



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 2013/2014



DECSAI

**Departamento de Ciencias
de la Computación e I.A.**

Universidad de Granada

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. 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

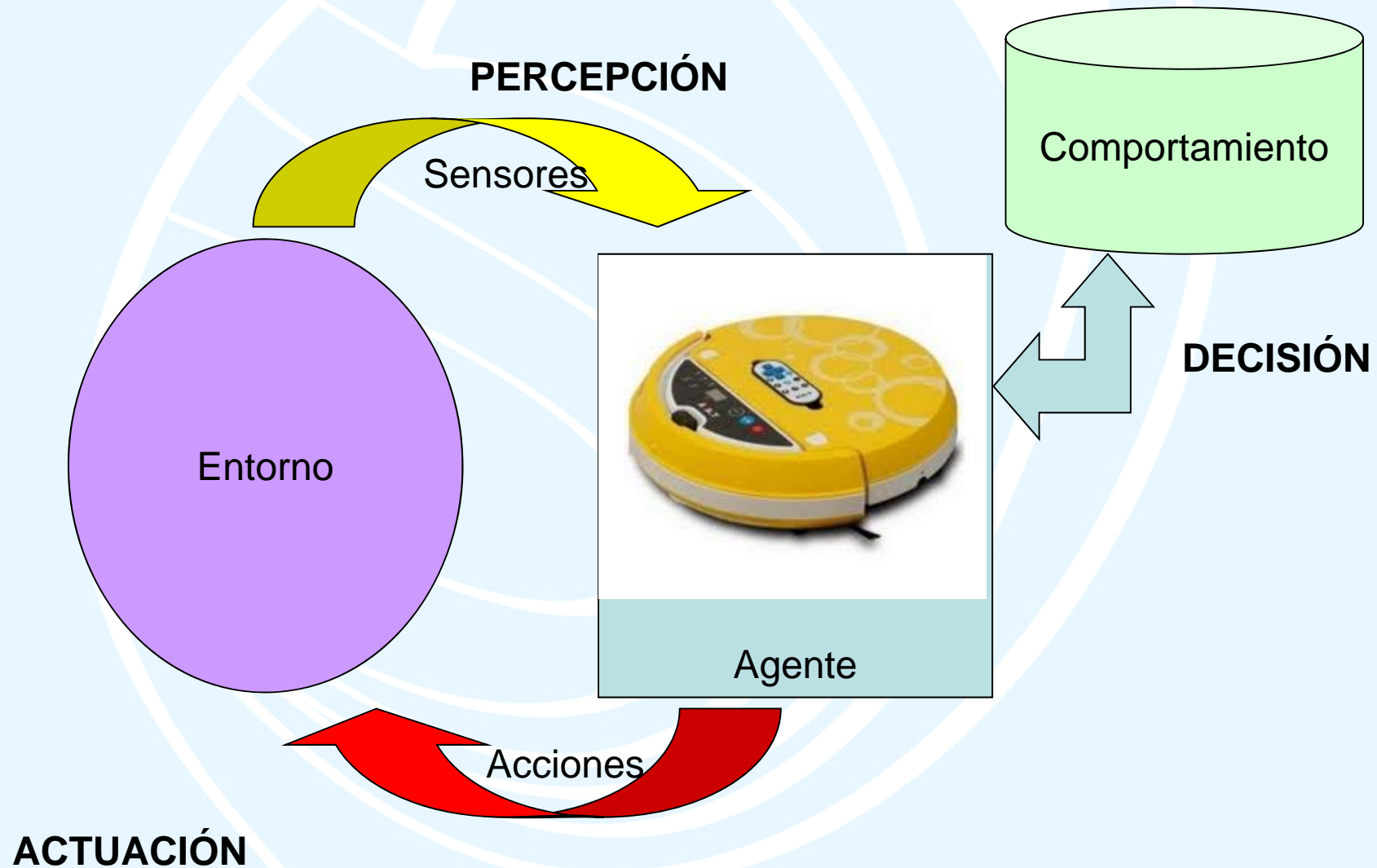
1. Introducción
2. **Presentación del Problema**
3. Presentación del Simulador
4. Implementación de un agente
5. Evaluación de la práctica

