LUIS FELIPE ZAPATA RIVERA

SISTEMAS MULTIAGENTE COLABORATIVOS

Proyecto presentado a la Universidad
Eafit para la obtención del título de
Ingeniero de Sistemas

Área de concentración:
Inteligencia Artificial

Asesor:
Ph.D. Juan Diego Zapata Rivera

Medellín 2010

RESUMEN

Este proyecto examina el área de los sistemas multiagente colaborativos. El proyecto describe seis simulaciones de sistemas multiagente. Cada simulación representa un tipo de esquema de colaboración y comunicación diferente. Los resultados de las simulaciones son analizados y comparados. El proyecto también describe la implementación en hardware de un prototipo de sistema multiagente correspondiente a una de las simulaciones utilizando la plataforma Lego MindStorms.

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

CAPITULO 1

INTRODUCCIÓN

1.1 Objetivos y motivación del proyecto

Los sistemas multiagente han sido aplicados en diferentes campos como por ejemplo en la programación de robots para la exploración de terrenos hostiles (ej., campos minados) [1,2], para realizar operaciones de rescate [3,4], para explorar otros planetas [5], entre otros.

El objetivo de este proyecto es dar a conocer el estado del arte y reportar los resultados de varios algoritmos para la colaboración entre agentes. Los algoritmos han sido implementados en la herramienta de software *NetLogo*. Adicionalmente, el articulo presenta la implementación de un prototipo de un sistema de agentes colaborativos en la plataforma *Lego MindStorms.*

1.1.1. Objetivo General

Conocer el estado del arte y evaluar varios algoritmos para la colaboración entre agentes.

1.1.2. Objetivos Específicos

- Desarrollar varias simulaciones que puedan servir como material educativo en la universidad para facilitar e ilustrar el desarrollo de agentes colaborativos.
- Implementar un demo de un sistema de agentes colaborativos en *Lego* usando recursos disponibles.

Los productos del proyecto son seis simulaciones en la plataforma *NetLogo*, la implementación de un demo prototipo en *Lego MindStorms,* dos videos del demo prototipo en *Lego*, 6 videos tutoriales con pruebas de cada simulación en *NetLogo* y un artículo que presenta los resultados del proyecto. Todo este material y el presente informe se incluyen en un CD. La implementación en hardware de todas las simulaciones esta fuera del alcance de este proyecto.

1.2. Impacto esperado

Los beneficios potenciales de este proyecto incluyen:

- Este proyecto contribuye al continuo desarrollo del área de la Inteligencia Artificial IA y robótica en la universidad EAFIT.
- Apoya los temas que se dictan en cursos relacionados con la IA en la universidad, ya que los resultados del proyecto podrán servir de punto de partida para los estudiantes que deseen profundizar en este campo.
- Facilita la participación de la universidad en competencias nacionales e internacionales de IA, gracias a la implementación de herramientas que permitan el avance en esta área.
- Posible presentación de los resultados del proyecto en una conferencia nacional o internacional.

1.3. Descripción de los capítulos del proyecto

El capítulo 1 describe los objetivos, motivación e impacto del proyecto. El capítulo 2 presenta las bases teóricas para la implementación de los sistemas multiagentes y describe varias plataformas para la implementación de agentes. El capitulo 3 muestra las características de la plataforma *NetLogo*. El capítulo 4 describe una serie de simulaciones que representan varios esquemas de colaboración. Este capítulo incluye el análisis de resultados. El capitulo 5 presenta una descripción detallada de la simulación realizada en la plataforma *Lego MindStorms*. El capítulo 6 presenta las respectivas conclusiones del trabajo y las sugerencias para trabajos futuros.

CAPITULO 2

MARCO TEÓRICO Y ESTADO DEL ARTE

2.1. Inteligencia artificial

La inteligencia artificial (IA) se basa en la idea que una máquina pueda simular comportamiento inteligente, principalmente reproduciendo la función de toma de decisiones adecuadas como consecuencia a unos estímulos e información del medio [6].

Autores como *Russell* [6] han escrito acerca del tema de la inteligencia artificial manifestando la dificultad que tiene el poder representar de manera eficaz todas las características de la inteligencia humana en un sistema, lo que se propone entonces es facilitar el trabajo dividiendo el sistema en subsistemas independientes sobre los cuales se pueda trabajar y profundizar.

En este enfoque de especialización de tareas existen productos en robótica, algunos comerciales y otros para uso de instituciones gubernamentales, que han tenido gran reconocimiento mundial, a continuación se muestran algunos de estos.

La Inteligencia artificial tiene aplicaciones en áreas como el desarrollo de software, educación personalizada, aeronáutica, medicina, aplicaciones militares, robótica entre otros, desde hace más de dos décadas [6], teniendo como pionero la aplicación militar.

En la figura 2.1, se muestra un grupo de robots con el objetivo compartido de levantar y desplazar una placa metálica a un sitio determinado.

En la Figura 2.2. Se presenta un grupo de Robots de uso militar especializado en la detección de minas en campos y terreno de difícil acceso, peligrosos para los seres humanos.

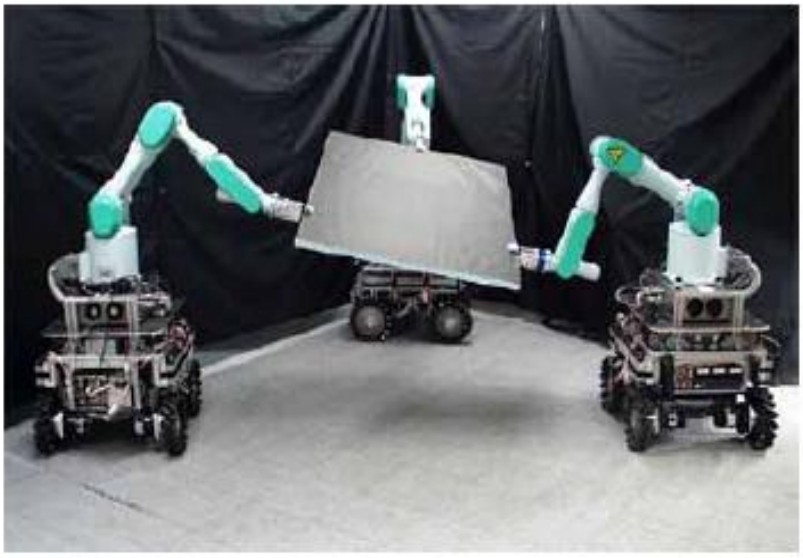

Figura 2.1. Trabajo cooperativo entre robots [7]

Figura 2.2. Robots Militares para la detección de minas [7]

Dentro de la inteligencia artificial se tratan temas como [8]:

- Aprendizaje Automático (*Machine Learning*)
- Ingeniería del conocimiento (*Knowledge Engineering*)

- Sistemas reactivos (*Reactive Systems*)
- Sistemas multiagente (*Multi-Agent Systems*)
- Sistemas basados en reglas (*Rule-Based Systems*)
- Visión artificial
- Audición artificial
- Lingüística computacional
- Procesamiento del lenguaje natural (*Natural Language Processing*)

En este proyecto se profundizó en el tema de sistemas multiagente colaborativos.

2.2. Sistemas multiagente

Un sistema multiagente es un sistema distribuido en el cual los nodos o elementos del sistema son agentes. En estos sistemas la conducta combinada de los agentes produce un resultado inteligente. Los sistemas multiagente se preocupan por coordinar la conducta inteligente de agentes autónomos. Estos agentes hacen parte de una colección y pueden coordinar o compartir su conocimiento, objetivos, habilidades y planes para tomar una acción o resolver una meta global.

Los sistemas multiagente han sido aplicados en varios campos. Por ejemplo, en educación se ha aplicado en la construcción de sistemas de tutoriales inteligentes [ej., 9, 10]. La clasificación de los agentes de software se presenta en la figura 2.3.

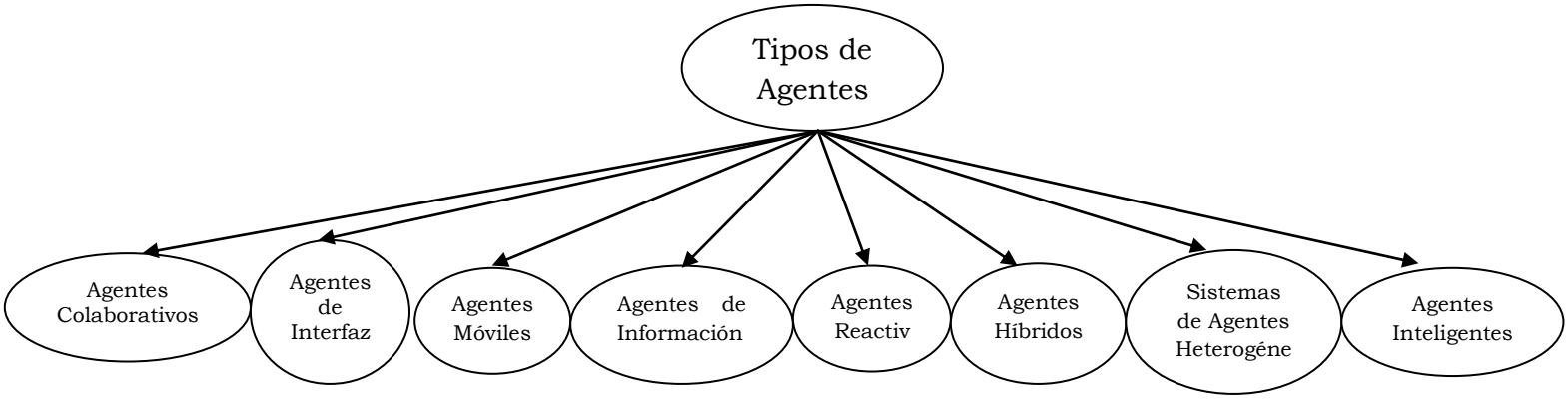

Figura 2.3. Clasificación de los agentes de software [11]

- Los agentes colaborativos

Los agentes colaborativos basados en objetivos son agentes autónomos que colaboran con otros agentes para cumplir sus objetivos.

- Agentes de Interfaz

Los agentes de interfaz se implementan para facilitar el uso de las aplicaciones de software y proporcionan ayuda, generalmente a usuarios que están aprendiendo a utilizar una aplicación concreta. Los agentes de interfaz aprenden típicamente a mejorar su ayuda al usuario de cuatro maneras [11]:

1. Observando e imitando al usuario, es decir, aprendiendo del usuario.
2. Con la recepción de *feedback* positivo y negativo del usuario (que aprende del usuario).
3. Recibiendo instrucciones explícitas del usuario que aprende del usuario.
4. Pidiendo consejo a otros agentes, es decir, aprendiendo de pares.

- Agentes Móviles

Los agentes móviles pueden ejecutarse y moverse por diferentes nodos de una red, su objetivo es ejecutar tareas e interactuar con otros agentes a través de mensajes. Ventajas de los agentes móviles incluyen: eficiencia, adaptación al cliente, reducción del tráfico de la red, gestión de gran volumen de información y permiten la comunicación en tiempo real.

- Agentes de Información

En la definición de agentes de información se encuentran la mayoría de los tipos de agentes, lo importante es que estos agentes implementan funcionalidades que apoyan la localización, recuperación y organización de información.

- Agentes Reactivos

Los agentes reactivos se caracterizan por no utilizar conocimiento previo ni representaciones del medio en el que interactúan, su comportamiento es consecuencia a un estímulo del medio mediante primitivas almacenadas en su código.

- Agentes Inteligentes

Un agente inteligente se puede definir como una entidad capaz de percibir su entorno, procesar tales percepciones y responder o actuar en su entorno de manera racional y tendiendo a lograr un resultado esperado.

Otras clasificaciones:

Existen varios criterios de clasificación para los agentes de software. Agentes deliberativos y reactivos, los primeros son agentes que tienen un previo conocimiento de su entorno y los reactivos son agentes que no poseen ningún conocimiento de su ambiente o entorno. La última clasificación consiste en agentes colaborativos o cooperativos, agentes de aprendizaje (*learning*) y agentes autónomos [8].

2.3. Aplicaciones de sistemas multiagente en robótica

En robótica se usan los sistemas multiagente en contextos como la planeación de rutas, prevención de obstáculos, arquitecturas de comunicación, metodologías de cooperación, aprendizaje artificial y control inteligente.

Algunos ejemplos de la vida real de sistemas multiagente aplicados en robótica son:

- Competencia de fútbol simulado en computador [ej., 13, 14]
- Robots de sumo (sacar del campo a otro robot) [ej., 15, 16]
- El Robot que se muestra en la figura 2.4, participa en la Competencia mundial de fútbol de robots humanoides. Este tipo de robot se caracteriza por implementar muchas de las características del cuerpo humano, además están programados con agentes colaborativos (*Robocup* y FIRA) [ej., 13, 14].

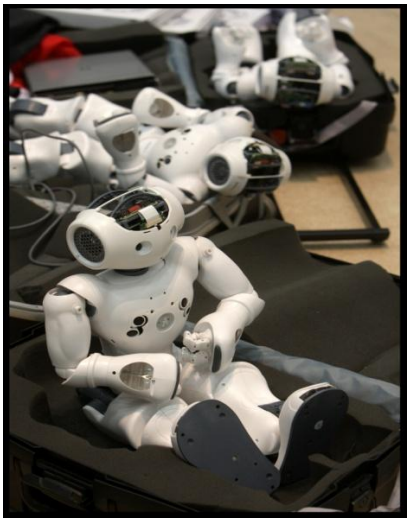

Figura 2.4. Robot Humanoide jugador de fútbol [13, 14]

Figura 2.5. Competencia de fútbol de robots mascotas AIBO de Sony [13, 14]

2.4. Comunicación entre agentes

La comunicación entre agentes se basa en protocolos o esquemas de intercomunicación, entre los protocolos más conocidos se encuentran *Knowledge Query Manipulation* Lenguaje (KQML) y (Agent Comunication Language) ACL.
KQML es un lenguaje y protocolo para comunicar agentes, fue creado a principios de los años 90 como parte del DARPA *knowledge Sharing Effort*. El KQML se puede utilizar para interactuar con un sistema inteligente, ya sea por un programa de aplicación, o por otro sistema inteligente [17].
FIPA-ACL *Agent Communication Language* fue propuesto por FIPA (*The Foundation for Intelligent Physical Agents*), y es un lenguaje estándar para la comunicación entre agentes, fue el sucesor del KQML [18].

2.5. Implementación de agentes en software y hardware

La implementación de agentes de software puede ser realizada con la ayuda de modelos de software como BDI (*The Belief-Desire-Intention*) [19] que es un modelo de software desarrollado para la programación de agentes inteligentes. En general esta arquitectura se caracteriza por

permitir la aplicación de las creencias, deseos e intenciones en los agentes.

Para implementar sistemas multiagente en software se deben tienen en cuenta 3 elementos:

- Autonomía
  Realizar una separación en procesos o hilos
  Comunicación con otros programas
  Mecanismos para procesamiento de eventos
- Inteligencia
  Codificación procedimental
  Lógica orientada a objetos
  Razonamiento sofisticado y habilidades de aprendizaje
- Movilidad
  Por ejemplo a través de *bytecodes* portables y archivos en *Java* (JAR) [20]

2.6. Plataformas para la implementación de agentes

Existen varias plataformas para la implementación de agentes (ej., JADE, Jess, IBM's *Robocode* [21], *JavaLog*, FIPA OS, *Voyeguer* ORB), y varios protocolos para realizar la comunicación.

La plataforma JADE [22] es una de las más usadas y conocidas, esta herramienta permite realizar la implementación de sistemas multiagente, permite la coordinación de múltiples agentes FIPA y proporciona una implementación estándar del lenguaje de comunicación FIPA-ACL.

En la etapa inicial del proyecto, se instalaron y ejecutaron varias plataformas de sistemas multiagentes incluida la plataforma JADE [23]. Esto incluye la configuración del equipo de computo para tener un funcionamiento correcto de dicha plataforma (ver anexo – figura 3), luego se realizaron varias pruebas del comportamiento de sistemas multiagente, visualizando los diferentes agentes y la comunicación (el paso de mensajes) entre ellos (ver anexo – figura 1) mediante herramientas de monitoreo (*sniffer*) incluidas en la plataforma (ver anexo – figura 2).

- Plataformas para implementar agentes inteligentes en *Lego MindStorms*

Igualmente, se realizó un análisis de plataformas que pudieran facilitar la implementación de agentes que puedan ser usados en el sistema *Lego MindStorms* y se encontró que en el lenguaje *Prolog* (lenguaje de programación lógico e interpretado, usado en el medio de investigación en Inteligencia Artificial), se pueden implementar sistemas de agentes, y mediante un software adicional se puede transformar el código generado al lenguaje nativo del sistema *Lego MindStorms*. Esta es una de las posibilidades que se tienen para la realización de implementaciones de sistemas multiagente para dicha plataforma [24].

- Herramientas para visualizar Agentes en JADE

Adicionalmente se trabajo en herramientas del proyecto *Player/Stage* (*Brian Gerkey*, *Richard Vaughan* y *Andrew Howard*, 2002 [25]) que posee utilidades como Player (interfaz para dispositivos robóticos), *Stage* y *Gazebo*, que permiten la simulación de un sistema de agentes en un ambiente gráfico, *Stage* y *Gazebo* permiten visualizar robots simulados con sensores y actuadores un ambiente en 2.5d y 3d respectivamente, (ver Anexo 1, figuras 4, 5, 6 y 7).
La plataforma *Player/Stage* es de uso gratuito y soporta lenguajes como Java, c++, entre otros, *JavaClient* 1.0 y 2.0 [26] son programas que brindan una interfaz para el trabajo con lenguaje *Java* con *Player/Stage*, cabe anotar que la mayoría de estos programas solo funciona en sistema operativo Linux.

Después de analizar varias plataformas se decidió trabajar en la plataforma *NetLogo*. Esta plataforma posee herramientas que facilitan la simulación y evaluación de sistemas multiagente. El siguiente capítulo describe esta herramienta.

CAPITULO 3

SISTEMAS MULTIAGENTES EN *NETLOGO*

La herramienta *NetLogo* sirve para implementar sistemas multiagente colaborativos. *NetLogo* está basado en el lenguaje de programación logo y cuenta con una interfaz simple y potente que permite realizar una gran variedad de simulaciones y modelamiento de fenómenos, los modelos en *NetLogo* pueden tener un alto grado de complejidad y pueden contar con cientos de agentes ejecutándose concurrentemente.

NetLogo fue diseñado por *Uri Wilensky* en 1999, director de la Universidad de *Northwestern*. *NetLogo* fue escrito en *Java* y *Scala* y corre sobre la maquina virtual de *Java*. *NetLogo* funciona bajo un esquema hibrido con parte interpretada y parte compilada en JVM *bytecode*.

NetLogo hace parte de una serie de modeladores de sistemas multiagente que empezó con *StarLogo*. Nace a partir de la funcionalidad del producto *StarLogoT*, adicionando nuevas e importantes características, con un lenguaje rediseñado y con una interfaz de usuario [27].
NetLogo funciona en la mayoría de plataformas importantes (Mac, Windows, Linux, etc.). Siendo ejecutada como una aplicación independiente. Las simulaciones pueden ser corridas como *applets* de Java dentro de un navegador Web.

En *NetLogo* se tienen cuatro tipos nativos de agentes [27]:

- ***Turtles*** (tortugas).
- ***Patches*** (celdas).
- ***Links*** (relaciones entre tortugas).
- ***Observer*** (observador).

Las tortugas son los agentes que se mueven por el mundo. Interaccionan entre sí y con el medio. Cada tortuga viene identificada por un identificador que es único para cada tortuga.

NetLogo denomina “mundo” (*world*) al terreno en el que se mueven las tortugas. Cada porción cuadrada de mundo se denomina *patch*. Cada *patch* está identificado por las coordenadas de su punto central.

Las tortugas se mueven por el mundo (y, por tanto, por encima de los *patches*). Las tortugas interaccionan entre sí según unas reglas de comportamiento y con el medio (es decir, con los *patches*).

Se puede modelar la relación entre distintas tortugas mediante links, que es el tercer tipo de agente presente en *NetLogo*. Los links se designan mediante un par (tortuga1, tortuga2), que indica las dos tortugas relacionadas mediante dicho link.

Finalmente, se tiene el observador (*observer*). Éste no está representado en el mundo, pero puede interactuar con él (crea y destruye agentes, asigna propiedades a los agentes, etc.).

Cada uno de los agentes que se crean, ya sean tortugas, *patches* o *links*, tienen una serie de propiedades. Algunas de estas propiedades vienen predefinidas en *NetLogo*.

Por ejemplo, las tortugas tienen las siguientes propiedades predefinidas [22]:

Who ;; identificador (no se puede modificar)
color ;; color
heading ;; orientación
xcor ;; coordenada x
ycor ;; coordenada y
shape ;; forma
label ;; etiqueta
label-color ;; color de la etiqueta
breed ;; raza
hidden? ;; ¿visible o no visible?
size ;; tamaño
pen-size ;; tamaño del trazo al desplazarse (cuando *pen-mode=down*)
pen-mode ;; ¿dejar trazo al desplazarse o no?

Estas propiedades se usan mediante el comando "*ask turtle*" o *ask* acompañado del *breed* o tipo de agente por ejemplo:

```
ask turtles
[
  set color red
]
```

El usuario también puede agregar nuevas propiedades a los agentes las cuales se deben incluir en la sección *turtles-own* o el nombre del agente acompañado de la palabra –*own*.

El manejo de interno de los agentes se hace mediante hilos lanzado de acuerdo a la cantidad de agentes instanciados, estos hilos están asociados a un proceso principal correspondiente al programa que crea el usuario en *NetLogo*.

Es importante saber cómo realiza *NetLogo* la generación de números aleatorios, *NetLogo* utiliza el *Mersenne Twister*, generador aleatorio. Este generador es a veces llamado un generador de números "pseudo-aleatorios", ya que utiliza un algoritmo matemático para producir ciertos números que aparecen al azar, pero en realidad son predecibles dado el tiempo suficiente para averiguar el patrón [28].

El próximo capítulo presenta varios sistemas multiagente que aplican diferentes estrategias de comunicación. El objetivo final es evaluar estas simulaciones y analizar las implicaciones para futuras aplicaciones de sistemas multiagente en la robótica.

CAPITULO 4

SIMULANDO SISTEMAS MULTIAGENTES EN *NETLOGO*

Las simulaciones en este capítulo implementan varias propiedades de los sistemas multiagente como: los ciclos de vida de los agentes, esquemas de comunicación entre los agentes, trabajo colaborativo, entre otros. La interfaz de la simulación se basa en un sistema vial compuesto de vehículos (Agentes) y carreteras (calles y carreras).

