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 2021.

MISIONES EN CHAPIN WARRIORS

202006688 – Karla Ernestina González Polanco

Resumen

El segundo proyecto del curso "Introducción a la Programación y Computación 2", tiene como objetivo, que el estudiante pueda comprender el proceso de manipulación que hace la computadora para algunas estructuras ya definidas dentro de un lenguaje de programación, implementando las suyas desde 0.

Para la implementación de estas estructuras determinadas como "TDA", se asignó al estudiante el análisis de un archivo con extensión XML, el cual contenía una lista de ciudades y una lista de robots, para las ciudades tendrá que hacerse la simulación de misiones a través de algoritmos de pathfinding. Cada ciudad está compuesta de unidades como:

- Celdas transitables e instransitables.
- Unidades civiles y militares.
- Recursos.

Los anteriores ayudarán al desarrollo del algoritmo desde un punto de entrada indicado a cierto objeto.

Palabras clave

TDA: Tipo de Dato Abstracto
Pathfinding: Algoritmo para resolución de laberintos.
Nodo: Registro que contiene datos y apuntadores.
XML: Lenguaje de marcado extensible.
Matriz: Arreglo bidimensional de objetos.

Abstract

The second "Introducción a la Programación y computación 2" course project, has as purpose, allow the student to understand de manipulation process a computer must go through with the predefined structures a programming language has, by implementing his own structures right from the start.

For the implementation of these structures called "TDAs", the student was assigned to analyze an XML extension file, which contains a list of cities and a list of robots, for certain city a simulation of a pathfinding algorithm will have to run. Every city is composed of unities such as:

- *Passable and impassable cells.*
- *Civilians and military unities*
- *Resources*

All of the above, will help with de algorithm develop from a start point to certain object the user will have to choose from a objects list.

Keywords

ADT: Abstract Data Type

Pathfinding: Maze solving algorithm.

Node: Register which contains data and pointers.

XML: Extensible Markup Language.

Matrix: Rectangular array or table of numbers,

Introducción

Para el desarrollo del proyecto fue necesaria la implementación de TDAs, como listas simplemente enlazadas, doblemente enlazadas y una matriz. En este caso se utilizaron dos tipos de nodos ajenos a los necesarios para la matriz. Uno que almacena la información de cada ciudad y el segundo con los datos de los robots asignados a las ciudades. La información almacenada en estos nodos será necesaria para la validación de ciertas condiciones que deben cumplirse para que una misión (algoritmos de pathfinding), pueda llevarse a cabo.

La vista gráfica de las misiones realizadas fue hecha con la extensión ".dot" de graphviz, para la cual se usó como base la matriz que se generó por cada ciudad al cargar un archivo de configuración. También está contemplado que el usuario sea capaz de asignar un tipo de misión (de rescate o extracción), indicando el punto de partida, el robot que enviará a la misión y la unidad civil o recurso al que se llegará.

Desarrollo del tema

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.

Las celdas están clasificadas de la siguiente forma:

• Punto de entrada: Celda donde puede iniciar una misión de rescate o una misión de extracción. Estas celdas son transitables.

• Intransitable: Celda donde no es posible transitar.

• Camino: Celda donde si es posible transitar.

• Unidad militar: Celda donde existe una unidad militar, toda unidad militar tiene una valoración que consiste en un número entero que representa su capacidad de combate, mientras mayor es el número entero, mayor es su capacidad de combate. Una celda de tipo "Unidad militar" es transitable, siempre y cuando se pueda superar su capacidad de combate

. • Unidad civil: Celda donde existe una unidad civil. Una celda de tipo "Unidad civil" siempre es transitable. • Recurso: Celda que contiene algún recurso físico dentro de la ciudad. Una celda de tipo "Recurso" no puede ser transitada.

Para poder completar las misiones de rescate o extracción de recursos de ciudades en conflicto, Chapín Warriors, S. A., ha creado unidades robóticas que pueden realizar dichas misiones.

Algunos de los métodos utilizados para la resolución del proyecto son los siguientes:

1. **Lista de robots/ciudades**

- + insertarAlFinal(parámetros del nuevo nodo): Utilizado para listas simple y dóblemente enlazadas. Verifica si ya existe un nodo en la lista. En caso de que no exista, se seteará al nuevo nodo como el primero y el último de la lista. En caso de que ya haya uno, lo setea como último y aquel que anteriormente era el último ahora apuntará a este nuevo nodo que se creó. También suma uno al tamaño (número de objetos) de la lista.

