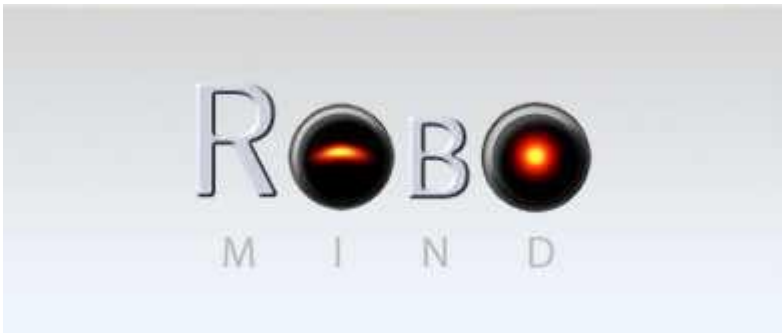
COLEGIO AGUSTINIANO CIUDAD SALITRE
AREA DE TECNOLOGIA E INFORMATICA
GRADO SEGUNDO

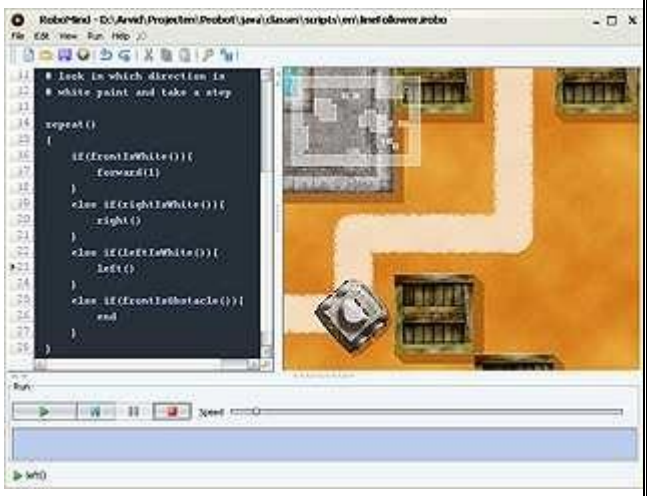
INTRODUCCIÓN A LA PROGRAMACIÓN.



1. LENGUAJE DE PROGRAMACIÓN ROBOMIND.

Para facilitar la comprensión de las técnicas y mecanismos de programación, en este curso se estudiará el lenguaje ROBOMIND. Robomind es un lenguaje que permite programar robots móviles. Se trata de un lenguaje muy sencillo, con un juego de instrucciones muy simple, pero que empela las mismas estructuras de programación que cualquier otro lenguaje de programación. Por todo ello es un lenguaje muy adecuado para introducir a los estudiantes de grado segundo en el complejo mundo de la programación.

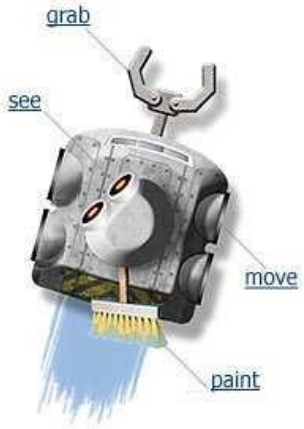
Como ya se ha dicho, Robomind es un lenguaje que permite controlar robots móviles. En nuestro caso, no dispondremos de robots "reales" cuyo funcionamiento controlar. Sin embargo, el entorno de programación Robomind ofrece un "robot móvil virtual" que simula el comportamiento de un robot real en la pantalla del ordenador. Nuestra tarea será programar dicho robot virtual para controlar su funcionamiento.



Como introducción, se pueden ver unos videos que describen las posibilidades básicas de RoboMind:

http://www.robomind.net/en/demo.html
http://www.robomind.net/en/demoNewInTwo.html

2. EL ROBOT VIRTUAL ROBOMIND.

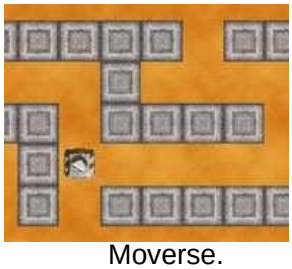


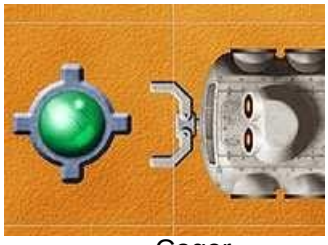
A continuación se muestra el robot que se programará en RoboMind. Se trata de un robot móvil equipado con los varios dispositivos que le permiten moverse, mirar alrededor, coger objetos, y pintar.

- Sensores: para captar información del exterior, el robot dispone de uan videocámara, que usará como sensor de sensor de contacto o de presencia, y como sensor de color.

- Actuadores: los actuadores permiten al robot realizar acciones.
  - Motores y ruedas: permiten al robot moverse
  - Brazo: permite al robot recoger objetos (balizas = beacons).
  - Brocha: permite al robot dibujar en color blanco o negro.


Mirar.


Moverse.


Coger.


Pintar.