El objetivo en las simulaciones es que los vehículos busquen un objeto de color rojo (Objetivo) que se debe ubicar en una de las intersecciones de calle y carrera del tablero. Los vehículos inician con un color diferente de acuerdo al caso y cambian su color a rojo una vez que encuentran el objeto buscado o reciben el mensaje de que el objeto ha sido encontrado por otro agente.

Los vehículos cuentan con varios atributos (ej., mensaje, color, ubicación actual, creencias e intenciones). Estos atributos pueden ser consultados en cualquier momento de la simulación. Las simulaciones son presentadas en orden de complejidad iniciando con el caso base.

La inteligencia de los agentes radica en la capacidad que tienen de comunicarse entre sí, en algunos casos usando las funciones para comunicaciones que posee *NetLogo* y en los dos últimos casos aprovechando la comunicación a través de mensajes FIPA-ACL. De esta manera los agentes mejoran el rendimiento de las simulaciones al lograr el objetivo y comunicarlo a los otros vehículos en un número menor de iteraciones. Los vehículos también exhiben comportamientos inteligentes en la manera en que buscan el objetivo cuando la ruta está limitada o fue modificada. La última simulación “Buscar o no buscar he ahí el dilema” incorpora nuevas capacidades que permiten que los vehículos tengan éxito en un esquema más realista con consumo de energía y la posibilidad de quedar parado en medio de la simulación. En este caso, los vehículos evalúan una función de utilidad basada en una probabilidad de seguir buscando el objeto y un premio o recompensa que se obtiene por lograr el

objetivo. De acuerdo al valor de la función de utilidad se define el comportamiento que va a tener el vehículo (creencias).
Se han implementado seis simulaciones de sistemas multiagente. En todas las simulaciones el objetivo consiste en ubicar un objeto de color rojo que se encuentra ubicado en un tablero. Este objetivo puede ser extendido a otros contextos. Por ejemplo, una tarea de búsqueda y rescate en la que varios agentes necesitan localizar un objetivo.

4.1. Estructura de la simulación

Esta sección describe la estructura general de la simulación de sistemas multiagente en *NetLogo*. Ver figura 4.1.

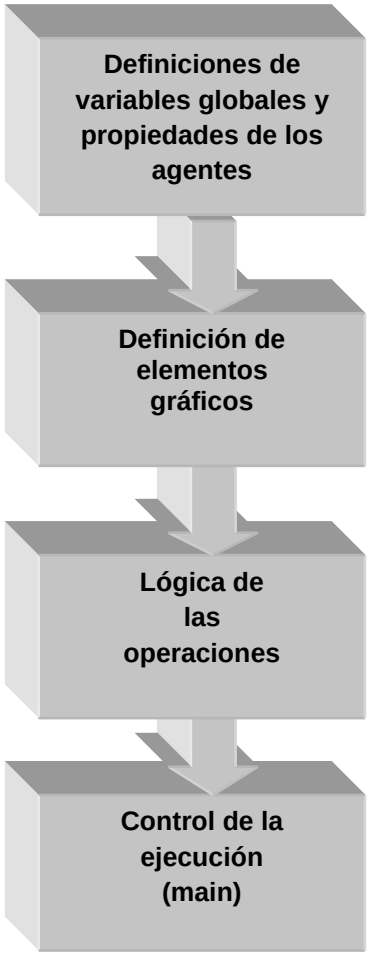

Figura 4.1. Elementos de la simulación

- Definiciones de variables globales, de agentes y propiedades de los agentes.

Aquí se definen las variables globales y los tipos de los agentes con sus características particulares. Ejemplo de definición de variables globales:

```
globals
 [
   x                ;; coordenada x
   y                ;; coordenada y
 ]
```

Ejemplo de definición del agente con sus propiedades:

```
breed [carros carro]     ;; nuevo tipo de agente o raza en NetLogo
carros-own               ;; propiedades del agente agregadas por el usuario
[
 mensaje?
 colorcarro?
]
```

- Definición de elementos gráficos

En esta parte del programa se crean algunos de los elementos de la interfaz: el tablero, los agentes y paneles de configuración.

Ejemplo de creación de 5 Agentes del tipo carro.

```
to setup-cars                    ;; creación y configuración de los carros
   create-carros 5               ;; crear los carros
    [
      set color green
    ]
 end
```

- Lógica de las operaciones

En esta sección están implementadas las funciones que dan dinámica al programa como las funciones del movimiento de los agentes, validación de reglas, comunicaciones etc.

Ejemplo de función que cambia el color de los agentes de color verdes por color rojo:

```
to repintar                    ;; nombre de la función

  if (color = green)
   [
      set color red
   ]
end
```

- Control de la ejecución (*main*)

Aquí se controla el orden de la ejecución de las partes del programa, el nombre de la función principal es empezar. La ejecución de esta función está asociada a la acción de un botón que se crea en la interfaz grafica, este botón como todos los botones de *NetLogo* puede configurarse para ejecución continua (como un ciclo continuo) o para realizar una sola iteración del código de la función.

Ejemplo de función empezar (*main*)

```
to empezar
  [
      ask carros
      [
        mover
       ]
   tick
  ]
end
```

4.2. Casos de simulación

En el proyecto se han implementado seis simulaciones de sistemas multiagente. En todas las simulaciones el objetivo consiste en ubicar un objeto de color rojo que se encuentra ubicado en un tablero. Este objetivo puede ser extendido a otros contextos. Por ejemplo, una tarea de búsqueda y rescate en la que varios agentes necesitan localizar un objetivo.

Estas simulaciones van incrementando su nivel de complejidad partiendo del primer caso llamado "Todos Buscamos" en el cual los vehículos no se comunican entre ellos; un segundo caso llamado "Te Cuento que Ya Terminamos", el cual implementa un esquema de comunicaciones básico entre los vehículos el cual consiste en que cada vehículo que tiene la facultad de comunicar a otros vehículos que el objeto rojo ha sido encontrado, esta comunicación solo es posible realizarse cuando un vehículo se encuentra en frente a otro (ej., comunicación por contagio); el tercer caso llamado "Les Cuento a Todos que Ya Terminamos", que permite una comunicación global de la información, es decir que todos los vehículos se enteran que el objeto fue encontrado al mismo tiempo (mensaje *broadcast*); el cuarto caso llamado "Nos Repartimos los Sectores y Nos Comunicamos a Través de los Nodos", este caso divide el terreno en zonas e implementa un nodo por cada zona, los cuales apoyan el proceso de comunicaciones; el quinto caso se llama "Les Cuento a Todos que Ya Terminamos usando Mensajes FIPA-ACL", que realiza básicamente lo mismo del tercer caso pero utilizando en esquema explicito de mensajes entre agentes; por último está el caso "Buscar o No Buscar, He Ahí el Dilema", que implementa nuevas propiedades a los vehículos como el consumo de energía y una función de utilidad que ayuda a los agentes a tomar decisiones respecto de moverse o no moverse en determinado momento, este caso se basa en la arquitectura BDI implementando un esquema de *beliefs* (creencias) e *intentions* (intenciones) y también hace uso de mensajes explícitos FIPA-ACL.

4.2.1. Todos buscamos (Navegación aleatoria; no hay comunicación)

Descripción:
Esta simulación es el caso base en el que se implementa un sistema compuesto de varios vehículos que buscan un objeto de color rojo en un tablero con vías (horizontales y verticales) en forma de cuadricula, el

sistema es totalmente configurable en la cantidad de vías en dirección x, cantidad de vías en dirección y, numero de vehículos y velocidad. Además permite editar y rediseñar las vías del tablero. En la parte izquierda se encuentran los controles que permiten presionando el botón crear mostrar el tablero con las características escogidas en la sección de parámetros, luego de debe usar el editor para ubicar un punto rojo en una de las calles del tablero y por últimos se debe presionar el botón empezar para dar inicio a la simulación. En este tablero cuando los vehículos llegan a un extremo, pasan al otro lado de este. Esto elimina los límites del tablero.

La política de navegación de los vehículos en las vías es totalmente aleatoria y no se implementa ningún tipo de comunicación entre los agentes (vehículos), ni se almacena información del tablero en los vehículos. La simulación finaliza cuando todos los vehículos ubican el objetivo y logran llegar a la salida ubicada en la esquina inferior derecha, si el objeto rojo es ubicado en un sitio valido (sito en color blanco) la simulación siempre termina, pues la política de navegación aleatoria de los vehículos lo garantiza. El sistema muestra en la parte de superior de la ventana la cantidad de iteraciones en *ticks* (unidad de medición de iteraciones) que toma la simulación en finalizar. El código de esta simulación aparece en el Apéndice 3, Caso 1.

La figura 4.2, muestra la situación inicial de la simulación. En este punto están creados los vehículos, el objetivo ha sido pintado y solo han transcurrido 8 iteraciones.

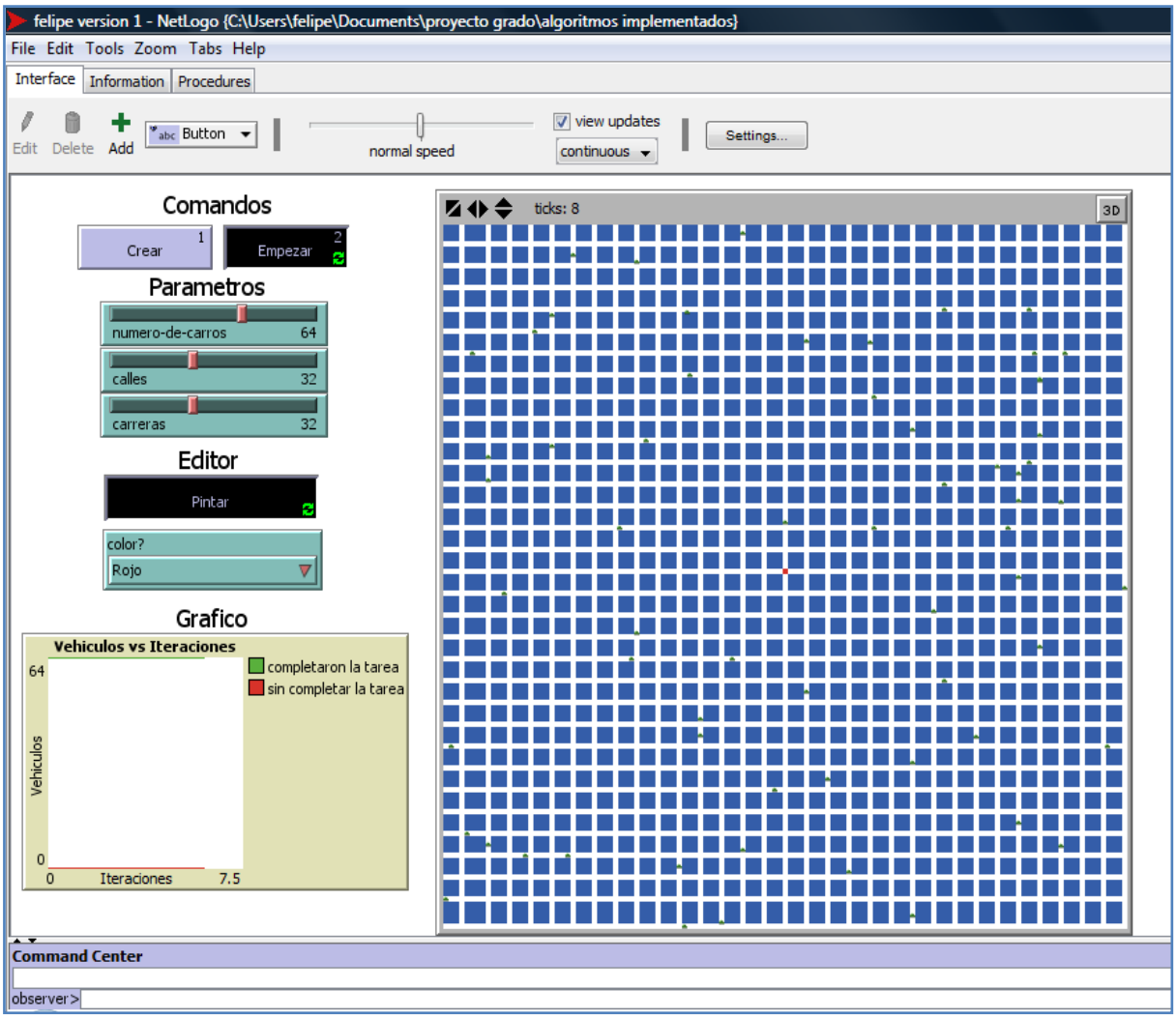

Figura 4.2. Situación inicial de la simulación Todos Buscamos.

La figura 4.3, muestra el momento en el que falta uno de los 64 vehículos por encontrar el objeto, han transcurrido 118484 iteraciones o *ticks*.

Se observa que el número de iteraciones que usan los últimos vehículos para ubicar el objeto es alto respecto al número de iteraciones que tardan los primeros vehículos en cumplir su objetivo, lo que hace pensar que es factible la implementación de un esquema de comunicaciones entre los vehículos que permita minimizar el tiempo de ejecución de la simulación. Esto se aborda en los siguientes casos.

La flecha roja señala el vehículo que falta por completar la tarea.

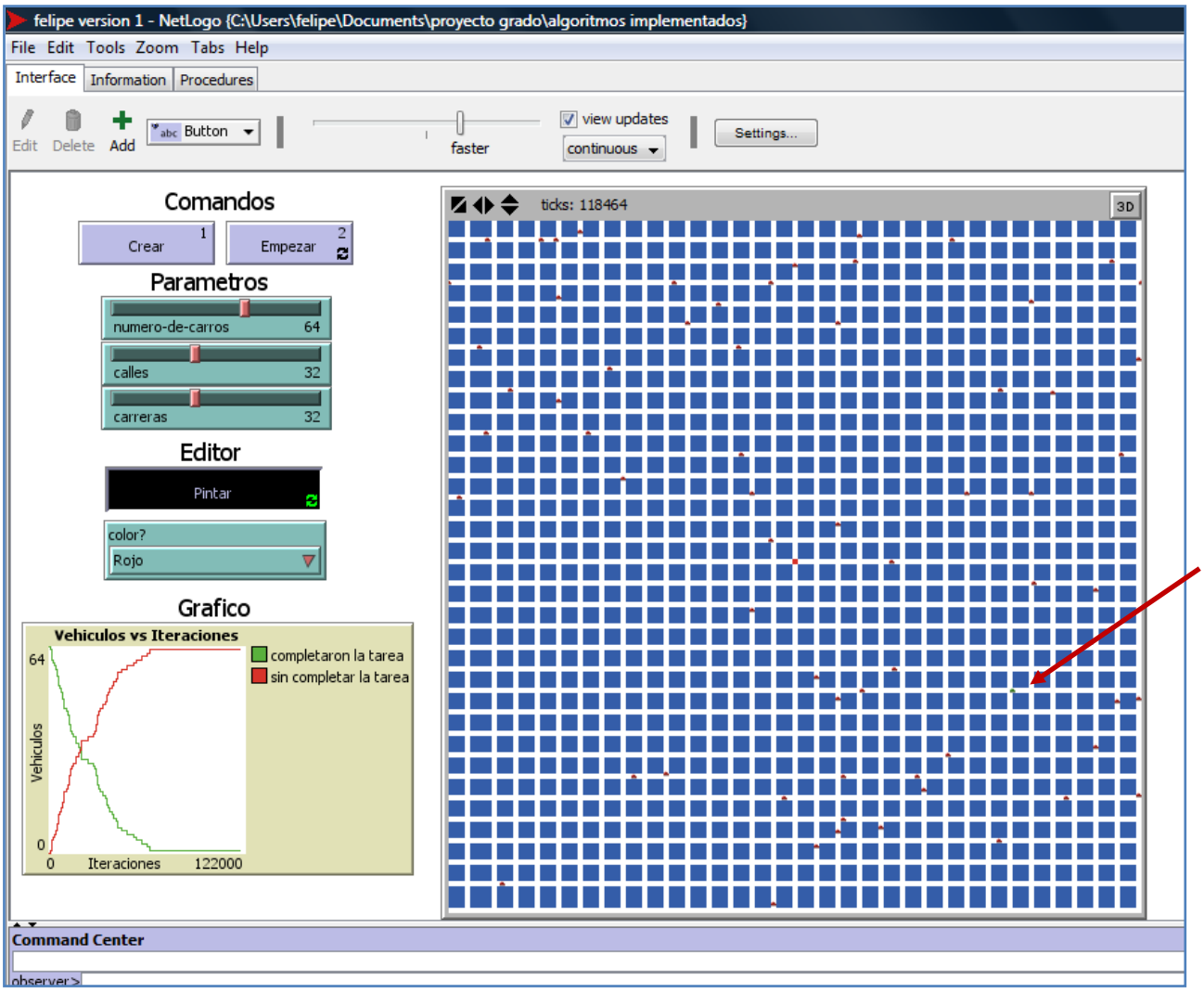

Figura 4.3. Simulación Todos buscamos por finalizar.

La figura 4.4 muestra el momento ha terminado la simulación y el programa se detiene, la grafica ubicada a la izquierda muestra el comportamiento de la cantidad de vehículos que fueron encontrando el objeto durante el transcurso de la simulación (Figura 4.4).

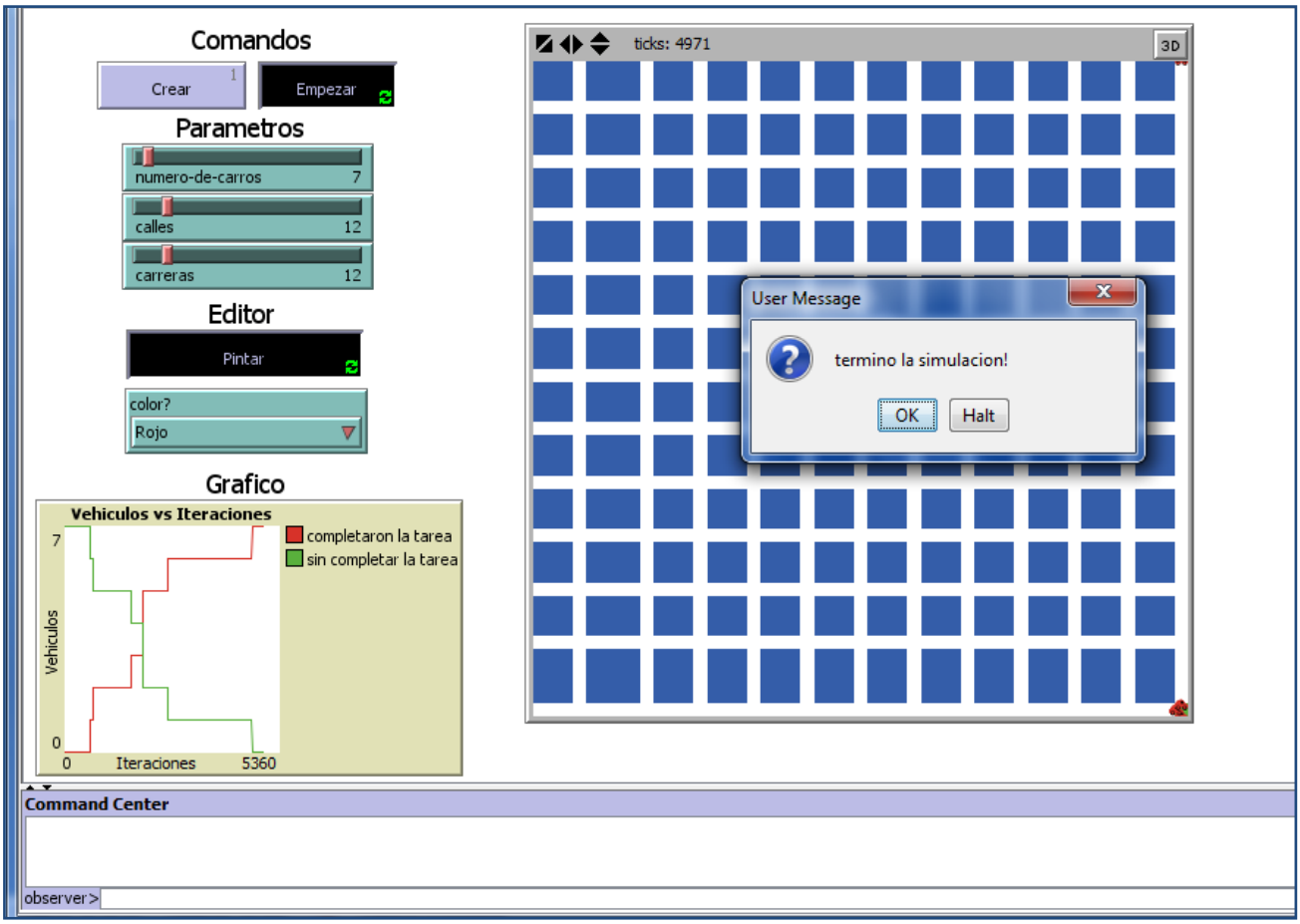

Figura 4.4. Simulación Todos buscamos Finalizada

Luego de haber ejecutado esta simulación 200 veces con 64 vehículos en un escenario 64 x 64 con 32 calles x 32 carreras y ubicando el objetivo siempre en el centro del tablero, estos son los resultados (Ver tabla 4.2):

Vehículos	Tamaño del tablero	Calles	Carreras
64	64 x 64	32	32

Tabla 4.1. Parámetros en el caso Todos buscamos.

Media (número de iteraciones)	112838.1
Desviación	23883.146
Máximo	253188
Mínimo	71067

Tabla 4.2. Estadísticas del número de iteraciones para el caso Todos buscamos

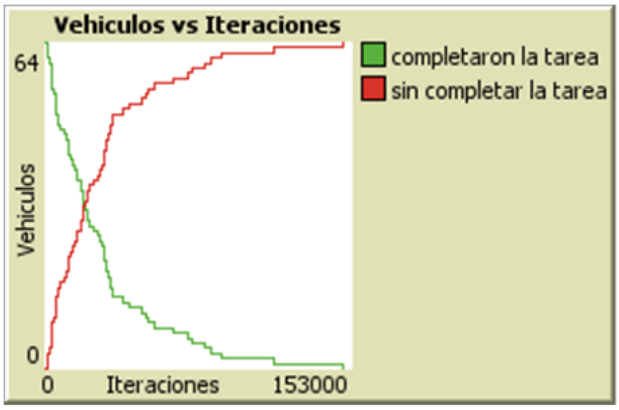

Figura 4.5. Gráfica de resultados caso Todos buscamos.

Análisis de la Gráfica.
Se realizaron los gráficos de todas las simulaciones tomando en el eje vertical el número de vehículos y en el eje horizontal el número de iteraciones (unidad de duración de la simulación).

