CHAPINEYES CHAPÍN WARRIORS, S. A

202000510 – Javier Alexander Girón Donis

Resumen

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. Empleando estructuras de datos dinámicas como: listas enlazadas y listas dobles; y el paradigma de programación orientada a objetos (POO).

Consiste en la lectura de un archivo de entrada de tipo XML, archivo que contiene la información, que al ser ingresado se interpretará su información creando las estructuras necesarias para cada ciudad inscrito en él, procesándolo y recorriendo su información para crear la ruta de armado optima. Ruta que será exportada en forma de un archivo graphviz de salida.

Las listas facilitaron el acceso a la información de las ciudades, así como la adaptabilidad a las coordenadas para poder identificar cada movimiento del Dron.

Palabras clave

Estructura de Datos, Clases, XML, POO, Interfaz Grafica

Abstract

The company Chapín Warriors, S.A. has developed automated equipment to rescue civilians and extract resources from cities that are immersed in armed conflicts. To carry out rescue and extraction missions, Chapín Warriors, S.A. has built autonomous and radar-invisible drones called ChapinEyes. The ChapinEyes fly over the cities and build a two-dimensional map of it, this two-dimensional map consists of a grid of cells, where each cell is identified as a road, an entry point, a defense unit, a civil unit, a resource, or an impassable cell. Using dynamic data structures such as: linked lists and double lists12110455; and the object-oriented programming (OOP) paradigm.

It consists of reading an input file of type XML, a file that contains the information, which when entered, its information will be interpreted creating the necessary structures for each land registered in it, processing it and going through its information to create the optimal assembly route. Path that will be exported in the form of an output graphviz file.

The lists facilitated the access to the information of the cities, as well as the adaptability to the coordinates to be able to identify each movement of the Drone.

Keywords

Data Structure, Classes, XML, *OOP, Graphic Interface*

Introducción

Se implementaron las estructuras de datos dinámicas para el desarrollo del software para así contar con un almacenamiento que sea capaz de procesar una gran cantidad de coordenadas.

El paradigma de programación orientada a objetos permite la creación de objetos, cruciales para el desarrollo de este software, separando instrucción de la ciudad con sus respectivas características, siendo cada interpretación de mayor facilidad.

Explicación del programa

Al ejecutarse el software, mostrara una ventana con distintos componentes que se habilitaran y según se trabaje con el programa, permitiendo desarrollar diferentes acciones: Realizar estrategia de una misión, Mostrar Mapas de ciudades ingresadas, Cargar archivos en formato XML y Salir. Cada una de estas opciones serán explicadas a continuación.

El funcionamiento principal del software consiste la lectura y análisis de archivos de tipo XML, guardando su información agrupada en simulaciones, líneas de armado y productos que se pueden realizar por medio de las líneas de armado.

XML es el acrónimo de Extensible Markup Language, es decir, es un lenguaje de marcado que define un conjunto de reglas para la codificación de documentos. Simplificando los conceptos expuestos podemos comenzar con que:

El lenguaje de marcado es un conjunto de códigos que se pueden aplicar en el análisis de datos o la lectura de textos creados por computadoras o personas. El lenguaje XML proporciona una plataforma para definir elementos para crear un formato y generar un lenguaje personalizado.

Los archivos XML que se ingresaran al software contienen información que será almacenada con el uso de distintos tipos de nodos y listas programadas de forma manual.

La programación informática considera que un nodo es cada uno de los elementos de una lista enlazada, un árbol o un grafo en una estructura de datos. Cada nodo tiene sus propias características y cuenta con varios campos; al menos uno de éstos debe funcionar como punto de referencia para otro nodo.

Una lista enlazada es una estructura de datos que puede utilizarse para la implementación de nuevas estructuras (tales como las colas, las pilas y sus derivados) y está formada por una serie de nodos que almacenan, además de la información deseada, un enlace, un puntero o una referencia al nodo que lo precede, al posterior, o bien uno a cada uno. La ventaja fundamental de una lista enlazada en comparación con un vector convencional es que sus elementos no presentan un orden rígido ni relacionado con el que tuvieron al momento de ser almacenados, sino que éste depende del enlace que posee cada nodo, y puede ser modificado cuando así se desee.

