PROGRAMACIÓN (1º BACH).

1. INTRODUCCIÓN.

Los ordenadores, máquinas, robots, etc. funcionan siguiendo un comportamiento cíclico en 3 fases:

1) Fase de entrada. Introducción de los datos necesarios, mediante teclado, micrófono, lector de código de barras, sensores, etc.

2) Fase de proceso. Tras introducir los datos, la máquina comienza su tratamiento, y realiza cálculos y operaciones sobre ellos. **Las operaciones y cálculos a realizar con los datos se especifican en los programas**. **Un programa es un conjunto de órdenes que indican a una máquina qué acciones hay que realizar sobre los datos para obtener los resultados deseados.**

3) Fase de salida. Una vez operados los datos de entrada, se obtiene de ellos un resultado, que es enviado a los dispositivos de salida (pantalla, impresora, actuadores de un robot o una máquina, etc.).

En este tema se estudiará cómo realizar programas de control para definir el funcionamiento de una máquina determinada (un ordenador, un robot, un autómata, etc.)

Ejemplos:

a) Calculadora:
- Fase de entrada: teclear los números y operaciones a realizar.
- Fase de proceso: el programa interno de la calculadora procesa los números en función de la operación indicada.
- Fase de salida: el resultado de la operación se envía a la pantalla para ser mostrado.

b) Climatizador del coche:
- Fase de entrada: leer la temperatura deseada indicada en el panel frontal del climatizador por el usuario. Medir la temperatura real del coche mediante un sensor de temperatura.
- Fase de proceso: se ejecuta el programa → comparar la temperatura medida con la temperatura deseada. Si se desea menos temperatura, se ha de inyectar aire frío. Si se desea más temperatura, se ha de inyectar aire caliente.
- Fase de salida: enviar una señal al sistema para activar el aire frío o el aire caliente.

c) Robot evita-obstáculos:
- Fase de entrada: leer los datos captados por el sensor de contacto del robot.
- Fase de proceso: se ejecuta el programa → si el sensor detecta un obstáculo, hay que hacer girar al robot. Si el robot no detecta obstáculo, debe seguir avanzando.
- Fase de salida: actuar sobre los motores del robot para que avance o gire.

2. LENGUAJES DE PROGRAMACIÓN.

Como se ha comentado antes, el funcionamiento de una máquina se define mediante un programa de control.

Para escribir programas de control se emplean **lenguajes de programación**.

Un lenguaje de programación es un conjunto de reglas, símbolos, y normas de sintaxis, que se aplican para desarrollar programas.

En general, los lenguajes de programación proporcionan un conjunto de instrucciones básicas que la máquina a controlar puede entender. Así mismo, definen el conjunto de reglas, símbolos y normas de sintaxis que permitirán emplear dicho juego de instrucciones básicas para escribir los programas que controlarán el funcionamiento de dicha máquina.

En definitiva, escribir un programa de control de una máquina consistirá en escribir una secuencia de instrucciones que le indiquen a la máquina la operación a realizar, respetando las normas y reglas que el lenguaje de programación impone.

Ejemplo:

- Juego de instrucciones de una barrera de aparcamiento:
  - leerSensorPresencia
  - subirBarrera
  - bajarBarrera.

- Programa:

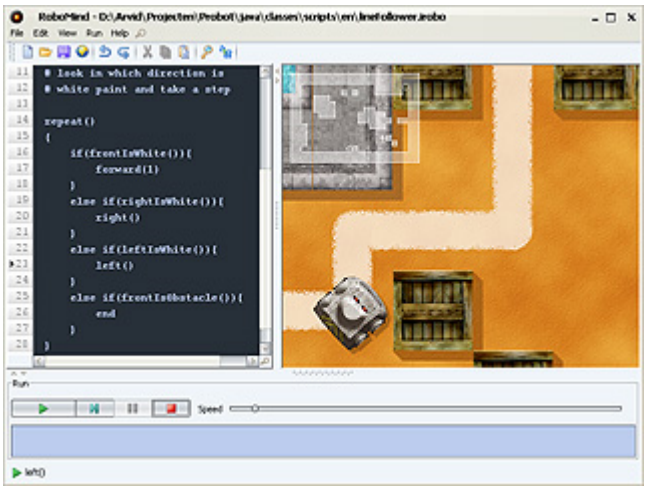
```
Si (leerSensorPresencia = "coche detectado")
        { subirBarrera }
En caso contrario
        { bajarBarrera}
Fin del programa
```

