TUTORIAL de PROGRAMACIÓN del NXT con MINDSTORMS NXT 2.0 (NXT-G) y NXC (Not eXactly C)

Gustavo Adolfo Raya Casero.

Reglas léxicas básicas.

- NXC detecta minúsculas y mayúsculas.
  - Ej. if ≠ If ≠ iF ≠ IF

- Para hacer comentarios se utiliza la siguiente notación:
  - // Cuando el comentario ocupa 1 línea.
  - /* Cuando el comentario ocupa más de una línea. En este caso hay que cerrar el comentario. */

Reglas léxicas básicas.

- Los espacios son utilizados para separar los símbolos del código utilizados en la programación.
  - Ej.:
    - int x;
    - x = 1;

Reglas léxicas básicas.

- Los identificadores son utilizados para nombrar TAREAS, FUNCIONES, SUBRUTINAS, VARIABLES y CONSTANTES. El primer carácter puede ser el símbolo '_', mayúsculas y minúsculas. El resto de caracteres pueden ser también '_', mayúsculas, minúsculas y números. Está PROHIBIDO utilizar los identificadores reservados por el entorno de programación. Ej.: "if", "for", "else", "do", "while", "int",…

Estructura de programas.

- Un programa de NXC está compuesto de bloques de códigos y variables.
- Hay dos distintos tipos de bloques de códigos:
  - Tareas.
  - Funciones.

Estructura de programas.
Tareas.

- El ROBOT NXT es multitarea. El número máximo de tareas es 256.
- Todos los programas deben empezar con la tarea principal:

```
task main(){}
```

- Otras tareas se definen así:

```
task nombre_tarea()
    {
      //código de la tarea;
    }
```

Estructura de programas.
Tareas.

- StartTask(task); da comienzo a una tarea específica.
- StopTask(task); detiene una tarea específica.
- StopAllTasks(); detiene TODAS las tareas. Detendrá el programa en ejecución completamente.
- Una tarea se detiene sí alcanza el final de la misma.
- ExitTo(nextTask); detiene la tarea actual y empieza a ejecutar la tarea especificada.

Estructura de programas.
Funciones.

- Cuando queremos que un grupo de instrucciones realicen una tarea determinada, las podemos agrupar y darles un nombre asociado a la operación que realizan. Ese es el concepto de FUNCIÓN.
- Las funciones soportan argumentos (entradas) y pueden retornar valores (salidas).

Estructura de programas. Funciones.

- Declaración de funciones:

```
tipo_valor_retorno nombre_función (lista_argumentos_entrada)
{
    //cuerpo de la función;
}
```

- Si queremos que la función no retorne ningún valor la declararemos como sigue:

```
void nombre (lista_argumentos_entrada)
{
    //cuerpo de la función;
}
```

- El número máximo de funciones y tareas es 256.

Estructura de programas. Funciones.

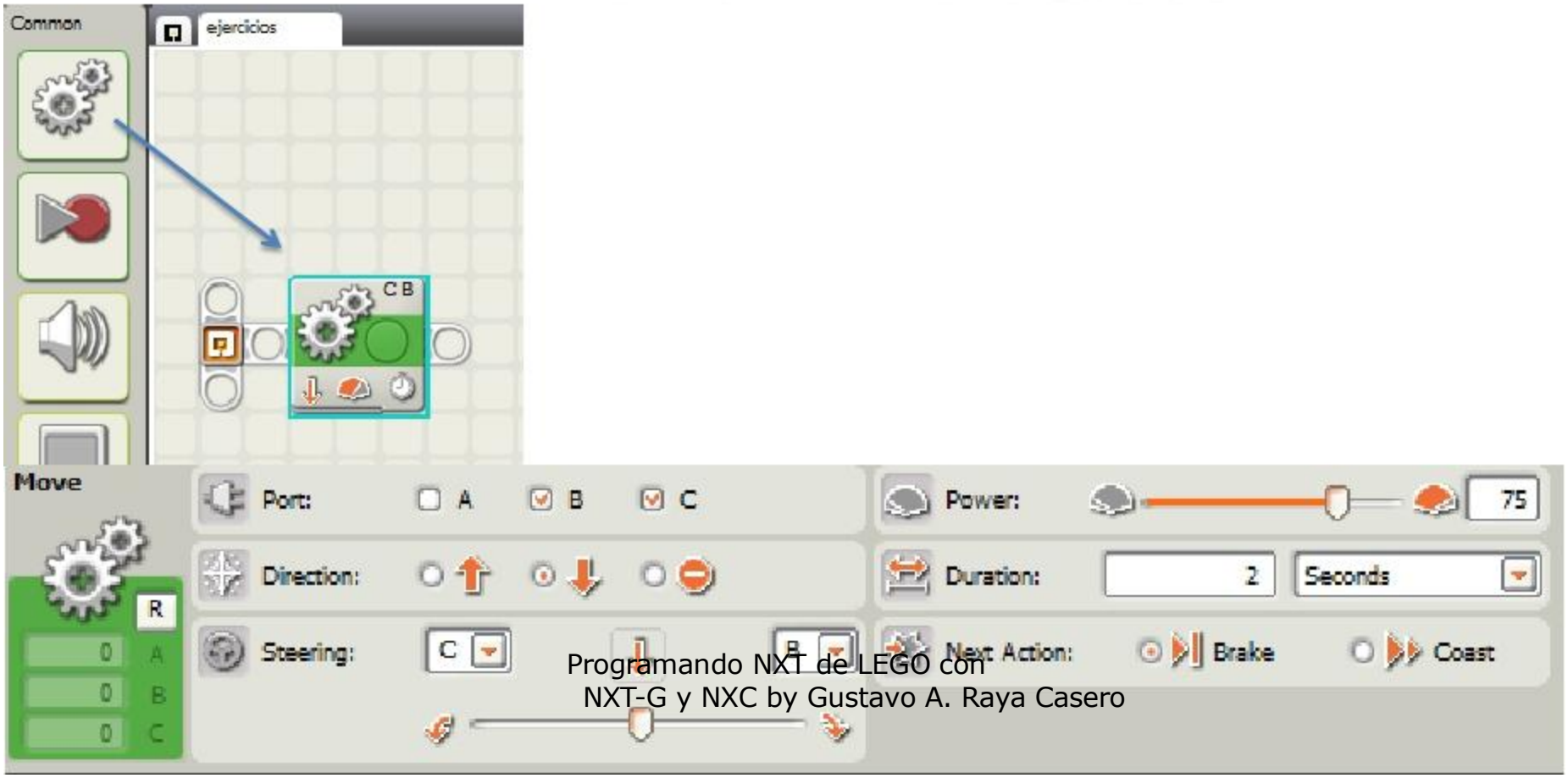
- La lista de argumentos de entrada puede estar vacía, o puede contener 1 o más argumentos de entrada definidos.
- Ejemplo

```
void prueba (int &x, int y, const int z)
{
        x = (y*2)+z;
}
task main (){
        int a=1,b=2;
        prueba(a,b,3);//Ahora las variables contienen a=7, b=2.
}
```

Ejercicio 1.

Encender los motores conectados a los puertos B y C durante 2 segundos en reversa (marcha atrás); después, detener ambos motores.

Ejercicio 1. Solución LEGO MINDSTORMS NXT.



Ejercicio 1. Solución NXC.

```
task main(){
    OnRev(OUT_B, 75);
    OnRev(OUT_C, 75);
    Wait(2000);
    Off(OUT_B);
    Off(OUT_C);
}
```

Comienza la tarea principal.

Ponemos en funcionamiento "marcha atrás" los motores conectados a las salidas B y C. La potencia de los motores es de 75(MÁXIMO=100). Observar que OnRev() es una FUNCIÓN con dos parámetros de entrada.

Los motores estarán en marcha durante 2 sgs. Advertir que el tiempo se expresa en msgs. Wait() es una función con un parámetro de entrada.

Paramos los motores conectados a las salidas B y C. Off() es una función con un parámetro de entrada.

Finaliza la tarea principal.

Ejercicio 1. Solución NXC.

```
task main(){
  OnRev(OUT_BC, 75);
  /* Pone en marcha el motor conectado a las salidas B y C con
    una potencia de 75.*/

  Wait(2000);
  /* Los motores estarán en marcha durante DOS SEGUNDOS. El
    tiempo está en milisegundos -> 2000 msg = 2 sg */

  Off(OUT_BC); // Paramos los motores B y C.
}
```

Ejercicio 1. Solución NXC.

```
/* Utilizamos la definición de MACROS = CONSTANTES, para hacer que el código sea más fácil de entender. De
      esta manera, para programas largos, sólo tendríamos que cambiar el valor de la MACRO una vez para todo
      el código.*/

#define MOTOR_TIME 2000
#define MOTOR_POWER 75

task main(){
  OnRev(OUT_BC, MOTOR_POWER);
  /* Pone en marcha el motor conectado a las salidas B y C con una potencia de 75. Vemos que ahora se están
      usando las MACROS definidas anteriormente.*/

  Wait(MOTOR_TIME);
  /* Los motores estarán en marcha durante DOS SEGUNDOS. El tiempo está en milisegundos -> 2000 msg = 2
      sg. Vemos que ahora se están usando las MACROS definidas anteriormente.*/

  Off(OUT_BC); // Paramos los motores B y C.
}
```

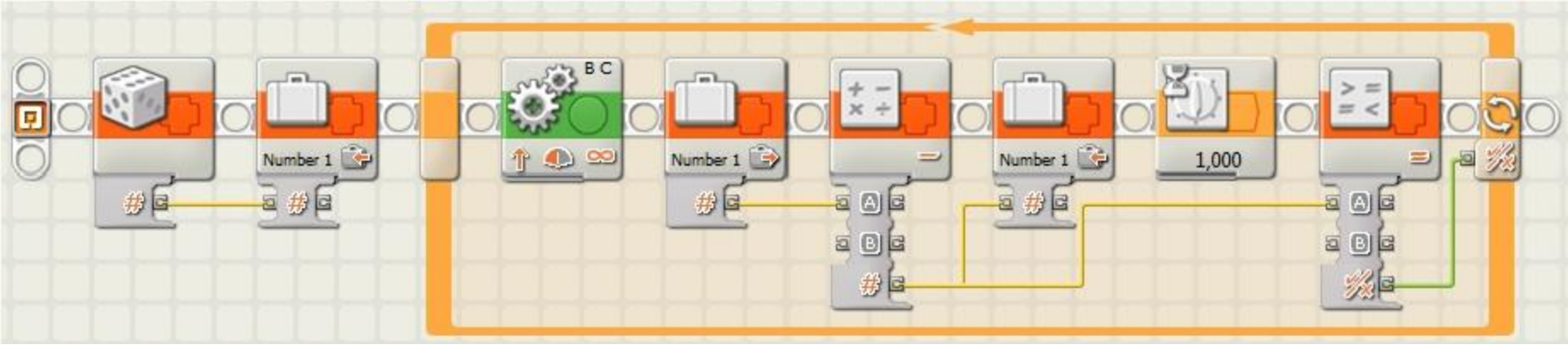
Ejercicio 2.

Encienda el motor B y C para que avancen una cantidad aleatoria de tiempo (máximo 10 segs.), después apáguelo.

Ejercicio 2. Solución NXC.

```
#define MOTOR_POWER 100

task main() {
  OnFwd (OUT_BC, MOTOR_POWER);
  /*Pone en marcha el motor conectado a la salida B, hacia adelante, con su
    potencia máxima = 100. */

  Wait(Random(10000));
  /*Estará en marcha durante un tiempo aleatorio entre 0 y 9,99.. sgs.*/

  Off(OUT_BC);
  //Para el motor conectado a la salida B y finaliza la tarea.
}
```

Ejercicio 2.
Solución Mindstorms NXT.



Programando NXT de LEGO con
NXT-G y NXC by Gustavo A. Raya Casero

Ejercicio 3.

- Encienda el motor B con una potencia máxima de 100 para que avance durante 2 segundos, apáguelo, y emita un sonido durante 1 segundo.

- Encienda el motor C en reversa con una potencia de 50 durante 2 segundos, apáguelo, y emita DOS sonidos distintos durante 1 segundo cada uno.

- Termine el programa y la tarea.

Ejercicio 3. Solución Mindstorms NXT.

Ejercicio 3. Solución NXC.

```
#define MOTOR_TIME 2000
#define MUSIC_TIME 1000

task main() {
    OnFwd(OUT_B,100); //Enciende el motor B con una potencia máxima de 100.
    Wait(MOTOR_TIME); //Espera para que avance durante 2 segundos.
    Off(OUT_B);       //Apaga el motor B.
    PlayTone(440, MUSIC_TIME);//Emite un tono de 440Hz durante 1sg.
    Wait(MUSIC_TIME); //Espera 1sg. que termine la función de sonido.

    OnRev(OUT_C,50); //Enciende el motor C con una potencia de 50.
    Wait(MOTOR_TIME); //Espera para que retroceda durante 2 segundos.
    Off(OUT_C);      //Apaga el motor C.
    PlayTone(440, MUSIC_TIME); //Emite un tono de 440Hz durante 1sg.
    Wait(MUSIC_TIME); //Espera 1sg. que termine la función de sonido.
    PlayTone(880, MUSIC_TIME); //Emite un tono de 880Hz durante 1sg.
    Wait(MUSIC_TIME); //Espera 1sg. que termine la función de sonido.
}
```

BUCLE. repeat (condición) { …}

```
repeat (condición)
{
   …  /*Se repetirá el bucle mientras
         la condición no sea falsa.*/
}
```

Ejercicio 4.1

Ejecute un tono de 440Hz. durante 1sg. un número aleatorio de veces entre 0 y 10 veces.

Casero

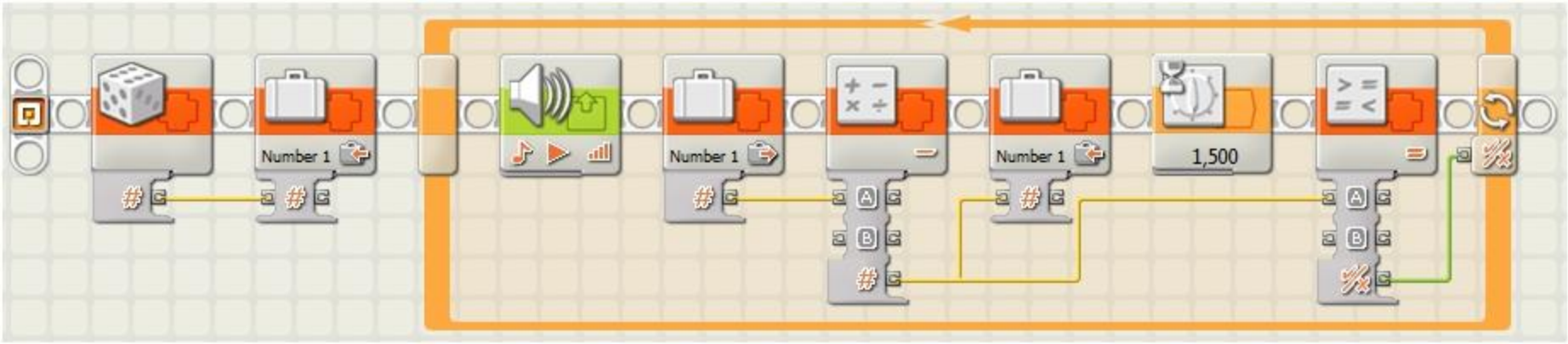
Ejercicio 4.1 Solución Lego Mindstorms NXT.

Esta solución no implementa un número aleatorio de veces con el fin de simplificarla. Lo que tenemos es un bucle(loop) repeat que se ejecuta tres veces.

Ejercicio 4.1. Solución NXC.

```
task main()
{
repeat ( Random(10) ) //Entre 0 y 10 veces.
{
    PlayTone(440, 1000);/*Emite un tono de 440Hz
                                    durante 1sg.*/
    Wait(1500);/*Espera 1sg. que termine la
                        función de sonido.*/
}
}
```

Ejercicio 4.1.
Solución Mindstorms NXT.

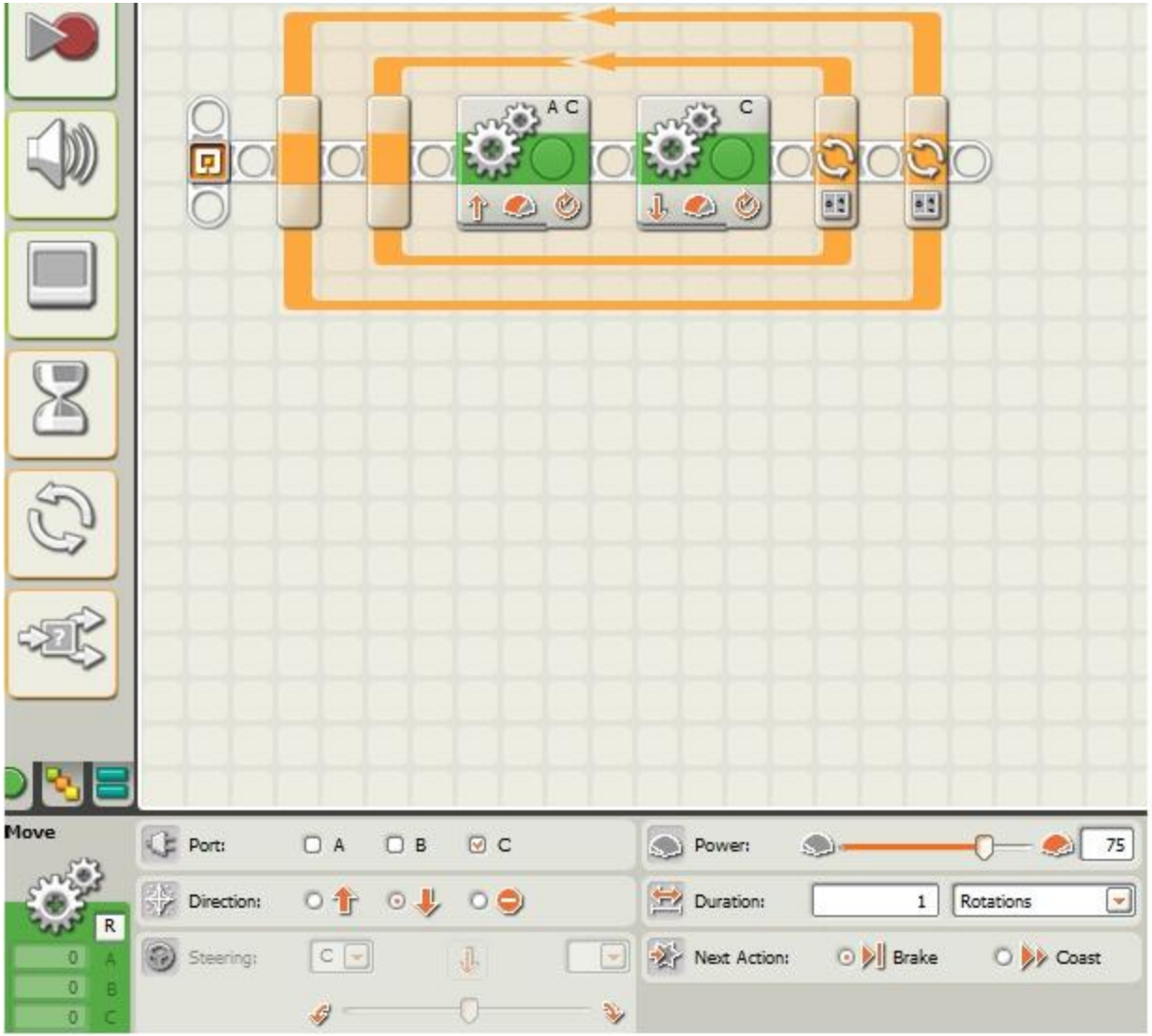


Esta es la solución implementada con un número aleatorio (random).

