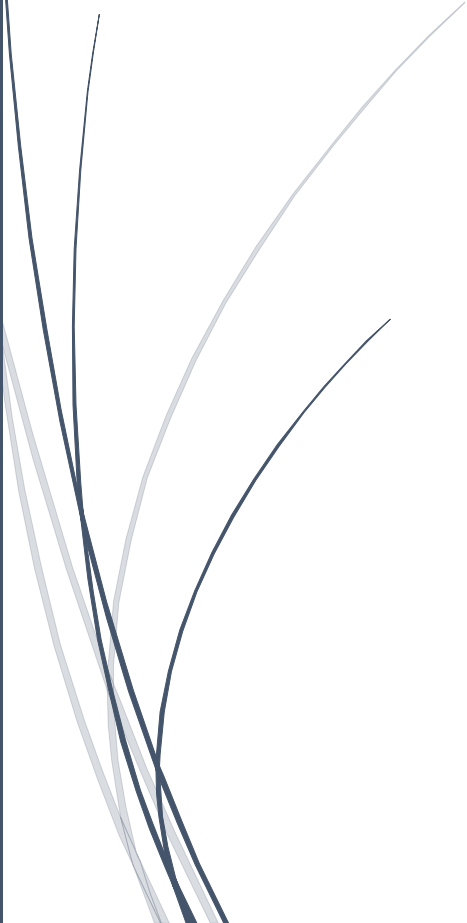
A dark blue vertical bar on the left side of the page. A blue arrow points to the right from the bar, containing the date.

14-12-2014

# Informe IA

Estrategias de búsqueda

Several thin, curved lines in dark blue and light grey originate from the bottom left and curve upwards and to the right.

Fabián Díaz Lorenzo  
Juan José Gregorio Díaz Marrero  
Miguel Aurelio García González  
UNIVERSIDAD DE LA LAGUNA

## Índice

<b>Introducción.....</b>	<b>2</b>
<b>¿En qué consiste nuestro modelo?.....</b>	<b>3-6</b>
<b>Entorno de desarrollo.....</b>	<b>7</b>
<b>Estrategia de búsqueda empleada.....</b>	<b>8, 9</b>
<b>Nuestro código.....</b>	<b>10, 11</b>
<b>Diseño basado en agentes.....</b>	<b>12, 13</b>
<b>División del trabajo.....</b>	<b>14</b>
<b>URL al código.....</b>	<b>15</b>

## Introducción

En esta última práctica de la asignatura hemos abordado las estrategias de búsqueda, básicamente el objetivo de esta práctica se basa en la utilización de estas estrategias como propuesta de resolución en la determinación de la planificación de trayectorias para la navegación de un robot humanoide. Antes de ver nuestro modelo, veamos qué tipos de estrategias de búsqueda podemos encontrar:

- **Estrategias de búsqueda sin información**
  - Búsqueda primero en anchura
  - Búsqueda primero en profundidad
  - Búsqueda limitada por profundidad
  - Búsqueda por profundización iterativa
  - Búsqueda bidireccional
- **Estrategias de búsquedas Heurísticas**
  - Estrategia en Escalada
  - Estrategia por Haces
  - Estrategia Primero el Mejor
  - Estrategias Óptimas
    - Estrategia de Coste Uniforme
    - Estrategia de Coste Uniforme con Subestimación
    - Estrategia de Coste Uniforme con Programación Dinámica
    - Estrategia A\*
- **Estrategias Anytime**
- **Estrategias en Tiempo Real**
  - LRTA\*
  - RTA\*
  - MTS

Vemos que hay diferentes tipos de estrategia, en este caso solo las nombramos y más adelante veremos cuál de estos tipos de estrategia hemos seguido, lo describiremos y veremos el por qué hemos seguido dicha estrategia.

## ¿En qué consiste nuestro modelo?

En esta práctica, se pide de manera general un entorno para el robot humanoide que se puede suponer rectangular de dimensiones  $M \times N$  y constituido por celdas libres y ocupadas, donde el robot puede efectuar acciones de movimiento, una cada vez, desde la casilla actual a una de las 4 casillas vecinas (Norte, Sur, Este, Oeste) que no se encuentre ocupada. Las casillas ocupadas corresponden a que hay algún obstáculo en dicha casilla, y las casillas libres corresponden a celdas que están libres de obstáculos.

