INFORME IA

Inteligencia Artificial
12/11/2015 Grado en Ingeniería Informática

Eric Ramos Suárez eric.ramos.suarez@gmail.com
Lucas Ruíz González alu0100785265@ull.etsii.es
Germán Paz Méndez gcpmendez@gmail.com

Índice:

1. Introducción

Procederemos a diseñar un sistema de agente inteligente que pueda resolver de manera virtual un problema. Situaremos en el planeta rojo “N” robots los cuales tendrán que ir esquivando los obstáculos que se les presenten. Tendrán que buscar la existencia de agua, para ello evitarán todo tipo de obstáculo e interacciona con los objetos que se le presenten.

2. Percepciones

Los agentes serán distribuidos en sectores proporcionados a la matriz asignada, podrán elegir entre las cuatro casillas posibles que les rodean (arriba, abajo, izquierda, derecha), existirá un coordinador que distribuirá las acciones que debe realizar cada agente. Podrán detectar la presencia de obstáculos y la presencia de otros agentes, y no podrán interferir en la ruta de los otros, en caso de quedar atrapado y no poder avanzar los agentes podrán destruir un objeto. Con el fin de calcular la ruta mínima.

3. Acciones

Los agentes dispondrán de cuatro movimientos:

- Los agentes podrán moverse en una matriz M x N en todas las direcciones.

- En el caso de que encuentren un obstáculo (borde o cráter) y no puedan avanzar sólo podrán dirigirse en las otras casillas posibles.

- En el caso de encontrarse con un obstáculo los agentes avanzan esquivando los cráteres o bordes siempre y cuando el camino hacia el objetivo sea mínimo.

- Los agentes escogen un camino mínimo sin interrumpir el de otros. En el caso de interferir en la ruta de otro agente, deberá recalcular su ruta sin interferir.

- En el caso de que no puedan avanzar tendrán la posibilidad de destruir un obstáculo y avanzar por el mismo.

- Se podrá generar de forma aleatoria los obstáculos y meta, incluso se podrán añadir nuevos obstáculos durante el trayecto de los agentes.
- Dispondrán también de la opción de elegir los puntos donde se situarán los

obstáculos y la meta de forma manual.

● Los agentes cogen una muestra del agua de su sector.

● En el caso de que un agente no pueda avanzar sin interferir la ruta de otro agente se detendrá desactivando su ruta.

● Cada agente se moverá dentro de un sector asignado.

● Los agentes pueden pedir apoyo según el nivel de agua encontrado en ese sector. Será el coordinador quien acepte el apoyo sugerido.

● El agente coordinador distribuirá el rol de cada agente.

4. Objetivo

Los agentes deberán encontrar el camino mínimo entre su posición y la posición final marcada por el usuario, en este caso hábitat será el planeta Marte (Matriz M x N con tamaño definido por el usuario) y el robot deberá encontrar la ruta hacia la casilla que contenga agua.

5. Entorno

Nuestro entorno será una matriz rectangular de M x N. Cada celda representa una franja de terreno en Marte. Los agentes empezarán su andadura partiendo de diferentes partes de la matriz. Las celdas vacías representan el espacio por donde los agentes podrán moverse. Las celdas pintadas de marrón representarán cráteres. Los objetos se compondrán de distintas celdas pintadas. Cuando encuentre la celda pintada de azul (agua) que será el objetivo, habrá finalizado la ruta.

6. Arquitectura

La arquitectura elegida para los agentes está basada en objetivos, tendrán información acerca de las metas asignadas (ubicaciones) por el coordinador a cada agente. Los agentes estarán condicionados cuando se encuentren una situación indeseable. El coordinador asignará a “n” agentes en caso de que alguno de ellos detecte una mayor cantidad de agua en la zona asignada

7. Planteamiento del problema:

Se nos han pedido los siguientes requisitos para la aplicación:

1. Debe tener una matriz de MxN.
2. Se pueden definir los obstáculos de forma aleatoria y manual.
3. Se debe permitir una visualización de la trayectoria determinada.
4. Debe contar con una interfaz gráfica.
5. Se dispone de un robot con 4 sensores para detectar posición.

Con esto en mente hemos dispuesto lo siguiente:

1. Como entorno de desarrollo utilizaremos Eclipse.
2. Para construir la interfaz visual se usará la librería Java Swings de la que utilizaremos varias clases.
3. Se crearán varias clases: una clase Applet, una clase Lienzo , una clase Node,una clase Canvas,una clase Astart,una clase Cargar mapa y por último la clase Guardar mapa .
4. La visualización se visualizará mediante refresco.

