

Memoria
Práctica 2
Agentes reactivos

Inteligencia Artificial

Mario Ruiz Calvo (2º D)

Análisis del problema.

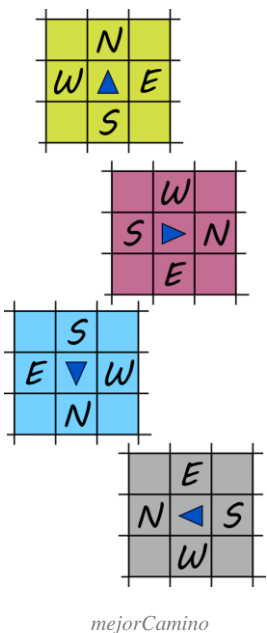
Se pide diseñar un agente reactivo que simule el comportamiento de una aspiradora inteligente situado en un mundo cuadrado en el que no se conocen a priori los diferentes obstáculos que se puede encontrar ni la posición ni orientación desde la que parte el robot. Se deberá conseguir además que el consumo de energía de la aspiradora sea mínimo y el mundo quede lo más limpio posible.

Descripción de la solución planteada.

Variables

Las variables que se han empleado en el desarrollo de la solución propuesta son las siguientes:

- mapa: Matriz de enteros que indica en cada posición si hay un obstáculo (con un valor 0), y si no lo hay, almacena un valor de potencial de componentes atractivos.
Como no sabemos la posición de la que parte el robot la matriz tendrá una dimensión de 20x20 para asegurarnos de que tendremos espacio suficiente para ir almacenando los datos necesarios.
- x,y: Variables enteras que representan las coordenadas de la posición del robot en el mapa en cada momento. Dadas las dimensiones de la matriz y por el mismo motivo, las inicializaremos en la coordenada (9,9).
- NORTH, EAST, SOUTH, WEST: Constantes utilizadas para representar la orientación del robot respecto al mapa y para indicar el mejor camino a tomar respecto al robot.
- orientation: Variable entera que puede tomar los valores *NORTH*, *EAST*, *SOUTH* o *WEST* e indica la orientación del robot respecto al mapa. Inicialmente supondremos que el robot tiene orientación norte.
- mejorCamino: Variable entera que indica cual es el mejor camino que puede tomar el robot en cada momento. Puede tomar valores *NORTH*, *EAST*, *SOUTH* o *WEST* dependiendo de si el robot debe avanzar, girar a la derecha, retroceder o girar a la izquierda para dirigirse al mejor camino en el siguiente paso. Es decir, que no toma los valores en el mismo sentido que *orientation*, sino que dependerá de la orientación del robot en cada momento (Ver imagen ←).
- dobleGiroL: Variable booleana utilizada para hacer el doble giro necesario para orientarse hacia el sur en la búsqueda del mejor camino. Tiene el valor *true* si es necesario hacer el giro en el siguiente paso.
- accAnterior: Variable de tipo *AccionType* que indica la acción que se ha tomado en el paso anterior y se utilizará para actualizar la posición y la orientación del robot.



Análisis del diseño

- Descripción de la función de potencial

Para la realización de esta práctica se han tenido en cuenta los ejercicios 5 y 6 de la relación de agentes reactivos en los que se hace uso de una función de potencial artificial para guiar al robot a través del mundo cuadrulado en el que se encuentra. En estos ejercicios la función de potencial consta de una componente atractiva que lo atrae al objetivo indicado y una componente repulsiva que lo alejará de los obstáculos lo que hará que el robot encuentre el mejor camino sin encontrar ningún obstáculo a su paso.

En nuestro caso, como no conocemos a priori los obstáculos que el robot se puede encontrar, partiremos de la idea de que las casillas por las que hace más tiempo que no hemos pasado tendrán una mayor probabilidad de que tengan más suciedad que por las que acabamos de pasar. Por tanto, utilizaremos una función de potencial que atraerá al robot hacia las casillas en las que hace más tiempo que no ha pasado y será repelido por las casillas en las que ha pasado recientemente. Además, conforme nos vamos encontrando nuevos obstáculos los añadimos a la función de potencial para que también sean repelidos. Todo esto lo conseguimos mediante la variable *mapa* y la función *ActualizarFuncionPotencia*.

