SmartCar

Inteligencia Artificial

Informe del desarrollo el proyecto de coche autónomo realizado en la asignatura de inteligenciar artificial recogiendo información del proyecto que se ha realizado.

Angel Luis Igareta Herraiz
Martín Belda Sosa
Alberto Jesús González Álvarez

Índice

SmartCar - Informe

1. Introducción

1.1 Descripción del problema

El objetivo de esta práctica consiste en la implementación de una estrategia de búsqueda en un lenguaje de programación para la resolución de la determinación de la trayectoria óptima de un punto a otro en un entorno compuesto por celdas libres y ocupadas. Este escenario representa la determinación de la ruta óptima de un coche autónomo en el caso real.

El entorno se representa como una matriz de dimensiones MxN constituido por celdas libres y ocupadas, un origen y un destino. Se explorará cada celda libre que pueda formar un camino siguiendo el orden que establece el algoritmo de búsqueda A*. Este algoritmo usa la función de evaluación f(n) = g(n)+h'(n) donde g(n) representa el coste del camino recorrido desde el origen hasta llegar a la celda n y h'(n)el valor heurístico de la celda a evaluar desde la celda actual. Las funciones heurísticas implementadas en esta práctica son las heurísticas Manhattan, Chebyshev y Euclídea.

1.2 Determinación de la trayectoria

Suponiendo un único coche autónomo según el escenario definido en el apartado anterior, la estrategia de búsqueda implementada calcula la trayectoria óptima partiendo desde una posición inicial hasta alcanzar una posición final. Ambas posiciones son definidas por el usuario, en el entorno de simulación desarrollado previamente.

1.3 Evaluación de las funciones heurísticas

Para la estrategia implementada en el apartado anterior, se evalúa el rendimiento de las tres funciones heurísticas implementadas en tres tamaños de escenarios diferentes: 25x25, 50x50 y 100x100 celdas respectivamente.

1.4 Evaluación de las funciones heurísticas

Para la estrategia implementada en el apartado anterior, se evalúa el rendimiento de las tres funciones heurísticas implementadas en tres tamaños de escenarios diferentes: 25x25, 50x50 y 100x100 celdas respectivamente.

1.5 Realización de tareas por cada miembro. Coordinación.

- Primera semana de prácticas (13-oct-17): formación del grupo, elección del lenguaje de programación con el que implementar la práctica: C++ con Qt framework. Creado grupo de whatsapp y repositorio de github.

- Segunda semana de prácticas (20-oct-17): interfaz creada. Ventana principal para elegir número de columnas, filas y obstáculos. Ventana con grid que permite introducir obstáculos, origen y destino para representar la matriz. Fórmula de ajuste a los márgenes de la ventana. Clase tablero que permite acceder a los elementos de una matriz.

- Tercera semana de prácticas (27-oct-17): Implementación del algoritmo A*. Botones ok y reiniciar en la interfaz. Implementación de obstáculos aleatorios.

- Cuarta semana (3-nov-17): Mejora de los elementos visuales de la interfaz gráfica. Corregidos errores en el algoritmo A*. Funcionamiento del procedimiento de obstáculos automáticos corregido.

- Quinta semana (9-nov-17): Entrega del software.

Los elementos que se han utilizado para gestionar la coordinación del trabajo han sido Whatsapp, github y las horas de prácticas de la asignatura.

2. HERRAMIENTAS UTILIZADAS

- **Qt framework** (https://www.qt.io/qt-for-application-development/)

Qt es un framework multiplataforma orientado a objetos ampliamente usado para desarrollar programas (software) que utilicen interfaz gráfica de usuario, así como también diferentes tipos de herramientas para la línea de comandos y consolas para servidores que no necesitan una interfaz gráfica de usuario.

Qt utiliza el lenguaje de programación C++ de forma nativa, adicionalmente puede ser utilizado en varios otros lenguajes de programación a través de bindings. También es usada en sistemas informáticos embebidos para automoción, aeronavegación y aparatos domésticos como frigoríficos.

Funciona en las principales plataformas y tiene un amplio apoyo. El API de la biblioteca cuenta con métodos para acceder a bases de datos mediante SQL, así como uso de XML, gestión de hilos, soporte de red, una API multiplataforma unificada para la manipulación de archivos y una multitud de otros métodos para el manejo de ficheros, además de estructuras de datos tradicionales.

- **Git** (https://git-scm.com)

Git es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente. Al principio, Git se pensó como un motor de bajo nivel sobre el cual otros pudieran escribir la interfaz de usuario o front end como Cogito o StGIT. Sin embargo, Git se ha convertido desde entonces en un sistema de control de versiones con funcionalidad plena. Hay algunos proyectos de mucha relevancia que ya usan Git, en particular, el grupo de programación del núcleo Linux.

