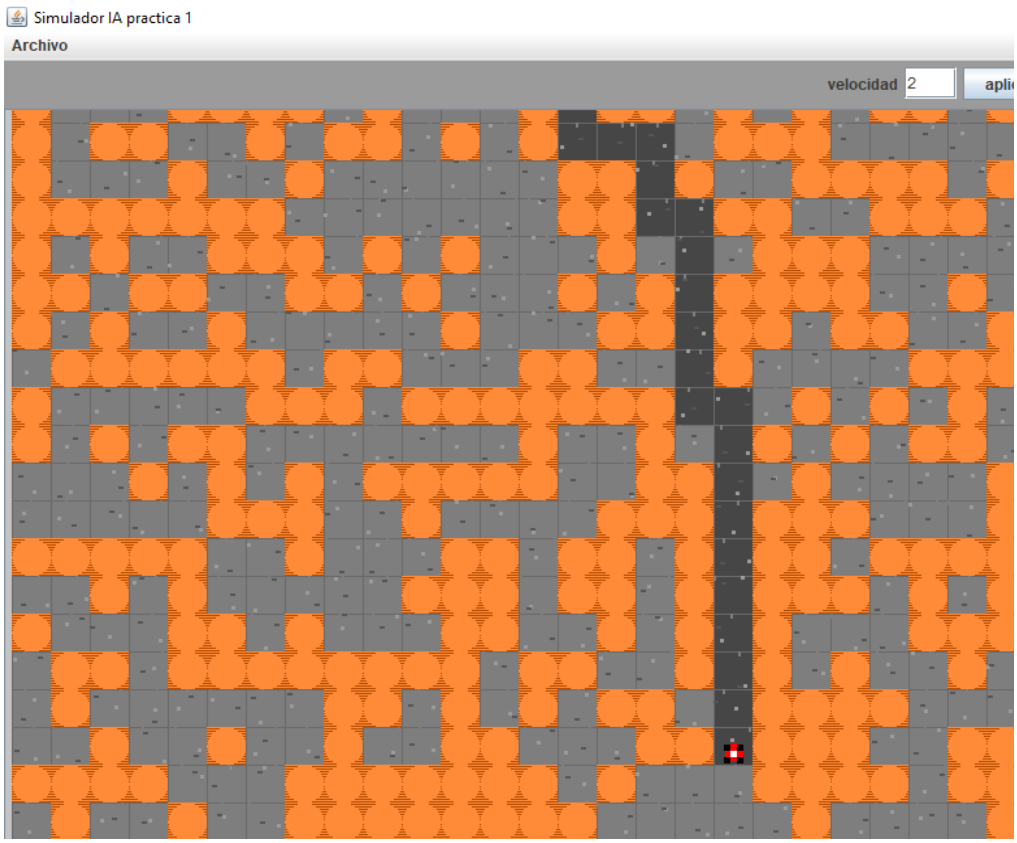
IA 2020-2021: Práctica 1 Búsqueda Estrategias de búsqueda

