hospital.

- ***Police Force*:** Tienen como tarea desbloquear las calles.
- ***Fire Brigade*:** Se encargan de apagar los edificios en llamas.

A pesar de conocer todo el conjunto de calles y edificios que componen el mundo, cada agente tiene una visión limitada pues solo conoce los atributos de un componente si se encuentra en su rango de visión. En *Robocup Rescue* el rango de visión está compuesto por un radio de 10 metros alrededor del agente. Una lista con todos los objetos del mundo dentro de este campo de visión es enviada por el kernel a cada agente al inicio de cada turno. Un ejemplo de campo de visión puede ser observado en detalle en la figura 2.

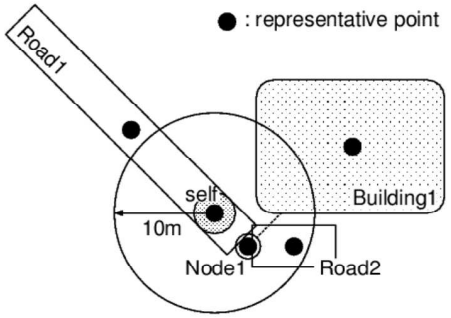


Figura 2: Rango de Visión de un Agente, tomado de [Morimoto, 2002]

Otro detalle importante a considerar es la comunicación. Los agentes de *Robocup Rescue* poseen 2 maneras de compartir información: por mensajes de voz y por mensajes de radio.

Los mensajes de radio emitidos por un agente pelotón son escuchados por todos los agentes de su mismo tipo y por su respectivo agente base, cada mensaje puede contener un máximo de 256 bytes de datos y cada agente pelotón puede enviar hasta 4 mensajes por turno. Vale la pena decir que la comunicación por radio no es totalmente confiable y que un agente pelotón sólo puede escuchar 4 de estos mensajes por turno.

Los mensajes de voz pueden ser escuchados por todos los agentes que se encuentren en un radio de 30 metros a la redonda con respecto al emisor.

Agentes base

Su tarea principal es la de servir de puente de comunicación para los mensajes de radio entre las distintas clases de agentes humanoides. El flujo de la comunicación por mensajes

de radio puede ser observado en la figura 3. Estos agentes también pueden desempeñar la tarea de coordinador en una arquitectura centralizada de toma de decisiones.

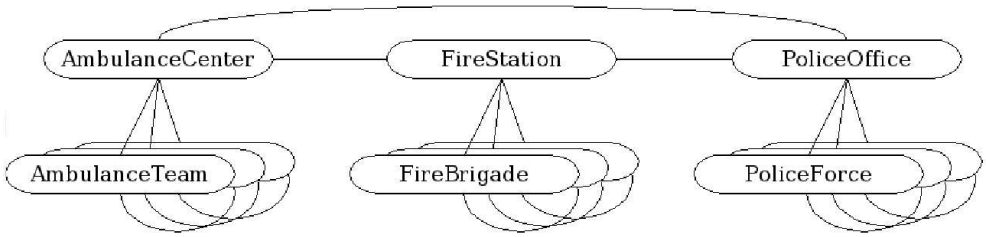


Figura 3: Flujo de Mensajes de RCRSS

Este tipo de agentes puede escuchar hasta $2N$ número de mensajes siendo N el número de agentes pelotón del mismo tipo que esta central.

Agentes civiles

Los agentes civiles son agentes humanoides y se comportan de acuerdo a las reglas definidas por el simulador de civiles como se explica en la sección 2.3.1.

Comandos

Existen dos tipos de comandos, las acciones, que tienen efectos sobre el mundo y los mensajes que sirven para el intercambio de información entre los agentes. Podemos observar las acciones y comandos de *Robocup Rescue Simulation System* en el cuadro 2, así como también, la capacidad de cada tipo de agente para realizarlas.

En el apéndice C se encuentra una descripción detallada de las acciones que pueden realizar los agentes de *Robocup Rescue Simulation System*.

2.3.3. Funcionamiento de la simulación

Una corrida del simulador tiene 300 ciclos. Cada ciclo toma aproximadamente un segundo y representa un minuto en el mundo real (escala 1:60). En cada una de las iteraciones se cumplen los siguientes pasos [RCRCommitee, 2000]:

1. El kernel envía información sensorial visual o auditiva a cada uno de los agentes. Es en este punto que se verifican las limitaciones sensoriales de cada agente.

	AT	FB	PF	AC	FS	PO
Percepción del mundo						
SENSE(visual)	✓	✓	✓	✓	✓	✓
HEAR(auditiva)	✓	✓	✓	✓	✓	✓
Acciones						
Move(visual)	✓	✓	✓			
Rescue(visual)	✓					
Load(visual)	✓					
Unload(visual)	✓					
Extinguish(visual)		✓				
Clear(visual)			✓			
Mensajes						
Say(mensaje de voz)	✓	✓	✓	✓	✓	✓
Tell(radiomensaje)	✓	✓	✓	✓	✓	✓
Cantidad de mensajes escuchables	4	4	4	2n	2n	2n
Cantidad de mensajes emitibles	4	4	4	2n	2n	2n

Cuadro 2: Capacidades de los agentes de RCRSS. (AT) *Ambulance Team*, (FB) *Fire Brigade*, (PF) *Police Force*, (AC) *Ambulance Center*, (FS) *Fire Station* y (PO) *Police Office*. 2n significa: tantos como el doble de la cantidad de agentes pelotón del mismo tipo

2. Los agentes deciden cual acción tomar y esta es enviada al kernel.

3. El kernel recibe cada una de las acciones enviadas por los agentes y las reenvía a todos los subsimuladores. Los agentes deben enviar sus acciones en un tiempo menor a 0.5 segundos, de lo contrario el comando es ignorado.

4. Los subsimuladores computan los cambios en la ciudad basándose en la información recibida y para luego enviar los resultados al kernel.

5. El kernel combina los resultados de los subsimuladores y los distribuye. Luego incrementa el tiempo de simulación y notifica a los visualizadores de la actualización.

6. Los visualizadores solicitan la información actualizada de la ciudad al GIS y muestran la nueva información.

7. El GIS actualiza el registro de la simulación.

El funcionamiento de *Robocup Rescue Simulation System* está ilustrado en la figura 4.

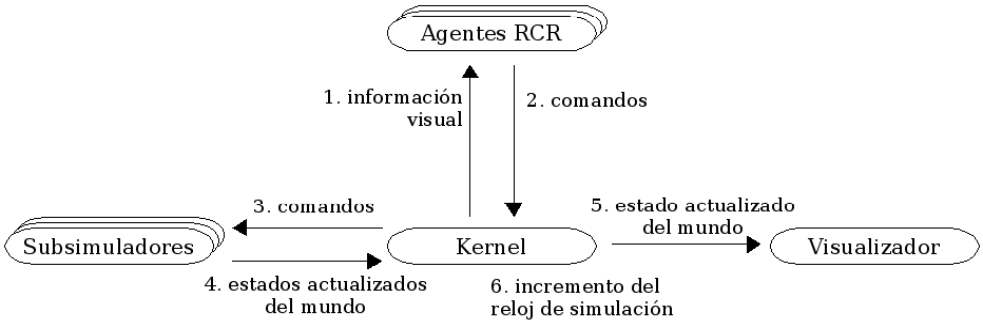


Figura 4: Intercambio de información de un turno de *Robocup Rescue*, tomado de [Llona, 2004]

2.4. Sistemas Multiagentes

"El estudio de los sistemas multiagentes se enfoca en sistemas donde múltiples agentes inteligentes interactúan entre sí" [Sycara, 2005].

Michael Wooldridge define un agente como un sistema computacional que está situado en un ambiente y tiene la capacidad de actuar de manera autónoma para alcanzar sus objetivos [Wooldridge, 2002]. Para este estudio se añadirá a esta definición que los agentes tienen la habilidad de interactuar con otros agentes.

Definiremos agente como un sistema computacional que tiene un objetivo específico. Puede percibir elementos del ambiente, es capaz de actuar en éste, y tiene la habilidad de interactuar con otros agentes.

Cuando se tienen dos o más agentes con objetivos comunes, surgen necesidades de coordinación y cooperación para que las acciones de un agente no interfieran con los objetivos globales y específicos de los demás agentes.

Las interacciones entre múltiples agentes pueden ser cooperativas o egoístas. En el caso de RCRSS, los agentes deben cooperar entre sí para alcanzar un objetivo común.

Para que un conjunto de agentes coopere es necesario que se comuniquen y trabajen

en equipo. También es necesario que los agentes negocien y realicen acuerdos, pero esto es particular de sistemas con agentes de naturaleza egoísta.

La comunicación es un mecanismo simple de interacción que puede ser implementado con un protocolo basado en mensajes simples o utilizando un lenguaje de comunicación entre agentes como KQML (Knowledge Query and Manipulation Language) o KIF (Knowledge Interchange Format) [Wooldridge, 2002]. Con éstos, los agentes pueden enviar información sobre elementos comunes del ambiente y hacer peticiones.

Existen diversos enfoques de coordinación para que los agentes trabajen en equipo. En primer lugar es necesario que el problema que se desea resolver sea analizado y descompuesto en sub-problemas, luego cada uno de éstos se resuelven, solucionando el problema original. Este procedimiento de planificador puede estar enfocado en uno o varios agentes. Cuando un grupo "maestro" de agentes realiza esta tarea, utilizando a otros agentes como "esclavos" que llevan a cabo las sub-tareas identificadas, se dice que existe un enfoque *centralizado* para resolver problemas. Por otro lado, cuando los agentes definen su propio plan de acción sin ayuda de un sistema central y coordinan sus acciones a medida que sea necesario, entonces existe un enfoque *distribuido*. Este enfoque es más difícil de manejar, especialmente si los agentes tienen objetivos particulares diferentes.

2.5. Clasificadores Genéticos XCS

Los *Sistemas de Clasificadores Genéticos XCS* son sistemas de aprendizaje basados en reglas de tipo condición/acción que evolucionan para mejorar la ejecución de una tarea. Se puede notar que este método de aprendizaje está basado en los clasificadores genéticos tradicionales [Holland, 1986] y su única diferencia está en el componente de reforzamiento [Wilson, 1997].

Tal como lo hacen los *Algoritmos Genéticos* [Mitchell, 1997] (AG), los XCS hacen la evaluación de sus decisiones por medio de un castigo o recompensa dado por el ambiente. Este es el marco de lo que conocemos hoy en día como *Aprendizaje por reforzamiento*.

Los XCS están compuestos de clasificadores o reglas que se encuentran contenidas en un

conjunto $[P]$. Cada uno de los clasificadores se compone de los siguientes elementos:

- Una condición
- Una acción
- Una predicción de recompensa
- Un estimado del error en la predicción ε
- Y una aptitud F cuyo valor es inversamente proporcional a ε

2.5.1. Selección de Acción en un Sistema XCS

En un sistema de clasificadores XCS la *selección de acción* es el mecanismo que se usa para obtener una acción a partir de una entrada del ambiente. Los pasos que se deben seguir para obtener la acción son:

1. Seleccionar las reglas del conjunto $[P]$ cuyas condiciones correspondan con el input dado
2. Con las reglas seleccionadas se forma el conjunto $[M]$
3. Luego se forma un *arreglo de predicciones* donde se colocan las predicciones de cada acción pesadas por sus aptitudes como se muestra en la fórmula (2) donde i es la regla con acción j

$$p_j = \frac{\sum_{i=0}^{n} (F_i \cdot p_i)}{\sum_{i=0}^{n} F_i} \tag{2}$$

   donde F_i es la aptitud de la regla i y P_i es la predicción de la recompensa de la regla i.
4. De este arreglo se escoge la acción con mayor predicción y se envía al ambiente
5. Las reglas con la acción elegida forman el conjunto de acciones $[A]$ y son evaluadas por el componente de reforzamiento como se explica en la sección 2.5.2

En la figura 5 se ejemplifica cada uno de los pasos descritos anteriormente.

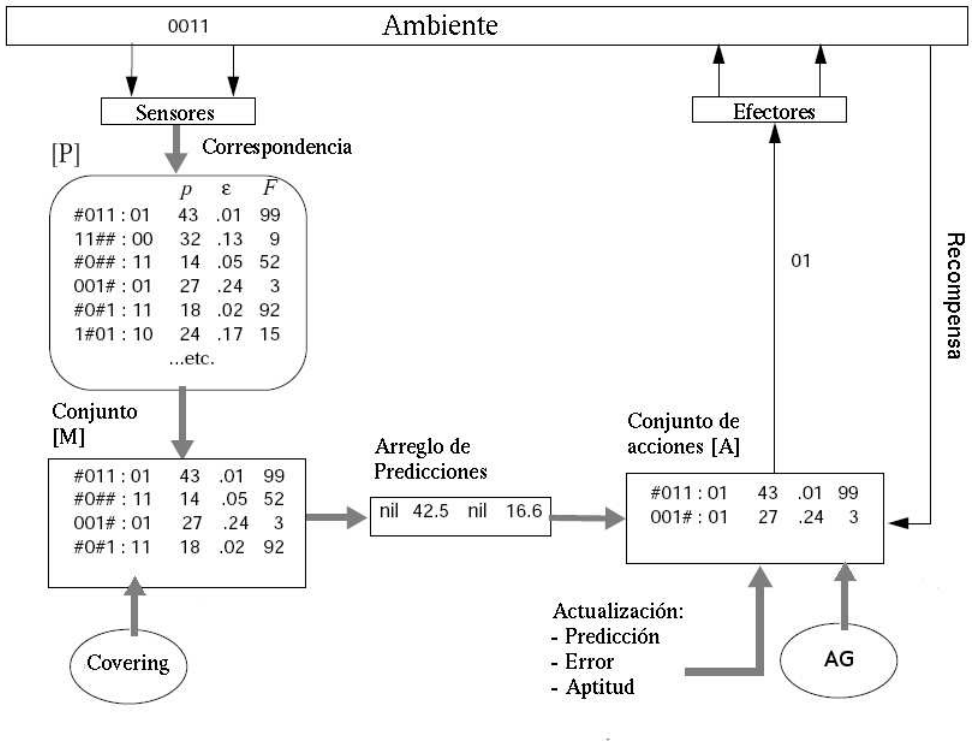


Figura 5: Ejemplo de un sistema XCS completo

2.5.2. Componente de reforzamiento de un sistema XCS

El componente de reforzamiento de un sistema XCS es el encargado de evaluar cada uno de los clasificadores basados en la recompensa obtenida del ambiente luego de hacer una selección de acción.

Con la recompensa obtenida se realizan ajustes en las predicciones de las reglas del conjunto $[A]$. Estas se hacen según la fórmula (3), donde R es la recompensa obtenida del ambiente, p_j es la predicción actual y α es la tasa de actualización ($0 < \alpha \leq 1$).

$$p_j \leftarrow p_j + \alpha(R - p_j) \tag{3}$$

Es importante también la actualización del error ε. Para esto se debe encontrar la diferencia entre la predicción y la recompensa obtenida. La actualización se puede hacer según la formula (4) donde ε_j es el error actual de la regla j, α es la tasa de corrección, p_j la predicción de la regla j y R la recompensa.

$$\varepsilon_j \leftarrow \varepsilon_j + \alpha(|R - p_j| - \varepsilon_j) \tag{4}$$

Y por último, se actualiza la aptitud F de cada regla del conjunto $[A]$. F va a a ser actualizado calculando la diferencia entre el F actual y la exactitud relativa con respecto a las demás reglas del conjunto $[A]$. A su vez, este necesita conocer la exactitud de la regla. De esta manera el cálculo queda resumido en las fórmulas (5), (6) y (7), donde k_j es la exactitud de la regla j, ε_0 es el mínimo error permitido (limita k_j a un valor finito como indica [Wilson, 1997]), k'_j es la exactitud relativa, α la tasa de ajuste de la aptitud, F_j la aptitud actual y n la penalización del error en la aptitud.

$$k_j = \begin{cases} \varepsilon_j^{-n} & \text{si } \varepsilon_j > \varepsilon_0 \\ \varepsilon_0^{-n} & \text{si } \varepsilon_j \leq \varepsilon_0 \end{cases} \tag{5}$$

$$k'_j = \frac{k_j}{\sum_{i \in [A]} k_i} \tag{6}$$

$$F_j \leftarrow F_j + \alpha(k'_j - F_j) \tag{7}$$

Por otra parte, es necesario asegurar una búsqueda amplia de soluciones a los inputs dados por el ambiente. Es decir, no siempre se debe escoger la acción con la mejor predicción, sino que en algunos casos se debería escoger acciones sub-óptimas para hacer una mayor exploración. Es por esta razón que al momento de hacer la elección de la acción en el arreglo de predicciones, es importante seleccionar una regla según una probabilidad dada; esta probabilidad pudiera ser proporcional a la predicción que se tenga.

Primeras reglas

Usualmente el entrenamiento de XCS se inicia con una población vacía. La manera en que XCS llena el conjunto $[P]$ es haciendo *covering*. Esto quiere decir que al momento de la llegada de una entrada que no tenga correspondencia en el conjunto $[P]$, se genera una regla

aleatoria capaz de hacerlo. A esta nueva regla se le asigna una predicción baja.

El mayor uso de esta técnica se da al principio de la ejecución del XCS. Sin embargo puede suceder que aunque el conjunto $[P]$ esté lleno, una entrada no coincida con ninguna regla. Si es así, se hace *covering* y se reemplaza alguna regla por la nueva generada mediante cualquier algoritmo de reemplazo conocido.

Derivación de nuevas reglas

Para derivar nuevas reglas, con una probabilidad dada se ejecuta una iteración de un algoritmo genético donde la población estará contenida en el conjunto $[A]$. Este AG sigue los siguientes pasos:

1. Escoge a dos padres de $[A]$ según un probabilidad p que depende de la aptitud del individuo.
2. Se hace una réplica de estos padres para formar h_1 y h_2.
3. Se aplica un algoritmo de cruce para h_1 y h_2.
4. Luego los individuos resultantes se mutan con una probabilidad pequeña.
5. Y por último se incorporan al conjunto $[P]$ utilizando un algoritmo de reemplazo.

Dado que la cardinalidad del conjunto $[P]$ no debe sobrepasar un máximo, el último paso del AG da pie a la eliminación de dos individuos en el conjunto. Este problema da la oportunidad de mantener al XCS balanceado en cuanto a acciones se refiere. Por ejemplo podemos aumentar las probabilidades de eliminar un individuo cuya acción se encuentre muy repetida. Sin embargo es válido el uso de cualquier algoritmo conocido para AG que permita la sustitución de individuos de una población.

2.6. Arquitectura de Subsunción

La arquitectura de subsunción, propuesta por Rodney Brooks [Brooks, 1991], surge como una respuesta alternativa para diseñar sistemas de control para robot móviles, inteligentes y autónomos. Está compuesta de una serie de módulos asíncronos (comportamientos)

ordenados en capas, que realizan tareas relativamente sencillas. Existe un mecanismo de comunicación entre los distintos comportamientos, de manera que aquellos módulos que se encuentran en capas superiores subsuman la acción de las capas inferiores suprimiendo su salida, logrando así, que de la interacción de comportamientos sencillos surja un robot capaz de realizar tareas complejas.

Estos sistemas de control deben cumplir con una serie de requerimientos esenciales entre los cuales podemos mencionar los siguientes:

- **Múltiples Objetivos:** Un robot, usualmente, debe cumplir con varias tareas de forma simultánea, algunas de éstas, conflictivas entre sí. Por ejemplo, trasladarse en espacio no estructurado y evitar obstáculos.

  La prioridad de un objetivo con respecto al otro muchas veces es relativa al ambiente en donde se encuentre el robot. Si un robot se encuentra examinando el estado de una ruta ferroviaria y siente que se avecina un tren, el sistema debe dar más importancia a esquivar al tren que a revisar las vías, así como también, debe mantener funciones básicas como el equilibrio del robot para evitar que éste pierda el control

- **Múltiples Sensores:** Es relevante considerar que un robot puede tener sensores de diversos tipos como infrarojos, cámaras y sonares. De igual manera es conveniente tomar en cuenta que los sensores poseen errores y pueden generar lecturas inconsistentes que afectan la manera en como el robot percibe el mundo real

- **Robustez:** Si algún sensor falla o se produce un cambio drástico en el ambiente, el sistema de control del robot debe ser capaz de adaptarse y producir comportamientos coherentes que concuerden con el mundo real

- **Aditividad:** Mientras más sensores y funcionalidades tenga un sistema de control, es necesario más poder de cómputo para evitar que las funcionalidades iniciales se vean afectadas en su ejecución

2.7. Algoritmos de Búsqueda

La búsqueda de soluciones a un problema es un área conocida de la inteligencia artificial, donde se utilizan algoritmos para determinar la secuencia de acciones que conducen a un estado deseable.

Para estudiar y aplicar algoritmos de búsqueda es necesario formular problemas bien definidos. Según Russell y Norvig [Rusell y Norvig, 2003], un problema está definido por los siguientes elementos:

- Un conjunto de estados S
- Un estado inicial $s_0 \in S$
- Un conjunto de acciones A que se pueden realizar
- Una función de sucesores $F : S \rightarrow A \times S$ que para un estado $s \in S$ retorna un conjunto de pares acciones-estados, que representan los estados alcanzables al aplicar cada acción. Con esta función se puede definir el espacio de estados, un grafo cuyos nodos son todos los estados alcanzables desde el estado inicial y con arcos entre estados representando las acciones permitidas en cada uno de los estados
- Un conjunto objetivo $G \subseteq S$ que define los estados que se desean alcanzar
- Una función que determine el costo de aplicar una acción $a \in A$ en un estado $s \in S$, generalmente denotada $c(s_1, a, s_2)$ y significa "el costo de aplicar la acción a en el estado s_1 y obtener el estado s_2"

De esta misma manera, se puede definir una solución como un camino desde el estado inicial s_0 hasta un estado objetivo $g \in G$, entendiendo que un camino es una secuencia de estados conectadas por una secuencia de acciones, como lo definen Russell y Norvig [Rusell y Norvig, 2003].

Todos los caminos tienen un costo asociado, que puede ser calculado con la función de costo del problema, éste permite comparar los caminos y por lo tanto las soluciones. Una solución óptima es aquella que tiene el menor costo de todas las soluciones posibles.

Los algoritmos que resuelven los problemas de búsqueda pueden ser clasificados según la información disponible del problema. Los algoritmos de búsqueda no informada resuelven problemas que no presentan información adicional a su definición. Cuando se cuenta con conocimiento específico del problema, se pueden utilizar algoritmos de búsqueda informada.