- **Github** (https://github.com/)

GitHub es una forja (plataforma de desarrollo colaborativo) para alojar proyectos utilizando el sistema de control de versiones Git. Utiliza el framework Ruby on Rails por GitHub, Inc. (anteriormente conocida como Logical Awesome). Desde enero de 2010, GitHub opera bajo el nombre de GitHub, Inc. El código de los proyectos alojados en GitHub se almacena típicamente de forma pública, aunque utilizando una cuenta de pago, también permite hospedar repositorios privados.

- **Whatsapp** (https://www.whatsapp.com/)

WhatsApp es una aplicación de mensajería instantánea para teléfonos inteligentes, que envía y recibe mensajes mediante Internet, complementando servicios de correo electrónico, mensajería instantánea, servicio de mensajes cortos o sistema de mensajería multimedia. Además de utilizar la mensajería en modo texto, los usuarios de la libreta de contacto pueden crear grupos y enviarse mutuamente, imágenes, vídeos y grabaciones de audio.

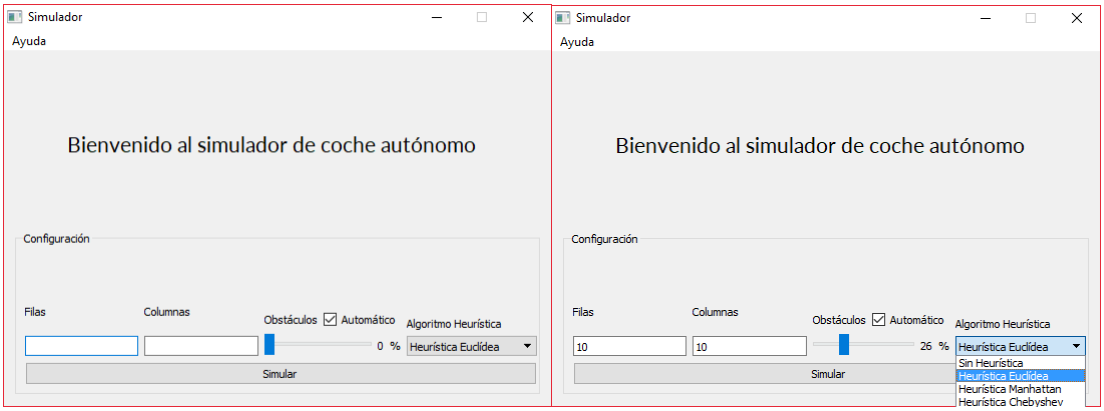
3. Interfaz Gráfica

A continuación explicaremos las funcionalidades de nuestro juego con sus correspondientes capturas de pantalla.

3.1 Pantalla principal

En ella tenemos:

- Dos casillas para introducir texto para indicar el número de filas y columnas.
- Un modo automático o manual para los obstáculos. Si elegimos el manual en la pantalla al presionar una celda se añadiría un obstáculo hasta pulsar "Siguiente".
- Una slider para poder seleccionar el porcentaje de obstáculos que queremos si elegimos el modo automático.
- Una lista de opciones para poder seleccionar el algoritmo de heurística: Hay 4 opciones:
    - Sin heurística.
    - Heurística Euclídea.
    - Heurística Manhattan.
    - Heurística Chebyshev.



1. Ventana para poder elegir la configuración de nuestro simulador

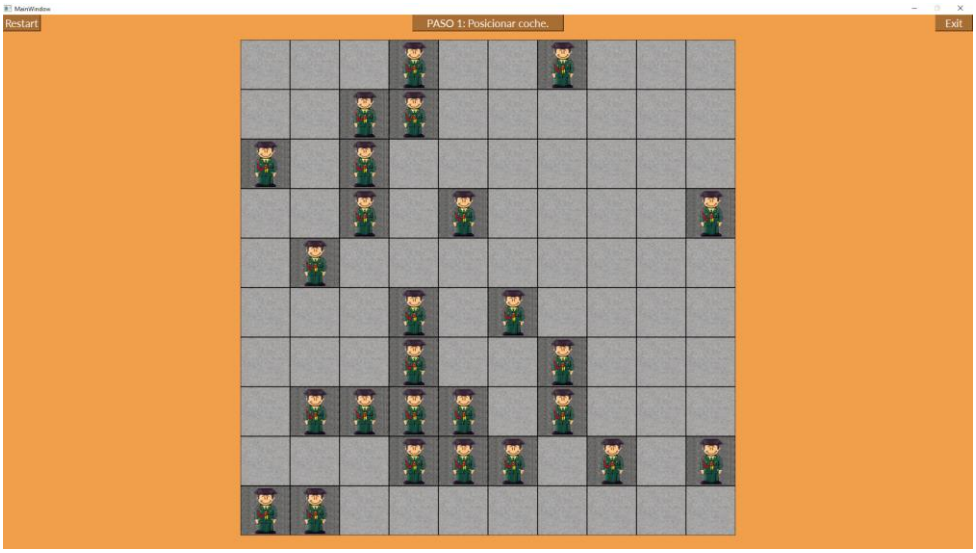
Tras presionar simular se nos abrirá la ventana de juego, donde tendremos tres botones en la parte superior de la pantalla.

