TECNOLÓGICO NACIONAL DE MEXICO

INSTITUTO TECNOLÓGICO DE LA LAGUNA

Inteligencia artificial

INGENIERÍA EN SISTEMAS COMPUTACIONALES

DOCENTE: LAMIA HAMDÁN M.

UNIDAD 1 PROYECTO UNIDAD 1

NÚM DE CONTROL	NOMBRE
19130527	Fatima Gorety Garcia Yescas
19130519	Roberto Esquivel Troncoso
19130535	Ivan Herrera Garcia
19130514	Isaias Gerardo Cordova Palomares
19130547	Jesus Rafael Medina Dimas

FECHA DE ENTREGA: 02/10/2022

Índice

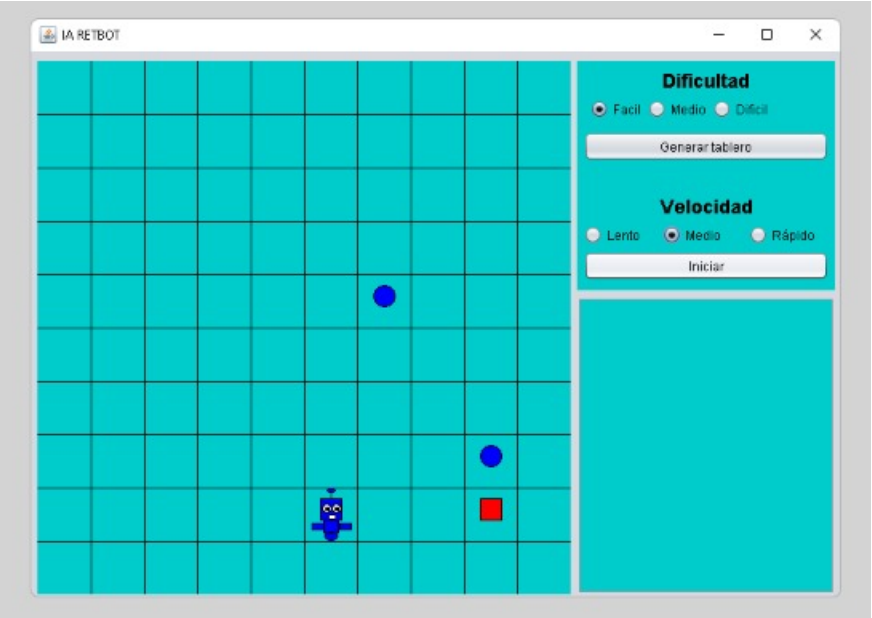
Introducción

El algoritmo A* (conocido también como A estrella o asterisco) es un algoritmo de búsqueda que encuentra la ruta de menor coste entre dos puntos siempre y cuando se cumplan una serie de condiciones. Está clasificado dentro de los algoritmos de búsqueda en grafos ya que tiene la necesidad de dar a los mecanismos robóticos, vehiculares o virtuales un sistema de navegación un sistema de navegación navegación autónomo [1, autónomo [1, 2].
Es un algoritmo que se encarga de encontrar la ruta que representa un menor costo entre un punto A y otro B. Es una técnica de búsqueda para grafos que utiliza una función heurística (ya que trata de métodos exploratorios durante la resolución del problema en el cual la solución se descubre por la evaluación del progreso logrado en la búsqueda de un resultado final) denotada f ’(x), la cual es una aproximación a f(x) que proporciona la verdadera evaluación de un nodo para determinar el orden en que se realiza la visita a los otros nodos del árbol.

Consiste entonces en atravesar una gráfica (de forma definida normalmente) siguiendo una ruta con el menor costo conocido manteniendo una cola sorteada de prioridades con segmentos de rutas alternas a lo largo del camino para que si en un momento dado la ruta que se está siguiendo arroja un valor (costo) superior en comparación con otro segmento de ruta encontrado, esta abandona esa ruta actual de alto coste y en vez de ella toma la que le brinda un camino más “barato”. De esta forma el proceso sigue hasta alcanzar la meta u objetivo.

Descripción general del proyecto

El primer proyecto consiste de un robot que recoge objetos (pelotas color azul), en caso de que al robot se le acabe la batería o se le esté por acabar, se deberá pausar la búsqueda de los objetos e ir a recargar la batería (color rojo) buscando hacer un recorrido óptimo en el que el robot no muera.



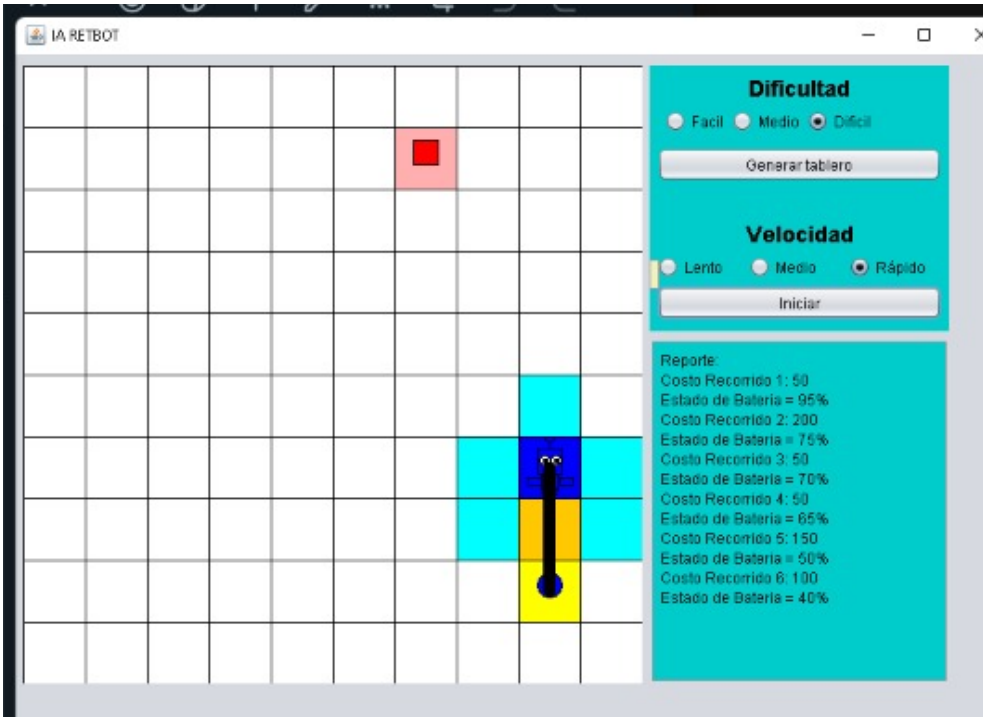
El proyecto tiene diferentes dificultades y velocidades, para la dificultad fácil se generan dos objetos, en la dificultad media se generan 4 y en dificultad difícil se generan 6.

Al iniciar el recorrido se genera simultáneamente un reporte qué es el costo del recorrido que va haciendo de hacia el objeto 1 y este a su vez hacia el objeto n, aunque cómo se mencionó anteriormente en caso de que en medio del recorrido se le está acabando la pila (por ejemplo le quedan 30 de batería y necesita 50 para recoger el objetivo por lo que le que no logra ir por el objetivo así que va por la batería ya que se está quedando sin pila) se generará un recorrido hacia la recarga y nuevamente hacia el siguiente objetivo.

El reporte es el costo de los recorridos, el estado de la batería antes de recoger la pila y muestra después los siguientes recorridos.

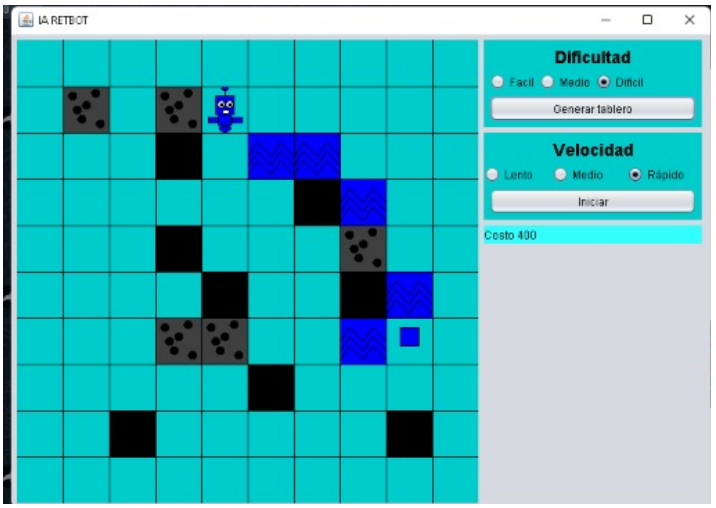
Cuando el robot comienza su recorrido tendrá distintos nodos por los cuales podrá pasar para esto nosotros decidimos implementar una serie de colores qué ayudarán a comprender mejor el funcionamiento:

- Blanco: terreno por explorar (aún no visitados)
- Cyan: terreno próximo a explorar (nodos explotados y abiertos)
- Naranja: terreno explorado (nodos explotados y cerrados)



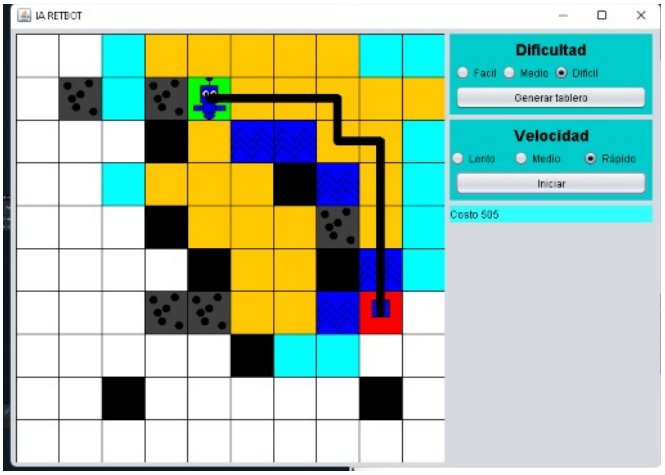
Parte 2 del proyecto:

La segunda parte del proyecto también es un recorrido, este tiene como objetivo el recoger el objeto, de igual forma se buscará realizar un recorrido óptimo, pero en él cual se presentarán algunas dificultades, para nuestro proyecto decidimos colocar dos dificultades por medio de bloques, en este caso grava y agua.