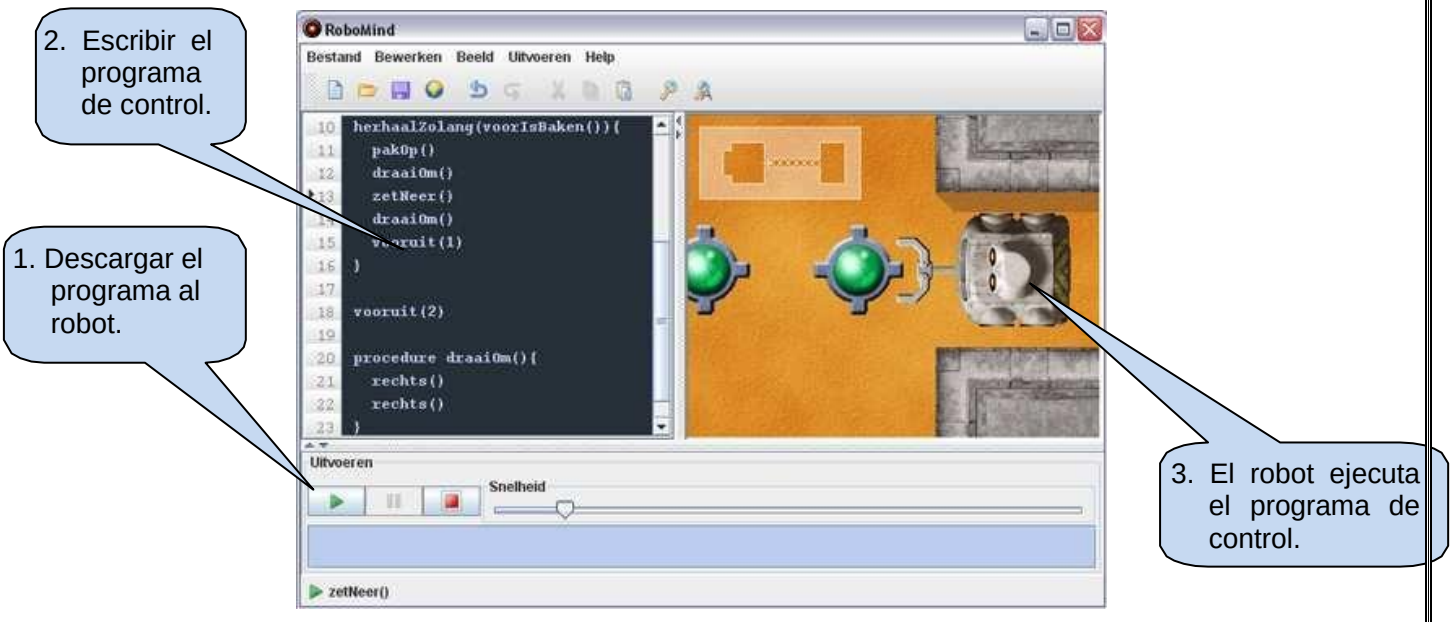
El programa de control se encargará de definir el comportamiento del robot. El programa de control deberá leer la información que los sensores capten del entorno (fase de entrada), interpretar y manipular dicha información (fase de proceso), y modificar el comportamiento de los actuadores en función de las decisiones tomadas al procesar los datos captados (fase de salida).

3. EL ENTORNO DE TRABAJO ROBOMIND.

Para programar el robot móvil virtual de RoboMind se utiliza un sencillo lenguaje de programación, que servirá de aprendizaje a las técnicas de programación.

¿Cómo programar el robot RoboMind? La secuencia de trabajo es siempre la misma:

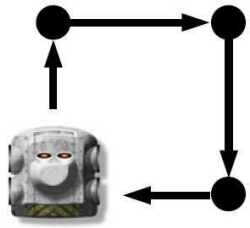
1) Escribir el programa de control.
2) Descargar el programa de control al robot, para definir su comportamiento.
3) El robot ejecuta el programa de control.



Ejemplo:

El siguiente programa de control hace que el robot realice un recorrido con forma de cuadrado. Escríbelo en el área de programación, descarga el programa, y observa el resultado.

```
forward(1)
right()
forward(1)
right()
forward(1)
right()
forward(1)
right()
```



4. EL LENGUAJE DE PROGRAMACIÓN ROBOMIND.

Para hacer que una máquina, un ordenador o un robot funcionen, hay que darle órdenes o ***instrucciones***.

Ejemplos:

- Para un video: play, pause, record, fast forward (FF), etc.
- Para un procesador de textos (Word): Poner texto en negrita, insertar imagen, copiar, pegar, etc.

Pues bien, para programar el robot virtual Robomind también se dispone de un conjunto de instrucciones que permiten gobernarlo. Estas instrucciones se le proporcionarán una tras otra, hasta escribir el programa de control con la secuencia de órdenes que se desea que el robot realice.

El lenguaje de programación RoboMind incorpora el conjunto de instrucciones y estructuras de programación que permitirán escribir los programas de control para dirigir el funcionamiento del robot.

A continuación, se revisarán paso a paso el conjunto de instrucciones y estructuras de programación disponibles en RoboMind.