La variable *mapa* se inicializará a 1 en cada posición. A partir de ahí, en cada paso se utilizará la función *ActualizarFuncionPotencial* que incrementará en 1 el valor de cada posición (excepto en la última posición por la que ha pasado, en la que volverá a tomar el valor de inicialización). De esta forma, el robot se verá atraído hacia las casillas con valores más altos (que serán las casillas en las que hace más tiempo que no ha pasado). Además, cada vez que el robot encuentre un obstáculo almacenará en la posición adecuada (según la orientación) de la variable *mapa* un valor 0 (mediante la función *ActualizarMapa*), con lo que servirá para distinguir los obstáculos en *ActualizarFuncionPotencial* y que no se incrementen (se elige un valor bajo como el 0 para representar los obstáculos en la variable *mapa* para que el robot nunca se vea atraído hacia esas casillas).

- Reglas de producción

Se han empleado un total de seis reglas para el desarrollo de la práctica:

```
//Regla 1
if (dirty_){
    accion=actSUCK;
    cout<<"Regla 1"<<endl;
}

//Regla 2
else if(dobleGiroL){
    accion=actTURN_L;
    dobleGiroL=false;
    cout<<"Regla 2"<<endl;
}

//Regla 3
else if(!siguienteOcup && mejorCamino==NORTH){
    accion=actFORWARD;
    cout<<"Regla 3"<<endl;
}

//Regla 4
else if(mejorCamino==EAST){
    accion=actTURN_R;
    cout<<"Regla 4"<<endl;
}

//Regla 5
else if(mejorCamino==SOUTH){
    accion=actTURN_L;
    dobleGiroL=true;
    cout<<"Regla 5"<<endl;
}

//Regla 6
else if(mejorCamino==WEST){
    accion=actTURN_L;
    cout<<"Regla 6"<<endl;
}
```

En primer lugar se da prioridad a la acción de limpiar una casilla si el sensor detecta que la casilla actual está sucia, de manera que no se pueda avanzar a la siguiente sin que la actual esté totalmente limpia.

En segundo lugar debe darse prioridad a la acción de hacer el doble giro. Para ello la variable *dobleGiroL* estará a true. Entonces se hará un giro a la izquierda (el segundo) y se pondrá de nuevo a false la variable *dobleGiroL* para indicar que ya se ha hecho el giro.

Las siguientes reglas son relativas a la decisión del agente sobre la dirección a tomar. Si el mejor camino se encuentra hacia el norte (respecto al robot) el robot avanzará, si el mejor camino se encuentra al oeste o el este hará un giro a la izquierda o a la derecha respectivamente (reglas 6 y 4) y si el mejor camino se encuentra hacia el sur hará un giro a la izquierda e indicará que se debe realizar un doble giro para orientarse hacia el sur (regla 5 y posteriormente regla 2).

Hay que señalar que en la regla 3 siempre se podrá realizar el avance. Cuando se encuentra un obstáculo por primera vez se actualiza la variable mapa dándole un 0 a esa posición de modo que la siguiente vez que se pase por esa casilla el robot será repelido porque verá un valor bajo y *mejorCamino* no podrá ser norte en esa posición.

En el caso de las reglas 6 y 4 el robot solo necesita hacer el giro que corresponda, pues en el siguiente paso, al seguir en la misma posición, la casilla que tiene el mejor camino seguirá siendo la misma aunque esta vez orientada hacia el norte (respecto al robot) con lo que intentara avanzar (si puede).

De esta manera el robot siempre elige en cada paso la dirección más apropiada.

- Función actualizar

Para que el robot pueda tomar la decisión adecuada en cada paso es necesario conocer de antemano que ha ocurrido en el paso anterior, actualizando todas las variables antes de tomar la siguiente decisión mediante el sistema de reglas. Esto lo hace la función *Actualizar* y otra serie de funciones asociadas en las que se divide para una mayor claridad.