2.7.1. Algoritmos de Búsqueda Informada

Best-First-Search es un algoritmo que realiza una búsqueda en profundidad sobre el grafo del espacio de estados escogiendo los nodos a expandir con una función de evaluación y seleccionando al que tiene el menor valor. Esta función de evaluación $f(n)$ puede depender de las características de n, la descripción de los estados objetivo, la información obtenida en la búsqueda y cualquier información adicional del problema [Pearl, 1984]; en este caso se llama función heurística. Una versión simple basada en caminos de Best-First-Search se expone en el algoritmo 2.7.1.

Es importante notar que a pesar de que el nombre Best-First-Search pareciera garantizar la mejor solución, este algoritmo no asegura que la solución encontrada sea óptima. Esto dependerá de la exactitud de la estimación de la función de evaluación.

Si se puede encontrar una heurística admisible, es posible usar algoritmos como A* o alguna de sus variantes. Una heurística $h(n)$ es admisible si nunca sobreestima el costo para llegar al objetivo [Rusell y Norvig, 2003]. En A* la función de evaluación para escoger al mejor nodo es $f(n) = g(n) + h(n)$, donde $g(n)$ es el costo real para llegar al nodo n desde el nodo inicial. A* y la familia de algoritmos derivados de éste si garantizan una solución óptima.

Algoritmo 2.7.1: BEST-FIRST-SEARCH(s_0: estado inicial)

```
inicial ← (s_0)
abiertos ← COLA-PRIORIDAD-VACIA()
INSERTAR(abiertos, inicial)
while not ES-VACIO(abiertos)
    camino ← REMOVER-PRIMERO(abiertos)
    if OBJETIVO(camino)
       return camino
    estado ← ULTIMO-NODO(camino)
    for each (accion, sucesor) ∈ F(estado)
        expansion ← EXPANDIR(camino, sucesor)
        expansion.costo ← f(sucesor)
        INSERTAR(abiertos, expansion)
```

2.8. Estructura de datos: KDtree

Es una estructura de datos que permite almacenar un conjunto de puntos de k-dimensiones en un árbol binario. Gracias a su estructura, permite realizar la busqueda de los n puntos más cercanos a punto con un reducido número de operaciones. Para más detalles ver apéndice D.

2.9. Soluciones previas: equipos exitosos

Para aprender técnicas y acercamientos al problema de Robocup Rescue Simulation System se estudiaron los equipos que han participado en la competencia en los últimos tres encuentros, especialmente los equipos que obtuvieron los mejores puestos y que publicaron sus hallazgos. Estos son: *ResQ Freiburg* [Kleiner et al., 2004], *DAMAS Team* [Paquet et al., 2004] y *5Rings* [Silva y Coelho, 2004].

2.9.1. *ResQ Freiburg*

Este equipo resultó ganador del premio al primer lugar en la competencia de Robocup Rescue Simulation en el año 2004.

Los agentes del equipo ResQ Freiburg [Kleiner et al., 2004] mantienen un modelo probabilístico del mundo, el cual indica la probabilidad de conseguir un civil en cada edificio de la ciudad. Estos valores se actualizan con la información sensorial del agente y la búsqueda de civiles se inclina hacia objetos con mayores probabilidades.

Este equipo representa las acciones que puede realizar en el mundo como una *Tarea*. Cada *Tarea* posee precondiciones, acciones y un valor de prioridad que puede cambiar según las condiciones del ambiente.

Para la búsqueda y planificación de caminos, los agentes tienen un grafo más pequeño del modelo del mundo, manejando un concepto de calles que contienen muchos nodos y otras calles llamado *LongRoad*. El concepto de un *LongRoad* es simple: debido a que muchos nodos en el espacio conectan únicamente dos calles, se puede construir una "super calle" que agrupe todos los nodos y calles que son contiguos y comparten esta propiedad.

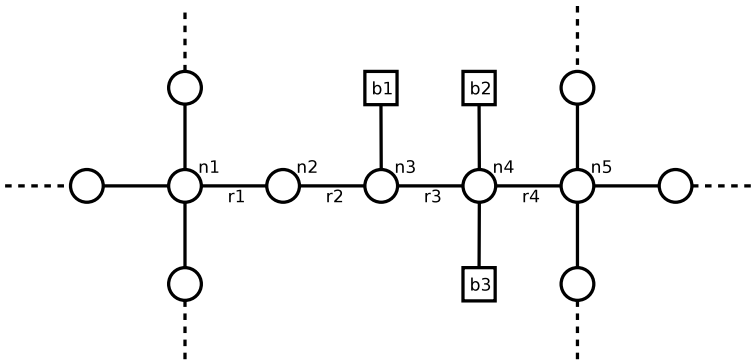


Figura 6: Grafo de nodos, calles y edificios del mundo, basado en [Kleiner et al., 2004], los cuadros son edificios, los círculos representan nodos y las conexiones entre nodos son las calles

Con la figura 6 se puede explicar mejor el concepto de *LongRoads*. El hecho de que los edificios $b1, b2$ y $b3$ estén conectados a algún nodo causa que el grafo tenga múltiples nodos que solamente tienen dos calles vecinas. En este caso, los nodos $n2, n3$ y $n4$ presentan esta característica. Entonces se puede construir un *LongRoad* desde los nodos $n1$ y $n5$ porque

éstos sí tienen múltiples calles que los conectan. El *LongRoad* contendrá los nodos $n2, n3, n4$ y las calles $r1, r2, r3, r4$.

El rescate de las víctimas es abordado con dos técnicas: una que predice el tiempo de vida de los civiles a partir de la información de un número importante de simulaciones; la segunda utiliza un algoritmo genético (véase apéndice A) para escoger el orden de rescate.

Para extinguir incendios se determina la vecindad de los edificios en llamas utilizando estructuras como el árbol *kdtree*, explicado en el apéndice D. La selección del edificio a extinguir depende de una función pesada que toma en cuenta el costo de moverse al edificio, número de edificios no quemados en la vecindad y el número de civiles en peligro.

La comunicación de los agentes ResQ permite enviar múltiples datos en un solo mensaje, dividiendo el mensaje en pequeños elementos llamados *Tokens* y retransmitiendo los mensajes importantes para asegurar la llegada al destinatario.

2.9.2. *DAMAS Team*

DAMAS Team [Paquet et al., 2004] es el equipo ganador del segundo lugar en el año 2004 en el *RoboCup Rescue Simulation League*. Para el desarrollo de sus agentes tienen soluciones sencillas a cada uno de los problemas, generalmente con una arquitectura centralizada.

En el caso de las Ambulancias, el AmbulanceCenter es el encargado de decidir el orden de rescate de cada uno de los agentes tapiados. La idea es evaluar cuál agente rescatar para maximizar el número de rescates posteriores. Esto se logra estimando el número de turnos que se necesita para salvar a cada uno de los agentes. Luego se ejecuta el algoritmo 2.9.1 donde *Agentes* es el conjunto de agentes por rescatar, *primerAgente* es el primer agente a rescatar, *cpt* es el número de posibles agentes a rescatar luego de salvar a x, *maxCpt* es el máximo número de agentes a rescatar luego de salvar a *primerAgente*, *tiempo* es el tiempo de rescate para el agente y *limiteVida* es el momento en la simulación de la muerte de un agente.

Algoritmo 2.9.1: ENCONTRAR-MEJOR-AGENTE(*Agentes*)

```
primerAgente ← null
maxCpt ← 0
for each x ∈ Agentes
    cpt ← 0
    for each y ∈ Agentes − {x}
        if x.tiempo + y.tiempo + CURRENT_TIME ≤ y.limiteVida
            cpt ← cpt + 1
    if cpt > maxCpt
        maxCpt ← cpt
        primerAgente ← x

return primerAgente
```

Luego de seleccionar un agente para rescatar, todas las Ambulancias se dirigen al lugar, desentierran a la víctima y la ambulancia con menor identificador la transporta al refugio más cercano.

Por otra parte, los policías son los agentes menos centralizados del equipo. La única función del PoliceOffice es dividir el mapa en nueve partes iguales al principio de la simulación y asignar una a cada uno de los policías. Luego los policías en cada de una de sus regiones tienen el deber de: desbloquear cada uno de los caminos solicitados por las Ambulancias y Bomberos, desbloquear los caminos desde los incendios hasta los refugios, desbloquear los alrededores de un refugio y por último desbloquear toda su zona mientras explora en busca de víctimas.

Los Bomberos por su parte usan el modelo centralizado para llevar a cabo sus tareas. El *Fire Station* es el encargado de elegir cuantos bomberos se deben asignar por cada uno de los incendios. Esto se logra mediante un método de aprendizaje por reforzamiento conocido como *Selective Perception* [Mccallum, 1996]. Luego de que los bomberos llegan al incendio, ellos se encargan de seleccionar que edificio apagar. Esto lo hacen evaluando para

cada edificio en fuego una suma pesada del número de civiles en el edificio y el número de edificios sin incendiar que tienen a su alrededor. Para determinar la cantidad de bomberos que será enviada al edificio también es usado *Selective Perception* [Mccallum, 1996].

2.9.3. *5Rings*

El equipo *5Rings* [Silva y Coelho, 2004] comenzó su participación en la liga de *Robocup Rescue* en el año 2004 y propone una aproximación teórica muy interesante al problema de *Robocup Rescue* y al modelado de sistema multiagentes.

Representación de Agentes

5Rings utilia una representación de cada agente como una tupla de 7 elementos. Esta tupla se define de la siguiente manera:

$$ag = \langle G, M, Wm(E_{self}, E_{other}, Mr), P_{Ac}, H_{Ac}, E, C \rangle$$

- G**:** es el conjunto de metas que el agente planea cumplir, las cuales pueden ser: cableadas por el diseñador, metas de otros agentes y metas del usuario (si existe interacción).
- M**:** Modelo Teórico del mundo, el cual contiene un modelo en donde las acciones son consecuencias directas de eventos externos.
- W_m**:** Working Memory, en donde se almacena la representación de mapa del desastre de un agente.
- P_{Ac}**:** Acciones y protocolos primarios que el agente puede usar.
- H_{Ac}**:** Acciones y protocolos de alto nivel que el agente puede usar.
- E**:** Ambiente en donde se desenvuelve el agente.
- C**:** Es el canal de comunicación por el cual el agente puede interactuar, de aquí se desprenden 2 canales lógicos: C_{ag} mediante el cual el agente se comunica con otros agentes y C_{agE}, por el cual el agente se comunica con el ambiente.

Niveles de Razonamiento

La implementación de los agentes de este equipo se realizó usando un arquitectura llamada RRR en donde coexisten 3 niveles de razonamiento:

- **R0** Compuesto por una serie de reglas condición-acción las cuales están separadas por conveniencia en dos subconjuntos:

  - **I:** para interacción o comunicación con otros agentes.
  - **A:** para realizar acciones que afecten al mundo.

  Cada lista de acciones para cada tipo de agentes se encuentra ordenada de manera que la primera regla cuya condición sea satisfecha esa será la acción elegida.

  Cuando la condición para realizar una acción es satisfecha la intención de realizar esta acción es pasada a un filtro que permite que la capa inmediatamente superior evalúe si se realizará esta acción u otra.

- **R1** En este nivel se actualizan los edificios y calles visitadas con la finalidad de poder hacer búsquedas de víctimas y de vías bloqueadas (*beliefs*). Lo que *5Rings* planea implementar en esta capa es una variación de una arquitectura de agentes *Belief-Desire-Intention (BDI)* [Wooldridge, 2002].

  Luego de actualizar los *beliefs*, basado en esta información se actualizan los deseos(*desires*) del agente, cada deseo es una lista de argumentos que esta capa cree que son las más "interesantes" para cada acción. Esta lista se encuentra ordenada por prioridades.

  Luego de la actualización de los deseos, entra en juego el planificador, el cual mediante una función de utilidad evalúa los deseos para obtener la acción que se debe realizar en el próximo turno. Por lo tanto, el deseo de costo mínimo se convierte en la intención del agente.

  La intención es pasada a un monitor que se encarga de ver si la acción se cumplió o si se puede cumplir y fija un plazo para su cumplimiento. Este monitor es capaz de realizar reparaciones al plan en caso de no poder llevarse a cabo.

Adicionalmente, en el nivel R1 se encuentra un planificador de rutas que posee una función de costos para caminos que toma en cuenta lo siguiente: vías despejadas, vías cortas y vías anchas. Además, el planificador de rutas tiene una lista de *must go* y otra de *taboo*. La lista de *must go* indica que la ruta debe pasar al menos por un elemento de la misma y la de *taboo* nos dice que la ruta no puede pasar por ningún nodo contenido en ella.

Por lo tanto la tarea de buscar un camino es subdividida en dos partes: encontrar un camino del origen a un elemento de la lista *must go* y luego encontrar un camino de este lugar a la meta deseada.

El plan de acción de cada agente es revisado en cada ciclo de la simulación y se calcula la desviación con respecto al plan original. Si la desviación excede cierto umbral entonces se realizan cambios para poder llevarlo a cabo y evitar problemas como *sub-goal frustation* [Murphy, 2000].

Por último, luego de un número determinado de reparaciones al plan, o de su culminación, se dice al planificador que seleccione una nueva intención.

- **R2**

  En este nivel se divide el mapa en una cuadrilla y se asigna a un agente de cada tipo a una celda de ésta. Se deja fuera uno o más agentes a los cuales se les asigna todo el espacio de desastre como área de acción.

  Para cada celda de la cuadrilla se define una distancia con respecto al centro y de no existir nodos de interés dentro de esa distancia, los agentes asignados a ese cuadro se asignan al mapa completo.

Capítulo 3

Diseño

Como se explicó en el capítulo 2 el problema de *Robocup Rescue* es muy complejo. Debido a ello se descompone en varios sub-problemas. Este capítulo presenta el diseño de cada una de las soluciones a los sub-problemas. A continuación se presentan el diseño de las soluciones a cada uno de estos problemas:

3.1. Problemas Identificados y Soluciones

En el comienzo de esta investigación se presentó un problema de diseño fundamental para el desarrollo de las estrategias de ataque y las aproximaciones a la solución al problema de *Robocup Rescue* el cuál consistía en definir si la toma de decisiones debería ser centralizada o distribuida.

Tuqueque Team optó por tomar un enfoque híbrido, con énfasis en la centralizada. Los agentes pueden tomar algunas decisiones de poca relevancia por sí mismos, pero dependen de una instancia mayor o agente central para poder realizar sus tareas más importantes. El resto de los problemas encontrados por el equipo son descritos a continuación:

3.1.1. Búsqueda de Civiles

La búsqueda de civiles es una de las tareas más importantes que los agentes deben realizar, ya que el rescate de éstos constituye el objetivo más importante en cualquier situación de desastre natural. Esta actividad consiste en observar todos los lugares de la ciudad donde podría encontrarse algún civil. Como se explicó en la sección 2.3.2, los civiles intactos se dirigen a los refugios y los que están atrapados se quedan inmóviles dentro de los edificios; por esto la búsqueda de civiles se traduce en observar los edificios de la ciudad.

Como todos los agentes pelotón pueden realizar la tarea de búsqueda de civiles, es necesario la coordinación de la exploración de edificios para conseguir a las víctimas en el menor tiempo posible.

Además, debe existir un mecanismo de reporte de víctimas encontradas. Éstos reportes deben llegar al *Ambulance Center* el cual es el agente que cordina las labores de rescate.

Exploración coordinada

Para resolver el problema de realizar una exploración coordinada los agentes de **Tuqueque Team** tienen un comportamiento básico que realizan siempre a menos que exista alguna tarea de mayor importancia. Este comportamiento consiste en escoger el edificio más cercano al agente y moverse hasta algún punto desde el cual el agente pueda verlo para poder actualizar las propiedades del mismo.

Cada agente mantiene actualizada una lista de edificios que no ha explorado. De esta manera un agente evita explorar varias veces el mismo punto. Cada edificio explorado es comunicado mediante mensajes de radio, así la lista de edificios que se desean explorar se va complementando con la información de otros agentes y la tarea de búsqueda se agiliza.

Es importante mencionar que los agentes deben evitar entrar a un edificio en llamas. Por las limitaciones de visión explicadas en la sección 2.3.2 del capítulo 2, a veces es necesario entrar al edificio para poder observarlo. Tomando en cuenta esto, los agentes solamente entran a los edificios cuando es necesario y huyen de éste si esta en fuego.

Por último, los agentes mantienen informados a los compañeros que se encuentran en su rango de audición de los edificios explorados mediante mensajes de voz. Con ésta técnica se trata de evitar que varios agentes escojan el mismo edificio para explorar.

Comunicación de información de víctimas

En el instante que un agente visualiza una víctima atrapada en un edificio envía un mensaje de radio a un agente central con la información completa de la víctima (ver sección B.4). Este mensaje se debe tratar con una prioridad alta, según el protocolo que se explica con detalle en la sección 3.1.7. Las centrales reenvían este mensaje al agente coordinador de rescate, que en el caso de **Tuqueque Team** es el *Ambulance Center*.

3.1.2. Rescate de Civiles

Una vez obtenida la información sobre la posición y el estado de las víctimas es necesario generar una estrategia para su rescate, esta tarea requiere de coordinación y cooperación por parte de todo el grupo de ambulancias.

El primer paso para desarrollar esta estrategia es decidir el enfoque para la coordinación, en el caso de **Tuqueque Team** se hará de manera centralizada, es decir, el *Ambulance Center* estará encargado de decidir cuál es la próxima acción y qué agente o grupo de agentes será designado para realizarla.

Una vez elegido el tipo de estrategia a utilizar, es necesario resolver los siguientes problemas:

a.- Orden de rescate de víctimas

El *Ambulance Center* debe establecer un orden de prioridad para el rescate de las víctimas con la finalidad de minimizar la pérdida de vida por parte de los civiles.

La selección de la próxima víctima a rescatar se realiza utilizando una modificación al algoritmo 2.9.1 que es usado por *DAMAS Team*. La principal diferencia es que en la selección de la próxima víctima se asume que será rescatada por un solo agente, es decir, el tiempo necesario para rescatar a un agente es igual a su *buriedness*, mientras que el *limiteVida* (ver sección 2.9.2) es calculado de acuerdo a la siguiente fórmula:

$$limiteVida = \frac{HP}{damage} \tag{8}$$

b.- Número de ambulancias necesarios para rescatar a una víctima

Luego de seleccionar al próximo civil a rescatar el *Ambulance Center* debe decidir cuántas ambulancias debe enviar a realizar el trabajo. Esta decisión es importante ya que el enviar más ambulancias de las necesarias podría ser un desperdicio de recursos y enviar pocas podría causar la muerte de la víctima, vale decir que es posible que el *Ambulance Center* considere que es preferible dejar morir a la víctima, ya sea porque es imposible rescatarla antes de que muera o porque requiere de mucho esfuerzo.

Para tomar esta decisión se cuenta con la ayuda de un sistema de clasificadores genéticos XCS como los explicados en la sección 2.5. La estructura de este sistema y los parámetros del algoritmos genético serán explicados en la sección 3.4.1.

c.- Selección de ambulancias para el rescate de víctimas

Además de cuántas ambulancias enviar es necesario saber cuáles ambulancias son las más adecuadas para realizar el rescate del civil, **Tuqueque Team** envía las ambulancias cuyo costo para alcanzar a la víctima sea el menor. Utiliza la función de costos descrita en la sección 3.1.6.

En caso de que se cuente con menos ambulancias libres de las necesarias para rescatar a la víctima se envían a todas las disponibles y en la medida que vayan reportándose como libres son despachadas.

d.- Selección de la ambulancia encargada de llevar a la víctima al refugio

Luego de realizar el rescate la víctima debe ser llevada al refugio más cercano, pero como el rescate es coordinado y varias ambulancias se encuentran desenterrando a la víctima, es necesario decidir cual de ellas será la encargada de llevarla hasta el refugio.

Para realizar esta tarea se le otorga a la ambulancia más cercana (menor costo de alcance) el puesto de *coordinador* el cual le confiere la atribución de encargarse de la víctima luego de haber sido desenterrada. De haber dos ambulancias con el menor costo la decisión se toma de manera aleatoria.

3.1.3. Rescate de Agentes Pelotón

Es posible que mientras realiza sus tareas un agente pelotón quede sepultado en alguno de los edificios colapsados, por lo tanto, es necesario prever esta situación y llevar a cabo las acciones necesarias para su rescate. Es importante aclarar que los agentes pelotón que se encuentren atrapados tienen prioridad sobre las víctimas ya que éstos además de aportar al puntaje, cumplen distintas funciones de rescate.

La cantidad de ambulancias que son enviadas a rescatar a cada agente pelotón se obtuvo

de forma experimental, cada vez que se recibe una petición de rescate para un agente se envían 2 equipos de ambulancias a rescatarlos en el momento que éstos se reportan como libres. De haber más de dos equipos de ambulancias desocupados se envían las más cercanas. Si se recibe más de una petición de rescate de agente, las solicitudes son atendidas en orden de llegada.

3.1.4. Extinción de Incendios

La extinción de incendios es una tarea muy importante en el RCRSS pues evita la destrucción de los edificios de la ciudad y alarga la vida de los civiles atrapados. Para realizar esta labor es necesaria una cadena de acciones.

En primer lugar, se necesita conocer la posición de los incendios en el mapa. Para esto, todos los agentes de **Tuqueque Team** al tener en su rango de visión un edificio en fuego que no haya sido reportado envían un mensaje radial informando el estado del edificio. De esta manera se acelera la búsqueda de incendios y la respuesta por parte del cuerpo de bomberos.

De los incendios encontrados, se elije cuales apagar. Luego, la estación de bomberos selecciona cuales de los bomberos libres deben ir a cada incendio y envía un mensaje radial listando los identificadores de los bomberos seleccionados.

Los bomberos al recibir la orden de la central, realizan una serie de pasos: construyen su propio modelo del incendio, seleccionan el mejor edificio a apagar de la manera indicada en la sección 3.5.2 y por último se coordinan para no obstaculizarse el paso y permitir que todos puedan extinguir el edificio seleccionado[1]. Los últimos dos pasos se repiten continuamente hasta lograr apagar el incendio por completo.

a.- Selección de incendios y cantidad de bomberos

La selección de incendios y la cantidad de bomberos para apagarlos son determinados mediante un clasificador genético XCS (sección 2.5). Al XCS se le da como entrada la información de tres incendios y genera como salida el número de bomberos que deberá ser enviado a cada uno de los incendios representados con 3 enteros sin signo de 4 bits. Se puede

[1]La acción de extinguir se explica en el apéndice C

notar que el rango de valores es $[0, 16]$. Es decir, se tiene un máximo de 16 bomberos para atender un incendio o se puede ignorar el fuego enviando 0 bomberos.