2. Presentación del Problema

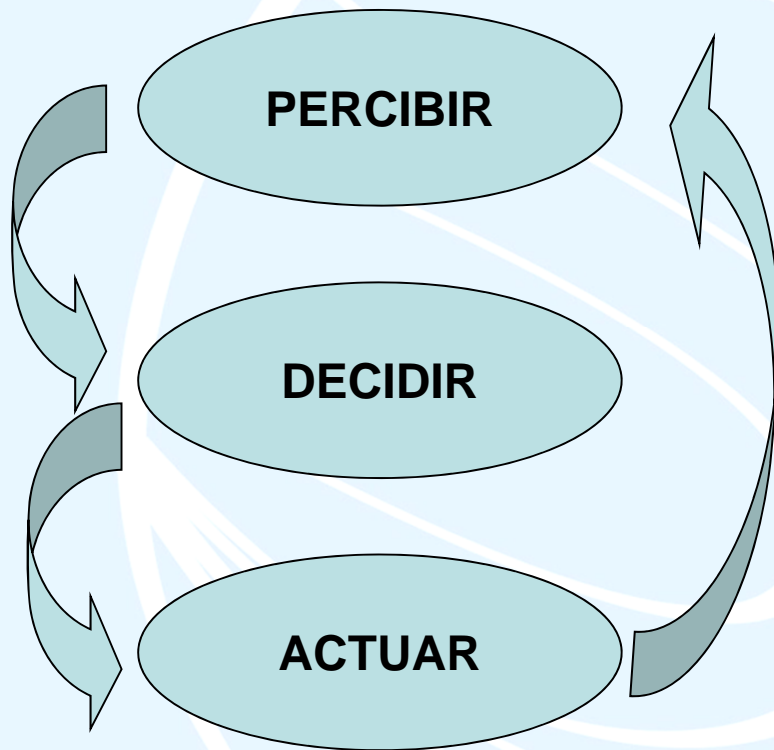
- Aspiradora Inteligente



2. Presentación del Problema



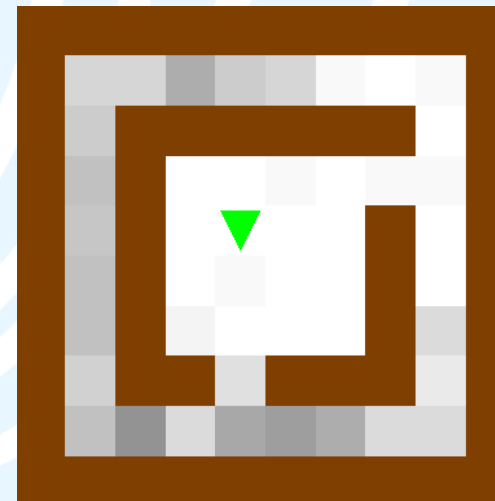
2. Presentación del Problema



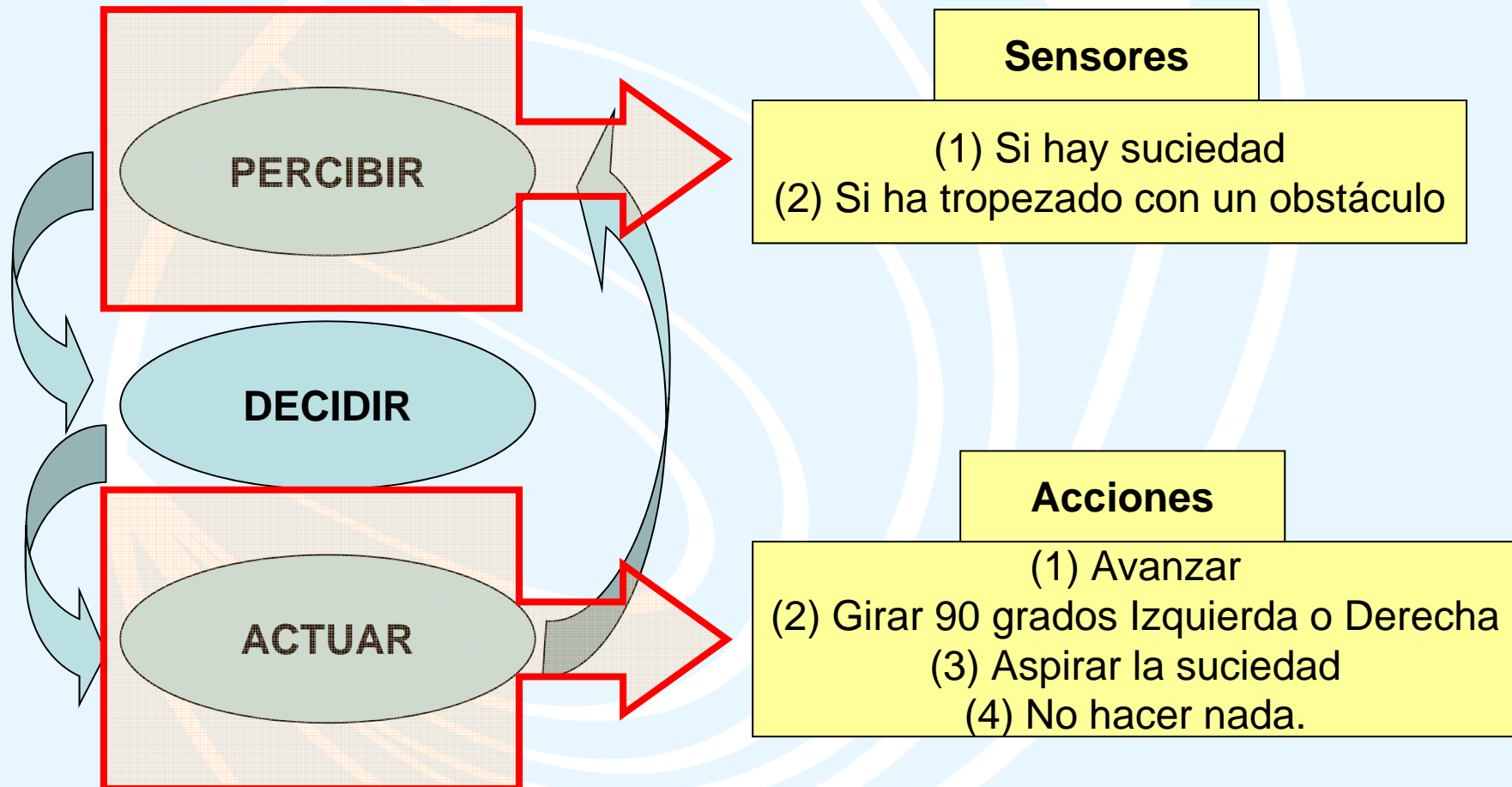
Controlador de ciclo cerrado

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

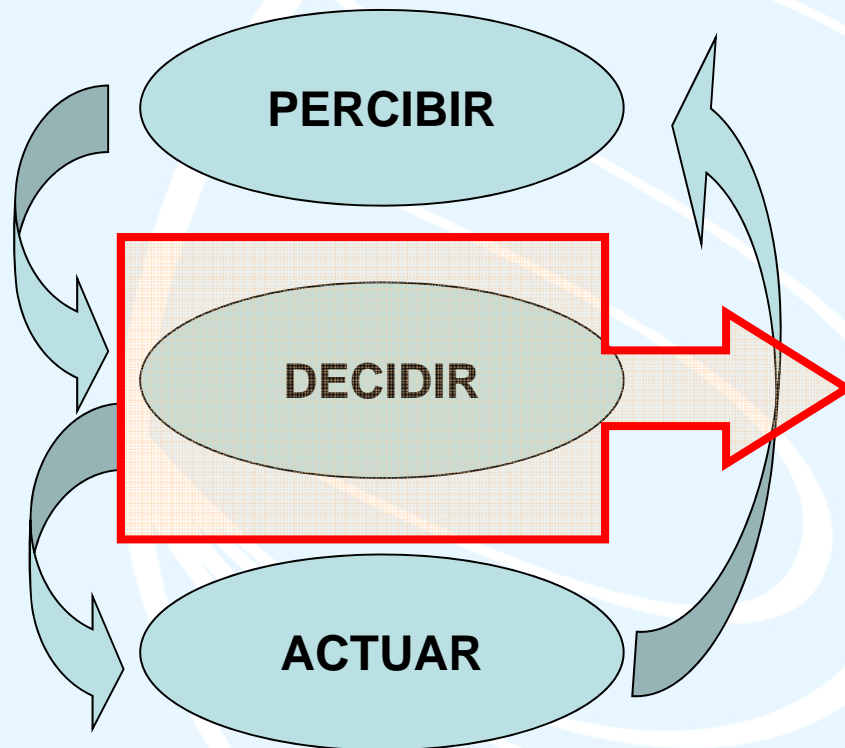
- Lo que es capaz de percibir el agente del entorno y,
- las acciones que el agente puede realizar.



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. Si no es así, en el guión de la práctica se indica como proceder a su instalación.

1. Cread la carpeta “U:\IA\practica2”
1. Descargar **Agent-P2_2013-14.rar** desde la [web](#) de la asignatura y cópielo en la carpeta



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

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

3. Presentación del Simulador

3.1. Compilación del Simulador

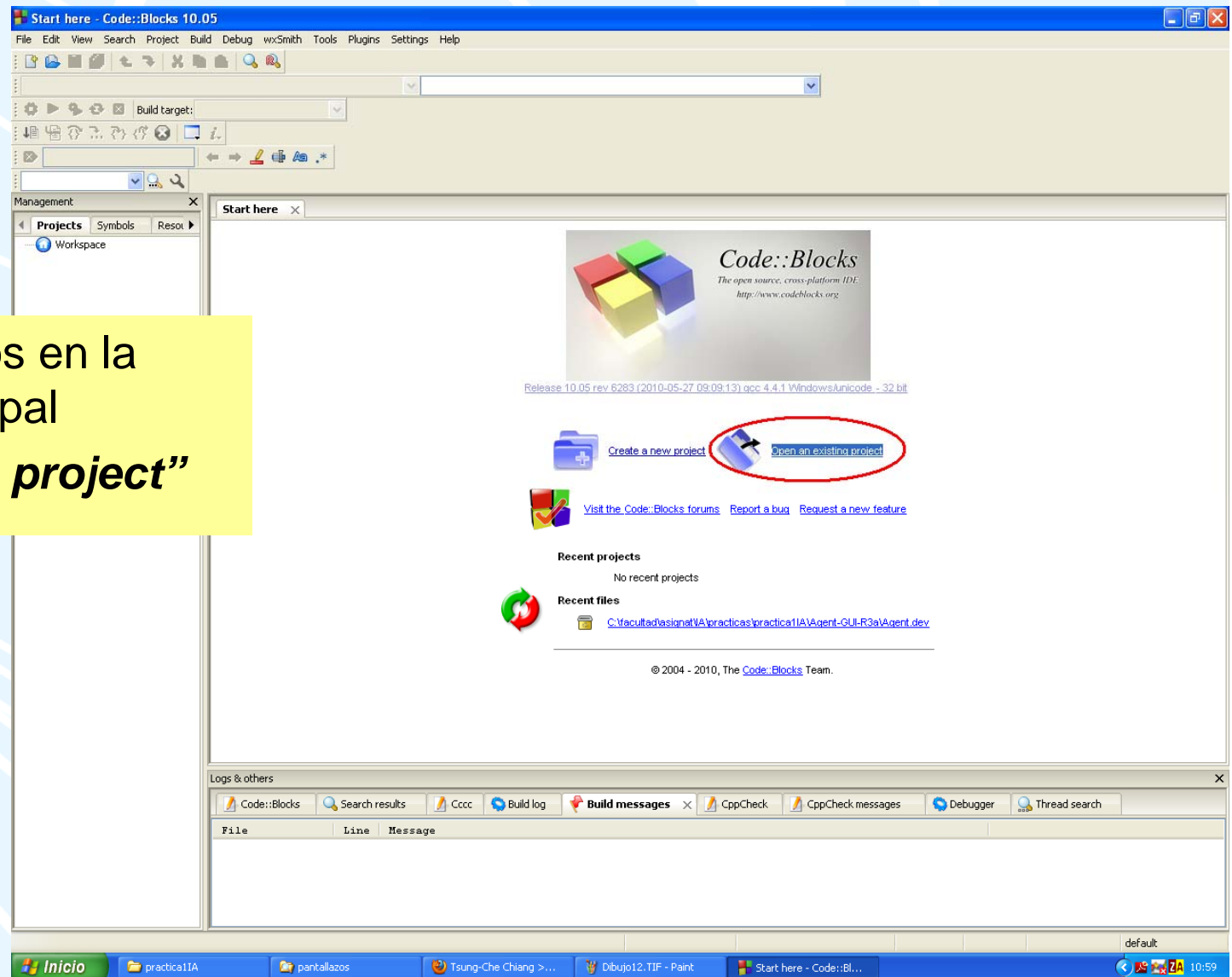
1. Abrimos “CodeBlocks”

