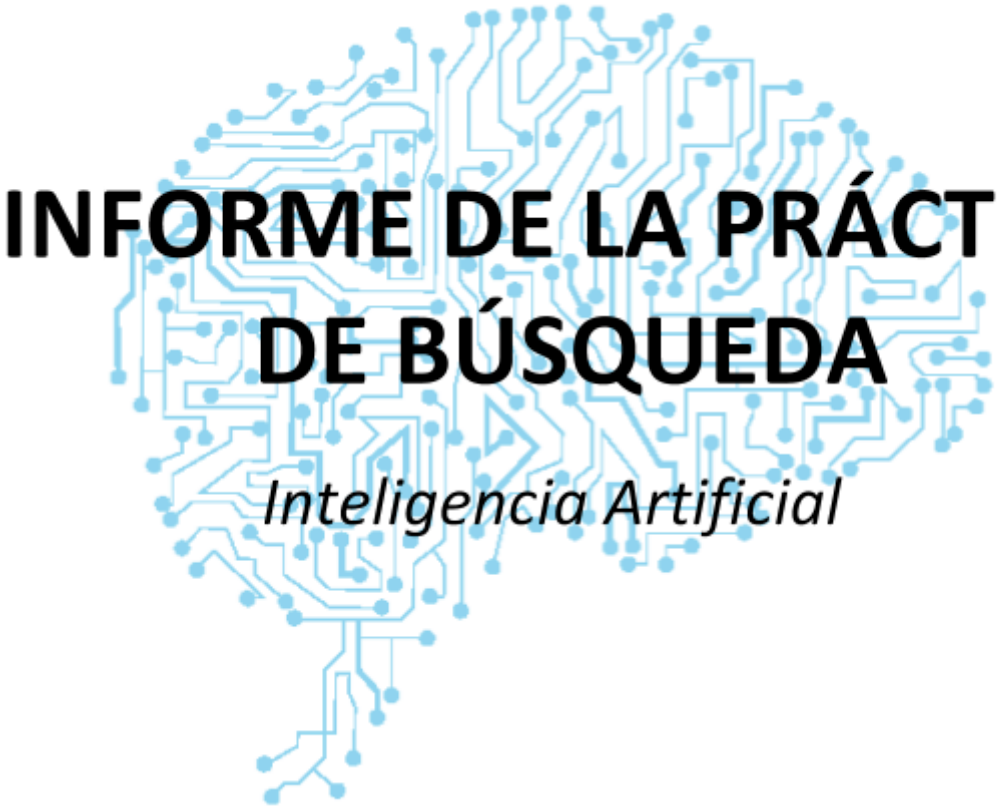


INFORME DE LA PRÁCTICA DE BÚSQUEDA

Inteligencia Artificial

EUGENIO JOSÉ GONZÁLEZ LUIS

MANUEL JESÚS PERAZA ALONSO

JESÚS RAMOS ÁLVAREZ

Índice

Introducción

El problema propuesto para llevar a cabo a lo largo de 6 semanas ha sido la resolución del camino óptimo, haciendo uso de las estrategias de búsqueda, por parte de un coche dispuesto sobre una matriz de dimensiones M x N constituida por celdas libres y ocupadas. Este coche podrá efectuar movimientos (Norte, Sur, Este y Oeste) en las casillas que no se encuentren ocupadas (obstáculos). El coche debe tener un sensor de proximidad que le permita saber cuáles de sus posibles casillas de movimiento está siendo ocupada por un obstáculo. La importancia de esta práctica está en la posible extrapolación a un coche autónomo.

La repartición de trabajo ha sido algo difícil debido a que casi siempre se tenía que hacer de manera conjunta puesto que al principio necesitábamos todos los datos y estructuras perfectamente claras para el grupo y así poder realizar funciones de manera individual posteriormente.

En una práctica como esta, en la que realmente el código no es tan extenso, es complicado atribuir las tareas ya que cada uno aportó lo que pudo en el momento que podía.

La coordinación de este grupo ha sido supervisada por Eugenio ayudándose de herramientas como github y explicando a los miembros de su grupo el sistema de versiones. Nuestro archivo de github siempre tenía en su rama master una versión funcional de la práctica que era usada a modo de plantilla cada vez que se requería añadir alguna funcionalidad al código, estos códigos eran editados en cada una de nuestras ramas (1 para cada uno de nosotros y una para el cc donde estaba lo que hubiéramos hecho en las clases práctica) Cada vez que la funcionalidad fuera terminada se añadía con un merge a la rama master.

Herramientas utilizadas

Java

Java es un lenguaje de programación de propósito general, concurrente, imperativo y orientado a objetos. Está fuertemente influenciado por otros lenguajes como Pascal o C++.

Uno de los principales motivos para usar Java frente a otros lenguajes fue su similitud con la sintaxis de C++ por lo que los miembros del grupo no se encontrarán perdidos pero al mismo tiempo evitaría tener que trabajar con código a bajo nivel de manera innecesaria.

A su vez la gran cantidad de librerías integradas en el JDK facilitan respecto a otros lenguajes de programación en gran manera la creación de elementos más complejos como el entorno gráfico gracias a Swing y AWT.

Otros motivos para usar Java fueron el sencillo manejo de hilos, la baja cantidad de líneas de código necesarias para escribir un programa en Java frente a otros lenguajes, la no necesidad de controlar la carga de la memoria (pues el recolector de basura lo hace de forma automática) y la facilidad para ejecutar el mismo código en múltiples máquinas aunque estas usen sistemas operativos distintos, pudiendo incluso ejecutarse en web o en plataformas móviles sin necesidad de cambiar ni una sola línea de código.

Eclipse

Eclipse es un IDE, entorno de desarrollo integrado, es decir, una aplicación que facilita la producción de software mediante la integración de servicios para hacer más sencilla la labor del programador.

Concretamente el IDE Eclipse está basado en módulos, que se pueden añadir o eliminar a gusto del programador, a diferencia de otros IDE que tienen una serie de funcionalidades determinadas.

El motivo principal que nos ha hecho decantarnos por Eclipse para el desarrollo de nuestra aplicación ha sido que, puesto que íbamos a utilizar Java como lenguaje de programación, necesitábamos un entorno de desarrollo con herramientas que nos hicieran más cómoda la programación en dicho lenguaje y Eclipse nos ofrecía todo lo que queríamos.

Un editor de texto con funciones de autocompletado y detección de errores para el lenguaje utilizado, un sistema muy intuitivo para la depuración del código, posibilidad de ejecutar la aplicación directamente desde el IDE, integración con Github y facilidad a la hora de documentar el código mediante javadoc.

Github

Github es una plataforma de desarrollo colaborativo en línea que proporciona multitud de herramientas para la gestión de proyectos de desarrollo de software. Está basado en un sistema de repositorios en los que alojar todos los archivos de tu proyecto, código, documentación, etc.

Al disponer de todos esos archivos en una plataforma en línea permite que la compartición entre los distintos desarrolladores sea mucho más sencilla, pues sólo tendrán que acceder a la web de Github para tener acceso a dichos archivos. Muchas herramientas de desarrollo como Eclipse, de la que hablamos anteriormente, proporciona además un sistema de integración con Github, permitiendo directamente desde el IDE llevar a cabo la modificación y actualización de los archivos almacenados en Github.

Sin embargo, Github no es sólo un lugar para almacenar código. Uno de los servicios que lo hacen una herramienta tan útil a la hora de desarrollar software es su sistema de control de versiones basado en git. Esto permite a los colaboradores del proyecto, entre otras cosas, trabajar en distintas ramas y luego fusionarlas, comprobando que no existan conflictos o en el caso de que existan saber cuál es su causa, o tener siempre disponibles las versiones anteriores del programa en desarrollo.

Aparte de esto, Github también permite la creación de issues, lo que hace posible llevar un control de los diversos problemas que hayan surgido durante la creación del software, o el de crear una página web de forma fácil y rápida que contenga toda la información de tu proyecto.

En nuestro caso decidimos hacer uso de Github principalmente por la facilidad a la hora de compartir los archivos, ya que en muchas ocasiones hemos trabajado cada uno en distintos sitios, por la integración que tiene con Eclipse y por el control de versiones.

Google docs

Para la realización del informe final de la práctica hemos hecho uso del procesador de texto que proporciona Google docs. Decidimos esto debido a que, aunque no disponga de todas las funcionalidades que pueden tener otros procesadores de texto, si nos aporta las necesarias para

realizar un informe de este tipo, y sobre todo, lo más útil de Google docs es su sistema de guardado automático en la nube y la posibilidad de que todos los usuarios con permiso de edición del documento puedan trabajar de forma simultánea. Además de esto, incluye la utilidad de poder descargar el documento directamente en pdf que es el formato que se requiere en la entrega.

