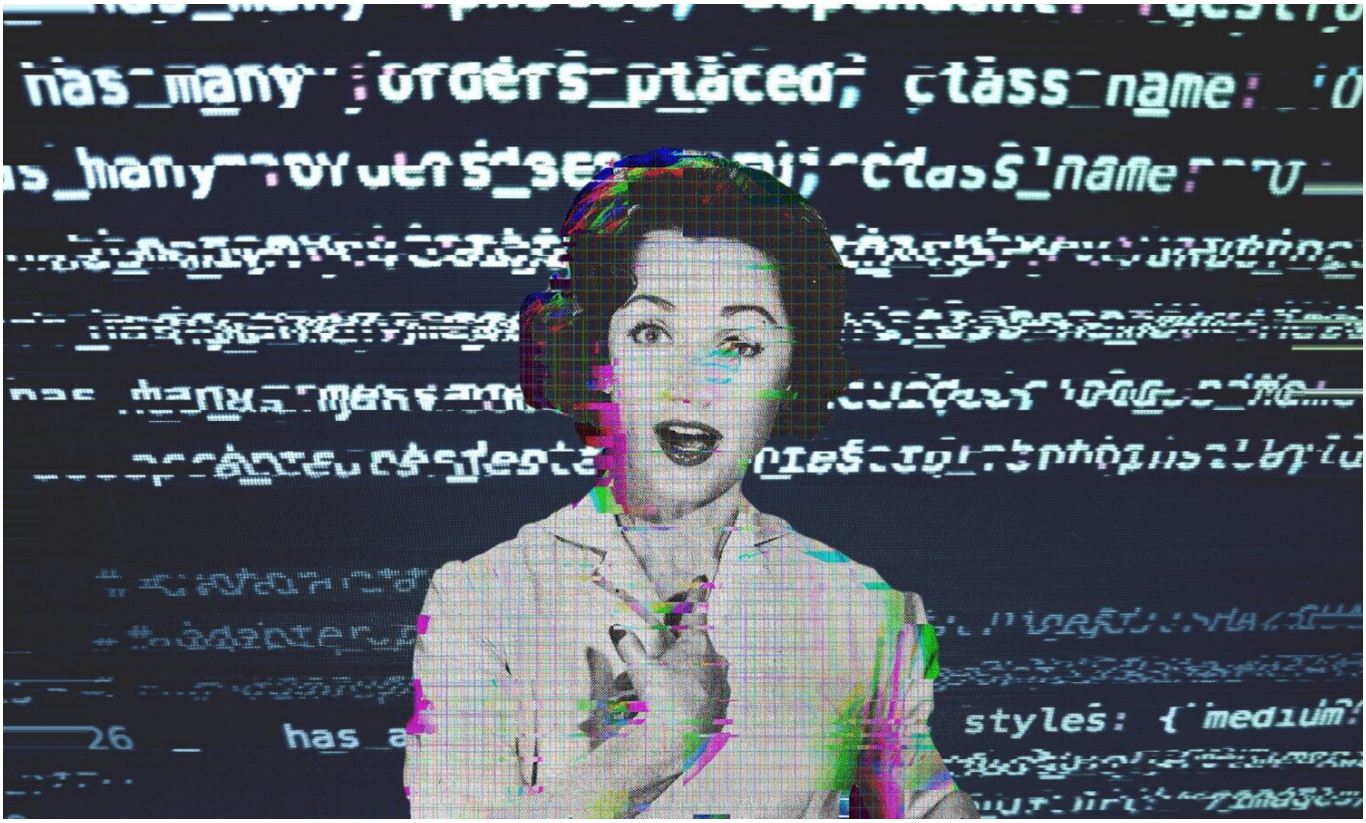
Práctica 1. Búsqueda heurística

Asignatura: Sistemas inteligentes
Alumno: Georg Usin
DNI/NIE: X8174555X



Índice

Explicación del algoritmo A*.

El algoritmo A* es un método de búsqueda que encuentra el camino más corto desde un punto A hasta un punto B. Es un algoritmo muy potente, muy utilizado en videojuegos y en aplicaciones como el GPS, que no son poca cosa.

A diferencia de los humanos, que usamos nuestra intuición y experiencias pasadas para elegir un camino más corto (por ejemplo, desde nuestra casa al kebab), los robots y sistemas informáticos necesitan algoritmos de búsqueda como A* para encontrar el camino óptimo hasta su destino (aunque, está claro que la mejor solución es pedir a domicilio).

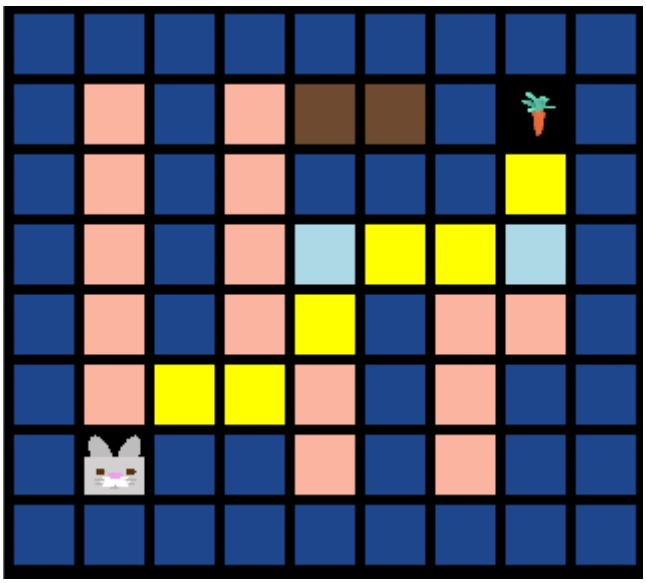
Para lograr esto, A* asigna un coste a cada posible camino, de manera que el camino con el menor coste es el más óptimo. Los movimientos ortogonales (horizontal o vertical) tienen un coste de 1, mientras que los diagonales tienen un coste de 1.5. Esto es por culpa de Pitágoras, si, el del teorema de Pitágoras.