- 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

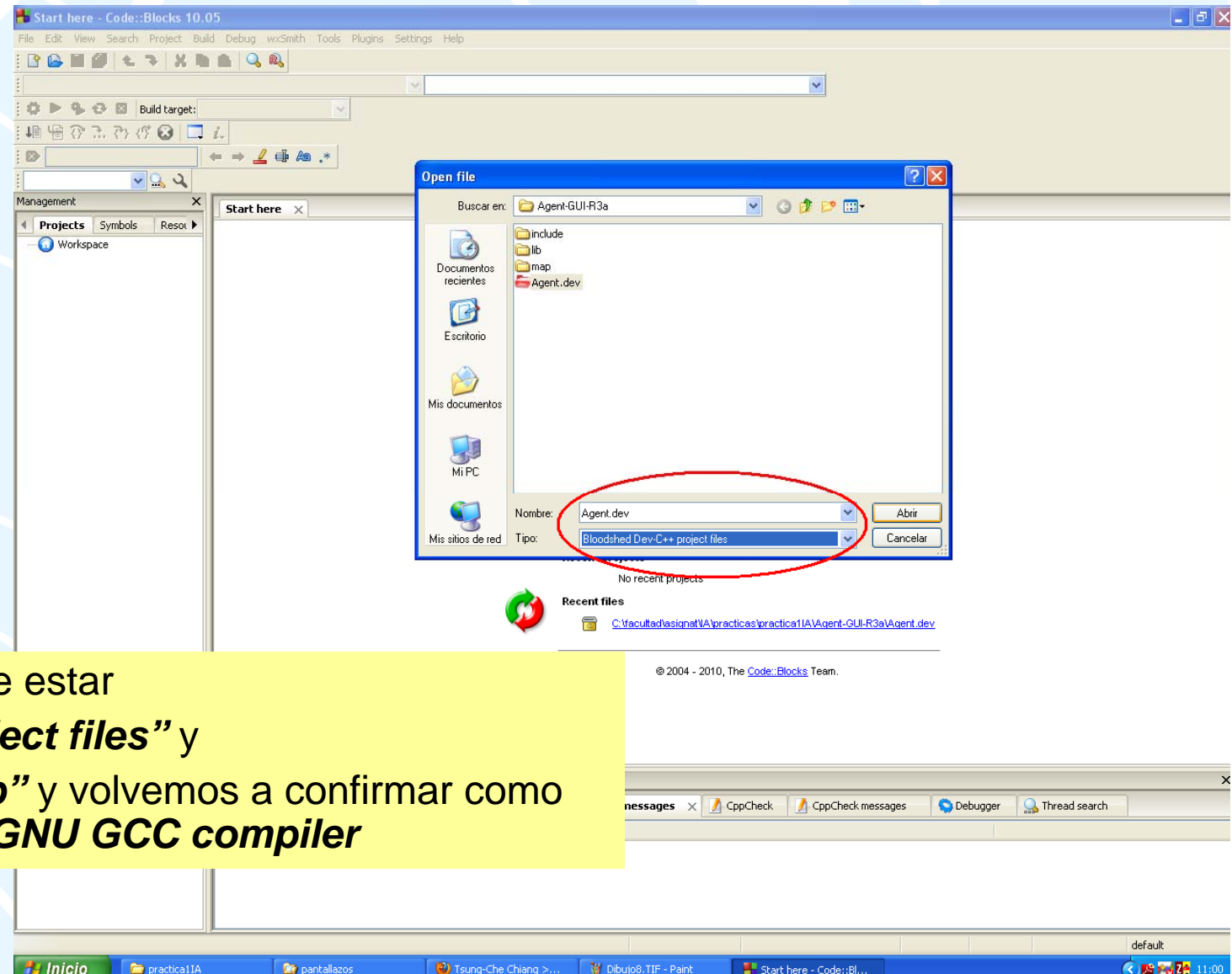
3.1. Compilación del Simulador

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



3. Presentación del Simulador

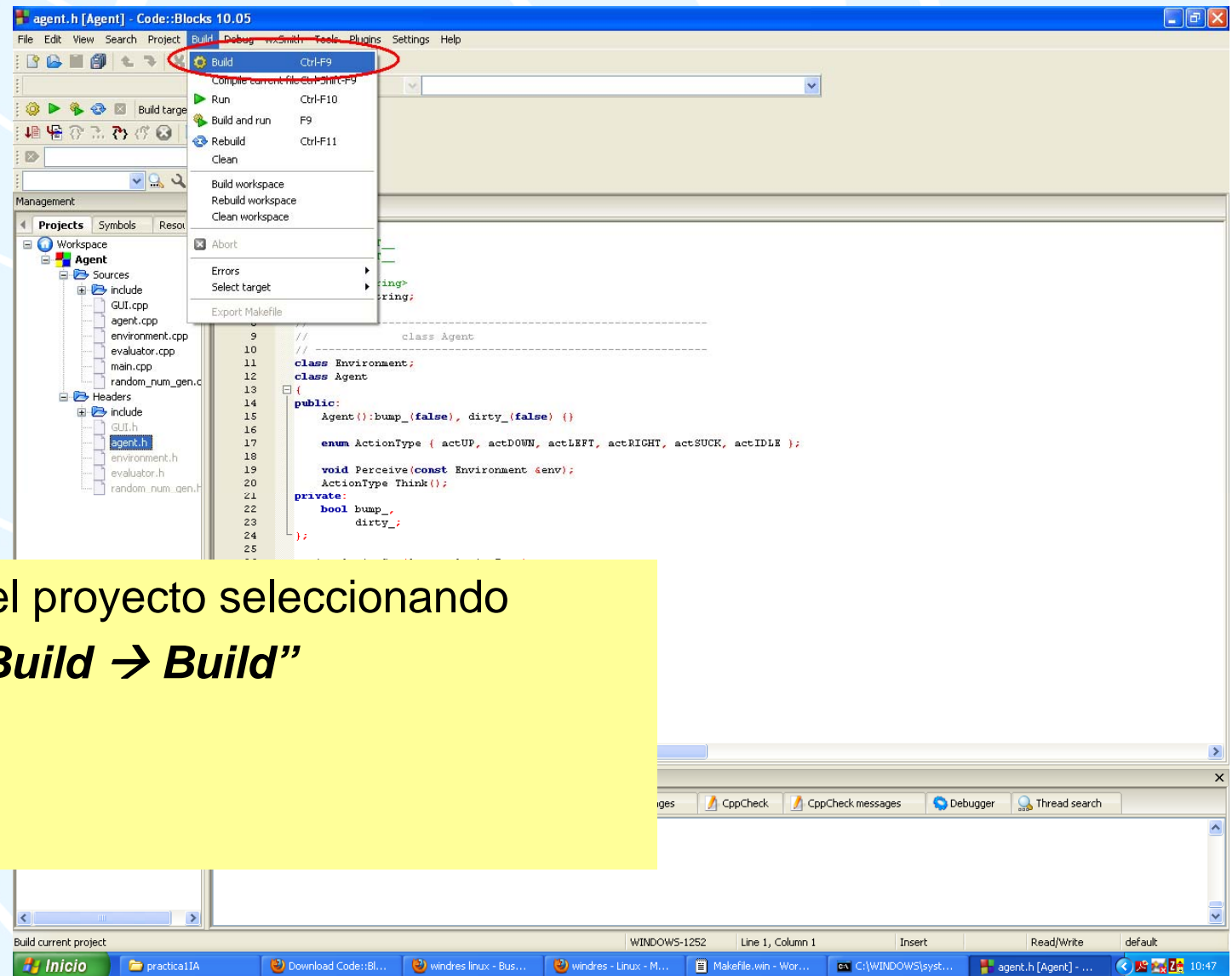
3.1. Compilación del Simulador



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

3. Presentación del Simulador

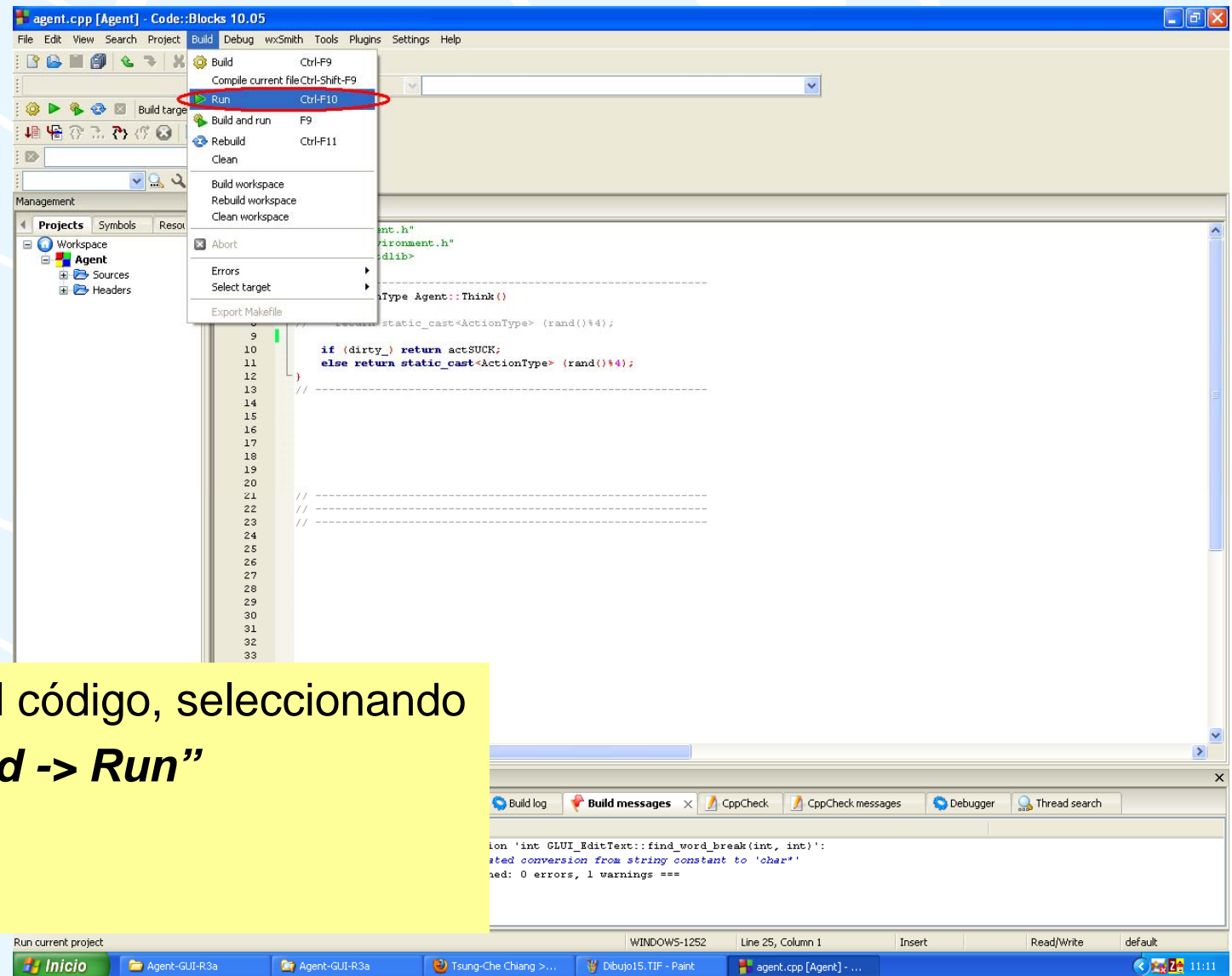
3.1. Compilación del Simulador