Interfaz

La interfaz de nuestra aplicación consta de las siguientes funcionalidades:

1. GUI
    a. O modo consola (ambos completos)
2. Creación de la matriz
    a. Aleatoria
    b. Manual
    c. Por fichero
3. Modificación “en vivo” del porcentaje de obstáculos
4. Display de información sobre la matriz actual (Tamaño, porcentaje, tiempo para resolver, pasos para resolver)
5. Selección de la velocidad de movimiento del coche
6. Seguimiento del coche tras la resolución
7. Efectos de audio
8. Selección del método para calcular las distancias (manhattan, euclídea y Mahalanobis)

GUI

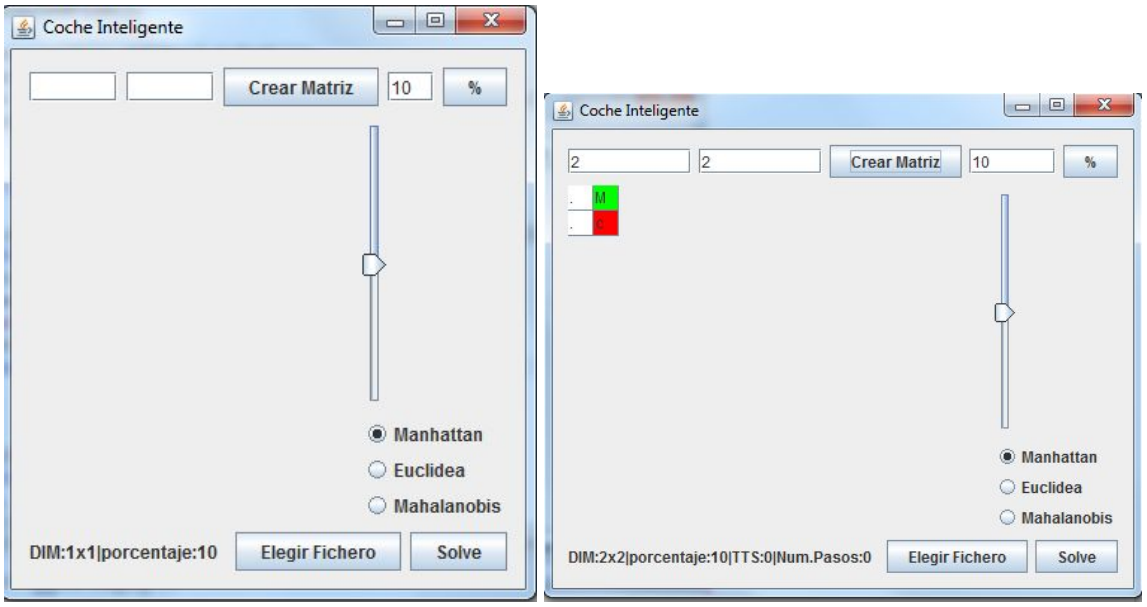
Nuestra aplicación se puede ejecutar con dos visualizaciones diferentes dependiendo del parámetro proporcionado a la hora de la ejecución. Por un lado está el modo interfaz gráfica (modo por defecto) y por otro el modo consola.

- Modo interfaz gráfica

Al ejecutar el programa en este modo interfaz gráfica inicial muestra, de izquierda a derecha y de arriba a abajo los siguientes campos:

- Dos espacios de introducción de texto, que corresponden al alto y ancho de la matriz.
- Un botón “Crear Matriz”.
- Otro campo de introducción de texto, para establecer el porcentaje de obstáculos.
- Un botón “%”.
- Un control deslizante para modificar la velocidad del coche.
- Un grupo de botones para seleccionar el método de cálculo de distancia.
- Un campo de texto con información relevante sobre la matriz.
- Un botón “Elegir fichero”.
- Un botón “Solve”.

Tras la creación de una matriz la interfaz sufre una ligera modificación, aparece la matriz creada y al campo de texto, barras de desplazamiento (si la matriz supera el tamaño de la ventana) y se añade más información sobre la ejecución actual al campo de texto.



A la izquierda, ejecución inicial en modo interfaz gráfica, a la derecha, tras crear una matriz

- Modo consola

Si ejecutamos la aplicación en modo consola lo primero que se nos muestra en consola es una menú con las siguientes opciones:

1. Constructor aleatorio

Al seleccionar esta opción solicita al usuario un tamaño N y M y crea una matriz de este tamaño, con el último porcentaje de obstáculos indicado en el que las posiciones de estos obstáculos, el coche y la meta son aleatorias.

2. Constructor manual

Permite aparte del tamaño, elegir el número de obstáculos y las coordenadas para estos, para el coche y para la meta.

3. Constructor por fichero

Permite crear una matriz a partir de un fichero determinado indicandole la ruta del archivo.

4. Elegir porcentaje de obstáculos aleatorios

Esta opción establece el porcentaje de obstáculos con el que se crearán las nuevas matrices.

5. Resolver

Resuelve el problema de la última matriz creada mediante el uso del método seleccionado.

6. Elegir método para el cálculo de distancias

Permite seleccionar el método para la resolución de la matriz entre calcular la distancia según Manhattan, Mahalanobis o por distancia Euclídea.

7. Salir

Cierra la aplicación.

```
1. Constructor aleatorio
2. Constructor manual
3. Constructor por fichero
4. Elegir porcentaje de obstaculos aleatorios
5. Resolver
6. Elegir metodo
7. Salir
Escribe una de las opciones
```

Menú principal del modo consola

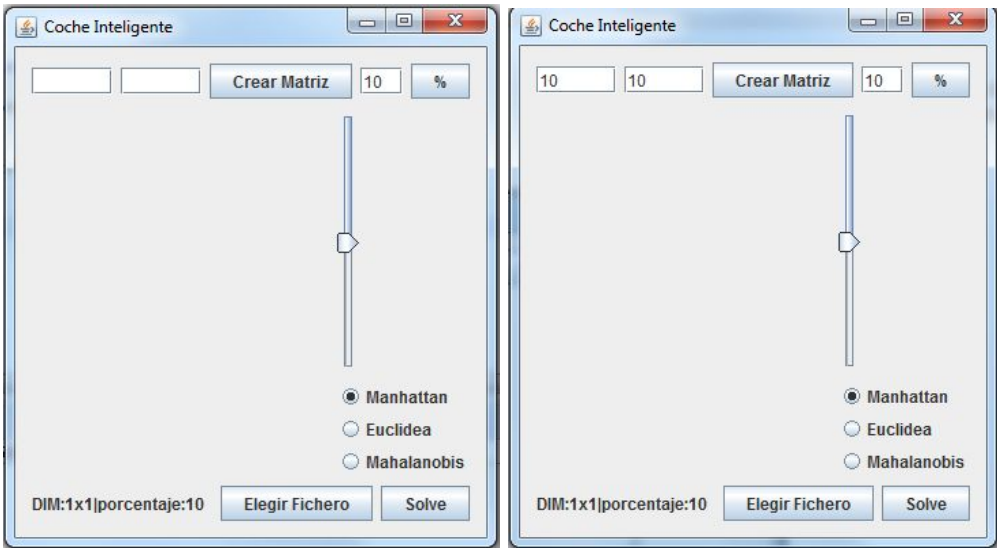
Creación de la matriz

Tanto en este como en los siguientes apartados en los que se detallan las funcionalidades estas serán explicadas únicamente para el modo con interfaz gráfica debido a que el modo consola ya ha sido completamente explicado en el apartado correspondiente a la GUI.

A la hora de crear una nueva matriz los usuarios pueden hacerlos de tres formas diferentes: aleatoria, manual y por fichero.

- Aleatoria

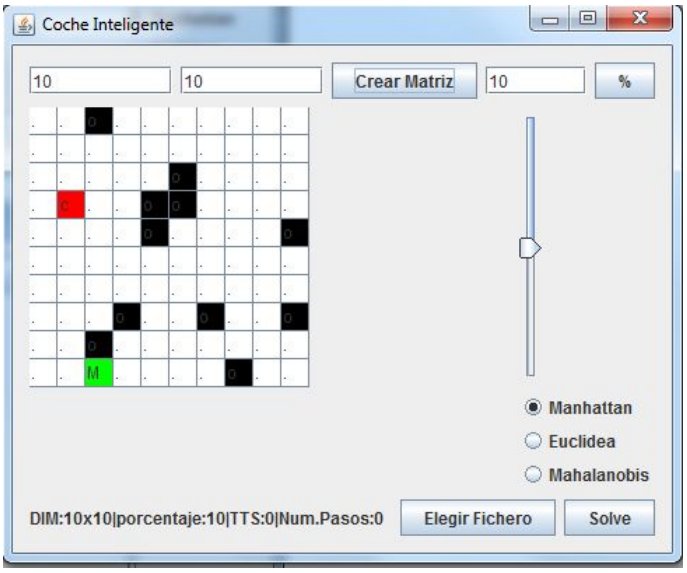
Para la creación aleatoria simplemente será necesario rellenar los dos campos de introducción de texto que corresponden al tamaño de la matriz y modificar, en caso de que se quiera, el porcentaje de obstáculos por defecto (10%) cambiando el campo correspondiente y haciendo click en el botón "%".



