Desarrollo Basado en Agentes 2023/2024

Grupo 304 – PRÁCTICA 2

Javier Linde Martínez

Ana López Mohedano

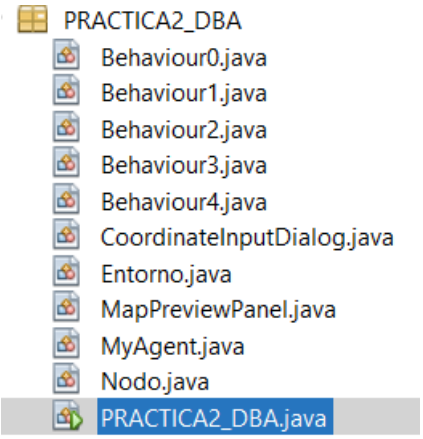
Jesús Palomares Fernández

Luis Miguel Guirado Bautista

ÍNDICE

Estructura de clases:

Encontraremos la siguiente estructura de clases en el programa:



- **PRACTICA2_DBA**: Es nuestra clase principal donde se ejecuta la función main la cual abre el selector de mapas. Esta clase se encarga de leer la carpeta maps y previsualizarlos a través de MapPreviewPanel.
- **MapPreviewPanel**: Es la clase que se encarga de mostrar una previsualización del mapa en la ventana de selección de mapas
- **CoordinateInputDialog**: Una vez el mapa ha sido seleccionado esta clase se encarga de abrir una ventana para introducir las coordenadas de origen y comprobar si son válidas, una vez se ha realizado esto inicializa el agente y el entorno.
- **Nodo**: Clase encargada de la representación de las casillas almacenan las coordenadas y la distancia Manhattan al nodo destino.
- **Entorno**: Clase encargada de la gestión del mapa y el agente, le devuelve al agente sus casillas adyacentes y registra la posición actual del agente también imprime el estado de la ejecución (en texto)
- **MyAgent**: Se trata del agente una vez este inicia abrirá una ventana donde te permitirá elegir el comportamiento que va a usar en la ejecución además de las variables necesarias para cada comportamiento implementa métodos set y get de estas
- **Behaviour4**: Es un comportamiento de prueba para comprobar el correcto funcionamiento de la selección de comportamientos implementada en el agente
- **Behaviour3**: Comportamiento inicial ya llega al objetivo, va recorriendo las casillas de menor distancia al destino y en caso de no encontrar el destino y haber visitado todas las adyacentes o que estas sean muros vuelve hacia atrás hasta la última casilla que dejo sin recorrer
- **Behaviour2**: Funciona igual que behaviour3 pero en caso de que la distancia al destino en 2 adyacentes sea la misma elegirá la que tenga menor distancia euclidea con el destino
- **Behaviour1**: Funciona como behaviour2 pero, tiene en cuenta cuando se pasa a una casilla que tiene mayor distancia que en la que estábamos antes de desplazarnos a la siguiente, es decir, el agente ha encontrado un muro donde estaría la casilla con menor distancia en caso de que recorramos (n * 2) - 2 casillas siendo n la mayor entre las filas y las columnas del entorno, ese valor es el máximo de pasos en un mapa sin obstáculos desde una esquina a otra, consideramos que el camino no es el correcto en el caso de que el contador que monitorea este estado llegue a este umbral y

retrocedemos (n * 2) - 2 para coger la otra casilla que habría disponible cuando elegimos la casilla con heurística mayor. Ya que ese camino tiene más posibilidades de ser el correcto, también en el caso de que mientras contamos desde que llegamos a una mayor, pasáramos n veces casillas con menor distancia podría significar que sí que estamos en un camino correcto, luego reiniciamos el contador de pisar mayores a 0 para que siga por ese camino.

- **Behaviour0**: Funciona igual que Behaviour1, pero no tiene en cuenta la distancia euclídea para elegir el siguiente nodo.

Dificultades durante la implementación:

- **Reinicio del proyecto:**
  El proyecto comenzó muy pronto y todos nos coordinamos para generar una interfaz de texto y un comportamiento que funcionará (que fue Behaviour3), pero, al finalizar todo esto rápido y ver los proyectos de otros compañeros por intentar tener algo mejor reiniciamos el proyecto a una semana de la entrega por lo que con las entregas de otras asignaturas el tiempo se nos echó encima.
- **Clase entorno:**
  La interfaz pasó a tener un selector de mapas, un selector de coordenadas y un selector de comportamientos, tras todas estas ventanas debería comenzar una simulación con una IU para visualizarla en la clase entorno, pero, debido a falta de tiempo nos quedamos al final con una interfaz de texto en la simulación.
- **Falta de tiempo:**
  Hasta que conseguimos llegar a los comportamientos Behaviour0 y Behaviour1 nos tomó mucho tiempo resolver errores de la idea que teníamos con estos llegándonos a tomar hasta el día final de la entrega (hoy) para conseguirlos, esto junto con una interfaz bastante más compleja nos ha supuesto mucha carga y nos ha llevado a un desarrollo exprés muy cargante.

