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.

Proyecto No. 2

201807154 – Denny Alexander Chalí Miza

Resumen

Para la realización del proyecto No. 2 se realizó una aplicación cuyas funciones eran las de analizar archivos XML, las cuales contienen datos de distintas ciudades, su estructura (mapa de celdas), enemigos en el mapa y los robots que exploraran el mapa. La función principal del programa es que dependiendo del tipo de misión que se seleccione se pueda encontrar una ruta evitando obstáculos y enemigos del punto de entrada al punto hacía donde se ubica el recurso o unidad civil.

Para la solución del problema se hizo uso de TDA, usando listas simples, listas doblemente enlazadas y listas ortogonales, siendo la lista ortogonal la que contiene los datos de las mallas de cada ciudad.

Para mostrar gráficamente el mapa y la ruta encontrada se hizo uso del software Graphviz, la cual muestra la matriz que contiene la lista ortogonal y la lista de celdas que componen el camino de la entrada a la unidad civil o al recurso.

Palabras clave
Pila, Lista, Funciones, Métodos y Clases.

Abstract

To carry out project No. 2, an application was made whose functions were to analyze XML files, which contain data from different cities, their structure (map of cells), enemies on the map and the robots that explored the map. The main function of the program is that depending on the type of mission selected, a route can be found avoiding obstacles and enemies from the point of entry to the point where the resource or civil unit is located.

For the solution of the problem, TDA was used, using simple lists, doubly linked lists and orthogonal lists, being the orthogonal list the one that contains the data of the grids of each city.

To graphically display the map and the route found, the Graphviz software was used, which shows the matrix that contains the orthogonal list and the list of cells that make up the path of entry to the civil unit or resource.

Keywords
Stack, List, Functions, Methods and Classes.

Introducción

Se desarrollo una aplicación que permite la lectura y análisis de archivos con formato XML, las cuales contienen una malla de celdas con caminos, obstáculos, enemigos, recursos y unidades civiles. La aplicación es capaz de definir una serie de misiones (rescate o extracción) e indicar los robots que pueden completar la misión, así como mostrar gráficamente la malla de celdas con el camino trazado hacia el objetivo

Desarrollo del tema

1. Preparando el entorno de Trabajo

Para la resolución del problema se utilizó el lenguaje de programación Python, el cual deberá de descargar e instalar en su ordenador, para el correcto funcionamiento del programa se recomienda instalar una versión superior a la 3.6.

1.1. Graphviz

Graphviz es un software de visualización de gráficos de código abierto que nos servirá para graficar los patrones de los pisos de forma gráfica.
Para su instalación debemos de tener previamente instalado Python.
Abrimos el símbolo del sistema (CMD) e ingresamos:

```
$ pip install graphviz
```

Para renderizar el código fuente DOT generado se puede usar el software Grahviz descargándola desde su sitio oficial.

2. TDA implementados

Para almacenar los datos de los archivos y manejar las rutas en cada malla fueron necesarios implementar distintos tipos de TDA**.**

2.1. Listas simples enlazadas

Una lista simple o lista simplemente ligada esta constituida por un conjunto de nodos alineados de manera lineal (uno después de otro) y unidos entre sí por una referencia (apuntador).
A diferencia de un arreglo, el cual también es un conjunto de nodos alineados de manera lineal, el orden está determinado por una referencia, no por un índice, y el tamaño no es fijo.
La unidad básica de un alista simple es un elemento nodo, cada elemento de la lista es un objeto que contiene la información que se desea almacenar, así como una referencia (NEXT) al siguiente elemento (sucesor).

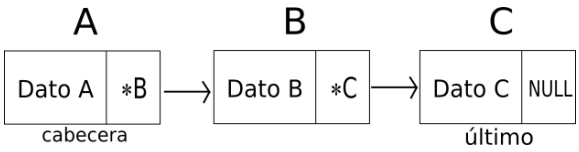

Figura 1. Lista enlazada simple

Fuente: Alberto Castillo G, obtenido de: www.betoissues.com.

2.2. Listas Doblemente enlazadas

Es una estructura de datos dinámica que se compone de un conjunto de nodos en secuencia enlazados mediante dos apuntadores (uno hacia adelante y otro hacia atrás).

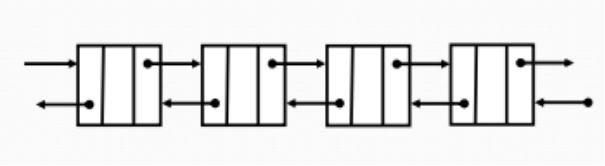

Figura 2. Lista doblemente enlazada.