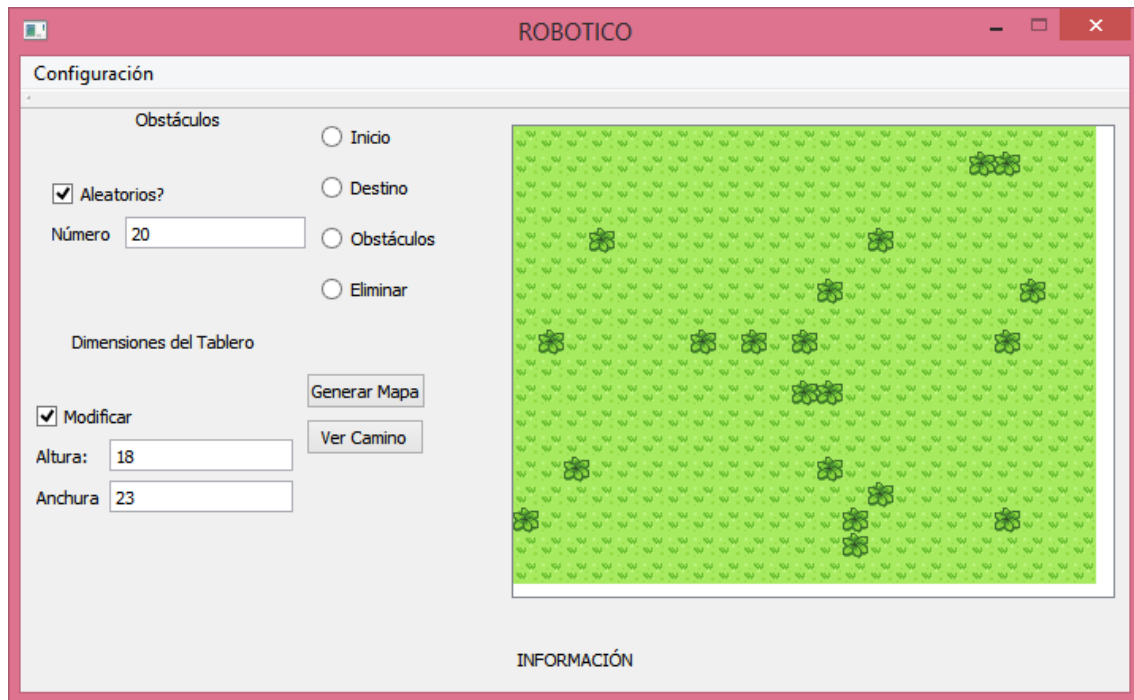
En nuestro caso, el entorno de nuestro robot humanoide se trata de un bosque de dimensiones  $M \times N$ , en el que encontramos árboles que se corresponderían con los obstáculos que el robot debe evitar. El robot debe llegar desde el punto donde se encuentra en un principio (punto inicial) hasta la empresa maderera (punto final) por el camino más corto posible.

De manera general, nuestro robot debe ir cogiendo madera por el bosque y llegar a la empresa maderera (punto final) de manera óptima.

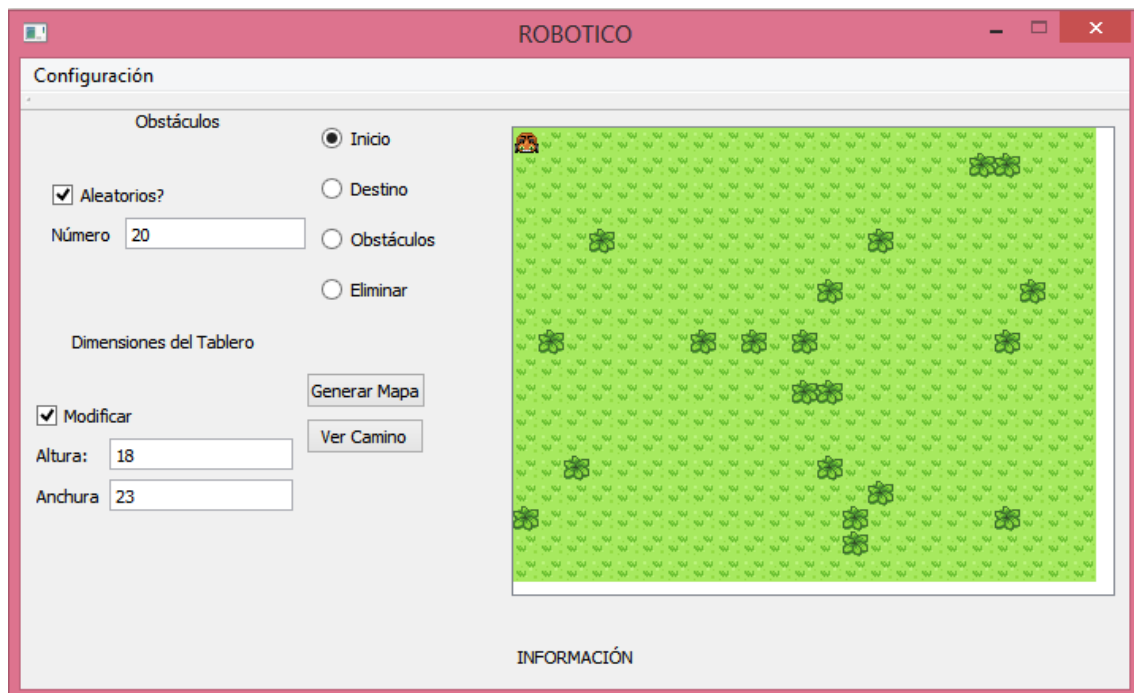
Veamos cómo funciona el modelo que hemos llevado a cabo:



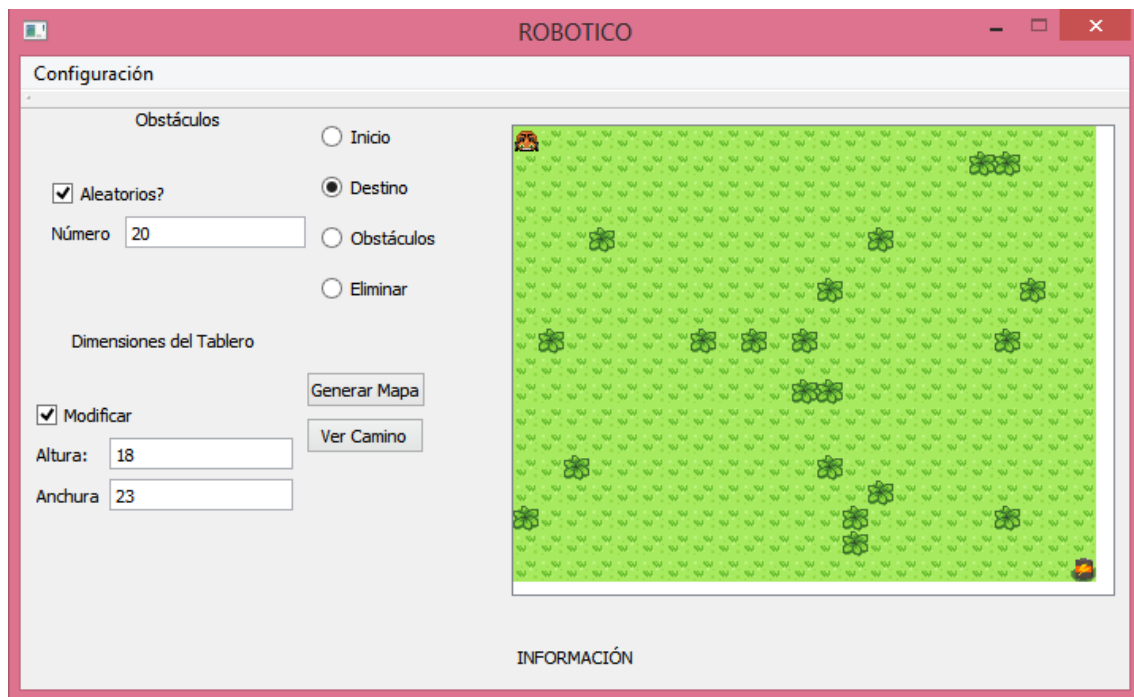
En primer lugar, nos saldría esta pantalla donde encontramos un menú donde vamos a seleccionar el número de obstáculos que queremos que aparezcan de manera aleatoria en nuestro escenario, y también el tamaño de nuestro escenario. En este caso pongamos que nuestra escena va a tener de altura 18 y de anchura 23, y además tendrá 20 obstáculos:



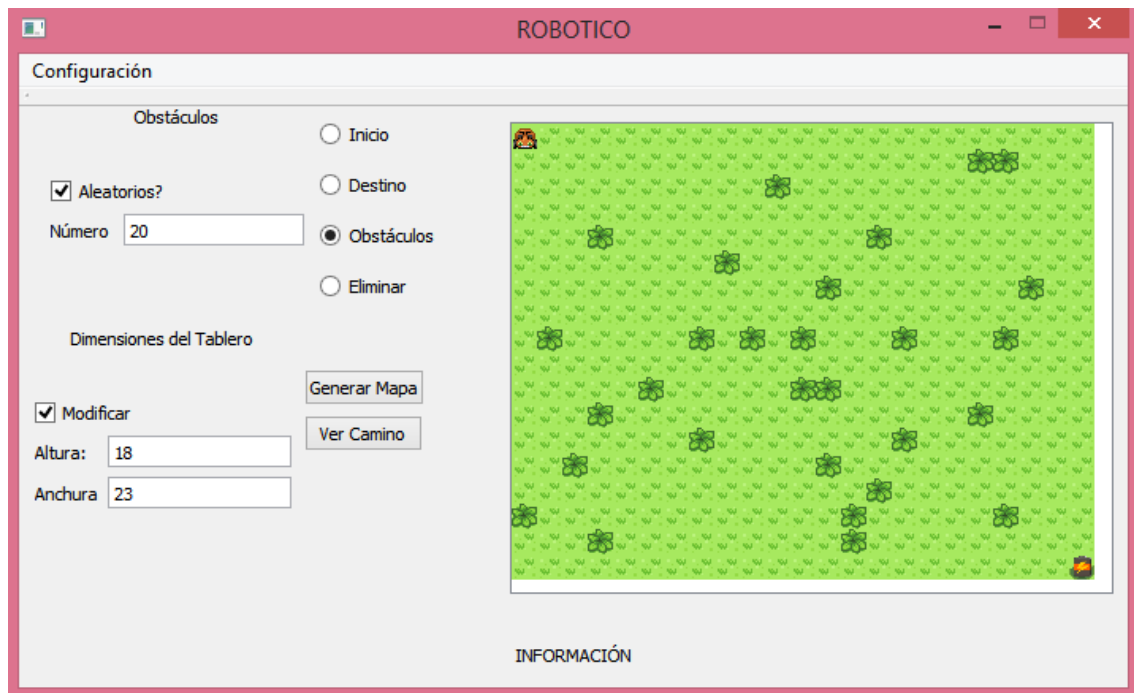
Este es el escenario que se nos ha generado, a continuación vamos a seleccionar donde queremos que comience nuestro robot, es decir, el punto de inicio:



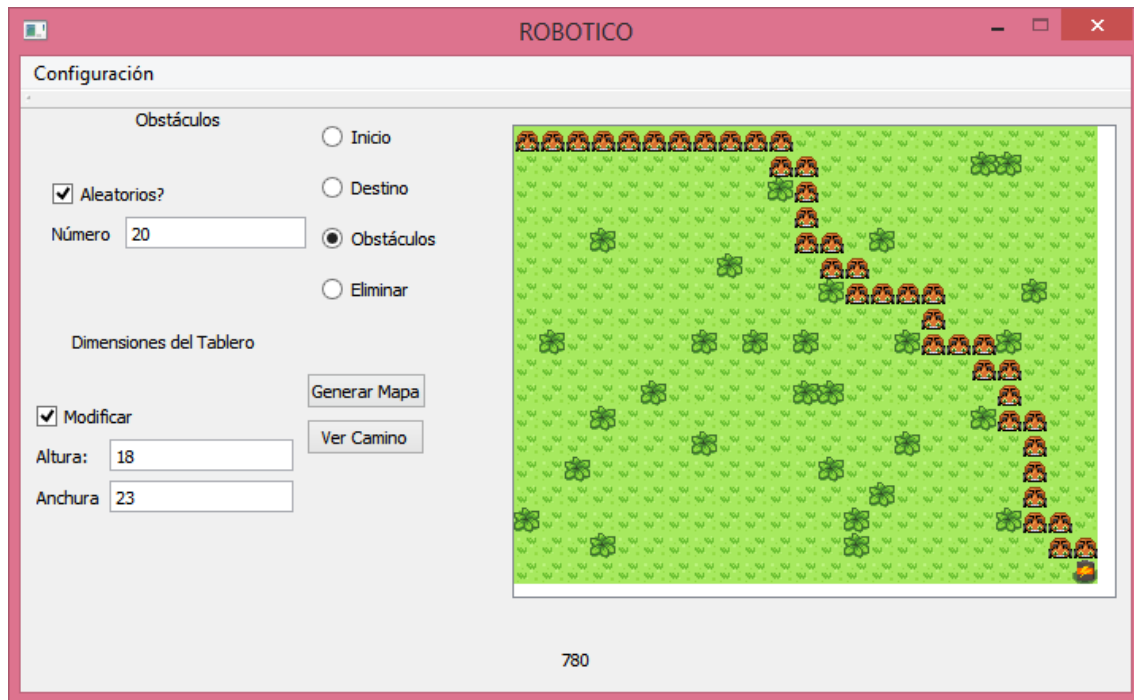
Vemos que el punto de inicio de nuestro robot será, en este caso, la esquina superior izquierda, ahora seleccionemos el destino, es decir, el punto final:



En esta ocasión hemos seleccionado como punto final que debe alcanzar el robot la esquina inferior derecha de nuestro escenario, ahora añadiremos algunos obstáculos más de manera manual:



Finalmente, solo nos queda ver el camino que debe recorrer nuestro robot para llegar desde su punto de partida al punto final de una manera óptima:



Lo que vemos sería el camino que debe seguir nuestro robot para llegar al punto final de manera óptima. Podemos observar que en la parte inferior del recuadro aparece un número, en este caso 780, esto nos indica el coste acumulado del camino que debe realizar el robot.

## Entorno de desarrollo

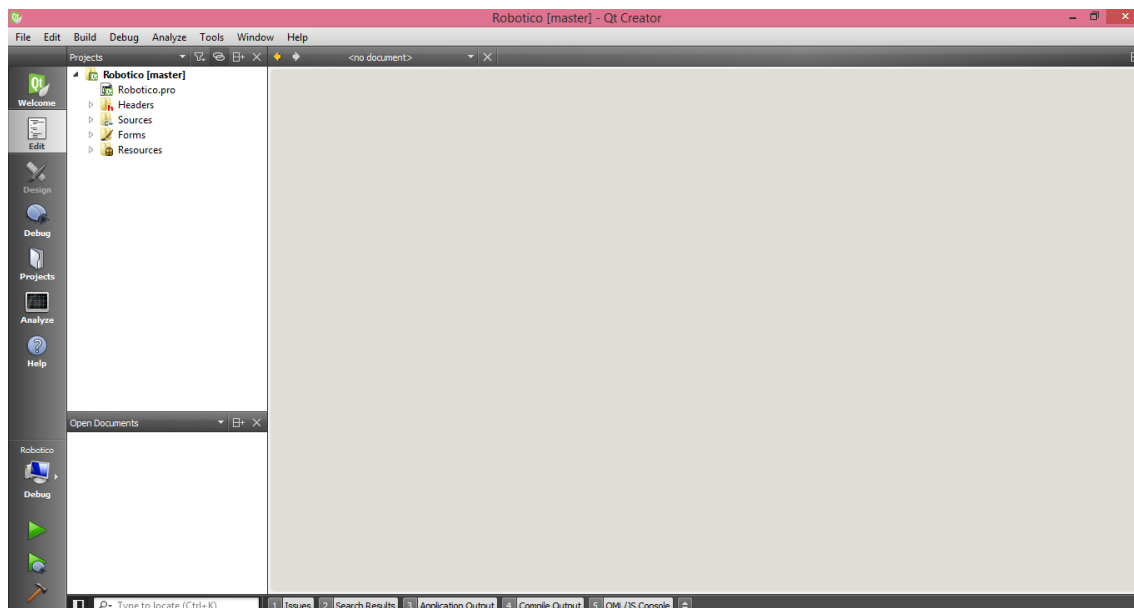
El entorno de desarrollo que hemos seleccionado para llevar a cabo nuestra práctica es Qt Creator ya que es un entorno con el que ya hemos trabajado y consideramos que nos iba a resultar más cómodo trabajar con él que investigar con algún otro entorno de desarrollo.

Algunas de las características que tiene Qt son las siguientes:

- Editor de código con soporte para C++, QML y ECMAScript.
- Herramientas para la rápida navegación del código.
- Resaltado de sintaxis y auto-completado de código.
- Control estático de código y estilo a medida que se escribe.
- Soporte para refactoring de código.
- Ayuda sensitiva al contexto.
- Plegado de código.
- Paréntesis coincidentes y modos de selección.

Además, Qt Creator cuenta con un depurador que muestra la información en bruto procedente de GDB de una manera clara y concisa.