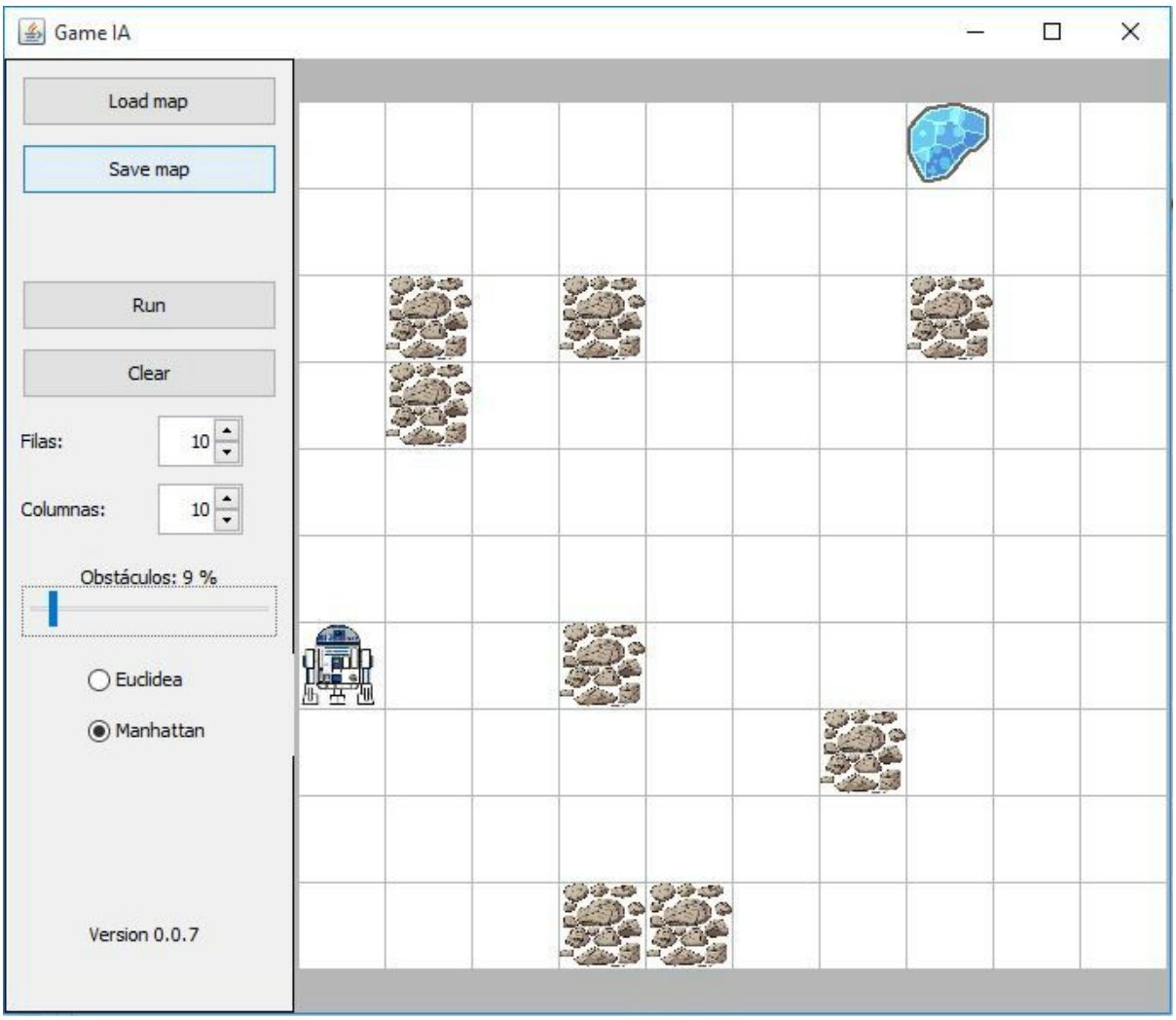
8. La Interfaz

Para trabajar en el proyecto hemos elegido el lenguaje de programación **JAVA**, por ser fácil de usar y tener bastantes librerías útiles que podremos utilizar. Como IDE utilizaremos **ECLIPSE**, por su buena integración con el lenguaje y la facilidad que nos ofrece para programar en él.