3.2 Pantalla del tablero

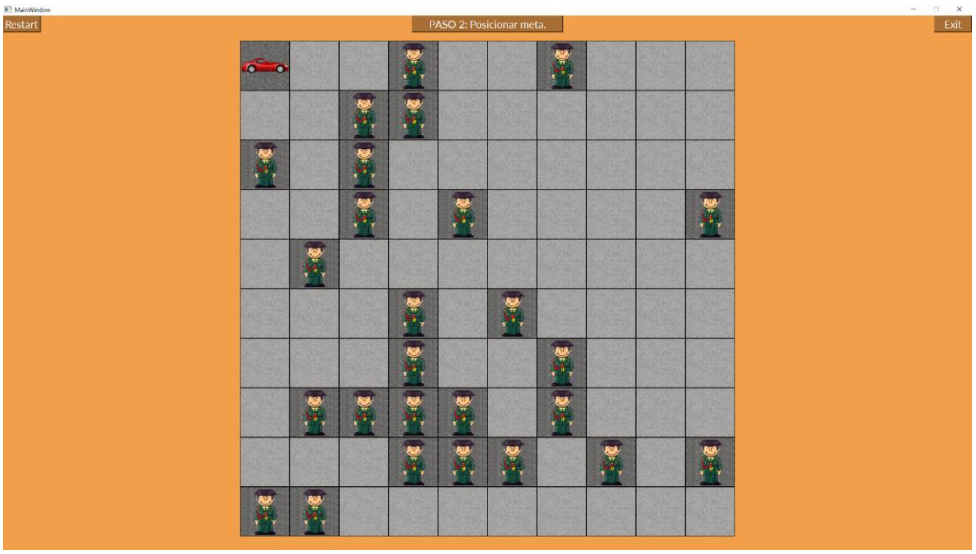
Esta es la pantalla principal del juego, donde se mostrará cómo se llega a la casilla destino y cuál es el camino mínimo para llegar a nuestro objetivo. Describimos los siguientes botones:

- *RESTART*: Sirve para volver a configurar el juego y cargarlo de nuevo.
- *EXIT*: Sirve para abandonar el juego.
- *PASO X*: Es un botón interactivo en el que va dando las instrucciones del juego y los pasos que se deben seguir. Tiene 4 estados:
    - Paso 1: Posicionar coche

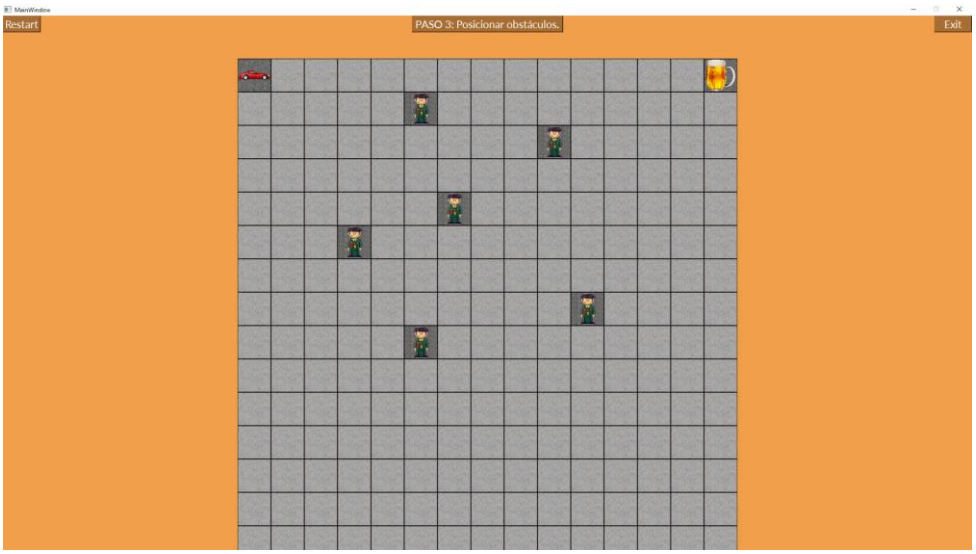
- Paso 2: Posicionar meta
- Paso 3: Elegir obstáculos (en caso de no haberlo puesto automático)
- Paso 4: ¡Pulse aquí para comenzar!



