



ugr

Universidad  
de Granada

# SEMINARIO 2

## Presentación Práctica 1

### Agentes Reactivos

**Inteligencia Artificial**

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

ETSI Informática y de Telecomunicación  
UNIVERSIDAD DE GRANADA

*Curso 2011/2012*



**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. 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



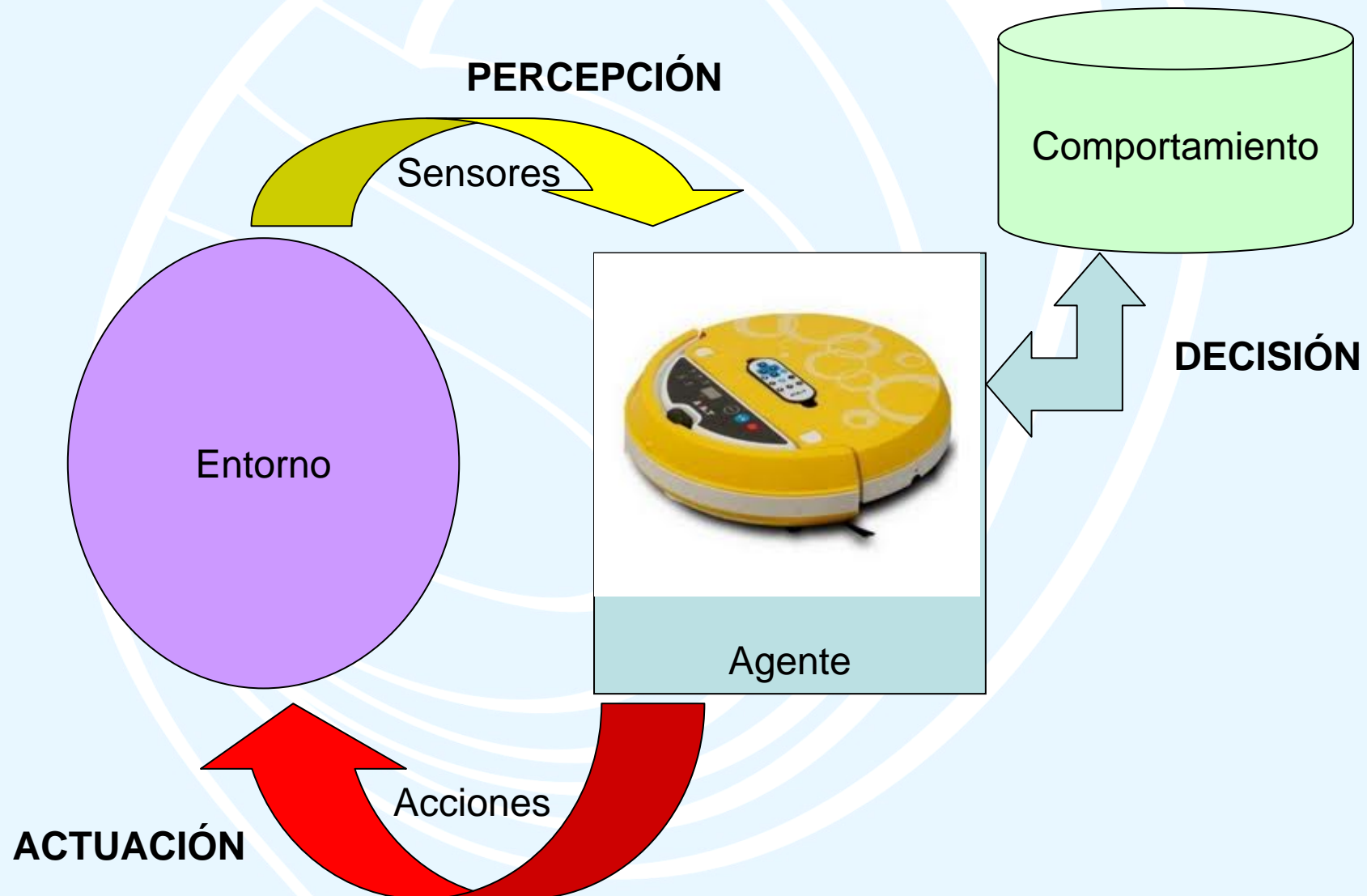
## 2. Presentación del Problema

- Aspiradora Inteligente

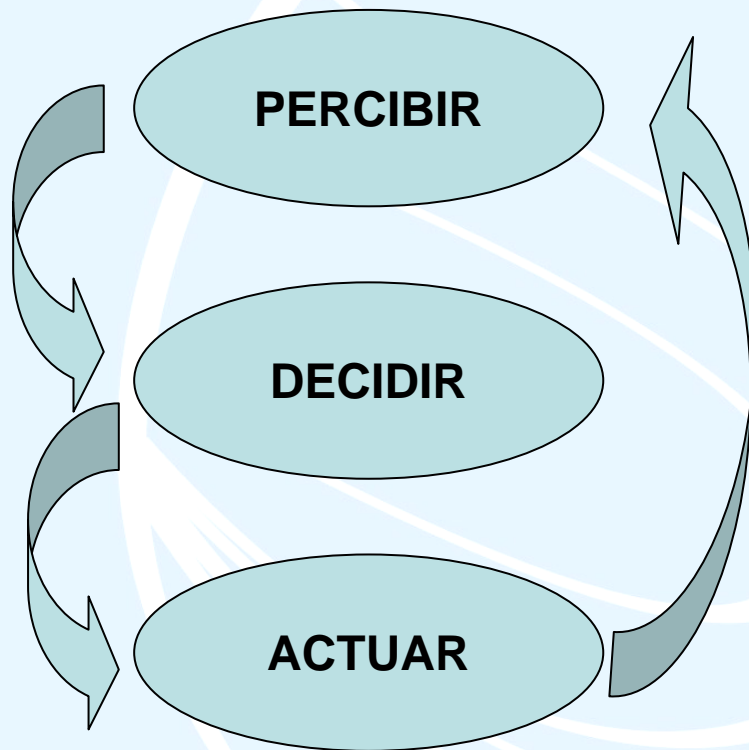




## 2. Presentación del Problema



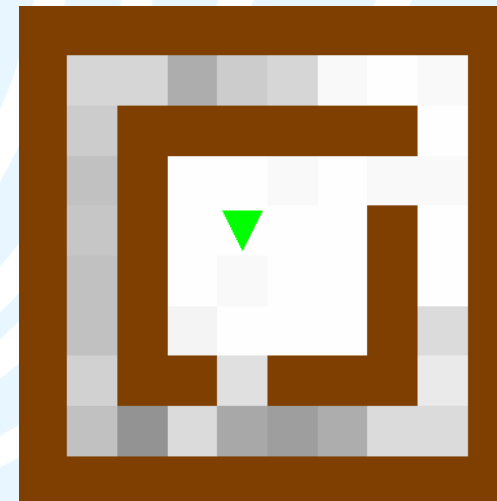
## 2. Presentación del Problema



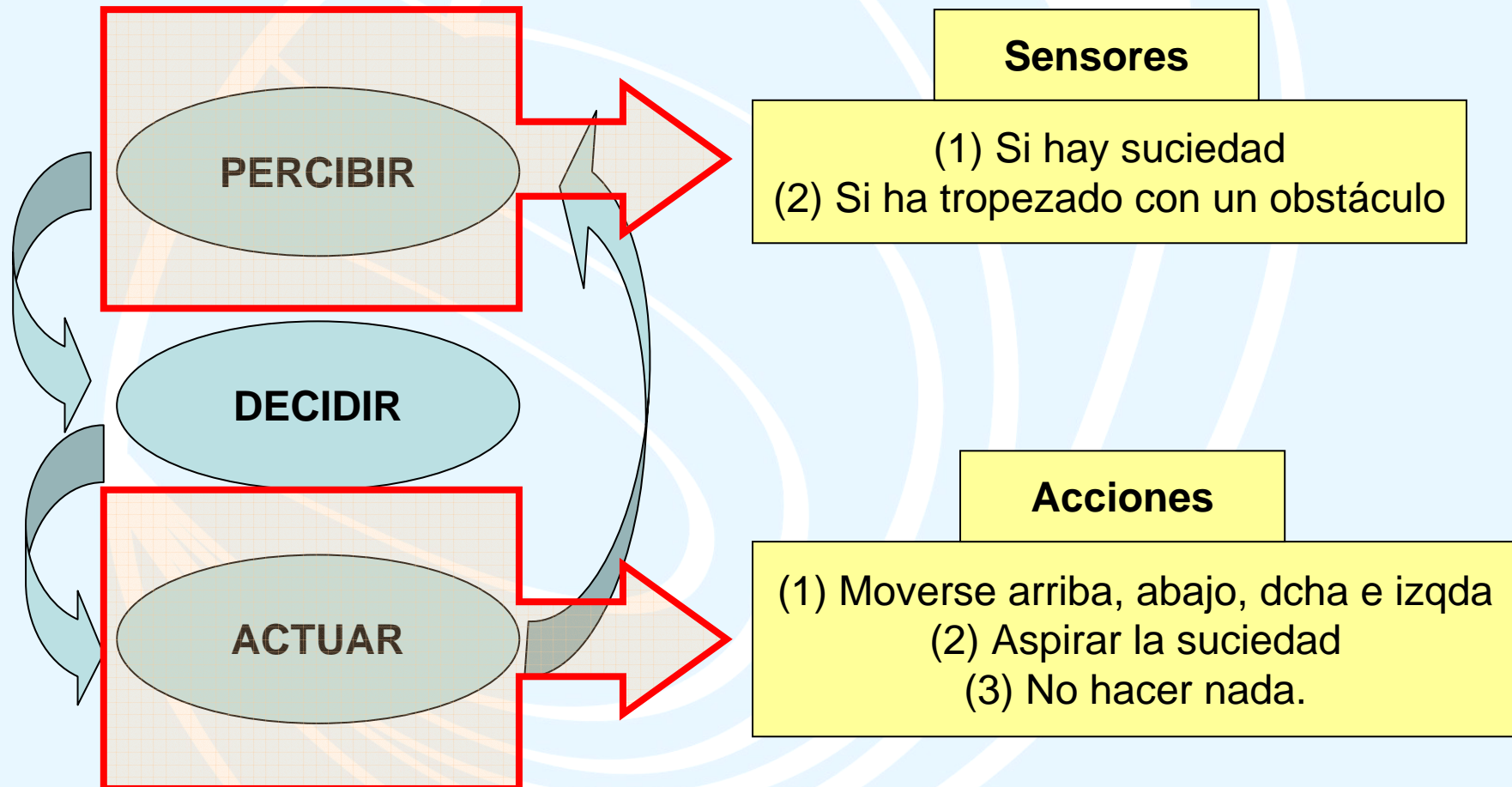
Controlador de ciclo cerrado

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

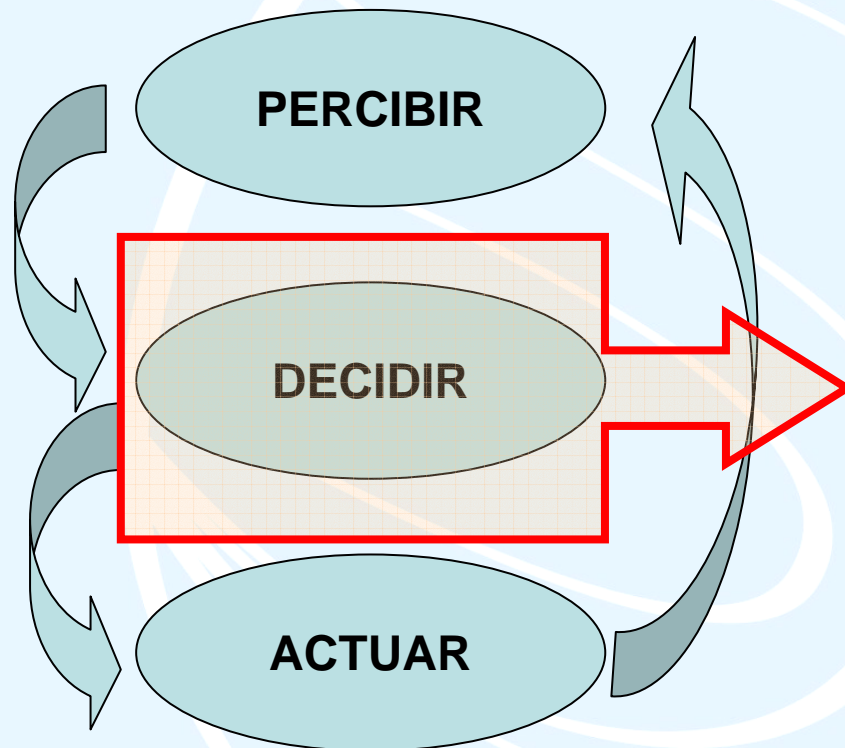
- Lo que es capaz de percibir el agente del entorno y,
- las acciones que el agente puede realizar.



## 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 enunciado de la práctica se indica como proceder a su instalación.