Ejercicio 4.2

Realizar un programa que haga girar al ROBOT de manera que complete tantos cuadrados como queramos.
Para ello:

1. Debe caminar hacia adelante durante un tiempo.
2. Girar hacia el lado (izquierda o derecha) que queramos para formar el cuadrado. El tiempo de este giro determinará los grados del giro.
3. Repetir los pasos 1 y 2 cuatro veces para completar un cuadrado.
4. Repetir los pasos 1,2, y 3 tantas veces como queramos para hacer el número de cuadrados que queramos.
5. Apagar los motores.



Ejercicio 4.2 Solución Lego Mindstorms NXT..

Ejercicio 4.2. Solución NXC.

```
/*  10 CUADRADOS
Con este programa el robot dibujará el número de cuadrados indicados en la
macro SQUARE_NUMBERS.
*/
#define MOVE_TIME    1300     // Tiempo caminando en línea recta.
#define TURN_TIME    315      // Tiempo de giro. 90 grados.
#define SQUARE_NUMBERS 10     // Número de cuadrados que el NXT trazará.

task main()
{
  repeat(SQUARE_NUMBERS)     // Dibuja SQUARE_NUMBERS cuadrados.
  {
    repeat(4)               // Completa 1 cuadrado.
    {
      OnFwd(OUT_AC, 75);
      Wait(MOVE_TIME);
      OnRev(OUT_C, 75);
      Wait(TURN_TIME);
    }
  }
  Off(OUT_AC);              // Apagamos los motores.
}
```

Ejercicio 4.3

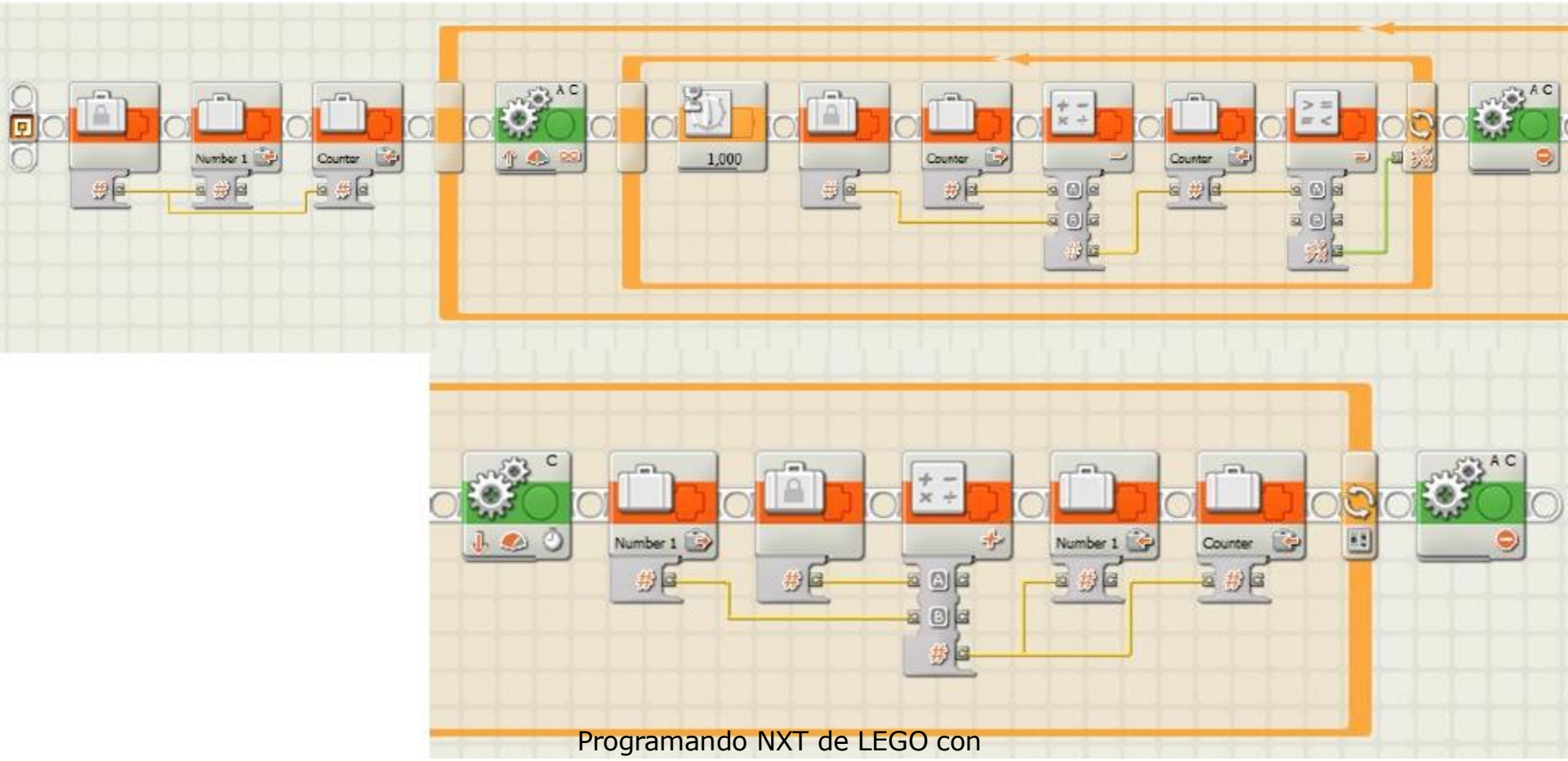
Realizar un programa que haga girar al ROBOT en ESPIRAL.
Para ello:

1. Debe caminar hacia adelante durante un tiempo que se irá INCREMENTANDO cada vez, por lo tanto este valor tiene que ser VARIABLE.
2. Girar hacia el lado (izquierda o derecha) que queramos para formar la espiral. El tiempo de este giro determinará los grados del giro.
3. Incrementar el tiempo en el que los motores caminan hacia adelante con el fin de hacer que el robot dibuje una espiral.
4. Repetir los pasos 1,2, y 3 tantas veces como grande en diámetro queramos que sea la espiral.
5. Apagar los motores.

Ejercicio 4.3. Solución NXC.

```
#define TURN_TIME     360
#define REPEATING_NUMBER 10

int move_time;// Definimos una variable entera que indica el tiempo en movimiento.

task main()
{
  move_time = 200;           // Inicializamos la variable.
  repeat(REPEATING_NUMBER)
  {
    OnFwd(OUT_AC, 75);
    Wait(move_time);          // Tiempo en movimiento hacia adelante.
    OnRev(OUT_C, 75);
    Wait(TURN_TIME);
    move_time += 200;         // Incrementamos la variable.

    /* Como move_time será cada vez mayor, esto hará que el robot vaya haciendo
    movimientos en espiral. Esto lo conseguimos haciendo que el tiempo en que el
    NXT camina hacia delante sea cada vez mayor con la instrucción
    move_time += 200;
    */
  }
  Off(OUT_AC);              //Apagamos los motores.
}
```

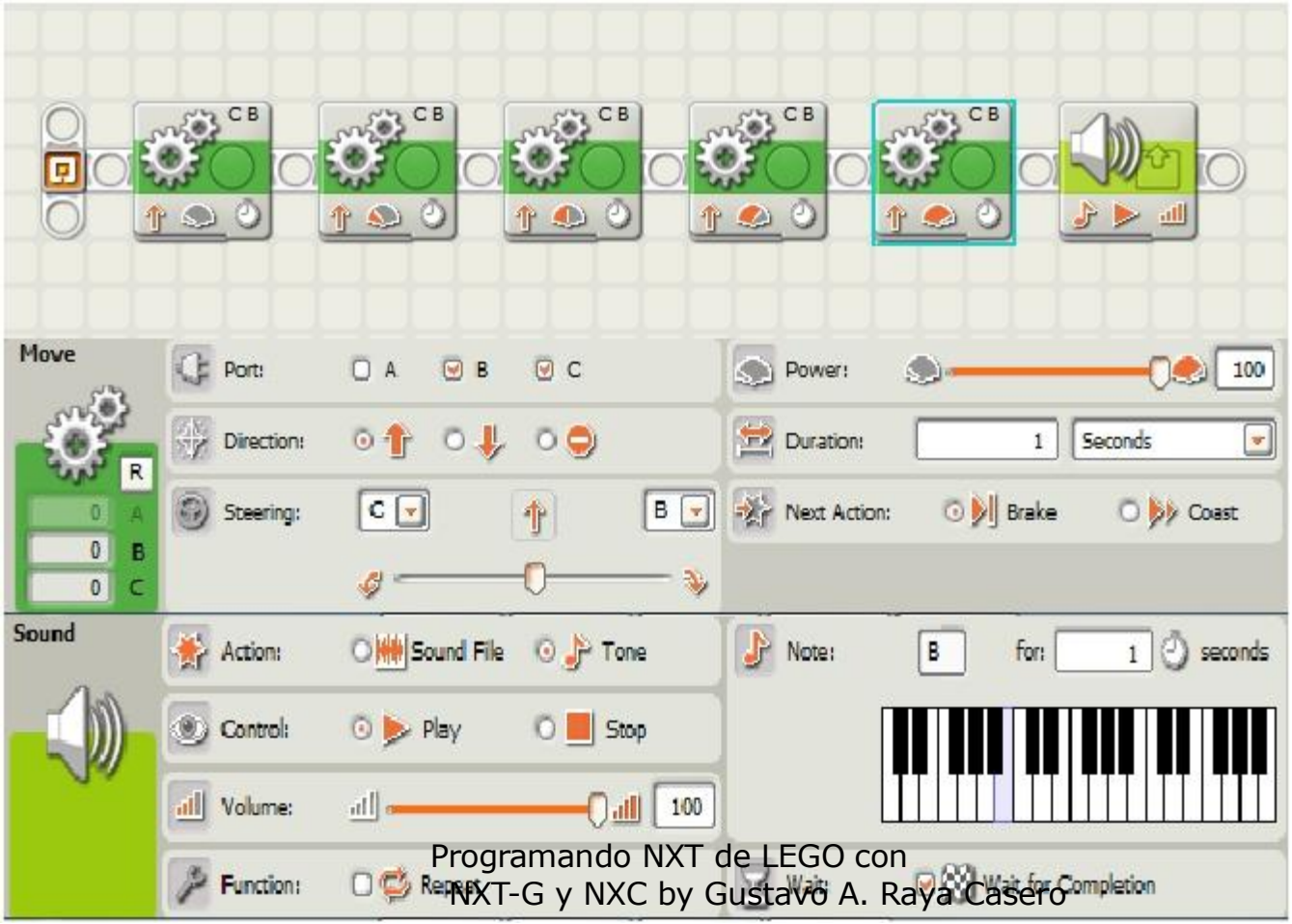
Ejercicio 4.3. Solución Lego Mindstorms NXT.



Ejercicio 5.

- Encienda los motores B y C para que avancen a nivel de potencia 20 durante un segundo, luego a nivel 40 durante un segundo, etc…, hasta llegar al nivel 100 durante un segundo.
- Posteriormente apague los motores y emita un tono de 220Hz. durante 1 sg.
- Finalice el programa.

Ejercicio 5. Solución Lego Mindstorms NXT.



Programando NXT de LEGO con NXT-G y NXC by Gustavo A. Raya Casero

Ejercicio 5. Solución NXC.

```
#define MOTOR_TIME 1000
#define MUSIC_TIME 1000

task main() {
    OnFwd(OUT_BC,20); //Motores B y C hacia adelante con una potencia de 20.
    Wait(MOTOR_TIME); //Espera a que avancen durante 1 segundo.
    OnFwd(OUT_BC,40); //Motores B y C hacia adelante con una potencia de 40.
    Wait(MOTOR_TIME); //Espera a que avancen durante 1 segundo.
    OnFwd(OUT_BC,60); //Motores B y C hacia adelante con una potencia de 60.
    Wait(MOTOR_TIME); //Espera a que avancen durante 1 segundo.
    OnFwd(OUT_BC,80); //Motores B y C hacia adelante con una potencia de 80.
    Wait(MOTOR_TIME); //Espera a que avancen durante 1 segundo.
    OnFwd(OUT_BC,100); //Motores B y C hacia adelante con una potencia de 100.
    Wait(MOTOR_TIME); //Espera a que avancen durante 1 segundo.
    Off(OUT_BC);
    PlayTone(220, MUSIC_TIME);//Emite un tono de 220Hz. durante 1sg
    Wait(MUSIC_TIME);//Espera 1sg. que termine la función de sonido.
}
```

BUCLE. while (condición) { …}

```
while (condición)
{
   …  /*Se repetirá el bucle mientras
         la condición sea cierta =          true.*/
}
```

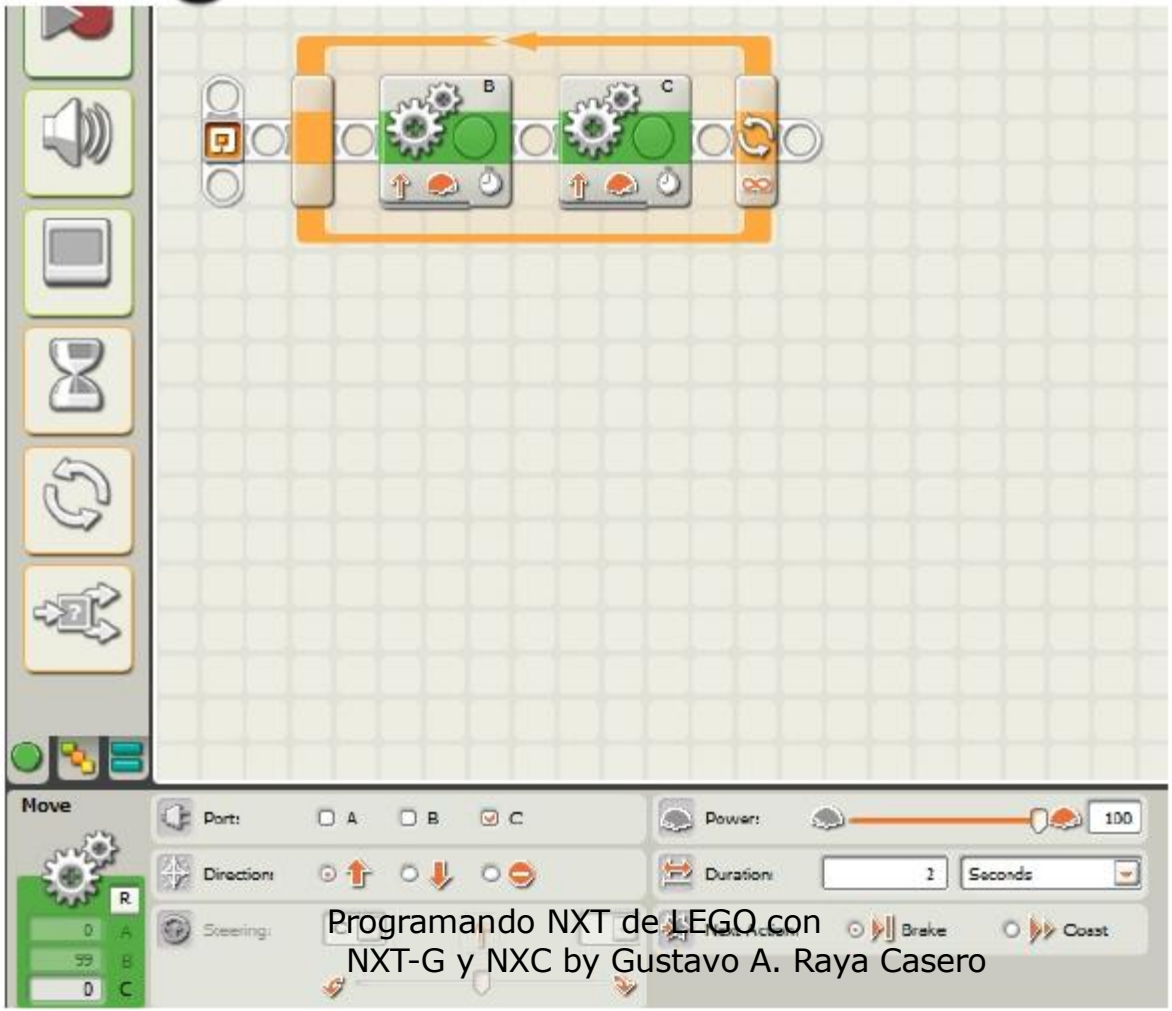
Ejercicio 6.

Encienda el motor B para que avance durante 2 segundos, y posteriormente detenga el motor B.

Encienda el motor C para que avance durante 2 segundos, después apague el motor C.

Repita el proceso de forma indefinida = BUCLE INFINITO.

Ejercicio 6. Solución Lego Mindstorms NXT.



Ejercicio 6. Solución NXC.

```
#define MOTOR_TIME 2000
task main() {
    while (true){ /*Bucle INFINITO porque la condición
                                    siempre es cierta.*/
        OnFwd(OUT_B, 100);/*Enciende el motor B
                                                a potencia 100.*/
        Wait(MOTOR_TIME);//Durante 2 sgs.
        Off(OUT_B); //Apaga el motor B.
        OnFwd(OUT_C, 100);/*Enciende el motor C
                                                a potencia 100.*/
        Wait(MOTOR_TIME);//Durante 2 sgs.
        Off(OUT_C);//Apaga el motor C.
    }
}
```

Ejercicio 7.