Comportamientos: Recomendamos leer los códigos en el orden del 4 al 0 ya que explican iterativamente cómo funciona el comportamiento Behaviour0 al final (qué es el comportamiento resultado).

- **Behaviour4:**
- **Código fuente:**

```
- import jade.core.behaviours.CyclicBehaviour;
- import jade.core.behaviours.OneShotBehaviour;
-
- // Agente de prueba para el selector de comportamientos
- public class Behaviour4 extends OneShotBehaviour {
-     public String name = "Stuart";
-     public String descripcion = "Le da miedo andar como lo dejes
  solo chilla de miedo y se auto elimina.";
-     protected MyAgent agente;
-
-     Behaviour4(MyAgent a) {
-         this.myAgent = agente = a;
-     }
-
-     @Override
```

```
-       public void action() {
-           System.out.println("ME DA MIEDO IR SOLO NO PUEDO MÁS (SE
    TIRA POR UN BARRANCO)\n");
-           agente.doDelete();
-       }
-
-   }
```

- **Behaviour3:**
- **Código fuente:**

```
-
-   import jade.core.behaviours.Behaviour;
-   import jade.core.behaviours.CyclicBehaviour;
-   import java.util.ArrayList;
-   import java.util.Comparator;
-   import java.util.List;
-   import java.util.PriorityQueue;
-
-   public class Behaviour3 extends CyclicBehaviour {
-       public String name = "Rayito";
-       public String descripcion = "Intenta ir de casilla en casilla a
    la siguiente que esté más cerca aunque a veces se pierde sabe
    volver sobre sus pasos.";
-       protected MyAgent agente;
-
-       Behaviour3(MyAgent a) {
-           this.myAgent = agente = a;
-       }
-
-       @Override
-       public void action() {
-           agente.getEntorno().mostrarEstado();
-           System.out.println("CONTADOR: " +
    agente.getContadorHeuristicaMayor() + " \n");
-           mover();
-           if
    (agente.getEntorno().agente_actual.equals(agente.getEntorno().desti
    no)) {
-               System.out.println("¡El agente ha alcanzado el
    objetivo!");
-               agente.doDelete();
-           }
-           try {
-               Thread.sleep(200);
-           } catch (InterruptedException e) {
-               e.printStackTrace();
-           }
-       }
-
```

```
-       // Nos desplazamos al nodo con menor distancia manhattan
-       public void mover() {
-           List<Nodo> adyacentes =
    agente.getEntorno().obtenerNodosAdyacentes();
-           PriorityQueue<Nodo> colaPrioridad = new
    PriorityQueue<>(Comparator
-                   .comparingInt((Nodo n) -> n.heu));
-           colaPrioridad.addAll(adyacentes);
-           colaPrioridad.removeAll(agente.getVisitedNodos());
-
-           Nodo siguienteNodo = colaPrioridad.poll();
-
-           if (siguienteNodo != null && siguienteNodo.val != -1) {
-               moverAlNodo(siguienteNodo);
-               // En caso de que no queden pasos disponibles (todos
    visitados o muros)
-               // retrocedemos en pasos hasta llegar
-               // a una casilla libre (siguienteNodo dejará de ser
    null)
-           } else {
-               regresarSobrePasos();
-           }
-       }
-
-       private void regresarSobrePasos() {
-           regresarSobrePasos(1);
-       }
-
-       // Regresamos un paso atrás utilizando VisitedNodos
-       private void regresarSobrePasos(int pasos) {
-           System.out.println("REGRESAMOS sobre " + pasos + "
    pasos\n");
-           for (int i = 0; i < pasos; i++) {
-               Nodo ultimoNodoVisitado =
    agente.getVisitedNodos().get(agente.getVisitedNodos().size() - 2);
-               agente.addVisitedNodo(0,
    agente.getVisitedNodos().get(agente.getVisitedNodos().size() - 1));
-               agente.removeVisitedNodo(agente.getVisitedNodos().size(
    ) - 1);
-               agente.setOrigen(ultimoNodoVisitado);
-               agente.getEntorno().setAgenteActual(ultimoNodoVisitado)
    ;
-               agente.plusIters();
-               agente.getEntorno().mostrarEstado();
-           }
-       }
-
-       // Función encargada de actualizar los valores del agente y el
    entorno
```

```
-       private void moverAlNodo(Nodo siguienteNodo) {
-           agente.setOrigen(siguienteNodo);
-           agente.getEntorno().setAgenteActual(siguienteNodo);
-           agente.addVisitedNodo(siguienteNodo);
-           agente.plusIters();
-       }
-   }
```

