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 2 sistema de control de Drones Chapín Warriors, S. A.

202010044 – Ramiro Agustín Télles Carcuz

Resumen

El programa sistema de control de drones Chapin Warriors puede leer distintas listas de drones y mapas 2d de ciudades de un archivlo xml y después poder gestionar de buena manera las misiones encargadas a los drones de Chapin Warriors.

El programa muestra todas las ciudades y drones cargados , y también incluye otras funcionas para hacer mejor gestión de los drones y las misiones que tiene que manejar Chapin Warriors S. A.

Este programa incluye la función de graficar las rutas que los drones tendrían que tomar para poder visualizar la ruta correcta que hará que el dron de Chapin Warriors S.A. tenga éxito en su misión.

Esto sumándole una interfaz amigable con el usuario para que sea mucho más simple e interactivo su uso para los trabajadores.

Estas y otras funciones implementadas con una gestión de memoria dinámica para lograr la mayor eficiencia en el programa y que sea de gran utilidad para Chapin Warriors S. A.

Palabras clave
TDA,Pila,Memoria Dinámica, laberinto

Abstract

The control system of drones can read different list of drones and 2d maps of cities by a xml file and it can manage the missions of the drones of chapin warriors in a good way.

The program shows the whole cities and drones, and also includes another functions to do a good manage of the drones and the missions that Chapin warriors S. A. managed.

This program includes the function to graph the paths the drones need to follow to make the drone of Chapin Warriors S. A. success in the mission.

It has a friendly interface to make the program more simpler and interactive for workers.

This and another functions implemented with a dynamic memory manager to make the program more efficient and more useful to Chapin Warriors S.A.

Keywords
TDA,Pile,Dynamic Memory, labyrinth

Introducción

Aquí se presenta la solución de software pedida por Chapin Warriors S. A.l la cual pidió una solución de software en la que este pueda llevar el control de una mejor manera de su producto.

Este pide un software el cual pueda manejar todos los drones, mapas de ciudades y misiones que los drones tendrán que realizar en las ciudades. Este programa se encarga también de mostrar una ruta que el dron pueda seguir para que pueda tener éxito en su misión.

Esto logrado con la implementación de una lista doblemente enlazada para la gestión dinámica de la memoria, se utilizó listas dentro de listas para los arrays 2d y un algoritmo para poder diseñar una ruta que el dron pueda seguir.

Y también esta la función de graficar para poder ver el camino que el dron tiene que seguir.

Se utilizó una pila para poder resolver la ruta que seguirá el dron para completar la misión.

Desarrollo del tema

A. Lista doblemente enlazada

Este programa implementa una lista doblemente enlazada para poder llevar a cabo la buena gestión y los cálculos que se tengan que hacer al momento de manipular las casillas de las ciudades, la lista de drones, etc.

Se implementaron los nodos que serán los elementos que tendrá la lista, los cuales se les asignaran los datos que el programa necesite para poder hacer una buena gestión de memoria.

Los nodos tienen 3 atributos que son el dato que contendrán, el apuntador al siguiente nodo, y el apuntador al nodo anterior.

La clase lista se encargará de poder gestionar todos los nodos para poder ingresar cada nodo de la lista y poder obtener la información que tienen.

Tienen como atributos una variable booleana que es vacio y esta dice si la lista está vacia o no, otra variable esntera llamada cant que lleva la cantidad de elementos que tiene la lista y un apuntador llamado inicio que apunta al primer nodo de la lista.

Esta lista tiene los siguientes métodos:

- Imprimir: Imprime los elementos de la lista.
- Remplazar: remplaza el dato de un nodo en la posición deseada.
- EliminarPos: Elimina uin nodo de la lista según la posición deseada.
- EliminarDato: ELiminar un nodo de la lista según el dato dado.
- Agregar_inicio: Agrega un nodo al inicio
- Agregar_Final: Agrega un nodo al final de la lista.
- Buscar: busca si existe un dato en la lista.
- getPos: devuelve el dato de la lista según la posición dada.

