Determinación de la Trayectoria Óptima para un Taxi Autónomo

Dana Belén Choque Zárate
Escuela Técnica Superior de Ingeniería Informática
Universidad de La Laguna
La Laguna, España
alu0101328348@ull.edu.es

Stephanie Andreina Arismendi Escobar
Escuela Técnica Superior de Ingeniería Informática
Universidad de La Laguna
La Laguna, España
alu0101351728@ull.edu.es

Gabriel Alberto Luis Freitas
Escuela Técnica Superior de Ingeniería Informática
Universidad de La Laguna
La Laguna, España
alu0101348421@ull.edu.es

***Resumen*— El problema al que nos enfrentamos es encontrar un camino entre un punto A y un punto B de una malla que posee obstáculos. Básicamente se nos pide un simulador de un GPS, que dado un inicio y un objetivo, encuentre el camino más corto entre ambos.**

Hemos utilizado el algoritmo A*, el cuál utilizará funciones heurísticas, que tratan de asemejarse al pensamiento humano, para calcular qué nodo o celda vecina a la que nos encontramos es la mejor para desplazarnos a ella y repetir el algoritmo.

Las funciones heurísticas utilizadas son la distancia de Manhattan, la distancia Euclídea, y la distancia de Chebyshev. Estas tres funciones nos darán un coste F, que será siempre menor o igual al coste que tendrá el camino obtenido por ese nodo. Gracias a esto, siempre elegiremos el nodo más cercano a la solución en lugar de dispersarse evaluando todos los nodos posibles.

Keywords*— *Función heurística, Manhattan, Euclídea, Chebyshev, algoritmo A*.

I. INTRODUCCIÓN

El objetivo de esta práctica es la implementación de estrategias de búsqueda para la determinación de trayectorias óptimas para taxis autónomos.

El entorno del coche autónomo tendrá dimensiones *M* (columnas) x *N* (filas) y constituido por celdas libres y ocupadas, donde el vehículo puede efectuar cuatro posibles movimientos desde la casilla actual a una de las 4 casillas vecinas (Norte, Sur, Este u Oeste) que no se encuentre ocupada. Las casillas ocupadas corresponden a obstáculos y otros vehículos. Las casillas libres corresponden con celdas libres de obstáculos y vehículos.

II. FORMULACIÓN DEL PROBLEMA

Los estados de este problema vendrán dados por la posición del coche autónomo. El conjunto de celdas libres serán todos los estados factibles del problema. Sin embargo, no todos los estados que sean factibles serán alcanzables desde el estado inicial.

El estado inicial del problema del coche autónomo vendrá dado por el usuario, y se tratará de la celda libre en donde se posicione el coche. Desde este estado, el coche se irá moviendo a sus celdas adyacentes siguiendo el algoritmo que se utilice. Habrá celdas libres que sean inalcanzables desde el estado inicial, ya que pueden estar separadas por una barrera de obstáculos o rodeada de estos. Por último, el estado final, u objetivo, será aquel que el usuario haya marcado como destino. Una vez que el coche autónomo vaya recorriendo los diferentes estados, si alcanza el estado objetivo, se puede decir que ha alcanzado la solución y deberá mostrarla, en caso de que el estado objetivo no sea alcanzable, también deberá decirse.

III. ENTORNO DE SIMULACIÓN

Para la ejecución de este proyecto, se ha elegido el lenguaje de programación C++ debido a la facilidad que este ofrece para el manejo de datos. Esto será de gran utilidad a la hora de desarrollar el algoritmo de búsqueda.

Respecto a el entorno de simulación, nos encontramos de lleno con un programa ejecutado en consola.

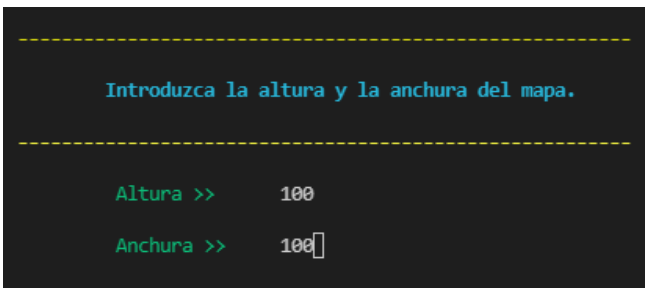

Ilustración 1. Solicitud de las dimensiones del circuito.

El programa, al iniciarse, no pedirá las medidas del mapa *(illustración 1)*. Esto recogerá una alto y un ancho, medido por casillas, que podrá alcanzar cualquier dimensión. A su vez, con menús similares, pedirá la posición *x* e *y* inicial, al igual que la que pretendemos alcanzar.

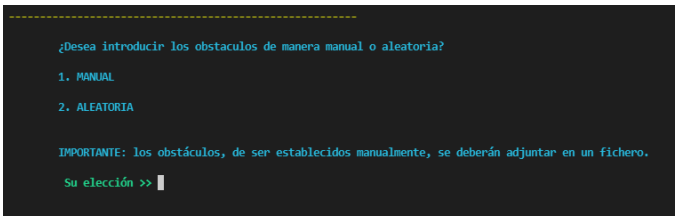

Ilustración 2. Menú de selección porcentaje de obstáculos.