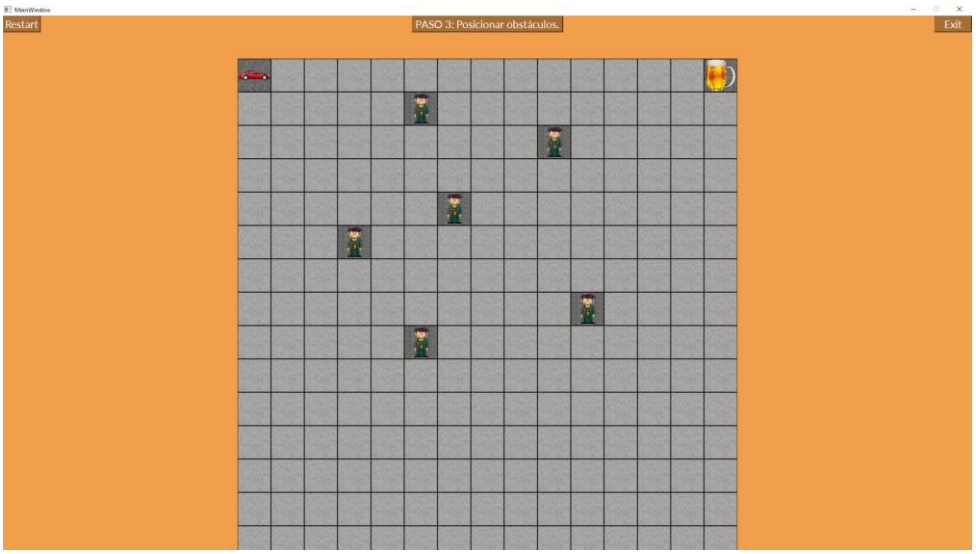
PASO 1: Posicionar coche.



PASO 2: Posicionar meta.



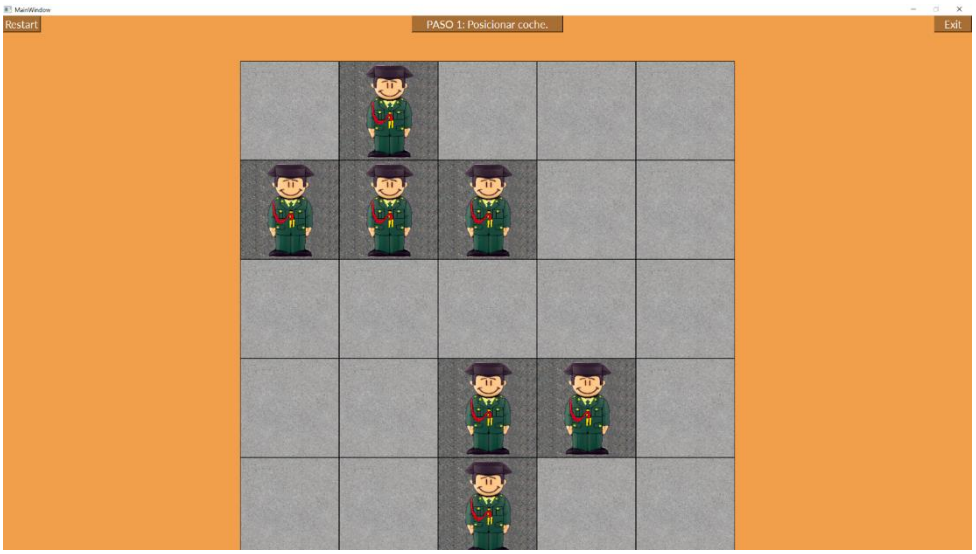
PASO 3: Posicionar obstáculos.



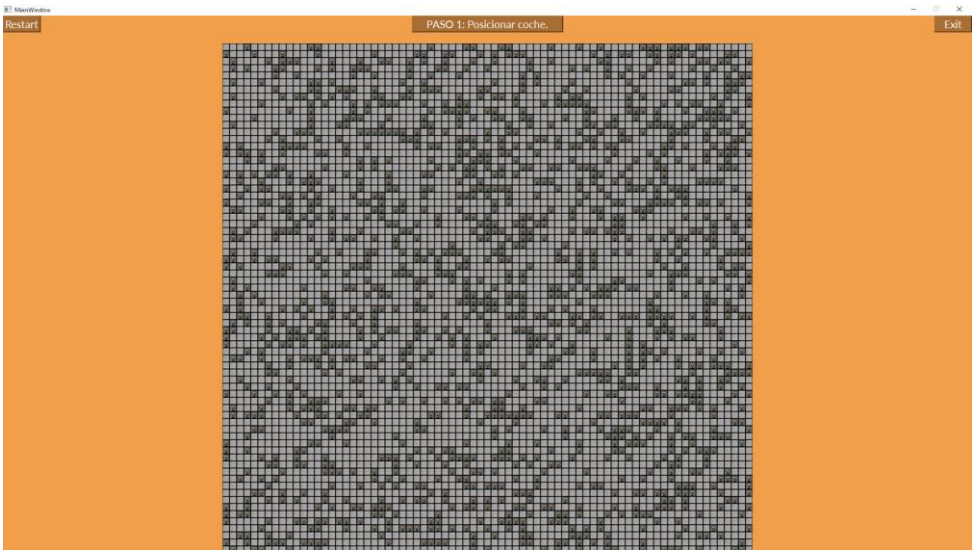
PASO 4: Pulse aquí para comenzar.

