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

202001144 – Edgar Rolando Alvarez Rodriguez

Resumen

Se presento un problema el cual consiste en la lectura de un archivo XML para el manejo de ciudades y robots, las ciudades con sus propios obstáculos y los robots en listas distintas para su manejo en búsqueda y rescate, dichos datos se encuentran dentro del archivo XML que deberá ser procesado con código en lenguaje Python.

Dichos procesos realzados con el inconveniente de utilizar listas creados a partir de clases

Palabras clave

Código, Patrones, Python, Listas

Abstract

A problem is presented which consists of reading an XML file for the management of cities and robots, the cities with their own obstacles and the robots in different lists for their management in search and rescue, these data are found within the XML file which must be processed with code in Python language.

These processes carried out with the inconvenience of using lists created from classes

Keywords

Code, Patterns, Python, Lists

Introducción

En el siguiente ensayo se darán a conocer las funcionalidades creadas para la lectura y manejo de los datos dentro del archivo XML para crear los patrones de las ciudades y a su vez calcular con los robots existentes el camino para llegar a sus objetivos.

Se dan a conocer las funciones más importantes para el manejo de los datos, así como la estructura principal del código.

Desarrollo del tema

Como punto inicial se presenta el problema a resolver, el cual 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

Para poder completar el sistema de rescate y extracción, la empresa Chapín Warriors, S.A., lo ha contratado para crear un sistema de control que recibirá la malla de celdas del dron ChapinEye, a continuación, el sistema de control será capaz de definir una serie de misiones e indicará los robots que pueden completar la misión, así como los caminos que deben seguir en la malla de celdas para ejecutar las misiones.

Para resolver dicho problema se utilizó el lenguaje Python, junto con la librería “Minidom” para leer el archivo XML, la librería “Graphviz” para la creación de los patrones en pantalla y la librería “pahtfinding” para la búsqueda de los caminos.

Para el uso de la librería denominada “pahtfinding” se tuvo que modificar dicha librería para evitar el uso de caminos en diagonal, así como la modificación de este para su uso con TDAS.

La estructuración del código se separó en 3 archivos cada uno encargado de un proceso diferente para poder manejar un orden en específico en la estructuración de los datos.

a. Main.py

b. ListaDoble.py

c. escritorGraphviz.py

d. algoritmo.py

e. pathGraficador.py

ListaDoble.py

El código al interior de este archivo es el encargado de manejar una serie de listas creadas a partir de clases, dichas listas guardan objetos tipo ciudad y tipo robot.

Los objetos tipo “Ciudad” son instanciados dentro de este archivo, estos objetos incluyen los datos de la ciudad, los cuales son:

- Nombre de la ciudad
- Filas de la ciudad
- Columnas de la ciudad

- Datos de cada fila
    - Entradas, salidas, enemigos, civiles y recursos
- Unidades militares dentro de la ciudad

Así mismo los objetos tipo "Robot" se encuentran dentro de este archivo que incluyen:

- Nombre del Robot
- Tipo (ChapinFighter o ChapinRescue)
- Capacidad

Como no se pueden usar las listas que por defecto puede manejar el lenguaje Python, dentro de los objetos tipo "Ciudad" la variable que almacena las ciudades con sus respectivos datos es a su vez una clase que funciona como una lista, dicha clase es instanciada dentro del objeto para poder consultar los datos de las filas una por una sin ningún problema.

Main.py

Este archivo es el encarga de iniciar el menú, así como leer el archivo XML y mandar dichos datos leídos al archivo "ListaDoble.py" para luego consultar dicho archivo y utilizar los datos necesarios según sean necesarios.

La función "lectorXML()" se encarga de leer el XML, la función "main()" se encarga de iniciar el programa. El programa de vista al usuario consta de un menú que le permite:

1. Cargar un archivo XML
2. Ver las ciudades y robots
3. Salir del programa

Para obtener los datos de las ciudades dentro de las listas, se crearon diferentes funciones cada una encargada de devolver un tipo de dato(s) en específico.

```
==================== MENU ====================
1. Ingrese la ruta de su archivo XML          |
2. Ver Ciudades                               |
3. Salir                                      |
----Elige una opcion
==============================================
```

Figura I. Menú en Consola.
Fuente: elaboración propia,

```
--Escoja una ciudad para ver su informacion--
0) CiudadGotica
1) CiudadGuate
ingrese el numero de la ciudad: 0
Filas de la Ciudad: 15
Columnas de la Ciudad: 20
```

Figura 2. Menú en Consola.
Fuente: elaboración propia,

```
Ingrese las Coordenas del Civil a rescatar:
ingrese x: 18
ingrese y: 13
```

Figura 3. Opciones dentro del menú de la opción 2
Fuente: elaboración propia,

Explicación del Menú:

La opción #1 Es la más importante ya que podemos ingresar la ruta del archivo que queremos leer.

