# CAPITULO 4

# IMPLEMENTACIÓN

## 4.1 INTRODUCCIÓN

Podemos decir que a lo largo de muchos años los programadores han ido desarrollando *patrones* o guías para resolver ciertos problemas, y con el paso del tiempo y mientras más se usan éstas guías han ido evolucionando. El hecho de que una misma estructura se vaya usando con cierta frecuencia para tratar de resolver algún problema es porque en efecto es una buena solución para un cierto tipo de problema, porque tampoco se pueden creas patrones universales capaces de dar solución a una amplia variedad de problemas. En el ámbito de la programación este conjunto de soluciones se denomina como "Patrones de Programación" (*Programming patterns*) [Bagnall, 2002] y son una colección de estructuras de programación, modelos orientados a objetos que describen algunos objetos principales y las relaciones entre ellos. La programación de un robot es un tanto diferente a la programación de una aplicación para algún usuario, y es por eso que dentro del área de la Robótica se han desarrollado patrones de programación específicos para el área. Uno de los más populares es el desarrollado para robots pequeños generalmente usado para hacer una programación a nivel de insectos ("insect"-level programming) y se conoce con el nombre de "Control de Comportamiento" (*Behavior Control*). [Bagnall, 2002]

### 4.1.1 TEORÍA DEL CONTROL DE COMPORTAMIENTO

Esta teoría fue originalmente definida por Rodney Brooks en el Laboratorio de Inteligencia Artificial en el MIT. [Bagnall, 2002] Él tomo esta idea del mundo de los insectos,

cuando notó que estos eran capaces de desenvolverse en el mundo real con éxito a pesar de tener un cerebro demasiado pequeño. Obviamente si comparamos el tamaño del cerebro de un insecto con una computadora pues ésta última tiene una memoria y capacidad mucho mayor, de hecho se ha demostrado que los insectos no tienen la capacidad de recordar cosas del pasado o asociar conductas y estímulos como demostró Pavlov que lo hacen los perros. Los insectos puramente tienen reacciones simples y sencillas hacia el mundo, pero cuando se combinan varias de estas conductas los insectos son capaces de responder asombrosamente a los cambios del ambiente. Esta es la idea que Brooks tomó para empezar a desarrollar esta nueva teoría y que describe en una de sus diversas publicaciones: "Al trabajar bajo este esquema no se construyen módulos funcionales sino que se definen comportamientos, los cuáles van a ser los bloques para construir el sistema y la funcionalidad va a ser el resultado." [Brooks, 1991*a*]

En un principio todas estas ideas fueron originalmente desarrolladas bajo el nombre de "Arquitectura Subsumption" pero en la actualidad se ha agrupado todo bajo el nombre de Control de Comportamiento ("*Behavior Control*"). [Bagnall, 2002]

Este tipo de arquitectura hace énfasis en la importancia de sincronizar los sensores con las acciones y lo que se busca es hacer una descomposición del problema en unidades significativas dentro del contexto para el cual se desarrolla el sistema. Los robots programados bajo este esquema funcionan mejor cuando el mundo real no puede ser modelado o representado con gran precisión, esto se debe a que esta arquitectura fue desarrollada como una respuesta a la dificultad de quitar los factores no deseados (*noise*) en el ambiente. [Arkin, 1998] Otro aspecto que se debe de tener en mente cuando se desarrolla un sistema basado en comportamientos es que las conductas que se implementen deben de entran en funcionamiento al menos una vez. Aunque es muy probable que las conductas

estén cambiando continuamente de una a otra, el robot aún debe de mostrar coherencia entre sus acciones y el objetivo, se espera que no esté cambiando rápidamente entre un comportamiento y otro que sea inconsistente con el anterior, es decir no se esperan acciones incoherentes debido al cambio entre los comportamientos activos. De igual manera no puede haber dos comportamientos activos al mismo tiempo, ya que mutuamente se estarían interfiriendo o limitando y ninguno de las dos se realizaría con éxito. [Brooks, 1991*a*] Obviamente se espera que el comportamiento que se ejecute sea coherente con la situación actual y las condiciones del ambiente, por ejemplo si el robot debe recargar su batería cuando éstas se bajan, se esperaría que verdaderamente lo haga cuando el nivel de las baterías ha bajado y no cuando aún están llenas.

Existen esencialmente dos estructuras necesarias para construir un comportamiento: los sensores (input) y los *actuadores* (output). Esto es cierto y necesario no solo para los robot, sino para todos los organismos, incluidos nosotros mismos. Nuestros sensores son los ojos, la vista, el oído, entre otros; y nuestros actuadores básicos son las piernas. Cuando vemos que algo nos impide pasar (input) pues simplemente cambiamos el rumbo hacia donde caminamos (output). En los insectos y en los robots es igual. Este conjunto de condiciones es lo que se llama conductas ó comportamientos (*behaviors*).

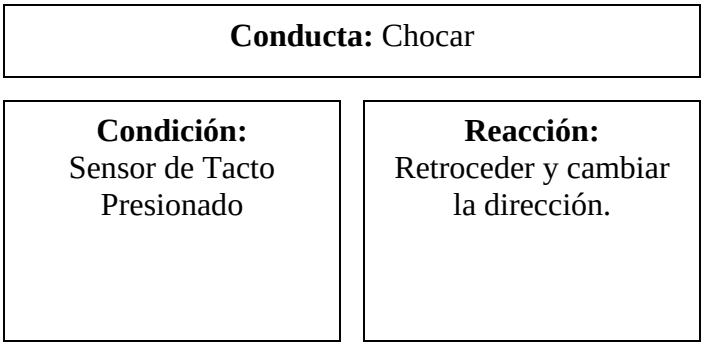


**Figura 4.1** Diagrama con el ejemplo de una conducta de choque. [Bagnall, 2002]

### 4.1.2 TEORÍA DEL CONTROL DE COMPORTAMIENTO y leJOS

Dado que esta teoría ha resultado ser muy útil para entender y programar robots reactivos, el API de leJOS ha tratado de hacer una implementación. Muchos programadores, cuando comienzan a diseñar su robot, piensan en su sistema como algo lineal, o un conjunto de condiciones del estilo *if-then* que se encuentran lejos de ser una programación estructurada y mucho menos orientada a objetos, que es el eje alrededor del cual gira leJOS dado que está basado en Java. Este tipo de códigos son muy comunes porque no requieren de un trabajo previo, simplemente el desarrollador se sienta frente a la computadora y comienza a teclear las condiciones, reacciones y básicamente las acciones que espera del robot, pero es muy probable que para el final el código sea toda una maraña de líneas entrelazadas que no llevan a ningún lado y que difícilmente puede ser reutilizable o incluso modificable.

