INTELIGENCIA ARTIFICIAL Y SISTEMAS INTELIGENTES

PROYECTO FINAL

2022/2023

Alumnos:

- Ignacio Alcalde Torrescusa
- Darío Álvarez Barrado

1. Resumen del problema de la práctica:

La práctica realizada consistía en diseñar e implementar un robot con la capacidad de explorar laberintos a través de objetos obstaculizantes como muros, con el fin de encontrar la salida. El objetivo principal del robot se basaba en descubrir la vía de escape, después de haber recolectado un conjunto de monedas específicas. Esto debía resolverse mediante técnicas y heurísticas previamente aprendidas en clase de teoría.

2. Descripción de la instalación, interfaz y manejo de la práctica:

Para poder probar esta práctica debemos realizar los siguientes pasos:

1. Instalar eclipse para java en nuestro ordenador.
2. Descargar el archivo zip de nuestro proyecto.
3. Descomprimir el archivo y guardarlo donde se prefiera.
4. Abrir eclipse (Que hemos instalado previamente).
5. Hacer click en “file” (Situado arriba a la izquierda).
6. Seleccionamos “import”.
7. Seleccionamos “Existing Projects into Workspace”.
8. Seleccionamos “Browse…” (Arriba a la derecha).
9. Buscamos donde hayamos guardado el archivo y hacemos doble click en él.
10. Hacemos click en Finish (Abajo).
11. Ya podremos ejecutar el proyecto con la técnica heurística que prefiramos (comentando las demás).

Una vez tenemos abierto el proyecto realizaremos los siguientes pasos:

1. Nos dirigiremos a la línea 42 para cambiar el laberinto con el que se quiera jugar. Lo único que habrá que hacer será cambiar el número: LABECOIN1.txt si se quiere jugar al primer laberinto, LABECOIN2.txt si se quiere jugar al segundo y así sucesivamente.

2. Nos dirigiremos a la línea 1184 y la descomentamos si queremos probar con escalada Máxima Pendiente, a la línea 1185 y la descomentamos si queremos probar con escalada estocástica y a la línea 1186 y la descomentamos si queremos probar con escalada simple.

Los módulos se encuentran en las siguientes líneas:

1. Movimientos: De la línea 106 a la 214.
2. Método del cálculo de la distancia a la salida: línea 216.
3. Método del cálculo de la distancia a la moneda más cerca: línea 229.
4. Método del cálculo de la distancia a la moneda de más valor: línea 251.
5. Método que calcula si una coordenada es un muro: línea 273.
6. Método de máxima pendiente: línea 280.
7. Método de escalada simple: línea 828.
8. Método de escalada simple estocástico: línea 529.
9. Método de Generar vector: línea 507.
10. Método de Actualizar Posición: línea 1134.
11. Main: línea 1181.

3. Descripción de los algoritmos de resolución usados en la solución:

En el proyecto desarrollado, las heurísticas implementadas se estructuran en torno a un mismo criterio basado en dividir el problema en dos partes, a partir de la identificación de dos objetivos bien definidos. En primer lugar, se busca localizar los puntos necesarios que permitan avanzar hacia la salida y, en segundo lugar, se enfoca en alcanzar dicha salida de manera exitosa.

3.1 Algoritmo de Escalada de Máxima Pendiente:

Para la resolución del problema mediante el algoritmo de escalada de máxima pendiente, implementamos un vector de 8 posiciones para almacenar todas las posibles distancias que el robot podría recorrer desde su posición actual. Posteriormente, se selecciona la distancia más corta y se ejecuta el movimiento correspondiente para avanzar hacia el objetivo. Esto se realiza para cada uno de los objetivos.

3.2 Algoritmo de Escalada simple:

Para la resolución del problema mediante el algoritmo de escalada simple generamos un vector de 8 casillas con los 8 posibles movimientos. Cuando hayamos escogido un movimiento que mejore nuestra situación actual nos movemos hasta conseguir nuestro objetivo. Esto se realiza para cada uno de los objetivos.

3.3 Algoritmos de Escalada Simple Estocástico:

Este algoritmo es igual que el simple. Sin embargo para mover no basta con que mejore la situación actual del robot sino que lo debe de hacer mejorando un valor determinado (α) hasta alcanzar el objetivo. Esto se realiza para cada uno de los objetivos.

4 Descripción de las soluciones seguidas para resolver el problema:

4.1 Algoritmo de Escalada de Máxima Pendiente:

Para la resolución del problema mediante el algoritmo de escalada de máxima pendiente, comenzamos por calcular la distancia actual hacia el objetivo y, a continuación, creamos un vector de 8 posiciones para almacenar todas las posibles distancias en función del movimiento que realice el robot desde su ubicación actual. Tras completar el vector, se examinan las distancias para identificar la menor hacia el objetivo. Si no se logra mejorar la distancia actual con ninguno de los movimientos, el algoritmo finaliza, puesto que no es posible mejorar la posición del robot. No obstante, si se identifica una distancia menor a la actual, se ejecuta el movimiento correspondiente para acercarse al objetivo. Esto se realiza para cada uno de los dos objetivos.