Fuente: Bruno López Takeyas, M.C, obtenido de: www.itnuevolaredo.edu.mx/Takeyas

2.3. Listas Ortogonales

Este tipo de lista se utiliza para representar matrices. Los nodos contienen cuatro apuntadores. Uno para apuntar al nodeo izquiedo, otro para apuntar al derecho, otro al nodo al nodo inferior y por último un apuntador al nodo superior.

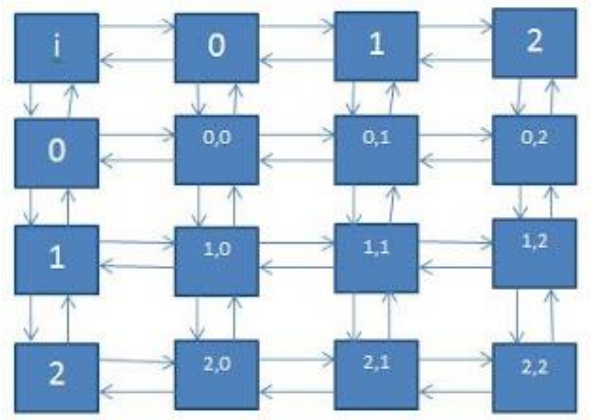

Figura 3. Lista ortogonal.

Fuente: Humberto Herrado, obtenido de: ciencias-y-sistemas.blogspot.com/

3. Iniciando el programa

3.1. Posicionarse en la carpeta (CMD)

Una vez el entorno de trabajo está listo podemos iniciar el programa, para ello abrimos una pestaña en el símbolo del Sistema (CMD) y navegamos hasta estar en la carpeta donde se ubica el ejecutable de nuestro programa:

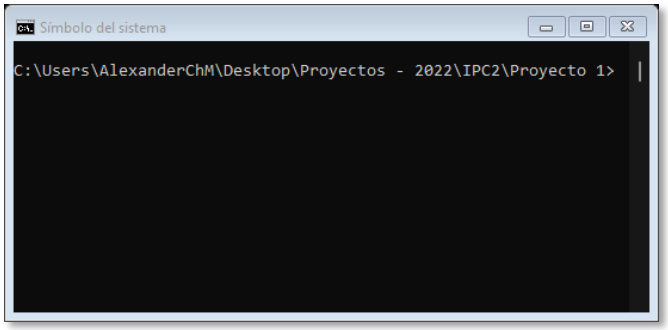

Figura 4. Ejemplo de ubicación en carpeta

Fuente: elaboración propia.

Ingresamos el siguiente comando una vez dentro de la carpeta que contine el ejecutable:

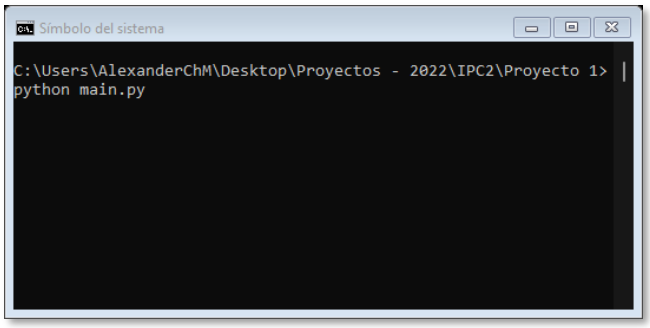

Figura 5. Inicializando el programa

Fuente: elaboración propia.

Una vez realizado los pasos anteriores se desplegará el menú de inicio del programa.

3.2. Funciones del programa

3.2.1. Menú de inicio

Al iniciar el programa se desplegará el menú inicial con las principales funciones como se muestra en el ejemplo:

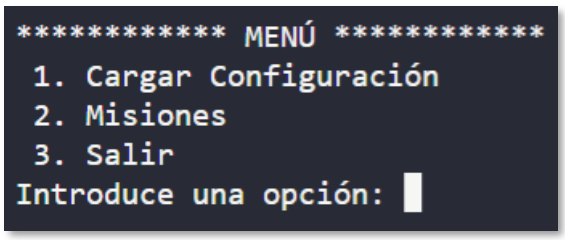

Figura 6. Pantalla de inicio.

Fuente: elaboración propia.

Para seleccionar una función se deberá de ingresar el número de la opción correspondiente como se muestra a continuación:

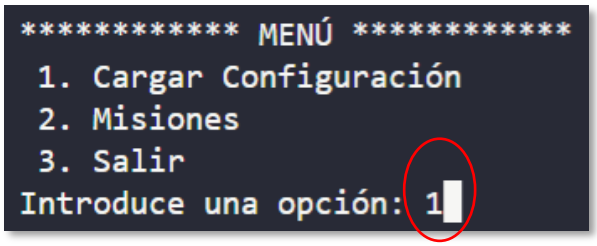

Figura 7. Ejemplo de selección de opciones.