Cuando el robot vaya por el objetivo y si dentro de su recorrido está un bloque de grava o de agua se le agregaran extras al costo total.
Si pasa por un bloque de grava se le suman 25 adicionales y por el contrario si pasa por un bloque de agua son 55 adicionales, esto se verá reflejado en el costo total.

De igual manera se implementaron los tres colores para visualizar mejor la manera en la que realiza el recorrido.



Código documentado:

Dibujo.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package ia;

import java.awt.Color;
import java.awt.Component;
import java.awt.Graphics;
import java.awt.Point;
import java.util.Random;

/**
 *
 * @author RETBOT
 */

// Clase Dibujo creada para crear figuras y el tablero
public class Dibujo {

   // Método para dibujar el tablero sobre el componente establecido
    public static void dibujarTablero(Component componente){
        // Librería Graphics utilizada para crear los dibujos sobre el tablero.
        Graphics hoja = componente.getGraphics();

        // Se elimina todo del tablero
        hoja.setColor(componente.getBackground()); // obtenemos el color de fondo
            // y eliminamos todo lo que este desde el inicio, hasta el fin del tamaño del
```

```
componente
        hoja.fillRect(0, 0, componente.getWidth(), componente.getHeight());

        // Establecemos el color Negro
        hoja.setColor(Color.BLACK);
        // y dibujamos las casillas del tablero
        // 50 px * 50 px
        for(int i=50; i<=500; i+=50){
            hoja.drawLine(0, i, componente.getWidth(), i); // líneas horizontales
            hoja.drawLine(i, 0, i, componente.getWidth()); //líneas verticales
        }

        try {
            Thread.sleep(100);
        } catch (InterruptedException ex) {

java.util.logging.Logger.getLogger(IAFrame.class.getName()).log(java.util.logging.Le
vel.SEVERE, null, ex);
        }
    }

    // metodo para dibujar el robot en el tablero
    public static void dibujarBOT(int x, int y, Component componente){
        // Librería Graphics utilizada para crear los dibujos sobre el tablero.
        Graphics hoja = componente.getGraphics();
        // Establecemos el color del robot
        hoja.setColor(Color.BLUE); // Cuerpo Relleno
        hoja.fillRect(x + 15, y + 10, 20, 20); // Cabeza
        hoja.fillRect(x + 18, y + 30, 14, 12); // Cuerpo
        hoja.fillOval(x + 19, y + 41, 12, 8); // Rueda
        hoja.fillRect(x + 7, y + 33, 11, 6); // Brazo <-
        hoja.fillRect(x + 33, y + 33, 11, 6); // Brazo ->
        hoja.fillOval(x + 21, y + 0, 8, 4); // antena
```

```
        hoja.setColor(Color.WHITE); // Cara Relleno
        hoja.fillOval(x + 17, y + 15, 8, 8); // Ojos <-
        hoja.fillOval(x + 26, y + 15, 8, 8); // Ojos ->
        hoja.fillOval(x + 22, y + 25, 8, 4); // Boca

        hoja.setColor(Color.BLACK); // Cara Relleno
        hoja.fillOval(x + 19, y + 17, 5, 5); // Ojos <-
        hoja.fillOval(x + 28, y + 17, 5, 5); // Ojos ->

        hoja.setColor(Color.BLACK); // Contorno
        hoja.drawRect(x + 15, y + 10, 20, 20); // Cabeza
        hoja.drawRect(x + 18, y + 30, 14, 12); // Cuerpo
        hoja.drawOval(x + 19, y + 41, 12, 8); // Rueda
        hoja.drawRect(x + 7, y + 33, 11, 6); // Brazo <-
        hoja.drawRect(x + 33, y + 33, 11, 6); // Brazo ->
        hoja.drawOval(x + 17, y + 15, 8, 8); // Ojos <-
        hoja.drawOval(x + 26, y + 15, 8, 8); // Ojos ->
        hoja.drawOval(x + 22, y + 25, 8, 4); // Boca
        hoja.drawLine(x + 25, y + 5, x + 25, y + 10); // antena
        hoja.drawOval(x + 21, y + 0, 8, 4); // antena

         try {
            Thread.sleep(100);
        } catch (InterruptedException ex) {

java.util.logging.Logger.getLogger(IAFrame.class.getName()).log(java.util.logging.Le
vel.SEVERE, null, ex);
        }
    }

    public static void dibujarBOTSinPila(int x, int y, Component componente){
        // Librería Graphics utilizada para crear los dibujos sobre el tablero.
        Graphics hoja = componente.getGraphics();
        // Establecemos el color del robot
```

```
hoja.setColor(Color.BLUE); // Cuerpo Relleno
hoja.fillRect(x + 15, y + 10, 20, 20); // Cabeza
hoja.fillRect(x + 18, y + 30, 14, 12); // Cuerpo
hoja.fillOval(x + 19, y + 41, 12, 8); // Rueda
hoja.fillRect(x + 7, y + 33, 11, 6); // Brazo <-
hoja.fillRect(x + 33, y + 33, 11, 6); // Brazo ->
hoja.fillOval(x + 21, y + 0, 8, 4); // antena

hoja.setColor(Color.WHITE); // Cara Relleno
hoja.fillOval(x + 17, y + 15, 8, 8); // Ojos <-
hoja.fillOval(x + 26, y + 15, 8, 8); // Ojos ->
hoja.fillOval(x + 22, y + 25, 8, 4); // Boca

hoja.setColor(Color.RED); // Cara Relleno
hoja.drawLine(x + 18, y + 16, x + 24, y + 22);// Ojos <- XP
hoja.drawLine(x + 24, y + 16, x + 18, y + 22);
hoja.drawLine(x + 27, y + 16, x + 33, y + 22);// Ojos ->
hoja.drawLine(x + 33, y + 16, x + 27, y + 22);

hoja.setColor(Color.BLACK); // Contorno
hoja.drawRect(x + 15, y + 10, 20, 20); // Cabeza
hoja.drawRect(x + 18, y + 30, 14, 12); // Cuerpo
hoja.drawOval(x + 19, y + 41, 12, 8); // Rueda
hoja.drawRect(x + 7, y + 33, 11, 6); // Brazo <-
hoja.drawRect(x + 33, y + 33, 11, 6); // Brazo ->
hoja.drawOval(x + 17, y + 15, 8, 8); // Ojos <-
hoja.drawOval(x + 26, y + 15, 8, 8); // Ojos ->
hoja.drawOval(x + 22, y + 25, 8, 4); // Boca
hoja.drawLine(x + 25, y + 5, x + 25, y + 10); // antena
hoja.drawOval(x + 21, y + 0, 8, 4); // antena

 try {
    Thread.sleep(100);
} catch (InterruptedException ex) {
```

```
java.util.logging.Logger.getLogger(IAFrame.class.getName()).log(java.util.logging.Le
vel.SEVERE, null, ex);
        }
    }

    //  Método para dibujar pelota en tablero
    public static void dibujarPelota(int x, int y, Component componente){
        // Librería Graphics utilizada para crear los dibujos sobre el tablero.
        Graphics hoja = componente.getGraphics();

        // establecemos el color de la pelota
        hoja.setColor(Color.blue);
        // y la dibujamos
        hoja.fillOval(x + 15, y + 10, 20, 20);

        // establecemos el color del contorno de la pelota
        hoja.setColor(Color.BLACK); // Contorno
        hoja.drawOval(x + 15, y + 10, 20, 20); // y se dibuja
    }

     //  Método para dibujar Pila en tablero
     public static void dibujarPila(int x, int y, Component componente){
        Graphics hoja = componente.getGraphics();

        hoja.setColor(Color.RED); // Seleccionamos el color de relleno
        hoja.fillRect(x + 15, y + 10, 20, 20); // y lo dibujamos

        hoja.setColor(Color.BLACK); // Seleccionamos el color del Contorno
        hoja.drawRect(x + 15, y + 10, 20, 20); // y lo dibujamos
    }

     // dibuja las pelotas en el tablero
    public static void generarPelotas(int evil[][],int cant,Component componente){
```

```
        for(int i=1;i<cant-1;i++){
            dibujarPelota(evil[i][0], evil[i][1], componente);
        }
    }


  // Generar numeros aleatorios del 50 al 400(450 limite del panel)
    public static int genRanEnteros(){
        Random azar = new Random(); // Método para generar números aleatorios
        int num = 1; // Inicia en uno, porque si es 0, no se puede dividir
        while(num%50!=0){ // si el numero no es multiplo del 50
           num = azar.nextInt(400)+50; // genera un numero al azar entre el 50 y el 450
        }
        return num; // y lo retorna
    }

   // generar numeros distintos
    public static void aleatorios(int evil[][], int cant) {
        // recordido de la cantidad de los numeros aleatorios
        for (int i = 0; i < cant; i++) {
            evil[i][0] = genRanEnteros();
            evil[i][1] = genRanEnteros();
        }

            // y después hacemos otro recorrido para validar que las posiciones en el
arreglo
        // para que  no se repitan
        for (int k = 0; k < cant; k++) {
            // Obtenemos X y Y del arreglo
            int auxX = evil[k][0];
            int auxY = evil[k][1];