3.3 Información extra

Cabe destacar que nuestro simulador se adapta a cualquier tamaño que le introduzcamos, adaptando el tamaño de las casillas según el algoritmo que seguimos. Así, si ponemos un tablero de dimensiones pequeñas, las celdas se verán más grandes que las celdas de un tablero mayor.



1. Tablero 5 x 5



2. Tablero 70 x 70

4. ALGORITMO DE BÚSQUEDA

4.1 Descripción

El algoritmo de búsqueda que hemos implementado ha sido el A* pues es uno de los más algoritmos más utilizados a la hora de encontrar rutas y recorrer grafos de una forma eficiente. Esto se debe a su precisión y rendimiento. Sin embargo, en la práctica generalmente se eligen algoritmos que puedan preprocesar el grafo para ganar eficiencia.

A diferencia de algunos algoritmos de búsqueda en grafos en los que sólo se tiene en cuenta la función heurística, el A* es capaz de tener en cuenta dos factores, el valor heurístico de los nodos y el coste real del recorrido. De esta manera 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 hasta el final y g(n) el camino real recorrido para llegar a ese nodo.

Para la heurística se pueden elegir entre varios tipos, en nuestro programa los tenemos todos implementados:

- Heurística Euclídea.
- Heurística Manhattan.
- Heurística Chebyshev.

El algoritmo es una combinación entre búsquedas del tipo **DFS o BFS**, mientras **h'(n)** atiende primero en profundidad, **g(n)** atiende primero en anchura. De este modo el algoritmo es capaz de encontrar una mejor ruta cuando existan nodos más prometedores **(min(f'(n))**.

4.2 Complejidad computacional

La complejidad computacional depende de la calidad de la heurística utilizada. En el caso peor, la complejidad será exponencial, mientras que en el caso mejor, se ejecutará en tiempo lineal. Para ello se debe satisfacer lo siguiente.

$$h'(x) <= g(y)-g(x)+h'(y)$$

4.3 Pseudocódigo

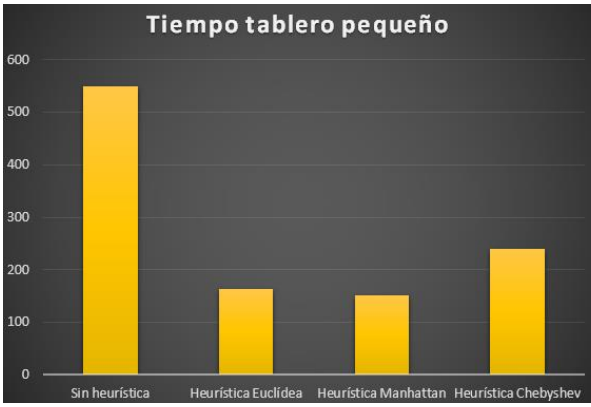
```
#   CADA NODO TIENE (F_SCORE, G_SCORE, H_SCORE, IS_START, IS_GOAL, IS_OBSTACLE,
    FATHER)
#
#   FUNCTION A*(START, GOAL)
#       CLOSEDSET := {} // CONJUNTO DE NODOS EVALUADOS
#       OPENSET := {START} // CONJUNTO DE NODOS NO EVALUADOS
#
#       WHILE OPENSET IS NOT EMPTY {
#           CURRENT_NODE :=  NODO EN OPENSET CON MENOR F_SCORE
#           IF CURRENT = GOAL {  RETURN RECONSTRUCT_PATH(CAMEFROM, CURRENT) } // SI
    ACABAMOS
#           OPENSET.REMOVE(CURRENT) CLOSEDSET.ADD(CURRENT) // LO MARCO
#
#           FOR EACH NEIGHBOR OF CURRENT
#               IF NEIGHBOR NOT IN OPENSET     // NUEVO NODO
#                   OPENSET.ADD(NEIGHBOR)
#                     TENTATIVE_GSCORE := GSCORE[CURRENT] + DIST_BETWEEN(CURRENT,
    NEIGHBOR)
#                   IF TENTATIVE_GSCORE >= GSCORE[NEIGHBOR] { CONTINUE } // NO ES
    MEJOR
#                   // ¡ES MEJOR CAMINO, LO GUARDAMOS!
#                      NEIGHBOR.FATHER := CURRENT
#                      GSCORE[NEIGHBOR] := TENTATIVE_GSCORE
#                     FSCORE[NEIGHBOR] := GSCORE[NEIGHBOR] + ESTIMATE_COST(NEIGHBOR,
    GOAL)
#   }
```