1. Compilamos el proyecto seleccionando
“Build → Build”

3. Presentación del Simulador

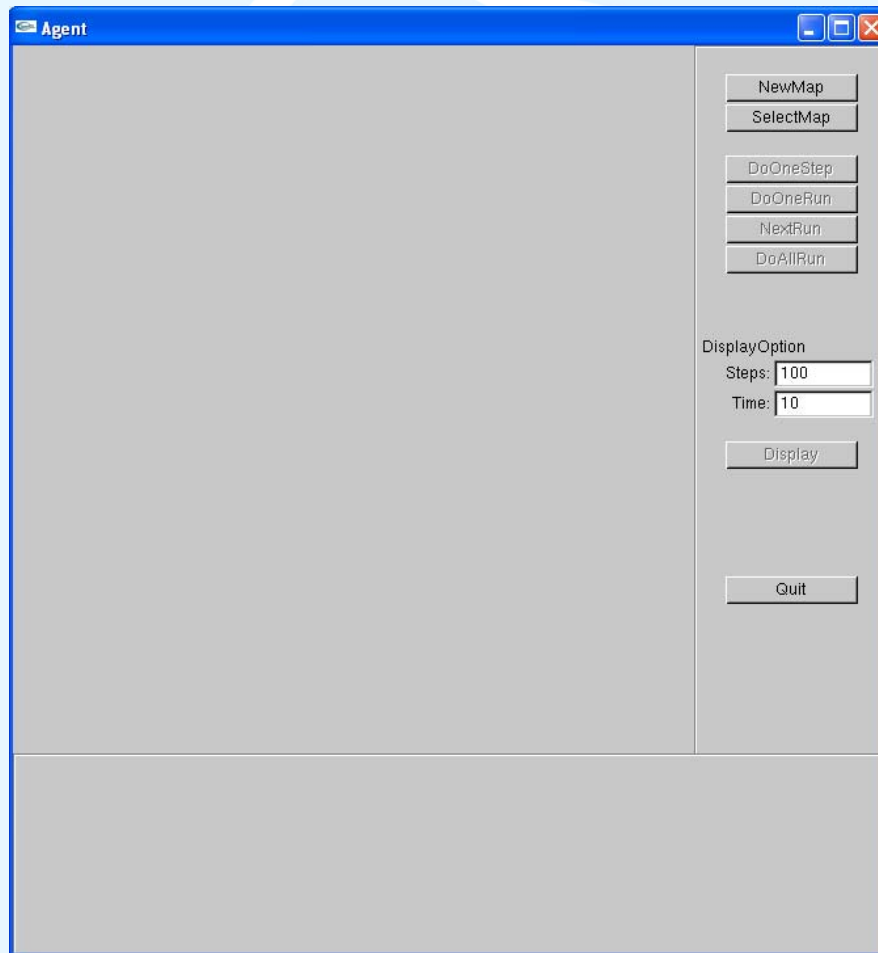
3.1. Compilación del Simulador



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

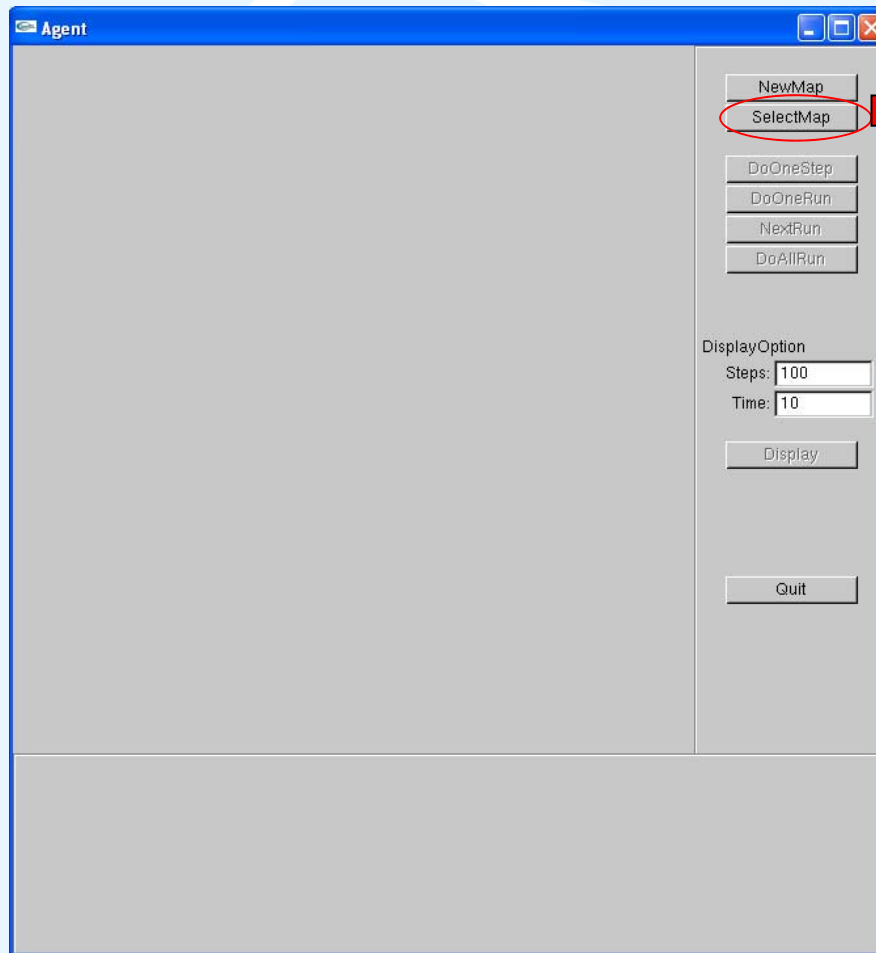
3. Presentación del Simulador

3.2. Ejecución del Simulador



3. Presentación del Simulador

3.2. Ejecución del Simulador

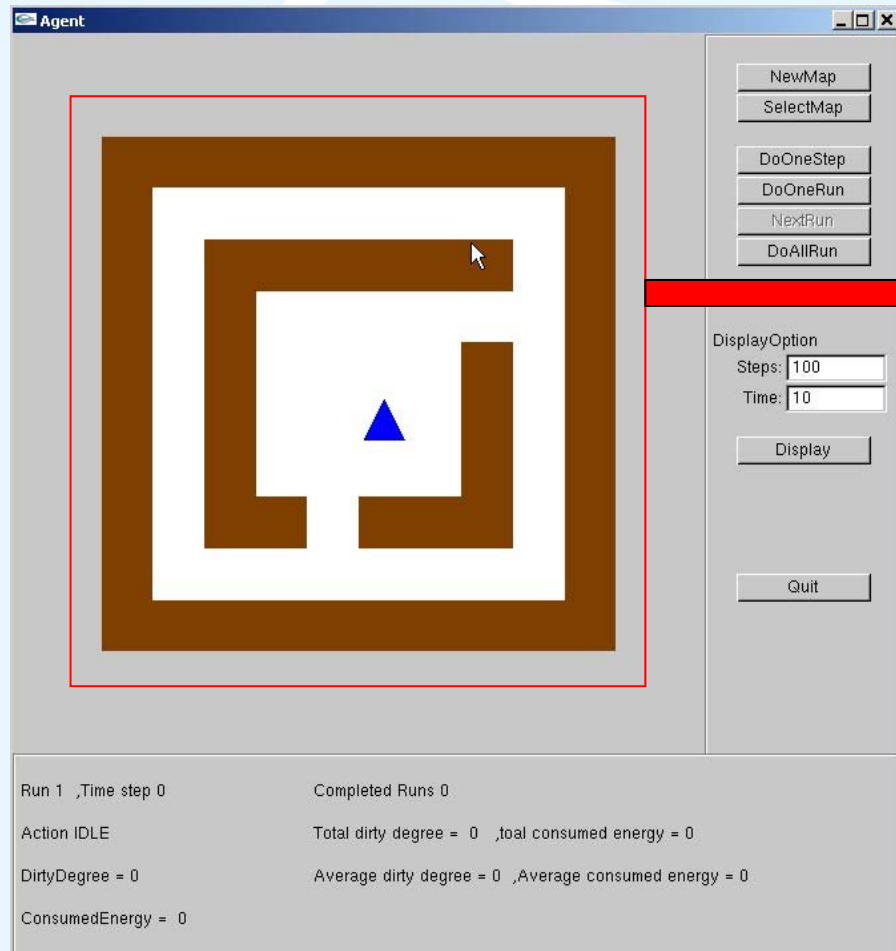


“SelectMap” Selecciona el fichero de mapa sobre el que Realizar la simulación.