5.1.- INSTRUCCIONES DE MOVIMIENTO.

Las instrucciones de movimiento controlan los motores del robot, y por tanto, el movimiento del motor.

MOVIMIENTO		
forward(n)		Avanzar n pasos.
backward(n)		Retroceder n pasos.
left()		Girar 90º a la izquierda.
right()		Girar 90º a la derecha.
north(n)		Orientarse al norte y avanzar n pasos.
south(n)		Orientarse al sur y avanzar n pasos.
east(n)		Orientarse al este y avanzar n pasos.
west(n)		Orientarse al oeste y avanzar n pasos.

Actividades “movimiento del robot”.

IMPORTANTE:
Antes de empezar, acude a la carpeta:
C:\Archivos de programa\RoboMind\scripts\en

Borra todos los archivos que encuentres en dicha carpeta.

Programa 1) Programa al robot para que llegue a la casilla a la izquierda de la baliza del sur. Al llegar a dicha posición, el robot se para. Guarda el programa en tu carpeta de trabajo como ***prog1.irobo***.

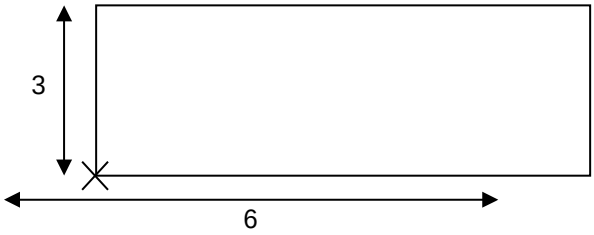

maps.rar


Scripts.rar

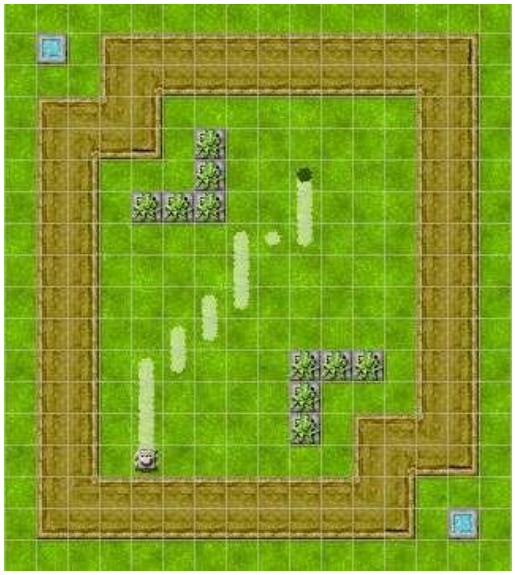
Programa 2) Programa al robot para llevarlo hasta la casilla frente la baliza del nordeste. Al llegar a dicha posición, el robot se para. Guarda el programa en tu carpeta de trabajo como ***prog2.irobo***.

Programa 3) Programa al robot para que trace un recorrido en forma de rectángulo, de altura 3 unidades y anchura 6 unidades. El robot termina de trazar el rectángulo en su posición inicial, donde se para. Guarda el programa en tu carpeta de trabajo como ***prog3.irobo***.

X 👁 posición inicial.



Programa 4) Escribe un programa que lleve al robot al sendero blanco y lo fuerce a recorrerlo. El robot se parará en la última casilla del sendero blanco. Guarda el programa en tu carpeta de trabajo como ***prog4.irobo***.



Programa 5) Abre el mapa “verticalLines1.map” (File Open Map). Escribe un programa que lleve al robot al punto negro. El robot sólo puede pisar en las casillas blancas. Guarda el programa en tu carpeta de trabajo como ***prog5.irobo***.

5.2.- INSTRUCCIONES PARA PINTAR Y RECOGER OBJETOS.

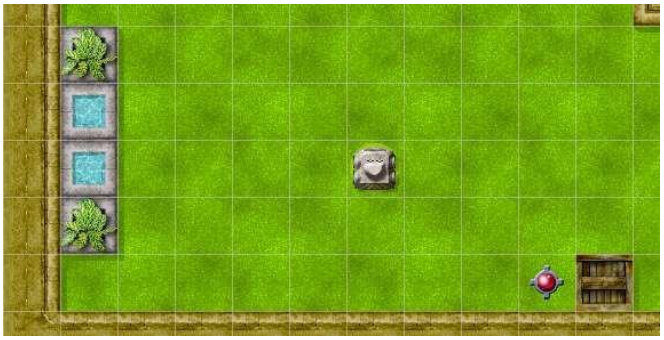
Estas instrucciones controlan la brocha y el brazo del robot. Permiten pintar líneas y recoger objetos (balizas).

