Práctica Agente Inteligente: Aspiradora

5.2. Implementación del agente

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 dirty_: Esta variables la obtiene automáticamente el agente desde el entorno. Está asociada al sensor de suciedad, y tiene el valor true si el agente ha detectado suciedad en la casilla donde se encuentra, y el valor false en caso contrario.

Además, el agente dispone de los siguientes métodos:

- Método void Perceive(const Environment &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 dirty_ según la información leída por los sensores de choque y suciedad. 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(const Environment &env) {
bump_ = env.isJustBump();
dirty_ = env.isCurrentPosDirty();
}
```

- 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 dirty_. 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 { actUP, actDOWN, actLEFT, actRIGHT, actSUCK, actIDLE };
```

Por tanto, los valores que puede tener una variable de tipo ActionType podrán ser alguno de los 6 siguientes:

- Valor actUP: Acción de moverse a la casilla inmediatamente superior a la actual. Al ser tipo enumerado, esta acción tiene el valor entero asociado 0.
- Valor actDOWN: Acción de moverse a la casilla inmediatamente inferior a la actual. Al ser tipo enumerado, esta acción tiene el valor entero asociado 1.

Valor actLEFT: Acción de moverse a la casilla inmediatamente a la izquierda a la actual. Al ser tipo enumerado, esta acción tiene el valor entero asociado 2.
Valor actRIGHT: Acción de moverse a la casilla inmediatamente a la derecha a la actual. Al ser tipo enumerado, esta acción tiene el valor entero asociado 3.
Valor actSUCK: Acción de limpiar 1 unidad de suciedad 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 succión de suciedad en caso de que la casilla en la que se encuentra el agente esté sucia, o un movimiento a una casilla adyacente seleccionada de forma aleatoria en otro caso:

```
Agent::ActionType Agent::Think() {
        int i;
if (dirty_) return actSUCK;
else i= rand()%4;
switch(i) {
case 0: return actUP; break;
case 1: return actDOWN; break;
case 2: return actLEFT; break;
case 3: return actDOWN; break;
default: return actIDLE;
        }
}
```

1.- Análisis del Problema.

Nos proponian a darle memoria a la aspiradora, es decir que recordará cuando se había chocado. Este agente está en una de dos localizaciones, cada una de las cuales puede , o no, contener suciedad. Así, hay $2x2^2 = 8$ posibles estados del mundo. Sobre el estado inicial de la aspiradora no se sabe nada, podría ser cualquiera, el agente desconoce donde se encuentra.

En definitiva, lo más complicado de este problema, era el tratar de limpiar lo máximo posible, y consumir el minimo de energía, cuya suma daba el total de eficacia del agente.

2.- Descripción de la solución planteada.

Una vez me leí el capítulo 2 de teoría, busqueda no informada, llegue a la conclucion de que el agente esta en una de dos localizaciones, sucio o limpio, con lo cual podría expresarlos con 0 = limpio, 1 = sucio, por ejemplo, ademas esto tambien me permitía, establecer otra igualdad, para saber cuando ha chocado, 0 = Camino Libre, 1 = Pared , es un ejemplo. Mi primera solución trivial fue pues creo que lo ha hecho todo el mundo, con una matrix de bool, inicializada a 0 (false), y donde hubiera limpiado o chocado colocar un 1(true).
Pero en cuanto comence a implementarlo, vi que no era demasiado eficiente, y me dije qué como podía implementar una especie de tiempo, para que el agente recordará a cual debía ir, es decir, limpiar antes el que se visito antes del ultimo visitado. Una especie de variable Tiempo.

Al final me decanté por una matriz de caracteres(char), de 20x20, con unas coordenadas X e Y iniciadas a 10, para situarlas en el centro de la matriz, ademas de esto tres vectores char. Y esta idea me la dió la función del final puesto que retornaba “UP” , “DOWN” y esto, de forma en strings, ademas de que en agent.h, esta la biblioteca string.