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

3. Presentación del Simulador

3.2. Ejecución del Simulador

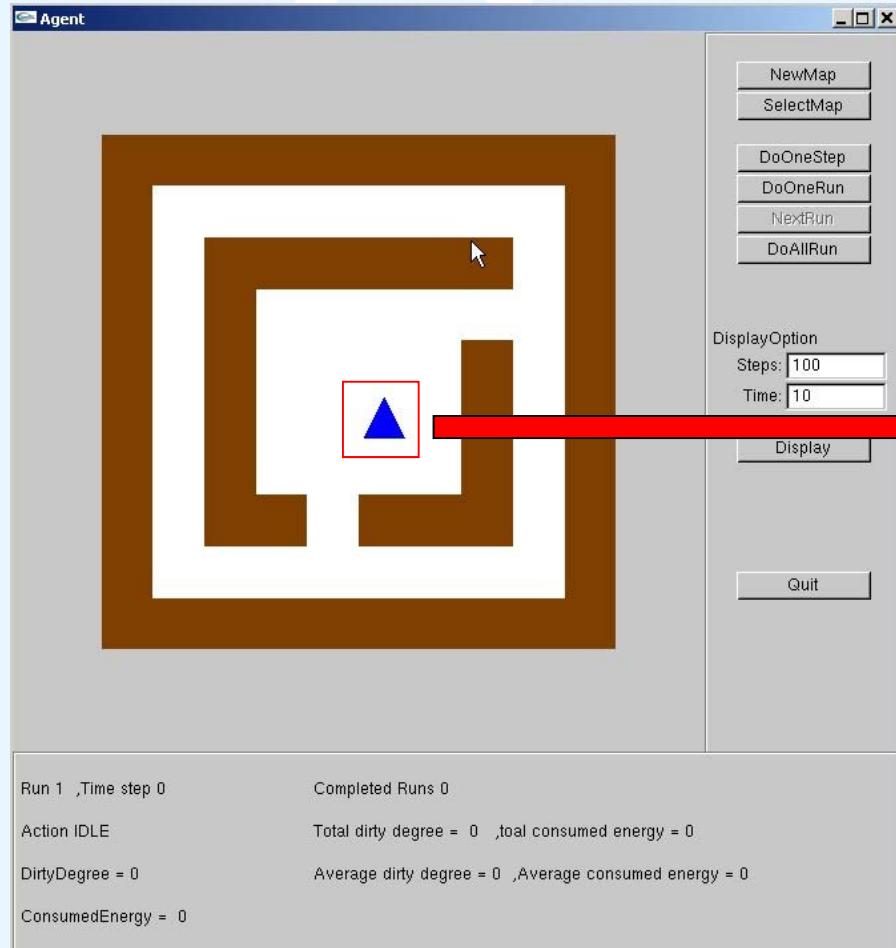


Mundo simulado:

- 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



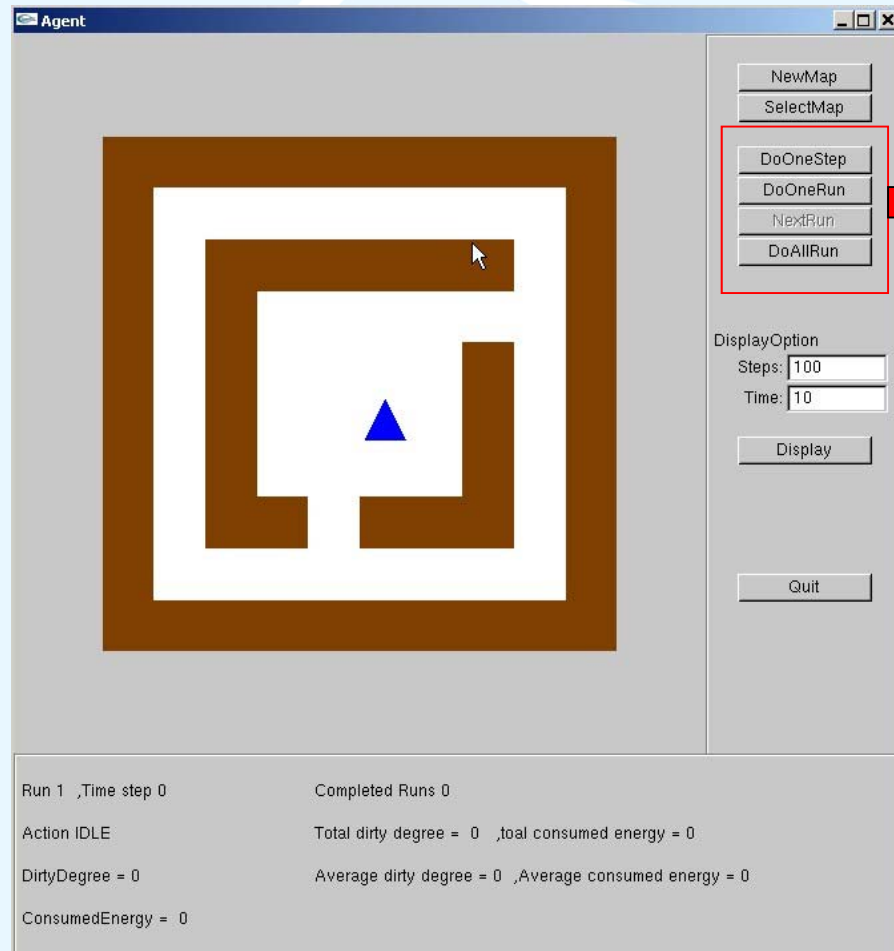
Aspiradora:

Los diferentes estados representan el resultado de la última acción:

-  **Aspiró** la suciedad 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.