B. Robots

El programa lleva la clase robot que es una clase en la que guardará todos los datos que sean necesarios para el manejo de los robots en el programa

Atributos de la clase robot:

- nombre: nombre del robot
- tipo: tipo del robot
- capacidad: capacidad del dron o poder de combate, si el dron no es de tipo ChapinFighter, su capacidad es 1.

Para poder llevar una buena gestión de los robots que se llevan en el programa, se hizo uso de una lista doblemente enlazada para poder almacenar los robots y hacer un uso correcto y eficiente de la memoria del sistema.

C. Ciudad

El programa lleva la clase ciudad que es una clase en la que guardará todos los datos que sean necesarios para el manejo de las ciudades en el programa.

Atributos de la clase ciudad:

- nombre: nombre de la ciudad
- filas: numero de filas del tablero de la ciudad
- columnas: numero de columnas del tablero de la ciudad
- unidadesMilitares: lista de en que casillas se encuentran las unidades militares y que poder de combate tienen, haciendo uso de la clase casilla para cada casilla de la lista.
- entradas: lista de casillas que son entradas
- recursos: lista de casillas que son recursos
- civiles:lista de casillas que son civiles.
- tablero: representación 2d de la ciudad en una lista dentro de una lista doblemente enlazada.

Métodos de la clase ciudad:

- imprimirTablero: imprime el tablero de la ciudad en consola
- colocarMilitares: coloca una bandera donde se ubican las unidades militares en el tablero de la ciudad.

Para poder llevar una buena gestión de las ciudades que se llevan en el programa, se hizo uso de una lista doblemente enlazada para poder almacenar las ciudadess y hacer un uso correcto y eficiente de la memoria del sistema.

D. Casilla

El programa lleva la clase casilla que es una clase en la que guardará todos los datos que sean necesarios para el manejo de las casillas del tablero de ciudad en el programa

Atributos de la clase casilla:

- fila: fila en la que se encuentra empezando desde 1
- columna: columna en la que se encuentra empezando desde 1.
- poder: poder de combate de una casilla, si la casilla no es militar, su poder es 0

Para poder llevar una buena gestión de las casillas que se llevan en el programa, se hizo uso de una lista doblemente enlazada para poder almacenar los robots y hacer un uso correcto y eficiente de la memoria del sistema.

E. Ingreso de los robots y ciudades al programa

Para poder ingresar los robots y ciudades en el programa, se hizo uso del lenguaje xml y de una estructura simple en la que almacenaba la información de todos los robots y las ciudades.

Para poder leer el xml se utilizó una librería llamada element tree para poder leer todo el archivo xml y después poder leerlo y extraer la información que el archivo xml traía.

F. Graficador

Aqui se guarda el método "crearGráfica" el cual recibe como parametro el tablero que se desea crear una representación gráfica para poder mejorar la visualización del tablero de la ciudad.

G. Pila

La clase pila hereda de la clase lista doblemente enlazada, y esta simplemente agrega dos nuevos métodos.

Métodos propios de Pila:

- append: recibe como parámetro un dato y lo agrega al final de la pila
- pop: elimina el ultimo dato en la pila, y este dato lo retorna

H. resolverLaberinto

Aqui se guardaron los distintos métodos para poder encontrar un camino viable el cual el robot pueda tomar en la mision que se le encargó para poder tener éxito en su misión.

Métodos de resolverLaberinto:

- crearCopiaTablero: Crea una copia del tablero para poder manipular la copia y no modificar los datos del original. Recibe como parametro el tablero que se desea copiar, y retorna la copia.

- resolverElLaberinto: Algoritmo que recibe como parametro el tablero de la ciudad, la posición de la casilla de inicio y la casilla de la posición final deseada.

- obtenerPoder: Busca el nivel de poder que le corresponde a cierta casilla. Se recibe como parametro la lista de unidadesMilitares y la posición de la casilla la cual se quiere saber el poder.