Estableciendo los valores del alto y ancho de la matriz

Una vez hecho esto, al pulsar "Crear Matriz" la matriz será creada teniendo en cuenta los valores anteriores y aparecerá en pantalla. En la matriz aleatoria creada podrán aparecer los siguientes elementos:

- Coche: casilla de color rojo y letra c.
- Meta: casilla de color verde y letra M.
- Obstáculo: casilla de color negro y letra o.
- Casilla libre: casilla de color blanco y símbolo ".".

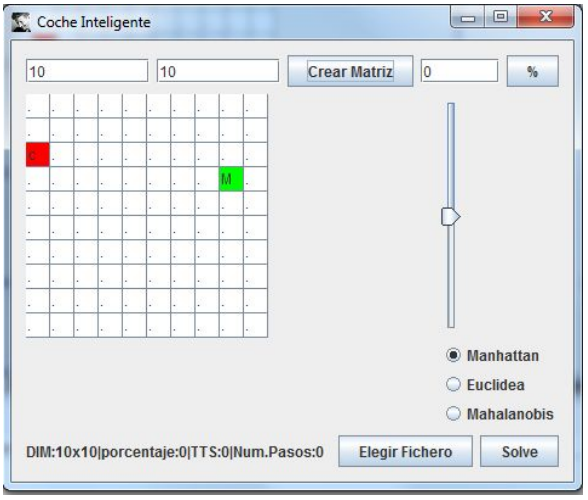


Matriz creada de forma aleatoria

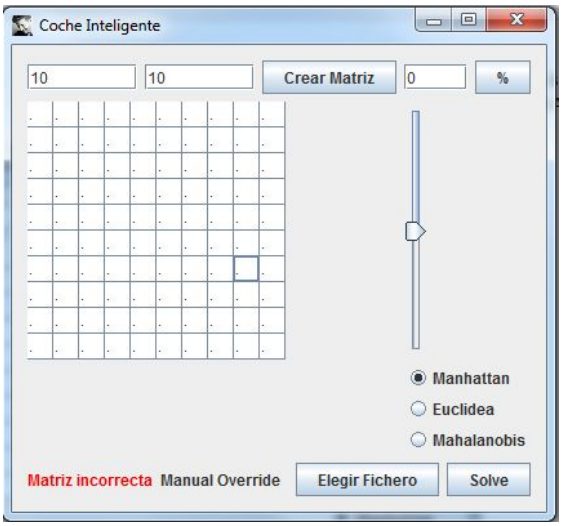
- Manual

Para la creación manual, lo primero es indicar el tamaño y clicar en “Crear Matriz”. Una vez aparezca la matriz o si ya había una en pantalla el usuario podrá modificar los elementos que aparecen en ella a su antojo, seleccionando y reescribiendo las distintas casillas siguiendo siempre la sintaxis de la aplicación (c = coche, M = meta, o = obstáculo, . = casilla libre).

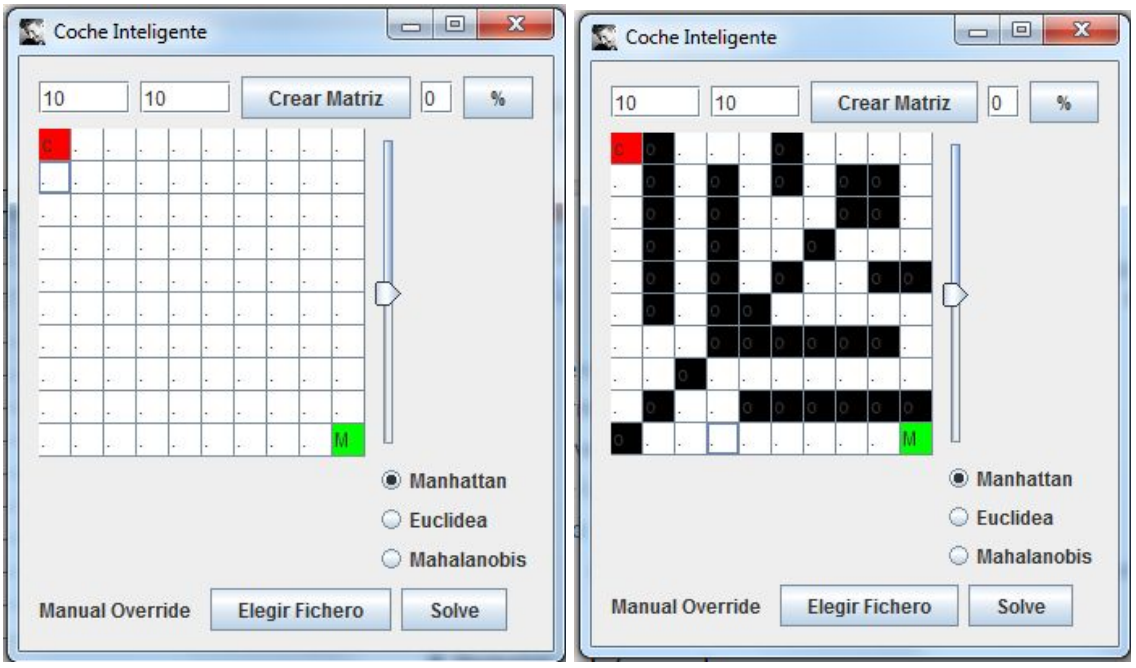
El programa detectará si una matriz no cumple con las condiciones necesarias que son: que exista un único coche, que haya una sola meta y que la sintaxis sea la correcta. En caso de no cumplir las condiciones, en el apartado de texto de información de la interfaz se mostrará un mensaje de error.



Creando matriz con tamaño específico



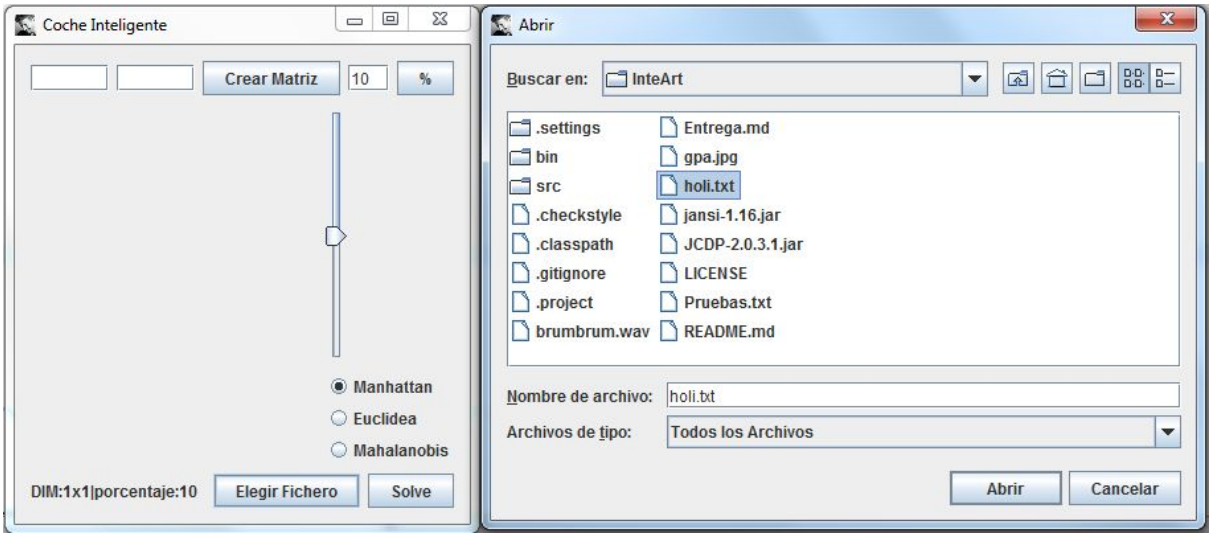
Mensaje de error al detectar que no existe coche ni meta



Añadidos primero coche y meta (ya la matriz es correcta) y luego los obstáculos de forma manual