Los archivos XML de Lista de Ciudades está conformado por: Ciudad que contiene el nombre de la misma, filas, columnas caracteres con los cuales posteriormente diseñaremos la ciudad y las unidades militares con su respectiva coordenada de ubicación.

De la misma manera en este mismo archivo podemos encontrar un apartado llamado robots donde se nos proporcionara el tipo de dron con el que contamos y su capacidad caso este sea ChapinFighter o caso contrario como seria ChapinRescue contaremos solo con su nombre.

La lectura de los archivos de entrada se utilizó la librería “xml.etree.ElementTree”, importándola internamente desde Python, que trabaja este tipo de archivos con la estructura de árbol de nodos, es decir, trabaja los archivos con ramificaciones y distintas propiedades, cada ramificación equivale a un nivel de profundidad, para acceder a niveles más profundos se agregaran un par de corchetes con el identificador de hijo que queramos acceder.

La biblioteca ElementTree incluye herramientas para analizar XML usando APIs basadas en eventos y documentos, buscando documentos analizados con expresiones XPath, y creando nuevos o modificando documentos existentes.

Para el análisis de los datos y su almacenamiento se optó por la creación de nodos, contando con cinco

tipos distintos de nodos, conectados entre sí por medio de los enlaces de sus respectivas listas enlazadas. La primera lista fue la de las líneas que por medio de una estructura repetitiva "for" se accedió a todos los registros de este tipo del primer archivo de entrada, por cada registro encontrado se creara un nodo de tipo "Nueva_lista" y se almacenara su información encontrada y mencionada anteriormente.

La siguiente lista creada durante el proceso de lectura será la lista de "Nuevo_guerrero", que sigue los mismos pasos de la lista anterior, con la diferencia del tipo de nodo e información que almacenara; los nodos serán utilizados para almacenar las posiciones en el tablero donde se encuentre una unidad militar que posteriormente serán mostradas dentro del mismo.

Por último, en la misma lectura del XML tenemos la creación de la lista "Nuevo_dron" y esta almacenara el nombre, tipo y capacidad de cada dron ingresado dentro del archivo en lectura. Estos se almacenarán para posteriormente utilizarlo como usuario en la realización de misiones.

Todas estas listas se declararán en formato global para poder ser utilizadas en las diferentes funciones que utilizaremos dentro de nuestro programa.

Al leer los archivos mencionados, nuevamente imprimiremos nuestro menú de opciones el cual brindara la opción de poder ingresar mas archivos de entrada los cuales podrán tener la opción de poder modificar la información del archivo previamente ingresado o agregar nuevas ciudades y drones dentro de nuestra lista enlazada, esto se controlará por medio de una función establecida que verificara dentro de nuestra lista enlazada si el nombre ingresado en el nuevo XML ya existe o no.

Una vez cargados los archivos que deseamos, podemos acceder a graficar nuestras ciudades por medio de la función "2. Mostrar Mapa de ciudades" las cual solicitará el nombre de la ciudad que deseamos gráficar por medio de Graphviz, esta función desplegará las ciudades guardadas dentro de nuestros nodos en caso no conozcamos su nombre.

Graphviz se interpretará su contenido y devolverá una imagen con la ciudad solicitada (en caso esta no exista, el programa indicará que no existe).

Graphviz (Graph Visualization) es un conjunto de herramientas de software para el diseño de diagramas definido en el lenguaje descriptivo DOT.1 Fue desarrollado por "AT&T Labs" y liberado como software libre con licencie tipo Eclipse.

Graphviz consiste en un lenguaje de descripción de gráficos llamado DOT, un conjunto de herramientas y librerías que pueden generar o procesar archivos DOT.