La línea verde representa la cantidad de vehículos que no han terminado y la línea roja la cantidad de vehículos que han encontrado el objeto rojo. Todas las graficas generadas por la simulación del caso Todos Buscamos, muestran un comportamiento similar al de la figura 4.5.

En esta figura se observa que al inicio de la simulación hay un crecimiento acelerado de la cantidad de vehículos que completan la tarea (línea roja) y después de 70000 iteraciones aproximadamente, los vehículos tardan más en completar la tarea.

4.2.2. Te cuento que ya terminamos (Navegación aleatoria; Comunicación incremental de resultado - Contagio)

Descripción:
Este caso posee los mismos elementos usados en el caso anterior principalmente en términos de la interfaz de usuario, pero se adicionaron algunas características que se describen a continuación: en este caso la comunicación entre los vehículos se realiza de manera incremental, es decir el vehículo que encuentra el objeto va transmitiendo el mensaje a los vehículos que se encuentran en un radio determinado (contagio) y a su vez estos vehículos pueden también transmitir el mensaje a otro vehículo, esto permite que existan muchos puntos de transmisión del mensaje y acelerar el proceso de propagación del mensaje. Parte del código de esta simulación aparece en el Apéndice 3, Caso 2.

Luego de realizar 200 pruebas de este caso con 64 vehículos en un escenario 64 x 64 con 32 calles x 32 carreras y ubicando el objetivo siempre en el centro del tablero estos son los resultados (ver tabla 4.4.).

Vehículos	Tamaño del tablero	Calles	Carreras
64	64 x 64	32	32

Tabla 4.3. Parámetros pruebas en el caso Ya terminamos 1.

Estadísticas	
Media (número de iteraciones)	5227.878
Desviación	576.921
Máximo	7204
Mínimo	3785

Tabla 4.4. Estadísticas del número de iteraciones del caso Te cuento que ya terminamos

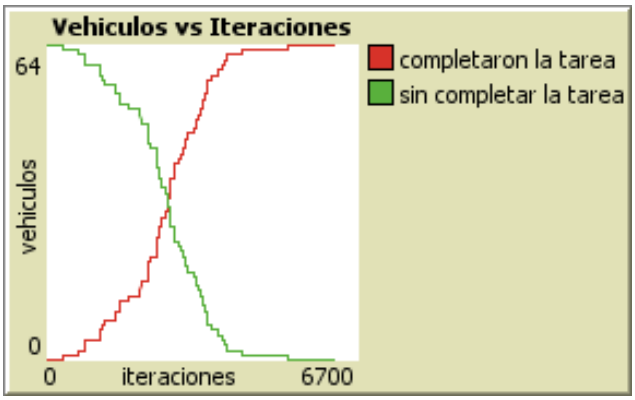

Figura 4.6. Gráfica de resultados caso Te cuento que ya terminamos

Análisis de la Gráfica.
En las gráficas generadas por la simulación del caso "Te cuento que ya terminamos" (Figura 4.6), se observa como la propagación del mensaje de objeto encontrado se hace cada vez más rápida debido a que a medida que transcurre el tiempo existen más carros que actúan de emisores del mensaje.

4.2.3. Les cuento a todos que ya terminamos (Navegación aleatoria; Comunicación global de resultado)

Descripción:
Este caso cuenta con las mismas características de los casos anteriores en términos de interfaz, pero implementa varias modificaciones respecto a los otros casos principalmente por desarrollo de un esquema de comunicación global entre los vehículos lo que permite que cuando el primer vehículo encuentre el objeto rojo inmediatamente envíe un mensaje al resto de vehículos avisando que el objetivo se ha logrado. Algunos elementos del código de esta simulación aparecen en el Apéndice 3, Caso 3.

Luego de realizar 200 pruebas de este caso con 64 vehículos en un escenario 64 x 64 con 32 calles x 32 carreras y ubicando el objetivo siempre en el centro del tablero estos son los resultados (ver tabla 4.6.).

Vehículos	Tamaño del tablero	Calles	Carreras
64	64 x 64	32	32

Tabla 4.5. Parámetros pruebas en el caso Les cuento a todos que ya terminamos.

Media (número de iteraciones)	1235.97
Desviación	224.8816
Máximo	2531
Mínimo	839

Tabla 4.6. Estadísticas del número de iteraciones de caso Les cuento a todos que ya terminamos

Figura 4.7. Gráfica de resultados caso Les cuento a todos que ya terminamos

Análisis de la Gráfica.
Las gráficas generadas por la simulación del caso "Les cuento a todos que ya terminamos", muestran un comportamiento similar al visto en la figura 4.7. Se observa un salto de la gráfica cuando un vehículo encuentra el objeto instantáneamente el resto vehículos pasan a estar rojos y las iteraciones que hay posteriormente son las del trayecto a la salida.

4.2.4. Nos repartimos los sectores y nos comunicamos a través de los nodos (Navegación por sectores; Comunicación incremental de resultado)

Descripción:
Este caso cuenta con todas las características desarrolladas en los casos anteriores, y además cuenta con elementos adicionales que se describen a continuación. Los vehículos se encuentran asignados a una zona especifica y solo pueden moverse en esta zona, además cuentan con una identificación y un color diferente exclusivo de la zona. La comunicación se realiza a través de unos nodos ubicados en cada zona, el vehículo que

encuentra el objeto envía un mensaje al nodo de sus zona, este nodo se encargan de propagar el mensaje a los nodos de otras zonas y estos a los vehículos de su zona. Apartes del código de esta simulación aparece en el Apéndice 3, Caso 4.

La figura 4.8 muestra una imagen del tablero presentado en esta simulación. Los cubos grises representan las antenas a través de las cuales se comunican los vehículos.

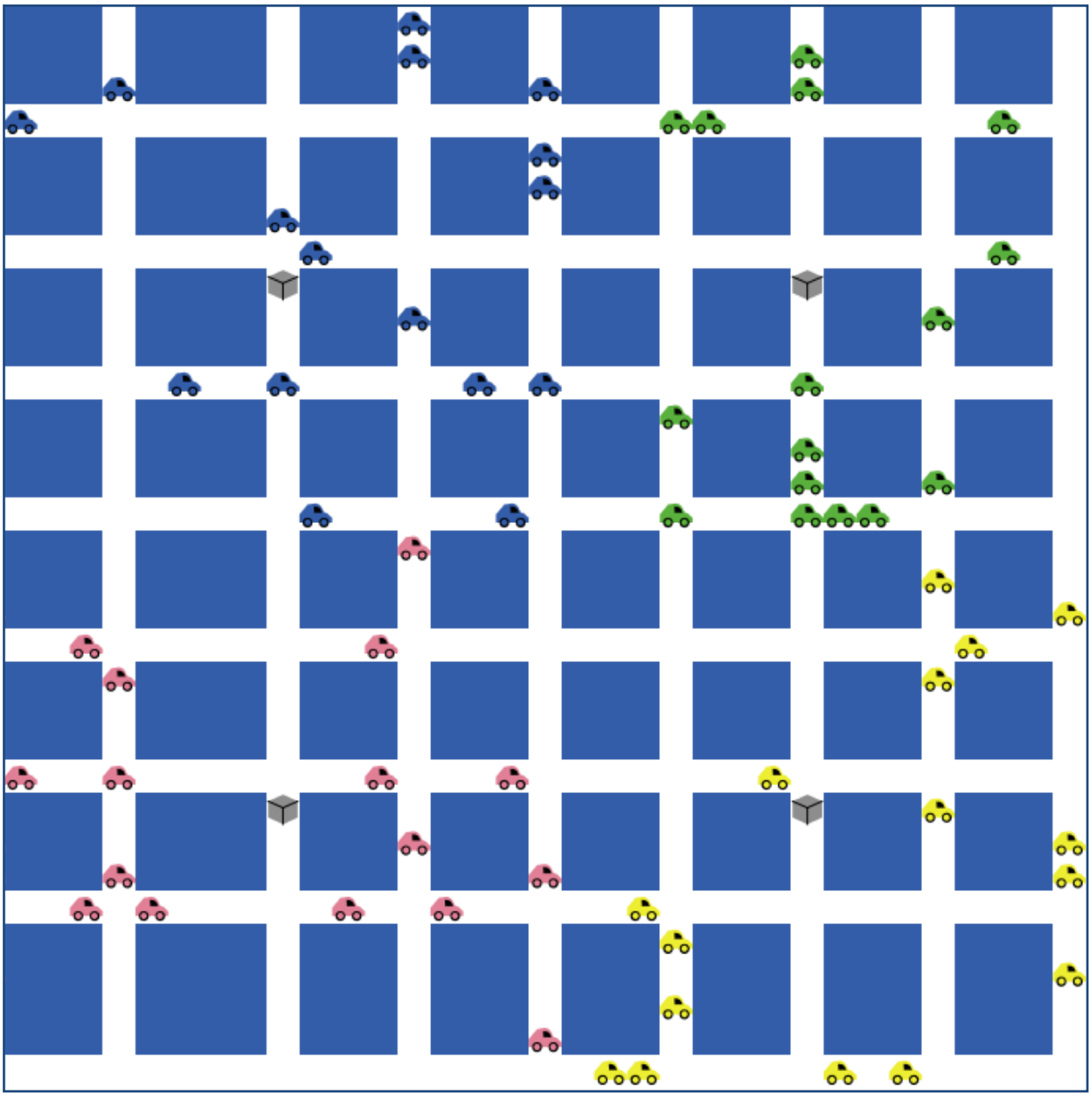

Figura 4.8. Interfaz de la simulación Nos repartimos los sectores y nos comunicamos a través de los nodos (tablero con 4 zonas y los vehículos con un color distintivo de su zona).

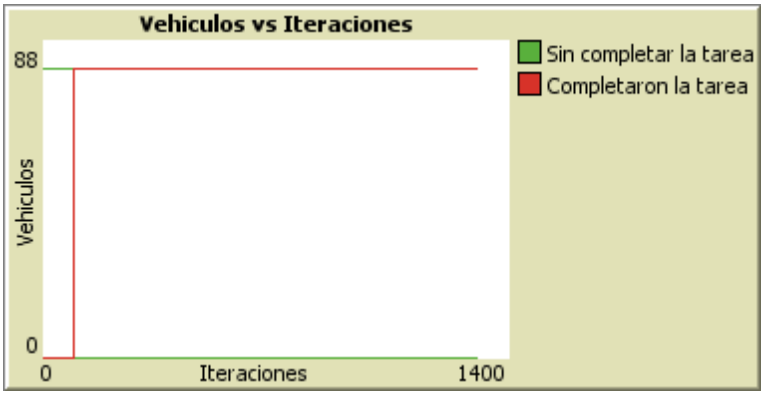

Figura 4.9. Gráfica de resultados caso Nos repartimos los sectores y nos comunicamos a través de los nodos

Análisis de la Gráfica.

Las graficas generadas por la simulación del caso "Nos repartimos los sectores y nos comunicamos a través de los nodos", muestran un comportamiento similar al visto en la figura 4.9. Se observa un salto de la grafica cuando un vehículo encuentra el objeto instantáneamente todos los nodos se cambian a color rojo y avisan a los vehículos de su zona. Las iteraciones que hay posteriormente son del trayecto a la salida.

Luego de realizar 200 pruebas de este caso con 64 vehículos en un escenario 64 x 64 con 32 calles x 32 carreras y ubicando el objetivo siempre en el centro del tablero estos son los resultados (ver tabla 4.8.).

Vehículos	Tamaño del tablero	Calles	Carreras	Calles por zona	Carreras por zona	zonas
64	64 x 64	32	32	2	2	16

Tabla 4.7. Parámetros pruebas en el caso Nos repartimos los sectores y nos comunicamos a través de los nodos.

Media (número de iteraciones)	1562,27
Desviación	571,11
Máximo	5810
Mínimo	892

Tabla 4.8. Estadísticas caso Nos repartimos los sectores y nos comunicamos a través de los nodos

4.2.5. Les cuento a todos que ya terminamos con mensajes FIPA-ACL (Navegación aleatoria; Comunicación global de resultado)

Descripción:
Este caso es similar al caso: "les cuento que ya terminamos", los cambios fundamentales están en la arquitectura utilizada, pues en este caso se usa una arquitectura BDI (*Belief-Desire-Intention*) y la manera de implementar la comunicación, en este caso se hace de manera global a través de mensajes FIPA-ACL. Se usan tres *intention*s (intenciones): buscar el objeto, comunicar a los otros e ir a la salida. Estas intenciones se van ejecutando una por una. En la figura 4.10, se muestra la ventana donde están los mensajes que se enviaron a cada uno de los vehículos y sus respectivas confirmaciones.

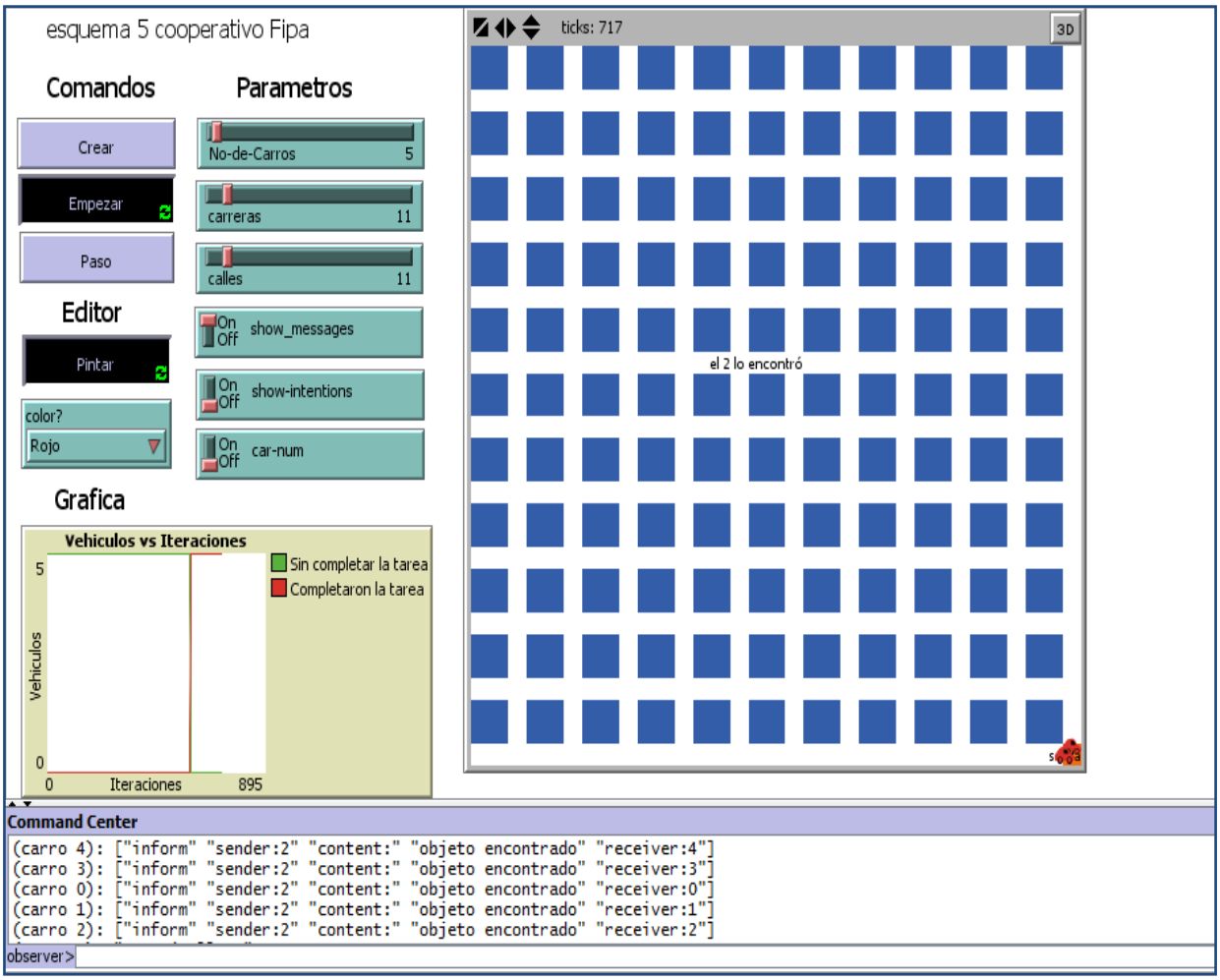

Figura 4.10. Interfaz y mensajes FIPA de la simulación Les cuento a todos que ya terminamos con mensajes FIPA-ACL

De acuerdo al propósito FIPA ofrece en su protocolo varios actos de comunicación, algunos de los actos más usados son: *Accept, Proposal, Call for Proposal, Inform, Inform If, Query If*. En esta simulación se hace uso del inform cuando el vehículo encuentra el objeto rojo y envía un mensaje al resto de vehículos informando que el objeto fue encontrado, la estructura de este tipo de comunicación es [29]:

(inform
:sender (agent-identifier :name i)
:receiver (set (agent-identifier :name j))
:content
"(objeto encontrado)".

Los mensajes FIPA-ACL son usados en la mayoría de sistemas multiagente y definen un esquema para el intercambio de mensajes, en *NetLogo* no se

trabaja con este sistema de mensajes FIPA-ACL de manera nativa, es entonces necesario el uso de 2 librerías adicionales [30] que se agregan al proyecto de la simulación. En este caso el vehículo que logra encontrar el objeto envía un mensaje explicito FIPA-ACL a el resto de vehículos. Parte del código de esta simulación aparece en el Apéndice 3, Caso 5.

Luego de realizar 200 pruebas de este caso con 64 vehículos en un escenario 64 x 64 con 32 calles x 32 carreras y ubicando el objetivo siempre en el centro del tablero estos son los resultados (ver tabla 4.10.).

Vehículos	Tamaño del tablero	Calles	Carreras	zonas
64	64 x 64	32	32	1

Tabla 4.9. Parámetros pruebas en el caso Nos repartimos los sectores y nos comunicamos a través de los nodos con mensajes CL-FIPA.

Media (número de iteraciones)	1261,783
Desviación	247,5418
Máximo	3063
Mínimo	915

Tabla 4.10. Estadísticas caso Nos repartimos los sectores y nos comunicamos a través de los nodos con mensajes CL-FIPA

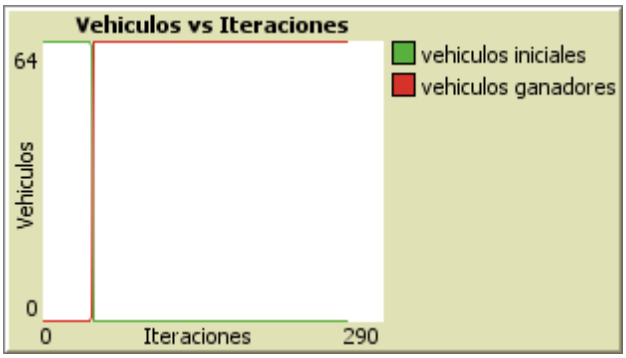

Figura 4.11. Gráfica de resultados del caso Les cuento a todos que ya terminamos con mensajes FIPA-ACL

Análisis de la Gráfica.
Las graficas generadas por la simulación de la simulación “Les cuento a todos que ya terminamos con mensajes FIPA-ACL”, muestran un

comportamiento similar al visto en la figura 4.11. En ella se ve como luego de que el objeto es encontrado todos los vehículos se enteran y cambian su estado, posteriormente se dirigen a la salida.

4.2.6. Buscar o no buscar, he ahí el dilema (Navegación para maximizar utilidad esperada; Comunicación por mensaje a todos usando FIPA)

Descripción:
Este caso implementa una función de utilidad esperada que es evaluada por los vehículos antes de cada movimiento, esta función ayuda a los agentes a tomar decisiones respecto de continuar o abandonar la búsqueda en determinado momento de la simulación (Ver ecuación 1).

$$UE = (Prob * Recompensa) - (EnergiaMaxima - EnergiaInicial) \quad (1),$$

Donde UE es la utilidad esperada, la cual es calculada como el producto de la probabilidad por la recompensa menos la resta entre la energía máxima (1000 unidades) y la energía inicial (valor entre 150 y 1000 unidades).

Prob, es la probabilidad que tiene el vehículo en un momento dado de seguir buscando el objeto y es calculada con base en la cantidad de energía inicial y la energía consumida hasta el momento.

$$Prob = \left(1 - \frac{(Energía\ inicial - Energía\ actual)}{Energía\ inicial}\right)$$

Cabe anotar que todos los vehículos inician con una cantidad de energía diferente entre 150 y 1000 unidades.

La Recompensa es un parámetro que configura el usuario en el panel de parámetros de la interfaz gráfica. En las pruebas se usa el valor 1000, este valor tiene una gran influencia en las decisiones de los vehículos por ejemplo si en una simulación se configura un premio bajo de 50 unidades los vehículos cuando evalúan su función de utilidad tendrán una creencia pesimista y abandonaran la búsqueda, por el contrario si el premio es alto ejemplo 10000 el resultado de la evaluación de la función de utilidad

esperada dará como resultado una creencia optimista por un largo periodo de tiempo lo que puede tener como consecuencia que los vehículos prefieran quedar varados en medio del tablero antes que cambiar su creencia a indeciso y a pesimista.

Basado en esta utilidad esperada se implementan tres niveles de creencias para los vehículos. El primer nivel de creencia es el optimista, el cual consiste en tener una utilidad esperada mayor a 666 unidades, por lo tanto los vehículos estarán realizando la búsqueda del objeto. El segundo nivel es el indeciso, el cual se da cuando la utilidad esperada está entre un valor de 333 y 666 unidades, en este nivel los vehículos tiene 2 opciones que son, quedarse buscando o abandonar la búsqueda del objeto, para ello toma una única decisión en el momento que pasa de ser optimista a indeciso (utilidad esperada = 666 unidades) o en el momento que al evaluar su función de utilidad por primera vez obtiene un valor entre 333 y 666 unidades, esta decisión la hace basada en el cálculo de un valor aleatorio de 1 a 10, si el valor es menor o igual que 5 decide continuar la búsqueda del objeto y si el valor es mayor que 5 decide abandonar la búsqueda. El tercer nivel es el pesimista, el cual se define como el nivel donde la utilidad esperada es menor a 333 unidades, si es el caso el vehículo abandona la búsqueda del objeto y se dirige directamente a la salida.

Según la creencia actual del vehículo se asignan una serie de intenciones, en este caso se cuenta con 3 intenciones que son: buscar el objeto, comunicar a los otros e ir a la salida. Por ejemplo, en el caso en que el vehículo es optimista (creencia nivel 1), el vehículo ejecuta las intenciones buscar el objeto, comunicar a los otros e ir a la salida, contrario al pesimista que solo ejecuta la intención de ir a la salida.

