23-4-2014

PRÁCTICA 2 – ASPIRADORA INTELIGENTE

INTELIGENCIA ARTIFICIAL

Francisco Javier Bolívar Lupiáñez

2º GRADO EN INGENIERÍA INFORMÁTICA – GRUPO B1

Contenido

Análisis del problema

Análisis del entorno y dificultades

Se pretende crear un agente reactivo que simule la acción de una **aspiradora inteligente**. Esta aspiradora se encontrará en un mundo cuadriculado dentro de una habitación bien limitada de tamaño 10x10 con obstáculos dentro. La aspiradora deberá moverse por ese entorno y recoger la basura que se encuentre.

La suciedad se encontrará en distintos niveles y la aspiradora no podrá detectarla hasta que este sobre ella. Por lo que si estás en una esquina de la habitación y hay mucha suciedad en la otra, la aspiradora no lo podrá saber y hasta que no se mueva hasta allí no se dará cuenta. Además podrá encontrarse cualquier tipo de obstáculo como pasillos estrechos sin salida.

Sensores del agente

El agente contará con dos sensores:

- **`bump_`**: Devuelve `true` si el agente ha chocado intentando hacer el movimiento anterior, `false` en caso contrario.
- **`dirty_`**: Devuelve `true` si hay suciedad, `false` en caso contrario.

Actuadores del agente

El agente contará con cinco actuadores:

- **`actFORWARD`**: Acción de moverse a la casilla de delante.
- **`actTURN_L`**: Acción de girar 90º hacia la izquierda sin avanzar ninguna casilla.
- **`actTURN_R`**: Acción de girar 90º hacia la derecha sin avanzar ninguna casilla.
- **`actSUCK`**: Acción de limpiar una unidad de suciedad de la casilla actual.
- **`actIDLE`**: No produce ninguna acción. El agente permanece inmóvil.

Estructura interna del agente

El agente dispondrá de dos métodos:

- **`void Perceive(const Environment &env)`**: Este método se utiliza para poder percibir el entorno. Como entrada tiene una variable de tipo `Environment` con información sobre el mundo del agente. El cometido de este método es el de asignar valores a las variables `bump_` y `dirty_` según la información leída por los sensores.
- **`ActionType Think()`**: Este método contiene la función de selección de acciones del agente que se detallará más adelante.

El agente dispondrá de las siguientes variables enteras:

- **`last_turn_`**: 0 si su última acción no fue un giro, 1 si fue un giro a la izquierda, 2 si lo fue hacia la derecha.
- **`orientacion_`**: 0 si está orientado hacia el norte, 1 si lo está hacia el este, 2 si lo está hacia el sur y 3 si lo está hacia el oeste.
- **`x_`**: Posición actual del robot en el eje de coordenadas.
- **`y_`**: Posición actual del robot en el eje de ordenadas.

Y de una estructura de datos:

- **`M_`**: matriz en la que se guardará información sobre las distintas casillas de la habitación.

Diseño de la función de selección de acciones

Se ha usado un **sistema de producción** con el siguiente esquema general:

```
ACTUALIZAR TIEMPO SIN VISITAR POSICIONES

IF suciedad THEN limpiar
ELSE IF choque THEN
  IF derecha=pared THEN girar_izquierda, giro=1
  ELSE IF izquierda=pared THEN girar_derecha, giro=2
  ELSE
    IF ultima_visita_der<ultima_visita_izq THEN girar_izquierda, giro=1
    ELSE IF ultima_visita_der>ultima_visita_izq THEN girar_derecha, giro=2
    ELSE rand(derecha or izquierda)
ELSE IF giro>0 THEN
  IF delante!=pared THEN avanza, giro=0
  ELSE giro=0
ELSE IF giro=1 THEN girar_izquierda, giro=1
ELSE IF giro=2 THEN girar_derecha, giro=2
ELSE
  IF derecha=izquierda=delante=pared THEN girar_izquierda
  ELSE
    IF ultima_visita_izq<=ultima_visita_der and izquierda!=pared THEN
      IF ultima_visita_izq>=ultima_visita_arriba and arriba!=pared THEN avanza, giro=0
      ELSE girar_izquierda, giro=1
    ELSE IF derecha!=pared THEN
      IF ultima_visita_der>= ultima_visita_arriba and arriba!=pared THEN avanza, giro=0
      ELSE girar_derecha, giro=2
    ELSE avanza, giro=0
```