Dot es una herramienta en línea de comandos para producir imágenes en capas de grafo dirigido en una variedad de formatos de salida (PostScript, pdf, svg, png, etc.).

Los archivos tipo DOT que esta extensión interpretara se generaron a través de la función "llenado_lista" la cual solo solicitará el nombre de la ciudad a graficar y esta hará la búsqueda dentro del nodo principal las características de esta ciudad como lo es: filas, columnas y caracteres junto con sus respectivas unidades militares las cuales hará un llamado por medio de la función Devolver al nodo guerreros y tomara todos los guerreros con el nombre de la ciudad en cuestión

Una vez llevados los datos necesarios para su implementación, estos serán llevados a la función mas importante dentro del programa y es dentro del archivo Matriz.py el cual recibirá estos datos y llevara los siguientes pasos para completar la creación de una matriz dispersa la que se usara tanto para graficar las ciudades, como para poder ejecutar las misiones que más adelante se mencionaran.

Una vez dentro del archivo Matriz.py usaremos la función reiniciaraiz la cual borrará cualquier matriz ya ingresada, una vez eliminados utilizaremos la función llenar_matriz la cual heredaremos las filas y columnas de la ciudad solicitada, esto debido a que la función creara por medio de la función insertar, recorrerá con un doble for que simulara cada posición en Y y X dentro del nodo, estas posiciones se irán almacenando dentro de un nodo exclusivo del archivo Matriz.py los cuales cuentan con x, y, izquierda, derecha, arriba, abajo, color y capacidad. Estos cumplen con la función siguiente:

X: Almacenará el dato del primer for que recorreremos por medio del dato fila.

Y: Almacenaremos el dato proporcionado simulando una coordenada por medio del dato proporcionado por columnas.

Izquierda: Por medio de este valor haremos la conexión con el nodo que se encuentre una posición anterior al que deseemos, estas conexiones se harán mas adelante por medio de las coordenadas ingresadas por estos for´s múltiples.

Derecha: La posición derecha será la conexión con el nodo posterior al creado, sin embargo, esta posición juega un rol importante dentro del funcionamiento ya que será al inicio la única conexión con todos los nodos y posteriormente se recorrerá cada nodo por medio de esta y se irán desenlazando por medio de otra función que más adelante se describirá.

Arriba: Por medio de este valor haremos la conexión con el nodo que se encuentre una posición superior al que deseemos.

Abajo: Por medio de este valor haremos la conexión con el nodo que se encuentre una posición inferior al que deseemos.

Al finalizar el proceso de llenado, utilizaremos la función "unir_nodos" la cual nuevamente heredaremos las filas y columnas ya que por medio de esta utilizaremos un for doble para buscar cada coordenada y con tres if evaluaremos cual coordenada es su izquierda, derecha, arriba y abajo y la adjuntaremos al nodo en cuestión.

Una vez hechas las conexiones, eliminaremos la unión de cada nodo del lado derecho en cada final de columnas por medio de la función "borrar_derecha".

Por último, nos encontramos con función de "llenar_colores" la cual almacenara en cada nodo el color que le corresponde a cada coordenada ingresando el texto que se proporciono por medio del XML, esta función identificara los siguientes caracteres:

- * será color negro
- E será color verde
- C será color azul
- R será color gris
- " " espacio en blanco será color blanco

Habiendo llenado los colores en su coordenada correspondiente, llamaremos al nodo guerreros con la función devolver y los agregaremos a nuestra matriz por medio de la función "editar_coordenadas_robots" la cual ingresara el color rojo que significa unidad militar y la capacidad de esta unidad (todo nodo que en su apartado de capacidad sea igual a None, esta significa que no cuenta es unidad militar).

Una vez llenada la matriz ya podremos llamar la función de "imprimir_total" la cual hará que se recorra cada coordenada de nuestra matriz por medio de un doble for como se ha mencionado anteriormente con herencia, se hará una evaluación de que color se encuentra en el nodo que se encuentre en esta posición y con una sentencia de condición If, evaluaremos cual es el color que corresponde para agregar la línea de código creada en Graphviz.