El pseudocódigo que muestra de manera más clara esto es el siguiente:

```
MaximaPendiente(){

      mientras Puntos < Precio hacer {

         -Calcular la distancia actual al objetivo.
        -Creamos un vector de 8 posiciones.
         Para i desde 0 hasta 7 hacer {
            seleccionar caso i:
               caso 0(Arriba):
                  Si no es muro:
                    -En la posición del vector “i” ponemos la distancia al objetivo si se mueve a
arriba
                  Si es muro:
                        -Ponemos un valor muy alto en la posición i del vector para que no se
tenga en cuenta
                 break;

      //Cada caso se corresponde con un movimiento y se realiza lo mismo en cada uno de
ellos                                    teniendo en cuenta hacia dónde se movería:

                     caso 1= Arriba Izquierda.
                     caso 2=Arriba derecha.
                     caso 3= Izquierda.
                     caso 4=Derecha.
                     caso 5=Abajo.
                     caso 6=Abajo Izquierda.
                     caso 7=Abajo derecha.


   }//fin del for
```

```
        Para i de 0 hasta 7:
            - Recorremos el vector y seleccionamos la distancia mínima y el movimiento que
debe hacer el robot que se corresponderá con la posición del vector con la distancia
mínima.
        }//fin del for

          Si no hay ningún movimiento que mejore la situación actual termina el programa ya
que no tiene solución.

        Realiza el mejor movimiento posible.

          Si el número de movimientos realizados es superior al establecido en nuestro caso
15 se termina el programa.

}fin del mientras

mientras no sea salida hacer{
     Realizar el mismo while pero esta vez en busca de la salida
}
```

4.2 Algoritmo de Escalada simple:

Para la resolución del problema mediante el algoritmo de escalada simple generamos un vector de 8 casillas con números del 0 al 7 (cada uno de los movimientos posibles) posicionados de forma aleatoria. Esto se realiza cada vez que el robot se mueve y tiene que realizar otro movimiento. Una vez tenemos el vector calculamos la distancia al objetivo desde la posición del robot y se recorre el vector hasta encontrar un movimiento que mejore la situación actual hasta alcanzar el objetivo. Esto se realiza para cada uno de los objetivos.

El pseudocódigo que muestra de manera más clara esto es el siguiente:

```
Simple(){

      mientras Puntos < Precio hacer {

          -Calcular la distancia actual al objetivo.
             -Creamos un vector de 8 posiciones con números del 0 al 7 colocados de forma aleatoria
llamando al módulo “GenerarVector()”.
          Para i desde 0 hasta 7 hacer {
             seleccionar caso i:
                caso  0(Arriba):
                   Si no es muro:
                     -Se calcula la distancia al objetivo si el robot se mueve hacia arriba.
                      Si la posición a la que se va a mover es mejor que la actual o es el objetivo ponemos
en la variable “posición” el movimiento que corresponda (0 en este caso) y finalizamos el “for”.
```

Si es muro:
-Ponemos un valor muy alto y que no corresponde a ningún movimiento en la variable "posición" para que el robot no realice movimiento.
break;

//Cada caso se corresponde con un movimiento y se realiza lo mismo en cada uno de ellos teniendo en cuenta hacia dónde se movería:

caso 1= Arriba Izquierda.
caso 2=Arriba derecha.
caso 3= Izquierda.
caso 4=Derecha.
caso 5=Abajo.
caso 6=Abajo Izquierda.
caso 7=Abajo derecha.

En este caso el for termina cuando escojamos un movimiento que sea mejor a la situación actual

}//fin del for

Realizamos el movimiento seleccionado.

Si el número de movimientos realizados es superior al establecido en nuestro caso 15 se termina el programa.

Si no hay ningún movimiento que mejore la situación actual termina el programa ya que no tiene solución.

}fin del mientras

mientras no sea salida hacer{
Realizar el mismo while pero esta vez en busca de la salida
}

4.3 Algoritmo de Escalada Simple Estocástico:

Este algoritmo es igual que el simple. Sin embargo para mover no basta con que mejore la situación actual del robot sino que lo debe de hacer mejorando un valor determinado (α) hasta alcanzar el objetivo. Esto se realiza para cada uno de los objetivos.