Existen multitud de lenguajes de programación, que se utilizan para diferentes aplicaciones:

a) Para desarrollar aplicaciones informáticas: Visual Basic, C++, etc.
b) Para desarrollar aplicaciones de internet: Java.
c) Para controlar robots: LOGO, RoboLab, RoboMind.
d) Para controlar máquinas de fábricas (autómatas programables): Grafset, Ladder, etc.

Los distintos lenguajes de programación son muy similares entre sí, utilizan las mismas estructuras de programación y procedimientos de desarrollo de programas. Lo único en que se diferencian es en el juego de instrucciones que proporcionan al usuario (no es lo mismo controlar una aplicación informática que un robot, por lo que el juego de instrucciones ha de ser diferente en cada caso).

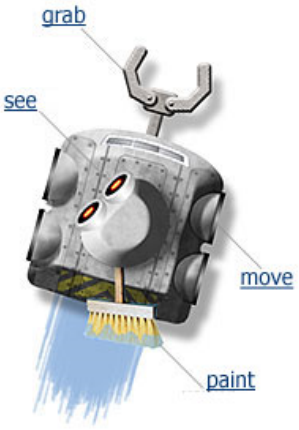
2. LENGUAJE DE PROGRAMACIÓN ROBOMIND.



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 alumnos de 4º de ESO 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.

3. EL ROBOT VIRTUAL ROBOMIND.

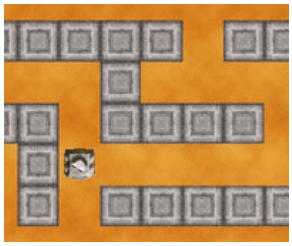


A continuación se muestra el robot que se programará en RoboMind. Se trata de un robot móvil equipado con los varios dispositivos que le permiten moverse, mirar alrededor, coger objetos, y pintar.

- 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.