Como una posible solución al problema anterior en leJOS se ha implementado el Modelo de Control de Comportamiento ("*Behavior Control Model*"), un API sencillo de entender y de programar. Cabe señalar que el usar este modelo requiere un poco de trabajo previo como es planeación y estructuración de las conductas que deseamos integrar al robot, cuándo entran en funcionamiento y cuándo dejan de usarse, entre otras cosas. Una vez que tenemos definido todos nuestros comportamientos podemos comenzar a programar cada uno de ellos (una clase por comportamiento definido) teniendo como beneficio que cada uno de nuestros comportamientos se encuentra programado uno independiente del otro, así que podemos modificar solo una conducta sin tener que alterar todo el programa. No olvidemos que debemos entender por comportamiento a todos los pares formados por una reacción y una acción consecuente, como observamos con anterioridad en la figura 4.1. Este tipo de programación es particularmente útil para programar los llamados Robots

Autónomos, que son aquellos que trabajan independientemente y deciden en base a lo que hay en el ambiente cuál será su próxima acción.

Uno de los principales problemas que se presentan al trabajar bajo este esquema es el control de los comportamientos. De alguna manera se debe de decidir cual de los comportamientos debe estar activo para que de esta manera sea esa conducta quien tome el control de los actuadores por un cierto momento. [Maes, 1990] Habrá ocasiones en que más de un comportamiento buscará entrar en funcionamiento y para que el robot pueda decidir cuál de ellos es el que deberá estar activo los diferentes comportamientos deben de tener prioridades. Por ejemplo un animal presenta varios objetivos básicos como son comer, defenderse, explorar y reproducirse, entre otros. Algunos de estos comportamientos son más importantes que otros, pero todos son necesarios para que pueda sobrevivir en el medio. Por ejemplo el comer es más importante que explorar el ambiente de su alrededor, pero no lo es más que defenderse de algún enemigo. Es importante que el código interrumpa la acción actual si una acción con mayor prioridad debe de tomar el control del sistema en algún momento. Cuando esto sucede se dice que el comportamiento de nivel menor ha sido suprimido. [Bagnall, 2002]

En leJOS el programador únicamente debe de definir cuándo es que el comportamiento debe estar activo y especificar la importancia de ese comportamiento, es decir fijarle una prioridad en relación al resto de los comportamientos definidos, puesto que ya está implementado un árbitro (*arbitrator*) el cual tomará todos los comportamientos definidos y los mandará a ejecutar.

En la siguiente sección se explicará más a detalle como fue que se implementaron todas estas ideas en el robot, se explicarán las conductas que fueron definidas para el robot y cuándo es que entran en acción. También se hablará sobre el robot que se construyó y el

porqué se hizo así. Finalmente en la última parte del capítulo haremos referencia al modelo de pruebas planeado para el robot y hablaremos sobre los resultados obtenidos.

## 4.2 DESARROLLO

### 4.2.1 SOFTWARE

Originalmente este trabajo estaba orientado por los trabajos realizados en el Laboratorio de Robots Móviles del Georgia Tech. [WEB 10] En algunas de estas investigaciones, desarrolladas principalmente por Ronald Arkin, se realizó la implementación de modelos de comportamiento animal en un robot. En base a ello fue que se decidió tratar de desarrollar una pequeña aplicación de algunos comportamientos básicos de los animales en un robot pequeño y hasta cierto punto limitado como el que se construyó usando Lego Mindstorms.

Los tres comportamientos básicos que fueron implementados en este trabajo son: comer, aparearse y huir de la presencia de un predador. También fueron incorporadas tres variables *motivacionales*, dos para llevar un control de los niveles de hambre y de apareamiento del robot y una variable boolean que indica la presencia de un predador en el ambiente. El funcionamiento de las variables es bastante simple, al inicio de la ejecución sus valores son 0 (cero) y aproximadamente cada 5 segundos comienzan a incrementarse, el apareamiento se incrementa en una unidad cada 5 segundos pero el hambre se incrementa en 2 unidades. La variable Miedo como sólo pude tener dos posibles valores (verdadero o falso) al inicio de la ejecución es falsa y así se mantiene siempre hasta que se encuentra un predador en el ambiente y es cuando su valor cambia a verdadero.

Dado que la implementación a nivel de software que se realizó está basada en el API del Modelo de Control de Comportamiento que como ya mencionamos es parte de leJOS antes de empezar a programar se tuvieron que diseñar los comportamientos (conductas) que definirían las acciones del robot. A continuación se muestran los diagramas de las conductas identificadas:

| **Conducta:** Avanzar | |
|---|---|
| **Condición:** Siempre que otra conducta no esté activada. (Nivel más bajo.) | **Reacción:** Avanzar hacia el frente. |

**Figura 4.2** Conducta que define que el robot avance hacia el frente. [Elaboración Propia]

| **Conducta:** Golpe en sensor 1 | |
|---|---|
| **Condición:** Sensor de Tacto S1 Liberado | **Reacción:** Acercarse e identificar estímulo. Reaccionar al estímulo. |

**Figura 4.3** Conducta que define qué hacer cuando el sensor izquierdo toca un objeto. [Elaboración Propia]

| **Conducta:** Golpe en sensor 3 | |
|---|---|
| **Condición:**<br>Sensor de Tacto S3<br>Liberado | **Reacción:**<br>Acercarse e identificar estímulo.<br>Reaccionar al estímulo. |

**Figura 4.4** Conducta que define qué hacer cuando el sensor derecho toca un objeto. [Elaboración Propia]

| **Conducta:** Incrementa Variable | |
|---|---|
| **Condición:**<br>El contador marca que han transcurrido 5 segundos. | **Reacción:**<br>Incrementar las variables:<br>Aparearse + 1<br>Hambre + 2 |