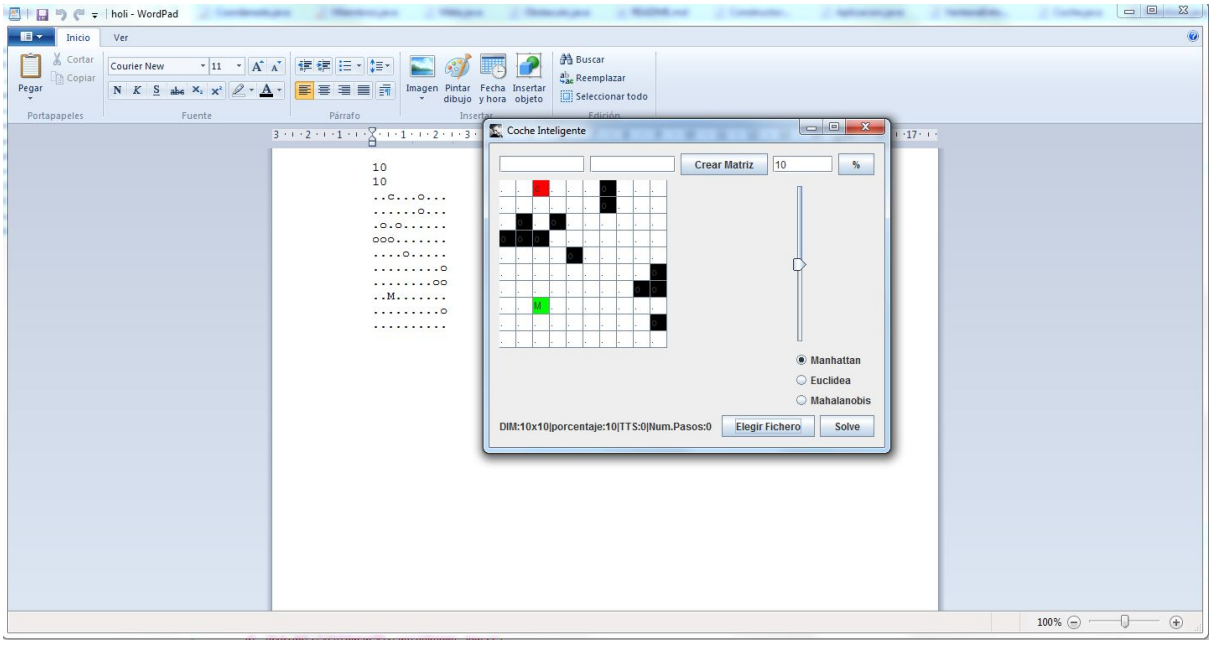
- Por fichero

Si el usuario desea crear su matriz a partir de un fichero de texto debe seleccionar la opción “Elegir Fichero”. Esto permitirá a dicho usuario buscar un archivo en su equipo y si este archivo sigue el modelo establecido que requiere el programa creará la matriz a partir de él.



Selección del archivo a partir del cual crear la matriz

El fichero debe seguir el modelo que se puede observar en la siguiente captura, el primer número es el alto de la matriz, el segundo el ancho y respetando estos valores se debe dibujar una matriz siguiendo la sintaxis explicada con anterioridad.

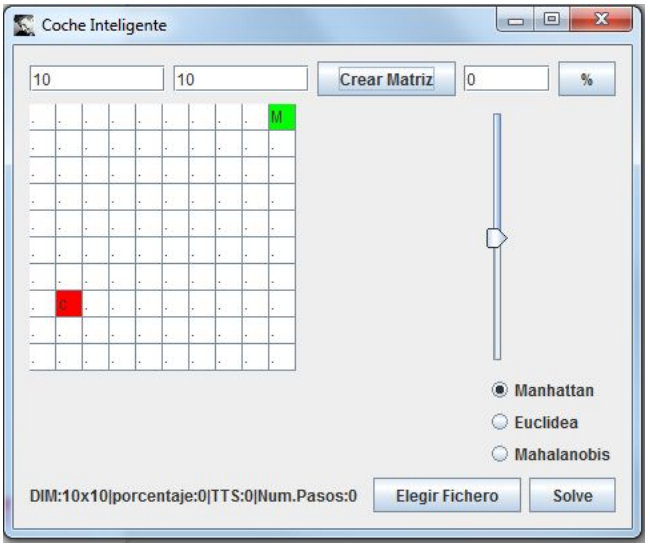


Matriz creada y fichero correspondiente a ella

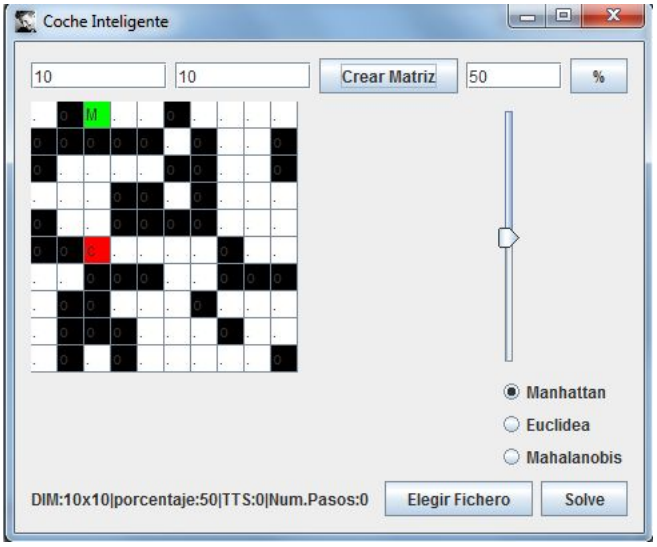
Modificación “en vivo” del porcentaje de obstáculos

En cualquier momento el usuario antes de crear una nueva matriz puede cambiar el porcentaje de obstáculos (por defecto 10%). Este porcentaje se mantendrá para todas las nuevas matrices hasta que se establezca uno diferente o se cierre la aplicación.

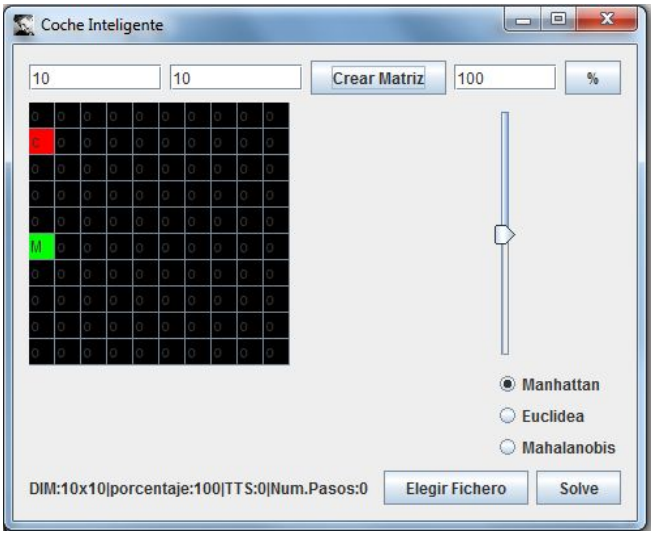
Para realizar este cambio de porcentaje en el campo de introducción de texto que le corresponde se introduce en valor deseado y se hace click en “%”, la siguiente matriz aparecerá con el nuevo porcentaje.



Matriz aleatoria creada con un 0% de obstáculos



Matriz aleatoria creada con un 50% de obstáculos



Matriz aleatoria creada con un 100% de obstáculos

Display de información sobre la matriz actual

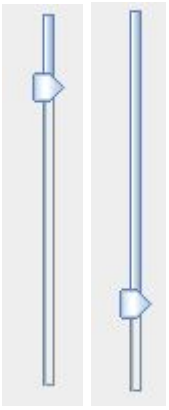
El display de información está formado por cuatro campos:

- DIM: Muestra la dimensión de la matriz de la forma NxM.
- Porcentaje: Muestra el porcentaje actual de obstáculos.
- TTS: Muestra el tiempo en milisegundos que se tardó en resolver el problema.
- Pasos: Muestra el número de pasos necesarios hasta resolver el problema.

Display de información

Selección de la velocidad de movimiento del coche

Mediante un control deslizante el usuario va a ser capaz de interactuar con la velocidad a la que se desplaza el coche. Cuanto más cerca esté el selector de la parte superior menos milisegundos tardará en realizar cada paso, es decir, la velocidad del coche aumenta. Por otro lado, si el selector se desplaza hacia la parte inferior del control deslizante sucederá lo contrario.



Control deslizante

Encontrar la solución

Tras tener la matriz deseada en pantalla pulsar el botón “Solve” hará que se resuelva el problema usando el algoritmo y el método de calcular la distancia escogido.

