TESIS
II2005
A66
v.2

UNIVERSIDAD CATÓLICA ANDRÉS BELLO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

Simulación del mecanismo de búsqueda y recolección de alimento de las colonias de hormigas utilizando Algoritmos Evolutivos y Bioinspirados con enfoque multiagente

APÉNDICE DEL TRABAJO ESPECIAL DE GRADO
presentado ante la
UNIVERSIDAD CATÓLICA ANDRÉS BELLO
Como parte de los requisitos para optar al título de

INGENIERO EN INFORMÁTICA

REALIZADO POR Juan Carlos Arvelo Flores
Ricardo Casanova Nébola-Kopak

PROFESOR GUIA Dr. Wilmer Pereira

FECHA Caracas, 29 de Noviembre de 2005

INDICE DE CONTENIDO

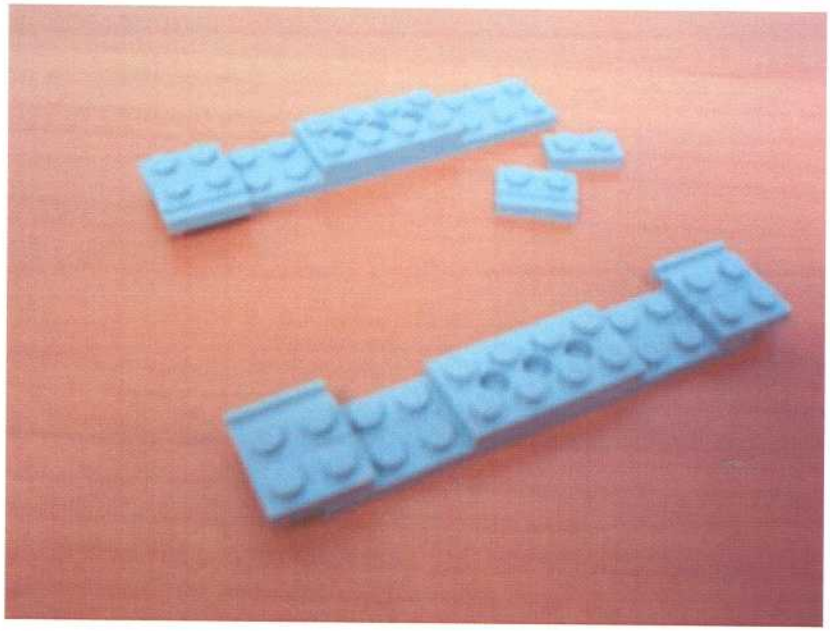
APÉNDICE A: CONSTRUCCIÓN DEL ROBOT 1
CONSTRUCCIÓN DE LA BASE 1
CONSTRUCCIÓN DE LAS RUEDAS (TIPO ORUGA) 11
CONSTRUCCIÓN FINAL DEL ROBOT 20
APÉNDICE B: MLCAD 28
APÉNDICE C: API DE WINDOWS 29
API Win32 29
Llamadas a las funciones del API de Windows 30
Librerías Dinámicas 31
Declaración de una DLL 31
Algunas funciones de la API 32
Ejemplo de una llamada al API de Windows 33
APÉNDICE D: SISTEMAS OPERATIVOS UTILIZADOS PARA PROGRAMAR EL RCX 35
APÉNDICE E: SIMULADOR DE COLONIAS DE HORMIGAS 37
APÉNDICE F: ALGORITMOS 42
Q-Learning 42
Value Iteration 42
Ant-Learning 43

