



ugr

Universidad  
de Granada

# SEMINARIO 2

## Presentación Práctica 2

### Agentes Reactivos

**Inteligencia Artificial**

**Dpto. Ciencias de la Computación e  
Inteligencia Artificial**

ETSI Informática y de Telecomunicación  
UNIVERSIDAD DE GRANADA

*Curso 2014/2015*



**DECSAI**

# Índice

1. Introducción
2. Presentación del Problema
3. Presentación del Simulador
4. Implementación de un agente
5. Método de evaluación de la práctica

# Índice

1. Introducción
2. Presentación del Problema
3. Presentación del Simulador
4. Implementación de un agente
5. Método de evaluación de la práctica

# 1. Introducción

- El objetivo de esta práctica consiste en el diseño e implementación de un agente reactivo que es capaz de:
  - *percibir el ambiente y*
  - *actuar de acuerdo a un comportamiento simple predefinido*
- Trabajaremos con un simulador software de una aspiradora inteligente basada en los ejemplos del libro *Stuart Russell, Peter Norvig, “Inteligencia Artificial: Un enfoque Moderno”*
- El simulador que utilizaremos fue desarrollado por el profesor **Tsung-Che Chiang** de la NTNU (*Norwegian University of Science and Technology, Taiwan*)

# 1. Introducción

- Esta práctica cubre los siguientes objetivos docentes:
  - Entender la IA como conjunto de técnicas para el desarrollo de sistemas informáticos que exhiben comportamientos reactivos, deliberativos y/o adaptativos (sistemas inteligentes)
  - Conocer el concepto de agente inteligente y el ciclo de vida "percepción, decisión y actuación"
  - Comprender que el desarrollo de sistemas inteligentes pasa por el diseño de agentes capaces de representar conocimiento y resolver problemas y que puede orientarse a la construcción de sistemas bien completamente autónomos o bien que interactúen y ayuden a los humanos
  - Conocer distintas aplicaciones reales de la IA. Explorar y analizar soluciones actuales basadas en técnicas de IA.

# 1. Introducción

- Para seguir esta presentación:
  - Encender el ordenador
  - En la petición de identificación poned
    1. Vuestro identificador (Usuario)
    2. Vuestra contraseña (Password)
    3. Y en Código **codeblocks**
    4. Pulsar “Entrar”

# Índice

1. Introducción
2. **Presentación del Problema**
3. Presentación del Simulador
4. Implementación de un agente
5. Evaluación de la práctica



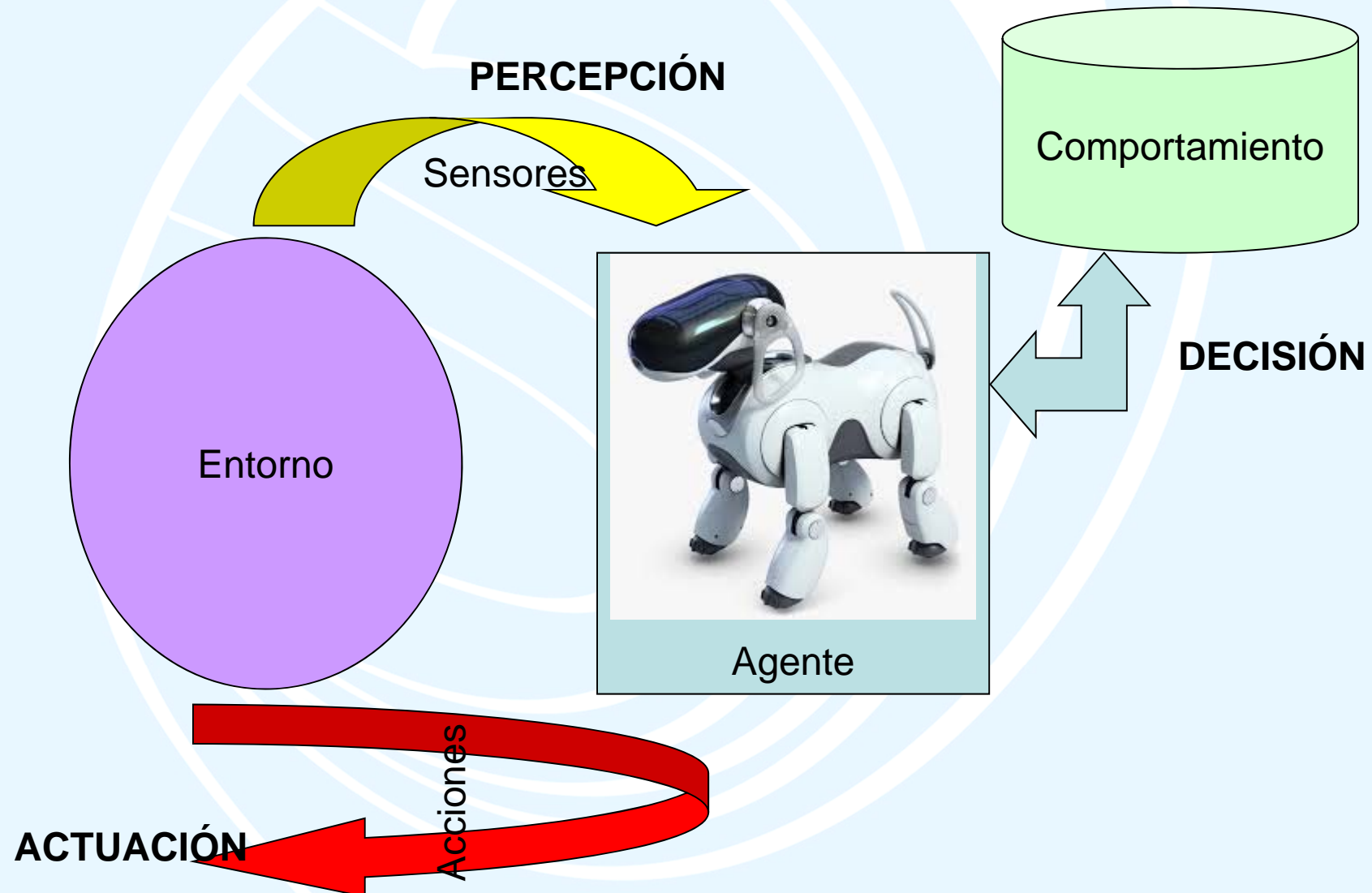
## 2. Presentación del Problema

### **Robot trufero**

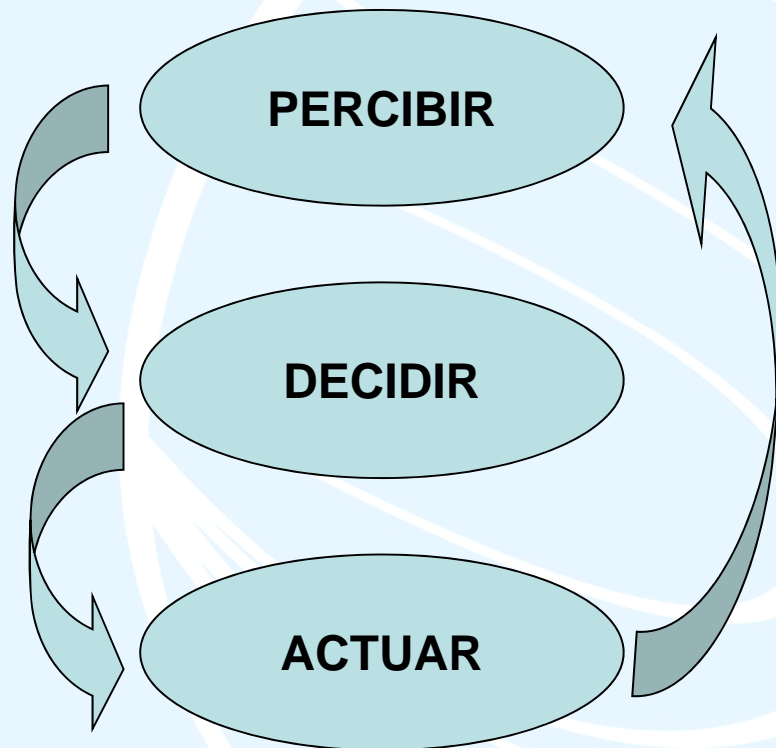




## 2. Presentación del Problema



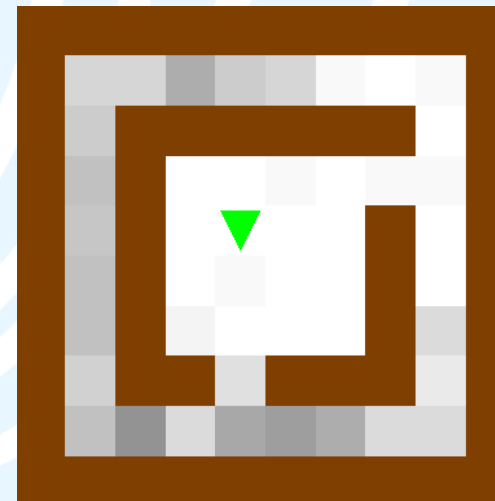
## 2. Presentación del Problema



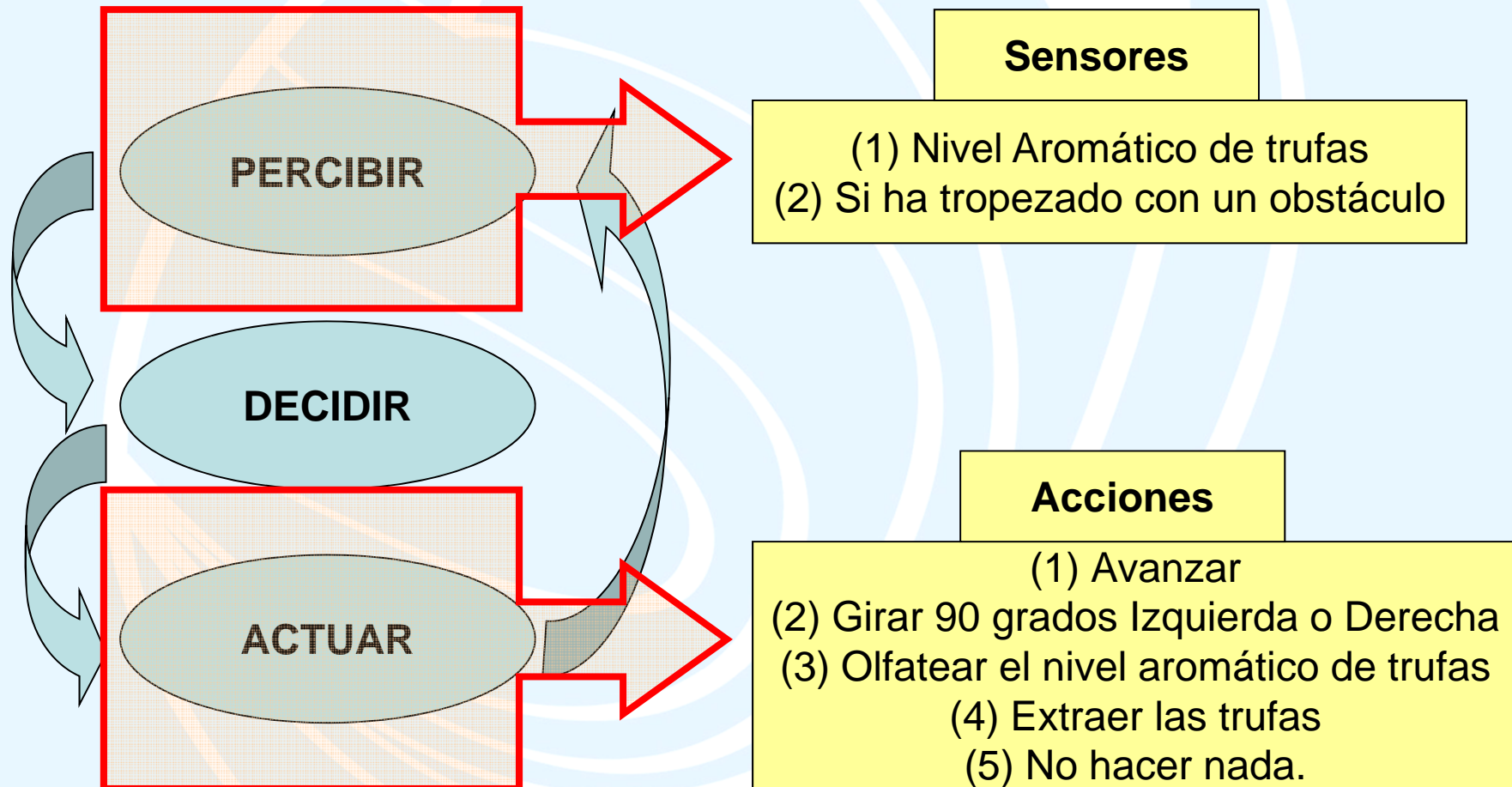
Controlador de ciclo cerrado

Vamos a trabajar con una versión simplificada del problema real restringiendo:

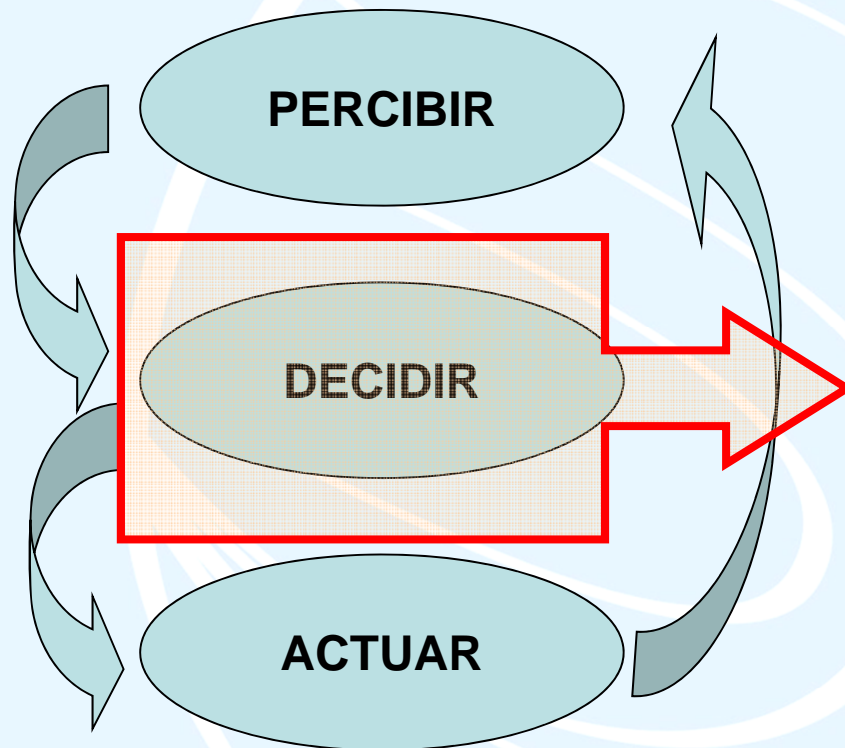
- Lo que es capaz de percibir el agente del entorno y,
- las acciones que el agente puede realizar.



## 2. Presentación del Problema



## 2. Presentación del Problema



El objetivo de la práctica será:

***Diseñar e implementar un modelo de decisión para este agente reactivo con las restricciones fijadas***

# Índice

1. Introducción
2. Presentación del Problema
3. **Presentación del Simulador**
4. Implementación de un agente
5. Evaluación de la práctica

### 3. Presentación del Simulador

- Compilación del simulador
- Ejecución del simulador

# 3. Presentación del Simulador

## 3.1. Compilación del Simulador

**Nota:** En esta presentación, asumimos que el entorno de programación **CodeBlocks** está ya instalado.

1. Cread la carpeta “U:\IA\practica2”
2. Descargar **Agent\_2014\_15.zip** desde la [web](#) de la asignatura y cópielo en la carpeta anterior.



- (a) <http://decsai.ugr.es>
- (b) Entrar en acceso identificado
- (c) Elegir la asignatura “Inteligencia Artificial”
- (d) Seleccionar “Material de la Asignatura”
- (e) Seleccionar “Práctica 2”
- (f) Descargar “Software Agente Reactivo”



# 3. Presentación del Simulador

## 3.1. Compilación del Simulador

1. Descomprimir en la raíz de esta carpeta y aparecerán los directorios:

- “.objs”,
- “include”
- “lib” y
- “map”

2. Los directorios “.objs”, “include” y “lib” son necesarios para la compilación de la práctica.

La subcarpeta “**map**” contiene la descripción de distintas habitaciones donde probar la aspiradora.

# 3. Presentación del Simulador

## 3.1. Compilación del Simulador

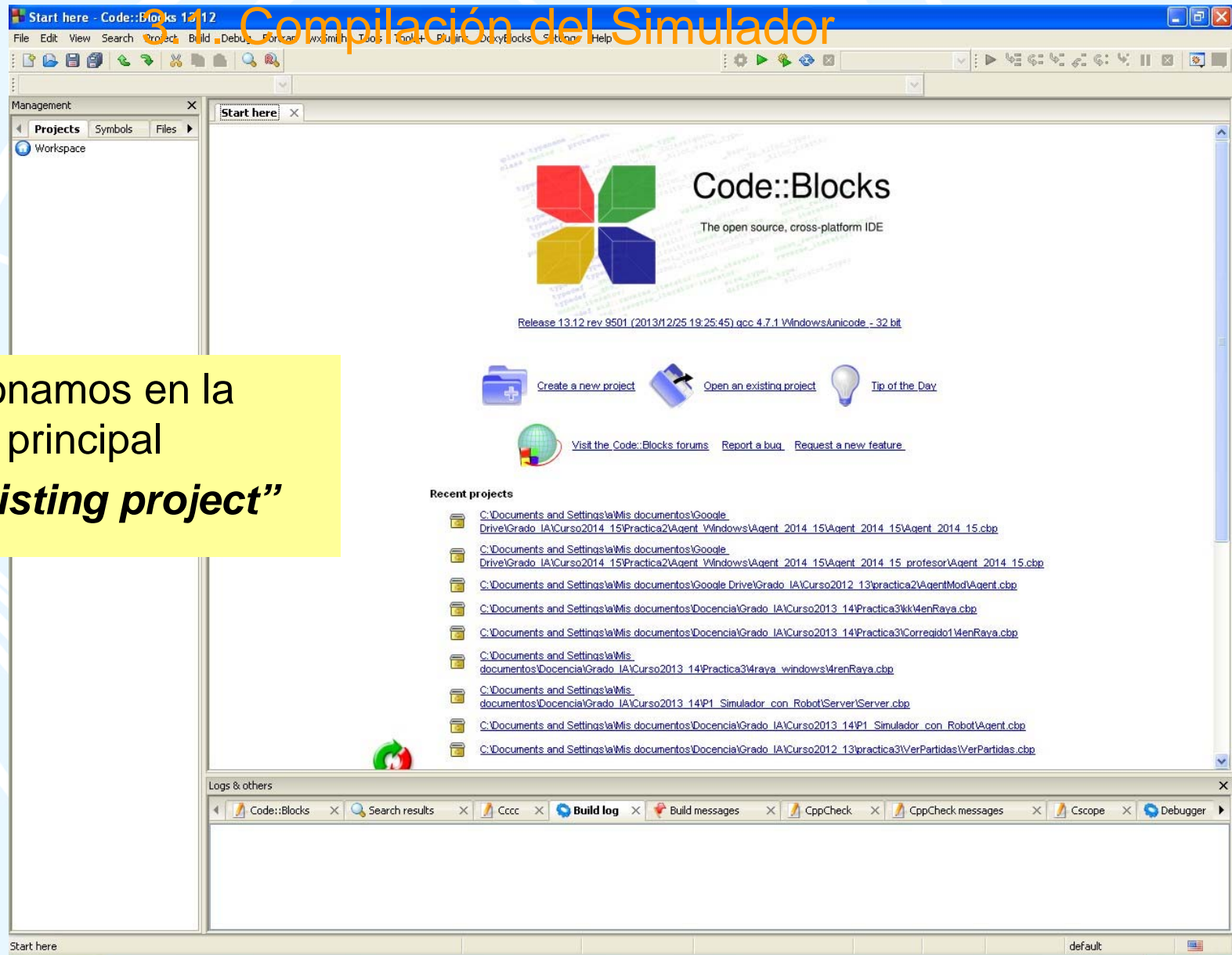
### 1. Abrimos “CodeBlocks”

- Si es la primera vez que lo lanzamos nos preguntará el compilador de C/C++ a usar:
  - Seleccionaremos la primera opción, “GNU GCC Compilar”
- Si es la primera vez, también nos preguntará si queremos asociar los ficheros C++ a este entorno de programación:
  - Seleccionaremos ***“Yes, associate Code::Blocks with every supported type (including project files from other IDEs)”***