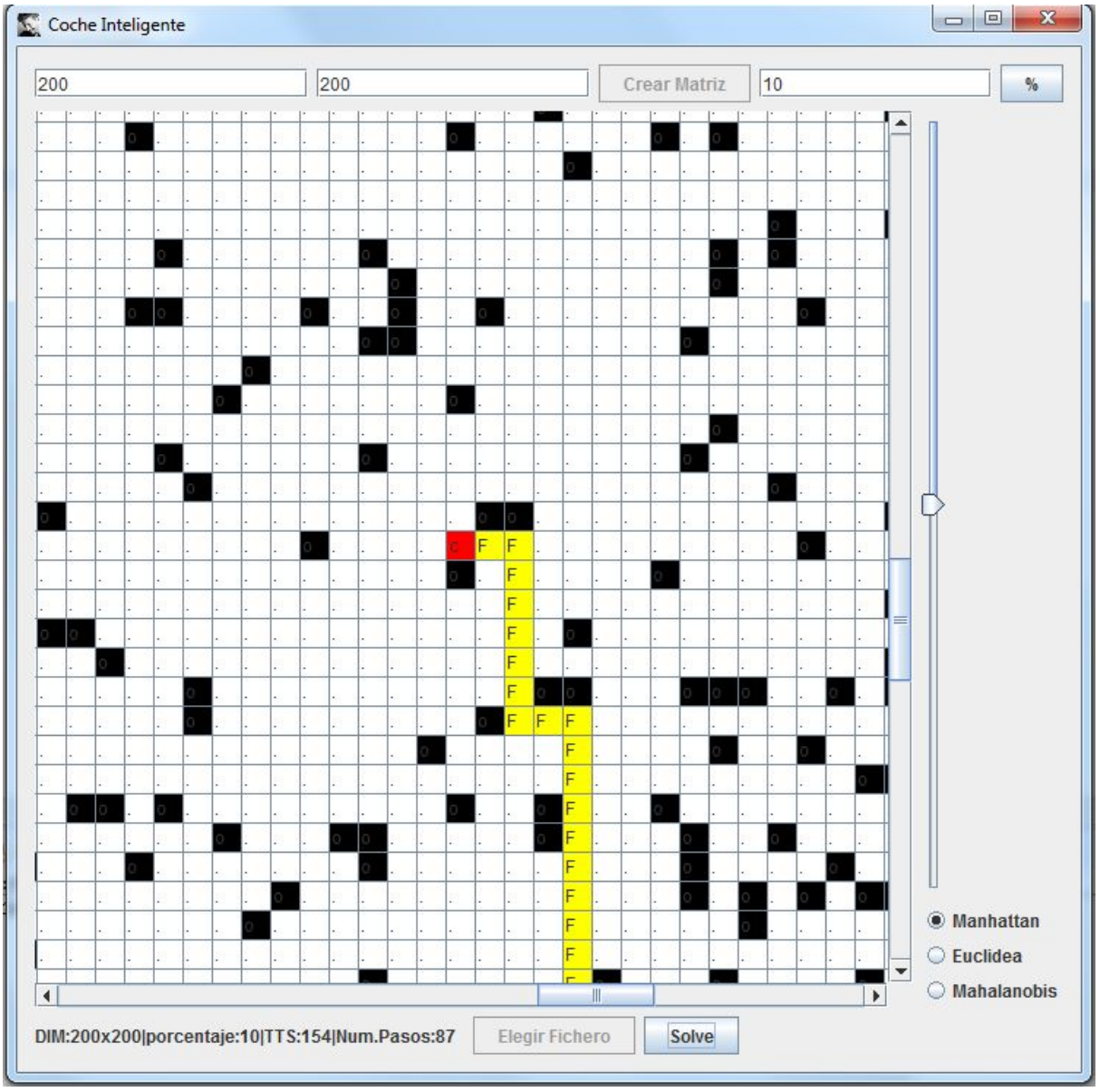
Una vez se haya encontrado solución el coche se actualizará la información correspondiente al tiempo y a los pasos y el coche empezará a avanzar, el camino marcado con la letra F y de color amarillo indica el camino óptimo según el algoritmo.

Si el problema no tiene solución se mostrará un mensaje indicando el error.

Interfaz gráfica después de resolver el problema

Seguimiento del coche tras la resolución

Esta funcionalidad de nuestra aplicación se ejecuta cuando las matrices superan el tamaño de la ventana y hace que en todo momento en el coche se encuentre en movimiento este se sitúe en el centro de la misma, es decir, se hace un seguimiento del recorrido del coche. Esto unido al hecho de que tenga un color específico evita el poder perder el coche de vista en matrices muy grandes.

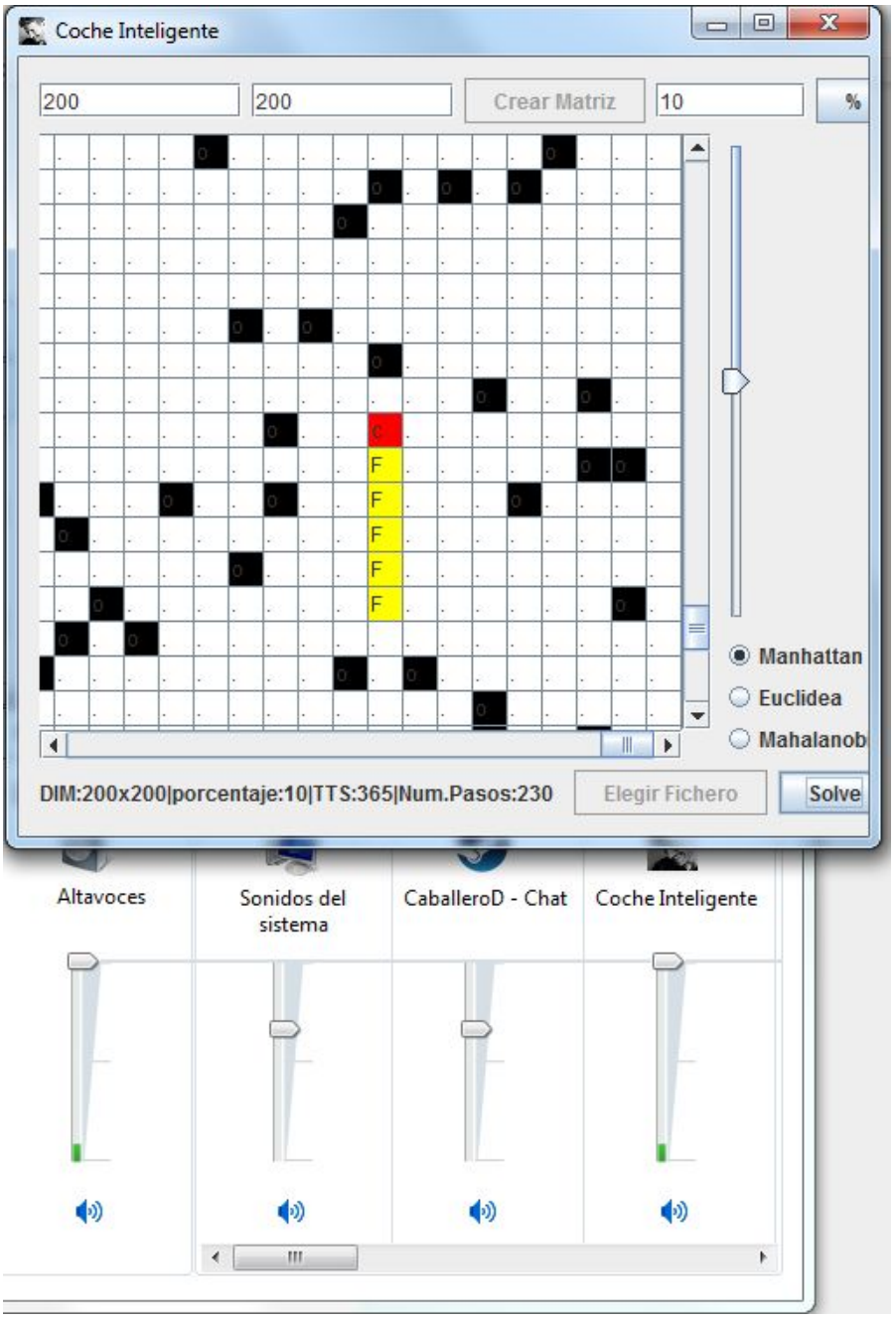


Seguimiento del coche (se observa el coche en el centro) en una matriz de 200x200

Efectos de audio

En nuestra aplicación aparte de lo visual hemos decidido incluir un efecto sonoro correspondiente a un coche. Gracias a esto conseguimos una experiencia más completa para el usuario y que realmente imagine que es un coche el que realiza el camino.