APÉNDICE A: CONSTRUCCIÓN DEL ROBOT

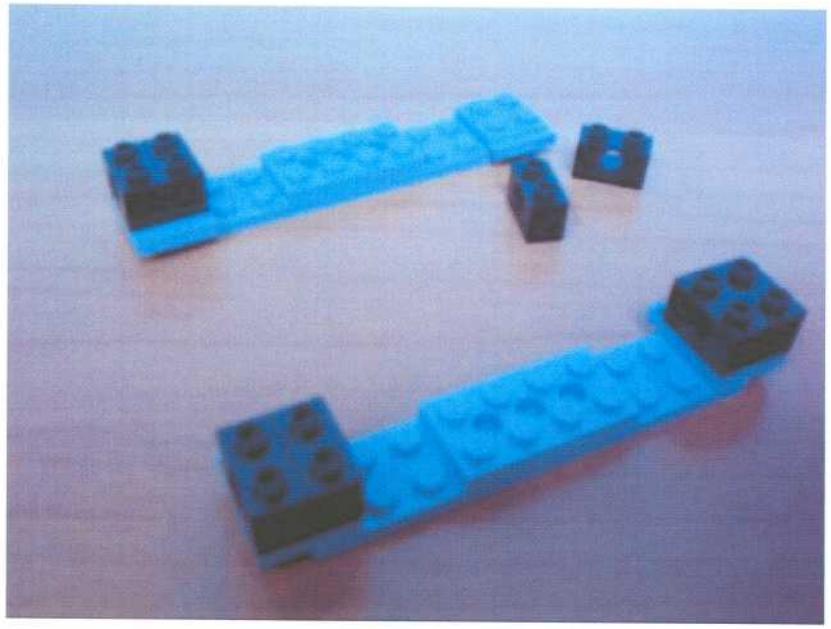
CONSTRUCCIÓN DE LA BASE

1

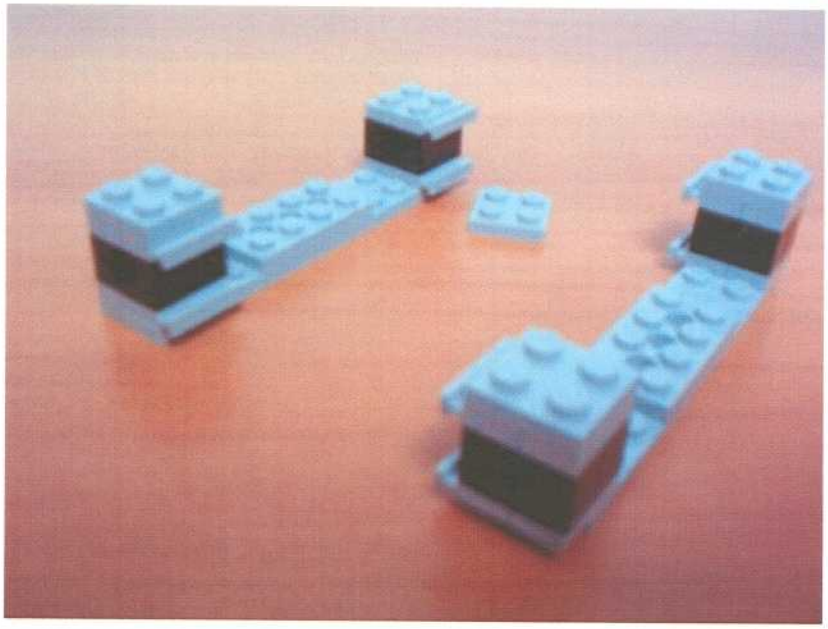
2



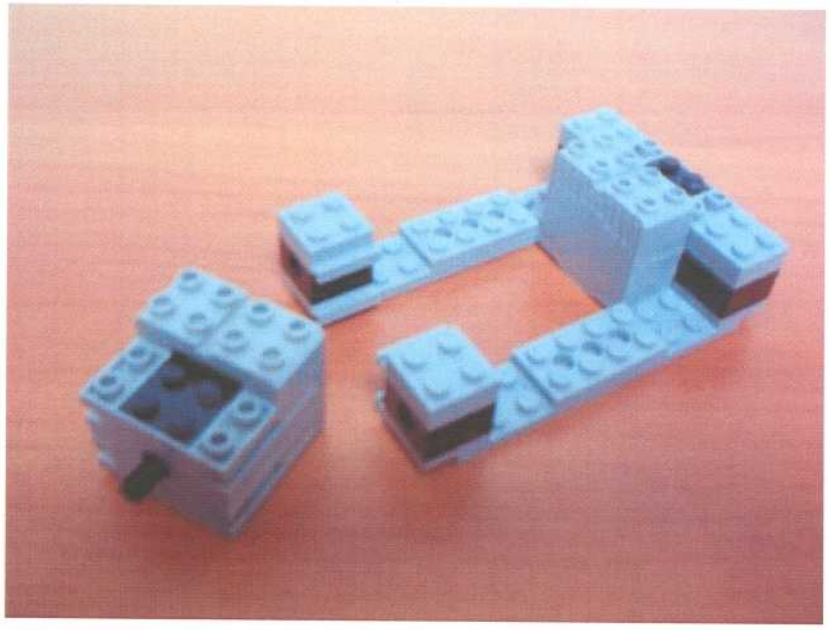
3



4

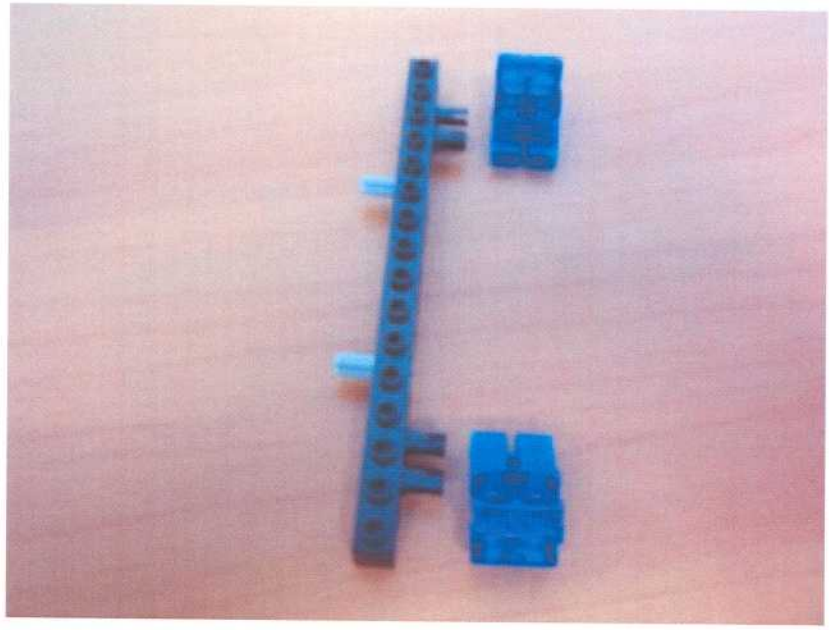


5

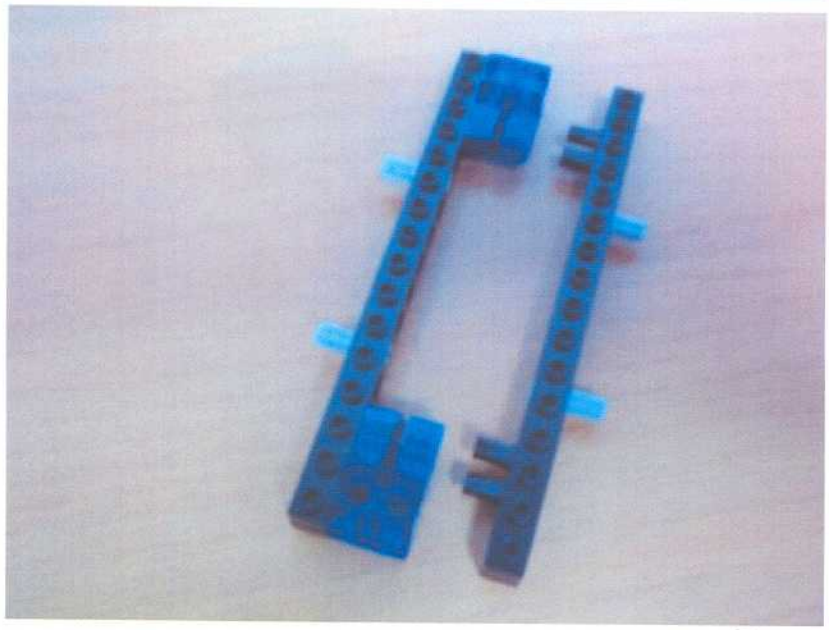


6

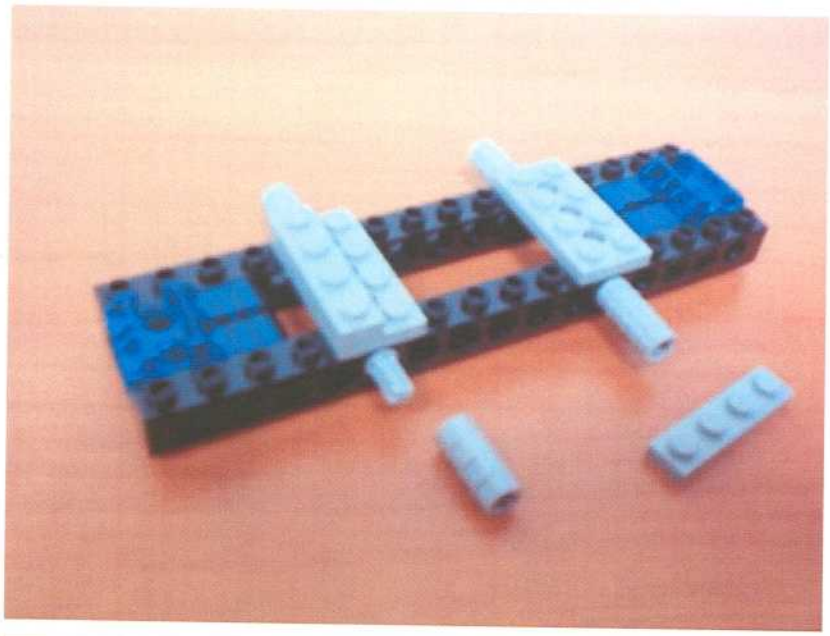
7



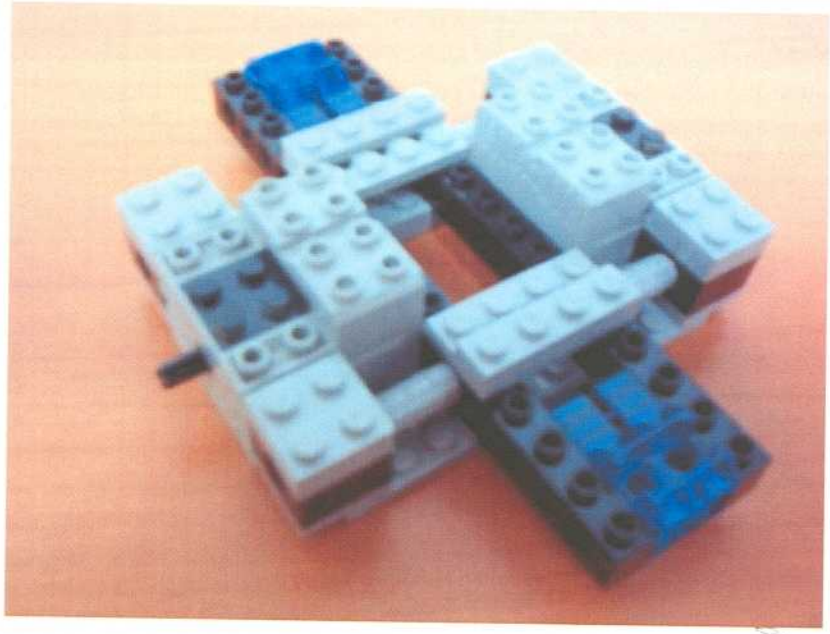
8



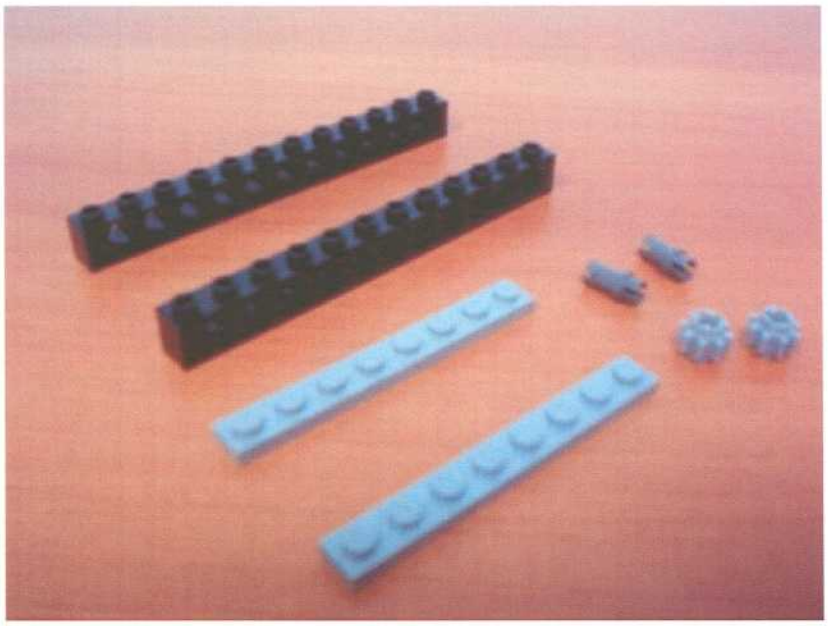
9



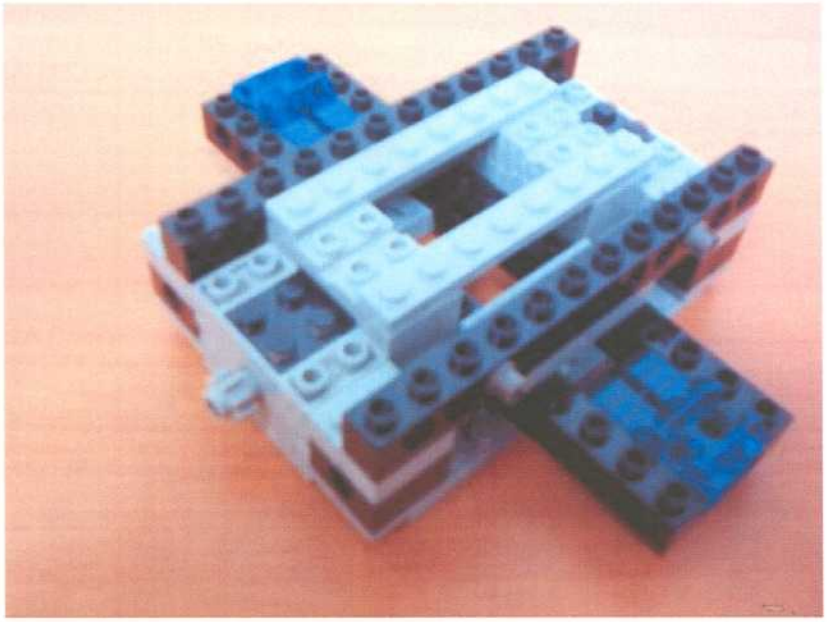
10



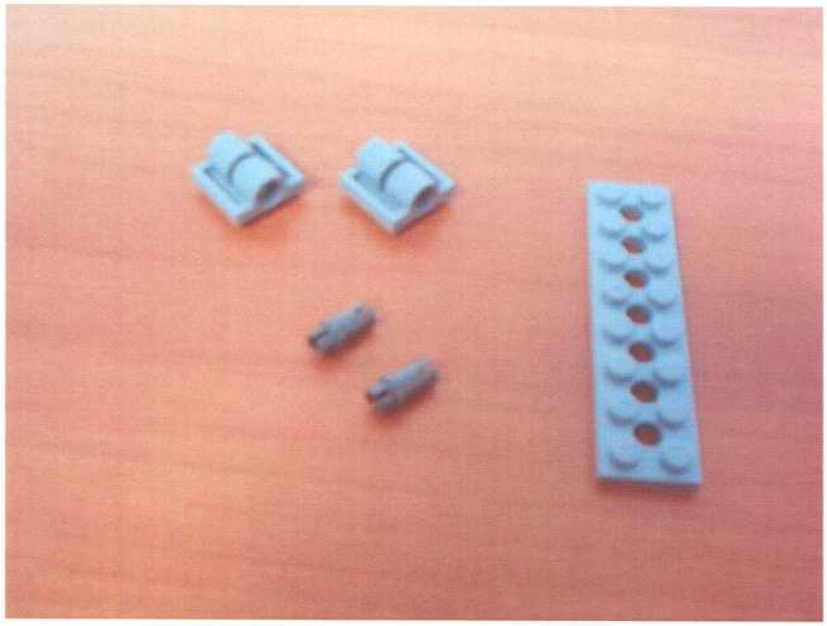
11