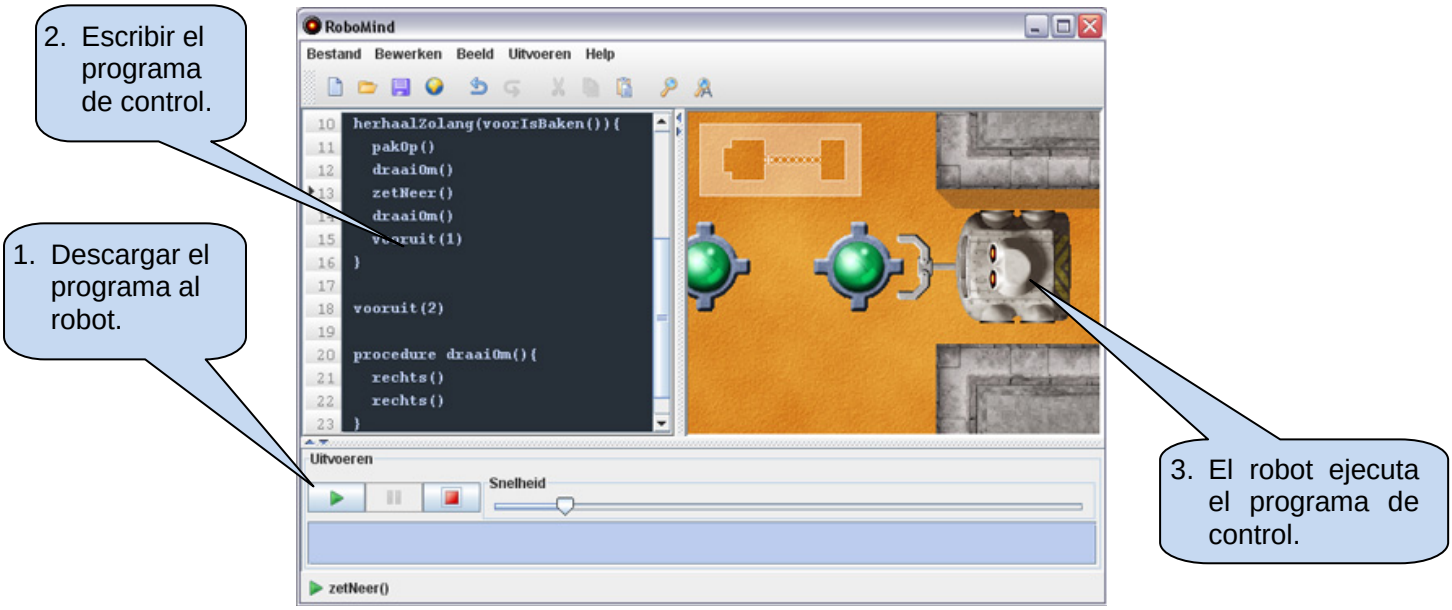
El programa de control se encargará de definir el comportamiento del robot. El programa de control deberá leer la información que los sensores capten del entorno (fase de entrada), interpretar y manipular dicha información (fase de proceso), y modificar el comportamiento de los actuadores en función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

4. 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:

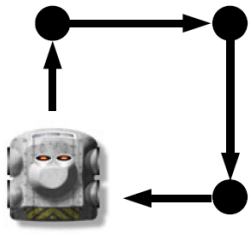
1) Escribir el programa de control.
2) Descargar el programa de control al robot, para definir su comportamiento.
3) El robot ejecuta el programa de control.