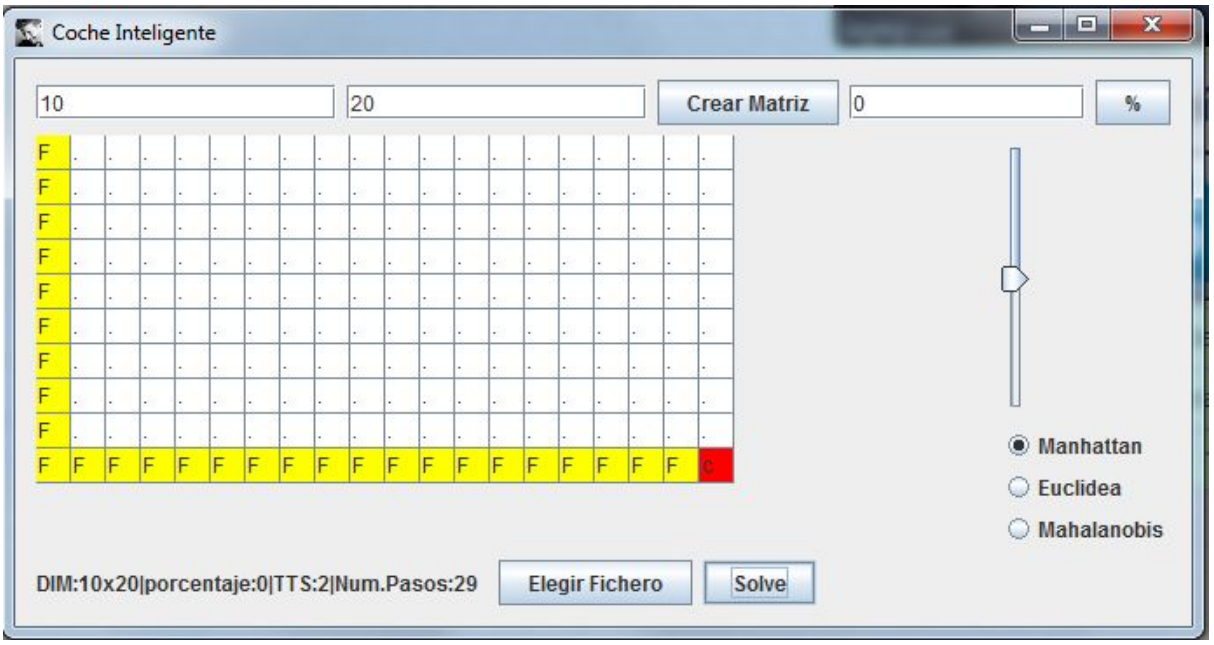
También sirve a modo de notificación, pues como el algoritmo en algunas ocasiones tarda un tiempo en resolver el problema, en el momento en el que se escucha el sonido significa que se ha resuelto el con éxito y que el coche ha arrancado y se dirige hacia su destino.



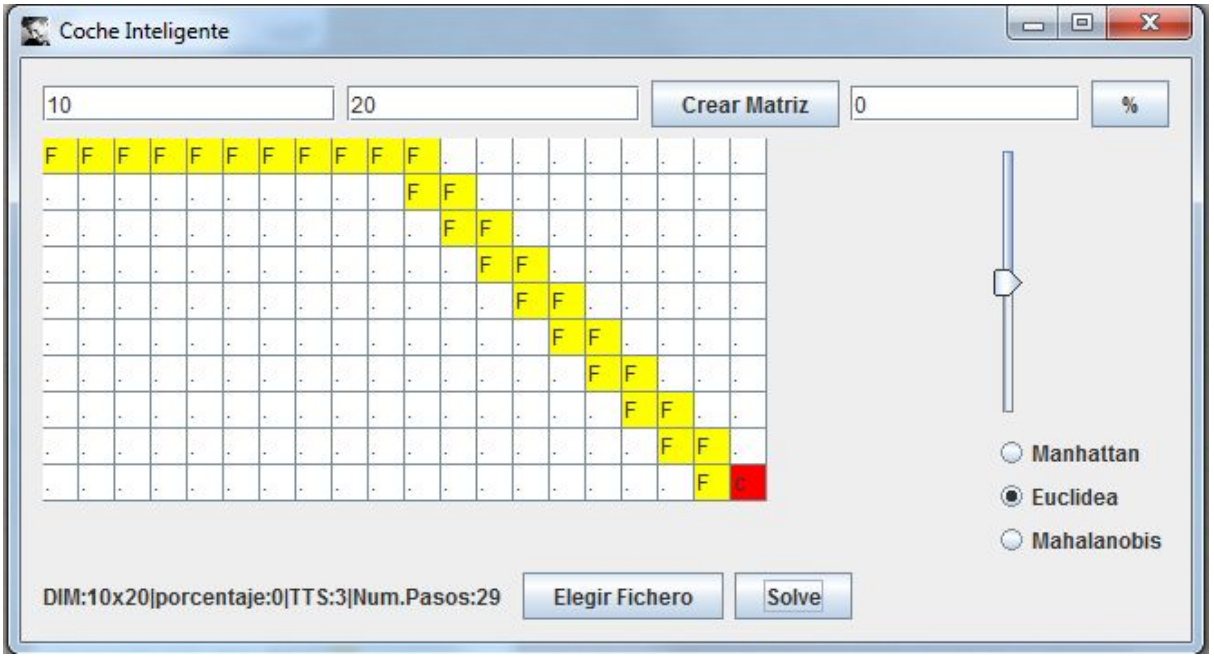
Aplicación y mezclador de volumen de Windows

Selección del método para calcular las distancias (manhattan, euclídea y Mahalanobis)

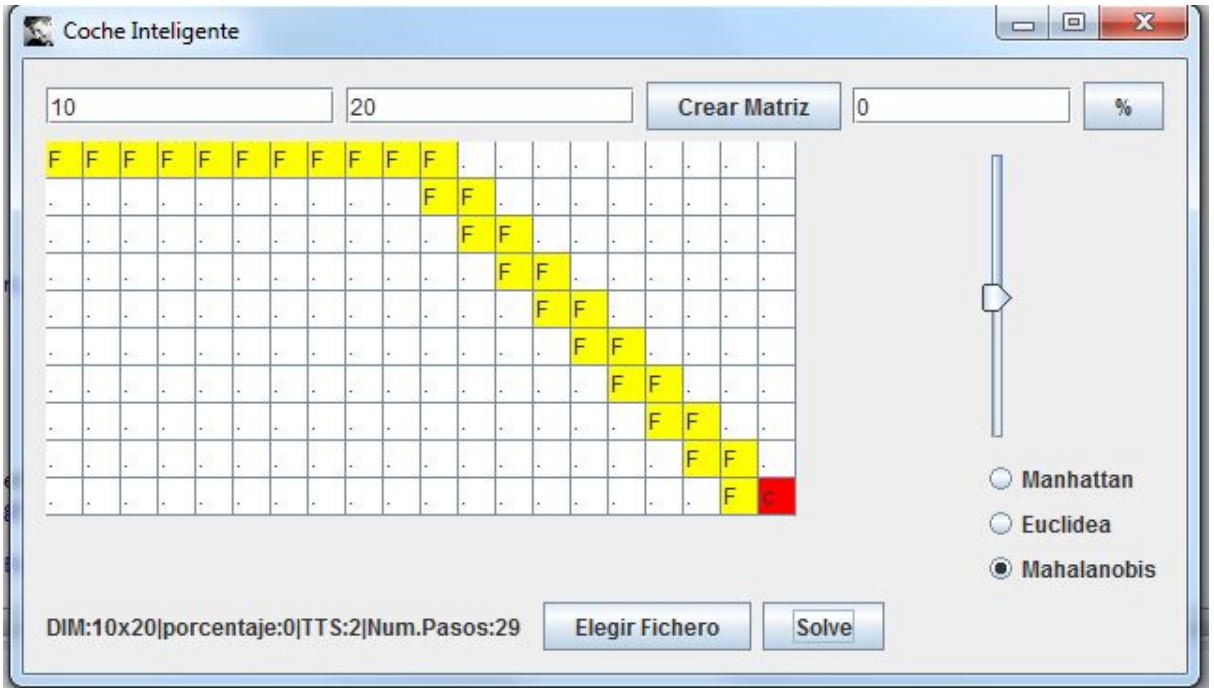
Debido a que para el estudio experimental se solicitó que el algoritmo calcule las distancias mediante diversos métodos decidimos incluir en nuestra interfaz un selector con el que cambiar entre un método y otro y hacer no tener que estar modificando el código cada vez que se quisiera probar un método diferente.



Problema resuelto siguiendo el método de la distancia Manhattan



Problema resuelto siguiendo el método de la distancia Euclídea



Problema resuelto siguiendo el método de la distancia de Mahalanobis.

Algoritmo de búsqueda

El algoritmo de búsqueda que seleccionamos fue “Primero el mejor” que trata de partir de una lista donde se guardan los nodos de trayectoria (el inicial o aquellos a los que se puede llegar derivando del primero de la lista). El primero de la lista es comprobado, se introducen sus descendientes y se meten en la lista, luego se borra el estudiado. Sólo se ha de comprobar que el que está primero ahora, no es el mismo que buscamos, o en caso de que se quede vacía, diríamos que no hay solución. Importante decir que esta lista se ordena según las expectativas del objeto, cuanto más altas sean, más al principio se colocarán.

