PRÁCTICA

Antonio Marí González y Juan Marí González

Indice

Introducción.

Este informe presenta los resultados obtenidos en la práctica de Inteligencia Artificial, donde se implementaron diversos algoritmos de búsqueda para resolver un laberinto virtual. El objetivo principal de esta investigación fue comparar la eficiencia y el rendimiento de diferentes estrategias de búsqueda en un entorno de juego, con el fin de determinar cuál es la más adecuada para encontrar la solución más óptima.

Se implementaron tres algoritmos de búsqueda: búsqueda en profundidad, A* y MiniMax con poda alfa-beta. Cada uno de estos algoritmos presenta características y ventajas particulares que fueron evaluadas en diferentes escenarios.

A lo largo de este informe se detallarán los algoritmos implementados, la metodología utilizada para evaluar su rendimiento y un análisis detallado de los resultados obtenidos. Finalmente, se presentarán las conclusiones.

Algoritmo de búsqueda por profundidad

La búsqueda en profundidad (Depth-First Search, DFS) es un algoritmo de búsqueda no informada que explora profundamente cada camino antes de retroceder, en caso de que no pueda continuar. Este enfoque sigue una estructura LIFO (Last In, First Out) gracias al uso de una pila, por lo que el algoritmo prioriza siempre explorar completamente un camino antes de pasar a otras opciones.

En contextos como laberintos, la búsqueda en profundidad puede resultar útil para encontrar una solución, pero no es óptima en términos de coste ni de tiempo. Esto se debe a que no tiene en cuenta el peso de las acciones (en este caso, las acciones tienen diferentes pesos) ni la distancia al objetivo, y puede explorar zonas que no son relevantes para llegar de manera eficiente. Por tanto, el rendimiento va a ser peor comparando con otros métodos.

Haciendo varias ejecuciones obtenemos estos datos(tiempo ejecución: cuanto tarda en hacer la búsqueda):

Abiertos	Visitados	Tiempo ejecución
60	8	0.0031104087829589844
74	15	0.0053348541259765625
35	10	0.0026645660400390625
245	39	0.026791095733642578
95	13	0.0063288211822509766

Como la búsqueda de profundidad depende de empezar por la rama izquierda del árbol, depende de cuán lejos de la rama izquierda esté la solución, de eso dependerá los visitados y abiertos, porque el tiempo de ejecución de la búsqueda es casi despreciable.

Algoritmo de búsqueda A*

La búsqueda A* (A estrella) es un algoritmo de búsqueda informada que se utiliza para encontrar el camino más corto entre un punto inicial y un objetivo en un entorno como un tablero, similar al problema planteado en el enunciado. A* combina la búsqueda de costo uniforme y una heurística para guiar al agente a través del laberinto hacia la casilla objetivo de la forma más eficiente posible.

Para implementar A*, hemos tenido que introducir un atributo llamado “peso” dentro de la clase Estado, además de usar una heurística, en nuestro caso hemos decidido utilizar la distancia manhattan desde el punto en el que nos encontramos hasta el destino, simplemente porque nos parecía la más sencilla y evidente.

Abiertos	Visitados	Tiempo ejecución
91	13	0.004550457000732422
107	15	0.006400585174560547
64	7	0.003088712692260742
172	23	0.008761167526245117
27	3	0.001010894775390625

Tenemos un resultado algo obvio, conocemos que A* por ser más óptimo, tendrá que revisar más nodos, para conocer su heurística total(coste + heurística), como en profundidad, empezamos desde la rama izquierda siguiendo un orden preestablecido, tendremos menos nodos abiertos por búsqueda, sin embargo tenemos que revisar la heurística total en A* para conocer el camino óptimo. Por lo tanto, esto equivale a más costo de memoria.

Por otro lado(y en contraparte a profundidad), cómo buscamos un camino óptimo con parámetros, es obvio que elegiremos menos nodos finales que búsqueda(en algunos casos).

Algoritmo Minimax

El algoritmo Minimax es una técnica de búsqueda utilizada principalmente en juegos de dos jugadores con información completa, como el ajedrez o el tic-tac-toe. Su objetivo es ayudar a un jugador a tomar la mejor decisión, asumiendo que el oponente también juega de manera óptima.

Por otro lado, Minimax es un método que genera mucho más abanico de estados, por esto hemos decidido definir una profundidad máxima(que depende el tamaño del tablero). Por tanto, generaremos el árbol hasta y profundidad, elegiremos el estado correspondiente según el turno(MAX o MIN) y elegiremos el camino hasta llegar a dicho estado, para posteriormente continuar generando(desde el estado elegido).

Por lo tanto, el tiempo de ejecución (por su recursividad) será mayor y además usaremos más memoria y CPU, porque es un algoritmo más costoso.