



INSTITUTO POLITECNICO NACIONAL



ESCUELA SUPERIOR DE CÓMPUTO

FUNDAMENTOS DE INTELIGENCIA ARTIFICIAL

Prof. Hernández Cruz Macario

PRÁCTICA 03

4BM1

Corona Reyes Mauricio Dassel

Martinez Méndez Diego

Pacheco Sánchez Rodrigo

Ingeniería en Inteligencia Artificial

Resumen:

Para la elaboración de nuestra práctica desarrollamos un programa que emplea lo previamente realizado en la práctica uno, donde realizamos un programa de agentes reactivos lo cuales eran capaces de “encontrar” ciertas recompensas y dirigirlas a la “base” por medio de funciones de movimientos específicos y movimientos aleatorios, con la diferencia en esta ocasión de que los agentes deben dejar un “rastro” o “migajas” de acuerdo con sus movimientos si es que encontraron alguna recompensa en su camino.

Introducción:

En el presente trabajo desarrollamos un programa empleando el lenguaje de programación JAVA que emplee a agentes reactivos.

Los agentes reactivos son una clase de agentes de inteligencia artificial que basan sus decisiones únicamente en la información disponible en el momento, sin considerar el pasado o el futuro.

Estos agentes reaccionan a los estímulos del entorno y toman medidas en consecuencia. A diferencia de los agentes basados en metas, los agentes reactivos no tienen una visión a largo plazo y se centran en maximizar su rendimiento en el corto plazo.

Utilizan una función de mapeo de entradas a salidas para determinar la mejor acción posible en función de la información sensorial presente en el momento.

Los agentes reactivos son ideales para aplicaciones en tiempo real, como sistemas de control de robots o de tráfico, donde la velocidad de respuesta es crucial.

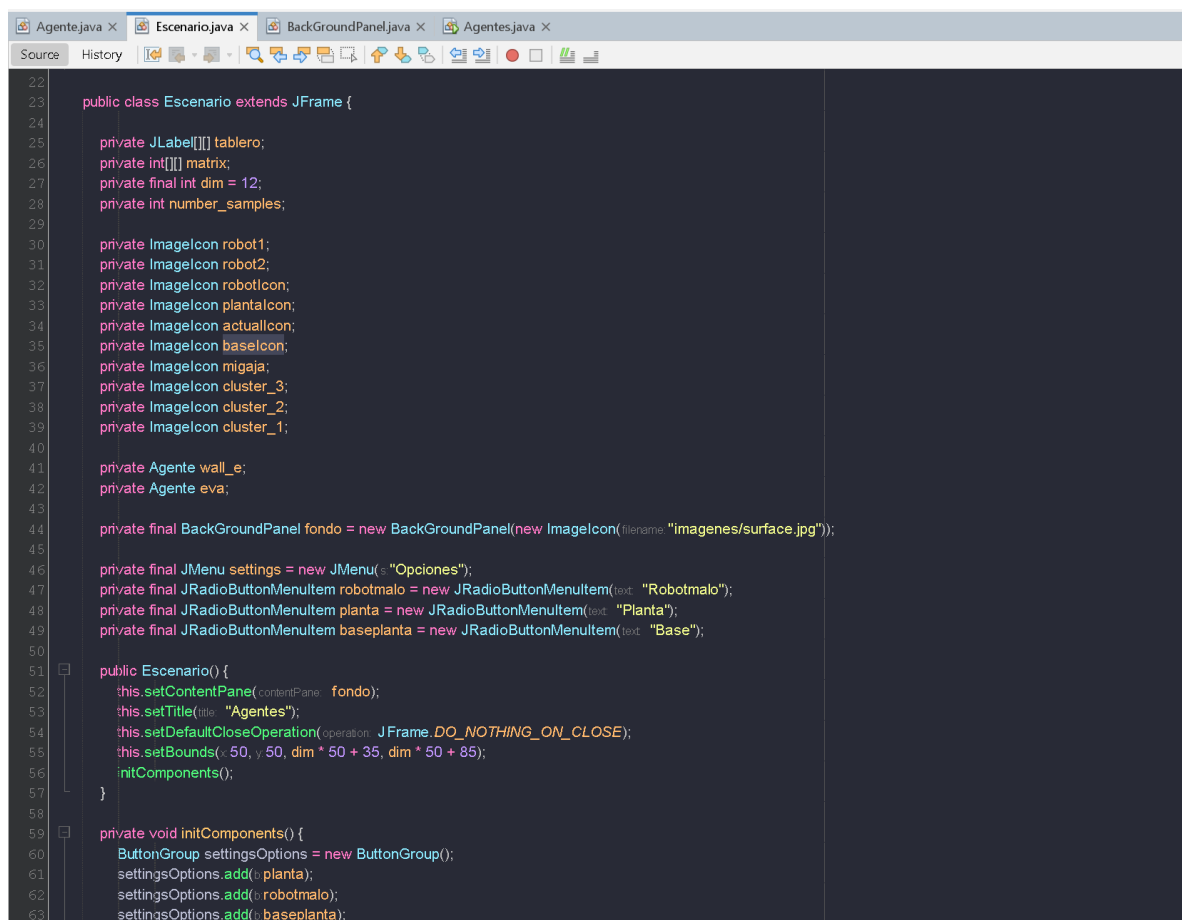
Aunque son más simples y rápidos que otros tipos de agentes, pueden tener dificultades para adaptarse a cambios inesperados en el entorno o para planificar a largo plazo debido a la falta de representación interna del mundo.