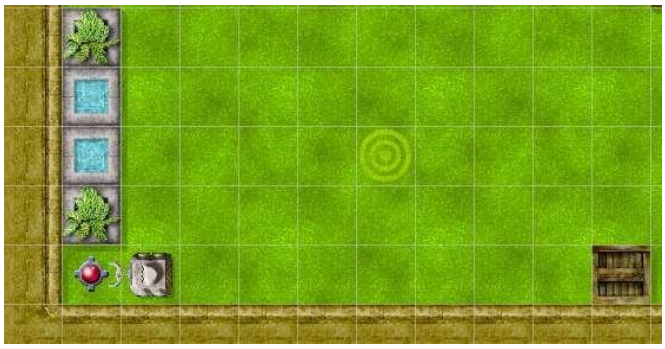
PINTAR		
paintWhite()		Bajar la brocha con pintura blanca al suelo para pintar en blanco.
paintBlack()		Bajar la brocha con pintura negra al suelo para pintar en negro.
stopPainting()		Dejar de pintar. Esconder la brocha.

COGER		
pickUp()		Coger la baliza situada frente al robot.
putDown()		Soltar la baliza y dejarla frente al robot.

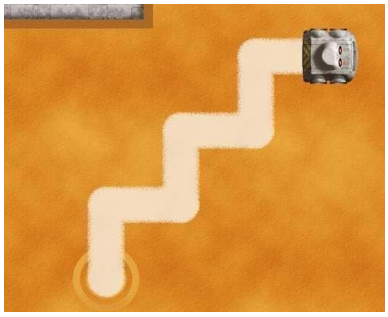
Actividades “pintar y recoger”.

Programa 6) Abre el mapa “default.map”. Escribe un programa para mover la baliza localizada al sudeste del robot a la esquina sudoeste del mapa. Guarda el programa en tu carpeta de trabajo como ***prog6.irobo***.





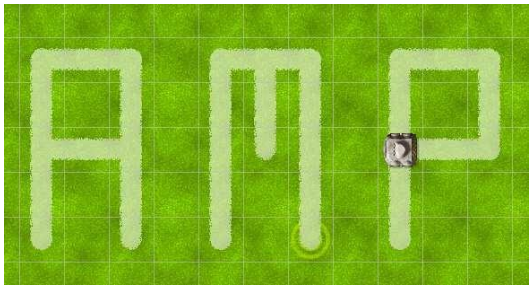
Programa 7) Escribe un programa para que el robot dibuje una escalera blanca con 3 escalones, como la mostrada en la imagen inferior. Guarda el programa en tu carpeta de trabajo como ***prog7.irobo***.



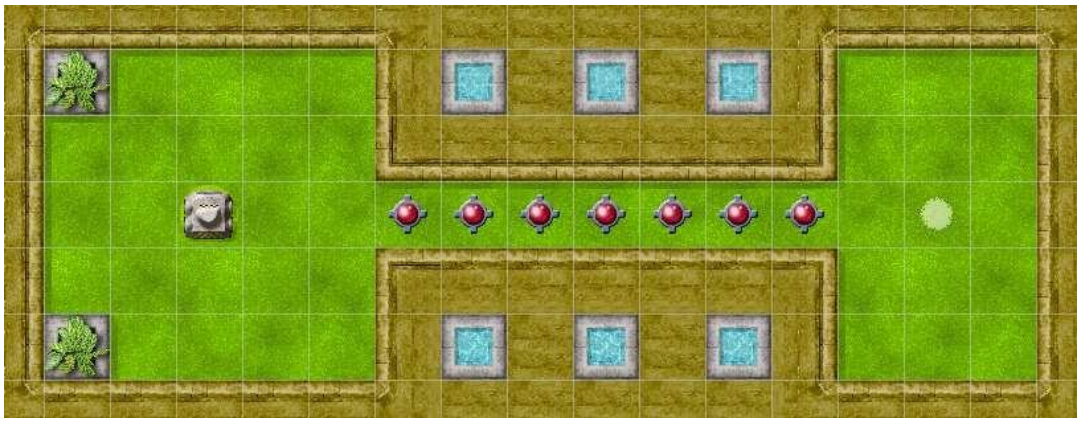
Programa 8) Abre el mapa “goRightAtWhite.map”.
Escribe un programa mediante el cual el robot rodee la caja de madera situada a su derecha, mientras va pintando el contorno de negro. Después, el robot vuelve al punto inicial. Guarda el programa en tu carpeta de trabajo como ***Prog8.irobo***.

Programa 9) Abre el mapa “openArea.map”.
Escribe un programa para que el robot escriba las iniciales de tu nombre y apellidos (3 letras).

Debes respetar los espacios en blanco entre cada inicial. Guarda el programa en tu carpeta de trabajo como ***prog9.irobo***.



Programa 10) Abre al mapa “passBeacons.map”. Escribe un programa para conseguir que el robot llegue hasta el punto blanco. Para ello, el robot deberá coger y retirar las balizas del pasillo que lleva al objetivo. Guarda el programa en tu carpeta de trabajo como ***prog10.irobo***.



5.3.- INSTRUCCIONES DE VISIÓN Y DE ALEATORIEDAD.

INSTRUCCIONES DE VISIÓN.

Las instrucciones de visión controlan las videocámaras con las que el robot virtual es capaz de “ver”. Las videocámaras actúan como sensores, que permitirán al robot detectar la presencia de obstáculos (paredes, cajas, etc.), detectar la presencia de balizas (objetos que puede recoger), y detectar colores en el suelo.