            // recorremos todas las posiciones
            for (int i = 0; i < cant; i++) {
```

```
            // Si k e i son iguales, no hace nada (porque es la misma posición y lo va a
modificar)
            if (k != i) {
               if (evil[i][0] == auxX) { // verifica que X sea diferente a auxX
                  // si son iguales, entra a verificar que la Y sea difetente a auxY
                  if (evil[i][1] == auxY) {
                     // si son iguales, entonces cambia la posición de auxY,
                     //  para que no estén en la misma coordenada
                     evil[i][1] = genRanEnteros();
                  }
               }
               if (evil[i][1] == auxY) { // verifica que Y sea diferente a auxY
                  // si son iguales, entra a verificar que la X sea difetente a auxY
                  if (evil[i][0] == auxX) {
                     // si son iguales, entonces cambia la posición de auxX,
                     //  para que no estén en la misma coordenada
                     evil[i][0] = genRanEnteros();
                  }
               }
            }
         }
      }
   }

    // Método para dibujar todas las piezas del tablero
        public static void dibujarComponentes(Color color, Node<Point> node, int
TILE_SIZE ,Component componente){
      // Librería Graphics utilizada para crear los dibujos sobre el tablero.
      Graphics hoja = componente.getGraphics();

      hoja.setColor(color);// Seleccionamos el color
      int rx = node.getObj().x; // Obtenemos la posición x
      int ry = node.getObj().y; // Obtenemos la posición y
      hoja.fillRect(rx, ry, TILE_SIZE, TILE_SIZE); // dibujamos el relleno de la casilla
```

```
        hoja.setColor(Color.BLACK); // Seleccionamos el color
            hoja.drawRect(rx, ry, TILE_SIZE, TILE_SIZE);// dibujamos el contorno de la
casilla

        // Y seleccionamos el personaje, según el color establecido
        if(color.equals(Color.BLUE)){ // si es verde, es la casilla de salida
            dibujarBOT(rx, ry, componente); // y se dibuja
        }else if(color.equals(Color.RED)){// si es rojo, es la casilla de llegada
            dibujarPelota(rx, ry, componente);
        }
        else if (color.equals(Color.PINK)) {
            dibujarPila(rx, ry, componente);// y se dibuja
        }
        else if (color.equals(Color.YELLOW)) {
             dibujarPelota(rx, ry, componente);
        }
    }
}
```

Calculos.java

```
package ia;

/**
 *
 * @author RETBOT
 */
// Clase utilizada para cálculos variados
public class Calculos {

    // Coordenadas X e Y en el tablero
    public static void cooordenadasXY(int evil[][], int cant, int coordenadas[][]) {
            int contaPelotas = 1; // contador de pelotas inicia en 1, porque el robto es el
primero en generarse en el mapa
```

```
        while (contaPelotas <= cant - 2) {
            int x = dePixelesACoordenadas(evil[contaPelotas][0]);
            int y = dePixelesACoordenadas(evil[contaPelotas][1]);

            coordenadas[contaPelotas-1][0] = x;
            coordenadas[contaPelotas-1][1] = y;
            contaPelotas++;
            System.out.println("X = " + x + " Y = " + y);
        }
    }

    // Método Hheurística utilizado para identificar el nodo más cercano al robot
    public static double heuristica(int x1, int y1, int x2, int y2) {
        return Math.abs(x1 - x2) + Math.abs(y1 - y2);
    }

    // Método utilizado para transformar de píxeles a coordenada
    public static int dePixelesACoordenadas(int aux) {
        int i = 0; // Posición inicial
        while (aux > 0) { // Hasta llegar a 0
            i++; // Aumentan las posiciones
            aux -= 50; // Disminuyen los píxeles
        }
        return i; // y retorna la coordenada
    }

}
```

Edge.java

```
package ia;

/**
 * Edge class.
```

```
 *
 * @author RETBOT
 */
// clase creada para la busqueda de nodos
public class Edge<T> {

    private double g;// costo del nodo
    // para identificar los nodos
    private final Node<T> a;// nodo a
    private final Node<T> b; // nodo b

    // constructor
    public Edge(double g, Node<T> a, Node<T> b) {
        this.g = g;
        this.a = a;
        this.b = b;
    }

    // obtenemos G
    public double getG() {
        return g;
    }

    // ponemos G
    public void setG(double g) {
        this.g = g;
    }

    // nodo a
    public Node<T> getA() {
        return a;
    }

    // nodo b
```

```
    public Node<T> getB() {
        return b;
    }
    // metodo para obtener el nodo opuesto
    public Node<T> getOppositeNode(Node<T> thisNode) {
        if (thisNode == a) { // si el nodo principal es a
            return b; // retornamos b
        }
        else if (thisNode == b) { // si el nodo principal es b
            return a; // retornamos b
        }
        return null; // si no es ninguno
    }
    @Override
    public String toString() {
        return "Edge{" + "g=" + g + ", a=" + a + ", b=" + b + '}';
    }

}
```

Graph.java

```
package ia;

import static ia.Node.State.CLOSED;
import static ia.Node.State.OPEN;
import static ia.Node.State.UNVISITED;
import java.util.ArrayList;
import java.util.List;
import java.util.PriorityQueue;

/**
 * Graph class.
 *
```

```
 * @author RETBOT
 */
// clase creada para Graficar los dibujos
public class Graph<T> {

    // lista de los nodos
    private final List<Node<T>> nodes = new ArrayList<>();
    // lista para almacenar todas los caminos posibles
    private final Heuristic<T> heuristic;

    // constructor
    public Graph(Heuristic<T> heuristic) {
        this.heuristic = heuristic;
    }

    // metodo para agregar un nuevo nodo
    public void addNode(Node n) {
        nodes.add(n);
    }

    // metodo obtener los nodos
    public List<Node<T>> getNodes() {
        return nodes;
    }

    // metodo obtener la Heuristica
    public Heuristic<T> getHeuristic() {
        return heuristic;
    }

    // metodo para el enlace de los nodos
    public void link(Node a, Node b, double cost) {
        // Creamos un nuevo objeto para determinar el costo de los nodos
        Edge edge = new Edge(cost, a, b);
```

```
        a.addEdge(edge); // agregamos el nodo a y b
        b.addEdge(edge);
    }

    // metodo para encontrar la ruta entre los nodos
        public void findPath(Node<T> start, Node<T> target, List<Node<T>> path, boolean pilafuera, boolean pelotafuera) {
        // creamos los estados del nodo
        nodes.forEach(node -> {
            node.setState(UNVISITED); // ponemos el estado sin visitas
            node.setBackPathNode(null); // el nodo anterior en nulo
            node.setG(Double.MAX_VALUE); // y el costo
        });

        start.setG(0); // ponemos el costo inicial en 0
        // y calculamos la distancia entre los nodos
        start.setH(heuristic.calculate(start, target, start));

          // utilizamos la clase PriorityQueue, integrada en java, para darle prioridad al nodo que entro primero
        PriorityQueue<Node<T>> activeNodes = new PriorityQueue<>();
        // y lo iniciamos
        activeNodes.add(start);

        // si los nodos no estan vacios
        while (!activeNodes.isEmpty()) {
            Node<T> currentNode = activeNodes.poll();// nodo actual
            currentNode.setState(CLOSED);// estado del nodo

            // Si se encontro el nodo destino
            if (currentNode == target) {
                path.clear(); // se limpia el path
                target.retrievePath(path); // y se obtiene la ruta
                return;
```

```
        }

        // si aun no encontro el nodo destino, continua con las busquedas
        currentNode.getEdges().forEach((edge) -> {
            // se establece el nodo vecino
            Node<T> neighborNode = edge.getOppositeNode(currentNode);
            // y el costo de la ruta
            double neighborG = currentNode.getG() + edge.getG();

              // si el nodo vecino, no se encuentra bloqueado, pero en esta ocasión no se utilizan bloqueos
            if (!neighborNode.isBlocked()
                    && neighborG < neighborNode.getG()) {
                if (pilafuera) { // Si la pila está activa
                    if (!neighborNode.isPila()) { //  Va a evitar chocar con ella
                        // se ingresa al path
                        neighborNode.setBackPathNode(currentNode);

                        // si se guarda
                        neighborNode.setG(neighborG);

                        // calculo de la ruta
                        double h = heuristic.calculate(start, target, neighborNode);
                        neighborNode.setH(h);

                        // si los nodos activos no contienen nodo vecino
                        if (!activeNodes.contains(neighborNode)) {
                                activeNodes.add(neighborNode); // se agrega a los nodos activos
                            neighborNode.setState(OPEN); // y se pone el estado abierto
                        }
                    }
                }
                if (pelotafuera) { // Si la pila está activa y va por la pelota
```

```
                if (!neighborNode.isPelota()) { //  Va a evitar chocar las pelotas
                    // se ingresa al path
                    neighborNode.setBackPathNode(currentNode);

                    // si se guarda
                    neighborNode.setG(neighborG);

                    // calculo de la ruta
                    double h = heuristic.calculate(start, target, neighborNode);
                    neighborNode.setH(h);

                    // si los nodos activos no contienen nodo vecino
                    if (!activeNodes.contains(neighborNode)) {
                            activeNodes.add(neighborNode); // se agrega a los nodos
activos
                        neighborNode.setState(OPEN); // y se pone el estado abierto
                    }
                }
            }
            if (!pilafuera) { // si la pila no esta activa
                // se ingresa al path
                neighborNode.setBackPathNode(currentNode);

                // si se guarda
                neighborNode.setG(neighborG);

                // calculo de la ruta
                double h = heuristic.calculate(start, target, neighborNode);
                neighborNode.setH(h);