Si se tiene en un momento determinado más de tres incendios sin atender, es necesario un método de selección de los mejores incendios para ser introducidos en el XCS y atenderlos. Los mejores incendios para atender son los que causen más daño cuando se propagan. Mientras más centrado se encuentre el incendio, más área tiene a su alrededor para destruir cuando se expanda. Es por esta razón que la estación de bomberos de **Tuqueque Team** hace la selección de los tres incendios más centrados en el mapa para introducirlos al XCS, determinar el número de bomberos necesarios para cada uno y atenderlos.

Cuando el XCS retorna el número de bomberos por cada incendio, se envían los bomberos libres con el menor costo para llegar desde su ubicación hasta su incendio correspondiente. Con esta acción se asegura que, si el incendio es alcanzable, los bomberos podrán llegar fácilmente al incendio y empezar su extinción lo más rápido posible.

El número de bomberos disponibles en el RCRSS es limitado, por lo tanto no siempre se puede cumplir al pie de la letra la acción enviada por el XCS. Hay dos posibles panoramas donde ocurre este problema:

- El número de bomberos solicitado por el XCS para un incendio excede el número total de bomberos existente en la simulación. En este caso, la estación de bomberos asume que el XCS envió el número máximo de bomberos disponibles en la simulación.

- El número de bomberos solicitado por el XCS excede la cantidad de bomberos libres. En este caso se ignora la orden del XCS y el incendio se queda inatendido hasta que se tenga la cantidad de bomberos requerida. Esto se hace para procurar el buen entrenamiento del XCS, pues si en este panorama se envían todos los bomberos libres y el incendio se extiende, el XCS puede ser mal recompensado cuando en realidad pudo haber tomado una buena decisión.

b.- Coordinación de los bomberos en el incendio

Uno de los principales problemas en la extinción de incendios es la coordinación de los bomberos para evitar obstaculizarse el paso. Dada la naturaleza de las calles en el RCRSS, hay un número máximo de agentes que se pueden colocar en cada una de ellas. En el momento que se topa el límite de agentes en una calle puede considerarse bloqueada, es decir, ningún otro agente puede pasar sobre ella. Esto es un grave problema en la extinción de edificios si todos los bomberos desean colocarse en una misma calle para extinguir, pues en el momento de topar el máximo de agentes de la calle algunos bomberos no podrán llegar a su objetivo y empezar su tarea. Para solucionar este problema, aprovechando la capacidad de los edificios de contener muchos agentes, los sitios seleccionados por los bomberos desde donde extinguir son otros edificios que no estén en fuego. Con esta acción los bomberos no solo evitan el problema ya mencionado, sino que además no obstaculizan las labores de otros agentes que deseen pasar por el sitio.

Por otra parte, para poder aprovechar todo el poder de los bomberos en un incendio lo ideal es que todos ataquen el mismo fuego. Esta es una tarea que los bomberos de **Tuqueque Team** hacen bastante bien, gracias a que cada uno de ellos tiene una visión del incendio muy parecida (pues se encuentran muy cerca) y que la decisión del mejor incendio a apagar (sección 3.5) está basada en esta información, en la mayoría de los casos atacan el mismo incendio.

c.- Recarga de agua de los bomberos

Para atacar el problema de recarga del tanque se ideó una estrategia que permite llenarlo de agua de manera oportuna. Esta estrategia se ejecuta en dos posibles escenarios:

- Un bombero se encuentra extinguiendo un edificio y el agua de su tanque se agota. En este caso el bombero se dirije al refugio más cercano y llena su tanque parcialmente, esto para evitar estar mucho tiempo en el refugio mientras el fuego se propaga y descontrola.

- Un bombero terminó de apagar un incendio. En este caso, y mientras no tenga otra tarea que realizar, el bombero se dirije al refugio mas cercano y llena su tanque por

completo.

3.1.5. Desbloqueo de Calles

Para resolver el problema de desbloqueo de calles es necesario que los policías seleccionen adecuadamente cuáles calles deben desbloquear. También es importante decidir cuántos agentes deben desbloquear una calle para, de ser posible, acelerar la tarea mediante el trabajo en equipo.

Aunque la tarea de desbloqueo resulte importante, no aporta ningún puntaje para la evaluación de un equipo. Se podría decir que limpiar las calles es una tarea de apoyo para cooperar con los bomberos y ambulancias. Tomando esto en cuenta, los policías deben ser capaces de atender peticiones de desbloqueo de calles específicas, áreas alrededor de algún punto y el conjunto de calles que se deben cruzar para trasladarse de un lugar a otro.

Finalmente es importante que los policías reporten a todos los agentes de la simulación los estados actualizados de las calles que desbloquean, de manera que cualquier agente este informado de nuevas rutas que son transitables, mejorando la precisión de su búsqueda de caminos y cálculo de costos.

a.- Selección de la calle que se debe desbloquear

Utilizar un enfoque ingenuo para seleccionar las calles como escoger aleatoriamente una calle y desbloquearla resulta inútil para los demás agentes. Por esto, **Tuqueque Team** resuelve el problema de selección de calles utilizando varias técnicas:

- Para aprovechar el subgrafo de *LongRoads* del mundo (véase optimización de búsqueda de la sección 3.1.6), los agentes *Police Force* no escogen una calle para desbloquear, escogen un *LongRoad* completo. Esto quiere decir que el objetivo del agente no es desbloquear una calle específicamente sino todas las calles del *LongRoad*. El agente mantiene una lista de *LongRoads* por desbloquear y escoge siempre el que está más cercano.

- Si el *Police Office* lo ordena, el policía mantendrá una lista de todos los *LongRoads*

más utilizados del mundo, esto se estima haciendo búsquedas aleatorias en el espacio del mundo y revisando cuáles son las calles más utilizadas para los caminos. El policía escoge el *LongRoad* más cercano para desbloquear.

- Si el objetivo del agente es desbloquear un área, construye una lista de los *LongRoads* cercanos al punto y escoge el *LongRoad* que no haya sido desbloqueado más cercano al agente.

- Solamente cuando todos los *LongRoads* del mundo han sido desbloqueados el policía escoge la calle más cercana que no se encuentre totalmente limpia.

b.- Número de policías necesarios para desbloquear una calle

Generalmente utilizar más de un agente para realizar una tarea es deseable ya que minimiza el tiempo de trabajo; sin embargo, para desbloquear una calle esta suposición es falsa. La cantidad de agentes que pueden ubicarse en una calle depende de los canales libres. A su vez, los canales libres dependen del nivel de bloqueo de la calle, así que éstas son de difícil acceso para más de un agente.

Para evitar un problema de coordinación al utilizar varios policías para desbloquear una calle, los policías de **Tuqueque Team** no trabajan en equipo para limpiar la misma calle.

c.- Desbloqueo de calles por petición

Para manejar ordenadamente las peticiones de desbloqueo, **Tuqueque Team** utiliza un enfoque centralizado. El *Police Office* está encargado de atender las necesidades de limpieza, por lo que mantiene una lista detallada de todas las calles que se deben limpiar.

Tuqueque Team identificó tres tipos de peticiones de desbloqueo:

- Desbloqueo de calle: indica que un agente necesita que una calle en específico sea limpiada, ya que su bloqueo hace que su objetivo sea inalcanzable.

- Desbloqueo de ruta o camino: indica que es necesario desbloquear alguna ruta desde un punto inicial hasta un punto final. Los agentes de **Tuqueque Team** hacen este

tipo de petición cuando necesitan llegar con urgencia a un objeto del mundo por lo que toda la ruta a éste debe ser desbloqueada.

- Desbloqueo de áreas: indica que se debe desbloquear todas las calles cercanas a un objeto del mundo. Este tipo de desbloqueo resulta útil para mantener ciertos edificios como los centros de los incendios o los refugios, alcanzables desde varios puntos.

d.- Actualización de la información sobre las calles

Para mantener informados a todos los agentes, los policías envían cada ciclo un radio-mensaje con los detalles de las calles observadas.

Dadas las limitaciones de comunicación del sistema expuestas en el cuadro 2 del capítulo 2, las centrales reenvían éstos mensajes cuando contienen información actualizada. Además, cuando se trata de los datos de calles totalmente desbloqueadas, se aumenta la prioridad del mensaje. De esta manera los agentes que no son policías pueden conseguir mejores caminos a sus objetivos.

3.1.6. Búsqueda y Planificación de Caminos

La búsqueda de caminos es una de las tareas fundamentales de los agentes. Consiste en construir una secuencia de calles, nodos y edificios, desde un punto inicial hasta otro punto de interés. Afortunadamente todos los objetos del mundo están interconectados, formando un grafo, por lo que se pueden aplicar algoritmos de búsqueda para resolver este problema.

El problema de búsqueda se puede definir de la siguiente manera utilizando la estructura explicada en la sección 2.7:

- El conjunto de estados S está compuesto por todos los nodos, calles y edificios del espacio de desastre.

- El estado inicial $s_0 \in S$ es la posición de partida del camino, generalmente la posición del agente.

- Cada acción perteneciente a A es un movimiento del agente desde un objeto hasta otro, siempre y cuando éstos estén directamente conectados.

- La función de sucesores F, que utiliza un estado como entrada, devuelve todos los objetos del mundo que están conectados a este estado.
- El conjunto objetivo G está formado por los objetos que el agente desea alcanzar.
- La función de costo expresa el número de ciclos que son necesarios para que un agente se mueva desde un objeto a otro.

La función de costo que se acaba de mencionar es de vital importancia y requiere un análisis detallado ya que no se puede determinar exactamente cuál es el costo de realizar una acción en un estado. Elementos que no son totalmente controlables por los agentes, como la simulación de tráfico, movimiento entre canales de una calle, disminución de velocidad por escombros en la vía; influyen directamente en el mínimo número de ciclos necesarios para atravesar cada objeto.

Como los costos no son determinables, se usa entonces una heurística para estimar el valor. Dicha estimación asume que los agentes se mueven a su máxima velocidad y sin interrupciones de otros agentes, civiles o escombros. Estas suposiciones relajan el problema, asegurando que la heurística no lo sobrestime, permitiendo el uso de un algoritmo de la familia de A*, como se explica en la sección 2.7.1.

Sin embargo, hay que considerar que es probable que el camino óptimo de un nodo a otro este conformado por calles que no han sido observadas o que pueden estar bloqueadas.

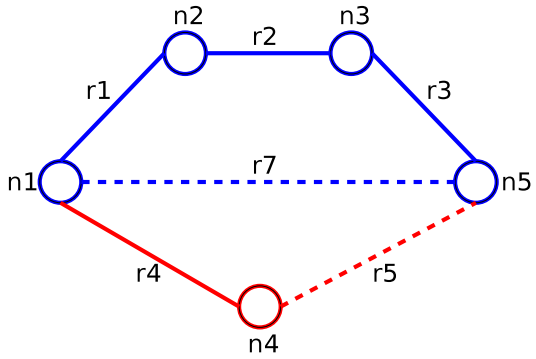


Figura 7: Ejemplo de un grafo del mundo. Los círculos representan nodos y las aristas que los conectan representan las calles. Los elementos en rojo no han sido observados, mientras que los azules si. Las líneas punteadas representan calles bloqueadas.

La figura 7 puede utilizarse para explicar el problema: los nodos y arcos azules representan nodos y calles del mundo que han sido observados, los rojos no han sido observados y los arcos de líneas punteadas son calles que están bloqueadas. Suponiendo que se desea buscar un camino desde el nodo $n1$ hasta el nodo $n5$, utilizar A* u otro algoritmo de búsqueda con la heurística mencionada anteriormente encontraría el camino $\langle n1, r7, n5\rangle$. Este camino no es aceptable para un agente ya que la calle $r7$ está bloqueada y no permite el paso. Si no se consideran a las calles intransitables en la búsqueda, entonces el camino encontrado sería $\langle n1, r4, n4, r5, n5\rangle$. Esta opción parece aceptable, pero este camino contiene calles que no han sido observadas y podrían estar bloqueadas. El camino que debe encontrar el algoritmo de búsqueda es $\langle n1, r1, n2, r2, n3, r3, n5\rangle$. En el ejemplo este es el camino más largo para llegar al destino, pero cada uno de los elementos que lo componen han sido observados y no están bloqueados. La solución podría no ser óptima, pero al menos asegura que el agente llegará a su destino.

Tomando esto en cuenta, el planificador de caminos de **Tuqueque Team** debe dar prioridad a las calles conocidas y libres de bloqueo, en segundo lugar a las calles que no se han observado y por último lugar a las calles bloqueadas. Esto se realiza estimando que las calles libres y observadas tienen un costo pequeño, las calles no exploradas con un costo alto y las bloqueadas con valor infinito.

Con este sistema de selección de calles, la heurística que se propone sobrestima el costo real, A* no puede ser utilizado, así que se escoge como algoritmo de búsqueda a Best-First-Search explicado en la sección 2.7.1.

Optimización a la búsqueda de caminos

Para optimizar la búsqueda de caminos, el planificador de caminos de **Tuqueque Team** aprovecha el concepto de *LongRoads* explicado en la sección 2.9.1. Utilizando un grafo más pequeño que agrupa nodos y calles, la búsqueda explora menos nodos y por lo tanto es más rápida.

Para **Tuqueque Team** la estructura de *LongRoad* se define como un par de nodos que representan el inicio y final del *LongRoad* y un conjunto de nodos y calles que conforman a

la estructura. Por ejemplo, en la figura 6 de la sección 2.9.1, los nodos n_1 y n_5 son el inicio y el final, mientras que el conjunto $\{n_1, n_2, n_3, n_4, n_5, \} \cup \{r_1, r_2, r_3, r_4\}$ son los nodos y las calles que agrupa el *LongRoad*. El costo de un *LongRoad* es el costo de la ruta desde el inicio de éste hasta su final, recorriendo todos sus nodos y calles.

La búsqueda de caminos utilizando *LongRoads* se puede resumir en tres pasos:

1. Buscar un camino de costo estimado mínimo desde el estado inicial s_0 hasta el nodo inicial o el nodo final del *LongRoad* que contiene a s_0. El estado final de éste camino será N_{lr1}.

2. Buscar un camino mínimo desde el estado objetivo g hasta el nodo inicial o final del *LongRoad* que lo contiene. El estado final de este camino será N_{lr2}.

3. Utilizando el grafo de *LongRoads*, buscar un camino mínimo entre los nodos N_{lr1} y N_{lr2}, la concatenación de los caminos $\textsc{Camino}(s_0, N_{lr1})$+$\textsc{Camino}(N_{lr1}, N_{lr2})$+$\textsc{Camino}(N_{lr2}, g)$ será la solución devuelta.

3.1.7. Comunicación entre Agentes

La cooperación entre agentes depende de que exista un buen mecanismo de comunicación entre éstos. Dadas las limitaciones que se presentan en la sección 2.3.2 del capítulo 2, se puede observar que existen dos problemas de comunicación: primero, el número de mensajes que se pueden enviar y recibir es reducido; segundo, el tamaño de los mensajes es relativamente corto.

Tuqueque Team ataca este problema con la creación de un protocolo para el uso y transmisión de mensajes durante la simulación. Éste define las reglas para dos aspectos importantes:

- el uso optimizado de todos los bytes disponibles en un mensaje, para enviar la mayor cantidad de información por mensaje.

- el control del número de mensajes que pueden transmitir los agentes, para minimizar el número de mensajes perdidos por transmitir más mensajes que el límite de escucha.

a.- *Tokens*

El protocolo de comunicación creado para comunicar a los agentes define de que manera se debe transmitir información aprovechando todo el tamaño de cada mensaje.

Se creó una estructura llamada *Token*, que representa la mínima unidad de información transmisible. Esta idea no es innovadora ya que es muy común en las implementaciones de los equipos actuales del proyecto RCRSS tal como la del equipo ResQ Freiburg [Kleiner et al., 2004]. Un *Token* es un paquete que contiene un encabezado que identifica su tipo, un número que indica el tamaño de los datos y una sección variable de datos.

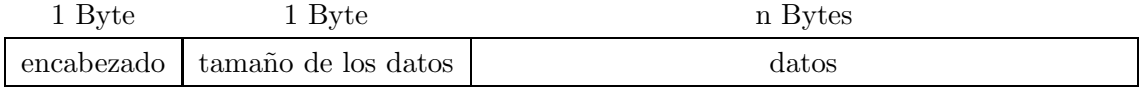


Figura 8: Estructura de un *Token*, el tamaño del *Token* es de $n + 2$ bytes

Los *Tokens* se concatenan como cadenas de bytes hasta alcanzar el tamaño máximo de un mensaje o agotarse su existencia, para luego enviar la cadena de bytes con radiomensajes o mediante la voz, como lo permite el API. Si es necesario se utilizan dos o más mensajes para enviar todos los *Tokens* en un turno.

Cada agente tiene entonces una lista de *Tokens* que desea enviar. Al final de su turno éste realiza la construcción del mensaje final y lo emite. Además se consideró que un nivel de prioridad debe relacionarse con cada *Token*, de manera que los elementos más importantes sean enviados primero y si no es posible enviar todos los *Tokens*, entonces se encolan para el próximo turno.

Existen muchos tipos de *Tokens*, identificados por su encabezado. Cada uno tiene un significado diferente, principalmente son órdenes de una central a un agente en especial o información de objetos del mundo observados. Todos los mensajes se explican en detalle en el apéndice E.

Por último, debido al diseño del simulador, es posible que algunos mensajes no lleguen a su destino por pérdida de paquetes UDP u otras razones externas a la simulación. Los elementos que se deseen transmitir que sean importantes, como las ordenes de las centrales,

deben utilizar un mecanismo de confirmación. Esto significa que para cada *Token* importante, debe existir otro para confirmar su recepción.

b.- Control de Transmisiones

El protocolo de comunicación de RCRSS indica exactamente el número de radiomensajes que los agentes pueden transmitir en cada turno. El propósito de este control es asegurar que los agentes centrales siempre puedan escuchar todos los mensajes de los demás agentes.

Las reglas que se deben considerar para transmitir son las siguientes:

1. Todos los agentes centrales pueden transmitir todos los mensajes que deseen, es decir pueden transmitir los 4 mensajes que les corresponden.

2. La cantidad de transmisiones por parte de los agentes pelotón no debe sobrepasar el límite de mensajes escuchables por parte de las centrales. Esto quiere decir que entre todos éstos no deben exceder $2n$ transmisiones (siendo n el número de agentes peloton del mismo tipo), como se indica en el cuadro 2 del capítulo 2. Sin embargo, tomando en cuenta la regla 1, cada central escuchará a lo sumo 8 mensajes de las demás centrales, así que el límite de transmisiones para los pelotones disminuye a $2n - 8$.

3. Por el límite de transmisiones definido en la regla 2, algunos agentes del mismo tipo tendrán oportunidad de transmitir 2 mensajes, mientras que el resto sólo podrá transmitir 1 mensaje. Debe existir entonces una ventana deslizante que indique cuáles agentes tienen el privilegio de transmitir más mensajes.

4. Ya que el límite de escucha de los agentes pelotón es de 4 mensajes y no es posible determinar cuáles mensajes escuchará el agente, es necesario asegurar que los agentes escuchen todos los mensajes de las centrales en algunas ocasiones. Se definió entonces un conjunto de turnos en el que ningún agente pelotón transmite mensajes, pero sus centrales sí lo hacen. En este turno de silencio, se asegura que los agentes recibirán toda la información que las centrales quieran enviar.

3.1.8. Coordinación y Cooperación

Para resolver problemas en equipo, se decidió que **Tuqueque Team** tendría un enfoque centralizado, donde los agentes centrales coordinan las actividades que le competen a cada grupo de agentes pelotón. Esto quiere decir que el *Ambulance Center* es el agente "maestro" para las actividades de rescate de víctimas, el *Fire Station* para la extinción de incendios y el *Police Office* para el desbloqueo de vías.

Sin embargo, existen otros problemas de coordinación que involucran a todos los agentes rescatistas del sistema. En primer lugar, como los agentes comparten el ambiente y utilizan los mismos elementos para moverse, se crea un evidente problema de tráfico. En segundo lugar, para realizar las tareas los agentes no poseen todas las habilidades necesarias para completarlo, así que deben cooperar con otros agentes que sí tienen las capacidades para ayudar a alcanzar el objetivo.

a.- Manejo de problemas de tráfico

El tráfico de agentes en el sistema podría paralizar a los agentes. Por esto es necesario que éstos detecten cuando se encuentran bloqueados debido al tráfico. Una vez identificado el problema, los agentes deben realizar alguna acción que resuelva el problema, o que al menos permita que otros agentes lo resuelvan.

El comportamiento intuitivo para evitar el bloqueo es evitar cualquier lugar donde se conoce que hay tráfico. Para lograr esto, se puede mantener una lista de elementos que deben evitarse a toda costa al planificar caminos. Los agentes de **Tuqueque Team** mantienen una "lista negra" o tabú, como lo hace 5Rings (ver 2.9.3). Los objetos en esta lista son la última elección del planificador y se mantienen en ella durante varios turnos, esperando que el problema de tráfico se resuelva en este lapso.

Si un agente no consigue una ruta alternativa, entonces se mueve a un edificio cercano para permitir el paso de otros agentes. Allí espera tres turnos y vuelve a intentar atravesar el camino bloqueado. Es importante que el agente haga una petición de desbloqueo por derrumbe de la calle que no puede atravesar para que los policías la limpien.

Si el bloqueo por tráfico se debe a una calle bloqueada, los policías podrían resolverlo al

limpiar la calle, siempre y cuando los demás agentes se mantengan fuera del área afectada.

De esta manera, evitando las calles bloqueadas y haciendo peticiones a los policías, se establece la cooperación para resolver los problemas de tráfico por calles bloqueadas.

b.- Cooperación entre agentes de distintos tipos

Debido a la complejidad del ambiente, es común que los agentes no puedan realizar sus tareas específicas por sí solos. En particular, las ambulancias no pueden realizar rescates si no hay manera de moverse hasta las víctimas y a los refugios. Los bomberos no pueden apagar un incendio si no pueden alcanzar una distancia mínima para extinguir cada edificio.

Tuqueque Team resuelve este problema permitiendo que los agentes hagan peticiones entre sí. Durante este estudio solamente se modeló un sistema de peticiones para el *Police Office*, que se explica en la sección 3.1.5. Las ambulancias deben anunciar a la central de policías que necesitan trasladarse de un lugar a otro para realizar un rescate. Los bomberos también anuncian sus rutas y las zonas donde existen incendios a extinguir. Estos casos son analizados por el *Police Office*, para enviar los agentes libres, cooperando directamente en la extinción de fuegos y rescate de civiles.