# 3. Presentación del Simulador

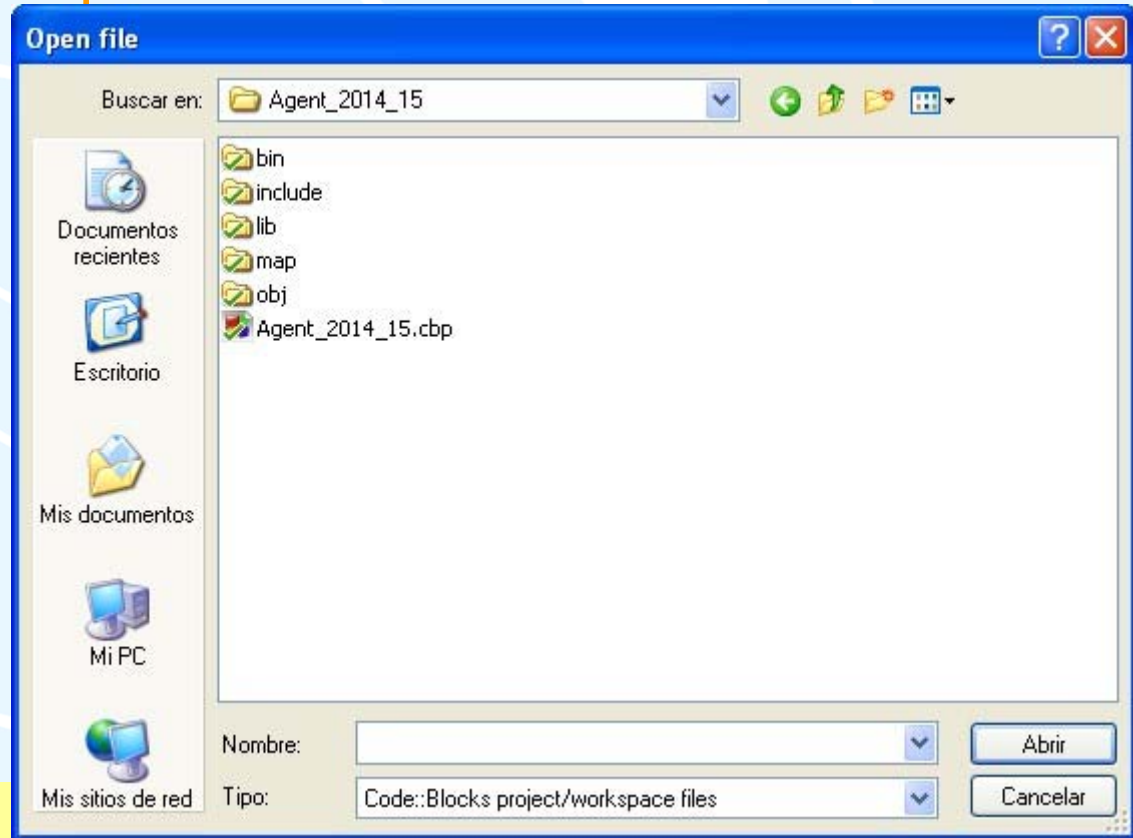
## 3.1. Compilación del Simulador

1. Seleccionamos en la pantalla principal ***“Open an existing project”***



# 3. Presentación del Simulador

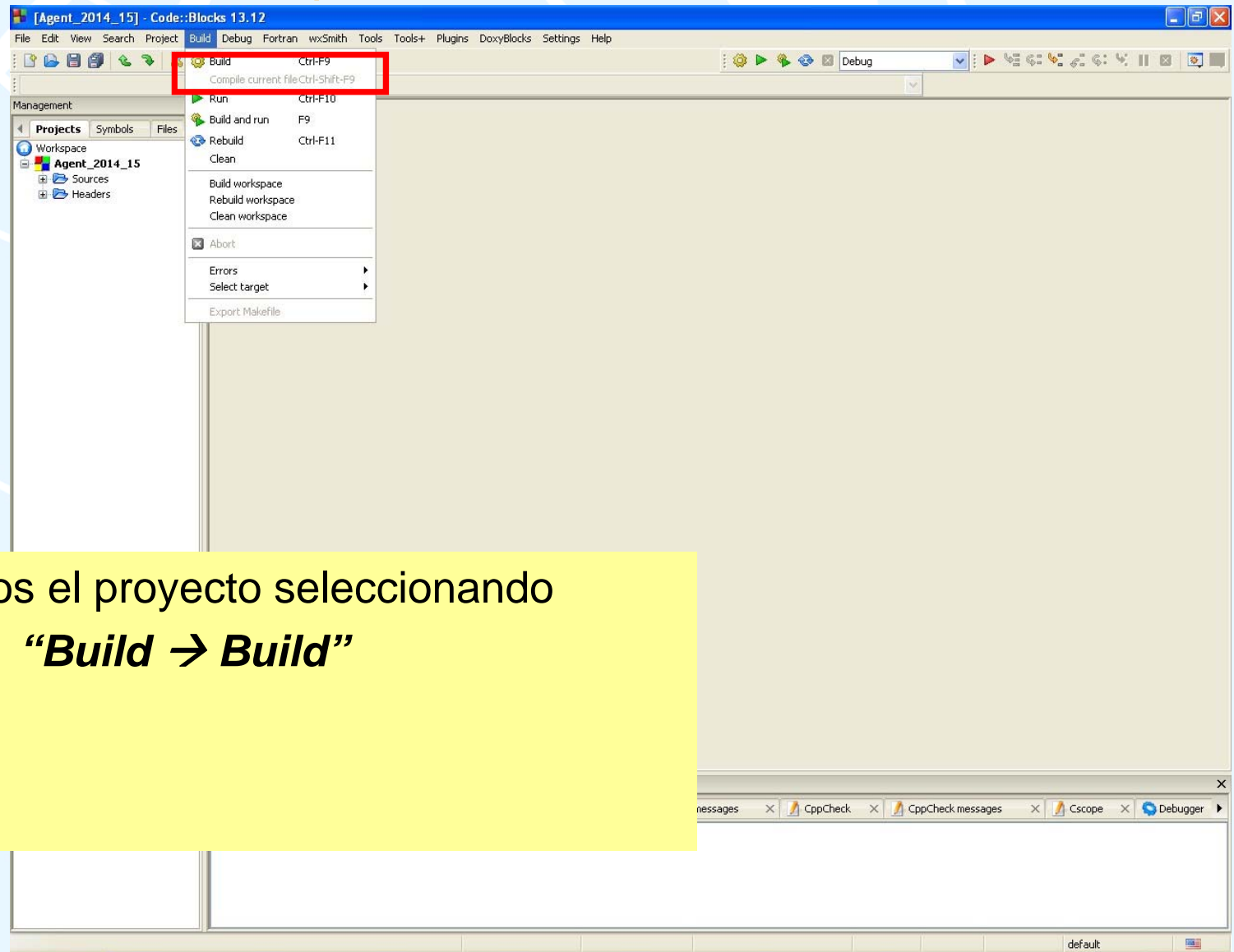
## 3.1. Compilación del Simulador



1. En “Tipo:” debe estar **“Code::Blocks project/workspace files”** y Abrimos **“Agent.cbp”** y volvemos a confirmar como Compilador a **GNU GCC compiler**