Los tres vectores consisten en:

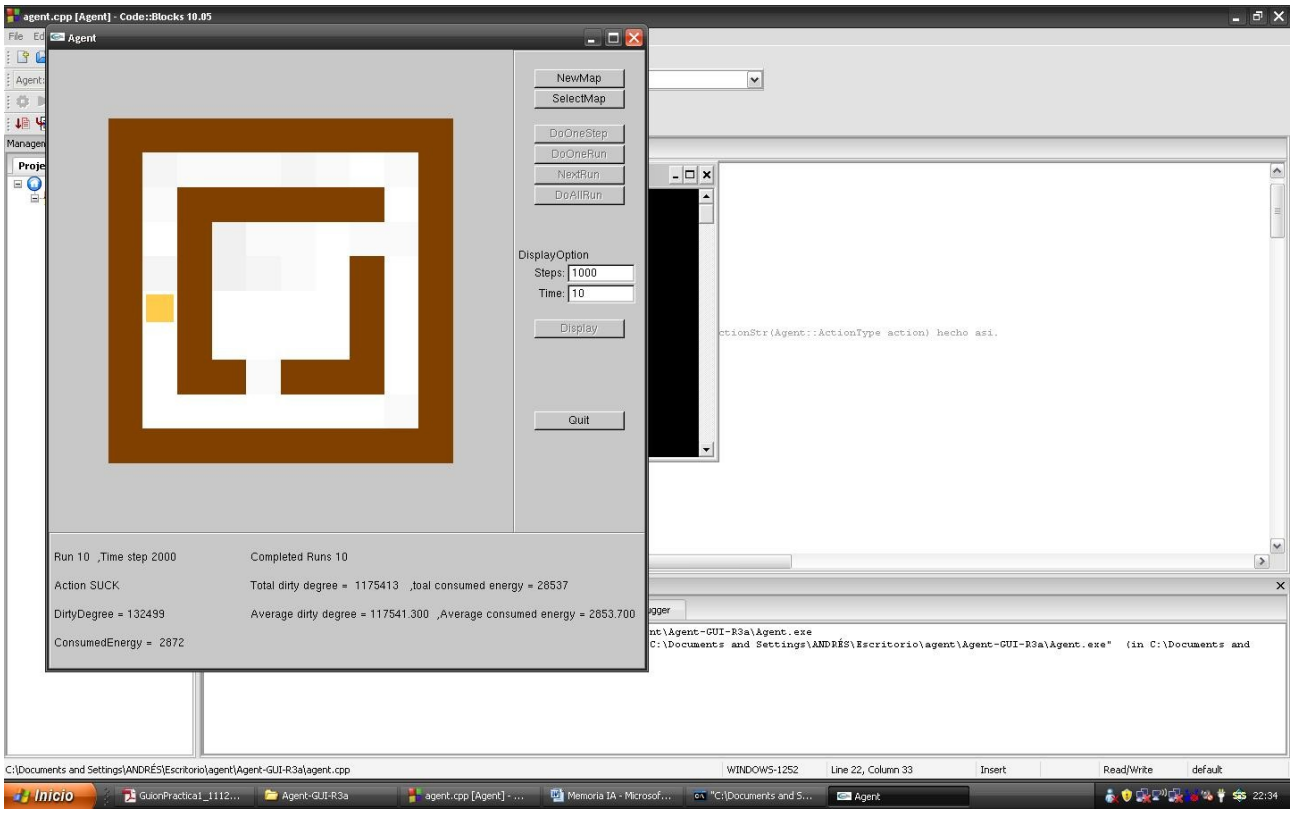
- Un array para controlar lo que habia hecho la aspiradora anteriormente, con 2000 elementos, ACT[2000] inicialiado a 'N' , y ACT[0] = 'I' de Inicio del vector.
- El segundo array, lo utilizo para controlar la etapa actual y dejarla memorizada en ese mismo vector, para recordar donde se ha movido, si ha aspirado y tal, de tamaño 10000.
- El tercer vector, llamado posicion, para que recuerde donde se ha chocado, de 4000 elementos.

Aparte de esto, he declarado un variable string estado, para ir actualizando el estado de la aspiradora, y he implementado 4 métodos string y uno void, donde se queda registrado el moviemiento, al que se le pasa un string. (Mi duda era si utilizar paso por referencia o no, por el tema de eficiencia).