3.2. Arquitectura de Subsunción

El comportamiento de los agentes de **Tuqueque Team** fue modelado utilizando *Arquitectura de Subsunción*2.6. El esquema de subsunción de cada tipo de agentes es diferente al de los otros pero tienen los mismos niveles. Los detalles de éstos se encuentran en la sección 3.2.2.

3.2.1. Información Sensorial

En *Robocup Rescue Simulation System* la información sensorial es simulada mediante dos canales, en primer lugar, el canal auditivo, mediante el cual los agentes obtienen información proveniente de otros agentes y el canal visual de donde obtienen información del estado de los objetos que se encuentran en el ambiente. La información auditiva es obtenida en cada turno mediante los comandos *hearTell* y *hearSay* (sección 2.3.2) que le permiten al agente

recibir todos los mensajes de radio y de voz que sus limitaciones le permitan.

3.2.2. Niveles de Comportamientos

La arquitectura de subsunción de **Tuqueque Team** cuenta con cuatro niveles de control que permitieron realizar un diseño ordenado y coherente de la conducta de cada uno de los agentes. Estos niveles así como sus comportamientos son explicados a continuación.

Nivel 0 Este nivel contiene los comportamientos más básicos de los agentes, los cuales estos coinciden con los comandos de *Robocup Rescue*(ver sección 2.3.2).

Nivel 1 Este nivel contiene comportamientos que los agentes pelotón realizan por defecto en caso de no tener ordenes de los agentes centrales. Estos consisten en la exploración de edificios en búsqueda de víctimas, y los reportes de los objetos del mundo vistos por el agente. Adicionalmente, los *Police Force* tienen un comportamiento que realizan al encontrarse sin ninguna actividad pendiente, el cual consiste en desbloquear todas las calles que no han sido exploradas y no desbloqueadas.

Nivel 2 En este nivel se encuentran los comportamientos que son recibidos como órdenes de los agentes centrales a los agentes pelotón; este nivel que garantiza la coordinación en la realización de las tareas de **Tuqueque Team**. Los comportamientos que pertenecen a este nivel comprenden las actividades principales de *Robocup Rescue* como el desbloqueo de calles, el rescate de víctimas y la extinción de incendios. Cada uno de estos serán explicados en detalle en la sección 3.2.3

Nivel 3 Este nivel contiene los comportamientos que tienen la mayor prioridad y garantizan que las actividades de un agente no interfieran con las de otro. En primer lugar, se encuentra un comportamiento que debe realizar en caso de acercarse el momento de su muerte; en segundo lugar, un comportamiento que resuelve problemas de tráfico entre los agentes pelotón.

3.2.3. Comportamientos de los Agentes

En esta sección se describen los comportamientos más importantes de los agentes de **Tuqueque Team**. Los diagramas de interacción de la arquitectura de subsunción de los agentes *Ambulance Team*, *Police Force* y *Fire Brigade* se encuentra en las figuras 9, 10 y 11.

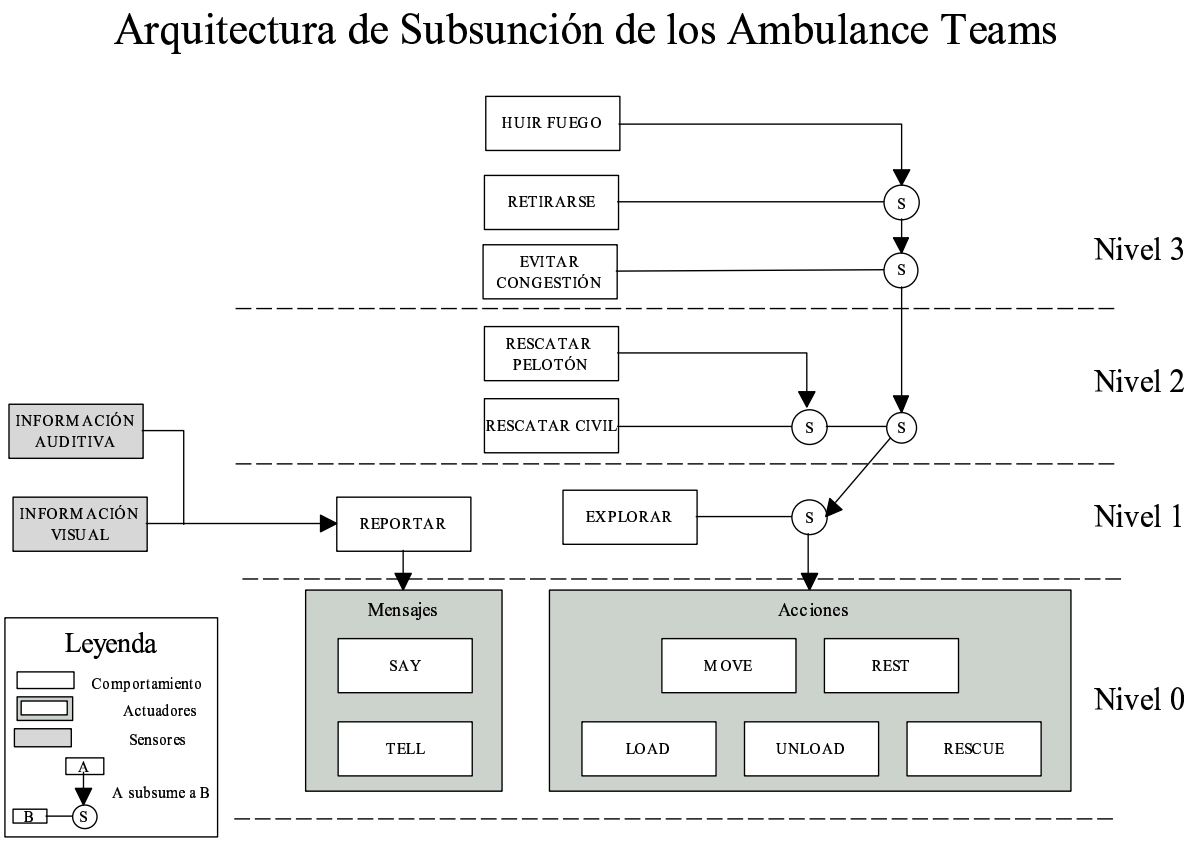


Figura 9: Diagrama de Arquitectura de Subsunción de los *Ambulance Team*

a.- Comportamientos generales

Los agentes pelotón comparten algunos comportamientos para asegurar la coordinación y cooperación:

Explorar: Consiste en realizar una exploración de edificios en búsqueda de civiles como se explica en detalle en la sección 3.1.1.

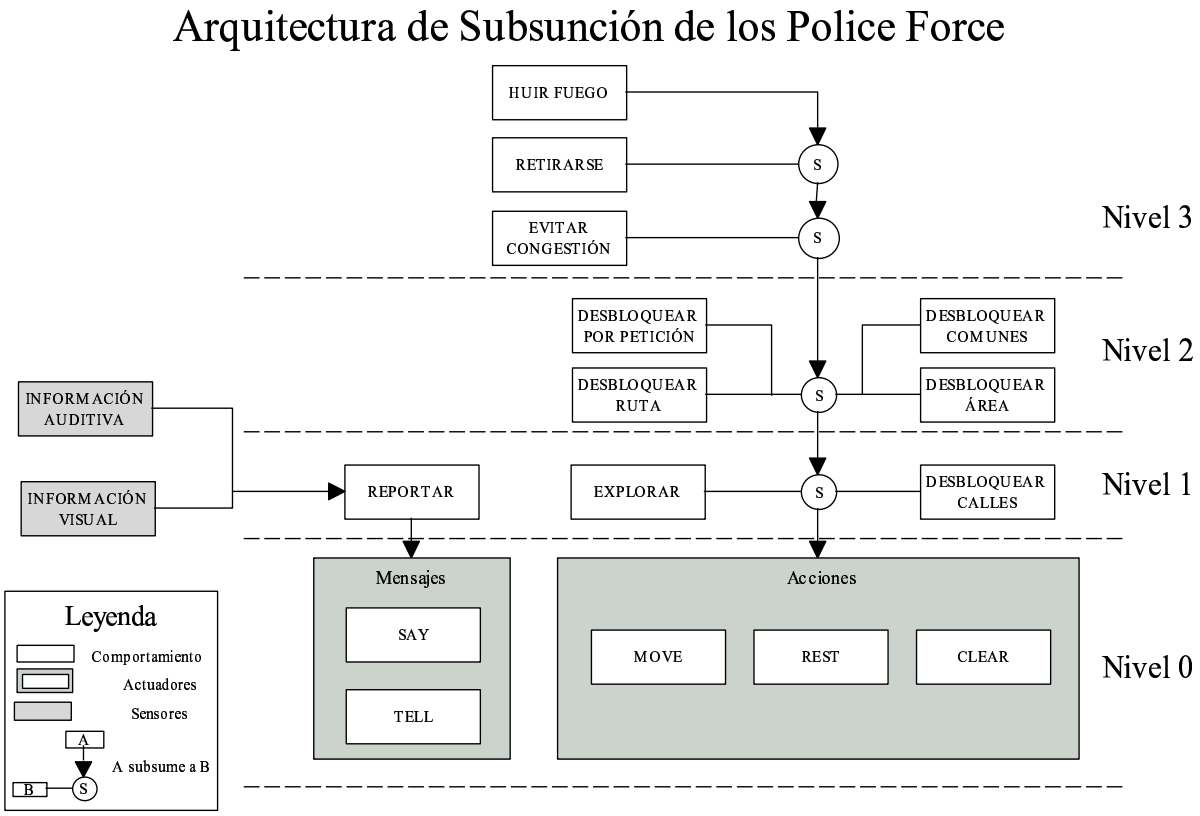


Figura 10: Diagrama de Arquitectura de Subsunción de los *Police Force*

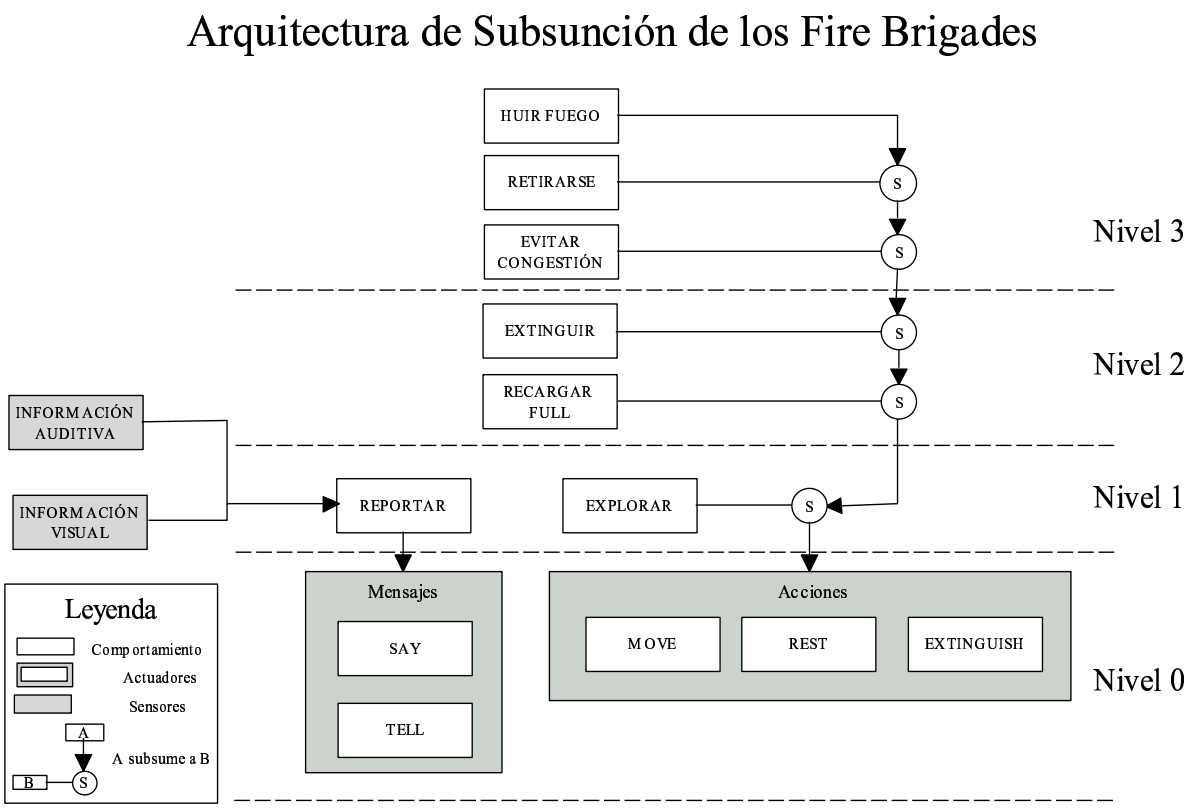


Figura 11: Diagrama de Arquitectura de Subsunción de los *Fire Brigade*

Reportar: El agente reporta todos los elementos del mundo que observa en cada turno utilizando mensajes de radio y voz.

EvitarCongestión: El agente se mueve a un edificio cercano cuando detecta congestión en la calle donde está parado, permitiendo el paso de otros agentes.

Retirarse: Cuando el agente detecta que sus puntos de vida llegarán a cero se mueve a un edificio para morir allí y no entorpecer el paso.

HuirFuego: El agente huye de los edificios que están en llamas.

b.- Comportamientos del agente *Fire Brigade*

Los *Fire Brigade* tienen dos comportamientos que permiten la extinción de los incendios. Estos comportamientos se activan como respuesta a las ordenes del *Fire Station*. Cada comportamiento se describe a continuación:

Extinguir: Aquí el agente busca el mejor edificio para extinguir y con la acción *move* se dirige a cualquiera de los edificios desde donde se puede apagar. Luego con la acción de *extiguish* empieza la extinción del edificio. En caso de vaciarse el tanque de agua del agente, se dirige al refugio más cercano con *move* y al llegar realiza la acción *rest* tantas veces como sea necesario para obtener el nivel de agua en el tanque deseado.

RecargarFull: En este caso el agente se dirige con *move* al refugio más cercano y ejecuta *rest* hasta que el tanque este completamente lleno.

c.- Comportamientos del agente *Ambulance Team*

Los *Ambulance Team* tienen dos comportamientos para el rescate de víctimas, estos comportamientos se activan como respuesta a una orden del agente central de ambulancias. Estos comportamientos se describen a continuación:

RescatarCiviles: El agente se dirige a la víctima mediante una acción *move*, cuando se encuentra en el lugar realiza la acción *rescue* hasta que el *buriedness* del la víctima sea 0. En caso de ser el coordinador, el agente hace la acción *load*, se mueve hasta el refugio más cercano mediante una acción *move* y cuando llega hace la acción *unload* para dejar a la víctima en el refugio.

RescatarAgentesPelotón: Se realizan las mismas acciones que en **RescatarCiviles** pero se omite totalmente la parte del coordinador ya que al ser desenterrado un agente pelotón no necesita ser llevado a un refugio.

d.- Comportamientos del agente *Police Force*

Los *Police Office* tienen cuatro comportamientos de desbloqueo y un comportamiento alternativo a la exploración de edificios.

DesbloquearCalles: El agente explora todas las calles que no ha visto y que tienen un nivel de bloqueo mayor a cero, moviéndose con la acción *move*. En estas calles realiza la acción *clear* hasta eliminar el bloqueo.

DesbloquearPorPetición: El agente se mueve mediante la acción *move* hasta alcanzar la calle que se pidió desbloquear. Al llegar hace *clear* hasta que exista al menos un canal libre.

DesbloquearRuta: El policía se mueve con la acción *move* hasta el inicio de la ruta que se pide limpiar. Después se mueve hasta el punto final de la ruta. Durante estos movimientos el agente limpia las calles hasta que sean transitables.

DesbloquearComunes: El agente determina cuáles son los *LongRoads* más utilizados haciendo varias consultas aleatorias al planificador de caminos. A continuación éste se mueve hasta el *LongRoad* más cercano y desbloquea cada calle de éste.

DesbloquearÁrea: El agente calcula los *LongRoads* alrededor del objeto del mundo que se pide desbloquear. Luego el agente desbloquea los *LongRoads* con un procedimiento igual al de **DesbloquearComunes**.

3.2.4. Agentes centrales

Los agentes centrales se encargan de coordinar a los agentes pelotón con la finalidad de optimizar las acciones realizadas por éstos. Los agentes pelotón son sumisos a las órdenes de los agentes centrales y abandonan cualquier tarea que no sea prioritaria para cumplir con las órdenes asignadas.

a.- *Fire Station*

La estación de bomberos asigna cuál edificio apagar a cada uno de los bomberos. Para esto mantiene una lista de bomberos libres. La lista se modifica sólo cuando se envía una petición de extinción a un bombero, en cuyo caso se elimina de la lista o se recibe un mensaje de libre con lo cual se incluye al emisor en la lista.

En el caso del *Fire Station* no es necesaria una asignación de prioridades en las tareas, pues tiene una sola.

b.- *Ambulance Center*

El centro de ambulancias se encarga de asignar rescatistas para la misión de salvamento de una víctima, en cada turno verifica si se encuentran ambulancias libres y las despacha para realizar las tareas de rescate de víctimas.

El orden de prioridades para asignar una tarea es el siguiente: si una tarea en curso necesita más agentes de los que ya le fueron asignados se envían las ambulancias necesarias a culminar la misma. En segundo lugar, verifica si hay agentes pelotón que no hayan sido rescatados y en caso de haber alguno se envía parte del personal a su rescate. Por último, si hay civiles en la cola de rescate entonces se pregunta al XCS cuantas ambulancias se deben enviar y éstas son despachadas a realizar la tarea.

c.- *Police Office*

La central de policías se encarga de recibir y atender las peticiones de desbloqueo de calles, rutas y áreas. Determina cuáles policías están libres y les envía ordenes con radiomensajes.

Las peticiones son atendidas con el siguiente orden de prioridad: primero las de desbloqueo de rutas, luego las peticiones de desbloqueo de áreas y por último las peticiones de desbloqueo de calles.

3.3. Sintetización de Valores

En un sistema de clasificadores genéticos XCS es importante dar como entrada sólo la información necesaria. Si se tiene una entrada con un rango de valores muy grande es probable

que la evolución sea en extremo lenta o incluso nunca se logre el aprendizaje deseado. Por ejemplo, si se tiene como entrada un valor que se encuentra en el rango $R = [0, 1000000]$ serían necesarios 20 bits para la representación de cada uno de los enteros. Para un mejor aprovechamiento del XCS, este número se puede resumir haciendo una correspondencia con un intervalo más pequeño como lo es $S = [0, 31]$ que solo ocupa 5 bits. Para lograr esto se divide a R en 32 sub-intervalos y se le asigna a cada uno un valor del rango S. Esto se puede generalizar siguiendo la función f en la ecuación (9) donde n representa la cantidad de sub-intervalos en que se divide R, v es el máximo valor del rango R y $j = \frac{v}{n}$

$$f_{n,v} : Rs \longrightarrow S \tag{9}$$

$$Rs = \{\ [a, b) \mid a = j \cdot i \wedge\ b = j \cdot (i + 1)\ \wedge\ 0 \leq i < n - 1\ \}\ \cup\ \{\ [j \cdot (n - 1), v]\ \} \tag{10}$$

$$S = \{\ x \mid 0 \leq x < n\ \} \tag{11}$$

La función (9) tiene como dominio un conjunto Rs y como rango conjunto de enteros S. El conjunto Rs consta de $n - 1$ intervalos del mismo tamaño y un intervalo diferente a los demas. Todos los intervalos del conjunto Rs son sub-intervalos del rango $[0, v]$. Por su parte el conjunto S está compuesto por n enteros. La función f hace una correspondencia uno a uno de los elementos del conjunto Rs con los elementos del conjunto S. Es importante destacar una serie de factores de esta función:

- Cada una de las variables planteadas son de tipo entero.
- El valor de n debe ser preferiblemente $2^i - 1$ para todo $i \geq 0$. Así se aprovecha el máximo de números representables por i bits.
- El último elemento del conjunto Rs debe ser separado para tomar en cuenta los casos en el que (v mód $n \neq 0$).
- Esta función solo es valida para sintetizar rangos cuyos valores sean mayores que cero.

Parámetro	Valor	Fuente
Probabilidad de ejecutar el AG en una iteración	0.2	experimental
Tasa de actualización en el reforzamiento (α)	0.1	experimental
Mínimo error permitido (ε_0)	0.5	experimental
Penalización del error (n)	5	[Wilson, 1997]
Covering		
Probabilidad de cambiar un bit a *don't care*	0.2	experimental
Predicción para la nueva regla	0	experimental
Error de la nueva regla	100	experimental
Aptitud de la nueva regla	0	experimental
Máximo tamaño de la población ($\|[P]\|$)	100	experimental

Cuadro 3: Parámetros del XCS para ambos centros

3.4. Parámetros de los Clasificadores Genéticos

Para la ejecución y entrenamiento del XCS usado por los agentes *Ambulance Center* y *Fire Station* es necesario establecer una serie de parámetros, como se indicó en la sección 2.5. Algunos de los valores establecidos son experimentales, otros son recomendados por autores del área. Los valores determinados se describen en el cuadro 3.

3.4.1. Parámetros del Algoritmo Genético Utilizado por el XCS

Los sistemas de clasificadores XCS requieren el uso de un AG para el descubrimiento de nuevas reglas como lo indica la sección 2.5. Para cualquier AG es necesario establecer una serie de parámetros y algoritmos[2]. Los algoritmos y valores usados para el AG se describen en el cuadro 4.

3.4.2. Parámetros del Clasificador de Selección de Incendio

La información de los incendios que usa el XCS de la estación de bomberos es la siguiente:

- **Edificios sanos:** Cantidad de edificios sanos que se encuentran alrededor del incendio

[2]Ver apéndice A y para más detalle [Mitchell, 1997]

Parámetro	**Algoritmo/Valor**	**Cómo se determina**
Algoritmo de reemplazo	Elitista	experimentalmente
Algoritmo de selección	k-Torneo con $k = 8$	experimentalmente
Algoritmo de cruce	Cruce de un punto	experimentalmente
Algoritmo de mutación	Mutación de un punto	experimentalmente
Probabilidad de mutación de un cromosoma	0.02	[Wilson, 1997]

Cuadro 4: Parámetros del Algoritmo Genético del XCS

- **Área del incendio:** Suma del área de los edificios en fuego, apagados y completamente consumidos por las llamas
- **Distancia al centro del mapa:** La distancia que se tiene desde el centro del incendio al centro del mapa
- **Costo:** Costo de la ruta desde el edificio principal del incendio a el refugio más cercano

La información de los incendios introducidos al XCS es traducida a una cadena de bits de tamaño 20 como se ejemplifica en la figura 12. Es importante destacar que todos los elementos del incendio son transformados como se explica en la sección 3.3.

