

Máquinas de estados finitos

Vamos a empezar con nuestro recorrido para crear aplicaciones de inteligencia artificial. En este capítulo cubriremos a las máquinas de estados finitos. Este tipo de máquinas computacionales son muy simples; se utilizan principalmente en sistemas de control y parecen presentar una inteligencia muy sencilla.

¿Qué son?	28
Crear una máquina de estados finitos	32
Desarrollo de la máquina de estados finitos	34
Resumen	51
Actividades	52

¿QUÉ SON?

En primer término, conozcamos qué es una **máquina de estados finitos**. Consiste en un modelo de comportamiento de un sistema. Este modelo tiene una cantidad **finita** de posibles **estados** y el sistema puede cambiar de estado cuando se cumple una determinada condición. A esta condición la llamaremos **condición de transición**. Cada uno de los estados determina el tipo de acción que la máquina lleva a cabo. Una transición indica el cambio de estado.

Para representar una máquina de estados finitos usamos un **diagrama de estados**. En este diagrama podemos observar fácilmente a la máquina. Los estados se representan por círculos. Colocamos el nombre del estado en su interior. Por medio de flechas, indicamos las transiciones entre los estados. Estas flechas llevan un rótulo que indica la condición de transición. Los estados son numerados y en su interior pueden indicarse las acciones que ejecutan.

Una de las máquinas de estados finitos más conocidas de la historia es la **máquina de Turing**, creada en 1936. Como vimos en el capítulo anterior, Alan Turing fue quizás el científico computacional más importante del siglo XX. La máquina de Turing es una máquina de estados finitos muy simple, pero con capacidad de simular la lógica de cualquier computadora, ya sean computadoras que existan o que vayan a existir. La máquina de Turing se considera más un experimento mental que una máquina práctica.

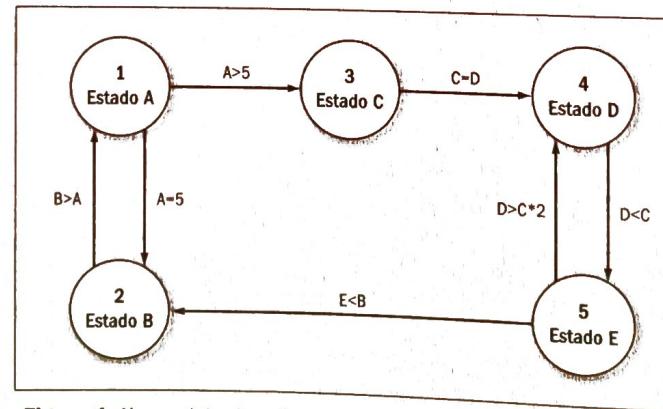


Figura 1. Un modelo de máquina de estados con sus componentes.
Podemos observar los diferentes estados y las transiciones entre ellos.

La máquina de estados finitos permanecerá en el mismo estado hasta que se lleve a cabo una transición. Mientras se encuentra en ese estado, llevará a cabo una acción.

Hay cuatro tipos básicos de acciones: de entrada, de salida, de ingreso y de transición.

Cuando se llega a un nuevo estado y se ejecuta una acción, a ésta se la conoce como **acción de ingreso**. Es lo primero que se hace cuando llegamos al nuevo estado. Si tenemos que llevar a cabo una acción al salir de un estado, decimos que esa acción es una **acción de salida**. Por ejemplo, podemos necesitar que un estado tenga que leer o escribir un archivo en el disco duro. Para hacerlo correctamente y evitarnos problemas, podemos colocar, como acción de ingreso, abrir el archivo y, como acción de salida, cerrarlo.

Frecuentemente, nos encontraremos con que el sistema recibe una entrada y debemos llevar a cabo una acción. Esta acción se conoce como **acción de entrada**, y dependerá de la entrada que recibimos y del estado en el cual se encuentre la máquina en ese momento. Si fuera necesario que la acción se llevara a cabo durante una transición, entonces la acción se conoce como **acción de transición**.

Las máquinas de estados finitos se clasifican básicamente en dos grupos: los reconocedores y los transductores. Los **reconocedores** se caracterizan por su salida **binaria**. Esto es, que en su salida obtenemos el valor de **falso** o **verdadero**. Los **transductores**, por su parte, nos dan una salida, que depende de la entrada, y el estado, con sus acciones.

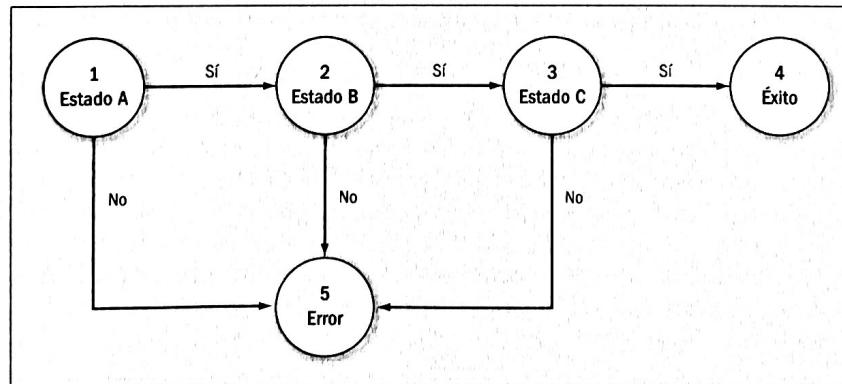


Figura 2. Este diagrama es un ejemplo de reconocedor.
Podemos ver que finaliza en Sí o No, es decir es binario en su salida.