- **Behaviour2:**
- **Código fuente:**

```
-   import jade.core.behaviours.Behaviour;
-   import jade.core.behaviours.CyclicBehaviour;
-   import java.util.ArrayList;
-   import java.util.Comparator;
-   import java.util.List;
-   import java.util.PriorityQueue;
-   public class Behaviour2 extends CyclicBehaviour {
-       public String name = "Rayo";
-       public String descripcion = "El padre de rayo dicen que se fue
    a por tabaco usando la distancia euclídea en lugar de la Manhattan
    y nunca volvió a casa";
-       protected MyAgent agente;
-
-       Behaviour2(MyAgent a) {
-           this.myAgent = agente = a;
-       }
-
-       @Override
-       public void action() {
-           agente.getEntorno().mostrarEstado();
-           System.out.println("CONTADOR: " +
    agente.getContadorHeuristicaMayor() + " \n");
-           mover();
-           if
    (agente.getEntorno().agente_actual.equals(agente.getEntorno().desti
    no)) {
-               System.out.println("¡El agente ha alcanzado el
    objetivo!");
-               agente.doDelete();
-           }
-           try {
-               Thread.sleep(200);
-           } catch (InterruptedException e) {
-               e.printStackTrace();
-           }
-       }
-
-       private double calcularDistanciaEuclidiana(Nodo nodo1, Nodo
    nodo2) {
```

```
-           int deltaX = nodo1.x - nodo2.x;
-           int deltaY = nodo1.y - nodo2.y;
-           return Math.sqrt(deltaX * deltaX + deltaY * deltaY);
-       }
-
-       // Funciona igual que Behaviour3 pero en caso de que haya 2
    casillas con menor
-       // distancia igual
-       // se obtendrá la casilla que tenga menor distancia euclidea
-       public void mover() {
-           List<Nodo> adyacentes =
    agente.getEntorno().obtenerNodosAdyacentes();
-           PriorityQueue<Nodo> colaPrioridad = new
    PriorityQueue<>(Comparator
-                   .comparingInt((Nodo n) -> n.heu)
-                   .thenComparingDouble(n ->
    calcularDistanciaEuclidiana(n, agente.getEntorno().destino)));
-           colaPrioridad.addAll(adyacentes);
-           colaPrioridad.removeAll(agente.getVisitedNodos());
-
-           Nodo siguienteNodo = colaPrioridad.poll();
-
-           if (siguienteNodo != null && siguienteNodo.val != -1) {
-               moverAlNodo(siguienteNodo);
-           } else {
-               regresarSobrePasos();
-           }
-       }
-
-       private void regresarSobrePasos() {
-           regresarSobrePasos(1);
-       }
-
-       private void regresarSobrePasos(int pasos) {
-           System.out.println("REGRESAMOS sobre " + pasos + "
    pasos\n");
-           for (int i = 0; i < pasos; i++) {
-               Nodo ultimoNodoVisitado =
    agente.getVisitedNodos().get(agente.getVisitedNodos().size() - 2);
-               agente.addVisitedNodo(0,
    agente.getVisitedNodos().get(agente.getVisitedNodos().size() - 1));
-               agente.removeVisitedNodo(agente.getVisitedNodos().size(
    ) - 1);
-               agente.setOrigen(ultimoNodoVisitado);
-               agente.getEntorno().setAgenteActual(ultimoNodoVisitado)
    ;
-               agente.plusIters();
-               agente.getEntorno().mostrarEstado();
-           }
```

```
            // agente.setVisitedNodos((ArrayList<Nodo>) visitedaux);
        }

        private void moverAlNodo(Nodo siguienteNodo) {
            agente.setOrigen(siguienteNodo);
            agente.getEntorno().setAgenteActual(siguienteNodo);
            agente.addVisitedNodo(siguienteNodo);
            agente.plusIters();
        }
    }
```

