Rodolfo Cacacho 15223
Fredy España 15060

René Olivet 15238
Diego Soler 15415

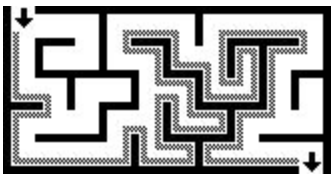
Fase 1 - Proyecto “Un robot sale del laberinto”

Algoritmos existentes para resolver un laberinto

Wall Follower:

El algoritmo Wall follower es el algoritmo más conocido para la resolución de laberintos. Este es conocido por la regla de la mano derecha o izquierda. Si el laberinto está “simply connected”, es decir no hay separaciones entre las paredes o están pegadas a la pared exterior del laberinto.

Por lo cual, el algoritmo consiste en seguir una pared manteniendo la mano derecha siempre junto a la pared. Este puede ser resuelto de manera contraria, al seguirlo con la mano a izquierda. Un laberinto de mayor complejidad puede tenerlo atrapado y llevarlo a un ciclo infinito sin poder encontrarle solución al laberinto.



Pledge:

El algoritmo de Pledge, diseñado para eludir obstáculos, requiere una dirección arbitrariamente elegida. Cuando se halla un obstáculo, por un lado (Wall Follower) se mantiene a lo largo del obstáculo, mientras que los ángulos girados se cuentan. Cuando el solucionador se mueve en dirección original de nuevo, y la suma angular de las vueltas es 0, el solucionador sale del obstáculo y continúa moviéndose en su dirección original.

A diferencia del wall follower, la mano se retira de la pared solamente cuando ambos "suma de vueltas realizadas" y "rumbo actual" están a cero. Esto permite que el algoritmo, evitar obstáculos. Suponiendo que el algoritmo gira a la izquierda en la primera pared, uno se dio la vuelta en 360 grados por las paredes. Un algoritmo que sólo se realiza un seguimiento de "rumbo actual" conduce en un bucle infinito, ya que deja la pared del extremo derecho inferior partida hacia la izquierda y se encuentra con la sección curvada en el lado izquierdo de nuevo. El algoritmo Pledge no sale de la pared de la derecha, debido a que la "suma de vueltas realizadas" no es cero en ese punto. Así se desprende de la pared en todos los sentidos, finalmente dejándolo fuera.

Este algoritmo permite a una persona con una brújula para encontrar su camino desde cualquier punto dentro de una salida exterior de cualquier laberinto bidimensional finito, independientemente de la posición inicial. Sin embargo, este algoritmo no funcionará a la inversa, es decir, encontrar el camino desde una entrada en el exterior de un laberinto para algún objetivo dentro de él.

Algoritmo Recursivo:

Para resolver el laberinto, un simple algoritmo recursivo puede hallar una forma de llegar hasta el final. El algoritmo se le dará una X de partida y el valor de Y. Si los valores X y Y no son en una pared, el método se llama a sí mismo con todos los valores X adyacentes y, asegurándose de que no se utilizan ya los valores X y Y antes. Si los valores X y Y son los de la ubicación final, se guardaran todos los casos anteriores del método que es el camino correcto. Este algoritmo posee la ventaja de la simplicidad de sus métodos, lo cual beneficia su ejecución en tiempo real y su implementación.

Trémaux Algorithm:

Este algoritmo fue inventado por Charles Pierre Trémaux, y es un algoritmo eficiente que requiere dibujar líneas en el suelo para marcar un camino. Además, este algoritmo garantiza para funcionar en laberintos con caminos bien definidos.

Un camino puede no haber sido recorrido, marcado una o dos veces. Cada vez que una nueva dirección es elegida, (desde una unión), el suelo es marcado. Al llegar a una unión, se escoge una dirección aleatoria que no haya sido visitada anteriormente, al llegar un tope, regresa, y si llega a otra unión este escoge otra dirección que no haya visitado previamente. En caso de no existir solución, el método lo lleva a la entrada del laberinto.

Animación de resolución por medio del algoritmo de Trémaux:
https://www.youtube.com/watch?v=6OzpKm4te-E

Dead-end filling:

Dead-end filling o llenado sin salida, es un algoritmo para resolver laberintos cuya metodología es llenar todos los callejones sin salida dentro del laberinto, de manera que solo las formas correctas queden sin llenar. Se puede utilizar para la resolución de laberintos en papel o con un programa de ordenador, pero no es útil para una persona dentro de un laberinto desconocido ya que este método se ve en todo el laberinto a la vez. El método consiste en dos pasos: primero, encontrar todos los callejones sin salida en el laberinto; y segundo, "rellenar" la trayectoria de cada callejón sin salida hasta que se cumpla el primer cruce. Es importante mencionar que algunos caminos no se convierten en partes de caminos sin salida hasta que otros callejones sin salida se eliminen.

Maze-Routing:

Es un método de bajo costo operativo para encontrar el camino entre dos localizaciones del laberinto. El algoritmo se propuso inicialmente para el chip de multiprocesadores de dominio (CMP) y para trabajar con cualquier laberinto basado en grid. Además de encontrar caminos entre dos ubicaciones del laberinto. El algoritmo puede detectar cuando no hay camino entre la fuente y el destino. Además, el algoritmo es para ser utilizado por un viajero de interior sin conocimiento previo del laberinto con la complejidad de memoria fija, independientemente del tamaño laberinto; requiere en total 4 variables para encontrar el camino y detectar los lugares inalcanzables. Sin embargo, el algoritmo no es encontrar la ruta más corta.