Los transductores se utilizan principalmente en aplicaciones de control, y son las máquinas de estados finitos que desarrollaremos en este capítulo. Existen diferentes tipos de transductores, pero los más básicos son la máquina de Moore y la máquina de Mealy, que explicaremos a continuación.

En la **máquina de Moore**, la salida dependerá del estado en el que se encuentra la máquina. En comparación, en la **máquina de Mealy**, la salida dependerá del estado en el que se encuentra la máquina y la entrada.

Ya hemos visto que podemos utilizar diagramas para las máquinas de estados finitos, pero también existe la posibilidad de usar otra herramienta conocida como la **tabla de transición de estados**. Ésta es una tabla en la cual podemos indicar los estados, las condiciones de transición y las acciones. Este tipo de tablas resulta muy útil en el momento de codificar la máquina de estados en la computadora.

Existen diversos tipos de tablas de transición, pero nosotros veremos una variante muy sencilla. Básicamente, crearemos una tabla en la cual vamos a colocar los posibles estados en el eje vertical y el eje horizontal. El eje vertical indicará el estado actual en el que se encuentra la máquina de estados, y el eje horizontal, el estado al cual se dirigirá después de la transición. En las celdas de la tabla, colocaremos la condición de transición. Si algún estado no tiene transición hacia otro estado, entonces, en la celda, se coloca una línea horizontal para indicar que no hay transición.

ESTADO ACTUAL/ NUEVO ESTADO	ESTADO A	ESTADO B	ESTADO C
Estado A		A>5	A<C
Estado B	Costo=0		
Estado C	Costo>100	C==A	-

Tabla 1. Esta tabla nos muestra las transiciones entre los diferentes estados.

Las máquinas de estados finitos son muy fáciles de programar. Hemos visto las nociones teóricas de ellas, pero también veremos que no presentan ninguna dificultad en su programación, de hecho, puede hacerse de manera rápida una vez que tenemos el diagrama y la tabla de estados. También puede depurarse fácilmente, trazando el problema a un estado en particular o al código donde se llevan a cabo las transiciones.

Este tipo de máquinas, cuando se codifican correctamente, resultan muy eficientes, pues el código necesario para implementar su lógica es pequeño y rápido. Se pue-

III DIAGRAMAS PARA MÁQUINAS DE ESTADOS FINITOS

Las máquinas de estados finitos también se pueden diagramar usando **UML** (*Unified Modeling Language*). El diagrama usado en UML para estas máquinas desciende de las **cartas de estados** creadas por **David Harel** en 1987. Sin importar el estilo de diagrama que seleccionemos, resulta fundamental que nos provea de todos los datos importantes de manera sencilla y rápida.

den programar con diferentes metodologías, ya sean **estructuradas** u **orientadas a objetos**. Una vez conocida cuál es su programación, nos sentiremos libres de implementarlas usando otras estructuras de control. Si la implementación se hace de manera correcta, entonces, la expansión de la máquina para incluir nuevos estados no debe presentar ninguna dificultad.

Para que podamos diseñar una máquina de estados finitos, primero observemos el comportamiento del sistema. A este comportamiento, lo dividiremos en diferentes estados. Por ejemplo, un calentador de agua tiene dos estados: cuando está prendido calentando agua y cuando está apagado en espera. Una lámpara también presenta dos estados: prendido y apagado. Naturalmente, los sistemas complejos podrán tener más estados.

Una vez definidos los estados, procedemos a ver cuáles son las transiciones que existen, es decir, observamos qué estado puede cambiar a qué posible estado. Al conocer estas transiciones, nos daremos cuenta de que el cambio de estado realiza cuando determinada condición se cumple. En nuestro ejemplo, el calentador de agua puede de ir de prendido a apagado y de apagado a prendido. Tenemos dos transiciones en la máquina. Ahora bien, hemos observado que el calentador se prende cuando la temperatura del agua es menor a 50 grados centígrados. Ésa es una condición de transición. También vimos que el calentador se apaga cuando la temperatura del agua es mayor a 85 grados centígrados. Ésa es la otra condición de transición.

Ya tenemos todos los elementos de la máquina de estados finitos que representan al calentador de agua.

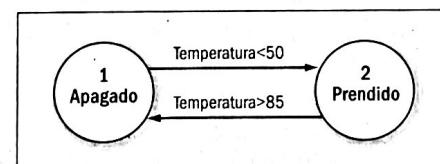


Figura 3. Éste es el diagrama de máquina de estados del calentador de agua. Tenemos dos estados y dos transiciones.

INDICAR EL ESTADO DE INICIO

Para indicar cuál es el estado en que inicia una máquina de estados finitos, podemos utilizar una flecha que no provenga de ningún otro estado. Esto, con el fin de no confundirla con una transición. Algunos diseños de nuestras máquinas tendrán un estado de inicio bien definido y no funcionarán correctamente si arrancan en otro estado, por eso es importante diagramarlo.

2. MÁQUINAS DE ESTADOS FINITOS

Ahora veamos cómo queda la tabla de transiciones. Como sólo tenemos dos estados, la tabla que creamos es muy sencilla.

ESTADO ACTUAL/NUEVO ESTADO	PRENDIDO	APAGADO
Prendido		Temperatura>85
Apagado	Temperatura<50	

Tabla 2. Tabla de transiciones del calentador de agua.

