Reporte Proyecto Estructuras de Datos Segundo Parcial

Grupo 10: Jairo Rodríguez

Clase ProyectoESTD2P(Main)

Se creó un atributo de tipo String que guarda la ruta del archivo dentro de la carpeta recurso del proyecto, seguido de la instancia de la clase GrafoVisibilidad para llamar a los métodos generarGrafo(), mostrarGrafo(), mostrarCamino(), simularMovimiento().

Clase GrafoVisibilidad

La clase empieza con las importaciones de librerías e interfaces, así como declarar atributos privados de tipo Graph para el grafo a mostrar, String para los puntos inicial, final y un Random usado para asignar pesos, atributos que son utilizados en los siguientes métodos:

generarGrafo(): recibe un archivo txt y lo lee usando try with resources, se declaran dos listas que guardaran Strings y arreglos de Double para los vértices y obstáculos respectivamente. Se cuenta si es primera línea para identificarlo como vértice de inicio y segunda línea para vértice de meta, se ira añadiendo a la lista vértice las coordenadas del grafo y se añaden dichos puntos al grafo a mostrar, se los configura para que sean fijos (no se muevan conforme nuevas ejecuciones en display(false)). Cuando termine de leer la primera y segunda línea, los guardara como vértice de inicio y fin para el camino a mostrar más adelante.

Después continúa leyendo y separando los vértices para los obstáculos, se forman las figuras y se agrega al grafo por medio de sus coordenadas. Para cada coordenada(nodo) que forma a la figura, se lo añade a la lista de arreglo double obstáculo, lo que sirve para iterar entre pares de vértices y comprobar si se cruzan, si no hay intersección se conectan los vértices mediante la arista y se le da un peso aleatorio entre 1 y 10.

interseca(): recibe dos pares de nodos y la lista de obstáculos, sirve para verificar si la línea entre vértices interseca con algún obstáculo. Con los arreglos de String c1 y c2 se separa cada componente del vértice v1 y v2, y de c1 y c2 se separan los valores numéricos de las coordenadas con x1, x2, y1 y y2. Con un lazo doble se itera cada vértice de obstáculo y se obtiene sus coordenadas, invocando al método esCruzado para verificar si los arcos se cruzan, si se cruzan retorna verdadero, caso contrario falso.

esCruzado(): devuelve true si se cruzan dos segmentos de recta entre dos pares de vértices, calculando un denominador que debe ser cero para líneas paralelas o colineales (no se cruzan) entonces devuelve falso, sino valida que los parámetros t y u estén entre 0 y 1 para decir que una arista se cruza con otra que pertenece al obstáculo, si se cumplen dichos valores, se cruzan y devuelve true.

mostrarGrafo(): muestra el grafo sin el camino más corto.

mostrarCamino(): usa el algoritmo Dijkstra se recorre el grafo desde el nodo inicio hasta la meta por el camino más corto(menos pesado), mostrando el camino tanto en pantalla como en consola (muestra el peso total del camino y el camino como una lista de la forma [x1,y2, x2,y2, … xn,yn]. Se le da color al camino usando styles.css visto en clase.

simularMovimiento(): implementa una simulación de como sería la visita del robot a cada nodo para llegar a la meta por el camino más corto, por medio de un hilo imprime cada llegada saltando un segundo.

Librería GraphStream: usada para mostrar el grafo en ventanas emergentes, conectar vértices mediante arco, darle pesos a las aristas, crear el camino más corto y mostrarlo.

Observaciones:

Los métodos interseca, esCruzado y simularMovimiento no cumplen totalmente la funcionalidad solicitada, puesto que no se eliminan todos los vértices que pasan por un obstáculo y no se muestra un objeto moviéndose por el camino en pantalla, simplemente se simula el movimiento por consola imprimiendo la llegada a cada nodo desde inicio a meta con un hilo.