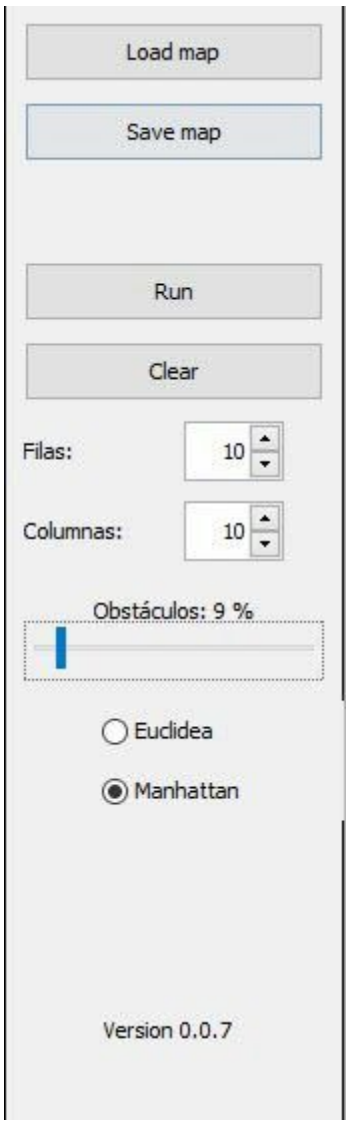
Optamos por un diseño basado en paneles que se habiliten según el modo marcado. Una vez trazadas las interacciones y un esbozo de lo que se quiere, procedemos a realizar el trabajo utilizando las librerías de **Swings.** Se debería también poder elegir entre distintos tipos de distancias heurísticas, por ello habilitamos inicialmente una choicebox para escoger una de ellas para ver cómo funcionan y ver qué diferencias se pueden apreciar .Las distancias que se han implementado son la distancia **Euclídea** y la distancia de **Manhattan** que serán explicadas más adelante

El modo será totalmente manual, especificando las casillas de destino y del robot. Y facilitamos un modo de previsualización en un slide para seleccionar el porcentaje de

obstáculos. Aplicando distintos estilos gracias al uso de las librerías que nos ofrece Swings, el panel principal queda de la siguiente manera.