El algoritmo utiliza la noción de distancia de Manhattan (MD) y se basa en la propiedad de las redes que los incrementos de MD / decrementos exactamente por 1 cuando se desplazan de un lugar a ninguna ubicación de 4 vecinos.

- Explique las razones por las cuales seleccionó el algoritmo que implementará. Si desechó los algoritmos documentados y decidió hacer uno propio explique por qué. Recuerde que puede basarse en los análisis de complejidad para poder justificar sus elecciones. Tenga en cuenta que debe utilizar el mismo algoritmo que diseñó y probó en la segunda fase del proyecto.

Se decidió utilizar el algoritmo recursivo debido a que pensamos es el más adecuado para la situación planteada. Para realizar ésta decisión, fue crucial un análisis comparativo respecto a los demás algoritmos de solución. Con ésto, se determinó que éste algoritmo tiene las características necesarias para resolver el problema que se asemeja más al discutido grupalmente. Por ésto, el algoritmo recursivo presenta una solución eficiente ante laberintos aleatorios y además la complejidad del algoritmo es adecuada para una implementación en *Java*.

Se descartaron los otros algoritmos investigados, esto se debe a varios defectos que vemos en estos algoritmos. En el algoritmo “Dead-end filling” por ejemplo, es necesario visualizar el laberinto completamente para que el algoritmo *rellene* los callejones que no conducen a la meta, dejando únicamente los caminos que resuelven el laberinto. Debido a que esta no es la situación a la que está sujeta el proyecto, se descartó este algoritmo.

Se descartó el algoritmo Tremaux debido a que pensamos que al hacer movimientos aleatorios al elegir un camino en una intersección podría perjudicar la eficiencia del algoritmo puesto que en el peor de los casos recorrería todo el laberinto antes de alcanzar la meta por lo que el tiempo de ejecución subiría.

Se descartó también el algoritmo “wall follower” y el “pledge” debido a que existe un detalle importante. El problema que tienen estos algoritmos se encuentra en las paredes físicas del laberinto, si estas no se encuentran pegadas, los algoritmos pueden llegar a fallar. Ya que desconocemos las condiciones del laberinto a resolver, decidimos eliminar estos algoritmos de las posibles a utilizar.

- Explique la estructura de datos a utilizar, y diseñe el diagrama de clases que utilizará.

UML - GitHub

- Explique el algoritmo que sirve para salir del laberinto agregando el pseudocódigo o el diagrama de flujo.

Función:

If (posicion actual es la salida)
Devuelve un mensaje de que se logró el objetivo o un true
Else
If (derecha está vacío)
Poner posición actual la posición de la derecha
Marcar posición anterior
Volver a llamar a la función
If (abajo está vacío)
Poner posición actual la posición de la abajo
Marcar posición anterior
Volver a llamar a la función
If (izquierda está vacío)
Poner posición actual la posición de la izquierda
Marcar posición anterior
Volver a llamar a la función
If (arriba está vacío)
Poner posición actual la posición de la derecha
Marcar posición anterior
Volver a llamar a la función
If (derecha está cubierto una vez)
Poner posición actual la posición de la derecha
Marcar posición anterior como segunda vez
Volver a llamar a la función
If (abajo está cubierto una vez)
Poner posición actual la posición de la abajo
Marcar posición anterior como segunda vez
Volver a llamar a la función

If (izquierda está cubierto una vez)

Poner posición actual la posición de la izquierda
Marcar posición anterior como segunda vez
Volver a llamar a la función

If (arriba está cubierto una vez)

Poner posición actual la posición de la arriba
Marcar posición anterior como segunda vez
Volver a llamar a la función

Link github

https://github.com/rodolfocacacho/ProyectoLaberinto/tree/master/ProyectoLaberinto/src

Referencias Bibliográficas:

Mishra, S., & Bande, P. (2008, November). Maze solving algorithms for micro mouse. In *Signal Image Technology and Internet Based Systems, 2008. SITIS'08. IEEE International Conference on* (pp. 86-93). IEEE.

Kazerouni, B. H., Moradi, M. B., & Kazerouni, P. H. (2003, August). Variable priorities in maze-solving algorithms for robot's movement. In *Industrial Informatics, 2003. INDIN 2003. Proceedings. IEEE International Conference on*(pp. 181-186). IEEE.

Babula, M. (2009). Simulated maze solving algorithms through unknown mazes. *Organizing and Program Committee*, 13.

Dang, H., Song, J., & Guo, Q. (2010, August). An efficient algorithm for robot maze-solving. In *Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2010 2nd International Conference on* (Vol. 2, pp. 79-82). IEEE.

Sharma, M. (2009, April). Algorithms for Micro-mouse. In *Future Computer and Communication, 2009. ICFCC 2009. International Conference on* (pp. 581-585). IEEE.

https://github.com/gogs21/maze-solver/blob/master/src/maze/ai/Tremaux.java

https://github.com/gogs21/maze-solver/blob/master/src/maze/ai/RobotBase.java