# 3. Presentación del Simulador

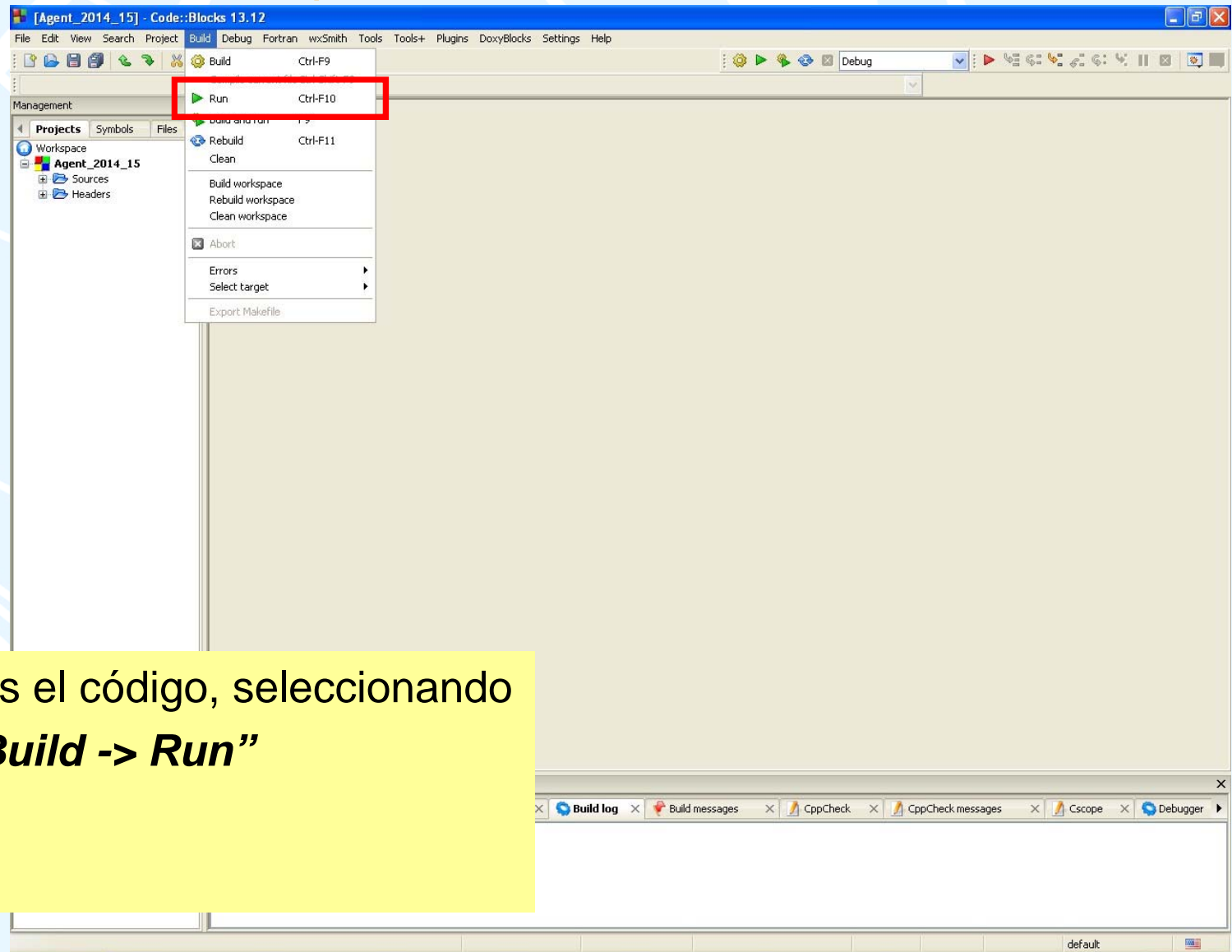
## 3.1. Compilación del Simulador



1. Compilamos el proyecto seleccionando  
***“Build → Build”***

# 3. Presentación del Simulador

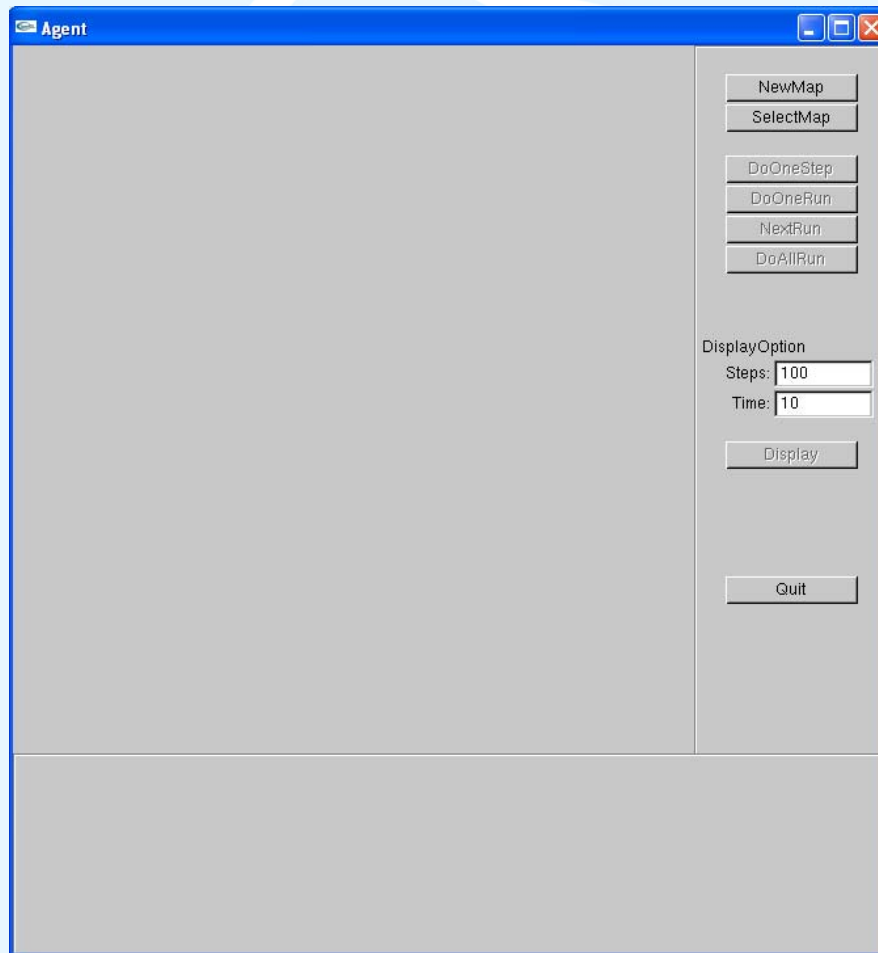
## 3.1. Compilación del Simulador



1. Ejecutamos el código, seleccionando  
***“Build -> Run”***

# 3. Presentación del Simulador

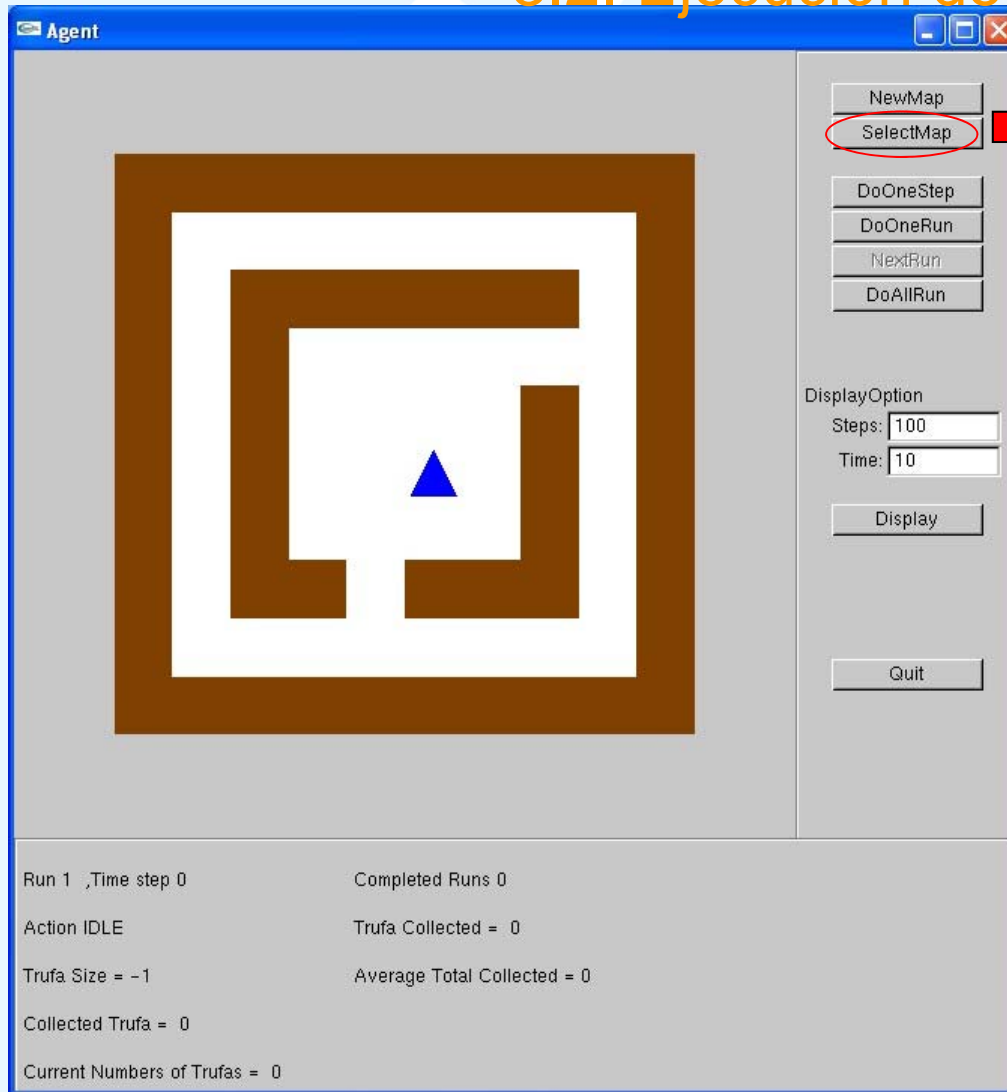
## 3.2. Ejecución del Simulador





# 3. Presentación del Simulador

## 3.2. Ejecución del Simulador

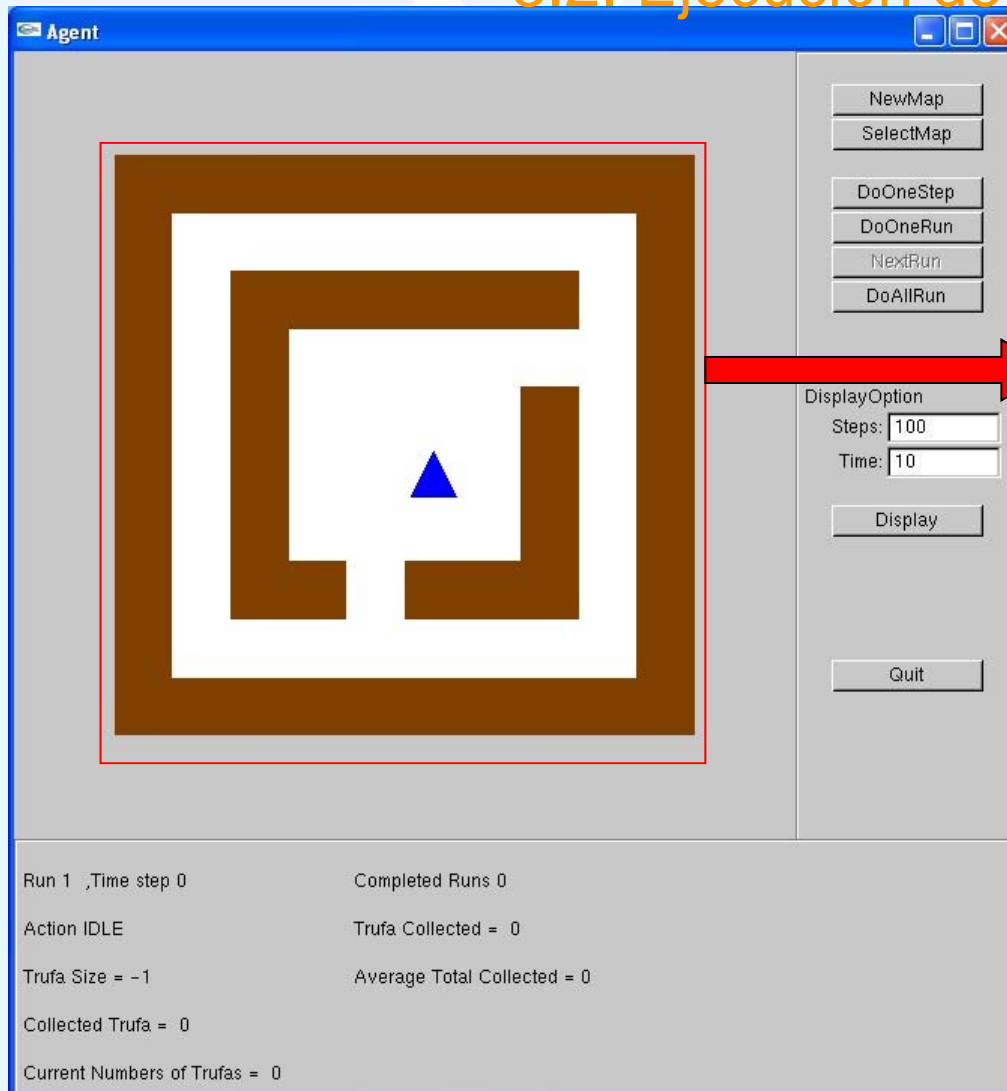


***“SelectMap”*** Selecciona el fichero de mapa sobre el que Realizar la simulación.

Seleccionamos el fichero ***“agent.map”***

# 3. Presentación del Simulador

## 3.2. Ejecución del Simulador

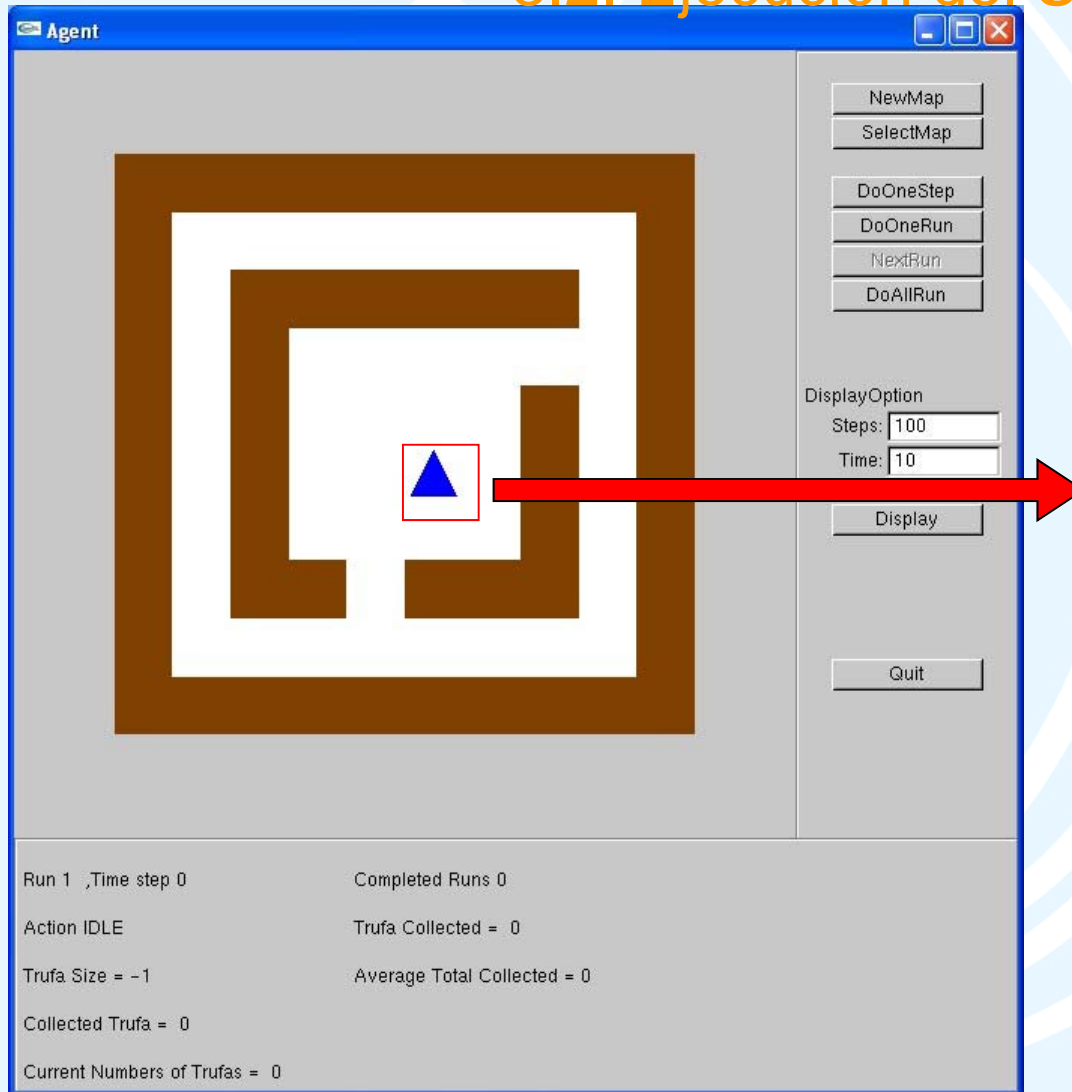


### ***Mundo simulado:***

- Los cuadrados marrones representan las paredes de la habitación.
- El resto de casillas representan la zona transitable.