                // si los nodos activos no contienen nodo vecino
                if (!activeNodes.contains(neighborNode)) {
                    activeNodes.add(neighborNode); // se agrega a los nodos activos
                    neighborNode.setState(OPEN); // y se pone el estado abierto
```

```
                }
            }
        }
    });
  }
 }
}
```

IA Frame.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/GUIForms/JFrame.java to edit this template
 */
package ia;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.Stroke;
import java.util.ArrayList;
import java.util.List;

/**
 *
 * @author RETBOT
 */
public final class IAFrame extends javax.swing.JFrame {

    /**
     * RETBOT
     */
```

```
// cantidad de posiciones en el tablero
int cant = 0;
// arreglo para almacenar las posiciones
int evil[][];

// Pos BOT
int x = 0, y = 0;

// Pos bateria
int xB = 0, yB = 0;
// bateria del robot
int bateria = 1000;
// bandera para activar y desactivar la batería
boolean banBateria = true;

 // Altura de la casilla
private static final int TILE_SIZE = 50;
 // Cantidad de columnas
private static final int GRID_COLS = 10;
 // Cantidad de filas
private static final int GRID_ROWS = 10;
// Tablero
private Node[][] grid;
// Objetos en el tablero
private Graph<Point> graph;
// inicio
private Node<Point> startNode;
// fin
private Node<Point> targetNode;
// lista de todos los nodos
private List<Node<Point>> path;
// recorrido
private final Stroke stroke = new BasicStroke(10
    , BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND);
```

```
// inicio
public IAFrame() {
    initComponents();
}

// Método para seleccionar la dificultad del juego y dibujo de las mismas
public void pelotitas() {
   // Selección del modo de juego FACIL, MEDIO o DIFICIL
    if(jRBDificultadFacil.isSelected()){
        cant = 2 + 2;
    }else if(jRBDificultadMedio.isSelected()){
        cant = 4 + 2;
    }else if(jRBDificultadDificil.isSelected()){
       cant = 6 + 2;
    }
    // se dimensiona el arreglo de las posiciones en el tablero
    evil = new int[cant][2];

    // se generan los números aleatorios, para las posiciones en el tablero
    Dibujo.aleatorios(evil, cant);
    // y se dibujan en el tablero las pelotas
    Dibujo.generarPelotas(evil, cant, JPTablero);
}


// crea el tablero vacio y recorrido del tablero
private void createGrid() {
    // primer ciclo para crear renglones
    for (int y2 = 0; y2 < GRID_ROWS; y2++) {
        // segundo ciclo para crear columnas
        for (int x2 = 0; x2 < GRID_COLS; x2++) {
            // otenemos el tamaño de las casillas
            int nx = x2 * TILE_SIZE;
            int ny = y2 * TILE_SIZE;
```

```
            // se crea un nodo y se establecen las posiciones
            Node node = new Node(new Point(nx, ny));
            // se agrega al tablero
            graph.addNode(node);
            grid[y2][x2] = node;
        }
    }

    // recorrido en el tablero
    // Recorrido vertical
    for (int y2 = 0; y2 < GRID_ROWS - 1; y2++) {
        for (int x2 = 0; x2 < GRID_COLS; x2++) {
            // vertical '|'
            Node top = grid[y2][x2];
            Node bottom = grid[y2 + 1][x2];
            graph.link(top, bottom, TILE_SIZE);
        }
    }

    for (int x2 = 0; x2 < GRID_COLS - 1; x2++) {
        for (int y2 = 0; y2 < GRID_ROWS; y2++) {
            // horizontal '-'
            Node left = grid[y2][x2];
            Node right = grid[y2][x2 + 1];
            graph.link(left, right, TILE_SIZE);
        }
    }
}

// metodo para dibujar los Nodos
private void drawNode(Graphics2D g, Node<Point> node) {
    Color color = Color.BLACK; // color principal

    switch (node.getState()) { // dependiendo del estado, cambia el color del nodo
```

```
            case OPEN: // si esta abierto
                color = Color.CYAN; // se pone cyan
                break;
            case CLOSED: // si esta cerrado
                color = Color.ORANGE; // se pone naranja
                break;
            case UNVISITED: // si no esta vicitado se pone blanco
                color = Color.WHITE;
                break;
        }

        if (node == startNode) {// nodo inicio
            color = Color.BLUE;
        } else if (node == targetNode) {// nodo destino
            color = Color.RED;
            if (node.isPila()) { // si es pila
                color = Color.PINK;
            } else if (node.isPelota()) { // si es pelota
                color = Color.YELLOW;
            }

        } else if (node.isBlocked()) {// si esta bloqueado es negro
            color = Color.BLACK;
        } else if (node.isPila()) {// si es pila
            color = Color.PINK;
        } else if (node.isPelota()) {// si es pelota
            color = Color.YELLOW;
        }
        //y lo manda a dibujar
        Dibujo.dibujarComponentes(color, node, TILE_SIZE, JPTablero);
    }

    // dibujar el recorrido
    private void drawPath(Graphics2D g) {
```

```
        // Se utiliza la clase Stroke, para el contorno de la línea
        Stroke originalStroke = g.getStroke();
        g.setColor(Color.BLACK); // se pone de color rosa
        g.setStroke(stroke);
        // y se empieza a trazar la linea en el tablero
        for (int i = 0; i < path.size() - 1; i++) {
            // recorriendo hasta llegar al nodo destino
            Node<Point> a = path.get(i);
            Node<Point> b = path.get(i + 1);

            // y obtenemos el inicio y el final del nodo A y B, para trazar
            // la línea por en medio
            int x1 = a.getObj().x + TILE_SIZE / 2;
            int y1 = a.getObj().y + TILE_SIZE / 2;
            int x2 = b.getObj().x + TILE_SIZE / 2;
            int y2 = b.getObj().y + TILE_SIZE / 2;
            // y de dibuja la linea
            g.drawLine(x1, y1, x2, y2);
            // Velocidad de trazado en pantalla
            if (jRBVelocidadLento.isSelected()) {
                try {
                    Thread.sleep(500);
                } catch (InterruptedException ex) {

java.util.logging.Logger.getLogger(IAFrame.class.getName()).log(java.util.logging.Le
vel.SEVERE, null, ex);
                }
            } else if (jRBVelocidadMedio.isSelected()) {
                try {
                    Thread.sleep(350);
                } catch (InterruptedException ex) {

java.util.logging.Logger.getLogger(IAFrame.class.getName()).log(java.util.logging.Le
vel.SEVERE, null, ex);
```

```
            }
        } else if (jRBVelocidadRapido.isSelected()) {
            try {
                Thread.sleep(100);
            } catch (InterruptedException ex) {

java.util.logging.Logger.getLogger(IAFrame.class.getName()).log(java.util.logging.Le
vel.SEVERE, null, ex);
            }
        }

    }
    // y al final lo decora
    g.setStroke(originalStroke);
}



/**
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    buttonGroup1 = new javax.swing.ButtonGroup();
    buttonGroup2 = new javax.swing.ButtonGroup();
    JPTablero = new javax.swing.JPanel();
    jScrollPane1 = new javax.swing.JScrollPane();
    jTAResultados = new javax.swing.JTextArea();
    jPanel1 = new javax.swing.JPanel();
    jButTablero = new javax.swing.JButton();
```

```
        jButIniciar = new javax.swing.JButton();
        jLabel2 = new javax.swing.JLabel();
        jRBDificultadFacil = new javax.swing.JRadioButton();
        jRBDificultadMedio = new javax.swing.JRadioButton();
        jRBDificultadDificil = new javax.swing.JRadioButton();
        jLabel3 = new javax.swing.JLabel();
        jRBVelocidadLento = new javax.swing.JRadioButton();
        jRBVelocidadMedio = new javax.swing.JRadioButton();
        jRBVelocidadRapido = new javax.swing.JRadioButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("IA RETBOT");
        setBackground(new java.awt.Color(153, 153, 255));
        setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
        setLocation(new java.awt.Point(0, 0));
        setMinimumSize(new java.awt.Dimension(760, 550));
        setPreferredSize(new java.awt.Dimension(500, 500));

        JPTablero.setBackground(new java.awt.Color(0, 204, 204));
        JPTablero.setPreferredSize(new java.awt.Dimension(500, 500));

                                    javax.swing.GroupLayout JPTableroLayout = new
javax.swing.GroupLayout(JPTablero);
        JPTablero.setLayout(JPTableroLayout);
        JPTableroLayout.setHorizontalGroup(

JPTableroLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
            .addGap(0, 500, Short.MAX_VALUE)
        );
        JPTableroLayout.setVerticalGroup(

JPTableroLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
```

```
    .addGap(0, 500, Short.MAX_VALUE)
);

jTAResultados.setBackground(new java.awt.Color(0, 204, 204));
jTAResultados.setColumns(20);
jTAResultados.setRows(5);
jScrollPane1.setViewportView(jTAResultados);

jPanel1.setBackground(new java.awt.Color(0, 204, 204));
jPanel1.setToolTipText("");

jButTablero.setText("Generar tablero");
jButTablero.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButTableroActionPerformed(evt);
    }
});

jButIniciar.setText("Iniciar");
jButIniciar.setEnabled(false);
jButIniciar.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButIniciarActionPerformed(evt);
    }
});

jLabel2.setFont(new java.awt.Font("Cantarell", 1, 18)); // NOI18N
jLabel2.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
jLabel2.setText("Dificultad");

buttonGroup1.add(jRBDificultadFacil);
jRBDificultadFacil.setSelected(true);
jRBDificultadFacil.setText("Facil");
```

```
        buttonGroup1.add(jRBDificultadMedio);
        jRBDificultadMedio.setText("Medio");

        buttonGroup1.add(jRBDificultadDificil);
        jRBDificultadDificil.setText("Dificil");

        jLabel3.setFont(new java.awt.Font("Cantarell", 1, 18)); // NOI18N
        jLabel3.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        jLabel3.setText("Velocidad");

        buttonGroup2.add(jRBVelocidadLento);
        jRBVelocidadLento.setSelected(true);
        jRBVelocidadLento.setText("Lento");

        buttonGroup2.add(jRBVelocidadMedio);
        jRBVelocidadMedio.setText("Medio");

        buttonGroup2.add(jRBVelocidadRapido);
        jRBVelocidadRapido.setText("Rápido");

                                javax.swing.GroupLayout jPanel1Layout = new
javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(

jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addContainerGap()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.
LEADING)
                        .addComponent(jLabel2, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addGroup(jPanel1Layout.createSequentialGroup()
```

```
                .addGap(6, 6, 6)
                .addComponent(jRBDificultadFacil)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jRBDificultadMedio)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jRBDificultadDificil)
                .addGap(0, 0, Short.MAX_VALUE))
                  .addComponent(jLabel3, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
              .addComponent(jButTablero, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
              .addComponent(jButIniciar, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                          .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel1Layout.createSequentialGroup()
                                  .addComponent(jRBVelocidadLento,
javax.swing.GroupLayout.PREFERRED_SIZE,                    68,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                  .addComponent(jRBVelocidadMedio,
javax.swing.GroupLayout.PREFERRED_SIZE,                    75,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                                  .addComponent(jRBVelocidadRapido,
javax.swing.GroupLayout.PREFERRED_SIZE,                    75,
javax.swing.GroupLayout.PREFERRED_SIZE)))
            .addContainerGap())
    );
    jPanel1Layout.setVerticalGroup(
```

```
jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabel2)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.
BASELINE)
                    .addComponent(jRBDificultadFacil)
                    .addComponent(jRBDificultadMedio)
                    .addComponent(jRBDificultadDificil))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(jButTablero)
                .addGap(30, 30, 30)
                .addComponent(jLabel3)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.
BASELINE)
                    .addComponent(jRBVelocidadLento)
                    .addComponent(jRBVelocidadMedio)
                    .addComponent(jRBVelocidadRapido))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jButIniciar)
                .addGap(9, 9, 9))
        );

                                        javax.swing.GroupLayout      layout      =      new
```

```
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                                .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                .addContainerGap()
                                                        .addComponent(JPTablero,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADIN
G, false)
                        .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                                                    .addComponent(jScrollPane1,
javax.swing.GroupLayout.PREFERRED_SIZE, 0, Short.MAX_VALUE))
                .addContainerGap(15, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGap(9, 9, 9)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADIN
G, false)
                    .addGroup(layout.createSequentialGroup()
                                                        .addComponent(jPanel1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addComponent(jScrollPane1))