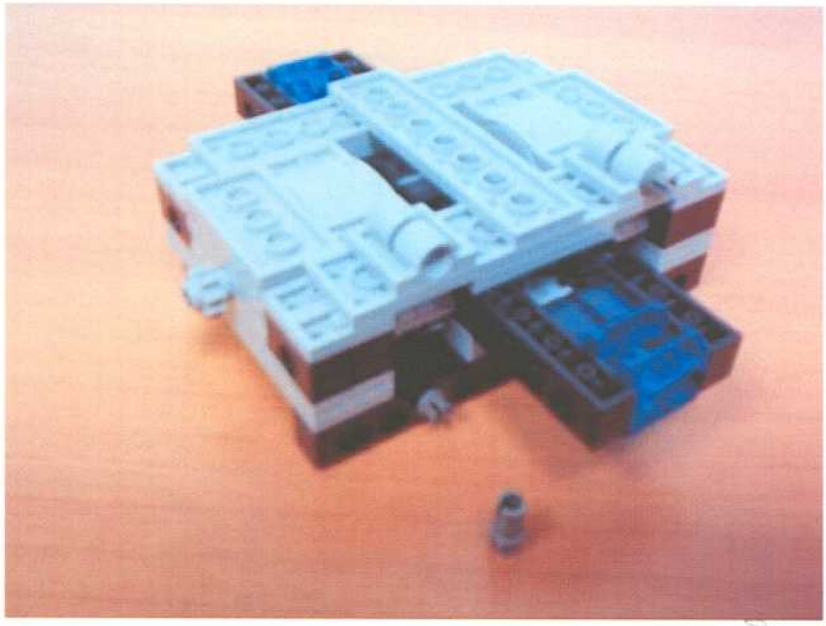
12



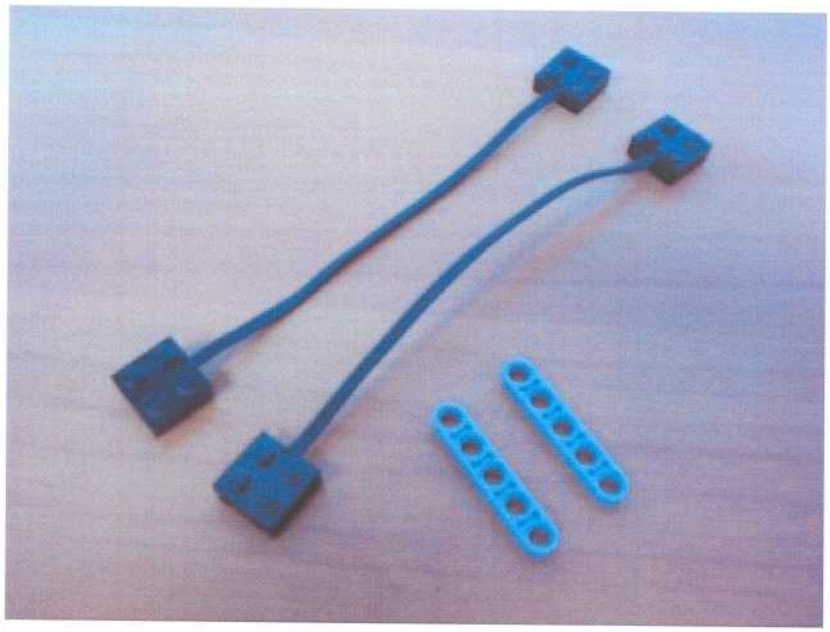
13



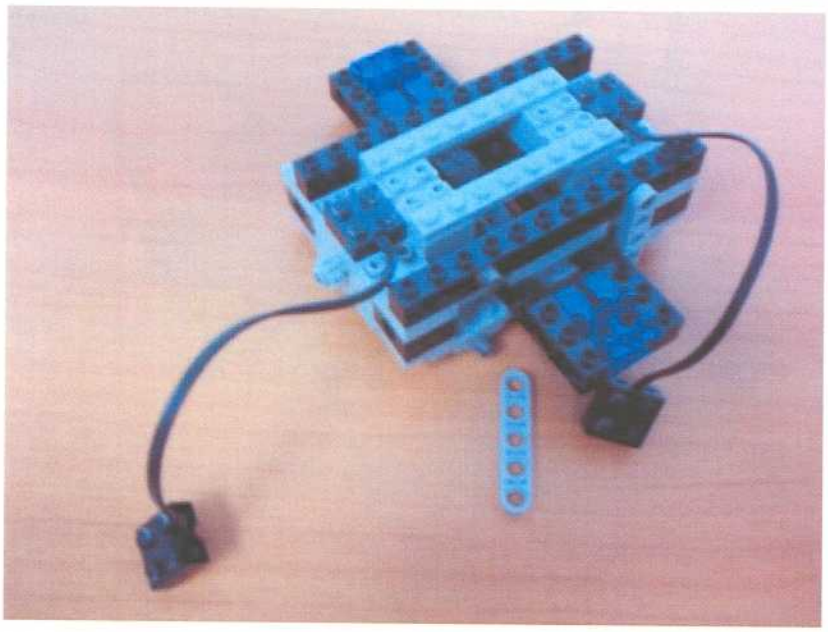
14



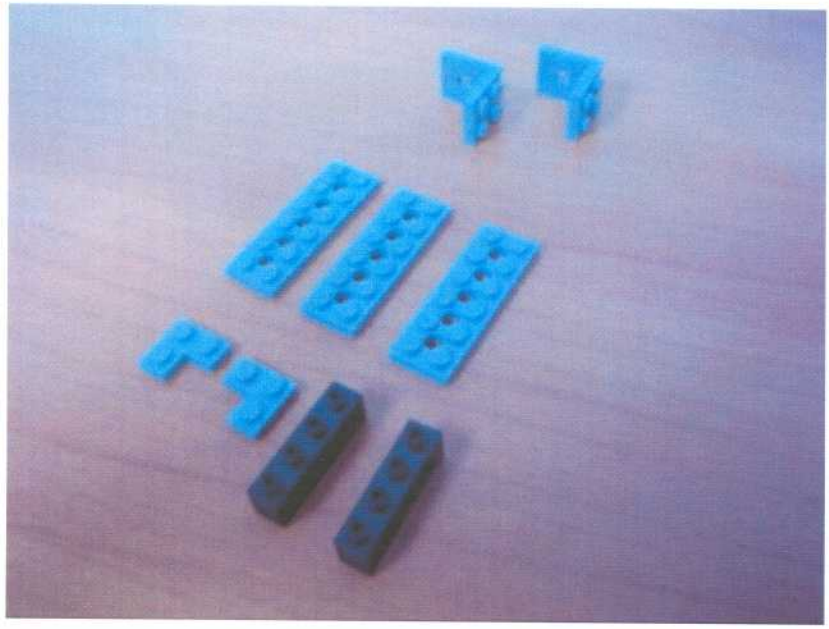
15



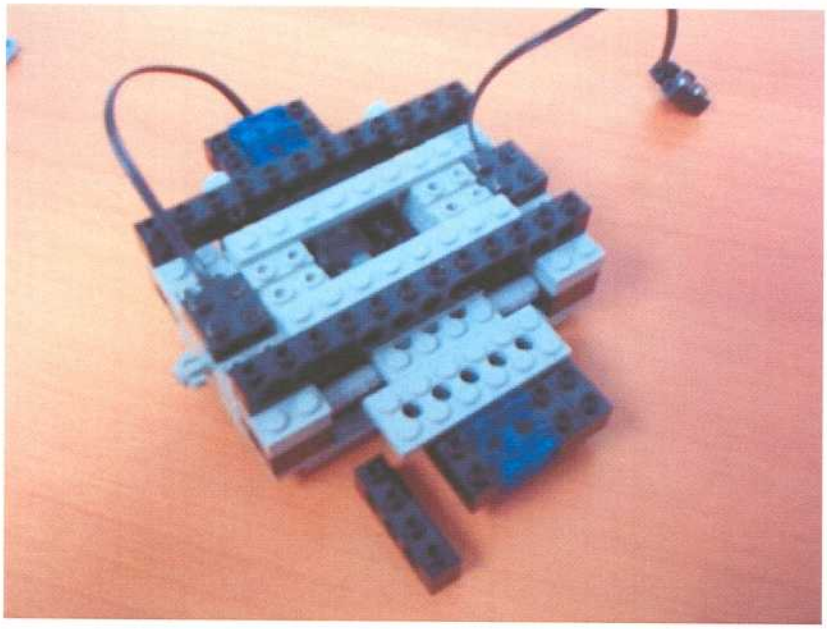
16



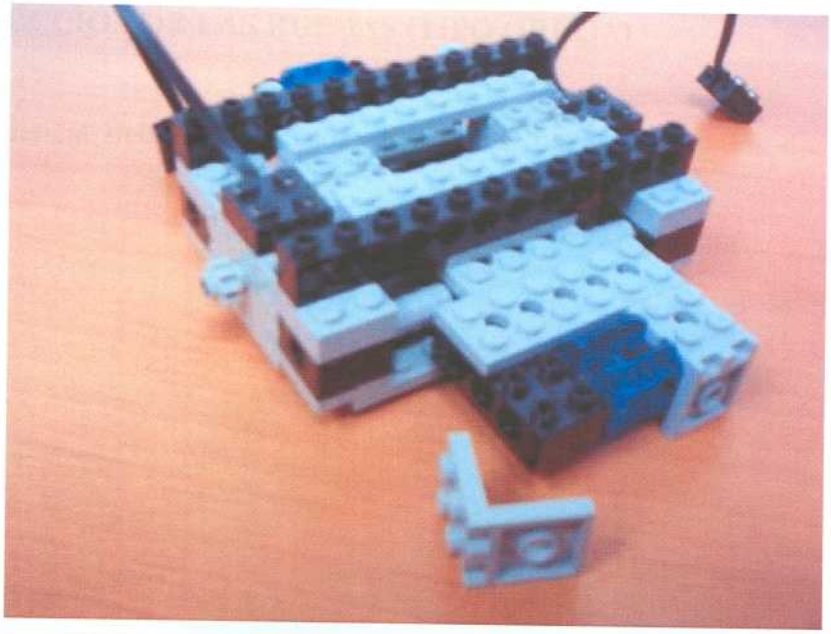
17



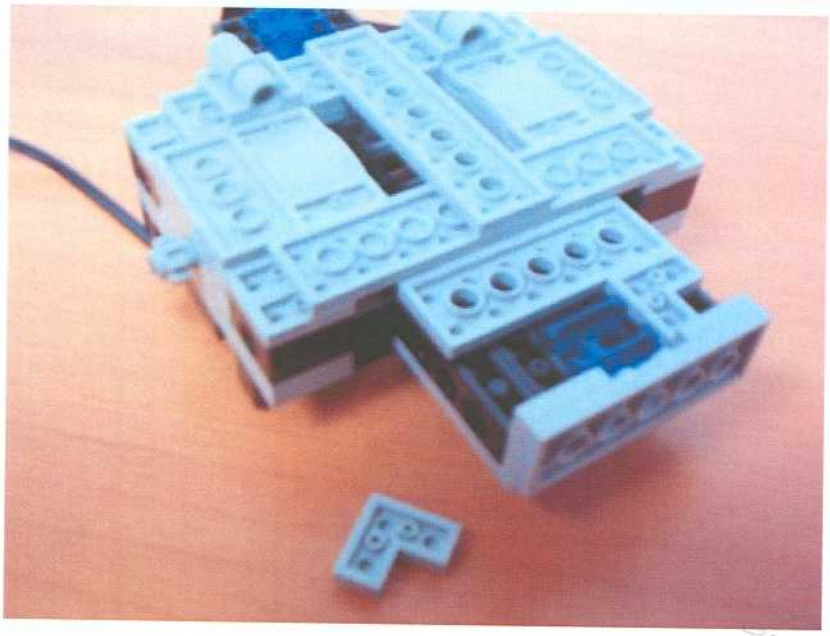
18



19



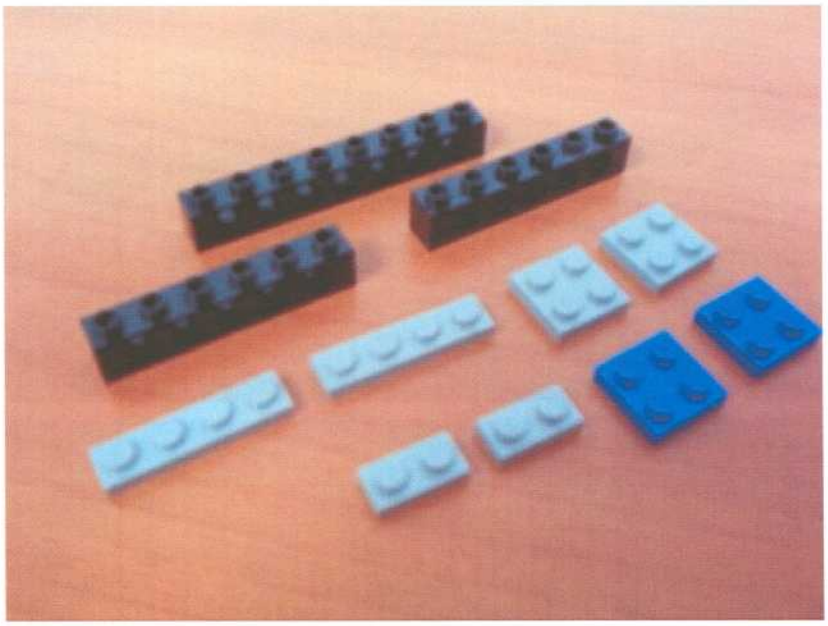
20



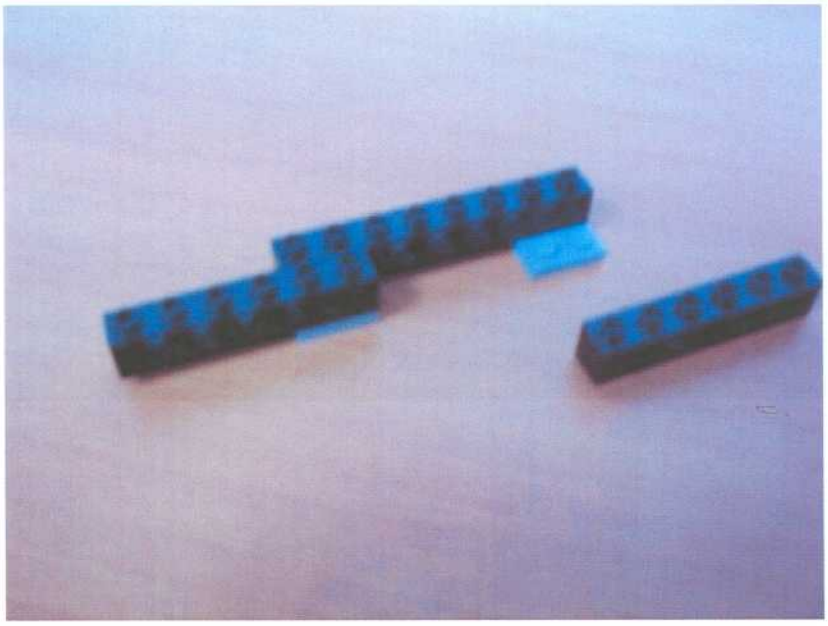
CONSTRUCCIÓN DE LAS RUEDAS (TIPO ORUGA)

NOTA: Esta secuencia debe ser realizada dos veces, para obtener las dos ruedas tipo oruga.