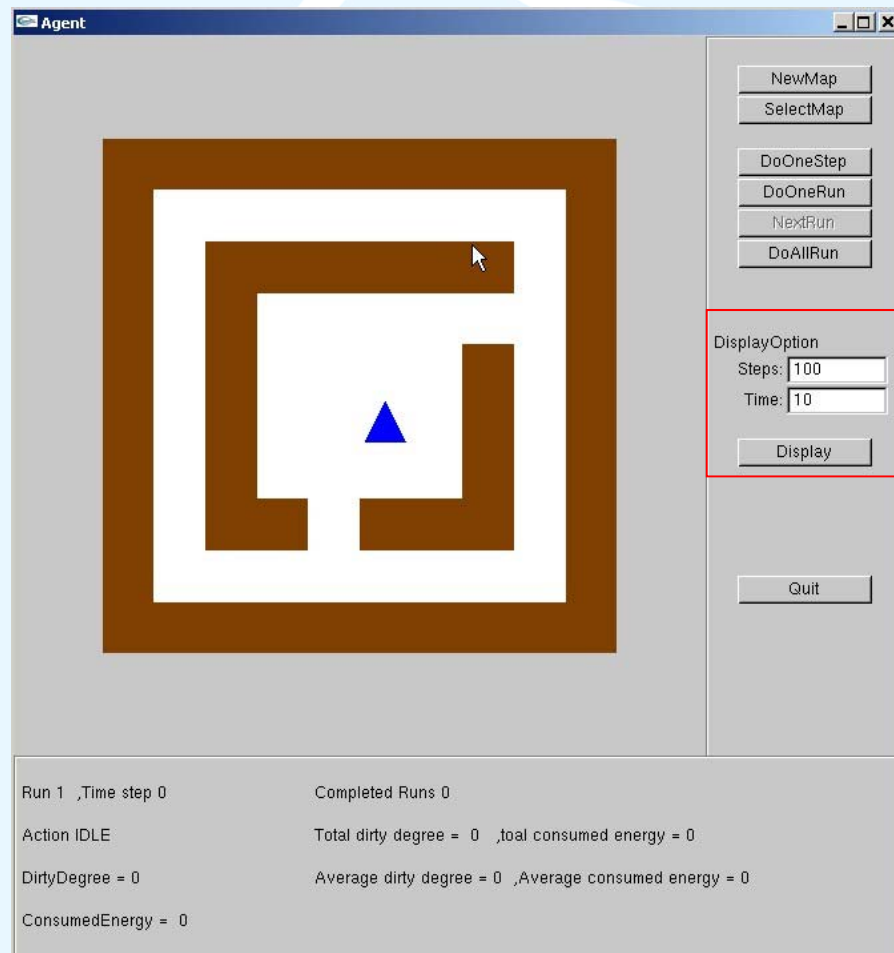
“DoOneRun” Produce una simulación completa.

“NextRun” Pasa a la siguiente ejecución.

“DoAllRun” Ejecuta todas las ejecuciones de la simulación”

3. Presentación del Simulador

3.2. Ejecución del Simulador



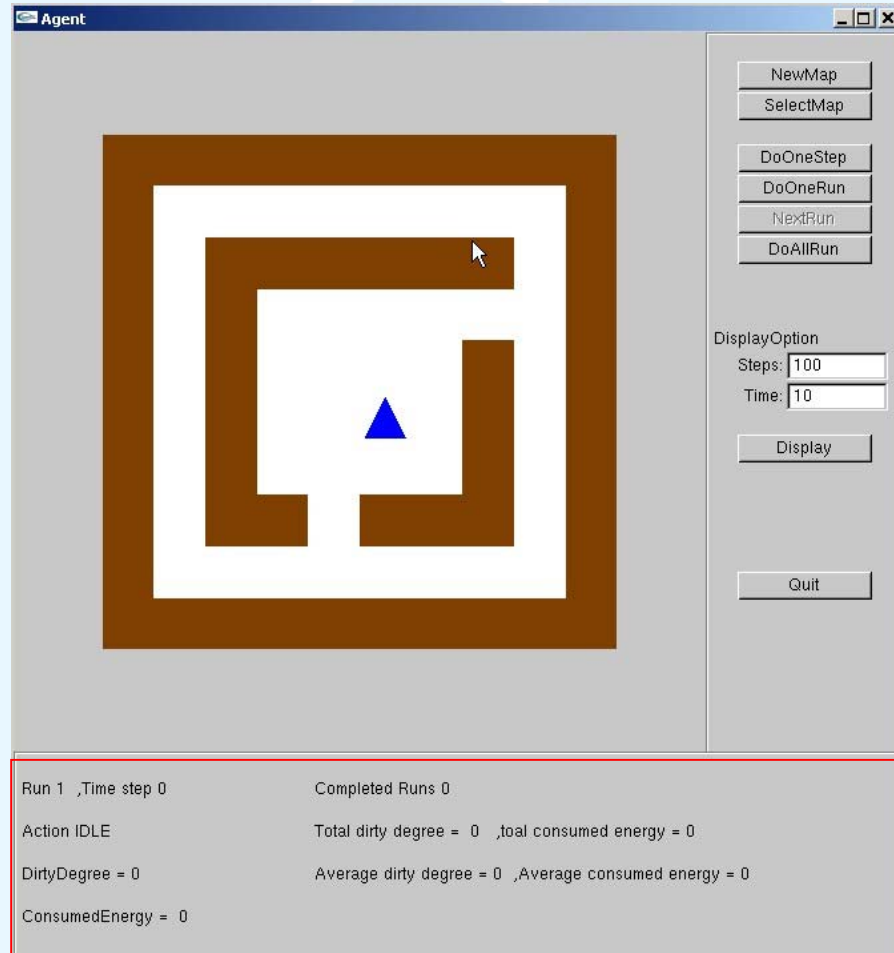
“Display”

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 suciedad actual.
- Cantidad de energía consumida.
- Ejecuciones ya completadas
- Grado de suciedad y cantidad de energía en la última ejecución.
- Media del grado de suciedad y media de la cantidad de energía consumida en las ejecuciones ya completadas.

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.

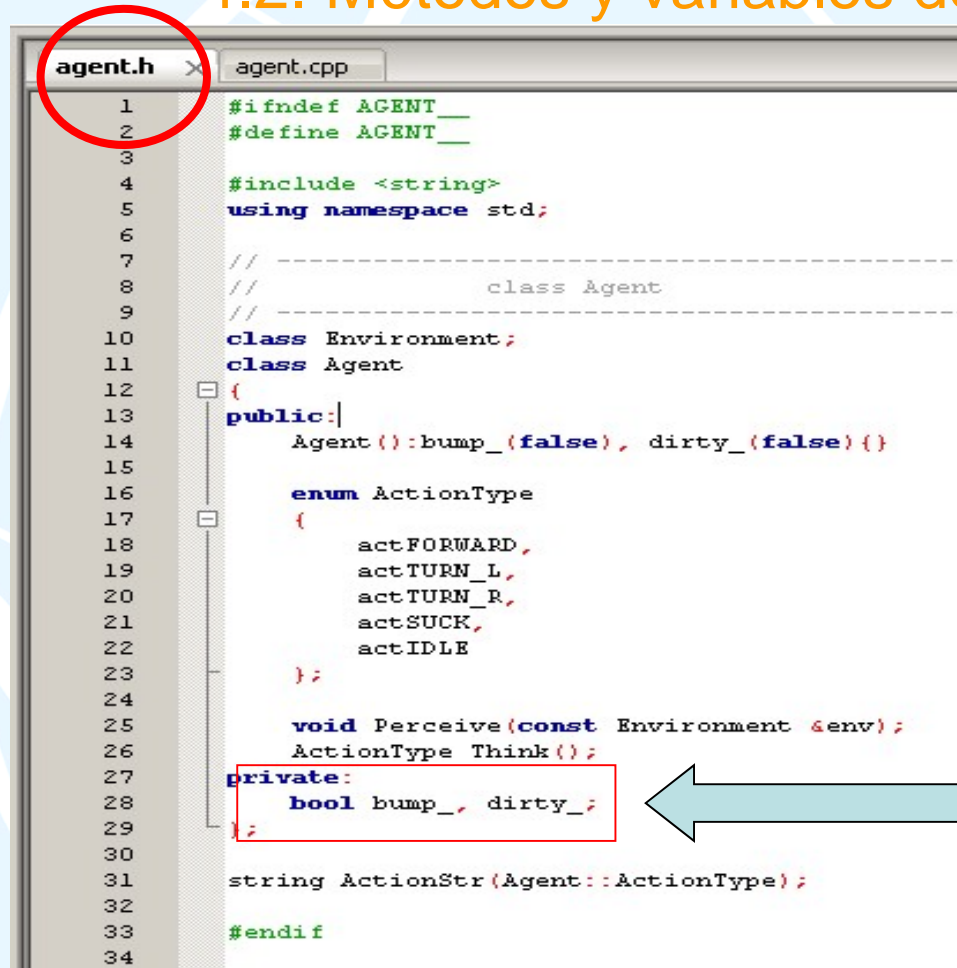
4. Implementación de un agente

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
 - **dirty_** (boolean) *true* indica que el sensor ha detectado suciedad en la casilla actual.