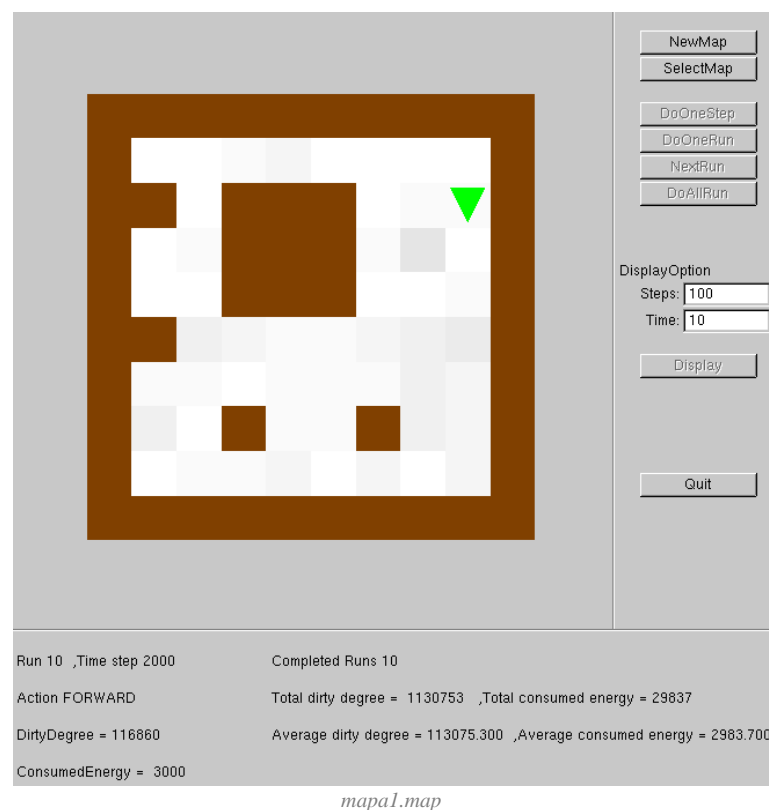
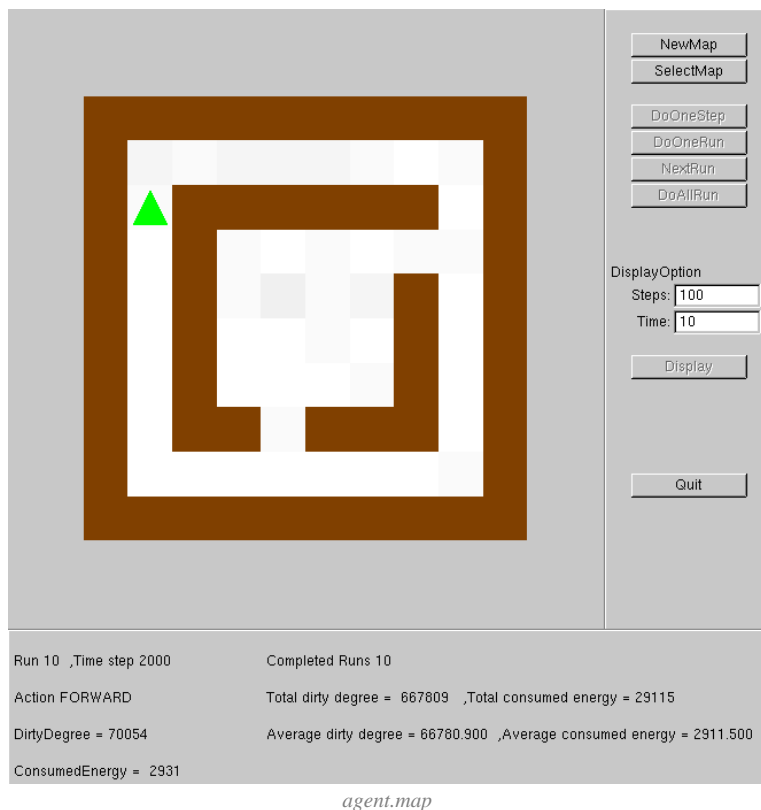
En primer lugar es necesario saber que acción se ha realizado en el paso previo (*accAnterior=acción*) para poder actualizar correctamente las demás variables. A continuación se actualiza la orientación del robot respecto al mapa que solo cambiará cuando en el paso anterior se ha hecho un giro a la izquierda o a la derecha. Después se actualizan las coordenadas del robot en nuestra variable mapa de acuerdo a la orientación (*orientation*) del robot respecto al mapa y solo si en el paso previo se ha avanzado (*actFORWARD*) y no se ha chocado.

Si el robot ha chocado se pone un 0 en la posición adecuada del mapa con la ayuda de *orientation*. A continuación se actualiza la función de potencial (que ya se ha explicado) y por último la variable *mejorCamino*.

La variable *mejorCamino* se actualiza en *ActualizarMejorCamino* y será el máximo de los valores de las casillas que se encuentran al norte, este, sur u oeste del robot con lo que también dependerá de la variable *orientation*. Si hay más de un camino posible la prioridad será norte, sur, oeste y este.

Resultados obtenidos por la solución aportada en los distintos mapas.

El agente se ha probado en los mapas *agent.map*, *mapa1.map*, *mapa2.map* y *mapa3.map*, además de en *maze1.map* del ejercicio 5 y en un mapa propio. En todos ellos se ha alcanzado una solución bastante buena y se ha podido observar que, tras cierto tiempo, el robot conoce el mapa por completo. Esto es así porque el robot se ve atraído por los obstáculos que no conoce ya que al pasar un tiempo esas casillas tendrán valores muy altos. Y una vez que conoce el mapa completo es capaz de establecer un camino óptimo en el que repite solo las casillas que son estrictamente necesarias (no permitiendo así que se acumule demasiada suciedad) y que realiza una y otra vez hasta terminar cada ejecución.