Desarrollo:

Comenzando con la descripción de nuestro programa, elegimos el lenguaje de programación JAVA para desarrollar nuestra práctica y utilizamos el entorno Netbeans para poderlo implementar puesto que ofrece diversas herramientas que facilitaron la elaboración de nuestro programa.

Comenzaremos describiendo nuestras dos clases principales, Agentes y Escenario, por el momento omitiremos las clases restantes ya que no aportan directamente información relevante a nuestro programa ya que solo ejecutan tareas sencillas como desplegar el tablero y ejecutar nuestro programa a través de un main.

Comenzando por la clase escenario en la cual especificamos todos los datos necesarios de nuestros agentes, algunos elementos esenciales como las recompensas y la base a la cual nuestros agentes deben dirigirse, así como la asignación de imágenes para cada uno de todos nuestros elementos.



```
22
23
24 public class Escenario extends JFrame {
25
26     private JLabel[][] tablero;
27     private int[][] matrix;
28     private final int dim = 12;
29     private int number_samples;
30
31     private ImageIcon robot1;
32     private ImageIcon robot2;
33     private ImageIcon robotIcon;
34     private ImageIcon plantIcon;
35     private ImageIcon actualIcon;
36     private ImageIcon baseIcon;
37     private ImageIcon migaja;
38     private ImageIcon cluster_3;
39     private ImageIcon cluster_2;
40     private ImageIcon cluster_1;
41
42     private Agente wall_e;
43     private Agente eva;
44
45     private final BackgroundPanel fondo = new BackgroundPanel(new ImageIcon(Iterame "imagenes/surface.jpg"));
46
47     private final JMenu settings = new JMenu("Opciones");
48     private final JRadioButtonMenuItem robotmalo = new JRadioButtonMenuItem(text: "Robotmalo");
49     private final JRadioButtonMenuItem planta = new JRadioButtonMenuItem(text: "Planta");
50     private final JRadioButtonMenuItem baseplanta = new JRadioButtonMenuItem(text: "Base");
51
52     public Escenario() {
53         this.setContentPane(fondo);
54         this.setTitle("Agentes");
55         this.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
56         this.setBounds(x: 50, y: 50, dim * 50 + 35, dim * 50 + 85);
57         initComponents();
58     }
59
60     private void initComponents() {
61         ButtonGroup settingsOptions = new ButtonGroup();
62         settingsOptions.add(plant);
63         settingsOptions.add(robotmalo);
64         settingsOptions.add(baseplanta);
65     }
66 }
```

La función principal de nuestra clase escenario además de ser “initComponents()”, es asignar el objetivo deseado de nuestro programa, lo cual realizamos en la siguiente función.

```
private void formaPlano() {
    tablero = new JLabel[dim][dim];
    matrix = new int[dim][dim];

    for (int i = 0; i < dim; i++) {
        for (int j = 0; j < dim; j++) {
            int row = i;
            int col = j;
            matrix[i][j] = 0;
            tablero[i][j] = new JLabel();
            tablero[i][j].setBounds(j * 50 + 10, i * 50 + 10, width: 50, height: 50);
            tablero[i][j].setBorder(border: BorderFactory.createDashedBorder(Color.black));
            tablero[i][j].setOpaque(false);
            this.add(tablero[i][j]);

            tablero[i][j].addMouseListener(new MouseAdapter() {
                @Override
                public void mousePressed(MouseEvent e) {
                    insertaObjeto(e);
                    if (actualcon == robotcon) {
                        matrix[row][col] = 1;
                    } else if (actualcon == plantacon) {
                        matrix[row][col] = 12;
                        number_samples++;
                        wall_e_objetivo = number_samples * 4;
                        eva_objetivo = number_samples * 4;
                    } else if (actualcon == baselcon) {
                        matrix[row][col] = 2;
                        wall_e.yBase = row;
                        wall_e.xBase = col;

                        eva.yBase = row;
                        eva.xBase = col;
                    }
                }
            });
        }
    }
}

private void gesRobomalo(ItemEvent eventObject) {
    JRadioButtonMenuItem opt = (JRadioButtonMenuItem) eventObject.getSource();
}
```