Durante su ejecución, el algoritmo explora las casillas adyacentes a la posición actual y calcula el coste de moverse a cada una de ellas. La casilla con el coste más bajo se elige para avanzar, y así continúa hasta llegar al destino. Dicho así parece fácil, que lo es, pero si solo utilizara esta lógica, se explorarían absolutamente todos los caminos posibles, y eso, no nos interesa...

Para evitar la exploración de casillas innecesarias, se utiliza una heurística. La heurística es una fórmula que calcula una estimación de la distancia desde una casilla hasta la casilla final. Las casillas más alejadas del objetivo tendrán un valor de heurística más alto, mientras que las más cercanas tendrán un valor más bajo. Las casillas con una heurística más baja serán las primeras en ser exploradas, ya que son las que están más cerca del nodo final.

El camino más corto será el de menor coste total.

Traza de un problema pequeño de A*.



1.) El conejo comienza en la casilla (6,1), esta casilla inicial se añade a lista Frontera y se calcula su heurísitca (usaremos la distancia de Octile) hacia la zanahoria que está en la posición (1,7).

En la primera iteración se selecciona como nodo a explorar el (6,1), evaluá las casillas adyacentes y concluye que la mejor opción seria hacer un movimiento en diagonal hacia (5,2) pues tiene un coste de 1,5 que es menor que realizar dos movimientos ortogonales que tendrían un coste de 2. La casilla (5,2) se añade a la lista Frontera y se selecciona como siguiente nodo a explorar.

2.) En el segundo movimiento, se mueve hacia (5,3) que añade 1 al coste. En este movimiento se podría haber movido en diagonal, resultado en un coste final de 9, igualmente la elección del movimiento horizontal es correcta y no afecta al al coste mínimo. Si modificásemos el código para priorizar los movimientos diagonales, en este movimiento lo habría hecho. Se añade el nodo (5,3) a la lista y lo exploramos.

3.) En el tercer y cuarto movimiento, el conejo se desplaza en diagonal de (5,3) → (4,4) y (4,4) → (3,5). Estos dos movimientos consecutivos en diagonal añaden 3 al coste final. Ambos nodos son añadidos a la lista Frontera y para el quinto movimiento se explorara el nodo (3,5).

4.) En el quinto movimiento el conejo se mueve en horizontal hacia (3,6), moverse en diagonal en este caso seria incorrecto, ya que aumentaría el coste sin ningún beneficio adicional en la reducción de distancia. Se añade la casilla (3,6) a la lista. Y añade 1 al coste.

5.) En el penúltimo movimiento explora (3,6) y decide moverse en diagonal a la posición (2,7) añadiendo 1,5 al coste y el nodo (2,7) a lista frontera.

6.) Por ultimo explora las casillas adyacentes a (2,7) y observa que el nodo destino o nodo final esta justo arriba en la casilla (1,7). Solo se puede realizar un movimiento vertical para llegar, y este añade 1 al coste final. Al alcanzar la zanahoria el conejo se pone contento, rompe su ayuno y el algoritmo termina la búsqueda.

El coste final es 9.

Al alcanzar la zanahoria, el algoritmo reconstruye el camino óptimo hacia atrás desde (1,7) hasta el inicio en (6,1), usando los nodos padre de cada casilla para trazar la ruta que siguió el conejo para desayunar.

Análisis comparativo de las heurísticas.

Las heuristicas que elegido para el desarrollo de esta práctica han sido:

- Distancia de Manhattan

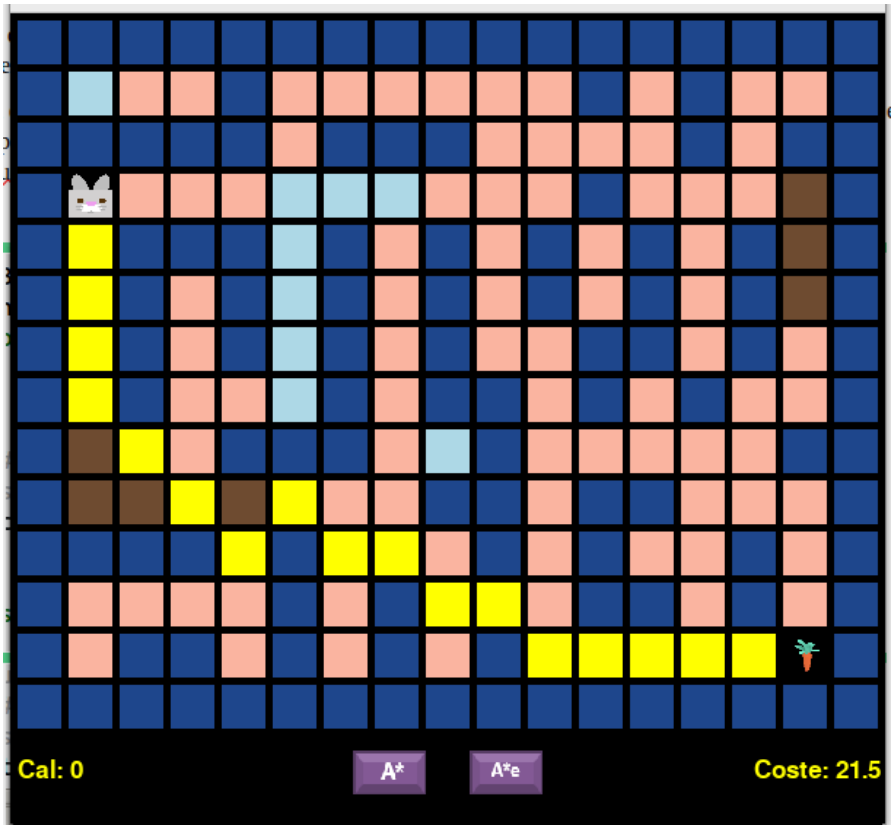
- Distancia Euclide

- Octile

- Chebyshev

La distancia de Manhattan ha sido eliminada por este ejemplo en el que no se cumple el principio de que debería de ser un resultado admisible.

En esta ejecución del mapa.txt colocamos al conejo en la posición (3,1) y a la preciada zanahoria en la posición (12, 15), el resultado correcto es 20,5; pero la ejecución del algoritmo A* con la heurística de la distancia de Manhattan devuelve 21,5. Por lo tanto esta heurística ha quedado DESCARTADA por no ser admisible.



Las otras tres heurísticas propuestas devuelven el resultado correcto, por lo tanto si son admisibles.

La **heurística de Octile** considera un costo adicional para los movimientos diagonales por lo que es muy útil para problemas en los que moverse en diagonal tiene un coste mayor a los movimientos ortogonales.

```
#Octile
dx = abs(pos1.getFila() - pos2.getFila())
dy = abs(pos1.getCol() - pos2.getCol())
return max(dx, dy) + (1.5 - 1) * min(dx, dy)
```

La **heurística de Chebyshev** es buena en escenarios en el que se permiten movimientos en todas las direcciones y el costo de todos los movimientos es el mismo por lo tanto es bastante mala para nuestro problema pues no funciona bien si los movimientos diagonales tienen un coste mayor.

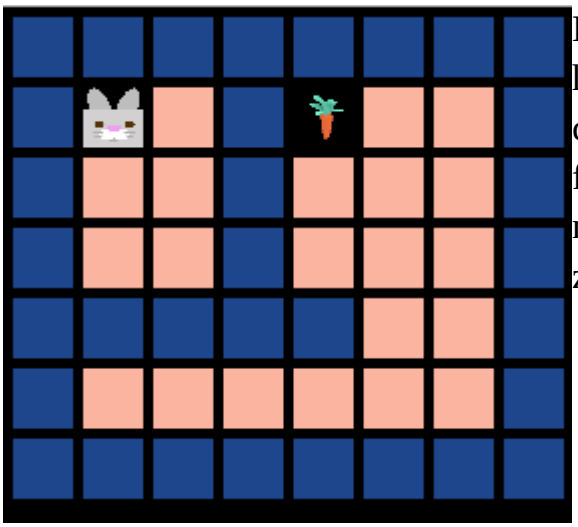
```
#Chebyshev
return max(abs(pos1.getFila() - pos2.getFila()), abs(pos1.getCol() - pos2.getCol()))
```

Por ultimo, la **heurística de Euclides** es ideal para movimientos en línea recta en entornos continuos, aunque menos efectiva para cuadrículas con restricciones de movimiento.

```
#Euclides
return math.sqrt((pos1.getFila() - pos2.getFila())**2 + (pos1.getCol() - pos2.getCol())**2)
```

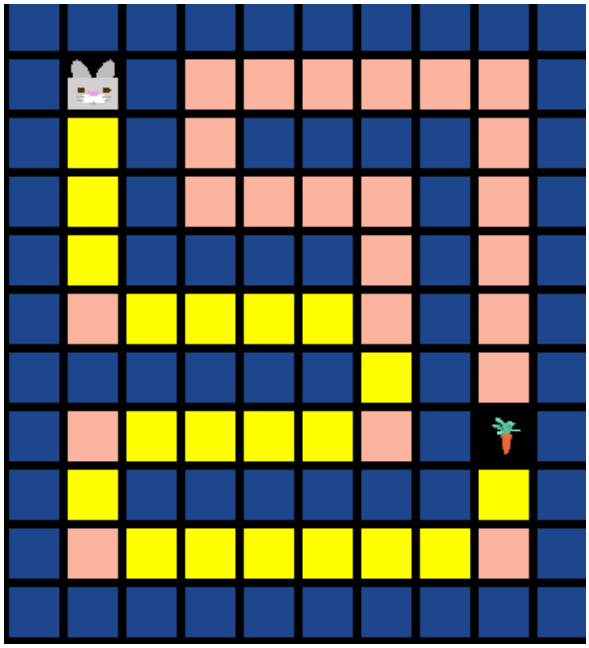
Viendo estas tres definiciones, podemos deducir que la que mas se adapta a nuestro problema es la heurística de Octile. Veamos que dicen las pruebas.