VISIÓN		
IZQUIERDA	EN FRENTE	DERECHA
leftIsObstacle()	frontIsObstacle()	rightIsObstacle()
leftIsClear()	frontIsClear()	rightIsClear()
leftIsBeacon()	frontIsBeacon()	rigthIsBeacon()
leftIsWhite()	frontIsWhite()	rightIsWhite()
leftIsBlack()	fontIsBlack()	rightIsBlack()

Ejemplo 1: leftIsObstacle ⓞ permite determinar si a la izquierda hay un obstáculo.
Ejemplo 2: rightIsBlack ⓞ permite determinar a la derecha hay una casilla pintada de negro.
Ejemplo 3: frontIsClear ⓞ permite determinar si al frente está despejado de obstáculos (clear = despejado).
Ejemplo 4: leftIsBeacon ⓞ permite determinar si en la casilla de la izquierda hay una baliza (beacon = baliza).

INSTRUCCIONES DE ALEATORIEDAD.

La instrucción de aleatoriedad permite al robot lanzar una moneda al aire para realizar una elección aleatoria (ir a la derecha o a la izquierda, al este o al oeste, pintar de negro o de blanco, etc.).

TOMA DE DECISIONES ALEATORIAS	
flipCoin()	Lanzar una moneda al aire para tomar una decisión aleatoria. El resultado puede ser cara (TRUE) o cruz (FALSE) con una probabilidad del 50%.

5.4.- ESTRUCTURAS DE PROGRAMACIÓN (1).

Además de las instrucciones, el lenguaje RoboMind proporciona ciertas estructuras de programación que permiten un mayor control sobre el robot virtual. Estas estructuras permiten ejecutar un conjunto de instrucciones varias veces (bucles o estructuras *repeat*), o ejecutar un conjunto de instrucciones sólo si se cumple una condición (condicionales o estructuras *if-else*).

5.4.1. - BUCLES (REPEAT y REPEAT-WHILE).

repeat (n) { instrucciones }

Esta estructura permite repetir la ejecución de las instrucciones entre llaves un número '*n*' de veces.
NOTA: Si se omite el parámetro n, se repetirán las instrucciones entre llaves indefinidamente 👁 por tanto, si se quiere que una serie de instrucciones se estén ejecutando siempre hay que usar *repeat()*.

Ejemplo: escribe y ejecuta el siguiente programa:

a) ¿Para qué sirve?

b) Explica su funcionamiento.

```
1  # ¿Qué hace el programa?
2
3  repeat (4)
4  {
5      forward(2)
6      backward(2)
7  }
8
```

repeatWhile (condición) { instrucciones }

Esta estructura repite (repeat) la ejecución de las instrucciones entre llaves mientras (while) la condición que evalúa sea verdadera. Si la condición no se hace verdadera, o deja de ser verdadera, continúa ejecutando la instrucción tras la llave que cierra el bucle.

NOTA: Las condiciones que utiliza el estructura repeatWhile suelen ser instrucciones de visión (leftIsObstacle(), frontIsBeacon(), rightIsClear(), etc.).

Ejemplo: escribe y ejecuta el siguiente programa:

a) ¿Para qué sirve?

b) Explica su funcionamiento.

```
1  # ¿Qué hace el programa?
2
3  repeatWhile ( frontIsClear() )
4  {
5      forward(1)
6  }
7
```

break

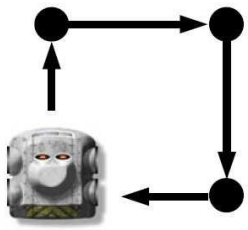
Esta instrucción termina con la ejecución del bucle ("rompe" el bucle), y continua ejecutando la instrucción que sigue a la llave que cierra el bucle. Sirve para forzar la salida de un bucle.

De momento no usaremos esta instrucción, pero será muy útil en cuanto veamos las estructuras condicionales.

Cuestiones "bucles".

Programa 11) Reescribe el programa que hace que el robot trace un cuadrado, pero utilizando un bucle. Guarda el programa en tu carpeta de trabajo como ***prog11.irobo***.

```
forward(1)
right()
forward(1)
right()
forward(1)
right()
forward(1)
right()
```



Programa 11b) Programa ahora al robot para que permanezca trazando cuadrados de forma indefinida. Necesitarás un bucle infinito repeat().Guarda el programa en tu carpeta de trabajo como ***prog11b.irobo***.

Programa 12) Buscando el límite: haz que el robot avance hasta que encuentre un obstáculo. Cuando lo encuentre, debe retroceder una posición. Ejecuta el mismo programa en varios mapas, debe funcionar cualquiera que sea el entorno (File Open map). Guarda el programa en tu carpeta de trabajo como ***prog12.irobo***.

Programa 13) Haz un programa que, utilizando un bucle, haga que el robot dibuje una escalera de 4 escalones. Ayúdate del esquema que se te proporciona a continuación para escribir tu programa. Guarda el programa en tu carpeta de trabajo como ***prog13.irobo***.

```
1  # empezar a pintar
2  repeat(4)
3  {
4      # dibujar un sólo escalón
5  }
6  # dejar de pintar
7
```

Programa 14) Vuelve a realizar el programa 10. En este caso, el proceso de coger y retirar cada baliza del camino lo realizarás mediante un bucle. Guarda el programa en tu carpeta de trabajo como ***prog14.irobo***.