Área					**# Edificios sanos**					**Distancia centro**					**Costo al refugio**				
1	1	0	1	0	1	1	0	0	0	1	0	1	0	0	0	1	0	1	0

Figura 12: Cadena de bits de la información de un incendio

Las cadenas de bits generadas de los tres incendios son unidas para formar la entrada del XCS de tamaño 60. Es importante destacar que si hay menos de tres incendios la entrada es completada con ceros en los bits faltantes.

Por su parte, los clasificadores del XCS constan de 60 bits de entrada y 12 bits de salida. Los 12 bits de salida se dividen en 3 partes iguales donde cada una indica el número de bomberos que se debe enviar a cada incendio. La correspondencia entre los bits de salida y los incendios viene dada por la posición en la cadena de bits, es decir, los primeros 4 bits de la salida corresponden a el incendio de los primeros 20 bits de entrada y así con el resto de los incendios.

3.4.3. Parámetros del Clasificador de Rescate de Víctimas

El sistema de clasificadores XCS usado por *Ambulance Center* toma en cuenta los siguientes parámetros: *Health Points*, *Damage*, *Buriedness* y *World Time*.

Cada clasificador tiene 21 bits de entrada ordenados de la siguiente manera:

HP/Damage								**Buriedness**							**WorldTime**						**Salida**		
0	1	1	1	0	0	1	0	0	1	1	0	0	0	1	0	1	0	0	0	1	1	0	1

Figura 13: Estructura del Cromosoma de los Clasificadores del XCS de Selección de Ambulancias

Los primeros 8 bits contienen la división de la *cantidad de vida* de la víctima entre su *damage*, lo cual da un aproximado de la cantidad de turnos que le restan por vivir a la víctima. Los bits del 9 al 15 contienen el *buriedness* de la víctima y los últimos 6 bits contienen el tiempo de la simulación de manera tal que el XCS sepa de cuanto tiempo dispone para salvar a la víctima, la traducción de entero al binario es explicada en detalle en la sección 3.3.

La salida está compuesta de 3 bits los cuales son interpretados como un entero sin signo que indica el número de ambulancias que serán enviados a la víctima a rescatar. Un ejemplo de un cromosoma se muestra en la figura 13.

3.5. Agrupamiento de Edificios en Fuego

Para la toma de decisiones de los agentes es necesario sintetizar la gran cantidad de información recibida a un nivel de abstracción que sea útil y sencillo de manejar. Este es el caso de los edificios que se encuentran en fuego en una ciudad. Generalmente lo que se necesita es obtener la información de un grupo completo de edificios incendiados y no de cada uno individualmente.

Inspirados en la vecindad de fuego de ResQ Freiburg (sección 2.9.1) y siguiendo la propuesta de [Llona, 2004] de sintetizar elementos espaciales agregados, **Tuqueque Team** desarrolló el agrupamiento de edificios en fuego (AEF). Los AEF cumplen la función de unir en un solo objeto los edificios en fuego cercanos y extraer información relevante. Estos grupos de edificios los llamaremos *incendios*.

Un AEF consta básicamente de dos listas. Una que contiene todos los edificios vecinos en el borde del incendio que aun no han sido tomados por las llamas, y otra de los edificios que: están en fuego, han sido apagados o han sido completamente consumidos por el incendio.

3.5.1. Inicialización y actualización del incendio

El agrupamiento comienza a partir de un edificio en fuego, que será el principal edificio del incendio. Luego se toman los vecinos del edificio principal y se colocan en la lista de edificios sin quemar.

Cada cierto tiempo es necesario actualizar el grupo, pues los que antes eran edificios sanos seguramente ya no lo son. Para hacer esto se actualizan las dos listas del AEF. Esto se logra haciendo un algoritmo parecido a Dijkstra [Dijkstra, 1959] donde la lista de abiertos se inicializa con todos los elementos del borde que se encuentren en fuego y la lista de cerrados con los elementos que se conocían ya incendiados. Luego se expanden los edificios en la lista de abiertos con sus vecinos. Los edificios expandidos se incluyen en la lista de cerrados y los vecinos obtenidos, si se estan quemando, se incluyen en la lista de abiertos. Este algoritmo se repite hasta que la lista de abiertos este vacía, es decir, hasta que todos los edificios vecinos en fuego sean explorados. El algoritmo 3.5.1 muestra con más detalle el procedimiento explicado anteriormente.

3.5.2. Información Sintetizada

Luego de hacer el agrupamiento, la información que se logra extraer de cada uno de los incendios es la siguiente:

- **Mejor edificio para extinguir:** Esto se logra gracias a una función pesada de cuatro variables:
  - La suma del área de los edificios vecinos sin incendiar
  - El área del edificio evaluado
  - Distancia euclidiana de la posición del bombero al edificio

  - Una medida que indica que tan avanzado se encuentra el fuego en el edificio (*fireness*).

- **Área del incendio:** Se suma el área de todos los edificios incendiados.
- **Numero de edificios sin incendiar:** Cantidad de edificios en el borde del incendio que se encuentran sanos
- **Ubicación del incendio:** Distancia del centro del incendio al centro del mapa.

Algoritmo 3.5.1: ACTUALIZARINCENDIO()

```
//burned: edificios apagados, en fuego o completamente quemados
//borderHealthy: edificios sanos alrededor del incendio

cerrados ← burned
abiertos ← {e ∈ borderHealthy | ESTAENFUEGO(e)}
burned ← burned ⋃ abiertos
borderHealthy ← borderHealthy − abiertos
while not ESVACIO(abiertos)
   b ← REMOVER-PRIMERO(abiertos)
   INSERTAR(cerrados, b)
   for each v ∈ VECINDAD(b)
      if v ∉ cerrados
         if ESTAENFUEGO(v)
            INSERTAR(abiertos, v)
            INSERTAR(burned, v)
          else
            INSERTAR(borderHealthy, v)
```

Capítulo 4

Implementación

En este capítulo describe los cambios que se hicieron al sistema durante el desarrollo de este trabajo. Además se explica los detalles de implementación de los agentes de **Tuqueque Team**, los cuales fueron desarrollados en *Java*, utilizando la interfaz *yabAPI*. Durante la implementación se presentaron algunos problemas que no fueron previstos en la fase de diseño, los cuales son detallados en el apéndice F.

4.1. Interfaz de Programación

Actualmente existen dos interfaces de programación (API) que encapsulan la comunicación de los agentes con el kernel. La primera interfaz, llamada ADK, fue desarrollada en C++ por Michael Bowling ([Bowling, 2000]). La segunda, yabAPI, está desarrollada en Java por Takeshi Morimoto ([Morimoto, 2004]) y es la interfaz oficial del proyecto.

Inicialmente se estudió la posibilidad de utilizar ADK como interfaz de programación, pues su lenguaje de implementación permite que el código sea rápido y optimizable en el momento compilación. Sin embargo, la implementación de [Bowling, 2000] no está completa y no soporta algunas acciones básicas de los agentes como la extinción de incendios. Además su documentación es pobre y no existe un grupo de personas que brinde soporte.

Se decidió utilizar yabAPI para programar a los agentes ya que provee una implementación más completa y su documentación, aunque aun es limitada, es mejor que la de ADK. Posee una lista masiva de correos que provee soporte a todas las versiones del API y sigue siendo desarrollada por el Comité Técnico de Robocup Rescue.

4.2. Modificaciones al Sistema de RCRSS

Para facilitar el proceso de implementación y depuración fue necesario realizar algunos cambios al sistema de simulación. Además se desarrolló un script para iniciar la simulación

que aprovecha la modularidad del sistema y facilita la modificación de los parámetros para cada corrida.

4.2.1. Script de arranque

Actualmente el procedimiento para iniciar todos los simuladores del sistema consiste en correr ocho scripts manualmente y luego iniciar los agentes. Esto suele ser tedioso, sobre todo cuando se desean hacer varias corridas de forma distribuida.

Se decidió implementar un script versátil que encapsulara todo el procedimiento y aprovechara la capacidad de correr los subsimuladores en máquinas distintas. Básicamente el script permite iniciar y terminar simulaciones completas, guardando registros de cada corrida. Este desarrollo y más detalles de implementación se explican en profundidad en el apéndice G.

4.2.2. Módulos añadidos

El sistema no cuenta actualmente con ningún elemento que calcule estadísticas como número de agentes vivos y edificios salvados en cada corrida. Solamente el visor *3DRescue* desarrollado por [Kleiner y Göbelbecker, 2005] tiene esta funcionalidad. Sin embargo, para extraer esta información de los registros, construye una interfaz gráfica considerablemente pesada que resulta innecesaria. Por esta razón se decidió hacer cambios sobre este visor, dejando solamente la funcionalidad de estadísticas y agregando otros elementos calculables que no fueron contemplados por sus autores, como por ejemplo, número de agentes vivos y porcentaje de calles no transitables.

Aprovechando la flexibilidad del script explicado en la sección 4.2.1, se agregó un módulo al sistema que grafica los valores de interés de la simulación a tiempo real. Más detalles sobre este módulo se explica en el apéndice G.

4.2.3. Modificaciones al kernel

A pesar de estar mantenido y soportado por un numeroso grupo de usuarios, el kernel posee algunos errores de programación. Los más importantes de los hallados por **Tuqueque Team** se reportaron a las listas masivas de *Robocup Rescue* pero no se recibió respuesta y fue

necesario depurar y modificar el código fuente. Los errores detectados fueron los siguientes:

- Cuando se inicia el kernel utilizando el script automático, éste pierde la habilidad de manejar señales correctamente. Este manejo utiliza un código que ha sido abandonado y fue reemplazado por un procedimiento actualizado. Se reportó esta solución para versiones futuras del sistema.

- En situaciones aún no determinadas, el kernel no puede iniciar los datos del mapa y su ejecución se congela. Este problema no fue resuelto por no encontrarse dentro del alcanze de esta investigación, sin embargo ocurre en muy pocas ocasiones.

4.2.4. Modificaciones al visor

Durante este estudio se contó con las implementaciones de visores de Kuwata [Kuwata, 2005], ResQ [Kleiner y Göbelbecker, 2005] (visor tridimensional) y el visor por defecto del sistema. Lamentablemente, el primero no es compatible con todos los mapas de ciudades disponibles y el uso del segundo resulta inconveniente para el desarrollo de los agentes de **Tuqueque Team** por los altos requerimientos gráficos del visor. Se decidió utilizar el visor por defecto, pero fue necesario realizar las siguientes modificaciones al mismo:

- Los colores que representan a los objetos del mundo no pueden ser diferenciados por personas que sufren de daltonismo. Teniendo en cuenta que uno de los integrantes de **Tuqueque Team** es daltónico fue necesario modificar los colores del visor.

- El visor no identifica a ningún elemento del mundo, haciendo que la depuración sea imposible. Se agregó la funcionalidad de mostrar los identificadores de los agentes, edificios, calles y nodos.

- Para mejorar la depuración del planificador de caminos, se agregó la funcionalidad de resaltar los *LongRoads*.

4.2.5. Limitaciones del sistema

Con el repetido uso del sistema de simulación se identificaron las siguientes limitaciones:

- Los simuladores de bloqueo y tráfico no pueden ser ejecutados en máquinas remotas y solamente funcionan adecuadamente si se corren en la misma máquina que el kernel.
- El simulador de tráfico y el visor son los componentes que tienen más posibilidad de detenerse por errores. La estructura de sus códigos no permite su depuración.
- En ocasiones los datos de los paquetes UDP contienen información inválida o errónea. Según las listas de correo de soporte, se trata de un problema conocido derivado de las limitaciones del protocolo, el cual no asegura que los datos lleguen a su destino.
- La documentación del código fuente del sistema es pobre y en algunos casos inexistentes. La mayoría de los comentarios están en japonés.

4.3. Detalles de Implementación

En esta sección se presentan una serie de detalles importantes de implementación de los agentes del **Tuqueque Team**. Ellos ayudan a comprender en un nivel más específico las funciones que cumplen los agentes para realizar sus tareas.

4.3.1. Generación de reglas aleatorias

Para comparar el desempeño de unos agentes con reglas generadas mediante un entrenamiento fue necesario agregar una funcionalidad al sistema XCS para inicializar el sistema con reglas aleatorias. Con esta adición se pueden obtener agente *Ambulance Center* y *Fire Station* que toman decisiones de rescate y extinción de incendios de manera aleatoria.

La creación de una regla aleatoria consiste en inicializar todos los bits de la condición y la acción con cualquier valor posible, es decir, 0, 1 ó *don't care*. Cada valor tiene la misma probabilidad de ser escogido en cada bit. Este proceso se repite hasta llenar todo el conjunto de clasificadores del sistema.

4.3.2. Listas *HashSets*

Cada uno de los objetos del mundo de RCRSS tiene un identificador único. YabAPI aprovecha esto y define que la función de hash de cada elemento es su identificador. Por

este detalle cada vez que fue necesario utilizar listas de elementos se decidió almacenarlos en *HashSets*, ya que la búsqueda en esta estructura es una operación de orden constante.

Los elementos añadidos al modelo del mundo, como los *LongRoads*, también se implementaron con un identificador único.

4.3.3. Selección de objetivos aleatorios

Como se indicó en la sección 3.1.1, para explorar edificios los agentes escogen el edificio más cercano y se mueven hasta algún punto donde se pueda observar a éste. Cuando varios agentes están en la misma posición y tienen la misma información del mundo, es posible que escojan los mismos objetivos. Para resolver esto se implementó la selección de objetivos con un componente aleatorio. Una de cada diez selecciones de edificios escogerá un elemento aleatorio y no el más cercano. Este detalle también se aplicó para la selección de calles de desbloqueo en los policías.

4.3.4. Construcción de grafo de *LongRoads*

La construcción del grafo de *LongRoads* del mundo se implementó con un procedimiento de dos pasos como se muestra en el algoritmo 4.3.1: primero se copia el grafo de nodos del mundo y se conectan utilizando *LongRoads* en lugar de calles, y en segundo lugar se buscan todos los nodos que tienen solamente dos *LongRoads* como vecinos, reemplazando a cada nodo encontrado y sus dos vecinos por un nuevo *LongRoad*.

Algoritmo 4.3.1: INICIALIZARLONGROADS(G_{mundo} : Grafo de nodos y calles del mundo)

```
// contrucción inicial a partir del grafo del mundo
G ← OBTENER-NODOS(G_mundo)
for each r ∈ OBTENER-CALLES(G_mundo)
    inicio ← NODO-INICIAL(r)
    final ← NODO-FINAL(r)
    lr ← NUEVO-LONGROAD(inicio, final, {r})
    INSERTAR(G, lr)
// eliminación de nodos con dos vecinos
listanodos ← {n ∈ G | NUMEROVECINOS(n) = 2}
while not ES-VACIO(listanodos)
    n ← REMOVER-PRIMERO(listanodos)
    lr_1, lr_2 ← VECINOS(n)
    inicio ← NODO-INICIAL(r_1)
    final ← NODO-FINAL(r_2)
    lr_nuevo ← NUEVO-LONGROAD(inicio, final, {lr_1, n, lr_2})
    AGREGAR(G, lr_nuevo)
    ELIMINAR-TODOS(G, {lr_1, lr_2, n})
```

4.3.5. Descomposición de *LongRoads* bloqueados

Cuando un *LongRoad* contiene una calle bloqueada, el costo de éste es penalizado y se asume que el *LongRoad* no es transitable. Por esto fue necesario implementar una descomposición de *LongRoads* para evitar que los elementos que son transitables del *LongRoad* sean castigados con un costo elevado.

Suponiendo que se tiene un *LongRoad* lr_b que tiene una calle r_b que está bloqueada. Sean n_1 y n_2 los nodos inicial y final de lr_b. Sean n_{b1} y n_{b2} los nodos de la calle r_b. El procedimiento de descomposición elimina a lr_b del grafo y lo reemplaza por tres nuevos *LongRoads*: uno desde n_1 hasta n_{b1}, otro similar desde n_{b2} hasta n_2 y por último el *LongRoad* bloqueado

desde n_{b1} hasta n_{b2}, que contiene a la calle no transitable.

4.3.6. Desesperación para Detectar Congestiones

Para resolver el problema de congestión de agentes en las vías se implementó un límite de espera en los agentes pelotón. Este límite funciona de la siguiente manera: en cada turno el agente debe verificar si se ha movido, si el agente detecta que, a pesar de tener como tarea moverse hacia un lugar específico, no se ha movido por α turnos, entonces modifica su ruta para moverse al edificio más cercano. El agente espera por los siguientes β turnos en el edificio para dar paso a otros agentes que se encuentran transitando por el lugar o que vienen a desbloquear las vías.

α y β son números que varían dependiendo del tipo de agente (ver cuadro 5). Es importante destacar que los agentes policiales tienen niveles de desesperación más altos ya que estos deben ser más persistentes para llegar a los lugares de alto tránsito y poder desbloquear las vías para permitir el paso a otros agentes.

Tipo de Agente	α	β
Ambulance Team	1	3
Police Force	2	N/A
Fire Brigade	1	3

Cuadro 5: Cuadro de Niveles de Desesperación de los Agentes Pelotón de **Tuqueque Team**

4.3.7. Desesperación de Ambulancias en Rescate de Agentes Pelotón

Para evitar que una ambulancia invierta demasiado tiempo rescatando a un agente se implementó un límite de espera, es decir, se le otorga a la ambulancia un lapso de 40 iteraciones para realizar la tarea luego del cual la ambulancia se "desespera" y abandona al agente sin culminar su rescate.

4.3.8. Desesperación de Policías

Al igual que la desesperación de ambulancias, los policías deben evitar perder ciclos tratando de alcanzar su objetivo. En este caso es importante que los policías sean pacientes ya que sólo ellos pueden resolver la congestión por bloqueo de las calles, por esto el límite de desesperación de los policías es más flexible.

Cuando un policía se desespera no abandona inmediatamente sus objetivos ni la tarea que está realizando. Primero intenta rutas alternativas usando la "lista negra" mencionada en la sección 3.1.8. Cuando las vías alternativas también presentan congestión el policía cancela su labor de limpieza.

4.3.9. Cantidad de Agua de Bomberos

Los *Fire Brigade* tal y como estaban modelados en la interfaz de programación de aplicaciones *YabAPI* no contenía un atributo *WaterQuantity* (sección 2.3.2), por lo tanto, fue necesario modificar el código fuente del mismo para poder contar con este atributo. *WaterQuantity* es utilizado por los bomberos para observar la cantidad de agua en su tanque cuando lo deseen.

4.3.10. Vértices de un Edificio

Al igual que el atributo *WaterQuantity*, el atributo *BuildingApexes* (cuadro 9 del apéndice B) no se encontraba presente en el *YabAPI*, por lo que fue necesario realizar modificaciones al código fuente de la interfaz para poder contar con esta información de los edificios.

Los vértices son utilizados en los algoritmos de búsquedas de vecinos para el agrupamiento de incendios.

4.3.11. Mensajes con información obsoleta

Cuando un agente recibe un mensaje que contiene información sobre objetos del mundo, ésta podría estar obsoleta. Para evitar que los agentes reemplacen información más actualizada del mundo éstos siempre verifican cuál es el tiempo de observación indicado por el mensaje, como se explica en la definición del protocolo en el apéndice E.

4.3.12. Manejo de *charsets* para Mensajes

Para construir los mensajes de radio y de voz utilizando el protocolo especificado en el apéndice E, se concatenan por orden de prioridad cada *Token* en una cadena de caracteres. Sin embargo, este procedimiento puede cambiar los valores de cada caracter porque Java solamente permite que sus cadenas de caracteres contengan valores definidos en un *charset*.

Esta limitación podría impedir la implementación del protocolo de comunicación, pero se logró configurar la creación de cadenas de caracteres para que utilizaran un *charset* de caracteres rusos que permite el uso de cualquier valor en un caracter.

4.3.13. Víctimas Muertas

Existe la posibilidad de que el *Ambulance Center*, como consecuencia de la pérdida de vigencia de la información del mundo, envíe a un grupo de ambulancias a rescatar a una víctima que se encuentre muerta. En este caso el equipo de ambulancias reportará al *Ambulance Center* que la víctima ha muerto y seguirá con sus labores habituales.

4.3.14. Víctimas en Incendios

Las ambulancias de **Tuqueque Team** no hacen el rescate de víctimas que se encuentren en un edificio en llamas, ya que si lo hicieran el fuego produciria daños en las ambulancias. Es por esto que si una víctima que está programada para ser rescatada se encuentra en un edificio en llamas las ambulancias ignorarán la orden de rescate e informarán al centro de ambulancias que la víctima no puede ser rescatada.

4.3.15. Búsqueda de vecinos de un edificio

Para la búsqueda de edificios vecinos (EV) de un edificio (E) primero es necesario definir que se considera vecino. En el caso de **Tuqueque Team** vecinos son los 20 edificios más cercanos que se encuentran en un radio de aproximadamente 16 metros. De esta menera la búsqueda de vecinos se puede dividir en las dos operaciones que se explican a continuación:

- **Encontrar los 20 edificios más cercanos:** Para lograr esto, cada uno de los agentes arma un árbol KDtree (ver sección D) al principio de la simulación tomando como

puntos los centros de todos los edificios. En el momento en que se desee buscar los 20 edificios más cercanos a un edificio, se usan los operadores de KDtree sobre el árbol generado y se obtienen. La implementación de KDtree utilizada para esta tarea fue tomada de [Levy,].

- **Verificar radio de vecindad:** Luego de encontrar los 20 edificios más cercanos, se verifica cuales de ellos contienen un vértice dentro de un círculo centrado en E con radio de 16 metros. Esto se logra evaluando si se cumple la ecuación (12), donde xc y yc son las coordenadas del centro, x y y son las coordenadas del punto que se verifica y R es el radio del círculo.

$$(x - xc)^2 + (y - yc)^2 \leq R^2 \tag{12}$$

Capítulo 5

Experimentos y Resultados

En éste capítulo se definen los esquemas para el entrenamiento de los XCS de **Tuqueque Team** y se presentan los resultados de este proceso. También se diseñan y analizan experimentos para observar y comparar el funcionamiento de los agentes de **Tuqueque Team** con respecto a otras soluciones elaboradas por otros equipos de la competencia.