                                                        .addComponent(JPTablero,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addContainerGap(8, Short.MAX_VALUE))
        );

        pack();
    }// </editor-fold>
//  Botón para generar el tablero
    private void jButTableroActionPerformed(java.awt.event.ActionEvent evt) {
        // Se borra el contenido de los resultados
        jTAResultados.setText("");
        // Se dibuja el tablero
        Dibujo.dibujarTablero(JPTablero);
        // Se dibujan las pelotitas
        pelotitas();

        // robot posicion inicial
        x = evil[0][0];
        y = evil[0][1];
        // bateria
        xB = evil[evil.length-1][0];
        yB = evil[evil.length-1][1];

        // dibuja el robot (Inicio)
        Dibujo.dibujarBOT(x,y,JPTablero);
        // y un cuadrado (Fin)
        Dibujo.dibujarPila(xB, yB,JPTablero);
         // Una vez establecido el tablero, podemos iniciar el recorrido
        jButIniciar.setEnabled(true);
```

```
}

private void jButIniciarActionPerformed(java.awt.event.ActionEvent evt) {
    // se borran los resultados anteriores
    jTAResultados.setText("Reporte:\n");
    // se reinicia la bateria
    bateria = 1000;
    banBateria = true;

    //CREAMOS ORIGEN DE LA RUTA
    int nx = Calculos.dePixelesACoordenadas(x);
    int ny = Calculos.dePixelesACoordenadas(y);
    int nxFin;
    int nyFin;

    // Bateria
    int posBateriaX = Calculos.dePixelesACoordenadas(xB);
    int posBateriaY = Calculos.dePixelesACoordenadas(yB);
    int estadoBateria;

    // MATRIZ PARA GUARDAR LAS CORRDENADAS
    int[][] coord = new int[cant-2][2];
    int cantCoord = coord.length;

    ArrayList<Integer> coordX = new ArrayList<>();
    ArrayList<Integer> coordY = new ArrayList<>();

    //CREAMOS LA MATRIZ //AÑADIMOS LOS OBJETOS CASILLAS
    Calculos.cooordenadasXY(evil, cant, coord);

    // almacenar todas las pelotitas
    for (var coord1 : coord) {
        coordX.add(coord1[0]);
        coordY.add(coord1[1]);
```

```
        }

        //CREAMOS ORIGEN DE LA RUTA
        for (int i = 0; i < coord.length; i++) {
            if (bateria >= 0) {
                // tablero
                path = new ArrayList<>();
                grid = new Node[GRID_ROWS][GRID_COLS];

                graph = new Graph<>((start, target, current) -> {
                    // heuristic = linear distance
                    int dx = target.getObj().x - current.getObj().x;
                    int dy = target.getObj().y - current.getObj().y;
                    return Math.sqrt(dx * dx + dy * dy);
                });
                // se crea el tablero
                createGrid();

                // se agregan las pelotitas
                for (int j = 0; j < coordX.size(); j++) {
                    int row = coordX.get(j);
                    int col = coordY.get(j);
                    grid[col][row].setPelota(true);
                }
                // y agregamos la pila
                if (banBateria) { // si aun no toma la bateria
                    grid[posBateriaY][posBateriaX].setPila(true); // activa la bateria
                } else { // si no
                        grid[posBateriaY][posBateriaX].setPila(false); // quita la bateria del
tablero
                }

                // Distancia de todas las coordenadas
                ArrayList<Double> distCoord = new ArrayList<>();
```

```
            // y empezamos a calcular la distancia
            int j = 0;
            while (j < cantCoord) // Distancia entre el bot y las pelotas
            {
               // x e y de cada pelotita
               nxFin = coordX.get(j);
               nyFin = coordY.get(j);
               // y lo agregamos al arraylista
               distCoord.add(Calculos.heuristica(nx, ny, nxFin, nyFin));
               j++;
            }
            //Para después ver cuál es la distancia más corta
            int indice = 0;
            double menor = distCoord.get(0); // Iniciando por la primera pelotita
                  for (int d = 1; d < distCoord.size(); d++) { // Para después verificar las
demás
               if (distCoord.get(d) < menor) {
                  menor = distCoord.get(d);
                  indice = d; // y almacenar la posición de la pelotita
               }
            }
            // Una vez obtenido la pelotita más cercana, obtenemos sus coordenadas
            nxFin = coordX.get(indice);
            nyFin = coordY.get(indice);

            // incio del recorrido
            startNode = grid[ny][nx];
            targetNode = grid[nyFin][nxFin];

            // se limpia el recorrido
            path.clear();
            // y se traza
            graph.findPath(startNode, targetNode, path, banBateria, false);
```

```
// Verificamos la distancia entre el robot y la batería
                double distanciaRyB = Calculos.heuristica(nx, ny, posBateriaX, posBateriaY) * 50;
// Obtenemos el costo del recorrido inicial
int rec1 = (int)targetNode.getG();
     //y después obtenemos el recorrido de la distancia entre el robot y la batería
int rec2 = (int)distanciaRyB;
// Obtenemos el valor de la batería después de los dos recorridos
int val1 = bateria - rec1;
int val2 = bateria - rec2;

// y la bateria restante despues del primer recorrido
int bateriaRestante = bateria - rec1;

// si el valor del primer recorrido es menor o igual a 0
   // o la batería restante es menor o igual a 300 y no es el último recorrido del tablero
    if (val1 <= 0 || (bateriaRestante<=300 && (i+1)<coord.length)) { // Si no puede llegar a la pelota
      if ((val2 >= 0 && val2<=bateria) && banBateria) { // Verificamos que pueda llegar a la batería, si puede, va por la batería
        // incio del recorrido
        startNode = grid[ny][nx];
        // fin del recorrido (Bateria )
        targetNode = grid[posBateriaY][posBateriaX];
        // indicamos que vamos por la bateria
        banBateria = false;

        // se limpia el recorrido
        path.clear();
        // y se traza
        graph.findPath(startNode, targetNode, path, banBateria, true);
```

```
                Graphics2D g = (Graphics2D) JPTablero.getGraphics();
                 // recorrido en el tablero
                graph.getNodes().forEach(node -> {
                   drawNode(g, node);
                });
                // se dibuja el reccorrido
                drawPath(g);
                 // y se establece el costo
                int costo = (int) targetNode.getG();
                // y se agrega a Reporte
                     jTAResultados.append("Costo Recorrido " + (i + 1) + ": " + costo +
"\n");
                // Se actualiza la batería, con el recorrido para agregar más batería
                bateria -= costo;
                estadoBateria = bateria * 1 / 10;
                jTAResultados.append("\nEstado de batería\n antes de recoger pila =
" + estadoBateria + "% \n");
                // y despues se agrega la bateria
                bateria = 1000;
                estadoBateria = bateria * 1 / 10;
                // y se agrega al reporte
                 jTAResultados.append("Estado de batería\n después de recoger pila
= " + estadoBateria + "% \n\n");
                // se actualiza la posición del robot
                nx = posBateriaX;
                ny = posBateriaY;

                // Distancia de todas las coordenadas con la posición actual del robot
                ArrayList<Double> distCoordAux = new ArrayList<>();
                // y empezamos a calcular la distancia
                j = 0;
                while (j < cantCoord) // Distancia entre el bot y las pelotas
                {
                   // x e y de cada pelotita
```

```
                    nxFin = coordX.get(j);
                    nyFin = coordY.get(j);
                    // y lo agregamos al arraylista
                    distCoordAux.add(Calculos.heuristica(nx, ny, nxFin, nyFin));
                    j++;
                }

            // Una vez obtenido la pelotita más cercana, obtenemos sus coordenadas
                indice = 0; // /Para después ver cuál es la distancia más corta
                menor = distCoordAux.get(0);// Iniciando por la primera pelotita
                 for (int d = 1; d < distCoordAux.size(); d++) {// Para después verificar
las demás
                    if (distCoordAux.get(d) < menor) {
                        menor = distCoordAux.get(d);
                        indice = d; // y almacenar la posición de la pelotita
                    }
                }
                // y obtener la posición final del recorrido
                nxFin = coordX.get(indice);
                nyFin = coordY.get(indice);

              } else { // si no puede terminamos las busquedas
                Dibujo.dibujarTablero(JPTablero);

                // se agregan las pelotitas al tablero
                for (j = 0; j < coordX.size(); j++) {
                    int row = coordX.get(j);
                    int col = coordY.get(j);
                    Dibujo.dibujarPelota(row * 50, col * 50, JPTablero);
                }
                // si la batería está activa
                if (banBateria) {
                    Dibujo.dibujarPila(xB, yB, JPTablero); // la agrega al tablero
                }
```

```
                // Dibujamos al robot sin batería
                Dibujo.dibujarBOTSinPila(nx * 50, ny * 50, JPTablero);
                        jTAResultados.append("\nEl robot no cuenta con la suficiente
energía\npara recoger la pelota o la batería \n");

                // el costo final y lo agregamos
                int costo = (int) targetNode.getG();
                    jTAResultados.append("Costo Recorrido " + (i + 1) + ": " + costo +
"\n");

                bateria -= costo;
                // y agregamos el estado de la batería
                estadoBateria = bateria * 1 / 10;
                   jTAResultados.append("Estado de Bateria = " + estadoBateria + "%
\n");

                // Estado de la batería para el recorrido de la siguiente pelotita
                int bateriaRec1 = (int) targetNode.getG() * 1 / 10;
                // Estado de la batería para el recorrido de la recarga de batería
                int bateriaRec2 = (int) distanciaRyB * 1 / 10;
                // se agrega al tablero
                  jTAResultados.append("Bateria necesaria para \nrecoger sig. pelota
= " + bateriaRec1 + "%\n");
                if(banBateria) // si la batería esta activa, se agrega
                    jTAResultados.append("Bateria necesaria para \nrecoger bateria =
" + bateriaRec2 + "%\n");
                // se pone la bateria en 0
                bateria = 0;
                // y termina el for
                break;
              }
            }
            // Si la bateria se encuentra bien
```

```
            // vuelve a iniciar el recorrido
            startNode = grid[ny][nx];
            targetNode = grid[nyFin][nxFin];

             // se limpia el recorrido
            path.clear();
            // y se traza
            graph.findPath(startNode, targetNode, path, banBateria, false);
            Graphics2D g = (Graphics2D) JPTablero.getGraphics();
            // recorrido en el tablero
            graph.getNodes().forEach(node -> {
                drawNode(g, node);
            });
            // se dibuja el reccorrido
            drawPath(g);
            // se obtiene el costo
            int costo = (int) targetNode.getG();
            // y se agrega al Reporte
            jTAResultados.append("Costo Recorrido " + (i + 1) + ": " + costo + "\n");
            bateria -= costo;
            // junto con el estado de la bateria
            estadoBateria = bateria * 1 / 10;
            jTAResultados.append("Estado de Bateria = " + estadoBateria + "% \n");
            // y se actualiza el estado actual del robot
            nx = nxFin;
            ny = nyFin;
            // se eliminan las coordenadas
            coordX.remove(indice);
            coordY.remove(indice);
            cantCoord -= 1;
        }
    }
}
```

```
    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
            //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
                                                    * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
                                    for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {
            java.util.logging.Logger.getLogger(IAFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex) {
            java.util.logging.Logger.getLogger(IAFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {
            java.util.logging.Logger.getLogger(IAFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
```

```
java.util.logging.Logger.getLogger(IAFrame.class.getName()).log(java.util.logging.Le
vel.SEVERE, null, ex);
        }
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new IAFrame().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JPanel JPTablero;
    private javax.swing.ButtonGroup buttonGroup1;
    private javax.swing.ButtonGroup buttonGroup2;
    private javax.swing.JButton jButIniciar;
    private javax.swing.JButton jButTablero;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JRadioButton jRBDificultadDificil;
    private javax.swing.JRadioButton jRBDificultadFacil;
    private javax.swing.JRadioButton jRBDificultadMedio;
    private javax.swing.JRadioButton jRBVelocidadLento;
    private javax.swing.JRadioButton jRBVelocidadMedio;
    private javax.swing.JRadioButton jRBVelocidadRapido;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JTextArea jTAResultados;
    // End of variables declaration
```

```
}
```