- colocarRecorrido: Recibe como parametro el laberinto, y la lista de posiciones de las casillas que hay que pasar para poder llegar al destino. Devuelve el tablero con el recorrido marcado.

I. Algoritmo utilizado para resolver la misión

Para poder tomar la decisión de que camino el dron tiene que tomar para completar con éxito su misión se tomó el tablero de la ciudad como un laberinto y se implementó el algoritmo de Tremaux.

El algoritmo de Tremaux consiste en siempre dejar un rastro por donde se pasa y siguiendo las siguientes reglas:

- No seguir el mismo camino dos veces.

- Si se llega a una intersección, tome cualquier camino

- Si un camino nuevo lleva a una intersección ya visitada, o a un callejón sin salida, se vuelve hacia atrás hasta la última intersección visitada

- si un camino ya visitado lo lleva a una intersección ya visitada, tome un camino nuevo, y si no hay, puede tomar cualquier camino.

En el caso de inclusión de figuras, deben ser nítidas, legibles en blanco y negro. Se denomina figuras a gráficas, esquemas, fotografías u otros elementos gráficos.

J. Interfaz gráfica

Primero el programa le pedirá que ingrese la ruta del archivo xml.

Figura 1. Petición de la ruta del archivo xml.

Fuente: elaboración propia.

luego de ingresar la ruta, se le pedirá que elija un tipo de misión.

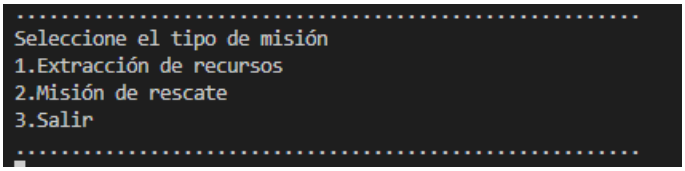

Figura 2. Petición de misión.

Fuente: elaboración propia.

Luego el programa pedirá que ingrese uno de los robots para completar la misión, si no hay robots para la misión, esta no se llevará a cabo, y si solo hay un robot capaz de la misión, este se elegirá automáticamente.

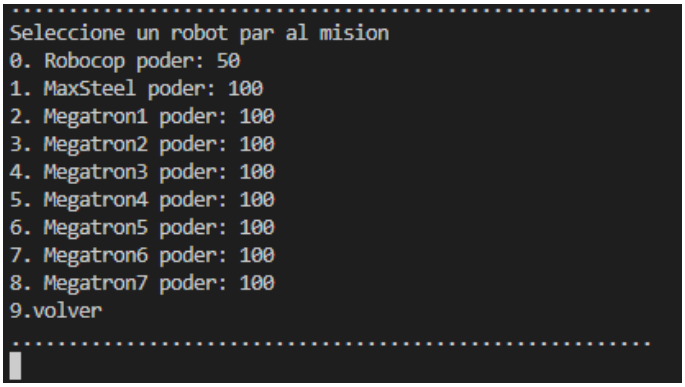

Figura 3. Elección de robot.

Fuente: elaboración propia.

Luego de haber elegido el robot, se pedira que seleccione la ciudad en la que se llevará a cabo la misión.

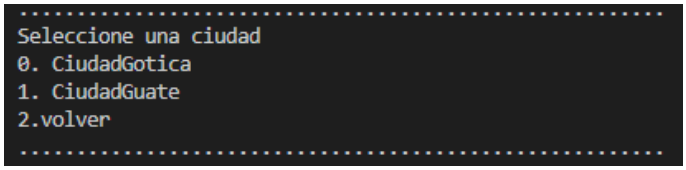

Figura 4. Elección de ciudad.

Fuente: elaboración propia.

Luego le pedirá la casilla a la que el robot tiene que llegar según que tipo de misión halla elegido. si solo hay una opción se elegirá automáticamente, y si no hay opción se cancelará la misión.