# 3. Presentación del Simulador

## 3.2. Ejecución del Simulador



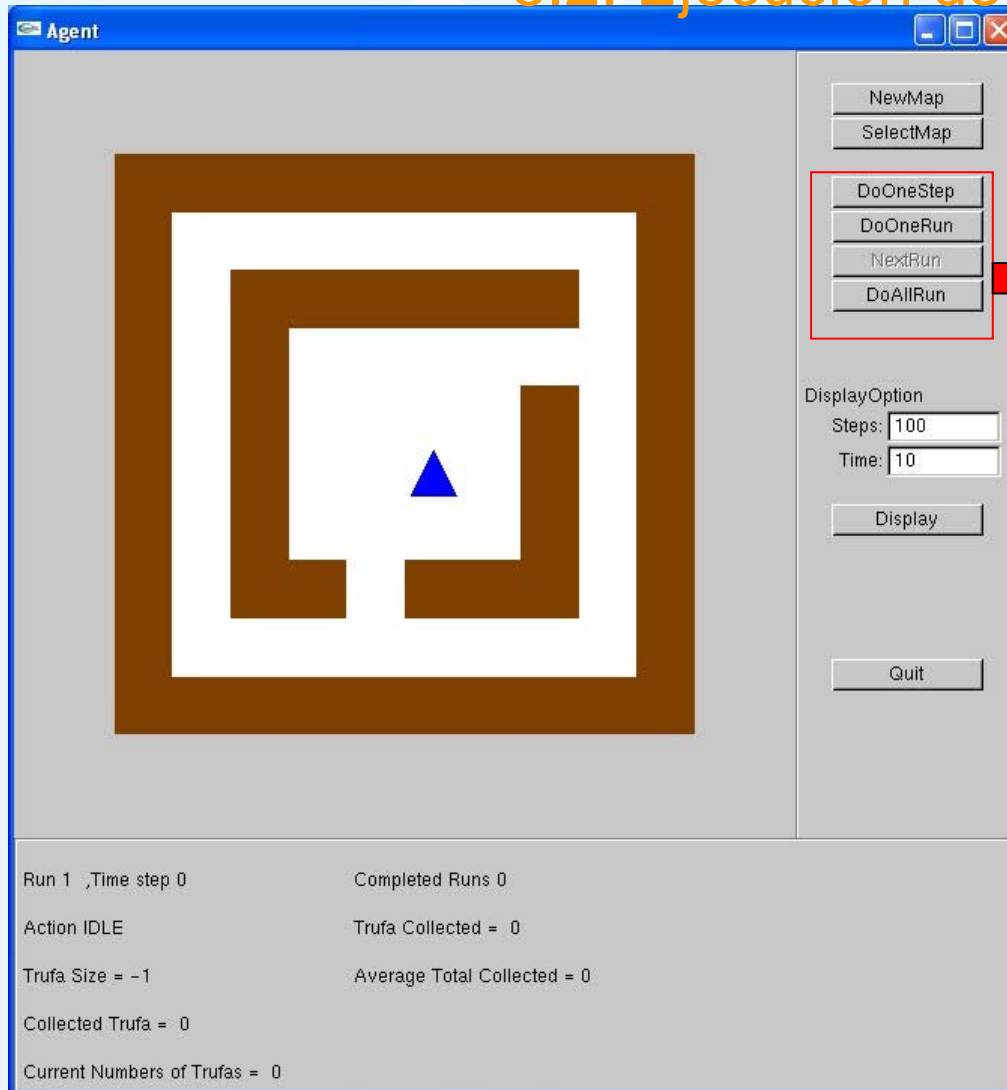
### ***Robot Trufero:***

Los diferentes estados representan el resultado de la última acción:

-  **Olió** el número de trufas de la casilla.
-  **Extrajo** las trufas de la casilla.
-  **No hizo nada**
-  **Hizo** un movimiento en la dirección indicada por la flecha.
-  **Chocó** con un obstáculo en la dirección que indica la flecha.

# 3. Presentación del Simulador

## 3.2. Ejecución del Simulador



***“DoOneStep”*** Produce el siguiente paso en la simulación.

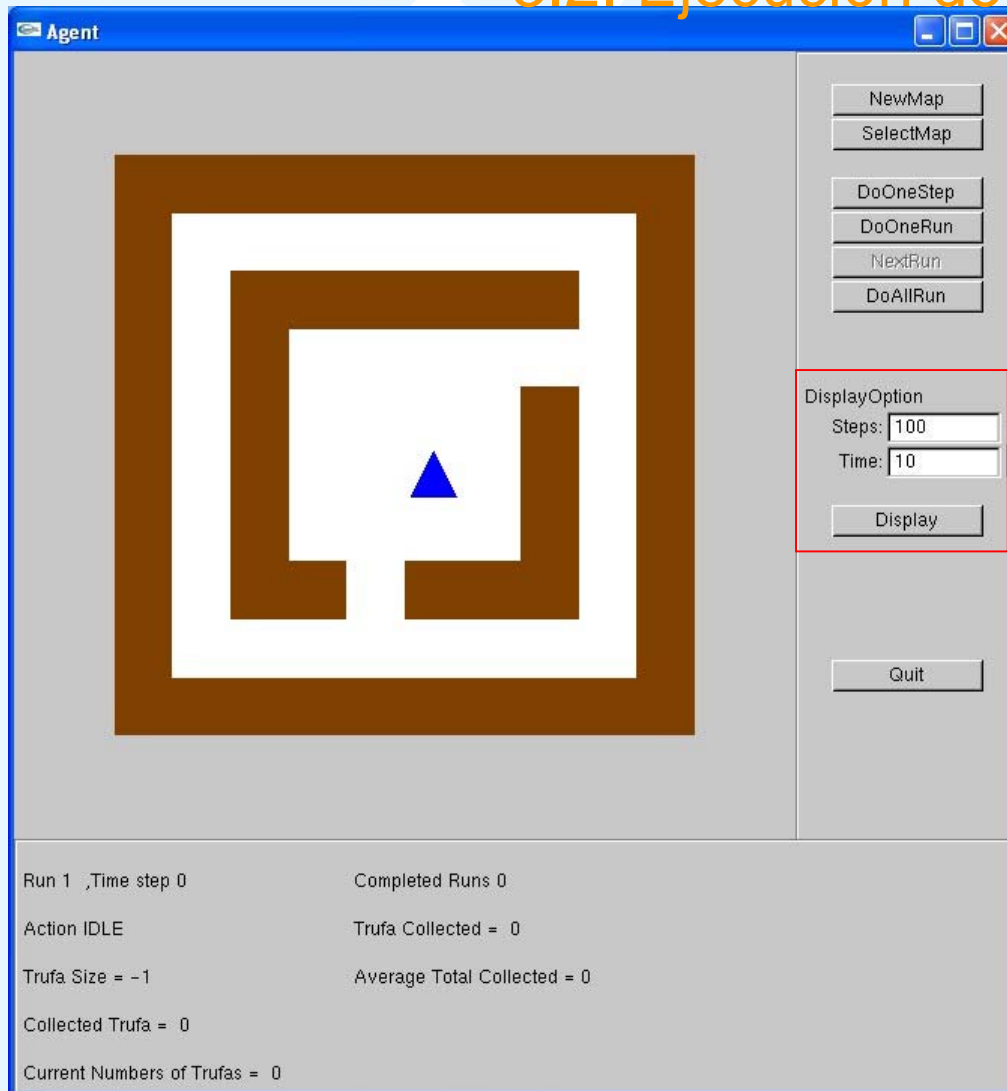
***“DoOneRun”*** Produce una simulación completa.

***“NextRun”*** Pasa a la siguiente ejecución.

***“DoAllRun”*** Ejecuta todas las ejecuciones de la simulación”

# 3. Presentación del Simulador

## 3.2. Ejecución del Simulador



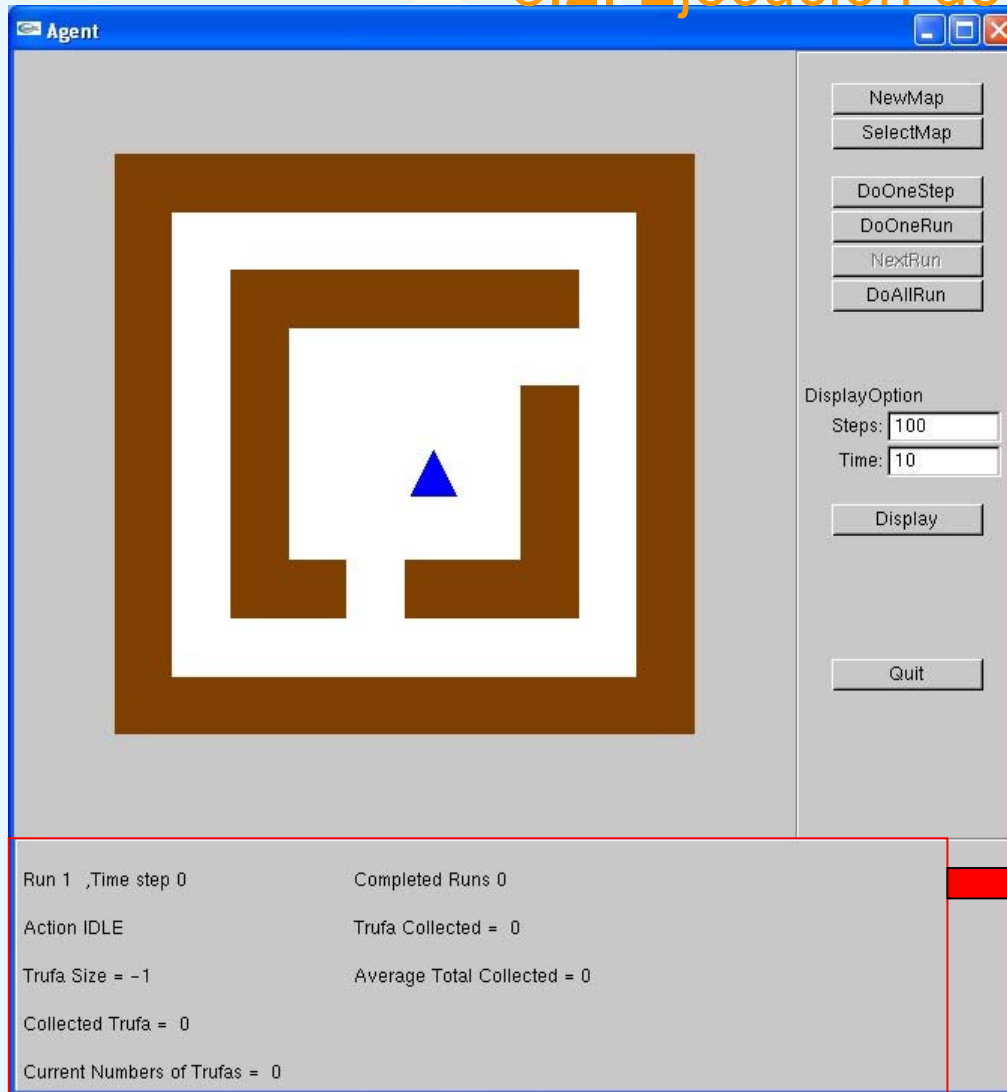
### ***“Display”***

Permite ver una secuencia continua de “Steps” pasos en el mundo simulado.

El valor **“Steps”** indica el número de pasos. El valor máximo aquí es el fijado en el mapa de la simulación.