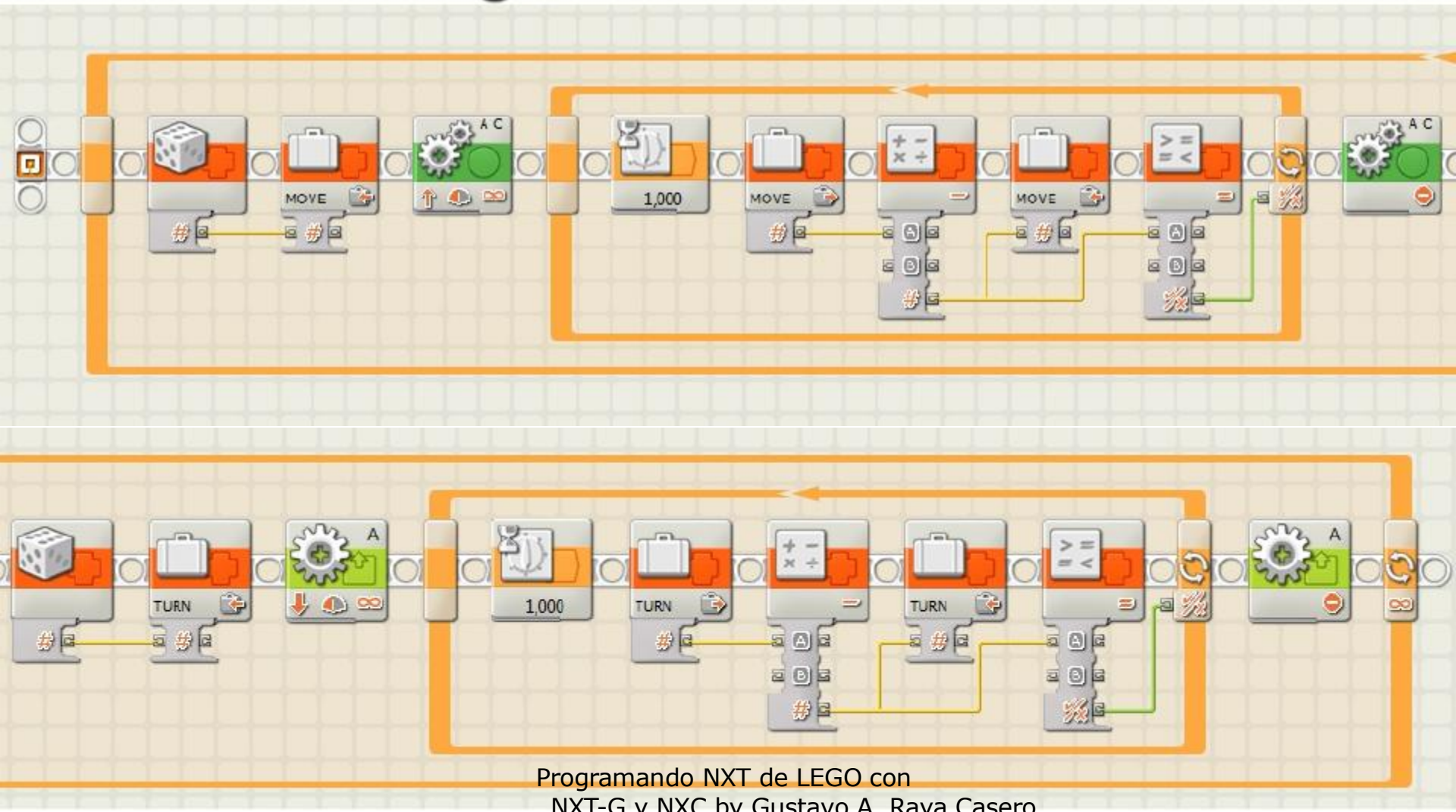
Implementar un programa que haga que el robot se mueva de forma aleatoria sin parar (infinitamente). Para ello, dentro de un bucle WHILE, haremos lo siguiente:

1. Generar un número aleatorio entre 0 y 0,6 segundos, que guardaremos en una variable entera, y que será el tiempo que el robot se estará moviendo hacia adelante.
2. Generar un número aleatorio entre 0 y 0,4 segundos, que guardaremos en otra variable entera distinta a la anterior, y que será el tiempo que el robot estará girando hacia un lado.
3. Hacer que el robot camine hacia adelante durante el tiempo de la primera variable.
4. Hacer que el robot GIRE hacia un lado durante el tiempo de la segunda variable.
5. Repetir el proceso infinitamente.

Ejercicio 7. Solución NXC.

```
int move_time, turn_time;

task main()
{
  while(true)
  {
    move_time = Random(600);
    turn_time = Random(400);
    OnFwd(OUT_AC, 75);
    Wait(move_time);
    OnRev(OUT_A, 75);
    Wait(turn_time);
  }
}
```

Ejercicio 7. Solución
Lego Mindstorms NXT.



Estructura de control IF

```
if (condition==true)
{
    statements;
}
else
{
    statements;
}
```

Ejercicio 8.

Realizar un programa que haga que el NXT haga giros a derecha y a izquierda de forma aleatoria, Random().

El NXT debe caminar hacia delante durante un tiempo constante y mayor que el tiempo de giro a derecha e izquierda.

El tiempo de giro a derecha debe ser constante e igual que el tiempo de giro a izquierda.

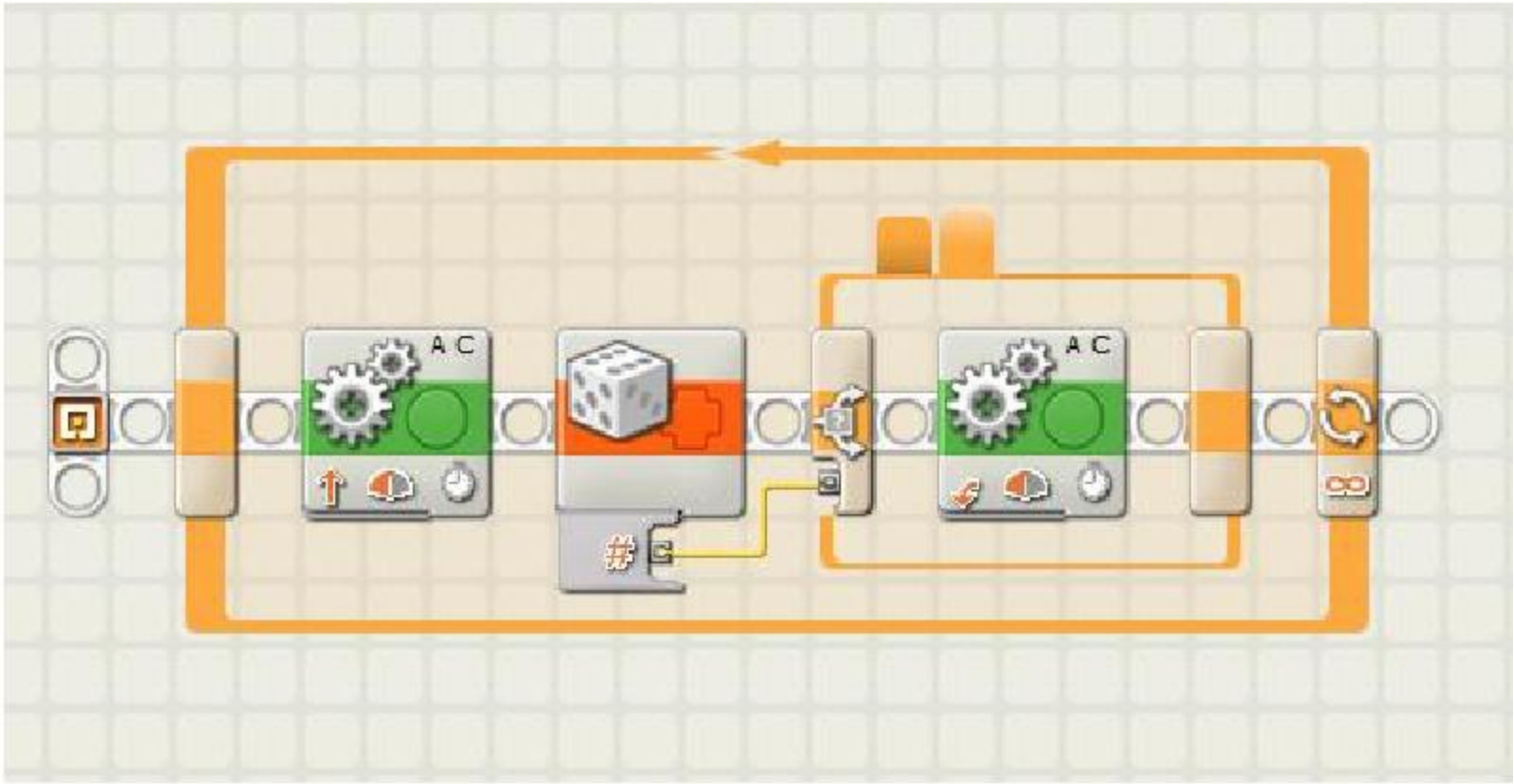
Debe hacer estas acciones un número infinitos de veces.

Ejercicio 8. Solución NXC.

```
#define MOVE_TIME    500
#define TURN_TIME    360

task main()
{
  while(true)
  {
    OnFwd(OUT_AC, 75);
    Wait(MOVE_TIME);
    if (Random() > 0)
      OnRev(OUT_C, 75); // Si el valor generado es mayor que 0.
    else
      OnRev(OUT_A, 75); /* Si el valor generado es menor o igual
                           que 0.*/
    Wait(TURN_TIME);
  }
}
```

EJEMPLO

Ejercicio 8. Solución
Lego Mindstorms NXT.



Estructura de control do {statements;} while();

```
do
{
statements;
}
while (condition);
```

Ejercicio 9.

Realizar un programa que haga que el NXT camine hacia delante durante un tiempo aleatorio, Random(), y que también haga giros con el motor C durante OTRO tiempo aleatorio, Random().

El NXT debe caminar hacia delante durante un tiempo NO constante y aleatorio, Random(), cada vez.

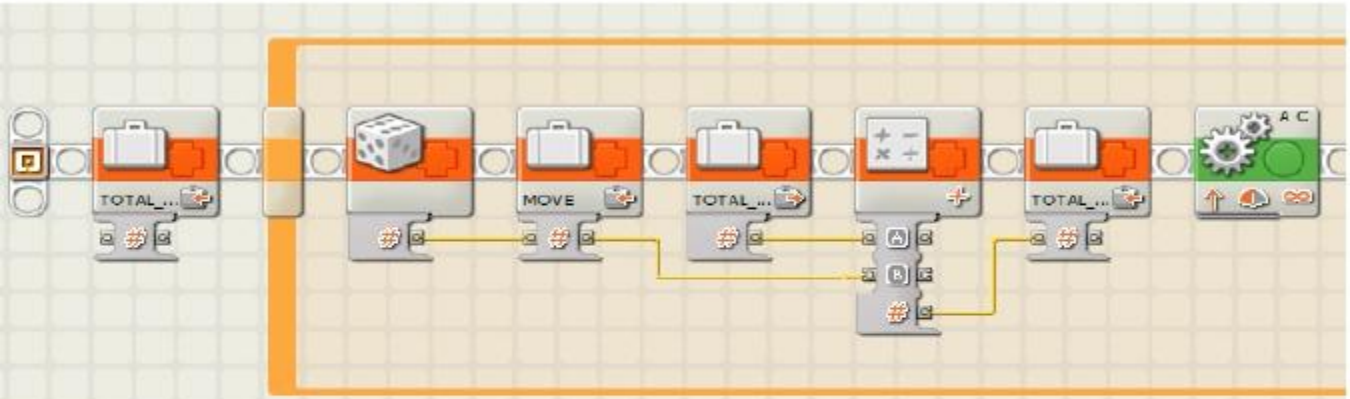
De la misma manera, el NXT debe hacer giros con el motor C durante OTRO tiempo aleatorio, Random(), cada vez.

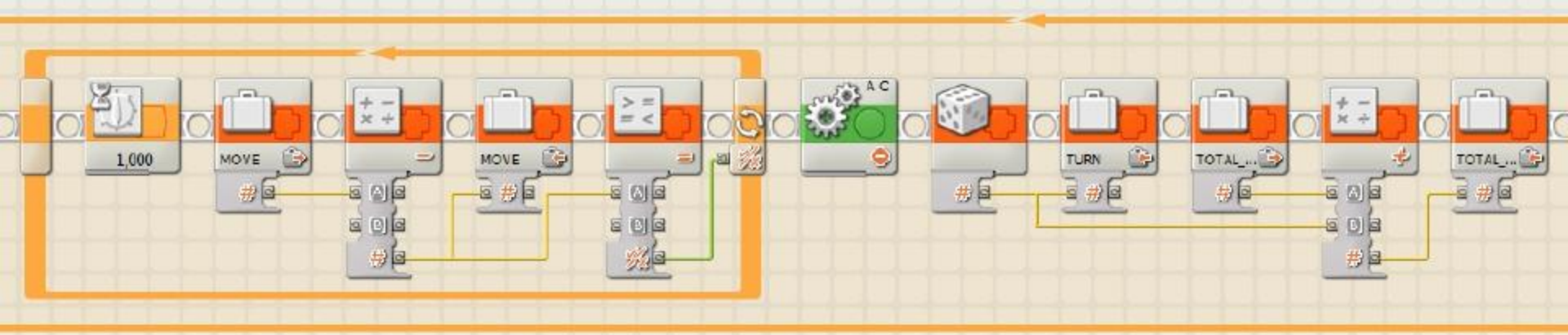
Debe hacer estas acciones un TIEMPO TOTAL INFERIOR a 20 segudos sumando todos
los movimientos.

Ejercicio 9. Solución NXC.

```
int move_time, turn_time, total_time;

task main()
{
  total_time = 0; /*total_time se inicializa. Aquí guardaremos el tiempo total
                                que el NXT ha estado en movimiento.*/
  do
  {
    move_time = Random(1000);  //Genera un número aleatorio entre 0 y 1000. Lo guardamos en
                                          move_time.
    turn_time = Random(1000);  //Genera un número aleatorio entre 0 y 1000. Lo guardamos en
                                          turn_time.
    OnFwd(OUT_AC, 75);           //Camina hacia adelante...
    Wait(move_time);            //...durante el tiempo indicado en move_time.
    OnRev(OUT_C, 75);            //Hace un giro a contra marcha con el motor C...
    Wait(turn_time);            //...durante el tiempo indicado en turn_time.
    total_time += move_time;      //Incrementa la variable total_time que almacena el tiempo...
    total_time += turn_time;      //...total que el NXT ha estado en movimiento.
  }
  while (total_time < 20000);  //Estará haciendo estos movimientos mientras total_time sea menor
                                          que 20sgs.
  Off(OUT_AC);                 //Cuando total_time>=20sgs saldrá del bucle y parará los motores.
}
```

Ejercicio 8. Solución Lego Mindstorms NXT.





Sensor mode and type.

El comado SetSensor() realiza dos operaciones:
Define el tipo de sensor,
Define el modo en el que el sensor opera.

A continuación vemos los modos de operación más comunes:

SENSOR_TOUCH, sensor de choque,

SENSOR_LIGHT, sensor de luz,

SENSOR_SOUND, sensor de sonido,

SENSOR_LOWSPEED, sensor ultrasónico.

Los sensores de luz y sonido ofrecen una lectura que oscila de 0 a 100.

Ejercicio 10_1.

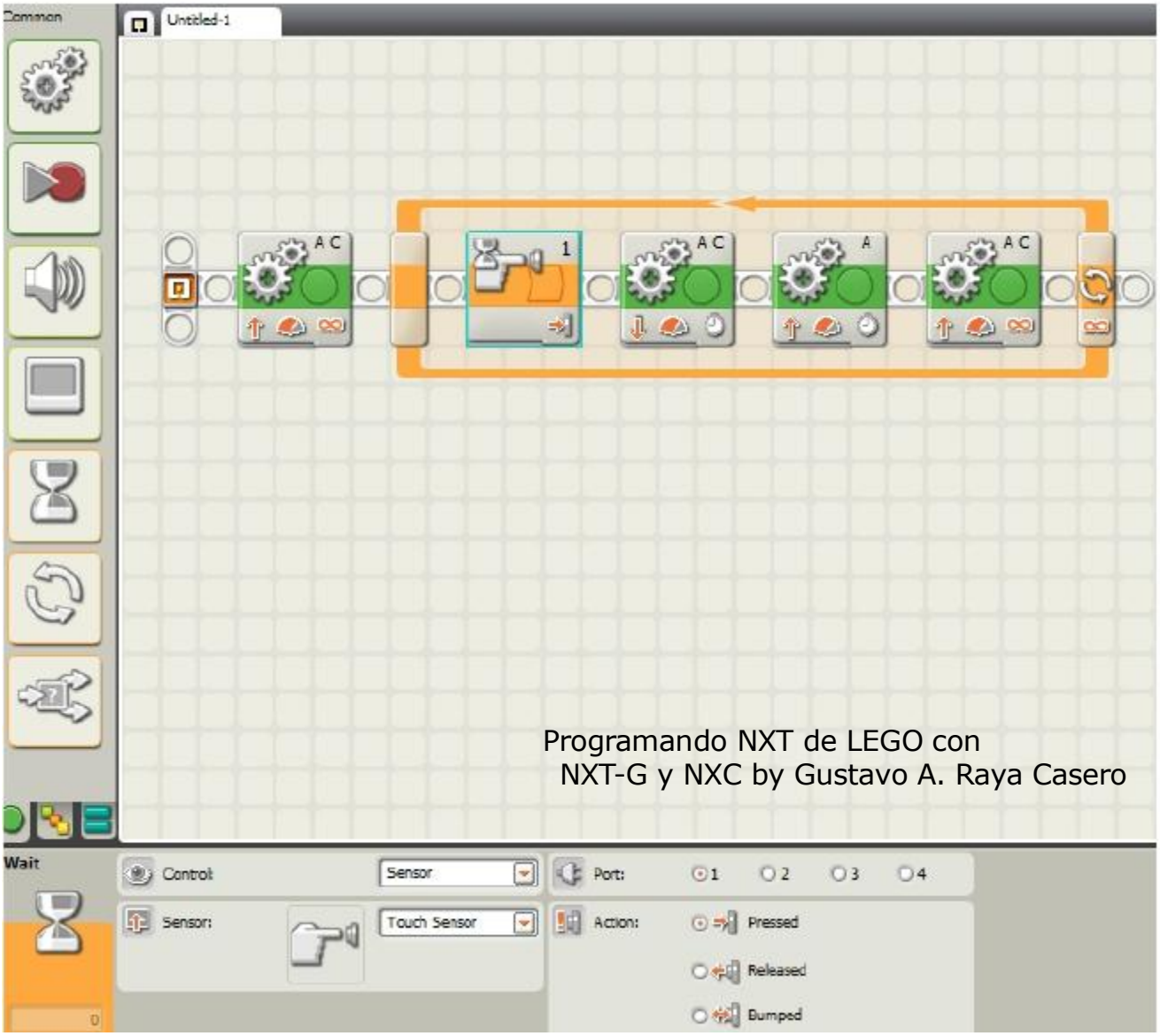
Implementar un programa que haga hacer al NXT las siguientes acciones:
Definir la entrada 1 como entrada de sensor de choque y poner en robot a caminar
hacia delante hasta que detecte un obstáculo con el sensor definido.
SI detecta un obstáculo con el sensor de choque conectado en la entrada 1 entonces:

- dará marcha atrás durante 300 msgs.
- a continuación girará durante 300msgs con el motor A girando hacia delante y el C hacia atrás.
- volverá a caminar hacia delante de forma indefinida hasta que detecte otro obstáculo con el sensor de choque conectado a la entrada 1. En tal caso, procederá de la misma manera.