Fuente: elaboración propia.

Para los submenús se utiliza el mismo método para seleccionar opciones y seleccionar elementos.

3.2.2. Cargar configuraciones (Mallas)

Esta función se encarga de desplegar una ventana emergente y seleccionar el archivo de entrada que contine los datos obtenidos por el dron ChapinEye:

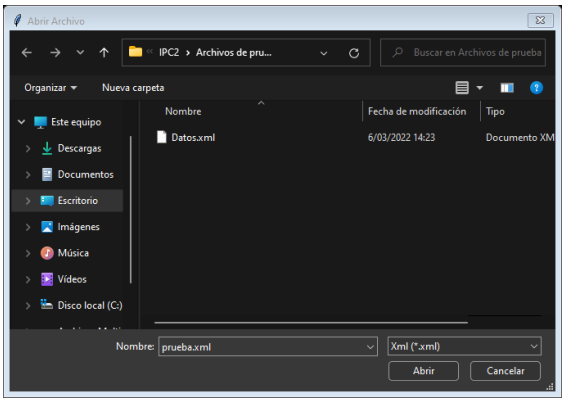

Figura 8. Ventana emergente para la selección del archivo.

Fuente: elaboración propia.

Una vez se localiza el archivo XML con los datos se selecciona, si los datos son correctos se mostrará un mensaje indicando los resultados satisfactorios de lectura, en caso contrario se mostrará un mensaje indicando que los datos no se cargaron.

3.2.3. Misiones

En esta sección deberá de seleccionar el tipo de misión que se desea realizar.

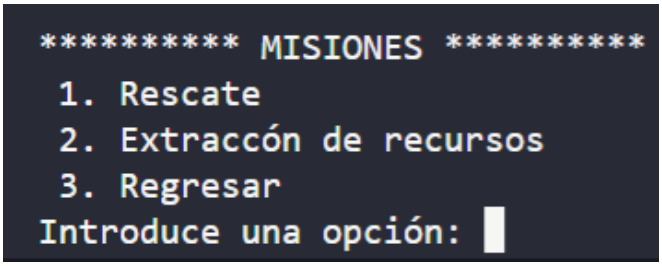

Figura 9. Menú de inicio.

Fuente: elaboración propia.

3.2.3.1. Rescate

3.2.3.1.1. Seleccionar robot

Estas misiones son realizadas por robots de tipo ChapinRescue. Si se tiene disponibles más de un robot ChapinRescue, el usuario deberá seleccionar uno. Si no hay robots ChapinRescue estas misiones no pueden ser llevadas a cabo

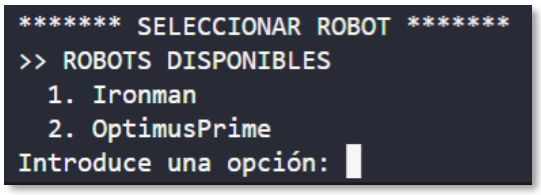

Figura 10. Menú selección de unidad robotica.
Fuente: elaboración propia.

3.2.3.1.2. Seleccionar ciudad

Se debe identificar la ciudad donde se realizará la misión de rescate, solamente las ciudades con "unidades civiles" pueden tener misiones de rescate

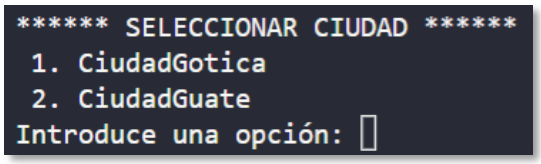

Figura 11. Ventana emergente para la selección del archivo.
Fuente: elaboración propia.

3.2.3.1.3. Seleccionar entrada

Toda misión de rescate debe iniciar en un punto de entrada

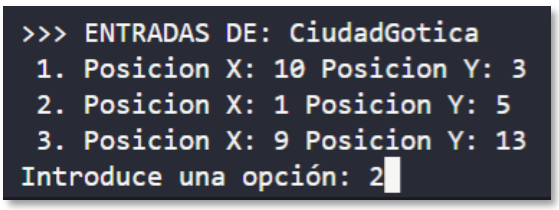

Figura 12. Menú selección de entrada.
Fuente: elaboración propia.