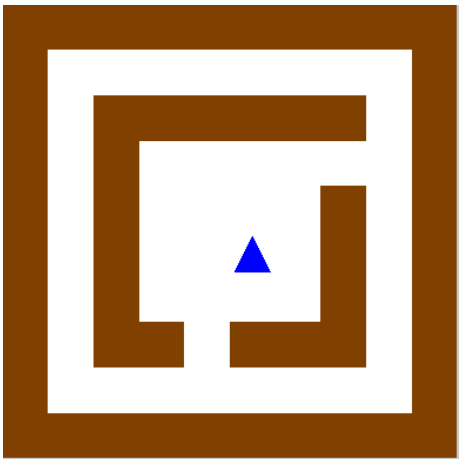
Lógicamente se ha tenido en cuenta, aunque lo he omitido en el esquema para que al echar un vistazo se vea más claramente la idea, la orientación del agente hacia el norte, este, sur y oeste en cada momento y cómo cambia tras las acciones, la posición el registro en la matriz de las posiciones donde hay paredes y el número de visitas que se han hecho a cada casilla.

He usado este sistema porque me parecía la forma más sencilla y esquemática de implementar el comportamiento de este agente, además, con este tipo de función podía describir un **modelo icónico del entorno** del agente y utilizarlo para tomar decisiones de que acción realizar.

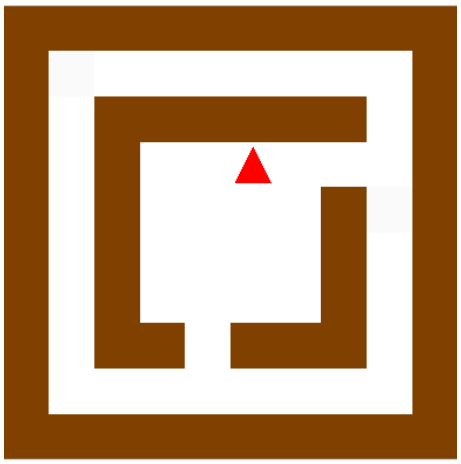
La principal ventaja de este sistema se encuentra en el historial de memoria que se almacena en el modelo icónico del entorno, es decir, conoce que casilla lleva más tiempo sin visitar para poder elegir una u otra cuando se da la ocasión. Además su usabilidad es muy buena pues se pueden añadir nuevas reglas fácilmente sin perjudicar al resto.

Ejemplo de uso

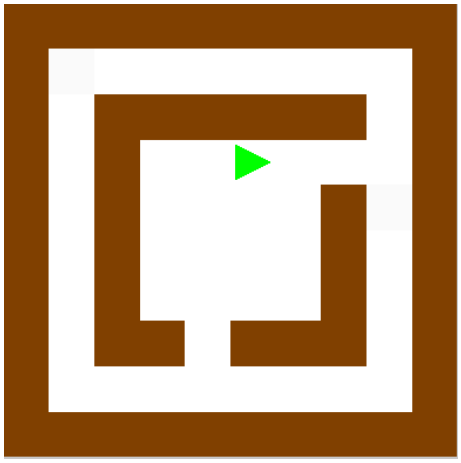
Partiendo desde esta posición:



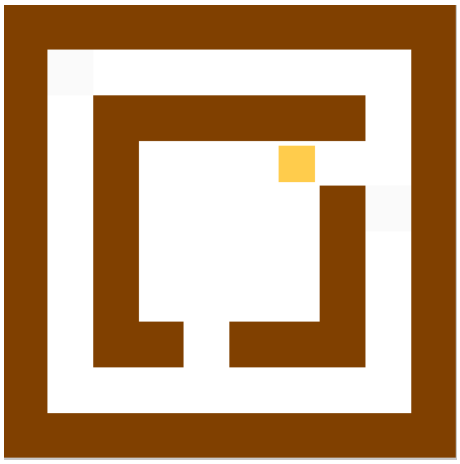
El agente avanzaría hasta chocar pues no hay paredes en su camino ni ha hecho visitas con anterioridad a otras casillas y se prioriza la acción de avanzar sobre la de girar.



Guarda la posición donde se ha chocado. Al no conocer ni la casilla de la derecha, ni la de la izquierda, escogería un movimiento al azar entre la derecha y la izquierda. En este caso, gira a la izquierda.

