# INTELIGENCIA ARTIFICIAL

E.T.S. de Ingenierías Informática y de Telecomunicación

# Práctica 2

## Agentes Reactivos

(El robot trufero)

**DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIAL**

**UNIVERSIDAD DE GRANADA**

Curso 2014-2015

# 1. Introducción

## 1.1. Motivación

La segunda práctica de la asignatura ***Inteligencia Artificial*** consiste en el diseño e implementación de un agente reactivo, capaz de percibir el ambiente y actuar de acuerdo a unas reglas simples predefinidas. Se trabajará con un simulador software. Para ello, se proporciona al alumno un entorno de programación, junto con el software necesario para simular el entorno. Es esta práctica, se diseñará e implementará un agente reactivo basado en los ejemplos del libro *Stuart Russell, Peter Norvig, "Inteligencia Artificial: Un enfoque Moderno", Prentice Hall, Segunda Edición, 2004*. El simulador que utilizaremos fue desarrollado por el profesor Tsung-Che Chiang de la NTNU (NorwegianUniversity of Science and Technology, Taiwan).

Originalmente, el simulador estaba orientado a experimentar con comportamientos en aspiradoras inteligentes. Las aspiradoras inteligentes son robots de uso doméstico de un coste aproximado entre 100 y 300 euros, que disponen de sensores de suciedad, un aspirador y motores para moverse por el espacio (ver Figura 1). Cuando unas aspiradora inteligente se encuentra en funcionamiento, esta recorre toda la dependencia o habitación donde se encuentra, detectando y succionando suciedad hasta que, o bien ha terminado de recorrer la dependencia, o bien se le aplica algún otro criterio de parada (batería baja, tiempo límite, etc.). Algunos enlaces que muestran el uso de este tipo de robots son los siguientes:

- http://www.youtube.com/watch?v=C1mVaje_BUM
- http://www.youtube.com/watch?v=dJSc_EKfTsw

**Figura 1: Aspiradora inteligente**

Este tipo de robots es un ejemplo comercial más de máquinas que implementan técnicas de Inteligencia Artificial y, más concretamente, mediante la Teoría de Agentes. En su versión más simple (y también más barata), una aspiradora inteligente presenta un comportamiento reactivo puro: Busca suciedad, la limpia, se mueve, detecta suciedad, la limpia, se mueve, y continúa con este ciclo hasta que se da una cierta condición de parada. Otras versiones más sofisticadas permiten al robot *recordar* mediante el uso de representaciones icónicas como mapas, lo cual hace que el aparato ahorre energía y

sea más eficiente en su trabajo. Finalmente, las aspiradoras más elaboradas (y más caras) pueden, además de todo lo anterior, realizar un plan de trabajo de modo que se pueda limpiar la suciedad en el menor tiempo posible y de la forma más eficiente. Son capaces de detectar su nivel de batería y volver automáticamente al cargador cuando esta se encuentra a un nivel bajo. Estas últimas pueden ser catalogadas como *agentes deliberativos* (se estudiarán en el tema 3 de la asignatura *Búsqueda en espacios de estados*).

En esta práctica, centraremos nuestros esfuerzos en implementar el comportamiento de un "robot trufero" asumiendo un comportamiento reactivo. Utilizaremos las técnicas estudiadas en el tema 2 de la asignatura para el diseño de agentes reactivos. ¿Qué es un robot trufero? Bueno, como su propio nombre indica un robot qué recolecta trufas.

Se ha adaptado el software anterior para los satisfacer los requisitos que se plantean en esta práctica.

### 1.2. ¿Cómo funciona el robot trufero?

Como todos sabéis, las trufas son una variedad de hongo muy apreciada en gastronomía y cotizadísima (Un kilo puede llegar a costar 6000 euros). Este hongo crece bajo tierra y no es fácil de localizar. Tradicionalmente, se han utilizado animales con una alta capacidad olfativa (perros fundamentalmente) para detectarlos. La misión del robot trufero será en esta práctica la de sustituir al perro trufero.

Explicaremos, de una forma muy simplificada, cuál es el funcionamiento interno de un robot trufero. El robot (agente) vive en un mundo modelado mediante cuadrículas. Entre sus acciones, tiene la posibilidad de olfatear el suelo, extraer las trufas, moverse hacia adelante o girar. Adicionalmente, dispone de un sensor de choque que se activa cuando el robot encuentra un obstáculo durante su movimiento hacia la casilla donde pretendía moverse. En cada momento, el robot selecciona una acción entre las disponibles, atendiendo a la función interna de selección de acciones que se ha implementado en el agente. En nuestro caso, las acciones serán: avanzar (el robot está orientado hacia la casilla de arriba, a la de abajo, a la de la izquierda o a la derecha, por lo tanto avanza en una de esas direcciones), girar a la derecha sin avanzar, girar a la izquierda sin avanzar, olfatear la casilla, extraer las trufas de la casilla y no hacer nada. Los sensores en nuestro robot son de choque (que se activa cuando intenta avanzar y no puede al detectar objeto) y de nivel aromático de trufas (que detecta cuando se coloca sobre una casilla y aplica la acción de oler). Las Figuras 2, 3 y 4 muestran un ejemplo de las acciones del agente a diseñar según el simulador que utilizaremos en la práctica. En la Figura 2 se observa, con tonos de gris, el grado aromático de las trufas de cada casilla del mundo. El robot ha realizado un movimiento hacia abajo desde la casilla inmediatamente superior (flecha verde apuntando hacia abajo). Por otra parte, en la figura 3 se observa que, desde la posición inicial del robot, éste seleccionó la acción de moverse hacia arriba, hecho que activó el sensor de choque al encontrar un obstáculo (flecha de color rojo). Por tanto, el robot no pudo hacer tal movimiento. Por último, la Figura 4 muestra que el robot activó el sensor de oler en la casilla donde se encuentra (recuadro amarillo).