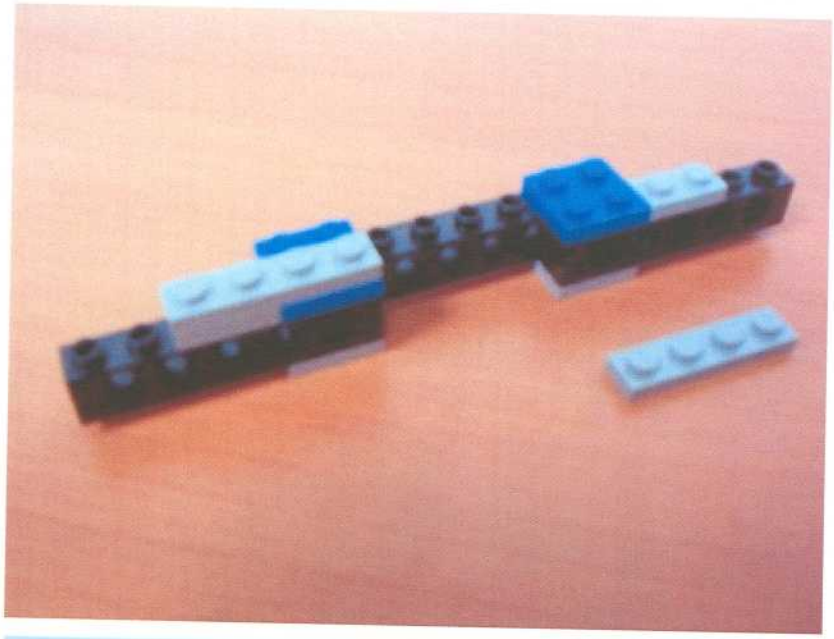
1



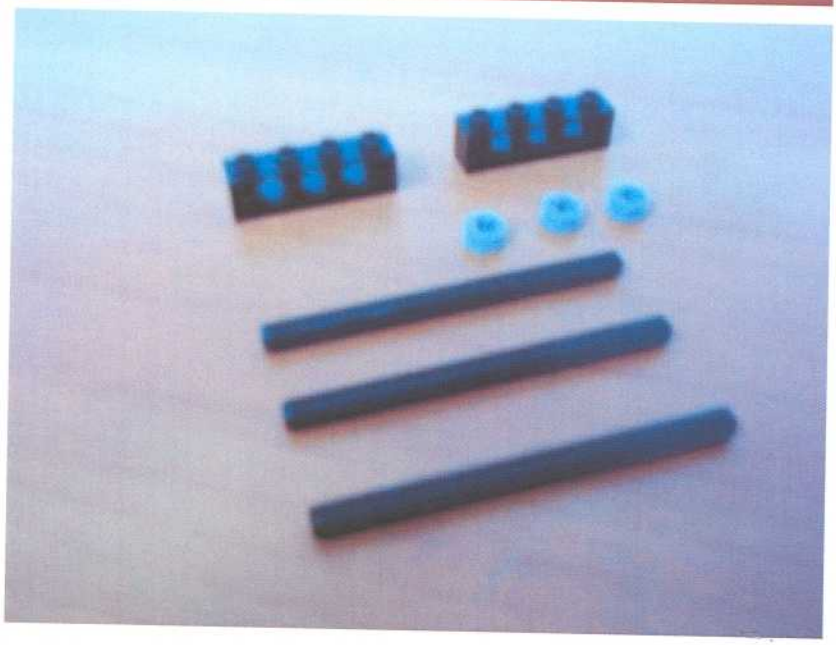
2



3

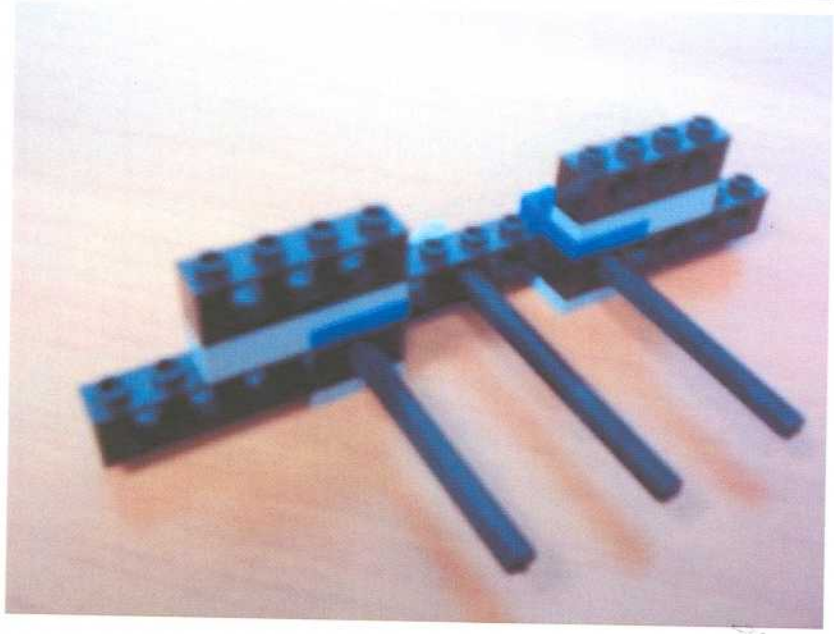


4



5

6



7

8

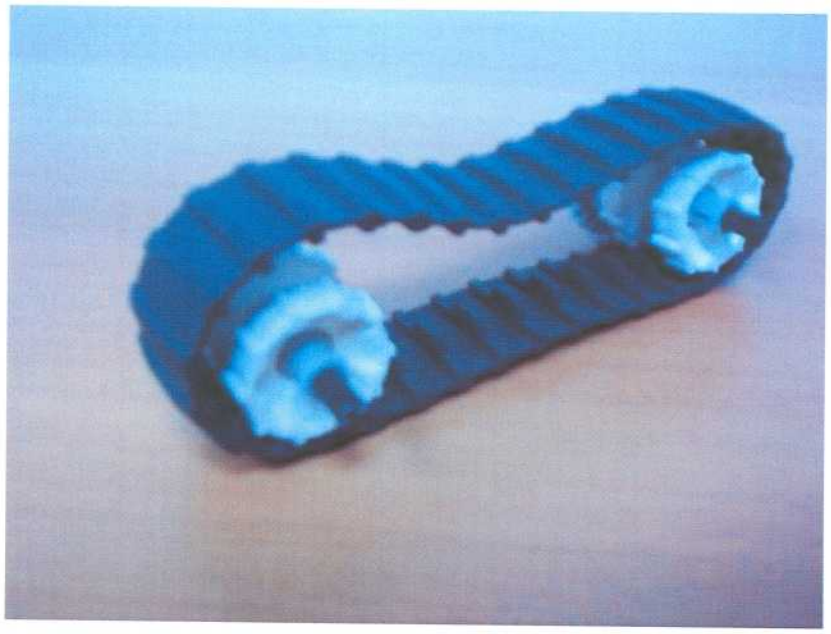
9

10

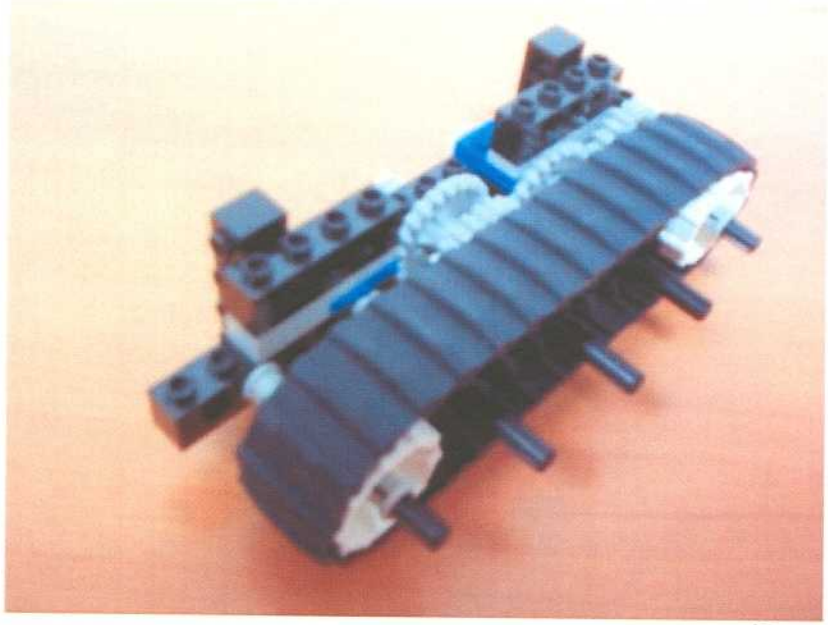
11

12

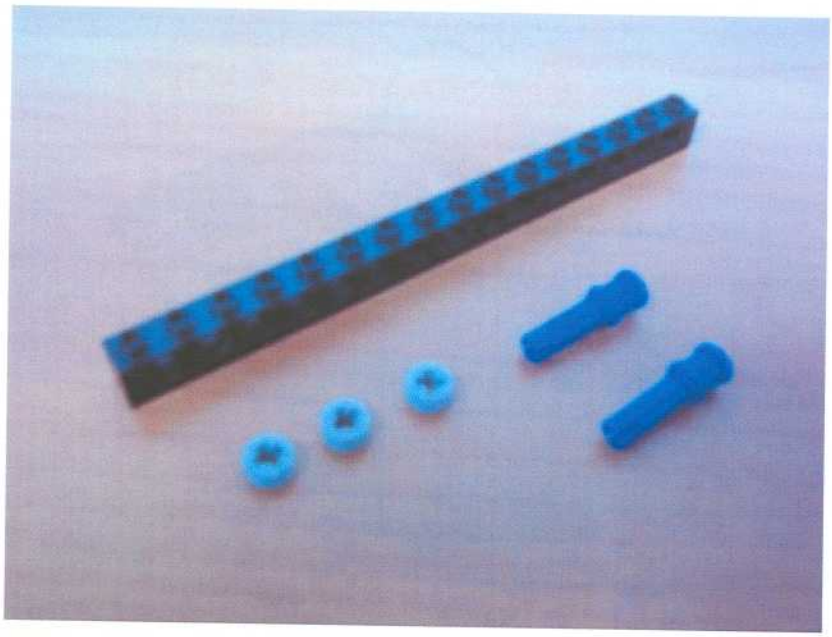
13



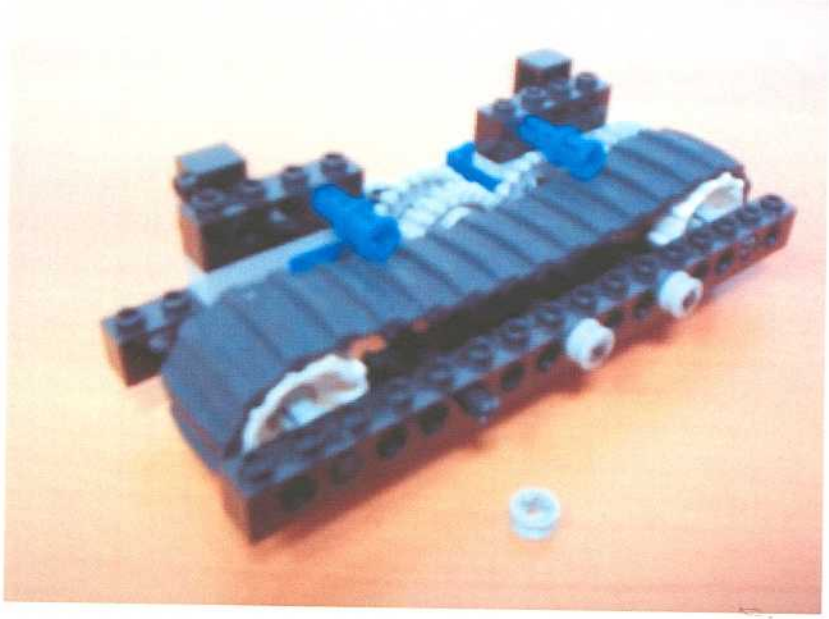
14



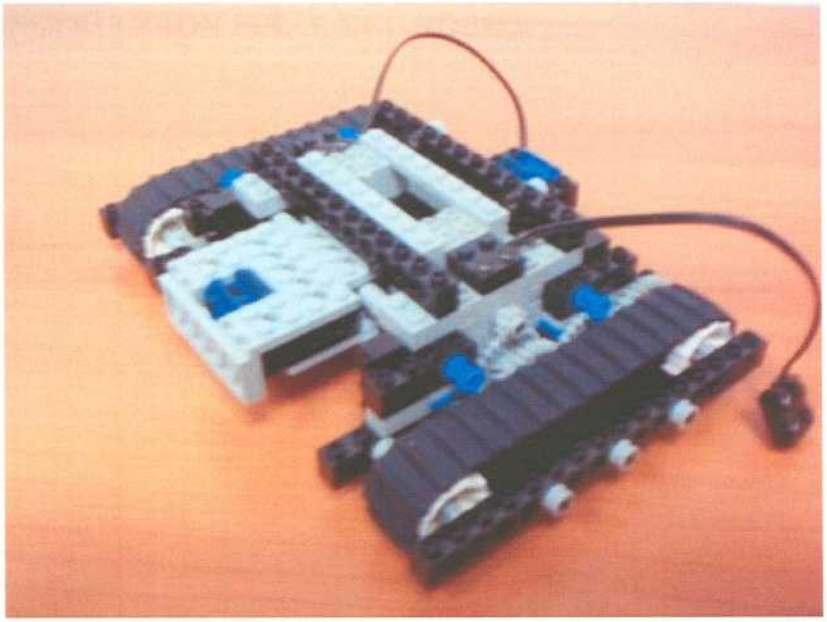
15



16

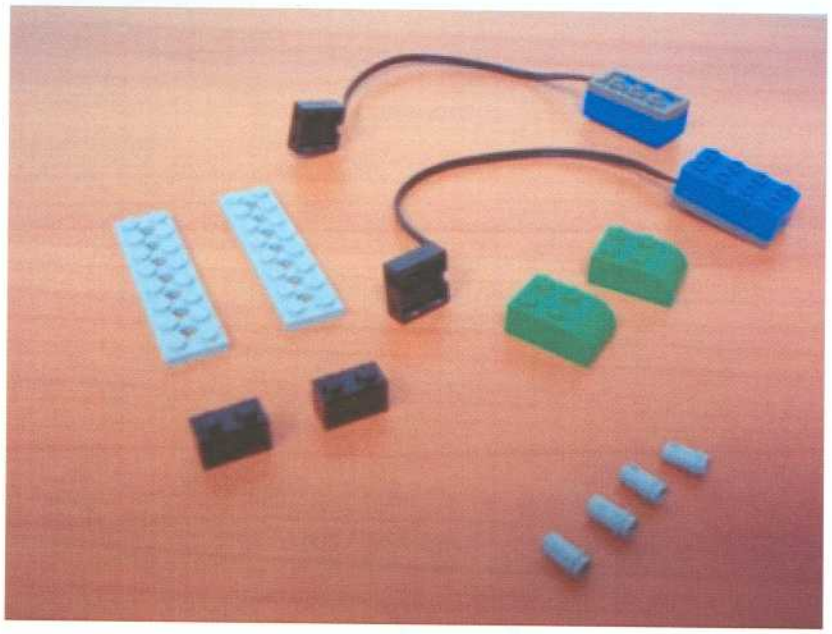


17

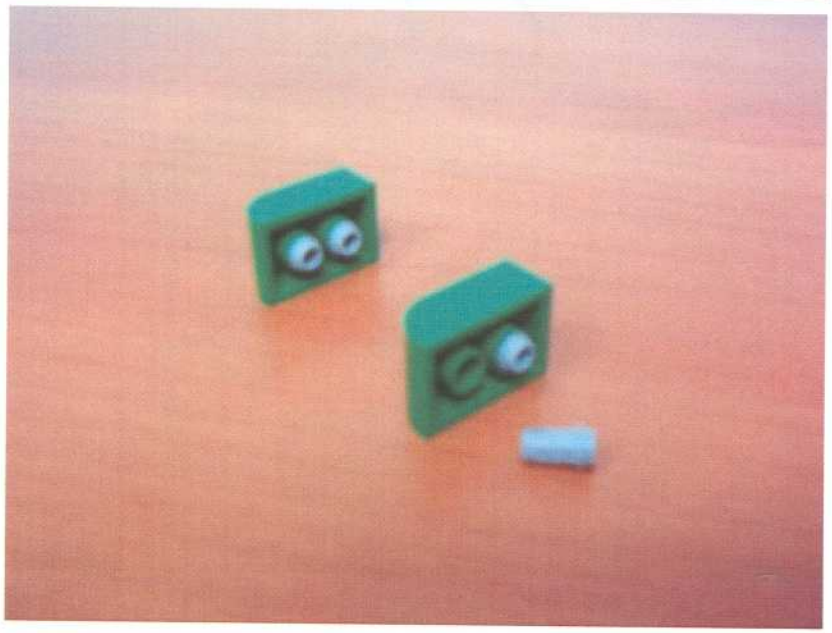


CONSTRUCCIÓN FINAL DEL ROBOT

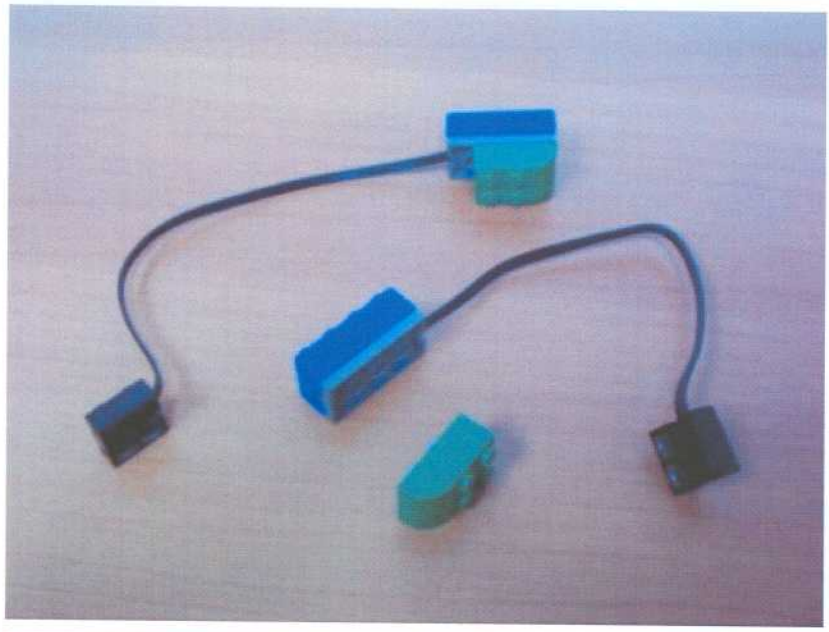
1



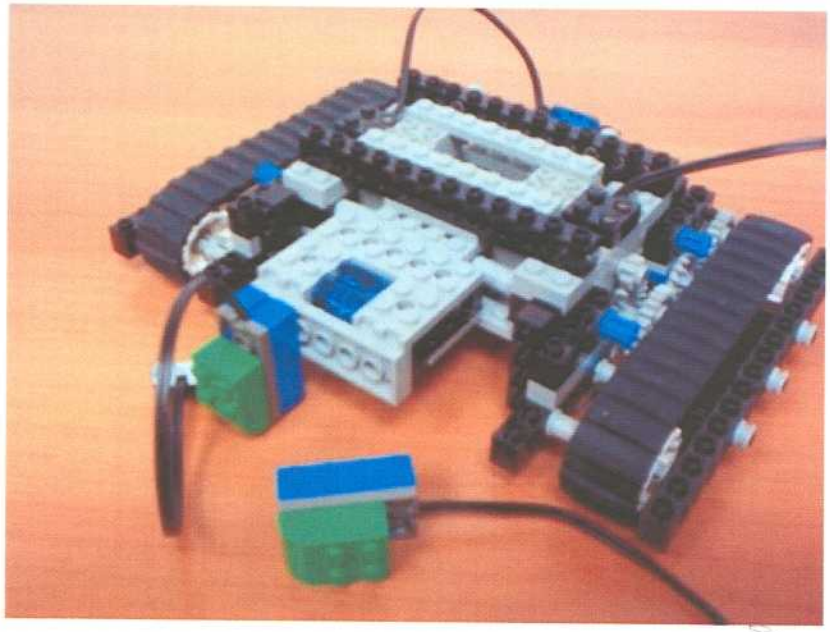
2



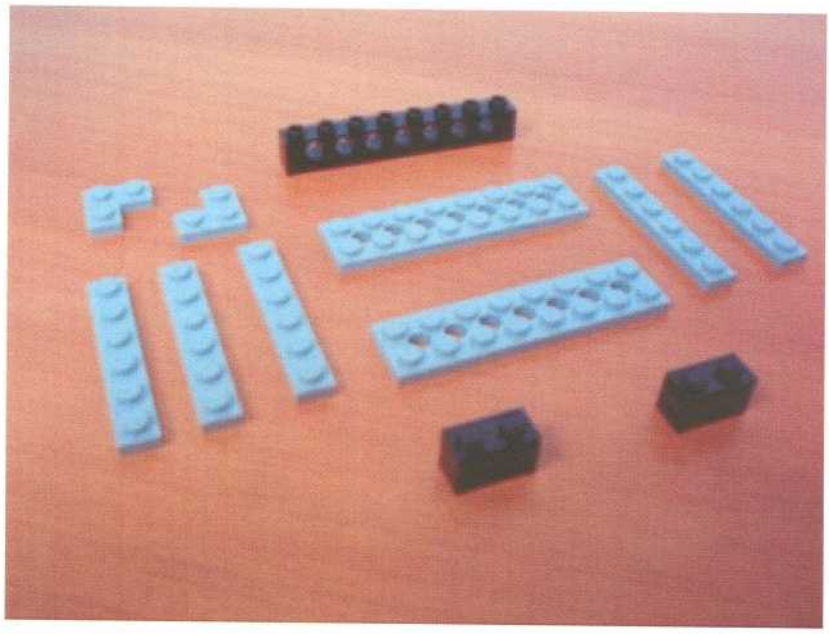
3



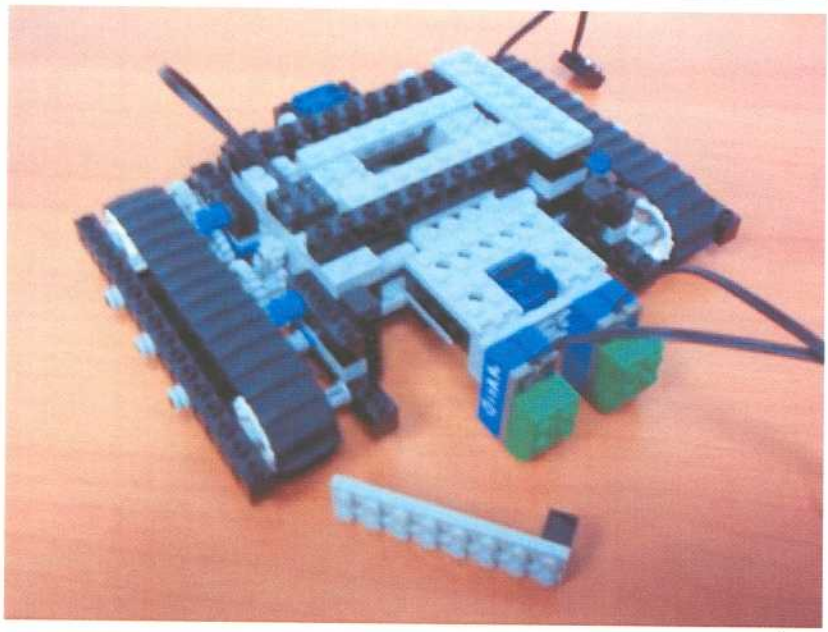
4



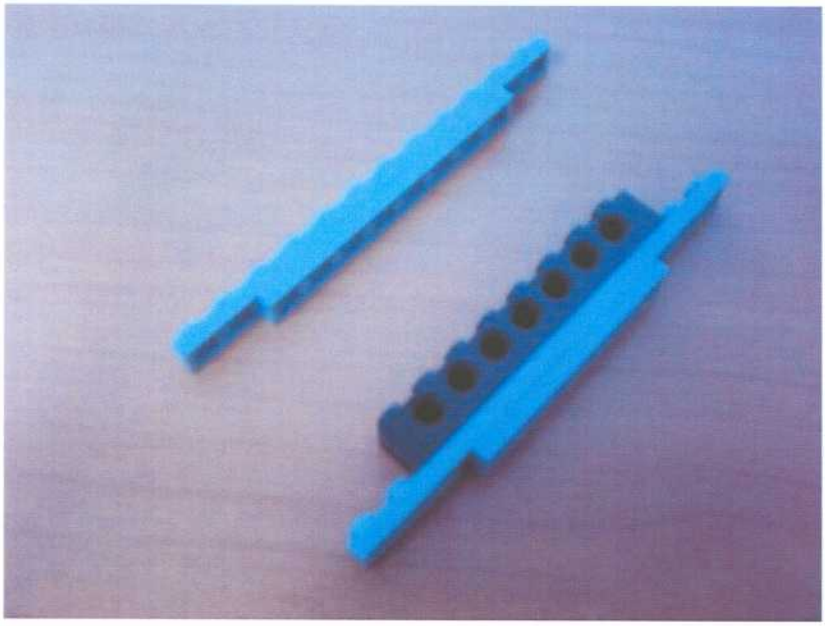
5



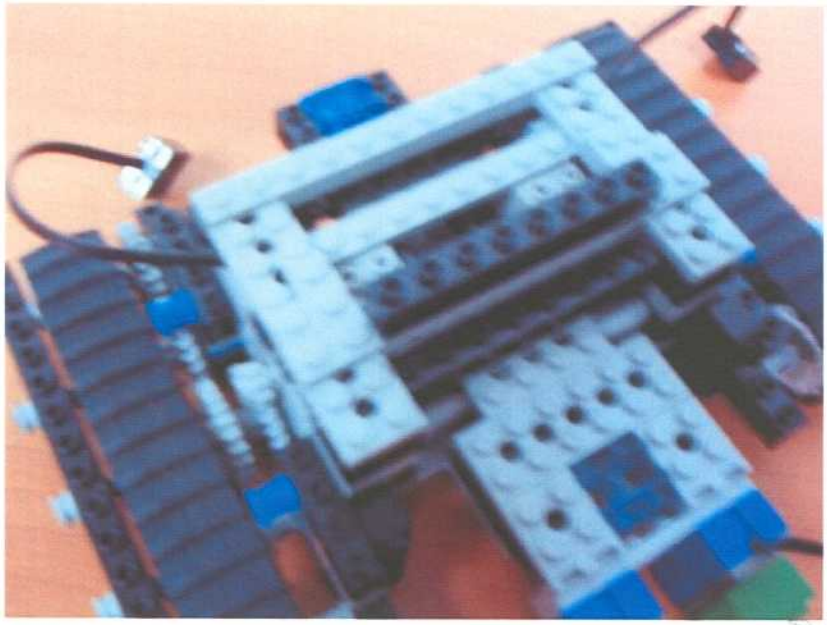
6



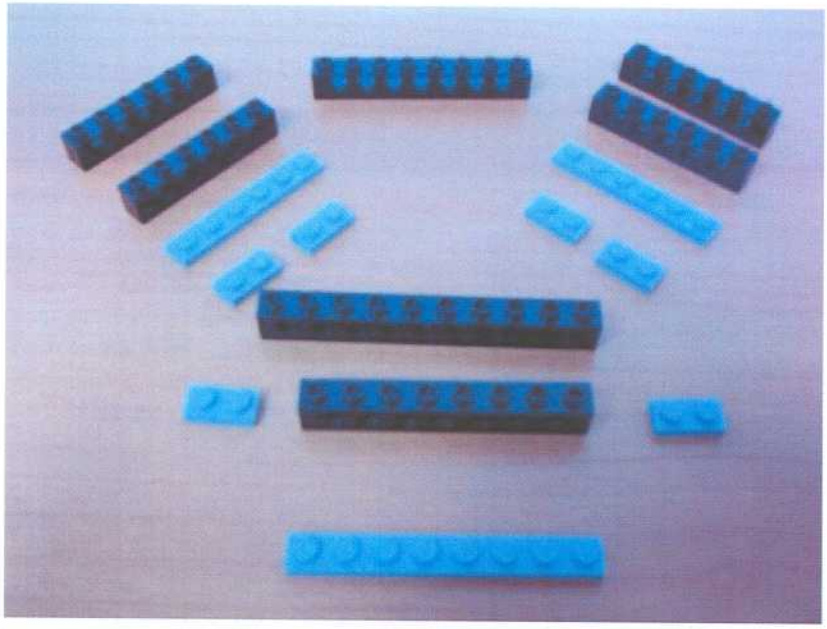
7



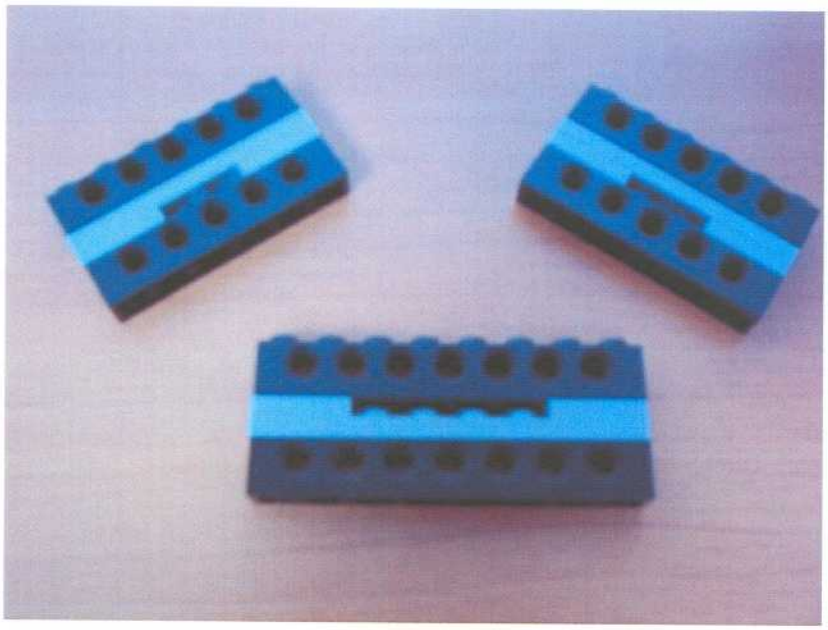
8



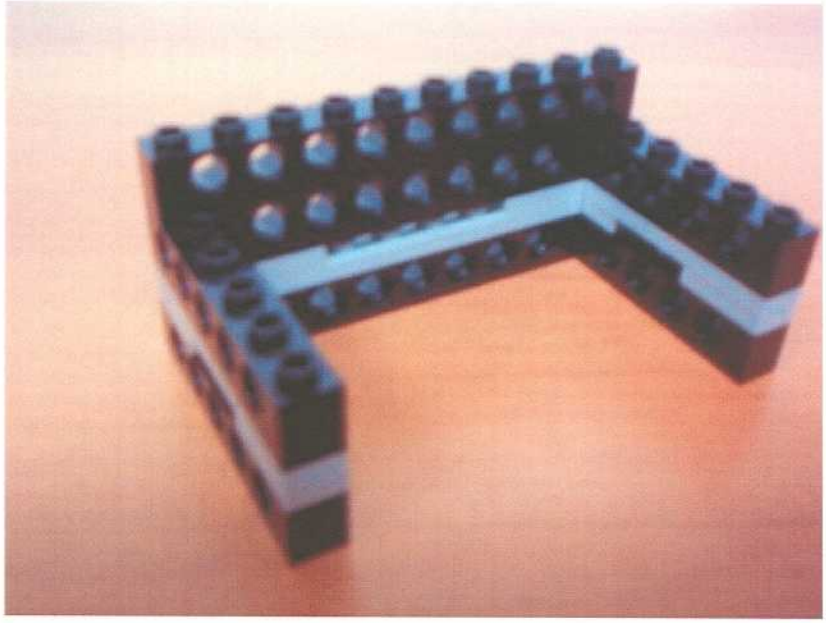
9



10

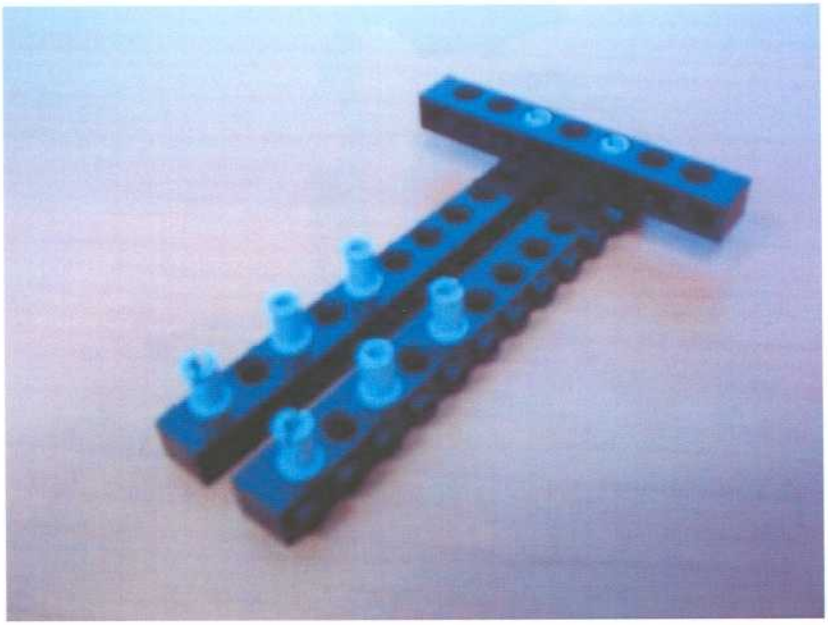


11

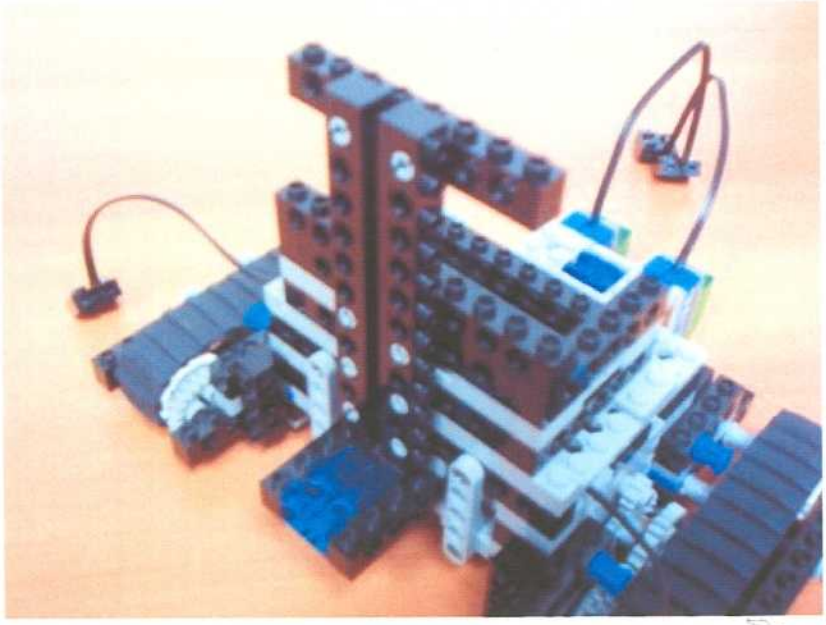


12

13



14



15

APÉNDICE B: MLCAD

MLCAD es una herramienta gráfica de diseño, desarrollada por Michael Lachmann en 1999; concebida especialmente para crear y mostrar paso a paso como se construye un modelo hecho con el kit LEGO MINDSTORMS.

En el entorno gráfico de MLCAD la construcción del robot puede ser visualizada desde cuatro puntos distintos, dando una perspectiva completa. Las piezas están hechas previamente, y básicamente lo que hay que hacer es seleccionarlas y colocarlas en la posición de deseada. Ofrece las herramientas básicas de cualquier sistema CAD, por ejemplo: mover, girar, seleccionar vistas, etc.