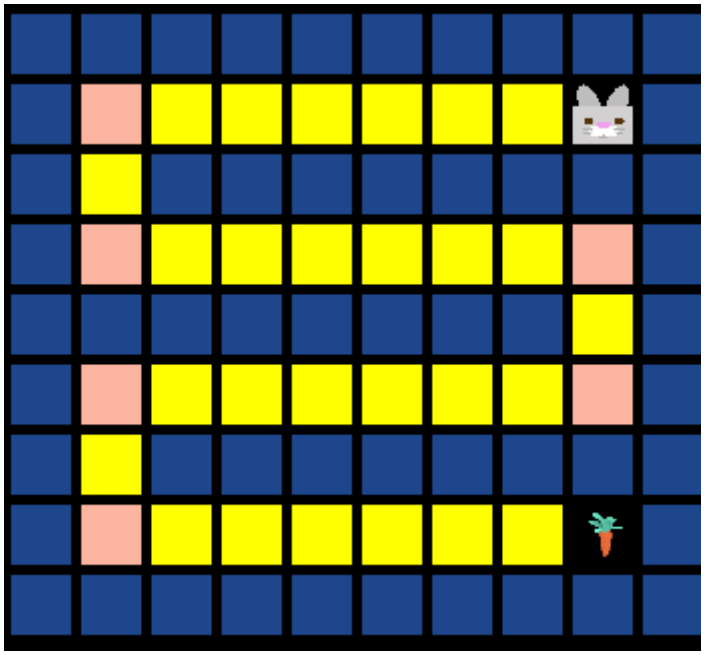
1. Detectar camino inexistente



En está prueba no utilizaremos ninguna heurística en específico, solo la realizamos para comprobar que el Algoritmo A* y A*e funcionan correctamente cuando no existe ningún camino posible para llegar a la zanahoria.

2. Laberinto y Ortogonal.



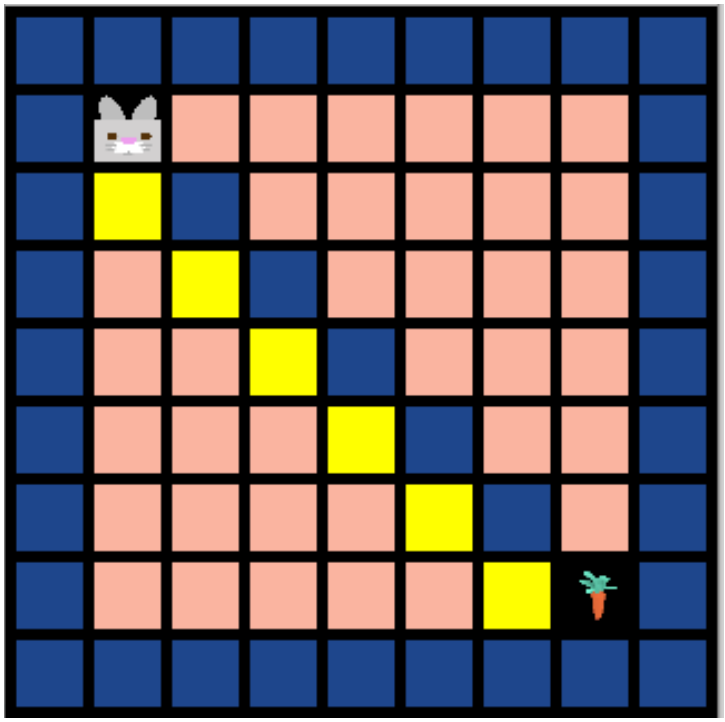


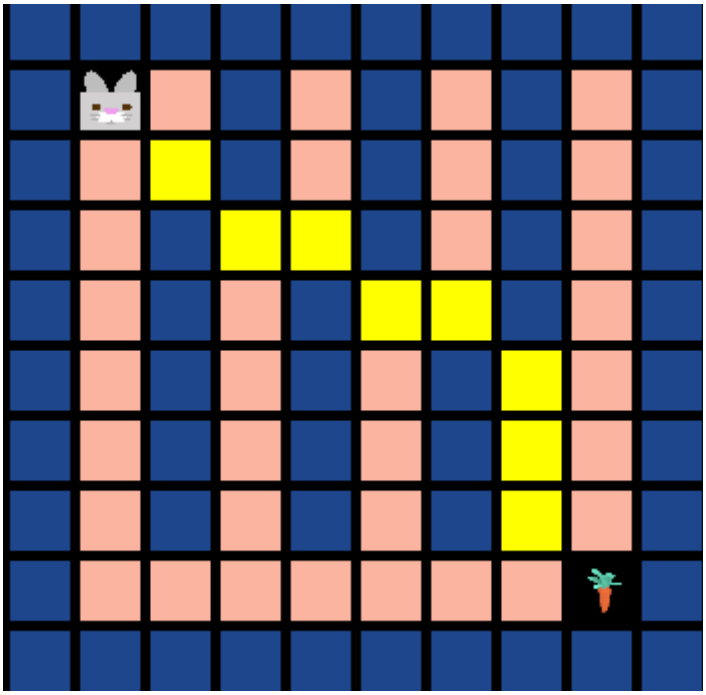
Ambas pruebas tienen un objetivo muy similar, comprobar cual de las tres heuristicas es la mejor encontrando un camino en un mapa en que sus caminos son simples lineas rectas. En la prueba “Laberinto” comprobamos también, la capacidad de elección de algoritmo para saber si es mejor llegar a la zanahoria por arriba o por abajo.