Nuestra implementación consiste en 3 listas “caminoFinal”, “mejor” y “visitados”, caminoFinal es una copia de visitados hasta que se acaba de resolver el problema. El primer paso consiste en vaciar las listas por si quedasen datos de una solución antigua y luego se añade en las tres listas la posición inicial del coche. Posteriormente se le pide al coche seleccionar a todas las posiciones a las que puede ir (No salen del mapa ni tienen obstáculos) y de estas se añaden a mejor aquellas que no estén en visitados, luego se ordena mejor en base a la distancia con la meta y se elimina su cabecera hasta que ésta no esté en visitados, una vez hecho esto se desplaza el coche a la cabecera y se añade la nueva posición del coche a visitados, esto se repite hasta que el coche llega a la meta. Por último se eliminan duplicidades de visitados y se selecciona de caminoFinal aquellas coordenadas pertenecientes al camino óptimo eliminando las demás, esto se consigue recorriendolo de atrás hacia adelante y comprobando que no se puede llegar a una coordenada dada en un solo paso desde otra anterior a la que le precede en la lista.

Estudio experimental

Para realizar el estudio experimental hemos usado dos ordenadores en distintos estados para comprobar el rendimiento del programa de forma independiente al entorno en que se ejecute:

- Ordenador 1: Ubuntu 8gb RAM 3.3GHz (4 nucleos) CPU 44% carga RAM 2% carga CPU.
- Ordenador 2:Windows 8 GB ram 2,5GHz CPU (4 nucleos) 85% Carga RAM 12% Carga CPU.

Luego se han establecido los distintos tamaños de pruebas en base al tamaño máximo permitido por el programa:

- 25*25 Tamaño pequeño
- 100*100 Tamaño medio
- 200*200 Tamaño máximo

Y por último se han ejecutado en cada uno de los ordenadores 4 Pruebas con cada uno de los tamaños usando distintas formas de calcular la distancia entre el coche y la meta:

- Manhattan: $D(X1,\ X2)\ =\ ||X1 - X2||\ =\ \sum_{i=1}^{n} |X_{1i} - X_{2i}|$
- Euclídea: $D(X1, X2)\ =\ \sqrt{\sum_{i=1}^{n}(x_{1i} - x_{2i})^2}$
- Mahalanobis: $D(X1, X2)\ =\ \vee\ \{x_{11} - x_{21}\ ,\ x_{12} - x_{22}\}$

		Pequeño								Mediano								Grande							
		Ordenador1				Ordenador2				Ordenador1				Ordenador 2				Ordenador1				Ordenador 2			
Manhattan	Tiempo	39	3	5	1	3	4	38	1	318	2741	23104	0	13896	42665	2700	88	423	17476	2939	6177	75	91	175	5
	Pasos	61	59	221	25	9	11	98	10	752	1855	5415	7	4925	7265	2896	272	559	7551	3815	5021	133	163	293	7
Euclídea	Tiempo	0	14	25	76	2	1	1	29	6	7	16866	76	122	373	83	53	1503	19	2030	191	610	64	841	3
	Pasos	1	115	125	531	24	21	17	399	42	57	4905	531	236	424	263	163	1987	111	2285	364	430	113	560	5
	Tiempo	67	1	2	0	1	0	1	5	2425	2	23798	5769	6163	45237	369	3257	5049	3	42	300	74	2	108	>120000
Mahalanobis	Pasos	417	18	30	4	14	29	33	211	2481	9	5409	1710	3561	7441	1295	1529	3087	19	110	481	61	3	51	NAN

Tabla de resultados de las pruebas

		Total			Pequeño			Mediano			Grande		
		Ambos	Ordenador1	Ordenador2	Ambos	Ordenador1	Ordenador2	Ambos	Ordenador1	Ordenador2	Ambos	Ordenador1	Ordenador2
Mahattan	Tiempo medio	4706,9	4435,5	4978,4	11,75	12	11,5	10689	6540,75	14837,25	3420,12	6753,75	86,5
	Pasos medios	1726	2112	1341	62	92	32	2924	2008	3840	2193	4237	149
	Velocidad media	0,36	0,47	0,26	5,27	7,66	2,78	0,27	0,3	0,25	0,64	0,62	1,72
Euclídea	Tiempo medio	958,1	1734,4	181,8	18,5	28,75	8,25	2198,25	4238,75	157,75	657,62	935,75	379,5
	Pasos medios	572	922	222	155	193	116	828	1384	272	732	1187	277
	Velocidad media	0,59	0,53	1,22	8,37	6,71	14,06	0,37	0,32	1,72	1,11	1,27	0,73
Mahalanobis	Tiempo medio	4029,34	3121,5	5019,72	9,62	17,5	1,75	10877,5	7998,5	13756,5	796,85	1348,5	61,3
	Pasos medios	1218	1148	1294	95	118	72	2930	2403	3457	545	925	39
	Velocidad media	0,3	0,37	0,26	9,88	6,74	41,14	0,27	0,3	0,25	0,68	0,69	0,64
Conjunto	Tiempo medio	3220,23	3097,13	3346,85	13,29	19,41	7,16	7921,58	6259,33	9583,83	1660,86	3012,66	186,18
	Pasos medios	1171	1394	942	104	134	73	2227	1932	2522	1183	2116	166
	Velocidad media	0,36	0,45	0,28	7,83	6,9	10,2	0,28	0,29	0,26	0,71	0,7	0,89

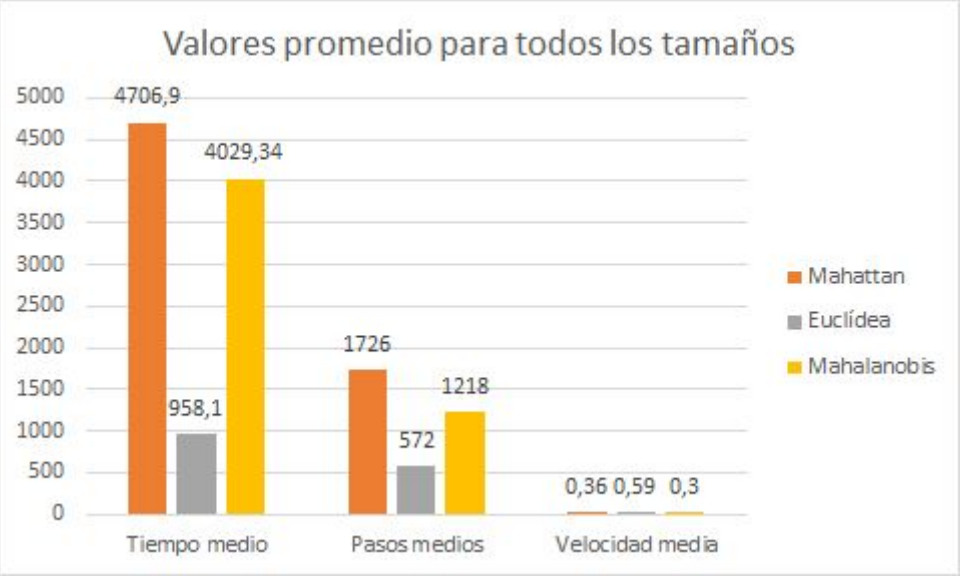
Tabla con los valores promedio de las pruebas

Conclusiones

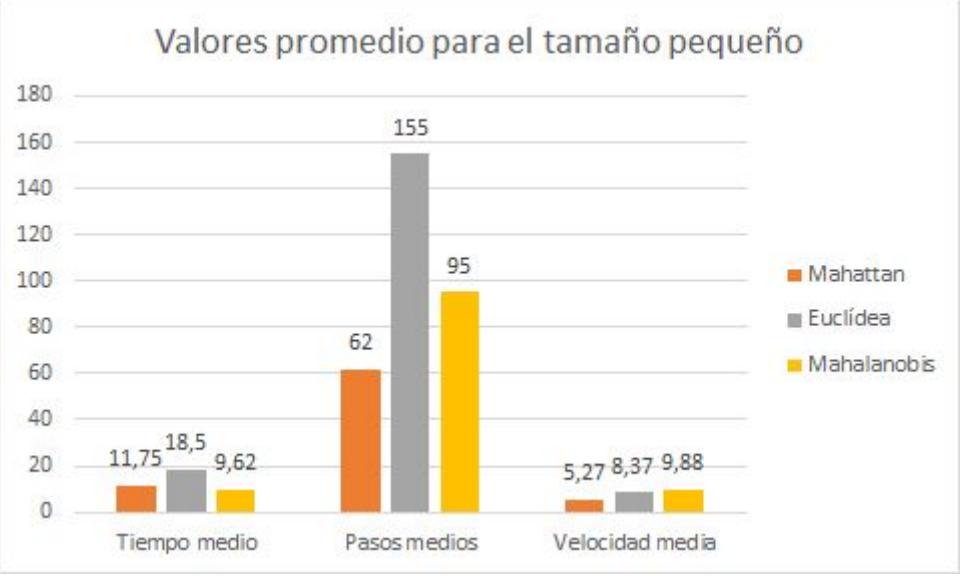
Analizando los datos recogidos en las diversas pruebas realizadas hemos sacado una serie de conclusiones que enumeramos a continuación. Lo primero aclarar que para estas conclusiones hemos tenido en cuenta tres variables distintas, el tiempo, los pasos y la velocidad (pasos/tiempo). Esta última variable la hemos añadido porque nos ha parecido más representativa que simplemente mirar tiempo o pasos ya que a la velocidad no le va a afectar tanto ese factor suerte de dónde hayan sido colocados el coche y la meta al crear la matriz aleatoria.