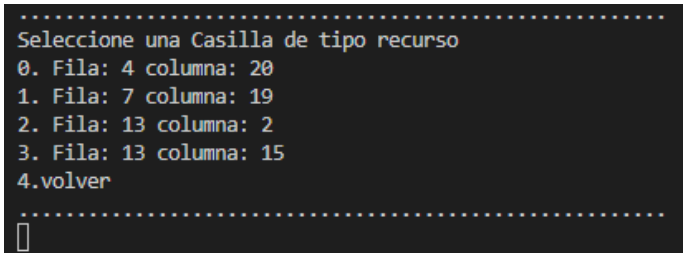

Figura 5. Elección de casilla.

Fuente: elaboración propia.

Luego le pedirá que elija una casilla en la que se pueda iniciar la misión. Si no hay, se cancela la misión, y si solo hay una se elegirá automáticamente.

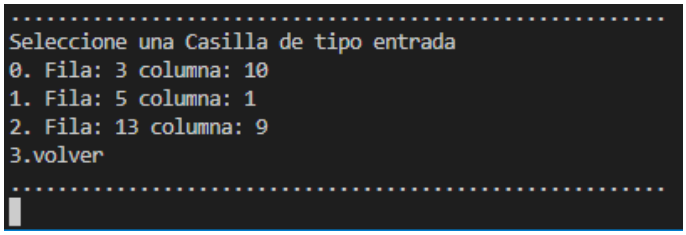

Figura 6. Elección de Entrada.

Fuente: elaboración propia

Luego se llevará a cabo la misión, y cuando esta acabe mostrará una imagen del camino que el dron debe tomar, y si no hay camino posile para poder completar la mision, se desplegará el mensaje Misión Implosible.

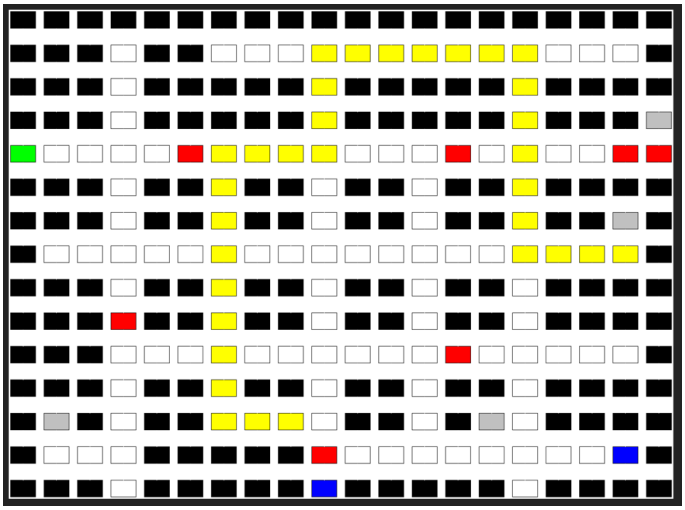

Figura 7. Misión desde la casilla 13,9 a la casilla 7 19.

Fuente: elaboración propia

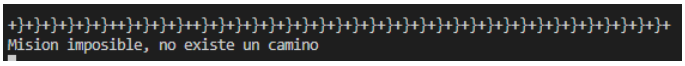

Figura 8. Mision imposible.

Fuente: elaboración propia

Conclusiones

Se logró realizar un programar que usa la memoria de manera más eficiente gracias a los TDAs

Se mostró una forma sencilla en la que una computadora o un humano puede resolver un laberinto.

Se logró realizar un programa que es capaz de resolver un laberinto utilizando memoria dinámica y un simple algoritmo.

Referencias bibliográficas

Delgado A., Esparza C. (2014). Robot seguidor de línea, modo solucionador laberinto. Universidad Santo Tomás de Bogotá.

M. seidl, M. Scholz, C. Huemer, G. Kappel, (2012). An introduction to Object-Oriented modeling. Springer International Publishing AG.