	Octile		Chebyshev		Euclides	
Mapas	Nodos	Tiempo (ms)	Nodos	Tiempo (ms)	Nodos	Tiempo (ms)
Laberinto	39	8,735	39	7,491	39	8,366
Ortogonal	33	7,5	33	7,3	33	5,4

Las tres heurísticas han añadido el mismo número de nodos interiores en ambas pruebas. Respecto al tiempo de ejecución, en la prueba ‘Laberinto’ ha habido una diferencia despreciable, pero podemos dar por ganador a Chebyshev. En la prueba ‘Ortogonal’, Euclides gana con algo más de diferencia.

3. Diagonal y Zigzag





Estas pruebas están diseñadas para comprobar cuál de las tres heurísticas es la mejor, teniendo en cuenta que el movimiento diagonal tiene un costo de 1.5. La que mejor se adapta a esta definición es la heurística de Octile; por lo tanto, debería ser la ganadora de estas pruebas.

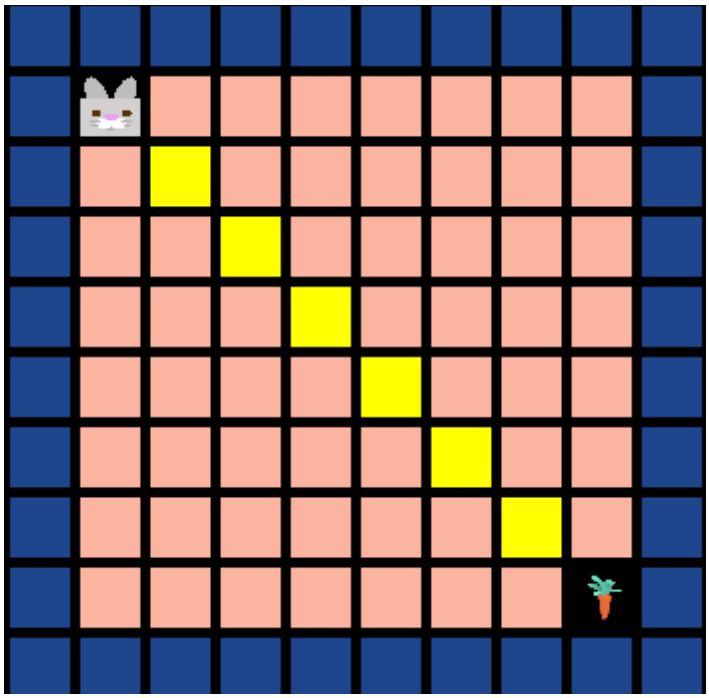
	Octile		Chebyshev		Euclides	
Mapas	Nodos	Tiempo (ms)	Nodos	Tiempo (ms)	Nodos	Tiempo (ms)
Diagonal	13	3,7	21	4,061	19	4,272
Zigzag	17	2,94	22	4,602	18	3,766

Se puede discernir claramente que la heurística de Octile es superior a las otras dos heurísticas en mapas en los que mayoritariamente se utilizan movimientos diagonales. Octile inserta un número mucho menor de nodos en la lista interior durante la prueba "Diagonal", y aunque sea una diferencia despreciable, es el más rápido de los tres en completar la ejecución.

En la prueba "Zigzag", la heurística de Octile es superior respecto a las otras dos en el tiempo de ejecución y en los nodos añadidos, aunque en este caso solo añade un nodo menos que Euclides.

Por lo tanto, Octile es mucho más recomendable para este tipo de escenarios, quedando en segundo lugar la heurística de Euclides y bastante por debajo la de Chebyshev.

4. Plano



En esta última prueba, se nos presenta un mapa totalmente plano sin ningún obstáculo. La heurística de Octile debería dar una estimación más precisa por considerar el costo de los movimientos diagonales. Chebyshev podría sobreestimar el costo al considerar que todos los movimientos tienen el mismo coste, y Euclides debe ofrecer una buena estimación, pero es computacional-mente más costosa debido al uso de raíces cuadradas.

