Inteligencia Artificial



Control de un dron de reconocimiento

Grupo 82

Andrea Martín González 100363947

Sabrina Riesgo Reyes 100363834

Índice

[**Introducción** 3](#_Toc513977056)

[**Parte básica** 4](#_Toc513977057)

[Descripción y explicación de la representación de búsqueda diseñada. 4](#_Toc513977058)

[Descripción de los experimentos realizadas y sus resultados. 5](#_Toc513977059)

[**Parte avanzada** 8](#_Toc513977060)

[Descripción y explicación de la representación de búsqueda diseñada. 8](#_Toc513977061)

[Descripción de los experimentos realizadas y sus resultados. 8](#_Toc513977062)

[**Conclusiones** 9](#_Toc513977063)

[**Comentarios personales** 9](#_Toc513977064)

# **Introducción**

El objetivo de esta práctica es programar el código necesario para resolver un problema de búsqueda propuesto. Para ello hemos tenido que hacer un diseño general, con estados, acciones, etc. Primero hemos completado el código necesario relativo a los estados, es decir, la lista de estados posibles, el estado siguiente al que te encuentras y también la comprobación de los estados meta. Posteriormente hemos insertado costes a cada una de las casillas y creado una heurística, los cuales se van a implementar respectivamente en dos métodos ajenos el uno al otro. Por último, realizamos una batería de experimentos con sus respectivas explicaciones para afianzar el código y comprobar su funcionamiento.

# **Parte básica**

## Descripción y explicación de la representación de búsqueda diseñada.

|  |  |  |  |
| --- | --- | --- | --- |
| MÉTODO | PARÁMETROS | RETURN | EXPLICACIÓN |
| actions | s: estado | lista de posibles acciones sobre el estado s | Controla cada uno de los posibles movimientos del dron en una determinada casilla.  Norte: se puede realizar cuando no se encuentra en la primera fila y cuando no tiene en la casilla superior un agua.  Sur: se puede realizar cuando no se encuentra en la última fila y cuando no tiene en la casilla inferior un agua.  Este: se pude ejecutar cuando no se encuentra en la última columna y cuando no tiene en la casilla derecha un agua.  Oeste: se puede ejecutar cuando no se encuentra en la primera columna y cuando no tiene en la casilla izquierda un agua. |
| result | s: estado  a: acción | estado resultante de aplicar la acción a sobre el estado s | Para obtener el nuevo estado, tenemos que saber la posición de ese estado resultante y además, la posición de las fotos que quedan por hacer.  Posición del estado resultante: aumentamos o disminuimos en 1 la fila o columna del estado actual.  Posiciones de las fotos: creamos una nueva lista, que convertiremos a tupla, en la cual copiaremos todas aquellas posiciones que se encontraban en el estado s y que son distintas de la posición del estado resultante. |
| is\_goal | s: estado | True si el estado s es uno de los objetivos y False en caso contrario | Comprueba si el estado en el que se encuentra coincide con el estado de alguno de los objetivos. |
| heuristic | s: estado | valor de la función heurística para el estado s | Calcula la distancia en número de casillas entre la posición del dron y todas las fotos que le quedan por hacer en ese estado. De los valores obtenidos se queda con el menor. |
| setup | --- | estado inicial  estado final  algoritmo | Inicializa los estados inicial y final y el algoritmo que se va a utilizar. |

## Descripción de los experimentos realizadas y sus resultados.

Para poder comparar los algoritmos y ver como se comportan el coste, el tiempo y la memoria en cada uno de ellos, hemos creado 5 escenarios distintos. La diferencia entre estos escenarios es la complejidad. Para todos mantendremos las mismas dimensiones del mapa, sin embargo, a medida que vayamos cambiando de escenario agregaremos más casillas con agua y más objetivos. En este caso no tendremos en cuenta el resto de los tipos de casillas porque todas tendrán el mismo coste.

La siguiente tabla muestra los resultados de los experimentos realizados.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | Breadth first | Depth first | Iterative limited depth  first | Uniform cost | Greedy | Astar |
| Escenario 1 | Cost | 10 | 18 | 10 | 10 | 14 | 10 |
| Nodes/Iterations | 116 | 131 | 423 | 113 | 54 | 68 |
| Max fringe size | 19 | 54 | 18 | 23 | 32 | 26 |
| Escenario 2 | Cost | 16 | 54 | 26 | 16 | 18 | 16 |
| Nodes/Iterations | 270 | 116 | 2901 | 247 | 93 | 206 |
| Max fringe size | 30 | 60 | 38 | 34 | 44 | 52 |
| Escenario 3 | Cost | 30 | 64 | 40 | 30 | 34 | 30 |
| Nodes/Iterations | 592 | 155 | 8327 | 589 | 248 | 534 |
| Max fringe size | 34 | 88 | 51 | 38 | 44 | 57 |
| Escenario 4 | Cost | 40 | 82 | 62 | 40 | 50 | 40 |
| Nodes/Iterations | 1150 | 157 | 31210 | 1150 | 534 | 1088 |
| Max fringe size | 57 | 101 | 70 | 57 | 54 | 70 |
| Escenario 5 | Cost | 52 | 108 | 76 | 52 | 62 | 52 |
| Nodes/Iterations | 2063 | 142 | 67198 | 2057 | 836 | 1961 |
| Max fringe size | 87 | 90 | 69 | 99 | 126 | 102 |