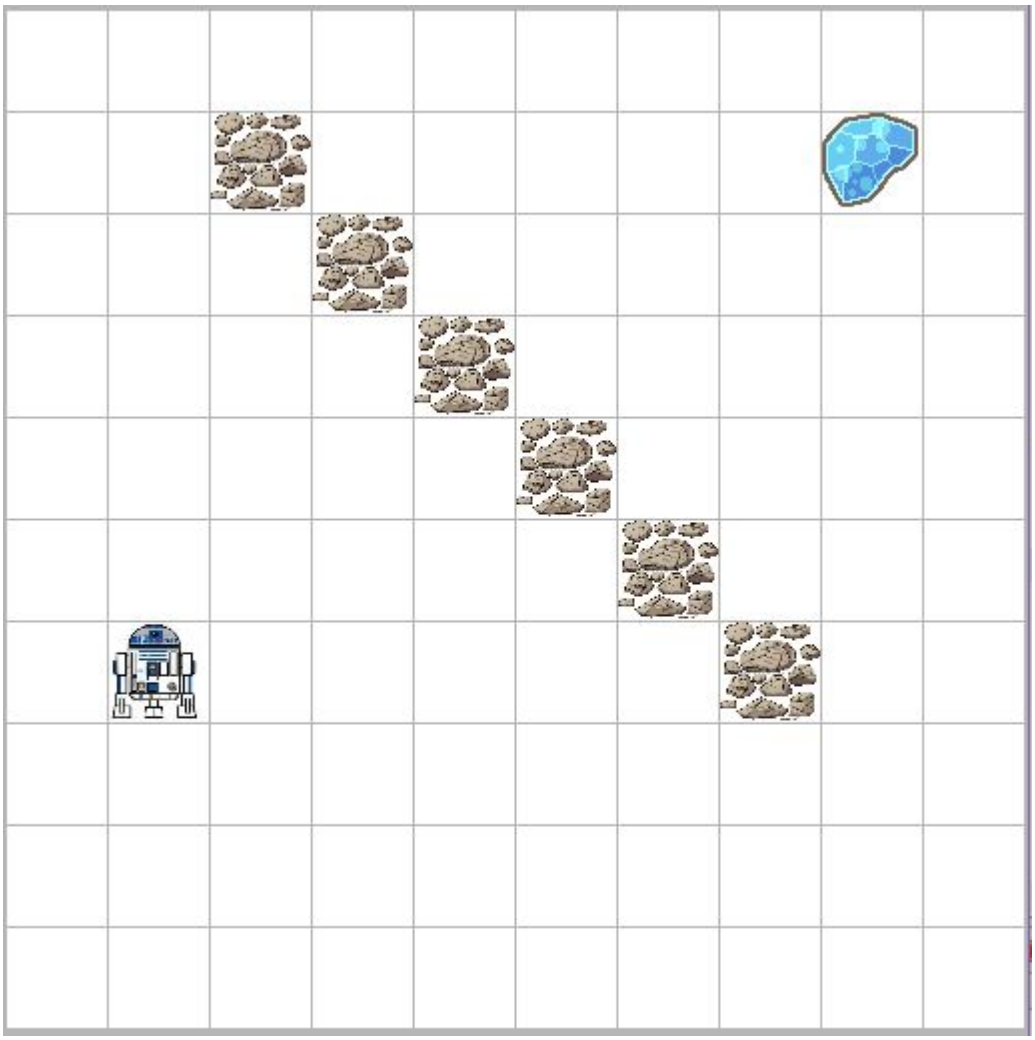


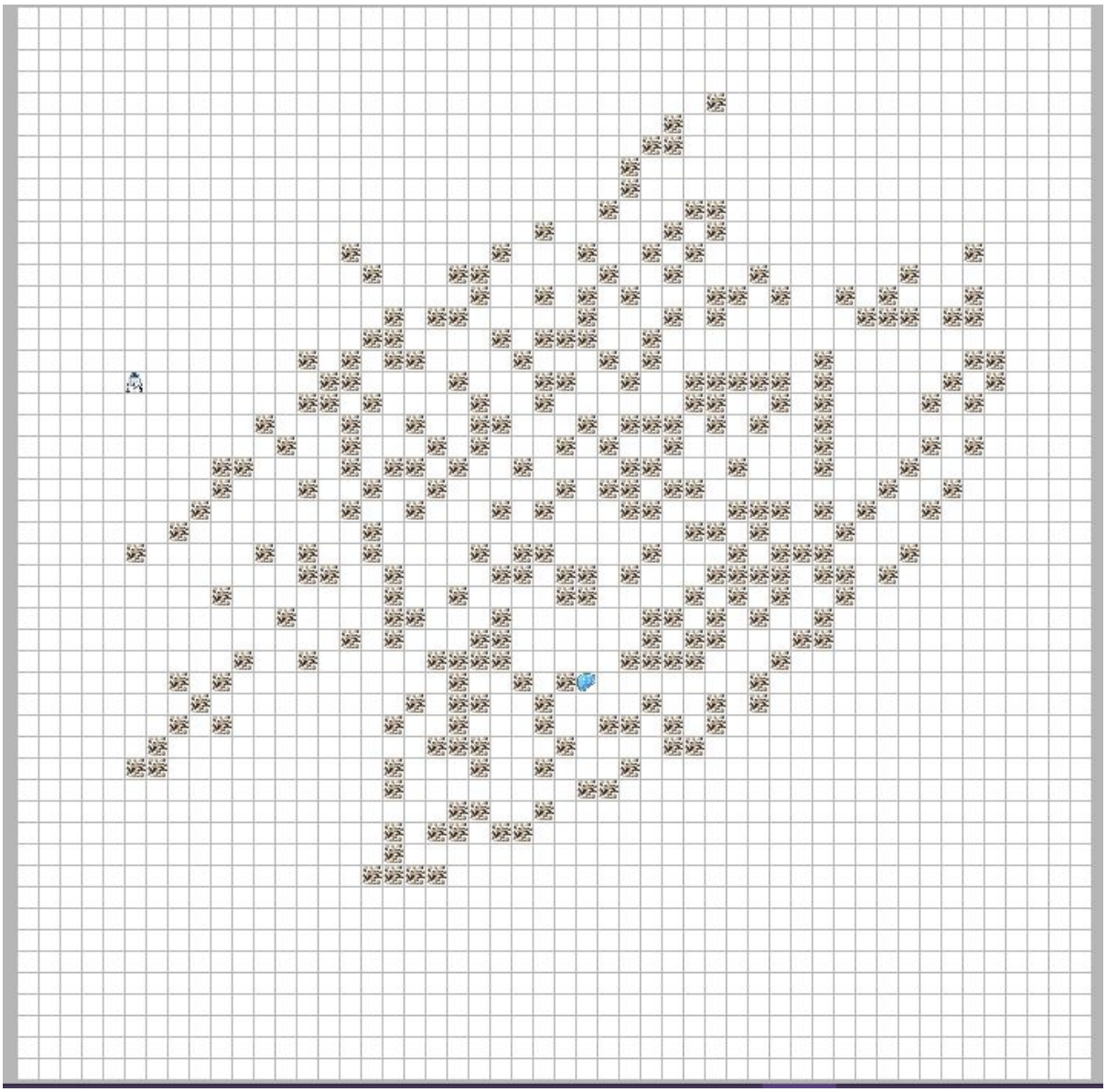
En esta imagen se puede observamos los diferentes botones que aparecen en el panel.