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()
```



5. EL LENGUAJE DE PROGRAMACIÓN ROBOMIND.

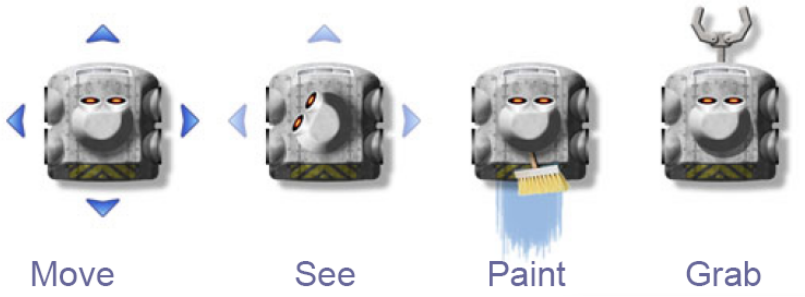
Para hacer que una máquina, un ordenador o un robot funcionen, hay que darle órdenes o ***instrucciones***.

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.

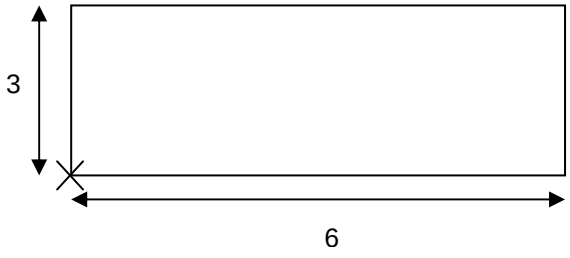
Las instrucciones de movimiento controlan los motores del robot, y por tanto, el movimiento del motor.

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".

Programa 1) Abre el mapa "default.map". 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 al programa como ***prog1.irobo***.

X → posición inicial.



Programa 2) Abre el mapa “default.map”. 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 al programa como ***prog2.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).

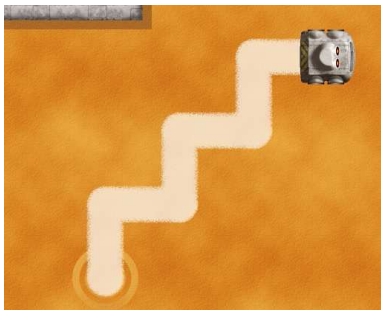
PINTAR		
paintWhite()		Bajar la brocha con pintura blanca al suelo para pintar en blanco.
paintBlack()		Bajar la brocha con pintura negra al suelo para pintar en negro.
stopPainting()		Dejar de pintar. Esconder la brocha.

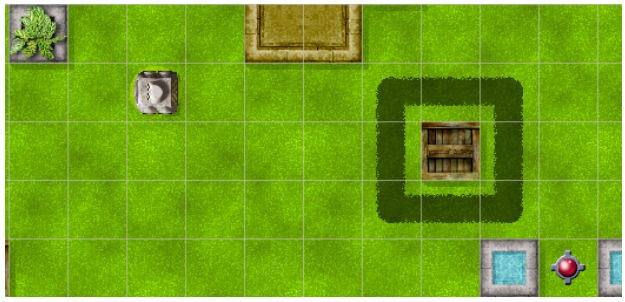
COGER		
pickUp()		Coger la baliza situada frente al robot.
putDown()		Soltar la baliza y dejarla frente al robot.

Actividades “pintar y recoger”.

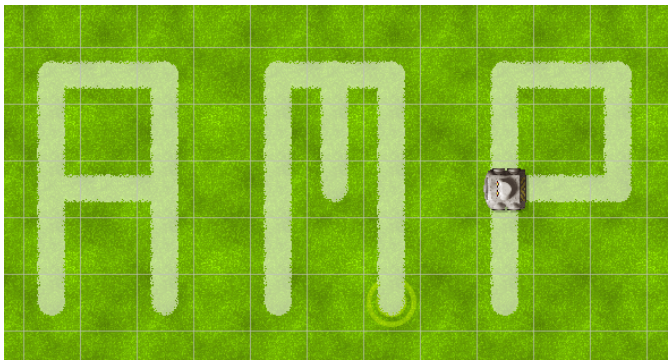
Programa 3) En el mapa default.map, escribe un programa para que el robot dibuje una escalera blanca con 3 escalones. Guarda al programa como ***prog3.irobo***.

Programa 4) 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 al programa como ***Prog4.irobo***.

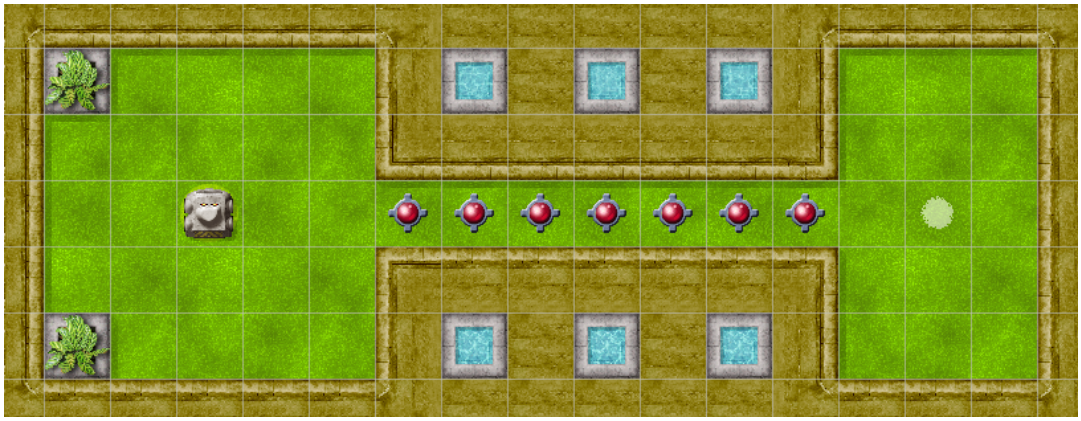




 Programa 5) Abre el mapa “openArea.map”.
Escribe un programa para que el robot escriba las iniciales de tu nombre y apellidos (3 letras).
Debes respetar los espacios en blanco entre cada inicial. Guarda al programa como ***prog5.irobo***.



Programa 6) 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 al programa como ***prog6.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).