- **Behaviour1:**
- **Código fuente:**

```
    import jade.core.behaviours.Behaviour;
    import jade.core.behaviours.CyclicBehaviour;
    import java.util.ArrayList;
    import java.util.Comparator;
    import java.util.List;
    import java.util.PriorityQueue;

    public class Behaviour1 extends CyclicBehaviour {
        public String name = "Reglitas";
        public String descripcion = "Siempre encontrará el camino,
    cuanto más compleja la situación menos tardará aunque es cierto que
    a veces le gusta complicarse la vida.";
        protected MyAgent agente;

        Behaviour1(MyAgent a) {
            this.myAgent = agente = a;
        }

        @Override
        public void action() {
            agente.getEntorno().mostrarEstado();
            System.out.println("CONTADOR: " +
    agente.getContadorHeuristicaMayor() + " \n");
            mover();
            if
    (agente.getEntorno().agente_actual.equals(agente.getEntorno().desti
    no)) {
                System.out.println("¡El agente ha alcanzado el
    objetivo!");
                agente.doDelete();
            }
            try {
                Thread.sleep(200);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
```

```
-       }
-
-       private double calcularDistanciaEuclidiana(Nodo nodo1, Nodo
    nodo2) {
-           int deltaX = nodo1.x - nodo2.x;
-           int deltaY = nodo1.y - nodo2.y;
-           return Math.sqrt(deltaX * deltaX + deltaY * deltaY);
-       }
-
-       // Funciona como behaviour2 pero tiene en cuenta cuando se pasa
    a una casilla
-       // que tiene mayor distancia que en la que estabamos
-       // Es decir, el agente ha encontrado un muro donde estaría la
    casilla con menor
-       // distancia
-       // En caso de que recorramos (n * 2) - 2 siendo n la mayor
    entre las filas y las
-       // columnas del entorno
-       // ese valor es el máximo de pasos en un mapa sin óbtaculos
    desde una esquina a
-       // otra consideramos que el camino no es el correcto
-       // y retrocedemos (n * 2) - 2 para coger la otra casilla que
    habría disponible
-       // ya que tiene más posibilidades de ser el camino
-       // correcto. También en el caso de que mientras contamos desde
    que llegamos a
-       // una mayor pisamos n veces casillas con menor distancia
-       // podría significar que sí que estamos en un camino correcto
    luego reiniciamos
-       // el contador de pisar mayores a 0 para que siga por
-       // ese camino
-       public void mover() {
-           List<Nodo> adyacentes =
    agente.getEntorno().obtenerNodosAdyacentes();
-           PriorityQueue<Nodo> colaPrioridad = new
    PriorityQueue<>(Comparator
-                   .comparingInt((Nodo n) -> n.heu)
-                   .thenComparingDouble(n ->
    calcularDistanciaEuclidiana(n, agente.getEntorno().destino)));
-           colaPrioridad.addAll(adyacentes);
-           colaPrioridad.removeAll(agente.getVisitedNodos());
-
-           Nodo siguienteNodo = colaPrioridad.poll();
-
-           if (siguienteNodo != null && siguienteNodo.val != -1 &&
    !agente.getVisitedNodos().contains(siguienteNodo)) {
-               if (!agente.isCuenta()) { // No estamos contando
-                   if (siguienteNodo.heu > agente.getOrigen().heu) {
    // Pasamos a una casilla con mayor heu
```

```
-                        agente.setCuenta(true); // Comenzamos a contar
-                        agente.setMayores(agente.getContadorHeuristicaM
    ayor() + 1);
-                        int pasos =
    agente.getContadorHeuristicaMayor();
-                        moverAlNodo(siguienteNodo);
-
-                        if (agente.isCuenta() && pasos >=
    (agente.getEntorno().getTamEntorno()) * 2 - 2) { // Llegamos al
-
                                        // umbral
-                            System.out.println("Volver sobre " + pasos
    + " pasos"); // Volvemos sobre nuestros pasos
-                            regresarSobrePasos(pasos);
-                            agente.setCuenta(false);
-                            agente.setMayores(0);
-                        }
-                    } else {
-                        moverAlNodo(siguienteNodo);
-                    }
-                } else { // Si estamos contando
-                    if (siguienteNodo.heu < agente.getOrigen().heu) {
    // Contamos cuantas veces caemos en menores heu
-                        agente.setMenores(agente.getMenores() + 1);
-                    }
-                    if (agente.getMenores() <
    agente.getEntorno().getTamEntorno()) { // Si llegamos al umbral de
    menores
-                        agente.setMayores(agente.getContadorHeuristicaM
    ayor() + 1); // Reiniciamos la cuenta de mayores
-                        int pasos =
    agente.getContadorHeuristicaMayor();
-                        moverAlNodo(siguienteNodo);
-
-                        if (agente.isCuenta() && pasos >=
    (agente.getEntorno().getTamEntorno() * 2) - 2) { // Si llegamos al
-
                                        // umbral de
-
                                        // mayores
-                            System.out.println("Volver sobre " + pasos
    + " pasos"); // Volvemos sobre nuestros pasos
-                            regresarSobrePasos(pasos);
-                            agente.setCuenta(false);
-                            agente.setMayores(0);
-                            agente.setMenores(0);
-                        }
-                    } else {
-                        agente.setCuenta(false);
```

```
                    agente.setMayores(0);
                    agente.setMenores(0);
                }
            }
        } else { // No hay movimientos disponibles
            regresarSobrePasos();
            agente.setMayores(0);
        }
    }

    private void regresarSobrePasos() {
        regresarSobrePasos(1);
    }

    private void regresarSobrePasos(int pasos) {
        System.out.println("REGRESAMOS sobre " + pasos + "
pasos\n");
        for (int i = 0; i < pasos; i++) {
            Nodo ultimoNodoVisitado =
agente.getVisitedNodos().get(agente.getVisitedNodos().size() - 2);
            agente.addVisitedNodo(0,
agente.getVisitedNodos().get(agente.getVisitedNodos().size() - 1));
            agente.removeVisitedNodo(agente.getVisitedNodos().size(
) - 1);
            agente.setOrigen(ultimoNodoVisitado);
            agente.getEntorno().setAgenteActual(ultimoNodoVisitado)
;
            agente.plusIters();
            agente.getEntorno().mostrarEstado();
        }
    }

    private void moverAlNodo(Nodo siguienteNodo) {
        agente.setOrigen(siguienteNodo);
        agente.getEntorno().setAgenteActual(siguienteNodo);
        agente.addVisitedNodo(siguienteNodo);
        agente.plusIters();
    }
}
```