Respecto a la generación de objetos, tanto manual como aleatoria, se generará otro menu *(illustración 2)*. Este aclarará que, en caso de que la elección sea "manual", se deberán añadir los objetos por medio de un fichero. En cambio, de ser aleatoria, se hará uso de un porcentaje del 0 al 100. Por el contrario, de seleccionarse manual, se pedirá que se introduzca por pantalla el nombre del fichero.

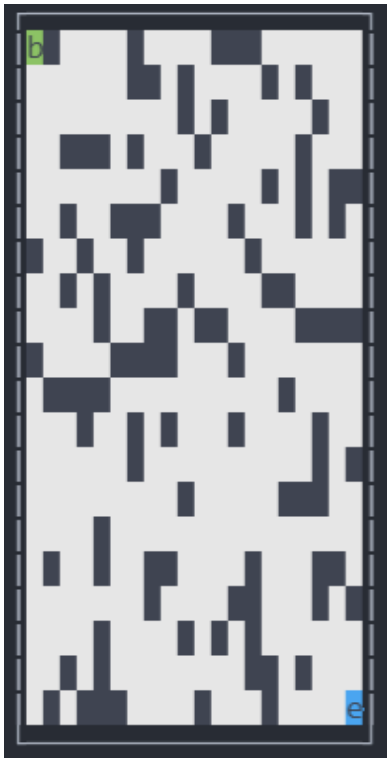

Ilustración 4. Mapacon obstáculos aleatorio de 25%, tamaño 20x20.

Los obstáculos son las celdas de color negro y las casillas libres están de color blanco *(illustración 3)*.

Luego, se preguntará de cuantas direcciones realizará el vehículo para moverse *(illustración 6)*: de 4 direcciones (norte, sur, este y oeste) *(illustración 4)* o de 8 direcciones (las mismas direcciones de las 4 anteriores más las diagonales de un punto a otro) *(illustración 8)*.

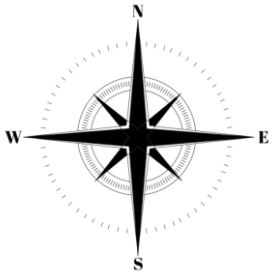

Ilustración 6. 4 direcciones.

Ilustración 6. 8 direcciones.

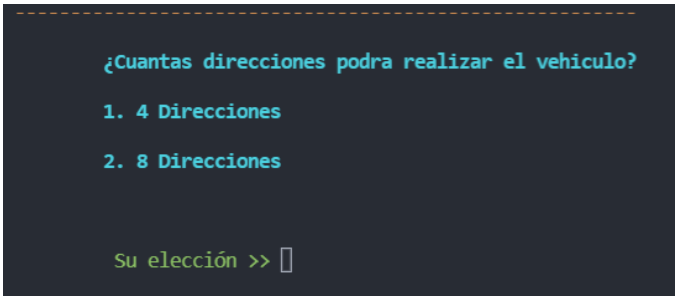

Ilustración 3. Menú de elección de direcciones.

Después, habrá otro menú en el que se pedirán las función a resolver el circuito. En el menú recomendará la mejor opción de las funciones heurísticas para resolver el problema en determinados casos de la dimensión y dirección escogida.

Finalmente, se imprimirá por pantalla el mapa con la posición del coche (marcado con el carácter b, de begin, de color verde), la posición que se pretende alcanzar (marcada con la e, de end, de color azul) y la solución del camino óptimo en color rosa.

IV. METODOLOGÍA DE TRABAJO

El desarrollo del código se divide en cinco partes: estructura de datos, algoritmo de búsqueda, interfaz de usuario, funciones. Cada componente del grupo tiene diversas tareas: desarrollo en la partes funcionales, actualizar el repositorio de Github, implementación del código, desarrollo del entorno gráfico. En cuanto el informe, todos redactarán el apartado que escojan.

Tareas realizadas/ futuras de cada miembro:

- Gabriel se encargó en la implementación de la interfaz de usuario y estructura de datos.
- Stephanie investigará la implementación y funcionamiento de Opengl para la interfaz gráfica, y desarrollará el algoritmo de búsqueda.
- Dana implementará las funciones básicas, documentación del código y desarrollará el algoritmo de búsqueda.

Todos redactarán el informe y actualizarán el repositorio de github[4].

Para el entorno de trabajo se ha utilizado GitHub[1], ya que es el controlador de versiones que conocen todos los desarrolladores. Se ha creado un repositorio conjunto donde se almacenará el código fuente y el informe[2].

A la hora de desarrollar código empleamos la técnica de programación en grupo, y en ocasiones, individual. Para trabajar en el código de manera cómoda y en vivo, utilizamos la extensión de Visual Studio Code "Live Share[3]" que permite crear un entorno colaborativo donde varias personas desarrollan código de manera simultánea.

Para las reuniones para comentar partes del código, utilizamos el google meet o hablamos por el grupo de whatsapp.

V. ALGORITMO DE BÚSQUEDA

En las clases de teoría se ha visto dos tipos de algoritmos de búsqueda informadas: la búsqueda voraz primero el mejor (búsqueda Greedy) y la búsqueda en estrella A*.

Para el cálculo de la distancia hay dos tipos: la distancia de Manhattan o la distancia Euclídea.

