DEPARTAMENTO
INGENIERÍA INFORMÁTICA
Y CIENCIAS DE LA COMPUTACIÓN
FACULTAD DE INGENIERÍA UNIVERSIDAD DE CONCEPCIÓN

Tarea Computacional:

"Hrognan y el Castillo"

Asignatura: Matemáticas Discretas

Integrantes: Gonzalo Fuica
Hugo Contreras
Ignacio Garrido

Profesor: Guillermo Cabrera

Fecha de entrega: 28/06/2019

Índice

Introducción

Hrognan el bárbaro está atrapado en las mazmorras del castillo del malvado mago Trumptin y debe escapar. Para eso, debe recorrer el castillo teniendo cuidado con los monstruos que lo habitan.

Para esto debemos analizar y tener en cuenta algunas condiciones como:

- Hrognan cuenta con *n vidas* y al terminar su recorrido debe tener al menos 1 vida. Y para derrotar a cada monstruo debe utilizar un número de vidas que depende del monstruo.
- El castillo puede tener uno o más pisos conectados por escaleras y también portales mágicos que transportan de un lugar a otro y que Trumptin ha puesto para confundir a Hrognan.
- Además, existen distintas puertas que necesitan de llaves específicas para abrirse.

El objetivo principal que involucra a nuestro bárbaro es saber cual es el camino a recorrer dentro del castillo de manera que sea, el más óptimo, corto y eficiente dentro de todas sus posibilidades y salga con vida.

Nuestra propuesta de solución para el problema anterior consiste en representar el problema a través de un grafo en donde cada casillero del castillo es representado como un nodo, las puertas, escaleras y portales están representados por aristas, las llaves se representan por un atributo en los nodos.

Se calcula las rutas óptimas desde el nodo inicial el cual representa la ubicación del bárbaro en el grafo, hasta todos los demás nodos, esto con el objetivo de conocer la ruta óptima hasta la salida, para esto el primer paso es calcular el camino óptimo considerando todas las puertas como abiertas, es decir considerando la existencia de la arista que representa cada puerta.

Las puertas que no poseen llave se dejan cerradas y las que sí poseen llave se asumen abiertas, es decir, se crea arista para la puerta, luego se asigna el valor del ID de la puerta + 2 a la arista donde se encuentra la puerta para obtener así el ID de todas las puertas que se necesitan atravesar. Luego de esto se cierran todas las puertas que no estaban cerradas con el objetivo de empezar a buscar las llaves necesarias para abrirlas. Al llegar a un casillero con una llave, la puerta correspondiente se abre, es decir, la conexión entre los nodos es agregada, y se procede a calcular el camino mínimo entre todas las llaves necesarias de conseguir para finalmente calcular el camino mínimo desde la última llave a la salida. Luego todos estos caminos mínimos son unidos, lo que resulta en un camino mínimo final.

Objetivos

El objetivo principal de la tarea es utilizar los conocimientos adquiridos durante el semestre con el fin de poder resolver un problema a través de la generación de un programa computacional que permita realizar lo siguiente:

Se busca crear un programa que lea un archivo con la descripción de un castillo, que incluye:

- Caracterización del castillo: pisos, ancho y alto.
- Posición inicial de Hrognan y *Exit* del castillo.
- Posición de cada muro y llave.
- Posición de cada puerta y número de llave correspondiente.
- Posición de Monstruos y vida asignada.
- Posición de portales y puertas.

Esto con el fin de encontrar la salida del castillo siguiendo el camino más corto tal que Hrognan salga con al menos una vida.

El archivo con la descripción del castillo debe ser ingresado por entrada estándar, así como la información que entrega el programa.

El castillo debe estar implementado mediante grafos que conecten sus pisos a través de arcos que reflejan al castillo de manera correcta , y el programa debe ser capaz de leer los archivos que se le entreguen sin caerse. Se le pregunta al usuario si se desea ingresar otro archivo tras la ejecución.