- **Behaviour0:**
- **Código fuente:**

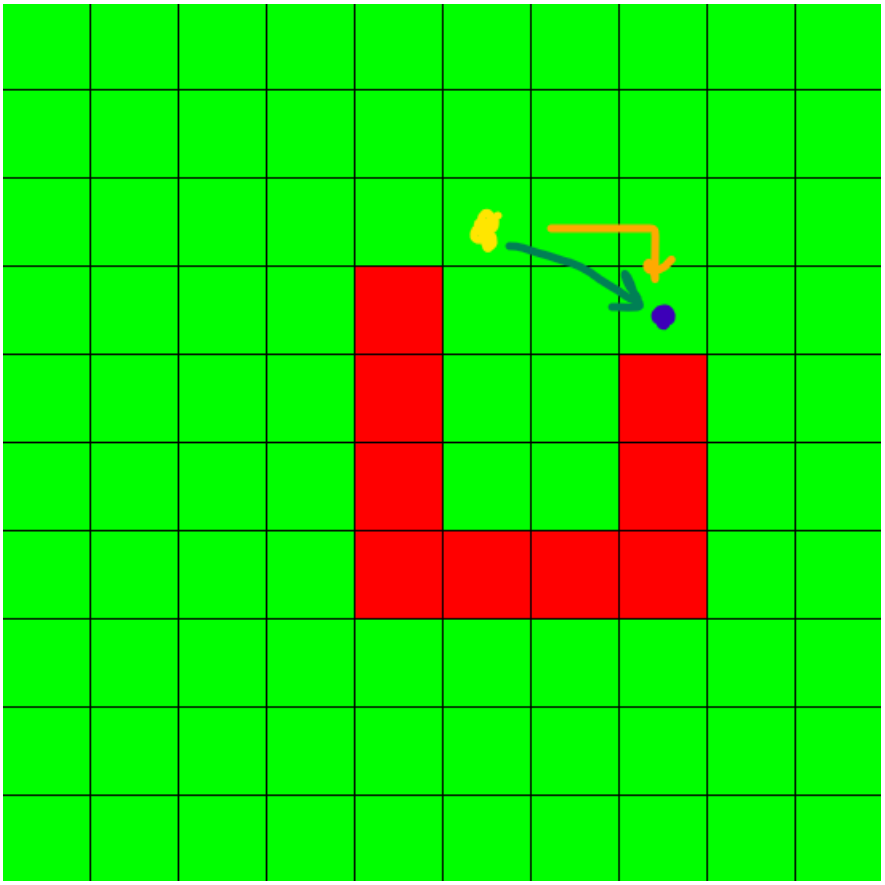
```
import jade.core.behaviours.Behaviour;
import jade.core.behaviours.CyclicBehaviour;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.PriorityQueue;