MLCAD tiene como ventaja que genera archivos muy pequeños, pues se basan en referencias a las piezas que se encuentran en la librería LEGO del software LDraw. Esta librería fue creada por James Jessiman con la finalidad de crear modelos de LEGO en 3D. Es muy útil para generar manuales de cómo crear determinados tipos de robots.

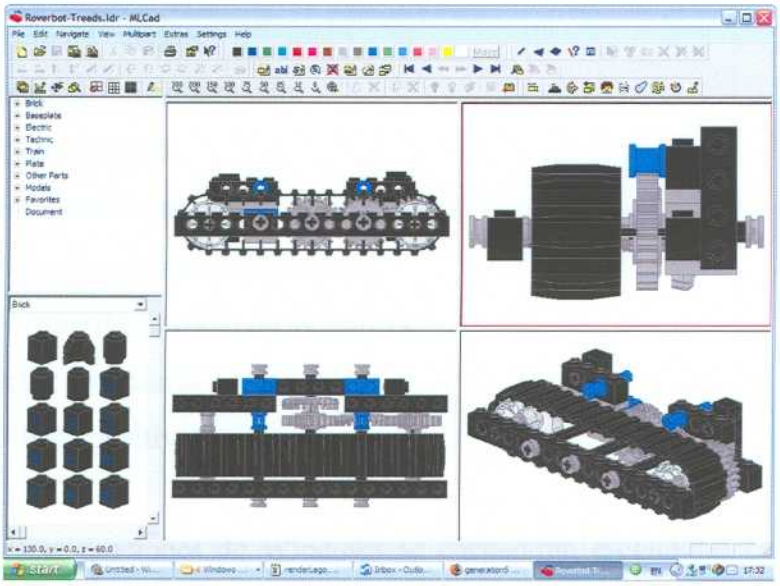


Figura 1. Ventana Principal del MLCAD.

Fuente: Elaboración propia.

APÉNDICE C: API DE WINDOWS

API Win32

La API ("Aplication Programming Interface" o "Interfaz de Programación de Aplicaciones") de Windows son un conjunto de funciones propias del sistema operativo que pueden ser usadas en cualquier programa.

También la Interfaz de Programación de Aplicaciones puede ser definida como un conjunto de funciones residentes en bibliotecas (generalmente dinámicas) que permiten que una aplicación pueda ser ejecutada bajo el sistema operativo Windows. Las versiones más recientes de Windows utilizan la API de 32 bits llamada Win32.

Las funciones API se dividen en varias categorías:

- Manejo de memoria.
- Almacenamiento.
- Información del sistema.
- Manejo de energía.
- Monitoreo del desempeño.
- Procesos e hilos.
- Comunicación entre procesos.
- Depuración y manejo de errores.
- GDI (interfaz gráfica) de Windows.
- Interfaz de usuario de Windows.
- E/S de dispositivos.

Todas estas funciones de Windows se encuentran escondidas dentro de DLL's ("Dynamic Link Library" o "Librería de Enlace Dinámico"), las cuales se encuentran agrupadas según la función que realizan:

- Kernel32.dll: operaciones de archivos y gestión de memoria.
- Gdi32.dll: operaciones gráficas.

- User32.dll: interacción con el usuario.