- Interrupción de la ejecución del programa.
- Ejecución línea por línea o instrucción a instrucción.
- Puntos de interrupción (breakpoints).
- Examinar el contenido de llamadas a la pila (stack), los observadores y de las variables locales y globales.





## Estrategia de búsqueda empleada

En esta práctica se nos pedía seguir una de las estrategias heurísticas para planificar el camino del robot humanoide, es decir, que partiendo de una posición inicial el robot pueda determinar la trayectoria óptima para alcanzar la posición final. Recordemos como se clasificaban las estrategias heurísticas, y veamos en qué consiste cada una de ellas:

### - Estrategias de búsquedas Heurísticas

- Estrategia en Escalada
- Estrategia por Haces
- Estrategia Primero el Mejor
- Estrategias Óptimas
  - Estrategia de Coste Uniforme
  - Estrategia de Coste Uniforme con Subestimación
  - Estrategia de Coste Uniforme con Programación Dinámica
  - Estrategia A\*

### Estrategias de búsquedas Heurísticas

Dentro de la literatura de búsqueda heurística, el término heurístico generalmente se refiere a un caso especial de función de evaluación heurística.

La función de evaluación heurística de un nodo  $n$ , se denota como  $h(n)$

- $h^*(n)$  representa el coste estimado
- $h(n)$  representa el coste verdadero

La función de evaluación produce un número que sirve para representar lo deseable o indeseable que sería la expansión de un nodo (p.e: para un estado final fijo: estima de la distancia a partir del nodo  $n$  al nodo objetivo).

### Estrategia en Escalada

Esta estrategia se basa en continuar por el mejor de los hijos del nodo actual, basándonos en alguna evidencia que nos permita ordenar los nodos hijos, de tal forma que se exploren primero aquellos que presentan mayor expectativa de éxito.

### Estrategia por Haces

Esta estrategia se basa en una exploración por niveles, pero no se desciende por todas las ramas, sino por un conjunto seleccionado de ellas o “haz”, ignorándose el resto.

La selección de las ramas adecuadas se realiza en base a criterios específicos del problema, produciéndose una especie de “poda de ramas” poco prometedoras.

### Estrategia Primero el Mejor

Esta estrategia es similar a la de escalada, salvo que la exploración no continua por el mejor de los hijos del nodo actual, sino por el mejor nodo de todos los presentes en la lista.

### Estrategias Óptimas

Estas estrategias se basan en la determinación de trayectorias óptimas en la resolución de problemas basados en estados.

### **Estrategia de Coste Uniforme**

Se basa en utilizar la información que proporciona una función de coste acumulado desde el inicio o raíz hasta cada nodo.

Se asume que el coste total de una trayectoria consiste en la acumulación de los costes de transiciones entre estados consecutivos.

### **Estrategia de Coste Uniforme con Subestimación**

El método anterior puede ser mejorado, utilizando una estimación del coste, o distancia total al nodo objetivo a través de la trayectoria en estudio.

Para ello, se supone que el coste total para un nodo  $n$ ,  $f^*(n)$  es:

$$f^*(n) = g(n) + h^*(n)$$

- $g(n)$ : coste acumulado hasta el nodo de trabajo
- $h^*(n)$ : estimación del coste restante desde el nodo de trabajo hasta el nodo actual
- $f^*(n)$ : estimación del coste total de la trayectoria parcial

### **Estrategia de Coste Uniforme con Programación Dinámica**

Esta estrategia se basa en el principio de programación dinámica. Según este principio: en una trayectoria óptima, todas las subtrayectorias de la misma también son óptimas.

### **Estrategia A\***

Esta estrategia es una combinación del método Coste Uniforme con Subestimación y Programación Dinámica.

Escogimos la estrategia de búsqueda A\* dado que este es el algoritmo más eficiente con respecto a los demás de los algoritmos óptimos y nos da siempre el mejor camino, a diferencia de los algoritmos heurísticos, si lo hay.

## Nuestro código

El algoritmo se ha desarrollado mediante pseudomapa, esto quiere decir, que el autómata no se va a mover del sitio hasta que no tenga el recorrido óptimo marcado pero el algoritmo realizara la búsqueda del camino óptimo como si no tuviera mapa, actualizándose en cada iteración. Al final del algoritmo es cuando el autómata se moverá guiándose por el camino marcado.