**Figura 4.5** Conducta que define cómo se incrementan las variables. [Elaboración Propia]

| **Conducta:** Cambiar Dirección | |
|---|---|
| **Condición:**<br>El contador marca que han transcurrido 20 segundos. | **Reacción:**<br>Cambiar dirección hacia un punto claro o el más obscuro. |

**Figura 4.6** Conducta que define cuándo se cambia la dirección del robot. [Elaboración Propia]

| **Conducta:** Hay Predador | |
|---|---|
| **Condición:** La variable Miedo = 1 lo cual indica que hay un predador en el ambiente. | **Reacción:** Detener el movimiento y alejarse del predador. |

**Figura 4.7** Conducta que define qué se hace ante la presencia de un predador. [Elaboración Propia]

Una vez que las conductas fueron definidas sigue el paso de decidir cuáles son más importantes que otras para así ponerlas en un nivel superior. Recordemos que las conductas situadas en los niveles superiores cuando tomen el control van a suprimir a los comportamientos inferiores, y una vez que se termina de ejecutar el comportamiento que toma el control es el que se encuentra en el nivel más bajo (prioridad menor), en este caso sería el comportamiento de *Avanzar*. El comportamiento de mayor prioridad será el de *hayPredador* pues cuando el robot percibe que alguien desea atacarlo tratará de alejarse del lugar y en todo momento su prioridad será la de salvar su vida. En el siguiente cuadro podemos observar la prioridad asignada a los comportamientos que se definieron anteriormente:

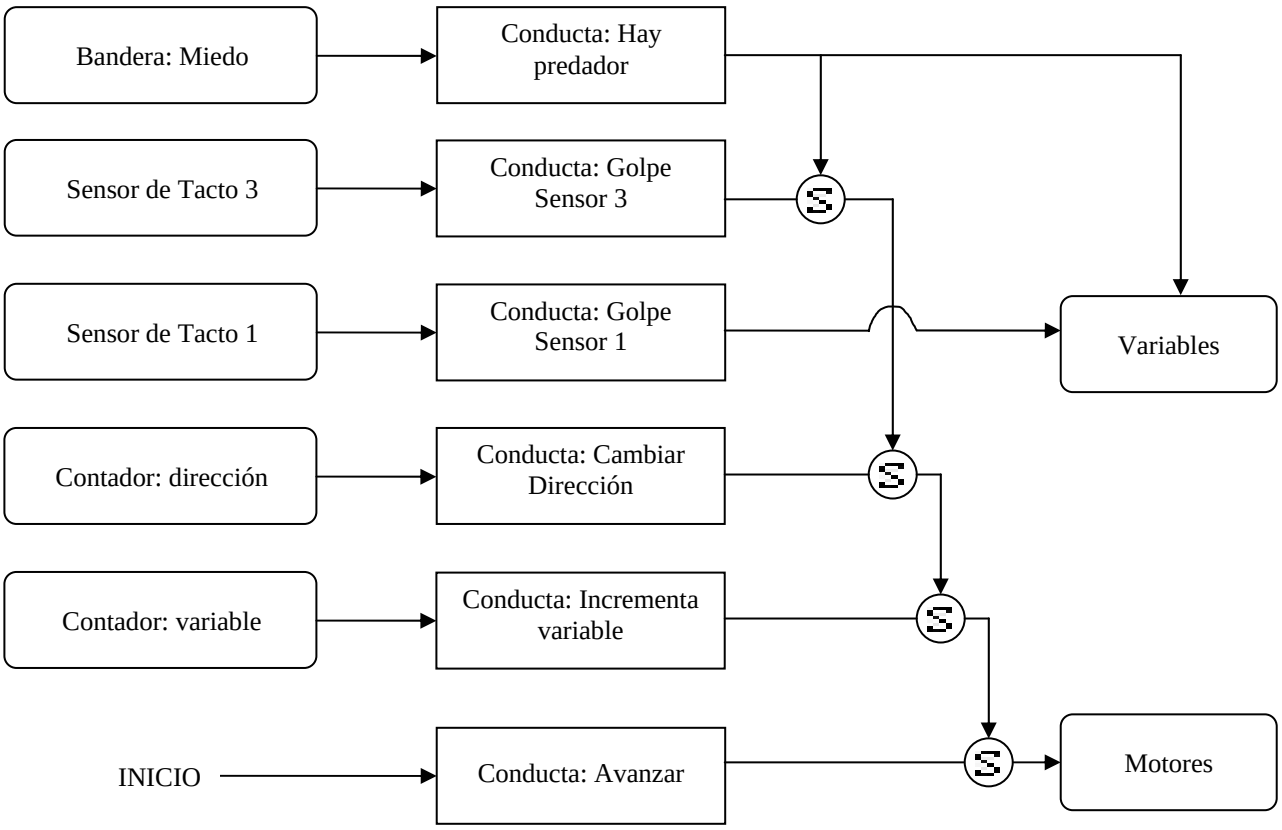


**Figura 4.8** Jerarquía de los comportamientos implementados. [Elaboración Propia]

Como podemos observar en el esquema anterior los comportamientos superiores tiene una prioridad mayor que los comportamientos ubicados en la parte inferior del diagrama y para realizarlo se tomó como base el estilo de diagrama desarrollado por Rodney Brooks. [Bagnall, 2002] Cómo podemos observar los puntos marcados como S definen los puntos en que los comportamientos superiores *suprimen* a los inferiores. La regla del Control de Comportamiento es que cualquier comportamiento del nivel inferior va a ser suprimido cuando cualquier comportamiento con mayor prioridad entre en ejecución, lo cual da como resultado que solo una conducta puede estar en ejecución en cualquier

momento. Aunque pareciera que de esta manera se limita el desempeño del robot pues solo ejecuta un comportamiento a la vez, este comportamiento puede incluir más de una función a realizar cuando se tiene el control.