Básicamente el dibujo de la matriz consiste en un un panel que ajusta las celdas automáticamente en función del número de filas y columnas que le digas y el ancho que le pongas.

Aquí podemos ver un escenario normal, con una matriz pequeña y uno con una matriz grande.

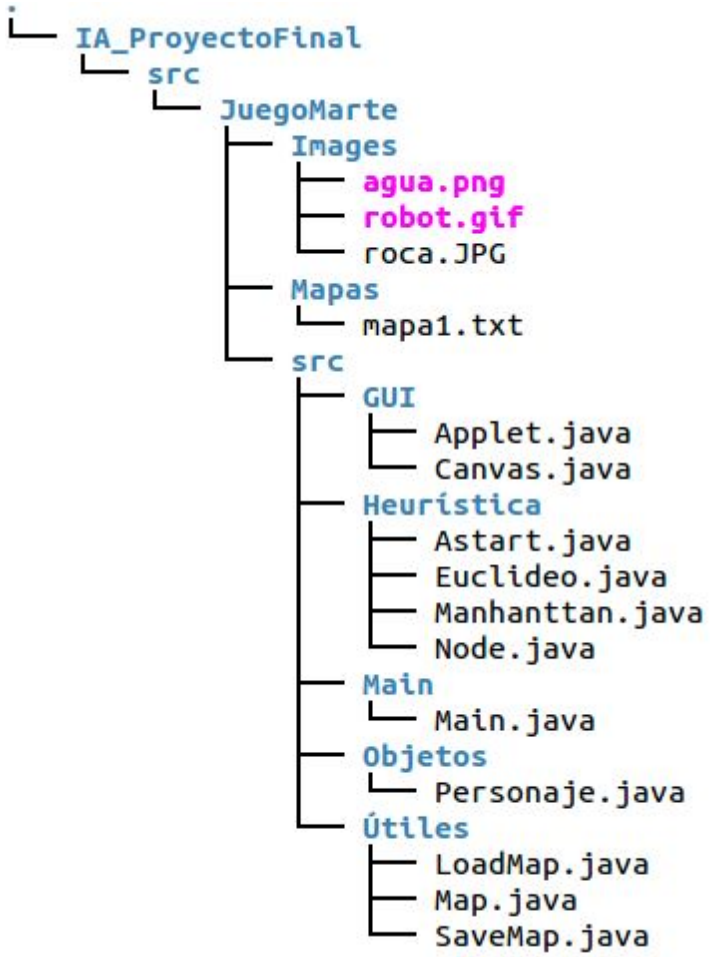




Para mejorar la robustez del programa y que no hayan errores, se han revisado que todos los campos introducidos cumplan con las normas. Se ha hecho uso de la estructura try / catch para poder manejar los errores.

9. Desarrollo de la estructura lógica

Para el desarrollo de la estructura lógica de nuestro programa hacemos uso de varias clases:



Descripción:

GUI

- **Applet** contiene la estructura de botones de la parte izquierda, situada en un panel y estando añadida a ella sliders, campos de texto, etc
- **Lienzo** crea la matriz de juego en un panel central, permitiéndonos visualizar y poder pintar en ella.

