|  |
| --- |
| **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.

Texto

Descripción generada automáticamente

Figura I. Menú en Consola.

Fuente: elaboración propia,

Texto

Descripción generada automáticamente

Figura 2. Menú en Consola.

Fuente: elaboración propia,

Texto

Descripción generada automáticamente

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”

Imagen que contiene texto, juego

Descripción generada automáticamente

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

**Anexo:**

Diagrama de Clases:

Diagrama

Descripción generada automáticamente

Código dentro de los archivos:

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Clase Nodo:

Texto

Descripción generada automáticamente

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

Texto

Descripción generada automáticamente