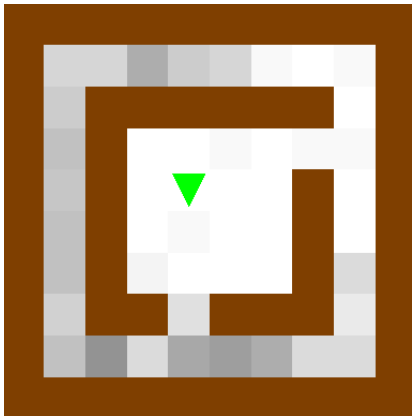

**Figura 2: Movimiento hacia debajo del robot desde la casilla superior**

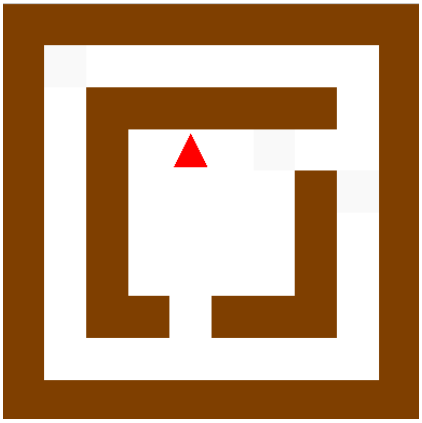

**Figura 3: Movimiento hacia arriba del robot. Choque con obstáculo**

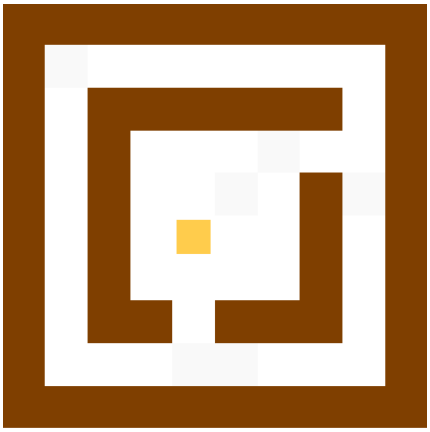

**Figura 4: Acción de oler el nivel el aroma de trufa que hay en una casilla**

A continuación, explicamos cuáles son los requisitos de la práctica, los objetivos concretos que se persiguen, el software necesario junto con su instalación, y una guía para poder programar el simulador.

## 2. Requisitos

Para poder realizar la práctica, es necesario que el alumno disponga de:

- Conocimientos básicos del lenguaje C/C++: tipos de datos, sentencias condicionales, sentencias repetitivas, funciones y procedimientos, clases, métodos de clases, constructores de clase.
- El entorno de programación **CodeBlocks** para Windows, que tendrá que estar instalado en el computador donde vaya a realizar la práctica. Este software se puede descargar desde la URL: http://www.codeblocks.org/.
- El entorno de simulación disponible en la web de la asignatura en el fichero ***Agent_2014_15.zip***.
- Los mapas del mundo del agente para validar su comportamiento, que ya vienen en el directorio map del fichero anterior.

Se proporciona una guía de instalación del software previamente mencionado en la sección 4 de este guión de prácticas.

## 3. Objetivo de la práctica

La práctica tiene como objetivo diseñar e implementar un agente reactivo que resuelva el problema del robot trufero de la mejor forma posible. Asumimos, como complejidad adicional, que el entorno es dinámico. Para ello, haremos las siguientes suposiciones:

- El entorno puede representarse en una matriz de tamaño máximo 10x10, donde los bordes de la matriz sólo pueden ser obstáculos (es decir, asumimos que el mundo es cerrado y no tiene salida).
- Cada casilla de la matriz puede ser transitable, y por consiguiente el robot puede posicionarse encima de ella, o intransitable, y por consiguiente contiene un obstáculo que impide que el robot avance o se posicione en ella. Si la casilla es transitable, entonces puede contener trufas.
- Tanto la estructura del mundo como la posición del robot dentro de ese mundo, como su orientación son desconocidas para el agente a priori.
- El nivel aromático de trufa de cada casilla se mide en números enteros comprendidos entre {0,20}, en concreto, los niveles que se pueden alcanzar son los siguientes:

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

- En cada instante de tiempo, cada casilla tiene probabilidad ***P*** de incrementar el nivel aromático de trufa. Esta probabilidad está asociada al mundo y puede tomar dos valores:
  - ***P*** = 0.01 que corresponde a mundos con trufas de crecimiento lento.
  - ***P*** = 0.02 que corresponde a mundos con trufas de crecimiento rápido.
- El agente no puede atravesar los obstáculos.
- La energía consumida por el agente es irrelevante para esta práctica.

- El tiempo es discreto (no continuo) para el agente. Así, hablaremos de **instantes de tiempo** t=1, 2, 3…, etc., simplificando el problema de esta forma.
- La cantidad de tiempo en cada simulación será de 2000 instantes.
- En cada instante de tiempo, el agente sólo puede realizar una única acción.
- En caso de realizar un movimiento para cambiar de posición en el entorno, el agente sólo puede moverse una casilla (avanzar, izquierda o derecha) en un instante de tiempo.
- En caso de realizar la acción de oler, el agente conocerá el número de trufas de la casilla en la que se encuentre, consumiendo un instante de tiempo.
- El agente puede extraer todas las trufas de la casilla en la que se encuentra consumiendo un instante de tiempo.