4. Implementación de un agente

4.2. Métodos y variables del agente



```
1  #ifndef AGENT__
2  #define AGENT__
3
4  #include <string>
5  using namespace std;
6
7  // -----
8  //             class Agent
9  // -----
10 class Environment;
11 class Agent
12 {
13 public:
14     Agent(): bump_(false), dirty_(false) {}
15
16     enum ActionType
17     {
18         actFORWARD,
19         actTURN_L,
20         actTURN_R,
21         actSUCK,
22         actIDLE
23     };
24
25     void Perceive(const Environment &env);
26     ActionType Think();
27 private:
28     bool bump_, dirty_;
29 };
30
31 string ActionStr(Agent::ActionType);
32
33 #endif
34
```

Variables *bump_* y *dirty_*

4. Implementación de un agente

4.2. Métodos y variables del agente

```
agent.h x agent.cpp
1  #ifndef AGENT_
2  #define AGENT_
3
4  #include <string>
5  using namespace std;
6
7  // -----
8  //          class Agent
9  // -----
10 class Environment;
11 class Agent
12 {
13 public:
14     Agent(): bump_(false), dirty_(false) {}
15
16     enum ActionType
17     {
18         actFORWARD,
19         actTURN_L,
20         actTURN_R,
21         actSUCK,
22         actIDLE
23     };
24
25     void Perceive(const Environment &env);
26     ActionType Think();
27 private:
28     bool bump_, dirty_;
29 };
30
31 string ActionStr(Agent::ActionType);
32
33 #endif
34
```

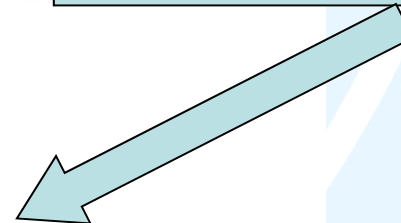
Constructor de Clase
Pone a *false* bump_ y dirty_

4. Implementación de un agente

4.2. Métodos y variables del agente

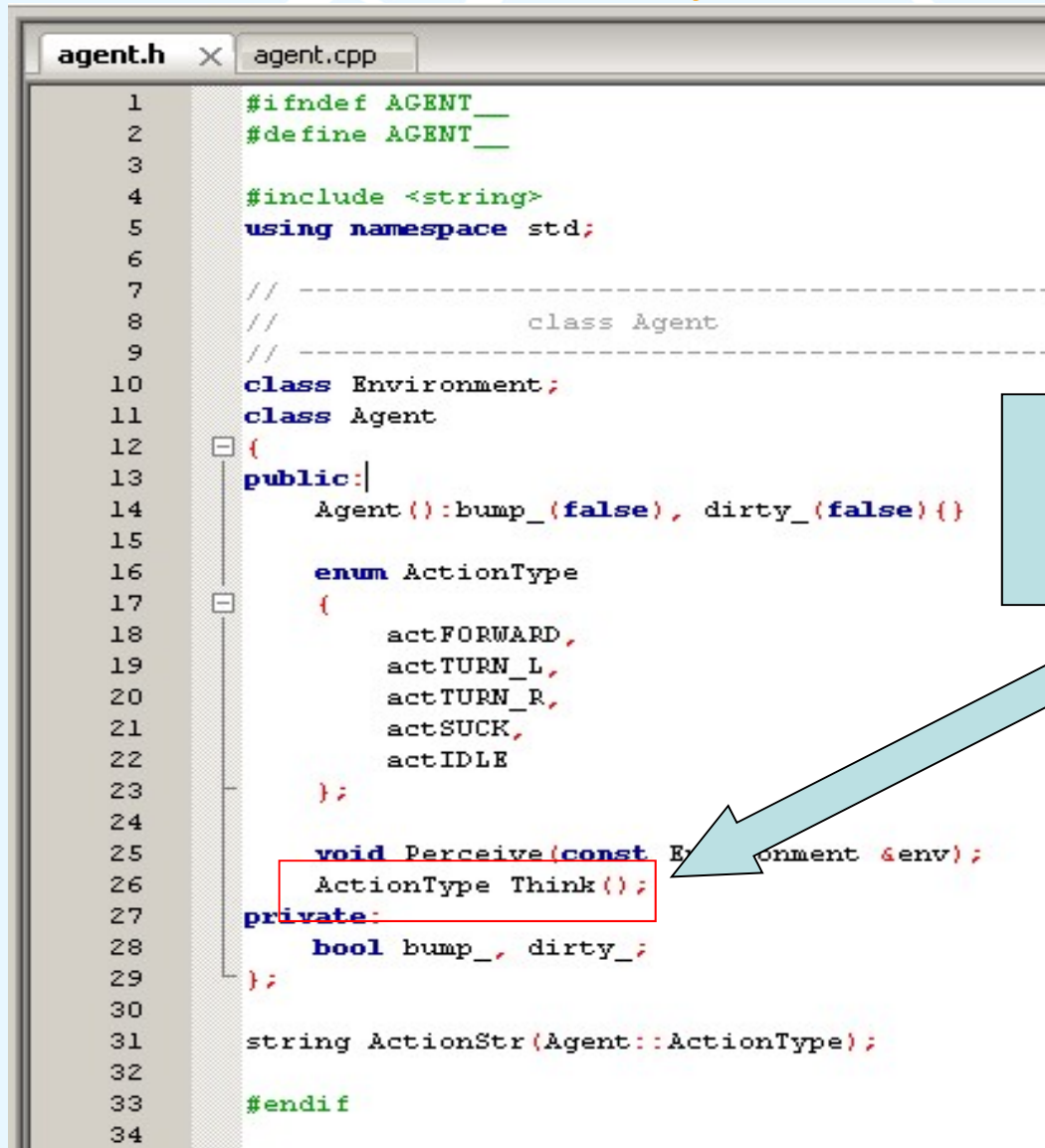
```
agent.h  x  agent.cpp
1  #ifndef AGENT__
2  #define AGENT__
3
4  #include <string>
5  using namespace std;
6
7  // -----
8  //          class Agent
9  // -----
10 class Environment;
11 class Agent
12 {
13 public:
14     Agent(): bump_(false), dirty_(f
15
16     enum ActionType
17     {
18         actFORWARD,
19         actTURN_L,
20         actTURN_R,
21         actSUCK,
22         actIDLE
23     };
24
25     void Perceive(const Environment &env);
26     ActionType Think();
27 private:
28     bool bump_, dirty_;
29 };
30
31 string ActionStr(Agent::ActionType);
32
33 #endif
34
```

Esta función toma una variable de tipo **Environment** que representa la situación actual del entorno, y da valor a las variables **bump_** y **dirty_**



4. Implementación de un agente

4.2. Métodos y variables del agente



```
1  #ifndef AGENT__
2  #define AGENT__
3
4  #include <string>
5  using namespace std;
6
7  // -----
8  //                               class Agent
9  // -----
10 class Environment;
11 class Agent
12 {
13 public:
14     Agent(): bump_(false), dirty_(false) {}
15
16     enum ActionType
17     {
18         actFORWARD,
19         actTURN_L,
20         actTURN_R,
21         actSUCK,
22         actIDLE
23     };
24
25     void Perceive(const Environment &env);
26     ActionType Think();
27 private:
28     bool bump_, dirty_;
29 };
30
31 string ActionStr(Agent::ActionType);
32
33 #endif
34
```