Ejercicio 10_1. Sensor de choque.

```
task main()
{
  SetSensor(IN_1,SENSOR_TOUCH);
  OnFwd(OUT_AC, 75);
  while (true)
  {
    if (SENSOR_1 == 1)
    {OnRev(OUT_AC, 75); Wait(300);
      OnFwd(OUT_A, 75); Wait(300);
      OnFwd(OUT_AC, 75);
    }
  }
}
```

Ejercicio 10_1. Solución Lego Mindstorms NXT.



Programando NXT de LEGO con
NXT-G y NXC by Gustavo A. Raya Casero

Ejercicio 10_2.

Implementar un programa que ejecute las siguientes acciones:

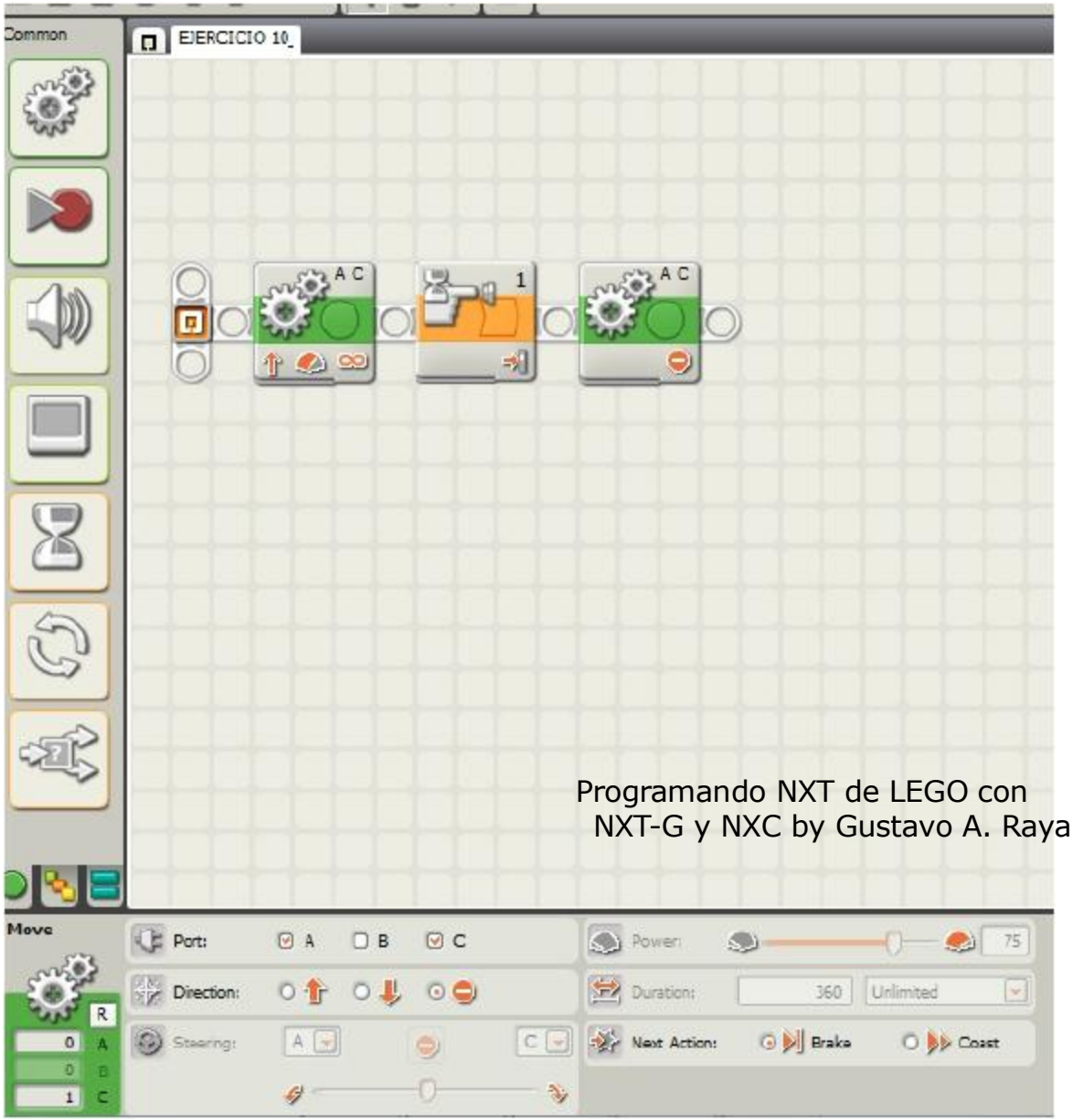
Definir la entrada 1 como entrada de sensor de choque y poner en robot a caminar hacia delante HASTA QUE detecte un obstáculo con el sensor de choque definido.

Después de detectar el obstáculo, se pararán los motores y se detendrá el programa.

Ejercicio 10_2. Sensor de choque.

```
task main()
{
  SetSensor(IN_1,SENSOR_TOUCH);

  OnFwd(OUT_AC, 75);

  until (SENSOR_1 == 1);

  Off(OUT_AC);
}
```

Ejercicio 10_2. Solución Lego Mindstorms NXT.



Programando NXT de LEGO con
NXT-G y NXC by Gustavo A. Raya Casero

Ejercicio 10_3. Sensor de choque.

Implementar un programa que ejecute las siguientes acciones:

Definir las entradas 1 y 2 como entradas conectadas a sensores de choque.

Encienda los motores A y C para que avancen hacia adelante.

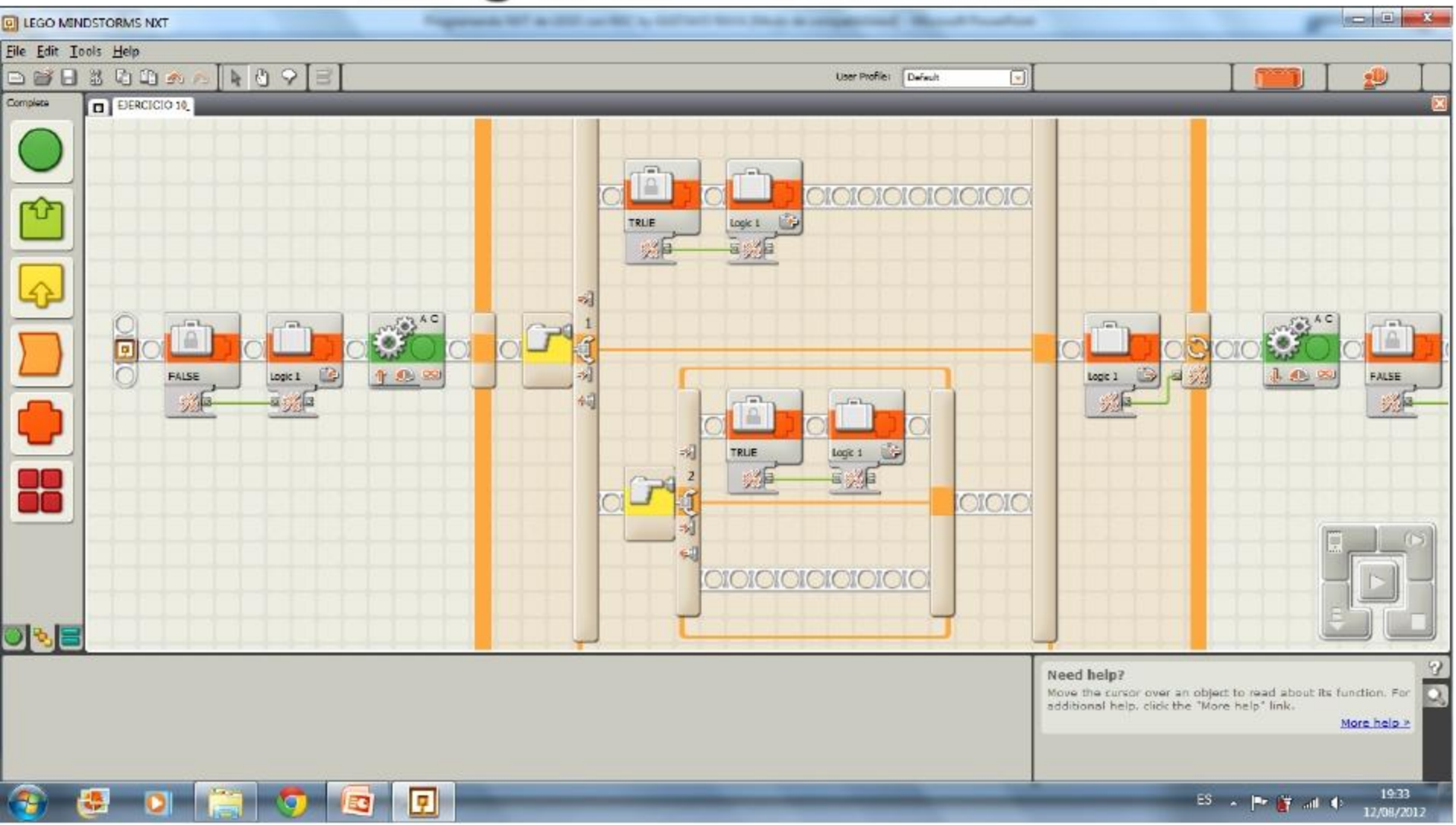
Si se presiona un sensor de contacto y se mantiene presionado, se invertirá la dirección de los motores.

Al liberar el sensor de contacto, los motores se apagarán y el programa terminará.

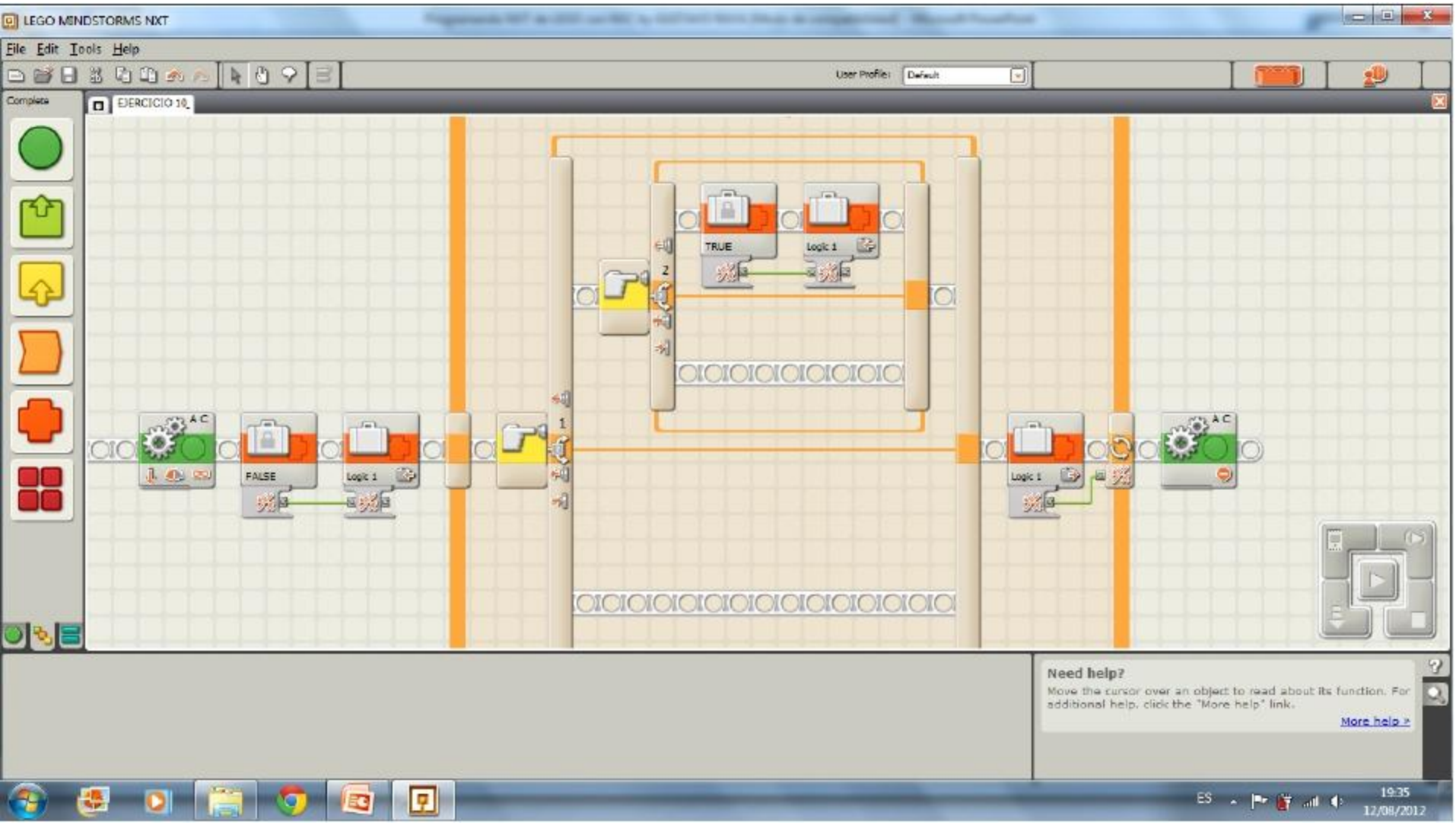
Ejercicio 10_3. Solución NXC.

```
task main()
{
        SetSensor(IN_1, SENSOR_TOUCH);
        SetSensor(IN_2, SENSOR_TOUCH);
        OnFwd(OUT_AC,50);

        until ((SENSOR_1 == 1)||(SENSOR_2==1));
/*Desde que se pulse al menos uno de los dos pulsadores, ||=OR, el
robot dará marcha atrás. */
        OnRev(OUT_AC,50);

        until ((SENSOR_1 == 0)&&(SENSOR_2==0));
/*Ambos sensores deben estar sin pulsar, &&=AND, para que el robot
se pare.*/
        Off(OUT_AC);
}
```

Ejercicio 10.3. Solución Lego Mindstorms NXT.



Ejercicio 10.3. Soluciém Lego Mindstorms NXT.



Ejercicio 10_4. Sensor de choque.

Implementar un programa que haga que el robot camine hacia delante hasta que choque con alguno de los sensores.

Si choca con el sensor izdo.:

dará marcha atrás con el motor izdo. manteniendo hacia delante el motor dcho. hasta que soltemos el sensor izdo.

al soltar el sensor izdo., volverá a caminar hacia delante con los dos motores.

Si choca con el sensor dcho.:

dará marcha atrás con el motor dcho. manteniendo hacia delante el motor izdo., hasta que soltemos dicho sensor.

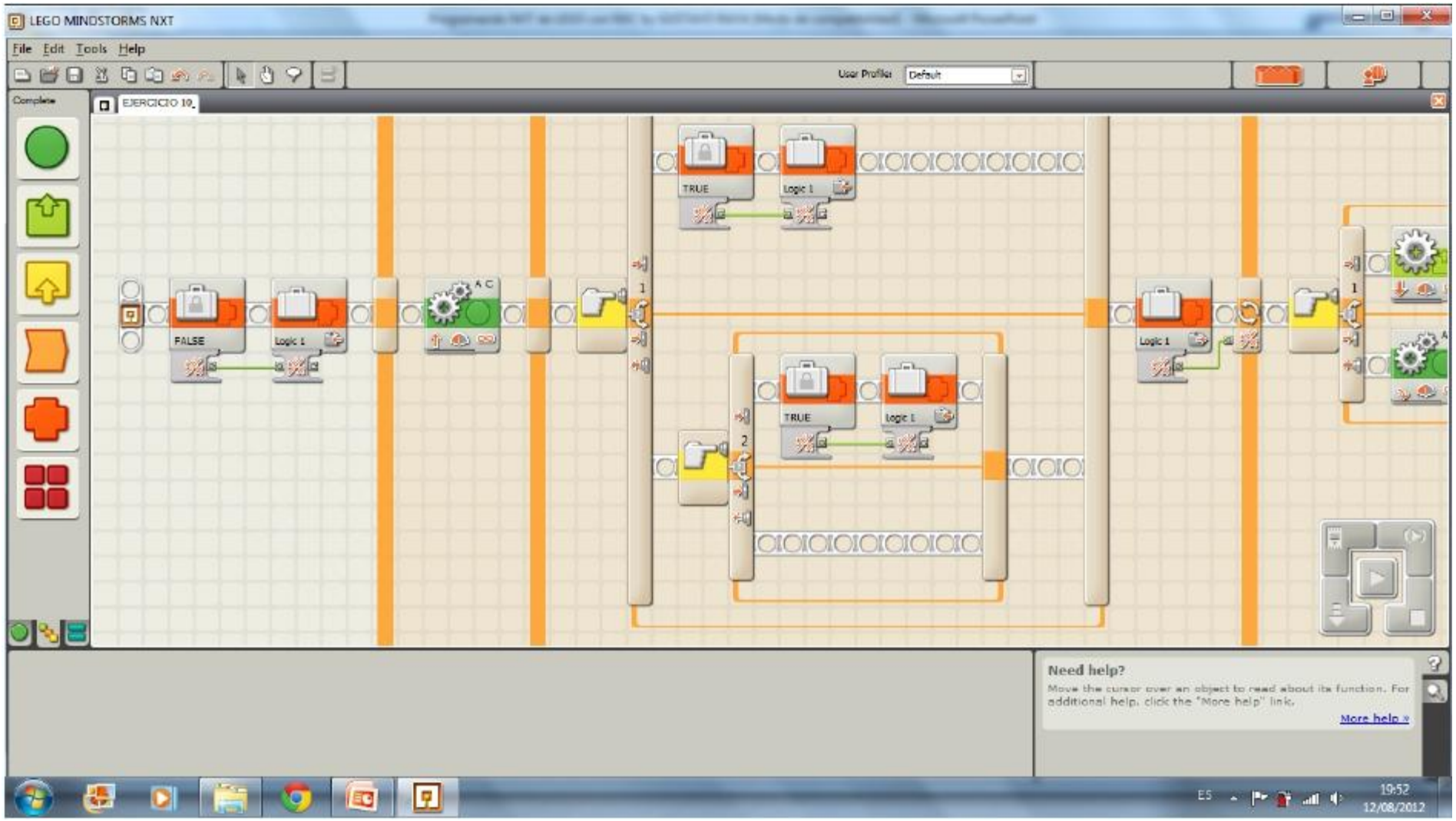
al soltar el sensor derecho, volverá a caminar hacia delante con los dos motores.

Repetirá este bucle de manera infinita.