El comportamiento del robot deberá intentar maximizar la cantidad de trufas recolectadas en cada simulación.

### 3.1. ¿Qué métodos de diseño se pueden utilizar?

La elección del tipo de agente y el diseño para realizar la selección de acciones sólo está restringida a que sea uno de los estudiados en clase de teoría, en el tema 2 del programa de la asignatura, o una variación del mismo diseñada o consultada por el alumno al profesor. En este contexto, el alumno es libre de seleccionar cualquier técnica o técnicas a implementar para resolver el problema (sistemas de producción, arquitectura de pizarra, etc.). En concreto, las tareas a realizar deberán incluir:

- **Análisis del problema** (análisis del entorno, dificultades, sensores del agente, actuadores del agente, restricciones, requisitos, etc.)
- **Estructura interna del agente:** Estructura detallada de los módulos que componen el agente y el funcionamiento interno del mismo. Justificación del diseño del agente seleccionado.
- **Diseño de la función de selección de acciones:** Tipo de función de selección de acciones elaborada (funciones lógicas, redes lógicas, sistemas de producción, arquitectura de pizarra, etc.), justificando su elección en función de sus ventajas e inconvenientes. Explicación (con un ejemplo simple) de su funcionamiento.
- **Implementación** del agente en el simulador.

Para mayor detalle, consulte la sección 6 de este documento: *Evaluación y entrega de prácticas*.

# 4. Software

## 4.1. Instalación de CodeBlocks

Se puede descargar directamente de forma gratuita desde la web oficial http://www.codeblocks.org/ o desde la web de la asignatura. Aquellos alumnos que deseen un entorno alternativo, también lo pueden utilizar, aunque este documento no da soporte para su uso.

Para la instalación de **CodeBlocks** seguir los pasos que se indican en su página web.

## 4.2. Instalación del simulador Agent_2014_15

El simulador nos permitirá implementar el comportamiento del agente y visualizar las acciones en una interfaz de usuario mediante ventanas. Para instalarlo, siga estos pasos:

1. Descargue el fichero **Agent_2014_15.zip** desde la web de la asignatura, y cópielo su carpeta personal dedicada a las prácticas de la asignatura de ***Inteligencia Artificial***. Supongamos, para los siguientes pasos, que esta carpeta se denomina "**U:\IA\practica2**".
2. Desempaquete el fichero en la raíz de esta carpeta y aparecerán estas tres carpetas "**include**", "**lib**" y "**map**".
3. Ya está instalado el simulador. A continuación, el siguiente paso es hacer la puesta a punto del entorno **CodeBlocks** para poder compilar el proyecto.

## 4.3. Compilación del simulador en CodeBlocks

Antes de poder compilar el proyecto del simulador, es necesario realizar ciertos ajustes para incluir las bibliotecas adicionales de la práctica. Para ello, siga los siguientes pasos:

1. Inicie **CodeBlocks** por primera vez. En esta primera ejecución, **CodeBlocks** nos pide que seleccionemos un compilador de C/C++ por defecto (Figura 5). Sólo lo pedirá esta vez, ya que esta información quedará registrada en su configuración interna. Seleccionaremos el compilador **GNU GCC Compiler** (la primera opción) y pulsaremos el botón **ok**.

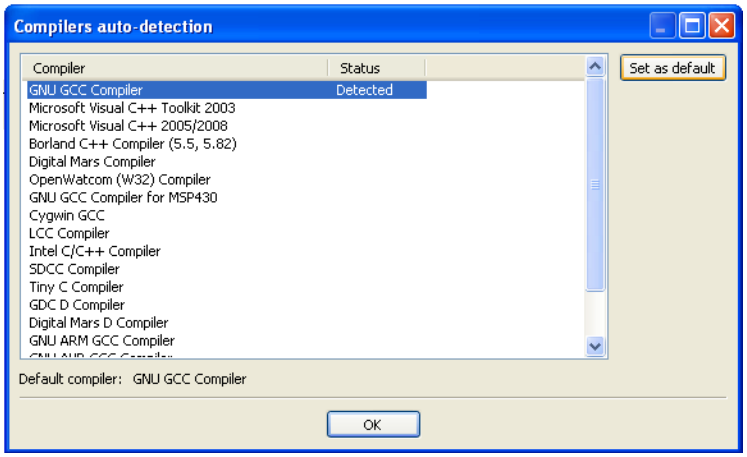


**Figura 5: Selección del compilador por defecto de CodeBlocks**

2. En esta primera ejecución de **CodeBlocks**, también se nos pregunta si deseamos que se asocien las extensiones de ficheros de programación en C/C++ a este entorno de programación. Si es lo que deseamos, pulsaremos sobre la opción
"**Yes, associateCode::Blocks with every supported type (including project files from other IDEs)**",
y pulsaremos sobre el botón **ok** (Figura 6). **CodeBlocks** tampoco volverá a hacer esta consulta en futuras ejecuciones.

