Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
Introducción a la programación y computación 2, 2do. Semestre 2021.

Ruta óptima para el robot R2E2

201903791 – Kevin Gerardo Ruíz Yupe

Resumen

Para las agencias de investigación espacial la exploración en diferentes planetas es una oportunidad para responder preguntas desde un punto de vista científico

Los robots son una herramienta indispensable para las investigaciones, sin embargo, funcionan con instrucciones predeterminadas. Es por ello que este ensayo explica un algoritmo para recorrer el camino óptimo y así, gastar menos combustible.

Palabras clave
Ortogonal

Algoritmo

Lenguaje de Marcado Extensible

Herramienta Graphviz

Abstract

For space investigation agencies, exploration on different planets is an opportunity to answer questions from a scientific point of view.

Robots are an indispensable tool for investigations, but work with predetermined instructions. That is why this essay explains an algorithm to travel the optimal road and spend less fuel

Keywords
Orthogonal

Algorithm

XML (Extensible Markup Language)

Graphviz

Introducción

La Agencia Guatemalteca de Investigación Espacial (AGIE) ha diseñado un nuevo robot de exploración, llamado r2e2 y tiene la habilidad de explorar nuevos terrenos, este nuevo robot puede moverse en todo tipo de terrenos, sólo necesita más combustible para moverse en terrenos accidentados, y menos combustible en terrenos planos.

Para solucionar el problema del combustible se ha implementado una aplicación en consola que verifica qué camino consume menos combustible. Esta aplicación permite el ingreso de datos en una estructura .xml, y guarda las posiciones en una lista simple enlazada.

Utilizando la herramienta Graphviz se selecciona un terreno para graficarlo.

Desarrollo del tema

Como inicio de la aplicación se crean las clases necesarias para el proceso de información de los archivos con extensión .xml. Como lo muestra la imagen:

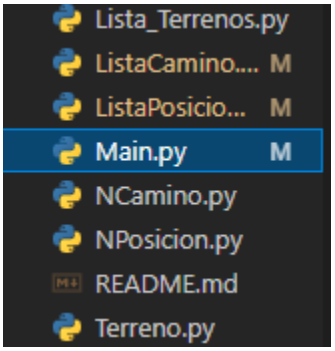


Figura 1. Creación de Clases

Fuente: Elaboración Propia.

A. Clase Nodo

Se define como Nodo a la parte de una lista enlazada cuyo objetivo es guardar los atributos (información del elemento a enlazar). Siempre se tendrá un apuntador hacia otro objeto Nodo

Se pueden definir los métodos *setters y getters*.

B. Clase Lista Simple

Se define como Lista al conjunto de procesos que se pueden realizar con un Nodo. Acá se encuentran métodos como: insertar, eliminar, obtener nodo y mostrar nodos.

Para aplicación se realizaron 3 clases Nodo como se muestra a continuación:

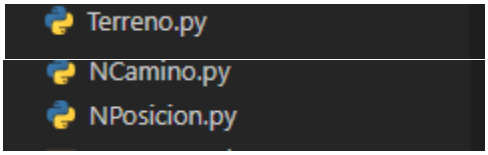


Figura 2. Clases Nodo

Fuente: Elaboración Propia.

C. Clase Terreno

En esta clase se obtienen los datos de cada mapa que el robot recorrerá.

```
from ListaPosiciones import ListaPosiciones

class Terreno:

    def __init__(self,nombremapa,posx,posy,posfx,posfy,nFilas,nColumnas):
        self.nombreMapa=nombremapa
        self.posInicialx=int(posx)
        self.posInicialY=int(posy)
        ####
        self.posFinalx=int(posfx)
        self.posFinaly=int(posfy)
        ###
        self.numFilas=int(nFilas)
        self.numColumnas=int(nColumnas)
                            Cargando...
        self.Lista_Posiciones=ListaPosiciones()
        self.siguiente = None
```

Figura 3. Vista de la Clase Terreno

Fuente: Elaboración Propia

Se observa que esta clase guarda datos como el nombre, posición en x inicial, posición en y inicial, posición en x final, posición en y final y finalmente el numero de columnas y filas (Dimensión) del terreno.

También, tiene un apuntador dirigido hacia el otro terreno (si existiera).

D. Clase NPosición

Esta clase obtendrá los datos de cada posición en el mapa:

```
class NodoPosicion:
    def __init__(self,posX,posY,combustible):
        self.posicionX=posX
        self.posicionY=posY
        # # # #  #   # # # #

        self.Combustible=combustible
        self.siguientePos=None

    def getPosicionX(self):
        return self.posicionX
    def getPosicionY(self):
        return self.posicionY
    def getCombustible(self):
        return self.Combustible
```

Figura 4. Vista de la Clase Posición

Fuente: Elaboración Propia

Datos como la posición en x, posición en y, combustible y el apuntador, se guardan en esta clase.