La opción #2 no d permite ver las ciudades dentro del archivo y escoger la ciudad que deseemos ver, así como los robots que podemos usar dentro de la ciudad.

La opción #3 nos permite salir de la aplicación.

escritorGraphviz.py

Este archivo es el encargado de generar las gráficas que nos permiten representar de manera más clara las ciudades seleccionadas, esto por medio de la edición de un archivo con extensión ".dot" que luego es transformado a un archivo ".pdf".

La función que genera la gráfica de la ciudad genera un nodo al que se le ingresa un código HTML con los datos del patrón seleccionado para luego generar el archivo ".dot"

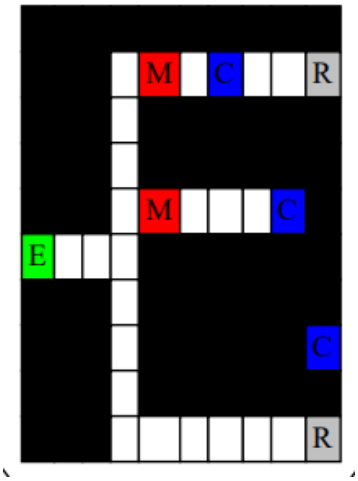


Figura 4. Grafica de una ciudad

Fuente: elaboración propia,

algoritmo.py

se encarga de encontrar el camino correcto para llegar al punto de rescate.

pathGraficador.py

Se encarga de graficar el camino correcto para llegar al punto final de rescate.

Conclusiones

Es posible implementar la memoria dinámica y un TDA dentro de otro para crear una lista dentro de otra y así tener acceso a los datos más fácilmente.

Mediante el uso correcto de librerías, el manejo de archivos y distintos datos se vuelve más fácil, por ello se recomienda no saturar un archivo de librerías y usar un numero adecuado sin exagerar.

El uso de TDAS puede ser una segunda opción para crear listas al tener algún tipo de obstrucción o conflicto con el uso de las listas nativas de Python.

Referencias bibliográficas

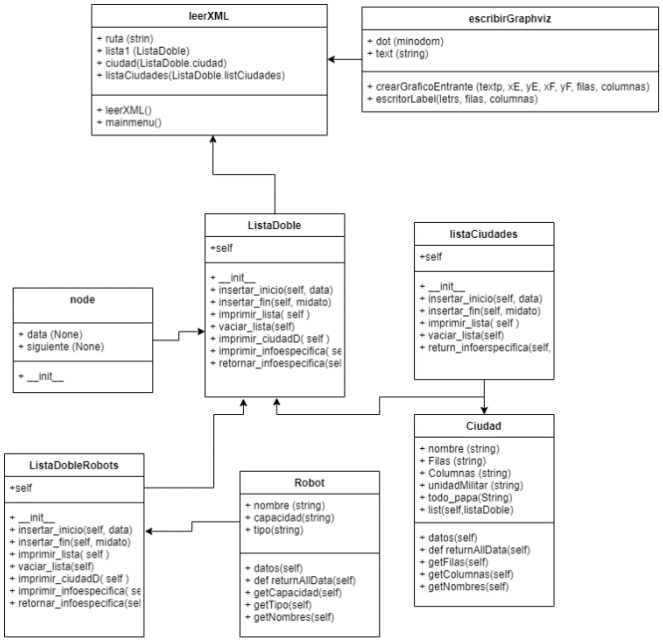
C. J. Date, (1991). An introduction to Database Systems. Addison-Wesley Publishing Company, Inc.

Node Shape (s.f.) Graphviz Recuperado 3 de marzo 2022, de https://graphviz.org/doc/info/shapes.html

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.

Anexo:

Diagrama de Clases:



Código dentro de los archivos:

```
ciudadx = ListaDoble.Ciudad('','','','','')
listaCiudades = ListaDoble.doubleList()

listMilitares = ListaDoble.Ciudad.lista_Militares()
militar = ListaDoble.unidadMilitar('','','')


listaRobots = ListaDoble.lista_Robots()
robot = ListaDoble.Robot('','','')

# VARIABLES GLOBALES
todo_papa = ''
ruta = ''
list_unidadesTexto = ''
unidadesTexto = ''
```

```
def lectorXML(rutanueva):
    global todo_papa, ruta, ciudadx, list_unidadesTexto, unidadesTexto, listMilitar
    ruta = rutanueva

    sub2 = 0

    mydoc = minidom.parse(ruta)

    ciudades = mydoc.getElementsByTagName('ciudad')

    """Lo que esta comentado de PRINT NO TOCAR"""
    # Intentando leer los pisos
    for x in ciudades:
        todo_papa=''
        if x.getElementsByTagName("nombre")[0]:
            e_nombre = x.getElementsByTagName('nombre')[0]
            print(e_nombre.nodeName, ':', str.strip(e_nombre.childNodes[0].data))

            filas = e_nombre.getAttribute('filas')
            columnas = e_nombre.getAttribute('columnas')
            # print('Filas: ' + str(filas))
            # print('Filas: ' + str(columnas))
```

```
unidadesTexto = divisorMapeador(todo_papa)
# print(unidadesTexto)
nuevaCadena = ''
for unidades in x.getElementsByTagName('unidadMilitar'):
    n_u_fila = unidades.getAttribute('fila')
    n_u_columna = unidades.getAttribute('columna')
    # print('#fila: '+ str(n_u_fila) + ' #columna: ' + str(n_u


    militar.fila = n_u_fila
    militar.columna = n_u_columna
    militar.capacidad = unidades.childNodes[0].data
    listMilitares.insertar_final(militar)
    militar = ListaDoble.unidadMilitar('','','')


    cont = 0
    for cosas in unidadesTexto[int(n_u_fila)-1]:
        # print('primero: '+cosas)
        if cont == (int(n_u_columna)-1):
            # print(cosas)
            cosas = 'M'
            # print(cosas)
        nuevaCadena += cosas
```

```
robots = mydoc.getElementsByTagName('robot')
for x in robots:
    if x.getElementsByTagName("nombre")[0]:
        e_r_nombre = x.getElementsByTagName('nombre')[0]
        # print(e_r_nombre.nodeName, ':', str.strip(e_r_nombre.chi

        e_r_tipo = e_r_nombre.getAttribute('tipo')
        e_r_capacidad = e_r_nombre.getAttribute('capacidad')
        # print('Tipo: ' + str(tipo))
        # print('Capacidad: ' + str(capacidad))
        # print('\n')
        """"""
        robot.tipo = str(e_r_tipo)
        robot.capacidad = (e_r_capacidad)
        robot.nombre = (str.strip(e_r_nombre.childNodes[0].data))
        listaRobots.insertar_final(robot)
        robot = ListaDoble.Robot('','','')
        """"""
```

```
def menu():
    global list_unidadesTexto, unidadesTexto, ciudadx
    salir = False

    while not salir:
        print('==================== MENU ====================')
        print ("1. Ingrese la ruta de su archivo XML           |")
        print ("2. Ver Ciudades                                |")
        print ("3. Salir                                       |")
        print ("----Elige una opcion")
        print('==============================================')
        opcion = pedirNumeroEntero()

        if opcion == 1:
            print('\n \n====================OPCION 1====================')
            # ruta = input("introduzca a ruta de tu archivo XML a analizar:
            # print('\nsu ruta escogida es: '+ruta)
            fn_abrirArchivo()
            print('Analizando XML...')
            lectorXML(ruta)
```

```
print('Filas de la Ciudad: ' + filas)
print('Columnas de la Ciudad: '+ columnas)
print('')
todo_papa = todo_papa.replace('\']','') #.spli
todo_papa = todo_papa.replace('[\'','')
todo_papa = todo_papa.split('\', \'')
"""""""
listicaMilitares = militares[:-1].split('L')
cont = 0
print('Unidades Militares en la Ciudad')
for x in listicaMilitares:
    if cont == 0:
        print('Fila: ' + x)
        cont += 1
    elif cont == 1:
        print('Columna: ' + x)
        cont += 1
    elif cont == 2:
        print('Capacidad: '+ x)
        print('')
        cont = 0
```

Clase Nodo:

```
class node:
    def __init__(self, data = None, siguiente = None, nodoHijin = None): #Constructor
        self.data = data #Este el atributo, puede mas atributo
        self.siguiente = siguiente  #Apunta al siguiente nodo
        self.nodoHijin = nodoHijin
```

Código para introducir los valeres de los patrones para generar la gráfica de estos:

```
def escritorLabel(letras, filas, columnas):
    text = ''
    filas = int(filas)
    columnas = int(columnas)
    contador = 0
    for x in range(filas):
        text += '<TR>'
        # filasL = ((len(letras))/2)
        for y in range(columnas):
            if contador >= filas:
                break
            for cosas in letras[contador]:
                text +='<TD'
                if cosas == '*':
                    text += ' BGCOLOR="black"'
                elif cosas == ' ':
                    text += ' BGCOLOR="white"'
                elif cosas == 'E':
                    text += ' BGCOLOR="green"'
                elif cosas == 'C':
                    text += ' BGCOLOR="blue"'
                elif cosas == 'R':
                    text += ' BGCOLOR="gray"'
                elif cosas == 'M':
```