Para el cálculo del camino óptimo mediante un algoritmo de búsqueda propuesto, se ha escogido el algoritmo de A* o A-star. Pues es más eficiente y óptimo que el algoritmo búsqueda Greedy, ya que la búsqueda A* es una búsqueda primero el mejor que evita expandir caminos que son caros. Este tipo de búsqueda, tiene una función heurística *f(n)* que combina dos funciones para evaluar los nodos del árbol de búsqueda:

- *g(n)* = costo de la ruta que va del nodo de partida hasta el nodo *n*.
- *h(n)* = coste que va desde *n* hasta el nodo objetivo.

Y por tanto *f(n) = g(n) + h(n)*. Donde *f(n)* es el costo total estimado de la solución más barata que pasa por el nodo *n*.

En detalle a la implementación del algoritmo, se detalla los siguientes tres apartados:

1. Las estructuras de datos utilizadas: Matrices de booleanos para marcar los obstáculos y la solución, listas enlazadas para las listas abiertas y cerradas.
2. Las funciones heurísticas utilizadas: función Manhattan, función euclidiana y función Chebyshev.
3. El pseudocódigo de la implementación del algoritmo.

a) Si la posición final es igual a la posición inicial, hemos alcanzado el final.
b) Si no lo es, implementamos el nodo inicial a la lista abierta.
c) Mientras la lista abierta no esté vacía, hacer lo siguiente:
    i. Tomar el primer nodo de la lista enlazada
    ii. Si la posición del nodo es igual a la posición final
        1. Marcaremos el nodo como una solución, Si llegamos otra vez a este punto, comprobaremos cuál de los dos tiene menor coste, y solo dejaremos marcado ese, el otro, lo incluimos en la lista cerrada pero no buscamos sus vecinos.
    iii. Si no lo es
        1. mover el nodo de la lista abierta a la cerrada.
        2. Si los nodos vecinos son posiciones válidas
            a. Si el nodo vecino válido está en la lista abierta, se compara el coste f del nodo vecino y el de la lista abierta. Si este es menor, se actualiza el nodo de la lista y se reordena la lista.
            b. Si el nodo no está en la lista abierta, pero está en la lista cerrada, no se hace nada con él y pasamos a comprobar el siguiente vecino.
            c. Si no está en ninguna lista, lo implementamos de forma ordenada (al coste) en la lista abierta.
            d. Una vez hayamos limpiado toda la lista abierta, tomaremos al nodo que habíamos

marcado como solución e iremos buscando a sus padres y marcandolos en una matriz que teníamos vacía, la cuál será el camino de nuestra solución

VI. EVALUACIÓN EXPERIMENTAL

En este apartado, se evaluarán los diferentes resultados experimentales. Dependiendo de los escenarios, se clasifican en dos casos:

- El peor caso: son los escenarios compuestos por mallas de tamaños muy grandes (más de 1000x1000) y ningún obstáculo, ya que el algoritmo tendrá que analizar más nodos.

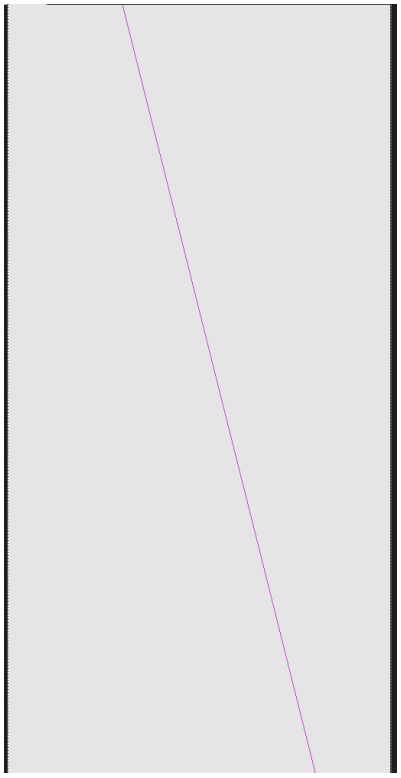

Ilustración 7. Ejemplo de un escenario del peor caso.

- El mejor caso: un mapa de tamaño máximo 200x200, con un nivel de obstáculos que ronde el 25%, logrando así un tiempo de procesamiento ínfimo.

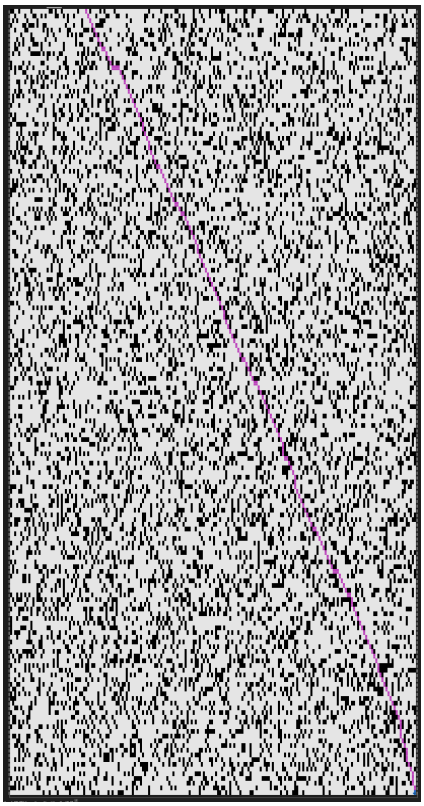

Ilustración 8. Ejemplo de escenario del mejor caso.