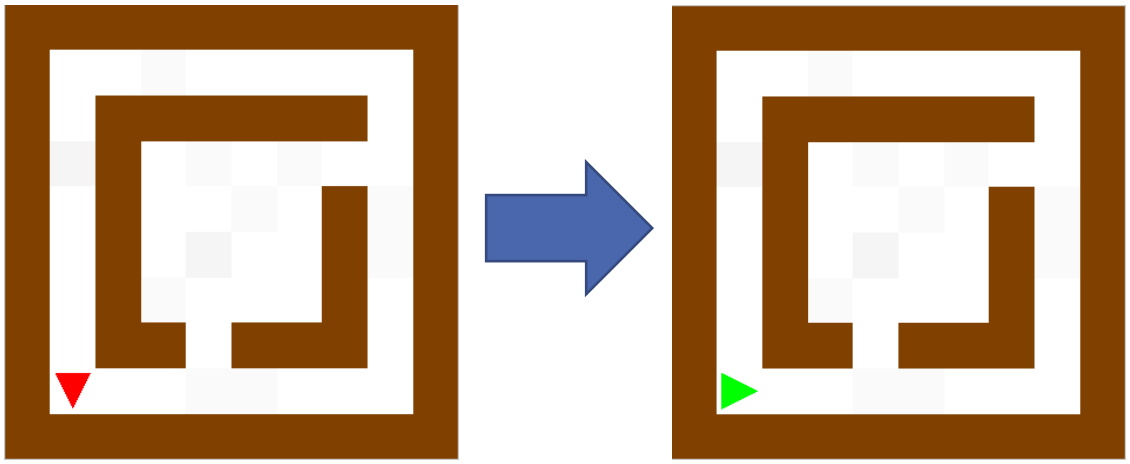


Después del giro, se prioriza avanzar. Y avanzaría hasta encontrar suciedad.

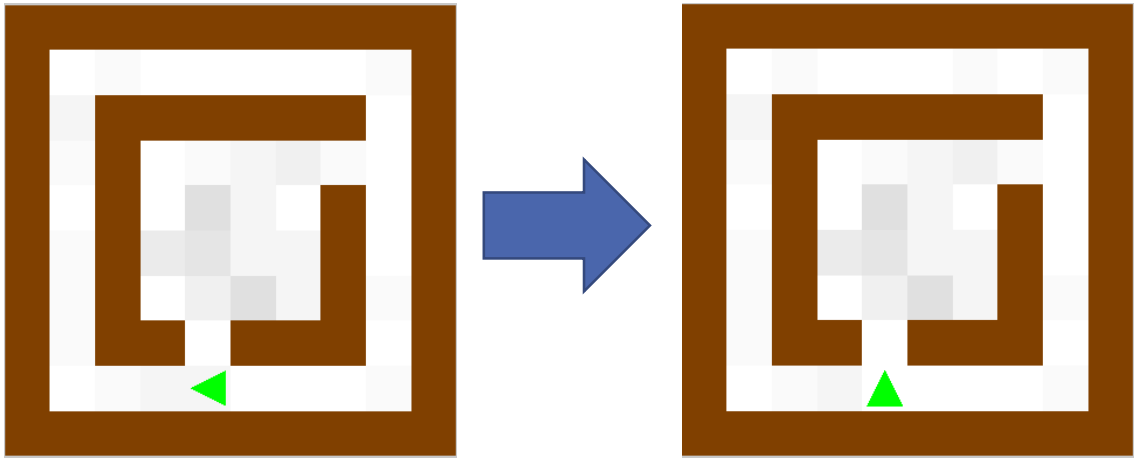


Si hay suciedad, limpia y sigue avanzando ya que no conoce las casillas que hay a la derecha ni a la izquierda.

Seguiría haciendo este comportamiento eternamente. Si se encuentra en una esquina y conoce las paredes que le rodean, directamente escogerá la opción correcta.