Es importante anotar que las intenciones son las que llevan el flujo de la simulación y dependen de la creencia actual del agente. La comunicación se hace usando mensajes FIPA-ACL mediante el uso de las 2 librerías adicionales mencionadas anteriormente [30].

La simulación también informa al usuario sobre varias estadísticas: el número de vehículos que abandonaron la búsqueda, ahorro total de energía que es la suma del combustible con el que llegan todos los vehículos a la salida, la cantidad total de energía consumida que es la suma de las diferencias entre la energía inicial y la energía con la que

llegan a la salida, promedio de energía consumida por vehículo y el promedio del ahorro de energía por vehículo. Ver figura 4.12.

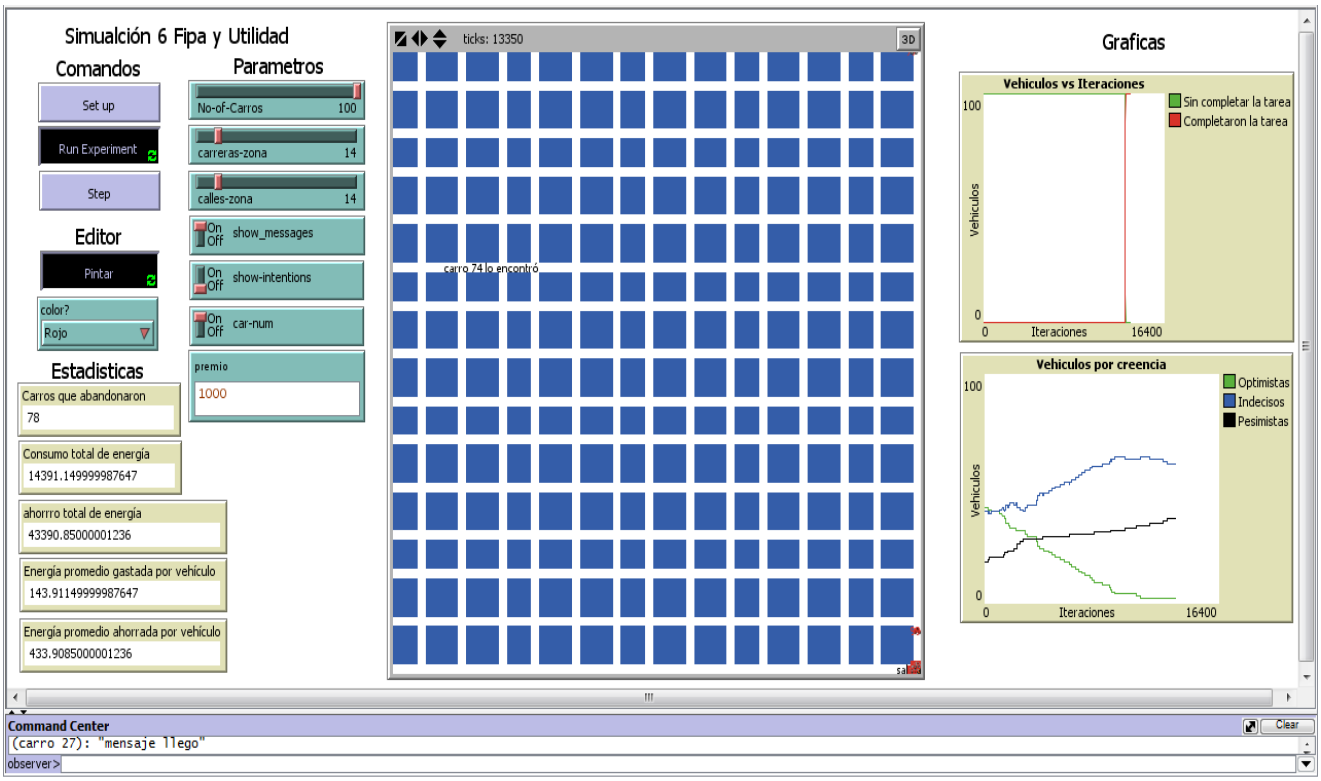

Figura 4.12. Gráfica de resultados caso Buscar o no buscar, he ahí el dilema con las nuevas características de estadísticas y la nueva grafica de Vehículos por creencias

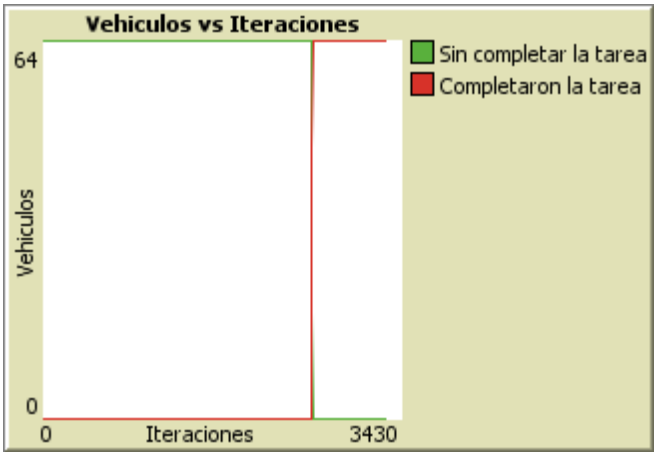

Figura 4.13. Gráfica de resultados caso Buscar o no buscar, he ahí el dilema

Análisis de las Gráficas:

Las graficas generadas por la simulación “Les cuento a todos que ya terminamos con mensajes FIPA-ACL”, muestran un comportamiento similar al visto en la figura 4.12., la cual muestra como la comunicación se realizó de manera global y los 64 vehículos pasaron de estar sin completar la tarea a completar la tarea casi que en el mismo momento.

En la gráfica 4.13 se muestra como van cambiando las creencias de los vehículos a medida que va pasando el tiempo, por ejemplo es evidente el incremento del número de vehículos pesimistas y la disminución de los vehículos optimistas, los indecisos fueron aumentando y disminuyendo de acuerdo al comportamiento de las otras 2 creencias.

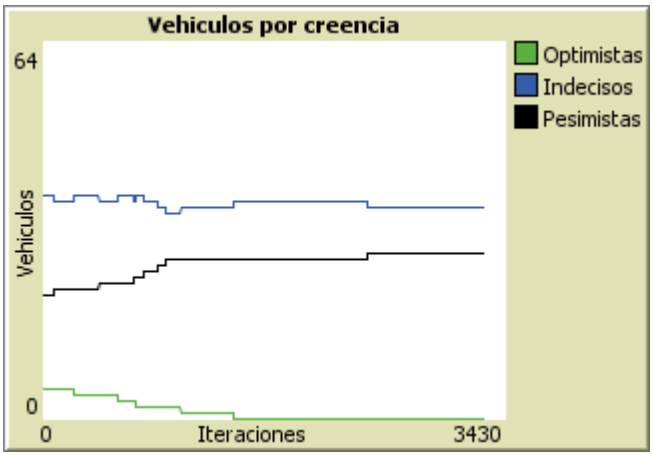

Figura 4.14. Cambio de las creencias de los agentes en la simulación Buscar o no buscar he ahí el dilema

Luego de realizar 200 pruebas de este caso con 64 vehículos en un escenario 64 x 64 con 32 calles x 32 carreras y ubicando el objetivo siempre en el centro del tablero estos son los resultados (ver tabla 4.11. y 4.12.).

Vehículos	Tamaño del tablero	Calles	Carreras	zonas
64	64 x 64	32	32	1

Tabla 4.11. Parámetros pruebas en el caso o no buscar, he ahí el dilema

Media (número de iteraciones)	1958,667
Desviación	376,6162
Máximo	3914
Mínimo	1251

Tabla 4.12. Estadísticas caso buscar o no buscar, he ahí el dilema

4.3. Análisis y comparación de resultados.

La tabla 4.13, contiene los resultados medidos en cantidad de iteraciones de cada una de las simulaciones (casos del 1 al 6). En la tabla se observa que el caso "Todos buscamos" gasta un número elevado de iteraciones, esto es debido a que no existe ningún esquema de comunicación que permita que los vehículos que van encontrando el objetivo puedan avisar al resto, entonces se debe esperar a que todos los vehículos por sus propios medios encuentran el objetivo.

El caso "Les cuento que ya terminamos" se mejora el rendimiento en cerca de 1/20 del numero de iteraciones del caso "Todos Buscamos" porque se implementa un esquema de transmisión del mensaje en forma de contagio, lo que permite que existan cada vez más puntos de transmisión a medida que avanza la simulación.

Los casos "les cuento que ya terminamos" y "Nos repartimos los sectores y nos comunicamos a través de los nodos" tienen un comportamiento similar y mucho más rápido que los casos anteriores porque en ambos la comunicación hace de manera global.

La diferencia se debe a que en el caso "Nos repartimos los sectores y nos comunicamos a través de los nodos" la comunicación se hace varias etapas (la primera para comunicar el mensaje a los nodos y otra etapa

para comunicar el mensaje a los vehículos). Sin embargo el comportamiento es similar.

Por último los casos “Les cuento a todos que ya terminamos con mensajes FIPA-ACL” y “Buscar o no buscar, he ahí el dilema” presentan una diferencia grande con los casos anteriores en el modo de implementar la comunicación, pero se puede decir que ambos tienen un buen rendimiento, el último caso “Buscar o no buscar he ahí el dilema” se ve afectado un poco en su rendimiento debido a que su sistema de creencias e intenciones basado en la evaluación de la utilidad esperada hace que los vehículos puedan decidir si continuar o abandonar la búsqueda del objeto, teniendo en cuenta si es un buen negocio seguir gastando su energía en la búsqueda.

Esto tiene como efecto que en un momento dado haya menos vehículos buscando el objeto y causa un impacto en el número de iteraciones si se compara con otros casos.

Como efecto positivo este caso muestra un ahorro en el consumo de energía total de los vehículos debido a su sistema de creencias e intenciones evita que vehículos sin muchas opciones de encontrar el objeto, continúen buscándolo.

Numero del caso	Nombre del caso	Media Número de iteraciones	Desviación	Máximo de iteraciones	Mínimo de iteraciones
1	Todos Buscamos	111494,56	21572,73	248316	67227
2	Te cuento que ya terminamos	5227.878	576.921	7204	3785
3	Les cuento que ya terminamos	1244,69	235,21	2676	908
4	Nos repartimos los sectores y nos comunicamos a través de los nodos	1562,27	571,11	5810	892

5	Les cuento que ya terminamos con mensajes FIPA-ACL	1260,67	246.69	3063	915
6	Buscar o no buscar, he ahí el dilema	1958,667	376,6162	3914	1251

Tabla 4.13. Tabla resumen de Estadísticas casos 1 al 6

4.4. Limitaciones de las simulaciones

Este proyecto es un primer paso en el estudio de sistemas multiagente. Con su desarrollo se han abierto un gran número de posibilidades para seguir explorando. Estas situaciones no se pudieron abordar por el tiempo y el alcance que se había definido para el trabajo, algunos ejemplos interesantes son:

- La posibilidad hacer que el terreno tenga importancia en las decisiones que toman los vehículos, esto podría lograrse incluyendo diferentes tipos de mapas con zonas de peligro para los vehículos, que afecten en un momento dado las creencias del vehículo y el tipo de mensajes usados para los agentes para negociar la toma de una decisión.

- Posibilidades como que los vehículos pesimistas puedan remolcar vehículos varados de regreso a la salida y así poder dejar las vías libres para el resto de vehículos.

- Un esquema en el que los vehículos puedan recargar su energía en algunos puntos del tablero y ver como este nuevo componente tiene efecto sobre las creencias e intenciones de los vehículos y en los resultados de la simulación.

- Establecer modos de consumo de energía de acuerdo a que sensores o elementos de comunicación tenga encendidos o se estén utilizando en el momento, esto podría hacer que los vehículos mediante una política interna de regulación del consumo de energía puedan mantenerse más tiempo en el tablero.

- Que los vehículos puedan marcar el terreno y evitar visitar zonas que ya han visitado previamente y medir los efectos en cuanto a mejora el tiempo que los vehículos necesitan para alcanzar el objetivo.

- Hacer todo el proceso de implementación de arquitectura BDI y comunicaciones a través de mensajes FIPA-ACL a los primeros 4 casos de simulación presentados en el proyecto.

- El esquema de intenciones maneja diferentes funcionalidades de los vehículos. Por ejemplo, cuando un vehículo tiene la intención de ir a la salida (porque su función de utilidad proporciona un valor pesimista), si en el trayecto coincide con el objeto buscado este no lo reconoce, ni reporta que lo encontró, ya que este vehículo en este momento va funcionando en un modo de funciones básicas que solo le permiten llegar a la salida. Esto se puede mejorar haciendo cambios a la simulación para que los agentes tengan en cuenta esta situación y comuniquen que han visto el objetivo en su camino a la salida.

CAPITULO 5

SISTEMAS MULTIAGENTES EMBEBIDOS EN *LEGO*

Este capítulo describe la implementación de una simulación de agentes inteligentes en *Lego* [Buscar o no buscar he ahí el dilema].

5.1. Materiales

2 unidades centrales NXT *Lego MindStorms* [31] incluyen comunicación vía Bluetooth.
2 sensores de colores originales *Lego* Complementarios.
2 sensores de ultrasonido.
4 motores DC incluidos en el kit de robótica *Lego MindStorms*.
1 Tablero cuadriculado para la simulación de vías.
Software NXT-G [29].

5.2. Descripción de ensamble

Los vehículos se ensamblaron en un esquema simple de 2 motores para las ruedas delanteras y una rueda omnidireccional en la parte posterior, en la parte frontal se equipó con un sensor de colores que facilitan la navegación sobre el tablero y con un sensor de ultrasonido que sirve para evitar las colisiones (ver figura 5.1).

Figura 5.1. Robots diseñados para la simulación en *Lego*

5.3. Software Para el diseño y la Programación

Para el diseño del vehículo se utilizó *Lego Digital Designer* versión 3.1 el cual permite modelar el robot antes de hacerlo físicamente.

La programación de la simulación se realizó con el software NXT-G incluido en el kit *Lego MindStorms,* este software permite realizar la programación de manera gráfica en forma de diagrama de flujo, incluye funciones para configurar los sensores, los motores y para el uso del dispositivo *Bluetooth*.

Posee estructuras básicas como decisiones y ciclos, permite el uso de operaciones matemáticas básicas, incluye funciones como comparadores, definición de rangos, conversión de tipos de datos y manejo de *string*s.

Está habilitado para realizar definición de variables con tipos de datos booleanos, numéricos y de texto, además incluye opciones para realizar una modularización de los programas en forma de bloques facilitando así la visualización y desarrollo de los programas.

5.4. Detalles sobre la implementación del demo de la simulación: Buscar o no buscar, he ahí el dilema

Para la implementación de este demo se seleccionó la simulación Buscar o no buscar he ahí el dilema (último caso que se implemento en *NetLogo*), porque hace uso dos características muy importantes en el tema de los sistemas multiagente, primero se desarrollo con una arquitectura BDI basada en creencias e intenciones y segundo porque realiza una comunicación explicita mediante el uso de mensajes FIPA-ACL, estas características se llevaron al sistema *Lego* de una manera simplificada a modo de demo, debido a limitaciones del hardware como el tamaño limitado de la memoria.

5.4.1. Restricciones en la implementación del demo en *Lego MindStorms*

Para La implementación se restringieron de la simulación original elementos como:

- Tamaño y personalización del tablero.

Por razones de costos el tablero para facilitar las pruebas se diseño de un tamaño constante de 4 calles por 4 carreras y con un patrón de diseño en forma de cuadricula.

- Número de vehículos.

Por facilidad y disponibilidad del *hardware* se realizaron las pruebas del programa demo para dos vehículos, tomando en cuenta que si el comportamiento del sistema es correcto con dos vehículos, es posible escalarlo en el futuro a una cantidad mayor de vehículos, además el programa quedo habilitado para ser instalado en más equipos posteriormente.

- Navegación de los vehículos en el tablero.

Al contar con solo un sensor de color por cada vehículo, la navegación se hace de manera menos fluida que en la simulación de software, pero conservando el criterio original de navegación aleatoria.

- Datos estadísticos y gráficos.

En el programa demo no se tienen ayudas graficas ni estadísticas y no se toman datos de varios experimentos en archivos como ocurre en la simulación en *software*.

- Comunicaciones.

En el programa demo no se utilizan las librerías de comunicaciones que poseen las funciones para el envío de mensajes con el protocolo FIPA a cambio de esto se uso el protocolo de comunicaciones de Bluetooth.

- Manejo de estructuras

El lenguaje grafico de *MindStorms* carece de un manejo de pilas para el almacenamiento de las creencias y las intenciones por lo cual se manejaron variables que representaban dichos elementos y se maneja a través de una estructura de decisiones que controlan el flujo de asignación de las variables.

5.4.2. Funcionamiento del demo

Para la ejecución por primera vez del demo primero se debe configurar manualmente la conexión de *Bluetooth* entre los vehículos, para ello se debe realizar el emparejamiento de los dispositivos, que incluye la búsqueda y autenticación con contraseña, una vez establecida la conexión se puede ejecutar el programa en cada uno de los vehículos. Los vehículos implementan un esquema de creencias basadas en la evaluación de la función de utilidad y al igual que en la versión en software, estas creencias pueden verse modificadas durante la ejecución, asociadas a cada creencia están un grupo de acciones que son moverse por el tablero, evitar los choques, enviar mensajes, recibir mensajes y detenerse (Ver figuras 5.1. y 5.2.). Estas acciones serán tomadas de acuerdo con la evaluación de la función de utilidad (ver figura 5.3.) que depende de la energía inicial (Ver figura 5.4); cuando uno de los vehículos ha encontrado el objeto rojo (ver figura 5.5.) envía el mensaje al otro y comienza a dar tres vueltas, mientras tanto el otro vehículo recibe el mensaje y también empieza a dar tres vueltas en señal de que recibió el mensaje. Luego de esto los vehículos paran y finaliza la ejecución del demo.

La figura 5.2, muestra como inicialmente se calcula un número aleatorio 0 o 1, luego está el comando de motor de dar un paso adelante, posteriormente se entra a un bloque de dos decisiones.

La primera es para chequear con el sensor de colores si esta sobre una zona negra o blanca, si esta en negro, entra a la segunda decisión y usa el valor del aleatorio para decidir si retroceder y girar a la izquierda o retroceder y girar a la derecha.

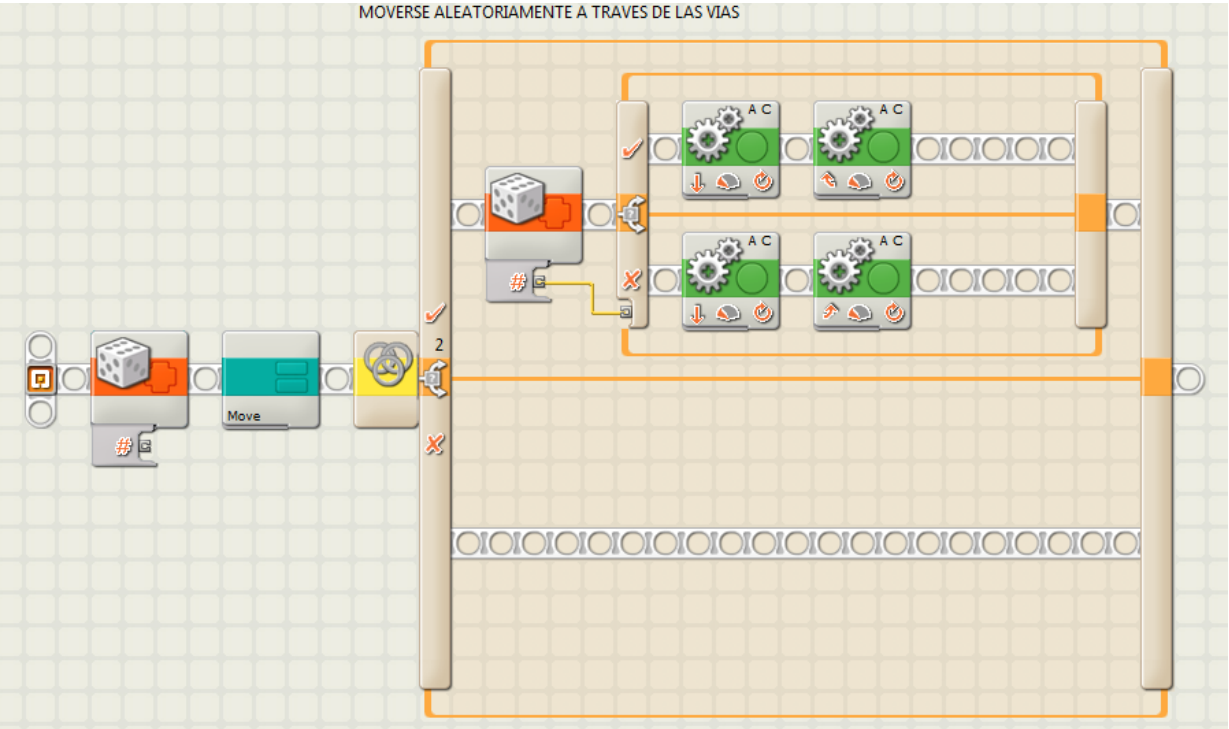

Figura 5.2. Moverse aleatoriamente a través de las vías

La figura 5.3, muestra el cálculo de la energía inicial basado en un valor aleatorio entre 150 y 1000 unidades este valor se guarda en dos variables la primera que solo se escribe al principio de la simulación y la segunda que va a ser actualizada a lo largo de la simulación.

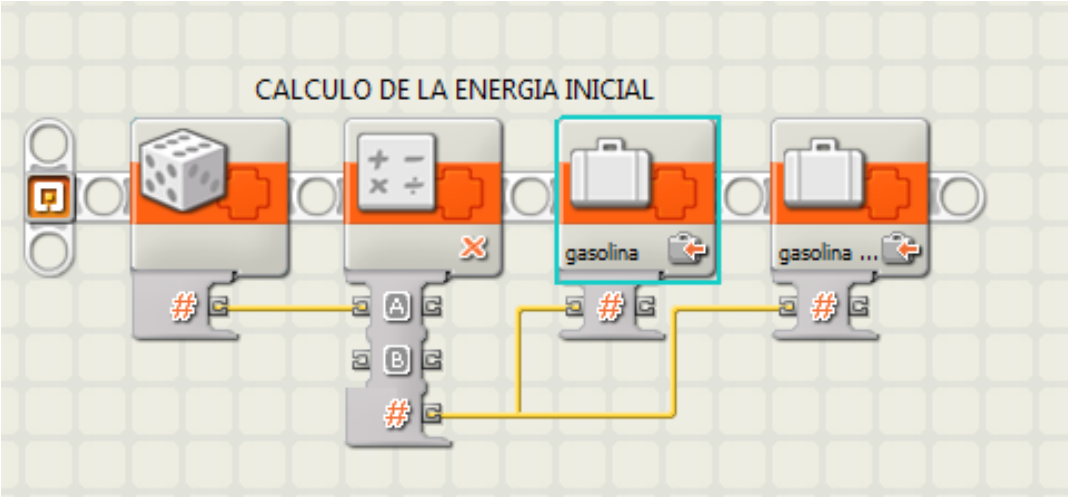

Figura 5.3. Cálculo de la energía inicial

La figura 5.4, muestra el cálculo de la probabilidad. Inicialmente se restan la energía inicial y la energía actual, luego se divide este resultado entre la energía inicial, finalmente este resultado se resta de 1 y se asigna a la variable probabilidad.