Para el estudio de este problema, evaluaremos los datos de los dos casos de escenario para cada función heurística. Así mismo, también se evaluará el tipo de dirección para cada función.

En la primera tabla se encuentra el estudio del tiempo de ejecución. Dicha tabla se divide en otras dos tablas para las situaciones del mapa con obstáculos o sin ellos.

1. En la tabla 1 *(illustración 9)* se encuentran 24 escenarios totales: 12 escenarios por dirección (4 direcciones u 8 direcciones), 6 escenarios por dimensión (pequeño y grande) y 2 escenarios por cada nivel de dificultad.

Tiempo de ejecución									
4 direcciones					8 direcciones				
Manhattan					Manhattan				
	Pequeño		Grande			Pequeño		Grande	
nivel\dimensión	20x20	80x80	110x110	200x200	nivel\dimensión	20x20	80x80	110x110	200x200
25	0,000314	0,006556	0,015499	0,069066	25	0,000643	0,017361	0,042358	0,215936
50	N/S	N/S	N/S	N/S	50	N/S	N/S	N/S	N/S
80	N/S	N/S	N/S	N/S	80	N/S	N/S	N/S	N/S
Euclídea					Euclídea				
	Pequeño		Grande			Pequeño		Grande	
nivel\dimensión	20x20	80x80	110x110	200x200	nivel\dimensión	20x20	80x80	110x110	200x200
25	0,000315	0,006541	0,013983	0,061856	25	0,000583	0,018631	0,035861	0,208729
50	N/S	N/S	N/S	N/S	50	N/S	N/S	N/S	N/S
80	N/S	N/S	N/S	N/S	80	N/S	N/S	N/S	N/S
Chebyshev					Chebyshev				
	Pequeño		Grande			Pequeño		Grande	
nivel\dimensión	20x20	80x80	110x110	200x200	nivel\dimensión	20x20	80x80	110x110	200x200
25	0,000269	0,005808	0,014637	0,06039	25	0,000518	0,015647	0,037098	0,187268
50	N/S	N/S	N/S	N/S	50	N/S	N/S	N/S	N/S
80	N/S	N/S	N/S	N/S	80	N/S	N/S	N/S	N/S

Ilustración 9. Tabla de ejecución con porcentajes de obstáculos.

En la tabla mencionada, observamos comportamiento en el porcentaje de obstáculos 25%. En los demás casos de porcentajes evaluados, vemos que no hay solución *(illustración 19).*

1.1. Tabla y gráfica para tiempo de ejecución en los casos de 4 direcciones.

Función\ tamaño	20x20	80x80	110x110	200x200
Manhattan	0,000314	0,006556	0,015499	0,069066
Euclídea	0,000315	0,006541	0,013983	0,061856
Chebyshev	0,000269	0,005808	0,014637	0,060639

Ilustración 10. Tabla de tiempo de ejecución para los casos de 4 direcciones.