# 3. Presentación del Simulador

## 3.2. Ejecución del Simulador

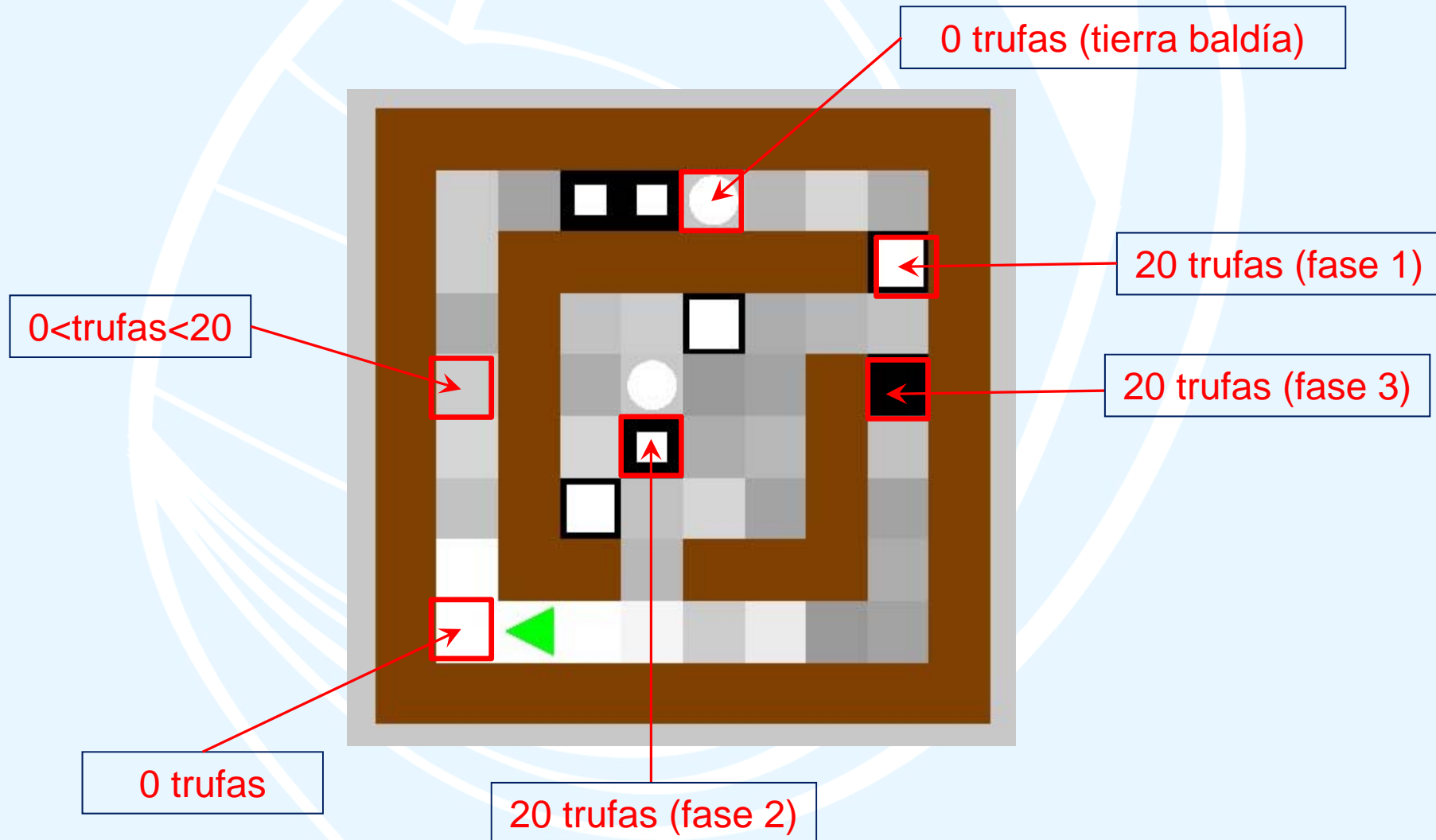


### ***Datos evolución de la simulación***

- Ejecución e iteración actual.
- Última acción ejecutada.
- Nivel de trufa olfateado en esta casilla.
- Cantidad de trufas recolectadas
- Ejecuciones ya completadas
- Total de trufas recolectadas en la última simulación ejecutada.
- Media de trufas recogidas en las simulaciones finalizadas.

# 3. Presentación del Simulador

## 3.2. Ejecución del Simulador





# Índice

1. Introducción
2. Presentación del Problema
3. Presentación del Simulador
4. Implementación de un agente
5. Método de evaluación de la práctica

## 4. Implementación de un agente

1. Descripción de los ficheros del simulador
2. Métodos y variables del agente
3. Modificando el comportamiento del agente: un ejemplo ilustrativo.

# 4. Implementación de un agente

## 4.1. Descripción de los ficheros

- Carpetas “include” y “lib”: Contiene ficheros de código fuente y bibliotecas necesarias para compilar la interfaz del simulador. **No son relevantes para la elaboración de la práctica**, aunque sí para que esta pueda compilar y ejecutarse correctamente.
- Carpeta map: Contiene los mapas disponibles en el simulador para modelar el mundo del agente.
- Fichero Agent.exe: Es el programa resultante, compilado y ejecutable, tras la compilación del proyecto.
- Ficheros Agent.cpb y Makefile.win: Son los ficheros principales del proyecto. Contienen toda la información necesaria para poder compilar el simulador.
- Ficheros Agent\_private.\* y agent.ico: Ficheros de recursos de Windows para la compilación (iconos, información de registro, etc.).

# 4. Implementación de un agente

## 4.1. Descripción de los ficheros

- Fichero **main.cpp**: Código fuente de la función principal del programa simulador.
- Ficheros **random\_num\_gen.\***: Ficheros de código fuente que implementan una clase para generar números aleatorios.
- Ficheros **GUI.\***: Código fuente para implementar la interfaz del simulador.
- Ficheros **evaluator.\***: Código fuente que implementa las funciones de evaluación del agente (energía consumida, suciedad acumulada, etc.).
- Ficheros **environment.\***: Código fuente que implementa el mundo del agente (mapa del entorno, suciedad en cada casilla, posición del agente, etc.).
- Ficheros **agent.\***: Código fuente que implementa al agente.

# 4. Implementación de un agente

## 4.2. Métodos y variables del agente

- Los dos únicos ficheros que se pueden modificar son “**agent.cpp**” y “**agent.h**” que son los que describen el comportamiento del agente.
- El agente **sólo** es capaz de percibir 2 señales del entorno:
  - **bump\_** (boolean) *true* indica que ha chocado con un obstáculo
  - **trufa\_size\_** (integer) *-1* indica que no se ha olfateado esa casilla. Los valores que toma son los siguientes:

0, 1, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20.

# 4. Implementación de un agente

## 4.2. Métodos y variables del agente

agent.h

```
7 // -----
8 //               class Agent
9 // -----
10 class Environment;
11 class Agent
12 {
13 public:
14     Agent() {
15         trufa_size_ = 0;
16         bump_ = false;
17     }
18
19     enum ActionType
20     {
21         actFORWARD,
22         actTURN_L,
23         actTURN_R,
24         actSNIFF,
25         actEXTRACT,
26         actIDLE
27     };
28
29     void Perceive(const Environment &env);
30     ActionType Think();
31 private:
32     int trufa_size_;
33     bool bump_;
34
35
36 };
```

Variables *bump\_* y *trufa\_size\_*

# 4. Implementación de un agente

## 4.2. Métodos y variables del agente

agent.h

```
7 // -----
8 // ----- class Agent -----
9 // -----
10 class Environment;
11 class Agent
12 {
13 public:
14     Agent() {
15         trufa_size_ = 0;
16         bump_ = false;
17     }
18
19     enum ActionType
20     {
21         actFORWARD,
22         actTURN_L,
23         actTURN_R,
24         actSNIFF,
25         actEXTRACT,
26         actIDLE
27     };
28
29     void Perceive(const Environment &env);
30     ActionType Think();
31 private:
32     int trufa_size_;
33     bool bump_;
34
35
36 };
```

Constructor de Clase  
Pone a **false** bump\_ y  
trufa\_size\_ a 0



# 4. Implementación de un agente

## 4.2. Métodos y variables del agente

agent.h

```
7 // -----
8 // class Agent
9 // -----
10 class Environment;
11 class Agent
12 {
13 public:
14     Agent() {
15         trufa_size_ = 0;
16         bump_ = false;
17     }
18
19     enum ActionType
20     {
21         actFORWARD,
22         actTURN_L,
23         actTURN_R,
24         actSNIFF,
25         actEXTRACT,
26         actIDLE
27     };
28
29     void Perceive(const Environment &env);
30     ActionType Think();
31 private:
32     int trufa_size_;
33     bool bump_;
34
35
36 };
```

Esta función toma una variable de tipo **Environment** que representa la situación actual del entorno, y da valor a las variables **bump\_** y **trufa\_size\_**

# 4. Implementación de un agente

## 4.2. Métodos y variables del agente

agent.h

```
7 // -----  
8 // class Agent  
9 // -----  
10 class Environment;  
11 class Agent  
12 {  
13 public:  
14     Agent() {  
15         trufa_size_ = 0;  
16         bump_ = false;  
17     }  
18  
19     enum ActionType  
20     {  
21         actFORWARD,  
22         actTURN_L,  
23         actTURN_R,  
24         actSNIFF,  
25         actEXTRACT,  
26         actIDLE  
27     };  
28  
29     void Perceive(const Environment &env);  
30     ActionType Think();  
31 private:  
32     int trufa_size_;  
33     bool bump_;  
34  
35  
36 };
```

En función de las variables de estado elige la siguiente acción a realizar.

# 4. Implementación de un agente

## 4.2. Métodos y variables del agente

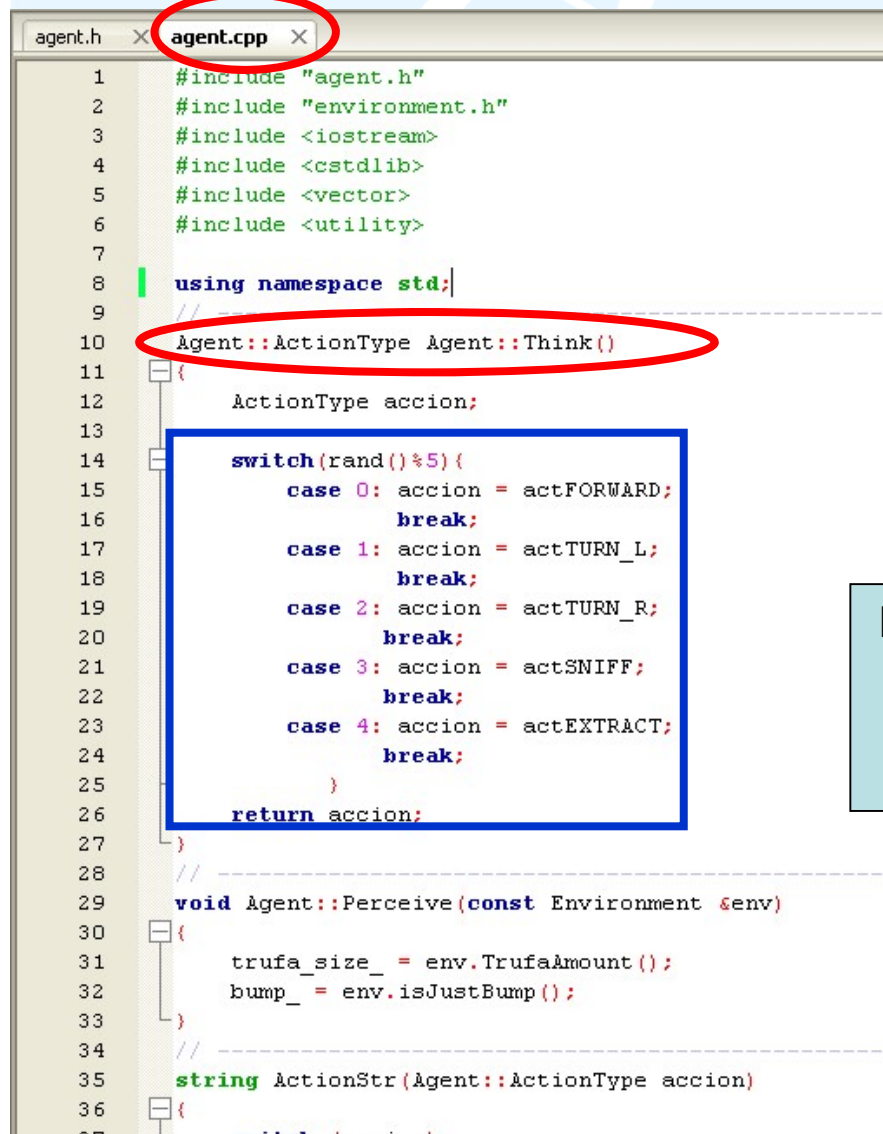
agent.h

```
7 // -----  
8 // class Agent  
9 // -----  
10 class Environment;  
11 class Agent  
12 {  
13 public:  
14     Agent() {  
15         trufa_size_ = 0;  
16         bump_ = false;  
17     }  
18  
19     enum ActionType  
20     {  
21         actFORWARD,  
22         actTURN_L,  
23         actTURN_R,  
24         actSNIFF,  
25         actEXTRACT,  
26         actIDLE  
27     };  
28  
29     void Perceive(const Environment &env);  
30     ActionType Think();  
31 private:  
32     int trufa_size_;  
33     bool bump_;  
34  
35  
36 };
```