Adicionalmente, existen otros DLL's, aunque menos importantes como:

- Winmm.dll: controla la parte multimedia.
- Comdlg.dll: controles comunes para todas las aplicaciones.

Llamadas a las funciones del API de Windows

La característica principal del API de Windows son las funciones y los mensajes internos/externos que maneja. Dichas funciones son consideradas como el corazón de las aplicaciones Windows. Hay más de 600 funciones de Windows que pueden ser llamadas por cualquier lenguaje de programación, como Visual Basic, C, C++, etc.

A través de estos mecanismos es posible realizar una gran cantidad de aplicaciones. La utilización de estas librerías que contienen las funciones del API pueden resolver una gran cantidad de problemas en la programación. Sin embargo, no puede ser ignorado el enorme poder destructivo que estas pueden ocasionar.

La depuración de un programa es completamente diferente si éste tiene incorporado llamadas a la API. En ocasiones, el resultado de ejecutar este tipo de programas suele ser incomprensible; por tal motivo, siempre es útil contar con una guía y de esta forma evitar posibles problemas.

Las principales ventajas que se pueden obtener con la utilización de la API son: la gran funcionalidad que podemos darle a las aplicaciones y la gran velocidad de ejecución, ya que generalmente es mucho más rápido realizar una función a través de la API que por medio del lenguaje de programación.

A diferencia de las funciones, los mensajes son utilizados por Windows para permitir que dos aplicaciones puedan comunicarse entre sí y con el propio sistema operativo. Se dice que las aplicaciones de Windows son conducidas por mensajes o sucesos.