Programa 15) Abre el mapa copyLine1.map. Crea un programa que haga que el robot camine paralelo a la línea negra que tiene a su izquierda. El robot debe avanzar paralelo a la línea negra mientras haya línea negra. Guarda el programa en tu carpeta de trabajo como ***prog15.irobo***.

Programa 15b) Robot bordeador: programa al robot para que llegue a la caja, y una vez allí se ponga a bordear la caja de forma indefinida. Guarda el programa en tu carpeta de trabajo como ***prog15b.irobo***.

Programa 15c) Robot bailón. Mediante un bucle que se repita de forma indefinida, programa al robot virtual para que "baile" (bailar será moverse girar la cabeza). Guarda el programa en tu carpeta de trabajo como ***prog15c.irobo***.
Pista: Usa leftIsClear() y rightIsClear() para mover la cabeza del robot, y simular que baila.

5.4.2.- SENTENCIAS CONDICIONALES (IF - ELSE).

Las estructuras condicionales permiten condicionar la ejecución de ciertas instrucciones al cumplimiento de una condición. Con ello se puede hacer que ciertas instrucciones no se ejecuten siempre, sino sólo en caso de que se den ciertas circunstancias.

if (condición) { instrucciones }

Ejecutará las condiciones entre llaves, únicamente si la condición se cumple, en caso contrario ejecuta la instrucción que sigue a la llave que cierra la sentencia if.

Ejemplo: escribe y ejecuta el siguiente programa:

a) ¿Para qué sirve?

b) Explica su funcionamiento.

```
1  repeat()
2  {
3      forward(1)
4      if (frontIsBeacon())
5      {
6          pickUp()
7      }
8  }
9
```

if (condición) { instrucciones }
else { instrucciones }

Si la condición se cumple, se ejecutan las instrucciones del bloque ***if***. En cambio, si la condición no se cumple se ejecutarán las instrucciones pertenecientes al bloque ***else***. (If 👁 si (condicional); else 👁 si no)

Es decir:
Si (ocurre esta condición) { haz estas instrucciones }
Si no { haz estas otras instrucciones }

Ejemplo: escribe y ejecuta el siguiente programa:

a) ¿Para qué sirve?

b) Explica su funcionamiento.

```
1  repeat()
2  {
3      if (frontIsObstacle())
4          {right()}
5      else
6          {forward(1)}
7  }
8
```

end

La instrucción end (fin) fuerza el final del programa. Cuando el robot ejecuta esta instrucción se termina el programa.

Actividades "sentencias condicionales".

 Programa 15d) robot borracho. Abre el mapa openArea.map. Crea un programa para que el robot avance de forma indefinida (repeat()), cambiando de dirección constantemente y de forma aleatoria.

Para cambiar de dirección aleatoriamente el robot deberá decidir al azar si gira a derechas (right()) o a izquierdas (left()) cada vez que avanza, mediante la función **flipCoin().** Esto se consigue de esta forma:

```
if (flipCoin())
        {haz esto}

else
        {haz esto otro}
```

Si (sale cara)
        {haz esto}

Si no –si sale cruz–
        {haz esto otro}

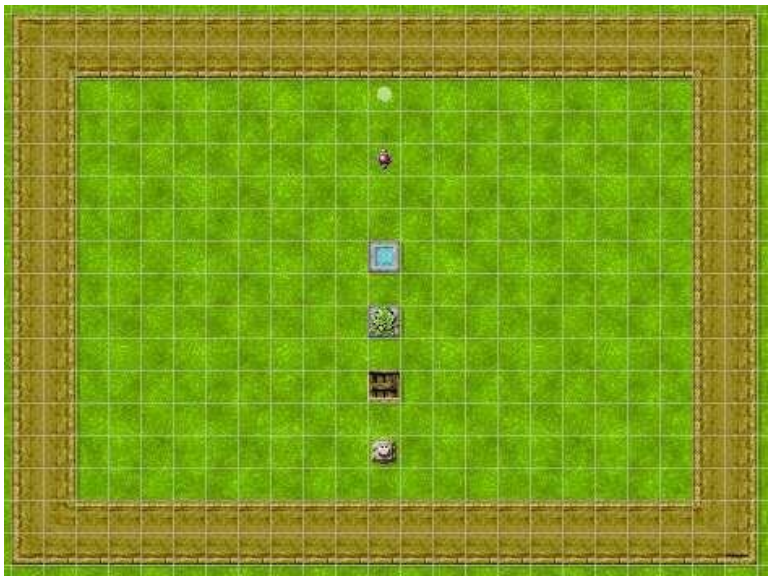
Guarda el programa en tu carpeta de trabajo como ***prog15d.irobo***.
NOTA: Este programa es para practicar el uso de la instrucción flipCoin(). Este concepto debe quedar claro.

