Algoritmo de resolución de laberintos

Iván Casado Álvarez

Introducción

Realizar un algoritmo para resolver un laberinto parecía algo poco alcanzable cuando entré de primeras a escribir el código al final he dado con un algoritmo que creo (hasta el momento) puede resolver cualquier laberinto que se le plantee. Para el propósito de este documento, visualizar la estructura del algoritmo y la lógica detrás de él, voy a dividir la información en cinco partes:

-Introducción
-Idea inicial
-Partes del programa
-Estructura y funcionamiento
-Por hacer

Idea inicial

Mi planteamiento inicial fue simple: Voy a dibujar un laberinto y un cubo va a recorrerlo y llegar a la meta, ese mismo cubo dejará un rastro indicando qué zonas ha visitado con anterioridad y otro rastro más oscuro para marcar las zonas que no llevan a la meta.

Quizá me vine un poco arriba pensando que podía también hacer que el algoritmo me dibujara la solución más corta para resolverlo (ya lo hará) pero me vi enfrascado en el algoritmo principal en muy poco tiempo como para preocuparse de que me indicara el camino más corto.

Con la excepción de dibujar el camino más corto el algoritmo actual hace todo lo que tenía planteado desde el inicio.

Partes del programa

En este apartado del programa no voy a explicar la estructura del algoritmo, voy a centrarme en mostrar la información que maneja el algoritmo para funcionar. El programa funciona a través de una serie de “div” en forma de cuadrícula, con distintas clases que determinan su función dentro del algoritmo.

La información más importante que maneja el algoritmo son los “div” con clase **“cuadradoBlanco”** y **“cuadradoNegro”,** el camino y la pared, respectivamente. Estos dos tipos de cuadrados son los que van a dibujar el laberinto como tal.

Después tenemos los “div” con clase **“meta”** e **“inicio”**. Estos se marcan con colores distintos al resto, la meta debe estar adyacente a un cuadrado blanco accesible para que el laberinto pueda tener solución, (de lo contrario el algoritmo recorrerá todas las opciones posibles y circulará por el laberinto sin parar). El inicio marca desde donde comienza el algoritmo a resolverse.

A parte de estos tipos de “div” tenemos una serie de botones que nos ayudarán a generar una cuadrícula al tamaño que deseemos hasta el máximo del abecedario (mal diseño por mi parte, aunque al tratarse de una estructura de datos se podría cambiar fácilmente y sustituir las id’s de los cuadrados por xx-xx así podríamos tener un ancho hasta 99). También tenemos la opción de pintar en ese lienzo en blanco nuestro laberinto, añadiendo cualquiera de los divs anteriormente nombrados.

Estructura y funcionamiento

He adjuntado un diagrama que representa el funcionamiento del algoritmo, sobre ese diagrama voy a realizar la explicación, dando por hecho que se ha leído con anterioridad la estructura.

El algoritmo posee una función principal que he llamado “Función principal” (vaya, qué original), ésta función realiza las siguientes acciones: **Lee la posición inicia**l, **calcula el destino y la dirección** del cuadrado, **ejecuta el movimiento**, **comprueba si ha acabado** y en caso de no haber llegado a la meta se **vuelve a iniciar la función** con un timeOut (para generar la sensación de movimiento).

Veamos cada función un poco en profundidad:

-Leer posición inicial: Lee todos los cuadrados del lienzo y devuelve cuál es el cuadrado con clase “actual”, esta función es útil sobre todo en la primera ejecución ya que posteriormente se actualiza solo.

-Calcular destino y dirección: Comprueba la dirección del cubo para así determinar dónde está la “derecha” ya que esta en caso de que todas las opciones tengan la misma prioridad irá a la derecha. Luego lee las opciones posibles, calcula el nuevo destino con el siguiente criterio de mayor a menor preferencia:

-Meta

-Opción menos visitada (hay un registro de número de veces que se ha visitado)

-Derecha, arriba, izquierda, abajo

-Ejecutar movimiento: Esta ejecuta el proceso de desplazar el cuadrado actual y marcar el cuadrado anteriormente ocupado como “transitado” o “intransitable”. Será intransitable si solo tienes una opción disponible, es decir, estás en un callejón sin salida. Al marcarlo como intransitable no podrás volver a ese callejón, así el código es más eficiente. Esta parte también es la que comprueba si hemos entrado en la meta, de ser así se marcará la variable “fin” como “true”.

Por hacer

Por hacer queda el infinito, pero una forma de optimizarlo es hacer que trabaje sobre los datos y no sobre el DOM, también estaría bien añadir la opción para generar laberintos de forma aleatoria. Y bueno, el estilo de la página.