1. Cread la carpeta “**U:\IA\practica1**”
2. Descargar **Material\_Practica1.rar** 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 1”
- (f) Seleccionar “Material para la Práctica 1”



# 3. Presentación del Simulador

## 3.1. Compilación del Simulador

3. Descomprimir en la raíz de esta carpeta y aparecerán:
  - “**Agent-GUI-R3a.rar**”,
  - “**BibliotecasP1.rar**” y
  - “**MapasPractica1.rar**”.
4. Desempaqueta el fichero “**Agent-GUI-R3a.rar**” en una subcarpeta llamada **Agent-GUI-R3a**. que contendrá un conjunto de archivos y las subcarpetas “**include**”, “**lib**” y “**map**”.
5. Desempaqueta el fichero “**BibliotecasP1.rar**” en la subcarpeta “**lib**” (“U:\IA\practica1\Agent-GUI-R3a\lib”).
6. Desempaquete el fichero “**MapasPractica1.rar**” en la subcarpeta “**map**” (“U:\IA\practica1\Agent-GUI-R3a\map”).

# 3. Presentación del Simulador

## 3.1. Compilación del Simulador

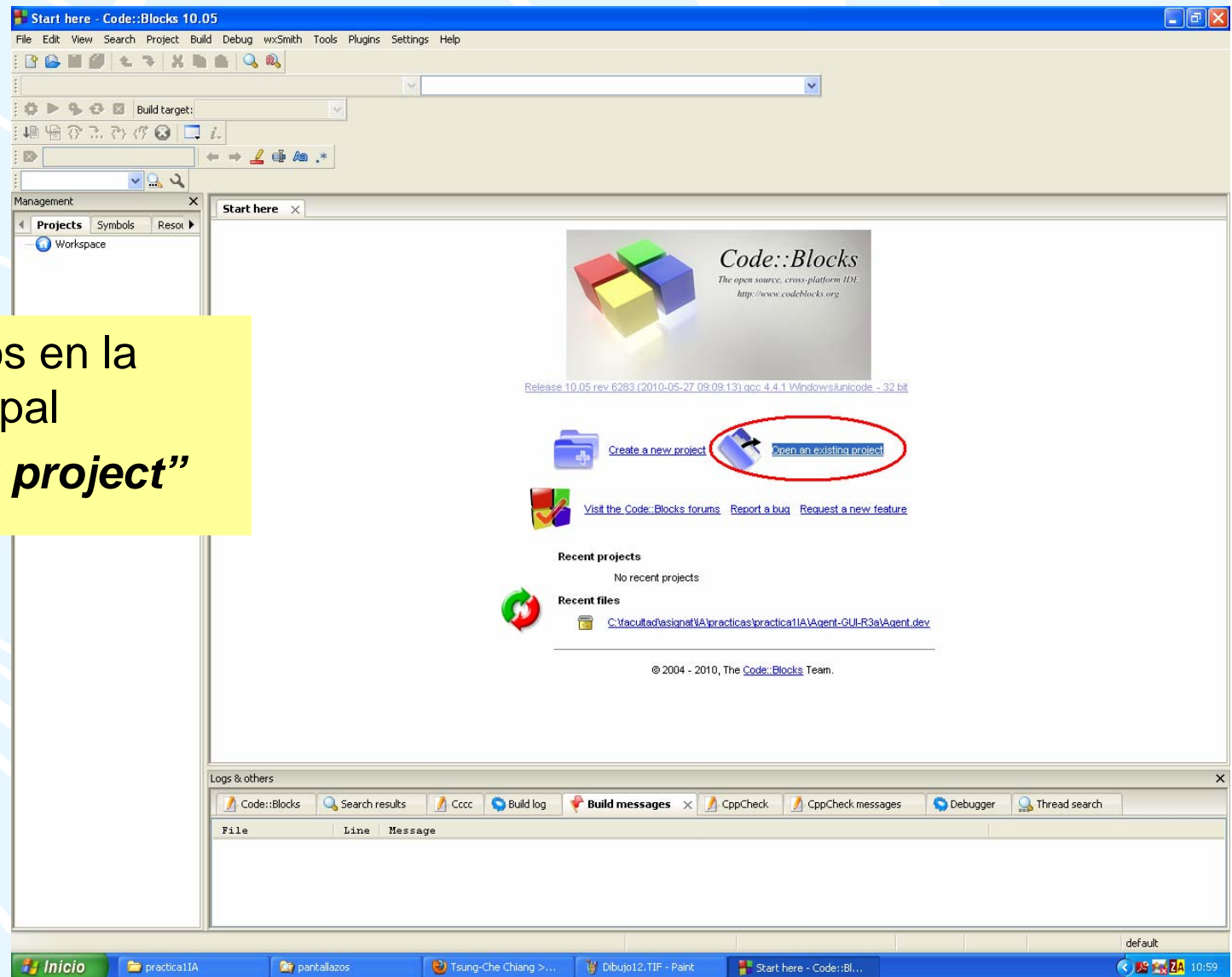
### 7. Abrimos “CodeBlocks”

- 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

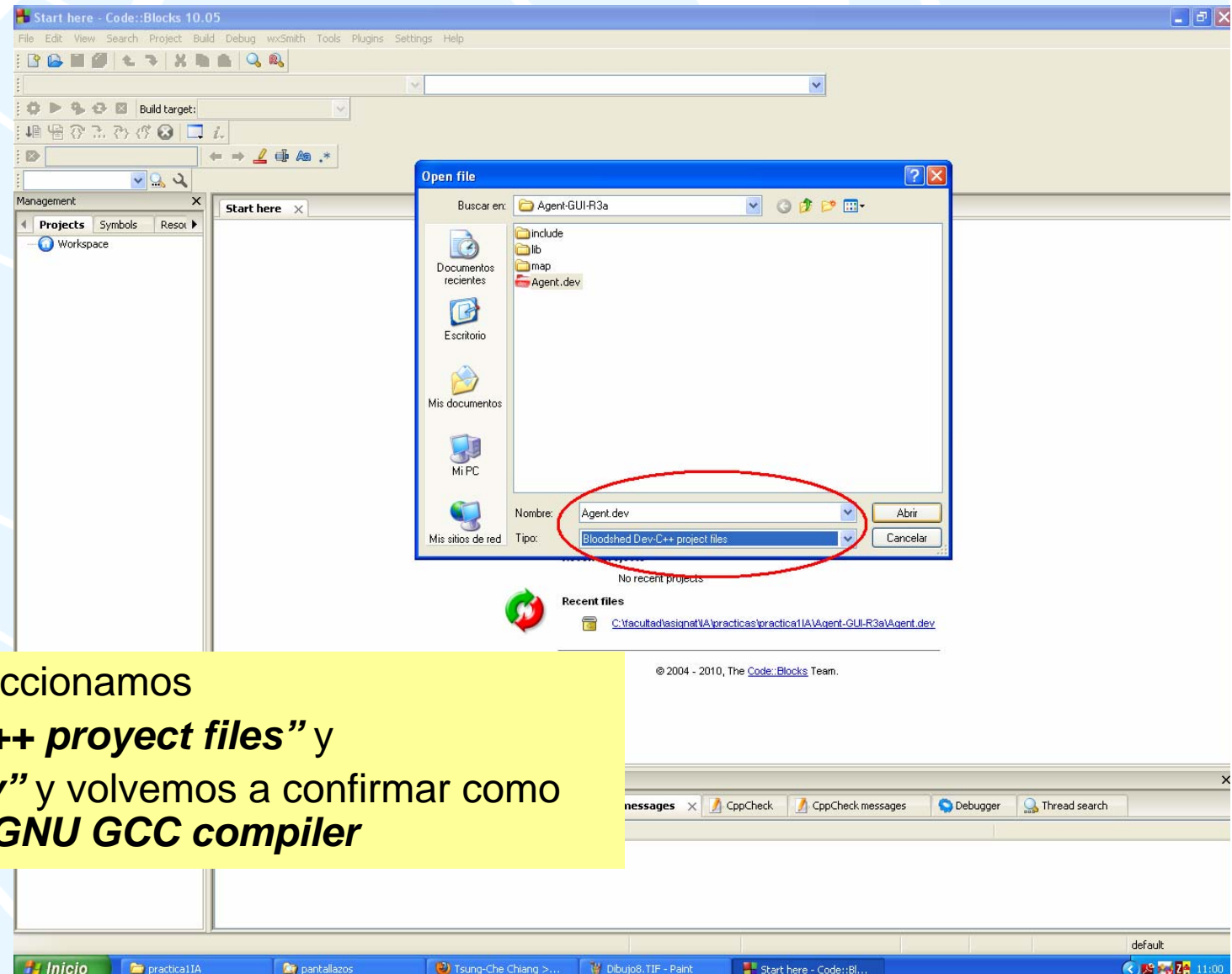
## 3.1. Compilación del Simulador

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



# 3. Presentación del Simulador

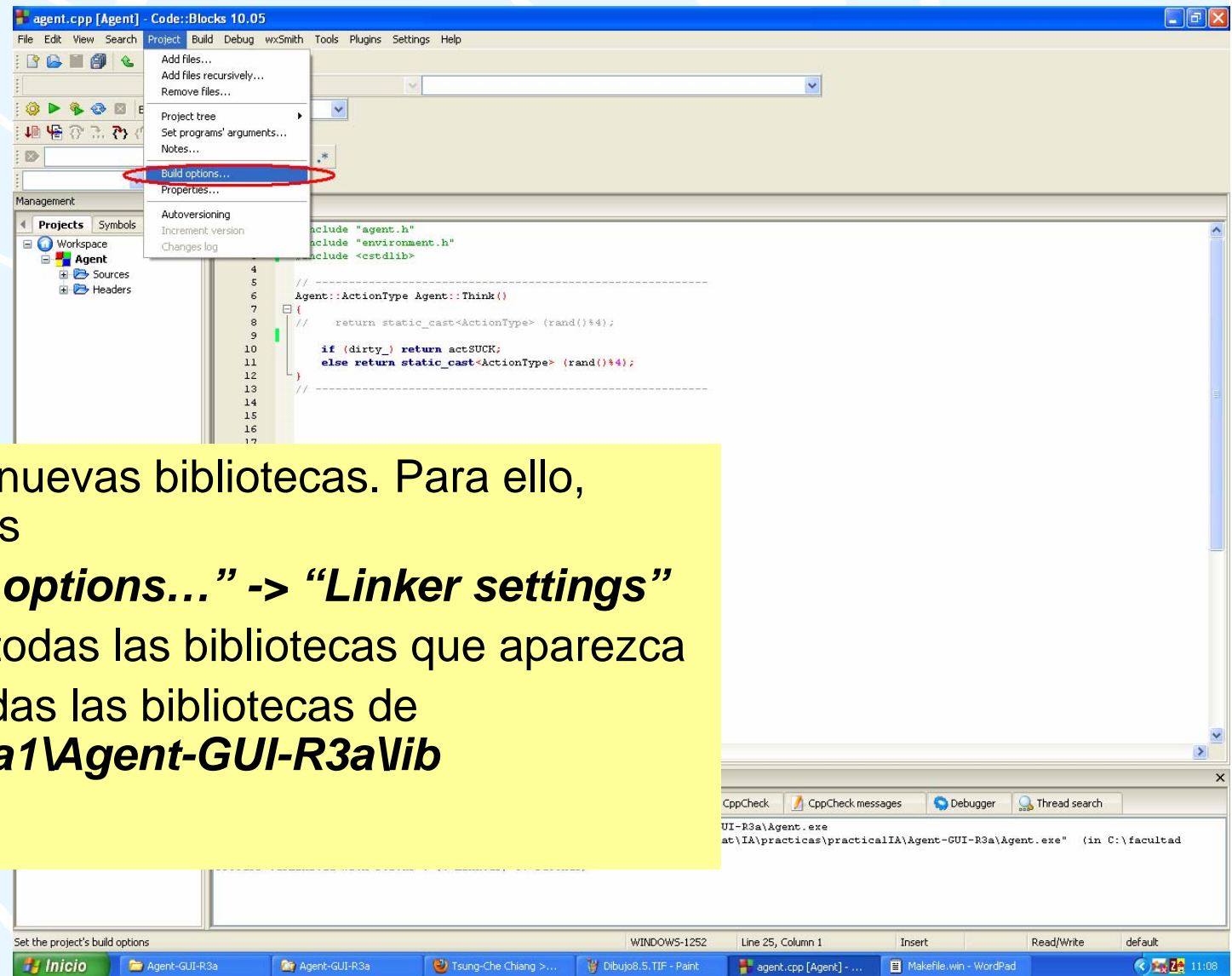
## 3.1. Compilación del Simulador



9. En “Tipo:” seleccionamos  
“**Bloodshed Dev-C++ project files**” y  
Abrimos “**Agent.dev**” y volvemos a confirmar como  
Compilador a **GNU GCC compiler**