En primer lugar nos encontramos el archivo *celda.cpp* y su cabecera *celda.h*, básicamente lo que nos encontramos aquí es la declaración de la clase *celda* con un método *get*, para conocer el estado de la celda, y un método *set*, para asignarle un estado. También podemos encontrar en la cabecera, una variable para poder conocer el objeto que contiene una celda determinada. Debemos decir que estos ficheros heredan de la clase *QGraphicsPixmapItem* para que una celda se pueda representar en una escena en Qt.

Lo siguiente que encontramos son los archivos *robot.cpp* y su cabecera *robot.h*, aquí encontramos las siguientes funciones:

- En primer lugar encontramos la función *Robot*, que realiza la creación del robot, otorgándole una posición inicial e inicializando los sensores a un array de 4 (que indicará el norte, el sur, el este y el oeste respectivamente), y poniéndolas a 0.
- Después encontramos el *destructor* y las diferentes funciones de *set* y *get*.
- A continuación encontramos la función *inicializar\_sensores*, la cual es clave para el desarrollo del algoritmo. Se encarga de buscar las distintas trayectorias a partir de la última posición de nuestra estructura nodo. Para ello tiene 3 estados posibles:
  - o 0 Posición válida.
  - o 1 Posición inválida debido a que hay un obstáculo (árboles).
  - o -1 Posición inválida debido a que está fuera del rango del terreno establecido.

Al final esta función devolverá un booleano el cual mandará un *true* cuando la suma del valor absoluto de los 4 sensores es menor que 4, esto indica que al menos uno de los sensores es 0 por lo que a partir de esa trayectoria es posible que se pueda encontrar un camino óptimo.

- Ahora encontramos la función *variantLessThan*, esta es una función que se le incorporará al método *Qsort* para que ordene la lista según su coste.
- La siguiente función es *RecEuristico*, esta es la función principal que desarrollara el algoritmo A\*. El principio crea dos lista (abierta (contendrá las trayectorias posibles que no han sido vistas) y cerrada (trayectorias ya visitadas)). Se inicializa la lista abierta con el nodo inicial en el que está el nodo añadiendo a su vector la posición inicial y el coste inicial.
- La función *manhattan* calcula la distancia manhattan de unas coordenadas iniciales a unas coordenadas finales.
- La función *is\_outrange*, examina si una trayectoria se va fuera de rango utilizada en la función *inicializar\_sensores()*.
- La función *del\_duplicate*, elimina los duplicados de la lista abierta.

- La función *del\_duplicate\_dual*, examina si hay que eliminar el nodo actual por que ya existe un nodo encerrado con un coste menor o igual al coste del nodo actual.
- La función *is\_invector*, examina que dado un nodo n, este no tenga una coordenada guardada que sea exactamente igual(devuelve true si existe esa coordenada en el vector o false cuando no exista esa coordenada en el vector).

A continuación tenemos los ficheros *terreno.cpp* y su cabecera *terreno.h*, veamos sus funciones:

- La función *Terreno* inicializa el terreno.
- La función *introducir\_automata* según una coordenada x y coloca el automata en esa posición y lo construye.
- La función *introducir\_arboles* según una coordenada x y coloca un árbol en esa posición.
- La función *introducir\_empresa* según una coordenada x y coloca la empresa en esa posición.
- La función *introducir\_terreno* según una coordenada x y coloca un terreno transitable en esa posición.
- La función *rellenar\_paso\_natural* rellena lo que falte de mapa con pasos naturales.
- La función *marcar\_recorridoAutomata* llama a la función heurística para conseguir el camino óptimo y si lo hay lo marca en el mapa con la imagen que le corresponda.

Ahora encontramos los ficheros *vistagrafica.cpp* y *vistagrafica.h*, en esta ocasión se crea una clase que nos permite la representación de vistas y la sobrecarga del *mousepressed* dentro de la misma para saber qué celda se ha presionado dentro de la vista.

Los ficheros *mainwindow.cpp* y *mainwindow.h*, se encargan de generar el mapa con las diferentes opciones que se le pasen en el menú. En la cabecera (*mainwindow.h*) se especificar el tamaño por defecto del mapa y el tamaño máximo que puede tener el mapa, así como el número máximo de obstáculos que puede haber.

## Diseño basado en agentes

Un Agente es un sistema que interactúa con su entorno, es decir, que percibe de él a través de sensores y actúa sobre él a través de efectores, realizando una cierta tarea, es decir, cumpliendo unos objetivos.