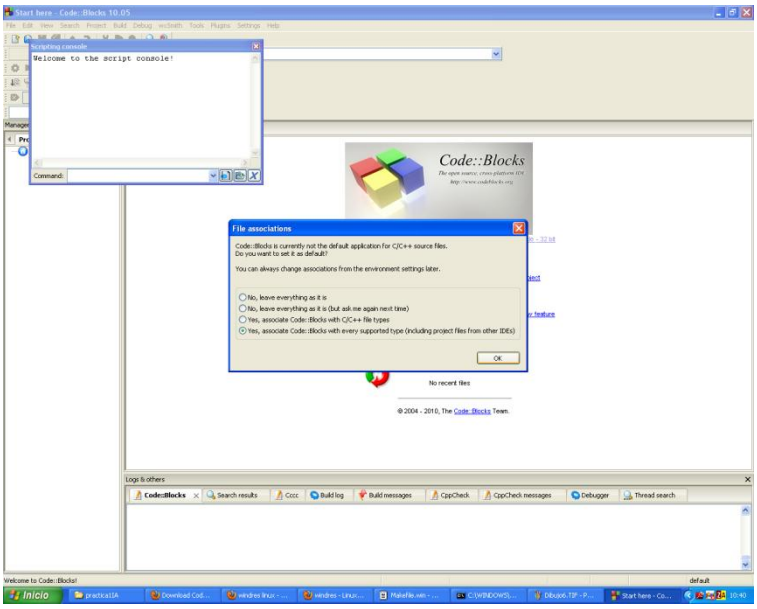


**Figura 6: Asociación de extensiones de ficheros de programación en C/C++**

3. Para abrir el **proyecto del simulador Agent_2015_15**, pulsaremos sobre la opción "**Open an existing project"** de la ventana principal del entorno **CodeBlocks**(Figura 7). Aparecerá una nueva ventana para que viajemos a la carpeta donde hemos instalado el simulador y lo

seleccionemos. Para ello, deberemos escoger el fichero "**Agent_2014_15.cpb"** y pulsaremos en el botón **Abrir**.

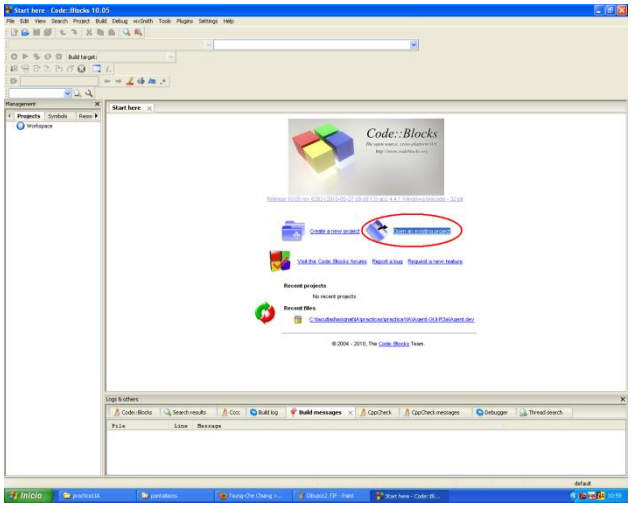

**Figura 7: Abrir un fichero existente en CodeBlocks**

4. De nuevo, **CodeBlocks** nos pide que confirmemos el compilador por defecto a usar para compilar el simulador. Por tanto, volvemos a seleccionar el compilador **GNU GCC Compiler** y pulsamos el botón **ok** (Figura 8).

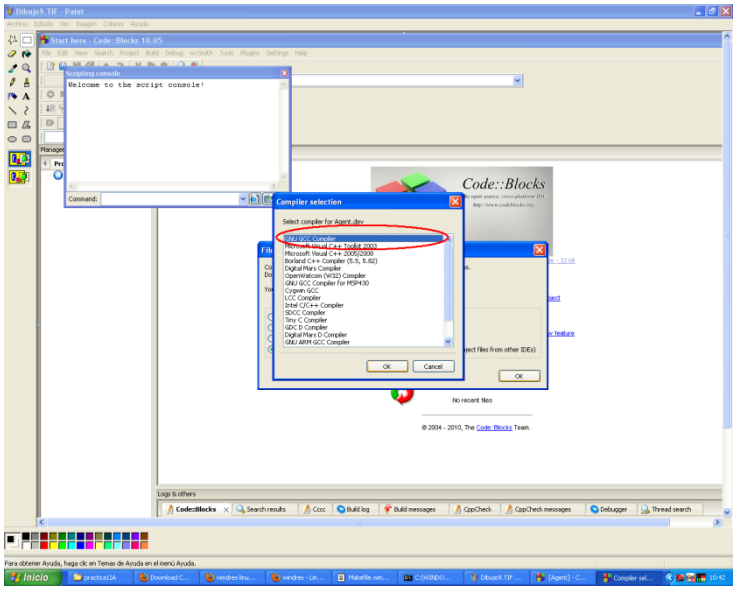

**Figura 8: Selección del compilador para el proyecto *Agent.dev***

5. Tras realizar los pasos anteriores, hemos conseguido abrir el proyecto. En la parte izquierda de la pantalla podremos ver el proyecto **Agent_2014_15**, junto con sus ficheros de código fuente

.**cpp** (subcarpeta **Sources**) y los correspondientes ficheros de cabecera **.h** (subcarpeta **Headers**), tal y como muestra la Figura 9. **Los ficheros que mayor interés proporciona para la práctica son *agent.h* y *agent.cpp,*** los cuales tendremos que modificar para implementar el comportamiento deseado del agente.