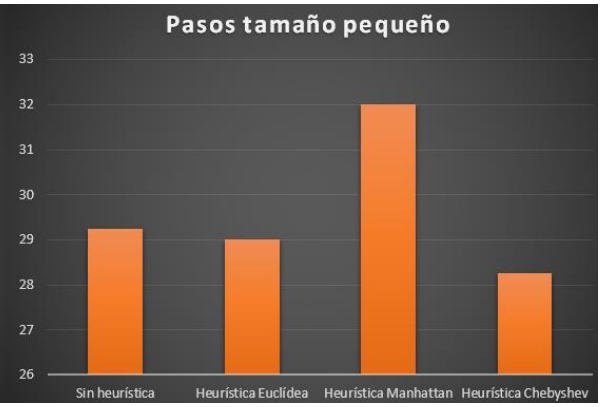
5. Estudio Experimental

Atendiendo a los diferentes tamaños del tablero y a las diferentes heurísticas usadas, obtenemos unos resultados diferentes a la hora de resolver el problema.

5.1 Tamaño pequeño (25 x 25)

Para esta ocasión hemos probado con un tablero de 25 filas y 25 columnas, rellenado con un 40% de obstáculos colocados automáticamente por nuestro programa. Obtuvimos los siguientes resultados.





Tamaño pequeño	
Pasos	25 x 25
28	574
28	527
30	562
31	538
Sin heurística	

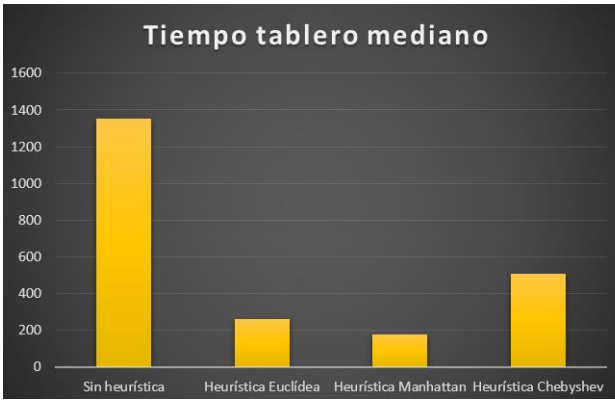
Tamaño pequeño	
Pasos	25 x 25
37	239
31	172
31	118
29	80
Heurística Manhattan	

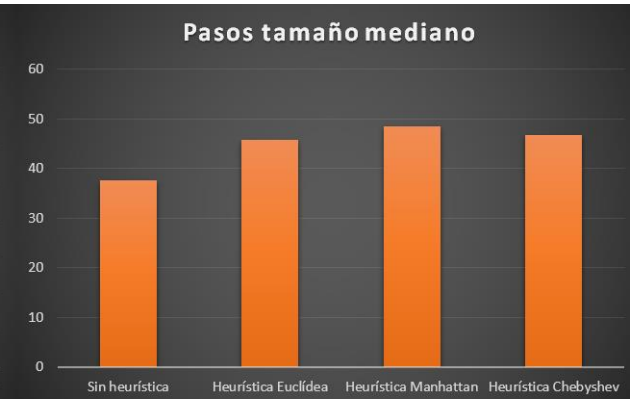
Tamaño pequeño	
Pasos	25 x 25
28	187
29	139
30	117
29	214
Heurística Euclídea	

Tamaño pequeño	
Pasos	25 x 25
28	208
27	225
30	270
28	258
Heurística Chebyshev	

5.2 Tamaño mediano (39 x 39)

Para esta ocasión hemos probado con un tablero de 39 filas y 39 columnas, rellenado con un 40% de obstáculos colocados automáticamente por nuestro programa. Obtuvimos los siguientes resultados.





Tamaño mediano		
Pasos	39 x 39	Tiempo (ms)
44		1390
46		1299
44		1327
16		1390
Sin heurística		

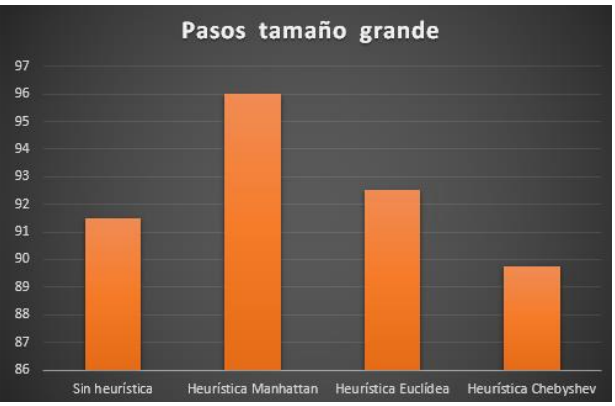
Tamaño mediano		
Pasos	39 x 39	Tiempo (ms)
52		174
50		193
47		173
45		172
Heurística Manhattan		

Tamaño mediano		
Pasos	39 x 39	Tiempo (ms)
45		227
42		191
47		236
49		398
Heurística Euclídea		

Tamaño mediano		
Pasos	39 x 39	Tiempo (ms)
48		622
46		354
48		567
45		497
Heurística Chebyshev		

5.3 Tamaño grande (77 x 77)

Para esta ocasión hemos probado con un tablero de 39 filas y 39 columnas, rellenado con un 40% de obstáculos colocados automáticamente por nuestro programa. Obtuvimos los siguientes resultados.