- +buscarObjeto(id): Recorre toda la lista tomando como temporal al primer elemento y moviendo esta referencia hasta llegar al último. Para cada objeto analiza si el parámetro que se busca es igual a cierto parámetro propio del objeto.
- +verDisponibilidad(tipo): Cuenta el número de objetos que tengan el tipo especificado como parámetro.
- +verTipo(): Verifica si existe al menos un objeto que tenga el tipo especificado como parámetro, en caso de que sí, lo devuelve.

2. Matriz

- +insertar(x,y,tipo): Crea un nuevo objeto que tendrá los parámetros especificados. A continuación, verifica si existe una cabecera para la fila (coordenada en x) y la columna (coordenada en y) que le pertenecen a esta nueva celda. En caso de que no existan se crean teniendo el id correspondiente al tipo de cabecera que sea. Para su inserción en una fila se verifica si el acceso de la cabecera a alguna celda exista, si no existe el nuevo nodo será el acceso que tendrá la fila. En caso contrario, se analiza por medio de las coordenadas de columna de los otros accesos, si esta nueva celda debe colocarse antes o después de cierto acceso. Si la coordenada en Y de la nueva celda es menor a la del acceso de fila, deberá setearse como siguiente acceso de la nueva celda al acceso inicial y a este acceso inicial setearse su anterior, que será la nueva celda, por último, se setea el acceso a la nueva celda. Se trabaja de forma similar, siguiendo las restricciones de si la coordenada en Y de la nueva celda es mayor, igual, o si ya existe una celda en esa posición. Luego de insertarla en la fila, se inserta de forma similar en la columna, pero ahora comparando las posiciones en X de la nueva celda, y los accesos de las columnas.
- +ubicarCoordenada(x,y,capacidad,tipo): Para la inserción de las unidades militares en un mapa fue necesario ubicar la coordenada en la que se posicionarán, partiendo de los parámetros x y y especificados. Conociendo el valor de la fila, se obtiene su acceso y se recorre hasta encontrar la coordenada donde se insertará. Por último, se setea a la celda el nuevo valor de capacidad y tipo.
- +verTipoCelda(tipo): Recorre cada fila de la matriz obteniendo a través de la cabecera a su acceso y en caso una de las celdas sea del tipo especificado en el parámetro, se imprime su coordenada en x y y.
- +graficarMatriz(ciudad): Se especifican las características generales que tendrá el archivo ".dot". Para graficar la matriz es necesario primero crear las cabeceras, por lo que se para las cabeceras para filas: se crea una variable temporal que almacena el valor de la primera cabecera de fila que tenga la matriz, posteriormente se va creando cada cabecera como un nuevo nodo en el documento y se hace la iteración para llegar hasta la última cabecera de fila que tenga la matriz, luego se especifica a cuál cabecera apuntará cada una, por ejemplo: F1->F2.
  El proceso anterior se hace también con las cabeceras de columna, pero a estas también se les asigna el mismo rango que la raíz (iterando de forma similar a como se hace para crear un nodo tipo cabecera), para que, al momento de ver la gráfica, todas aparezcan alineadas. Se enlazan la raíz con la

primera cabecera de fila y la raíz con la primera cabecera de columna.
Seguidamente, se crea cada nodo de tipo celda de la matriz, especificándoles un color dependiendo del tipo que tenga.
Se hace el enlace de las cabeceras a la primera celda de esa fila o columna y se hace un enlace entre todo el resto de las celdas, al usar sus apuntadores derecha y abajo para las iteraciones y poder enlazar cada una de las celdas. Por último, se especifica el mismo rango para aquellas celdas que tengan la misma posición en y. En caso de que la graficación pertenezca a una misión, luego de la gráfica de la matriz se especifican el tipo de misión, la unidad o recurso que se rescato o extrajo y el tipo y nombre del robot asignado a completarla.

- +asignarMision(): dependiendo del tipo de misión defino el tipo de robot que es capaz de realizarla. Cada ciudad cuenta con un atributo que especifica el número de unidades civiles y recursos que posee una ciudad. En caso de que si exista al menos uno de estos elementos, se despliega un menú para elegir a cuál civil o recurso se rescatará o extraerá. De forma similar, se utiliza el método verDisponibilidad y se pasa como parámetro el tipo de robot necesario para una misión. En caso exista al menos uno de estos robots, se realiza la misión después de elegir a cuál robot se le asignará. También para ambos tipos de misión es necesario especificar el punto de partida de la misión por lo que se utiliza el método verTipo para desplegar la lista de aquellas celdas que sean del tipo “punto de entrada” y el usuario tenga la posibilidad de elegirlo. Por último, se genera la gráfica correspondiente a la misión con los detalles en la parte de abajo.