Otro detalle importante a considerar al momento de programar fue el decidir qué serían los estímulos presentes en el ambiente. Era importante definir los requerimientos tanto en color como en tamaño y forma. Se decidió trabajar con latas de refresco llenas, para que el robot no pudiera empujarlas. Estas resultaron ser de buen tamaño para trabajar con el robot pues no eran ni exageradamente grandes ni tan chicas como para que las antenas no pudieran percibirlas; la forma de la lata facilitó el hecho de que el robot pudiera acercarse para identificar el color. El color es lo más importante pues va a definir el tipo de estímulo que tiene el robot frente de él. Esta fue una de las cosas más difíciles de decidir pues se eligieron varios colores con los cuales hacer pruebas al sensor de luz para poder registrar las diversas intensidades que el sensor leía. Después de varios colores de prueba, y de varios tipos de papel (no se podía elegir un papel muy brilloso pues se afectaba la lectura) se decidió utilizar los colores que se reflejan en la siguiente tabla, donde también se indica lo que ese color representa en cuanto a estímulos para el robot.

| **COLOR** | **REPRESENTA** |
|---|---|
| Negro | Alimento |
| Verde | Pareja |
| Blanco | Depredador |

**Figura 4.9** Relación del color con el estímulo que representa. [Elaboración Propia]

## PROGRAMACIÓN DE COMPORTAMIENTOS EN leJOS (BEHAVIOR)

Cómo se había mencionado anteriormente fue utilizado el API de Comportamiento (*Behavior API*) para tener clases más estructuradas y evitar un código largo y enredado. Antes de presentar la estructura de los comportamientos que se desarrollaron, hablaremos un poco sobre el API de Comportamiento.

El API de Comportamientos implementado en leJOS es muy sencillo y fácil de entender puesto que solo está compuesto de una interfaz y una clase. La interfaz de comportamiento (*Behavior interface*) es usada para definir cada conducta. Una vez que todas las conductas han sido especificadas se le proporcionan a un árbitro (*arbitrator*) el cuál va a regular cuál de los comportamientos deberá ser activado. Todas las clases y las interfaces que se necesitan para el control de comportamiento se encuentran almacenadas en el paquete *josx.robotics*. (El término *josx* se deriva de *Java Operating System eXtension*) [Bagnall, 2002] El API para la interfaz de Comportamiento [WEB 11] está compuesto por tres métodos que se explican a continuación:

**josx.robotics.Behavior**

- *boolean takeControl ( )*

  Dentro de este método se define cuándo es que una conducta se debe volver activa, la manera de indicarlo es devolviendo un valor verdadero o falso (*boolean*).

- *void action ( )*

  Aquí se especifican las acciones que esperamos que el robot ejecute cuando el comportamiento esté activo.

- *void suppress ( )*

  El código que escribamos dentro de este método debe de terminar las acciones que fueron especificadas en el método *action( )* e incluso se puede usar éste método para actualizar algunos datos después de que la conducta terminó de realizarse.

Como se puede observar los métodos incluidos en esta interfaz son sencillos de implementar, si por ejemplo un robot está compuesto de tres comportamientos entonces el programador tendrá que crear tres clases, cada una de la cuales implementará la interfaz de comportamiento y se entiende con esto que desarrollara los métodos para cada una. Una vez que se tienen listas las clases los objetos de Comportamiento deben de entregarse al árbitro (*arbitrator*) para que él las manipule. A continuación se muestra el API [WEB 11] para la clase del árbitro:

**josx.robotics.Arbitrator**

- *public Arbitrator (Behavior [] comportamientos)*

  Se debe de crear un objeto de tipo Arbitrator el cual va a regular cuando es que cada uno de los comportamientos está activo. La manera en que se pasan los comportamientos es en un arreglo de comportamientos y la prioridad se define por el índice que ocupe dentro del arreglo, mientras mayor sea el índice dentro del arreglo mayor es la prioridad de ese comportamiento.

  Parámetros: *comportamientos* Un arreglo de comportamientos.

- *public void start( )*

Empieza la ejecución del árbitro, es decir se empiezan a ejecutar los comportamientos.

La clase del árbitro (*Arbitrator*) es aún mas sencilla de entender que la del Comportamiento e incluso es más sencilla de programar. Cuando se hace la instancia de un objeto de tipo *Arbitrator* se le pasa como parámetro un arreglo que contiene los comportamientos, después de esto ya es posible mandar a llamar al método *start( )* con lo cuál se comienza a decidir cuál de los comportamientos debe de estar en ejecución.

La manera en que trabaja el árbitro es muy sencilla y se explica a continuación: Una vez que se ha iniciado el funcionamiento el árbitro recorre todos los métodos *takeControl ()* de los comportamientos incluidos en el arreglo, iniciando con el de mayor prioridad (es decir, el comportamiento que se encuentra en la mayor posición del arreglo) y así va recorriendo de manera descendiente hasta que se encuentra con un comportamiento que quiere (es decir que puede) tomar el control en ese momento. Una vez que encuentra un comportamiento a ejecutar entonces manda a llamar al método *action ()* de dicha conducta. Si dos comportamientos quieren tomar control al mismo tiempo, el comportamiento que se va a ejecutar es aquél que tenga la mayor prioridad. Ejemplificaremos esto con las conductas implementadas para este proyecto, como podemos observar en la Figura 4.8, en el comportamiento Avanzar su método *takeControl ( )* siempre regresa como valor verdadero, es decir, el robot siempre va a estar en movimiento, pero para el caso de la figura podemos suponer que el Sensor de Tacto 3 ha hecho contacto con algo, por lo cual su método *takeControl ( )* también está regresando un valor verdadero. En este caso como la prioridad del comportamiento asociado al sensor de tacto es mayor entonces se suprime al

comportamiento avanzar y es la conducta Golpe Sensor 3 la que se activa y tiene control de los motores en ese momento.