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

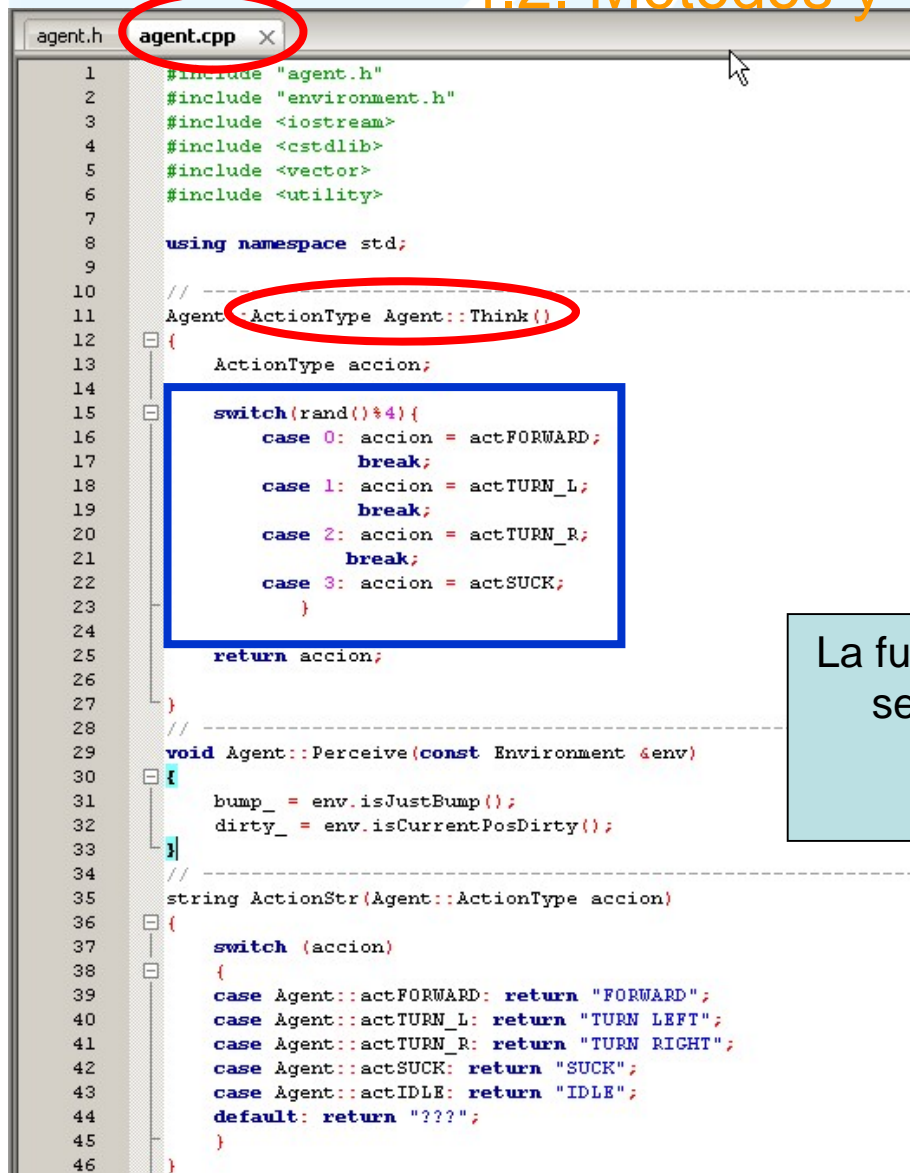
```
agent.h  x  agent.cpp
1  #ifndef AGENT__
2  #define AGENT__
3
4  #include <string>
5  using namespace std;
6
7  // -----
8  //      class Agent
9  // -----
10 class Environment;
11 class Agent
12 {
13 public:
14     Agent(): bump_(false), dirty_(false) {}
15
16     enum ActionType
17     {
18         actFORWARD,
19         actTURN_L,
20         actTURN_R,
21         actSUCK,
22         actIDLE
23     };
24
25     void Perceive(const Environment &env);
26     ActionType Think();
27 private:
28     bool bump_, dirty_;
29 };
30
31 string ActionStr(Agent::ActionType);
32
33 #endif
34
```

Las acciones posibles son:

actFORWARD,
actTURN_L,
actTURN_R,
actSUCK,
actIDLE

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 // -----
11 Agent::ActionType Agent::Think()
12 {
13     ActionType accion;
14
15     switch(rand()%4){
16         case 0: accion = actFORWARD;
17                 break;
18         case 1: accion = actTURN_L;
19                 break;
20         case 2: accion = actTURN_R;
21                 break;
22         case 3: accion = actSUCK;
23                 }
24
25     return accion;
26 }
27
28 // -----
29 void Agent::Perceive(const Environment &env)
30 {
31     bump_ = env.isJustBump();
32     dirty_ = env.isCurrentPosDirty();
33 }
34
35 // -----
36 string ActionStr(Agent::ActionType accion)
37 {
38     switch (accion)
39     {
40     case Agent::actFORWARD: return "FORWARD";
41     case Agent::actTURN_L: return "TURN LEFT";
42     case Agent::actTURN_R: return "TURN RIGHT";
43     case Agent::actSUCK: return "SUCK";
44     case Agent::actIDLE: return "IDLE";
45     default: return "???";
46     }
47 }
```

Las acciones posibles son:

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 esta sucia que entonces la limpie.

```
if (dirty_) accion = actSUCK;
```

4. Implementación de un agente

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

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

Incluyo la excepción en el caso de estar en una casilla sucia

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

4. Implementación de un agente

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

Ahora no queremos avanzar más de dos veces seguidas sin cambiar la dirección, a menos que el avance provoque un choque.

4. Implementación de un agente

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

```
agent.h x agent.cpp
1  #ifndef AGENT__
2  #define AGENT__
3
4  #include <string>
5  using namespace std;
6
7  // -----
8  //           class Agent
9  // -----
10 class Environment;
11 class Agent
12 {
13 public:
14     Agent(): bump_(false), dirty_(false), n_avances_(0) {}
15
16     enum ActionType
17     {
18         actFORWARD,
19         actTURN_L,
20         actTURN_R,
21         actSUCK,
22         actIDLE
23     };
24
25     void Perceive(const Environment &env);
26     ActionType Think();
27 private:
28     bool bump_, dirty_;
29     int n_avances_;
30 };
31
32 string ActionStr(Agent::ActionType);
33
34 #endif
35
```

Añado un nuevo dato miembro
“*int n_avances_*” y
lo inicializo con valor 0.

4. Implementación de un agente

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

```
10 // -----
11 Agent::ActionType Agent::Think()
12 {
13     ActionType accion;
14
15     if (dirty_)
16         accion = actSUCK;
17     else {
18         if (bump_ or n_avances_>2){
19             switch(rand()%2+1){
20                 case 1: accion = actTURN_L;
21                     break;
22                 case 2: accion = actTURN_R;
23             }
24             n_avances_=0;
25         }
26         else {
27             switch(rand()%3){
28                 case 0: accion = actFORWARD;
29                     n_avances_++;
30                     break;
31                 case 1: accion = actTURN_L;
32                     n_avances_=0;
33                     break;
34                 case 2: accion = actTURN_R;
35                     n_avances_=0;
36             }
37         }
38     }
39     return accion;
40
41 }
```

En el método `Think()` de `agent.cpp` mantengo la condición de aspirar

4. Implementación de un agente

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

```
10 // -----
11 Agent::ActionType Agent::Think()
12 {
13     ActionType accion;
14
15     if (dirty_)
16         accion = actSUCK;
17     else {
18         if (bump_ or n_avances_>2){
19             switch(rand()%2+1){
20                 case 1: accion = actTURN_L;
21                     break;
22                 case 2: accion = actTURN_R;
23             }
24             n_avances_=0;
25         }
26         else {
27             switch(rand()%3){
28                 case 0: accion = actFORWARD;
29                     n_avances_++;
30                     break;
31                 case 1: accion = actTURN_L;
32                     n_avances_=0;
33                     break;
34                 case 2: accion = actTURN_R;
35                     n_avances_=0;
36             }
37         }
38     }
39     return accion;
40
41 }
```

Si no hay suciedad y se ha producido una colisión o se avanzó más de 2 veces seguidas, entonces al azar cambio la orientación del robot

4. Implementación de un agente

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

```
10 // -----
11 Agent::ActionType Agent::Think()
12 {
13     ActionType accion;
14
15     if (dirty_)
16         accion = actSUCK;
17     else {
18         if (bump_ or n_avances_>2){
19             switch(rand()%2+1){
20                 case 1: accion = actTURN_L;
21                     break;
22                 case 2: accion = actTURN_R;
23             }
24             n_avances_=0;
25         }
26         else {
27             switch(rand()%3){
28                 case 0: accion = actFORWARD;
29                     n_avances_++;
30                     break;
31                 case 1: accion = actTURN_L;
32                     n_avances_=0;
33                     break;
34                 case 2: accion = actTURN_R;
35                     n_avances_=0;
36             }
37         }
38     }
39     return accion;
40
41 }
```

Si no se da nada de lo anterior,
al azar selecciono la siguiente acción

4. Implementación de un agente

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

```
10 // -----
11 Agent::ActionType Agent::Think()
12 {
13     ActionType accion;
14
15     if (dirty_)
16         accion = actSUCK;
17     else {
18         if (bump_ or n_avances_>2){
19             switch(rand()%2+1){
20                 case 1: accion = actTURN_L;
21                     break;
22                 case 2: accion = actTURN_R;
23             }
24             n_avances_=0;
25         }
26         else {
27             switch(rand()%3){
28                 case 0: accion = actFORWARD;
29                     n_avances_++;
30                     break;
31                 case 1: accion = actTURN_L;
32                     n_avances_=0;
33                     break;
34                 case 2: accion = actTURN_R;
35                     n_avances_=0;
36             }
37         }
38     }
39     return accion;
40
41 }
```

IMPORTANTE:
Actualizar de forma coherente
el dato miembro *n_avances_*

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**, llamado “***practica2.zip***”, sin carpetas que contenga:

- La memoria de la práctica (en formato pdf)
- Los archivos “***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. Análisis del problema
2. Descripción de la solución propuesta
3. Tabla con los resultados obtenidos sobre los distintos mapas.
4. Opcionalmente código fuente de los archivos ***“agent.cpp”*** y ***“agent.h”***

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 el nivel medio de suciedad sobre las 10 ejecuciones, es decir el valor de

“Average dirty degree”

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
- La fecha tope de entrega es el **miércoles 23 de abril a las 23:59**