# 3. Presentación del Simulador

## 3.1. Compilación del Simulador



10. Incluimos las nuevas bibliotecas. Para ello, seleccionamos

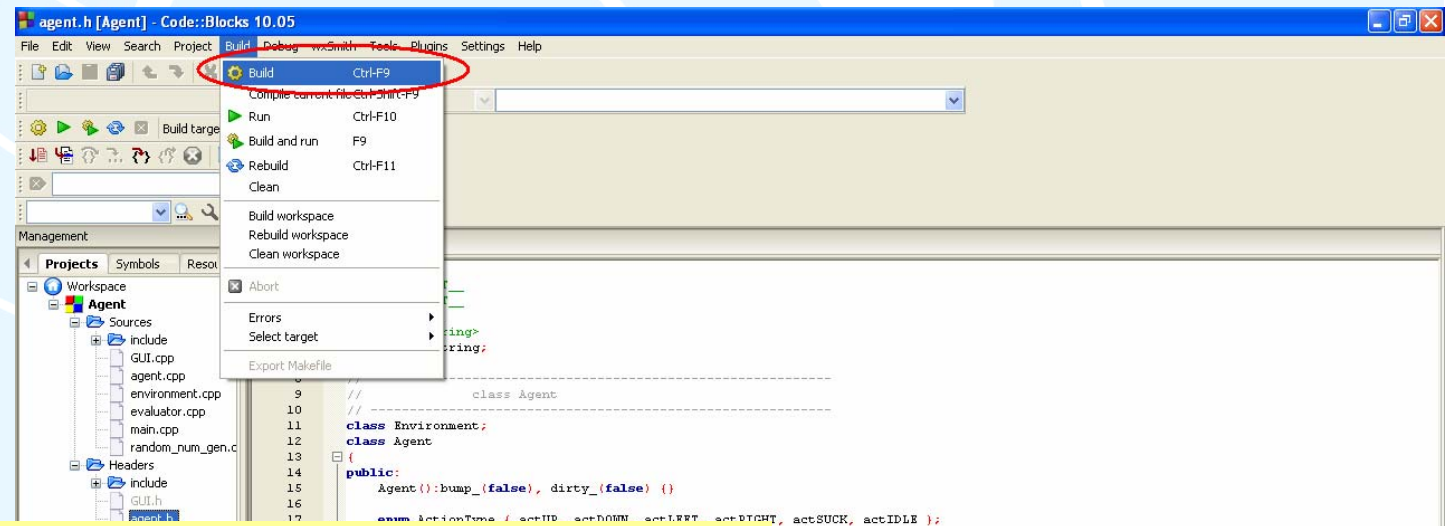
***“Project”->“Build options...” -> “Linker settings”***

10(a). Eliminamos todas las bibliotecas que aparezca

10(b). Incluimos todas las bibliotecas de  
***U:VA\practica1\Agent-GUI-R3aVib***

# 3. Presentación del Simulador

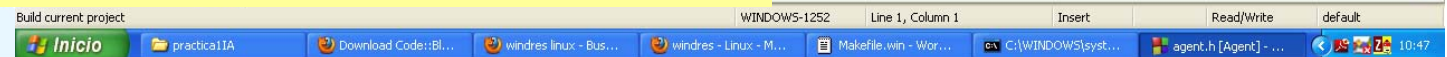
## 3.1. Compilación del Simulador



11. Compilamos el proyecto seleccionando

***“Build → Build”***

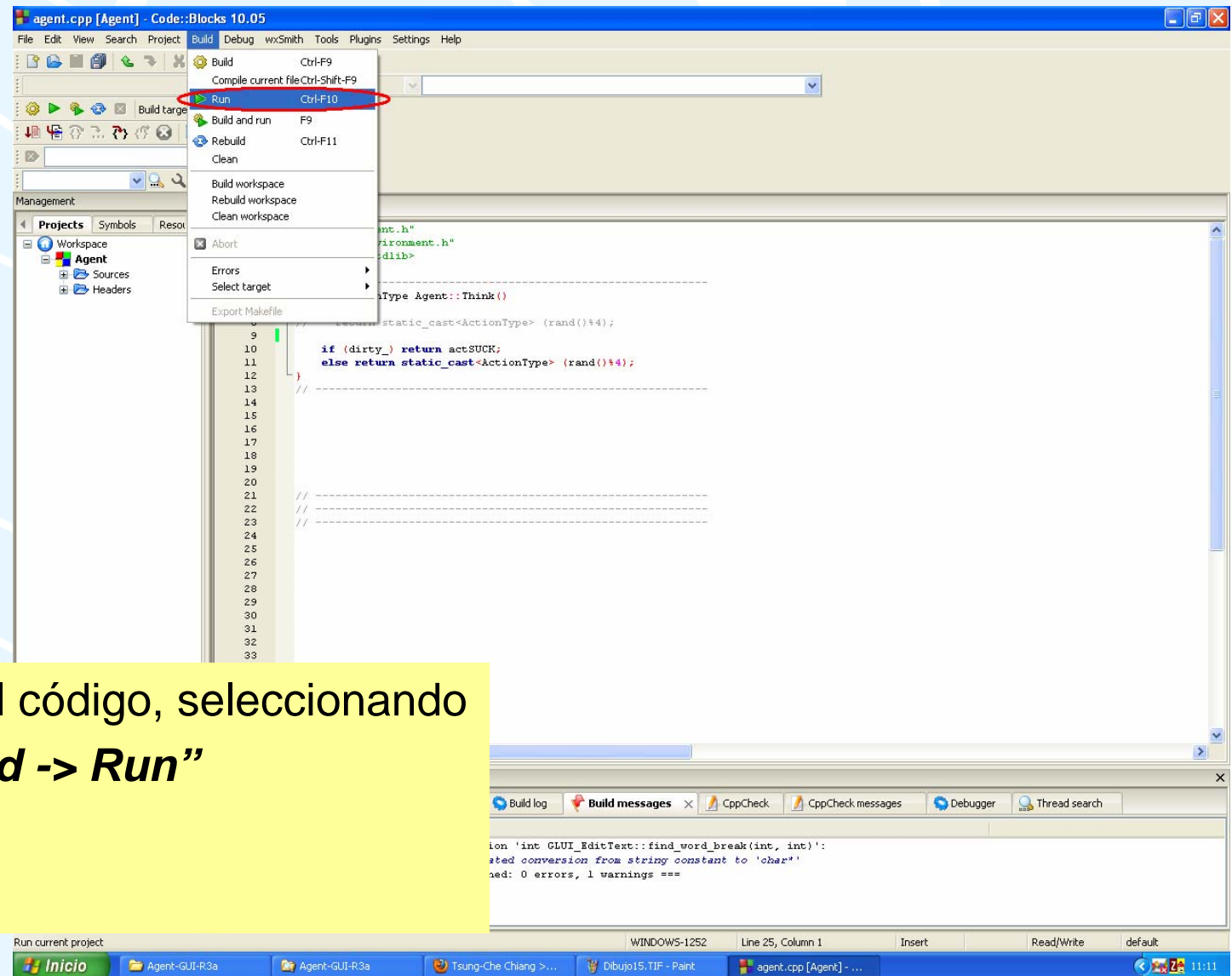
- La compilación nos da un error, indicando que no se reconoce el comando ***“rand()”***.
- Corregimos el error, incluyendo en la línea 3, la cabecera ***“#include <cstdlib>”*** en el archivo ***“agent.cpp”***
- Salvamos el archivo y ***“Build -> Build”***