Si se encuentra entre dos casillas y una lleva más tiempo sin ser visitada, girará para ir hacia allí:



Soluciones

agent.map

```
Completed Runs 10

Total dirty degree =  678139   ,Total consumed energy = 29114

Average dirty degree = 67813.900  ,Average consumed energy = 2911.400
```

mapa1.map

```
Completed Runs 10

Total dirty degree =  1029634   ,Total consumed energy = 29859

Average dirty degree = 102963.400  ,Average consumed energy = 2985.900
```

mapa2.map

```
Completed Runs 10

Total dirty degree =  963909   ,Total consumed energy = 29217

Average dirty degree = 96390.900  ,Average consumed energy = 2921.700
```

mapa3.map

```
Completed Runs 10

Total dirty degree =  1277494   ,Total consumed energy = 29749

Average dirty degree = 127749.400  ,Average consumed energy = 2974.900
```

Implementación

agent.h

```
#ifndef AGENT__
#define AGENT__

#include <string>
#include <vector>
using namespace std;

// ----------------------------------------------------------
//                              class Agent
// ----------------------------------------------------------
class Environment;
class Agent
{
    public:
        Agent():bump_(false), dirty_(false),last_turn_(0){
            vector<int> aux(20,0);
            vector<vector<int> > m(20,aux);
            M_ = m;
            x_=9;
            y_=9;
            orientacion_=0;
        }

        enum ActionType
        {
            actFORWARD,
            actTURN_L,
            actTURN_R,
            actSUCK,
            actIDLE
        };

        void Perceive(const Environment &env);
        ActionType Think();
    private:
        bool bump_, dirty_;
        int last_turn_, orientacion_, x_, y_;
        vector<vector<int> > M_;
    /**********************************
        MAPA:
        pared=-1

        ORIENTACIÓN:
        norte=0
        este=1
        sur=2
```

```
        oeste=3

        GIRO:
        no_giro=0
        izq=1
        der=2
    ***********************************/

};

string ActionStr(Agent::ActionType);

#endif
```