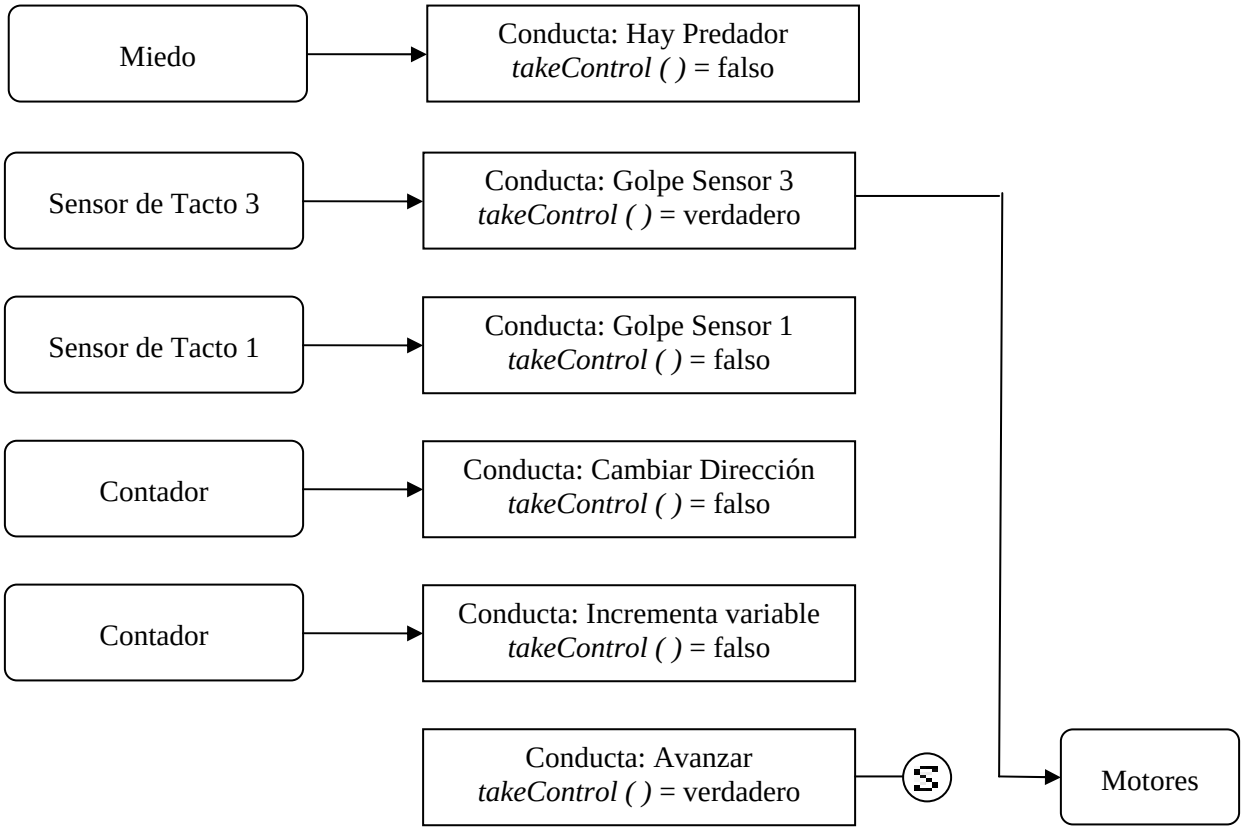


**Figura 4.10** Ejemplo sobre cómo un comportamiento con prioridad mayor suprime los comportamientos inferiores. [Elaboración Propia]

### 4.2.2 ROBOT

Para probar los comportamientos que se describieron en la sección anterior se decidió construir un robot sencillo pero que pudiera desplazarse sin ningún problema por el ambiente, además se esperaba que el robot pudiera girar de una manera tranquila para así poderse acercar lo suficiente e identificar el estímulo con el que hizo contacto. Los diseños

que se presentaran se construyeron en base a los mostrados en "*Robotics Invention System 2.0 Constructopedia*" la cual viene incluida en el kit de Lego Mindstorms.

Aunque entre la idea original que se tenía del robot y el modelo final presentado en este trabajo hubo una variedad de cambios significativos, hubo algunos detalles que no cambiaron y son los siguientes:

- Los motores se conectan a los puertos de salida A y C (out*puts* A y C) del bloque RCX. Se conectan los dos en el mismo sentido para que cuando giren lo hagan en la misma dirección.
- Los sensores de tacto se conectan en los puertos de entrada 1 y 3 (in*puts* 1 y 3) del bloque RCX
- El sensor de luz se conecta al puerto de entrada 2 (in*put* 2) del bloque del RCX.

Esto quiere decir es posible cambiar el diseño físico del robot si conservamos estas mismas condiciones en la manera de conectar los sensores y actuadores (motores).

Originalmente se había construido un robot que caminaba gracias a un par de pequeñas piernas, pero no mostraba mucho equilibrio y en ocasiones los engranes adaptados a los motores eran los que sostenían a la estructura. Esto no era bueno pues, además de que a la larga los engranes se iban a desgastar, el movimiento no era fluído ni muy preciso. Al momento de realizar este modelo no se implementó ningún tipo de defensa (*bumper*) pues desde el principio fue notable que las piernas no fueron una buena solución por lo que no se siguió trabajando más en esa idea.

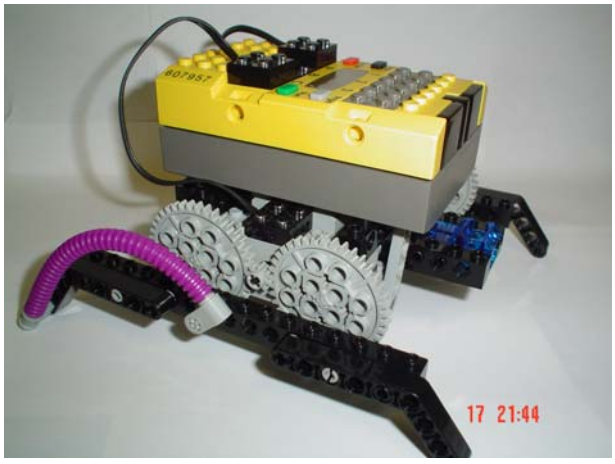


**Imagen 4.1** Diseño del primer robot construido que se apoyaba en 2 patitas. [Elaboración Propia]

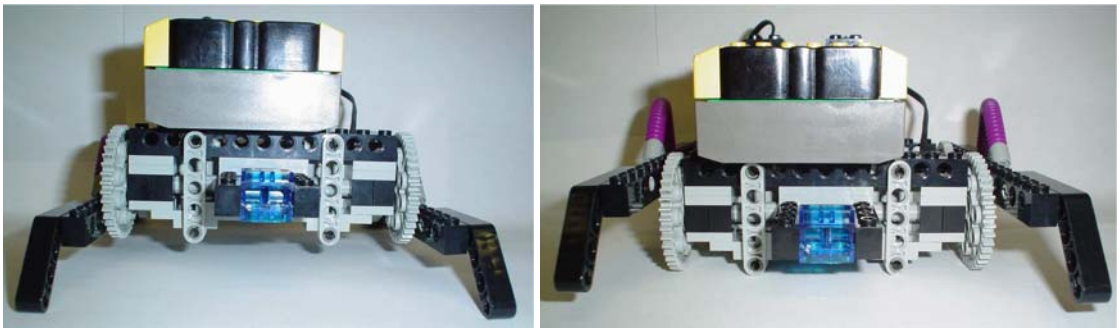

**Imagen 4.2** En estas imágenes podemos observar como en un principio el robot se apoyaba bien sobre las dos patas (izquierda) pero al comenzar a avanzar llegaba un punto en que las patas no tocaban el suelo y se sostenía con los engranes. [Elaboración Propia]