El pseudocódigo que muestra de manera más clara esto es el siguiente:

Estocástico(){

mientras Puntos < Precio hacer {

-Calcular la distancia actual al objetivo.
-Creamos un vector de 8 posiciones con números del 0 al 7 colocados de forma aleatoria llamando al módulo “GenerarVector()”.
Para i desde 0 hasta 7 hacer {
seleccionar caso i:
caso 0(Arriba):
Si no es muro:
-Se calcula la distancia al objetivo si el robot se mueve hacia arriba.
Si la posición a la que se va a mover es mejor que la actual y mejor que α o es el objetivo ponemos en la variable “posición” el movimiento que corresponda (0 en este caso) y finalizamos el “for”.
Si es muro:
-Ponemos un valor muy alto y que no corresponde a ningún movimiento en la variable “posición” para que el robot no realice movimiento.
break;

//Cada caso se corresponde con un movimiento y se realiza lo mismo en cada uno de ellos teniendo en cuenta hacia dónde se movería:

caso 1= Arriba Izquierda.
caso 2=Arriba derecha.
caso 3= Izquierda.
caso 4=Derecha.
caso 5=Abajo.
caso 6=Abajo Izquierda.
caso 7=Abajo derecha.

En este caso el for termina cuando escojamos un movimiento que sea mejor a la situación actual y se supere en un valor α.

}//fin del for

Realizamos el movimiento seleccionado.

Si el número de movimientos realizados es superior al establecido en nuestro caso 15 se termina el programa.

Si no hay ningún movimiento que mejore la situación actual termina el programa ya que no tiene solución.
}fin del mientras

mientras no sea salida hacer{
Realizar el mismo while pero esta vez en busca de la salida
}

5. Resultados obtenidos en las baterías de pruebas para cada una de las soluciones:

Laberinto:	**Técnica de resolución:**	**Solución:**	**Tiempo de ejecución:**	**Nodos generados:**
Laberinto 1	Máxima pendiente	Solución no encontrada.	9 ms	88
Laberinto 2	Máxima pendiente	D, AI, AI, AI, I, I	14 ms	48
Laberinto 3	Máxima pendiente	Alcanzado el número máximo de movimientos que se pueden realizar.	19 ms	120
Laberinto 4	Máxima pendiente	Solución no encontrada	8 ms	40
Laberinto 5	Máxima pendiente	Solución no encontrada	10 ms	48
Laberinto 6	Máxima pendiente	Solución no encontrada	10 ms	48
Laberinto 7	Máxima pendiente	Solución no encontrada	12 ms	64
Laberinto 8	Máxima pendiente	Solución no encontrada	9 ms	32
Laberinto 9	Máxima pendiente	Solución no encontrada.	11 ms	48
Laberinto 10	Máxima pendiente	Solución no encontrada	14 ms	48
Laberinto 1	Simple	Solución no encontrada	18 ms	47
Laberinto 2	Simple	D, AI, A, BI, A, I, I, A, I	11 ms	26
Laberinto 3	Simple	Alcanzado el número máximo de movimientos que se pueden realizar.	14 ms	29

Laberinto 4	Simple	Solución no encontrada	18 ms	34
Laberinto 5	Simple	Solución no encontrada	14 ms	39
Laberinto 6	Simple	Solución no encontrada	15 ms	29
Laberinto 7	Simple	Solución no encontrada	13 ms	21
Laberinto 8	Simple	Solución no encontrada	8 ms	20
Laberinto 9	Simple	Solución no encontrada	11 ms	33
Laberinto 10	Simple	Solución no encontrada	13 ms	23
Laberinto 1	Estocástico con α=1	Solución no encontrada	6 ms	26
Laberinto 2	Estocástico con α=1	D, AI, AI, AI ,I ,I	10 ms	26
Laberinto 2	Estocástico con α=2	Solución no encontrada	7 ms	14
Laberinto 3	Estocástico con α=1	Alcanzado el número máximo de movimientos que se pueden realizar.	16 ms	72
Laberinto 4	Estocástico con α=2	Solución no encontrada	5 ms	8
Laberinto 5	Estocástico con α=1	Solución no encontrada	12 ms	27
Laberinto 6	Estocástico con α=3	Solución no encontrada	4 ms	8
Laberinto 7	Estocástico con α=1	Solución no encontrada	15 ms	42
Laberinto 8	Estocástico con α=4	Solución no encontrada	6 ms	8
Laberinto 9	Estocástico con α=3	Solución no encontrada	5 ms	8
Laberinto 10	Estocástico con α=1	Solución no encontrada	10 ms	38

6. Conclusiones sobre los resultados para cada una de las técnicas de resolución empleadas:

Podemos determinar que el algoritmo de máxima pendiente al tener que expandir todas las posibilidades obtiene un mayor número de nodos generados que las otras dos heurísticas empleadas. Por el contrario el algoritmo estocástico va a ser el que menos nodos expanda y el de menor tiempo de ejecución respecto a las otras heurísticas ya que va a ser difícil superar la situación actual en el valor α. Por último, el algoritmo de escalada simple no tiene un comportamiento establecido debido a su comportamiento aleatorio a la hora de escoger la posibilidad a expandir.

7. Referencias:

- www.lawebdelprogramador.com: De esta página hemos sacado la lectura de una matriz por fichero, la cual hemos adaptado a nuestro proyecto.

- Chat-gpt: De esta inteligencia artificial hemos sacado el módulo de GenerarVectorAleatorio();