# 3. Presentación del Simulador

## 3.1. Compilación del Simulador

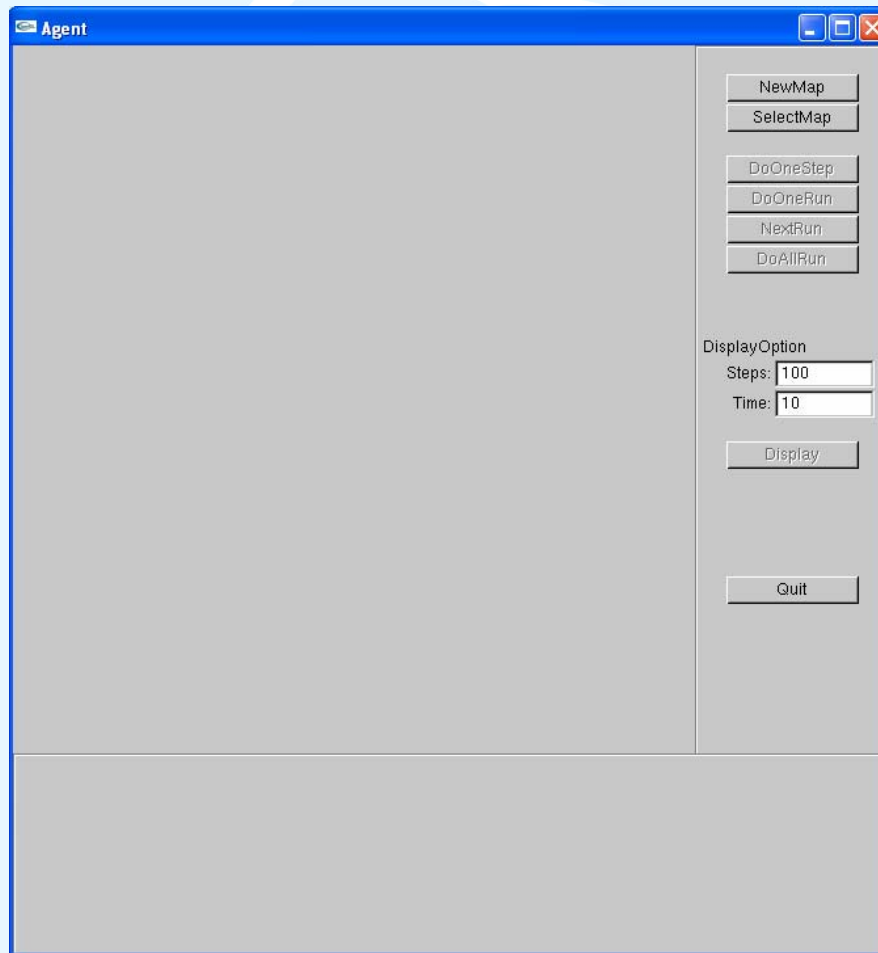


12. Ejecutamos el código, seleccionando  
***“Build -> Run”***



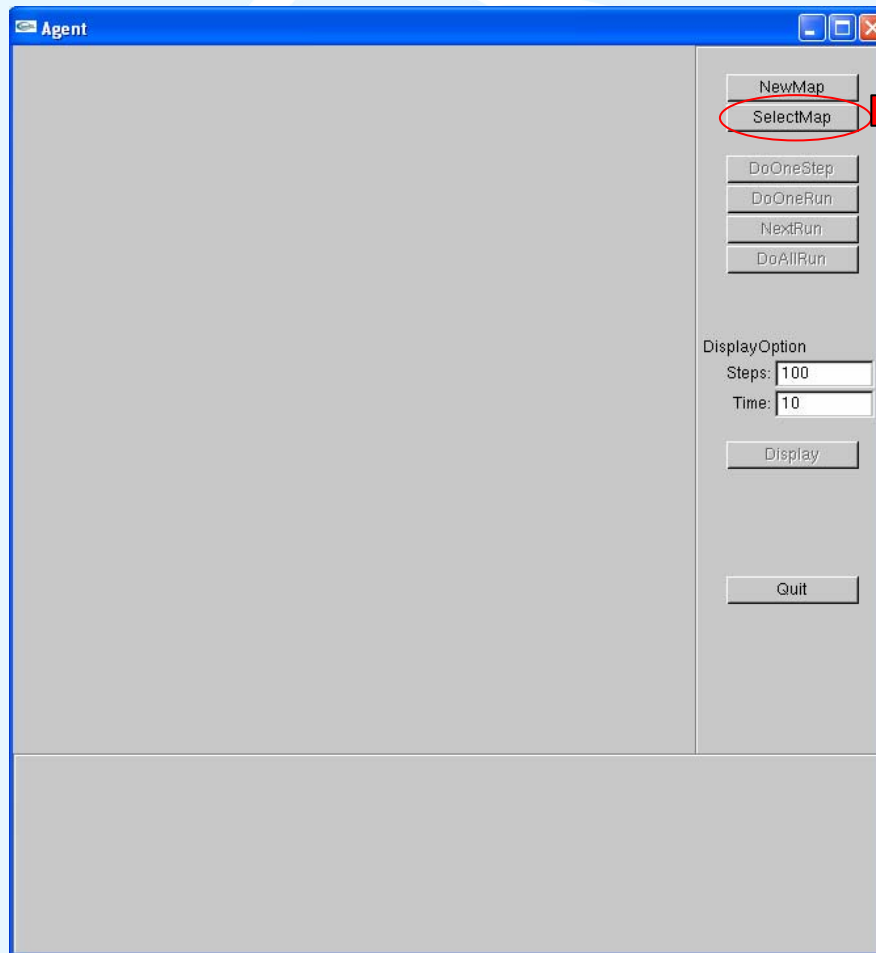
# 3. Presentación del Simulador

## 3.2. Ejecución del Simulador



# 3. Presentación del Simulador

## 3.2. Ejecución del Simulador

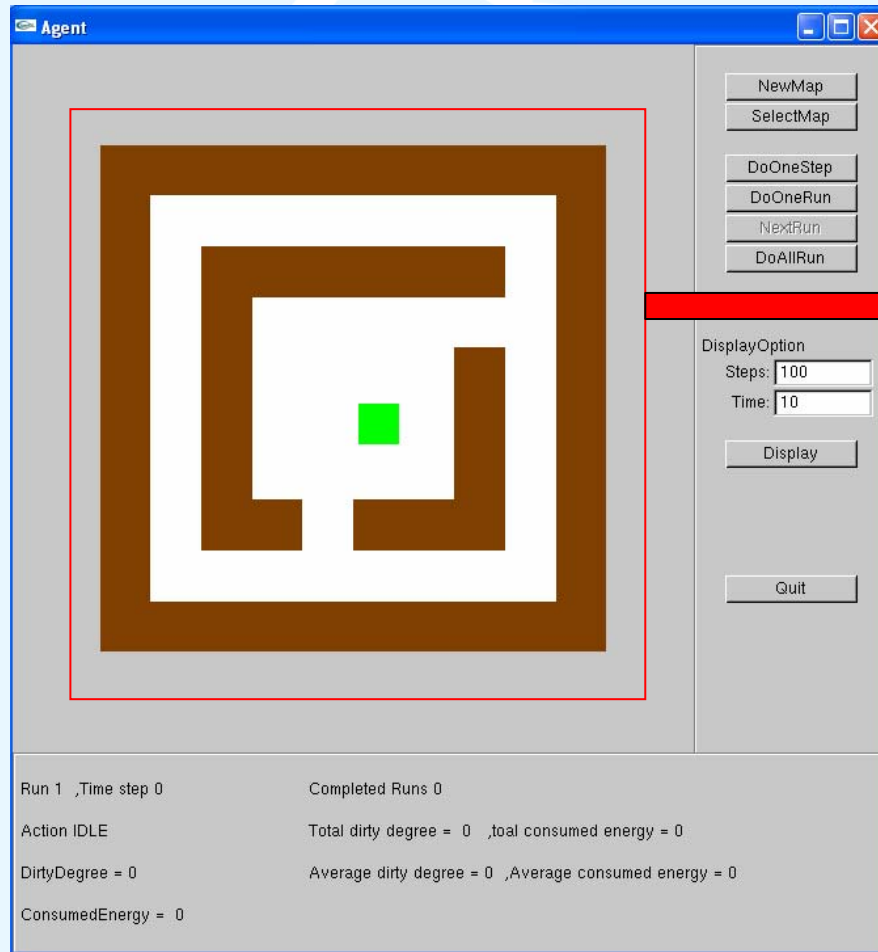


***“SelectMap”*** Selecciona el fichero de problema sobre el que Realizar la simulación.

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

# 3. Presentación del Simulador

## 3.2. Ejecución del Simulador

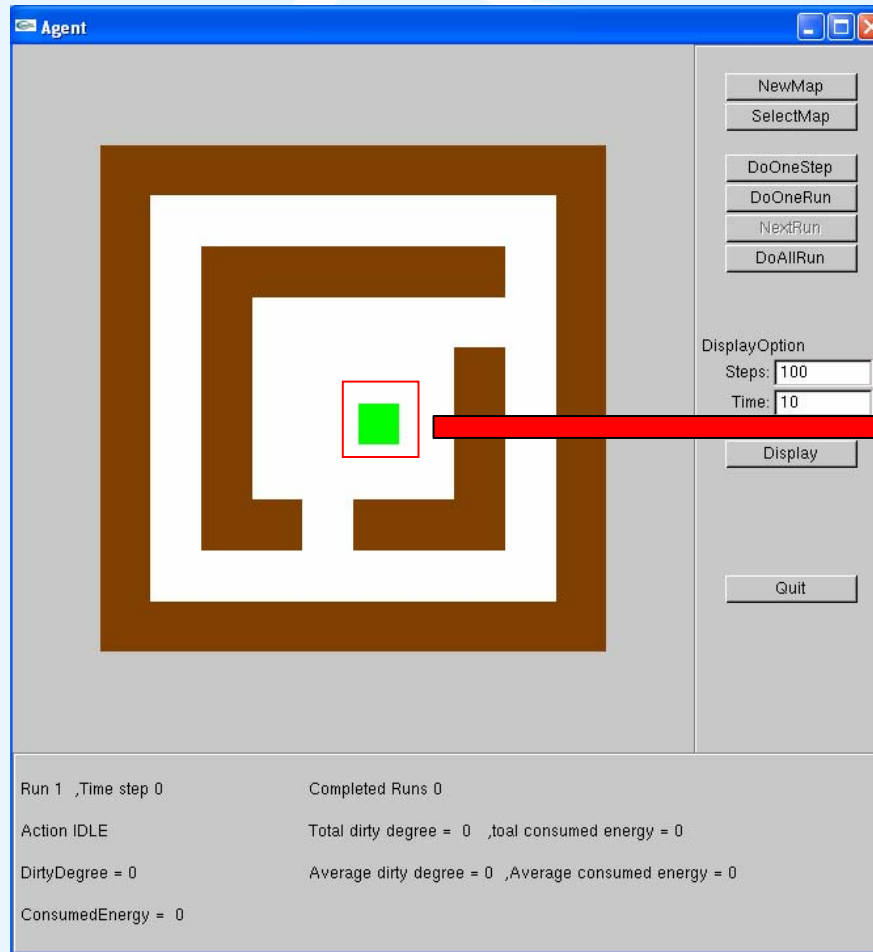


### ***Mundo simulado:***

- 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



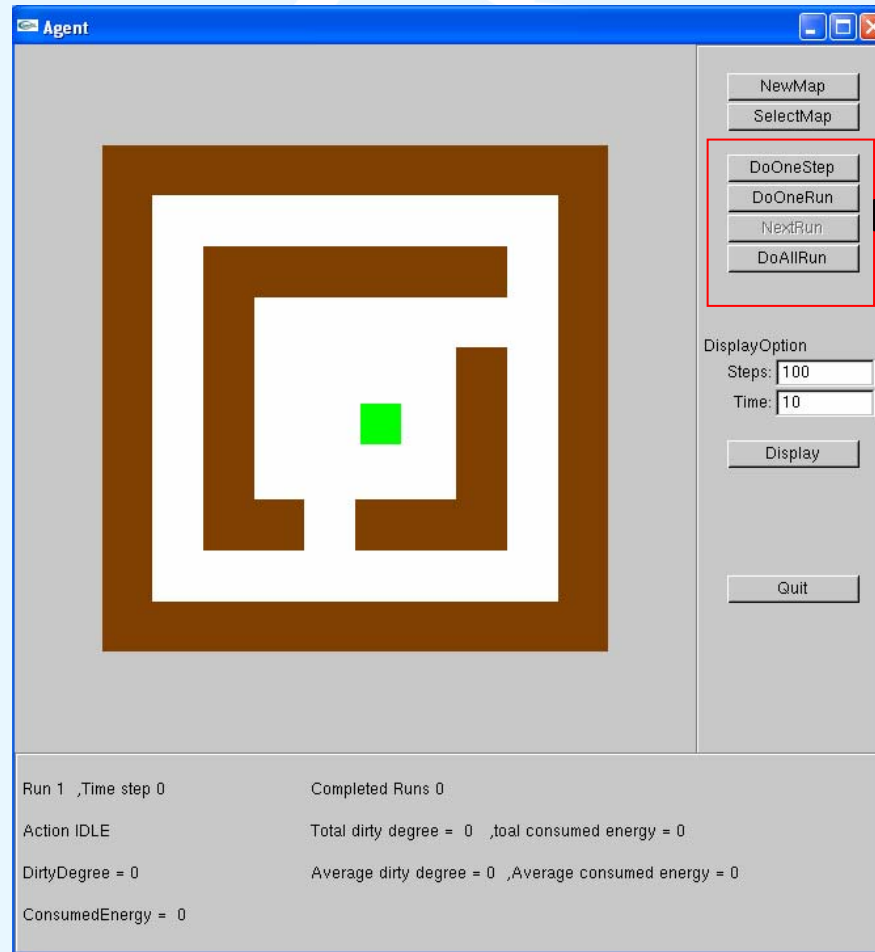
### ***Aspiradora:***

Los diferentes estados representan el resultado de la última acción:

-  **No** hizo nada.
-  **Aspiró** la suciedad de la casilla.
-  **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.

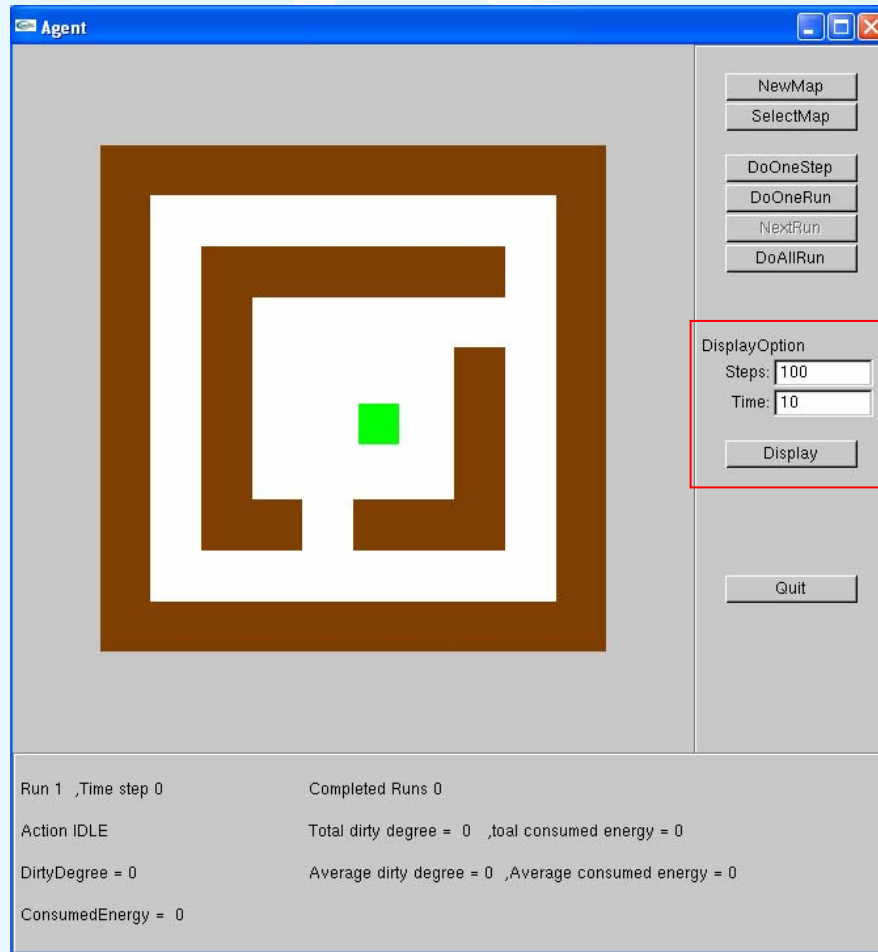
***“DoOneRun”*** Produce una simulación completa.

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

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

# 3. Presentación del Simulador

## 3.2. Ejecución del Simulador



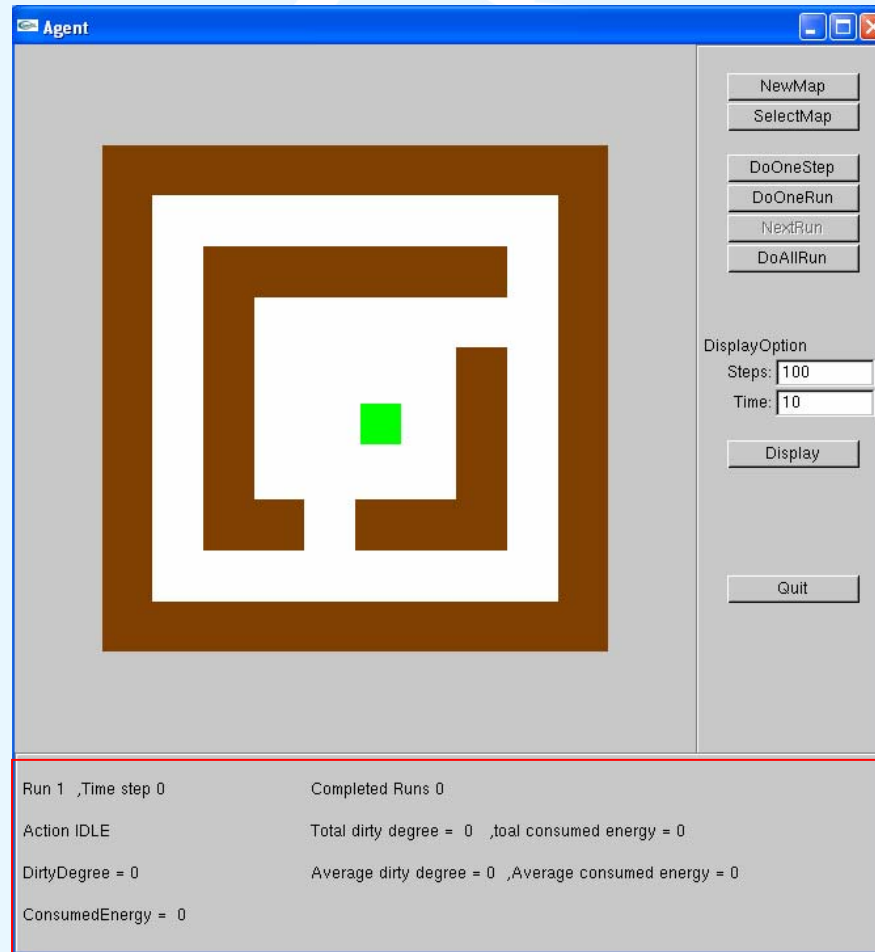
### ***“Display”***

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.dev 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.