IA Frame.java (programa 2)

```
package ia;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.Stroke;
import java.util.ArrayList;
import java.util.List;

/**
 *
 * @author RETBOT
 */
public final class IAFrame extends javax.swing.JFrame {
    // Posición inicial BOT
    int x = 0, y = 0;

    // Posición inicial bateria
    int xB = 0, yB = 0;

    // Cantidad de objetos en el tablero
    int cant = 2;

    // arreglo para almacenar los objetos en el tablero
    int evil[][];

    // Altura de la casilla
    private static final int TILE_SIZE = 50;
```

```
// Cantidad de columnas
private static final int GRID_COLS = 10;
// Cantidad de filas
private static final int GRID_ROWS = 10;

// Tablero
private Node[][] grid;

// Objetos en el tablero
private Graph<Point> graph;

// inicio
private Node<Point> startNode;

// fin
private Node<Point> targetNode;

// lista de todos los nodos
private List<Node<Point>> path;

// recorrido
private final Stroke stroke = new BasicStroke(10
    , BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND);

// inicio
public IAFrame() {
    initComponents();
}

// crea el tablero vacio y recorrido del tablero
private void createGrid() {
    // primer ciclo para crear renglones
    for (int y2 = 0; y2 < GRID_ROWS; y2++) {
        // segundo ciclo para crear columnas
```

```
        for (int x2 = 0; x2 < GRID_COLS; x2++) {
            // otenemos el tamaño de las casillas
            int nx = x2 * TILE_SIZE;
            int ny = y2 * TILE_SIZE;
            // se crea un nodo y se establecen las posiciones
            Node node = new Node(new Point(nx, ny));
            // se agrega al tablero
            graph.addNode(node);
            grid[y2][x2] = node;
        }
    }
    // recorrido en el tablero
    // Recorrido vertical
    for (int y2 = 0; y2 < GRID_ROWS - 1; y2++) {
        for (int x2 = 0; x2 < GRID_COLS; x2++) {
            // vertical '|'
            Node top = grid[y2][x2];
            Node bottom = grid[y2 + 1][x2];
            graph.link(top, bottom, TILE_SIZE);
        }
    }

    for (int x2 = 0; x2 < GRID_COLS - 1; x2++) {
        for (int y2 = 0; y2 < GRID_ROWS; y2++) {
            // horizontal '-'
            Node left = grid[y2][x2];
            Node right = grid[y2][x2 + 1];
            graph.link(left, right, TILE_SIZE);
        }
    }
}

// metodo para dibujar los Nodos
private void drawNode(Graphics2D g, Node<Point> node) {
```

```
    Color color = Color.BLACK; // color principal

    switch (node.getState()) { // dependiendo del estado, cambia el color del nodo
        case OPEN: // si esta abierto
            color = Color.CYAN; // se pone cyan
            break;
        case CLOSED: // si esta cerrado
            color = Color.ORANGE; // se pone naranja
            break;
        case UNVISITED: // si no esta vicitado se pone blanco
            color = Color.WHITE;
            break;
    }

    if (node == startNode)  // nodo inicio de color verde
        color = Color.GREEN;
    else if (node == targetNode)  // nodo destino de color rojo
        color = Color.RED;
    else if (node.isBlocked())  // si esta bloqueado es negro
        color = Color.BLACK;
    else if(node.isAgua()) // si es agua azul
        color = Color.BLUE;
    else if(node.isTierra()) // si es tierra gris
        color = Color.DARK_GRAY;

    //y lo manda a dibujar
    Dibujo.dibujarComponentes(color, node, TILE_SIZE, JPTablero);
}

// dibujar el recorrido
private void drawPath(Graphics2D g) {
    // Se utiliza la clase Stroke, para el contorno de la línea
    Stroke originalStroke = g.getStroke();
    g.setColor(Color.BLACK); // se pone de color rosa
```

```
        g.setStroke(stroke);
        // y se empieza a trazar la linea en el tablero
        for (int i = 0; i < path.size() - 1; i++) {
            // recorriendo hasta llegar al nodo destino
            Node<Point> a = path.get(i);
            Node<Point> b = path.get(i + 1);

            // y obtenemos el inicio y el final del nodo A y B, para trazar
            // la línea por en medio
            int x1 = a.getObj().x + TILE_SIZE / 2;
            int y1 = a.getObj().y + TILE_SIZE / 2;
            int x2 = b.getObj().x + TILE_SIZE / 2;
            int y2 = b.getObj().y + TILE_SIZE / 2;
            // y de dibuja la linea
            g.drawLine(x1, y1, x2, y2);
            // Velocidad de trazado en pantalla
            if (jRBVelocidadLento.isSelected()) {
                try {
                    Thread.sleep(500);
                } catch (InterruptedException ex) {

java.util.logging.Logger.getLogger(IAFrame.class.getName()).log(java.util.logging.Le
vel.SEVERE, null, ex);
                }
            } else if (jRBVelocidadMedio.isSelected()) {
                try {
                    Thread.sleep(250);
                } catch (InterruptedException ex) {

java.util.logging.Logger.getLogger(IAFrame.class.getName()).log(java.util.logging.Le
vel.SEVERE, null, ex);
                }
            } else if (jRBVelocidadRapido.isSelected()) {
                try {
```

```
                Thread.sleep(100);
            } catch (InterruptedException ex) {

java.util.logging.Logger.getLogger(IAFrame.class.getName()).log(java.util.logging.Le
vel.SEVERE, null, ex);
            }
        }

    }
    // y al final lo decora
    g.setStroke(originalStroke);
}

// Método para convertir de píxeles a coordenadas
private int dePixelesACoordenadas(int aux) {
    int i = 0; //  Inicia en la posición 0
    while (aux > 0) { // Ciclo hasta que los píxeles se terminen ( aux = 0)
        i++; // aumenta la posición
        aux -= 50; // y disminuye las coordenadas
    }
    return i; // al final retorna la coordenada en donde se encuentra
}

/**
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    buttonGroup1 = new javax.swing.ButtonGroup();
    buttonGroup2 = new javax.swing.ButtonGroup();
```

```
        JPTablero = new javax.swing.JPanel();
        jPanel1 = new javax.swing.JPanel();
        jButTablero = new javax.swing.JButton();
        jLabel1 = new javax.swing.JLabel();
        jRBDificultadFacil = new javax.swing.JRadioButton();
        jRBDificultadMedio = new javax.swing.JRadioButton();
        jRBDificultadDificil = new javax.swing.JRadioButton();
        jLabCosto = new javax.swing.JLabel();
        jPanel2 = new javax.swing.JPanel();
        jLabel2 = new javax.swing.JLabel();
        jRBVelocidadLento = new javax.swing.JRadioButton();
        jRBVelocidadMedio = new javax.swing.JRadioButton();
        jRBVelocidadRapido = new javax.swing.JRadioButton();
        jButIniciar = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("IA RETBOT");
        setBackground(new java.awt.Color(153, 153, 255));
        setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
        setLocation(new java.awt.Point(0, 0));
        setMinimumSize(new java.awt.Dimension(750, 550));
        setPreferredSize(new java.awt.Dimension(500, 500));

        JPTablero.setBackground(new java.awt.Color(0, 204, 204));
        JPTablero.setPreferredSize(new java.awt.Dimension(500, 500));

                                    javax.swing.GroupLayout JPTableroLayout = new
javax.swing.GroupLayout(JPTablero);
        JPTablero.setLayout(JPTableroLayout);
        JPTableroLayout.setHorizontalGroup(

JPTableroLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
            .addGap(0, 500, Short.MAX_VALUE)
```

```
        );
        JPTableroLayout.setVerticalGroup(
JPTableroLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
            .addGap(0, 500, Short.MAX_VALUE)
        );

        jPanel1.setBackground(new java.awt.Color(0, 204, 204));

        jButTablero.setText("Generar tablero");
        jButTablero.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButTableroActionPerformed(evt);
            }
        });

        jLabel1.setFont(new java.awt.Font("Cantarell", 1, 18)); // NOI18N
        jLabel1.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        jLabel1.setText("Dificultad");

        buttonGroup1.add(jRBDificultadFacil);
        jRBDificultadFacil.setSelected(true);
        jRBDificultadFacil.setText("Facil");

        buttonGroup1.add(jRBDificultadMedio);
        jRBDificultadMedio.setText("Medio");

        buttonGroup1.add(jRBDificultadDificil);
        jRBDificultadDificil.setText("Dificil");

                                    javax.swing.GroupLayout jPanel1Layout = new
javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
```

```
        jPanel1Layout.setHorizontalGroup(

jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addContainerGap()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.
LEADING)
                    .addComponent(jButTablero, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addComponent(jLabel1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addComponent(jRBDificultadFacil)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(jRBDificultadMedio)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(jRBDificultadDificil)
                        .addGap(0, 0, Short.MAX_VALUE)))
                .addContainerGap())
        );
        jPanel1Layout.setVerticalGroup(

jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabel1)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.
BASELINE)
```

```
                .addComponent(jRBDificultadFacil)
                .addComponent(jRBDificultadMedio)
                .addComponent(jRBDificultadDificil))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jButTablero)