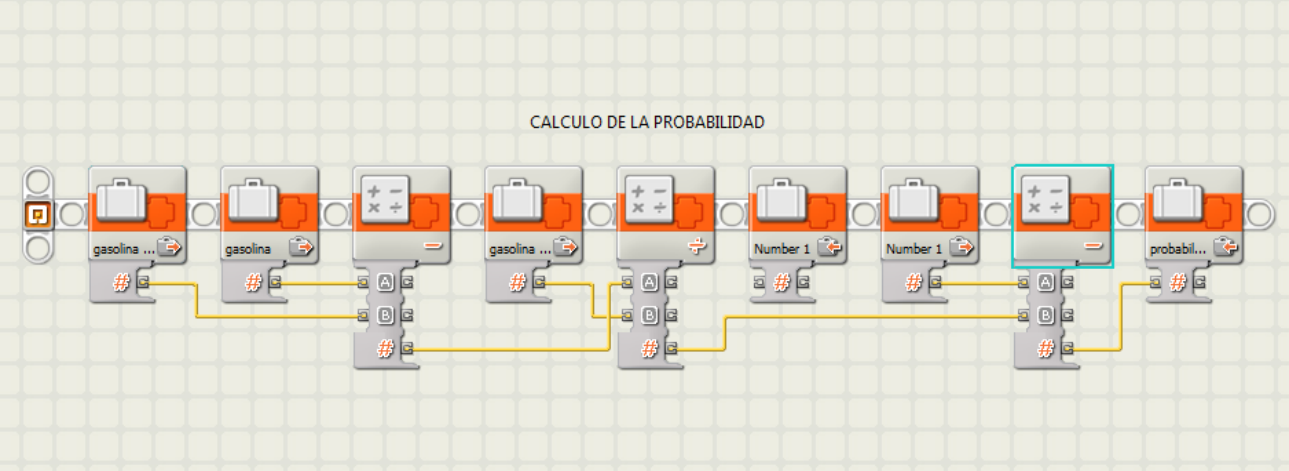

Figura 5.4. Cálculo de la probabilidad de continuar buscando el objeto.

La figura 5.5, muestra la implementación de la función de utilidad esperada. En el primer bloque de operación se realiza el producto de la probabilidad por el premio en el segundo bloque de operación se hace la resta entre la energía máxima y la energía actual y en la tercera operación se hace la resta de las dos operaciones anteriores y se asignan a la variable utilidad.

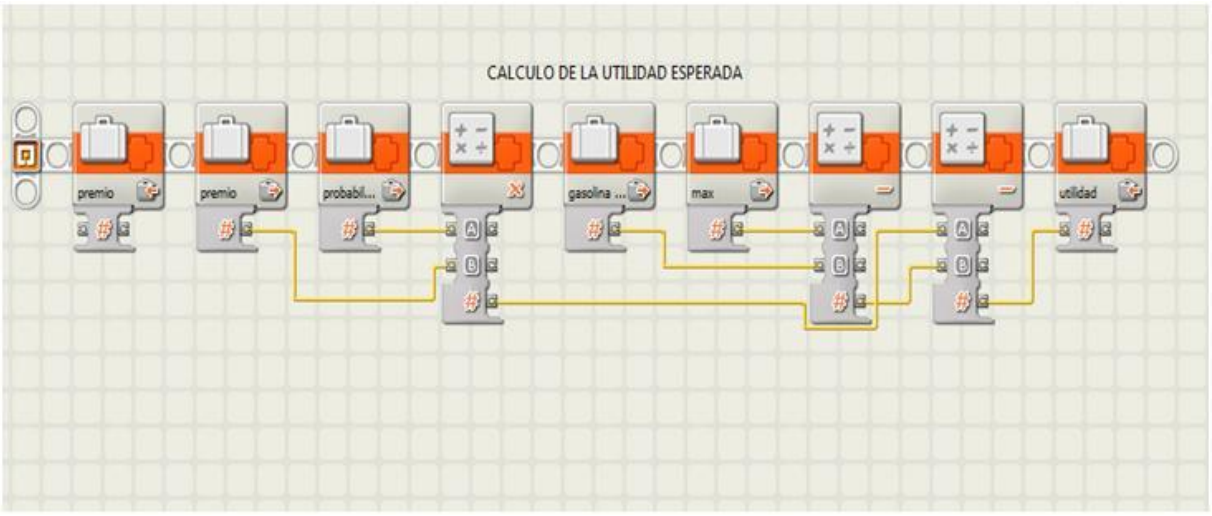

Figura 5.5. Cálculo de la utilidad esperada

La figura 5.6, muestra el bloque de recibir un mensaje por *Bluetooth*, si el mensaje contiene la palabra "encontrado" ingresa por la parte inferior de la decisión y toma la acción de dar tres vueltas y luego parar.

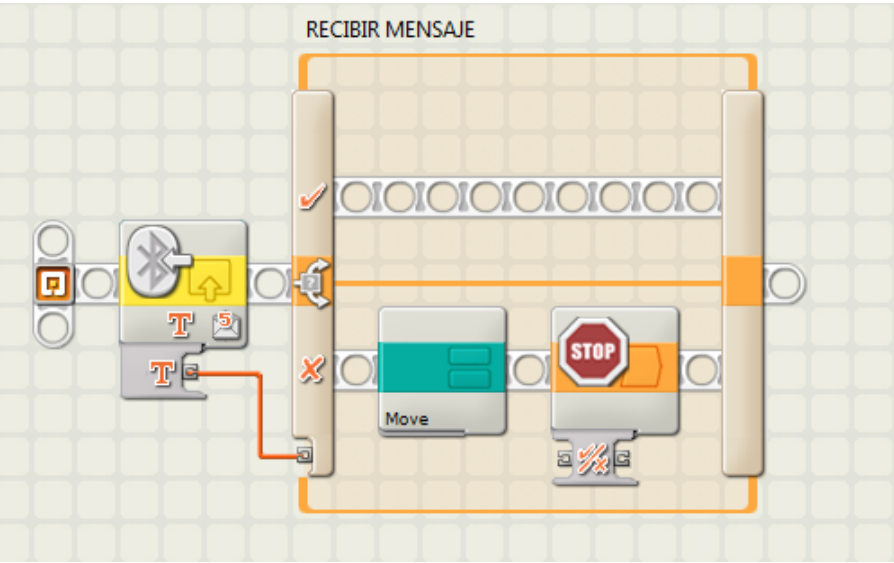

Figura 5.6. Recibir los mensajes y tomar acciones

La figura 5.7, muestra una decisión a partir del resultado de la lectura del sensor de colores, aquí se evalúa si el vehículo encontró el objeto rojo, si es así entonces ingresa a un bloque de envío de mensaje a través de *Bluetooth* desde donde envía la palabra "encontrado" al otro vehículo luego procede a dar tres giros en el mismo punto.

Figura 5.7. Detectar el objeto rojo y enviar el mensaje

La figura 5.8, muestra la situación inicial de la ejecución del demo, es este momento los vehículos evalúan su función de utilidad y toman su creencia.

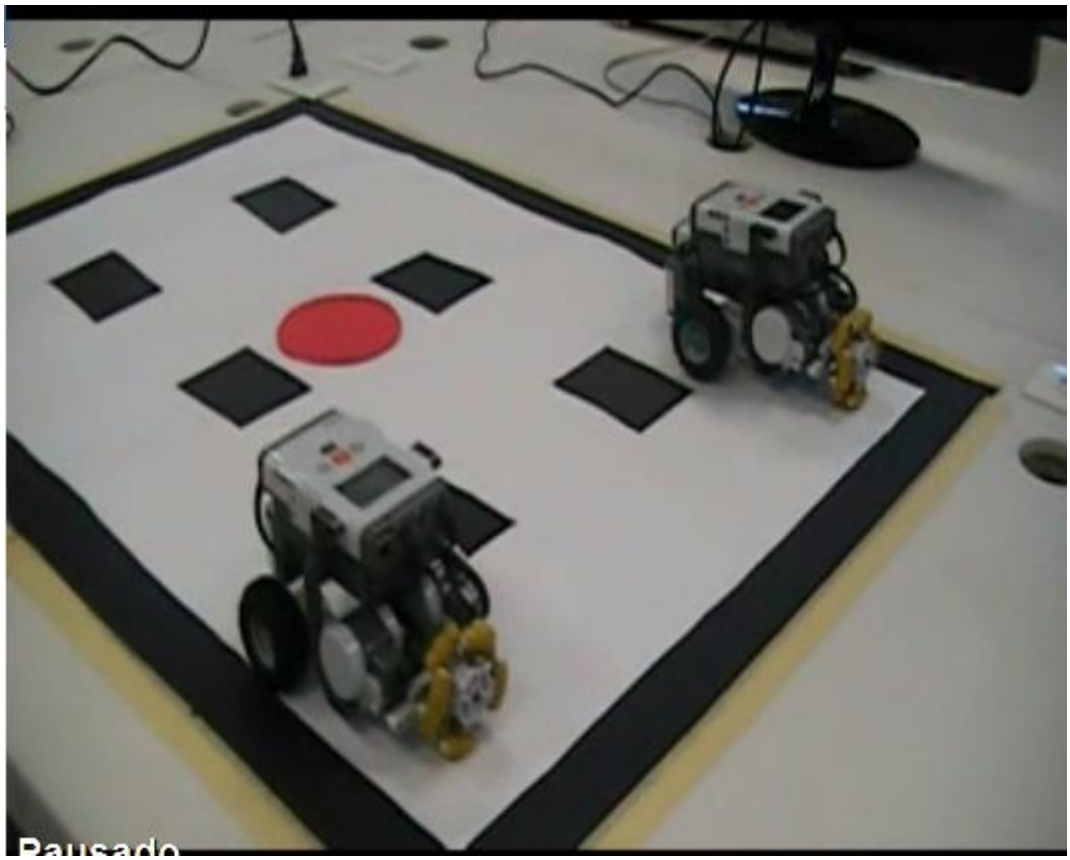

Figura 5.8. Simulación en *Lego MindStorms* del caso Buscar o no buscar he ahí el dilema.

La figura 5.8 muestra el momento en el que uno de los vehículos ubica el objeto rojo, aquí es cuando envía el mensaje al otro vehículo, empieza a dar tres vueltas y coloca un mensaje en la pantalla que dice “LO ENCONTRAMOS”.

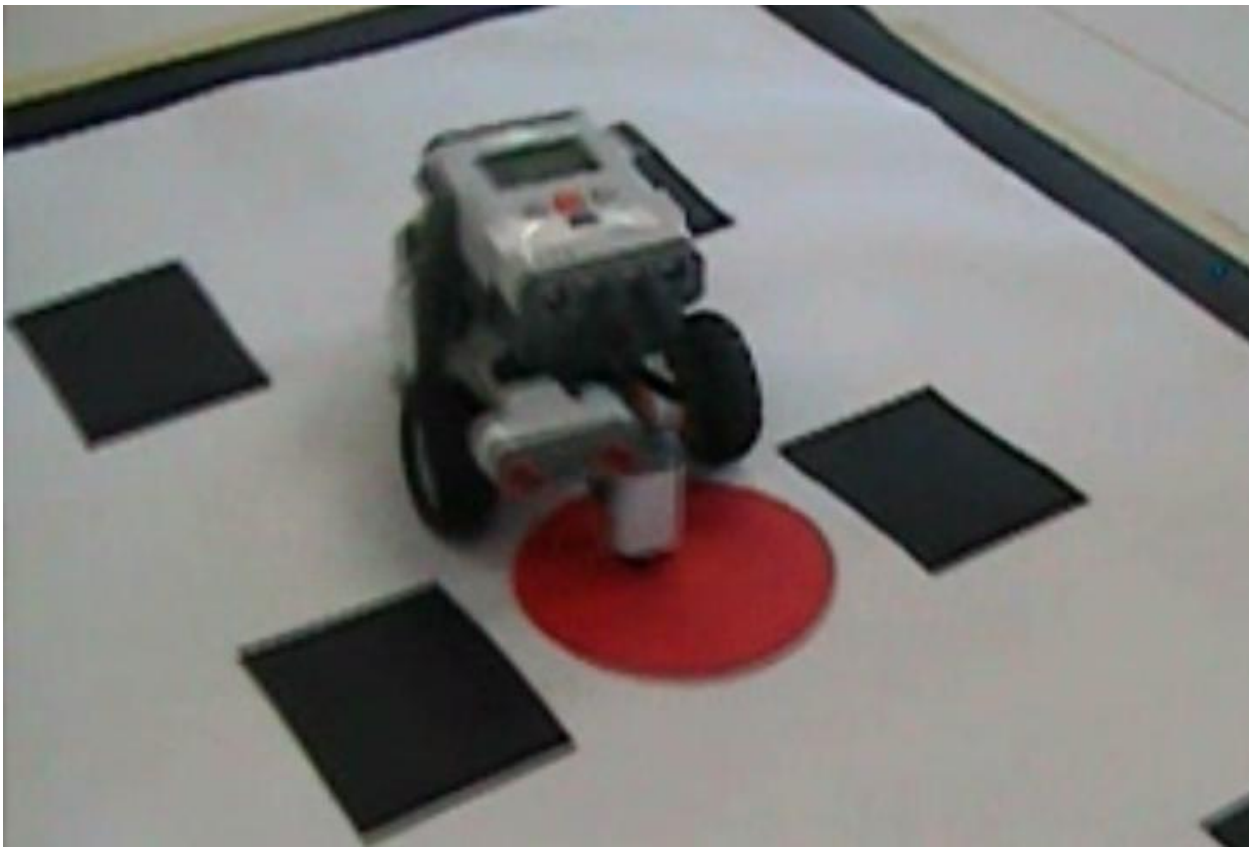

Figura 5.9. Se encontró el punto rojo

La figura 5.10, muestra el momento en que el vehículo recibe el mensaje “LO ENCONTRAMOS” y se encuentra dando las tres vueltas en señal de que se recibió el mensaje

Figura 5.10. Vehículo que recibe el mensaje

5.5. Futuro trabajo en *Lego* y agentes

Para el futuro queda pendiente la implementación completa en *Lego MindStorms* de otros casos de sistemas multiagente simulados en *NetLogo*, con la posibilidad de usar hardware adicional como más vehículos, más sensores de colores, GPS, y sistemas de orientación como brújulas entre otros, además existe también la posibilidad de usar otros lenguajes para la programación de robots que sean compatibles con *Lego MindStorms* y que implementen librerías para arquitecturas BDI y protocolo FIPA o en su defecto realizar la implementación de dichas librerías. El código completo de esta simulación aparece en el Apéndice 4.

En el prototipo implementado plataforma *Lego MindStorms* se identificó una posibilidad de un nuevo caso interesante para abordar posteriormente y ocurre cuando en la simulación se pierde la comunicación. Esto ocasiona que los vehículos actúen de forma similar al primer caso de simulación llamado todos buscamos pero con el componente de creencias e intenciones. Sería interesante explorar sistemas de *troubleshooting* basado en el estado de creencias.

CAPITULO 6

CONCLUSIONES

A continuación se presentan las conclusiones del proyecto.

- La plataforma *NetLogo*, es una herramienta útil para el aprendizaje y desarrollo de un sinnúmero de aplicaciones donde se utilicen sistemas multiagente colaborativos.

- La plataforma *Lego MindStorms* es una herramienta que permite unir el hardware y el software para realizar experimentos que permitan plasmar de una manera tangible los resultados de simulaciones. La implementación en hardware permite en cuenta la complejidad adicionada por las variables del entorno.

- El continuo incremento de la capacidad de la computación ha permitido que cada vez más, la implementación de sistemas multiagente sea más simple y eficiente, también ha permitido poder desarrollar sistemas multiagente en hardware lo que hace que campos como la robótica y los sistemas inteligentes estén estrechamente ligados.

- Las estadísticas de las pruebas realizadas durante el proyecto han mostrado que los esquemas de colaboración como repartir el trabajo entre varios agentes, la implementación de políticas diferentes de comportamiento entre un grupo de agentes y el hecho de implementar comunicaciones entre los agentes, permiten mejorar el rendimiento de los sistemas multiagente, en aspectos como velocidad para encontrar el objetivo y la comunicación a los demás agentes.

- Este trabajo tiene aplicación en operaciones de búsqueda y rescate a través de agentes. Algunos de los sistemas implementados incluyen esquemas de navegación y evaluación que calculan el beneficio de las acciones a través de una función de utilidad. Esta función usa

probabilidades de acuerdo a niveles de energía y tiene en cuenta la recompensa que puede recibir el agente al encontrar lo que busca. El uso de estos mecanismos pueden ser de gran utilidad para robots que realizan búsquedas.

- Durante el desarrollo de este proyecto algunos estudiantes se mostraron interesados en aprender más del funcionamiento de las simulaciones y como se implementaban en hardware. Este proyecto tiene el potencial de motivar a los estudiantes a aprender acerca de la inteligencia artificial y la robótica. Podría ser interesante incluir dentro del currículo temas de sistemas multiagente, no solo de forma teórica sino también mediante el uso de simuladores como *NetLogo* y tecnologías como *Lego MindStorms*, de esta manera los estudiantes podrán tener una experiencia más enriquecedora y útil para su futuro profesional.

REFERENCIAS

[1] Manzoor, U. and Nefti, S. and Hasan, H. and Mehmood, M. and Aslam, B. and Shaukat, O. (2008). A multi-agent model for mine detection—MAMMD. *Emerging Technologies and Information Systems for the Knowledge Society*. pp. 139—148

[2] Zafar, K. and Baig, A.R. and Badar, S. and Naveed, H. (2009). Multi Agent Based Mine Detection and Route Planning Using Learning Real Time A* Algorithm. *Computer Science and its Applications, 2009. IEEE CSA'09*. pp. 1–7

[3] Ferreira, P., dos Santos, F., Bazzan, A., Epstein, D. and Waskow, S. (2010). Robocup Rescue as Multiagent Task Allocation among Teams: experiments with task interdependencies. *JAAMAS, 20*(3):421–443.

[4] A. Farinelli, G. Grisetti, L. Iocchi, S. Lo Cascio, and D. Nardi. Design and evaluation of multi agent systems for rescue operations. *In Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'03)*. pp. 3138–3143

[5] Earon E J P, Barfoot T D, and D'Eleuterio G M T. (2001). "Development of a Multiagent Robotic System with Application to Space Exploration". In Proceedings of the IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM). pp. 8-11.

[6] Russell, S. J. y Norvig P. (2004). Inteligencia Artificial. Un Enfoque Moderno. Editorial

[7] Rutgers Laboratory or Realife Reinforcement Learning (En Línea) http://www.cs.rutgers.edu /rl3/. Consultado el 5 de Enero de 2010.

[8] Rivas, I. L. Inteligencia Artificial, (En Línea) http://www.slideshare.net/guest0d7e01/inteligencia-artificial-3178034

[9] Vassileva J., J. Greer, G. McCalla, R. Deters, D. Zapata-Rivera, C. Mudgal, S. Grant. (1999). A Multi-Agent Approach to the Design of Peer-Help Environments, In S. Lajoie and M. Vivet (eds.) *Artificial Intelligence in Education*, IOS Press: Amsterdam, 38-45.

[10] Zapata-Rivera, J.D. & Greer, J. (2001) SMODEL Server: Student Modelling in Distributed Multi-Agent Tutoring Systems. *International Conference on Artificial Intelligence in Education AIED 2001*. 446-455.

[11] Nwana, H. S. UMBC agent webs, “A Panoramic Overview of the Different Agent Types”. UMBC, an honors University of Maryland. (En Línea): http://agents.umbc.edu/introduction/ao/5.shtml

[12] Bautista Vallejo, J. M (2007), Los agentes de software inteligentes y la respuesta didáctica a la diversidad. Revista Electrónica Actividades investigativas en educación. Universidad de Costa Rica. (En Línea) http://redalyc.uaemex.mx/pdf/447/44770110.pdf

[13] Robocup. (En Línea): http://www.robocup.org/

[14] FIRA | Federation of International Robot-soccer Association. (En Línea): http://www.fira.net/

[15] FSI All Japan Robot Sumo Tournament. (En Línea): http://www.fsi.co.jp/sumo-e/

[16] Robot Sumo - All Japan Tournament - Kanto Regional Finals. (En Línea): http://video.google.com/videoplay?docid=20653470836006893 71#

[17] KQML. (En Línea): http://en.wikipedia.org/wiki/Multi-agent_system

[18] FIPA ACL. (En línea): http://en.wikipedia.org/wiki/Agent_Communication_Language

[19] BDI. (En Línea): http://en.wikipedia.org/wiki/BDI_software_agent

[20] Bigus J. P. y Bigus Wiley J., Constructing Intelligent Agents Using Java. 2001 (En Línea): http://ai.ee.ccu.edu.tw/ia/notes/ia02.pdf

[21] AI Depot. http://ai-depot.com/articles/ibms-robocode-a-platform-for-learning-ai/. Consultado el 6 de Enero de 2010.

[22] Java Agent Development Framework. (En Línea) http://jade.cselt.it/. Consultado el 15 de Enero de 2010.

[23] Universidad de Alicante, Departamento de Inteligencia artificial. http://www.dccia.ua.es/~pablo/tutorial_agentes/instalacion.html. Consultado el 8 de Enero de 2010.

[24] Ferme E. y Gaspar L., “RCX+PROLOG A platform to use *Lego MindStorms* Robots in Artificial Intelligence courses”. http://math.uma.pt/ferme/Papers/Ferme-Gaspar07.pdf. Consultado el 10 de Enero de 2010.

[25] The Player Project. http://playerstage.sourceforge.net. Consultado el 20 de enero de 2010.

[26] Java Client 2.0 Player/Stage. http://java-player.sourceforge.net/. Consultado el 20 de enero de 2010.