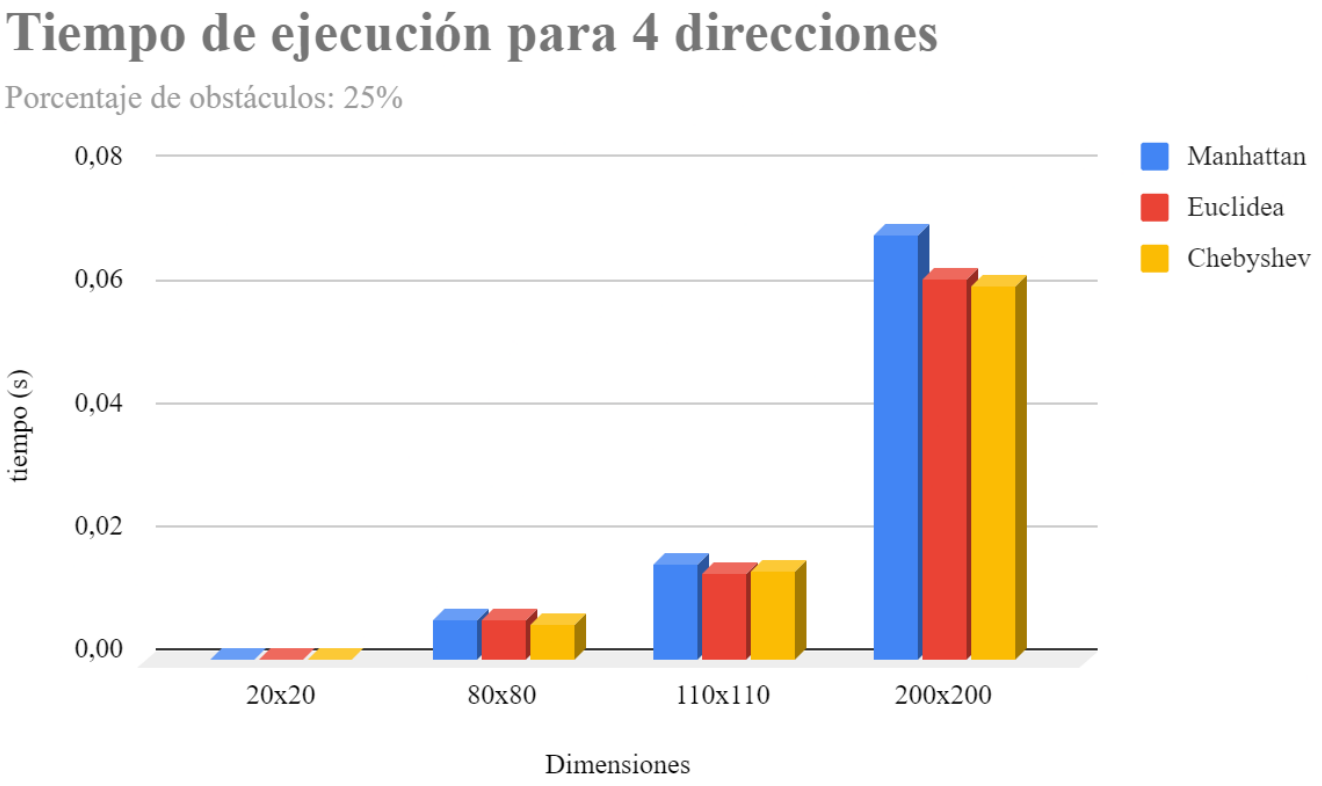

Ilustración 11. Gráfica de tiempo de ejecución para los casos de 4 direcciones.

1.2. Tabla y gráfica para tiempo de ejecución en los casos de 8 direcciones.

Función\ tamaño	20x20	80x80	110x110	200x200
Manhattan	0,000643	0,017361	0,042358	0,215936
Euclídea	0,000583	0,018631	0,035861	0,208729
Chebyshev	0,000518	0,015647	0,037098	0,187268

Ilustración 12. Tabla del tiempo de ejecución para los casos de 8 direcciones.

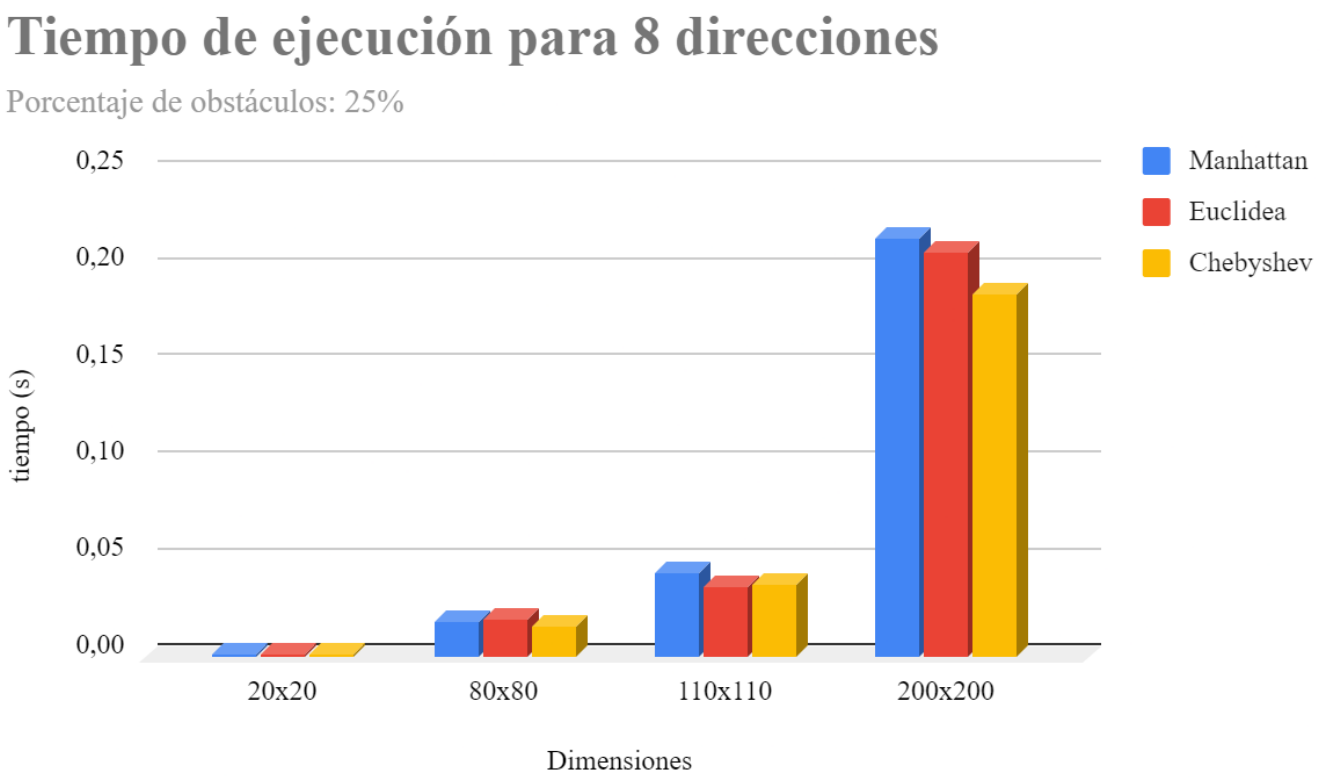

Ilustración 13. Gráfica del tiiempo de ejecución para los casos de 8 direcciones.

2. En la tabla 2, se encuentran 8 escenarios totales: 4 escenarios por dirección (4 direcciones u 8 direcciones) y 2 escenarios por dimensión (pequeño y grande).