Marcos Jesús Barrios Lorenzo

Universidad de La Laguna

alu0101056944@ull.edu.es

15/11/2020



Índice

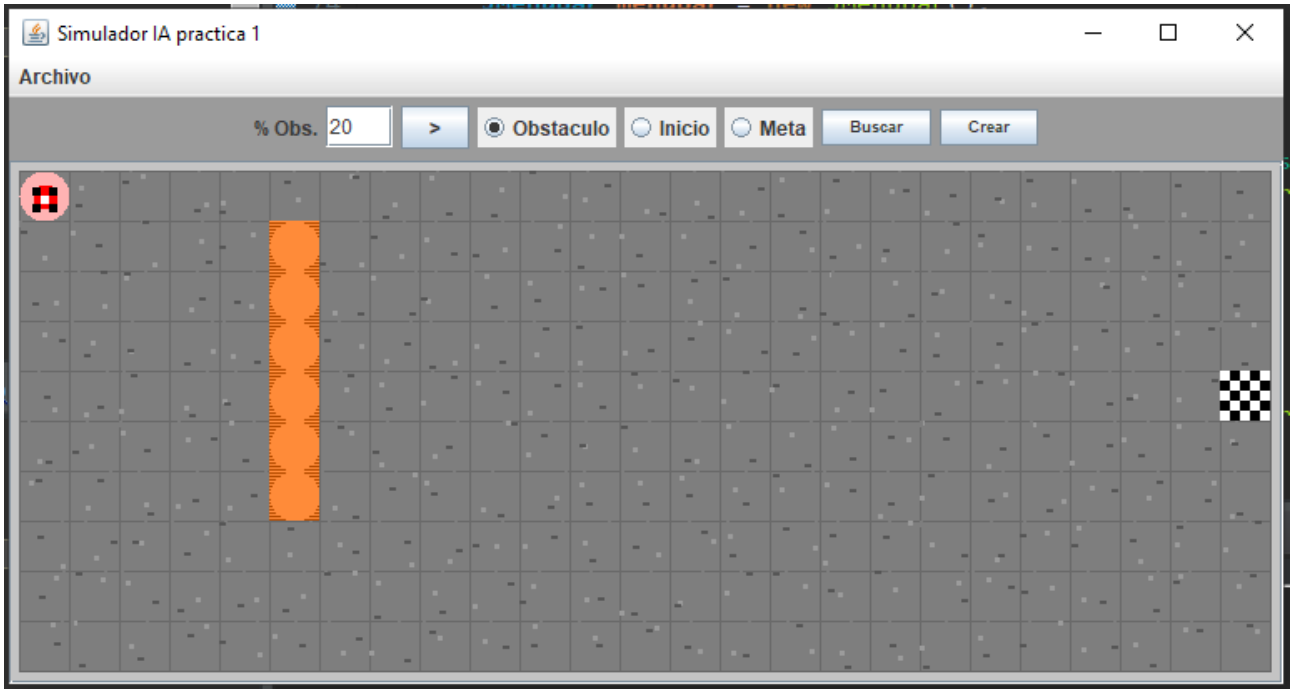
Introducción

Un agente debe **navegar** por un mapa de celdas con obstáculos hasta la localización **meta** por el camino más corto posible. El **problema** es encontrar el camino más corto a través de los obstáculos hasta la meta.

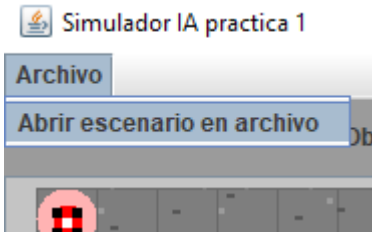
Hay 4 **operadores**: moverArriba, moverAbajo, moverDerecha, moverIzquierda. Un **estado** lo forma la celda que contiene el coche, el mapa de obstáculos y la celda que contiene la meta.

El estado final es la meta que hemos asignado explícitamente en el escenario. El estado inicial es la celda que contiene el coche al empezar la búsqueda.

Entorno de simulación y programación



Tenemos la barra de **menú** con una única opción "Archivo" que al clickearlo nos permite seleccionar la opción de abrir un **escenario** a partir de un fichero. El **fichero** debe tener la siguiente estructura:



Hay que respetar tanto los enunciados como los saltos de línea. Los 3 primeros enunciados deben tener la misma estructura exacta mientras que al último, el de obstáculos, le sigue un número arbitrario de pares fila, columna que indican las celdas que tendrán un obstáculo.

escenarioPrueba.txt | escenario2.txt | new 1

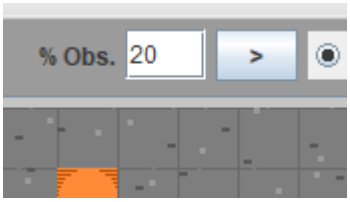
```
numFilas numColumnas | tamaño de la matriz
4 10

filaInicio columnaInicio | posicion celda inicio
3 9

filaMeta columnaMeta | posicion celda meta
1 0

filaObstaculo columnaObstaculo | posiciones de los obstaculos
0 0
0 1
0 2
1 2
1 3
1 4
```

A continuación un segundo ejemplo.