NOTA: El robot solo es capaz de ver en las posiciones inmediatamente adyacentes, pero no en diagonal. No es capaz de ver dos cuadrados más allá de su posición.

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

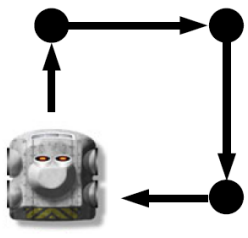
Esta instrucción termina con la ejecución del bucle ("rompe" el bucle), y continua ejecutando la instrucción que sigue a la llave que cierra el bucle. Sirve para forzar la salida de un bucle.

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

Cuestiones "bucles".

Programa 7) Escribe el programa que hace que el robot trace un cuadrado, pero utilizando un bucle. Guarda al programa como ***prog7.irobo***.

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



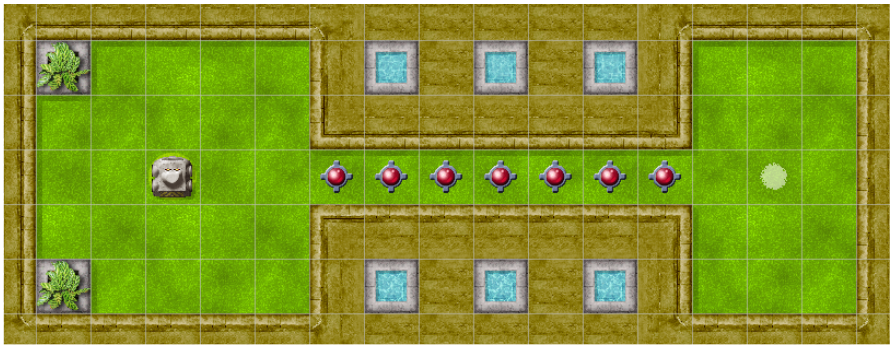
Programa 8) Programa ahora al robot para que permanezca trazando cuadrados de forma indefinida. Guarda al programa como ***prog8.irobo***.

Programa 9) 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. Guarda al programa como ***prog9.irobo***.

Programa 10) Haz un programa que, utilizando un bucle, haga que el robot dibuje una escalera de 4 escalones. Guarda al programa como ***prog10.irobo***.

Programa 11) 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. **Este programa debe funcionar para los mapas copyLine1.map y copyLine2.map**. Guarda al programa como ***prog11.irobo***.

Programa 12) Vuelve a realizar el programa 7 (pasar balizas). En este caso, el proceso de coger y retirar cada baliza del camino lo realizarás mediante un bucle. Programa al robot de forma que realice esta acción sin saber a priori cuántas balizas ha de retirar. Además, el robot tampoco sabe a qué distancia se situará la primera baliza. **Asegúrate que el programa funciona tanto para los mapas passBeacons.map, passBeacons2.map y passBeacons3.map**. Guarda al programa como ***prog12.irobo***



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:

```
repeat()
{
    if (frontIsObstacle())
        {right()}
    else
        {forward(1)}
}
```

a) ¿Para qué sirve?

b) Explica su funcionamiento.

end

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

Actividades "sentencias condicionales".

Programa 13) 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 al programa como ***prog13.irobo***.

NOTA: Este programa es para practicar el uso de la instrucción flipCoin(). Este concepto debe quedar claro.

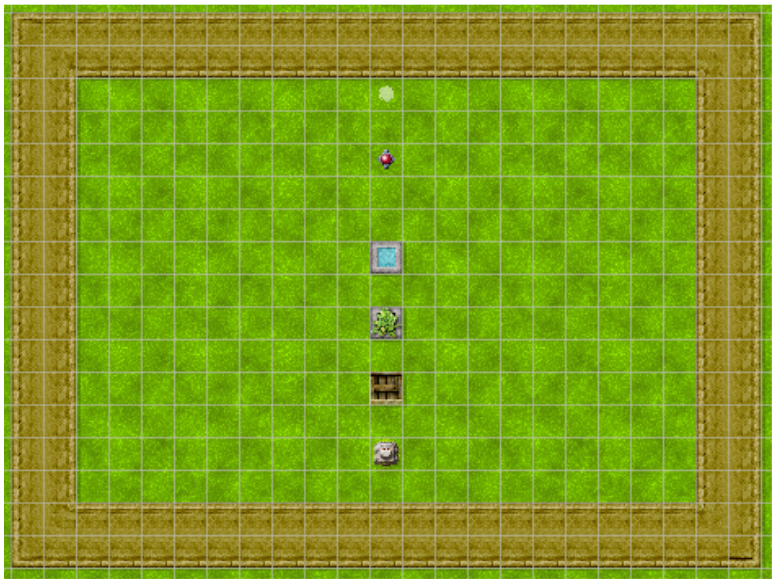
NOTA2: es posible que el robot colisione con las paredes en su movimiento. Es natural, se trata de un "robot borracho".

Programa 14) 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. **El programa debe funcionar para los mapas copyLine1.map y copyLine2.map**. Guarda al programa como ***prog14.irobo***.

Programa 15) Repite el programa 15 (copiar línea). En este caso, la línea podrá ser blanca o negra, y el robot deberá copiarla, pero con el color contrario (si la línea es blanca, la copia negra, y viceversa). Este programa debe funcionar para los mapas copyLine1.map, copyLine2.map, y copyLine3.map. Guarda al programa como ***prog15.irobo***.

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. **Asegúrate que el programa funciona para los mapas goRightAtWhite, goRightAtWhite1, goRightAtWhite2 y goRightAtWhite3**. Guarda al programa como ***prog16.irobo***.

Programa 17) 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), encima del cual se para. Al esquivar el objeto, el robot debe volver a la línea vertical de recorrido. **El programa debe funcionar para los mapas avoidObstacles, avoidObstacles1, avoidObstacles2 y avoidObstacles3**. Guarda al programa como ***prog17.irobo***.



Programa 18) 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 al programa como ***prog18.irobo***.

Programa 19) 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. Al esquivar el obstáculo largo, el robot debe volver a la línea vertical de recorrido **El programa debe funcionar para los mapas avoidContinuousObstacles.map, avoidContinuousObstacles2.map y avoidContinuousObstacles3.map**. Guarda al programa como ***prog19.irobo***.

Programa 20) Aparcando. Abre el mapa findSpot1.map. Programa al robot para que “aparque” en el hueco marcado con una señal blanca. **El programa debe funcionar para todos mapas findSpot**. Guarda al programa como ***prog20.irobo***

Programa 21) 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. **El programa debe funcionar para todos los mapas, y el robot nunca debe colisionar**. Guarda al programa como ***prog21.irobo***

Programa 22) Busca-balizas: programa al robot para buscar balizas. Para ello debe recorrer autónomamente el mapa default.map de igual forma que en el ejercicio 22, en busca de balizas. Cuando encuentra una baliza, debe cogerla y detenerse. Guarda al programa como ***prog22.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()

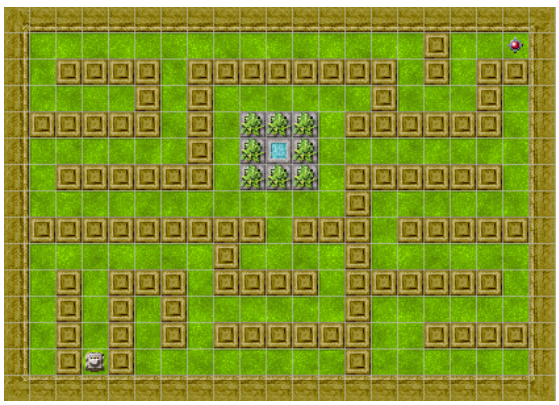
El robot avanzará mientras al frente esté despejado y a la izquierda también esté despejado (y = and)

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).
Ejemplo 4: leftIsObstacle() and not frontIsWhite() → la condición se cumple si a la izquierda hay un obstáculo y si al frente no está pintado de blanco.