Heurística

- **AStar** implementa el algoritmo A* visto a lo largo de la asignatura, y el cual permite que nuestro robot pueda encontrar un camino mínimo hacia su objetivo.
- **Manhattan** aplica el algoritmo de Manhattan y devuelve la distancia entre dos puntos de la matriz.
- **Node** crea una estructura que define todos los puntos de la matriz teniendo una X y una Y para poder situar en el mapa.

Main

- **Main** asigna un tamaño a la interfaz e inicializa el lienzo. Que contiene los distintos paneles, main nos permite visualizar el programa en su totalidad.

ADICIONAL

Útiles:

- **LoadMap** nos permite cargar un mapa de la carpeta mapas, e visualizarlo directamente en nuestro programa.
- **Map** define la estructura del mapa que queramos cargar o guardar.
- **SaveMap** nos permite en cualquier momento poder guardar la pantalla que hemos creado, creando así un nuevo mapa.

10. Algoritmos heurísticos.

El algoritmo de búsqueda que se ha elegido e implementado para la realización de la búsqueda del camino deseado en la práctica ha sido el algoritmo de A*.Se ha elegido esta opción porque de los posibles algoritmos propuestos el A* es la mejor para alcanzar una determinada meta,en este caso sería que el robot encontrará en Marte el objetivo(agua).
El **problema** de algunos algoritmos de búsqueda en grafos informados, como puede ser el algoritmo voraz, es que se guían en exclusiva por la función heurística, la cual puede no indicar el camino de coste más bajo, o por el coste real de desplazarse de un nodo a otro (como los algoritmos de escalada), pudiéndose dar el caso de que sea necesario realizar un movimiento de coste mayor para alcanzar la solución. Es por ello bastante intuitivo el hecho de que un buen algoritmo de búsqueda informada debería tener en cuenta ambos factores, el valor heurístico de los nodos y el coste real del recorrido.

Así, el algoritmo **A*** utiliza una función de evaluación **f(n) = g(n) + h'(n)**, donde h'(n) representa el valor heurístico del nodo a evaluar desde el actual, n, hasta el final, y g(n), el coste real del camino recorrido para llegar a dicho nodo, n, desde el nodo inicial.

A* mantiene dos estructuras de datos auxiliares, que podemos denominar abiertos, implementado como una cola de prioridad (ordenada por el valor f(n) de cada nodo), y cerrados, donde se guarda la información de los nodos que ya han sido visitados. En cada paso del algoritmo, se expande el nodo que esté primero en abiertos, y en caso de que no sea un nodo objetivo, calcula la f(n) de todos sus hijos, los inserta en abiertos, y pasa el nodo evaluado a cerrados.

El algoritmo es una combinación entre búsquedas del tipo primero en anchura con primero en profundidad: mientras que h'(n) tiende a primero en profundidad, g(n) tiende a primero en anchura. De este modo, se cambia de camino de búsqueda cada vez que existen nodos más prometedores.

Como todo algoritmo de búsqueda en amplitud, A* es un algoritmo completo: **en caso de existir una solución, siempre dará con ella.**

El algoritmo A* aplicado al nuestro programa Juego Marte

Cuando se haya asignado las posiciones del robot el agua y los obstáculos el algoritmo A* entra en escena.Cuando hacemos click sobre el botón Run, se generará el camino camino mínimo posible gracias a tener implementado el algoritmo A*.Una vez que se empieza a generar el camino se pueden encontrar varios casos.

Utilizaremos un bucle while(true) para inicializar nuestro programa si encuentra el camino al destino fijado,el algoritmo ha realizado su trabajo correctamente y retornará la lista abierta con la solución. Si no encuentra el camino es que no puede llegar al destino final porque no tiene camino posible por el que seguir debido a los diferentes obstáculos colocados por el camino,el algoritmo comprobará que la lista abierta esté vacía al final del mismo y saldremos del bucle imprimiendo un mensaje por pantalla diciéndole al usuario que no se ha encontrado la misma.

Para optimizar la búsqueda del camino para que encuentra el destino lo más rápido posible se utiliza una función Hash.Lo que hacemos con esta función es sobreescribirla para ordenar los elementos de la lista abierta y devuelva el coste heurístico F.

11. Complejidad computacional

La complejidad computacional del algoritmo está íntimamente relacionada con la calidad de la heurística que se utilice en el problema. En el caso peor, con una heurística de pésima calidad, la complejidad será exponencial, mientras que en el caso mejor, con una buena h'(n), el algoritmo se ejecutará en tiempo lineal. Para que esto último suceda, se debe cumplir que h'(x) <= g(y) - g(x) + h'(y) donde h* es una heurística óptima para el problema, como por ejemplo, el coste real de alcanzar el objetivo.