Tiempo de ejecución									
4 direcciones					8 direcciones				
Manhattan					Manhattan				
	Pequeño		Grande			Pequeño		Grande	
nivel\dimensión	20x20	80x80	110x110	200x200	nivel\dimensión	20x20	80x80	110x110	200x200
0	0,000596	0,014405	0,032846	0,198839	0	0,001287	0,042415	0,102553	0,614105
Euclídea					Euclídea				
	Pequeño		Grande			Pequeño		Grande	
nivel\dimensión	20x20	80x80	110x110	200x200	nivel\dimensión	20x20	80x80	110x110	200x200
0	0,000493	0,013786	0,03033	0,198839	0	0,000981	0,042842	0,102698	0,61407
Chebyshev					Chebyshev				
	Pequeño		Grande			Pequeño		Grande	
nivel\dimensión	20x20	80x80	110x110	200x200	nivel\dimensión	20x20	80x80	110x110	200x200
0	0,000636	0,012852	0,032695	0,160395	0	0,001089	0,039517	0,095533	0,669968

Ilustración 14. Tabla de tiempo de ejecución para 0% de obstáculos.

De los datos obtenidos de las dos tablas se visualizará en una gráfica de columnas. Habrán dos tablas por cada dirección *(ilustración 15 e ilustración 17)*.

La tabla 2 se divide en dos tablas para cada tipo de dirección con 0% de obstáculos.

2.1. Tabla y gráfica para tiempo de ejecución en los casos de 4 direcciones.

Función\ tamaño	20x20	80x80	110x110	200x200
Manhattan	0,000596	0,014405	0,032846	0,198839
Euclídea	0,000493	0,013786	0,03033	0,160685
Chebyshev	0,000636	0,012852	0,032695	0,160395

Ilustración 15. Tabla de ejecución para lso casos de 4 direcciones.

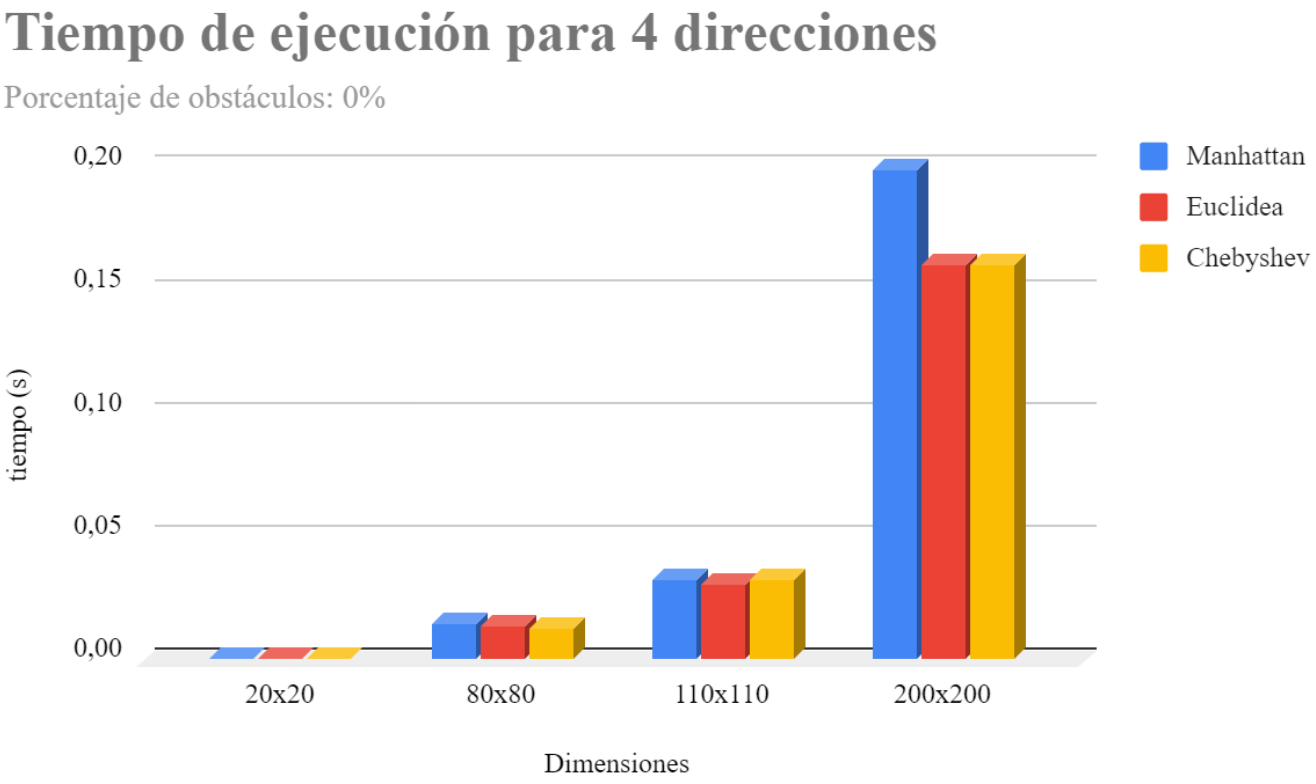

Ilustración 16. Gráfica del tiempo de ejecución para los casos de 8 direcciones.