```

```
-   public class Behaviour0 extends CyclicBehaviour {
-       public String name = "Reglitas americano";
-       public String descripcion = "Como Reglitas pero americano (solo
    usa la Manhattan).";
-       protected MyAgent agente;
-
-       Behaviour0(MyAgent a) {
-           this.myAgent = agente = a;
-       }
-
-       @Override
-       public void action() {
-           agente.getEntorno().mostrarEstado();
-           System.out.println("CONTADOR: " +
    agente.getContadorHeuristicaMayor() + " \n");
-           mover();
-           if
    (agente.getEntorno().agente_actual.equals(agente.getEntorno().desti
    no)) {
-               System.out.println("¡El agente ha alcanzado el
    objetivo!");
-               agente.doDelete();
-           }
-           try {
-               Thread.sleep(200);
-           } catch (InterruptedException e) {
-               e.printStackTrace();
-           }
-       }
-
-       // Exactamente igual que Behaviour 0 pero no tiene en cuenta la
    distancia
-       // euclidea para elegir
-       // el siguiente nodo
-       public void mover() {
-           List<Nodo> adyacentes =
    agente.getEntorno().obtenerNodosAdyacentes();
-           PriorityQueue<Nodo> colaPrioridad = new
    PriorityQueue<>(Comparator
-                   .comparingInt((Nodo n) -> n.heu));
-           colaPrioridad.addAll(adyacentes);
-           colaPrioridad.removeAll(agente.getVisitedNodos());
-
-           Nodo siguienteNodo = colaPrioridad.poll();
-
-           if (siguienteNodo != null && siguienteNodo.val != -1 &&
    !agente.getVisitedNodos().contains(siguienteNodo)) {
-               if (!agente.isCuenta()) {
-                   if (siguienteNodo.heu > agente.getOrigen().heu) {
```

```
-                       agente.setCuenta(true);
-                       agente.setMayores(agente.getContadorHeuristicaM
    ayor() + 1);
-                       int pasos =
    agente.getContadorHeuristicaMayor();
-                       moverAlNodo(siguienteNodo);
-
-                       if (agente.isCuenta() && pasos >=
    (agente.getEntorno().getTamEntorno()) * 2 - 2) {
-                           System.out.println("Volver sobre " + pasos
    + " pasos");
-                           regresarSobrePasos(pasos);
-                           agente.setCuenta(false);
-                           agente.setMayores(0);
-                       }
-                   } else {
-                       moverAlNodo(siguienteNodo);
-                   }
-               } else {
-                   if (siguienteNodo.heu < agente.getOrigen().heu) {
-                       agente.setMenores(agente.getMenores() + 1);
-                   }
-                   if (agente.getMenores() < 10) {
-                       agente.setMayores(agente.getContadorHeuristicaM
    ayor() + 1);
-                       int pasos =
    agente.getContadorHeuristicaMayor();
-                       moverAlNodo(siguienteNodo);
-
-                       if (agente.isCuenta() && pasos >=
    (agente.getEntorno().getTamEntorno() * 2) - 2) {
-                           System.out.println("Volver sobre " + pasos
    + " pasos");
-                           regresarSobrePasos(pasos);
-                           agente.setCuenta(false);
-                           agente.setMayores(0);
-                           agente.setMenores(0);
-                       }
-                   } else {
-                       agente.setCuenta(false);
-                       agente.setMayores(0);
-                       agente.setMenores(0);
-                   }
-               }
-           } else {
-               regresarSobrePasos();
-               agente.setMayores(0);
-           }
-       }
```

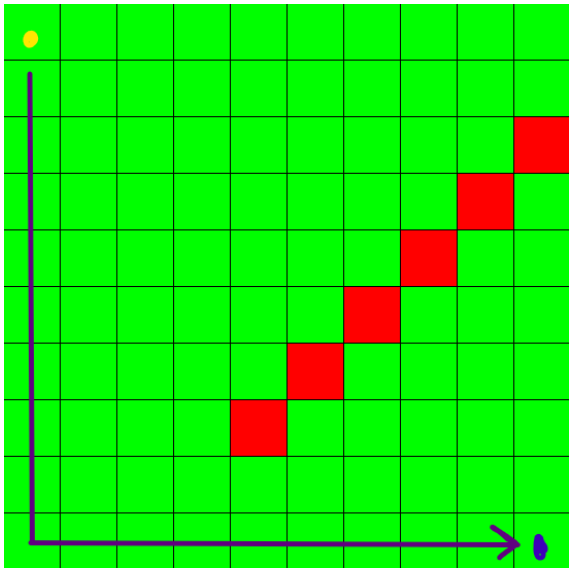
```
-
-       private void regresarSobrePasos() {
-           regresarSobrePasos(1);
-       }
-
-       private void regresarSobrePasos(int pasos) {
-           System.out.println("REGRESAMOS sobre " + pasos + "
    pasos\n");
-           for (int i = 0; i < pasos; i++) {
-               Nodo ultimoNodoVisitado =
    agente.getVisitedNodos().get(agente.getVisitedNodos().size() - 2);
-               agente.addVisitedNodo(0,
    agente.getVisitedNodos().get(agente.getVisitedNodos().size() - 1));
-               agente.removeVisitedNodo(agente.getVisitedNodos().size(
    ) - 1);
-               agente.setOrigen(ultimoNodoVisitado);
-               agente.getEntorno().setAgenteActual(ultimoNodoVisitado)
    ;
-               agente.plusIters();
-               agente.getEntorno().mostrarEstado();
-           }
-       }
-
-       private void moverAlNodo(Nodo siguienteNodo) {
-           agente.setOrigen(siguienteNodo);
-           agente.getEntorno().setAgenteActual(siguienteNodo);
-           agente.addVisitedNodo(siguienteNodo);
-           agente.plusIters();
-       }
-   }
```