Ya que hemos visto una máquina de estados muy sencilla, es tiempo de que creamos una máquina de estados finitos con C#.

CREAR UNA MÁQUINA DE ESTADOS FINITOS

Por fin llegó el momento de construir nuestra propia máquina de estados finitos. Primero, debemos definir la aplicación. Imaginemos un pequeño robot que recogerá diez objetos colocados al azar. Si en algún momento el robot empieza a quedarse sin batería, tendrá que ir a recoger una nueva. Si el robot se queda por completo sin batería, se muere.

Sobre la base de la aplicación que necesitamos crear, observamos los estados necesarios: es preciso un estado de búsqueda en el cual el robot se dirija hacia el objeto. Otro estado en el que decida cuál es el siguiente objeto por recoger. Un estado necesario, en el cual nuestro robot deberá ir por la nueva batería. También cuando llega a la nueva batería y recarga la energía, ése es otro estado. Cuando el robot muere es el siguiente estado y vamos a agregar un último estado en el cual el robot se mueva al azar cuando ya no haya más objetos que recoger.

Con estos estados, podemos trazar nuestro diagrama de estados. Recordemos que necesitamos colocar los estados adentro de círculos y, con flechas, indicamos hacia dónde va la transición.

III LA MÁQUINA DE MOORE

La máquina de Moore fue creada por **Edgard F. Moore**, quien se dedicó a la investigación de las máquinas de estados. Se la clasifica como un Autómata de estados finitos, es decir que es un modelo de comportamiento con una cantidad finita de posibles estados. Esta máquina incluye una señal de salida para cada uno de sus estados.

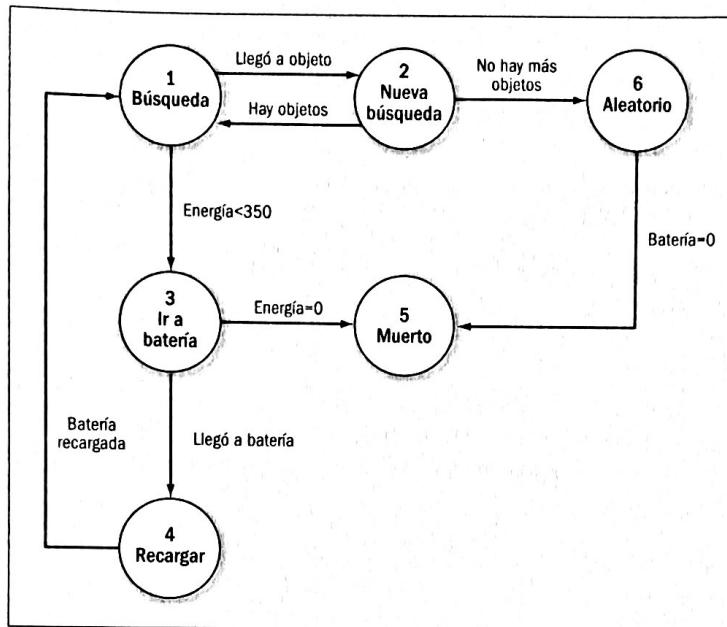


Figura 4. Éste es el diagrama de estados del robot que implementaremos.

Ahora pensemos en las transiciones. Si el robot se encuentra en el estado de búsqueda, puede seleccionar un nuevo objeto que buscar si ya llegó al objeto buscado, o dirigirse hacia la batería si su cantidad de energía es menor a 400 unidades. Ya hemos encontrado las condiciones de transición para ese estado. Continuemos haciendo lo mismo con los demás estados.

Si el robot se encuentra en el estado en el que escoge el siguiente objeto por buscar, puede cambiar de estado. Se cambia a búsqueda cuando haya seleccionado el nuevo objeto. Si ya no hay más objetos para recoger, entonces se pasa al estado de movimiento aleatorio.

En el estado aleatorio, se lleva a cabo la transición hacia el estado de muerto cuando la energía en la batería llega a cero. El estado de muerto **no tiene transición a ningún otro estado**.

Cuando se dirige a la batería y la energía de ésta llega al valor cero, entonces también la transición es hacia el estado de muerto. Si llega a la batería antes de que esto suceda, se pasa al estado donde recarga. En el estado de recargar la batería, hay una transición hacia el estado de búsqueda, para completar el ciclo de estados finitos de nuestro robot.

2. MÁQUINAS DE ESTADOS FINITOS

Estas transiciones las vemos en el diagrama de estados, pero también podemos construir la **tabla de estados**. La tabla de estados es útil y permite ver rápidamente la condición necesaria para la transición. En máquinas con muchos estados o transiciones, resulta más fácil de manejar que el diagrama.

ESTADO ACTUAL/NUUEO ESTADO	BÚSQUEDA	NUEVA BÚSQUEDA	IR A BATERÍA	RECARGAR	MUERTO	ALEATORIO
Búsqueda		Llegó al objeto Energía<400				
Nueva búsqueda	Hay objeto				No hay objeto	
			Ir a batería		Llegó a batería Energía=0	
Ir a batería						
Recargar	Energía recargada					
Muerto						
Aleatorio					Energía=0	

Tabla 3. Aquí tenemos la tabla donde se muestran los estados y las transiciones de la máquina de estados de nuestra aplicación.

Desarrollo de la máquina de estados finitos

Ahora que ya tenemos las bases del diseño, pasaremos al desarrollo del programa. Lo primero que necesitamos es crear una aplicación de formas para **Windows** en C#. El proyecto se puede llamar **MEF**, por máquina de estados finitos.