5.1. Descripción de Entrenamientos

La generación de reglas del XCS de **Tuqueque Team** se realizó entrenando dos conjuntos de clasificadores diferentes. El primer entrenamiento se realizó en los mapas de la ciudad de *Foligno* y el segundo en los mapas de la ciudad de *Kobe*. Ambos entrenamientos utilizaron los parámetros explicados en la sección 3.4.

5.1.1. Entrenamiento para *Foligno*

Para este entrenamiento se usaron dos mapas de la misma ciudad: *Foligno* y *FolignoEasy*. Esta ciudad presenta un compleja red de calles y nodos, una distribución de edificios y civiles de una ciudad real y otras propiedades del ambiente que hacen que la labor de extinción de incendios sea difícil.

El procedimiento consistió de 900 corridas hechas de maneras secuencial alternando los mapas del conjunto de entrenamiento. Los XCS del *Ambulance Center* y *Fire Station* comienzan con un conjunto de reglas vacío y luego, para cada corrida, se utilizaron las reglas de la corrida anterior. Este proceso se llevó a cabo utilizando las especificaciones de hardware y plataforma descritos en la sección 5.2.2, específicamente el modelo 2 del cuadro 7.

5.1.2. Entrenamiento para *Kobe*

El conjunto de mapas de entrenamiento contiene las siguientes ciudades: *KobeHard* y *KobeEasy*. A diferencia de *Foligno*, esta ciudad es más simple en cuanto a estructura de

calles, número de edificios y extinción de incendios. Presenta la ventaja de tener siete refugios lo que facilita las labores de recarga de agua y descarga de víctimas.

Al igual que el entrenamiento anterior, se hicieron 900 corridas secuenciales iniciando los sistemas XCS con conjuntos de reglas vacíos. Las especificaciones de hardware y plataforma fueron las mismas del entrenamiento de *Foligno*.

5.2. Descripción de Experimentos

Experimento	Mapa	Equipos	Número de corridas
Análisis detallado del desempeño de **Tuqueque Team**	*Foligno*	• **Tuqueque Team** entrenado en *Foligno*	1
Comparación de agentes	*KobeHard*	• **Tuqueque Team** entrenado en *Kobe* • **Tuqueque Team** con reglas aleatorias • **ResQ Freiburg** • **DAMAS Team** • **5Rings** • **sampleagents**	20
	FolignoEasy	• **Tuqueque Team** entrenado en *Foligno* • **Tuqueque Team** con reglas aleatorias • **sampleagents**	20
	RandomEasy	• **Tuqueque Team** entrenado en *Kobe* • **Tuqueque Team** con reglas aleatorias • **sampleagents**	20

Cuadro 6: Descripción de experimentos

Para observar el desempeño de los agentes de **Tuqueque Team** se diseñaron dos experimentos que se presentan en el cuadro 6. El primero consiste en completar una corrida en el mapa *Foligno*, y obtener estadísticas de agentes vivos, edificios a salvo y calles desbloqueadas

en cada ciclo de la simulación para observar en detalle el desempeño de **Tuqueque Team**. El segundo es una comparación de una corrida promedio de diversos tipos de agentes para comparar el funcionamiento de cada uno en distintos escenarios. Este experimento contiene dos grupos de agentes de **Tuqueque Team** con un parámetro diferente: uno con reglas entrenadas (veáse entrenamiento en la sección 5.1) y otro con reglas generadas aleatoriamente. El objetivo de esta selección es observar la influencia del proceso de aprendizaje.

5.2.1. Descripción de agentes del experimento

A continuación se describen los agentes utilizados en los experimentos del cuadro 6:

- **Tuqueque Team entrenado en *Foligno*:** Utiliza los agentes desarrollados durante este estudio con reglas del XCS obtenidas del proceso de entrenamiento en *Foligno*, como se explica en la sección 5.1.1.
- **Tuqueque Team entrenado en *Kobe*:** Agentes como los explicados aparte anterior, pero con las reglas aprendidas en el entrenamiento de *Kobe* (veáse sección 5.1.1).
- **Tuqueque Team con reglas aleatorias (Tuqueque Random):** Estos agentes poseen un XCS con reglas generadas aleatoriamente y sin ser entrenadas, como se explicó en la sección 4.3.1 del capítulo 4.
- **sampleagents:** Son los agentes de ejemplo que provee la distribución de yabAPI. Éstos están programados con comportamientos básicos pero funcionales.
- **ResQ Freiburg:** Los agentes de *ResQ Freiburg* [Kleiner et al., 2004]. La actividad de éstos se determinó a partir de los registros de la competencia publicados en [RCRCommitee, 2004a].
- **DAMAS Team:** Agentes de *DAMAS Team* [Paquet et al., 2004], participantes del 2004. Al igual que *ResQ Freiburg*, la información de las corridas se determinó a partir de los registros de la competencia.
- **5Rings:** Agentes de *5Rings* [Silva y Coelho, 2004]. También se determinó a partir de los registros publicados.

5.2.2. Descripción de Hardware y Plataforma

La descripción del hardware y la plataforma utilizada para los entrenamientos y experimentos se presentan a continuación en los cuadros 7 y 8.

Modelo 1

Componente	**Descripción**
Procesador	Intel ® Pentium ® 4 2.80GHz Cache 1024KB
Memoria	512MB DDR RAM

Modelo 2

Componente	**Descripción**
Procesador	Intel ® Pentium ® 4 2.80GHz Cache 512KB
Memoria	2.5GB DDR RAM

Cuadro 7: Descripción de Hardware

Componente	**Descripción**
Sistema de Operación	Gentoo Linux - Kernel Vanilla 2.6.11.7
Compilador de C	GCC 3.3.5
Entorno de Java	Sun Java Development Kit Versión 1.4.2
Versión de Python	Python 2.3.5
Shell	Bash Versión 3.00.16

Cuadro 8: Descripción de Plataforma de Software

5.3. Resultados y Análisis

En esta sección se presentan los resultados de la realización de los experimentos descritos en la sección 5.2.

5.3.1. Resultados de los entrenamientos

Para la evaluación de los resultados de los entrenamientos se establecieron, además del puntaje definido en la equación 1 de la sección 2.2, dos nuevas métricas que permiten describir mejor la labor de los bomberos y ambulancias. La primera es el porcentaje de edificios

destruidos, que es la cantidad de edificios que fueron consumidos por el fuego. Un alto porcentaje de edificios destruidos indica que los bomberos no realizaron su tarea de manera efectiva. La segunda es el porcentaje de agentes vivos, que indica el número de agentes con puntos de vida positivos. Una alta cantidad de agentes vivos indica que las labores de rescate llevadas a cabo por las ambulancias fueron satisfactorias.

La figura 14 contiene la evolución del puntaje de **Tuqueque Team** durante los entrenamientos de los sistemas XCS de selección de ambulancias e incendios. Los valores están interpolados mediante una curva de bezier.

Durante las primeras doscientas corridas, el puntaje y el porcentaje de víctimas vivas al final de cada simulación aumentan paulatinamente. De la misma manera, la cantidad de edificios destruidos decrece constantemente en este período. Es en estas corridas donde el sistema de clasificadores tienen su mayor tasa de aprendizaje.

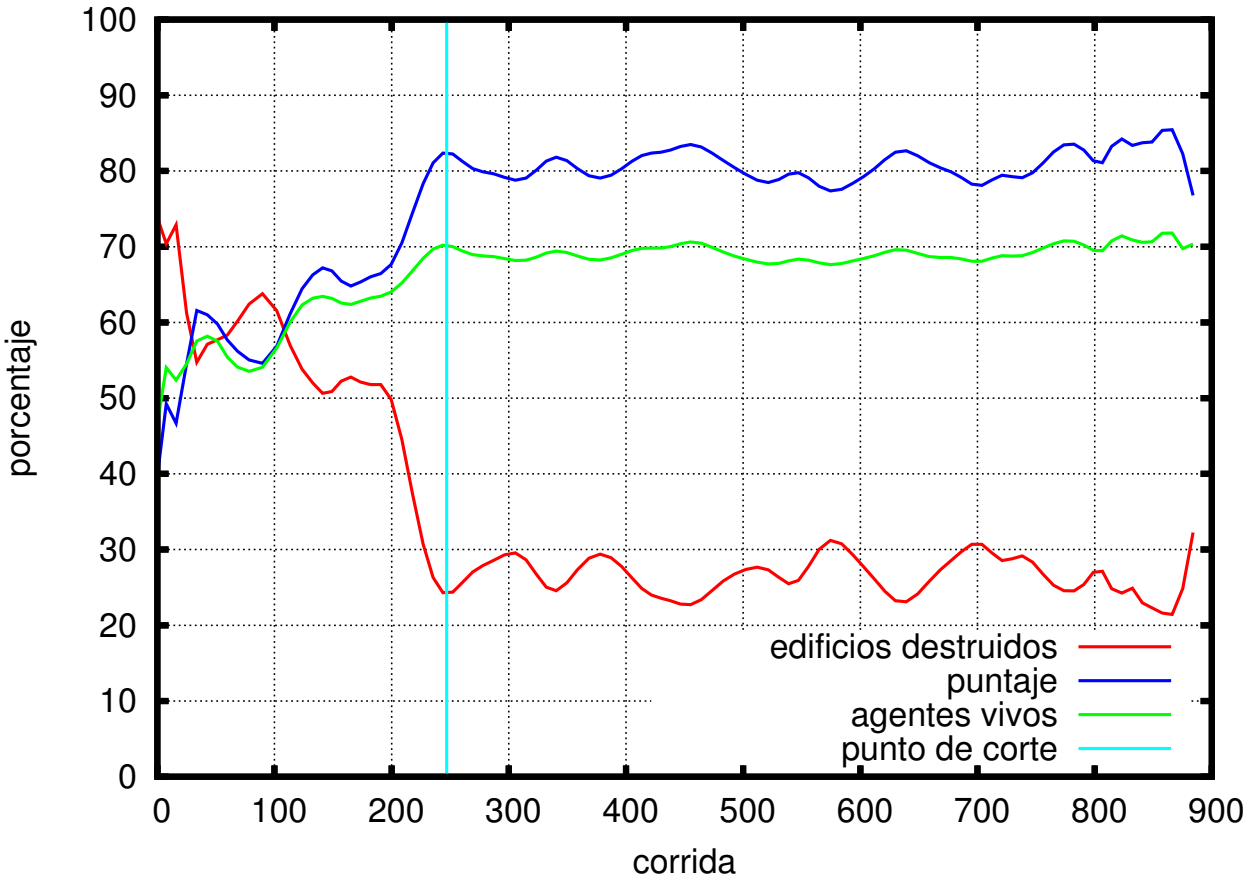


Figura 14: Gráfica de Entrenamientos en *Foligno*

La figura también muestra que entre las corrida 200 y 300 el puntaje del equipo llega a un tope y en las siguientes corridas se produce una estabilización de las tres variables de interés hasta el final del entrenamiento. Se estableció la corrida 247 como punto de corte para el entrenamiento de los agentes. El conjunto de clasificadores resultante de esta corrida fue el utilizado para realizar las pruebas de desempeño de **Tuqueque Team**.

La variación del porcentaje de agentes vivos entre las corridas mencionadas anteriormente se debe a que el sistema XCS, específicamente el de las ambulancias, evoluciona un buen conjunto de reglas. Durante el resto del proceso de aprendizaje, no se encuentran reglas capaces de mejorar los resultados de la generación del punto de corte.

En cuanto al sistema de aprendizaje de los bomberos, no se repite el mismo comportamiento que presentaron las ambulancias. Un análisis detallado se presenta más adelante en el experimento de comparación con otros agentes (sección 5.3.2).

5.3.2. Resultados de los experimentos

Experimento 1 - Análisis detallado de una corrida:

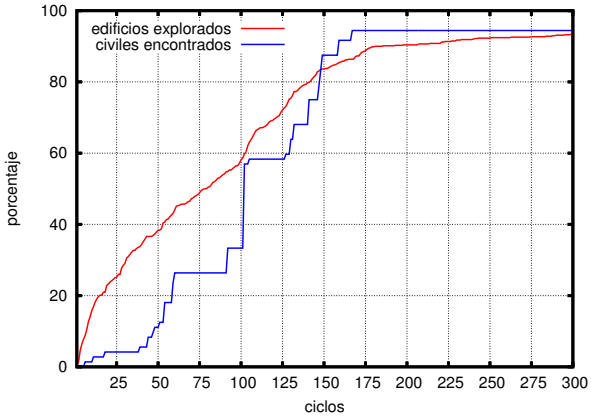
La figura 15 presenta los resultados detallados de una corrida de los agentes de **Tuqueque Team** en el mapa de la ciudad de *Foligno*.

En la figura 15(a) se muestra el desarrollo de las labores de exploración de **Tuqueque Team** en la corrida de muestra. Podemos observar que los agentes fueron capaces de encontrar casi el cien por ciento de las víctimas del mapa, así como también, de explorar casi la totalidad de los edificios de la ciudad. La incompletitud de la exploración se debe a que los agentes ignoran los edificios que están en llamas y por lo tanto no logran ver a las víctimas que se encuentran en estos edificios.

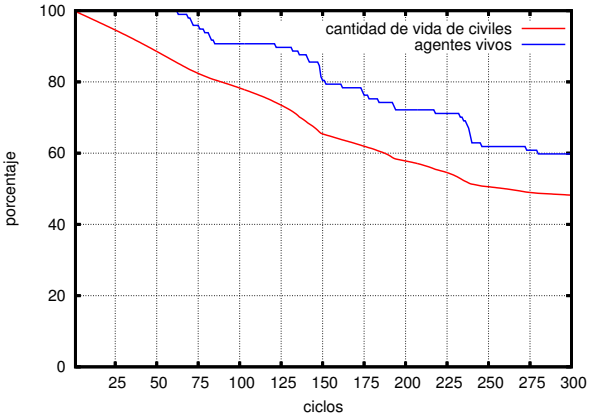
En la figura 15(b) se encuentra reflejado el desempeño del escuadrón de ambulancias. Se puede observar que aproximadamente el 60 % de los agentes logran sobrevivir al desastre lo cual es satisfactorio tomando en cuenta que el tamaño y nivel de bloqueo hacen que *Foligno* sea un mapa bastante complejo.

Podemos observar en la figura 15(c) que alrededor del ciclo 100 el 90 % de las calles son

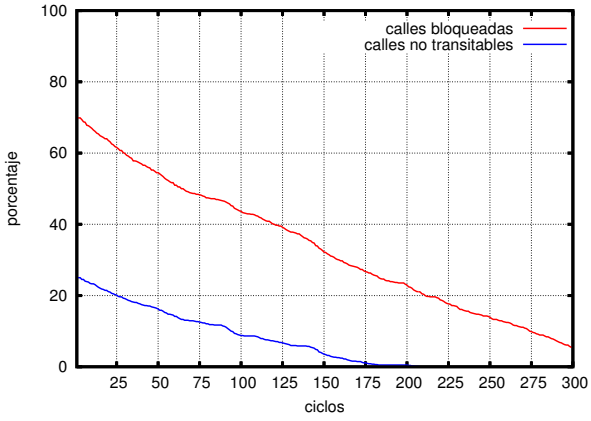
transitables. Esto facilita las labores de los demás agentes y aunque este valor no influye directamente en el puntaje del equipo, permite que otros agentes realicen sus tareas más eficientemente. A pesar de que el mapa inicialmente tiene el 70 % de sus calles bloqueadas, al final de la simulación los policías logran desbloquear casi todas las calles de la ciudad.



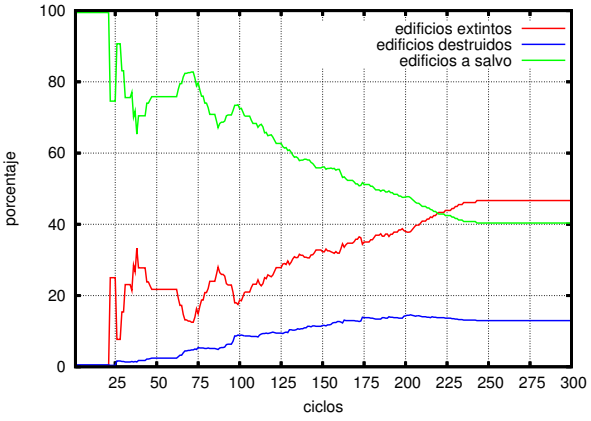
(a) Desempeño de exploración



(b) Desempeño de ambulancias



(c) Desempeño de policías



(d) Desempeño de bomberos

Figura 15: Desempeño detallado de los agentes de **Tuqueque Team** en una corrida en el mapa *Foligno*

En la figura 15(d) se observa cómo la cantidad de edificios extintos varía notablemente en pocos ciclos. Las disminuciones de este valor se deben a la expansión de los incendios y los aumentos del valor son consecuencia directa de la labor de extinción de los bomberos.

Se puede notar que en esta corrida el combate de los incendios fue una difícil labor, que culminó con el control de éste en el ciclo 230 aproximadamente. Al final de la simulación los bomberos lograron extinguir aproximadamente el 45 % de los edificios que estuvieron en llamas, permitiendo únicamente la destrucción de un 15 % de los edificios de la ciudad. El 40 % de los edificios de la ciudad quedaron intactos, es decir, nunca fueron alcanzados por ningún incendio.

Experimento 2 - comparacion entre agentes

a.- Comparación en *KobeHard*

En la figura 16 se observa que los agentes **Tuqueque Team** con un conjunto de clasificadores entrenados presentan resultados satisfactorios. El desempeño de los bomberos observado mediante la variable de porcentaje de edificios no destruido es el resultado más interesante pues mejora el de todos los grupos de agentes del experimento.

El alto desempeño de los bomberos de **Tuqueque Team** se debe a un buen conjunto de clasificadores obtenidos del entrenamiento. En el mapa *KobeHard* un conjunto de reglas aleatorias produce buenos resultados, pero no tan buenos como las reglas entrenadas. En cuanto el porcentaje de agentes vivos que mide la efectividad de las ambulancias, se puede observar que con las reglas entrenadas obtienen una mejoría leve con respecto al sistema de reglas aleatorias.

La labor de extinción de incendios obtuvo resultados satisfactorios gracias a un sistema de XCS que logró adaptarse en el mapa de *Kobe* mediante un buen proceso de entrenamiento. La información que utiliza este sistema es suficiente para determinar que acciones permiten que se ataquen los incendios de manera eficiente.

Por otro lado, la información que utiliza el sistema XCS de las ambulancias no parece ser suficiente para generar las mejores acciones para la labor de rescate de civiles. Las reglas producto del entrenamiento son buenas pero el sistema no decide sobre muchos elementos importantes del rescate, como el orden de rescate de víctimas.

Al comparar los resultados de los agentes **Tuqueque Team** con reglas aleatorias, sampleagents, *DAMAS Team* y *5Rings*, se puede concluir que el desarrollo de los agentes de

este estudio ha sido satisfactorio, ya que tomando decisiones aleatorias de rescate y extinción de incendios, **Tuqueque Team** puede arrojar resultados notablemente mejores que sampleagents y comparables con los equipos de la competencia del 2004.

Si se comparan los valores obtenidos de *ResQ Freiburg* con los de **Tuqueque Team** entrenado, se puede notar que el entrenamiento de los bomberos es una buena solución para el mapa *KobeHard*, mejorando la técnica implementada por el equipo *ResQ Freiburg*. Sin embargo, el trabajo de las ambulancias no superó a *ResQ Freiburg*, mostrando que existen mejores soluciones para la decisión de rescate de víctimas que el uso de un sistema de clasificadores genéticos XCS con los parámetros actualmente usados.

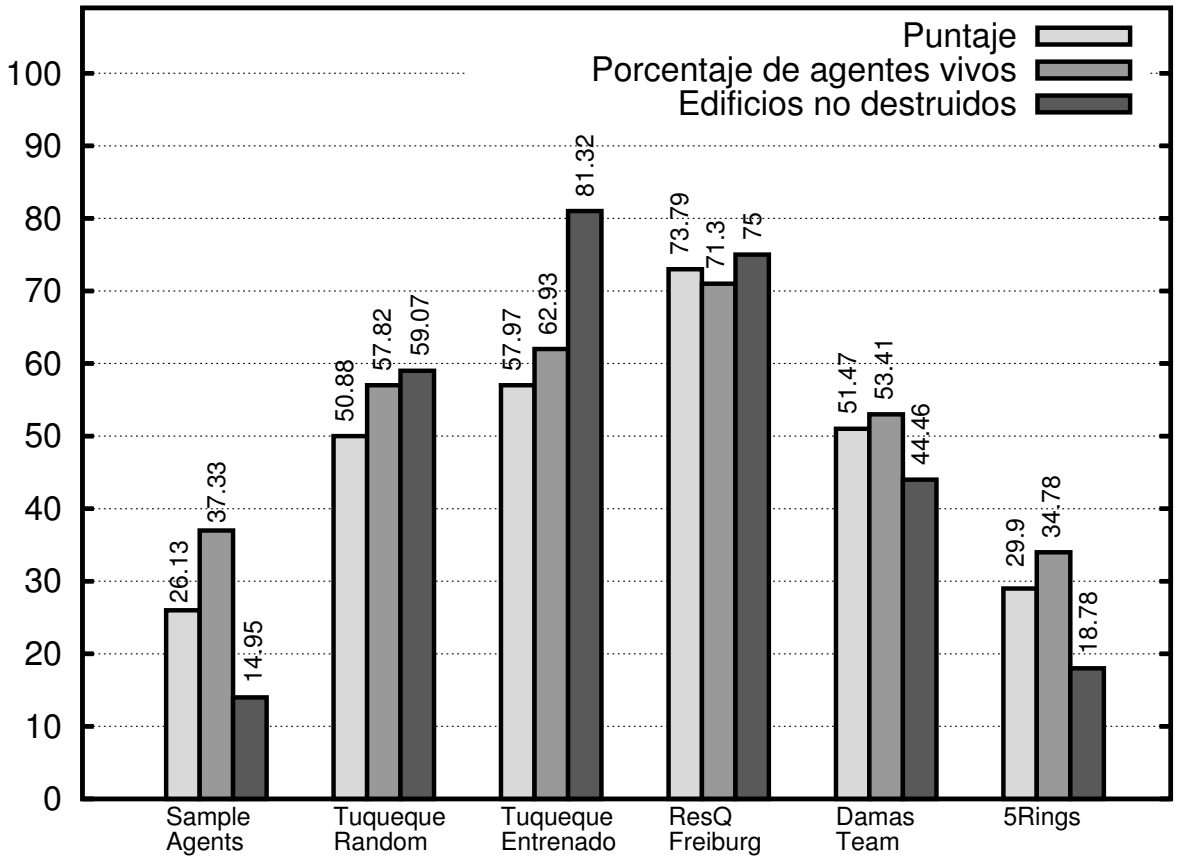


Figura 16: Comparación de Desempeño de Equipos en *KobeHard*

b.- Comparación en *FolignoEasy*

En la figura 17 podemos observar que las dos instancias de **Tuqueque Team** superan la puntuación, número de agentes vivos y cantidad de edificios no destruidos de los samplea-

gents. Sin embargo, la mejora que presentan no es tan notoria como en *KobeHard* o *RandomEasy*. Esto se debe a las complicaciones adicionales que presenta el mapa *FolignoEasy* como lo son:

- Mayor número de calles.
- Muchos edificios díficiles de extinguir gracias a sus propiedades.
- Número limitado de refugios.
- Un mayor tamaño del mapa.
- Se tiene casi la misma cantidad de agentes que en *KobeHard* o *RandomEasy*.
- Mayor número de derrumbes en las calles.
- Mayores problemas de tráfico.