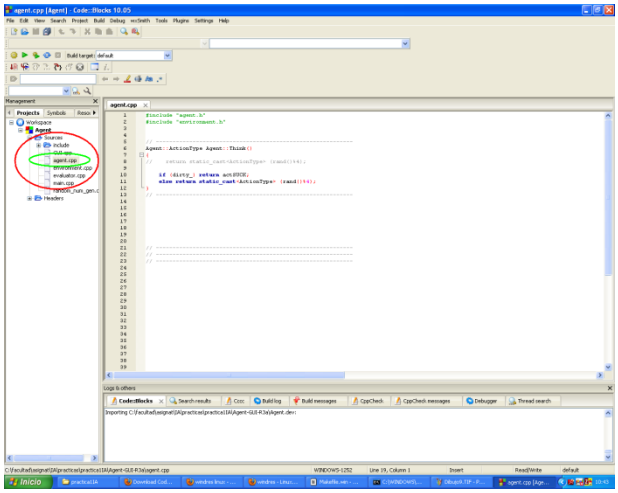

**Figura 9: Ficheros del proyecto *Agent***

6. **Compilación y depuración de errores**. El proyecto se compila desde la opción "**Build**" del menú "**Build**" (Figura 10). En el caso de existencia de errores, el programa no generará ningún fichero ejecutable y mostrará los errores encontrados durante el proceso de compilación.

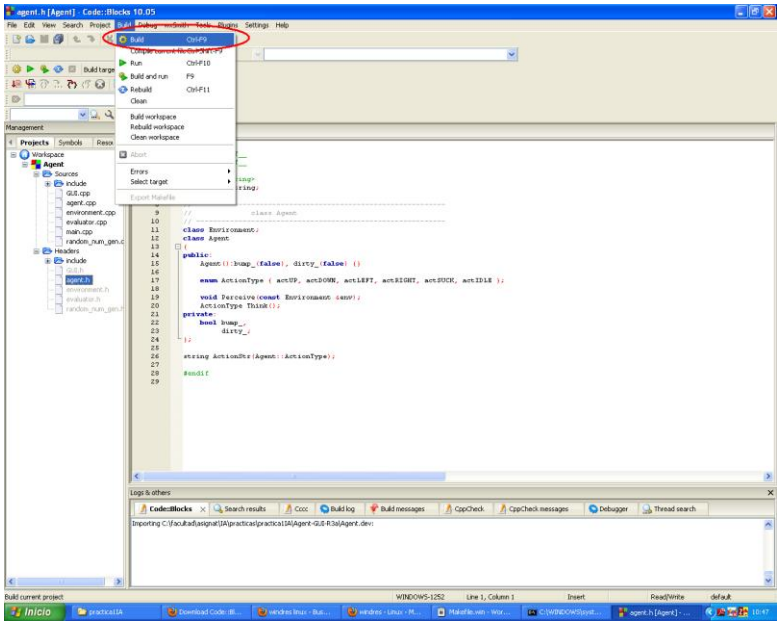

**Figura 10: Compilación del proyecto**

## 4.4. Ejecución del simulador

Una vez compilado el proyecto del simulador, para ejecutarlo pulsaremos sobre la opción "**Run**" del menú **"Build**" (alternativamente, también podemos hacer doble clic sobre el programa **Agent.exe** generado en la carpeta del proyecto tras su compilación). Aparecerá una ventana como la que se muestra en la Figura 11.

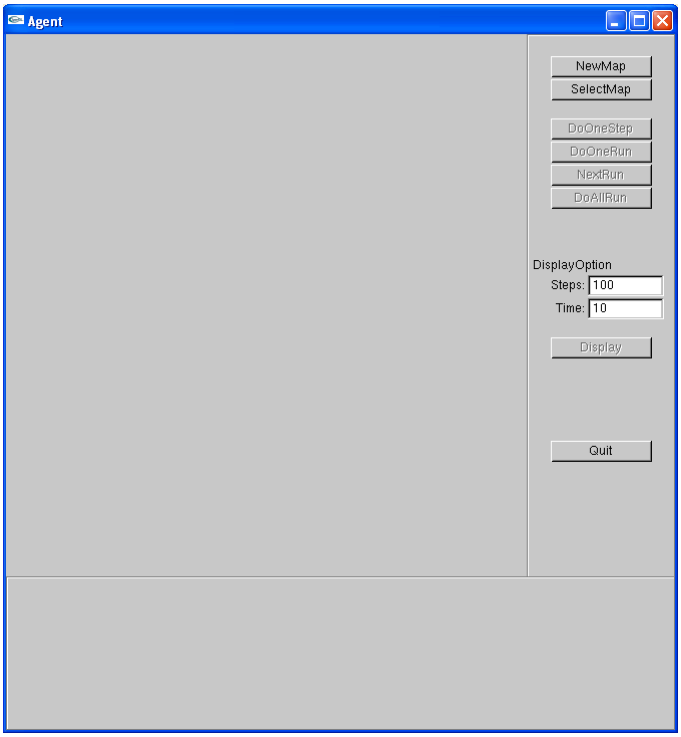


**Figura 11: Vista del simulador**

En esta ventana, la opción que nos interesa se encuentra en el botón "**SelectMap**", que nos permite seleccionar un mapa del entorno. Seleccionar "**Agent.map**". En la sección 5 describimos cómo podemos crearnos nuestros propios mapas para realizar la práctica.

Tras cargar el mapa "**Agent.map**", la vista del simulador cambia mostrando el mundo del agente, la posición donde comienza, y se activan las opciones de ejecución de simulación (Figura 12): "**DoOneStep**", "**DoOneRun**", etc.. Las que nos interesan son:

- **DoOneStep:** Permite avanzar un instante de tiempo, suficiente para que el agente realice una de las acciones disponibles (avanzar, izquierda o derecha, limpiar 1 punto de suciedad, o quedarse quieto).
- **DoOneRun:** Permite ejecutar un número de pasos establecidoa priori **en el campo de texto "Steps"**, en la sección "**DisplayOption**".
- **Display:** Es equivalente a "**DoOneRun**" pero, además, muestra el recorrido del agente en el simulador.