Estas son las acciones posibles

# 4. Implementación de un agente

## 4.2. Métodos y variables del agente



```
1  #include "agent.h"
2  #include "environment.h"
3  #include <iostream>
4  #include <cstdlib>
5  #include <vector>
6  #include <utility>
7
8  using namespace std;
9
10 Agent::ActionType Agent::Think()
11 {
12     ActionType accion;
13
14     switch(rand()%5){
15         case 0: accion = actFORWARD;
16                 break;
17         case 1: accion = actTURN_L;
18                 break;
19         case 2: accion = actTURN_R;
20                 break;
21         case 3: accion = actSNIFF;
22                 break;
23         case 4: accion = actEXTRACT;
24                 break;
25     }
26     return accion;
27 }
28
29 // -----
30 void Agent::Perceive(const Environment &env)
31 {
32     trufa_size_ = env.TrufaAmount();
33     bump_ = env.isJustBump();
34 }
35
36 // -----
37 string ActionStr (Agent::ActionType accion)
38 {
39     switch (accion)
```

Implementación del  
Método Think()



La función implementa el siguiente comportamiento:  
*selecciono aleatoriamente una acción excepto  
no hacer nada*

## 4. Implementación de un agente

### 4.3. Modificando el comportamiento del agente: ejemplo ilustrativo 1.

Supongamos que queremos mejorar el comportamiento del agente anterior, indicándole que *si la casilla actual tiene un nivel oloroso superior a cero que extraiga*.

```
if (trufa_size_>0)  
    accion = actEXTRACT;
```

# 4. Implementación de un agente

## 4.3. Modificando el comportamiento del agente: un ejemplo ilustrativo.

```
10  Agent::ActionType Agent::Think()  
11  {  
12      ActionType accion;  
13  
14      if (trufa_size_>0){  
15          accion = actEXTRACT;  
16      }  
17      else {  
18          switch(rand()%4){  
19              case 0: accion = actFORWARD;  
20                  break;  
21              case 1: accion = actTURN_L;  
22                  break;  
23              case 2: accion = actTURN_R;  
24                  break;  
25              case 3: accion = actSNIFF;  
26                  break;  
27          }  
28      }  
29      return accion;  
30  }
```

Añado la regla:  
Si hay un nivel de olor a trufa >0,  
entonces extraer trufas

En otro caso, genero al azar  
una acción (menos *extraer*)



## 4. Implementación de un agente

### 4.3. Modificando el comportamiento del agente: ejemplo ilustrativo 2.

La regla introducida mejora algo, pero para dispararse requiere que **trufa\_size** tome valor mayor de 0. Así que añadimos el siguiente comportamiento: *si aún no se ha olfateado la casilla, entonces olfatear.*

```
if (trufa_size_==-1)  
    accion = actSNIFF;
```



# 4. Implementación de un agente

## 4.3. Modificando el comportamiento del agente: ejemplo ilustrativo 2.

```
9 // -----
10 Agent::ActionType Agent::Think()
11 {
12     ActionType accion;
13
14     if (trufa_size > 0) {
15         accion = actEXTRACT;
16     }
17     else if (trufa_size == -1) {
18         accion = actSNIFF;
19     }
20     else {
21         switch(rand()%3) {
22             case 0: accion = actFORWARD;
23                     break;
24             case 1: accion = actTURN_L;
25                     break;
26             case 2: accion = actTURN_R;
27                     break;
28         }
29     }
30     return accion;
31 }
```

Se mantiene la regla anterior

Añado la regla:  
Si NO he olfateado la casilla,  
entonces oler

En otro caso, genero al azar  
una acción de movimiento

# 4. Implementación de un agente

## 4.3. Modificando el comportamiento del agente: ejemplo ilustrativo 3.

Parece que mejora, pero el movimiento es un desastre. Intentemos lo siguiente: *si al moverse choca con un obstáculo, no dejemos que intente avanzar en esa dirección otra vez hasta que cambie de casilla*. Para hacer esto necesitamos:

- a) Definir variables miembro de la clase que me permitan manejar esta situación.
- b) Antes de tomar la siguiente decisión, actualizar los valores de dichas variables.
- c) Definir las reglas necesarias para controlarlo.

# 4. Implementación de un agente

## 4.3. Modificando el comportamiento del agente: ejemplo ilustrativo 3.

*si al moverse choca con un obstáculo, no dejemos que intente avanzar en esa dirección otra vez hasta que cambie de casilla*

a) **Definir variables miembro de la clase que me permitan manejar esta situación.**

- En este caso utilizaré 2 variables:
  - una que llamaré **orientación\_**, que me dice la dirección de mi movimiento,
  - un vector de 4 posiciones que llamaré **obstaculo\_**, indicando si es viable avanzar en dicha dirección.

El proceso para añadir estas dos variables de estado, es tan simple como incluirlas en la parte privada de la clase, y darle un valor inicial en el constructor de clase.

# 4. Implementación de un agente

## 4.3. Modificando el comportamiento del agente: ejemplo ilustrativo 3.

```
11 class Agent
12 {
13 public:
14     Agent() {
15         trufa_size_=0;
16         bump_false;
17         orientacion_=0;
18         for (int i=0; i<4; i++)
19             obstaculo[i]=false;
20     }
21
22     enum ActionType
23     {
24         actFORWARD,
25         actTURN_L,
26         actTURN_R,
27         actSNIFF,
28         actEXTRACT,
29         actIDLE
30     };
31
32     void Perceive(const Environment &env);
33     ActionType Think();
34 private:
35     int trufa_size_;
36     bool bump_;
37
38     int orientacion_;
39     bool obstaculo[4];
40 }
```

y se inicializan estos datos en el constructor.

Se declaran los nuevos datos en la parte privada de la clase.

**orientacion\_** = {0,1,2,3} asociados a Norte, Este, Sur, Oeste

# 4. Implementación de un agente

## 4.3. Modificando el comportamiento del agente: ejemplo ilustrativo 3.

*si al moverse choca con un obstáculo, no dejemos que intente avanzar en esa dirección otra vez hasta que cambie de casilla*

### **b) Antes de tomar la siguiente decisión, actualizar los valores de dichas variables.**

- Este es un proceso común en este tipo de sistemas. Justo en la iteración anterior se envió una acción ¿Se pudo realizar la acción? ¿Cómo puedo saberlo? ¿Tanto si completó la acción como sino, cambia eso mi conocimiento sobre el mundo?
- No hay una respuesta en todos los casos a las preguntas anteriores. Sin embargo, en nuestro ejemplo podemos responder todas las preguntas.
- Un método simple para realizar este proceso es incluir una nueva variable que nos diga que acción se tomó en la iteración anterior. Seguimos el proceso del caso a) y incluimos ***ult\_accion\_***

# 4. Implementación de un agente

## 4.3. Modificando el comportamiento del agente: ejemplo ilustrativo 3.

```
11 class Agent
12 {
13 public:
14     Agent() {
15         trufa_size = 0;
16         bump = false;
17         orientation = 0;
18         for (int i=0; i<4; i++)
19             obstaculo[i] = false;
20         ult_accion_ = actIDLE;
21     }
22
23     enum ActionType
24     {
25         actFORWARD,
26         actTURN_L,
27         actTURN_R,
28         actSNIFF,
29         actEXTRACT,
30         actIDLE
31     };
32
33     void Perceive(const Environment &env);
34     ActionType Think();
35 private:
36     int trufa_size;
37     bool bump;
38
39     int orientation;
40     bool obstaculo[4];
41     ActionType ult_accion_;
42
43 };
44
```

y se inicializa en el constructor.  
En el instante anterior no hizo nada

Se añade la variable `ult_accion_` de tipo `ActionType` como dato miembro de la clase.



# 4. Implementación de un agente

## 4.3. Modificando el comportamiento del agente: ejemplo ilustrativo 3.

*si al moverse choca con un obstáculo, no dejemos que intente avanzar en esa dirección otra vez hasta que cambie de casilla*

**b) Antes de tomar la siguiente decisión, actualizar los valores de dichas variables.**

- El proceso de actualización se hace al principio del método Think() preguntando por la acción que se hizo y que nueva información recibo del mundo.
- Una posible forma de hacerlo se describe en el siguiente código:



# 4. Implementación de un agente

## 4.3. Modificando el comportamiento del agente: ejemplo ilustrativo 3.

Sensores: ***{bump\_ , trufa\_size\_}***

Variables de estado: ***{orientacion\_ , obstaculo\_ , ult\_accion\_ }***

```
10 Agent::ActionType Agent::Think()
11 {
12     // ACTUALIZATION
13     switch(ult_accion_){
14         case actFORWARD:
15             if (bump_){
16                 obstaculo_[orientacion_]=true;
17             }
18             else {
19                 for (int i=0; i<4; i++)
20                     obstaculo_[i]=false;
21             }
22             break;
23         case actTURN_R:
24             orientacion ++;
25             if (orientacion_>3)
26                 orientacion_=0;
27             break;
28         case actTURN_L:
29             orientacion --;
30             if (orientacion_<0)
31                 orientacion_=3;
32             break;
33     }
34 }
35
```

# 4. Implementación de un agente

## 4.3. Modificando el comportamiento del agente: ejemplo ilustrativo 3.

Sensores: **{bump\_ , trufa\_size\_}**

Variables de estado: **{orientacion\_ , obstaculo\_ , ult\_accion\_ }**

```
10 Agent::ActionType Agent::Think()  
11 {  
12     // ACTUALIZATION  
13     switch(ult_accion){  
14         case actFORWARD:  
15             if (bump_){  
16                 obstaculo_[orientacion_]=true;  
17             }  
18             else {  
19                 for (int i=0; i<4; i++)  
20                     obstaculo_[i]=false;  
21             }  
22             break;  
23         case actTURN_R:  
24             orientacion ++;  
25             if (orientacion_>3)  
26                 orientacion_=0  
27             break;  
28         case actTURN_L:  
29             orientacion --;  
30             if (orientacion_<0)  
31                 orientacion_=3;  
32             break;  
33     }  
34 }  
35
```

Si la ultima acción fue avanzar y se activó el sensor de choque, entonces tengo un obstáculo delante y lo almaceno en el vector.

Si la ultima acción fue avanzar y **NO** se activó el sensor de choque, He pasado a una nueva casilla e Inicializo los obstáculos

# 4. Implementación de un agente

## 4.3. Modificando el comportamiento del agente: ejemplo ilustrativo 3.

Sensores: **{bump\_ , trufa\_size\_}**

Variables de estado: **{orientacion\_ , obstaculo\_ , ult\_accion\_ }**

```
10 Agent::ActionType Agent::Think()  
11 {  
12     // ACTUALIZATION  
13     switch(ult_accion_){  
14         case actFORWARD:  
15             if (bump_){  
16                 obstaculo_[orientacion_]=true;  
17             }  
18             else {  
19                 for (int i=0; i<4; i++)  
20                     obstaculo_[i]=false;  
21             }  
22             break;  
23         case actTURN_R:  
24             orientacion ++;  
25             if (orientacion_>3)  
26                 orientacion_=0  
27             break;  
28         case actTURN_L:  
29             orientacion --;  
30             if (orientacion_<0)  
31                 orientacion_=3;  
32             break;  
33     }  
34 }  
35
```

Si la ultima acción fue girar a la derecha,  
entonces cambio mi orientación.

# 4. Implementación de un agente

## 4.3. Modificando el comportamiento del agente: ejemplo ilustrativo 3.

Sensores: **{*bump\_* , *trufa\_size\_*}**

Variables de estado: **{*orientacion\_* , *obstaculo\_* , *ult\_accion\_*}**

```
10 Agent::ActionType Agent::Think()
11 {
12     // ACTUALIZATION
13     switch(ult_accion_){
14         case actFORWARD:
15             if (bump_){
16                 obstaculo_[orientacion_]=true;
17             }
18             else {
19                 for (int i=0; i<4; i++)
20                     obstaculo_[i]=false;
21             }
22             break;
23         case actTURN_R:
24             orientacion ++;
25             if (orientacion_>3)
26                 orientacion_=0;
27             break;
28         case actTURN_L:
29             orientacion --;
30             if (orientacion_<0)
31                 orientacion_=3;
32             break;
33     }
34 }
35
```

Si la ultima acción fue girar a la izquierda, entonces cambio mi orientación.