A la forma que nos presenta el proyecto, le colocamos en la propiedad de **text** el texto: **Máquina de estados finitos**. En la propiedad de **size**, colocamos el tamaño de la forma que será de 700 por 550 pixeles. La siguiente figura muestra estos valores.

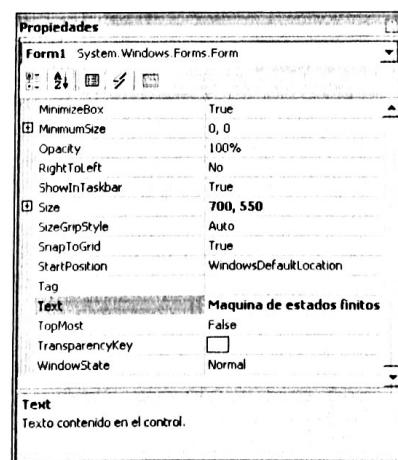


Figura 5. Aquí podemos ver el cuadro de propiedades de la forma, con los valores correspondientes.

Ahora vamos a programar el **handler** para el evento **Paint** de la forma. En el mismo cuadro de propiedades, oprimimos el botón de **Eventos** y hacemos doble clic sobre el evento llamado **Paint**.

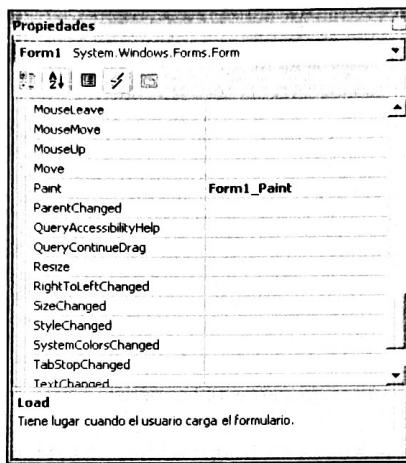


Figura 6. El botón de eventos tiene la imagen de un rayo amarillo.

Hacemos doble clic sobre el evento **Paint**. Esto genera el **handler** para el evento.

Al crear el evento, somos enviados al editor de código, por lo que es necesario regresar a la vista **Diseño** y continuar editando la forma. Ahora agregamos los menús de la aplicación. Comenzamos con un control de menús y colocamos dos submenús. El primer menú mostrará el texto de **Archivo** y contendrá únicamente un elemento que diga salir y que tenga como propiedad **Name** el valor de **mnuSalir**.

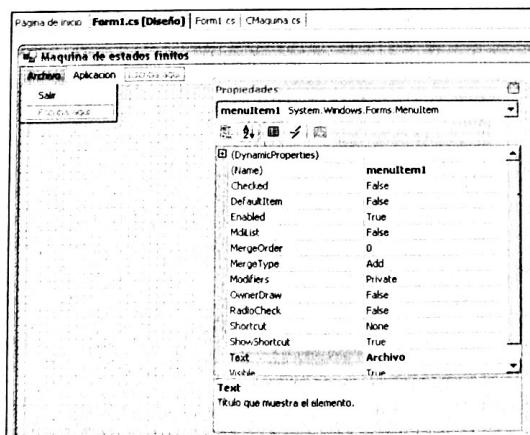


Figura 7. Aquí vemos el menú de **Archivo** con las propiedades del elemento **Salir**.

El otro submenú se llamará **Aplicación** y poseerá dos elementos. El elemento de **Inicio** nos va a servir para que la máquina de estados finitos se ejecute, y el elemento de **Paro** será utilizado para detenerla. El elemento de **Inicio** lleva en la propiedad de **Name** el valor **mnuInicio**, y el elemento de **Paro** lleva **mnuParo** en la propiedad de **Name**.

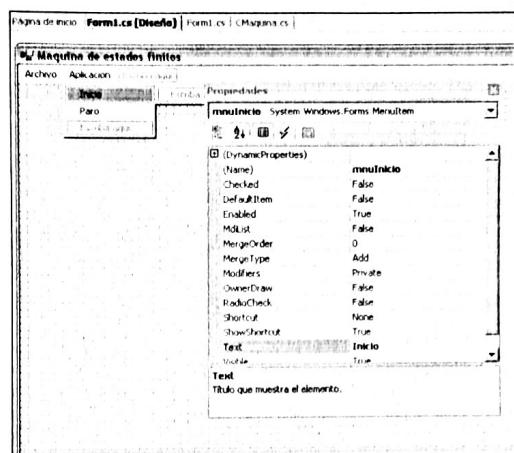


Figura 8. Observemos la estructura del menú **Aplicación** y las propiedades del elemento **Inicio**.

Ahora podemos crear los handlers para los elementos del menú. Para hacerlo, hacemos doble clic sobre el elemento del menú. Nuevamente somos enviados al editor de código, por lo que debemos regresar al editor de diseño en cada ocasión.

La máquina de estados finitos necesita ejecutarse dentro de un ciclo. No utilizaremos un ciclo de forma tradicional, sino un componente **Timer**. El **Timer** es un objeto que cada determinado tiempo genera un evento. Este evento puede ser capturado, y el handler se invoca. Así, tendremos una función que se invoca cada determinado tiempo. Cada invocación sería una iteración del ciclo. El **Timer** puede ser arrancado o detenido, y lo haremos con los menús que acabamos de generar.