Modelo

Para resolver esto, se ha modelado el castillo mediante un grafo, considerando lo siguiente:

- Cada casilla es parte del castillo en donde Hrognan puede desplazarse libremente en forma vertical y horizontal. A excepción de los portales y escaleras que desplazan a Hrognan hacia otra posición del castillo.
- Sólo las casillas adyacentes vertical y horizontalmente están conectadas entre sí, y tienen asignados peso 1 en sus aristas (coste 1 para desplazarse)
- Las escaleras y portales son representados en el grafo por aristas que están conectados desde su respectivo nodo y piso desde donde comienza hasta donde termina cada uno
- Los muros y puertas son representadas como “desconexiones” entre casillas, es decir, si una casilla posee uno de estos obstáculos,

Algunas estructuras de datos utilizados en el desarrollo del modelo son:

- Cada nodo del grafo está almacenada en una lista enlazada **tileListHead** de **Tiles** o casillas, que contiene posición, piso, id de la casilla. Además cada nodo posee variables de estado
- Posee una matriz de adyacencia para indicar adyacencia entre las casillas **adyMatr.** Esta matriz guarda 1s y 0s, excepto cuando hay puertas con llaves entre los nodos, en ese caso se guarda la **idLlave del nodo+2** en esa arista.
  Esto con el fin de localizar con mayor facilidad la id de las puertas que son atravesadas durante el cálculo de camino mínimo.
- Las escaleras y portales han sido adheridas a las aristas de la matriz de adyacencia, de forma que agregan nuevas conexiones de coste 1 entre las casillas donde se encuentran.
- Los muros son desconexiones o 0s en la matriz de adyacencia **adyMatr** entre los nodos afectados.
- Las puertas están almacenadas en una lista enlazada **doorHeadList,** las cuales son posteriormente añadidas como muro si es que no poseen llaves (imposibles de atravesar), o como **idLlave del nodo +2** si es que poseen su llave correspondiente
- Los nodos o **Tiles** poseen variables de estado **idLlave** y **lifeMonster**, las cuales indican la existencia de una llave en ese nodo junto a su id, y la existencia de un monstruo junto a su vida, respectivamente. Si **idLlave=** -1 significa que no hay llave en ese nodo. Si **lifeMonster=** 0 significa que no hay monstruo en ese nodo. De otra manera, tienen asignados los valores de su id de llave o vida en ese parámetro.

Cálculo de nodos

Para calcular los nodos correspondientes a las casillas, se leen los datos correspondientes a las dimensiones del castillo. Luego, se asignan las coordenadas X,Y correspondientes junto con el piso en el que se encuentra el nodo. Además se asigna una id a cada nodo, empezando desde id= 0 el primer nodo de la esquina inferior izquierda, y aumentando en 1 a medida que va recorriendo hacia la esquina superior derecha. Los nodos son guardados en una lista enlazada tileListHead.

Las id de las llaves y los monstruos son asignados al nodo correspondiente gracias a una función getIdNodo que devuelve una id dada una posición X, Y y un piso.

Además, las aristas entre nodos son asignadas con peso 1 si es que los nodos son adyacentes, y peso 0 en caso contrario. Cuando se leen los muros desde el archivo, estos asignan peso 0 a los nodos que intersecta. Las puertas son leídas en un principio y guardadas en una lista contenedora **doorHeadList,** luego se asignan como muros si es que no poseen una llave asociada, y con peso 1 si es que poseen una llave. Después de calcular el camino mínimo directo a la salida sin considerar las puertas (ya que tienen peso 1), se identifican las puertas atravesadas en este camino, y se asigna peso **id de puerta+2** a esa conexión de nodos, y finalmente se les asigna peso 0.

Datos almacenados en nodos

Los datos almacenados dentro de los nodos o Tiles son los siguientes:

Variables de posición e identificación: posX, posY, piso, id

Variables de estado: idLlave, lifeMonster

Variable de lista: next

Adyacencia de nodos

La adyacencia de los nodos viene dada por la posición de las casillas. Si una casilla 1 está al lado vertical u horizontalmente de otra casilla 2, entonces existe una adyacencia entre estos nodos. Además, existe el caso especial cuando existen portales o escaleras encima de los nodos, lo que crea nueva conexiones entre estas casillas.