Actividades finales.

Programa 23) 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 al programa como ***prog23.irobo.***

Programa 24) Laberinto (1): 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. **El programa debe funcionar en los mapas maze1.map, maze2.map y maze3.map**. Algoritmo 1: Para salir de estos laberintos basta con seguir siempre la pared de la derecha, o la pared de la izquierda. Guarda al programa como ***Prog24.irobo.***



Programa 25) Robot rastreador. Abre el mapa default.map. Realiza un programa para conseguir un robot rastreador de línea blanca. El programa consta de dos tareas: en primer lugar el robot debe moverse por el mapa de forma autónoma, como lo hacía en el programa 22, buscando la línea blanca. Cuando encuentra la línea blanca debe empezar a seguirla de forma automática (sin saber a priori por dónde discurre dicha línea), deteniéndose cuando detecte que la línea blanca se ha acabado. **Este programa debe funcionar en todos los mapas default.map**. Guarda al programa como ***prog25.irobo.***

Programa 26) Robot rastreador (2).
Abre el mapa rm-lf-task2.map. Se trata de programar al robot para seguir la línea blanca, pero la línea blanca está interrumpida por puntos negros. Consigue que el robot sea capaz de seguir la línea blanca a pesar de las interrupciones. Guarda al programa como ***prog26.irobo***

Programa 27) Parking.
Abre el mapa rm-lf-task3.map. El robot debe avanzar de forma aleatoria sobre el mallado de líneas blancas en busca de un lugar donde aparcar (punto negro). Para recorrer el parking de forma aleatoria, necesitarás cambiar de dirección aleatoriamente cada vez que llegues a un cruce de caminos (derecha, al frente, izquierda). ¿Cómo decidir aleatoriamente entre tres posibles opciones? Tal vez necesites "anidar" 2 instrucciones flipCoin().Guarda al programa como ***prog27.irobo***

Programa 28) Campo de minas.
Abre el mapa rm-lf-task4.map.El programa es muy similar al anterior, el objetivo es que el robot llegue al punto negro, recorriendo aleatoriamente la llama de líneas blancas. Sin embargo, cuando encuentre una baliza, debe evitarla y dirigirse por otro camino. Guarda al programa como ***prog28.irobo***

Programa 29) Borra marcas.
Abre el mapa paintSpots.map. Escribe un programa que haga que el robot avance de forma aleatoria y autónoma por el mapa. Cuando el robot encuentre una marca negra, deberá pintarla blanca. El programa no acabará nunca, puesto que el robot no sabe cuántas marcas negras debe pintar. Guarda al programa como ***prog29.irobo.***

Programa 30) A través de las balizas.
Abre el mapa findLines3.map. Escribe un programa que permita al robot llegar a la meta, indicada por las líneas blancas. Para ello, hay que tener en cuenta que el robot no sabrá dónde está la meta, por lo que deberá moverse aleatoriamente a través de las balizas, retirándolas para poder avanzar. El programa termina automáticamente cuando el robot llega a la meta y se posiciona encima**. El programa debe funcionar para todos los mapas FindLines**. Guarda al programa como ***prog30.irobo.***

Programa 31) BONUS.