Programa 16) Seguir las marcas blancas.
En el mapa goRightAtWhite1.map encontrarás una serie de marcas blancas.

Escribe un programa que haga al robot ir de una a otra marca. Para ello debes hacer avanzar paso a paso al robot hasta que encuentre una marca blanca al frente, y al encontrarla debe dirigirse hacia la siguiente. El recorrido terminará recogiendo la baliza del final del circuito.
Escribe el programa que hace esta tarea, asegurándote que funciona para los mapas goRightAtWhite1, goRightAtWhite2 y goRightAtWhite3. Guarda el programa en tu carpeta de trabajo como ***prog16.irobo***.

Programa 17) Robot copión: en el mapa copyLine1.map hay una línea negra a la izquierda del robot. El objetivo de este programa copiar la línea negra, dibujando una línea blanca de igual longitud a la derecha del robot. Haz el programa de forma que se ejecute correctamente incluso si no sabes el tamaño de la línea negra a priori. Guarda el programa en tu carpeta de trabajo como ***prog17.irobo***.

Programa 18) Esquivar objetos aislados: Abre el nada avoidObstacles.map. Escribe un programa para que el robot avance esquivando los objetos que tiene delante, hasta llegar al punto blanco (meta). El programa debe funcionar para los mapas avoidObstacles, avoidObstacles1 y avoidObstacles2. Guarda el programa en tu carpeta de trabajo como ***prog18.irobo***.



Programa 19) Esquivar objetos aislados y retirar balizas: modifica el programa anterior para que si el robot encuentra una baliza, en vez de esquivarla, la recoja y la deje a un lado. Guarda el programa en tu carpeta de trabajo como ***prog19.irobo***.

Programa 20) Esquivar objetos continuados: Basándote en el ejercicio 18 programa al robot para que sea capaz de sortear objetos continuados (uno a continuación de otro). Deberás utilizar la visión lateral para ver cuándo finaliza el obstáculo. Guarda el programa en tu carpeta de trabajo como ***prog20.irobo***. Mapa: avoidContinuousObstacles.map.

Programa 21) Aparcando. Abre el mapa findSpot1.map. Programa al robot para que “aparque” en el hueco marcado con una señal blanca. Guarda el programa en tu carpeta de trabajo como ***prog21.irobo***. El mismo programa también debe funcionar para el mapa findSpot1.map.

Programa 22) Robot evita-obstáculos. Programa un robot móvil que avance por el mapa de forma autónoma. Cuando en su avance detecte la presencia de un obstáculo (muro, caja, baliza, planta, agua, etc.), debe evitarlo cambiando de dirección aleatoriamente (flipCoin()).Guarda el programa en tu carpeta de

trabajo como ***prog22.irobo***

Programa 22b) Marcando colisiones: crea un programa que haga que el robot avance. Si el robot detecta un obstáculo, marca la colisión con un punto negro, y gira aleatoriamente para evitar el obstáculo. Guarda el programa en tu carpeta de trabajo como ***prog22b.irobo***. Usa el mapa default.map.

Programa 23) Busca-balizas: programa al robot para buscar balizas. Para ello debe recorrer autónomamente y de forma aleatoria el mapa default.map. Cuando encuentra una baliza, debe cogerla y detenerse. Guarda el programa en tu carpeta de trabajo como ***prog23.irobo***

5.4.3.- EXPRESIONES LÓGICAS. OPERADORES LÓGICOS.

Las condiciones a evaluar en las sentencias if y repeatWhile se denominan expresiones lógicas. Tal expresión será un valor verdadero si se cumple, o falso si no se cumple (TRUE o FALSE).

Una expresión lógica puede ser simple o compuesta:

- Expresión lógica simple:

```
1  repeatWhile(frontIsClear())
2  {
3      forward(1)
4  }
5
```

Ejemplo: frontIsClear().

El robot avanzará mientas al frente esté despejado de obstáculos.

- Expresión lógica compuesta: está formada por varias expresiones lógicas simples, unidas por los operadores lógicos NOT, AND, OR.

```
1  repeatWhile(frontIsClear() and leftIsClear())
2  {
3      forward(1)
4  }
5
```

Ejemplo: frontIsClear() and leftIsClear()

El robot avanzará mientras al frente esté despejado y a la izquierda también esté despejado (y = and)

OPERADORES LÓGICOS			
OPERADOR	NOTACIÓN ALTERNATIVA	Nº DE EXP. LÓGICAS SIMPLES QUE OPERA	EXPLICACIÓN
not	~ (ALT + 126)	1	NO: Niega el valor de la expresión lógica que le sigue.
and	**&**	2	Y: Resulta verdadero si las dos expresiones lógicas lo son, y falso si alguna de las dos o ambas son falsas.
or	**|**	2	O: Resulta verdadero si una de las dos expresiones lógicas lo es, y falso sólo si ambas son falsas.