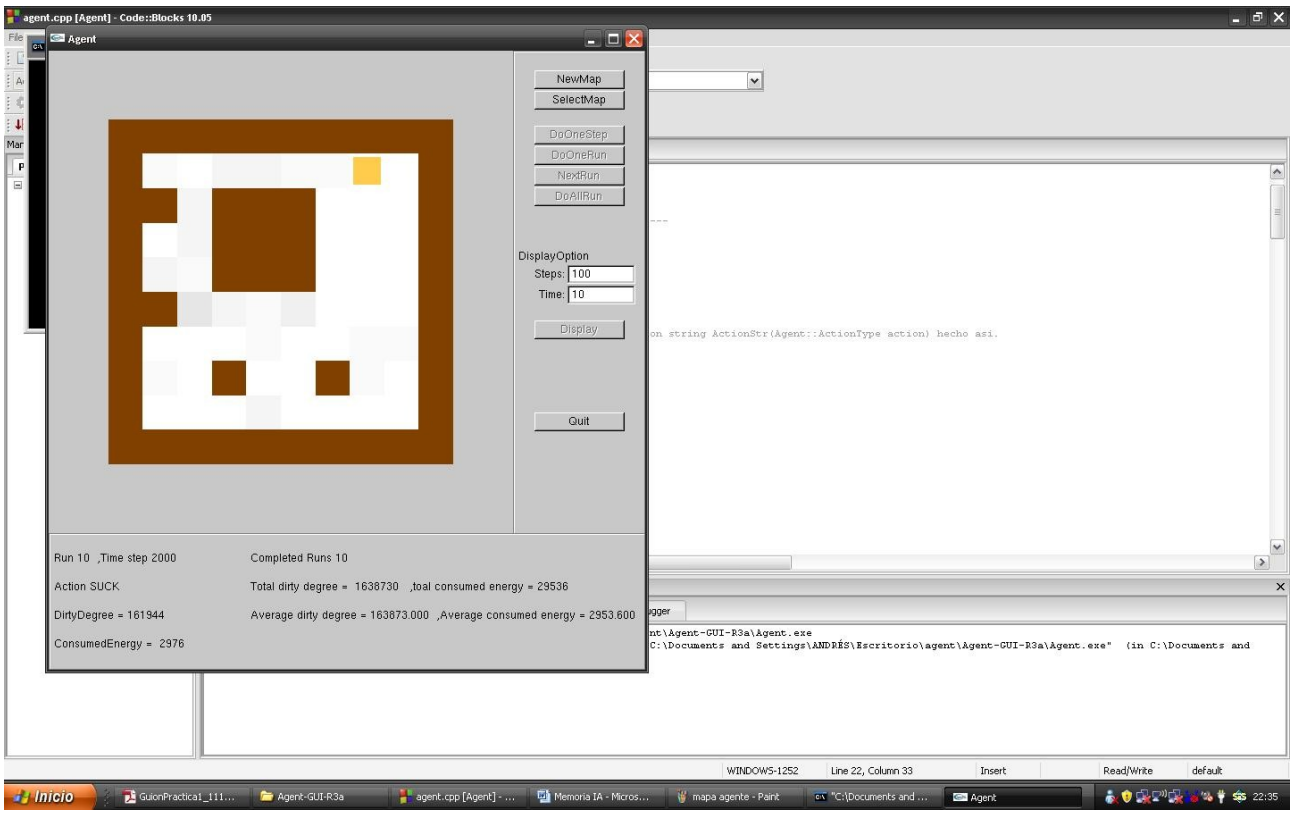
Basicamente mi gran problema ha sido el cómo recordar lo que hacía la aspiradora, para ello declare 3 etapas, una etapa actual, y la de despues. Y mediante if y manejando los arrays, creó que lo he podido hacer.

3.- Resultados obtenidos por la solución aportada en los distintos mapas.