Las funciones posteriores solamente indican que tipo de icono colocar dentro de nuestro tablero y en la celda seleccionada.

Entrando en detalle de nuestra clase principal la cual es “Agente.java” podemos notar toda la declaración de variables y los elementos necesarios que utilizaremos en esta clase.

```
2
3 import java.util.HashMap;
4 import java.util.Random;
5
6 import javax.swing.ImageIcon;
7 import javax.swing.JFrame;
8 import javax.swing.JLabel;
9 import javax.swing.JOptionPane;
10
11 public class Agente extends Thread {
12
13     String nombre;
14     int i;
15     int j;
16     ImageIcon icon;
17     ImageIcon Base;
18     ImageIcon migaja;
19     HashMap< Integer, ImageIcon> cluster = new HashMap<>();
20     int[][] matrix;
21     JLabel tablero[][];
22     int obtenidas;
23     int objetivo;
24     int entregas;
25     Agente companero;
26     int xBase;
27     int yBase;
28     Boolean irBase = false;
29
30     JLabel casillaAnterior;
31     int fila_ant;
32     int col_ant;
33     Random aleatorio = new Random(System.currentTimeMillis());
34
35     public Agente(String nombre, ImageIcon icon, int[][] matrix, JLabel tablero[], ImageIcon mother, ImageIcon migaja, HashMap< Integer, ImageIcon> cluster) {
36         this.nombre = nombre;
37         this.icon = icon;
38         this.matrix = matrix;
39         this.tablero = tablero;
40
41         this.i = aleatorio.nextInt(matrix.length);
42         this.j = aleatorio.nextInt(matrix.length);
43         tablero[i][j].setIcon(icon);
44     }
45 }
```

Contamos con una función llamada “regresarBase()” la cual va a ser la encargada de dirigir a nuestros agentes hacia la “base” que hemos colocado en el tablero, esto ocurrirá una vez nuestro agente haya encontrado alguna recompensa y lo realizamos simplemente moviendo de manera aleatoria hacia abajo, arriba, derecha o izquierda nuestro agente hasta encontrarse con algún obstáculo que le obligue a cambiar de dirección o hasta llegar con la base.

```
8
9 public void regresarBase() {
10
11     irBase = true;
12
13     while (true) {
14
15         int yDistance = yBase - i;
16         int xDistance = xBase - j;
17
18         if (yDistance == 0 && xDistance == 0) {
19             matrix[i][j] = 2;
20             break;
21         }
22
23         while ((yBase - i) > 0) {
24
25             casillaAnterior = tablero[i][j];
26             if (matrix[i + 1][j] == 0 || matrix[i + 1][j] == 2 || matrix[i + 1][j] == 4) {
27
28                 fila_ant = i;
29                 col_ant = j;
30                 i += 1;
31                 matrix[i][j] = 4;
32                 actualizaPosicion();
33
34                 try {
35                     sleep(100 + aleatorio.nextInt(bound 100));
36                 } catch (Exception ex) {
37                     ex.printStackTrace();
38                 }
39
40             } else {
41                 break;
42             }
43
44         }
45
46         while ((xBase - j) > 0) {
47
48             casillaAnterior = tablero[i][j];
```