Debido a los problemas que se observaron al tratar de trabajar con el robot sostenido por las patitas, se decidió cambiar el diseño a un modelo de orugas (del estilo de los tanques de guerra). El trabajar con las orugas da mayor precisión a los movimientos y al ser una superficie relativamente grande la que está en contacto con el suelo, la fricción es mayor, la rotación un poco más lenta y se vuelve más fácil tratar de realizar movimientos precisos. Por esta razón se decidió que el trabajar con las orugas era lo más factible pues se tenía como resultado un movimiento estable y preciso. En esta primera implementación de las

orugas se agregó al robot una defensa (*bumper*) doble en la parte de enfrente y en el centro se colocó el sensor de luz. Después de trabajar un tiempo con este esquema se concluyó que la defensa que había sido implementada no era del todo funcional y se necesitaba presionar con cierta fuerza los sensores para que se percibiera el estímulo, por lo cual se decidió cambiar de diseño.

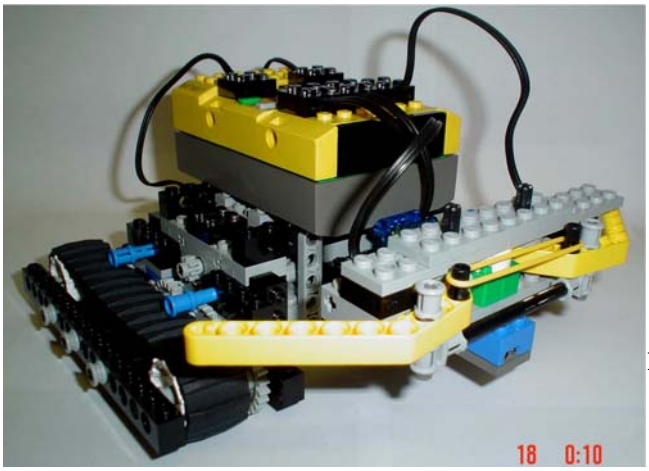


**Imagen 4.3** Diseño del segundo robot construido, ahora implementando las orugas. [Elaboración Propia]

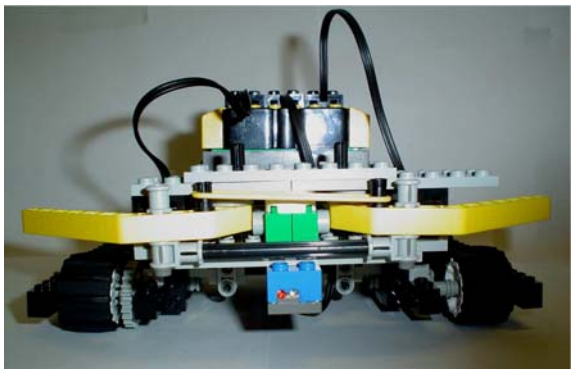

**Imagen 4.4** Detalle de la defensa colocada al diseño. El sensor de luz fue puesto en la parte baja de la defensa. [Elaboración Propia]

Como una tercera versión del robot se decidió dejar las orugas pero cambiar la defensa por unas antenas. Las antenas son más sensibles y trabajan de manera contraria a la defensa: las antenas siempre van presionadas y cuando hacen contacto con algún objeto se

liberan, solo que la presión que requieren para liberarse es menor que la requerida por la defensa para que se presione el sensor. Otro punto a favor de trabajar con las antenas es que el área de posible contacto es aún más grande que en el caso de la defensa. De igual manera que en la versión anterior del robot se volvió a colocar el sensor de luz en la parte delantera del robot, pero en esta ocasión quedó más salido que las antenas lo cual hacía que en ocasiones lo primero que chocara con los obstáculos fuera el sensor de luz y no las antenas, por lo cual no se podía percibir adecuadamente la presencia de objetos.

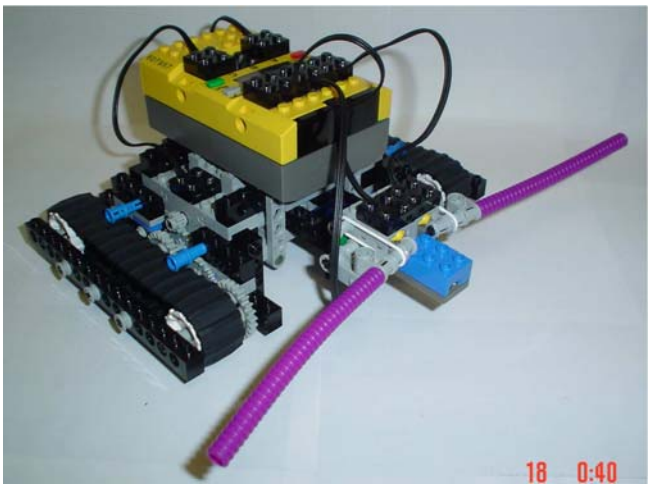


**Imagen 4.5** Tercer diseño desarrollado ahora implementando las antenas sensibles. [Elaboración Propia]

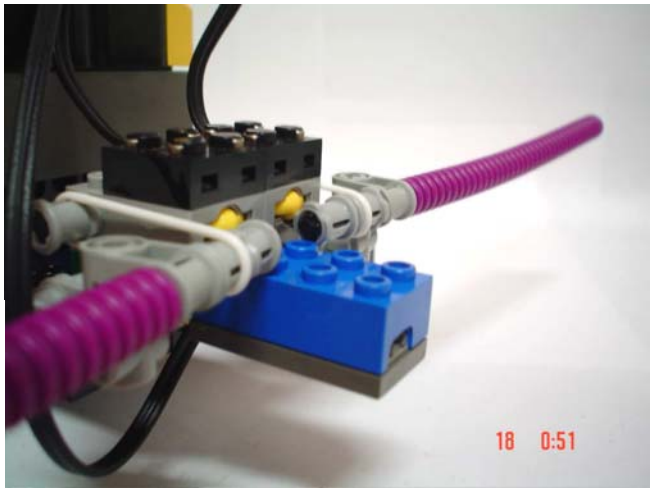


**Imagen 4.6** Detalle de los sensores de tacto presionados por las antenas, y en la parte de en medio y por delante se observa el sensor de luz. [Elaboración Propia]