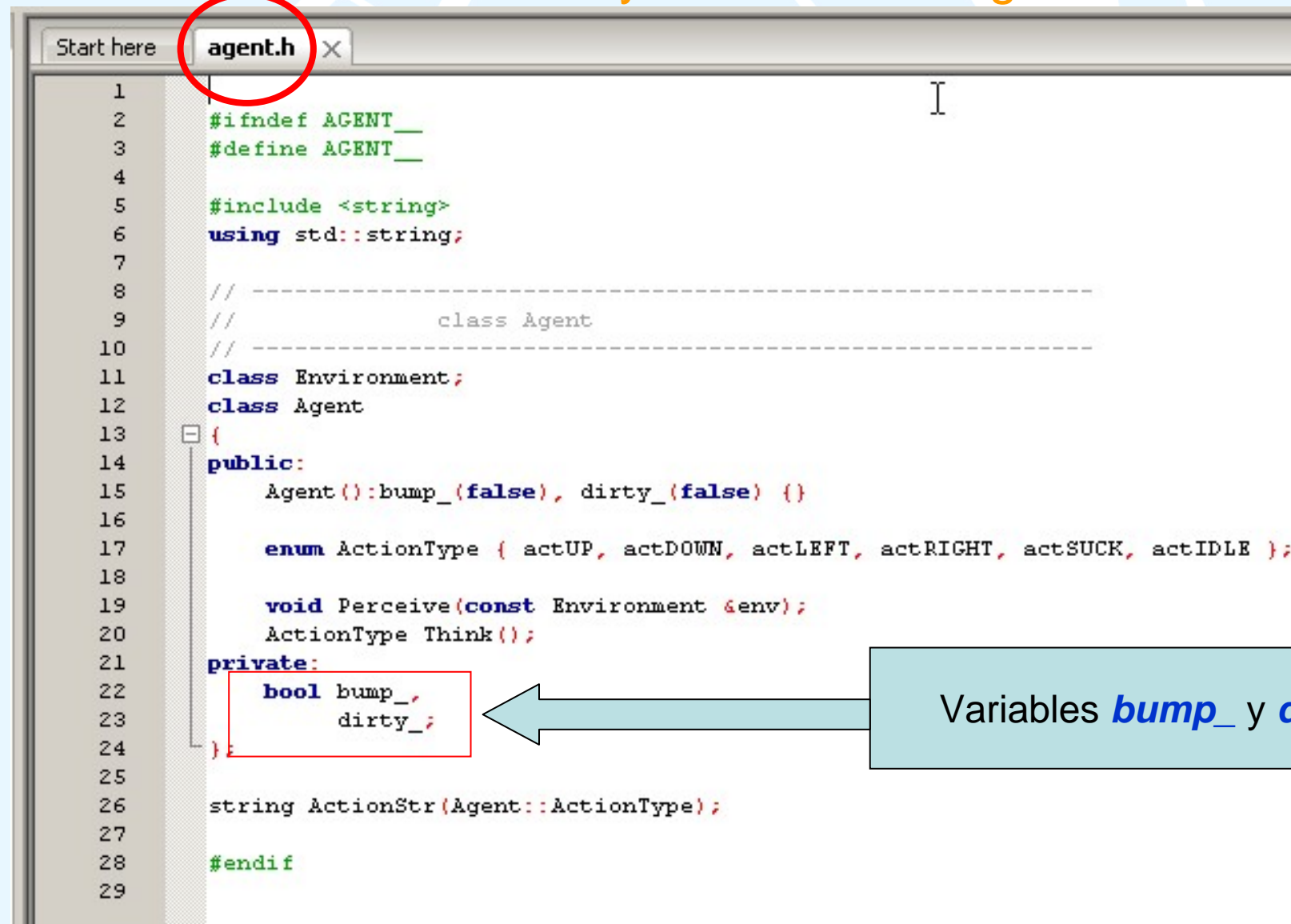
# 4. Implementación de un agente

## 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 indica que hay suciedad en la casilla actual.

# 4. Implementación de un agente

## 4.2. Métodos y variables del agente



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

Variables **bump\_** y **dirty\_**

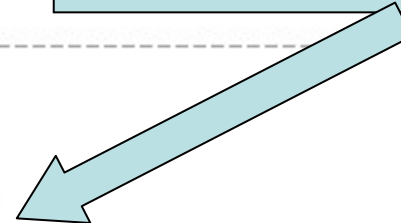


# 4. Implementación de un agente

## 4.2. Métodos y variables del agente

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

Constructor de Clase  
Pone a **false** bump\_ y dirty\_



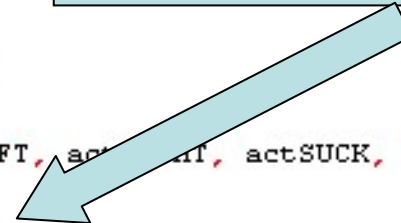


# 4. Implementación de un agente

## 4.2. Métodos y variables del agente

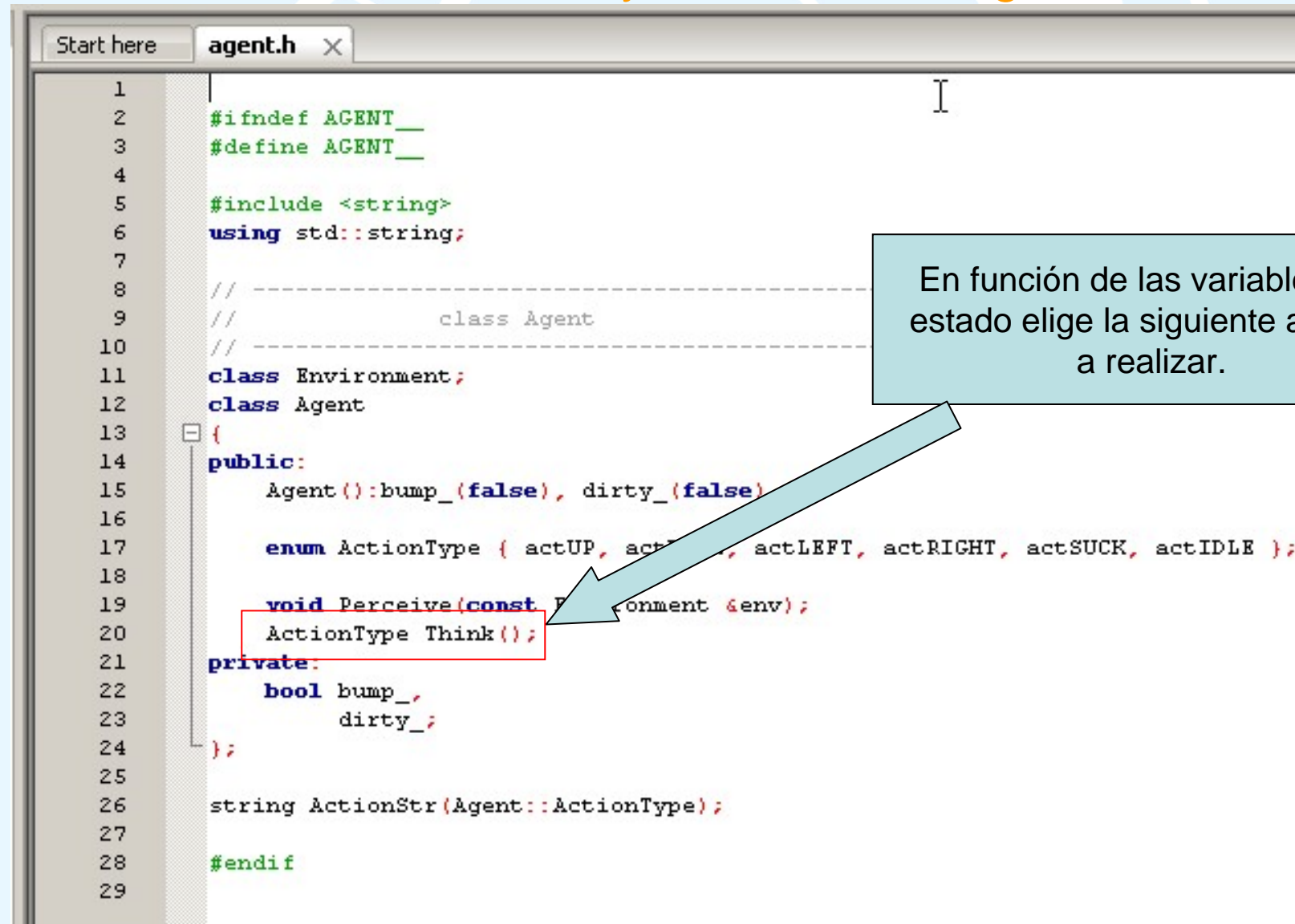
```
1  |
2  | #ifndef AGENT__
3  | #define AGENT__
4  |
5  | #include <string>
6  | using std::string;
7  |
8  | // -----
9  | //           class Agent
10 | // -----
11 | class Environment;
12 | class Agent
13 | {
14 | public:
15 |     Agent(): bump_(false), dirty_(false) {}
16 |
17 |     enum ActionType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

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

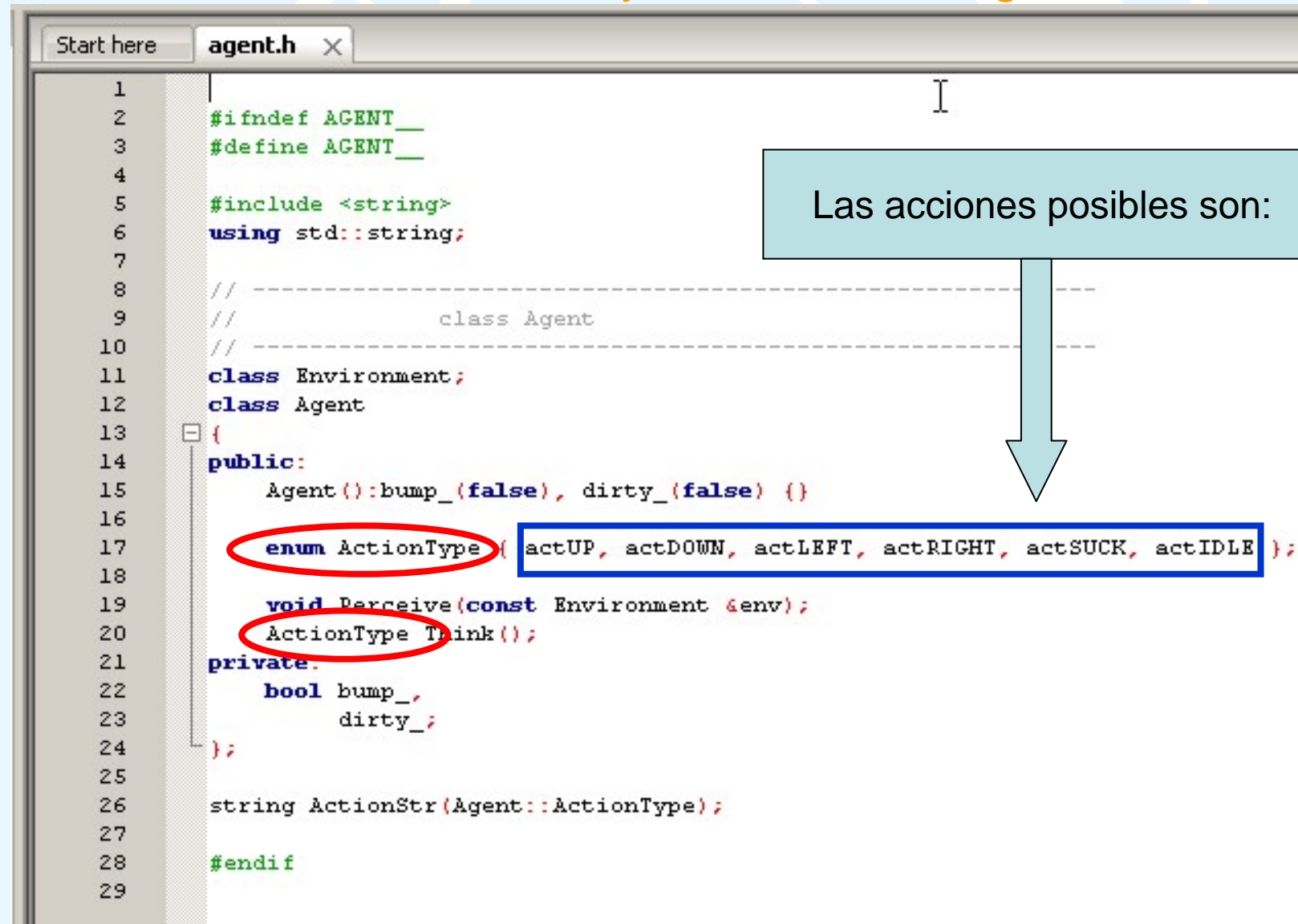


```
1
2 #ifndef AGENT__
3 #define AGENT__
4
5 #include <string>
6 using std::string;
7
8 // -----
9 //             class Agent
10 // -----
11 class Environment;
12 class Agent
13 {
14 public:
15     Agent():bump_(false), dirty_(false)
16
17     enum ActionType { actUP, actDOWN, actLEFT, actRIGHT, actSUCK,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

