Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
Introducción a la programación y computación 2, 1er. Semestre 2022.

TÍTULO DEL PROYECTO EN MAYÚSCULAS. EXTENSIÓN MÁXIMA DE 35 PALABRAS

202001568 – Nathan Antonio Valdéz Valdéz

Resumen

La empresa Chapín Warriors, S. A. es una empresa que se enfoca en desarrollar robots de rescate inteligentes para ayudar a las bases militares que están dentro de un país con inmersos conflictos bélicos, el objetivo por el cual solicitan nuestra ayuda es para construir el sistema de los drones autónomos que sobrevuelan las ciudades y crean un mapa bidimensional de la ciudad, identificando las calles transitables, no transitables, unidades militares, unidades civiles, puntos de entradas, ubicación de recursos. Este mapa se genera por medio de una malla de celdas en donde esta identificada cada posición, el robot tiene que tener la capacidad de buscar la ruta mas corta y mas segura para llegar a los objetivos que se le pida. La empresa ha desarrollado dos tipos de robots, uno especializado únicamente para el rescate de civiles y el otro para extracción de recursos, estos últimos poseen la capacidad de luchar contra unidades militares.

Palabras clave
Matriz Dispersa, Listas Enlazadas, Graphviz, Archivo XML, Nodos, TDA

Abstract

The company Chapín Warriors, S. A is a company that focuses on developing intelligent rescue robots to help military bases that are inside a country with immersed war conflicts, the objective for which they request our help is to build the system of autonomous drones that fly over cities and create a two-dimensional map of the city, identifying passable and non-passable streets, military units, civilian units, entry points, resource locations. This map is generated by means of a mesh of cells where each position is identified, the robot must have the ability to find the shortest and safest route to reach the objectives that are requested. The company has developed two types of robots, one specialized solely for the rescue of civilians and the other for resource extraction, the latter having the ability to fight against military units.

Keywords
Sparse Matrix, Linked Lists, Graphviz, XML File, Nodes, ADT..

Introducción

Crear un software el cual se debe de utilizar mediante línea de comandos y fácil de utilizar. El programa debe ser capaz de cargar un archivo XML y procesar su información para almacenarla en Matrices Dispersas y Listas enlazadas. Dicha aplicación recibe un archivo XML que contiene la configuración de todo nuestro sistema, como pueden ser las ciudades con su mapa bidimensional de calles, posición de los objetivos de interés, robots disponibles para sus misiones específicas, El usuario podrá seleccionar el archivo XML y el programa se configurará automáticamente para luego facilitarle la selección de misiones disponibles, una vez el usuario selecciona la misión el programa le indica al robot la ruta mas corta y mas segura para realizar dicha misión. Si en dado cazo no existieran robots disponibles para la misión seleccionada o si las unidades militares les ganan en combate, el programa retorna un mensaje de misión fallida.

Desarrollo del tema

El problema que se nos ha solicitado resolver es el siguiente “La empresa Chapín Warriors, S. A. ha desarrollado equipos automatizados para rescatar civiles y extraer recursos de las ciudades que se encuentran inmersas en conflictos bélicos.
Con el fin de realizar las misiones de rescate y extracción, Chapín Warriors, S. A. ha construido drones autónomos e invisibles para los radares llamados ChapinEyes. Los ChapinEyes sobrevuelan las ciudades y construyen un mapa bidimensional de la misma, este mapa bidimensional consiste en una malla de celdas, donde cada celda es identificada como un camino, un punto de entrada, una unidad de defensa, una unidad civil, un recurso o una celda intransitable.”

Para la solución de este problema se usó el lenguaje Python, el cual implementamos la librería “Element Tree” para el manejo de la información que se ingresa
al programa por medio de un archivo XML.
A continuación, se explicará detalladamente todas las
funciones que el sistema posee, así como también el algoritmo aplicado para realizar todo el proceso.