Por último, tenemos la función principal del programa, esta esta en la función No. 1 de nuestro menú principal el cual se denomina como "Realizar estrategia de una nueva misión" la cual nos desplegara un menú de selección donde dirá si deseamos elaborar un rescate o extracción ya que dependiendo de que función deseemos se utilizaran drones diferentes como ya se mencionó con anterioridad.

Si se ingresa "R" se iniciará la función "misión_rescate" la cual verificara que existan Drones de tipo ChapinRescue (esta función está contenida dentro de un Try Except ya que al estar el nodo vació el programa podría mostrar fallos), una vez verificada la existencia de dicho Dron y existan drones de este tipo, se nos desplegara el listado de nombres de dichos drones para poder ingresar el que deseamos utilizar.

Una vez seleccionado el dron para la misión, se iniciará la función buscar para encontrar dicho nombre dentro del nodo y si este existe, se utilizará la función imprimir de nuestro nodo principal para ingresar la ciudad en la cual deseamos realizar el rescate y se nos solicitará el nombre que deseemos.

Una vez ingresada la ciudad se utilizará nuevamente la función "llenado_lista" enviando el nombre de la ciudad, esto con el fin de crear la matriz con el mapa y mostrar el mapa de la ciudad solicitada ya que posteriormente se verificará la existencia de unidades civiles por rescatar, caso contrario el mapa se utilizará como evidencia que no existe una unidad civil.

Ya visualizado el mapa y la unidad civil que se desea rescatar, se solicitará la coordenada en Y y luego en X dentro del mapa, en caso solo exista una unidad civil, esto no será necesario y solo se utilizara la coordenada donde exista el único color azul dentro de la matriz. De la misma manera se hará con el punto de inicio, se verificará la existencia de este y si existen mas de un punto de ingreso, se solicitarán las coordenadas en Y y X que se desean utilizar.

Culminado este proceso se ejecutará la función "realizar_mision" la cual solicitará las coordenadas de la unidad civil, coordenadas de entrada, la matriz creada con cada coordenada del tablero y por ultimo las filas y columnas totales de dicho tablero para poder imprimir dicho tablero con la misión finalizada.

En caso de seleccionar "E" se ejecutara la función "misión_extraccion" la cual seguirá el mismo proceso que la función anterior, seleccionar dron, seleccionar ciudad, seleccionar coordenadas de ingreso y de los recursos que se desean extraer, sin embargo el cambio en esta función es la herencia del listado de guerreros dentro de este tablero ya que si el dron encuentra uno de ellos en el tablero tendrá que enfrentarlo y se hará la comparación de capacidad de cada uno y si el dron es superior, este podrá vencerlo. Caso contrario el dron no podrá ejecutar la misión de extracción.

Para la elaboración de las misiones, tanto de extracción, como de rescate, se utilizaron las mismas bases las cuales son:

Al iniciar trasladaremos la matriz que enviamos por medio de herencia de datos a la matriz raíz de nuestro archivo, con esto posteriormente utilizaremos la función "buscar_coordenadas" la cual le daremos las coordenadas tanto del civil/recurso y buscaremos el nodo de posición que encontremos dentro de la matriz.

Una vez contando con los nodos que deseamos haremos una comparación de posición tanto en Y como en X la cual con un if sabremos si el recurso o civil que deseamos rescatar se encuentra hacia abajo o hacia arriba, de igual manera compararemos los movimientos si deben ser hacia la izquierda o abajo ya que esto será esencial para la función que ejecutaremos. Una vez sabiendo la posición en base a las comparaciones de dichas coordenadas ejecutaremos cualquiera de las siguientes funciones:

- Abajo_derecha
- Abajo_izquierda
- Arriba_izquierda
- Arriba_derecha

Por medio de estas funciones se ejecutó el método de la "Recusividad" la cual evaluara si la casilla siguiente es blanca, caso contrario evaluara hacia arriba, derecha o izquierda, una vez finalizado enviaremos nuevamente el nodo al método y cambiaremos el color que recorrimos por amarillo.