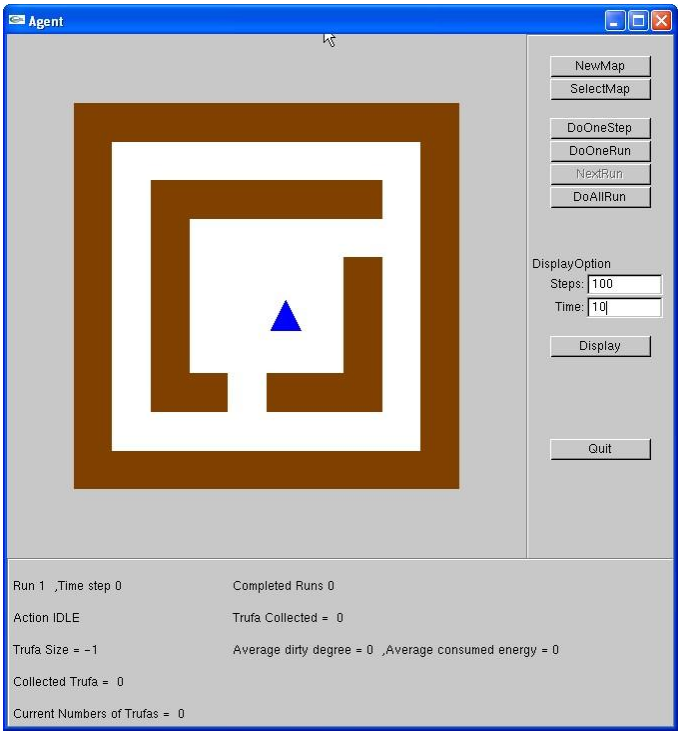


**Figura 12: Vista del simulador en mapa**

Adicionalmente, también resultan de interés para la práctica las variables de estado que se muestran en la parte inferior:

- **Run**: Ejecución actual.
- **Time step**: Instante de tiempo actual dentro de la ejecución en curso.
- **Trufa Size**: Número de trufas que hay en dicha casilla. (-1 indica que recientemente no se ha olfateado)
- **Collected Trufa**: Cantidad de trufas recolectadas hasta el momento.
- **CurrentNumbers of Trufas**: Cantidad de trufas que hay en este instante en el mapa.
- **Etc**.

Se invita al alumno a experimentar con las opciones anteriores. Aclaramos que, en este ejemplo, el comportamiento implementado en el agente consiste en seleccionar aleatoriamente una acción.

### 4.5. Mapas adicionales

Se pone a disposición del alumno mapas adicionales para que éste evalúe los comportamientos implementados. Estos mapas ya están dentro de la carpeta map y se llaman *mapa1.map*, *mapa2.map*, *mapa3.map* (versiones de mapas de crecimiento de trufas lento) y *mapa1_rap.map*, *mapa2_rap.map*, *mapa3_rap.map* (versiones de los mapas anteriores pero de crecimiento rápido de trufas).

## 5. Implementación del agente

### 5.1. Descripción de los ficheros del simulador

El simulador se encuentra instalado en la carpeta seleccionada en el apartado 4.2 de esta memoria (por defecto, **Agent_2015_15**). Esta carpeta contiene los ficheros de código fuente y las bibliotecas necesarias para su correcta compilación. En concreto, estos son:

- Carpeta **include**: Contiene ficheros de código fuente y objeto (**.o**) necesarios 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 **lib**: Contiene las bibliotecas (**.a**) básicas para poder acceder a la interfaz gráfica de OpenGL y la creación de ventanas en Windows.
- Carpeta **map**: Contiene los mapas disponibles en el simulador para modelar el mundo del agente. Discutiremos este aspecto en el apartado 5.3 de este guión.
- 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.).
- 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.), los cuales se muestran en la parte inferior de la ventana principal del simulador.
- 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.

## 5.2. Implementación del agente

Entre todos los ficheros descritos en el apartado 5.1 de este guión, únicamente será necesario modificar los archivos **agent.cpp** y **agent.h** para poder llevar a cabo las tareas de esta práctica. Concretamente, el agente se implementa en la clase **Agent**, definida en **Agent.h**. El agente tiene dos variables miembro, de tipo **bool**:

- Variable **bump_**: Esta variable la obtiene automáticamente el agente desde el entorno. Está asociada al sensor de choque, y tiene el valor **true** si el agente ha chocado contra un obstáculo intentando hacer el movimiento anterior, y el valor **false** en caso contrario.
- Variable **trufa_size_**: Esta variables se obtiene del entorno tras aplicar la acción de oler. Toma el valor -1 si no se ha realizada una operación de olfateo de la casilla, y un valor entre 0 y 20 si se realizar dicha operación.

Además, el agente dispone de los siguientes métodos:

- Método **void Perceive(constEnvironment&env)**: Este método lo utiliza el simulador para que el agente pueda percibir el entorno. Como entrada, tiene una variable de tipo **Environment** con información sobre el mundo del agente. El cometido de **Perceive** en esta práctica consiste en asignar valores a las variables **bump_** y **trufa_size_** según la información leída por los sensores de choque y olfateo. De este modo, la implementación de este método debe ser la siguiente (**no debe ser modificada por el alumno para la elaboración de la práctica**):

```
void Agent::Perceive(constEnvironment&env) {
    trufa_size_ = env.TrufaAmount();
    bump_ = env.isJustBump();
}
```

- Método **ActionType Think()**: Este método contiene la función de selección de acciones del agente. **Este método deberá ser modificado por el alumno para implementar el comportamiento deseado del agente.** Como mínimo, debe tener en cuenta los valores de las variables internas **bump_** y **trufa_size_**. Como salida, este método devuelve la acción a realizar seleccionada. El tipo de salida, **ActionType**, está definido como tipo enumerado **enum** dentro de la clase **Agent**:

```
enum ActionType { actFORWARD, actTURN_L, actTURN_R, actSNIFF, actEXTRACT, actIDLE };
```