LA MÁQUINA DE MEALY

La máquina de Mealy fue creada por **G.H. Mealy**, quien también era un investigador de las máquinas de estados finitos. Este tipo de máquinas pueden ser utilizadas para aplicaciones sencillas de encriptado, entre otras cosas. Esta máquina está clasificada como un transductor. En cada estado de la máquina de Mealy, se incluyen sus propias entradas y salidas.

Agregamos un **Timer** a la forma desde el cuadro de herramientas. En su propiedad de **Interval**, colocamos el valor de **5**. Este **Interval** nos permitirá indicar cada cuántos milisegundos se generará el evento.

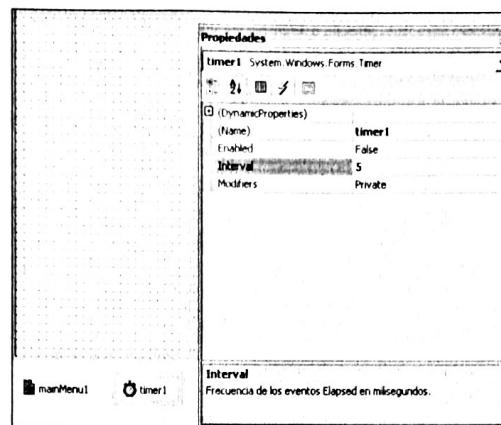


Figura 9. Aquí vemos el **Timer** y su cuadro de propiedades con el valor de **5** en la propiedad **Interval**. Al modificar este valor, cambiaremos la frecuencia con la que se activa el **Timer**.

El evento que genera el **Timer** se llama **Tick**; generamos el handler para **Tick**.

Ya está lista la estructura básica de la forma. Regresaremos a ella en unos momentos. Ahora, crearemos una nueva clase, y ésta será la clase de la máquina de estados finitos. A la nueva clase, la llamamos **CMaquina**.

Pero antes, necesitamos colocar una pequeña estructura de apoyo. Ésta contendrá los datos necesarios para indicar la posición y el estado de los objetos y de la batería.

```
public struct S_objeto
{
    public bool activo;
    public int x,y;           // Indica si el objeto es visible o no
                           // Coordenadas del objeto
```

La estructura se llama **S_objeto** y guarda las variables que permiten colocar las coordenadas del objeto y una variable de tipo **bool** que indica si el objeto está activo o no. Si el objeto no ha sido recogido, entonces lo marcamos como activo. Si el objeto ya ha sido recogido, lo marcamos como no activo al colocar el valor **false** en esa variable.

2. MÁQUINAS DE ESTADOS FINITOS

```
// Enumeracion de los diferentes estados
public enum estados
{
    BUSQUEDA,
    NBUSQUEDA,
    IRBATERIA,
    RECARGAR,
    MUERTO,
    ALEATORIO,
};

// Esta variable representa el estado actual de la maquina
private int Estado;

// Estas variables son las coordenadas del robot
private int x,y;

// Arreglo para guardar una copia de los objetos
private S_objeto[] objetos = new S_objeto[10];
private S_objeto bateria;

// Variable del indice del objeto que buscamos
private int indice;

// Variable para la energia;
private int energia;
```

Dentro de la clase encontramos los datos necesarios. Primero, una **enumeración**. En ésta, colocamos los diferentes estados que puede tener la máquina. Esto no es estrictamente necesario, pero lo vamos a utilizar para facilitarnos la programación. En este caso, **BUSQUEDA** tendrá el valor de cero, **NBUSQUEDA** el valor de 1, y así sucesivamente.

La siguiente variable es un entero que se llama **Estado**. Este dato nos permite guardar el estado actual en el que se encuentra la máquina. Luego tenemos las variables que utilizaremos para colocar las coordenadas donde se encuentra nuestro robot.

El robot necesita saber las coordenadas de los objetos y de la batería, por eso, creamos un arreglo para diez objetos y una variable para la batería. En ellos, se guardará una copia de la información sobre los objetos y la batería.

La siguiente variable guarda el índice dentro del arreglo del objeto que buscamos en ese momento. Si éste tiene el valor de -1, sabremos que ya no hay más objetos que buscar. La última variable es utilizada para saber cuánta energía tiene el robot.

Ya que tenemos los datos que va a utilizar la clase, es necesario colocar las propiedades que les corresponden.

```
// Creamos las propiedades necesarias
public int CoordX
{
    get {return x;}
}

public int CoordY
{
    get {return y;}
}

public int EstadoM
{
    get {return Estado;}
}
```

Podemos ver que todas las propiedades utilizadas son de solo lectura. Dos propiedades son usadas para las coordenadas del robot y otra, para conocer cuál es el estado actual en que se encuentra.

```
public CMquina()
{
    // Este es el constructor de la clase

    // Inicializamos las variables

    Estado=(int)estados.NBUSQUEDA
        // Colocamos el estado
        // de inicio.
    x=320;
    y=240;      // Coordenada X
    indice=-1;   // Coordenada Y
        // Empezamos como si no hubiera
        // objeto por buscar
```

```
        energia=800;
```

```
}
```

En el constructor, simplemente inicializamos las variables. El estado con el que se inicia la máquina de estados es **nueva búsqueda**, es decir, tal y como lo definimos en el diagrama. Luego colocamos las coordenadas del robot. Hasta este momento, no hay un objeto seleccionado para la búsqueda, por lo que la variable de **índice** se coloca en **-1** y le asignamos 800 unidades de energía al robot.

```
public void Inicializa(ref S_objeto [] Pobjetos, S_objeto Pbateria)
{
    // Colocamos una copia de los objetos y la bateria
    // para poder trabajar internamente con la informacion

    objetos=Pobjetos;
    bateria=Pbateria;

}
```

La siguiente función se llama **Inicializa()** y tiene como objetivo recibir el arreglo de objetos y de la batería, y guardar una copia de esa información dentro de los datos propios de la clase.