La recursividad se detendrá cuando las coordenadas deseadas sean las mismas que las del nodo que se envió tanto del recurso o del civil.

Una vez finalizada, realizaremos una impresión del mapa donde se mostrará el recorrido por el dron para cumplir dicha misión, en caso esta misión no se pueda realizar, solo se mostrará que no se pudo cumplir y se desplegara el menú de inicio.

Conclusiones:

El análisis y creación de nodos y estructuras de datos dinámicas se ha utilizado por ser un recurso que haga relaciones de distintos tipos entre la información y poder acceder entre ellos de una manera práctica.

Es importante modelar un problema antes de empezar a realizarlo, por medio de diagramas de flujo y clases para comprender, analizar y encontrar la solución de mejor manera.

Al utilizar una interfaz gráfica, permite al usuario tener mejor interacción con el programa.

Se usaron las herramientas de Graphviz, y XML para la creación de los reportes de cada misión creado.

Anexos:

Archivo de entrada solicitado (ciudades.xml)

```xml
<?xml version="1.0"?>
<configuracion>
    <listaCiudades>
        <ciudad>···
        </ciudad>
        <ciudad>
            <nombre filas="10" columnas="10">CiudadGuate</nombre>
            <fila numero="1">"**********"</fila>
            <fila numero="2">"***    C  R"</fila>
            <fila numero="3">"*** ******"</fila>
            <fila numero="4">"*** ******"</fila>
            <fila numero="5">"***      C*"</fila>
            <fila numero="6">"E   ******"</fila>
            <fila numero="7">"*** ******"</fila>
            <fila numero="8">"*** *****C"</fila>
            <fila numero="9">"*** ******"</fila>
            <fila numero="10">"***       R"</fila>
            <unidadMilitar fila="2" columna="5">10</unidadMilitar>
            <unidadMilitar fila="5" columna="5">20</unidadMilitar>
        </ciudad>
    </listaCiudades>
    <robots>
        <robot>
            <nombre tipo="ChapinFighter" capacidad="50">Robocop</nombre>
        </robot>
        <robot>
            <nombre tipo="ChapinRescue">Ironman</nombre>
        </robot>
        <robot>
            <nombre tipo="ChapinFighter" capacidad="10">MaxSteel</nombre>
        </robot>
        <robot>
            <nombre tipo="ChapinRescue">OptimusPrime</nombre>
        </robot>
        <robot>
            <nombre tipo="ChapinFighter" capacidad="100">Megatron</nombre>
        </robot>
    </robots>
</configuracion>
```

Figura 1. Archivos de Entrada

Fuente: Elaboración propia, 2022

Diagrama de Clases del software

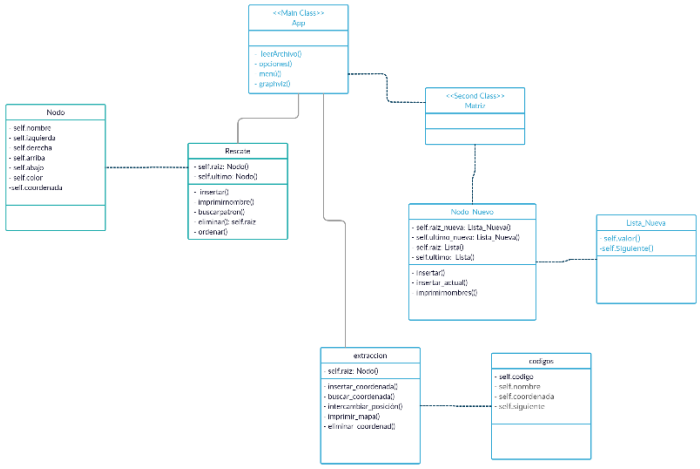

Figura II. Diagrama de Clases

Fuente: Elaboración propia, 2022