Por tanto, los valores que puede tener una variable de tipo **ActionType** podrán ser alguno de los 5 siguientes:

- Valor **actFORWARD**: Acción de moverse a una casilla adyacente a la actual. Al ser tipo enumerado, esta acción tiene el valor entero asociado 0. La casilla concreta viene determinada por la orientación actual del robot, valor que no es conocido por robot.
- Valor **actTURN_L**: Acción de girar su orientación 90% hacia la izquierda en relación a su orientación anterior, esta acción tiene el valor entero asociado 1.
- Valor **actTURN_R**: Acción de girar su orientación 90% hacia la derecha en relación a su orientación anterior. Al ser tipo enumerado, esta acción tiene el valor 2 entero asociado.
- Valor **actSNIFF**: Acción de olfatear el nivel de trufas que hay en una casilla. Al ser tipo enumerado, esta acción tiene el valor entero asociado 3
- Valor **actEXTRACT**: Acción de extraer las trufas que hay en la casilla actual donde se encuentra el agente. Al ser tipo enumerado, esta acción tiene el valor entero asociado 4.
- Valor **actIDLE**: No produce ninguna acción. El agente permanece inmóvil. Al ser tipo enumerado, esta acción tiene el valor entero asociado 5.

Como ejemplo de implementación de un comportamiento del agente, el siguiente código (a implementar en el fichero **agent.cpp**) realiza una acción aleatoria, exceptuando la acción de no hacer nada:

```
Agent::ActionType Agent::Think() {

    switch(rand()%5) {
            case 0: return actFORWARD; break;
            case 1: return actTURN_L; break;
            case 2: return actTURN_R; break;
            case 3: return actSNIFF; break;
            case 4: return actEXTRACT; break;
    }
}
```

Se recomienda al alumno modificar el código del método **Think** en el fichero **agent.cpp** para comprobar diferentes comportamientos simples (sólo moverse en una dirección, alternar entre moverse y oler y extraer, realizar algún movimiento dependiendo del sensor de choque, etc.), con el fin de familiarizarse con el proceso de implementación y prueba en el simulador.

Adicionalmente, si el alumno estima necesario generar nuevas variables, vectores o matrices miembro de la clase **Agent**, así como crear nuevos métodos auxiliares que sean necesarios para desarrollar las tareas (según el diseño que haya realizado para el comportamiento del agente en la práctica), estas deben declararse e inicializarse de la siguiente forma:

- **Declaración de nuevas variables miembro**: En el fichero **agent.h**, la sección privada de la clase (líneas 31-33) contiene declaradas las variables miembro **bump_** y **trufa_size_**. Las variables adicionales que desee crear el alumno deberán declararse en esta sección. Por ejemplo, si el alumno estimase necesaria una variable **aux**, el código privado de la clase quedaría de la siguiente forma:

```
…
private:
    bool bump_
    int trufa_size_;
    int aux;
…
```

- **Inicialización de variables miembro**: Las variables deben ser inicializadas en el **constructor por defecto** de la clase **Agent**. En el código inicial que se proporciona en el simulador:

```
Agent() {
        bump_= false;
        trufa_size_= -1;
}
```

Para inicializar la nueva variable **aux** en el constructor al valor 0, bastaría con modificar el código de la siguiente forma:

```
Agent() {
        bump_= false;
        trufa_size_= -1;
        aux= 0;
}
```

Una vez implementada la inicialización, la variable estaría lista para ser usada y modificada en el método **Think** del agente.

- **Declaración de nuevos métodos:** Deben declararse en el fichero **agent.h, dentro de la clase Agent**. Por cuestiones formales, deberían declararse en la sección privada de la clase. Por ejemplo, si deseamos crear un nuevo método **void MiMetodo(),** que se encargue de realizar algún tipo de procesamiento auxiliar, este se podría declarar de la siguiente forma:

```
…
    private:
        bool bump_
        int trufa_size_;
        int mimundo[10][10],

        void MiMetodo();
…
```

- **Implementación de los nuevos métodos:** Mientras que la declaración de los métodos se realiza durante la definición de la clase (fichero **agent.h**), su implementación debe incluirse en el fichero **agent.cpp**. Así, para implementar el método previamente declarado **MiMetodo**, se debería incluir en el fichero **agent.cpp** el siguiente código (NOTA: No olvidar incluir el prefijo **Agent::** previo al nombre del método para indicar que es método de la clase **Agent** y no una función independiente):

```
        …
    void Agent::MiMetodo() {
            // Código fuente del método
    }
    …
```

## 5.3. Generación de mapas

Además de los mapas proporcionados por defecto y por el profesorado en la web de la asignatura, se da la posibilidad al alumno de generar sus propios mapas para realizar pruebas adicionales. **Los mapas realizados deben colocarse en la carpeta "map" del simulador.**

Un mapa es un fichero de texto cuyo nombre acaba obligatoriamente en la extensión "**.map**". Su contenido se ilustra en la Figura 13, la cual muestra el contenido del mapa por defecto proporcionado por el simulador:

```
// Initial position, dirty probability, random seed, map
5 5 0.01 1
O O O O O O O O O O
O - - - - - - - - O
O - O O O O O O - O
O - O - - - - - - O
O - O - - - - O - O
O - O - - - - O - O
O - O - - - - O - O
O - O O - O O O - O
O - - - - - - - - O
O O O O O O O O O O
```