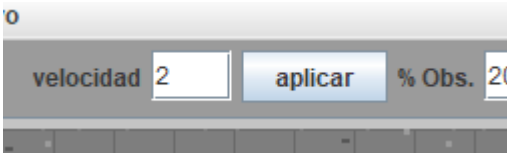
escenarioPrueba.txt | escenario2.txt | new 1

```
numFilas numColumnas | tamaño de la matriz
30 30

filaInicio columnaInicio | posicion celda inicio
2 2

filaMeta columnaMeta | posicion celda meta
15 10

filaObstaculo columnaObstaculo | posiciones de los obstaculos
0 0
3 2
15 10
15 11
15 12
15 13
15 14
15 16
15 17
```

En el panel superior tenemos la opción de aplicar obstáculos **aleatoriamente** sobre el escenario con el porcentaje especificado. Indicamos el porcentaje y apretamos al botón de la derecha para aplicarlo.



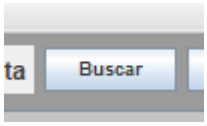
También tenemos la opción de **cambiar la velocidad del coche**, mínimo 1 y máx 3 porque la lógica depende de donde esté el coche y si el incremento es demasiado grande empezará a funcionar mal.



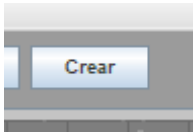
Los botones Obstáculo, inicio y meta nos permiten seleccionar lo que cambiaremos al **hacer click** sobre una celda. Si tenemos obstáculo al clickear una celda se obstaculizará. Si presionamos de

nuevo se liberará. La opción inicio permite mover la celda inicial. La de meta la celda donde está la meta.

El botón de **buscar** ejecuta la búsqueda. Al presionarlo el **coche** se moverá hacia la meta por el **camino** más corto encontrado. Si modificamos el escenario durante la búsqueda, esta fallará. Para que la búsqueda se inicie sobre los últimos cambios hay que presionar buscar de nuevo.



El botón **crear** enseña un diálogo para poner los datos del **nuevo escenario** que queremos crear.



He elegido el lenguaje de programación **Java** con el Entorno de desarrollo **Eclipse**. Dentro de Java la librería **Swing** (ya incluida en Java) para la parte gráfica del programa.

Es el lenguaje y el entorno de desarrollo donde más experiencia tengo. La programación orientada a objetos me permitirá reutilizar las clases programadas en un futuro posiblemente cercano, lo cual es una ventaja añadida. También facilitará el entendimiento del funcionamiento del programa.

Algoritmo de búsqueda

Pseudocódigo:

agregarNodoAExpandir(nodoInicial)

mientras hayan nodos por expandir hacer:
NodoBusqueda siguienteNodo = obtenerNodoSinExpandirConMenorMerito()

si siguienteNodo es meta y ((mejor meta que la encontrada previamente) ó no se ha encontrado meta):
guardar nodo meta encontrado
guardar lista de operadores que nos llevo a la meta

eliminar siguienteNodo de la lista de nodos por expandir
agregar siguienteNodos a lista de nodos expandidos

si ya se ha encontrado una meta:
expandir, calcular méritos y ordenar lista de expandidos *únicamente si siguienteNodo es prometedor (su f < f meta)*
si de lo contrario no tenemos ninguna meta encontrada:
expandir, calcular méritos y ordenar lista de expandidos

NodoBusqueda :
f, g, h, (enteros)
nodosHijo (lista de nodos)
nodoPadre (nodoBusqueda)
operador(accion) (cadena de carácteres)
celda asociada (celda)

Celda:
celdaIzq, celdaDer, celdaArriba, celdaAbajo (celda)
ocupada, inicial, meta (boolean)
fila y columna (entero)

MatrizCeldas:
matrizCeldas (Matriz de celdas)
celdaInicial y celdaMeta (celda)

Coche:
celdaConCoche, celdaAnterior (celda)
listaAcciones (lista de operadores)

Matriz de celdas permite inicializar una matriz para luego **asignar** apropiadamente a cada celda sus **vecinos** dependiendo de si están en una esquina (2 vecinos), en un borde (3 vecinos o en el interior (4 vecinos). Luego al algoritmo se le pasa la celda que contiene el coche para comenzar con su

funcionamiento, junto a la celda final para poder calcular las **heurísticas**.