Librerías Dinámicas

Casi todas las API's de Windows se unen e interactúan entre sí formando librerías de enlace dinámico.

Una librería dinámica, abreviada DLL (Dynamic Link Library), permite que una aplicación pueda compartir código y recursos. Una DLL es un archivo ejecutable que contiene funciones las cuales pueden ser llamadas por otras aplicaciones.

Una DLL no se incluye dentro del código, sino que en el momento en que se realiza la llamada a la función, ésta (DLL) se carga en memoria, se utiliza y posteriormente se descarga.

La ventaja que posee utilizar DLL's es que no es necesario tener una gran cantidad de código cargado en memoria a la hora de ejecutar una aplicación. Por otro lado la desventaja se presenta cuando un ejecutable decida llevarse a otra instalación, será necesario llevarse también las DLL's.

Otras ventajas de utilizar DLL's es que al estar separadas del programa se pueden actualizar sin tener que modificar los programas que las utilizan; gran ahorro de memoria y disco, pues al ser Windows quien administra su utilización no existe duplicidad de código cuando varias aplicaciones acceden a ellas.

La mayoría de estas librerías se encuentran ubicadas en el directorio de System de Windows. En cada versión suelen aparecer nuevas versiones de DLL's.

Para acceder a cualquier función de la API lo primero que se tiene que hacer es declararlas.

Declaración de una DLL

Para declarar una función DLL, esta tiene que ser escrita en el módulo global de aplicación, o en la sección de declaraciones del formulario.

Ejemplo:

```
Declare Function CreateThread Lib "kernel32" (ByVal
lpSecurityAttributes As Long, ByVal dwStackSize As Long, ByVal
lpStartAddress As Long, ByVal lpParameter As Long, ByVal
dwCreationFlags As Long, _lpThreadId As Long) As Long
```

La palabra reservada "Lib" le indica al lenguaje de programación la librería donde puede hallar la función de la API declarada. Todas las DLL's están en "Kernel", "GDI", "User", o en una de las DLL's correspondientes a un driver de dispositivo.

La palabra reservada "Alias" se utiliza en ocasiones e indica que la función tiene otro nombre en la librería dinámica. Suele ser útil cuando el nombre de la función coincide con alguna palabra clave, variable o constante, o contiene caracteres no reconocidos por el lenguaje.

Si una función no retorna valor se declara como un procedimiento o "Sub", aunque esta opción no se utiliza casi nunca ya que la mayoría de las API's retornan al menos una verificación de su realización, por lo que, se declara como función o "Function".

Algunas funciones de la API

Las funciones del API se agrupan mayormente en:

- Funciones de direccionamiento de memoria a 32 bits: recupera el API para aplicaciones que emplean 32 bits.
- Funciones de ejecución de aplicaciones: carga y ejecuta aplicaciones.
- Funciones bitmap: crea y maneja los bitmaps.
- Funciones callback o rellamada: recupera información sobre algo que se está ejecutando, con la posibilidad de hacer algún cambio.
- Funciones de cuadro de diálogo común: actúa sobre los diálogos.
- Funciones de comunicación: gestión de los dispositivos de comunicación.
- Funciones de cursor: gestión de todo lo relacionado con el cursor.
- Funciones DDE (Data Dynamic Exchange): gestiona el intercambio de datos dinámicos.
- Funciones de error: gestión de los errores de los dispositivos.
- Funciones de movimiento y visualización: determina si una ventana es minimizada.
- Funciones para las propiedades: informa y modifica el estado de las propiedades.

- Funciones pen: correspondientes al lápiz.
- Funciones de paleta de colores.
- Funciones de dibujo y pintura.
- Grupo de funciones OLE: manejo de objetos.
- Funciones Toolhelp: devuelve información sobre los objetos que hay en memoria.
- Funciones Drag & Drop.
- Funciones de controladores instalables.
- Funciones para la impresión: devuelve tamaños de letra, página, etc. y activa o desactiva algunas funciones, configura colores, etc. a la hora de imprimir.
- Funciones Shell: controlan el entorno.
- Funciones de versión: controla la versión de los ficheros.
- Funciones GDI: controla todo lo que se refiere a gráficos, tipos de letras, etc.
- Funciones Kernel: lleva a cabo una E/S a ficheros, control de errores, etc.
- Funciones de usuario: controla las acciones que realiza un usuario, recupera información de mensajes de hardware, abre dispositivos instalables, etc.

Ejemplo de una llamada al API de Windows

En el siguiente ejemplo, se muestra la declaración de la función "*CreateThread*".

```
Declare Function CreateThread Lib "kernel32" (ByVal
lpSecurityAttributes As Long, ByVal dwStackSize As Long, ByVal
lpStartAddress As Long, ByVal lpParameter As Long, ByVal
dwCreationFlags As Long, _lpThreadId As Long) As Long
```

La función "*CreateThread*" posee seis parámetros, los cuales se explican a continuación:

- lpSecurityAttributes: define los atributos de seguridad a usar; típicamente toma valor cero para usar los valores por defecto.
- dwStackSize: representa el tamaño de la pila; cada hilo tiene su propia pila.

- lpStartAddress: es la dirección de memoria donde el hilo comienza.
- lpParameter: es una variable tipo 'long' de 32 bits que se pasa como parámetro a una función que comienza el nuevo hilo.
- dwCreationFlags: es una variable de 32 bits que otorga el control de comienzo de un hilo (independientemente si está activado, suspendido, etc.).
- lpThreadId: es una variable que es cargada al instante de crear un nuevo hilo, simboliza la identificación del mismo.

Todo lenguaje de programación es finito, es decir, está limitado. En ocasiones es posible ampliar su funcionalidad utilizando librerías u otros complementos. Las librerías DLL's existen desde la versión 2.5 de Windows y son escritas principalmente en lenguaje C.

APÉNDICE D: SISTEMAS OPERATIVOS UTILIZADOS PARA PROGRAMAR EL RCX

Un aspecto muy importante es determinar que tipo de herramienta utilizar para programar el RCX. Es de hacer notar que el RCX es el corazón del kit de LEGO MINDSTORMS.

Hay distintas herramientas para programar el RCX, sin embargo no existe ningún Firmware (Sistema Operativo) ideal. La elección depende las características y condiciones del problema que se desea resolver. Entre ellas tenemos: RIS (Robotic Invention System), NQC (Not Quite C), BrickOS y LejOS.

A continuación se describen brevemente cada una de ellas:

- **RIS:** es el firmware estándar que provee LEGO MINDSTORMS en su kit. Es un lenguaje de programación gráfico. Está enfocado para todo tipo de personas, específicamente para individuos mayores de 12 años con poca o ninguna experiencia en cuanto a programación.
- **NQC:** es un firmware que permite programar la microcomputadora con una sencilla sintaxis, en un lenguaje muy similar a C. Está dirigido a aquellas personas con limitada experiencia en programación que desean desarrollar programas más complejos que con RIS. Fue desarrollado por Dave Baum, tomando como base el viejo código fuente de C. Emplea estructuras de control, variables, funciones, etc.
- **BrickOS:** es un firmware que permite la programación en lenguaje C. Ofrece muchas ventajas ya que proporciona flexibilidad, potencia y efectividad. Su aprendizaje se torna lento de no tener conocimientos de C. Fue creado por Markus L. Noga, quien desarrollo la mayor parte, aunque otras personas han realizado incorporaciones. Antiguamente este sistema operativo era llamado LegOS.
- **LejOS:** es un firmware que permite la programación en lenguaje Java. Proporciona numerables ventajas propias de un lenguaje de alto nivel, por ejemplo: concurrencia, clases, herencia, estructuras de datos dinámicas, funciones matemáticas, así como otras características. Fue desarrollado por José Solórzano y

está basado en la TinyVM. Actualmente está mantenido por Paul Andrews Jürgen Stuber.

La tabla que se muestra a continuación ilustra algunas de las características más importantes que permiten comparar cada uno de los Firmware.

	RIS	NQC	BrickOS	LejOS
Lenguaje de Programación	Gráfico.	Similar a C.	C.	Java.
Sistemas Operativos bajo los que trabaja	Windows.	Windows, Mac OS X.	Windows, Linux.	Windows, Linux, Solaris, Mac OS X.
Nivel de Programación	Bajo (para todo tipo de usuario).	Bajo (para todo tipo de usuario), se facilita su uso si se posee conocimientos de C.	Intermedio (de rápido aprendizaje si se tiene conocimientos de C).	Intermedio (de rápido aprendizaje si se tiene conocimientos de Java).
Nivel de Instalación	Bajo.	Bajo.	Alto.	Bajo.
Estructuras de Control	No aplica.	Sí.	Sí.	Sí.
Control de Concurrencia	No aplica.	No.	Sí.	Sí.

Tabla 1. Cuadro comparativo de los diferentes Sistemas Operativos en los cuales puede programarse el RCX.

Fuente: Elaboración propia.

APÉNDICE E: SIMULADOR DE COLONIAS DE HORMIGAS

La aplicación tiene como objetivo fundamental simular el mecanismo de búsqueda y recolección de alimento de las hormigas utilizando "Ant Learning". Dicho algoritmo (elaboración propia) tiene como finalidad lograr conseguir el alimento en un ambiente desconocido a fin de lograr la supervivencia de la colonia.

Utilizando distintos tipos de ambientes se logra simular el funcionamiento de las hormigas, adicionalmente se incorporan distintas condiciones capaces de alterar el correcto funcionamiento de dichos entes (hormigas), como: tasa de evaporación de la feromona, cantidad de hormigas al momento de la búsqueda del alimento, nivel de aprendizaje, etc.

La herramienta de programación utilizada para desarrollar la aplicación fue Visual Basic 6.0. En la figura 2 se puede apreciar la pantalla/ventana principal del simulador.

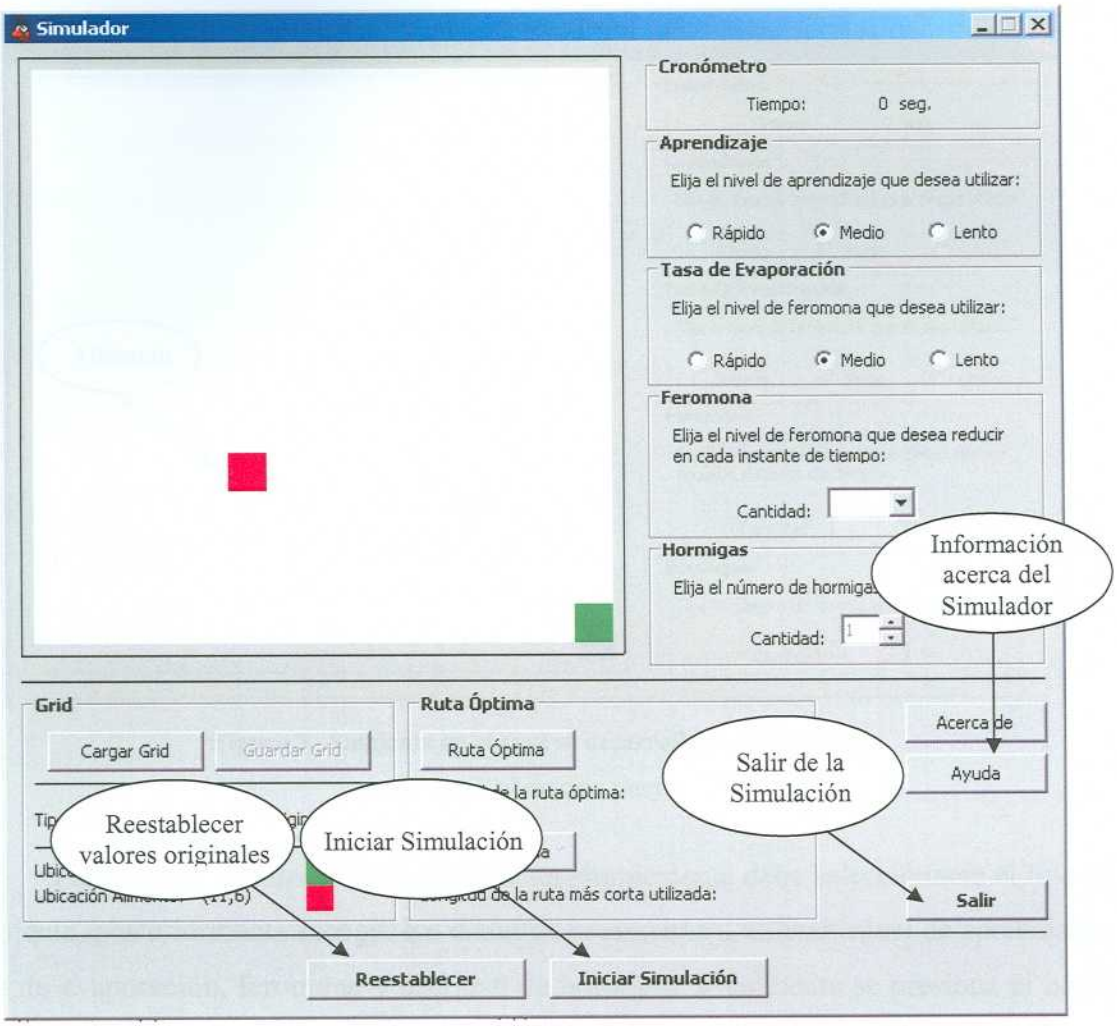


Figura 2. Pantalla Principal del Simulador.