Conclusiones

La implementación de nuestras propias estructuras dinámicas nos da un mejor terreno para entender cómo es que funcionan las estructuras de los lenguajes de programación. Al ser estas estructuras predefinidas tan accesibles y “fáciles” de usar, podría perderse el razonamiento para el desarrollo de la solución a ciertos problemas que podrían presentarse en otras ocasiones.

También, su implementación nos permite tener las bases necesarias para poder hacer uso de ellas en cualquier otro lenguaje de programación, y sin ningún problema manejar información en forma de una lista, aunque este lenguaje tenga o no algo así entre sus estructuras.

El manejo de la información en archivos con extensión XML permite el aprendizaje de su manejo con diferentes tipos de lenguajes capaces de construir expresiones que puedan recorrerlo y de él extraer información importante para su posterior manejo en una aplicación.

Al hacer uso de graphviz, en su extensión .dot la implementación de sus diferentes características a una lista lineal de datos nos permite hacer diferentes cambias que nos permitan mostrar la información de forma gráfica como algo mucho más entendible a como podría verse inicialmente.

Todos los conocimientos adquiridos con la realización del proyecto son piezas importantes para permitir el desarrollo de la lógica de implementación de nuevas estructuras y de hacer su correcto manejo para obtener los resultados necesarios al ejecutar su aplicación.

Referencias bibliográficas

Phyton TM, (2001-2022). *Minimal DOM implementation.* Python Software Foundation.

Emden R. Gansner and Eleftherios Koutsofios and Stephen North, (2015). *Drawing graphs whit dot.* Graphviz.

[Nombre del autor], (2006). *Tipos de datos abstractos (TDA).* Instituto tecnológico de Celaya.

Eduardo Zepeda (2021). Taller de graphviz completo.

Tech With Tim(2020). Python Path Finding Tutorial - Breadth First.

Anexos

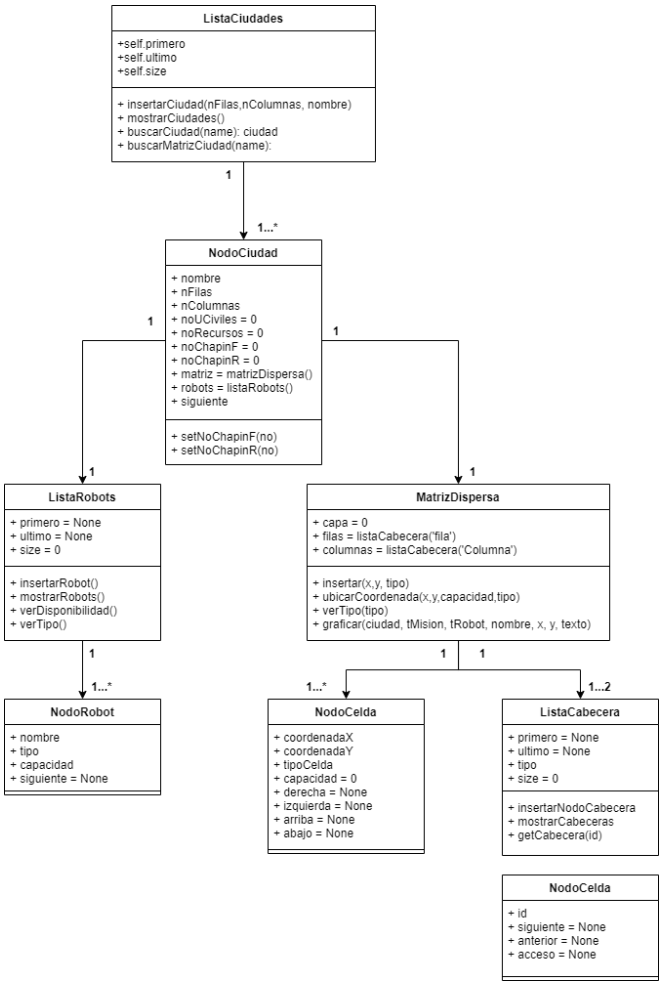

Figura 2. Diagrama UML.

Fuente: elaboración propia.