NewMap

SelectMap

DoOneStep

DoOneRun

NextRun

DoAllRun

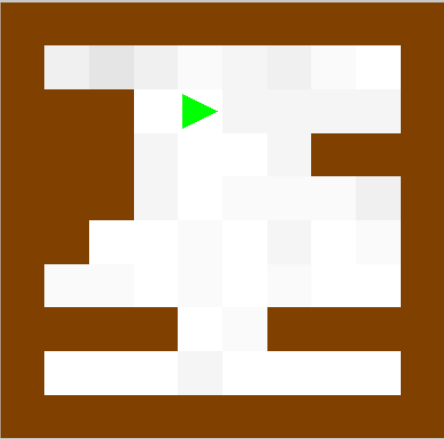
DisplayOption

Steps: 100

Time: 10

Display

Quit



Run 10 ,Time step 2000Completed Runs 10

Action TURN RIGHTTotal dirty degree = 856877 ,Total consumed energy = 29575

DirtyDegree = 87131Average dirty degree = 85687.700 ,Average consumed energy = 2957.500

ConsumedEnergy = 2973

mapa2.map

NewMap

SelectMap

DoOneStep

DoOneRun

NextRun

DoAllRun

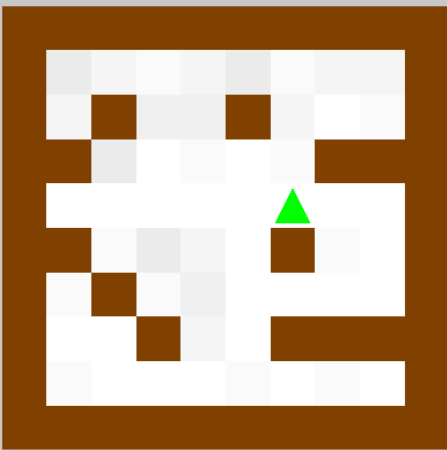
DisplayOption

Steps: 100

Time: 10

Display

Quit



Run 10 ,Time step 2000Completed Runs 10

Action TURN RIGHTTotal dirty degree = 1245194 ,Total consumed energy = 29981

DirtyDegree = 126596Average dirty degree = 124519.400 ,Average consumed energy = 2998.100

ConsumedEnergy = 3020

mapa3.map

NewMap

SelectMap

DoOneStep

DoOneRun

NextRun

DoAllRun

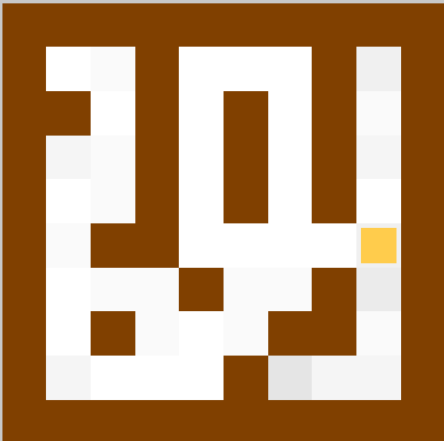
DisplayOption

Steps: 100

Time: 10

Display

Quit



Run 10 ,Time step 2000Completed Runs 10

Action SUCKTotal dirty degree = 761692 ,Total consumed energy = 28665

DirtyDegree = 81238Average dirty degree = 76169.200 ,Average consumed energy = 2866.500

ConsumedEnergy = 2884

maze1.map

NewMap

SelectMap

DoOneStep

DoOneRun

NextRun

DoAllRun

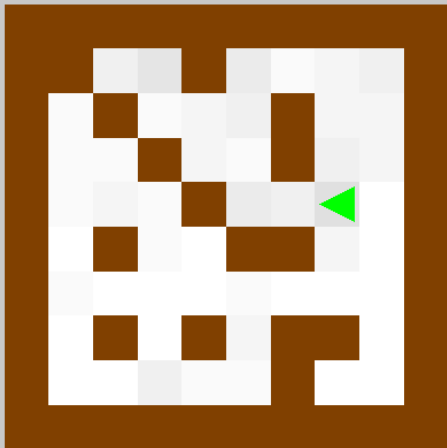
DisplayOption

Steps: 100

Time: 70

Display

Quit



Run 10 ,Time step 2000Completed Runs 10

Action FORWARDTotal dirty degree = 1064464 ,Total consumed energy = 29453

DirtyDegree = 115445Average dirty degree = 106446.400 ,Average consumed energy = 2945.300

ConsumedEnergy = 2954

propio.map