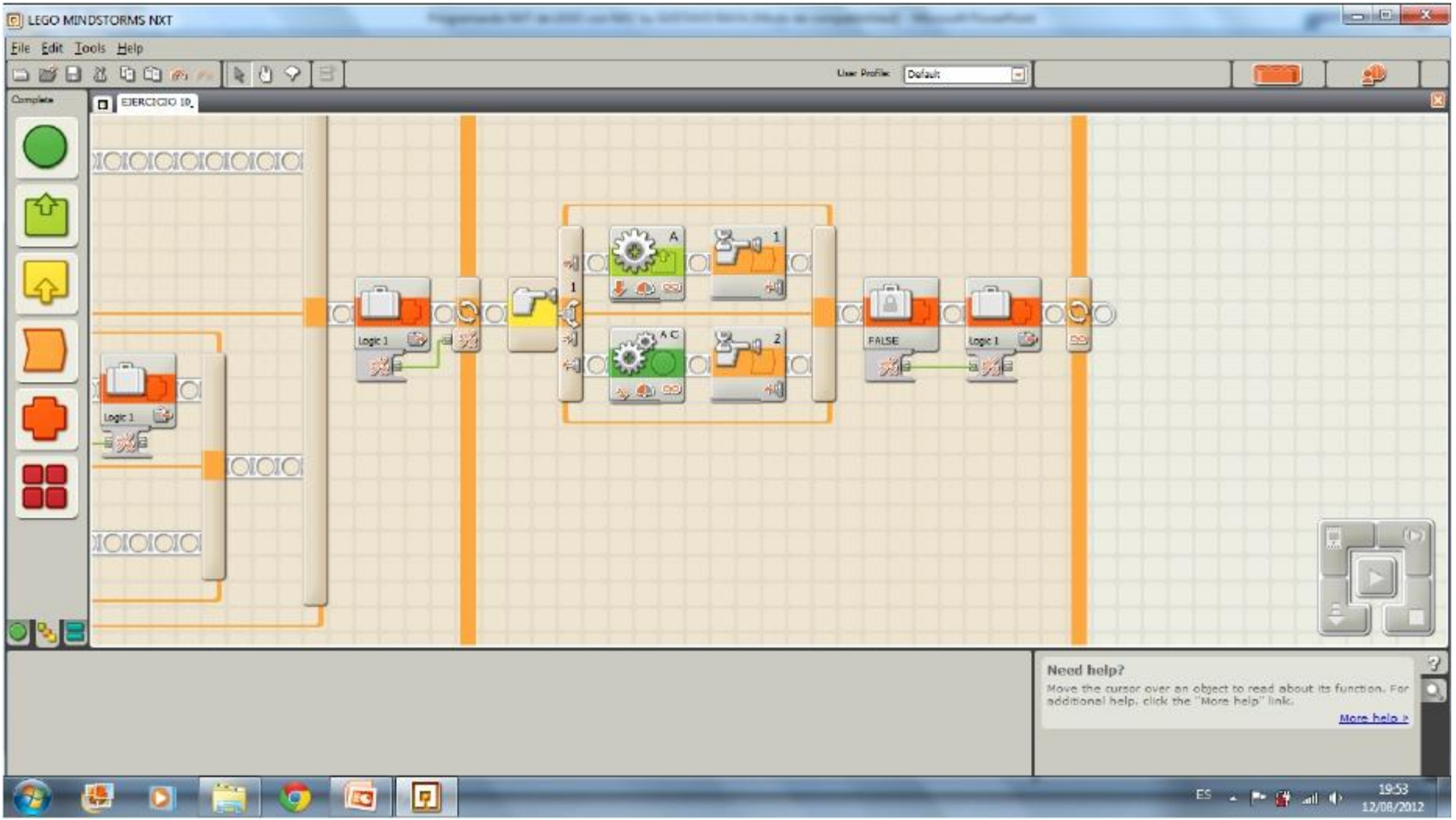
Ejercicio 10_4. Solución NXC.

```
task main()
{
    SetSensor(IN_1, SENSOR_TOUCH);
    SetSensor(IN_2, SENSOR_TOUCH);
    while (true) {
          OnFwd(OUT_AC,50);
          until ((SENSOR_1 == 1) || (SENSOR_2 == 1));
           /*Saldrá de la instrucción until cuando se haya pulsado el
              SENSOR_1 o el SENSOR_2*/

        if (SENSOR_1 == 1) {//Si se pulsa el SENSOR_1
           OnRev(OUT_A,50); //Marcha atrás hasta... (until)
           until (SENSOR_1 == 0);// ...(until) que se suelte el SENSOR_1.
        }
        else //Si no fue el SENSOR_1 entonces tiene que ser el SENSOR_2
        {
           OnRev(OUT_C,50);  //Marcha atrás hasta... (until)
           until (SENSOR_2 == 0);// ...(until) que se suelte el SENSOR_2.
        }
    }
}
```

Ejercicio 10.4.. Solucióm Lego Mindstorms NXT..



Ejercicio 10.4.. Solucióm Lego Mindstorms NXT..



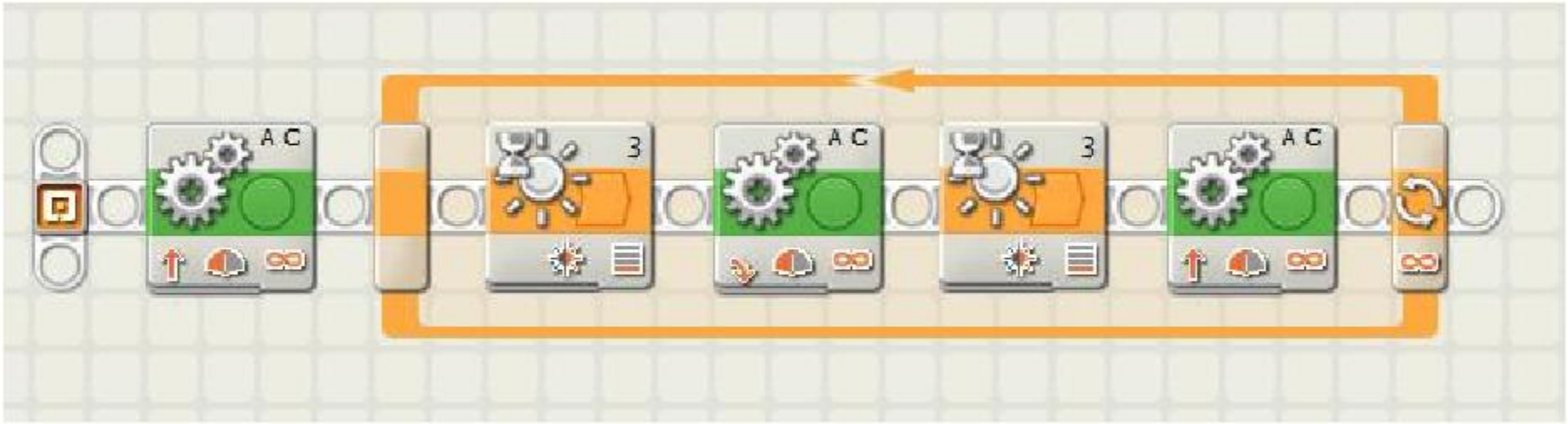
Ejercicio 11_1.

Implementar un programa que ejecute las siguientes acciones:
- definir la input 3 como entrada conectada al sensor de luz,
- el robot caminará hacia delante mientras el nivel de luz sea menor o igual al nivel definido en la macro THRESHOLD,
- SI el nivel de luz es mayor que el indicado en THRESHOLD, el NXT girará hacia atrás con el motor C hasta que el nivel de luz vuelva a ser menor o igual al indicado en THRESHOLD. Cuando esto ocurra volverá a caminar hacia adelante repitiendo todo el proceso en un bucle infinito.

Ejercicio 11_1.

```
#define THRESHOLD 40
/* SENSOR_LIGHT = IN_3 > 40 → NO NEGRO.
   SENSOR_LIGHT = IN_3 <= 40 → NEGRO. */
task main()
{
  SetSensorLight(IN_3); /*SetSensor(IN_3;SENSOR_LIGHT); orden equivalente.*/
  OnFwd(OUT_AC, 25);
  while (true)
  {
    if (Sensor(IN_3) > THRESHOLD) //Si es un color claro
      OnRev(OUT_C, 25); // Marcha atrás con el motor C
      Wait(100);
      until(Sensor(IN_3 <= THRESHOLD);//Detecta un color oscuro(negro).
      OnFwd(OUT_AC, 25);//Camina hacia adelante.
    }
  }
}
```

Ejercicio 11.1. Solución Lego Mindstorms NXT.



Ejercicio 11_2. Sensor de LUZ

Implementar un programa que haga lo siguiente:

El robot caminará hacia adelante hasta que detecte una superficie negra.

Al detectar la superficie negra girará sobre si mismo dando la vuelta.

Después de dar la vuelta, volverá a caminar hacia adelante durante un tiempo y se parará.

Ejercicio 11_2.
Solución Lego Mindstorms NXT.

Ejercicio 11_2. Solución NXC.

```
#define THRESHOLD 40

task main()
{
  SetSensorLight(IN_3);
  OnFwd(OUT_AC, 50);
  until (SENSOR_3<THRESHOLD);
  OnRev(OUT_C,50);
  Wait(1000);
  OnFwd(OUT_AC,50);
  Wait(1000);
  Off(OUT_AC);
}
```

Light Sensor. Ejercicio 11_3.

Implementar un programa que infinitamente haga lo siguiente:

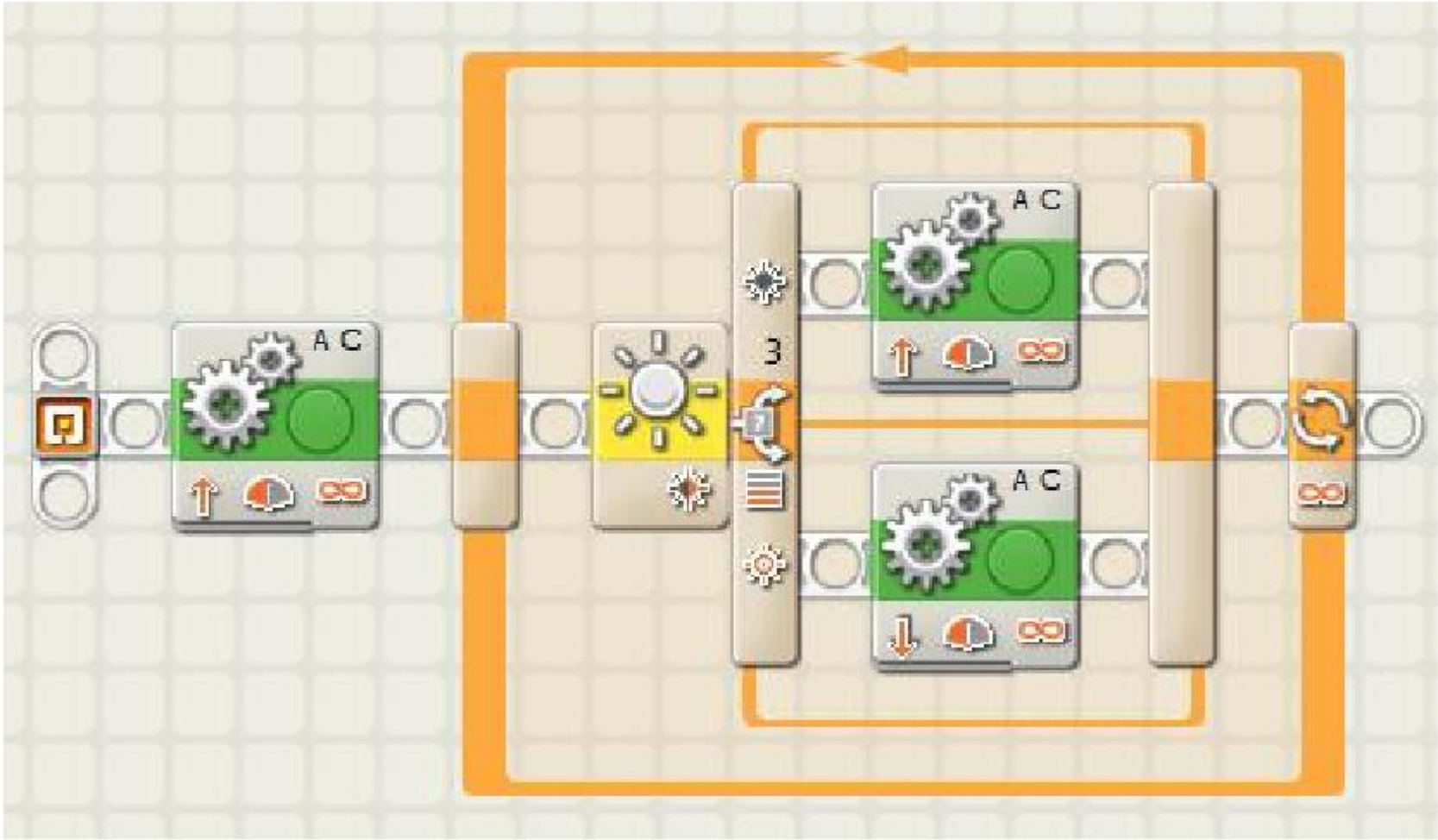
Si el haz del sensor de luz se incide sobre una superficie negra, el robot caminará hacia adelante.

Si el haz del sensor de luz se incide sobre una superficie no negra, el robot caminará hacia atrás.

Ejercicio 11_3. Solución

```
#define LIGHT           SENSOR_3
#define THRESHOLD    40

task main()
{
    SetSensorLight(IN_3);
   OnFwd(OUT_AC,50);
    while (true)
    {
          if (LIGHT <= THRESHOLD)
          {
            OnFwd(OUT_AC,50); Wait(500);
          }
          else
          {
            OnRev(OUT_AC,50); Wait(500);
          }
    }
}
```

Ejercicio 11_3.
Solución Lego Mindstorms NXT.



Sensor ULTRASÓNICO.

El sensor emite un sonido y mide el tiempo que la señal tarda en regresar, para luego calcular la distancia, a la cual se encuentra el objeto u obstáculo. Es el mismo principio utilizado por los murciélagos y el sonar de las naves, tiene una rango de 0 a 255 cm con una precisión de +/- 3 cm.

Ejercicio 12.

Implementar un programa que infinitamente haga lo siguiente:

Definir la entrada 4 como input del sensor de luz.

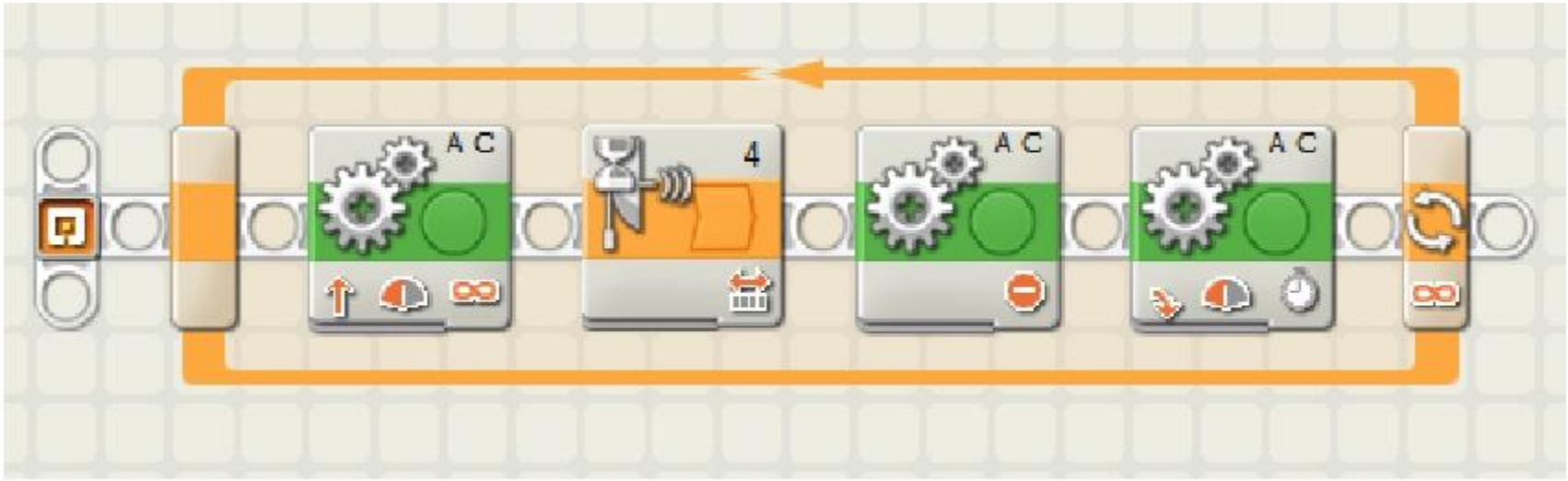
El robot caminará hacia adelante MIENTRAS no detecte un obstáculo a menos de
15cm.

Si detecta un obstáculo a 15 cm o menos de distancia, parará los motores,
dará marcha atrás con el motor C durante 800msgs y, posteriormente, volverá a
caminar hacia adelante repitiendo todo el proceso de forma indefinida.

Ejercicio 12.

```
#define NEAR 15 //MACRO = constante → NEAR=15cm.

task main(){

  SetSensor(IN_4,SENSOR_LOWSPEED); //SetSensorLowspeed(IN_4);

  while(true){
    OnFwd(OUT_AC,50);
    while(SensorUS(IN_4)>NEAR);
    Off(OUT_AC);
    OnRev(OUT_C,100);
    Wait(800);
  }
}
```

Ejercicio 12.
Solución Lego Mindstorms NXT.



Ejercicio 13.

Realizar un programa que ejecute las siguientes acciones:

•Definir el puerto 4 como entrada para el sensor de sonido.

•El robot estará andando en línea recta, de forma indefinida hasta que escuche un valor de sonido por encima de un umbral definido.