agent.cpp

```
#include "agent.h"
#include "environment.h"
#include <iostream>
#include <cstdlib>
#include <vector>
#include <utility>

#define NORTE 0
#define ESTE 1
#define SUR 2
#define OESTE 3
#define PARED -1
#define NO_GIRO 0
#define GIRO_IZQ 1
#define GIRO_DER 2

using namespace std;

// ----------------------------------------------------------

Agent::ActionType Agent::Think(){
    ActionType accion;

    for(int i=0; i<20; i++) {
        for(int j=0; j<20; j++) {
            if(M_[i][j]!=PARED) M_[i][j]++;
        }
    }

    M_[x_][y_]=0;

    // 1. SUCIEDAD -> LIMPIAR
    if(dirty_) {
        accion=actSUCK;
        last_turn_=NO_GIRO;

        // Si había chocado, desavanzo
        if(bump_) {
            if(orientacion_==NORTE) x_++;
            if(orientacion_==ESTE) y_--;
            if(orientacion_==SUR) x_--;
            if(orientacion_==OESTE) y_++;
        }
    }
    // 2. NO SUCIEDAD -> MOVERSE
    else {
        // 2.1. CHOCAR -> GIRAR
        if(bump_) {
            // GUARDA POSICIÓN PARED
            M_[x_][y_]=PARED;
            // DESAVANZO
            if(orientacion_==NORTE) x_++;
            if(orientacion_==ESTE) y_--;
            if(orientacion_==SUR) x_--;
            if(orientacion_==OESTE) y_++;
            M_[x_][y_]=0;
            // BUSCA POSICIÓN A LA QUE MOVERSE
            // Si estoy mirando al norte
            if(orientacion_==NORTE) {
```

```
                if(M_[x_][y_+1]==PARED) { // hacia la derecha hay pared
                    accion = actTURN_L;
                    last_turn_ = GIRO_IZQ;
                    orientacion_=OESTE;
                } else if(M_[x_][y_-1]==PARED) { // hacia la izquierda hay pared
                    accion = actTURN_R;
                    last_turn_ = GIRO_DER;
                    orientacion_=ESTE;
                } else { // no hay pared hacia ningún lado
                    if(M_[x_][y_+1]<M_[x_][y_-1]) { // derecha preferible
                        accion = actTURN_L;
                        last_turn_ = GIRO_IZQ;
                        orientacion_=OESTE;
                    } else if(M_[x_][y_+1]>M_[x_][y_-1]) { // izquierda preferible
                        accion = actTURN_R;
                        last_turn_ = GIRO_DER;
                        orientacion_=ESTE;
                    } else {
                        switch(rand()%2+1) {
                            case 1: accion = actTURN_L;
                                last_turn_=GIRO_IZQ;
                                orientacion_=OESTE;
                                break;
                            case 2: accion = actTURN_R;
                                last_turn_=GIRO_DER;
                                orientacion_=ESTE;
                        }
                    }
                }
            }
            // Si estoy mirando al este
            else if(orientacion_==ESTE) {
                if(M_[x_+1][y_]==PARED) { // hacia la derecha hay pared
                    accion = actTURN_L;
                    last_turn_ = GIRO_IZQ;
                    orientacion_=NORTE;
                } else if(M_[x_-1][y_]==PARED) { // hacia la izquierda hay pared
                    accion = actTURN_R;
                    last_turn_ = GIRO_DER;
                    orientacion_=SUR;
                } else { // no hay pared hacia ningún lado
                    if(M_[x_+1][y_]<M_[x_-1][y_]) { // derecha preferible
                        accion = actTURN_L;
                        last_turn_ = GIRO_IZQ;
                        orientacion_=NORTE;
                    } else if(M_[x_+1][y_]>M_[x_-1][y_]) { // izquierda preferible
                        accion = actTURN_R;
                        last_turn_ = GIRO_DER;
                        orientacion_=SUR;
                    } else {
                        switch(rand()%2+1) {
                            case 1: accion = actTURN_L;
                                last_turn_=GIRO_IZQ;
                                orientacion_=NORTE;
                                break;
                            case 2: accion = actTURN_R;
                                last_turn_=GIRO_DER;
                                orientacion_=SUR;
                        }
                    }
                }
            }
            // Si estoy mirando al sur
            else if(orientacion_==SUR) {
                if(M_[x_][y_-1]==PARED) { // hacia la derecha hay pared
                    accion = actTURN_L;
                    last_turn_ = GIRO_IZQ;
                    orientacion_=ESTE;
                } else if(M_[x_][y_+1]==PARED) { // hacia la izquierda hay pared
                    accion = actTURN_R;
                    last_turn_ = GIRO_DER;
                    orientacion_=OESTE;
                } else { // no hay pared hacia ningún lado
                    if(M_[x_][y_-1]<M_[x_][y_+1]) { // derecha preferible
                        accion = actTURN_L;
                        last_turn_ = GIRO_IZQ;
                        orientacion_=ESTE;
```

```
                    } else if(M_[x_][y_-1]>M_[x_][y_+1]) { // izquierda preferible
                        accion = actTURN_R;
                        last_turn_ = GIRO_DER;
                        orientacion_=OESTE;
                    } else {
                        switch(rand()%2+1) {
                            case 1: accion = actTURN_L;
                                last_turn_=GIRO_IZQ;
                                orientacion_=ESTE;
                                break;
                            case 2: accion = actTURN_R;
                                last_turn_=GIRO_DER;
                                orientacion_=OESTE;