En este experimento ocurrió un fenómeno interesante para los agentes de **Tuqueque Team**: las puntuaciones obtenidas por los agentes con reglas aleatorias fueron muy parecidas a las de los agentes entrenados. Específicamente los bomberos aleatorios tuvieron un desempeño mejor que los entrenados.

El hecho de que unos bomberos con reglas aleatorias presenten un mejor desempeño que los bomberos con clasificadores entrenados, indica que existen mejores soluciones que las alcanzadas por el sistema evolutivo. Nótese en la figura 14 que el aprendizaje de los bomberos se estabiliza después de las primeras trescientas corridas.

El estancamiento del aprendizaje de los bomberos se debe a un error en la parametrización del sistema de clasificadores. Tomando en cuenta que el número de posibles salidas de un clasificador de selección de incendios es de 2^{12} y que el el tamaño de la población es de 100, es muy probable que en el conjunto de clasificadores $[P]$ todas las reglas tengan acciones distintas entre sí. Como se indicó en la sección 2.5, se debe ejecutar un algoritmo genético sobre el *action set* $[A]$ para generar nuevas reglas. Pero si todas las reglas tienen acciones diferentes, todos los posibles *action sets* tendrán un solo elemento, por lo que no

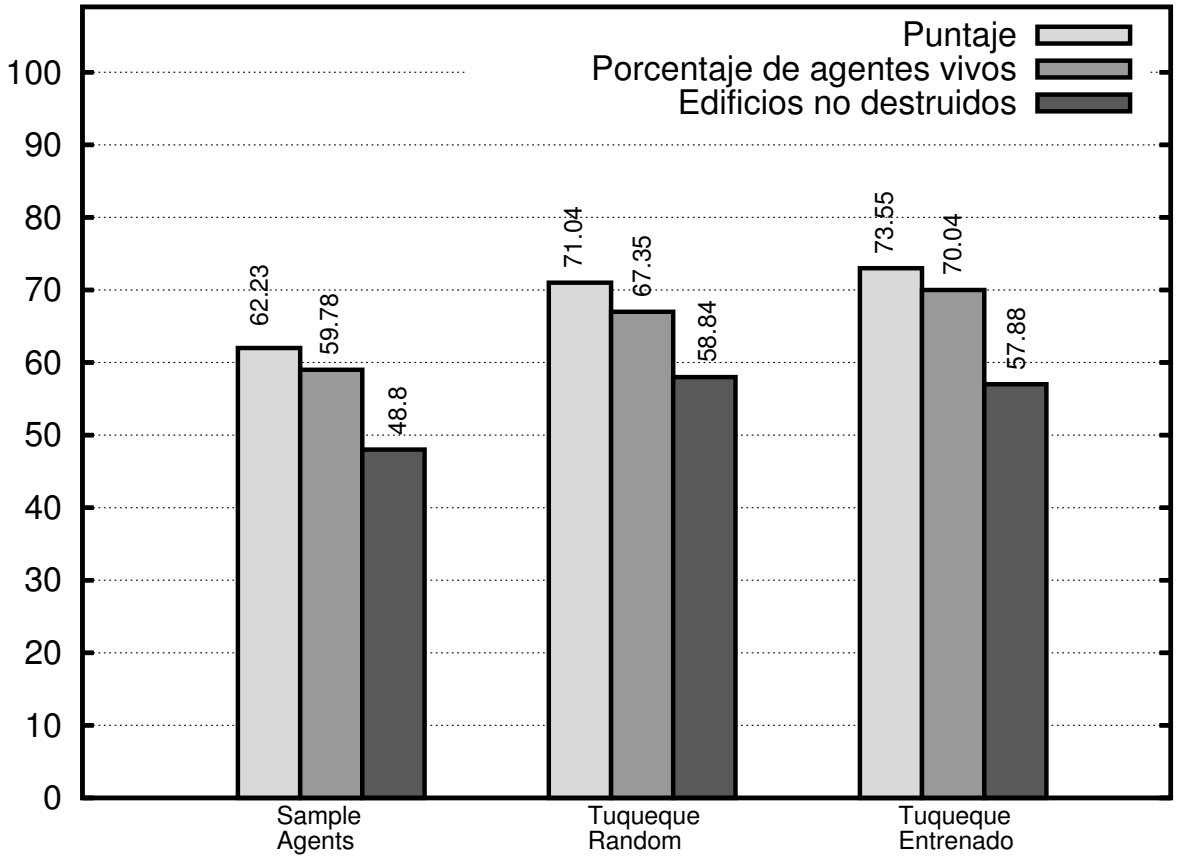


Figura 17: Comparación de Desempeño de Equipos en *FolignoEasy*

se podrá ejecutar el paso evolutivo. Una vez que se presenta este problema, el conjunto de clasificadores $[P]$ sólo puede ser modificado como consecuencia del covering. Por esto, disminuye la posibilidad de incluir nuevo material genético a la población y se decrementa la tasa de aprendizaje del sistema.

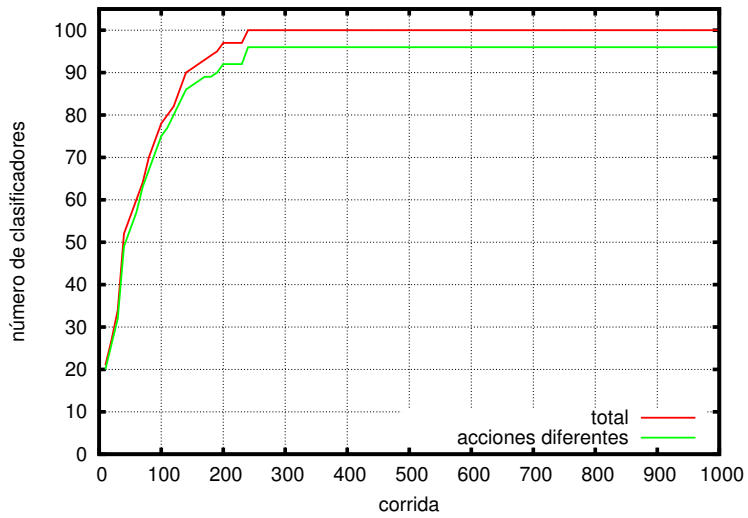


Figura 18: Estado de la Población durante el Entrenamiento

El problema de diseño del XCS explica la cantidad de edificios destruidos en la figura 15. Inicialmente se cuenta con un conjunto de clasificadores vacío que se pobla como consecuencia del covering y el algoritmo genético. El reemplazo elitista mantiene al conjunto $[P]$ con las mejores reglas hasta que su población llega al límite. Entre las corridas 200 y 300 ocurre el problema expuesto anteriormente y la tasa de aprendizaje decrece drásticamente. En la figura 18 se comprueba este comportamiento.

c.- Comparación en *RandomEasy*

En la figura 19 se muestra el desempeño de los agentes **Tuqueque Team** con reglas entrenadas en *Kobe*, con reglas generadas aleatoriamente y los agentes de sampleagents.

El puntaje obtenido por los agentes con reglas aleatorias es peor que el alcanzado por los agentes entrenados. Es importante destacar que a pesar de que los clasificadores fueron entrenados en la ciudad de *Kobe*, fueron los suficientemente generales como para resolver el problema en un ambiente diferente.

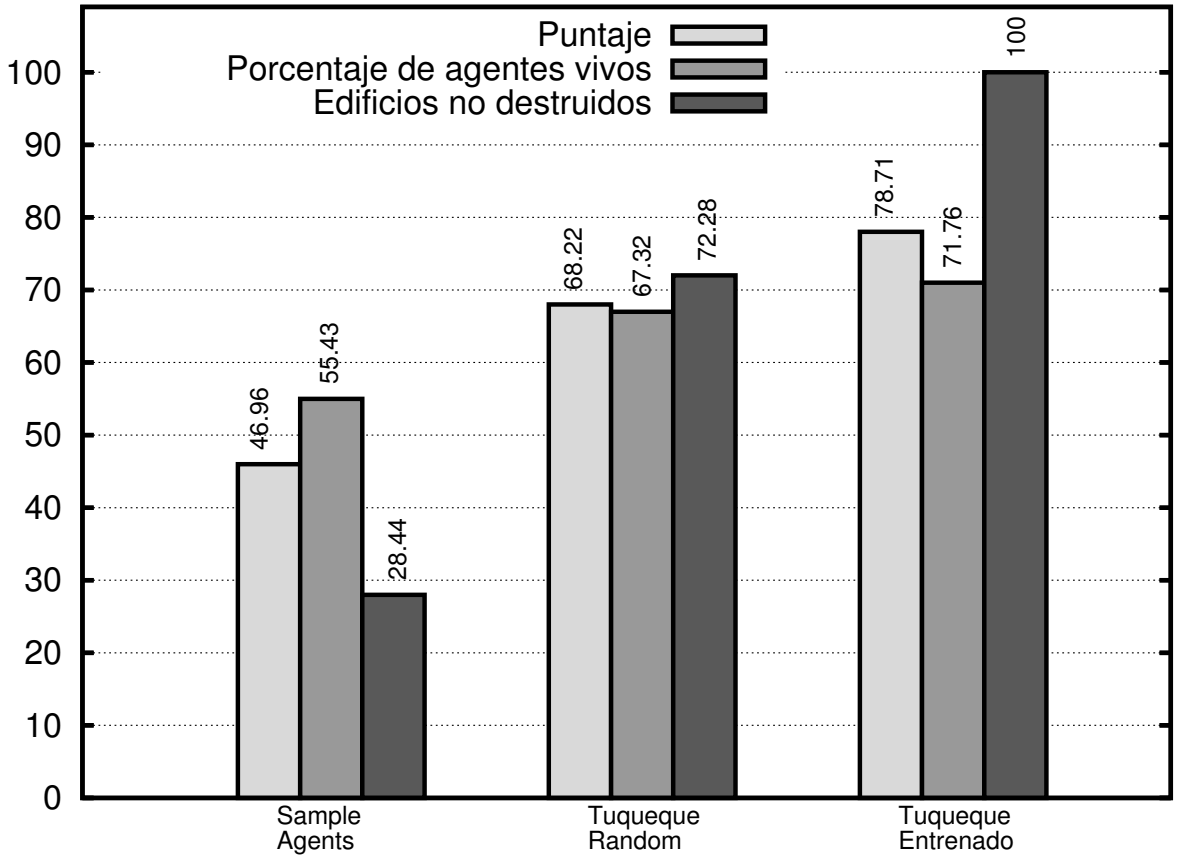


Figura 19: Comparación de Desempeño de Equipos en *RandomEasy*

El cuerpo de bomberos alcanzó un 100 % de efectividad en su tarea y en las 20 corridas apagaron todos los incendios que se presentaron en cada simulación. Por otra parte el equipo de ambulancias mantuvo vivos al 70 % de los agentes.

Capítulo 6

Conclusiones y Recomendaciones

6.1. Conclusiones

Se implementó una aproximación a la solución al problema de *Robocup Rescue* compuesta por los siguientes elementos:

- Esquema de coordinación distribuida para la búsqueda de víctimas civiles.
- Esquema de coordinación centralizado para el rescate de víctimas.
- Esquema de coordinación centralizado para la extinción de incendios.
- Agrupamientos de edificios para facilitar la planificación en la extinción de incendios.
- Técnicas de aprendizaje evolutivo como medio de soporte para la toma de decisiones, específicamente, un sistema de clasificadores genéticos XCS.
- Algoritmos de búsqueda informada en grafos y agrupamiento de calles para planificación de rutas.
- Arquitectura de subsunción como modelo de comportamiento individual de los agentes.
- Sistema de cooperación entre agentes heterogéneos, que permite que los bomberos, ambulancias y policías trabajen para lograr metas comunes.
- Protocolo de comunicación para maximizar el uso de los canales y priorizar el flujo de la información.

El diseño formulado demostró ser efectivo para la resolución del problema y presenta un alto nivel de competitividad con otros equipos, obteniendo puntajes similares a los equipos finalistas del campeonato mundial del 2004. Con un poco más de desarrollo los agentes

objetos de este estudio podrían estar preparados para realizar una participación exitosa en la próxima edición de *Robocup Simulation League*.

A pesar de que la selección de los parámetros de los sistemas de clasificadores de selección de incendio no fue muy apropiada, específicamente el conjunto de reglas fue reducida con respecto al número de acciones posibles. El uso de las técnicas de aprendizaje evolutivo son adecuadas para el problema de *Robocup Rescue*.

6.2. Recomendaciones y Direcciones Futuras

Sería conveniente realizar un estudio en profundidad sobre la elección de los distintos parámetros del sistema de clasificadores genéticos, así como también, una prueba de distintos operadores de cruce y mutación que se ajusten mejor a la estructura de los cromosomas del problema.

Agregar un modelo probabilístico que permita optimizar la búsqueda de civiles en la ciudad estableciendo un sistema de prioridades para los edificios dependiendo de la probabilidad de que una víctima se encuentre atrapada en su interior, como lo hace *ResQ Freiburg* [Kleiner et al., 2004].

Implementación de un sistema de planificación de caminos con memoria en donde se almacenen algunos caminos en un *cache* para evitar la repetición de cálculos innecesarios.

Sería interesante la creación de equipos de trabajos heterogéneos compuesto de agentes de varios tipos, por ejemplo, asignar a un grupo de ambulancias un grupo de policías que les sirva de escolta y recorran la ciudad con ellos para desbloquear las vías necesarias.

Extender las funcionalidades del sistema de aprendizaje, asignando la toma de una mayor cantidad de decisiones al XCS, como por ejemplo, el orden de rescate de las víctimas.

Bibliografía

[Bowling, 2000] Bowling, M. (2000). Agent development kit. http://www-2.cs.cmu.edu/~mhb/research/rescue. Disponible en: http://www-2.cs.cmu.edu/~mhb/research/rescue.

[Brooks, 1991] Brooks, R. A. (1991). How to build complete creatures rather than isolated cognitive simulators.

[Dijkstra, 1959] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271.

[Eker y Akin, 2004] Eker, B. y Akin, H. L. (2004). Roboakut 2004 rescue team description. RoboCupRescue simulation league.

[Holland, 1975] Holland, J. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press.

[Holland, 1986] Holland, J. H. (1986). *Machine learning an artificial intelligence approach volume II*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Honji et al., 2004] Honji, K., Furuta, S., y Ito, N. (2004). A consideration for cooperative behavior with heterogeneous agents'communication. RoboCupRescue simulation league.

[Kleiner et al., 2004] Kleiner, A., Brenner, M., Bräuer, T., Dornhege, C., Göbelbecker, M., Luber, M., Prediger, J., y Stückler, J. (2004). Resq freiburg: Team description paper and evaluation. RoboCupRescue simulation league.

[Kleiner y Göbelbecker, 2005] Kleiner, A. y Göbelbecker, M. (2005). Robocup rescue 3dviewer web site. Disponible en: http://kaspar.informatik.uni-freiburg.de/~rescue3D/.

[Kuwata, 2005] Kuwata, Y. (2005). Kuwata robocup rescue viewer web site. Disponible en: http://homepage1.nifty.com/morecat/Rescue/Rescue.html.

[Levy,] Levy, S. D. Simon levy's kdtree library. Disponible en: http://www.cs.wlu.edu/~levy/.

[Llona, 2004] Llona, G. G. (2004). Aproximación a robocup rescue simulation a partir de planificación central y síntesis de elementos espaciales agregados. Proyecto de Grado, Universidad Simón Bolívar.

[Mccallum, 1996] Mccallum, A. K. (1996). *Reinforcement learning with selective perception and hidden state*. PhD thesis. Supervisor-Dana Ballard.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine Learning*, chapter 9, pages 249–273. McGraw-Hill, New York.

[Moore, 1991] Moore, A. (1991). An introductory tutorial on kd-trees. Technical Report Technical Report No. 209, Computer Laboratory, University of Cambridge, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

[Morimoto, 2002] Morimoto, T. (2002). *How to Develop a RoboCupRescue Agent*.

[Morimoto, 2004] Morimoto, T. (2004). Yabapi: Api to develop a robocuprescue agent in java. `http://ne.cs.uec.ac.jp/~morimoto/rescue/yabapi`. Disponible en: `http://ne.cs.uec.ac.jp/~morimoto/rescue/yabapi`.

[Morimoto, 2005a] Morimoto, T. (2005a). Morimoto robocup rescue viewer web site. Disponible en: `http://ne.cs.uec.ac.jp/~morimoto/rescue/viewer/`.

[Morimoto, 2005b] Morimoto, T. (2005b). Robocup-rescue simulation project overview. Disponible en: `http://www.rescuesystem.org/robocuprescue/simoverview.html`.

[Murphy, 2000] Murphy, R. R. (2000). *Introduction to AI Robotics*. MIT Press.

[Paquet et al., 2004] Paquet, S., Bernier, N., y Chaib-draa, B. (2004). Damas-rescue description paper. RoboCupRescue simulation league.

[Pearl, 1984] Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

[RCRCommitee, 2000] RCRCommitee (2000). *RoboCup-Rescue Simulator Manual*. Disponible en: `http://robomec.cs.kobe-u.ac.jp/robocup-rescue/`.

[RCRCommitee, 2004a] RCRCommitee (2004a). Robocup 2004 rescue simulation league official home page. Disponible en: `http://robot.cmpe.boun.edu.tr/rescue2004`.

[RCRCommitee, 2004b] RCRCommitee (2004b). Robocup 2004 rescue simulation league rules v1.01.

[RCRCommitee, 2005] RCRCommitee (2005). Robocup-rescue simulation project. Disponible en: `http://www.rescuesystem.org/robocuprescue/simulation.html`.

[RoboCupFederation, 2004] RoboCupFederation (2004). Robocup official site. Disponible en: `http://www.robocup.org/`.

[Rusell y Norvig, 2003] Rusell, S. y Norvig, P. (2003). *Artificial Intelligence A Modern Approach*. Prentice Hall Series in Artificial Intelligence.

[Silva y Coelho, 2004] Silva, P. T. y Coelho, H. (2004). The 5rings "team report". RoboCupRescue simulation league.

[Skinner et al., 2004] Skinner, C., Teutemberg, J., Cleveland, G., Barley, M., Guesgen, H., Riddle, P., y Zuerch, U. (2004). The black sheep team description. RoboCupRescue simulation league.

[Sycara, 2005] Sycara, K. P. (2005). Multi-agent systems. Disponible en: `http://www.aaai.org/AITopics/html/multi.html`.

[Tadokoro et al., 2000] Tadokoro, S., Kitano, H., Takahashi, T., Noda, I., Matsubara, H., Shinjoh, A., Koto, T., Takeuchi, I., Takahashi, H., Matsuno, F., Hatayama, M., Nobe, J., y Shimada, S. (2000). The robocup-rescue project: A robotic approach to the disaster mitigation problem. In *ICRA*, pages 4089–.

[Wilson, 1997] Wilson, S. W. (1997). Structure and function of the xcs classifier system. Disponible en: `http://web.tiscali.it/LCS/Papers/TUT2.pdf.zip`.

[Wooldridge, 2002] Wooldridge, M. (2002). *An Introduction to MultiAgent Systems*. John Wiley and Sons.

Apéndice A

Algoritmos Genéticos

Un Algoritmo Genético (AG) [Holland, 1975] es un sistema de aprendizaje basado en la selección natural y evolución biológica. Esta compuesto por una población de cromosomas, una función de *aptitud* que evalúa el desempeño de un individuo y un conjunto de operadores sobre los cromosomas.

Los cromosomas representan la información de un individuo, son cadenas de caracteres generalmente en el alfabeto $\{0, 1\}$.

A.1. Operadores Genéticos

Cada uno de los operadores genéticos se ejecuta con una probabilidad dada que puede influir significativamente en el desempeño del AG. Por otra parte, los operadores genéticos en sí pueden influir en gran manera sobre el resultado final que arroje el algoritmo genético. A continuacíon se describen cada uno de los operadores:

Selección Este operador se encarga de seleccionar los dos candidatos para ejecutar el operador de cruce. Generalmente esta selección está basada en la aptitud del individuo, seleccionando los dos más aptos. Una particularización de este operador es el algoritmo de selección k-Torneo. En él se selecciona k clasificadores al azar y de este conjunto aparta los dos mejores individuos basado en su aptitud.

Cruce Este operador es de suma importancia en los algoritmos genéticos, pues es uno de los encargado de diversificar la población y probar nuevas opciones. Combina los cromosomas de un par de individuos y genera dos nuevos que serán considerados parte de una nueva generación. Una instancia de este operador es el algoritmo de cruce de un punto. En él se selecciona un pivote y se hace el intercambio de las partes del cromosoma haciendo dos nuevos clasificadores como se muestra en la figura 20.

Mutación Es otro de los operadores encargados de la diversidad en la poblacion del AG.

Siguiendo la analogía de mutación biológica, se encarga de alterar la carga genética de un individuo muy levemente. La mutación de un punto es una particularización de este operador. En él se selecciona un bit aleatorio del cromosoma y se altera. Cada uno de los bits tiene la misma probabilidad de ser cambiado.

Reemplazo Despues de realizar la nueva generación de individuos, es necesario incluirlos en la población. Dado que la población de un AG es finita, si se ha llegado a su máximo se debe sustituir algunos individuos. Generalmente los individuos reemplazados son los menos aptos en la población. Entre los algoritmos de reemplazo mas comunes se encuentra el reemplazo elitista. El reemplazo elitista se encarga de buscar los individuos menos capacitados de la población basandose en la aptitud.