Uso de memoria
El espacio requerido por A* para ser ejecutado es su mayor problema. Dado que tiene que almacenar todos los posibles siguientes nodos de cada estado, la cantidad de memoria que requerirá será exponencial con respecto al tamaño del problema.

En la clase Heurística se lleva a cabo la implementación de 4 funciones principales:
Función que devuelve el coste estimado.
Función que calcula la Distancia Manhattan.
Función que calcula la distancia de Euclídeo.
Función para saber si un nodo ha sido visitado previamente.
Función para diversificar el nodo

Función que optimiza la búsqueda del camino mínimo encontrado un 70% más rápido
El algoritmo principal A*

El algoritmo A* consta de los siguientes pasos:

1. Formar una lista de trayectorias parciales, abierta, con una trayectoria inicial que comienza en el nodo raíz. Formar una lista cerrada, de trayectorias desechadas mínimas, inicialmente vacía.
2. Hasta que la lista abierta esté vacía o se encuentre el objetivo, analizar su primera trayectoria:
   2.1 Si la trayectoria termina en el nodo objetivo, se finaliza el bucle.
   2.2 Si la primera trayectoria no termina en el nodo objetivo:
   2.3 Eliminar la primera trayectoria de la lista abierta, incluyéndose en la lista cerrada. En el caso de que ya exista una similar, eliminar la de mayor coste
   2.4 Formar nuevas trayectorias a partir de la trayectoria eliminada de abierta ramificando el último nodo de la misma.
   2.5 Añadir las nuevas trayectorias a la lista abierta, si existen.
   2.6 Ordenar la lista abierta en base al costo total estimado de cada una, colocando la de mínimo coste al inicio de la lista.
   2.7 Si dos o más trayectorias de abierta acaban en un nodo común, borrar las mismas excepto la que posee mínimo coste entre ellas. Eliminar ésta última también si existe una similar con menor coste en la lista cerrada. Al eliminar trayectorias de abierta deben insertarse en cerrada salvo que ya exista allí una similar de menor coste.

   3. Si se alcanza el nodo objetivo, el problema tiene solución y es óptima, en caso contrario no existe solución.

12. Manhattan e Euclídea

Manhattan:

La distancia entre dos puntos medido a lo largo de los ejes y los ángulos correctos.
En un plano entre:

p_1 at (x_1, y_1) y p_2 at (x_2, y_2), it is $|x_1 - x_2| + |y_1 - y_2|$.

Euclídea:

La línea recta entre dos puntos.
En un plano entre:

p_1 at (x_1, y_1) and p_2 at (x_2, y_2), it is $\sqrt{((x_1 - x_2)^2 + (y_1 - y_2)^2)}$.

13. Metodología del trabajo

Para realizar la práctica nos hemos dividido el trabajo realizado nos hemos organizado de la siguiente manera:

Las tareas marcadas con ** han sido realizadas de forma cooperativa

Eric Ramos Suárez:

- Implementación de generación de obstáculos
- Creación de interfaz**
- Redacción de informe**
- Implementación Java**

Lucas Ruíz González:

- Creación de la interfaz**
- Redacción de informe**
- Implementación Java**

Germán Paz Méndez

- Creación de la interfaz**
- Redacción de informe**
- Implementación algoritmo de búsqueda
- Implementación gestión de errores
- Implementación web: http://gcpmendez.github.io/IA_ProyectoFinal/
- Implementación applet
- Implementación Java**

- Reunión una vez por semana para debatir e implementar los objetivos de la práctica.
- Utilización de un sistema de control de versiones. Github, que hemos ido actualizando cada semana.
  Link: https://github.com/gcpmendez/IA_ProyectoFinal

14. Bibliografía

https://en.wikipedia.org/wiki/A*_search_algorithm

https://www.youtube.com/watch?v=eTx6HQ9Veas

https://es.wikipedia.org/wiki/Algoritmo_de_b%C3%BAsqueda_*

https://www.youtube.com/watch?v=kgr4AiV4eCY

https://xlinux.nist.gov/dads/HTML/euclidndstnc.html

https://xlinux.nist.gov/dads/HTML/manhattanDistance.html