[27] Manual *NetLogo* en español (En Línea) http://sites.google.com/site/manual*NetLogo*/

[28] *NetLogo* Models Library: Sample Models/Mathematics/Probability/ProbLab (En Línea) http://ccl.northwestern.edu/*NetLogo*/models/RandomBasic

[29] Foundation for intelligent physical agents, "FIPA communicative act library specification". (En Línea): http://www.fipa.org/specs/fipa00037/SC00037J.pdf

[30] Sakellariou I., Kefalas Petros K., y Stamatopoulou I. "Enhancing *NetLogo* to Simulate BDI Communicating Agents" http://users.uom.gr/~iliass/projects/*NetLogo*/Papers/Extending_*NetLogo*_SETN08_SVerlag_Camera_ready.pdf. Consultado el 6 de septiembre.

[31] Lego MindStorm, (En Línea): http://mindstorms.lego.com/en-us/Default.aspx

ANEXO 1 – FIGURAS

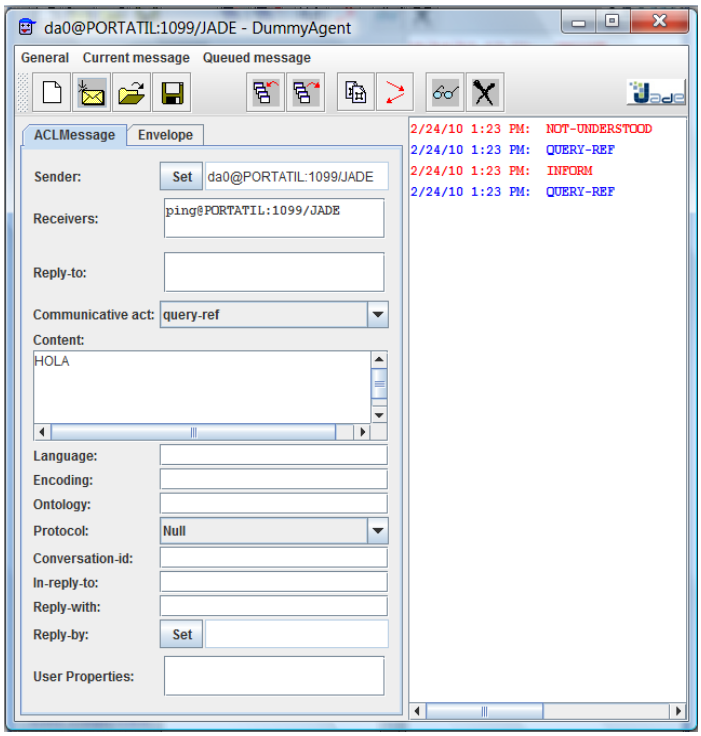

Figura 1. Ventana del *Dummy Agent* que tiene como función proveer una interfaz para el envío de mensajes a agentes en *JADE* mediante protocolo FIPA-ACL sucesor de la KQLM (*Knowledge Query and Manipulation Language*)

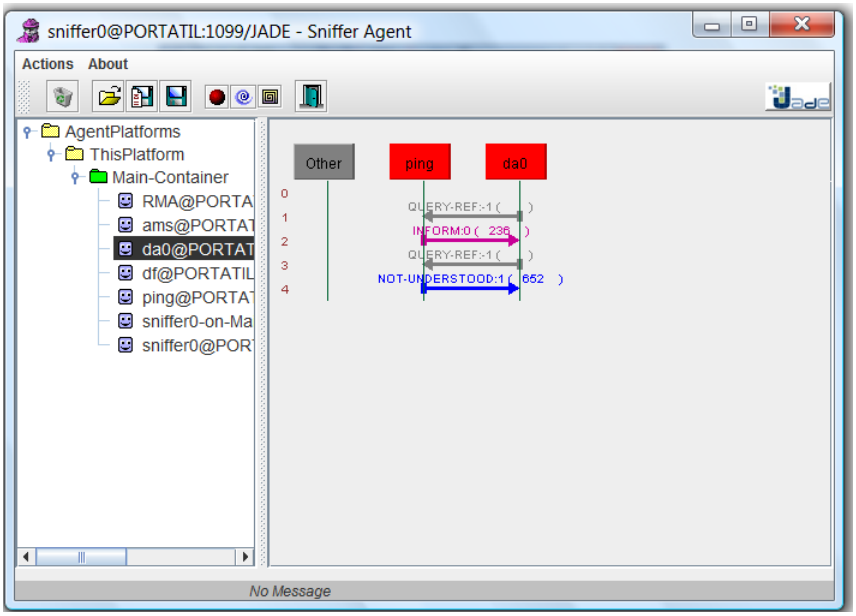

Figura 2. En esta figura se visualiza en agente *Sniffer* el cual esta monitoreando en tiempo real todos los agentes que tengamos en un momento dado en la plataforma, en este caso se muestra el monitoreo de los agentes *Dummy Agent* y *Ping*, en este seguimiento se observa como es el paso de mensajes entre es tos en una línea vertical de tiempo.

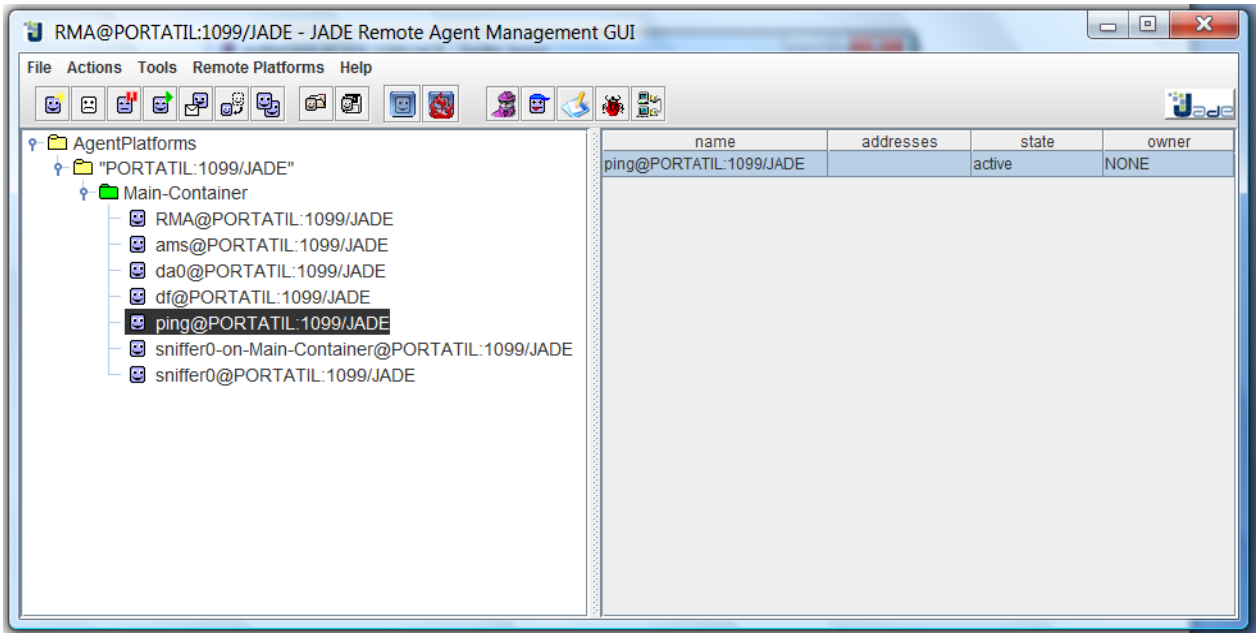

Figura 3. Ventana de interfaz RMS que provee una vista general de los agentes que existen en el *Main Container* Local y también los agentes que se en encuentra en máquinas remotas. En esta interfaz se muestra el nombre la dirección el estado y dueño de los agentes.

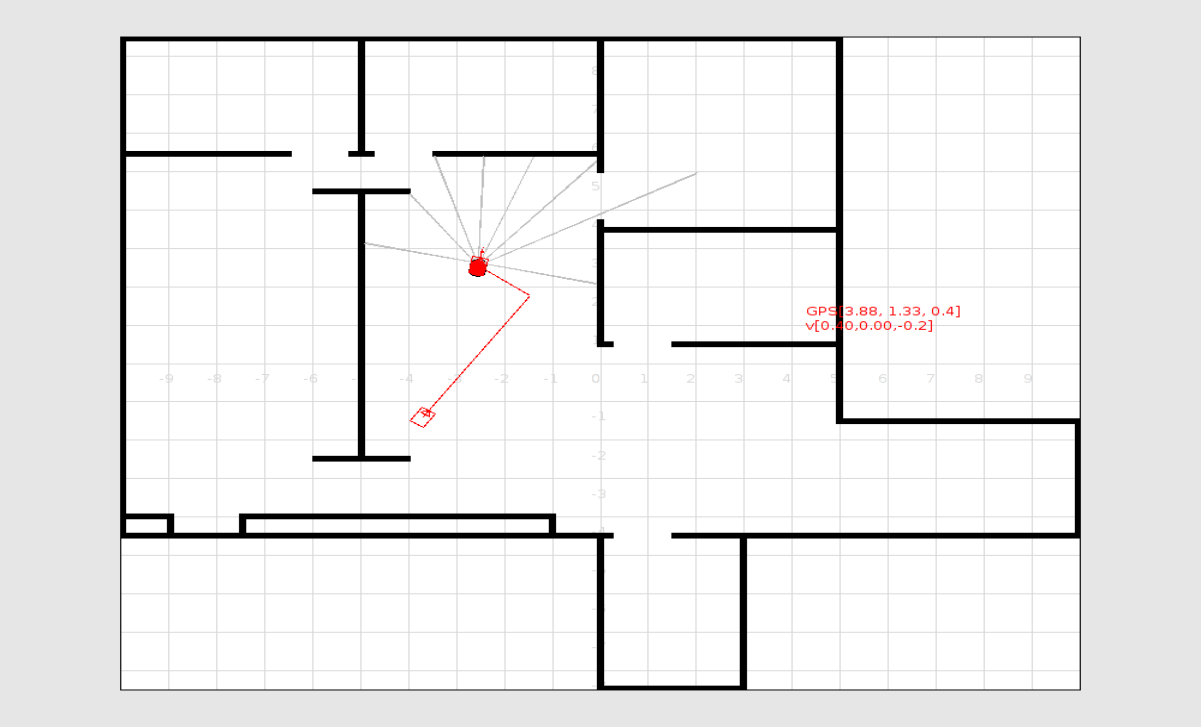

Figura 4. Modelo para el reconocimiento de un escenario en 2D usando *Player* y la interfaz de *Stage*, se basa en tomar la decisión de a dónde dirigirse basado en las distancias que el Robot tiene a los obstáculos.

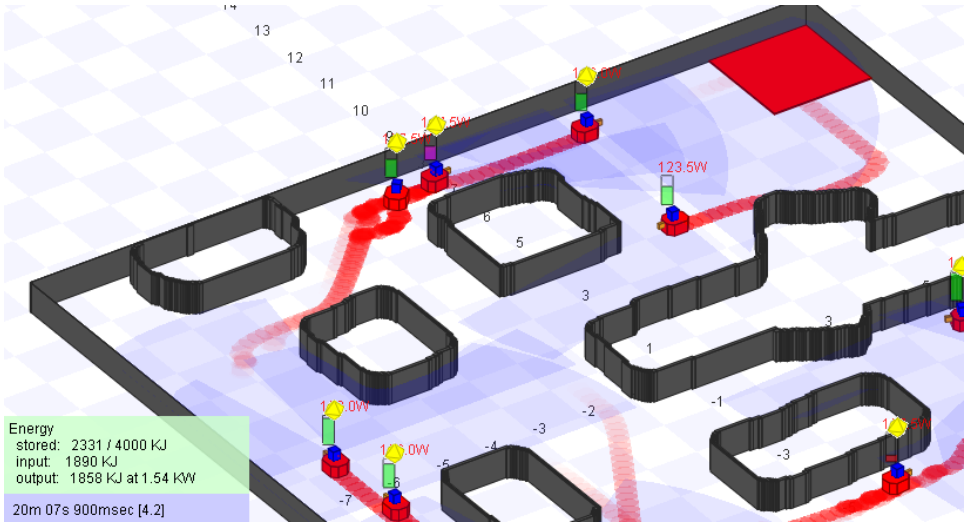

Figura 5. Modelo en perspectiva de un sistema de robots modelado en *Stage* el cual muestra la interacción de varios robots y su estado de energía.

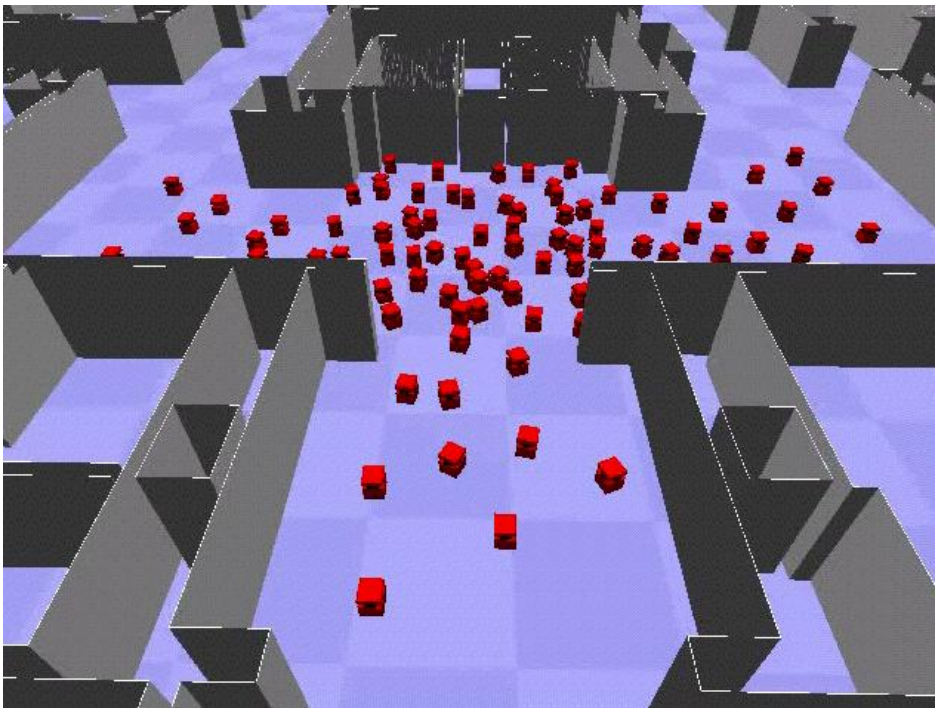

Figura 6. Interfaz de 100 robots que interactúan en un ambiente simulado en *Gazebo*, esta herramienta permite modelar un entorno en 3D y poner en ejecución muchos programas para analizar su comportamiento.

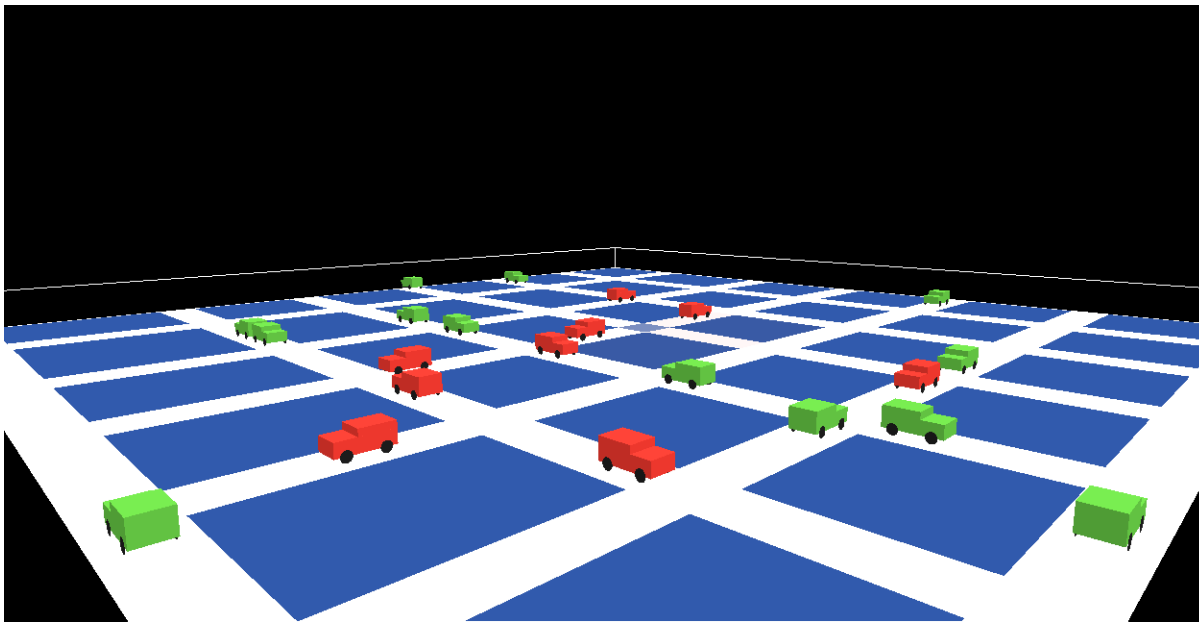

Figura 7. Modelo en 3D de un sistema de 22 vehículos diseñado en *NetLogo* 4.1 la simulación corresponde uno de los casos de simulación presentado en este trabajo.

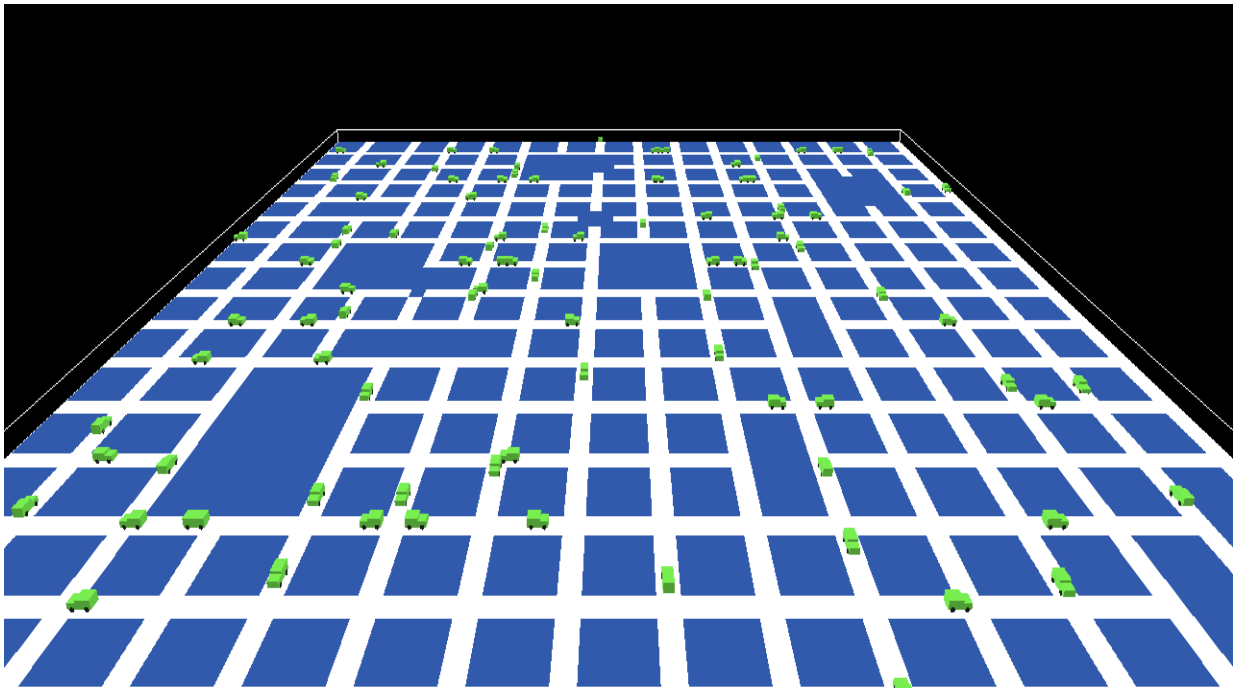

Figura 8. Modelo en 3D diseñado en *NetLogo* 4.1 la simulación corresponde uno de los casos de simulación presentado en este trabajo y muestra el uso del editor de vías diseñado para este proyecto.

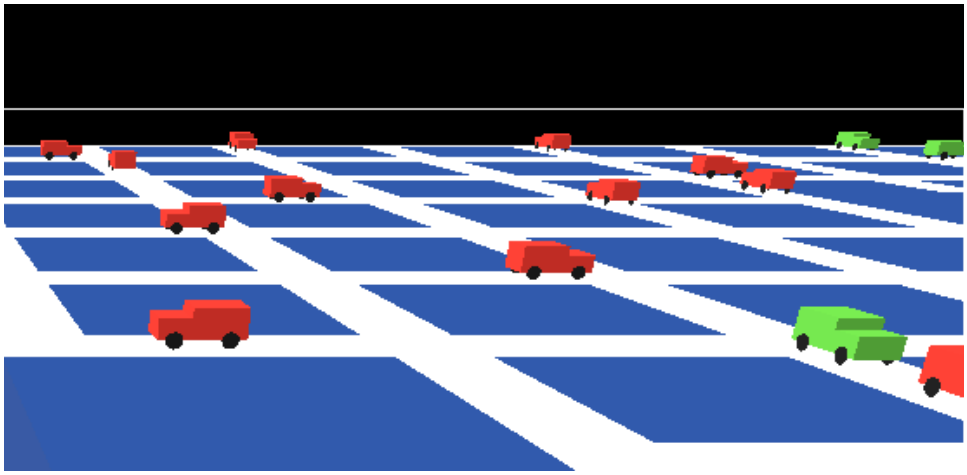

Figura 9. Modelo en 3D diseñado en *NetLogo* 4.1 la simulación corresponde uno de los casos de simulación presentado en este trabajo y muestra el uso del editor de vías diseñado para este proyecto.