Nota: los escenarios utilizados se encuentran en software/escenarios\_basica

Para la realización del siguiente gráfico no hemos tenido en cuenta el algoritmo iterative\_limited\_depth\_first porque, como se puede apreciar en la tabla anterior, sus valores correspondientes al aspecto nodes/iterations son demasiado altos en comparación con el resto de los algoritmos, por lo que al incluirlo no se podía observar con claridad el comportamiento de estos.

Resulta complicado ver la línea que representa el algoritmo breadth\_first. Esto se debe a los valores de dicho algoritmo son muy parecidos a los del algoritmo uniform\_cost, similitud que también se puede apreciar en la tabla anterior.

Greedy es uno de los algoritmos que utiliza la heurística. Para comprobar el correcto funcionamiento de esta, utilizando 4 escenarios distintos, obtenemos el coste de cada recorrido del dron y los representamos en un gráfico.

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Greedy | |
| Sin heurística | Con heurística |
| Escenario 2 | Cost | 40 | 18 |
| Nodes/Iterations | 263 | 93 |
| Max fringe size | 66 | 44 |
| Escenario 3 | Cost | 46 | 34 |
| Nodes/Iterations | 287 | 248 |
| Max fringe size | 127 | 44 |
| Escenario 4 | Cost | 58 | 50 |
| Nodes/Iterations | 523 | 534 |
| Max fringe size | 175 | 54 |
| Escenario 5 | Cost | 76 | 62 |
| Nodes/Iterations | 522 | 836 |
| Max fringe size | 169 | 126 |

Como podemos observar, el coste para cada escenario es mayor cuando no se está utilizando la heurística.

# **Parte avanzada**

## Descripción y explicación de la representación de búsqueda diseñada.

|  |  |  |  |
| --- | --- | --- | --- |
| MÉTODO | PARÁMETROS | RETURN | EXPLICACIÓN |
| cost | s: estado  a: acción  s’: siguiente estado | coste de ir del estado s al estado resultante s’ aplicando sobre s una acción a | Para cada tipo de casilla hemos establecido un coste que devolverá la función: desert: 1  plains: 2  hills: 3  forest: 2 |

## Descripción de los experimentos realizadas y sus resultados.

Para comprobar el correcto funcionamiento de la función cost, creamos escenarios que contengan distinto tipo de casillas. (software/escenarios\_avanzada). Una vez ejecutado el programa podemos observar que el coste de cada escenrio se corresponde con la suma de los costes de las casillas por las que pasa.

# **Conclusiones**

Atendiendo a los experimentos realizados hemos llegado a diversas conclusiones que recalcaremos a continuación:

**-Experimento del coste:** Observando la tabla se puede llegar a la conclusión de que el algoritmo depth first crece de forma lineal en cuanto al número de nodos ya que, según vas aumentando el coste, no siempre supera al número de nodos con coste inferior. Sin embargo, la técnica de iterative\_limited\_depth\_first, que es el anterior pero iterativo, crece de manera exponencial, pero de forma muy rápida, los números de nodos aumentan en gran cantidad. Además, ninguna de estas técnicas tiene un aumento de memoria uniforme. En este aspecto el algoritmo depth first tiene un aumento grande respecto a los demás al principio, pero luego empieza a disminuir. El resto de las búsquedas solo aumentan en memoria o se mantienen.

**-Experimento de la heurística:** Claramente siempre que aplicamos la heurística el coste acaba disminuyendo, y la memoria también es menor después de aplicar la heurística en los escenarios elegidos. Sin embargo, el número de iteraciones que realiza no siempre disminuye, en más de un escenario aumenta. Con esto podemos observar que el coste no depende del número de nodos que recorre.

# **Comentarios personales**

Esta práctica te ayuda a la hora de realizar un diseño para resolver un problema de búsqueda, es decir, a saber cómo abarcarlo de primeras. La parte básica es bastante asequible, no tiene una complejidad muy grande quitando la heurística que requiere un poco más de esfuerzo. La parte avanzada sin embargo sí que es un poco más difícil, en comparación. El hecho de tener que programarlo en Python tiene la desventaja de que al no tener llaves, paréntesis, etc. pues tienes que tener mucho control con el problema de la identación. No obstante, quitando eso, es un lenguaje muy similar a otros que hemos utilizado asique no ha sido un gran impedimento a la hora de realizar la práctica. En conclusión, es bastante útil para afianzar y comprender conceptos vistos en clase.