- **Diferencias entre distancia Manhattan y Euclídea:**
- La diferencia entra la distancia euclídea y la distancia Manhattan, es que la distancia Manhattan es a cuantas casillas está una casilla del destino y la euclídea es a cuantas unidades en una línea recta esta la casilla del destino. Si solo se tiene en cuenta la distancia Manhattan puede haber casos donde 2 casillas puedan ser seleccionables y el agente elegirá la primerá que le devuelva la cola con prioridad, si combinamos con la euclídea el agente elegiría la que tenga menor euclídea entre las casillas con misma manhattan es una manera de que no nos arriesguemos al orden de como se guardan los posibles movimientos la decisión del agente, sin embargo, hay casos donde solo usar la Manhattan nos ha beneficiado por eso la existencia de Behaviour0 y Behaviour1 ya que ambos pueden dar mejores soluciones respecto al otro según la situación.



En amarillo el agente, en azul el destino, la distancia Manhattan se trata de la flecha naranja y la euclídea de la fleche verde oscura.

- **Razonamiento de umbral de mayores y umbral de menores:**
- Tras una serie de intentos de mejorar los comportamientos Behaviour2 y Behaviour3 uno de los miembros del grupo propusó una idea si tenemos un mapa de tamaño cuadrado n, el camino más corto desde una esquina a otra del mapa es de (n * 2) – 2 casillas si no nos podemos mover en diagonal:

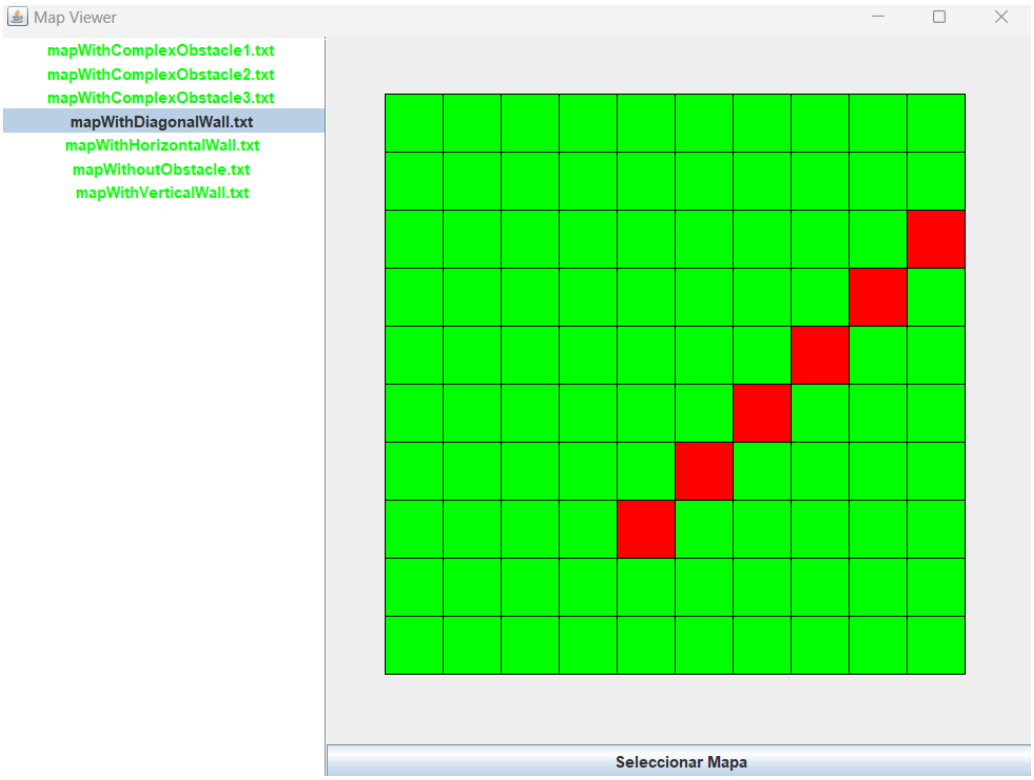


*Podemos ver que el camino más corto de una esquina a otra en un mapa de 10*10 es de 18 casillas*