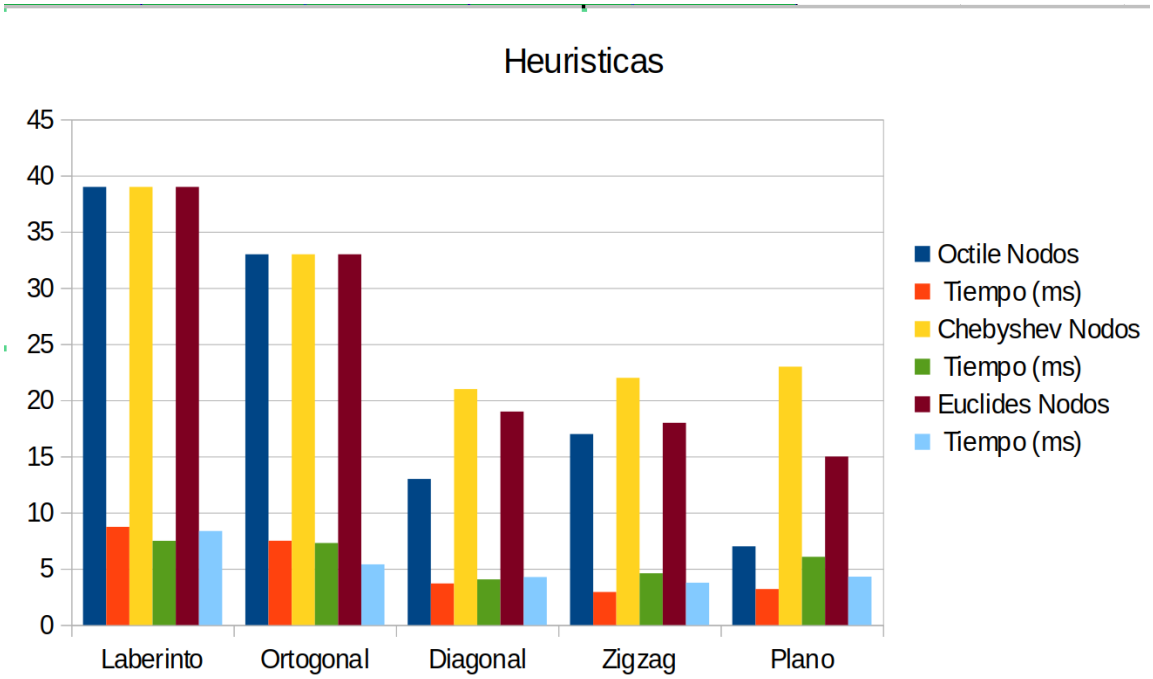
	Octile		Chebyshev		Euclides	
Mapas	Nodos	Tiempo (ms)	Nodos	Tiempo (ms)	Nodos	Tiempo (ms)
Plano	7	3,2	23	6,07	15	4,3

Como hemos dicho antes, Octile es la mejor opción para escenarios sin obstáculos. Con diferencia, es la que menos nodos inserta en la lista interior y la que menos tiempo de ejecución requiere.

La heurística de Euclides, aun utilizando raíces cuadradas, posee un tiempo de ejecución bastante menor que Chebyshev. Una vez más, Chebyshev resulta ser la peor de las tres.

Conclusión

	Octile		Chebyshev		Euclides	
Mapas	Nodos	Tiempo (ms)	Nodos	Tiempo (ms)	Nodos	Tiempo (ms)
Laberinto	39	8,735	39	7,491	39	8,366
Ortogonal	33	7,5	33	7,3	33	5,4
Diagonal	13	3,7	21	4,061	19	4,272
Zigzag	17	2,94	22	4,602	18	3,766
Plano	7	3,2	23	6,07	15	4,3



Puesto que en el problema propuesto del conejo y la zanahoria los movimientos ortogonales tienen un costo de 1 y los diagonales un costo de 1.5, la heurística de Octile resulta ser la mejor, pues inserta los mismos nodos en la lista interior que las otras dos heurísticas en escenarios en los que no se hace mucho uso de los movimientos diagonales. En escenarios en los que se hace uso de movimientos diagonales, inserta muchos menos nodos que las otras dos.

En cuanto al tiempo de ejecución, tarda un poco más que la heurística de Chebyshev en los escenarios de pocos movimientos diagonales y menos que las otras dos heurísticas en los escenarios con más movimientos diagonales.

Por lo tanto, si los movimientos diagonales valiesen lo mismo que los ortogonales, quizás la heurística de Chebyshev sería la mejor, pero como en nuestro caso los diagonales tienen un costo de 1.5, la heurística de Octile es la mejor de las tres.

Pruebas de A* ε

A*e es un algoritmo con el mismo funcionamiento que A*, pero este usará como heurística las calorías. En el mapa hay tres tipos de casillas accesibles, y cada una de ellas tiene un distinto costo calórico;

- Hierba, representado por '.' y un coste calórico = 2.

- Agua, representado por '~' y un coste calórico = 4.

- Roca, representado por '*' y un coste calórico = 6.

El objetivo de A*e será encontrar el camino óptimo, es decir, el camino con un menor coste calórico.

Con un valor de epsilon cercano a 0, el algoritmo se comporta de una forma muy similar al A* clásico, explorando una ruta mas directa y posiblemente de forma mas rápida, pero podría no encontrar el camino menos calórico.

Con un valor de epsilon cercano a 1, tiende a expandir más nodos en cada iteración, explorando caminos alternativos que podrían tener un coste menor en gasto calórico.

Vamos a realizar pruebas con tres valores distintos de epsilon:

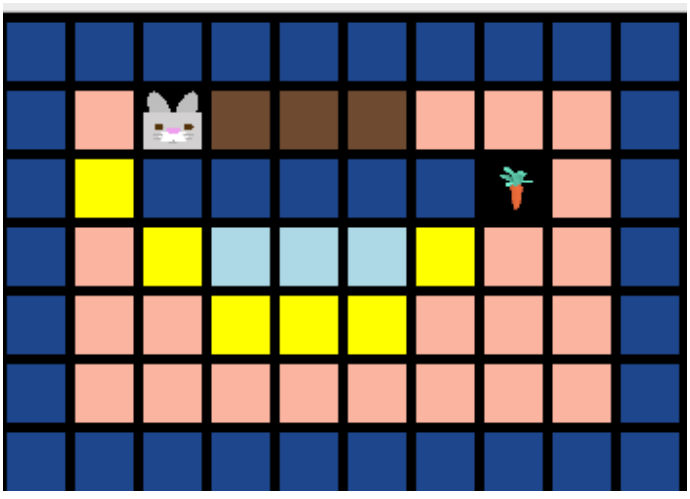
- E1 = 0.1

- E2 = 0.5

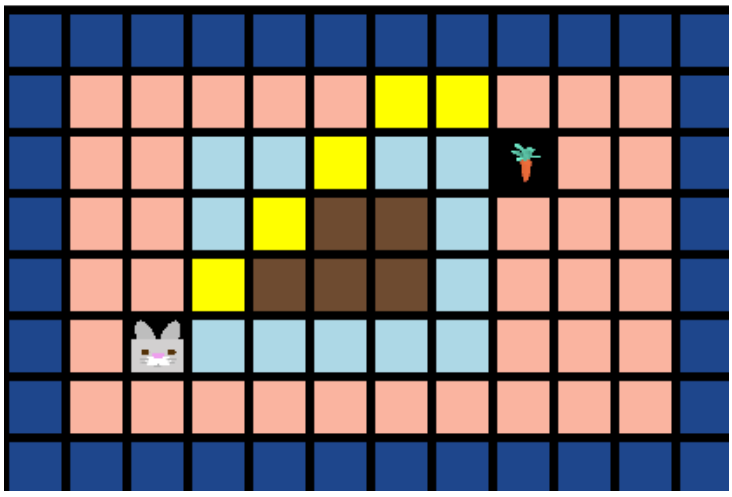
- E3 = 0.9

Someteremos al algoritmo A*e a las siguientes pruebas;