A. Funciones del sistema

1. Carga de Archivos XML.
2. Mostrar ciudades cargadas
3. Mostrar robots cargados
4. Graficar patron de calles por ciudad
5. Verificar misiones disponibles
6. Seleccionar misiones
7. Salir del sistema

B. Explicación del Programa

Al iniciar el programa lo que se le mostrará al usuarioes el menú principal de opciones que tenemos, el usuario deberá ingresar con el teclado numérico la opción de que desee ejecutar. Si el usuario ingresa una opción que no está dentro del menú el programa mostrara nuevamente el menú hasta que el usuario digite una opción valida.

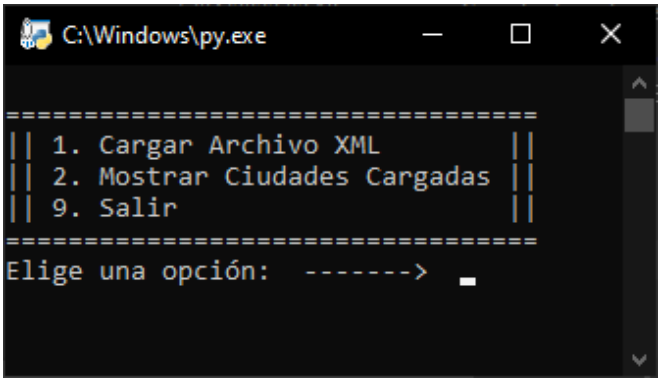

Figura 1. Menú Principal
Fuente: elaboración propia

1. Cargar Archivo XML

Al ejecutar esta función de nuestro menú principal se abrirá automáticamente el explorador de archivos de su sistema operativo para que el usuario pueda seleccionar cómodamente el archivo que desea cargar al programa, por defecto está configurado para que solo nos muestre archivos estrictamente con extensión ".xml" y carpetas, pero se puede cambiar la opción a mostrar todos los archivos de su almacenamiento y se mostraran todos los archivos con cualquier tipo de extensión

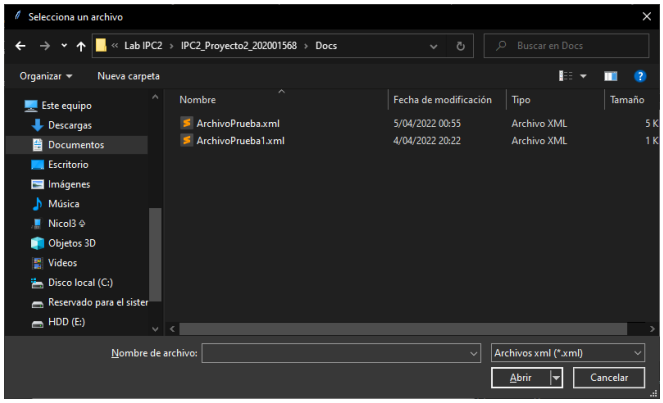

Figura 2. FileChooser
Fuente: elaboración propia

El Programa analizara automáticamente el archivo y lo almacenara en la memoria del sistema por medio de Listas enlazadas, objetos, Matrices Dispersas, etc.

Por ejemplo para almacenar una ciudad el programa genera una lista de ciudades y cada una tiene los siguientes atributos

- Nombre
- Numero de Filas
- Nunmero de Columnas
- Tamaño de Calles
- Patron de calles
- Numero de misiones disponibles para rescate
- Numero de misiones disponibles para extracción
- Apuntado hacia la siguiente ciudad en la lista

```
class NodeCity:
    def __init__(self, Name, Rows, Columns):
        self.Name = Name
        self.Rows = int(Rows)
        self.Columns = int(Columns)
        self.Size = self.Rows*self.Columns
        self.Pattern = SparceMatrix()
        self.Rescue = 0
        self.Extract = 0
        self.Next: NodeCity = None
```

Figura 3. Nodos Ciudad
Fuente: elaboración propia

El Patron de las calles se guarda por medio de una Matriz dispersa, esta matriz almacena únicamente las calles que son transitables, cada nodo de la matriz representa una posición de la ciudad en donde pueden estar ubicadas unidades militares, civiles, de rescate, puntos de entradas, recursos para extracción, estos es la información que guarda cada uno de estos nodos y a continuación un poco mas detallada:

- Tipo de nodo: paso libre, punto de entrada, unidad militar, unidad civil, recurso
- Coordenada en X
- Coordenada en Y
- Vida; si es una unidad militar
- Apuntador arriba, Apuntador abajo, Apuntador izquierda, Apuntador derecha

```
class MatrixNode():
    def __init__(self, x, y, Character):
        self.Character = Character
        self.Xcord = x
        self.Ycord = y
        self.Health = None
        self.up: MatrixNode = None
        self.down: MatrixNode = None
        self.right: MatrixNode = None
        self.left: MatrixNode = None
```

Figura 3. Nodos Ciudad
Fuente: elaboración propia

Para guardar los robots disponibles se usa una lista simple enlazada en donde los atributos de cada nodo son:

- Nombre
- Tipo de robot
- Vida, si es de tipo Extracción
- Apuntador al siguiente

Si algunos de estos datos se carga nuevamente el sistema solo actualizara los datos de dicho objeto repetido

2. Insert

La funcion para llenar la matriz dispersa es una de las mas complejas en este programa, ya que esta solo guarda los datos importantes que se van a utilizar, por ejemplo en este algoritmo no se guardan las calles intransitables porque no se hace ninguna operación con ellas.

Para poder llenar esta matriz lo primero que se hace es separar los patrones de calles y mandarlos cada uno a la inserción, nuestra matriz posee cabeceras que es como nosotros accedemos a los datos, si el tipo de celda que estamos ingresando no tiene cabecera el programa genera su propia cabecera para poder acceder al dato que estamos ingresando Si ingresamos un dato que ya posee una cabecera el programa lo insertara de forma ordenada, para ello también se necesitan cambiar los apuntadores de los nodos anteriores y el nodo siguiente, si el dato que estamos ingresando ya existe en la matriz solo se actualizara y no cambiara de apuntadores.

3. FindCord

Esta función es de las mas utilizadas en el programa ya que nos busca una celda en toda la matriz dispersa con tan solo mandarle como parámetros sus coordenadas en X & Y.

Lo primero que hará dicha función es obtener la cabecera de su coordenada en Y y devolver la primera de sus celdas, una vez tenemos la primera celda de su cabecera esta se ira iterando hacia la derecha hasta que encuentre la coordenada en x, de forma contraria si no encuentra dicha coordenada nos retornará un objeto Nulo y se nos imprimirá en pantalla el mensaje de No se encontró (x,y)

ANEXOS

```
    def Insert(self, PosX, PosY, Character):
        NewNode = MatrixNode(PosX, PosY, Character)
        NodeY = self.Rows.getHeader(PosY)
        NodeX = self.Columns.getHeader(PosX)

        if NodeX is None:
            NodeX = NodeHeader(PosX)
            self.Columns.insertNodeHeader(NodeX)

        if NodeY is None:
            NodeY = NodeHeader(PosY)
            self.Rows.insertNodeHeader(NodeY)

        if NodeX.access is None:
            NodeX.access = NewNode
        else:
            if NewNode.Ycord < NodeX.access.Ycord:
                NewNode.down = NodeX.access
                NodeX.access.up = NewNode
                NodeX.access = NewNode

            elif NewNode.Ycord == NodeX.access.Ycord and NewNode.Xcord == NodeX.access.Xcord:
                NodeX.access.Character = NewNode.Character

            else:
                tmp: MatrixNode = NodeX.access
                while tmp is not None:
                    if NewNode.Ycord < tmp.Ycord:
                        NewNode.up = tmp.up
                        NewNode.down = tmp
                        tmp.up.down = NewNode
                        tmp.up = NewNode
                        break

                    elif NewNode.Ycord == tmp.Ycord and NewNode.Xcord == tmp.Xcord:
                        tmp.Character = NewNode.Character
                        break

                    else:
                        if tmp.down is None:
                            NewNode.up = tmp
                            tmp.down = NewNode
                            break
                        else:
                            tmp = tmp.down
```