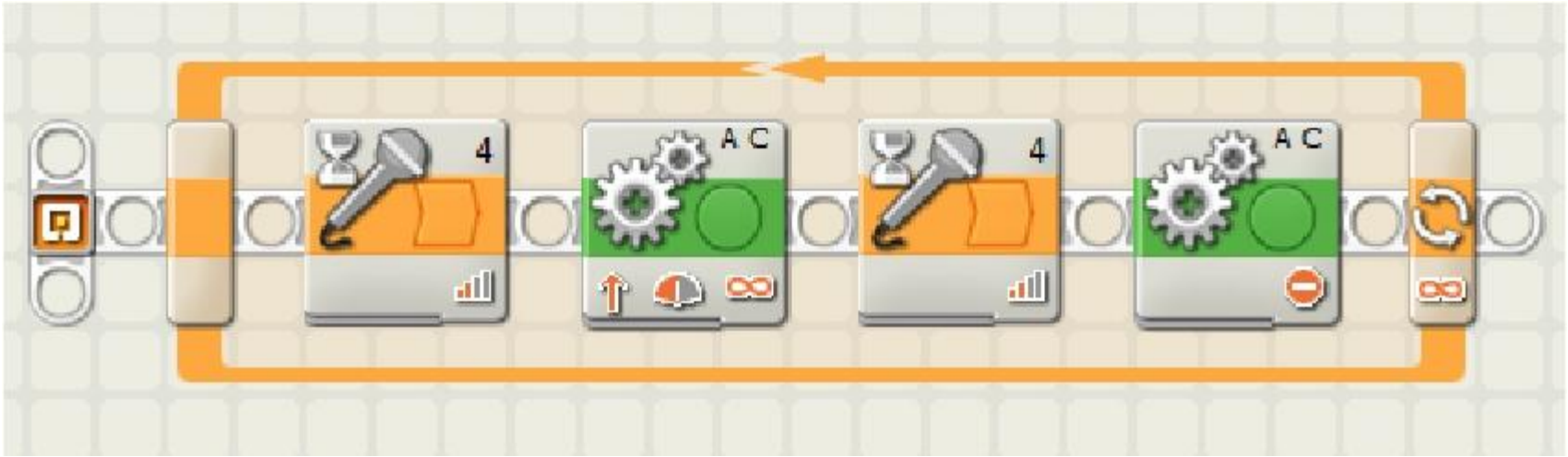
•Al escuchar un sonido por encima del umbral marcada, el robot se detendrá.

•El robot estará detenido hasta que vuelva a escuchar otro sonido por encima del umbral. Cuando esto ocurra volverá a andar nuevamente repitiendo todo el proceso descrito de forma indefinida (bucle infinito).

Ejercicio 13. Sound sensor.

```
#define THRESHOLD 60
#define MIC SENSOR_4 //Redefinimos la macro SENSOR_4 como MIC.
task main()
{
  //A continuación configuramos el puerto 4 como sensor de sonido.
  SetSensor(IN_4,SENSOR_SOUND); //SetSensorSound(IN_4);
  while(true){  // Bucle infinito.
     /* Con el comando until, el programa espera por un nivel de sonido mayor que
     el indicado en el umbral THRESHOLD: SENSOR_4 es una macro que devuelve
     el valor de sonido leído por el sensor.*/
     until(MIC > THRESHOLD); /*El programa se queda detenido en este punto hasta que
     el nivel de sonido recibido es mayor al indicado en THRESHOLD.*/
     OnFwd(OUT_AC, 75);
     Wait(300); /*Esta orden es para permitir al robot caminar al menos durante 0,3sgs
     antes de volver a leer el SENSOR_4=MIC de nuevo, si no la misma onda de retorno
     podría  hacer detener al robot inmediatamente debido a la velocidad de procesado del
     NXT. */
     until(MIC > THRESHOLD); /* El robot continuará moviéndose hasta que "escuche" un
     valor until(MIC > THRESHOLD. Otra orden alternativa a esta podría haber sido
     while(MIC <= THRESHOLD);*/
     Off(OUT_AC); /* El robot se parará hasta que vuelva a escuchar un sonido
     MIC > THRESHOLD.*/
     Wait(300);
  }
}
```

Ejercicio 13.
Solución Lego Mindstorms NXT.



TASKS = TAREAS

Un programa NXC puede contener 255 tareas como máximo, cada una de las cuales debe tener un nombre único.

La tarea **main()** debe existir **SIEMPRE**, puesto que es la primera se ejecuta.

El resto de tareas serán ejecutadas sólo cuando una tarea que esté corriendo (en ejecución) la invoque o la llame. Obviamente, puede ser la tarea **main()** la que invoque a otra tarea.

La tarea que invoca y la invocada, desde el momento de la llamada, estarán corriendo simultáneamente si ninguna de las dos termina antes su ejecución.

Ejercicio 14. Tareas.

Queremos hacer un programa que haga mover al robot en cuadrados, como ya habíamos hecho anteriormente. Sin embargo, en esta ocasión queremos que el robot reaccione cuando golpee un obstáculo.

No es aconsejable implementar este programa con una sola tarea, ya que el robot debe hacer dos tareas simultáneamente:

- Caminar dibujando cuadrados
- Detectar si se han activado los sensores de choque.

Por esta razón es mucho mejor utilizar dos tareas, una encargada de hacer mover al NXT dibujando cuadrados, y otra encargada de reaccionar a los sensores de choque.

Ejercicio 14.1

```
mutex moveMutex;

task move_square()
{
  while (true)
  {
    Acquire(moveMutex);    // Acquire = hacerse con, apoderarse de.
    OnFwd(OUT_AC, 50); Wait(1000);
    OnRev(OUT_C, 50); Wait(500);
    Release(moveMutex); //Release = liberar.
  }
}

task check_sensors()
{
  while (true)
  {
    if (SENSOR_1 == 1)
    {
      Acquire(moveMutex);
      OnRev(OUT_AC, 50); Wait(500);
      OnFwd(OUT_A, 50); Wait(500);
      Release(moveMutex);
    }
  }
}

task main()
{
  SetSensorTouch(IN_1);
  Precedes(move_square, check_sensors);
}
```

/* Es MUY IMPORTANTE recordar que las tareas lanzadas están corriendo a la vez, y esto puede provocar RESULTADOS INEXPERADOS si no se tiene en cuenta, por ejemplo si ambas tareas intentan mover los motores a la vez.

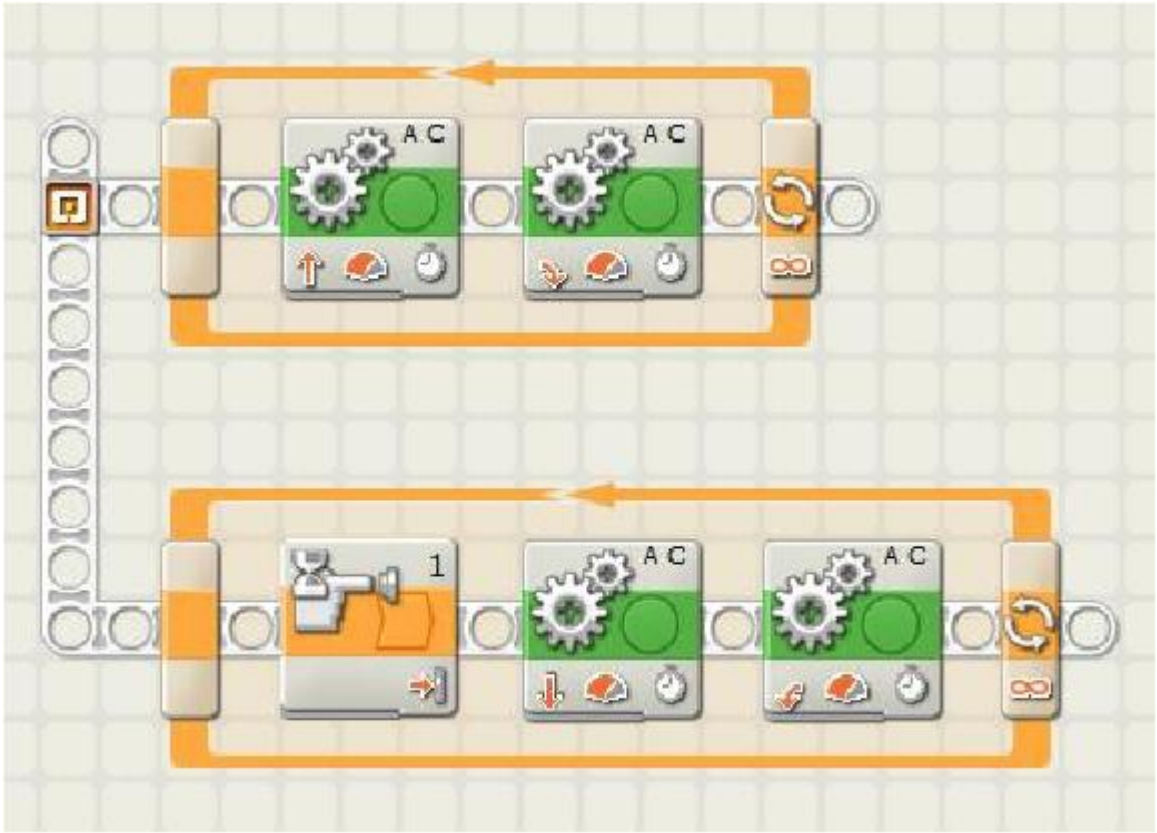
Para evitar estos problemas, declaramos un tipo de variable, mutex (mutual exclusion).

Así pues, sólo se puede actuar sobre este tipo de variables con las funciones Acquire() y Release(), escribiendo la parte de código crítica entre estas dos funciones.

Esto nos garantiza que sólo una tarea podrá tener control total de los motores en cada momento.

Estas variable de tipo mutex son conocidas como semáforos, y este tipo de programación es denominada programación concurrente.*/

Ejercicio 14.1
Solución Lego Mindstorms NXT.



Ejercicio 14.2 con semáforos.

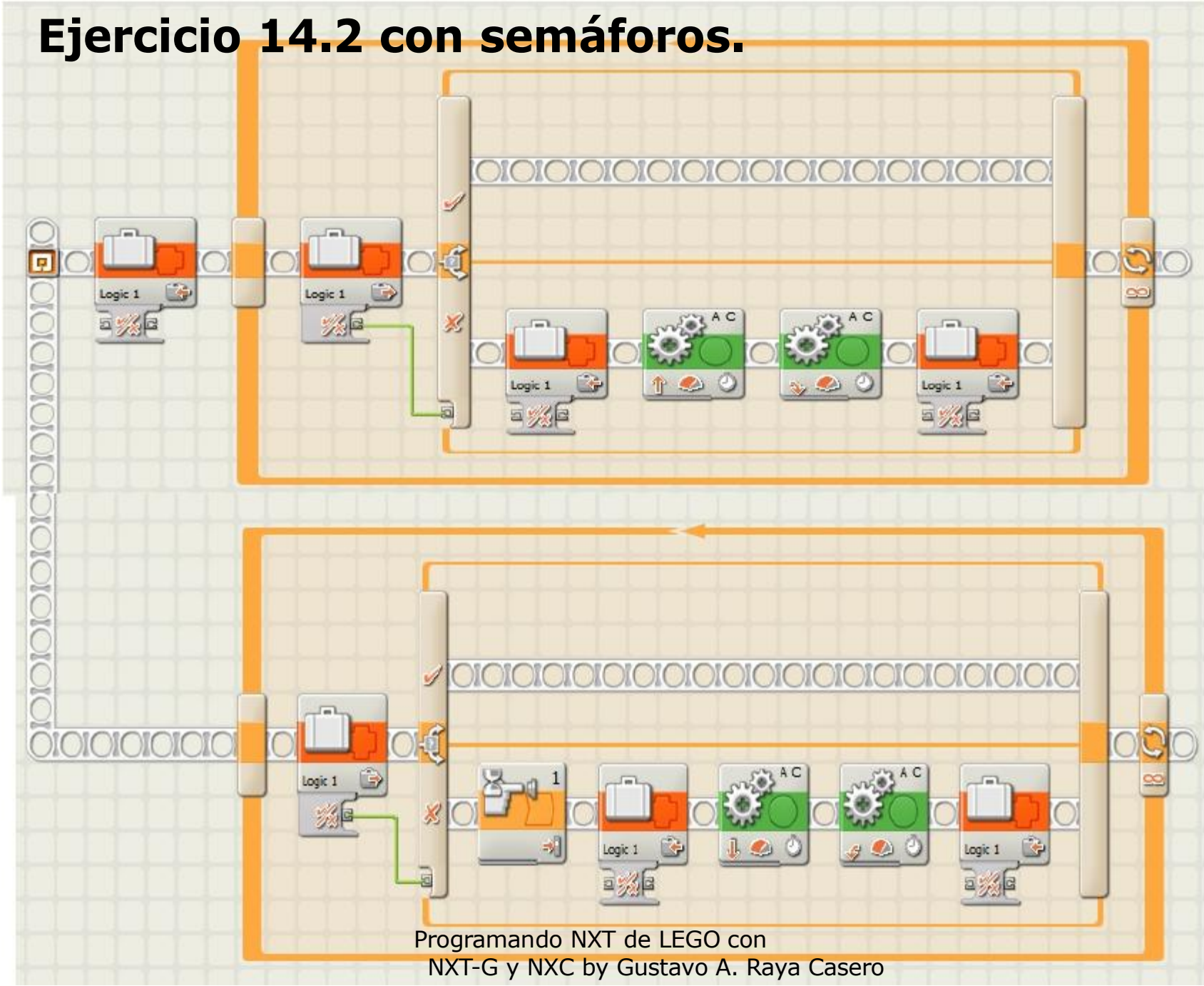
```
int sem;

task move_square()
{
  while (true)
  {
    until (sem == 0);
    sem = 1;
    OnFwd(OUT_AC, 50);
    sem = 0;
    Wait(1000);

    until (sem == 0);
    sem = 1;
    OnRev(OUT_C, 50);
    sem = 0;
    Wait(850);
  }
}
```

/*Podría discutirse la necesidad de poner en la tarea move_square el semáforo sem=1 y al final de la misma de nuevo sem=0. La razón es debido a que el comando OnFwd() es en realidad dos comandos.
Si no pusiéramos este semáforo, esta SECUENCIA de dos comandos podría ser interrumpida por la otra tarea a la mitad de su ejecución.*/

```
task submain()
{
  SetSensor(IN_1, SENSOR_TOUCH);
  while (true)
  {
    if (SENSOR_1 == 1)
    {
      until (sem == 0); sem = 1;
      OnRev(OUT_AC, 50); Wait(500);
      OnFwd(OUT_A, 50); Wait(850);
      sem = 0;
    }
  }
}
```

```
task main()
{
  sem = 0;
  Precedes(move_square, submain);
}
```

Ejercicio 14.2 con semáforos.



Ejercicio 14.3.

- Escriba un programa con 2 tareas.
- La primera chequea el sensor de contacto 1. Si este es presionado, el robot dará marcha atrás sólo con el motor que está conectado en el lado del sensor de contacto 1 mientras dicho sensor esté pulsado.
- La segunda chequea el sensor de contacto 3. Al accionar este otro contacto, el motor que está en el lado de este sensor avanzará hacia adelante mientras el sensor esté pulsado.
- En ambos casos, al soltar los pulsadores, el robot parará los motores hasta que se vuelva a tocar otro pulsador.
- El proceso se repite inffinitamente.

Ejercicio 14.3.

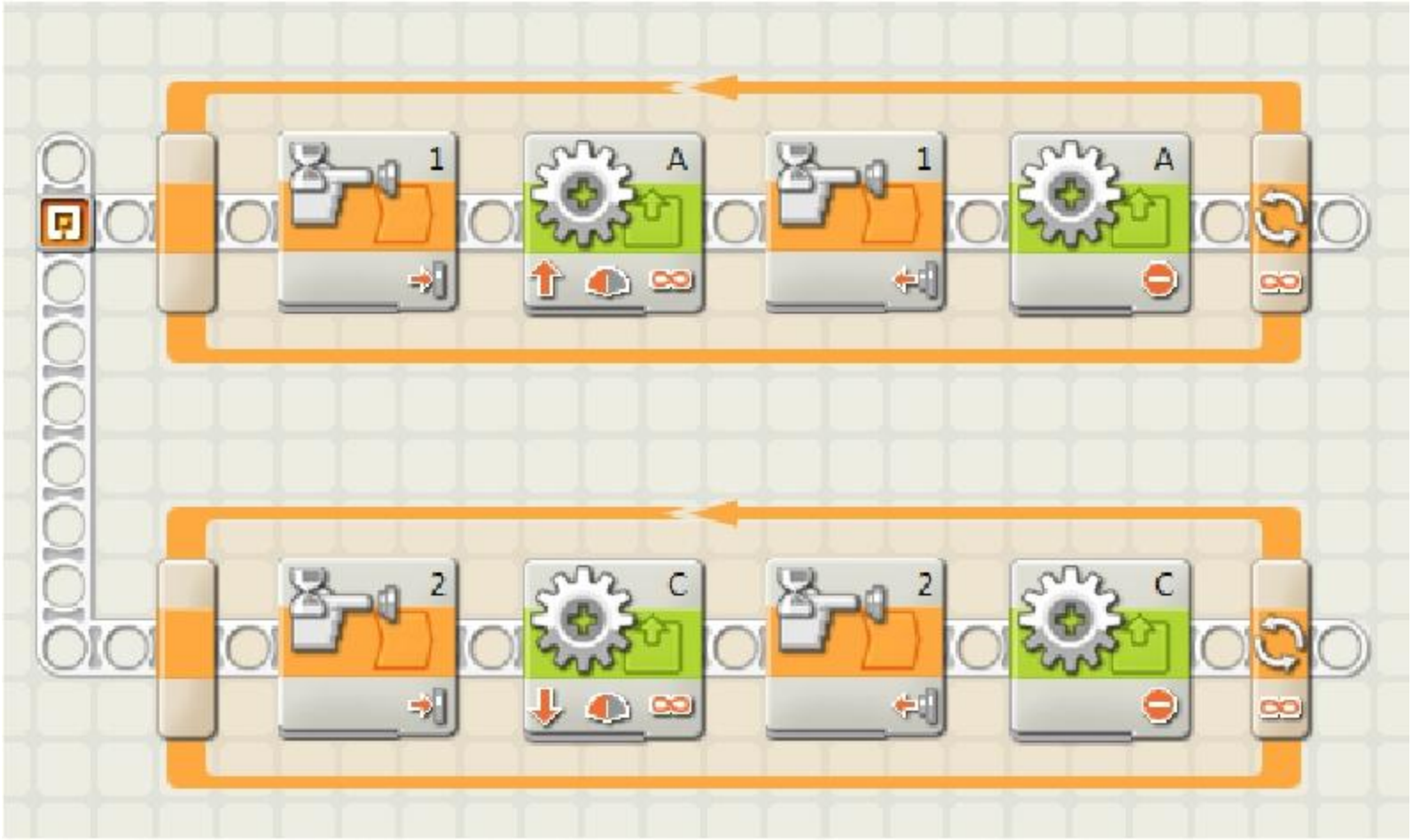
Solución NXC.

```
mutex moveMutex;

task sensor1()
{
  SetSensorTouch(IN_1);
  while (true)
  {
    until (SENSOR_1==1);
    Acquire(moveMutex);
    OnFwd(OUT_A, 50);
    until (SENSOR_1==0);
    Off(OUT_A);
    Release(moveMutex);
  }
}
```

```
task sensor2()
{
  SetSensorTouch(IN_2);
  while (true)
  {
    until (SENSOR_2==1);
    Acquire(moveMutex);
    OnRev(OUT_C, 50);
    until (SENSOR_2==0);
    Off(OUT_C);
    Release(moveMutex);
  }
}
```

```
task main()
{
  Precedes(sensor1,sensor2);
}
```

Ejercicio 14.3.
Solución Lego Mindstorms NXT.