La función que tenemos a continuación es la más importante de la clase. Se llama **Control()** y será invocada por el handler del **Timer** cada 5 milisegundos. Esta función invoca a una acción en particular dependiendo del estado actual del robot y, también, verifica si se cumple alguna condición de transición para llevar a cabo el cambio de estado. Podemos pensar en esta función como en el cerebro de la máquina de estados finitos.

Veamos por partes el contenido de esta función.

```
public void Control()
{
    // Esta funcion controla la logica principal
    // de la maquina de estados

    switch(Estado)
```

```

{
    case (int)estados.BUSQUEDA:
        // Llevamos a cabo la accion del estado
        Busqueda();

        // Verificamos por transicion
        if(x==objetos[indice].x &&
           y==objetos[indice].y)
        {
            // Desactivamos el objeto encontrado
            objetos[indice].activo=false;

            // Cambiamos de estado
            Estado=(int)estados.NBUSQUEDA;

        }
        else if(energia<400)
            // Chequeamos
            condicion
            de transicion
            Estado=(int)estados.IRBATERIA;

    break;
}

```

Dentro de la función, tenemos un **switch** referenciado a la variable **Estado**, que según el valor que contenga, es la parte del código que se ejecutará. El primer caso que tenemos es para el estado de búsqueda. En cada caso, primero invocamos a la acción y luego verificamos si se cumple alguna condición. La acción está contenida dentro de la función **Busqueda()**. Luego verificamos por la transición. Según nuestra tabla, debemos cambiar al estado de nueva búsqueda cuando se llega al objeto. Se codifica esa condición en un **if**. Si éste se cumple, desactivamos el objeto buscado y cambiamos el estado. Notemos que el estado se cambia al asignar un nuevo valor a la variable **Estado**.

```

case (int)estados.NBUSQUEDA:
    // Llevamos a cabo la accion del estado
    NuevaBusqueda();

    // Verificamos por transicion
}

```

```

        if(indice==-1)
            // Si ya no hay
            // objetos, entonces
            // aleatorio
            Estado=(int)estados.ALEATORIO;
        else
            Estado=(int)estados.BUSQUEDA;

        break;
    }
}

```

El siguiente caso es el de nueva búsqueda. De manera similar, primero invocamos a la función que tiene a la acción y luego verificamos si se cumple alguna condición de transición, indicando su nuevo estado.

```

case (int)estados.IRBATERIA:
    // Llevamos a cabo la accion del estado
    IrBateria();

    // Verificamos por transicion
    if(x==bateria.x && y==bateria.y)

        Estado=(int)estados.RECARGAR;

        if(energia==0)
            Estado=(int)estados.MUERTO;

        break;

case (int)estados.RECARGAR:
    // Llevamos a cabo la accion del estado
    Recargar();

    // Hacemos la transicion
    Estado=(int)estados.BUSQUEDA;

    break;

case (int)estados.MUERTO:
    // Llevamos a cabo la accion del estado
    Muerto();
}

```

```

        Muerto();

        // No hay condicion de transicion

        break;

        case (int)estados.ALEATORIO:
            // Llevamos a cabo la accion del estado
            Aleatorio();

            // Verificamos por transicion
            if(energia==0)
                Estado=(int)estados.MUERTO;

            break;

        }
    }
}

```

Los demás casos se codifican siguiendo el mismo esquema. Primero se invoca a la función con la acción y luego se verifica por las condiciones de transición. Notemos que podemos tener un estado que no presente condiciones de transición. Esto es válido, y la máquina de estados permanecería indefinidamente en ese estado.

Ahora podemos ver las funciones que representan las acciones de los estados.

```

public void Busqueda()
{
}

```

III PROBLEMAS CON LA MÁQUINA DE ESTADOS FINITOS

Si la máquina de estados finitos no funciona como se desea, un buen lugar para iniciar la depuración es la función de **Control()**. En particular, hay que revisar que las condiciones de transición estén colocadas correctamente. Un depurador nos puede ayudar a ver los valores de las variables y a comprobar que se cumplan las condiciones adecuadamente.

```
// En esta funcion colocamos la logica del estado Busqueda

// Nos dirigimos hacia el objeto actual
if(x<objetos[indice].x)
    x++;
else if(x>objetos[indice].x)
    x--;

if(y<objetos[indice].y)
    y++;
else if(y>objetos[indice].y)
    y--;

// Disminuimos la energia
energia--;

}
```

La función **Busqueda()** es utilizada para acercar el robot hacia el objeto. Compara las coordenadas y, dependiendo del valor, la incrementa o disminuye. Hay que recordar que esto ocurrirá dentro de un ciclo controlado por el **Timer**. Cada vez que se ejecute la función, disminuimos la energía.

```
public void NuevaBusqueda()
{
    // En esta funcion colocamos la logica del estado
    // Nueva Busqueda
    // Verificamos que haya otro objeto por buscar
    indice=-1;
```