3.2.3.1.4. Seleccionar Unidad Civil

Si la ciudad tiene varias "unidades civiles" se deberá elegir la unidad civil a rescatar, si solamente hay una "unidad civil" no se solicitará esta elección.

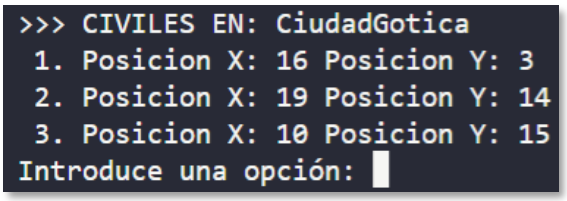

Figura 13. Menú selección de unidad civil.
Fuente: elaboración propia.

3.2.3.1.5. Grafica de la ruta segura.

Las misiones de rescate deben identificar un camino seguro, es decir, deben evitar cualquier camino que contenga "unidades militares"

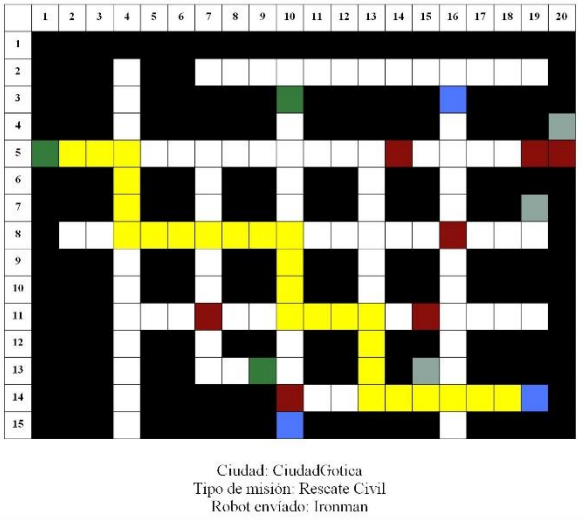

Figura 14. Gráfica de la ruta encontrada.
Fuente: elaboración propia.

3.2.3.2. Misión Extracción de recursos

3.2.3.2.1. Seleccionar robot

Estas misiones son realizadas por robots de tipo ChapinFighter. Si se tiene disponibles más de un robot ChapinFighter, el usuario deberá seleccionar uno. Si no hay robots ChapinFighter estas misiones no pueden ser llevadas a cabo

```
******* SELECCIONAR ROBOT *******
>> ROBOTS DISPONIBLES
  1. Robocop    Capacidad: 50
  2. MaxSteel    Capacidad: 10
  3. Megatron    Capacidad: 100
Introduce una opción:
```

Figura 15. Menú selección de unidad robotica.
Fuente: elaboración propia.

3.2.3.2.2. Seleccionar ciudad

Se debe identificar la ciudad donde se realizará la misión de extracción de recursos, solamente las ciudades con “Recursos” pueden tener misiones de extracción.

```
****** SELECCIONAR CIUDAD ******
 1. CiudadGotica
 2. CiudadGuate
Introduce una opción:
```

Figura 16. Ventana emergente para la selección del archivo.
Fuente: elaboración propia.

3.2.3.2.3. Seleccionar entrada

Toda misión de rescate debe iniciar en un punto de entrada

```
>>> ENTRADAS DE: CiudadGotica
 1. Posicion X: 10 Posicion Y: 3
 2. Posicion X: 1 Posicion Y: 5
 3. Posicion X: 9 Posicion Y: 13
Introduce una opción: 2
```

Figura 17. Menú selección de entrada.
Fuente: elaboración propia.

3.2.3.2.4. Seleccionar Recurso

Si la ciudad tiene varias celdas con “recursos” deberá elegir el recurso que desea extraer, si solamente hay un “recurso” no se solicitará esta elección.

```
>>> RECURSOS DE: CiudadGotica
 1. Posicion X: 20 Posicion Y: 4
 2. Posicion X: 19 Posicion Y: 7
 3. Posicion X: 15 Posicion Y: 13
Introduce una opción: 1
```

Figura 18. Menú seleccionar recurso a extraer.
Fuente: elaboración propia.

3.2.3.2.5. Grafica de la ruta segura.

Las misiones de extracción de recursos pueden transitar por celdas con “unidades militares” siempre y cuando su capacidad de combate sea mayor a la capacidad de combate de la “unidad militar”, en estos casos, el robot utilizado disminuirá su capacidad de combate restando la capacidad de combate de la “unidad militar” derrotada.