2.2. Tabla y gráfica para tiempo de ejecución en los casos de 8 direcciones.

Función\ tamaño	20x20	80x80	110x110	200x200
Manhattan	0,001287	0,042415	0,102553	0,614105
Euclídea	0,000981	0,042842	0,102698	0,61407
Chebyshev	0,001089	0,039517	0,095533	0,669968

Ilustración 17. Tabla de tiempo de ejecución para los casos de 8 direcciones.

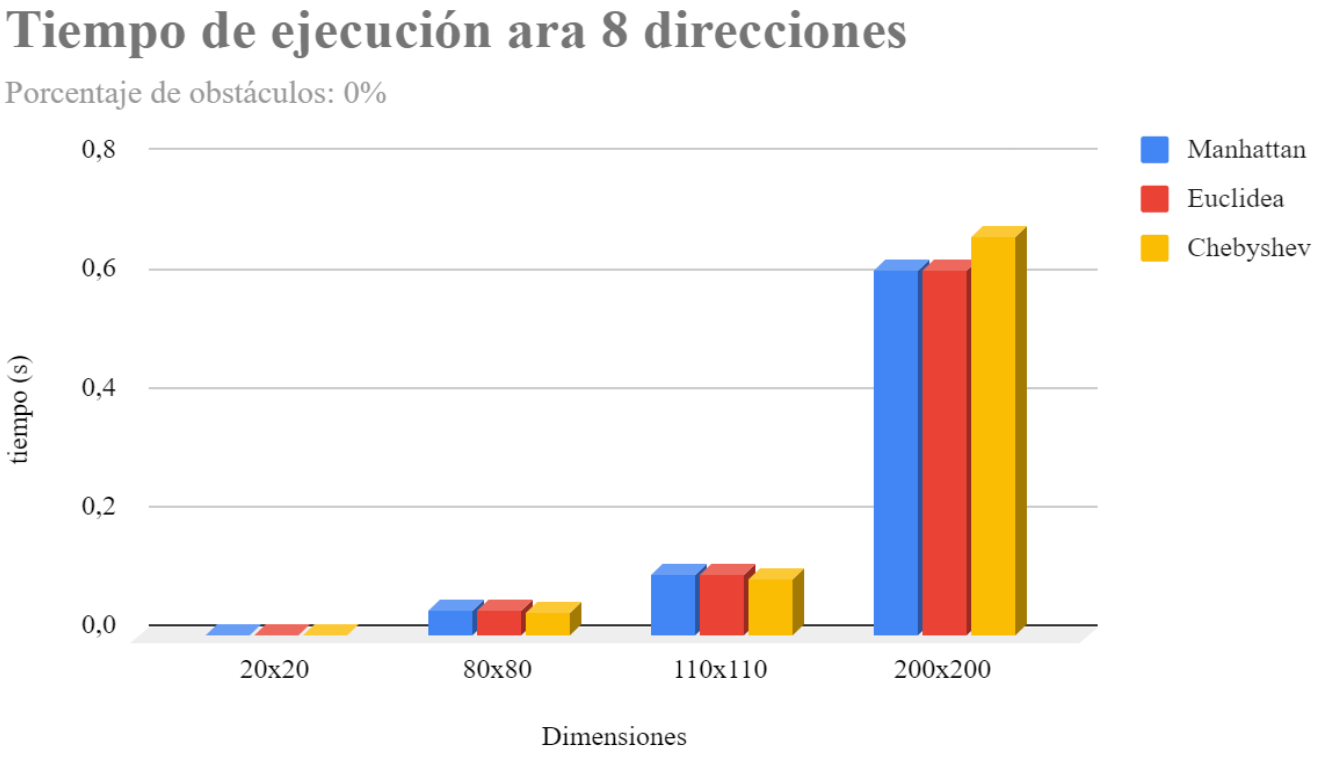

Ilustración 18. Gráfica para los casos de 8 direcciones.

En la siguiente tabla se ilustra los casos que no hubo solución para los dos tipos de direcciones con un porcentaje de dificultad 50% y 80%.

Porcentaje de obstáculos: 50%				
Función\ tamaño	20x20	80x80	110x110	200x200
Manhattan	N/S	N/S	N/S	N/S
Euclídea	N/S	N/S	N/S	N/S
Chebyshev	N/S	N/S	N/S	N/S
Porcentaje de obstáculos: 80%				
Función\ tamaño	20x20	80x80	110x110	200x200
Manhattan	N/S	N/S	N/S	N/S
Euclídea	N/S	N/S	N/S	N/S
Chebyshev	N/S	N/S	N/S	N/S

Ilustración 19. Tabla de tiempo de ejecución sin solución.

En las tablas siguientes se ilustran la cantidad de nodos, el tamaño de los caminos y los costes de las solución respectivamente. Las situaciones recreadas son:

- 4 direcciones con un 25% de obstáculos.

Función\ tamaño	20x20	80x80	110x110	200x200
Manhattan	0,000314	0,006556	0,015499	0,069066
C/N	299	4760	8998	29766
T/C	38	157	217	398
C/S	38	157	217	398
Euclidea	0,000315	0,006541	0,013983	0,061856
C/N	299	4760	8998	29766
T/C	38	157	217	398
C/S	38	157	217	398
Chebyshev	0,000269	0,005808	0,014637	0,060639
C/N	299	4760	8998	29766
T/C	38	157	217	398
C/S	38	157	217	398

Ilustración 20. Tabla de cantidad de nodos y costes para 4 direcciones.