Un sistema basado en múltiples agentes nos permite resolver ciertos problemas que con sólo un agente es imposible o demasiado complicado de resolver, en nuestro caso permite la recolección más eficiente de madera que hay por los suelos del bosque que dificultan el tránsito por el camino. Existen dos tipos de sistemas multiagente, los cooperativos, es decir, que colaboran entre ellos para llegar a una solución y, los competitivos, es decir, que compiten entre ellos para ver quién alcanza la solución en un menor tiempo posible. En nuestra representación, será un modelo cooperativo ya que los robots tienen una capacidad limitada de madera y es necesario que si hay varios agentes se ayuden entre ellos para equilibrar la carga, además de recorrer más caminos óptimos de un lugar A al lugar de destino.

Los agentes los clasificamos como agentes inteligentes y racionales porque a partir de una percepción realizan una acción determinada.

En nuestro diseño tendremos un número variable de robots que realizarán la labor de recolección de ramas que haya por el camino y que tendrán como único obstáculo al elemento árbol. Nos referimos a número variable de robots porque depende de las dimensiones del bosque para decidir un número óptimo de robots que realicen esta limpieza de caminos.

### **Sensores**

En lo referente a las percepciones, cada robot dispondrá de los siguientes sensores:

- Sensores de proximidad, para detectar la proximidad de los obstáculos (árboles).
- Sensores infrarrojos, para la visión nocturna (normalmente la limpieza de caminos se hará por la noche).
- Sensores de calor, para detectar animales y/o personas y evitar daños.

### **Movimientos**

Todos los agentes podrán realizar las siguientes acciones/movimientos:

- Movimiento hacia el norte.
- Movimiento hacia el sur.
- Movimiento hacia el este.
- Movimiento hacia el oeste.
- Recolección de ramas y otros elementos de los árboles.

### **Objetivos**

El objetivo de todo ellos es encontrar un camino óptimo para llegar a la empresa maderera y así generar caminos óptimos para el transporte de la madera por medio del bosque.

### **Entorno**

El entorno en el que se desarrollará dicha tarea será en un bosque en el cual se crearán varios caminos óptimos según las dimensiones del territorio.

## **Tipo de Arquitectura Software**

La arquitectura software propuesta es del tipo híbrida y está dividida en capas:

- Reactivo basado en modelo (nivel bajo): En general, los sensores en todo instante no suministran toda la información acerca del estado actual del entorno. Es preciso pues mantener alguna “memoria” de situación.
- Basado en objetivos: Para decidir qué hay que hacer, aparte de la información acerca del estado que prevalece en el ambiente, se necesita cierto tipo de información sobre su meta.
- Basado en utilidad: Las metas no bastan por sí mismas para generar una conducta de alta calidad (camino óptimo).

## **Funcionamiento y representación**

El funcionamiento y representación de dicho sistema es el siguiente:

1. Número variable robots (según las dimensiones de terreno) dispuestos cada uno en un lugar diferente del territorio, partiendo el territorio en partes iguales y en cada una de esas parte irá un autómata.
2. En algún punto del territorio está la empresa maderera, que es el lugar final al que tienen que llegar nuestros autómatas.
3. Cada autómata deberá hallar un camino óptimo para llegar a la empresa maderera evitando los obstáculos. Si por algún motivo dos o más autómatas llegaran a un mismo punto en común en su trayectoria, haría un intercambio de peso para que todos los robots tuvieran un peso equilibrado, esto es debido a que cada camino tenga más palos y ramas que otro camino.
4. El objetivo de cada robot humanoide es el de encontrar un camino óptimo a la empresa maderera.

## División del trabajo

ASPECTO DE LA PRÁCTICA	TIEMPO INVERTIDO	AUTOR/ES
<b>Escoger entorno de desarrollo</b>	30 min	Fabián Díaz Lorenzo Juan José G. Díaz Marrero Miguel A. García González
<b>Escoger estrategia de búsqueda</b>	30 min	Fabián Díaz Lorenzo Juan José G. Díaz Marrero Miguel A. García González
<b>Desarrollo de la parte gráfica</b>	600 min	Juan José G. Díaz Marrero
<b>Desarrollo del algoritmo</b>	1080 min	Fabián Díaz Lorenzo
<b>Desarrollo del informe</b>	100 min	Miguel A. García González

## URL del repositorio

[https://alu0100709476@bitbucket.org/alu0100709476/empresa\\_maderera.git](https://alu0100709476@bitbucket.org/alu0100709476/empresa_maderera.git)