Fuente: Elaboración propia.

Inicialmente la aplicación se carga con un ambiente predefinido. Allí se pueden visualizar originalmente dos colores. El primero, el verde, el cual indica la ubicación del nido dentro del ambiente; el segundo, el rojo, el cual refleja la ubicación del alimento. En la figura 3 muestra un tipo de ambiente.

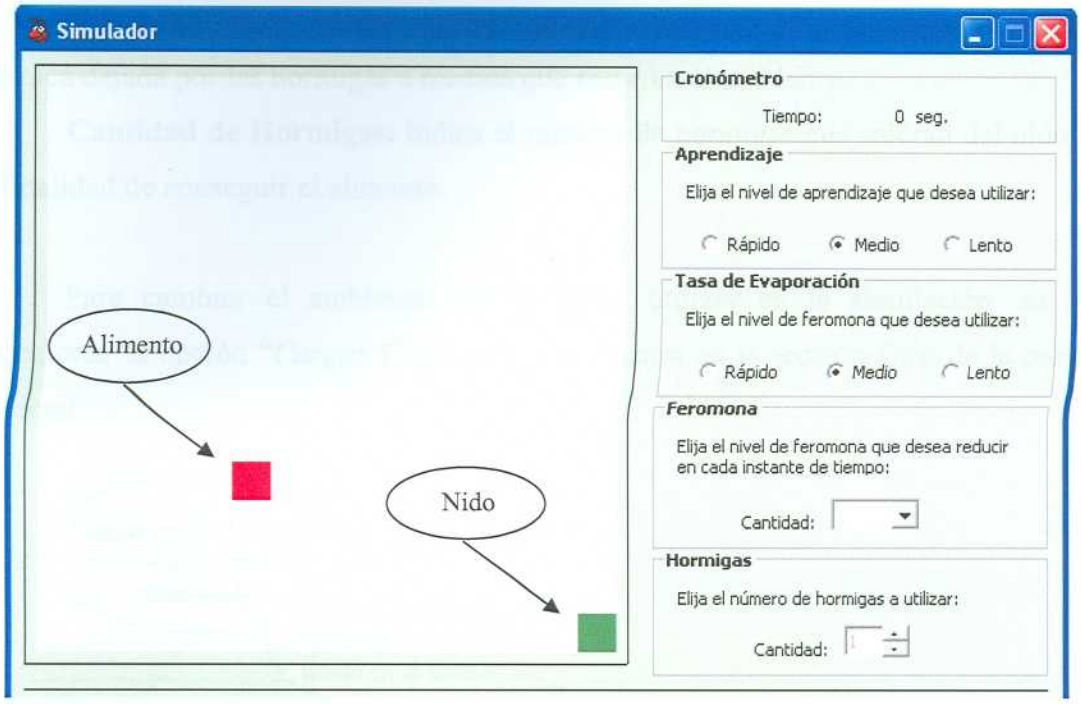


Figura 3. Ambiente en el cual se desarrolla la simulación.

Fuente: Elaboración propia.

Para iniciar la corrida de la simulación simplemente debe seleccionarse el tipo de ambiente, posteriormente escoger los distintos parámetros a utilizar: nivel de aprendizaje, tasa de evaporación, feromona y cantidad de hormigas. Finalmente se presiona el botón "*Iniciar Simulación*".

Para reinicializar los valores originales, se debe presionar el botón "*Reestablecer*". En caso de que se desee tener más conocimiento acerca del funcionamiento de la aplicación, basta con ir a la sección de "*Ayuda*", allí encontrará toda la información. Para salir de la aplicación debe presionarse el botón "*Salir*".

Las opciones de configuración de la aplicación se explican detalladamente a continuación:

Aprendizaje: refleja la velocidad con la cual las hormigas se desplazan alrededor del ambiente en búsqueda del alimento; indica el intervalo de tiempo que se tarda cada hormiga en procesar la información proveniente del ambiente y poder procesarlo.

Tasa de Evaporación: es la velocidad a la que se reduce la sustancia química dejada por las hormigas a medida que recorren el ambiente.

Feromona: es la cantidad que se reduce en un instante de tiempo de la sustancia química dejada por las hormigas a medida que recorren el ambiente.

Cantidad de Hormigas: indica el número de hormigas que saldrán del nido con la finalidad de conseguir el alimento.

Para cambiar el ambiente que se desea utilizar en la simulación, se debe seleccionar la opción "*Cargar Grid*" que se encuentra en la sección Grid de la pantalla principal.

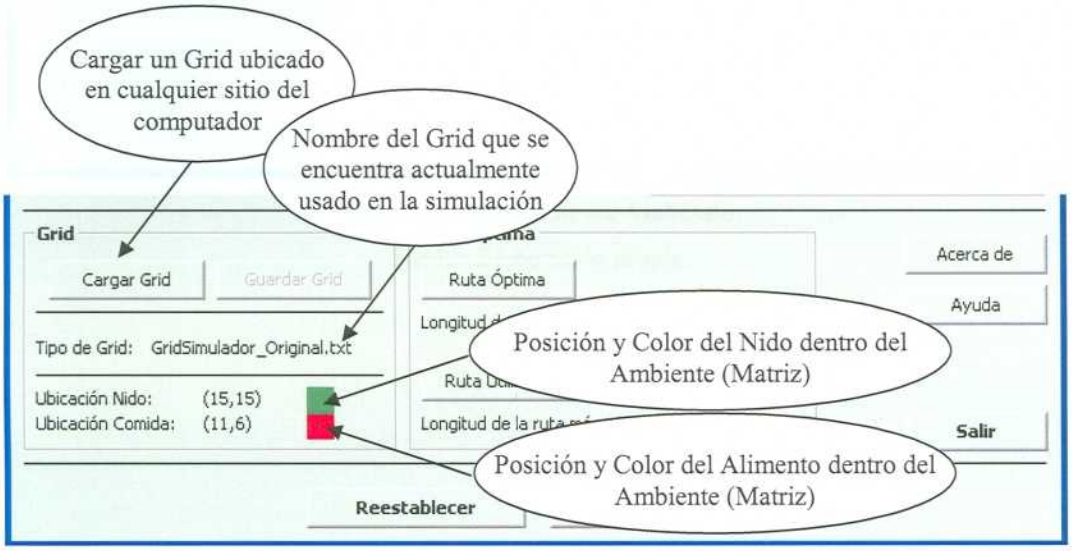


Figura 4. Información del Grid.

Fuente: Elaboración propia.

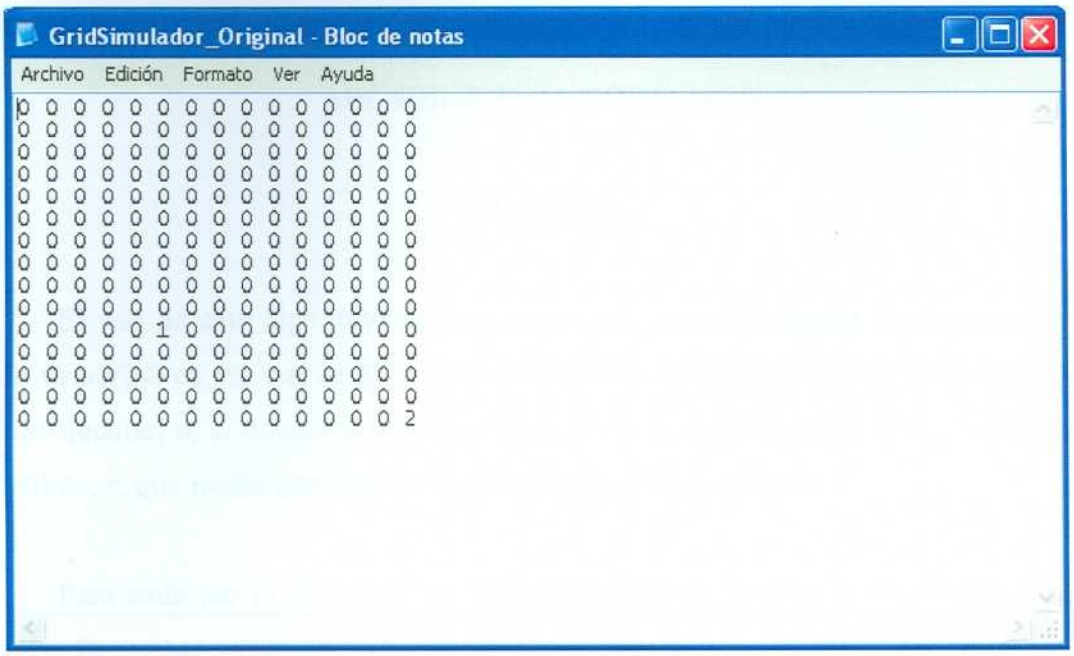
Los ambientes están guardados en formato .txt. Cada uno de ellos está compuesto por tres tipos de caracteres, ellos son: 0, 1, 2.

0: Camino.

1: Alimento.

2: Nido.

La figura 5 visualiza un tipo de archivo el cual representa un ambiente dentro de la aplicación.



GridSimulador_Original - Bloc de notas

Archivo Edición Formato Ver Ayuda

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 2
```

Figura 5. Ejemplo de un Ambiente.

Fuente: Elaboración propia.

APÉNDICE F: ALGORITMOS

Q-Learning

El uso de este algoritmo se basa en la generación de tuplas de experiencia del estilo <*s*,*a*,*s*',*r*> en las que se definen el estado del que parte el agente, *s*, la acción que decide ejecutar, *a*, el estado al que llega después de ejecutar esa acción, *s*', y el refuerzo inmediato, *r*, que recibe por llegar a ese estado desde el que estaba.

Para cada par (s ∈ S, a ∈ A) inicializar la tabla $Q(s,a)$ a 0. Observar el estado actual *s*. Hacer para siempre:

- Seleccionar una acción *a* y ejecutarla
- Recibir el refuerzo inmediato *r*
- Observar el nuevo estado *s*'
- Actualizar la entrada de la tabla, $Q(s,a)$ con la ecuación:

$$Q(s,a) \leftarrow r + \gamma \max_{a'} Q(s',a')$$

- Asignar a *s* el estado *s*'

Value Iteration

Siendo:
s: el estado del que parte el agente
a: la acción que decide ejecutar
s': el estado al que llega después de ejecutar la acción a

Se puede definir el algoritmo de Value Iteration de la siguiente forma:

Inicializa $V(s) = 0$ para toda $s \in S$
Repite
$\Delta \leftarrow 0$
Para cada $s \in S$
$v \leftarrow V(s)$
$V(s) \leftarrow max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^*(s')$
$\Delta \leftarrow max(\Delta, |v - V(s)|)$
Hasta que $\Delta < \theta$ (número positivo pequeño)
Regresa una política determinística tal que:
$\pi(s) = argmax_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^*(s')]$

Ant-Learning

En este caso, el algoritmo de divide en dos partes. Una, referente a la lógica empleada por cada uno de los robots para la navegación en el ambiente y la segunda, al funcionamiento del Servidor que actualiza la matriz de sectores para generar el aprendizaje de los entes.

Algoritmo empleado por cada uno de los robots:

```
Mientras (¡seEncuentreAlimento())
{
        irAdelanteHastaPuntoDeCruce();

        direccionATomar = generarDireccionAleatoria();
        posición(X,Y) ← nueva_Posicion_Geografica();

        Si (direccionATomar == IZQUIERDA)
        {
                nueva_orientacion_geografica(IZQUIERDA);
                girar_Izquierda();
        }
```

```
        Si (direccionATomar == DERECHA)
        {
                nueva_orientacion_geografica(DERECHA);
                girar_Derecha();
        }

        esperarHastaObtenerToken();

/* Se actualiza la matriz de sectores con la posicón X,Y , marcándola como ocupada */

        transmitirPosicion(X,Y);
}
```

Algoritmo empleado por el Servidor:

```
dispararHilo();
darTokenAPrimerRobot();
Hacer para siempre
{
        esperarLlegadaDePaquete();

        Si (LlegaUnPaquete)
        {
                posición(X,Y) ← obtenerPosicionesDelPaquete();
                actualizarMatrizDeSectores(X,Y);
                darTokenASiguienteRobot();
        }
}
```

Hilo del Servidor:

```
Cada cantidad de tiempo X
{
        reducirFeromonaEnMatrizDeSectores();
}
```