## 4. Implementación de un agente

### 4.3. Modificando el comportamiento del agente: ejemplo ilustrativo 3.

*si al moverse choca con un obstáculo, no dejemos que intente avanzar en esa dirección otra vez hasta que cambie de casilla*

#### **c) Definir las reglas necesarias para controlarlo.**

1. Si el nivel oloroso de trufas es mayor que 0 => EXTRAER
2. Si NO he medido el nivel oloroso de trufas => OLER
3. Si NO tengo un obstáculo delante => AVANZAR
4. Si NO tengo un obstáculo a la derecha => GIRAR DERECHA
5. En otro caso => GIRAR IZQUIERDA

# 4. Implementación de un agente

## 4.3. Modificando el comportamiento del agente: ejemplo ilustrativo 3.

Sensores: **{*bump\_* , *trufa\_size\_*}**

Variables de estado: **{*orientacion\_* , *obstaculo\_* , *ult\_accion\_*}**

```
10 Agent::ActionType Agent::Think()
11 {
12     // ACTUALIZATION
13     switch(ult_accion_){
14         case actFORWARD:
15             if (bump_){
16                 obstaculo_[orientacion_]=true;
17             }
18             else {
19                 for (int i=0; i<4; i++)
20                     obstaculo_[i]=false;
21             }
22             break;
23         case actTURN_R:
24             orientacion_++;
25             if (orientacion_>3)
26                 orientacion_=0;
27             break;
28         case actTURN_L:
29             orientacion_--;
30             if (orientacion_<0)
31                 orientacion_=3;
32             break;
33     }
34 }
35
```

```
36 // DECISION
37 ActionType accion;
38
39 if (trufa_size_>0){
40     accion = actEXTRACT;
41 }
42 else if (trufa_size_==1){
43     accion = actSNIFF;
44 }
45 else if (!obstaculo_[orientacion_]){
46     accion = actFORWARD;
47 }
48 else if (!obstaculo_[(orientacion_+1)%4]){
49     accion = actTURN_R;
50 }
51 else {
52     accion = actTURN_L;
53 }
54
55 ult_accion_ = accion;
56
57 return accion;
58 }
59 // -----
```



# 4. Implementación de un agente

## 4.3. Modificando el comportamiento del agente: ejemplo ilustrativo 3.

Sensores: *{bump\_ , trufa\_size\_}*

Variables de estado: *{orientacion\_ , obstaculo\_ , ult\_accion\_ }*

Si el nivel oloroso de trufas  
es mayor que 0 => EXTRAER

```
36 // DECISION
37 ActionType accion;
38
39 if (trufa_size_>0){
40     accion = actEXTRACT;
41 }
42 else if (trufa_size_==1){
43     accion = actSNIFF;
44 }
45 else if (!obstaculo_[orientacion_]){
46     accion = actFORWARD;
47 }
48 else if (!obstaculo_[(orientacion_+1)%4]){
49     accion = actTURN_R;
50 }
51 else {
52     accion = actTURN_L;
53 }
54
55 ult_accion_ = accion;
56
57 return accion;
58 }
59 // -----
```

**REGLA 1**



# 4. Implementación de un agente

## 4.3. Modificando el comportamiento del agente: ejemplo ilustrativo 3.

Sensores: *{bump\_ , trufa\_size\_}*

Variables de estado: *{orientacion\_ , obstaculo\_ , ult\_accion\_ }*

Si NO he medido el nivel  
oloroso de trufas => OLER

```
36 // DECISION
37 ActionType accion;
38
39 if (trufa_size_>0){
40     accion = actEXTRACT;
41 }
42 else if (trufa_size_== -1){
43     accion = actSNIFF;
44 }
45 else if (!obstaculo_[orientacion_]){
46     accion = actFORWARD;
47 }
48 else if (!obstaculo_[(orientacion_+1)%4]){
49     accion = actTURN_R;
50 }
51 else {
52     accion = actTURN_L;
53 }
54
55 ult_accion_ = accion;
56
57 return accion;
58 }
59 //
```

REGLA 2

# 4. Implementación de un agente

## 4.3. Modificando el comportamiento del agente: ejemplo ilustrativo 3.

Sensores: *{bump\_ , trufa\_size\_}*

Variables de estado: *{orientacion\_ , obstaculo\_ , ult\_accion\_ }*

Si NO tengo un obstáculo  
delante => AVANZAR

```
36 // DECISION
37 ActionType accion;
38
39 if (trufa_size_>0){
40     accion = actEXTRACT;
41 }
42 else if (trufa_size_==1){
43     accion = actSNIFF;
44 }
45 else if (!obstaculo_[orientacion_]){
46     accion = actFORWARD;
47 }
48 else if (!obstaculo_[(orientacion_+1)%4]){
49     accion = actTURN_R;
50 }
51 else {
52     accion = actTURN_L;
53 }
54
55 ult_accion_ = accion;
56
57 return accion;
58 }
59 //
```

REGLA 3

# 4. Implementación de un agente

## 4.3. Modificando el comportamiento del agente: ejemplo ilustrativo 3.

Sensores: *{bump\_ , trufa\_size\_}*

Variables de estado: *{orientacion\_ , obstaculo\_ , ult\_accion\_ }*

Si NO tengo un obstáculo a la derecha => GIRAR DERECHA

```
36 // DECISION
37 ActionType accion;
38
39 if (trufa_size_>0){
40     accion = actEXTRACT;
41 }
42 else if (trufa_size_==1){
43     accion = actSNIFF;
44 }
45 else if (!obstaculo_[orientacion_]){
46     accion = actFORWARD;
47 }
48 else if (!obstaculo_[(orientacion_+1)%4]){
49     accion = actTURN_R;
50 }
51 else {
52     accion = actTURN_L;
53 }
54
55 ult_accion_ = accion;
56
57 return accion;
58 }
59 //
```

**REGLA 4**

# 4. Implementación de un agente

## 4.3. Modificando el comportamiento del agente: ejemplo ilustrativo 3.

Sensores: *{bump\_ , trufa\_size\_}*

Variables de estado: *{orientacion\_ , obstaculo\_ , ult\_accion\_ }*

En otro caso

=> GIRAR IZQUIERDA

```
36 // DECISION
37 ActionType accion;
38
39 if (trufa_size_>0){
40     accion = actEXTRACT;
41 }
42 else if (trufa_size_==1){
43     accion = actSNIFF;
44 }
45 else if (!obstaculo_[orientacion_]){
46     accion = actFORWARD;
47 }
48 else if (!obstaculo_[(orientacion_+1)%4]){
49     accion = actTURN_R;
50 }
51 else {
52     accion = actTURN_L;
53 }
54
55 ult_accion_ = accion;
56
57 return accion;
58 }
59 // -----
```

REGLA 5

# 4. Implementación de un agente

## 4.3. Modificando el comportamiento del agente: ejemplo ilustrativo 3.

Sensores: *{bump\_ , trufa\_size\_}*

Variables de estado: *{orientacion\_ , obstaculo\_ , ult\_accion\_ }*

Almacena la acción seleccionada para usarla en la siguiente iteración.

```
36 // DECISION
37 ActionType accion;
38
39 if (trufa_size_>0){
40     accion = actEXTRACT;
41 }
42 else if (trufa_size_==1){
43     accion = actSNIFF;
44 }
45 else if (!obstaculo_[orientacion_]){
46     accion = actFORWARD;
47 }
48 else if (!obstaculo_[(orientacion_+1)%4]){
49     accion = actTURN_R;
50 }
51 else {
52     accion = actTURN_L;
53 }
54
55 ult_accion_ = accion;
56
57 return accion;
58 }
59 // -----
```



# 4. Implementación de un agente

## 4.3. Modificando el comportamiento del agente: ejemplo ilustrativo 3.

Sensores: **{bump\_ , trufa\_size\_}**

Variables de estado: **{orientacion\_ , obstaculo\_ , ult\_accion\_ }**

```
10 Agent::ActionType Agent::Think()
11 {
12     // ACTUALIZATION
13     switch(ult_accion_){
14         case actFORWARD:
15             if (bump_){
16                 obstaculo_[orientacion_]=true;
17             }
18             else {
19                 for (int i=0; i<4; i++)
20                     obstaculo_[i]=false;
21             }
22             break;
23         case actTURN_R:
24             orientacion ++;
25             if (orientacion_>3)
26                 orientacion_=0;
27             break;
28         case actTURN_L:
29             orientacion --;
30             if (orientacion_<0)
31                 orientacion_=3;
32             break;
33     }
34 }
35
```

```
36 // DECISION
37 ActionType accion;
38
39 if (trufa_size_>0){
40     accion = actEXTRACT;
41 }
42 else if (trufa_size_== -1){
43     accion = actSNIFF;
44 }
45 else if (!obstaculo_[orientacion_]){
46     accion = actFORWARD;
47 }
48 else if (!obstaculo_[(orientacion_+1)%4]){
49     accion = actTURN_R;
50 }
51 else {
52     accion = actTURN_L;
53 }
54
55 ult_accion_ = accion;
56
57 return accion;
58 }
59 // -----
```

# Índice

1. Introducción
2. Presentación del Problema
3. Presentación del Simulador
4. Implementación de un agente
5. Evaluación de la práctica



## 5. Evaluación de la práctica

1. ¿Qué hay que entregar?
2. ¿Qué debe contener la memoria de la práctica?
3. ¿Cómo se evalúa la práctica?
4. ¿Dónde y cuándo se entrega?

## 5. Evaluación de la práctica

¿Qué hay que entregar?

Un único archivo comprimido zip que llamado “***practica2.zip***” contenga sin carpetas los tres ficheros siguientes:

- La memoria de la práctica (en formato pdf)
- Los ficheros “***agent.cpp***” y “***agent.h***” propuestos como solución.

***No ficheros ejecutables***

## 5. Evaluación de la práctica

¿Qué debe contener la memoria de la práctica?

1. Descripción de la solución propuesta.
2. Tabla con los resultados obtenidos sobre los distintos mapas.
3. Descripción de otras estrategias descartadas.

***Documento 5 páginas máximo***

# 5. Evaluación de la práctica

## ¿Cómo se evalúa?

Se tendrán en cuenta tres aspectos:

1. El documento de la memoria de la práctica
  - se evalúa **de 0 a 3 puntos**.
2. La defensa de la práctica
  - se evalúa **APTO** o **NO APTO**. APTO equivale a 3 puntos, NO APTO implica tener un **0** en esta práctica.
3. Evaluación de la solución propuesta
  - se evalúa **de 0 a 4**.
  - el valor concreto es el resultado de interpolar entre la mejor y la peor solución encontrada.

# 5. Evaluación de la práctica

## ¿Cómo se evalúa?

Sobre la solución propuesta (*hasta 4 puntos*) :

- Será evaluada sobre un mapa distinto a los aportados junto con el material de la práctica.
- Ni la propuesta de agente que aparece en el guión, ni la que se incluye en esta presentación serán propuestas válidas entregables como solución a la práctica.
- Sólo se considerará en la evaluación del comportamiento del agente la media de las trufas recolectadas sobre las 10 ejecuciones, es decir el valor de

***“Average Total Collected”***

## 5. Evaluación de la práctica

¿Dónde y cuándo se entrega?

- Se entrega en la aplicación de gestión de prácticas de la asignatura **decsai.ugr.es → Entrega de Prácticas**

Grupos	Fecha límite
A3, B3, C3 y D2	13 de Abril hasta las 23:59H
A1, B1 y Doble Grado	14 de Abril hasta las 23:59H
C1	16 de Abril hasta las 23:59H
A2, B2, C2 y D1	17 de Abril hasta las 23:59H