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



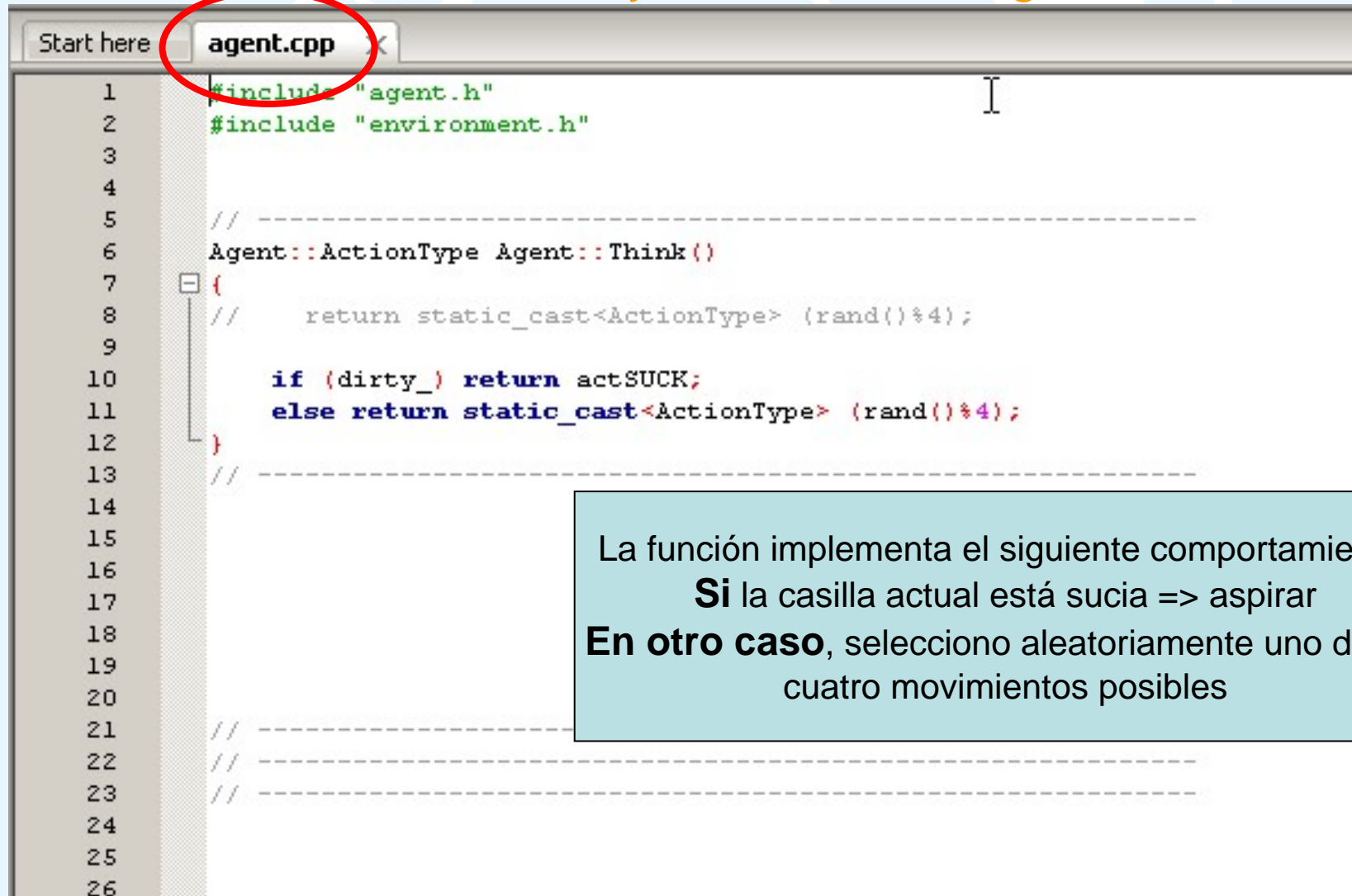
Start here agent.h

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

Las acciones posibles son:

# 4. Implementación de un agente

## 4.2. Métodos y variables del agente



```
1  #include "agent.h"
2  #include "environment.h"
3
4
5  // -----
6  Agent::ActionType Agent::Think()
7  {
8      // return static_cast<ActionType> (rand()%4);
9
10     if (dirty_) return actSUCK;
11     else return static_cast<ActionType> (rand()%4);
12 }
13 // -----
14
15
16
17
18
19
20
21 // -----
22 // -----
23 // -----
24
25
26
```

La función implementa el siguiente comportamiento:  
**Si** la casilla actual está sucia => aspirar  
**En otro caso**, selecciono aleatoriamente uno de los cuatro movimientos posibles

## 4. Implementación de un agente

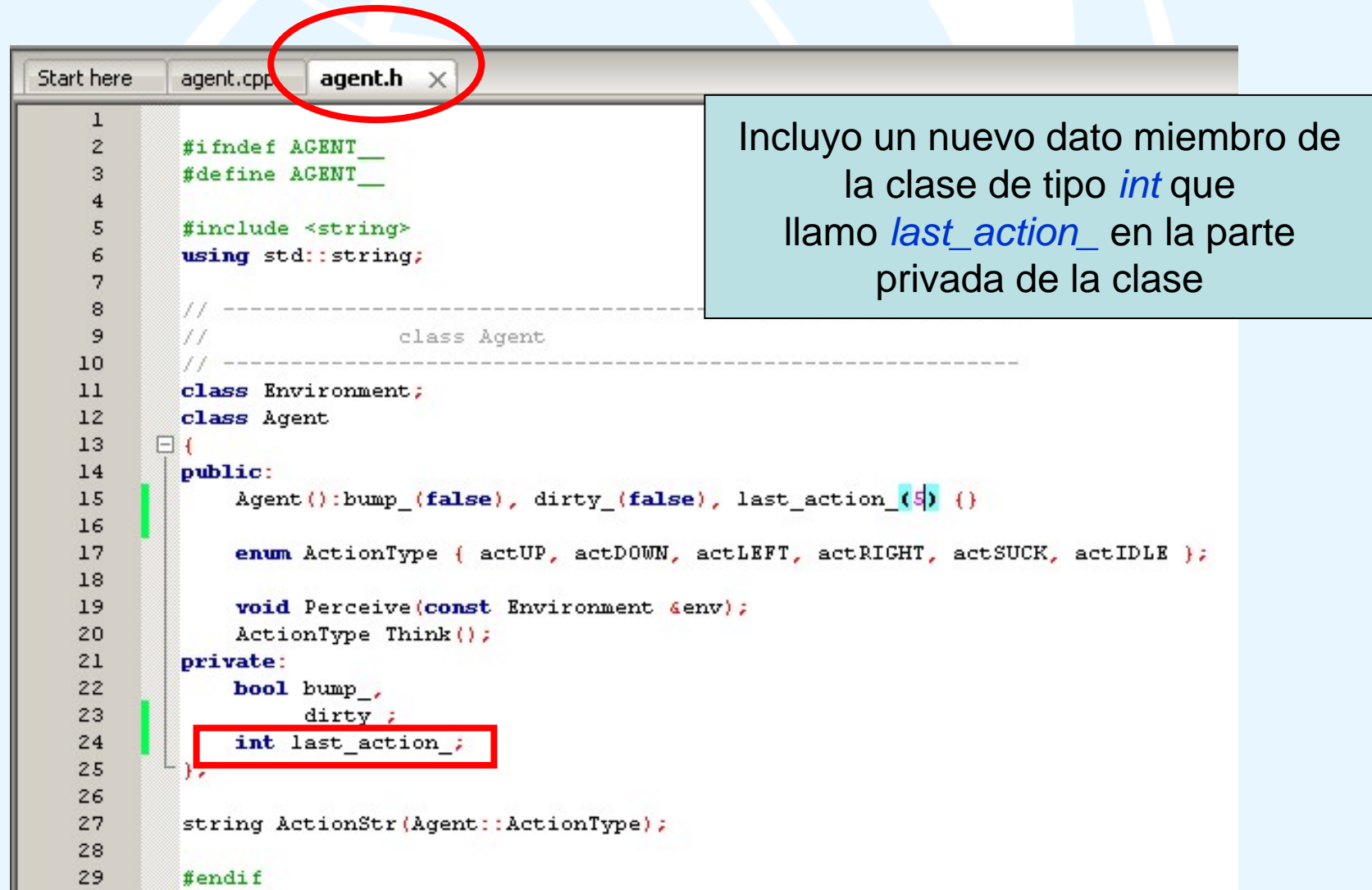
### 4.3. Modificando el comportamiento del agente: un ejemplo ilustrativo.

Supongamos que queremos mejorar el comportamiento del agente anterior, no permitiéndolo hacer un movimiento que sea el opuesto al que realizó en la acción anterior, es decir,

**si en la última acción se realizó un “actUP”,  
no vamos a permitir que la siguiente acción  
sea un “actDOWN” .**

# 4. Implementación de un agente

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



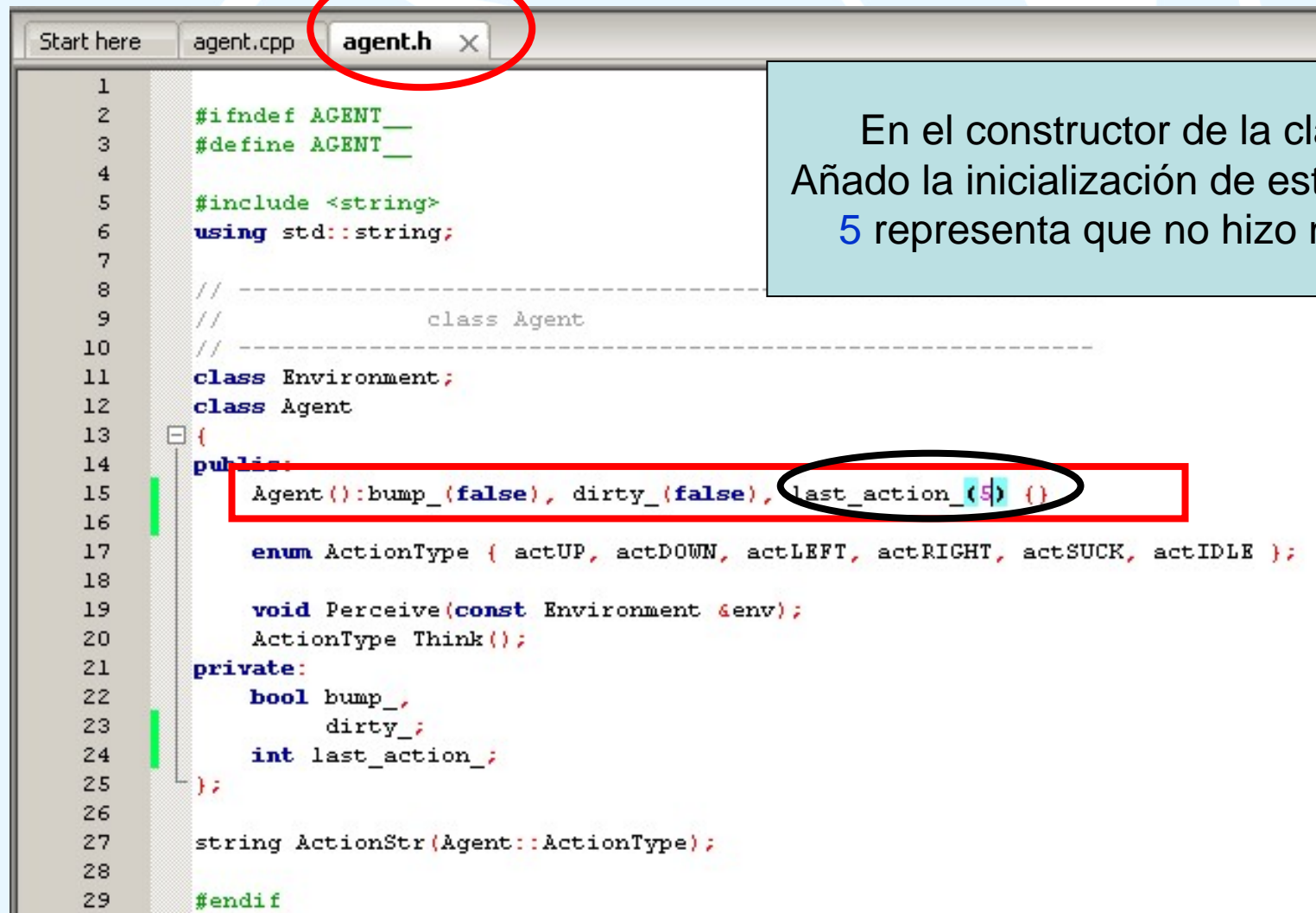
```
1
2  #ifndef AGENT_
3  #define AGENT_
4
5  #include <string>
6  using std::string;
7
8  // -----
9  //               class Agent
10 // -----
11 class Environment;
12 class Agent
13 {
14 public:
15     Agent(): bump_(false), dirty_(false), last_action_(5) {}
16
17     enum ActionType { actUP, actDOWN,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

Incluyo un nuevo dato miembro de la clase de tipo *int* que llamo *last\_action\_* en la parte privada de la clase



# 4. Implementación de un agente

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



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

En el constructor de la clase  
Añado la inicialización de este dato.  
5 representa que no hizo nada



# 4. Implementación de un agente

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

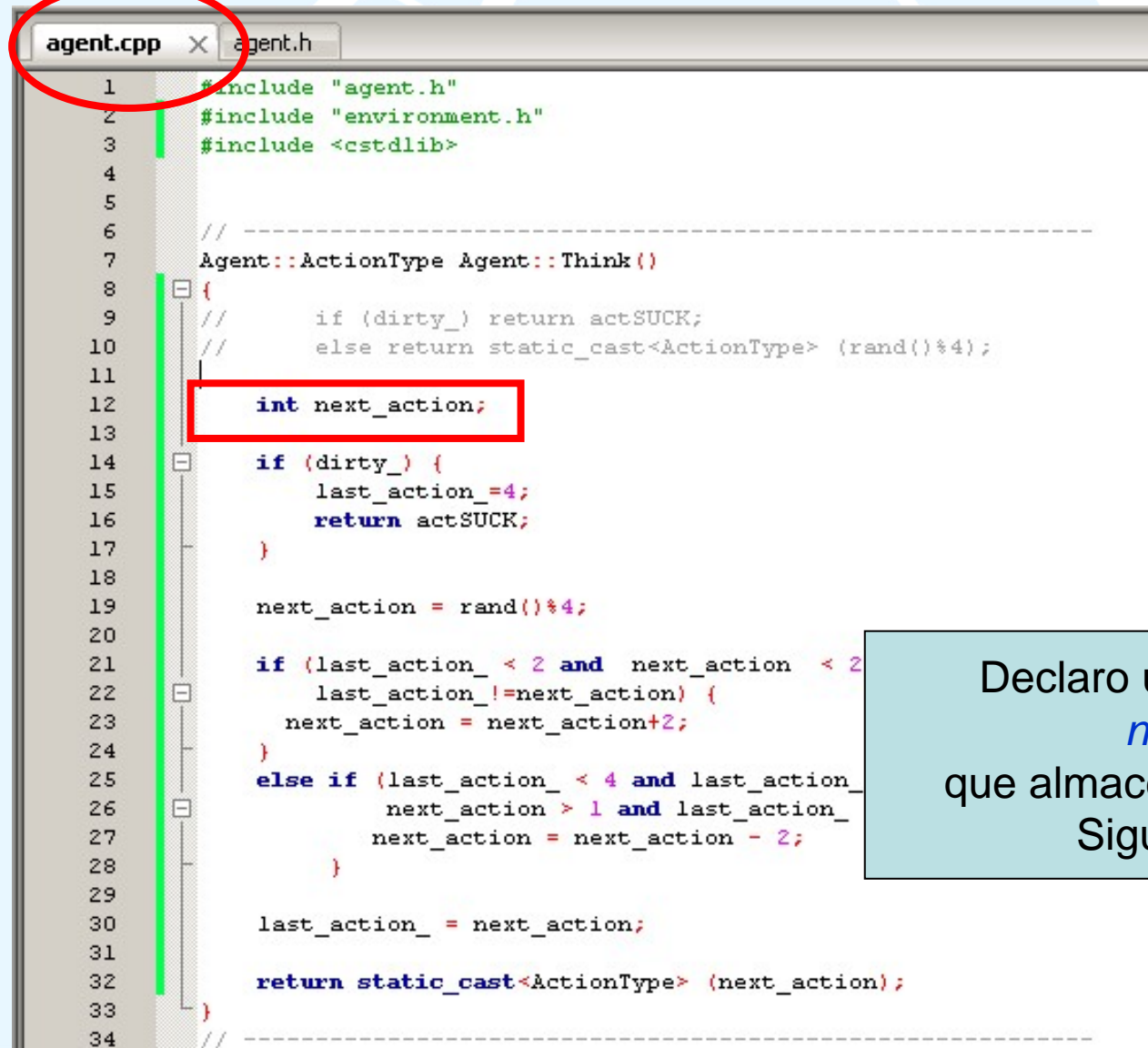