Nuestra función “migajas()” nos ayuda a que los agentes sigan el rastro de “migajas” una vez encontrado la recompensa y dirigirse hacia la base, esto con el fin de poder eliminar las imágenes del “rastro” y que nuestro agente siga el camino adecuadamente.

```

162 public void migajas() {
163     matrix[i][j] = 0;
164
165     int last_movement = 0;
166
167     while (true) {
168         casillaAnterior = tablero[i][j];
169         fila_ant = i;
170         col_ant = j;
171
172         if (matrix[i - 1][j] == 4 && i > 0) {
173             i--;
174             matrix[i][j] = 0;
175             actualizarPosicion();
176             last_movement = 1;
177
178             try {
179                 sleep(100 + aleatorio.nextInt(100));
180             } catch (Exception ex) {
181                 ex.printStackTrace();
182             }
183
184         } else if (matrix[i + 1][j] == 4 && i <= matrix.length - 2) {
185             i++;
186             matrix[i][j] = 0;
187             actualizarPosicion();
188             last_movement = 2;
189
190             try {
191                 sleep(100 + aleatorio.nextInt(100));
192             } catch (Exception ex) {
193                 ex.printStackTrace();
194             }
195
196         } else if (matrix[i][j + 1] == 4 && j <= matrix.length - 2) {

```

Finalmente nuestra función principal “run()” nos ayuda a mover nuestro agente por todo el tablero de manera aleatoria hasta encontrar alguna recompensa y poder ejecutar las funciones previamente mencionadas.

```

public void run() {
    int next_move_row = 0;
    int next_move_col = 0;

    while (true) {
        casillaAnterior = tablero[i][j];
        fila_ant = i;
        col_ant = j;

        int flag = 1;
        while (flag == 1) {
            next_move_row = aleatorio.nextInt((1 - (-1)) + 1) + (-1);
            next_move_col = aleatorio.nextInt((1 - (-1)) + 1) + (-1);
            if ((next_move_col == 0 && (next_move_row == 1 || next_move_row == -1)) || (next_move_row == 0 && (next_move_col == 1 || next_move_col == -1))) {
                flag = 0;
            }
        }

        if ((i > matrix.length - 2 || matrix[i + 1][j] == 1) && next_move_row == 1) {
            next_move_row = -1;
            next_move_col = 0;
        }

        if ((i < 1 || matrix[i - 1][j] == 1) && next_move_row == -1) {
            next_move_row = 1;
            next_move_col = 0;
        }

        if ((j > matrix.length - 2 || matrix[i][j + 1] == 1) && next_move_col == 1) {
            next_move_col = -1;
            next_move_row = 0;
        }

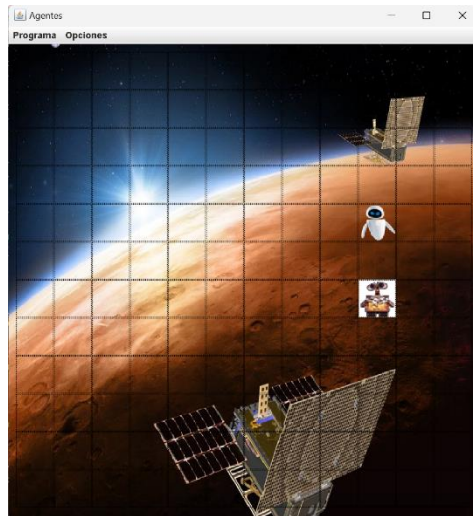
        if ((j < 1 || matrix[i][j - 1] == 1) && next_move_col == -1) {
            next_move_col = 1;
            next_move_row = 0;
        }

        if ((i < matrix.length - 1 && i > 1) && (matrix[i + 1][j] == 1 && matrix[i - 1][j] == 1)) {
            next_move_row = 0;

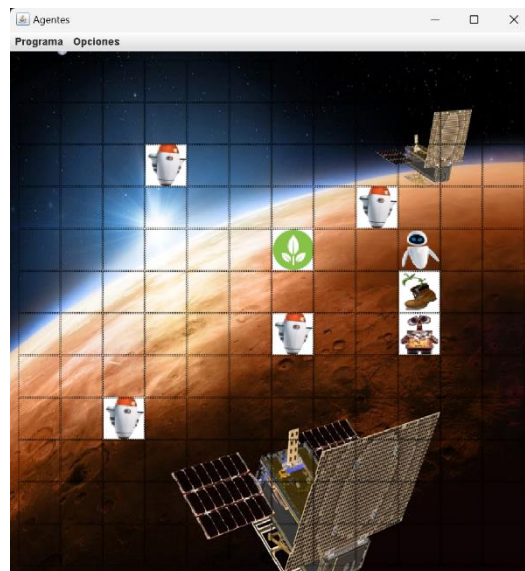
```