                        }
                    }
                }
            }
            // Si estoy mirando al oeste
            else {
                if(M_[x_-1][y_]==PARED) { // hacia la derecha hay pared
                    accion = actTURN_L;
                    last_turn_ = GIRO_IZQ;
                    orientacion_=SUR;
                } else if(M_[x_+1][y_]==PARED) { // hacia la izquierda hay pared
                    accion = actTURN_R;
                    last_turn_ = GIRO_DER;
                    orientacion_=NORTE;
                } else { // no hay pared hacia ningún lado
                    if(M_[x_-1][y_]<M_[x_+1][y_]) { // derecha preferible
                        accion = actTURN_L;
                        last_turn_ = GIRO_IZQ;
                        orientacion_=SUR;
                    } else if(M_[x_-1][y_]>M_[x_+1][y_]) { // izquierda preferible
                        accion = actTURN_R;
                        last_turn_ = GIRO_DER;
                        orientacion_=NORTE;
                    } else {
                        switch(rand()%2+1) {
                            case 1: accion = actTURN_L;
                                last_turn_=GIRO_IZQ;
                                orientacion_=SUR;
                                break;
                            case 2: accion = actTURN_R;
                                last_turn_=GIRO_DER;
                                orientacion_=NORTE;
                        }
                    }
                }
            }
        }

        // 2.2. TRAS GIRAR -> INTENTO AVANZAR
        else if(last_turn_>0) {
            if(orientacion_==NORTE) {
                if(M_[x_-1][y_]!=PARED) {
                    accion=actFORWARD;
                    last_turn_=NO_GIRO;
                    x_--;
                } else {
                    last_turn_=NO_GIRO;
                    accion=actIDLE;
                }
            } else if(orientacion_==ESTE) {
                if(M_[x_][y_+1]!=PARED) {
                    accion=actFORWARD;
                    last_turn_=NO_GIRO;
                    y_++;
                } else {
                    last_turn_=NO_GIRO;
                    accion=actIDLE;
                }
            } else if(orientacion_==SUR) {
                if(M_[x_+1][y_]!=PARED) {
                    accion=actFORWARD;
                    last_turn_=NO_GIRO;
                    x_++;
                } else {
```

```
                    last_turn_=NO_GIRO;
                    accion=actIDLE;
                }
            } else {
                if(M_[x_][y_-1]!=PARED) {
                    accion=actFORWARD;
                    last_turn_=NO_GIRO;
                    y_--;
                } else {
                    last_turn_=NO_GIRO;
                    accion=actIDLE;
                }
            }
        }

        // 2.3. ÚLTIMO GIRO IZQ -> GIRAR IZQ
        else if(last_turn_==GIRO_IZQ) {
            accion = actTURN_L;
            last_turn_=GIRO_IZQ;
            if(orientacion_==NORTE) {
                orientacion_=OESTE;
            } else if(orientacion_==ESTE) {
                orientacion_=NORTE;
            } else if(orientacion_==SUR) {
                orientacion_=ESTE;
            } else {
                orientacion_=SUR;
            }
        }

        // 2.4. ÚLTIMO GIRO DER -> GIRAR DER
        else if(last_turn_==GIRO_DER) {
            accion = actTURN_R;
            last_turn_=GIRO_DER;
            if(orientacion_==NORTE) {
                orientacion_=ESTE;
            } else if(orientacion_==ESTE) {
                orientacion_=SUR;
            } else if(orientacion_==SUR) {
                orientacion_=OESTE;
            } else {
                orientacion_=NORTE;
            }
        }

        // 2.5. NO VIENE DE GIRAR
        else {
            // Si estoy mirando al norte
            if(orientacion_==NORTE) {
                int valorARR=M_[x_-1][y_];
                int valorDER=M_[x_][y_+1];
                int valorIZQ=M_[x_][y_-1];

                if(valorDER==PARED and valorIZQ==PARED and valorARR==PARED) { //
Camino sin salida -> doy la vuelta
                    accion = actTURN_L;
                    last_turn_=GIRO_IZQ;
                    orientacion_=OESTE;
                } else {
                    if(valorIZQ>=valorDER and valorIZQ!=PARED) { // valor izq
preferible
                        if(valorIZQ<=valorARR and valorARR!=PARED) { // ARRIBA
                            accion = actFORWARD;
                            last_turn_=NO_GIRO;
                            x_--;
                        } else { // IZQUIERDA
                            accion = actTURN_L;
                            last_turn_=GIRO_IZQ;
                            orientacion_=OESTE;
                        }
                    } else if(valorDER!=PARED) { // valor der preferible
                        if(valorDER<=valorARR and valorARR!=PARED) { // ARRIBA
                            accion = actFORWARD;
                            last_turn_=NO_GIRO;
                            x_--;
                        } else { // DERECHA
                            accion = actTURN_R;
```

```
                            last_turn_=GIRO_DER;
                            orientacion_=ESTE;
                        }
                    } else { // paredes a los lados, no arriba -> ARRIBA
                        accion = actFORWARD;
                        last_turn_=NO_GIRO;
                        x_--;
                    }
                }
            }
            // Si estoy mirando al este
            else if(orientacion_==ESTE) {
                int valorARR=M_[x_][y_+1];