Anexos

Diagrama de clases

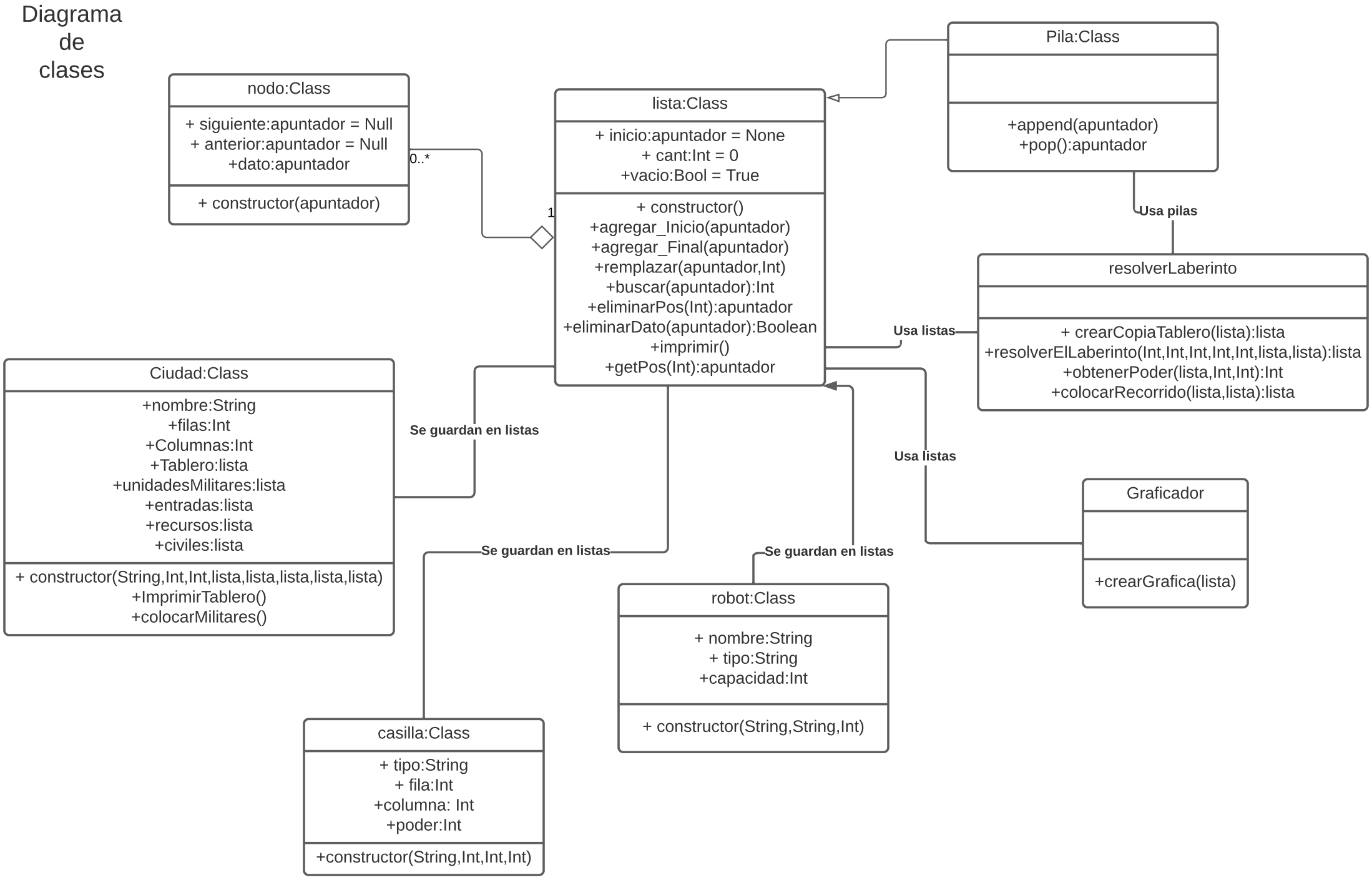

Diagrama de actividad para resolver el laberinto