Ejercicio 14.4.

Implemente un programa que cumpla con las siguientes condiciones:

Hacer que el robot avance hacia delante y que, al detectar un obstáculo con alguno de los sensores de choque (izdo. o dcho.) proceda como sigue...

...al detectar el obstáculo, parará los motores, dará marcha atrás durante 0,5 segs., girará hacia el lado contrario del obstáculo durante 0,5 segs., parará los motores de nuevo para inmediatamente continuar hacia adelante indefinidamente hasta que vuelva a encontrar otro obstáculo.

El proceso se repetirá infinitamente.

Ejercicio 14.4.

El programa debe tener las siguientes tareas:

- task main() {…}
- task detecta_izda() {…}
- task detecta_dcha() {…}

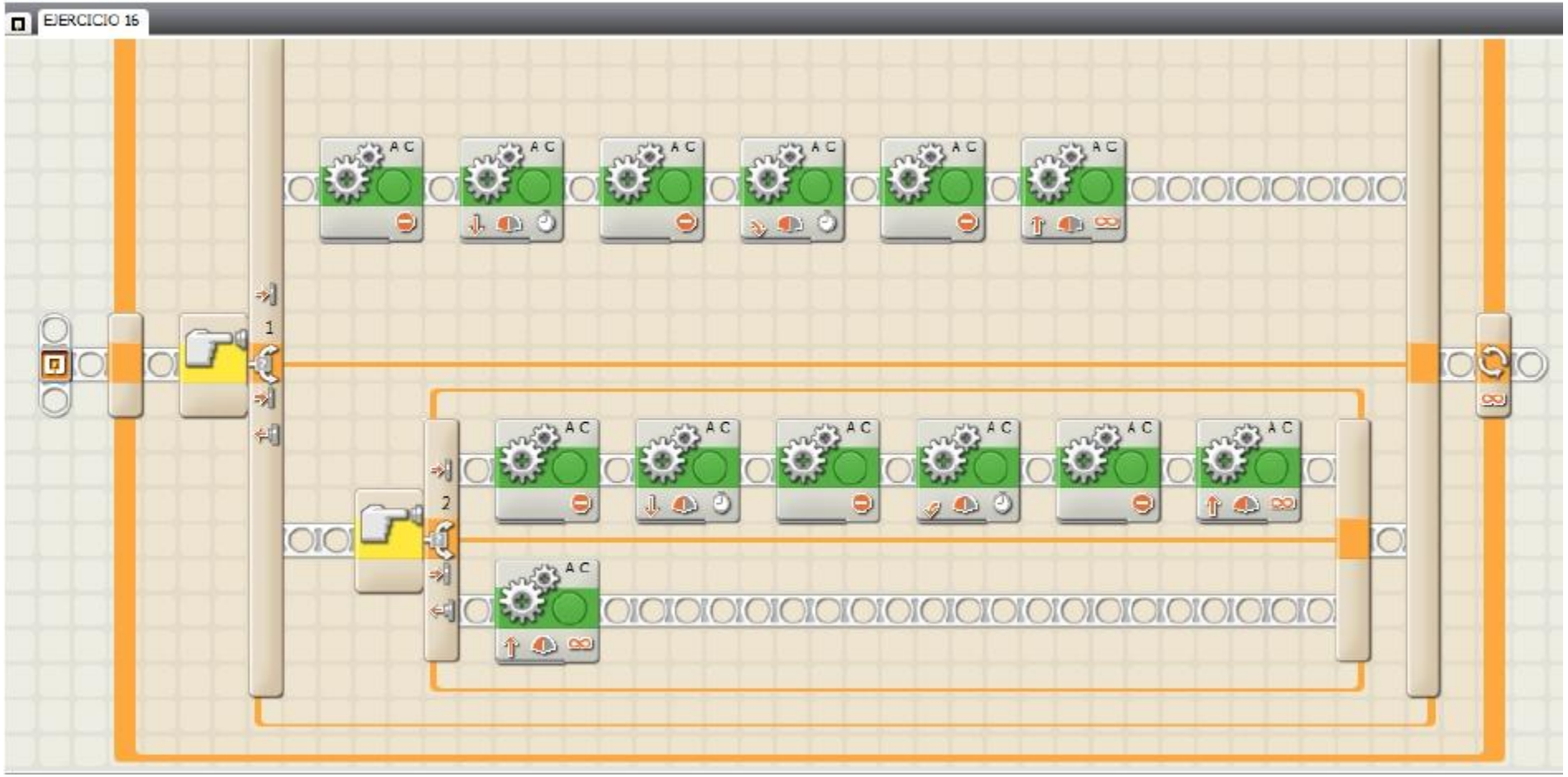
Ejercicio 14.4. Solución NXC.

```
mutex moveMutex;

task detecta_izda()
{
  SetSensorTouch(IN_2);
  while (true)
  {
    if (SENSOR_2==1)
    {
        Acquire(moveMutex);
        Off(OUT_AC);
        OnRev(OUT_AC,50);  Wait(500);
        Off(OUT_AC);
        OnFwd(OUT_C,50);  OnRev(OUT_A,50);
        Wait(500);
        Off(OUT_AC);
        OnFwd(OUT_AC,50);
        Release(moveMutex);
    }
  }
}
```

```
task detecta_dcha()
{
  SetSensorTouch(IN_1);
  while (true)
  {
    if (SENSOR_1==1)
    {
        Acquire(moveMutex);
        Off(OUT_AC);
        OnRev(OUT_AC,50);  Wait(500);
        Off(OUT_AC);
        OnFwd(OUT_A,50);  OnRev(OUT_C,50);
        Wait(500);
        Off(OUT_AC);
        OnFwd(OUT_AC,50);
        Release(moveMutex);
    }
  }
}
```

```
task main()
{
  OnFwd(OUT_AC,50);
  Precedes(detecta_dcha,detecta_izda);
}
```

Ejercicio 14.4.
Solución Lego Mindstorms NXT.



Subrutinas.

En ocasiones, un mismo código se necesita varias veces en diferentes partes de nuestro programa.

En estos casos, ese código que va a ser repetido se implementa en una subrutina, que es llamada mediante su invocación cada vez que ese código se hace necesario.

Una subrutina, a diferencia de una función, no devuelve un valor.

Ejercicio 15.

Implemente un programa que cumpla con las siguientes condiciones:
- El NXT caminará hacia adelante durante 1 seg. al 75% de su potencia máxima.
- Girará hacia la izquierda durante 0,9sgs. al 50% de su potencia máxima. Esta acción la realizará mediante la llamada a una subrutina.
- Caminará nuevamente hacia adelante durante 2 segs. al 50% de su potencia máxima. Esta última acción deberá estar integrada en la subrutina llamada previamente.
- Girará hacia la izquierda durante 0,9sgs. al 75% de su potencia máxima. Esta acción la realizará mediante la llamada a una subrutina.
- Caminará nuevamente hacia adelante durante 1 seg. al 75% de su potencia máxima. Esta última acción deberá estar integrada en la subrutina llamada previamente.
- Girará hacia la izquierda durante 0,9sgs. al 100% de su potencia máxima. Esta acción la realizará mediante la llamada a una subrutina.
- Finalmente parará los motores.

Ejercicio 15.

```
sub turn_around(int pwr)
{
  OnRev(OUT_C, pwr); Wait(900);
  OnFwd(OUT_AC, pwr);
}

task main()
{
  OnFwd(OUT_AC, 75);
  Wait(1000);
  turn_around(50);
  Wait(2000);
  turn_around(75);
  Wait(1000);
  turn_around(100);
  Off(OUT_AC);
}
```

/*En este programa hemos definido una subrutina que hace que el NXT gire y de marcha atrás.
La tarea principal llama a la subrutina tres veces.
A la subrutina se le pasa el valor de la potencia de los motores por parámetro (argumento), escribiendo el valor pasado entre paréntesis.
Si una subrutina no tiene parámetros, estos se escriben sin nada en su interior.*/

Funciones inline.

El principal beneficio de las subrutinas es que están almacenadas sólo en una posición de memoria, por lo que ahorramos "espacio".

Sin embargo, cuando las subrutinas son cortas, es más conveniente utilizar funciones inline. Estas funciones inline no son guardadas de formas separada en memoria, sino que son insertadas literalmente en cada una de las partes del código desde donde son invocadas. Obviamente esto ocupa más memoria al implicar más líneas de código, pero evita el salto de posición memoria que se hace en la llamada a subrutina.

Ejercicio 16.1.

Implemente un programa que cumpla con las siguientes condiciones:
- El NXT caminará hacia adelante durante 1 seg. al 75% de su potencia máxima.
- Girará hacia la izquierda durante 0,9sgs. al 50% de su potencia máxima. Esta acción la realizará mediante la llamada a una función inline.
- Caminará nuevamente hacia adelante durante 2 segs. al 50% de su potencia máxima. Esta última acción deberá estar integrada en la función inline llamada previamente.
- Girará hacia la izquierda durante 0,9sgs. al 75% de su potencia máxima. Esta acción la realizará mediante la llamada a una función inline.
- Caminará nuevamente hacia adelante durante 1 seg. al 75% de su potencia máxima. Esta última acción deberá estar integrada en la función inline llamada previamente.
- Girará hacia la izquierda durante 0,9sgs. al 100% de su potencia máxima. Esta acción la realizará mediante la llamada a una función inline.
- Finalmente parará los motores.

Ejercicio 16.1.

```
inline void turn_around()
{
  OnRev(OUT_C, 75); Wait(900);
  OnFwd(OUT_AC, 75);
}

task main()
{
  OnFwd(OUT_AC, 75);
  Wait(1000);
  turn_around();
  Wait(2000);
  turn_around();
  Wait(1000);
  turn_around();
  Off(OUT_AC);
}
```

/*Las funciones pueden retornar tipos de valores diferentes a void, por ejemplo integer, string, etc…*/

Ejercicio 16.2.

Implemente un programa que cumpla con las siguientes condiciones:
- El NXT caminará hacia adelante durante 1 seg. al 75% de su potencia máxima.
- Girará hacia la izquierda durante 0,9sgs. al 50% de su potencia máxima. Esta acción la realizará mediante la llamada a una función inline.
- Caminará nuevamente hacia adelante durante 2 segs. al 50% de su potencia máxima. Esta última acción deberá estar integrada en la función inline llamada previamente.
- Girará hacia la izquierda durante 0,9sgs. al 75% de su potencia máxima. Esta acción la realizará mediante la llamada a una función inline.
- Caminará nuevamente hacia adelante durante 1 seg. al 75% de su potencia máxima. Esta última acción deberá estar integrada en la función inline llamada previamente.
- Girará hacia la izquierda durante 0,9sgs. al 100% de su potencia máxima. Esta acción la realizará mediante la llamada a una función inline.
-Finalmente parará los motores.
Implementar esta solución pasando por parámetro a la función inline los argumentos potencia de giro y tiempo de giro. Una posible declaración de la función inline podría ser “inline void turn_around(int pwr, int turntime)”.

Ejercicio 16.2.

```
inline void turn_around(int pwr, int turntime)
{
  OnRev(OUT_C, pwr);
  Wait(turntime);
  OnFwd(OUT_AC, pwr);
}

task main()
{
  OnFwd(OUT_AC, 75);
  Wait(1000);
  turn_around(75, 2000);
  Wait(2000);
  turn_around(75, 500);
  Wait(1000);
  turn_around(75, 3000);
  Off(OUT_AC);
}
```

Definiendo macros.

Hay otra manera de darle nombre a pequeños trozos de código que van a ser utilizados en varias partes de un programa. NXC nos permite hacer esto con el uso de macros. Ya hemos visto que podemos definir constantes, usando #define nombre.
Sin embargo, utilizando esto mismo podemos definir trozos de código que será utilizado varias veces (ver ej 17.1).

Ejercicio 17.1.

Implemente un programa que cumpla con las siguientes condiciones:

- El NXT caminará hacia adelante durante 1 seg. al 75% de su potencia máxima.
- Girará hacia la izquierda durante 0,9sgs. al 50% de su potencia máxima. Esta acción la realizará mediante la llamada a una macro.
- Caminará nuevamente hacia adelante durante 2 segs. al 50% de su potencia máxima. Esta última acción deberá estar integrada en la macro llamada previamente.
- Girará hacia la izquierda durante 0,9sgs. al 75% de su potencia máxima. Esta acción la realizará mediante la llamada a una macro.
- Caminará nuevamente hacia adelante durante 1 seg. al 75% de su potencia máxima. Esta última acción deberá estar integrada en la macro llamada previamente.
- Girará hacia la izquierda durante 0,9sgs. al 100% de su potencia máxima. Esta acción la realizará mediante la llamada a una macro.
-Finalmente parará los motores.

Ejercicio 17.1.

```
#define turn_around      OnRev(OUT_B, 75); Wait(3400);OnFwd(OUT_AB, 75);


task main()
{
  OnFwd(OUT_AB, 75);
  Wait(1000);
  turn_around;
  Wait(2000);
  turn_around;
  Wait(1000);
  turn_around;
  Off(OUT_AB);
}
```

/*Después de #define, **turn_around** representa el texto que está a continuación. Por tanto, cuando se escriba **turn_around**, esta macro es sustituída por el código que se escribe justo después de su declaración. El #define debe estar declarado sólo en una línea. Aunque hay formas de poner un #define en múltiples líneas, esto NO es una práctica recomendada.*/

Ejercicio 17.2.

Implemente un programa que ejecute las siguientes acciones:
-El NXT caminará hacia atrás durante 1 seg. al 50% de su potencia máxima.
-El NXT caminará hacia adelante durante 1 seg. al 50% de su potencia máxima.
-El NXT girará hacia la izquierda durante 0,75 seg. al 75% de su potencia máxima.
-El NXT caminará hacia adelante durante 1 seg. al 75% de su potencia máxima.
-El NXT caminará hacia atrás durante 2 seg. al 75% de su potencia máxima.
-El NXT caminará hacia adelante durante 1 seg. al 75% de su potencia máxima.
-El NXT girará hacia la derecha durante 0,75 seg. al 75% de su potencia máxima.
-El NXT caminará hacia adelante durante 2 seg. al 30% de su potencia máxima.
-Parará los motores y terminará el programa.
Cada paso será implementado con la llamada a una macro.

Ejercicio 17.2.

```
#define turn_right(s,t)  OnFwd(OUT_A, s);OnRev(OUT_B, s);Wait(t);
#define turn_left(s,t)   OnRev(OUT_A, s);OnFwd(OUT_B, s);Wait(t);
#define forwards(s,t)    OnFwd(OUT_AB, s);Wait(t);
#define backwards(s,t)   OnRev(OUT_AB, s);Wait(t);

task main()
{
  backwards(50,1000);
  forwards(50,1000);
  turn_left(75,750);
  forwards(75,1000);
  backwards(75,2000);
  forwards(75,1000);
  turn_right(75,750);
  forwards(30,2000);
  Off(OUT_AB);
}
```

/*Define statements are actually a lot more powerful. They can also have arguments. For example, we can put the time to turn as an argument in the statement. Here is an example in which we define four macro's; one to move forwards, one to move backwards, one to turn left and one to turn right. Each has two arguments: the speed and the time. */

/*It is very useful to define such macros. It makes your code more compact and readable. Also, you can more easily change your code when you e.g. change the connections of the motors.*/

Tareas, subrutinas, funciones inline y macros.

Hasta ahora hemos visto el uso de tareas, subrutinas, funciones inline y macros. Todas tinen diferentes usos. A continuación tenemos un resumen que remarca sus diferencias:

• **Tareas.** Normalmente corren a la vez y se encargan de diferentes acciones, las cuales tienen que ser ejecutadas a la vez con otras acciones.

•**Subrutinas.** Son útiles cuando un mismo segmento de código, que es considerado no pequeño, deben ser utilizado en diferentes partes de una misma tarea.

•**Funciones inline.** Son útiles cuando el mismo segmento de código debe ser utilizado en diferentes lugares por diferentes tareas. Utilizan mas memoria.

•**Macros.** Son muy útiles para pequeños segmentos de código que necesita ser utilizado en diferentes partes del programa. Pueden tener parámetros.

Tocando música.

El NXT tiene un “built-in speaker” que puede reproducir tonos e incluso ficheros de sonido. Este último es particularmente útil cuando queremos que el NXT nos diga lo que está haciendo o lo que está pasando.
Podemos hacer que el NXT interprete música mientras se mueve.

Reproduciendo ficheros de sonido.

BricxCC tiene una utilidad built-in para convertir ficheros .wav en ficheros .rso, accesibles desde el menú Tools/Sound conversion.

Podemos guardar sonidos en ficheros .rso en la flash del NXT usando otra utilidad, el navegador de memoria del NXT (Tools/NXT explorer) y reproducir los ficheros con el comando:

PlayFileEx(filename, volume, loop?)

Los argumentos son el nombre del fichero de sonido, volumen (un número entre 0 y 7), y loop: este último argumento es puesto a 1 (TRUE) si queremos que el fichero se reproduzca con un bucle, y 0 (FALSE) si queremos que el fichero se reproduzca sólo una vez.

El NXT no espera a que la nota termine. Si diferentes ficheros son reproducidos consecutivamente, entonces sería conveniente añadir un pequeño retardo entre ellos.

Ejercicio 18.1.

```
#define TIME 300
#define MAXVOL 7
#define MINVOL 1
#define MIDVOL 3
#define pause_4th Wait(TIME)
#define pause_8th Wait(TIME/2)
#define note_4th PlayFileEx("! Click.rso",MIDVOL,FALSE); pause_4th
#define note_8th PlayFileEx("! Click.rso",MAXVOL,FALSE); pause_8th

task main()
{
  PlayFileEx("! Startup.rso",MINVOL,FALSE);
  Wait(2000);
  note_4th;
  note_8th;
  note_8th;
  note_4th;
  note_4th;
  pause_4th;
  note_4th;
  note_4th;
  Wait(100);
}
```

Reproduciendo música.

Para reproducir un tono podemos utilizar el comando:

PlayToneEx(frequency, duration, volume, loop?)

Tiene cuatro argumentos.

El primero frequency en Hertz, el segundo duration (en 1/1000 de sgs., como en el comando wait), y los dos últimos son volume and loop como el comando anterior (PlayFileEx(filename, volume, loop?).

PlayTone(frequency, duration) es otra opción; en este caso el volumen es fijado a través del menú del NXT, y loop está deshabilitado.

Como con PlayFileEx, el NXT no espera que una nota termine. Así que si usamos diferentes tonos consecutivamente, entonces sería conveniente añadir un pequeño retardo entre ellos.