III PATRONES

En la programación orientada a objetos, existe un patrón conocido como **State** que permite desarrollar máquinas de estados finitos usando una serie de reglas definidas por él mismo. Es otra forma distinta de implementar las máquinas de estados finitos, pero sus ideas fundamentales son las mismas.

```
// Recorremos el arreglo buscando algun objeto activo
for(int n=0;n<10;n++)
{
    if(objetos[n].activo==true)
        indice=n;
}
}
```

En la siguiente función, se selecciona el índice del nuevo objeto por buscar. Simplemente, recorremos el arreglo de objetos seleccionando como el último disponible. Si no hubiera objetos activos, entonces el valor que obtiene es -1.

```
public void Aleatorio()
{
    // En esta funcion colocamos la logica del estado Aleatorio
    // Se mueve al azar

    // Creamos un objeto para tener valores aleatorios
    Random random=new Random();

    int nx=random.Next(0,3);
    int ny=random.Next(0,3);

    // Modificamos la posicion al azar
    x+=nx-1;
    y+=ny-1;

    energia--;
}

}
```

En la función **Aleatorio()**, se seleccionan al azar dos valores, uno para cada eje. Este valor se suma a la posición actual. Al restar uno, podemos hacer que a veces el valor disminuya, a veces permanezca igual y a veces se incremente.

```
public void IrBateria()
{
    // En esta funcion colocamos la logica del estado Ir Bateria
}
```

```
// Nos dirigimos hacia la bateria
if(x<bateria.x)
    x++;
else if(x>bateria.x)
    x--;

if(y<bateria.y)
    y++;
else if(y>bateria.y)
    y--;

// Disminuimos la energia
energia--;

}
```

La función **IrBateria()** es similar a la función **Busqueda()**, pero se trabaja sobre las coordenadas de la batería, también se disminuye la energía.

```
public void Recargar()
{
    // En esta funcion colocamos la logica del estado Recargar
    energia=1000;

}
```

En la función **Recargar()**, solamente colocamos 1000 en la energía. Es muy sencilla.

```
public void Muerto()
```

CONDICIONES DE TRANSICIÓN

En nuestro ejemplo, colocamos la lógica de las condiciones de transición dentro de la función **Control()**, pero también es posible colocarlas dentro de la función propia de cada estado. La decisión de dónde colocarlo dependerá de nuestra filosofía de diseño y de sus necesidades, o bien de los estándares adoptados en nuestro entorno de desarrollo.

```

    {
        // En esta función colocamos la lógica del estado Muerto

        // Sonamos un beep de la computadora
    }

```

La función de **Muerto()** solamente suena el **beep** de la computadora para indicar que el robot se ha quedado sin energía.

Es momento de regresar a la clase de la forma y trabajar sobre ella. Primero colocaremos los datos que necesitamos y, luego, se escribirá el código de los handlers que ya se han creado.

```

    // Creamos un objeto para la máquina de estados finitos
    private CMaquina maquina=new CMaquina();

    // Objetos necesarios
    public S_objeto[] ListaObjetos = new S_objeto[10];
    public S_objeto MiBateria;

```

Creamos una instancia de nuestra clase de la máquina de estados finitos. A este objeto, simplemente lo llamaremos máquina. Luego se crea el arreglo para los objetos y una variable para la batería.

```

public Form1()
{
    //
    // Necesario para admitir el Diseñador de Windows Forms
    //
    InitializeComponent();

    //
    // TODO: agregar código de constructor después de
    // llamar a InitializeComponent
    //

    // Inicializamos los objetos
}

```

```

// Creamos un objeto para tener valores aleatorios
Random random=new Random();

// Recorremos todos los objetos
for(int n=0;n<10;n++)
{
    // Colocamos las coordenadas
    ListaObjetos[n].x=random.Next(0,639);
    ListaObjetos[n].y=random.Next(0,479);

    // Lo indicamos activo
    ListaObjetos[n].activo=true;
}

// Colocamos la bateria
MiBateria.x=random.Next(0,639);
MiBateria.y=random.Next(0,479);
MiBateria.activo=true;

maquina.Inicializa(ref ListaObjetos,MiBateria);

}

```

En el constructor de la forma, tenemos un objeto para poder generar números aleatorios. Se recorre el arreglo de los objetos colocándolos en posiciones al azar. Todos los objetos se inician como activos. A la batería, también se le da una posición al azar. Ya con estos elementos, invocamos al método **Inicializa()** de la máquina y pasamos los objetos correspondientes.

```
private void mnuSalir_Click(object sender, System.EventArgs e)
```



USO DE SWITCH

No es estrictamente necesario hacer uso de **switch**, se puede sustituir por una escalera bien planeada de **if-else**. En algunos casos, podemos obtener mejores resultados al ordenar las condiciones, sobre todo si sabemos cuáles son más comunes o tienen mayor posibilidad de ocurrir. Si bien no es necesario, este tipo de optimización es recomendable para sistemas en tiempo real.

```
{  
    // Cerramos la ventana y finalizamos la aplicacion  
    this.Close();  
}
```

En el handler del menú para salir de la aplicación, se cierra la ventana y se finaliza la aplicación.

```
private void mnuInicio_Click(object sender, System.EventArgs e)  
{  
    timer1.Enabled=true;  
}
```

Cuando se selecciona el menú de **Inicio**, activamos el **Timer** y, con esto, la máquina de estados empieza a funcionar dentro del ciclo creado por el **Timer**.

```
private void mnuParo_Click(object sender, System.EventArgs e)  
{  
    timer1.Enabled=false;  
}
```