Se definen los métodos setters y getters para llevar un mejor control con las posiciones.

E. Clase NCamino

Esta clase tiene los datos de posición en x y posición en y del camino a seguir.

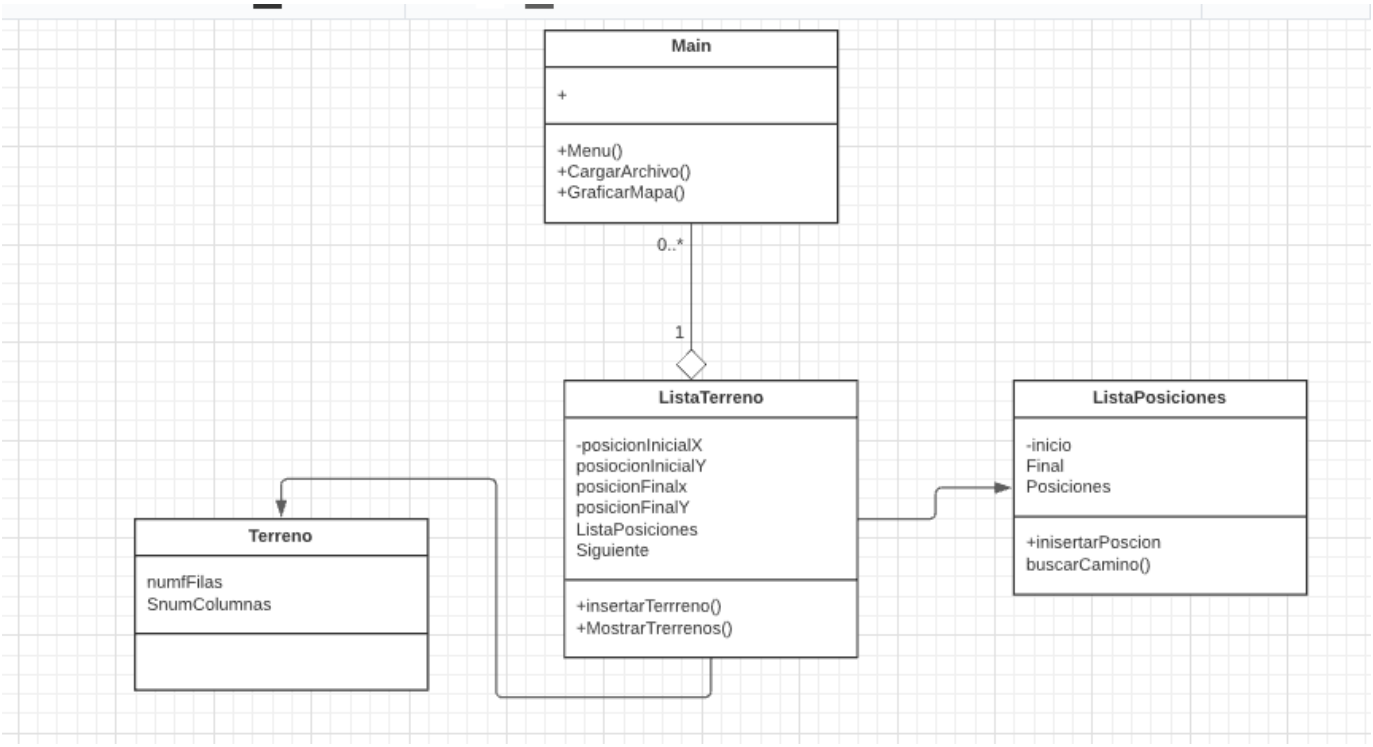
```
class NodoCamino:
    def __init__(self,posicionX,posicionY,combustible,marca):
        self.PosicionX=posicionX
        self.PosicionY=posicionY
        self.Combustible=combustible
        self.marca=marca
        self.siguiente=None

```

Figura 5. Vista de la Clase NCamino
Fuente: Elaboración Propia

Aquí se guarda el combustible gastado actualmente. El atributo marca contiene un string ‘1’ que indica que el nodo está disponible para recorrer.

Anexos


Main
+
+Menu()
+CargarArchivo()
+GraficarMapa()
0..*
1
ListaTerreno
-posicionInicialX
posiocionInicialY
posicionFinalx
posicionFinalY
ListaPosiciones
Siguiente
+insertarTerrreno()
+MostrarTrerrenos()
ListaPosiciones
-inicio
Final
Posiciones
+inisertarPoscion
buscarCamino()
Terreno
numfFilas
SnumColumnas