- 8 direcciones con un 25% de obstáculos.

Función\ tamaño	20x20	80x80	110x110	200x200
Manhattan	0,000643	0,017361	0,042358	0,215936
C/N	300	4800	9075	29999
T/C	22	88	121	219
C/S	28,6274	116,581	160,765	293,144
Euclidea	0,000583	0,018631	0,035861	0,208729
C/N	300	4800	9075	29999
T/C	22	88	122	219
C/S	28,6274	116,581	161,35	293,144
Chebyshev	0,000518	0,015647	0,037098	0,187268
C/N	300	4800	9075	29999

T/C	22	88	121	219
C/S	28,6274	116,581	160,765	293,144

Ilustración 21.Tabla de cantidad de nodos y costes para 8 direcciones.

- 4 direcciones con un 0% de obstáculos.

Función\ tamaño	20x20	80x80	110x110	200x200
Manhattan	0,000596	0,014405	0,032846	0,198839
C/N	400	6400	12100	40000
T/C	38	158	218	398
C/S	38	158	218	398
Euclidea	0,000493	0,013786	0,03033	0,160685
C/N	400	6400	12100	40000
T/C	38	158	218	398
C/S	38	158	218	398
Chebyshev	0,000636	0,012852	0,032695	0,160395
C/N	400	6400	12100	40000
T/C	38	158	218	398
C/S	38	158	218	398

Ilustración 22. Tabla de cantidad de nodos y costes para 4 direcciones.

- 8 direcciones con un 0% de obstáculos.

Función\ tamaño	20x20	80x80	110x110	200x200
Manhattan	0,012077	0,229875	0,449142	2,37956
C/N	400	6400	12100	40000
T/C	19	79	109	199
C/S	26,87	111,723	154,149	281,429
Euclidea	0,006766	0,227316	0,458423	2,31897
C/N	400	6400	12100	40000
T/C	19	79	109	199

C/S	26,87	111.723	154,149	281,429
Chebyshev	0,011304	0,163891	0,451453	2,33754
C/N	400	6400	12100	40000
T/C	19	79	109	199
C/S	26,87	111,723	154,149	281,429

Ilustración 23. Tabla de cantidad de nodos y costes para 8 direcciones.

VII. CONCLUSIONES

El algoritmo que hemos utilizado (A estrella) nos ofrece muy buenos tiempos para encontrar el camino óptimo en tamaños comedidos. Con una cpu de más de 7 años hemos conseguido tiempos de medio segundo para mapas de 400x400 celdas. Son tiempos muy buenos, pero cuando llevamos esto a mallas más grandes, el algoritmo empieza a verse superado. Para mallas de 1000x1000 tarda sobre los 5 segundos, y si nos vamos a mallas de 5000x5000 llega a tardar 54m 44s y 970 ms. Se puede ver que la función que sigue es exponencial. Este tema nos causó una gran curiosidad, y tomamos la decisión de buscar una solución. Encontramos una versión mejorada del algoritmo A*, llamado Jump Point Search. Esta versión no valora todos los nodos adyacentes al nodo que estamos evaluando, sino que toma la heurística utilizada para valorar más rápidamente los nodos adyacentes y añadir a la cola únicamente aquellos nodos que se encuentren con obstáculos. Aunque no hemos implementado esa versión, nos pareció muy curiosa, ya que, según ciertos desarrolladores de videojuegos, esta versión llegaba a mejorar el algoritmo A* hasta en 30 veces.

Centrándonos de nuevo en el algoritmo A*. Este algoritmo se basa en una función heurística que trata de valorar por defecto la distancia a la que se encuentra cierto nodo del destino. Gracias a esta función, determinamos que nodo elegir en la siguiente iteración. Tras experimentar con el código podemos llegar a la conclusión de que no es necesario utilizar la mejor función heurística posible para llegar a la solución, sino que con una heurística que sea menos costosa de calcular, aunque no otorgue la misma precisión, podemos llegar a la solución óptima consumiendo menos cantidad de tiempo en el desarrollo del algoritmo.

REFERENCES

[1] G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955. *(references)*

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] I. S. Jacobs and C. P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[4] K. Elissa, “Title of paper if known,” unpublished.

[5] R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.

[8] Madrid Dykinson, Metaheurística.

LINKS

[1] https://es.wikipedia.org/wiki/GitHub

[2] https://github.com/Dncz/IA_21-22.git

[3] https://ed.team/blog/comparte-tu-codigo-y-colabora-en-tiempo-real-con-visual-studio-code

[4] https://swcarpentry.github.io/git-novice-es/08-collab/#:~:text=El%20due%C3%B1o%20debe%20dar%20acceso,%3A%2F%2Fgithub.com%2Fnotifications

[5] https://dev.to/jansonsa/a-star-a-path-finding-c-4a4h

[6] https://www.redblobgames.com/pathfinding/a-star/introduction.html

[7] https://gamedevelopment.tutsplus.com/tutorials/how-to-speed-up-a-pathfinding-with-the-jump-point-search-algorithm--gamedev-5818

[8] https://escarbandocodigo.wordpress.com/2011/07/11/1051/

https://www.ieee.org/conferences/publishing/templates.html