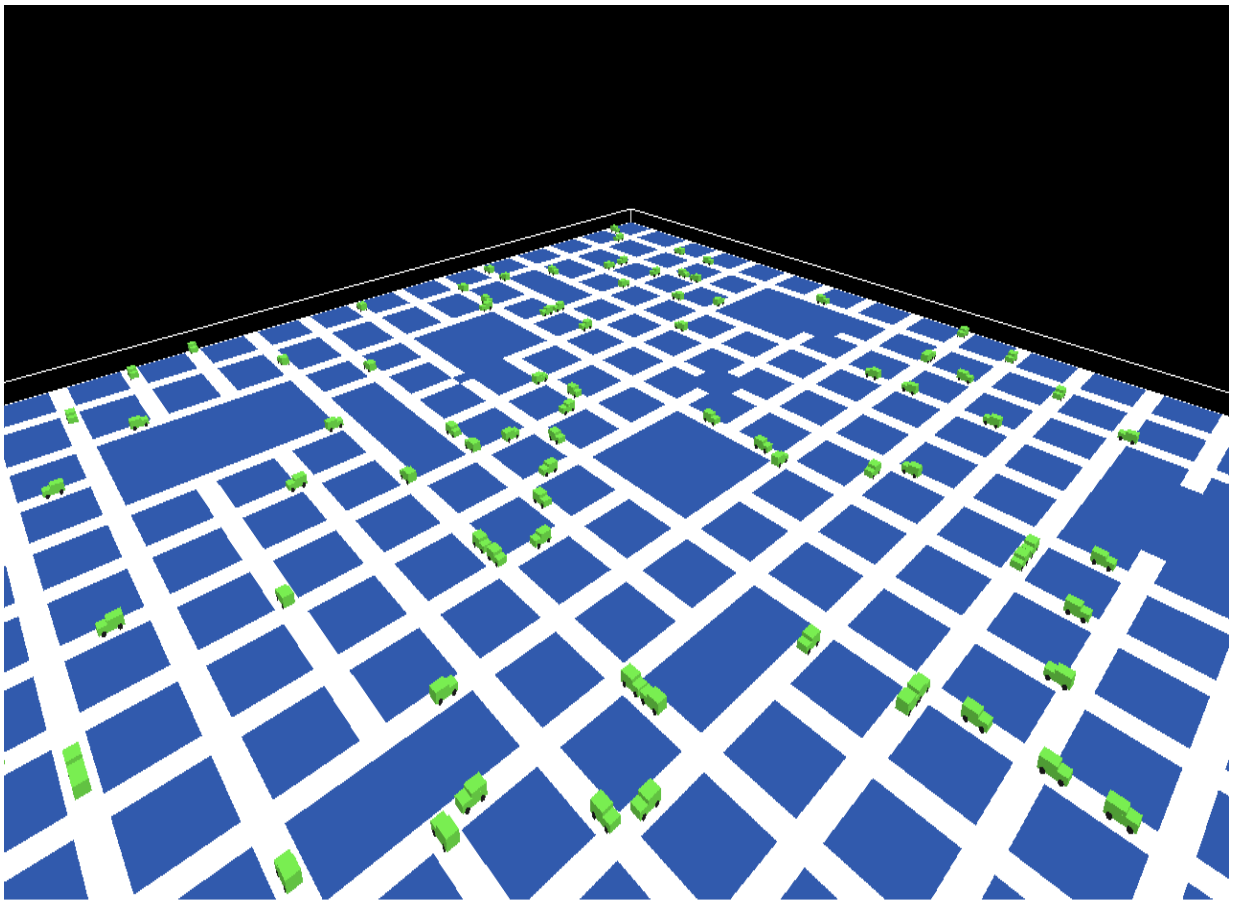

Figura 10. Una vista aérea de un tablero irregular diseñado en este proyecto.

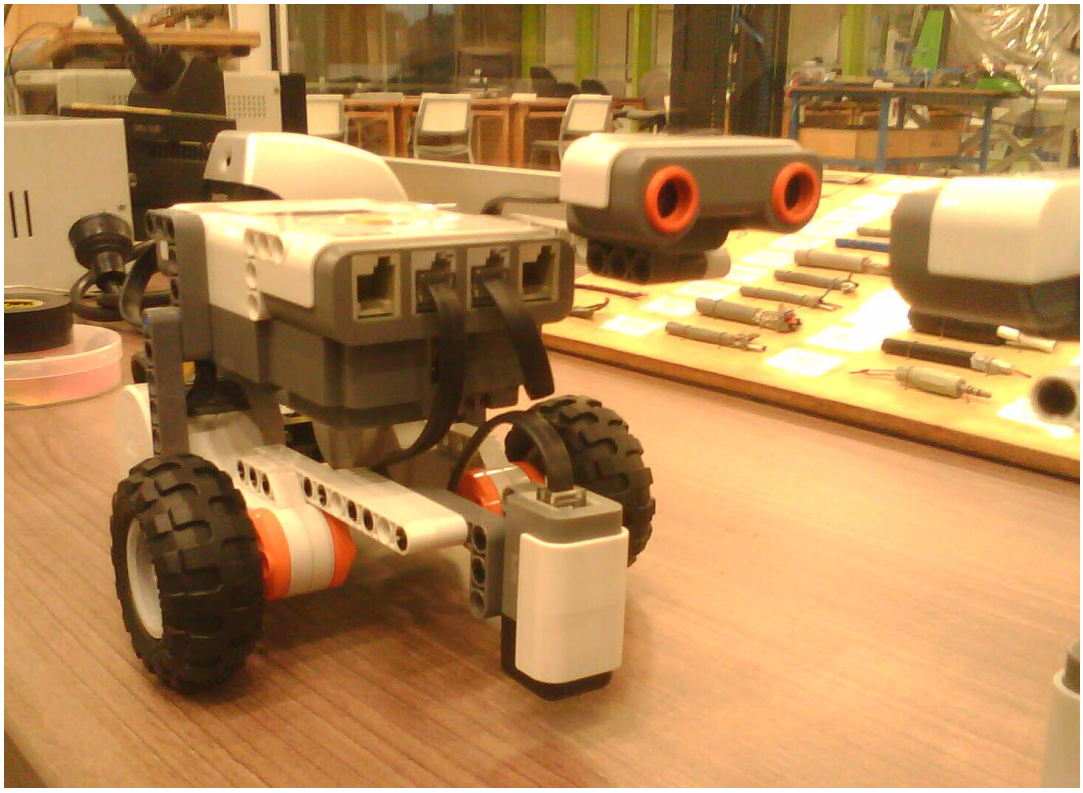

Figura 11. Sistema usado para el demo en *Lego MindStorms*

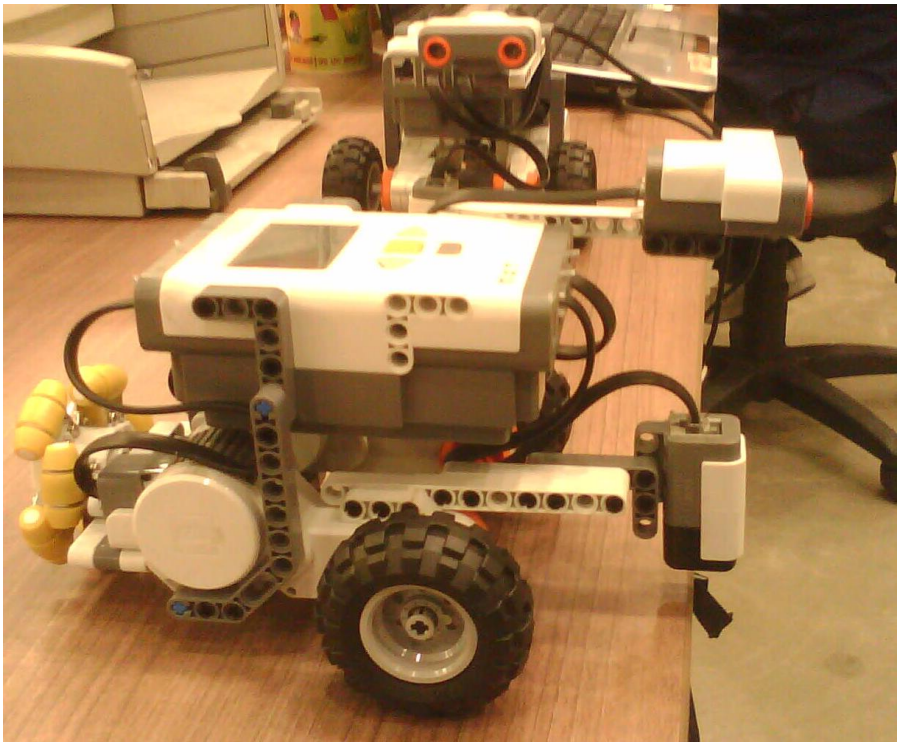

Figura 12. Vista lateral del sistema para el demo en *Lego MindStorms*

ANEXO 2 – DATOS SIMULACIONES

- Número de iteraciones de las 200 Simulaciones del caso Todos buscamos:

74451	138948	77014	88547	99202	107814
171372	98588	87555	153881	147175	110066
119493	130998	115916	103040	172454	94119
120092	132043	85004	85922	13765	129942
87702	108981	129339	98408	84406	90245
199633	116136	127588	75053	85479	87125
103996	166778	253188	107993	75881	86471
71067	116971	86756	72620	98320	156446
118734	109809	156847	80280	84385	91849
95799	137047	133760	135045	79247	77705
107407	125554	119686	131930	141149	116365
146329	169130	88719	94102	80772	88257
138560	131763	134603	144975	88728	75708
132151	118437	82347	139800	113318	121101
99271	134132	114420	90493	95655	140844
109052	77177	144194	92826	82627	81422
113930	131586	102411	98389	100824	134767
86519	67719	100087	103878	79392	82344
143267	125526	111067	124155	65266	88766
90512	141299	105387	92627	95489	186180
113494	142485	110577	107061	113114	81285
95588	77844	118932	162081	143801	135628
140338	104988	84337	95371	92141	120053
86399	117439	85340	109343	144538	118328
90706	121864	76881	72653	127743	77636
112591	121534	143578	117303	248316	90424
103545	67227	135371	90706	107950	125921
128957	131565	112885	149879	119965	115563
86586	130638	133311	100638	129177	86599
113331	79232	158424	113351	102411	147744
122362	97099	70295	94671	102944	109878
104626	113911	108924	148004	91875	114265
98743	122936	103329	94472	69681	162925

83650 98195

- Número de iteraciones de las 200 Simulaciones del caso Te cuento que ya terminamos:

6369 6021 4970 5158 6469 5181 4810 5146 4814 4998 5299
5623 5789 4896 3931 5064 4579 5614 5918 4811 4426 4440
5714 4800 5152 5751 4014 4446 4549 7204 3763 5372 4351
5090 4772 4866 5215 5627 5699 5717 6022 4636 4990 4954
6838 5087 5336 6360 5926 5493 3928 5391 5243 5522 4442
4388 5056 4836 6236 4605 5406 5345 5228 4172 4432 5476
5360 7189 5204 4818 5269 5905 4933 5464 5925 4239 6213
4507 5357 4479 4337 5470 6697 4880 4921 4974 5838 5469
5467 6068 4723 4606 5503 6230 4889 4243 4916 5600 4448
5777 5458 4952 5703 4010 4570 5651 4847 6064 6603 5013
5413 4223 4546 4654 5005 5021 5018 4833 5861 4798 3937
7156 6134 5532 4286 4239 5003 4247 6036 6918 7026 6092
5117 4754 5107 5158 5141 5080 4546 4626 5635 6025 5277
4814 6261 4779 7014 5075 5082 6296 5107 4774 4848 5598
4425 6422 4543 3801 7289 5048 5502 5770 5611 5348 4724
5951 4717 3999 6024 4394 4750 3785 4680 5772 5233 5055
4208 5184 4963 5330 4979 4919 5873 4765 5341 6155 5551
4072 5362 5516 4575 6422 7082 5052 5025 4494 5632 6421
4523

- Número de iteraciones de las 200 Simulaciones del caso Les cuento que ya terminamos:

983 1030 945 972 1631 1349 956 1121 1174 1600 1062
1062 1440 1409 1175 1217 940 1255 998 1084 998 1574
1062 1761 1052 999 1168 1589 1452 998 1359 1009 1089
1164 839 1227 1105 1793 993 908 940 1404 945 2005
1340 988 1185 1195 1387 1034 1025 1009 1014 1876 1158
1046 1820 1116 934 1201 1404 1649 1265 1185 1121 1296
1313 1148 1121 998 1343 1509 1046 1009 1734 1344 1046
1009 1046 1223 1292 1094 2531 998 1184 1026 1163 940
1349 1217 1734 1036 1430 1280 1020 1153 955 1742 2119
1361 1030 977 1436 2246 1275 1073 1857 1047 977 1142

1073 1068 1466 988 1067 983 1531 1286 1569 3286 1371
1302 1302 1749 1191 1435 1020 1435 945 988 978 1169
1793 2506 1068 1317 1014 1104 1665 1024 977 2676 1035
1079 1520 1387 1344 1456 929 993 1025 1695 977 1056
1052 1435 1015 982 961 1094 1244 1126 1169 924 1111
1387 1057 1036 1436 1217 1084 1249 1433 1200 993 1435
1057 924 1296 1764 1562 1425 1068 1621 993 983 1484
1068 1660 1195 961 956 1099 1036 1691 1099 1401 956
1137 998

- Número de iteraciones de las 200 Simulaciones del caso Nos repartimos los sectores y nos comunicamos a través de los nodos:

1173 1046 4164 1167 1509 1303 2251 1680 1067 3403 1134
1218 1107 1052 1620 1333 977 945 1072 1083 1373 1621
1235 2281 1983 1030 999 2940 1600 961 1578 1251 3892
924 1309 1046 1046 1885 994 988 1086 972 1236 1490
1009 1510 3075 1062 924 919 2015 956 930 2081 926
4091 1057 1020 1284 1920 935 1179 5810 1103 1040 1272
1025 2268 1438 1824 1024 1196 1401 2997 1132 966 1150
1099 1746 1733 1058 1474 982 3262 1035 1589 1422 1291
1951 1199 955 1195 4216 2705 1324 951 983 1057 1588
1638 956 945 1476 999 1219 1041 1877 1153 5851 4671
971 1191 1073 1053 1009 2179 982 1196 1185 1471 1506
1240 1642 1506 1759 887 1480 1370 1884 2057 1796 1740
1267 1463 1631 1251 1262 2714 961 2969 1157 1572 1361
1013 1116 2005 2011 2162 1003 1054 1231 1690 1051 1082
1004 978 2405 2000 1420 987 1485 1392 1188 3694 1063
1052 4989 4294 1540 3375 1083 1871 2715 1353 2132 903
950 2743 1143 1933 1102 1196 2193 2186 977 946 1120
982 1512 1094 1001 2069 892 1025 1102 1196 1200 1016
2547 1613

- Número de iteraciones de las 200 Simulaciones del caso les cuento que ya terminamos con mensajes FIPA-ACL:

1209 1068 984 1365 1478 3063 968 915 1263 1187 1166
1090 1522 1181 2243 1466 1262 1048 1393 936 1048 983
1228 1058 968 1505 1464 1026 1686 1080 1032 1591 1106
1732 1261 1778 1158 1202 1355 1143 1138 1198 1455 1021

1479 1231 1148 1446 1368 1373 946 1096 1153 1015 1416
1010 1936 1412 1747 1021 957 1356 1497 968 1095 1026
952 1101 1143 1186 1352 1164 952 1016 1063 1074 1437
1036 2302 1555 1110 1819 919 1084 1999 1053 925 1677
1011 1306 1084 1068 1148 1095 1303 941 1207 1038 983
1287 1090 914 1484 1205 1202 1060 989 978 957
1085 1400 973 925 1116 1104 1227 1015 1322 1048 1090
1251 1037 1430 1095 1079 967 994 1016 962 1708 1197
1793 1084 1085 1031 1327 978 1856 1032 1628 2127 1427
1183 1172 1090 2002 1244 2095 1335 1299 1127 1554 1021
1052 1512 1383 1438 1157 1078 1064 1074 1539 2246 1055
1657 1010 1058 1005 1294 1595 1286 946 999 1244 1517
999 1000 1037 1429 1299 1026 1090 2320 1889 1085 1053
2499 1000 1074 1449 1532 1024 1102 989 989 1324 3746
1069

- Número de iteraciones de las 200 Simulaciones del caso Buscar o no buscar, he ahí el dilema:

2796 1944 2735 1490 2056 1354 1559 2015 2062 2104 2073
1841 1925 1633 1375 1997 1596 3783 2222 1734 1978 1670
1984 1626 1346 1651 1820 2491 1747 1615 2215 1766 2700
1592 2046 1534 2099 1738 1748 1663 2232 1462 1677 1473
1753 1382 1545 1335 3097 1485 1769 2169 1518 1664 1799
2183 1684 1624 1873 1900 1825 1513 1835 1779 3114 1794
1963 2824 2691 1844 1524 1517 2579 1874 1599 2428 1737
1777 1612 2174 1615 1545 1891 1712 2046 1582 1984 1707
1251 1790 2959 1570 2375 1591 1692 2250 1786 2093 1942
2710 2607 1715 2212 2195 1658 1985 2899 1556 2067 1625
2132 2469 1596 2309 3620 1996 1510 3914 2185 1498 1618
1628 2532 2246 1688 2258 2342 2120 1287 2943 1889 1799
2577 1303 1334 1749 1338 1779 2356 1948 2405 1582 1778
2073 1764 1345 1584 1607 1535 1786 1810 2562 3270 1785
1798 1726 1601 1713 1776 1400 1613 2895 2119 2477 1675
1605 1890 1904 3207 1710 1562 1899 1918 1738 1864 2400
3523 1826 2143 1661 1840 1390 1719 1763 1486 1653 1903
2119 1763 2636 1741 2695 1546 1429