                int valorDER=M_[x_+1][y_];
                int valorIZQ=M_[x_-1][y_];

                if(valorDER==PARED and valorIZQ==PARED and valorARR==PARED) { //
Camino sin salida -> doy la vuelta
                    accion = actTURN_L;
                    last_turn_=GIRO_IZQ;
                    orientacion_=NORTE;
                } else {
                    if(valorIZQ>=valorDER and valorIZQ!=PARED) { // valor izq
preferible
                        if(valorIZQ<=valorARR and valorARR!=PARED) { // ARRIBA
                            accion = actFORWARD;
                            last_turn_=NO_GIRO;
                            y_++;
                        } else { // IZQUIERDA
                            accion = actTURN_L;
                            last_turn_=GIRO_IZQ;
                            orientacion_=NORTE;
                        }
                    } else if(valorDER!=PARED) { // valor der preferible
                        if(valorDER<=valorARR and valorARR!=PARED) { // ARRIBA
                            accion = actFORWARD;
                            last_turn_=NO_GIRO;
                            y_++;
                        } else { // DERECHA
                            accion = actTURN_R;
                            last_turn_=GIRO_DER;
                            orientacion_=SUR;
                        }
                    } else { // paredes a los lados, no arriba -> ARRIBA
                        accion = actFORWARD;
                        last_turn_=NO_GIRO;
                        y_++;
                    }
                }
            }
            // Si estoy mirando al sur
            else if(orientacion_==SUR) {
                int valorARR=M_[x_+1][y_];
                int valorDER=M_[x_][y_-1];
                int valorIZQ=M_[x_][y_+1];

                if(valorDER==PARED and valorIZQ==PARED and valorARR==PARED) { //
Camino sin salida -> doy la vuelta
                    accion = actTURN_L;
                    last_turn_=GIRO_IZQ;
                    orientacion_=ESTE;
                } else {
                    if(valorIZQ>=valorDER and valorIZQ!=PARED) { // valor izq
preferible
                        if(valorIZQ<=valorARR and valorARR!=PARED) { // ARRIBA
                            accion = actFORWARD;
                            last_turn_=NO_GIRO;
                            x_++;
                        } else { // IZQUIERDA
                            accion = actTURN_L;
                            last_turn_=GIRO_IZQ;
                            orientacion_=ESTE;
                        }
                    } else if(valorDER!=PARED) { // valor der preferible
                        if(valorDER<=valorARR and valorARR!=PARED) { // ARRIBA
                            accion = actFORWARD;
                            last_turn_=NO_GIRO;
```

```
                            x_++;
                        } else { // DERECHA
                            accion = actTURN_R;
                            last_turn_=GIRO_DER;
                            orientacion_=OESTE;
                        }
                    } else { // paredes a los lados, no arriba -> ARRIBA
                        accion = actFORWARD;
                        last_turn_=NO_GIRO;
                        x_++;
                    }
                }
            }
            // Si estoy mirando al oeste
            else {
                int valorARR=M_[x_][y_-1];
                int valorDER=M_[x_-1][y_];
                int valorIZQ=M_[x_+1][y_];

                if(valorDER==PARED and valorIZQ==PARED and valorARR==PARED) { //
Camino sin salida -> doy la vuelta
                    accion = actTURN_L;
                    last_turn_=GIRO_IZQ;
                    orientacion_=SUR;
                } else {
                    if(valorIZQ>=valorDER and valorIZQ!=PARED) { // valor izq
preferible
                        if(valorIZQ<=valorARR and valorARR!=PARED) { // ARRIBA
                            accion = actFORWARD;
                            last_turn_=NO_GIRO;
                            y_--;
                        } else { // IZQUIERDA
                            accion = actTURN_L;
                            last_turn_=GIRO_IZQ;
                            orientacion_=SUR;
                        }
                    } else if(valorDER!=PARED) { // valor der preferible
                        if(valorDER<=valorARR and valorARR!=PARED) { // ARRIBA
                            accion = actFORWARD;
                            last_turn_=NO_GIRO;
                            y_--;
                        } else { // DERECHA
                            accion = actTURN_R;
                            last_turn_=GIRO_DER;
                            orientacion_=NORTE;
                        }
                    } else { // paredes a los lados, no arriba -> ARRIBA
                        accion = actFORWARD;
                        last_turn_=NO_GIRO;
                        y_--;
                    }
                }
            }
        }
    }

    return accion;
}
// ----------------------------------------------------------
void Agent::Perceive(const Environment &env) {
    bump_ = env.isJustBump();
    dirty_ = env.isCurrentPosDirty();
}
// ----------------------------------------------------------
string ActionStr(Agent::ActionType accion) {
    switch (accion) {
        case Agent::actFORWARD: return "FORWARD";
        case Agent::actTURN_L: return "TURN LEFT";
        case Agent::actTURN_R: return "TURN RIGHT";
        case Agent::actSUCK: return "SUCK";
        case Agent::actIDLE: return "IDLE";
        default: return "???";
    }
}
```