En el menú de **Paro**, el **Timer** es detenido. Cuando éste se detiene, el evento ya no se genera. Esto nos puede servir para colocar en pausa a la máquina de estados finitos.

```
private void timer1_Tick(object sender, System.EventArgs e)  
{  
    // Esta funcion es el handler del timer  
    // Aqui tendremos la logica para actualizar nuestra  
    // maquina de estados  
  
    // Actualizamos la maquina  
    maquina.Control();  
  
    // Mandamos a redibujar la pantalla  
    this.Invalidate();  
}
```

En el handler del **Timer** se invoca a la función **Control()** de la máquina de estados finitos y, después que ésta llevó a cabo su proceso, se vuelve a dibujar la ventana para mostrar los cambios efectuados. Hay que recordar que esta función se invoca cada 5 milisegundos mientras el **Timer** esté activo.

```

private void Form1_Paint(object sender,
    System.Windows.Forms.PaintEventArgs e)
{
    // Creamos la fuente y la brocha para el texto
    Font fuente = new Font("Arial", 16);
    SolidBrush brocha = new SolidBrush(Color.Black);

    // Dibujamos el robot
    if(maquina.EstadoM==(int)CMaquina.estados.MUERTO)

        e.Graphics.DrawRectangle(Pens.Black,maquina.CoordX-4,maquina.CoordY-4,8,8);
    else

        e.Graphics.DrawRectangle(Pens.Green,maquina.CoordX-4,maquina.CoordY-4,8,8);

    // Dibujamos los objetos
    for(int n=0;n<10;n++)
        if(ListaObjetos[n].activo==true)

            e.Graphics.DrawRectangle(Pens.Indigo,ListaObjetos[n].x-4,ListaObjetos[n].y-4,8,8);

    // Dibujamos la bateria
    e.Graphics.DrawRectangle(Pens.IndianRed,MiBateria.x-
        4,MiBateria.y-4,8,8);

    // Indicamos el estado en que se encuentra la maquina
    e.Graphics.DrawString("Estado ->
        "+maquina.EstadoM.ToString(), fuente, brocha, 10, 10);

}

```

Por último, tenemos el handler de **Paint()**. Cuando se necesita dibujar la ventana, este handler es invocado. Primero creamos una fuente y una brocha que se usarán en el despliegue de un mensaje de texto. Luego dibujamos el robot. Si el robot está muerto, se dibuja un rectángulo negro; en caso contrario, un rectángulo verde.

Tenemos un ciclo para dibujar los objetos. Solamente dibujamos los objetos que se encuentren activos. El objeto se dibuja con un cuadro de color índigo. La batería se dibuja como un cuadro rojo.

Al final, simplemente, se agrega un mensaje en la ventana, indicando el número de estado en el que se encuentra la máquina.

Ahora podemos compilar y ejecutar la aplicación. Aparecerá la ventana mostrando al robot, los objetos y la batería. Seleccionamos el menú de **Inicio**, y la máquina empieza a funcionar. Podemos ver al robot recogiendo objetos y, si disminuye su energía demasiado, dirigiéndose hacia la batería

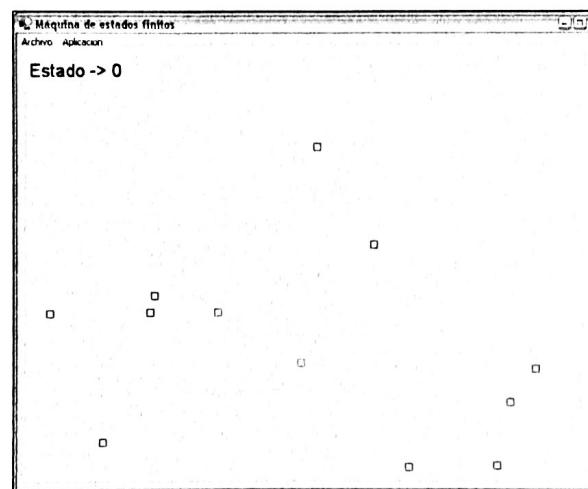


Figura 10. Aquí vemos a nuestra máquina de estados finitos ejecutándose, el robot recoge los objetos y recarga su batería.

Con esto, ya tenemos una primera aplicación que, al ejecutarla, parece mostrar un comportamiento ligeramente inteligente.



RESUMEN

Las máquinas de estados finitos nos permiten implementar sistemas de control de manera sencilla y eficiente. Tienen una cantidad definida de posibles estados. La máquina permanece en el mismo estado hasta que una condición de transición se cumpla. Las transiciones nos indican qué estado puede cambiar a cuál estado.



ACTIVIDADES

TEST DE AUTOEVALUACIÓN

- 1** ¿Qué es una máquina de estados finitos?
- 2** ¿Qué indica la transición?
- 3** ¿Qué es la condición de transición?
- 4** ¿De qué manera se puede diagramar una máquina de estados finitos?
- 5** ¿Cómo se indica el estado inicial de la máquina de estados finitos?
- 6** ¿Para qué sirve una tabla de estados?
- 7** ¿Por qué usamos un Timer?
- 8** ¿De qué otra forma se puede implementar la máquina de estados finitos?
- 9** ¿Cómo puede modificarse el código para que el robot recoja el objeto más cercano en lugar del último de la fila?
- 10** ¿Dónde se pueden aplicar las máquinas de estados finitos?
- 11** Modelar una impresora como máquina de estados finitos.
- 12** Modelar un televisor como máquina de estados finitos.