```
1  #include "agent.h"
2  #include "environment.h"
3  #include <cstdlib>
4
5
6  // -----
7  Agent::ActionType Agent::Think()
8  {
9      // if (dirty_) return actSUCK;
10     // else return static_cast<ActionType> (rand()%4);
11
12     int next_action;
13
14     if (dirty_) {
15         last_action_ = 4;
16         return actSUCK;
17     }
18
19     next_action = rand()%4;
20
21     if (last_action_ < 2 and next_action < 2
22         last_action_ != next_action) {
23         next_action = next_action + 2;
24     }
25     else if (last_action_ < 4 and last_action_
26             next_action > 1 and last_action_
27             next_action = next_action - 2;
28     }
29
30     last_action_ = next_action;
31
32     return static_cast<ActionType> (next_action);
33 }
34 // -----
```

Comento la implementación anterior  
del método `Think()`

# 4. Implementación de un agente

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

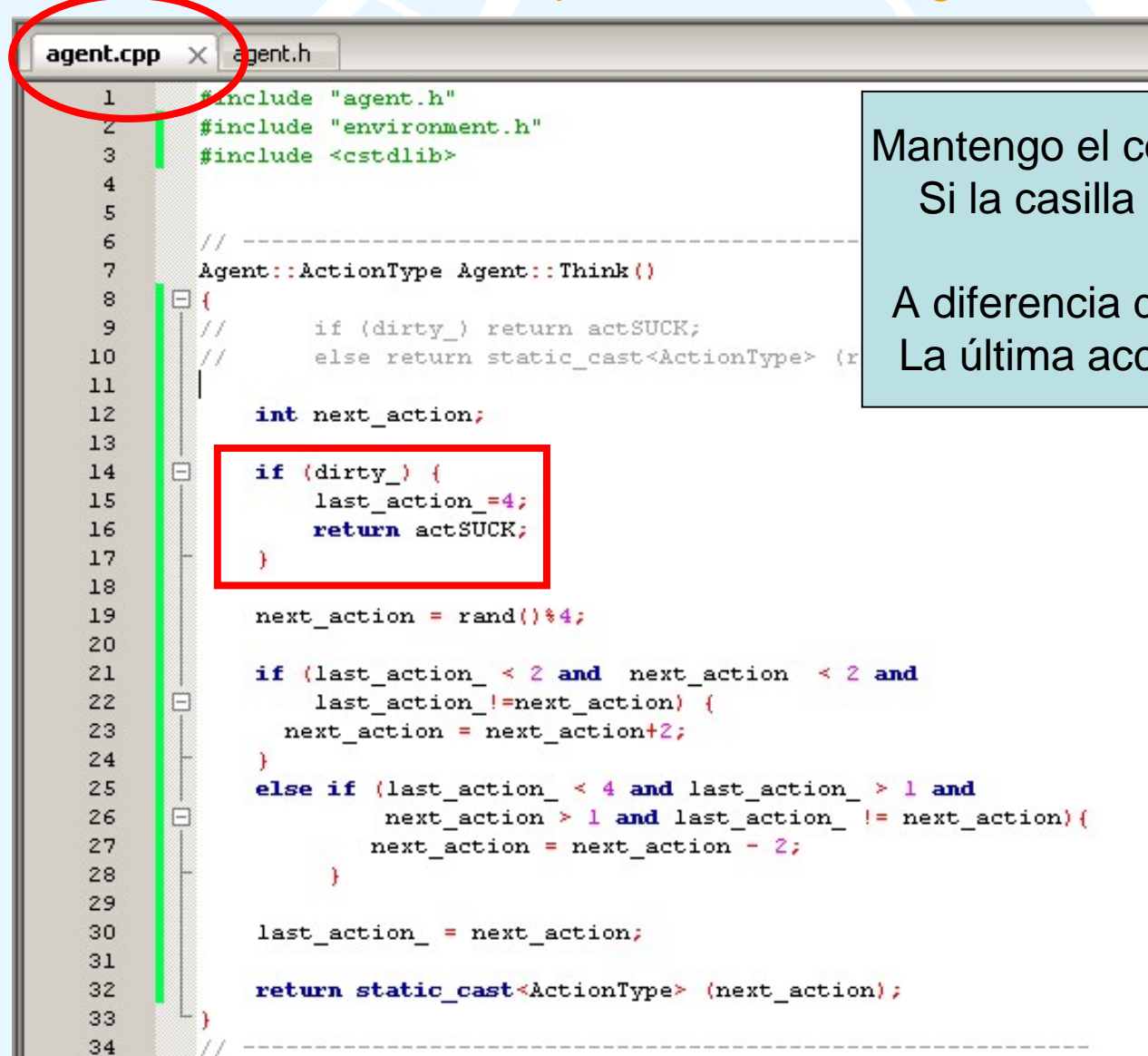


```
1  #include "agent.h"
2  #include "environment.h"
3  #include <cstdlib>
4
5
6  // -----
7  Agent::ActionType Agent::Think()
8  {
9      // if (dirty_) return actSUCK;
10     // else return static_cast<ActionType> (rand()%4);
11
12     int next_action;
13
14     if (dirty_) {
15         last_action_ = 4;
16         return actSUCK;
17     }
18
19     next_action = rand()%4;
20
21     if (last_action_ < 2 and next_action < 2
22         last_action_ != next_action) {
23         next_action = next_action + 2;
24     }
25     else if (last_action_ < 4 and last_action_
26             next_action > 1 and last_action_
27             next_action = next_action - 2;
28     }
29
30     last_action_ = next_action;
31
32     return static_cast<ActionType> (next_action);
33 }
34 // -----
```

Declaro una variable local  
*next\_action*  
que almacenará el valor de la  
Siguiente acción

# 4. Implementación de un agente

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



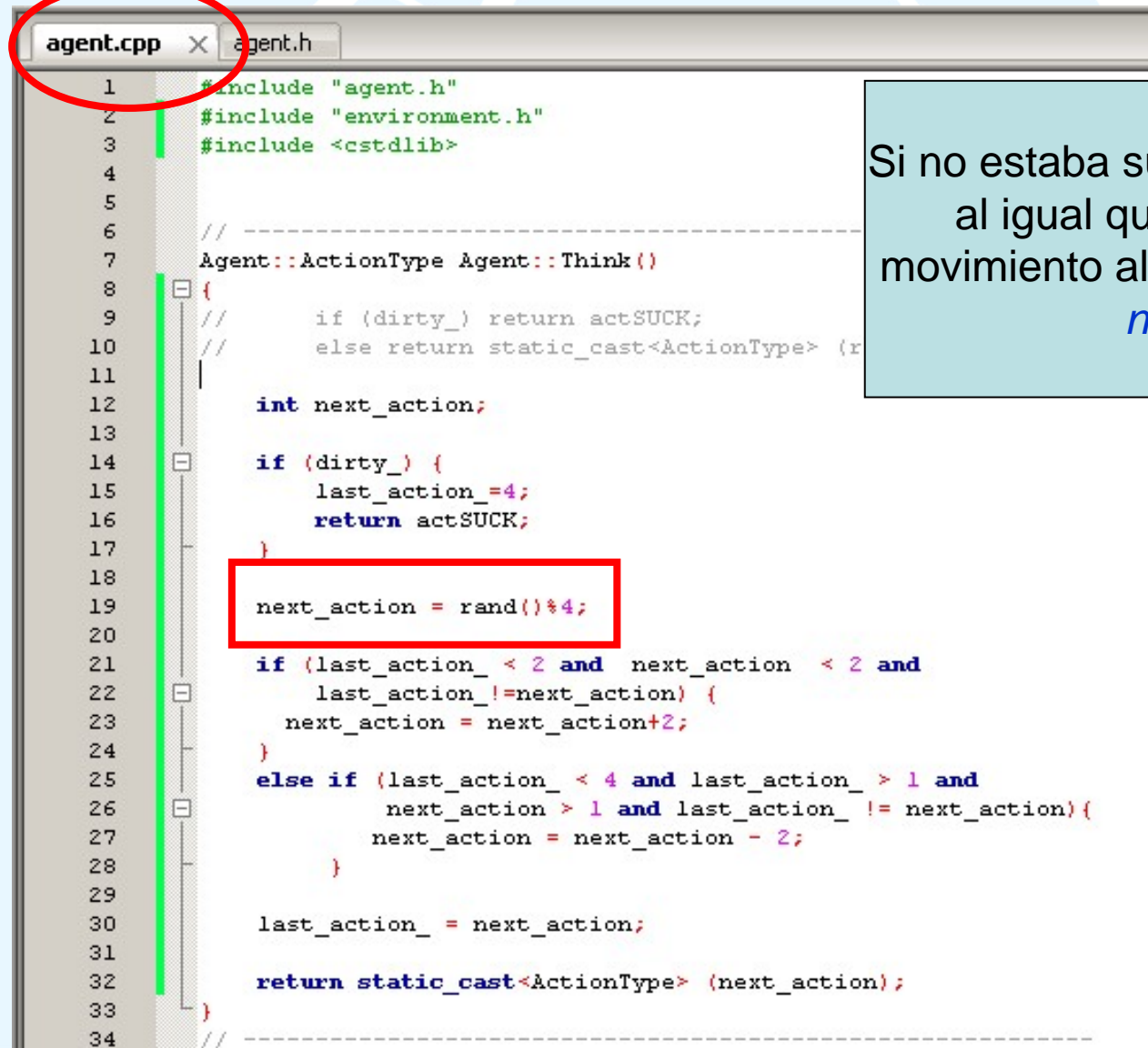
```
1 #include "agent.h"
2 #include "environment.h"
3 #include <cstdlib>
4
5
6 // -----
7 Agent::ActionType Agent::Think()
8 {
9     // if (dirty_) return actSUCK;
10    // else return static_cast<ActionType> (r
11
12    int next_action;
13
14    if (dirty_) {
15        last_action_ = 4;
16        return actSUCK;
17    }
18
19    next_action = rand()%4;
20
21    if (last_action_ < 2 and next_action < 2 and
22        last_action_ != next_action) {
23        next_action = next_action + 2;
24    }
25    else if (last_action_ < 4 and last_action_ > 1 and
26        next_action > 1 and last_action_ != next_action){
27        next_action = next_action - 2;
28    }
29
30    last_action_ = next_action;
31
32    return static_cast<ActionType> (next_action);
33 }
34 // -----
```

Mantengo el comportamiento anterior  
Si la casilla está sucia => limpiar

A diferencia del anterior, indicó que  
La última acción será *actSUCK* (4)

# 4. Implementación de un agente

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

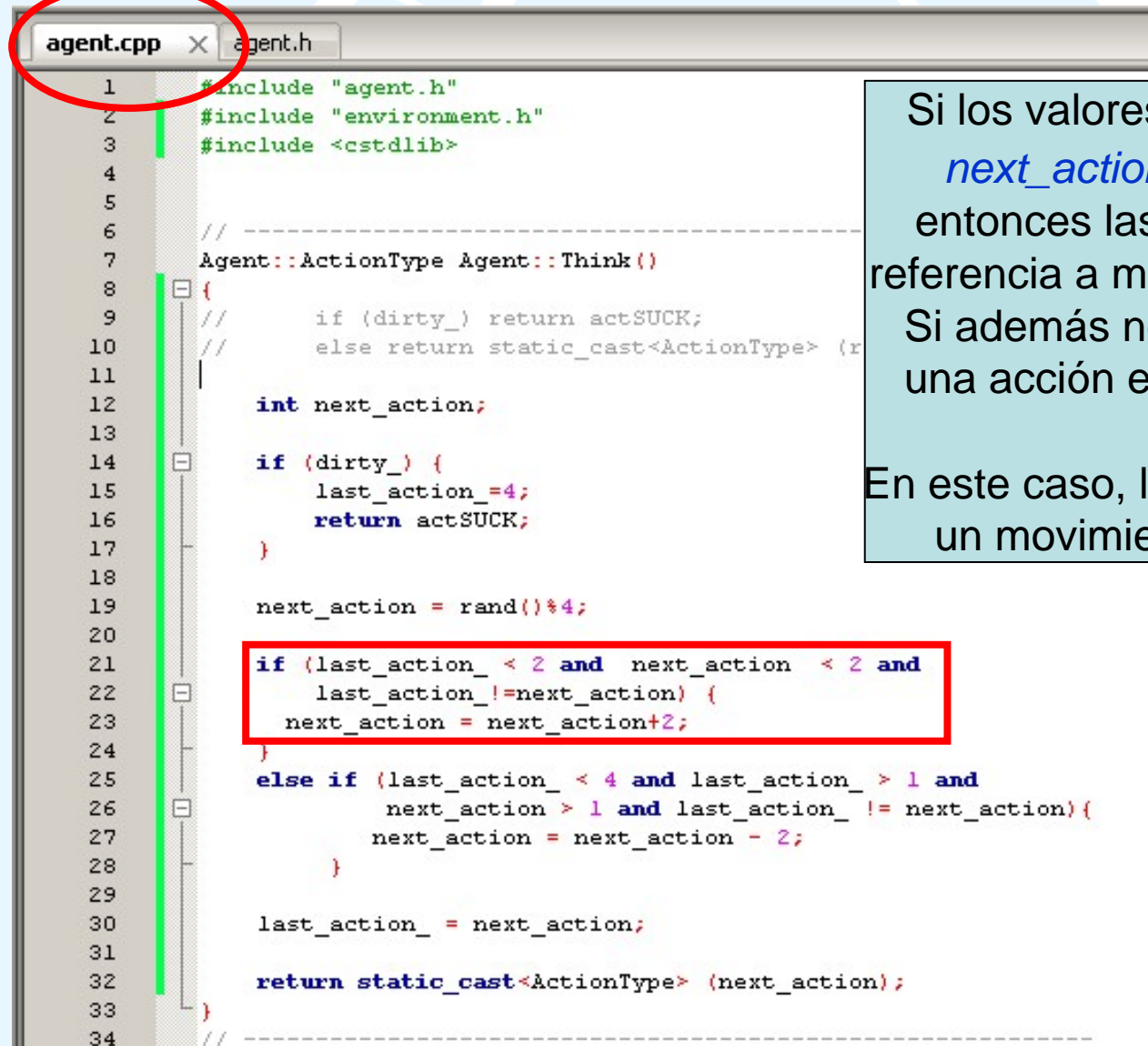