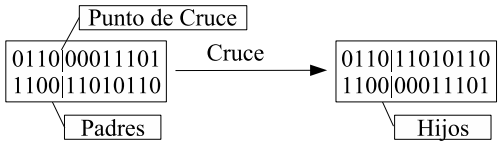


Figura 20: Ejemplo de cruce de un punto en un cromosoma

A.2. Entrenamiento de los Algoritmos Genéticos

El entrenamiento de los AG diversifica la población y permite un surgimiento de individuos más aptos en la medida en que se suceden las generaciones. Se entiende como generación una iteración del algoritmo de entrenamiento. En el algoritmo A.2.1 se presenta los pasos de un entrenamiento para un AG donde P es la población, N es el número de generaciones deseadas y m el número de cruces que se ejecutará en cada generación. Los pasos básicos que se siguen en este algoritmo son:

1. Se evalua la aptitud de los individuos de la población.

2. De la población se escojen algunos individuos que harán la función de padres para la siguiente generación mediante el operador de cruce.

3. Se le aplican los operadores de cruce a los individuos seleccionados y se generan dos nuevos individuos.

4. Se aplica los operadores de mutación en los individuos generados.

5. Los nuevos miembros son incorporados a la población y de ser necesario se seleccionan los individuos a ser sustituidos. Se repite el ciclo.

Algoritmo A.2.1: EntrenarAG()

```
for g ← 1 to N
   for each i ∈ P
      CalcularAptitud(i)
   P' ← ∅
   for h ← 1 to m
      (p1, p2) ← Seleccion(P)
      (h1, h2) ← Cruce(p1, p2)
      Mutacion(h1, h2)
      Incluir(P', h1, h2)
   Reemplazo(P, P')
```

Apéndice B

Cuadros de Atributos de Objetos de RCRSS

B.1. Cuadro de Atributos de Edificios

Atributo	**Descripción**	**Observaciones**
ID	Identificador numérico único para el edificio	
x	Abscisa del punto central del edificio en el espacio planar	medida en milímetros
y	Ordenada del punto central del edificio en el espacio planar	medida en milímetros
floors	Número de pisos del edificio	
buildingAttributes	Tipo de edificio: madera (0), estructura de acero (1) o concreto armado (2)	
buildingApexes	Coordenadas de los vértices del polígono que define la forma del edificio	
buildingAreaGround	Area de la base del edificio	medida en mm^2
buildingAreaTotal	Area total del edificio (todos sus pisos)	medida en mm^2
entrances	ID de los nodos entrada al edificio	
fieryness	Nivel de incendio del edificio: sin fuego (0), en fuego [1..3], consumido [5..7]	

Cuadro 9: Atributos de un Edificio, tomado de [RCRCommitee, 2000]

B.2. Cuadro de Atributos de Nodos

Atributo	**Descripción**	**Observaciones**
ID	Identificador numérico único para el nodo	
x	Abscisa del nodo en el espacio planar	medida en mm
y	Ordenada del nodo en el espacio planar	medida en mm

Cuadro 10: Atributos de un Nodo, tomado de [RCRCommitee, 2000]

B.3. Cuadro de Atributos de Calles

Atributo	**Descripción**	**Observaciones**
ID	Identificador numérico único para la calle	
x	Abscisa del punto central de la calle en el espacio planar	medida en mm
y	Ordenada del punto central de la calle en el espacio planar	medida en mm
head	ID del nodo de donde parte la calle	
tail	ID del nodo a donde llega la calle	
width	Ancho de la calle	medido en mm
length	Longitud de la calle	medida en mm
block	Ancho de la porción bloqueada de la calle	medido en mm
repairCost	Esfuerzo requerido para desbloquear la calle	medido en turnos agente
linesToHead	Cantidad de carriles desde el *head* hasta el *tail*	
linesToTail	Cantidad de carriles desde el *tail* hasta el *head*	

Cuadro 11: Atributos de una Calle, tomado de [RCRCommitee, 2000]

B.4. Cuadro de Atributos de Agentes

Atributo	**Descripción**	**Observaciones**
Humanoide		
ID	Identificador numérico único para el agente	
position	ID del objeto en donde se encuentra el humanoide	
positionExtra	Distancia hasta Head del calle en donde se encuentra el agente	
HP	Health Points: cantidad de vida de un civil, inicialmente 10000	
damage	Cantidad de vida que pierde un civil por turno.	
buriedness	Representa el esfuerzo requerido para rescatar al agente en caso de que haya sido inhabilitado por entierro.	medido en turnos-agente

Cuadro 12: Atributos de los Agentes,tomado de [RCRCommitee, 2000]

Apéndice C

Acciones de RCRSS

Las acciones de un agente son los comandos que de una o otra manera cambian el estado de la ciudad. Cada agente posee acciones distintas, estas se describen en el cuadro 2. Cada una de ellas junto con algunas de sus restricciones se explica a continuación:

Mover (*move*) Permite el movimiento de un agente a otro lugar siguiendo el grafo de calles de la ciudad. Esta acción se envía al kernel con la ruta que se debe seguir. El desplazamiento de los agentes se puede ver afectado por una serie de factores: carriles bloqueados, cruces, tráfico, cambios de carril, entre otros.

Rescatar (*rescue*) Acción que permite desenterrar a una víctima de un edificio derrumbado. Para lograr esto la ambulancia necesita estar en el edificio caido. En cada iteración al nivel de *buriedness* de la víctima que esta siendo rescatada se le resta una unidad tantas veces como ambulancias ejecutan esta acción sobre él.

Cargar Víctima (*load*) Permite cargar a la víctima, ya desenterrada, a la ambulancia para ser transportada al refugio. Solo las ambulancias pueden cargar a las víctima.

Descargar Víctima (*unload*) Permite que una ambulancia descargue la víctima en la posición actual.

Extinguir (*extinguish*) Acción que permite la extinción de un edificio en fuego por parte de un agente de tipo *Fire Brigade*. Según las reglas *RoboCup-Rescue Simulation Competition* del 2004 [RCRCommitee, 2004b] los tanques de agua de los bomberos tienen un máximo de 15000 litros. Mientras un agente ejecuta esta acción, serán sustraidos de su tanque de agua 1000 litros por cada turno. Se puede ejecutar mientras la cantidad de agua en el tanque del bombero sea distinta a 0. Además debe ser ejecutada a una distancia de no más de 30 metros entre el agente y el edificio en llamas.

Para la carga de agua de un bombero, es necesario dirigirse a cualquier refugio donde sera provisto de 1000 litros de agua por cada turno.

El RCRSS permite el uso de multiples mangueras por cada bombero. Sin embargo, el yabAPI [Morimoto, 2002] actual no soporta esta funcionalidad.

Limpiar Calle (*clear*) Esta acción permite a los policias quitar los escombros que estorbe el paso de agentes por cualquier calle. Para ejecutar la acción el policía debe ubicarse sobre el area afectada. Cada turno, se va aumentando en una unidad la cantidad de canales pasables en la calle.

Apéndice D

Estructura de datos: KDtree

Es una estructura de datos que permite almacenar un conjunto de puntos de k-dimensiones en un árbol binario. La idea es que el árbol divida el espacio donde se distribuyen todos los puntos en subespacios más pequeños. Cada nodo del árbol contiene las coordenadas del punto que representa, la dimensión en la cual se hace la división y los subarboles que contienen los subespacios generados [Moore, 1991]. En la Figura 21 se muestra un espacio con un conjunto de puntos y el árbol generado.

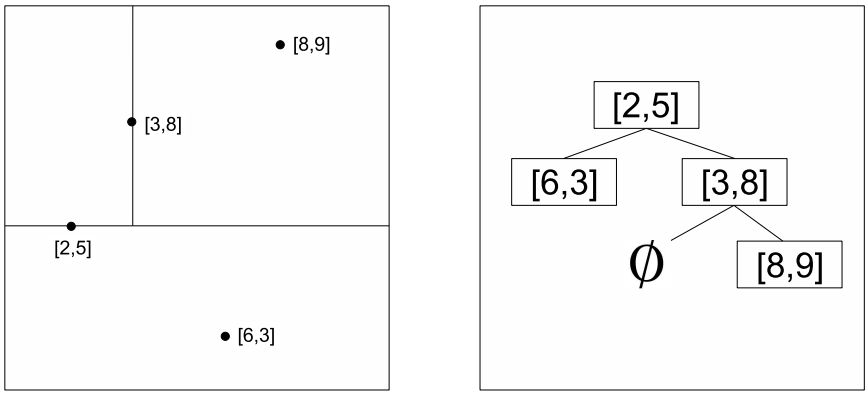


Figura 21: Espacio y subespacios de un conjunto de puntos y el kdtree generado. Se puede ver que el nodo [2,5] del árbol hace la división en el eje y y que el nodo [3,8] la hace en el eje x.

Para generar este árbol se utiliza el algoritmo D.0.1. Es necesario destacar que la selección del mejor pivote para la división del espacio es de vital importancia pues de ella depende la buena distribución de los puntos en el árbol.

Algoritmo D.0.1: CREAR-KDTREE(*puntos*: conjunto de puntos)

if ES-VACIO(*puntos*)
  return KDTREE-VACIO()
$p, eje \leftarrow$ SELECCIONAR-PIVOTE(*puntos*)
$d \leftarrow$ DOMINIO(p)
$puntos' \leftarrow puntos - \{p\}$
$r \leftarrow$ RANGO(p)
$puntos\_izquierda \leftarrow \{(d', r') \in puntos' \mid d'_{eje} \leq d_{eje}\}$
$puntos\_derecha \leftarrow \{(d', r') \in puntos' \mid d'_{eje} > d_{eje}\}$
$arbol\_izquierda \leftarrow$ CREAR-KDTREE($puntos\_izquierda$)
$arbol\_derecha \leftarrow$ CREAR-KDTREE($puntos\_derecha$)

return $\langle d, r, eje, arbol\_izquierdo, arbol\_derecho \rangle$

Entre las operaciones más utiles de *KDtree* se encuentra la búsqueda del punto más cercano a otro en el árbol. La estructura *KDtree* permite realizar una poda considerable en el árbol haciendo que la complejidad de la búsqueda sea $O(log(n))$ [Moore, 1991].

Apéndice E

Cuadro de Mensajes de Tuqueque Team

El cuadro que se presenta a continuación contiene la información de los mensaje utilizados en el protocolo de comunicación utilizado por el equipo **Tuqueque Team** en su implementación de solución al problema de *Robocup Rescue*.

Nombre	**ID**	**Tamaño**	**Descripción**
EXTINGUISH	0x00	$4 + (4 \cdot nfb)$	Orden de extinción de un incendio, contiene el identificador del edificio a apagar y los identificadores de todos los *Fire Brigade* que deben acudir al incendio. nfb es el número de bomberos enviados a realizar la tarea.
FREE	0x01	8	Reporte al *Fire Station* de que un *Fire Brigade* se encuentra libre.
POSITION	0x03	12	Informe a un agente central de la posición de un agente.

...continúa en la página siguiente...

...continuación del cuadro de la página anterior...

Nombre	**ID**	**Tamaño**	**Descripción**
ACK_EXTINGUISH	0x04	4	Mensaje de confirmación de un mensaje de EXTINGUISH.
BURNING_BUILDING	0x05	4	Información de un edificio en llamas.
UNBLOCK_ZONE	0x06	4	Petición de desbloqueo de zona, contiene el identificador de el objeto que se encuentra en el centro de la zona.
UNBLOCK_ROAD	0x07	4	Petición de desbloqueo de una calle, contiene el identificador de la calle a desbloquear.
VISITED_LONGROAD	0xAB	4	Reporte de *longroad* visitado, contiene el id del longroad en cuestión.

...continúa en la página siguiente...

...continuación del cuadro de la página anterior...

Nombre	**ID**	**Tamaño**	**Descripción**
VISITED_ROAD	0xAC	24	Reporte de *road* desbloqueado, lo envía un policía cuando ha visto o desbloqueado una calle. Contiene el identificador de la calle limpiada, su ancho, nivel de bloqueo, *linestohead*, *linestotail*, y turno en el que fue enviado el mensaje.
ORDER_UNBLOCK_AREA	0xAD	8	Orden de desbloqueo del area alrededor de un *MotionlessObject*. Contiene el identificador del agente y el identificador del *MotionlessObject*.
ACK_ORDER_UNBLOCK_AREA	0xAE	4	Confirmación de mensaje ORDER_UNBLOCK_AREA, contiene el identificador del objeto que se encuentra en el centro del área.

...continúa en la página siguiente...

...continuación del cuadro de la página anterior...

Nombre	**ID**	**Tamaño**	**Descripción**
ORDER_STOP_BUILDING	0xAF	4	Orden a un agente para que deje de explorar edificios, contiene el identificador del agente.
ACK_ORDER_STOP_BUILDING	0xB0	4	Confirmación de mensaje ORDER_STOP_BUILDING.
ORDER_UNBLOCK_ROAD	0xB3	8	Orden de desbloqueo una calle, contiene el id del agente y el id de la calle.
ACK_ORDER_UNBLOCK_ROAD	0xB4	4	Confirmación al mensaje al ORDER_UNBLOCK_ROAD, contiene el identificador de la calle que se quiere desbloquear.
ORDER_UNBLOCK_ROUTE	0xB5	12	Ordena a un policia que desbloquee una ruta desde un *motionless* a otro *motionlessobject*. Contiene el id del agente , el id del origen y el id del destino.

...continúa en la página siguiente...

...continuación del cuadro de la página anterior...

Nombre	**ID**	**Tamaño**	**Descripción**
ACK_ORDER_UNBLOCK_ROUTE	0xB6	8	Confirmación al mensaje ORDER_UNBOCK_ROUTE, contiene el id del origen y el id del destino.
ORDER_COMMON_ROADS	0xB7	4	Ordena a un policía que desbloquee los *longroads* más populares, contiene el id del policia al que va dirigido la orden.
ACK_ORDER_COMMON_ROADS	0xB8	0	Mesaje de respuesta al ORDER_COMMON_ROADS.
VICTIM	0x5A	20	Reporte de haber encontrado una víctima, con toda su información. Los datos que contiene el mensaje son: *posicionId*(int), *victimId*(int), *victimBuriedness*(int), *victimDamage*(int), *victimHp*(int).

...continúa en la página siguiente...

...continuación del cuadro de la página anterior...

Nombre	**ID**	**Tamaño**	**Descripción**
VISITED_BUILDING	0x5B	4	Reporta si un edificio esta en llamas. Los datos que contiene son buildingID.
FREE_AGENT	0x5D	0	Mensaje que indica que el emisor esta libre.
RESCUE	0x5E	$12 + 4 \cdot nat$	Orden de rescate a una víctima, se envía el identificador de la víctima, su posición, el identificador del coordinador de rescate. *nat* es el número de ambulancias enviadas a realizar la tarea.
ACK_RESCUE	0x5F	4	Mensaje de confirmación del mensaje RESCUE, se envía el identificador de la víctima.
RESCUED	0x60	4	Mensaje que indica que una víctima ha sido rescatada exitosamente, contiene el identificador de la víctima.

...continúa en la página siguiente...

...continuación del cuadro de la página anterior...

Nombre	**ID**	**Tamaño**	**Descripción**
UNBLOCK_ROUTE	0x61	8	Orden de desbloqueo de una ruta. Se envían los identificadores del inicio y del final.
PRIORITY_RESCUE	0x62	8	Reporte de agente atrapado,se envía el identificador del agente y su posición.
UNSUCCESSFUL_RESCUE	0x63	4	Mensaje que indica el fracaso de una operación de rescate, contiene el identificador de la víctima.
UNRESCUABLE_VICTIM	0x64	4	Mensaje que indica que la víctima no puede ser rescatada por encontrarse en un edificio en llamas.
RESCUE_AGENT	0x65	$8 + (4 \cdot nat)$	Orden de rescate a un agente pelotón, se envía el identificador del agente, su posición y los identificadores de las ambulancias que deben acudir a su rescate.

...continúa en la página siguiente...

...continuación del cuadro de la página anterior...

Nombre	**ID**	**Tamaño**	**Descripción**
ACK_RESCUE_AGENT	0x66	4	Confirmación del mensaje RESCUE_AGENT, contiene el identificador del agente.
FREE_TIME	0x67	4	Mensaje para que un agente se reporte libre en determinado momento de la simulación, contiene el turno en el cuál se encuentra la simulación.

Cuadro 13: Cuadro de Mensajes de **Tuqueque Team**

Apéndice F

Problemas Encontrados

En este apéndice se presenta una serie de problemas encontrados en en el transcurso del desarrollo de los agentes del **Tuqueque Team**.

F.1. Algoritmo de Visión del Kernel

En ocasiones es posible que un agente tenga un objeto dentro de su campo de visión pero el kernel no envía los datos de éste como información sensorial visual. Con frecuencia ocurre que la posición de un agente es un objeto que no puede ver. A pesar de que está bien documentada la problemática del rango de visión como se explica en la figura 2 del capítulo 2, los desarrolladores del kernel del sistema indican que el algoritmo que detecta los elementos observados tiene errores.

Estas inconsistencias del sistema se tomaron en cuenta en la implementación de los agentes para que éstos no perdieran tiempo esperando por información sensorial que probablemente nunca sea recibida por errores externos.

F.2. Problemas con *LinkedList*

Se presentaron problemas con la estructura *LinkedList*. Durante la implementación se utilizó este contenedor como cola de prioridades en el planificador de caminos, el actualizador de incendios y el manejador de *Tokens*. Se detectó que al insertar y eliminar muchos elementos de un *LinkedList* tiene efectos secundarios en la extracción del primer elemento. Se pudo observar cómo el tiempo de ejecución de esta simple operación aumentaba exponencialmente.

F.3. Votaciones de los Bomberos

Para la selección del mejor edificio a apagar en un incendio se intentó implementar un sistema votaciones de bomberos para cada uno de los edificios. Un bombero vota basandose

en la alcanzabilidad de los edificios desde donde se puede apagar el edificio evaluado. Así, al momento de actualizar el AEF de cada bombero, se verifica cual edificio es más cercano a cada compañero y se le da un voto. Es importante destacar, que este proceso se hacía en cada uno de los bomberos y no requería comunicación entre ellos.

El cálculo de este factor requiere una gran cantidad de búsqueda de rutas lo cual es muy lento. Tomando en cuenta las limitaciones de tiempo para emitir una acción (sección 2.3.3) no era posible hacerlo en el tiempo adecuado. Por esta razón fue eliminada esta funcionalidad como medidor de cercanía a un edificio y sustituida por el cálculo de distancias euclidianas, que resultó ser tan eficaz como las votaciones pero más rápida.

Apéndice G

Manual de Uso para el Script de Arranque

El script de arranque de *Robocup Rescue Simulation System*, llamado *rcrss-launcher*, tiene como objetivo iniciar el kernel y todos los sub-simuladores automáticamente, permitiendo que cada uno de éstos sean ejecutados en máquinas diferentes. El script de arranque está compuesto por otros scripts: uno maestro que determina los parámetros de la simulación, ejecutando los módulos necesarios y un script por cada sub-simulador del sistema. También cuenta con otras funcionalidades como detener la simulación y verificar que todos los módulos del sistema están ejecutándose.

G.1. Requerimientos

Los requerimientos de software de *rcrss-launcher* se presentan detalladamente en el cuadro 14.

Requerimiento	**Observación**
openssh	Esencial para establecer conexiones remotas con otros computadores. Tanto el cliente como el servidor son necesarios.
procps	Incluido en los paquetes básicos de todas las distribuciones GNU/ Linux, permite monitoreo y control de procesos
Python	Necesario para la ejecución de todos los scripts, sólo ha sido probado con versiones mayores a 2.3
java-sdk	Necesario para la ejecución de algunos módulos como el trafficsimulator, se recomienda la implementación de Sun y la versión 1.4
rescue simulator	El simulador de *Robocup Rescue Simulation System* incluye los programas de todos simuladores. Solamente ha sido probado con la versión 0.45

Cuadro 14: Requerimientos de software para el script *rcrss-launcher*

Además se recomienda que todas las máquinas donde corren los simuladores compartan la misma estructura de archivos y sistema de usuarios, para facilitar las conexiones remotas y configuración del script.

G.2. Funcionamiento

Para iniciar el sistema el script *rcrss-launcher* inicialmente se lee un archivo de configuración y a partir de éste obtiene los parámetros de ejecución de los sub-simuladores.

Para cada sub-simulador se realiza una conexión a un computador mediante *Secure Shell (ssh)* y en ésta ejecuta al sub-simulador. Una vez que se han ejecutado todos los módulos del sistema el script *rcrss-launcher* verifica que cada uno de éstos esté corriendo normalmente. En caso de fallas se reinician los módulos necesarios. Después de este paso el script termina su ejecución.

Si el usuario lo indica, se puede verificar de nuevo el estado de la simulación, indicando si algún simulador ha fallado.

Finalmente el usuario puede ordenar que la simulación se detenga. En este caso el script se conecta de nuevo a todos los computadores donde corren los sub-simuladores y detiene la ejecución de cada uno.

G.2.1. Modo de uso

La sintaxis para utilizar el script es la siguiente:

```
python rcrss-launcher [-c <configuración>] [--kill] [--check-running]
```

Donde `configuración` es el archivo de configuración del script. La opción `--kill` indica que se desea terminar la simulación. La opción `--check-running` permite verificar si todos los módulos del sistema están ejecutándose correctamente.

G.3. Archivo de Configuración

El archivo de configuración de *rcrss-launcher* permite la definición de variables específicas para cada módulo de simulación. Esta compuesto de un área global donde se definen las

variables que pueden compartir todos los módulos y un área local para cada sub-simulador. Un ejemplo simple del archivo de configuración se presenta en la figura 22.

La versión actual del script contiene un archivo de configuración de ejemplo que contiene la información necesaria para correr la simulación y contiene información de ayuda que explican cada opción y variable de éste.

```
RCRDIR      = ~/rescue
LOG         = logs/${TEAM_NAME}-${MAP}-c${CORRIDA}.log
RCRCONFIG   = robocup2004.rules
MAP         = Foligno
MAPDIR      = ${RCRDIR}/maps/${MAP}
KERNEL_HOST = localhost
TEAM_NAME   = tuqueque

[0gis]
runnable = yes
script   = 0gis.py
script_options = -c rcrss_launcher.conf
program  = ${RCRDIR}/program/gis/gis
shell    = xterm
machine  = ${KERNEL_HOST}
map_options = -logname ${LOG} \
              -shindopolydata ${MAPDIR}/shindopolydata.dat \
              -galpolydata ${MAPDIR}/galpolydata.dat
options  = -file ${RCRCONFIG} ${map_options} -mapdir ${MAPDIR}/
           -gisini ${MAPDIR}/gisini.txt
```

Figura 22: Ejemplo de configuración para el script *rcrss-launcher*