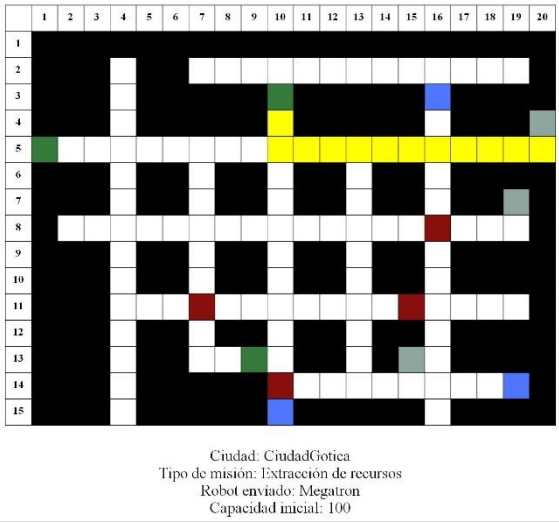

Figura 19. Gráfica de la ruta encontrada.
Fuente: elaboración propia.

Referencias bibliográficas

Lewis, John; Loftus, William (2009). *Java Software Solutions Foundations of Programming Design* 6th ed. Pearson Education Inc.

Chacón Sartori, Camilo. *Computación y programación funcional: introducción al cálculo lambda y la programación funcional usando Racket y Python*. [Barcelona]: Marcombo.

Anexos

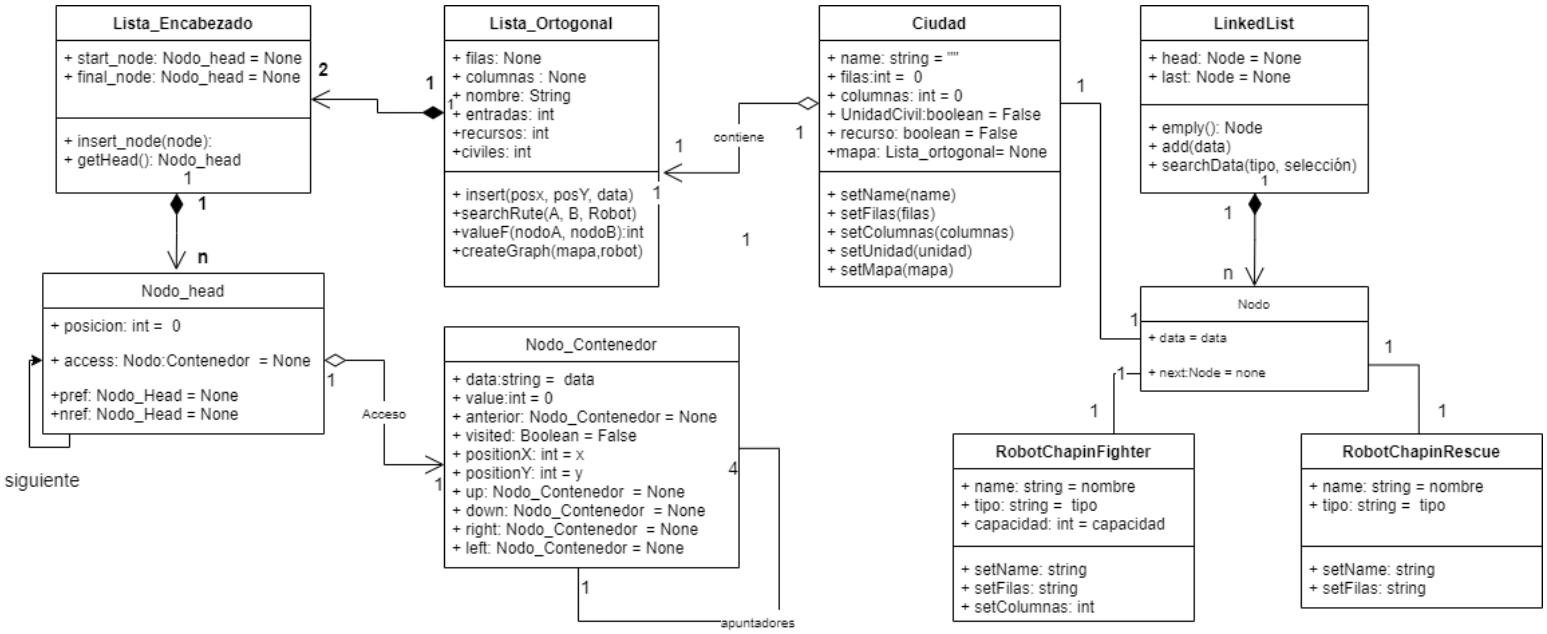

Figura 20. Diagrama de clases del programa desarrollado.

Fuente: elaboración propia.