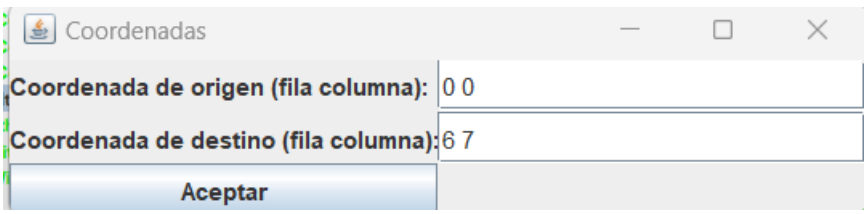
Es decir, lo coherente en un camino hacia al destino es que el número de pasos ronde esta cifra luego con la distancia Manhattan si no encontramos ningún óbstaculo esta

condición se cumple con facilidad y se puede comprobar en que durante todo el camino las casillas a las que se ha movido el agente tenían menor distancia al destino que en la que estaba previamente, es decir, en el caso de que pisemos una casilla con una distancia mayor a la actual significa que nos hemos topado con un óbstaculo y hemos elegido un camino que no es seguro que nos lleve al destino. Por lo tanto si una vez pisamos este óbstaculo nos movemos la cifra del umbral el agente volverá hacia atrás y tomará otro de los caminos que había disponibles cuando se cruzó con el obstáculo. Pero también hay una situación en la que quizás pudieramos alcanzar la distancia del umbral y que sí que se tratará del camino correcto para esto tenemos el contador menores en el agente, el contador menores cuenta cuantas casillas siguientes con menor distancia al destino que la actual hemos pisado desde que el agente comenzó a contar cuando se topó con un obstáculo, si encontramos n casillas así significa que nos estamos acercando aunque sea en muchos pasos (quizás sea un laberinto de obstáculos) al objetivo y reiniciaremos el contador mayores para seguir por dicho camino. Este razonamiento ha logrado recortar algunos pasos para el recorrido de mapas complejos respecto a los comportamientos que solo tienen en cuenta la distancia.

Ejemplo de ejecución del agente:



Selección de mapa



Selección de coordenadas

Selección de comportamiento a usar

```
Posición del agente: (8, 5)
Posición del objetivo: (6, 7)
Estado del mapa:
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 X
0 0 0 0 0 0 0 0 X 0
0 0 0 0 0 0 0 X 0 0
0 0 0 0 0 0 X 0 0 0
0 0 0 0 0 X 0 D 0 0
0 0 0 0 X 0 0 0 0 0
0 0 0 0 0 A 0 0 0 0
0 0 0 0 0 0 0 0 0 0
-------------
PASOS: 39
```

Instancia simulación

```
Posición del agente: (6, 6)
Posición del objetivo: (6, 7)
Estado del mapa:
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 X
0 0 0 0 0 0 0 0 X 0
0 0 0 0 0 0 0 X 0 0
0 0 0 0 0 0 X 0 0 0
0 0 0 0 0 X A D 0 0
0 0 0 0 X 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
-------------
PASOS: 63
```

Fin simulación

Capturas de pasos realizados por cada comportamiento en un mapa tipo:

El mapa será el mapa con muro diagonal el origen 0, 0 y el destino 6, 7.

- **Reglitas americano: (0)**

```
Posición del agente: (6, 6)
Posición del objetivo: (6, 7)
Estado del mapa:
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 X
0 0 0 0 0 0 0 0 X 0
0 0 0 0 0 0 0 X 0 0
0 0 0 0 0 0 X 0 0 0
0 0 0 0 0 X A D 0 0
0 0 0 0 X 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
-------------
PASOS: 63
```

- **Reglitas: (1)**

```
Posición del agente: (6, 6)
Posición del objetivo: (6, 7)
Estado del mapa:
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 X
0 0 0 0 0 0 0 0 X 0
0 0 0 0 0 0 0 X 0 0
0 0 0 0 0 0 X 0 0 0
0 0 0 0 0 X A D 0 0
0 0 0 0 X 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
-------------
PASOS: 28
```

- **Rayo: (2)**

```
Posición del agente: (6, 6)
Posición del objetivo: (6, 7)
Estado del mapa:
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 X
0 0 0 0 0 0 0 0 X 0
0 0 0 0 0 0 0 X 0 0
0 0 0 0 0 0 X 0 0 0
0 0 0 0 0 X A D 0 0
0 0 0 0 X 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
-------------
PASOS: 27
```

- **Rayito: (3)**

```
Posición del agente: (6, 6)
Posición del objetivo: (6, 7)
Estado del mapa:
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 X
0 0 0 0 0 0 0 0 X 0
0 0 0 0 0 0 0 X 0 0
0 0 0 0 0 0 X 0 0 0
0 0 0 0 0 X A D 0 0
0 0 0 0 X 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
-------------
PASOS: 155
```

- Podemos ver que este es un claro ejemplo donde el uso de la distancia euclídea nos benefecia bastante, pero a veces puede suceder el caso contrario.

Reparto de trabajo:

- **Ana López Mohedano y Jesús Palomares Fernández:** Behaviour2 y Behaviour3, además de ayudar en el desarrollo de los sucesivos (0 y 1).
- **Javier Linde Martínez:** Behaviour0 y Behaviour1 además de selector de mapas y de behaviours.
- **Luis Miguel Guirado Bautista:** Documentación, clases entorno y nodo y ayuda en el desarrollo de Behaviour0 y Behaviour1.