NOT

Exp	NOT (exp)
TRUE	FALSE
FALSE	TRUE

AND

Exp1	Exp1	Exp1 AND Exp2
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TUE	TRUE

OR

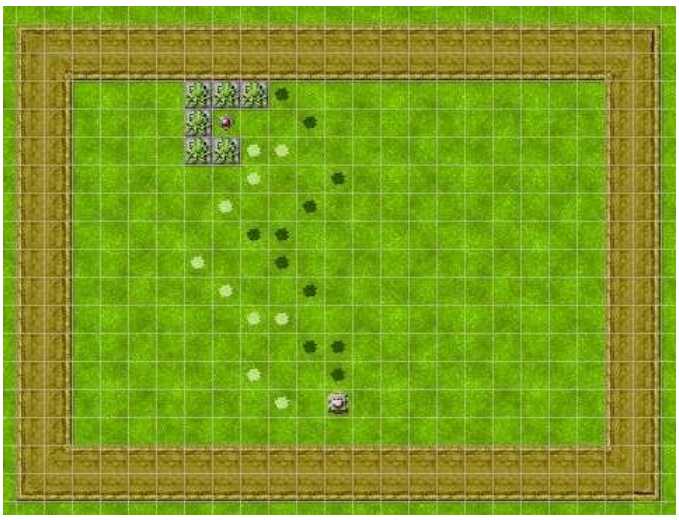
Exp1	Exp1	Exp1 OR Exp2
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TUE	TRUE

Ejemplo1: not frontIsWhite() 👁 será verdadera si al frente NO se detecta color blanco.
Ejemplo2: frontIsClear() and leftIsBlack() 👁 será verdadera si al frente está despejado Y a la izquierda se detecta negro (es decir, será verdadera sólo si se cumplen ambas expresiones simples)
Ejemplo3: frontIsWhite or frontIsBlack() 👁 será verdadera si al frente se detecta blanco O se detecta negro (es decir, será verdadera sólo con que se cumpla una de las expresiones)

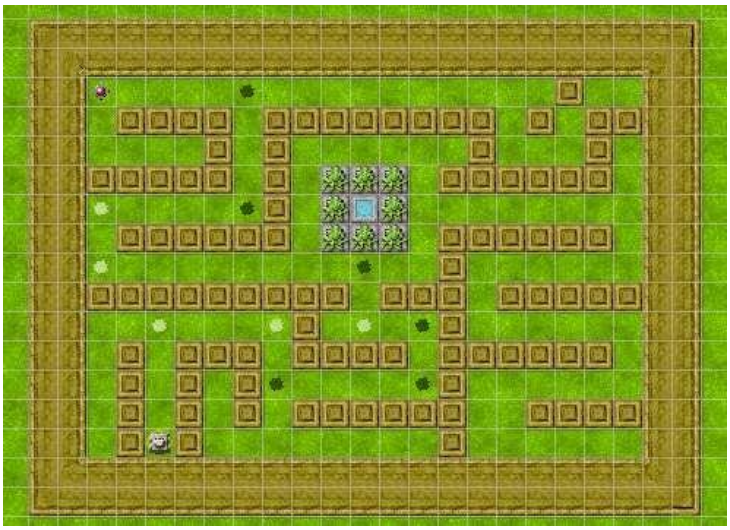
Actividades finales.

Programa 24) Enjaulado. Abre el mapa roboCage.map. Verás que el robot se encuentra dentro de un recinto limitado por una línea negra. Programa al robot para que se mueva aleatoriamente dentro de este recinto, pero sin poder abandonarlo (el robot se moverá encerrado en el recinto de la línea negra). Guarda el programa en tu carpeta de trabajo como ***prog24.irobo.***

Programa 25) Código de colores. Abre el mapa colorCode.map. Los puntos blancos simbolizan girar a la derecha y avanzar, mientras que los negros girar a la izquierda y avanzar. Dota de la inteligencia necesaria a tu robot para que encuentre y coja la baliza escondida siguiendo las pistas situadas en el suelo. Guarda el programa en tu carpeta de trabajo como ***prog26.irobo.***



Programa 26) Código de circulación. Abre el mapa drivingCode.map.



Haz que tu robot sigua un camino trazado en el suelo basándose en el siguiente código de circulación:

a) Un punto blanco significará que en la siguiente bifurcación se tome el camino de la derecha.
b) Un punto negro significará que en la siguiente bifurcación se tome el camino de la izquierda.

Las indicaciones de tráfico se situarán en la casilla de la bifurcación.
El programa termina cuando el robot llega a la baliza y la recoge. Guarda el programa en tu carpeta de trabajo como ***Prog27.irobo.***

Programa 27) Robot rastreador. Abre el mapa default.map. Realiza un programa para conseguir un robot rastreador de línea blanca. El robot se detendrá al final de la línea blanca al detectar una pared. Guarda el programa en tu carpeta de trabajo como ***prog24.irobo.***

Programa 28) Laberinto: Abre el mapa maze1.map. El objetivo del programa es conseguir que el robot escape del laberinto de forma autónoma. El robot encuentra la salida al localizar y coger la baliza, terminando el programa. Guarda el programa en tu carpeta de trabajo como ***Prog28.irobo.***
Pista: Para salir de un laberinto basta con seguir siempre la pared de la derecha, o la pared de la izquierda.