Here is a table of useful frequencies:

Sound	3	4	5	6	7	8	9
B	247	494	988	1976	3951	7902	
A#	233	466	932	1865	3729	7458	
A	220	440	880	1760	3520	7040	14080
G#		415	831	1661	3322	6644	13288
G		392	784	1568	3136	6272	12544
F#		370	740	1480	2960	5920	11840
F		349	698	1397	2794	5588	11176
E		330	659	1319	2637	5274	10548
D#		311	622	1245	2489	4978	9956
D		294	587	1175	2349	4699	9398
C#		277	554	1109	2217	4435	8870
C		262	523	1047	2093	4186	8372

Ejercicio 18.2.

```
#define VOL 3

task main()
{
  PlayToneEx(262,400,VOL,FALSE);  Wait(500);
  PlayToneEx(294,400,VOL,FALSE);  Wait(500);
  PlayToneEx(330,400,VOL,FALSE);  Wait(500);
  PlayToneEx(294,400,VOL,FALSE);  Wait(500);
  PlayToneEx(262,1600,VOL,FALSE); Wait(2000);
}
```

Ejercicio 18.3.

Si queremos que el NXT reproduzca música mientras camina, óbviamente debemos utilizar diferentes tareas.
Aquí tenemos un sencillo ejemplo en el que el NXT se moverá hacia adelante y hacia atrás mientras reproduce música.

```
task music()
{
  while (true)
  {
    PlayTone(262,400);  Wait(500);
    PlayTone(294,400);  Wait(500);
    PlayTone(330,400);  Wait(500);
    PlayTone(294,400);  Wait(500);
  }
}
```

```
task movement()
{
  while(true)
  {
    OnFwd(OUT_AC, 75); Wait(3000);
    OnRev(OUT_AC, 75); Wait(3000);
  }
}
```

```
task main()
{
  Precedes(music, movement);
}
```

Ejercicio 19.1.

Implementar un programa que haga que nuestro robot siga con el sensor de luz un camino pintado de negro.

El robot debe recordar el sentido (izquierda o derecha) de la última curva, de manera que cuando encuentre una nueva curva compruebe que sea hacia el mismo lado que la anterior o no.

Además, al tocar cualquiera de los sensores de choque, el robot se parará, emitirá un sonido que indique parada, y quedará en este estado hasta que se vuelva a tocar alguno de los sensores de choque.
Al tocar nuevamente cualquiera de los sensores de choque, esto implicará que el robot emitirá un sonido de reinicio de marcha, siguiendo nuevamente el camino pintado de negro.

Ejercicio 19.1.

```
#define UMBRAL 40
#define POTENCIA_AVANCE 50
#define POTENCIA_GIRO 25
#define TIEMPO_GIRO 45
#define TOP_GIROS 4
#define VOL 7
#define ejecuta_izda OnRev(OUT_C,POTENCIA_GIRO); OnFwd(OUT_A,POTENCIA_GIRO); Wait(TIEMPO_GIRO);
#define ejecuta_dcha OnRev(OUT_A,POTENCIA_GIRO); OnFwd(OUT_C,POTENCIA_GIRO); Wait(TIEMPO_GIRO);
#define sonido_stop    PlayFileEx("Woops.rso",VOL,FALSE); Wait(500);
#define sonido_run    PlayFileEx("OK.rso",VOL,FALSE); Wait(500);

sub giro_izda()
{
    do
    { ejecuta_izda;
      n_giros++;
    } while ((SENSOR_3 >= UMBRAL) && (n_giros < TOP_GIROS));

    n_giros = 0;

    if (SENSOR_3 >= UMBRAL)//Entonces es un giro de derechas
    {
      do
      { ejecuta_dcha;
      } while (SENSOR_3>=UMBRAL);//Gira a la derecha hasta encontrar el camino.
      izda = 0;
    }
    else izda = 1;
}
```

Ejercicio 19.1.

```
sub giro_dcha()
{
    do
    { ejecuta_dcha;
      n_giros++;
    } while ((SENSOR_3 >= UMBRAL) && (n_giros < TOP_GIROS));

    n_giros = 0;

    if (SENSOR_3 >= UMBRAL)
    {
      do
      { ejecuta_izda;
      } while (SENSOR_3>=UMBRAL);
      izda = 1;
    }
    else izda = 0;
}
```

Ejercicio 19.1.

```
task camino()
{
  while (true)
  {
   if (SENSOR_3 >= UMBRAL)  //Si dejamos de detectar el camino negro
   {
      Acquire(moveMutex);
      Off(OUT_AC);
      if (izda == 1) giro_izda();
      else giro_dcha();
      if (SENSOR_3 < UMBRAL) OnFwd(OUT_AC,POTENCIA_AVANCE);
      else Off(OUT_AC);
      Release(moveMutex);
   }
  }
}
```

Ejercicio 19.1.

```
task choque()
{
 while(true){
    if ((SENSOR_1 ==1) || (SENSOR_2 ==1)){
      Acquire(moveMutex);
      Off(OUT_AC); Wait(500);  sonido_stop;
      until ((SENSOR_1 ==1) || (SENSOR_2 ==1));
      sonido_run;
      OnFwd(OUT_AC,POTENCIA_AVANCE); Wait(500);
      Release(moveMutex);
    }
 }
}
```

Ejercicio 19.1.

```
task main()
{
  n_giros = 0;
  izda = 1; //Presuponemos que la 1ra curva será de izda.
  SetSensorTouch(IN_1);
  SetSensorTouch(IN_2);
  SetSensorLight(IN_3);
  OnFwd(OUT_AC,POTENCIA_AVANCE);
  Precedes(camino,choque);
}
```

Ejercicio 19.2.

Implementar un programa que haga que nuestro robot siga con el sensor de luz un camino pintado de negro.

El robot debe recordar el sentido (izquierda o derecha) de la última curva, de manera que cuando encuentre una nueva curva compruebe que sea hacia el mismo lado que la anterior o no.

Al tocar el sensor de choque conectado en la entrada derecha (input1), el robot aumentará de marcha (aumento de velocidad), emitiendo un sonido que será un barrido de frecuencias creciente. Si la velocidad es máxima y se pulsa incrementar la velocidad, el robot emitirá un sonido de error.
Al tocar el sensor de choque conectado en la entrada izquierda (input2), el robot disminuirá de marcha (decremento de velocidad), emitiendo un sonido que será un barrido de frecuencias decreciente. Si la velocidad es mínima y se pulsa decrementar la velocidad, el robot emitirá un sonido de error.

El robot empezará a avanzar con una potencia del 20% de su capacidad. Los incrementos y decrementos de velocidad los hará de 20 en 20.

Ejercicio 19.2.

```
#define UMBRAL 40
#define POTENCIA_GIRO 25
#define TIEMPO_GIRO 45
#define TOP_GIROS 4
#define VOL 7
#define ejecuta_izda OnRev(OUT_C,POTENCIA_GIRO); OnFwd(OUT_A,POTENCIA_GIRO); Wait(TIEMPO_GIRO);
#define ejecuta_dcha OnRev(OUT_A,POTENCIA_GIRO); OnFwd(OUT_C,POTENCIA_GIRO); Wait(TIEMPO_GIRO);
#define sonido_error    PlayFileEx("! Attention.rso",VOL,FALSE); Wait(500);

mutex moveMutex;
int n_giros,potencia_avance,izda;

sub giro_izda()
{
    do
    { ejecuta_izda;
      n_giros++;
    } while ((SENSOR_3 >= UMBRAL) && (n_giros < TOP_GIROS));

    n_giros = 0;

    if (SENSOR_3 >= UMBRAL)//Entonces es un giro de derechas
    {
      do
      { ejecuta_dcha;
      } while (SENSOR_3>=UMBRAL);//Gira a la derecha hasta encontrar el camino.
      izda = 0;
    }
    else izda = 1;
}
```

Ejercicio 19.2.

```
sub giro_dcha()
{
    do
    { ejecuta_dcha;
      n_giros++;
    } while ((SENSOR_3 >= UMBRAL) && (n_giros < TOP_GIROS));

    n_giros = 0;

    if (SENSOR_3 >= UMBRAL)
    {
      do
      { ejecuta_izda;
      } while (SENSOR_3>=UMBRAL);
      izda = 1;
    }
    else izda = 0;
}
```

Ejercicio 19.2.

```
task camino()
{
  while (true)
  {
   if (SENSOR_3 >= UMBRAL)  //Si dejamos de detectar el camino negro
   {
      Acquire(moveMutex);
      Off(OUT_AC);
      if (izda == 1) giro_izda();
      else giro_dcha();
      if (SENSOR_3 < UMBRAL) OnFwd(OUT_AC,potencia_avance);
      else Off(OUT_AC);
      Release(moveMutex);
   }
  }
}
```

Ejercicio 19.2.

```
//Realiza los cambios de marchas si se toca alguno de los sensores de choque.
task choque()
{
  while (true)
  {
    if (SENSOR_1 ==1) {//Aumentar marchas
       if (potencia_avance < 100)
       { potencia_avance=potencia_avance+20;
         Acquire(moveMutex);
         OnFwd(OUT_AC,potencia_avance);
         Release(moveMutex);
         PlayToneEx(262,400,VOL,FALSE);
         Wait(500);
         PlayToneEx(330,400,VOL,FALSE);
       }
       else sonido_error;
    }
    if (SENSOR_2 ==1) {// Disminuir marchas
       if (potencia_avance > 0)
       { potencia_avance=potencia_avance-20;
         Acquire(moveMutex);
         OnFwd(OUT_AC,potencia_avance);
         Release(moveMutex);
         PlayToneEx(330,400,VOL,FALSE);
         Wait(500);
         PlayToneEx(262,400,VOL,FALSE);
       }
       else sonido_error;
    }
  }
}
```

Ejercicio 19.2.

```
task main()
{
  n_giros = 0;
  izda = 1;
  potencia_avance=20;
  SetSensorTouch(IN_1);
  SetSensorTouch(IN_2);
  SetSensorLight(IN_3);
  OnFwd(OUT_AC,potencia_avance);
  Precedes(camino,choque);
}
```

Ejercicio 20.1.

Implementar un programa que haga que nuestro robot siga con el sensor de luz un camino pintado de negro.

El robot debe recordar el sentido (izquierda o derecha) de la última curva, de manera que cuando encuentre una nueva curva compruebe que sea hacia el mismo lado que la anterior o no.

Al tocar el sensor de choque conectado en la entrada derecha, el robot girará hacia ese lado emitiendo un sonido. Realizará el giro hasta que encuentre el camino negro.

Al tocar el sensor de choque conectado en la entrada izquierda, el robot girará hacia ese lado emitiendo un sonido. Realizará el giro hasta que encuentre el camino negro.

En ambos casos, una vez hecho el giro y encontrado el camino negro, el robot volverá a caminar hacia adelante siguiendo el camino negro hasta que se vuelva a chocar con alguno de los sensores de choque.

Ejercicio 20.1.

```
#define UMBRAL 40
#define POTENCIA_AVANCE 50
#define POTENCIA_GIRO 25
#define TIEMPO_GIRO 45
#define TOP_GIROS 4
#define VOL 7
#define ejecuta_izda OnRev(OUT_C,POTENCIA_GIRO); OnFwd(OUT_A,POTENCIA_GIRO); Wait(TIEMPO_GIRO);
#define ejecuta_dcha OnRev(OUT_A,POTENCIA_GIRO); OnFwd(OUT_C,POTENCIA_GIRO); Wait(TIEMPO_GIRO);
#define sonido    PlayFileEx("! Attention.rso",VOL,FALSE); Wait(500);

mutex moveMutex;
int n_giros,izda;

sub giro_izda()
{
    do
    { ejecuta_izda;
      n_giros++;
    } while ((SENSOR_3 >= UMBRAL) && (n_giros < TOP_GIROS));

    n_giros = 0;

    if (SENSOR_3 >= UMBRAL)//Entonces es un giro de derechas
    {
      do
      { ejecuta_dcha;
      } while (SENSOR_3>=UMBRAL);//Gira a la derecha hasta encontrar el camino.
      izda = 0;
    }
    else izda = 1;
}
```

Ejercicio 20.1.

```
sub giro_dcha()
{
    do
    { ejecuta_dcha;
      n_giros++;
    } while ((SENSOR_3 >= UMBRAL) && (n_giros < TOP_GIROS));

    n_giros = 0;

    if (SENSOR_3 >= UMBRAL)
    {
      do
      { ejecuta_izda;
      } while (SENSOR_3>=UMBRAL);
      izda = 1;
    }
    else izda = 0;
}
```

Ejercicio 20.1.

```
task camino()
{
  while (true)
  {
   if (SENSOR_3 >= UMBRAL)  //Si dejamos de detectar el camino negro
   {
      Acquire(moveMutex);
      Off(OUT_AC);
      if (izda == 1) giro_izda();
      else giro_dcha();
      if (SENSOR_3 < UMBRAL) OnFwd(OUT_AC,POTENCIA_AVANCE);
      else Off(OUT_AC);
      Release(moveMutex);
   }
  }
}
```

Ejercicio 20.1.

```
task choque()
{
  while (true)
  {
    if (SENSOR_1 ==1) //Sensor Dcho.
    {
      Acquire(moveMutex);
      ejecuta_dcha; sonido;
      until (SENSOR_3 < UMBRAL);
      OnFwd(OUT_AC,POTENCIA_AVANCE);
      Release(moveMutex);
    }
    else {
      if (SENSOR_2 ==1) //Sensor Izdo.
      {
        Acquire(moveMutex);
        ejecuta_izda; sonido;
        until (SENSOR_3 < UMBRAL);
        OnFwd(OUT_AC,POTENCIA_AVANCE);
        Release(moveMutex);
      }
    }
  }
}
```

Ejercicio 20.1.

```
task main()
{
  n_giros = 0;
  izda = 1;
  SetSensorTouch(IN_1);
  SetSensorTouch(IN_2);
  SetSensorLight(IN_3);
  OnFwd(OUT_AC,POTENCIA_AVANCE);
  Precedes(camino,choque);
}
```

Ejercicio 20.2.

Implementar un programa que haga que nuestro robot siga con el sensor de luz un camino pintado de negro.

El robot empezará a andar cuando le demos la orden con nuestra voz (cuando escuche un sonido por encima de un valor umbral). Si está andando y escucha una orden de parada se detendrá (al escuchar un sonido por encima de nuestro valor umbral). Este proceso lo realizará infinitamente.

Mientras está siguiendo el camino pintado de color negro, si choca con el sensor conectado a input1, el robot emitirá un sonido y girará hacia la derecha. Realizará el giro hasta que encuentre el camino negro nuevamente.

Si con el sensor de ultrasonidos conectado a input 4 detecta un obstáculo dentro de una distancia inferior a 10cm, el robot se detendrá hasta que el obstáculo vuelva a estar a una distancia mayor o igual a 10cm, reanudando en tal caso la marcha.

El robot estará ejecutando todas estas tareas infinitamente.

Ejercicio 20.2.

```
#define REFLEJO 40
#define ESTA_CERCA 10
#define VOZ 70
#define POTENCIA_AVANCE 50
#define POTENCIA_GIRO 25
#define TIEMPO_GIRO 45
#define TOP_GIROS 4
#define VOL 7
#define ejecuta_izda OnRev(OUT_C,POTENCIA_GIRO); OnFwd(OUT_A,POTENCIA_GIRO); Wait(TIEMPO_GIRO);
#define ejecuta_dcha OnRev(OUT_A,POTENCIA_GIRO); OnFwd(OUT_C,POTENCIA_GIRO); Wait(TIEMPO_GIRO);
#define sonido    PlayFileEx("! Attention.rso",VOL,FALSE); Wait(500);

mutex moveMutex;
int n_giros,izda;

sub giro_izda()
{
    do
    { ejecuta_izda;
      n_giros++;
    } while ((SENSOR_3 >= REFLEJO) && (n_giros < TOP_GIROS));

    n_giros = 0;

    if (SENSOR_3 >= REFLEJO)//Entonces es un giro de derechas
    {
      do
      { ejecuta_dcha;
      } while (SENSOR_3>=REFLEJO);//Gira a la derecha hasta encontrar el camino.
      izda = 0;
    }
    else izda = 1;
}
```

Ejercicio 20.2.

```
sub giro_dcha()
{
    do
    { ejecuta_dcha;
      n_giros++;
    } while ((SENSOR_3 >= REFLEJO) && (n_giros < TOP_GIROS));

    n_giros = 0;

    if (SENSOR_3 >= REFLEJO)
    {
      do
      { ejecuta_izda;
      } while (SENSOR_3>=REFLEJO);
      izda = 1;
    }
    else izda = 0;
}
```

Ejercicio 20.2.

```
task camino()
{
  while (true)
  {
   if (SENSOR_3 >= REFLEJO)  //Si dejamos de detectar el camino negro
   {
      Acquire(moveMutex);
      Off(OUT_AC);
      if (izda == 1) giro_izda();
      else giro_dcha();
      if (SENSOR_3 < REFLEJO) OnFwd(OUT_AC,POTENCIA_AVANCE);
      else Off(OUT_AC);
      Release(moveMutex);
   }
  }
}
```

Ejercicio 20.2.

```
task choque()
{
  while (true)
  {
    if (SENSOR_1 ==1) //Sensor Dcho.
    {
      Acquire(moveMutex);
      ejecuta_dcha; sonido;
      until (SENSOR_3 < REFLEJO);
      OnFwd(OUT_AC,POTENCIA_AVANCE);
      Release(moveMutex);
    }
  }
}
```

Ejercicio 20.2.

```
task ver_si_cerca()
{
  while(true){
    if (SensorUS(IN_4)<ESTA_CERCA) {
      Acquire(moveMutex);
      Off(OUT_AC);
      until (SensorUS(IN_4)>=ESTA_CERCA);
      OnFwd(OUT_AC,POTENCIA_AVANCE);
      Release(moveMutex);
    }
  }
}
```

Ejercicio 20.2.

```
task para_arranca()
{
    while(true)
    {
      if (SENSOR_2 >= VOZ){
        Acquire(moveMutex);
        Off(OUT_AC); Wait(500);
        until (SENSOR_2 >= VOZ);
        OnFwd(OUT_AC,POTENCIA_AVANCE);
        Wait(500);
        Release(moveMutex);
      }
    }
}
```

Ejercicio 20.2.

```
task main()
{
  n_giros = 0;
  izda = 1; //Presuponemos que la 1ra curva será de izda.
  SetSensorTouch(IN_1); //Sensor de choque dcho.
  SetSensorSound(IN_2); //Sensor de sonido en la input 2.
  SetSensorLight(IN_3); //Sensor de luz en la input 3.
  SetSensorLowspeed(IN_4);  //Sensor de ultrasonido en la input 4.
  until (SENSOR_2 >= VOZ);
  OnFwd(OUT_AC,POTENCIA_AVANCE); Wait (500);
  Precedes(para_arranca,camino,choque,ver_si_cerca);
}
```