Figura 4. Función Insert
Fuente: elaboración propia

```
122
123     def FindCord(self, CordX, CordY):
124         try:
125             tmp: MatrixNode = self.Rows.getHeader(CordY).access
126             while tmp != None:
127                 if tmp.Xcord == CordX and tmp.Ycord == CordY:
128                     return tmp
129                 tmp = tmp.right
130         except:
131             print('No se encontro ({},{})'.format(CordX,CordY))
132             return None
133
```

Figura 5. Función FindCord
Fuente: elaboración propia

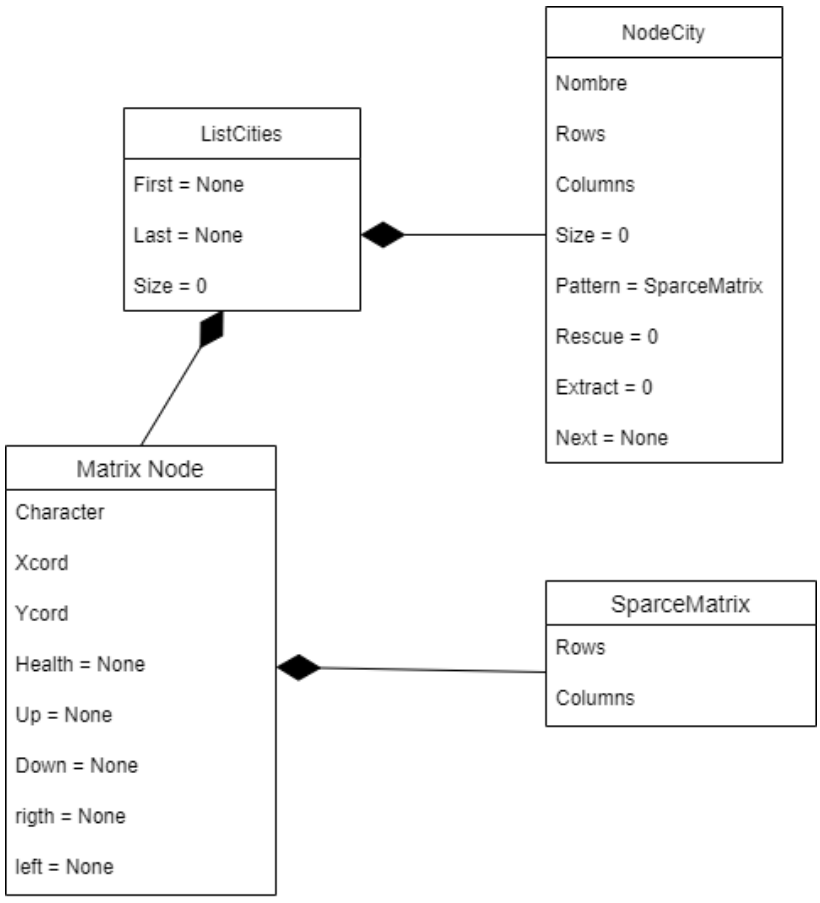

Figura 5. Función FindCord
Fuente: elaboración propia

Conclusiones

La implementación de Matrices dispersas en nuestros programas es una forma de utilizar exponencialmente la memoria que estaremos utilizando, puesto que podremos guardar solo los datos que vayamos a utilizar

Al finalizar este programa se concluyo que la mejor manera de tener nuestros datos guardados es con la implementación de POO (programación orientada a objetos)

La herramienta Graphviz nos ayuda a graficar cualquier matriz como nosotros queramos implementando el diseño que nos podamos imaginar

Referencias bibliográficas

- https://rico-schmidt.name/pymotw-3/xml.etree.ElementTree/parse.html
- https://docs.python.org/3/library/xml.etree.elementtree.html
- https://www.geeksforgeeks.org/xml-parsingpython.html