Tamaño grande		
Pasos	77 x 77	Tiempo (ms)
91		9584
93		9990
95		9576
87		9823
Sin heurística		

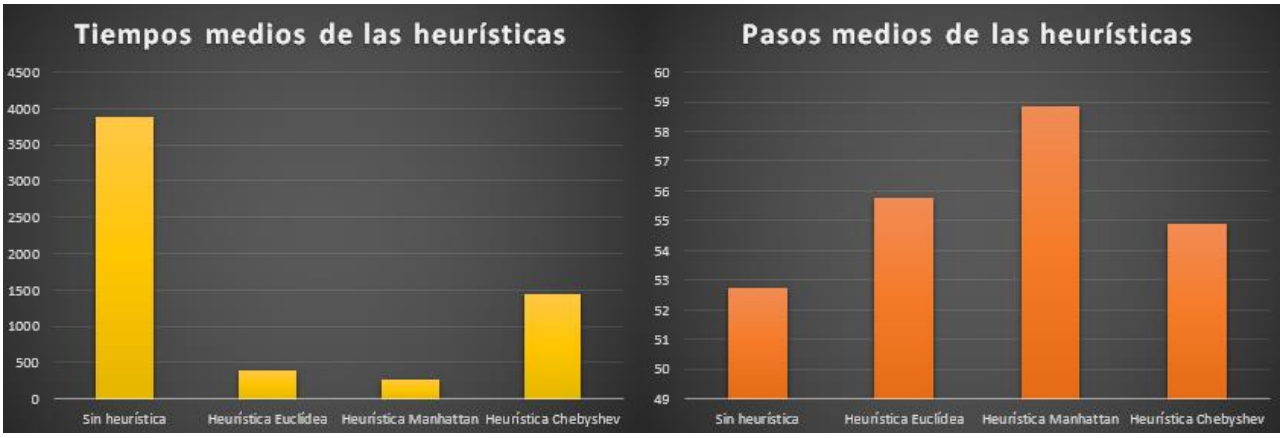
Tamaño grande		
Pasos	77 x 77	Tiempo (ms)
112		677
91		408
91		412
90		470
Heurística Manhattan		

Tamaño grande		
Pasos	77 x 77	Tiempo (ms)
91		632
95		874
92		762
92		726
Heurística Euclídea		

Tamaño grande		
Pasos	77 x 77	Tiempo (ms)
91		3950
91		3291
85		3093
92		4037
Heurística Chebyshev		

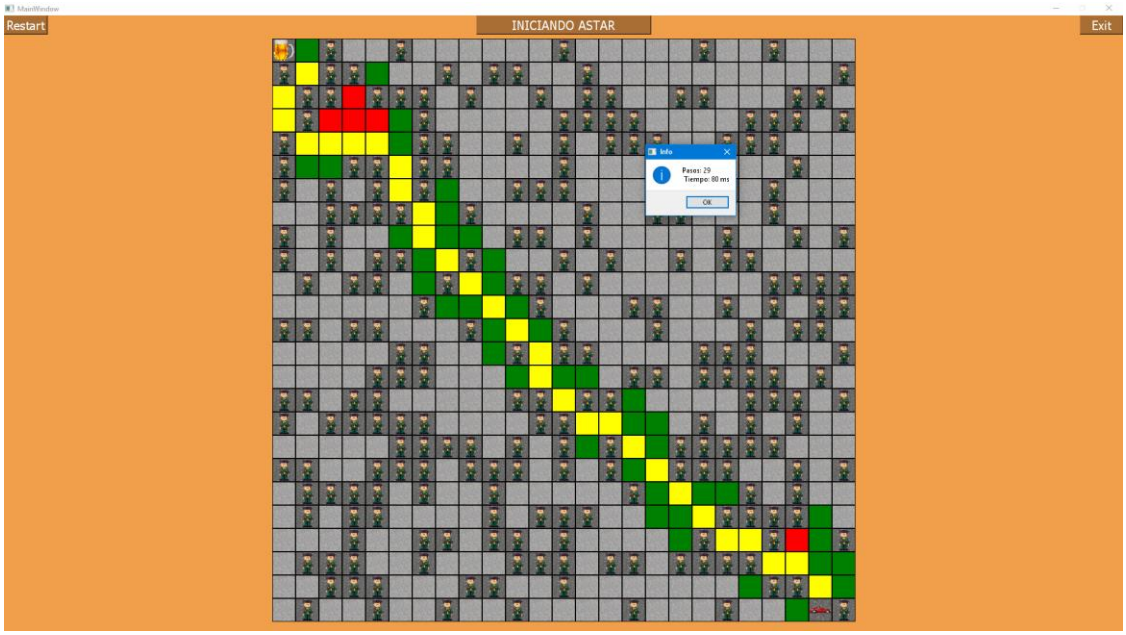
5.4 Estadísticas de las heurísticas

Después de realizar las pruebas anteriores, concluimos que en líneas generales la heurística Manhattan es la más rápida de todas, seguida por la euclídea. Esta primera tiene un inconveniente que hemos apreciado en las estadísticas y es que generalmente realiza un mayor número de pasos.