La versión final que se presenta del robot es prácticamente igual a la anterior pero con un ligero cambio en la colocación del sensor de luz. Dado que en la versión anterior estaba más al frente que las antenas, se hizo un cambio para que el sensor quedará más atrás que las antenas, además de que se añadió un bloque frente al sensor para que el rayo infrarrojo con que el cual realiza la medición fuera más exacto. A continuación se presentan algunas imágenes del robot con el que se trabajó.

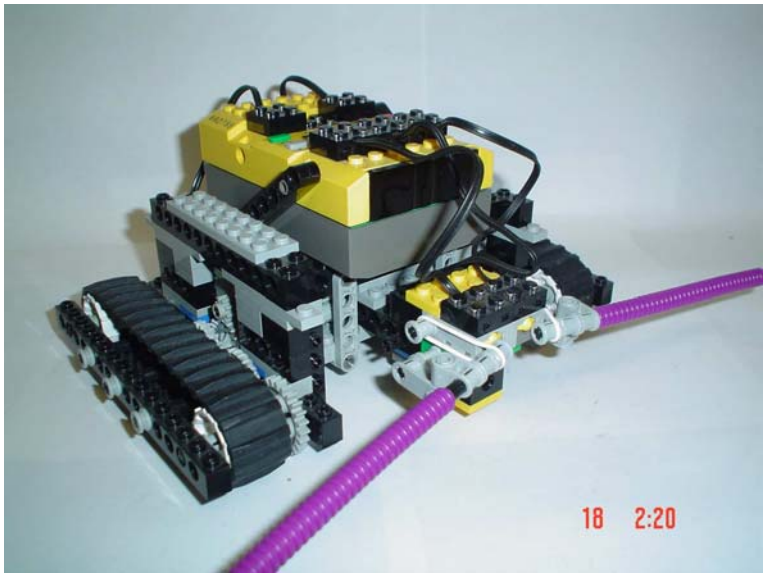


**Imagen 4.7** Diseño final con el que se trabajó. [Elaboración Propia]

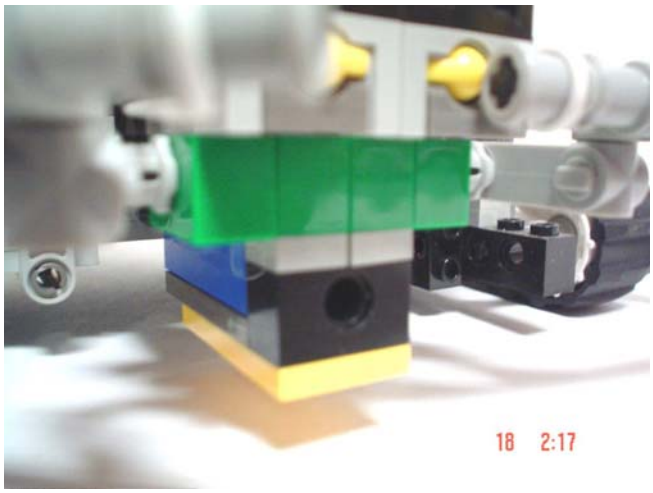


**Imagen 4.8** Detalle de la posición final de los sensores. En el fondo se observa el sensor de luz y más al frente quedaron colocados los sensores de tacto. [Elaboración Propia]

## 4.3 PRUEBAS

Antes de comenzar a hacer la implementación en leJOS se diseñaron los comportamientos que iban a conformar al robot, teniendo una idea de la situación en la que debían de ejecutarse y lo que se esperaba que sucediera, estos diagramas ya han sido mostrados en la sección anterior. Una vez que estos comportamientos estuvieron bien definidos, sabiendo lo que se esperaba del robot en ciertas situaciones, se comenzó a realizar la implementación en leJOS para poder comenzar a probar los módulos en el robot construido.

Como primera implementación se construyeron tres comportamientos para ser probados y estos fueron los de menor prioridad: el avanzar hacia el frente todo el tiempo, y los comportamientos para controlar los sensores de tacto y el sensor de luz. Se esperaba que el robot fuera capaz de recorrer un área y cuando chocara con algo se alejara del obstáculo tomando antes una medición de la intensidad de luz reflejada en el objeto con el cual había hecho contacto, es decir se buscaba desarrollar las bases para más adelante identificar el color del estímulo. En esta primera parte se observaron varios problemas que no habían sido considerados en un inicio. Por ejemplo el robot debía ser capaz de diferenciar cuando chocaba con una pared y cuando lo hacía en realidad con un estímulo útil para él. Se trató de controlar esto usando el sensor de luz y nos hallamos ante la necesidad de ser capaces de identificar con el sensor al menos cuatro colores diferentes, lo cual resultó prácticamente imposible pues después de varias pruebas de color e intensidades el robot sólo era capaz de distinguir entre tres colores. Entonces se consideró la idea de marcar un área dentro de la cual se moviera el robot, dicha área estaría delimitada por líneas negras que el robot reconocería con otro sensor de luz. Pero aquí nuevamente nos encontrábamos ante nuevos

obstáculos pues de los tres puertos disponibles en el RCX para conectar sensores dos estaban siendo ocupados por los sensores de tacto y sólo quedaba uno libre para los dos sensores de luz que se deseaban conectar. Debemos de considerar que un solo sensor de luz no podía realizar las dos actividades, pues por un lado se necesitaba llevar un sensor de frente que fuera capaz de tomar la medición de luz del objeto con el cual hiciera contacto, y por otro lado, se necesitaba un sensor señalando hacia el suelo el cual iba a registrar el cambio en la intensidad de la luz cuando se tratara de cruzar la línea negra que delimitaba el área de prueba.