En primer lugar, mencionar que sin tener en cuenta el tamaño en el que pongamos a prueba nuestra aplicación y algoritmo, es decir, uniendo los datos medios recogidos en el pequeño, mediano y máximo concluimos que en cuanto a tiempo medio, con mucha diferencia sobre el resto, es el

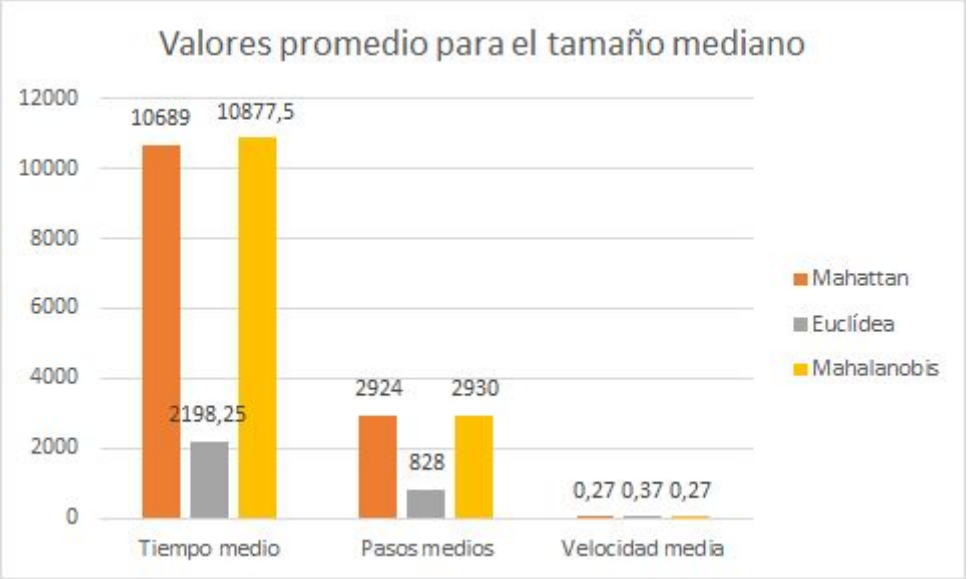
método de distancia euclídea el que menos requiere de los tres. En cuanto al número de pasos vuelve a ser el algoritmo usando la distancia euclídea el que resuelve los problemas planteados con un menor número de ellos. Además se trata del método más rápido con una velocidad de 0.59 st/ms mientras que con Manhattan los problemas se han resuelto con una velocidad de 0.36 st/ms y usando Mahalanobis, el más lento, se ha llegado a una velocidad de 0.3 st/ms.



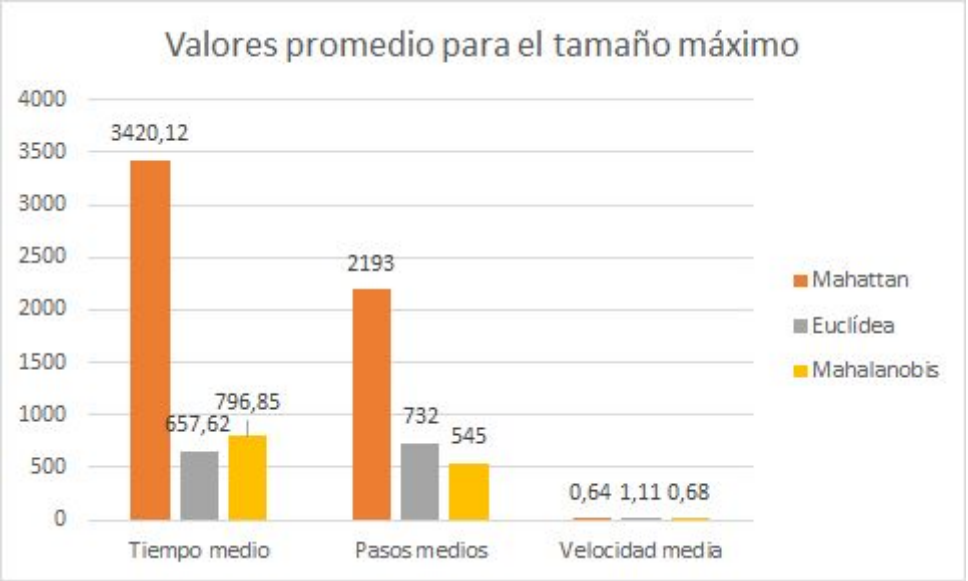
En caso de que la matriz sea pequeña, se obtendrá un mejor tiempo con el algoritmo Mahalanobis, en cuanto a número de pasos el mejor algoritmo bajo nuestros resultados ha sido Manhattan y por último, si miramos la velocidad obtenemos que el mejor algoritmo fue el Mahalanobis.



En caso de que la matriz sea mediana, el mejor resultado en lo que a tiempo se refiere será proporcionado por el método de distancia euclídeo, en cuanto a los pasos nuestro ganador fue euclídeo nuevamente y por último basándonos en su velocidad de nuevo el claro ganador es el algoritmo euclídeo, razón por la cual es lógico decir que el que mejor ha desempeñado su función en una matriz mediana ha sido el euclídeo.



En caso de que la matriz sea grande, observamos que el tiempo tiene por líder al euclídeo, en número de pasos al Mahalanobis y por último, en cuanto a la velocidad el euclídeo aunque relativamente cercano al Mahalanobis, lo que a nuestro modo de ver lo convertiría, de nuevo, en el mejor algoritmo para este último caso.

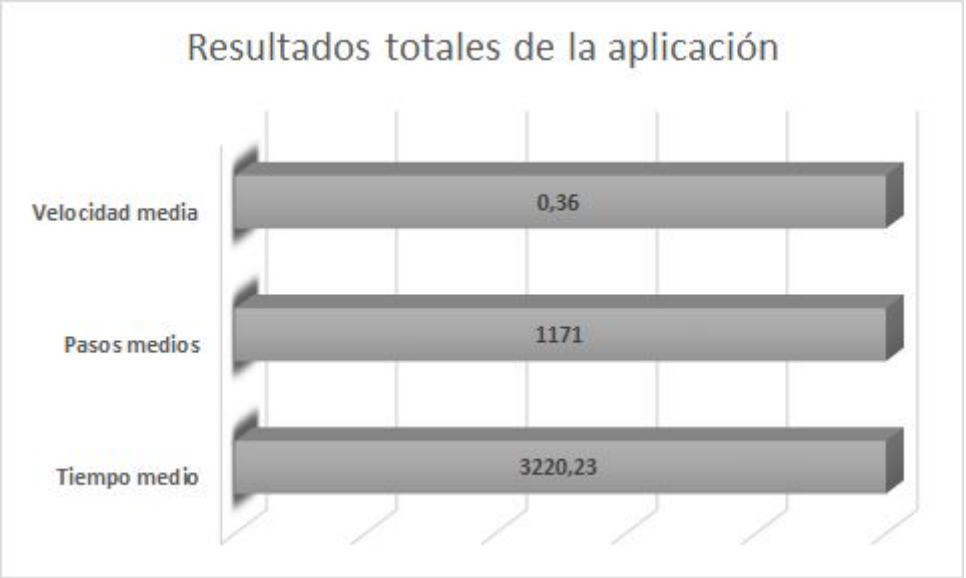


Decir que con el algoritmo de Mahalanobis se dieron casos en los con matrices de tamaño máximo el tiempo de ejecución era excesivo y se detuvo el programa antes de que terminara el algoritmo. En la tabla de resultados no decidimos contemplar todos los casos en los que esto sucedió sino que sólo representamos uno de ellos.

A pesar de que el ordenador cambie los tiempo medios bajan pero en sí la eficacia de los algoritmos no se ha visto demasiado afectada con el cambio de máquina así que deducimos que da igual el pc en que se ejecute el programa. Además, comprobando los distintos casos y calculando la complejidad del código, hemos podido comprobar que el tiempo escala de forma polinómica.

Otro cálculo que hemos realizado y vemos que puede tener relevancia ha sido obtener los datos medios de los tres algoritmos en conjunto. Esta información nos permitiría comparar nuestra

implementación de los algoritmos con la implementación de otros programadores, preferiblemente en el mismo pc para tener un criterio más exacto.



Bibliografía

1. Apuntes de clase
2. http://web.mit.edu/6.005/www/sp14/psets/ps4/java-6-tutorial/components.html
3. http://wwwae.ciemat.es/~cardenas/docs/lessons/MedidasdeDistancia.pdf
4. https://docs.oracle.com/javase/8/docs/api/
5. https://stackoverflow.com/