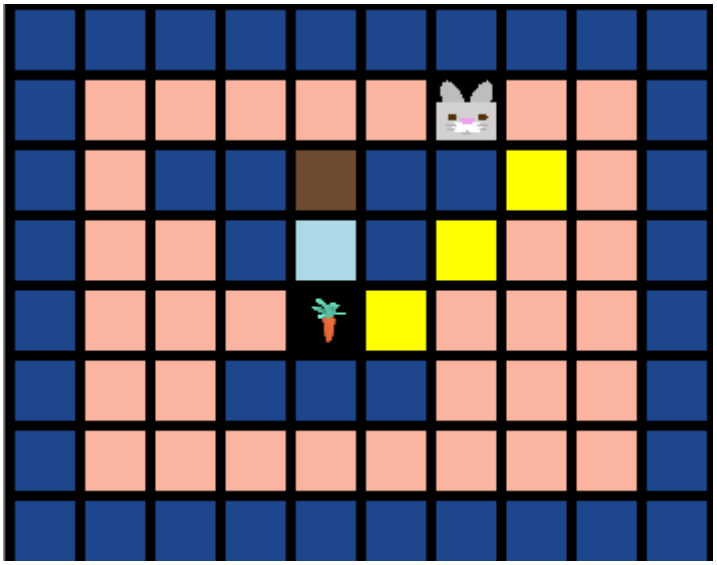
Terrenos



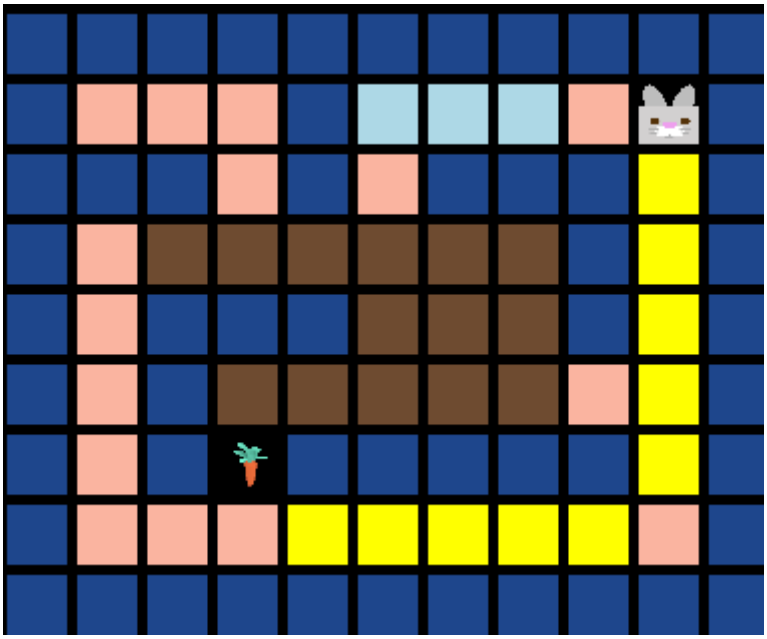
Terrenos 2



Cuello de Botella



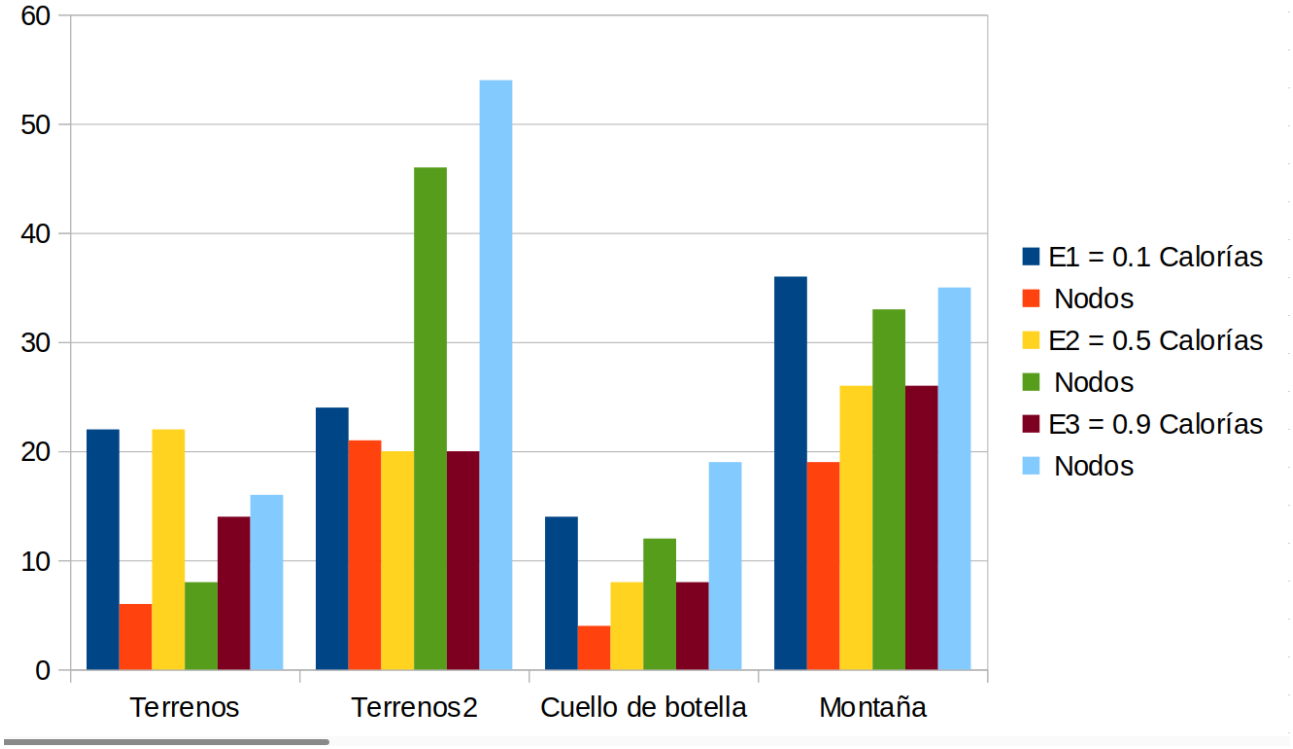
Montaña



Todas estas pruebas están diseñadas para que, dependiendo del valor de epsilon, se obtenga un camino que varía entre el resultado de A* y otros caminos optimizados en términos de gasto calórico. Como hemos explicado antes, cuanto más se acerque epsilon a 0, más se parecerá el resultado al obtenido por A*. Al ajustar epsilon, podemos obtener tanto el camino que encontraríamos usando A* con E1, como rutas de menor gasto calórico con E2 o incluso E3 en casos excepcionales.

Resultados Obtenidos

	E1 = 0.1		E2 = 0.5		E3 = 0.9	
Mapas	Calorías	Nodos	Calorías	Nodos	Calorías	Nodos
Terrenos	22	6	22	8	14	16
Terrenos2	24	21	20	46	20	54
Cuello de botella	14	4	8	12	8	19
Montaña	36	19	26	33	26	35



Con los resultados obtenidos, observamos que, a medida que se exploran más nodos, el coste calórico tiende a disminuir. Aunque en la mayoría de los casos no es estrictamente necesario utilizar E3 para obtener un camino óptimo en términos de calorías, en la prueba de "Terrenos" sí se requiere E3 para alcanzar el camino de menor gasto calórico. En esta prueba, E2 nos devuelve un camino con un coste de 22 calorías, mientras que solo con E3 logramos el camino óptimo real de 14 calorías.

Por lo tanto, si se desea garantizar que el algoritmo siempre encuentre el camino óptimo en términos de calorías, será necesario utilizar E3, aunque en la mayoría de los casos podría resultar menos eficiente en tiempo de ejecución.

Análisis comparativo de los algoritmos A* y A*ε

Tras realizar varias pruebas tanto con las heurísticas posibles para A* como con distintos valores de epsilon en A*e, llego a la siguiente conclusión:

El algoritmo A* se especializa en encontrar el camino más corto entre dos puntos A y B considerando únicamente la accesibilidad de las casillas, sin tener en cuenta el tipo de terreno. De las tres heurísticas admisibles evaluadas en este contexto, la heurística de Octile se seleccionará para el resultado final, ya que las pruebas han demostrado que es la más adecuada para nuestro problema, pues son escenarios donde los movimientos diagonales tienen un coste superior a los movimientos ortogonales.

En cambio, el algoritmo A*e se enfoca en obtener el camino más eficiente en términos de coste calórico entre dos puntos A y B, dándole importancia al tipo de terreno. Esto permite evitar rutas con un consumo calórico elevado, y con un valor apropiado de epsilon, A*e puede aproximarse al camino óptimo o bien priorizar caminos que minimicen el gasto calórico.

Bibliografía

Explicación del algoritmo A*, video que realmente me ayudó a entender el algoritmo y a poder implementarlo junto al pseudocodigo: ***https://www.youtube.com/watch?v=hQa9JTtq4Ok&t=709s&ab_channel=LlamaElitista***

Heurística de Chebyshev y Octile: ***https://chatgpt.com/***

Guía utilizada para poder hacer que la lista interior sea un set:

https://ellibrodepython.com/sets-python

Ayuda para diseñar las pruebas: ***https://claude.ai/***