**Figura 13: Esquema de fichero de definición de mapas**

- La primera línea del fichero contiene un comentario que indica el significado de los valores en la siguiente línea.
- La segunda línea del fichero contiene la posición (X,Y) donde el agente comienza la exploración (comenzando a contar desde la coordenada (0,0)), la probabilidad de que la suciedad de cada casilla aumente en cada instante de tiempo. El último valor es el número de mapa (valor fijo igual a 1, irrelevante para el desarrollo de esta práctica). Todos estos valores están separados por un único espacio en blanco.
- Las siguientes líneas contienen las filas y columnas del mapa que define el mundo del agente. Un valor **O** (**o** mayúscula) indica que en la casilla hay un obstáculo, mientras que un valor **–** (signo negativo) sirve para definir una casilla vacía. Todos los valores de una misma fila se deben separar por un único espacio en blanco.
- El mapa debe tener un tamaño de 10x10 fijo.
- Todas las casillas del borde del mapa deben tener valor **O**, con el fin de satisfacer la restricción comentada en el apartado 3 acerca de que la habitación esté cerrada.

Con estas indicaciones, y guardando el mapa como fichero de texto con extensión "**.map**" en la carpeta "**map**" del simulador (editándolo, por ejemplo, con el programa **notepad** de Windows), el alumno dispone de nuevos mecanismos para evaluar su implementación sobre el comportamiento del agente en entornos adicionales a los proporcionados.

## 6. Evaluación y entrega de prácticas

La calificación final de la práctica será la media ponderada de los aspectos susceptibles de evaluación en la práctica. Para cada comportamiento se realizarán 10 ejecuciones del algoritmo en un mapa distinto a los proporcionados junto con el software.

La calificación de la eficacia del agente diseñado se obtendrá como la media de la cantidad de trufas recolectadas a lo largo de las 10 ejecuciones. A mayor valor, mayor será la calificación. En cada ejecución, se realizarán 2000 acciones por el agente para comprobar su comportamiento.

La **calificación final** de la práctica se calculará de la siguiente forma:

- Se entregará una memoria de prácticas (ver apartado 6.1 de este guión) al finalizar las tareas a realizar. La fecha límite de la entrega de la memoria dependerá de cada grupo de prácticas (del lunes 13 al viernes 17 de abril) según se describe más abajo, y se realizará a través del acceso identificado de DECSAI.
- Se realizará una defensa de la práctica durante la clase en la semana del 20 al 24 de abril, según el grupo. El objetivo de esta defensa es verificar que la memoria entregada ha sido realizada por el alumno. Por tanto, esta defensa requerirá de la ejecución del simulador con los comportamientos realizados por los alumnos, en clase de prácticas, y de la respuesta a cuestiones del trabajo realizado. La calificación de la defensa será **APTO** o **NO APTO**. Una calificación **NO APTO** en la defensa implica el suspenso con calificación **0** en la práctica. Una calificación **APTO** permite al alumno obtener la calificación según los criterios explicados en el punto siguiente.
- La práctica se califica numéricamente de 0 a 10. Se evaluará como la suma de los siguientes criterios:
    - La memoria de prácticas se evalúa de 0 a 3. Para obtener la máxima calificación, es de especial relevancia que el alumno explique en la memoria claramente el diseño propuesto para el comportamiento del agente (diseño de la estructura del agente, diseño de la función de selección de acciones, etc.), así como un buen análisis de la solución aportada. También se valorará positivamente que el alumno explique con claridad la relación entre el diseño realizado y la implementación.
    - Las cuestiones realizadas por el profesor durante la defensa de prácticas y correctamente respondidas por el alumno se evalúan de 0 a 3. El valor de cada pregunta es el mismo.
    - La eficacia de la solución se evalúa de 0 a 4 puntos. El alumno que obtenga la mejor solución ***(menor valor en la cantidad final de suciedad en la habitación)*** obtendrá una calificación máxima en este apartado de 4 puntos. El alumno que obtenga la peor solución obtendrá una calificación mínima de 0. El resto de alumnos obtendrá una calificación proporcional a la de los compañeros que hayan obtenido la mínima y máxima calificación, en función de la bondad de la solución.

- La **fecha de entrega de la memoria práctica** será la semana del 13 de Abril. Las fechas concretas para cada grupo se describen en la siguiente tabla

| Grupo de prácticas | Fecha límite |
|---|---|
| **A3, B3, C3 y D2** | 13 de Abril hasta las 23:59 horas |
| **A1, B1 y Doble Grado** | 14 de Abril hasta las 23:59 horas |
| **C1** | 16 de Abril hasta las 23:59 horas |
| **A2,B2, C2 y D1** | 17 de Abril hasta las 23:59 horas |

## 6.1. Restricciones del software a entregar y representación.

Se pide desarrollar un programa (modificando el código de los ficheros del simulador (**agent.cpp** y **agent.h**) con el comportamiento requerido para el agente. Estos ficheros deberán entregarse mediante la plataforma web de la asignatura, en un fichero ZIP que no contenga carpetas. El archivo ZIP deberá contener **sólo el código fuente de estos dos ficheros** con la solución del alumno, y la versión electrónica de la memoria en **PDF**. **No se evaluarán aquellas prácticas que contengan ficheros ejecutables o virus.**

El fichero ZIP debe contener una memoria de prácticas en formato PDF (no más de 5 páginas) que, como mínimo, contenga los siguientes apartados:

1. Descripción de la solución planteada.
2. Resultados obtenidos por la solución aportada en los distintos mapas que se entregan con el agente.
3. Descripción de otras estrategias descartadas.