ANEXO 3 – CÓDIGOS FUENTE

Caso 1: Todos buscamos (Navegación aleatoria; No comunicación) código completo

```
globals
[
  grid-x-inc           ;; la cantidad de espacios entre 2 vias en direccion x
  grid-y-inc           ;; la cantidad de espacios entre 2 vias en direccion y
  x                    ;; coordenada x del punto rojo
  y                    ;; coordenada y del punto rojo
  i                    ;; contador ciclo de asignacion de mensajes
  j                    ;; coordenada x de la salida
  k                    ;; coordenada y de la salida
  bandera              ;; bandera para mensaje de debe pintar objetivo
  final                ;; bandera para fin de la simulacion
  roads

]

breed [carros carro]
carros-own
[
mensaje?
colorcarro?
objetoxcor?
objetoycor?
]

to setup-globals              ;; inicializar las variables globales
   set grid-x-inc world-width / carreras
   set grid-y-inc world-height / calles
end

to setup-patches              ;; configuracion del tablero,
   ask patches
  [
    set pcolor blue
  ]
  set roads patches with
    [(floor((pxcor + max-pxcor - floor(grid-x-inc - 1)) mod grid-x-inc) = 0)
  or
   (floor((pycor + max-pycor) mod grid-y-inc) = 0)]
  ask roads [ set pcolor white ]
end

to setup-objeto
```

```
   ask patches[
   if (pycor = 1 and pxcor = 0 )
     [
        set pcolor red
        set x 0
        set y 1
      ]

   ]
end

to setup-cars                    ;; configuracion del carro

  create-carros numero-de-carros          ;; crear los carros
 [
     set color green
     posicion-inicial
     set mensaje? false
     set objetoxcor? false
     set objetoycor? false
     set colorcarro? "Verde"
     ifelse (floor((pxcor + max-pxcor - floor(grid-x-inc - 1)) mod grid-x-inc) = 0)
     [
        set heading 180
     ]

     [
        set heading 90
     ]
 ]

end

to posicion-inicial
  move-to one-of roads with [not any? carros-on self]
end

to repintar                    ; color para carro con mensaje gris, carro sin mensaje verde
  if (color = green)
    [
      set color red
      set colorcarro? "Rojo"
    ]



end
```

```
to pintar-obstaculo [ color-obstaculo ]
  ask patch (round mouse-xcor) (round mouse-ycor)
  [
    set pcolor color-obstaculo
    if color-obstaculo = red
    [
      set x round mouse-xcor
      set y round mouse-ycor
    ]
     if color-obstaculo = green
    [
      set j round mouse-xcor
      set k round mouse-ycor
    ]
  ]
end

to pintar
  if mouse-down?
  [
    if color? = "Rojo"
     [ pintar-obstaculo red ]

     if color? = "Azul"
     [ pintar-obstaculo blue ]

     if color? = "Blanco"
     [ pintar-obstaculo white ]

     if color? = "Verde"
     [ pintar-obstaculo green ]
  ]
end

to chequear-objeto-rojo

        if pcolor = red
        [
        set mensaje? true
        set objetoxcor? x
        set objetoycor? y
        repintar

    ]
end

to pintar-salida
  ask patches[
```

```
   if (pycor = min-pycor and pxcor = max-pxcor )
     [
       set pcolor green
       set j round pxcor
       set k round pycor
     ]

  ]
end

to ir-a-salida
  if round xcor = j
    [
     if round ycor = k
     [
      stop
     ]
    ]
    ifelse heading = 90
    [

          ifelse (round ycor = min-pycor)
       [
        ifelse [pcolor] of patch-at 1 0 = white  and [pcolor] of patch-at 0 -1 = white and
[pcolor] of patch-at 0 1 = white
      [   fd 0.4
        set heading heading + one-of [ 0]
        fd 0.6    ]

      [   ifelse [pcolor] of patch-at 1 0 = white  and [pcolor] of patch-at  0 -1 = blue and
[pcolor] of patch-at 0 1 = white
        [  fd 0.4
          set heading heading + one-of [ 0]
          fd 0.6
        ]
        [ ifelse [pcolor] of patch-at 1 0 = white  and [pcolor] of patch-at 0 -1 = white and
[pcolor] of patch-at 0 1 = blue
          [ fd 0.4
            set heading heading + one-of [ 0]
            fd 0.6
          ]
          [ ifelse [pcolor] of patch-at 1 0 = blue  and [pcolor] of patch-at 0 -1 = white and
[pcolor] of patch-at  0 1 = white
            [ fd 0.4
              set heading heading + one-of [ -90]
              fd 0.6
            ]
```

```
          [ ifelse [pcolor] of patch-at 1 0 = blue  and [pcolor] of patch-at 0 -1 = blue and
[pcolor] of patch-at 0 1 = white
            [ fd 0.4
              set heading heading - 90
              fd 0.6
            ]
            [ ifelse [pcolor] of patch-at 1 0 = blue  and [pcolor] of patch-at 0 -1 = white and
[pcolor] of patch-at 0 1 = blue
              [ fd 0.4
                set heading heading + 90
                fd 0.6
              ]
              [ ifelse [pcolor] of patch-at 1 0 = blue  and [pcolor] of patch-at 0 -1 = blue and
[pcolor] of patch-at 0 1 = blue
                [
                  set heading heading - 180
                  fd 0.2
                ]
                [
                  fd 0.2
                ]
              ]
            ]
          ]
        ]
      ]

      ]

      [


       ifelse (round xcor = max-pxcor)
       [
         fd 0.4
         set heading heading + 90
       ]
      [

      ifelse [pcolor] of patch-at 1 0 = white  and [pcolor] of patch-at 0 -1 = white and [pcolor]
of patch-at 0 1 = white
      [   fd 0.4
        set heading heading + one-of [90 0]
        fd 0.6    ]
```

```
    [  ifelse [pcolor] of patch-at 1 0 = white  and [pcolor] of patch-at  0 -1 = blue and
[pcolor] of patch-at 0 1 = white
      [  fd 0.4
        set heading heading + one-of [ 0]
        fd 0.6
      ]
      [ ifelse [pcolor] of patch-at 1 0 = white  and [pcolor] of patch-at 0 -1 = white and
[pcolor] of patch-at 0 1 = blue
        [ fd 0.4
          set heading heading + one-of [90 0]
          fd 0.6
        ]
        [ ifelse [pcolor] of patch-at 1 0 = blue  and [pcolor] of patch-at 0 -1 = white and
[pcolor] of patch-at  0 1 = white
          [ fd 0.4
            set heading heading + one-of [ 90]
            fd 0.6
          ]
          [ ifelse [pcolor] of patch-at 1 0 = blue  and [pcolor] of patch-at 0 -1 = blue and
[pcolor] of patch-at 0 1 = white
            [ fd 0.4
              set heading heading - 90
              fd 0.6
            ]
            [ ifelse [pcolor] of patch-at 1 0 = blue  and [pcolor] of patch-at 0 -1 = white and
[pcolor] of patch-at 0 1 = blue
              [ fd 0.4
                set heading heading + 90
                fd 0.6
              ]
              [ ifelse [pcolor] of patch-at 1 0 = blue  and [pcolor] of patch-at 0 -1 = blue and
[pcolor] of patch-at 0 1 = blue
                [
                  set heading heading - 180
                  fd 0.2
                ]
                [
                  fd 0.2
                ]
              ]
            ]
          ]
        ]
      ]

    ]
  ]
  ]
```

```
  ]

 [
  ifelse (heading = 180)
  [
    ifelse (round ycor = min-pycor)
   [
    fd 0.4
    set heading heading - 90
   ]

   [

    ifelse (round xcor = max-pxcor)
    [

     ifelse [pcolor] of patch-at 0 -1 = white  and [pcolor] of patch-at -1 0 = white and
[pcolor] of patch-at 1 0 = white
     [   fd 0.4
       set heading heading + one-of [ 0]
       fd 0.6    ]
     [   ifelse [pcolor] of patch-at 0 -1 = white  and [pcolor] of patch-at -1 0 = blue and
[pcolor] of patch-at 1 0 = white
       [  fd 0.4
         set heading heading + one-of [ 0]
         fd 0.6
       ]
       [   ifelse [pcolor] of patch-at 0 -1 = white  and [pcolor] of patch-at -1 0 = white and
[pcolor] of patch-at 1 0 = blue
         [ fd 0.4
           set heading heading + one-of [ 0]
           fd 0.6
         ]
         [ ifelse [pcolor] of patch-at 0 -1 = blue  and [pcolor] of patch-at -1 0 = white and
[pcolor] of patch-at 1 0 = white
           [ fd 0.4
             set heading heading + one-of [90]
             fd 0.6
           ]
           [ ifelse [pcolor] of patch-at 0 -1 = blue  and [pcolor] of patch-at -1 0 = blue and
[pcolor] of patch-at 1 0 = white
             [ fd 0.4
               set heading heading - 90
               fd 0.6
             ]
             [ ifelse [pcolor] of patch-at 0 -1 = blue  and [pcolor] of patch-at -1 0 = white and
[pcolor] of patch-at 1 0 = blue
               [ fd 0.4
```

```
                set heading heading + 90
                fd 0.6
              ]
              [ ifelse [pcolor] of patch-at 0 -1 = blue  and [pcolor] of patch-at -1 0 = blue
and [pcolor] of patch-at 1 0 = blue
                [
                  set heading heading - 180
                  fd 0.2
                ]
                [
                  fd 0.2
                ]
              ]
            ]
          ]
        ]
      ]
    ]

  [


      ifelse [pcolor] of patch-at 0 -1 = white  and [pcolor] of patch-at -1 0 = white and
[pcolor] of patch-at 1 0 = white
      [   fd 0.4
        set heading heading + one-of [-90 0]
        fd 0.6    ]
      [   ifelse [pcolor] of patch-at 0 -1 = white  and [pcolor] of patch-at -1 0 = blue and
[pcolor] of patch-at 1 0 = white
        [  fd 0.4
          set heading heading + one-of [-90 0]
          fd 0.6
        ]
        [   ifelse [pcolor] of patch-at 0 -1 = white  and [pcolor] of patch-at -1 0 = white and
[pcolor] of patch-at 1 0 = blue
          [ fd 0.4
            set heading heading + one-of [ 0]
            fd 0.6
          ]
          [ ifelse [pcolor] of patch-at 0 -1 = blue  and [pcolor] of patch-at -1 0 = white and
[pcolor] of patch-at 1 0 = white
            [ fd 0.4
              set heading heading + one-of [-90]
              fd 0.6
            ]
            [ ifelse [pcolor] of patch-at 0 -1 = blue  and [pcolor] of patch-at -1 0 = blue and
[pcolor] of patch-at 1 0 = white
```

```
              [ fd 0.4
                set heading heading - 90
                fd 0.6
              ]
              [ ifelse [pcolor] of patch-at 0 -1 = blue  and [pcolor] of patch-at -1 0 = white and
[pcolor] of patch-at 1 0 = blue
                [ fd 0.4
                  set heading heading + 90
                  fd 0.6
                ]
                [ ifelse [pcolor] of patch-at 0 -1 = blue  and [pcolor] of patch-at -1 0 = blue
and [pcolor] of patch-at 1 0 = blue
                  [
                    set heading heading - 180
                    fd 0.2
                  ]
                  [
                    fd 0.2
                  ]
                ]
              ]
            ]
          ]
        ]
      ]
    ]
    ]  ; cerrar al final
    [ ;; cerrar al final

      ifelse heading = 270
      [
        ifelse [pcolor] of patch-at -1 0 = white  and [pcolor] of patch-at 0 1 = white and
[pcolor] of patch-at 0 -1 = white
        [   fd 0.4
          set heading heading + one-of [-90 90]
          fd 0.6
        ]
        [  ifelse [pcolor] of patch-at -1 0 = white  and [pcolor] of patch-at  0 1 = blue and
[pcolor] of patch-at 0 -1 = white
          [  fd 0.4
            set heading heading + one-of [-90 0]
            fd 0.6
          ]
          [ ifelse [pcolor] of patch-at -1 0 = white  and [pcolor] of patch-at 0 1 = white and
[pcolor] of patch-at 0 -1 = blue
            [ fd 0.4
              set heading heading + one-of [ 0]
```

```
            fd 0.6
           ]
           [ ifelse [pcolor] of patch-at -1 0 = blue  and [pcolor] of patch-at 0 1 = white and
[pcolor] of patch-at 0 -1 = white
            [ fd 0.4
              set heading heading + one-of [-90]
              fd 0.6
            ]
            [ ifelse [pcolor] of patch-at -1 0 = blue  and [pcolor] of patch-at 0 1 = blue and
[pcolor] of patch-at 0 -1 = white
              [ fd 0.4
                set heading heading - 90
                fd 0.6
              ]
              [ ifelse [pcolor] of patch-at -1 0 = blue  and [pcolor] of patch-at 0 1 = white and
[pcolor] of patch-at 0 -1 = blue
                [ fd 0.4
                  set heading heading + 90
                  fd 0.6
                ]
                [ ifelse [pcolor] of patch-at -1 0 = blue  and [pcolor] of patch-at 0 1 = blue and
[pcolor] of patch-at 0 -1 = blue
                  [
                    set heading heading - 180
                    fd 0.2
                  ]
                  [
                    fd 0.2
                  ]
                ]
              ]
            ]
          ]
        ]
      ]

      [ ;; cerrar al final

        if heading = 0
        [
          ifelse [pcolor] of patch-at 0 1 = white  and [pcolor] of patch-at 1 0 = white and
[pcolor] of patch-at -1 0 = white
          [   fd 0.4
            set heading heading + one-of [90]
            fd 0.6    ]
          [   ifelse [pcolor] of patch-at 0 1 = white  and [pcolor] of patch-at 1 0 = blue and
[pcolor] of patch-at -1 0 = white
```

```
          [  fd 0.4
            set heading heading + one-of [ 0]
            fd 0.6
          ]
          [ ifelse [pcolor] of patch-at 0 1 = white  and [pcolor] of patch-at 1 0 = white and
[pcolor] of patch-at -1 0 = blue
            [ fd 0.4
              set heading heading + one-of [90 ]
              fd 0.6
            ]
            [ ifelse [pcolor] of patch-at 0 1 = blue  and [pcolor] of patch-at 1 0 = white and
[pcolor] of patch-at -1 0 = white
              [ fd 0.4
                set heading heading + one-of [ 90]
                fd 0.6
              ]
              [ ifelse [pcolor] of patch-at 0 1 = blue  and [pcolor] of patch-at 1 0 = blue and
[pcolor] of patch-at -1 0 = white
                [ fd 0.4
                  set heading heading - 90
                  fd 0.6
                ]
                [ ifelse [pcolor] of patch-at 0 1 = blue  and [pcolor] of patch-at 1 0 = white and
[pcolor] of patch-at -1 0 = blue
                  [ fd 0.4
                    set heading heading + 90
                    fd 0.6
                  ]
                  [ ifelse [pcolor] of patch-at 0 1 = blue  and [pcolor] of patch-at 1 0 = blue and
[pcolor] of patch-at -1 0 = blue
                    [ fd 0.4
                      set heading heading - 180
                      fd 0.6
                    ]
                    [
                      fd 0.2
                    ]
                  ]
                ]
              ]
            ]
          ]
        ]
      ]
    ]
  ]
```

```
 end

to mover

  ifelse not all? carros [color = red]
[
  ifelse heading = 180
   [
    ifelse [pcolor] of patch-at 0 -1 = white  and [pcolor] of patch-at -1 0 = white and
[pcolor] of patch-at 1 0 = white
     [   fd 0.4
       set heading heading + one-of [90 -90 0]
       fd 0.6    ]
     [   ifelse [pcolor] of patch-at 0 -1 = white  and [pcolor] of patch-at -1 0 = blue and
[pcolor] of patch-at 1 0 = white
       [  fd 0.4
         set heading heading + one-of [-90 0]
         fd 0.6
       ]
       [    ifelse [pcolor] of patch-at 0 -1 = white  and [pcolor] of patch-at -1 0 = white and
[pcolor] of patch-at 1 0 = blue
         [ fd 0.4
           set heading heading + one-of [90 0]
           fd 0.6
         ]
         [ ifelse [pcolor] of patch-at 0 -1 = blue  and [pcolor] of patch-at -1 0 = white and
[pcolor] of patch-at 1 0 = white
           [ fd 0.4
             set heading heading + one-of [-90 90]
             fd 0.6
           ]
           [ ifelse [pcolor] of patch-at 0 -1 = blue  and [pcolor] of patch-at -1 0 = blue and
[pcolor] of patch-at 1 0 = white
             [ fd 0.4
               set heading heading - 90
               fd 0.6
             ]
             [ ifelse [pcolor] of patch-at 0 -1 = blue  and [pcolor] of patch-at -1 0 = white and
[pcolor] of patch-at 1 0 = blue
               [ fd 0.4
                 set heading heading + 90
                 fd 0.6
               ]
               [ ifelse [pcolor] of patch-at 0 -1 = blue  and [pcolor] of patch-at -1 0 = blue and
[pcolor] of patch-at 1 0 = blue
                 [
                   set heading heading - 180
```

```
              fd 0.2
            ]
            [
              fd 0.2
            ]
          ]
        ]
      ]
    ]
  ]
]

[  ; cerrar al final

ifelse heading = 90
[
  ifelse [pcolor] of patch-at 1 0 = white  and [pcolor] of patch-at 0 -1 = white and [pcolor] of patch-at 0 1 = white
    [   fd 0.4
      set heading heading + one-of [90 -90 0]
      fd 0.6    ]

    [   ifelse [pcolor] of patch-at 1 0 = white   and [pcolor] of patch-at   0 -1 = blue and [pcolor] of patch-at 0 1 = white
      [  fd 0.4
        set heading heading + one-of [-90 0]
        fd 0.6
      ]
      [ ifelse [pcolor] of patch-at 1 0 = white   and [pcolor] of patch-at 0 -1 = white and [pcolor] of patch-at 0 1 = blue
        [ fd 0.4
          set heading heading + one-of [90 0]
          fd 0.6
        ]
        [ ifelse [pcolor] of patch-at 1 0 = blue   and [pcolor] of patch-at 0 -1 = white and [pcolor] of patch-at  0 1 = white
          [ fd 0.4
            set heading heading + one-of [-90 90]
            fd 0.6
          ]
          [ ifelse [pcolor] of patch-at 1 0 = blue   and [pcolor] of patch-at 0 -1 = blue and [pcolor] of patch-at 0 1 = white
            [ fd 0.4
              set heading heading - 90
              fd 0.6
            ]
```

```
          [ ifelse [pcolor] of patch-at 1 0 = blue  and [pcolor] of patch-at 0 -1 = white and
[pcolor] of patch-at 0 1 = blue
            [ fd 0.4
              set heading heading + 90
              fd 0.6
            ]
            [ ifelse [pcolor] of patch-at 1 0 = blue  and [pcolor] of patch-at 0 -1 = blue and
[pcolor] of patch-at 0 1 = blue
              [
                set heading heading - 180
                fd 0.2
              ]
              [
                fd 0.2
              ]
            ]
          ]
        ]
      ]

    ]
  ]

  [ ;; cerrar al final

    ifelse heading = 270
    [
     ifelse [pcolor] of patch-at -1 0 = white  and [pcolor] of patch-at 0 1 = white and
[pcolor] of patch-at 0 -1 = white
      [   fd 0.4
        set heading heading + one-of [90 -90 0]
        fd 0.6    ]

      [  ifelse [pcolor] of patch-at -1 0 = white  and [pcolor] of patch-at  0 1 = blue and
[pcolor] of patch-at 0 -1 = white
        [  fd 0.4
          set heading heading + one-of [-90 0]
          fd 0.6
        ]
        [ ifelse [pcolor] of patch-at -1 0 = white  and [pcolor] of patch-at 0 1 = white and
[pcolor] of patch-at 0 -1 = blue
          [ fd 0.4
            set heading heading + one-of [90 0]
            fd 0.6
          ]
          [ ifelse [pcolor] of patch-at -1 0 = blue  and [pcolor] of patch-at 0 1 = white and
[pcolor] of patch-at 0 -1 = white
```

```
            [ fd 0.4
              set heading heading + one-of [-90 90]
              fd 0.6
            ]
            [ ifelse [pcolor] of patch-at -1 0 = blue  and [pcolor] of patch-at 0 1 = blue and
[pcolor] of patch-at 0 -1 = white
              [ fd 0.4
                set heading heading - 90
                fd 0.6
              ]
              [ ifelse [pcolor] of patch-at -1 0 = blue  and [pcolor] of patch-at 0 1 = white and
[pcolor] of patch-at 0 -1 = blue
                [ fd 0.4
                  set heading heading + 90
                  fd 0.6
                ]
                [ ifelse [pcolor] of patch-at -1 0 = blue  and [pcolor] of patch-at 0 1 = blue and
[pcolor] of patch-at 0 -1 = blue
                  [
                    set heading heading - 180
                    fd 0.2
                  ]
                  [
                    fd 0.2
                  ]
                ]
              ]
            ]
          ]
        ]
      ]

      [ ;; cerrar al final

        if heading = 0
        [
          ifelse [pcolor] of patch-at 0 1 = white  and [pcolor] of patch-at 1 0 = white and
[pcolor] of patch-at -1 0 = white
          [   fd 0.4
            set heading heading + one-of [90 -90 0]
            fd 0.6    ]
          [   ifelse [pcolor] of patch-at 0 1 = white  and [pcolor] of patch-at 1 0 = blue and
[pcolor] of patch-at -1 0 = white
            [  fd 0.4
              set heading heading + one-of [-90 0]
              fd 0.6
            ]
```

```
          [ ifelse [pcolor] of patch-at 0 1 = white  and [pcolor] of patch-at 1 0 = white and
[pcolor] of patch-at -1 0 = blue
            [ fd 0.4
              set heading heading + one-of [90 0]
              fd 0.6
            ]
            [ ifelse [pcolor] of patch-at 0 1 = blue  and [pcolor] of patch-at 1 0 = white and
[pcolor] of patch-at -1 0 = white
              [ fd 0.4
                set heading heading + one-of [-90 90]
                fd 0.6
              ]
              [ ifelse [pcolor] of patch-at 0 1 = blue  and [pcolor] of patch-at 1 0 = blue and
[pcolor] of patch-at -1 0 = white
                [ fd 0.4
                  set heading heading - 90
                  fd 0.6
                ]
                [ ifelse [pcolor] of patch-at 0 1 = blue  and [pcolor] of patch-at 1 0 = white and
[pcolor] of patch-at -1 0 = blue
                  [ fd 0.4
                    set heading heading + 90
                    fd 0.6
                  ]
                  [ ifelse [pcolor] of patch-at 0 1 = blue  and [pcolor] of patch-at 1 0 = blue and
[pcolor] of patch-at -1 0 = blue
                    [ fd 0.4
                      set heading heading - 180
                      fd 0.6
                    ]
                    [
                      fd 0.2
                    ]
                  ]
                ]
              ]
            ]
          ]
        ]
      ]
    ]
  ]
  ]

    [
      ir-a-salida
```

```
    ]

end
to setup-plot

  set-current-plot "Vehiculos vs Iteraciones"
  set-plot-y-range 0 (numero-de-carros)
  set-plot-x-range ticks (ticks + 1)
end

to update-plot
   set-current-plot-pen "sin completar la tarea"
   plot count turtles with [color = red]
   set-current-plot-pen "completaron la tarea"
   plot count turtles with [color != red]
  ;;numero-de-carros - count turtles with [color = red]


end




to crear
  ca
  setup-globals
  setup-patches
  set-default-shape carros "car"
  set x -500
  set y -500
  set final false
  set bandera true
  setup-cars
  setup-plot

end


to empezar

    ifelse (x = -500 and y = -500  )
  [
   if bandera = true
   [
    user-message "Debe pintar el objeto rojo en una interseccion"
    set bandera  false
   ]
  ]
```

```
 [
 if all? carros [ color = red ]
   [
     pintar-salida
     if final = false
     [
     if  all? carros [pxcor = j and pycor = k]
       [
         user-message "termino la simulacion!"
         set final true
       ]
     ]
   ]

 if final = false

   [
     ask carros
     [
       mover
       chequear-objeto-rojo
     ]

  tick
   update-plot

   ]
 ]

end
```

Caso 2: Te cuento que ya terminamos (Navegación aleatoria; Comunicación incremental de resultado) Código de la función de transmisión del mensaje.

```
to pegar-color

      ask carros in-radius 1              ;;   ubica un carro con el que hizo contacto fisico

      [

        if (color = green)                ;; chequea el estado del carro encontrado

        [

           Repintar                       ;; cambia el color del vehiculo
```

```
        set mensaje? True              ;; envía el mensaje
        set objetoxcor? x              ;; transmite información de las cordenadas del objeto
        set objetoycor? y              ;; transmite información de las cordenadas del objeto
      ]
    ]
End
```

Les cuento a todos que ya terminamos (Navegación aleatoria; Comunicación global de resultado) Código de la función de aviso a todos los vehículos

```
to pegar-color-a-todos
    ask carros in-radius 256     ;;   ubica los carros en todo el tablero
    [
     if (color = green)        ;; chequea el estado del carros encontrados
     [
       Repintar                       ;; cambia el color de los vehiculos
       set mensaje? True              ;; envía el mensaje
       set objetoxcor? x              ;; transmite información de las cordenadas del objeto
       set objetoycor? y              ;; transmite información de las cordenadas del objeto
     ]
    ]
End
```

Nos repartimos los sectores y nos comunicamos a través de los nodos (Navegación por sectores; Comunicación incremental de resultado)

Código de función que chequea el punto rojo y envía los respectivos mensajes

```
to chequear-objeto-rojo                ;; función para detectar el objeto buscado
    if pcolor = red                      ;; instrucciones para cambiar las variables del vehiculo
ganador
    [
     set mensaje? true
     repintar
     set zonaganadora zonaroad?
     set carroganador who
     ask carros with [ zonacarro? = zonaganadora ]        ;; mensaje a los carros de su zona
    [
           set color red
          set zonaganadora? zonaganadora
          set carroganador? carroganador

    ]
       ask nodos                  ;; mensaje a los nodos de las otras zonas
       [
         set color red
         set mensaje? true
        set zonaganadora? zonaganadora
       ]
     comunicar-a-los-otros-vehiculos       ;; mensaje de los nodos a sus vehiculos
    ]
 end
```

Les cuento a todos que ya terminamos con mensajes FIPA-ACL (Navegación aleatoria; Comunicación global de resultado) Código de funciones que muestran el comportamiento de los vehículos, la detección de colisiones y función para el envío de mensajes.

```
to carro-behaviour              ;; Comportamiento del vehiculo
    execute-intentions
  listen-to-messages
end
to look-for-objeto              ;; Primera intention
    mover
    if detect-carro             ;; detecar las colisiones
    [
      turn-away
    ]
  if  (chequear-objeto-rojo = true)
  [
    remove-intention ["look-for-objeto" "false"]     ;; cambio de intention
  ]
end


to comunicar-a-los-otros                                ;; envío del mensaje
   broadcast-to carros add-content ( "objeto encontrado")
  create-message "inform"
  remove-intention ["comunicar-a-los-otros" "false"]
end
```

Buscar o no buscar, he ahí el dilema (Navegación para maximizar utilidad esperada; Comunicación por mensaje a todos usando FIPA) Código de asignación de creencias y actualización de creencias e intenciones .

```
to asignar-creencia-inicial
      ifelse (utilidad-esperada < 333)
     [
       add-belief (list "estado" "pesimista")
     ]
     [
       ifelse (utilidad-esperada < 666)
       [
         add-belief (list "estado" "indeciso")
       ]
       [
         add-belief (list "estado" "optimista")
       ]
     ]
 end

to actualizar-creencias

  if (current-intention != "ir-a-salida")
  [
    ifelse (utilidad-esperada > 333 and utilidad-esperada < 333.03)
     [
       show "cambie de belief de indeciso a pesimista"
```

```
      remove-belief [["estado" "indeciso"]]
      add-belief (list "estado" "pesimista")
      remove-intention ["look-for-objeto" "false"]
      remove-intention ["comunicar-a-los-otros" "false"]

    ]
    [
      ifelse (utilidad-esperada > 666 and utilidad-esperada < 666.03 )
      [
        show "cambie de belief de optimista a indeciso"
        remove-belief [["estado" "optimista"]]
        add-belief (list "estado" "indeciso")
        set decision random 9

        ifelse (decision  > 4 )
        [
        remove-intention ["look-for-objeto" "false"]
        remove-intention ["comunicar-a-los-otros" "false"]
        ]
         [
         ]
      ]
        [
        ]
    ]
]
End
```