```
1  #include "agent.h"
2  #include "environment.h"
3  #include <cstdlib>
4
5
6  // -----
7  Agent::ActionType Agent::Think()
8  {
9      // if (dirty_) return actSUCK;
10     // else return static_cast<ActionType> (r
11
12     int next_action;
13
14     if (dirty_) {
15         last_action_ = 4;
16         return actSUCK;
17     }
18
19     next_action = rand()%4;
20
21     if (last_action_ < 2 and next_action < 2 and
22         last_action_ != next_action) {
23         next_action = next_action + 2;
24     }
25     else if (last_action_ < 4 and last_action_ > 1 and
26             next_action > 1 and last_action_ != next_action){
27         next_action = next_action - 2;
28     }
29
30     last_action_ = next_action;
31
32     return static_cast<ActionType> (next_action);
33 }
34 // -----
```

Si no estaba sucia la casilla, entonces al igual que antes, genera un movimiento al azar que almaceno en *next\_action*

# 4. Implementación de un agente

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



```
1  #include "agent.h"
2  #include "environment.h"
3  #include <cstdlib>
4
5
6  // -----
7  Agent::ActionType Agent::Think()
8  {
9      //      if (dirty_) return actSUCK;
10     //      else return static_cast<ActionType> (r
11
12     int next_action;
13
14     if (dirty_) {
15         last_action_ = 4;
16         return actSUCK;
17     }
18
19     next_action = rand()%4;
20
21     if (last_action_ < 2 and next_action < 2 and
22         last_action_ != next_action) {
23         next_action = next_action + 2;
24     }
25     else if (last_action_ < 4 and last_action_ > 1 and
26             next_action > 1 and last_action_ != next_action){
27         next_action = next_action - 2;
28     }
29
30     last_action_ = next_action;
31
32     return static_cast<ActionType> (next_action);
33 }
34 // -----
```

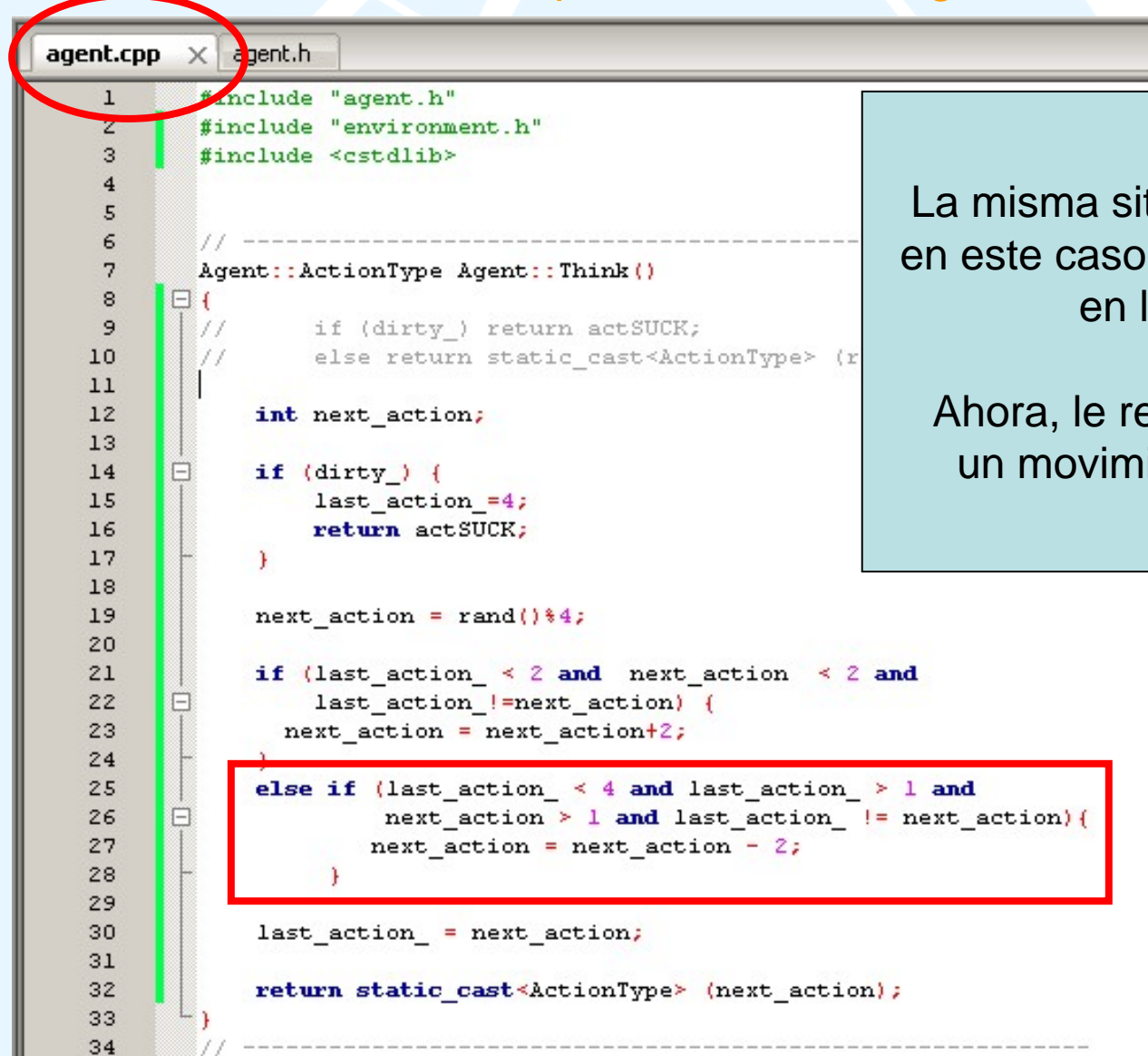
Si los valores de *last\_action\_* y de *next\_action* son inferiores a **2**, entonces las dos acciones hacen referencia a movimiento en la vertical. Si además no coinciden, entonces una acción es la opuesta a la otra.

En este caso, le sumo **2** para provocar un movimiento en la horizontal.



# 4. Implementación de un agente

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



```
1  #include "agent.h"
2  #include "environment.h"
3  #include <cstdlib>
4
5
6  // -----
7  Agent::ActionType Agent::Think()
8  {
9      //      if (dirty_) return actSUCK;
10     //      else return static_cast<ActionType> (r
11
12     int next_action;
13
14     if (dirty_) {
15         last_action_ = 4;
16         return actSUCK;
17     }
18
19     next_action = rand()%4;
20
21     if (last_action_ < 2 and next_action < 2 and
22         last_action_ != next_action) {
23         next_action = next_action + 2;
24     }
25     else if (last_action_ < 4 and last_action_ > 1 and
26             next_action > 1 and last_action_ != next_action){
27         next_action = next_action - 2;
28     }
29
30     last_action_ = next_action;
31
32     return static_cast<ActionType> (next_action);
33 }
34 // -----
```

La misma situación anterior, pero en este caso, con los movimientos en la horizontal.

Ahora, le resto **2** para provocar un movimiento en la vertical.

# 4. Implementación de un agente

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

```
1 #include "agent.h"
2 #include "environment.h"
3 #include <cstdlib>
4
5
6 // -----
7 Agent::ActionType Agent::Think()
8 {
9     // if (dirty_) return actSUCK;
10    // else return static_cast<ActionType> (r
11
12    int next_action;
13
14    if (dirty_) {
15        last_action_ = 4;
16        return actSUCK;
17    }
18
19    next_action = rand()%4;
20
21    if (last_action_ < 2 and next_action < 2 and
22        last_action_ != next_action) {
23        next_action = next_action + 2;
24    }
25    else if (last_action_ < 4 and last_action_ > 1 and
26        next_action > 1 and last_action_ != next_action) {
27        next_action = next_action - 2;
28    }
29
30    last_action_ = next_action;
31
32    return static_cast<ActionType> (next_action);
33 }
34 // -----
```

Termino el método, almacenado en *last\_action\_* la acción elegida y devolviendo dicha acción.



# 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 o rar) que llamado “***practica1***” contenga dos carpetas:

- Una de las carpetas con la memoria de la práctica (en formato pdf)
- La otra carpeta con los archivos “***agente.cpp***” y “***agente.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. 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 4 puntos.
2. La defensa de la práctica
  - se evalúa APTO o NO APTO. APTO equivale a 4 puntos, NO APTO implica tener un 0 en esta práctica.
3. Evaluación de la solución propuesta
  - se evalúa de 0 a 2.
  - 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 2 puntos*) :

- 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.
- Será evaluada sobre un mapa distinto a los aportados junto con el material de la práctica (***mapaExamen.map***)
- 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”***

- El **valor mínimo y máximo** para interpolar la calificación concreta de cada alumno en este apartado se obtendrá de los resultados obtenidos por las prácticas entregadas por todos los que pertenecen al mismo grupo de teoría.

# 5. Evaluación de la práctica

¿Dónde y cuándo se entrega?

- Se entrega en la aplicación de gestión de prácticas de la asignatura

***<http://decsai.ugr.es> → Entrega de Prácticas***

- La fecha tope de entrega será particular para cada grupo de prácticas y se realizará del día 21 al 27 de Marzo.
- La hora concreta para cada grupo se notificará mediante un correo electrónico.