                        .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );

        jLabCosto.setBackground(new java.awt.Color(51, 255, 255));
        jLabCosto.setOpaque(true);

        jPanel2.setBackground(new java.awt.Color(0, 204, 204));

        jLabel2.setFont(new java.awt.Font("Cantarell", 1, 18)); // NOI18N
        jLabel2.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        jLabel2.setText("Velocidad");

        buttonGroup2.add(jRBVelocidadLento);
        jRBVelocidadLento.setSelected(true);
        jRBVelocidadLento.setText("Lento");

        buttonGroup2.add(jRBVelocidadMedio);
        jRBVelocidadMedio.setText("Medio");

        buttonGroup2.add(jRBVelocidadRapido);
        jRBVelocidadRapido.setText("Rápido");

        jButIniciar.setText("Iniciar");
        jButIniciar.setEnabled(false);
        jButIniciar.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButIniciarActionPerformed(evt);
```

```
        }
    });

                                javax.swing.GroupLayout jPanel2Layout = new
javax.swing.GroupLayout(jPanel2);
    jPanel2.setLayout(jPanel2Layout);
    jPanel2Layout.setHorizontalGroup(

jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(jLabel2,  javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
        .addGroup(jPanel2Layout.createSequentialGroup()

.addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.
LEADING)
            .addGroup(jPanel2Layout.createSequentialGroup()
                                                .addComponent(jRBVelocidadLento,
javax.swing.GroupLayout.PREFERRED_SIZE,                                 68,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                                .addComponent(jRBVelocidadMedio,
javax.swing.GroupLayout.PREFERRED_SIZE,                                 75,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                                                .addComponent(jRBVelocidadRapido,
javax.swing.GroupLayout.PREFERRED_SIZE,                                 75,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGroup(jPanel2Layout.createSequentialGroup()
                .addContainerGap()
                                                        .addComponent(jButIniciar,
javax.swing.GroupLayout.DEFAULT_SIZE,
```

```
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)))
                .addContainerGap())
        );
        jPanel2Layout.setVerticalGroup(

jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel2Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabel2)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.
BASELINE)
                    .addComponent(jRBVelocidadLento)
                    .addComponent(jRBVelocidadMedio)
                    .addComponent(jRBVelocidadRapido))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addComponent(jButIniciar)
                .addContainerGap())
        );

                                        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                                .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                .addContainerGap()
                                                        .addComponent(JPTablero,
javax.swing.GroupLayout.PREFERRED_SIZE,
```

```
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADIN
G, false)
                    .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addComponent(jPanel2, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addComponent(jLabCosto, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
            .addContainerGap(9, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADIN
G)
                .addGroup(layout.createSequentialGroup()
                                                    .addComponent(jPanel1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                                    .addComponent(jPanel2,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                                .addComponent(jLabCosto,
javax.swing.GroupLayout.PREFERRED_SIZE,                                 20,
javax.swing.GroupLayout.PREFERRED_SIZE))
                                                .addComponent(JPTablero,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addContainerGap(14, Short.MAX_VALUE))
        );

        pack();
    }// </editor-fold>
        //  Botón para generar el tablero
    private void jButTableroActionPerformed(java.awt.event.ActionEvent evt) {

        // Selección del modo de juegoFACIL, MEDIO o DIFICIL
        if(jRBDificultadFacil.isSelected()){
            cant = 10 + 2;
        }else if(jRBDificultadMedio.isSelected()){
            cant = 15 + 2;
        }else if(jRBDificultadDificil.isSelected()){
           cant = 20 + 2;
        }

        // Se dibuja el tablero
        Dibujo.dibujarTablero(JPTablero);

        // y se dimensiona el arreglo de las posiciones en el tablero
        evil = new int[cant][2];

        // se generan los números aleatorios, para las posiciones en el tablero
        Dibujo.aleatorios(evil, cant);
        // inicio
```

```
x = evil[0][0];
y = evil[0][1];
// fin
xB = evil[1][0];
yB = evil[1][1];

// dibuja el robot (Inicio)
Dibujo.dibujarBOT(x,y,JPTablero);
// y un cuadrado (Fin)
Dibujo.dibujarPila(xB, yB,JPTablero);

// Esyablecemos el bloqueo en el tablero
for(int i=2;i<(cant-2)/2;i++){
    int row = evil[i][0];
    int col = evil[i][1];
    Dibujo.dibujarBloqueo(row, col, JPTablero);
}
// También tiramos en el tablero la tierra y el agua,
// para impedir el buen funcionamiento del bot
for(int i=(cant-2)/2;i<cant-2;i++){
    // fila y columna en donde se colocara el agua
    int row = evil[i][0];
    int col = evil[i][1];
    // se dibuja
    Dibujo.dibujarAgua(row, col, JPTablero);

    i++;
    // fila y columna en donde se colocara la tierra
    row = evil[i][0];
    col = evil[i][1];
    // se dibuja
    Dibujo.dibujarTierra(row, col, JPTablero);
}
// Una vez establecido el tablero, podemos iniciar el recorrido
```

```
    jButIniciar.setEnabled(true);
}

private void jButIniciarActionPerformed(java.awt.event.ActionEvent evt) {
    // inicio de la ruta
    x = evil[0][0];
    y = evil[0][1];
    int nx = dePixelesACoordenadas(x);
    int ny = dePixelesACoordenadas(y);

    // fin de la ruta
    int nxFin = dePixelesACoordenadas(evil[1][0]);
    int nyFin = dePixelesACoordenadas(evil[1][1]);

    // creamos el la ruta a segur y el tablero
    path = new ArrayList<>();
    grid = new Node[GRID_ROWS][GRID_COLS];

    // heuristica
    graph = new Graph<>((start, target, current) -> {
        // heuristic = linear distance
        int dx = target.getObj().x - current.getObj().x;
        int dy = target.getObj().y - current.getObj().y;
        return Math.sqrt(dx * dx + dy * dy);
    });
    // se crea el tablero
    createGrid();

    // y se ingresan los obstaculos
    // Nota: 0 y 1; es el inicio y el fin del recorrido, por eso inicia en 2
    // Agregamos el bloqueo en el tablero
    for(int i=2;i<(cant-2)/2;i++){
       int row = dePixelesACoordenadas(evil[i][0]);
        int col = dePixelesACoordenadas(evil[i][1]);
```

```
    grid[col][row].setBlocked(true);
}
//  También la tierra y agua
for(int i=(cant-2)/2;i<cant-2;i++){
    // Agua
    int row = dePixelesACoordenadas(evil[i][0]);
    int col = dePixelesACoordenadas(evil[i][1]);
    grid[col][row].setAgua(true);
    i++;

    // Tierra
    row = dePixelesACoordenadas(evil[i][0]);
    col = dePixelesACoordenadas(evil[i][1]);
    grid[col][row].setTierra(true);
}


// punto de partida
startNode = grid[ny][nx];
// punto de llegada
targetNode = grid[nyFin][nxFin];

// se limpia el recorrido
path.clear();
// y se traza
graph.findPath(startNode, targetNode, path);
Graphics2D g = (Graphics2D) JPTablero.getGraphics();

// recorrido en el tablero
graph.getNodes().forEach(node -> {
    drawNode(g, node);
});

// se dibuja el reccorrido
```

```
        drawPath(g);

        // y se establece el costo en un JLabel
        jLabCosto.setText("Costo "+((int) targetNode.getG()));
    }



    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
            //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
         /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
                                                                * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
                                        for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(IAFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(IAFrame.class.getName()).log(java.util.logging.Le
```

```
vel.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(IAFrame.class.getName()).log(java.util.logging.Le
vel.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(IAFrame.class.getName()).log(java.util.logging.Le
vel.SEVERE, null, ex);
        }
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new IAFrame().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JPanel JPTablero;
    private javax.swing.ButtonGroup buttonGroup1;
    private javax.swing.ButtonGroup buttonGroup2;
    private javax.swing.JButton jButIniciar;
    private javax.swing.JButton jButTablero;
    private javax.swing.JLabel jLabCosto;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JRadioButton jRBDificultadDificil;
    private javax.swing.JRadioButton jRBDificultadFacil;
    private javax.swing.JRadioButton jRBDificultadMedio;
```

```
    private javax.swing.JRadioButton jRBVelocidadLento;
    private javax.swing.JRadioButton jRBVelocidadMedio;
    private javax.swing.JRadioButton jRBVelocidadRapido;
    // End of variables declaration


}
```