El algoritmo registra sus acciones(**operadores**) en el coche al encontrar una meta (este contiene la lista de operadores aplicados) que luego el **simulador** utiliza para mover el coche gráficamente sobre el escenario.

He implementado tanto la heurística de distancia de **euclides** como la de distancia de **manhattan**, solo la de manhattan se encuentra en funcionamiento en el programa porque no he puesto ninguna opción dentro del programa para cambiar la heurística pero ambas se encuentran en el código. He probado las dos y la diferencia es que euclides tira más por **diagonales** que manhattan.

Evaluación experimental del algoritmo de búsqueda

...dos funciones heurísticas:

Heurística	Nº nodos expandidos	Longitud camino mínimo	Tiempo de resolución	Tamaño de escenario	Probabilidad de obstáculo
Manhattan	2500	98	83696718 ns = 0.0836967118 s	50x50 (pequeño)	0%
Manhattan	1763	104	27605760 ns = 0.02760576 s	50x50 (pequeño)	25%
Manhattan	382* (abrí un camino)	108* (manual)	74586107 ns = 0,074586107 s	50x50 (pequeño)	50%
Manhattan	137* (abrí un camino)	100* (manual)	613849 ns = 0,000613849 s	50x50 (pequeño)	80%
Euclidea	1457	98	211889746 ns = 0,211889746 s	50x50 (pequeño)	0%
Euclidea	1324	98	63295814 ns = 0,063295814 s	50x50 (pequeño)	25%
Euclidea	337	104* (manual)	25324996 ns = 0,025324996 s	50x50 (pequeño)	50%
Euclidea	125	98* (manual)	597242 ns = 0,000597242 s	50x50 (pequeño)	80%
Manhattan	9899	198	1219314930 ns =1,21931493 s	100x100 (mediano)	0%
Manhattan	7290	210	818649545 ns = 0,818649545 s	100x100 (mediano)	25%
Manhattan	700	220* (manual)	167562393 ns = 0,167562393 s	100x100 (mediano)	50%
Manhattan	-	-	-	100x100 (mediano)	80%
Euclidea	9164	198	3123549524 ns = 3,123549524 s	100x100 (mediano)	0
Euclidea	5885	204	926353328 ns = 0,926353328 s	100x100 (mediano)	25%
Euclidea	-	-		100x100 (mediano)	50%
Euclidea	-	-		100x100 (mediano)	80%
Manhattan	39800	398	68009634800ns = 68,0096348 s	200x200 (grande)	0%

Manhattan	29460	412	70062313100 ns = 70,0623131 s (menos intentos por lo que menor precisión)	200x200 (grande)	25%
Manhattan	-	-	-	200x200 (grande)	50%
Manhattan	-	-	-	200x200 (grande)	80%
Euclideo	38044	398	110604053600 ns = 110,6040536 s	200x200 (grande)	0%
Euclideo	25793	398	50427540400 ns = 50,4275404 s	200x200 (grande)	25%
Euclideo	-	-	-	200x200 (grande)	50%
Euclideo	-	-	-	200x200 (grande)	80%

Conclusiones sobre la evaluación experimental:

- El algoritmo en sí depende enormemente del algoritmo de ordenación que empleemos
- La complejidad espacial es al menos O(N^2)
- La temporal es al menos O(N^2) también.
- El heurístico euclídeo es mucho mejor que el manhattan. Especialmente en complejidad espacial. Excepto en tamaños grandes, que el manhattan ha dado mejores resultados sin obstáculos.
- A probabilidades mayors que 25% no se suele encontrar camino

Conclusiones

- Es relativamente fácil lograr que un agente encuentre un camino si la estructura de datos es la adecuada.
- A* es ineficiente para complejidades espaciales grandes, lo cual lo hace poco ideal para videojuegos de alto presupuesto

Bibliografía

(ninguna)