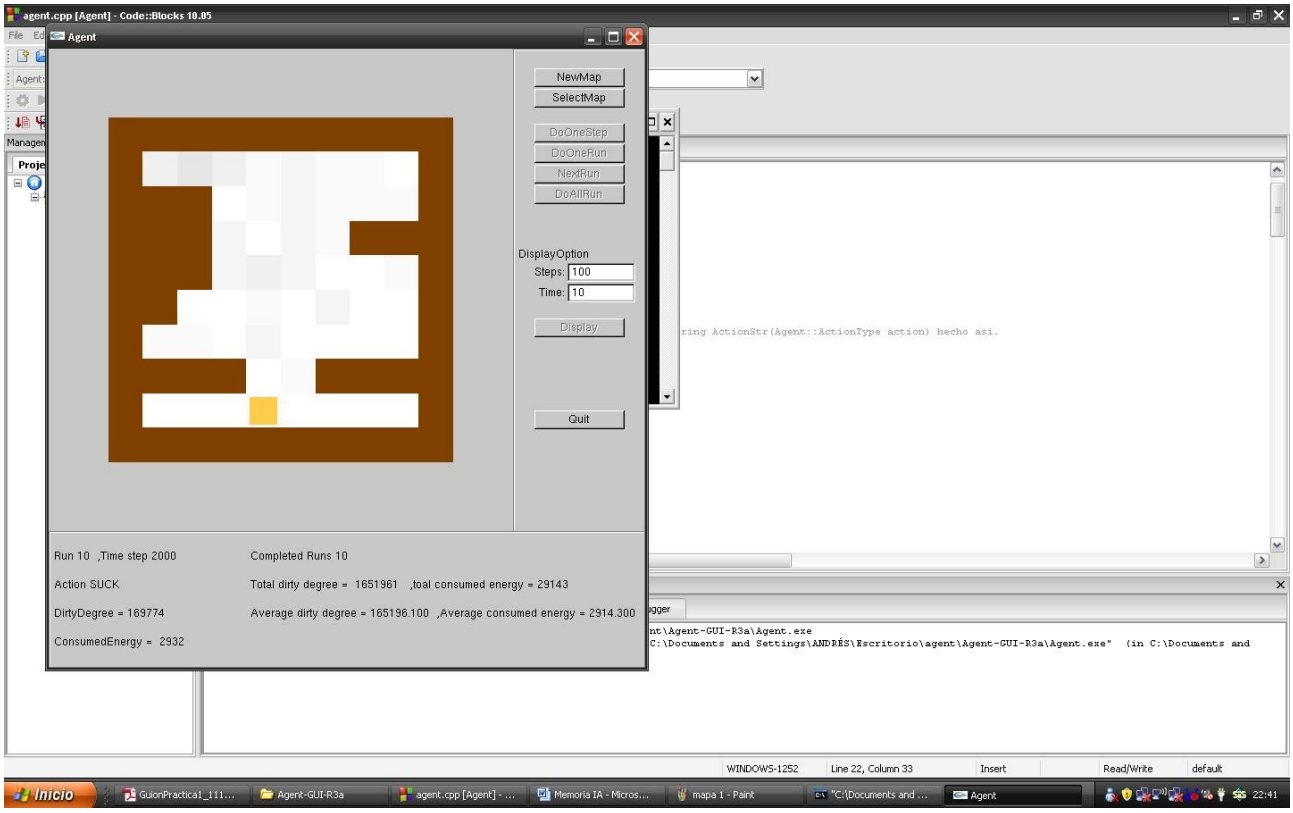
Agent.map



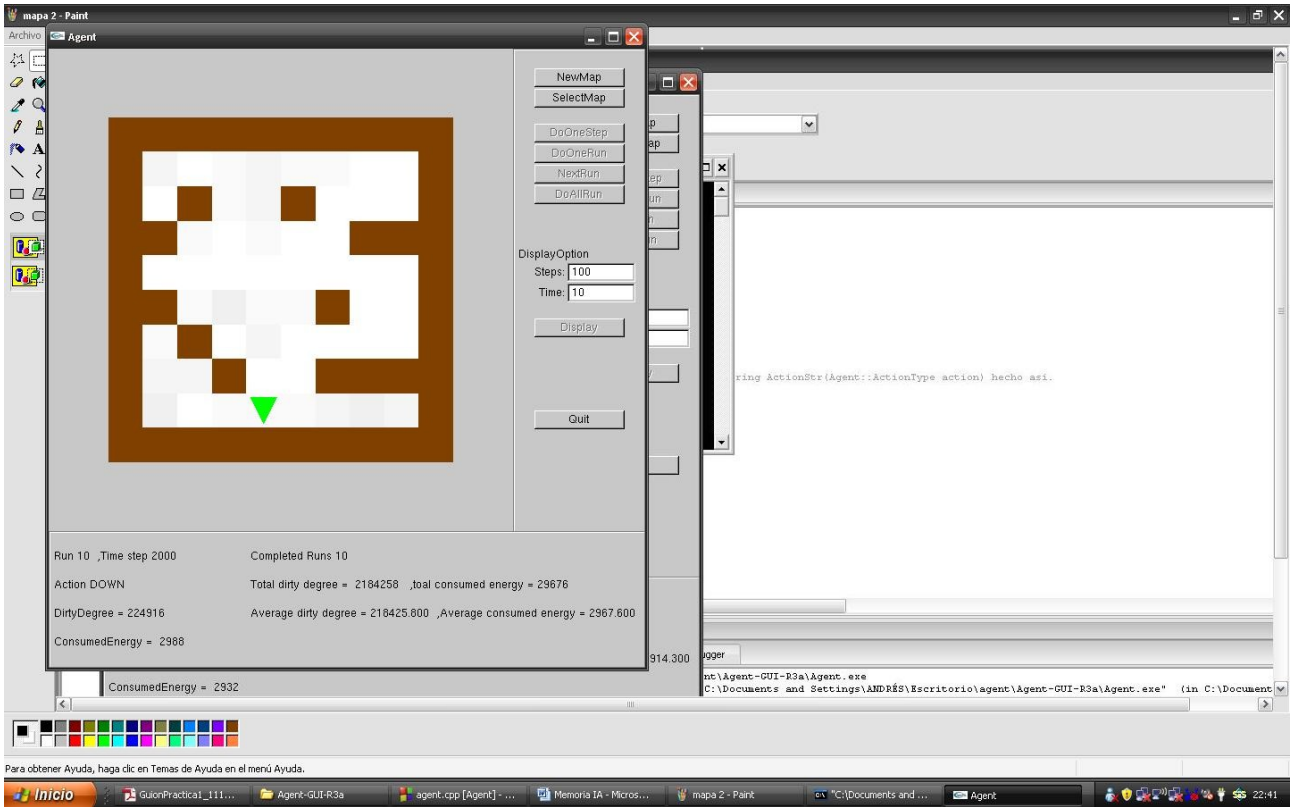
MAP1


agent.cpp [Agent] - Code::Blocks 10.05
Agent
NewMap
SelectMap
DoOneStep
DoOneRun
NextRun
DoAllRun
DisplayOption
Steps: 100
Time: 10
Display
Quit
Run 10 ,Time step 2000
Completed Runs 10
Action SUCK
Total dirty degree = 1638730 ,toal consumed energy = 29536
DirtyDegree = 161944
Average dirty degree = 163873.000 ,Average consumed energy = 2953.600
ConsumedEnergy = 2976
WINDOWS-1252
Line 22, Column 33
Insert
Read/Write
default
Inicio
22:35

MAP2


agent.cpp [Agent] - Code::Blocks 10.05
Agent
NewMap
SelectMap
DoOneStep
DoOneRun
NextRun
DoAllRun
DisplayOption
Steps: 100
Time: 10
Display
Quit
Run 10 ,Time step 2000
Completed Runs 10
Action SUCK
Total dirty degree = 1651961 ,toal consumed energy = 29143
DirtyDegree = 169774
Average dirty degree = 165196.100 ,Average consumed energy = 2914.300
ConsumedEnergy = 2932
WINDOWS-1252
Line 22, Column 33
Insert
Read/Write
default
Inicio
22:41

MAP3


mapa 2 - Paint
Archivo
Agent
NewMap
SelectMap
DoOneStep
DoOneRun
NextRun
DoAllRun
DisplayOption
Steps: 100
Time: 10
Display
Quit
Run 10 ,Time step 2000
Completed Runs 10
Action DOWN
Total dirty degree = 2184258 ,toal consumed energy = 29676
DirtyDegree = 224916
Average dirty degree = 218425.800 ,Average consumed energy = 2967.600
ConsumedEnergy = 2988
ConsumedEnergy = 2932
Para obtener Ayuda, haga clic en Temas de Ayuda en el menú Ayuda.
Inicio
22:41