Implementación

Para encontrar las rutas óptimas utilizamos el algoritmo Dijkstra, para esto se utilizó una matriz de costos (**cost[][]**) la cual se inicializa asignando la distancia entre cada cada par de nodos del grafo, si existe arista entre dos nodos, entonces su valor en la matriz de costos será el de la distancia entre ellos, en cambio si no existe arista, la distancia entre nodos será infinito (**INF**), estos valores son obtenidos desde la matriz de adyacencia (**G[][])** la cual es pasada como parámetro a la función dijkstra. Luego, se inicializan los arreglos: **pred** el cual almacena los nodos predecesores de los nodos que ya se ha determinado el camino más corto, **distance** almacena la distancia más corta entre un nodo inicial y cualquier otro nodo, y **visited** almacena los nodos ya visitados, es decir para los cuales ya fue calculado un camino mínimo. Por cada nodo del grafo se asigna una distancia entre ese nodo y el nodo inicial (**startNode**) en el arreglo distance, luego se asigna los valores de pred para cada nodo, este valor corresponde al nodo inicial, luego se los valores para el arreglo visited son asignados para cada nodo como valor cero indicando que no han sido visitados aún. Estos valores son asignados para cada nodo del grafo expecto el para el nodo inicial el cual es asignado con distancia cero en arreglo distance debido a que la distancia entre este nodo y él mismo es cero, además se asigna valor 1 en arreglo visited ya que es el nodo desde el cual se inicia el proceso, por último se inicializa un contador con valor uno indicando que existe un nodo ya procesado (el nodo inicial).

Luego, por cada nodo no visitado del grafo, seleccionamos alguno tal que su valor en arreglo distance sea el menor hasta el momento (esto se hace utilizando la variable minDistance inicializada con valor “infinito”), luego actualizamos minDistance con el valor del arreglo distance par ese nodo indicando de esta forma que la nueva distancia mínima corresponde a la de este nodo seleccionado, además se inicializa el valor para la variable nextNode la cual será la posición del nodo con menor distancia.
Por último, luego de visitar todos los nodos se asigna el nodo con menor distancia como visitado para luego repetir el proceso realizando un ligero cambio en la implementación: se revisa si existe un mejor camino a través de nextNode, es decir, compara con los nodos aún no visitados la distancia y se repite el proceso hasta obtener un camino de costo mínimo desde un nodo inicial hasta el nodo objetivo.

Para calcular el camino mínimo considerando los obstáculos como puertas, llaves y enemigos, de primeras se asignan a las aristas que poseen puertas con llave peso **ID de puerta +2** con el fin de almacenar la id de las puertas atravesadas en un arreglo **idDoorsPassed**. Con estas ID’s se determinan las posiciones de las llaves que hay que conseguir para poder ejecutar el camino mínimo originalmente obtenido, luego se cierran las puertas (se asigna peso 0 en esas aristas). Se calcula el camino mínimo hacia cada llave, desde cada una de las llaves, y después hacia la salida. Luego se unen todos estos caminos mínimos, el cual se convierte en el camino mínimo final.

En la consideración de monstruos, si el bárbaro muere durante el trayecto de este camino mínimo final debido a los monstruos existentes, el programa finaliza ya que no es posible llegar a una salida sin que el bárbaro muera en el trayecto.

Conclusión

En general, desde el punto de vista de trabajo en equipo tuvimos dificultades para poder coordinar nuestros horarios y lograr reunirnos, sin embargo el mayor inconveniente al cual nos enfrentamos fue poder formular una representación adecuada del problema; debiendo cambiar en más de una ocasión la estructura del mismo, lo cual provocó finalmente un retraso en la entrega del proyecto, esto debido a que todos teníamos ideas diferentes para formular el problema

Desde el punto de vista técnico se presentaron diversas dificultades, dentro de las cuales se encuentra la elección de un algoritmo para la encontrar las rutas óptimas que se ajustara a los requerimientos del problemas y más aún, a los requerimientos específicos para nuestra formulación.

En el manejo de los monstruos, inicialmente se tenía la idea de obtener nuevos caminos si es que el bárbaro muere en alguna de las casillas, desconectando aquellas casillas perjudiciales para la vida de Hrognan, es decir, asignándoles peso 0. Luego se calcula el camino mínimo por cada casilla desconectada, para finalmente sumar todos estos caminos mínimos y obtener un nuevo camino mínimo final donde Hrognan no muere. Pero por dificultades técnicas en el código y de tiempo esta idea no se pudo llevar a cabo, y se decantó por desarrollar el manejo de monstruos actual.

Junto con ésto, también podemos mencionar que si bien desde el punto de vista técnico el código necesario para implementar el proyecto en general no era especialmente complejo, sí requirió una gran inversión de tiempo debido a que requirió muchas líneas de código para poder lograr el objetivo de la tarea.