Ante estas limitaciones, de no poder agregar un sensor más al robot y no poder distinguir más de tres colores, se tuvo que tomar la decisión de tener que considerar algunas cosas por adelantado, es decir se crearon ciertas especificaciones dentro de las cuales el comportamiento del robot se espera sea normal. Dichas especificaciones son las siguientes: Consideramos que el robot se mueve en un espacio infinito, donde no hay más obstáculos que los estímulos significativos para él (pareja, alimento, predador). Debido a que el sensor de luz solo toma una lectura lo suficientemente confiable cuando está muy cerca de los objetos, consideramos normal que el robot se coloque justamente enfrente de él para tomar un valor, sin importar que este estímulo represente un predador o una pareja. Y por último y como se explicará mas adelante, también se considera que el robot nunca llegará exactamente de frente hacia el estímulo, es decir nunca chocará con la parte central de sus antenas, por ser este un “punto muerto” para él (carece de sensibilidad).

Prácticamente esta fue la parte de la implementación que mas trabajo tomó realizar, pues se tenían varias ideas para tratar de solucionar los problemas pero al final no se pudo dar una mejor solución que la anterior, considerando la situación del robot y su construcción entre otras cosas. Es muy probable que en otros casos y situaciones estos

problemas puedan ser resueltos de un mejor modo, pero para el caso de este trabajo no se encontró otra solución viable.

Después de este punto simplemente se trabajó en perfeccionar la reacción del robot cuando hacía contacto con un estímulo. Inicialmente se estaba trabajando con un robot que al chocar se alejaba inmediatamente del estímulo, pero al tratar de simular el comportamiento de un animal no se esperaba que el robot se alejara sino por el contrario que se acercara más a reconocer el estímulo. Se hicieron varias pruebas para controlar esta idea pues de principio no funcionaba del todo bien, pero después de algunas modificaciones se logró hacer que el robot se colocara lo mejor posible al frente del estímulo.

Otro detalle que faltaba por resolver era el que el robot cambiara de dirección en algún momento de su recorrido, independientemente de alejarse de los estímulos después de haberlos identificado. Esto se solucionó agregando un comportamiento más al robot, éste consiste en manejar una variable de tiempo (Timer) la cual cuenta aproximadamente 50 segundos y al llegar a este tiempo, el robot gira aleatoriamente hacia el punto más obscuro si su variable mayor es el Hambre o hacia un punto claro cuando su variable mayor es Aparearse, para después continuar vagando por el ambiente.

Una vez que se tuvo el diseño sobre el movimiento del robot por el ambiente y se logró reconocer los estímulos presentes según su color, se implementaron los métodos para incrementar el valor de las variables motivacionales del robot. Simultáneamente se implementaron los métodos para hacer que el robot reaccione a los estímulos y se creó el comportamiento encargado de reaccionar ante la presencia de algún predador en el ambiente. Este es el comportamiento de mayor prioridad para el robot pues cuando el robot se ve amenazado lo único que se espera es que trate de huir y ponerse a salvo.

Al llegar a este punto no restó más que comenzar a realizar las pruebas diseñadas para el robot y ver cómo reaccionaba éste ante los estímulos presentes en el ambiente para que, en los casos que se requiriera se hicieran los cambios y ajustes necesarios. Se decidió que el robot podía o no reaccionar a los estímulos presentes en el ambiente es decir, en ocasiones, aunque estuviera frente a un estímulo que representaba alimento, podría ser que el robot no tuviera tanta hambre como para tomarlo, entonces simplemente lo detectaba pero lo ignoraba. Esta decisión se toma en base al valor de la variable correspondiente al estímulo presente al momento de hacer contacto; durante la ejecución esto se refleja con algunos sonidos emitidos por el robot debido a que el desplegar un mensaje en la pantalla del RCX requiere que se duerma por un momento el hilo (thread) de la ejecución actual, lo cual impide que se observe un movimiento fluido del robot.

En las pruebas realizadas al robot lo que se pudo observar con mayor frecuencia es que el lugar donde se realicen y la luz del mismo influyen de manera extremadamente significativa en el buen comportamiento del robot, básicamente por el sensor de luz que en ocasiones no es capaz de distinguir entre los tres colores. En el 50% de las veces las pruebas no fueron exitosas porque la luz no era la adecuada, generalmente faltaba luz. De ese 50% de pruebas solo podemos considerar que un 10% no funcionó por exceso de luz. Como se mencionó con anterioridad el caso en que el robot se encuentre completamente de frente con un estímulo no se consideró en este trabajo y en las pruebas realizadas se habrá presentado en un máximo de 15% ese detalle, pero de manera general el robot alcanzaba a “sentir” el contacto con algún estímulo. Otro detalle que influye en el desempeño general del robot es que si el espacio donde se prueban es muy chico el robot tiende a topar con las paredes o los límites del área de prueba y, al no estar validando donde ya no es un área de prueba válida en un 30% de las veces el robot comenzaba a alejarse de los estímulos

presentes. En general cuando el robot se acerca sin problemas a los estímulos reacciona de buena manera, dependiendo del valor actual de sus variables.

Lamentablemente podemos decir que el buen funcionamiento del robot depende de la situación de la luz actual en el lugar donde se hagan las pruebas, pues con falta de luz para el robot todas las latas reflejan muy pocos valores y las considera a todas prácticamente negras; por el lado contrario tenemos que si hay demasiada presencia de luz entonces todas las latas son consideradas como blancas. A pesar de ser una aplicación interesante que nos permite ver cómo el robot puede reaccionar a diferentes cosas presentes en el ambiente se ve limitada por el sensor de luz, aún así podemos decir que el resultado final fue en efecto un comportamiento inteligentemente visible a los ojos humanos, tal vez no de una manera fácil y rápidamente evidente por las limitaciones propias del RCX pero el desempeño en general fue coherente y satisfactorio. Las clases implementadas se presentan en el Apéndice 1 junto con la documentación basada en *javadoc* que se realizó.