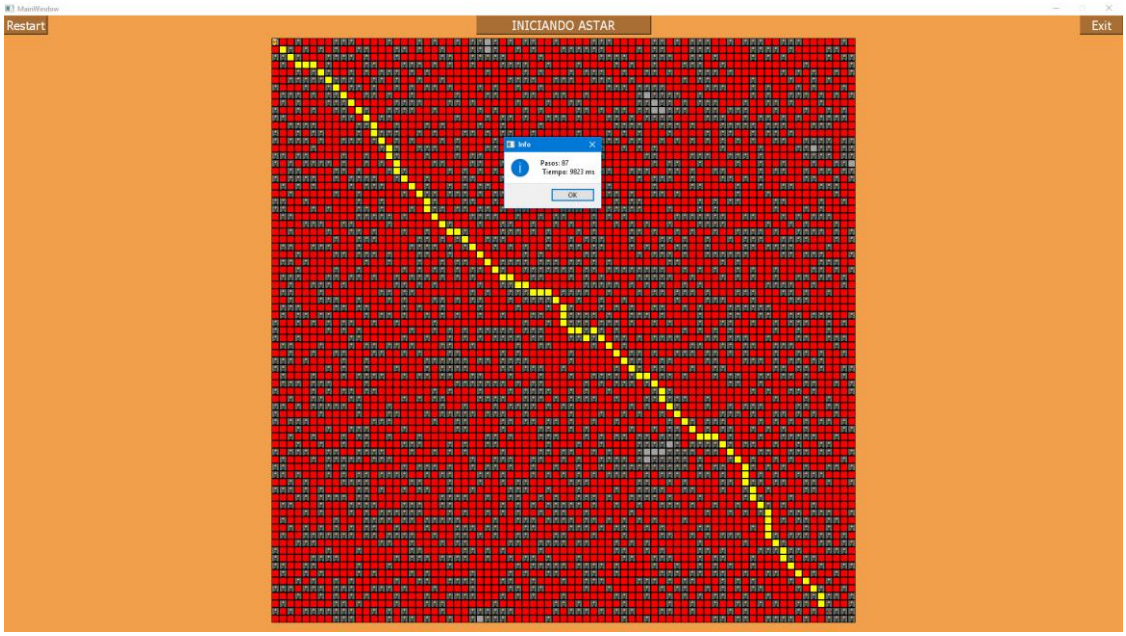
5.5 Prueba de la eficiencia del algoritmo

Se da constancia de que el algoritmo es eficiente para encontrar el camino mínimo independientemente de la heurística utilizada (en este caso fue la Manhattan).



5.6 Prueba de ineficiencia del algoritmo

No se constató en ninguna prueba que se realizó que el algoritmo fuera ineficiente, sin embargo



6. CONCLUSIONES

El proyecto que hemos desarrollado es eficiente, encontrando en todos los casos el camino más corto para llegar al destino.

Usando siempre el mismo algoritmo de búsqueda (A*) conseguimos unos buenos resultados, aunque no depende sólo del algoritmo de búsqueda sino de la función heurística que empleemos, así, si no usamos ninguna función heurística el tiempo para calcular la ruta es considerablemente mayor, por lo cual, el uso de cualquiera de las heurísticas supone una mejora importante del rendimiento en contraste con lo que pasa si no usamos ninguna de ella.

En líneas generales, concluimos que la función heurística "Distancia Manhattan" es la más eficiente en cuánto a tiempo de cómputo. Nótese que los tiempos cuando usamos Manhattan en un tablero grande son equivalentes a no usar ninguna heurística en el tablero más pequeño en el que realizamos las pruebas.

7. BIBLIOGRAFÍA

Se han usado los siguientes recursos para la elaboración de este proyecto

1. *Página web de QT*
   - http://doc.qt.io/qt-5/qtexamplesandtutorials.html

2. *Wiki para iniciarse en QT*
   - https://wiki.qt.io/Qt_for_Beginners

3. *Información a acerca del algoritmos de búsqueda y su pseudocódigo*
   - https://es.wikipedia.org/wiki/Algoritmo_de_b%C3%BAsqueda_A*

4. *Video tutoriales para el uso de QT y su interfaz gráfica*
   - https://www.youtube.com/playlist?list=PLS1QulWo1RIZiBcTr5urECberTITj7gjA