Heuristic.java

```
package ia;

/**
 * Heuristic functional interface.
 *
 * @author RETBOT
 */
// Clase para calcular la heuristica
public interface Heuristic<T> {
    // se crea un metodo para calcular la heuristica del nodo inicial, con el nodo final
    public double calculate(Node<T> start, Node<T> target, Node<T> current);
}
```

Node.java

```
package ia;

import java.util.ArrayList;
import java.util.List;

/**
 * Node class.
 *
```

```
 * @author RETBOT
 */
// clase para establecer los nodos
public class Node<T> implements Comparable<Node> {

    // estados de los nodos, Visitado, abierto y cerrado
    public static enum State { UNVISITED, OPEN, CLOSED };

    private final T obj;
    private State state = State.UNVISITED;
    private boolean blocked = false;
    private boolean pila = false;
    private boolean pelota = false;

    private double g; // cost
    private double h; // heuristic
                    // f = g + h

    private Node backPathNode;
    private final List<Edge<T>> edges = new ArrayList<>();

    // constructior
    public Node(T obj) {
        this.obj = obj;
    }

    // Método para obtener los objetos
    public T getObj() {
        return obj;
    }

    // Método para obtener el estado
    public State getState() {
        return state;
```

```
}

// Método para poner el estado
void setState(State state) {
    this.state = state;
}

// Método para obtener si el nodo esta bloquedo
public boolean isBlocked() {
    return blocked;
}

// Método para obtener si es pila
public boolean isPila() {
    return pila;
}

// Método para obtener si es pelota
public boolean isPelota() {
    return pelota;
}

// Método para bloquear el nodo
public void setBlocked(boolean blocked) {
    this.blocked = blocked;
}

// Método para poner tierra
public void setPila(boolean pila) {
    this.pila = pila;
}

// Método para poner agua
public void setPelota(boolean pelota) {
```

```
    this.pelota = pelota;
}

// Método para obtener costo
public double getG() {
    return g;
}

// Método para poner costo
void setG(double g) {
    this.g = g;
}

// Método para obtener la heuristica
public double getH() {
    return h;
}

// Método para poner la heuristica
void setH(double h) {
    this.h = h;
}

 // Método para obtener el nodo de la ruta de regreso
public Node getBackPathNode() {
    return backPathNode;
}

// Método para poner el nodo de la ruta de regreso
public void setBackPathNode(Node backPathNode) {
    this.backPathNode = backPathNode;
}

 // Método para obtener la lista de búsquedas
```

```
public List<Edge<T>> getEdges() {
    return edges;
}

//Método para agregar a la lista de búsquedas
public void addEdge(Edge edge) {
    edges.add(edge);
}

// f(n) = g(n) + h(n) -> cost + heuristic
public double getF() {
    return g + h;
}

// Método para recuperar ruta
public void retrievePath(List<Node<T>> path) {
    if (backPathNode != null) {
        backPathNode.retrievePath(path);
    }
    path.add(this);
}

// Método para comparar los nodos
@Override
public int compareTo(Node o) {
    double dif = getF() - o.getF();
    return dif == 0 ? 0 : dif > 0 ? 1 : -1;
}

@Override
public String toString() {
    return "Node{" + "id=" + obj + ", state=" + state + ", g=" + g + ", h="
        + h + ", parentNode=" + backPathNode + ", edges=" + edges + '}';
}
```

```
}
```

Conclusión

Para la realización de este proyecto se separó en dos partes, una para la generación de un recorrido óptimo (y a su vez que este no pasará en diagonal y la otra para lograr hacer un algoritmo que analizará la posible ruta de vida o muerte (hablando del robot con la batería), la implementación de los algoritmos y la heurística fue algo complicada al inicio, ya que no se tenía nociones de este tipo de algoritmos, sin embargo se pudo implementar de una manera sencilla utilizando colores para comprender mejor el funcionamiento al momento de hacer los recorridos, la manera en la que se toman las decisiones está calculada para que tenga éxito en su mayoría por lo que el resultado de la aplicación es satisfactorio.

Referencias

- Programar es increíble. (2 de enero de 2020). ⭐ *Algoritmo A* (estrella) en HTML5 y JavaScript* 🎮 *Tutorial paso a paso* [Video]. YouTube. https://www.youtube.com/watch?v=NWS-_VsMab4

- gammafp. (27 de julio de 2017). *Pathfinding A* (A estrella) - Tutorial completo en español* [Video]. YouTube. https://www.youtube.com/watch?v=X-5JMScsZ14

- Programar es increíble. (26 de diciembre de 2019). 🎮 *Búsqueda de caminos en los Videojuegos: Algoritmo A* (Estrella)* [Video]. YouTube. https://www.youtube.com/watch?v=1gszEk8rUS4

- Julio Sanchez. (15 de septiembre de 2012). *Algoritmos A * (ESTRELLA): Encuentra el Camino Optimo en un mapa* [Video]. YouTube. https://www.youtube.com/watch?v=LBvzyUbl2O4

- Sylvain Saurel. (15 de julio de 2018). *Implementing A Star (A*) Search Algorithm in Java for Pathfinding in Games* [Video]. YouTube. https://www.youtube.com/watch?v=oeT8B8sqbxQ

- Sylvain Saurel. (15 de julio de 2018b). *Implementing A Star (A*) Search Algorithm in Java for Pathfinding in Games* [Video]. YouTube. https://www.youtube.com/watch?v=oeT8B8sqbxQ

- coderodde. (1 de octubre de 2016). *A* and Dijkstra algorithms: full Java video tutorial* [Video]. YouTube. https://www.youtube.com/watch?v=uPXlsSd_wK4

- Computerphile. (15 de febrero de 2017). *A* (A Star) Search Algorithm - Computerphile* [Video]. YouTube. https://www.youtube.com/watch?v=ySN5Wnu88nE

- Leo Ono. (28 de julio de 2020). *[Java 2D / AI] from scratch - A* (A Star) Path Finding Algorithm Test* [Video]. YouTube. https://www.youtube.com/watch?v=oBJbYrfrMBU

- Simplilearn. (22 de julio de 2021). *A* Algorithm in Artificial Intelligence You Must Know in 2022*. Simplilearn.com. https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/a-star-algorithm#:~:text=What%20is%20an%20A*%20Algorithm?%20It%20is%20a,a%20robot%20that%20can%20find%20its%20own%20course.