Abre el mapa rm-lf-task5.map. El objetivo es aparcar al robot en el punto negro. ¿No ves el punto negro? Créelo, está en algún lugar del mapa, puede que escondido. En este caso, no se te proporciona indicación alguna, se supone que a estas alturas ya eres un experto, haz lo que creas más conveniente.

Programa 32) BONUS.
Laberinto (2). El algoritmo que has programado en el ejercicio 24 no es válido para cualquier tipo de laberinto, solo para algunos laberintos (como los laberintos en los que lo has probado). A continuación, utilizaremos un algoritmo distinto a seguir una pared para escapar de un laberinto (algoritmo de Tremaux).

El algoritmo de Tremaux es válido para cualquier laberinto, y siempre encuentra la salida. Si el laberinto no tiene salida, lleva de vuelta a la entrada. Los pasos a seguir son los siguientes:

- Conforme el robot avanza, dibuja una línea blanca detrás del robot para marcar el camino que ya has recorrido.
- Si llega a un callejón sin salida, da la vuelta y vuelve por el mismo camino.
- Cuando se llega a un cruce que no se haya visitado antes, elige un camino aleatoriamente.
- Cuando se llega a un cruce que ya se haya visitado antes, considéralo un callejón sin salida y vuelve por el mismo camino.
- Si estás recorriendo un camino que ya has recorrido antes (es decir, un camino que ya esté pintado) y encuentras un cruce, ve en la dirección que no hayas recorrido antes, y si no la hay, ve por un camino que ya hayas recorrido.

- Si estás recorriendo un camino por el que ya has pasado antes (es decir, pintado en blanco), y te ves obligado a retroceder en dirección opuesta, dibuja una línea detrás de ti en color negro. Ese camino no debes volver a recorrerlo.
- Todos los caminos estarán sin pintar (los que no hayas recorrido nunca), pintados de blanco (los que has recorrido una vez), o pintados d negro (los que has recorrido una vez, y te has visto obligado a retroceder en la dirección opuesta, que los debes evitar de ahora en adelante).
  Fuente: http://www.astrolog.org/labyrnth/algrithm.htm

Obsérvalo en funcionamiento: http://www.youtube.com/watch?v=dfwzjtjndks

Prueba tu programa en el mapa **bigmaze.map**.

NOTA:

- Todos los programas son de obligatoria realización. No puedes pasar al ejercicio siguiente sin haber acabado el ejercicio anterior. Además es necesario, ya que los ejercicios que realizas te van preparan y enseñan para los ejercicios que has de realizar a continuación.

- Además, los ejercicios marcados con una estrella presentan puntuación para tu nota final. La puntuación es la siguiente:

ACTIVIDAD	PUNTUACIÓN
1 rectángulo 3x6	
2 sendero blanco	
3 escalera	
4 rodea caja	
5 iniciales	**0,25**
6 balizas	**0,25**
7 cuadrado	
8 cuadrado infinito	
9 busca límite	**0,25**
10 escalera bucle	
11 paralelo a línea	
12 balizas bucle	**0,25**
13 borracho	**0,5**
14 copia línea	**0,5**
15 copia línea otro color	

ACTIVIDAD	PUNTUACIÓN
16 marcas blancas	**0,5**
17 obstáculos aislados	**0,5**
18 obstáculos y balizas	
19 obstáculos continuos	**0,5**
20 Aparcar	**0,5**
21 Evita colisiones	
22 busca balizas	**0,5**
23 Enjaulado	**0,5**
24 Laberinto (1)	**1**
25 Rastrea línea blanca	**1**
26 Rastreador (2)	
27 Parking	
28 Campo de minas	**1**
29 Borra marcas	**1**
30 Atraviesa balizas	**1**

31 Busca punto	**+1**

32 Laberinto (2)	**+1**