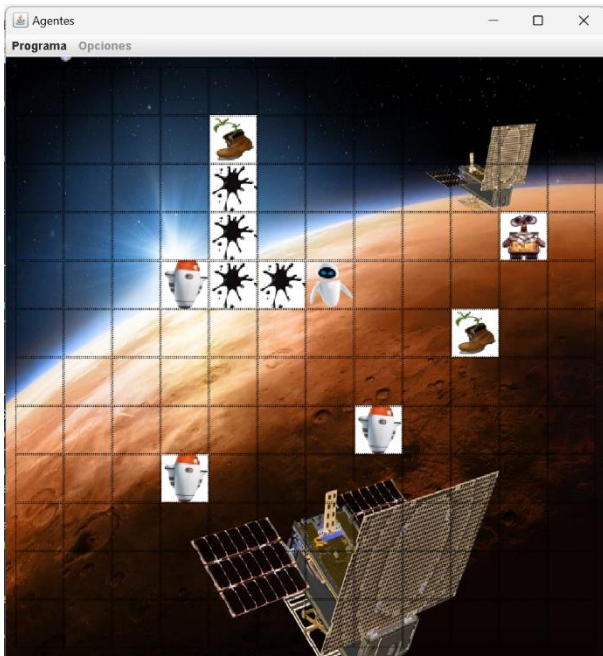
Ejecución del programa:

Primera vez al ejecutar nuestro programa nos aparecerá una situación muy similar a la siguiente, encontrándonos con dos agentes y el tablero vacío. Contamos con un menú en la parte superior izquierda que nos permitirá colocar los elementos que deseemos para probar nuestro programa.

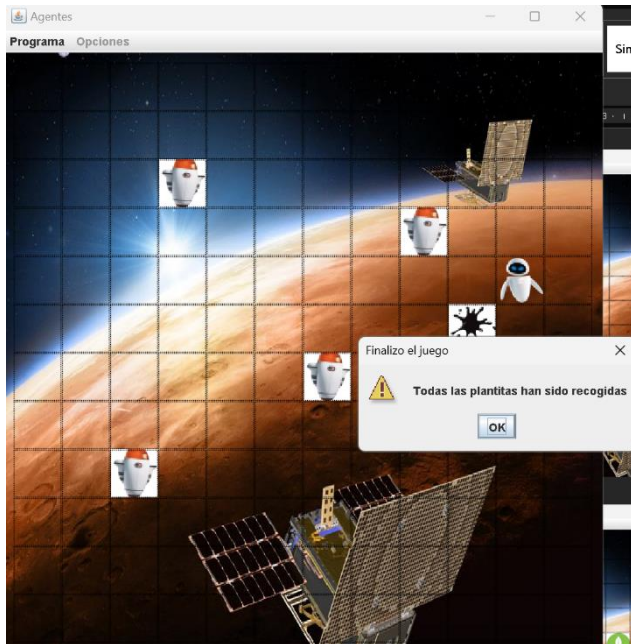


Una vez colocados los elementos que deseamos en nuestro programa procederemos a ejecutarlo y observar su funcionamiento.





Si nuestro agente se encuentra con alguna recompensa, dejará un rastro que marca su recorrido hasta llegar a alguna base cercana.



Cuando nuestros agentes acaben de recoger todas las recompensas en nuestro tablero observaremos un mensaje como el siguiente, indicando que han finalizado su tarea.

Conclusiones:

Los agentes reactivos son una de las formas más sencillas y eficaces de tomar decisiones en tiempo real basadas en información sensorial. Debido a su capacidad para reaccionar rápidamente a los cambios en el entorno, son adecuados para aplicaciones en tiempo real, como sistemas de control de robots, seguridad y tráfico.

Además, los agentes reactivos son menos complejos y más económicos en términos de implementación y costos computacionales en comparación con otros tipos de agentes más complejos. No necesitan una representación interna del mundo ni una visión a largo plazo, lo que hace que su diseño y desarrollo sean más simples.

En entornos predecibles y estables, donde la velocidad de respuesta es crucial y los cambios son mínimos, los agentes reactivos pueden ser especialmente efectivos. Aunque pueden tener dificultades para adaptarse a cambios impredecibles o planificar a largo plazo, siguen siendo una herramienta valiosa en la inteligencia artificial debido a su simplicidad y velocidad de respuesta.

Referencias

Mamani, M. L. (25 de 05 de 2020). *ENCORA*. Obtenido de DFS vs BFS:
<https://www.encora.com/es/blog/dfs-vs-bfs>

Búsqueda primero en profundidad (DFS) frente a búsqueda primero en amplitud (BFS).

(s. f.). <https://www.techiedelight.com/es/depth-first-search-dfs-vs-breadth-first-search-bfs/>